01

Introdução

Transcrição

[00:00] Olá, bem-vindo ao curso de testes em JavaScript com o Jasmine do Alura. Nesse curso, a minha ideia é discutir com vocês como escrever testes para códigos escrito em JavaScript. Afinal, JavaScript é uma linguagem que está cada vez mais popular. Os browsers dão suporte, então, toda aplicação faz uso do JavaScript para colocar alguma inteligência no client.

[00:28] Agora também com essa onda de Node.js, as pessoas têm inscrito códigos servers sides com JavaScript, estão, alguns código de JavaScript que antes eram basicamente manipular a interface, agora estão ficando complicados, cheio de regras de negócio e tudo mais. Escrever teste de JavaScript hoje é fundamental. E se você já conhece alguma coisa de testes, ou já fez algum outro curso da nossa formação, conhece o JUnit ou NUnit, você vê que nem é tão diferente assim.

[01:00] A grande diferença é na maneira de programar, como escrever o código JavaScript pra facilitar testabilidade. Se você não fez nenhum curso da formação de testes, não tem problema. Aqui eu vou começar bem do básico, motivando você do porquê testar, e como escrever o primeiro teste.

[01:18] Vamos começar. Eu vou programar aqui e eu vou usar o sublime, que é uma IDE bastante comum, estou usando o MAC aqui, mas você escolhe o seu editor de texto favorito, não vai fazer muita diferença para nós. O browser como vocês viram aqui, eu também estou usando o Chrome, porque não vai fazer muita diferença usar o Chrome ou o Firefox. Dá no mesmo.

[01:40] Então, aqui de volta para o sublime, a primeira coisa que vou fazer é começar como uma página HTML qualquer. Nem vou ser caprichoso no meu HTML porque não é esse o foco. Eu vou abrir com uma tag script, pra nós começar a programar em JavaScript. Aqui eu estou usando o browser, para facilitar a vida, mas lembre que o que fizemos aqui vai funcionar até pra aquele seu código Node.js, que não é código cliente side.

[02:06] E como eu falei para vocês, a maneira de programar em JavaScript tem que ser pensada também, então, desde já eu vou programar bonitinho. O primeiro problema que eu quero resolver é o seguinte: imaginem que eu tenho uma classe, e essa classe ela vai receber uma lista de números, e ela tem que me devolver o maior e o menor elemento dessa lista. Aquele algoritmo tranquilo, que você escreveu lá no seu primeiro ano de faculdade, ou no seu primeiro curso de linguagem de programação.

[02:3] Aqui eu vou programar o JavaScript simulando um pouco a orientação a objetos. Então, vou criar minha classe aqui, eu vou chamar essa classe de “MaiorEMenor”. E começar a declarar a minha classe, "var class". E eu vou criar a função aqui que eu vou chamar de “encontra”. A função “encontra” vai receber uma lista de números, e o que eu vou fazer com essa lista de números? Eu vou fazer aquele algoritmo mais convencional, eu vou varrer essa lista inteira e vou ficar guardando o maior e o menor elemento dessa lista.

[03:09] E vou o tempo todo comparar, o número atual é menor do que o menor de todos? Se for, eu troco. A mesma coisa pro maior. Se for maior de todos, eu também troco. Então, a primeiro coisa que eu vou fazer é o loop, “num.forEach”. Vou abusar um pouquinho dessa ideia do JavaScript, um pouquinho de linguagem funcional. Para cada número eu vou fazer comparação. Se o número atual foi menor do que o menor, eu vou trocar, “menor = num”.

[03:41] A mesma coisa pro maior, “maior = num”. Vou colocar um else aqui senão não vai funcionar. Eu preciso declarar essas variáveis maior e menor. Eu vou fazer isso aqui fora da classe, até pra usar e simular modificadores de acesso, então, não vou querer que essa variável menor seja enxergada por qualquer um. Vou mostrar pra vocês, “var menor”, e

“var maior”. A primeira coisa que eu vou fazer é inicializar esses caras. O menor eu vou inicializar assim. Eu vou colocar o maior número possível de JavaScript, pro número de JavaScript.

