Jasmine

Jasmine é um framework de código aberto para testes unitários, que não afeta a aplicação ou IDE. Ele suporta testes assíncronos, é compatível com Javascript e tem sintaxe simples.

Na página do Jasmine, há um exemplo da sintaxe Javascript usada para os testes. Confira a seguir.

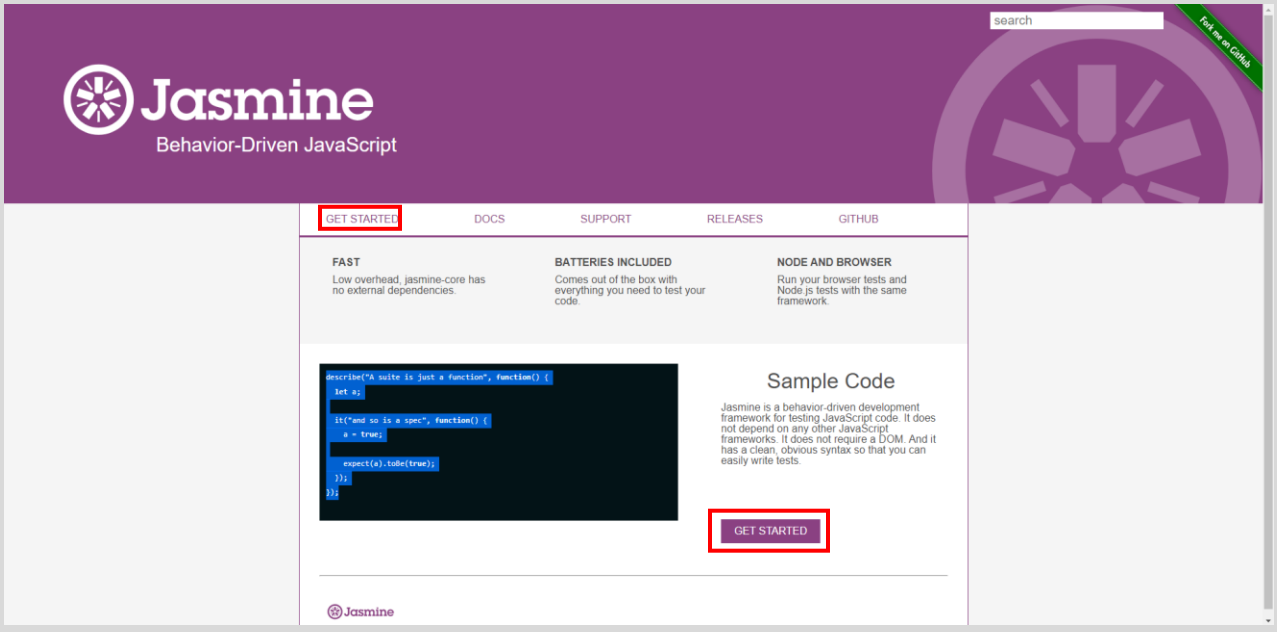
```
describe("A suite is just a function", function() {
  let a;

  it("and so is a spec", function() {
    a = true;

    expect(a).toBe(true);
  });
});
```

Nesse código, descrevemos o que será testado e o resultado esperado. Os resultados dos testes são renderizados de imediato em um navegador.