[04:20] Por quê? Porque pensa comigo, se o menor tem um número muito grande, a primeira vez que que cair nesse if, o if vai ser verdadeiro e vai trocar. Então, coloca um número bem grande, porque qualquer um vai ser menor que ele. No maior eu vou fazer exatamente o contrário, eu vou colocar um número bem pequenininho. Óbvio que para terminar esse código, eu preciso fazer alguma função para me devolver um maior e menor, senão não vai ter visibilidade.

[04:46] Se esse código JavaScript não está muito claro para você, essa jogada de classes, faça lá o nosso curso de JavaScript, que falamos um pouquinho sobre isso. Então, aqui eu vou fazer “return maior”. “pegaMenor”, retorna o menor. E o “pegaMaior”, retorna o maior.

[05:16] E aqui basta fazer agora o “return clazz”. Então, dando uma olhada esse código, function maior e menor, estou simulando uma classe, eu tenho duas variáveis escondidas, o menor e o maior, e eu tenho o método encontra. O método encontra faz um loop e o tempo inteiro ele olha o número e substitui se ele achou um menor do que o menor atual, ou se ele achou um maior do que o maior atual.

[05:39] Agora aqui eu vou escrever um teste para ele, vamos praticar igual você fez lá no primeiro código da sua vida. Você escreveu a função, e aí você escreveu um método main, ou alguma coisa que invocava essa função pela primeira vez. Então, vamos lá. "var algoritmo = new MaiorEMenor()”. “algoritmo.encontra”, e aqui vou passar uma lista de números quaisquer, então, 5, 15, 7 e 9. E eu vou imprimir esses números. “console.log(algoritimo.pegaMaior())”, e “console.log(algoritimo.pegaMaior())”.

[06:18] Então, se eu rodar esse programa, eu espero que a saída seja 15 e 5. Vou salvar isso aqui, Ctrl+S. Vou escolher o diretório. Eu estou usando a pastinha com meu nome aqui, meu sobrenome é Aniche. Vou criar uma pasta que eu vou chamar de “primeiro-js”, para facilitar. E vou chamar esse cara aqui de “teste.html”. Ele até coloriu pra nós. Uma boa ideia. No próximo você vai ver que eu vou fazer isso, já vou salvar arquivo desde o começo, porque tudo fica colorido, mais fácil de ver.

[07:00] Estou no meu browser, Command+O, file open, para abrir o arquivo. Vou achar o meu arquivo teste HTML. Vou abrir ele. Vamos lá. Eu vou abrir um inspector do Chrome, vou pro console. Dá uma olhada. Ele imprimiu 15 e 5. Está funcionando. O meu código está funcionando, é o que apareceu aqui. Agora, a pergunta que vou fazer para vocês é: bem, programamos, escreveu um teste, uma main que invocou o nosso código e deu certo. Será que eu posso colocar esse código em produção?

[07:41] Será que ele realmente funciona? Não tem nenhum outro problema nele? Veja só o teste que eu vou fazer aqui. Ao em vez de 5, 15, 7 e 9, eu vou passar para ele 7, 6, 5 e 4. Então, eu espero que o maior seja 7 e o menor seja 4. Vou voltar pro meu browser. Vou dar refresh na página e dá uma olhada. O menor ele colocou quatro, ele acertou. Só que olha o maior, ele colocou um número doido aqui, 5 e a menos 324. Não era o que eu estava esperando.

[08:13] Eu estava esperando que o maior fosse 7, igual eu passei aqui para ele. Só que veja só, esse número aqui, 5 vezes dez elevado a menos 324, é exatamente o nosso “MIN_VALUE”. Ou seja, o maior nunca fui substituído. Por quê? Dá uma olhada no meu código, esse else aqui, que eu coloquei de propósito para forçar o erro, é um bug. Eu tirei o else e veja só, funcionou, mas o ponto é que quando eu estava programando, e isso acontece no mundo real, você está programando um algoritmo que é complicado, que te dá trabalho, você escreve código errado sem querer.

[08:47] Você não está pensando em todas as possibilidades. Acontece. Como eu vou descobrir isso? Testando. Eu tenho que testar software. E como que eu tenho que fazer isso? De maneira automatizada. O teste manual é chato, é demorado, é suscetível ao erro, é por isso que ninguém testa, porque não dá para ficar pagando um desenvolvedor ou um testador pro tempo inteiro testar o programa inteiro, que você sabe que, na prática, toda vez que você muda qualquer coisa no seu código você tem que testar ele do começo ao fim. Tudo de novo.

[09:19] Até porque mexeu em A você quebra B, e tudo mais. A ideia então, é fazer um programa que testa o seu programa. Este código é quase um teste automatizado, porque eu tenho um cenário aqui, 7, 6, 5 e 4 é um cenário, e toda vez que eu testo o que eu faço é pensar em cenários. Eu quero ver o que acontece com a minha aplicação se eu passar números de ordem crescente, números em ordem decrescente, números em qualquer ordem, uma lista com um único elemento e assim por diante.

[09:47] Eu executo uma ação e eu valido uma saída. Isso aqui é quase automatizado, porque veja só, o cenário está automatizado. Depois que eu escrevi 7, 6, 5 e 4 é automático. Invocar o método encontro também, meu programa sempre vai invocar o método encontro. O único problema é a saída. Porque ainda depende de um ser humano olhar para ver se está funcionando.

[10:09] O que eu poderia fazer para corrigir é o seguinte, vamos lá. O “pegaMaior” eu sei que é 7, e o “pegaMenor” tem que ser 4. A. Se eu rodar false true, era pra imprimir true e true, se tudo tivesse funcionado. Não funcionou. Porque ainda tem esse else aqui. Deixa eu tirar. Se eu rodar agora é true e true.

[10:36] Veja só, eu escrevi um código agora que é totalmente automatizado, tem um cenário automatizado, uma invocação de método automatizado e um teste que é automatizado, porque o meu computador sabe que agora o “pegaMaior” tem que ser 7, e que o “pegaMenor” tem que ser 4. É o que eu espero. Isso é um teste automatizado.

[10:55] Só que isso é ruim ainda, porque imagina só um programa de verdade vai imprimir um milhão de trues, um milhão de falses, você não vai saber muito bem o que deu false, o que deu true. E é aí que vai entrar o framework de teste automatizado, que é o Jasmine.

[11:11] O que o Jasmine faz pra nós? Ele facilita não só a escrita do teste, mas essa saída final, se der um erro, ele vai mostrar onde deu erro para mim, qual foi o teste que falhou. Vou mostrar para vocês. O Jasmine é uma biblioteca JavaScript, como qualquer outra. Você baixa no site do Jasmine. No primeiro exercício do curso você vai ter a opção para baixar, o link certinho. Você vai baixar o arquivo, Jasmine standalone 2.0. Aqui eu tô usado a versão 2.00. Talvez quando você fizer esse curso tenha uma versão mais recente, se ainda é 2 está valendo.

[11:46] Vou deszipar ele aqui. Ele deszipou pra essa pasta Jasmine standalone. E dá uma olhada, aqui está a biblioteca, e essa pastinha lib é toda a biblioteca do Jasmine, que pra nós não importa muito. É uma biblioteca, não vou mexer nela. Para nós se importam três coisas, a pasta source, a pasta spec, e o SpecRunner. Spec é como ele chama o teu teste automatizado, ele chama de spec. É bastante comum, tem bastante framework que faz isso. Source é onde vai estar o nosso código fonte.

[12:20] SpecRunner é um cara bonitinho que vai mostrar pra nós a interface. Então, vamos lá. O que eu vou fazer é abrir o SpecRunner. Vou clicar duas vezes nele, e dá uma olhada. Ele já rodou. O Jasmine e já vem com alguns códigos de exemplo, ele tem a classe Player. Ele executou esse teste, "should be able to play a song", e assim por diante, cada um desses verdinhos aqui que estou passando o mouse é um teste que ele rodou.

[12:44] Vamos abrir esse código no sublime para dar uma olhada. Aqui eu vou abrir a pasta inteira no sublime para facilitar. A classe player é uma classe do JavaScript convencional, um código de JavaScript, e eu vou abrir aqui um player spec, que é o quem ele rodou. Esse aqui é um código do teste. Nesse momento, não assusta, não tenta entender, vamos escrever isso do zero. Por que ele rodou esse player spec? Dá uma olhada no SpecRunner.

[13:10] O que ele faz aqui não tem segredo. Veja só que aqui eu estou importando as bibliotecas do Jasmine, tem os JavaScripts, tem o CSS pras coisas ficarem bonitas. Aqui são as minhas bibliotecas, e em seguida, os specs. O que vamos fazer é escrever os nossos specs. Então, a primeira coisa que eu vou fazer é pegar aquele nosso código maior e menor e jogar aqui dentro. Eu vou criar aqui um arquivo que eu vou chamar de “MaiorEMenor.js”.

[13:34] O que vai ter nele? Aquele nosso código fonte do “MaiorEMenor”. Vou dar um Ctrl+c nele aqui e vou colar no outro projeto. E vou batizá-lo de “MaiorEMenor.js”. No SpecRunner agora eu vou incluir esse cara. Então, eu vou copiar essa linha do include, e source “MaiorEMenor.js”. Meu código fonte está lá, agora eu quero testar. Na pastinha spec eu vou escrever o arquivo que eu vou chamar de maior e MaiorEMenorSpec.js.

[14:13] Como que começa a escrever esse teste? É a API do Jasmine, e é o que eu vou mostrar para vocês agora. Todo começo de teste tem o que o Jasmine chama de "describe", o describe você vai passar para ele uma informação em português, por exemplo, em língua natural, que vai explicar o que estamos testando. Geralmente, nós colocamos o nome da classe, então, estou testando aqui o “MaiorEMenor”. É o meu teste.

[14:38] O describle ele recebe uma função. Function. Dentro desse describle é onde eu vou escrever os meus testes, e por testes entenda cada possível cenário do meu código. Então, o primeiro teste que eu vou fazer é um teste que eu vou chamar de "deve entender números em ordem não sequencial". Veja só que o nome do meu teste já deixa bem claro o cenário que eu vou passar, "números em ordem sequencial".

[15:03] E veja o nome do método que eu usei aqui, eu usei it. Por que it? Porque se você pegar o código em inglês geralmente o pessoal vai escrever "it should e blá blá blá", então, "o código deve tal coisa", por isso eu usei o deve aqui. Em inglês ia ser “it should”, eu vou programar em português, não tem problema. Então, esse primeiro texto aqui é uma explicação do teste. Eu geralmente uso para deixar claro o cenário que eu estou testando. "Deve entender números em ordem não sequencial". Essa é uma outra função. "Function".

[15:37] Aqui dentro agora eu vou programar. Eu vou programar exatamente o código que eu fiz antes, vamos lá, é a mesma. “var algoritmo = new MaiorEMenor()”, “algoritmo.encontra” eu vou passar aquela lista 5, 15, 7 e 9, e eu sei que a saída é 15 pro maior e 5 pro menor. Como eu vou fazer essa verificação? Antes era o console.log, e usamos igual a igual. Aqui eu vou usar a API do Jasmine. Como funciona?

[16:04] Eu vou sempre usar a palavra expect. E dentro do expect eu vou passar o que eu calculei, o algoritmo “pegaMaior”, eu espero que ele seja igual a 15. Dá uma olhada, o API é bem fácil de ler. Você entende que eu espero que o “pegaMaior” seja igual a 15, e eu espero que o “pegaMenor” seja igual a 5.

[16:27] É exatamente igual o outro código que fizemos. Dá uma olhada da linha 32 para baixo. Igual. A diferença é que em vez de “console.log”, eu usei o expect do Jasmine. Então, o teste automatizado é igual aquele código que você escrevia, dá um ew na sua classe, invoca a função que você quer testar e verifica se a saída é o que você quer, de acordo com o cenário que você passou.

[16:54] E o SpecRunner? Eu tenho que incluir essa especificação, “spec/MaiorEMenorSpec.js”, e agora eu vou rodar aqui. Dá uma olhada. Apareceu maior e menor. Eu aqui só para tirar, vou tirar as coisas das demonstrações do Jasmine que não precisamos. Vou deixar só o nosso. Então, “MaiorEMenor”, “MaiorEMenorSpec”. Roda de novo. Olha só, “MaiorEMenor”, deve entender números em ordem não sequencial. E ele está verde, geralmente, os frameworks pintam de verde quando o true dá certo. Se tivesse dado errado ele ia ter pintado de vermelho. Quer ver? Vamos ver um teste em vermelho aqui.

[17:32] Vamos escrever o próximo teste. Como eu faço isso? A mesma coisa "it" e o próximo cenário. "Deve entender números em ordem crescente". Function, “var algoritmo = new MaiorEMenor()”. “algoritmo.encontra”, vou passar para ele que será 5, 6, 7 e 8, e eu espero que algoritmo “pegaMaior”, seja "toEqual" a 8 e o menor é 5.

[18:15] Vou rodar. Tudo verdinho. Vamos forçar o bug, colocar um else aqui. Vamos ver... Tudo continua verde porque o bug era em ordem decrescente. Então, vamos fazer o mesmo teste para a ordem decrescente. Aqui eu vou até fazer um Ctrl+c pra não ter que digitar tudo de novo. Cuidado com o Ctrl+c, pra você não dar tudo certo. Em ordem decrescente aqui vai ser 8, 7, 6 e 5. Ele usa a palavra expect, mas o termo comum na área de teste é assert. Os assert são os mesmos. Eu vou rodar e dá uma olhada.

[18:49] Três testes, um falhou, e ele ai me falar qual. "Deve entender números em ordem decrescente", eu esperava que isso aqui fosse igual a 8 e não é, por algum motivo não é, bug do meu código. Ficou vermelho está errado. Volto pro meu código e, realmente, tem um bug. Corrijo o bug e rodo de novo. Estou dando refresh na página para rodar de novo. Não tem segredo.

[19:14] Tudo verde. Tudo verde me dá segurança, me diz que está tudo funcionando. E qual vantagem dessa segurança? A vantagem é que eu posso vir nesse código aqui e muda a implementação. Passo jogar isso aqui fora e implementar de outro jeito. Eu poderia ordenar lista de números, pegar o primeiro elemento como menor e o último elemento como maior. Uma outra implementação. Como que eu vou garantir que funciona? Eu vou vir aqui e vou rodar meu teste.

[19:37] Então, isso é um teste automatizado. Um teste automatizado é alguma coisa que dá um new no que eu quero testar, se eu estou usando classes de JavaScript, invoca o comportamento que eu quero testar passando um cenário em particular e dado o cenário, depois eu faço as asserções, as validações pra ver se meu programa funcionou. Isso é teste manual.

[20:03] Olha o temo que ele levou para rodar. Ele mostra ali no canto direito da tela. Zero pontos zero zero zero quatro segundos. Muito rápido. Muito mais rápido que um ser humano, imagina você testando isso aí de maneira manual, ia demorar demais. Então, isso é um teste automatizado. Veja só que não tem segredo, é fácil. Meu código JavaScript é simples, o teste é simples igual e você vê que ao longo do curso eu vou mostrar que o seu teste nunca vai ser muito diferente disso.

[20:30] Se seu código está bem escrito, seu teste sempre vai ser simples. Esse é o segredo. Então, nesse capítulo mostrei para vocês as vantagens do teste automatizado, ele roda rápido, ele me dá segurança na refatoração, ele não é tão chato igual é fazer um teste manual.

[20:46] E eu mostrei o Jasmine, como fazer ele funcionar. Baixei a biblioteca no site e saí escrevendo as specs, usando o describe, usando o it. Passando essas funções anônimas. Bem simples. Esse é o primeiro capítulo, mostrei para vocês o começo de um teste automatizado. Eu vejo vocês no próximo capítulo, que a coisa vai ficar mais legal. Obrigado.