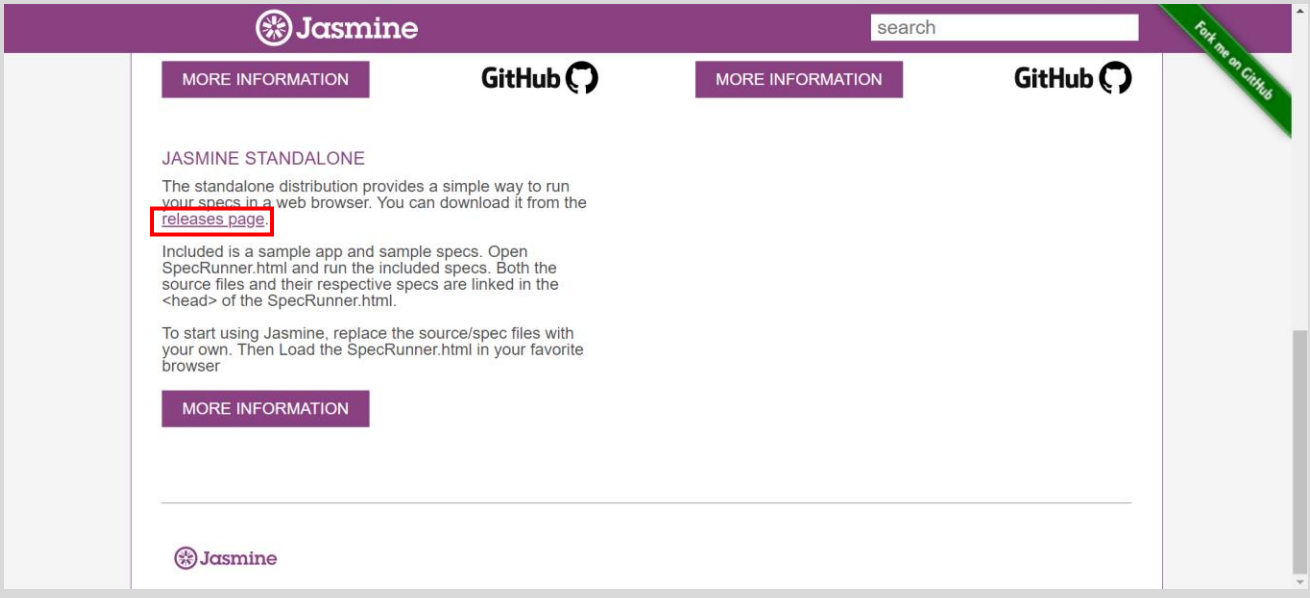
Há várias maneiras de instalar e executar o Jasmine. Nesse conteúdo, você vai acompanhar como usar a versão standalone, ou seja, a versão para desktop.

Instalação

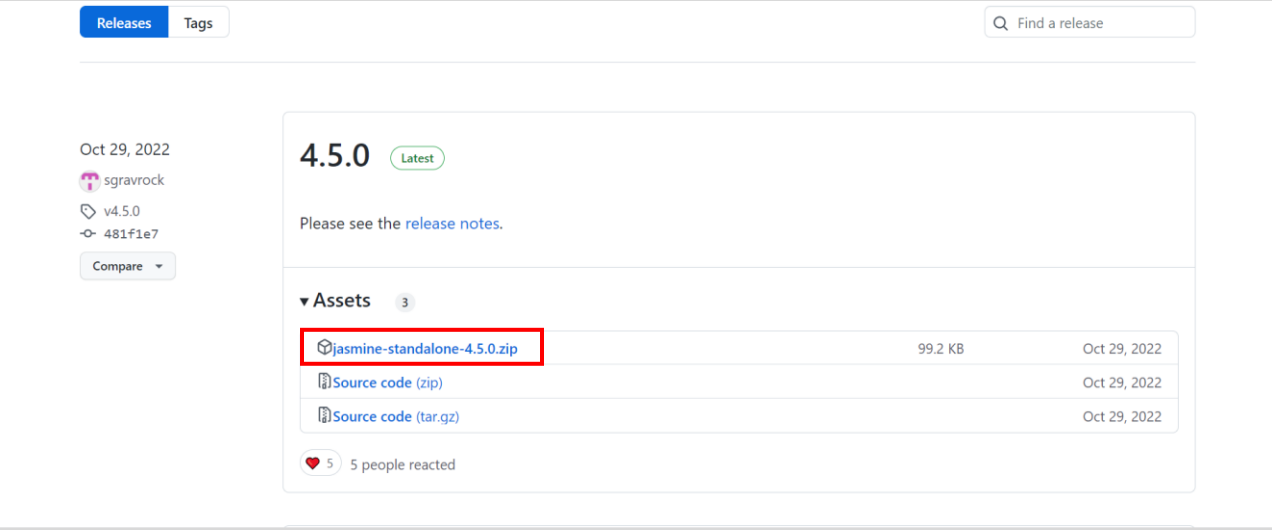
1. Acesse a documentação do Jasmine em **https://jasmine.github.io/** e clique em **GET STARTED**.



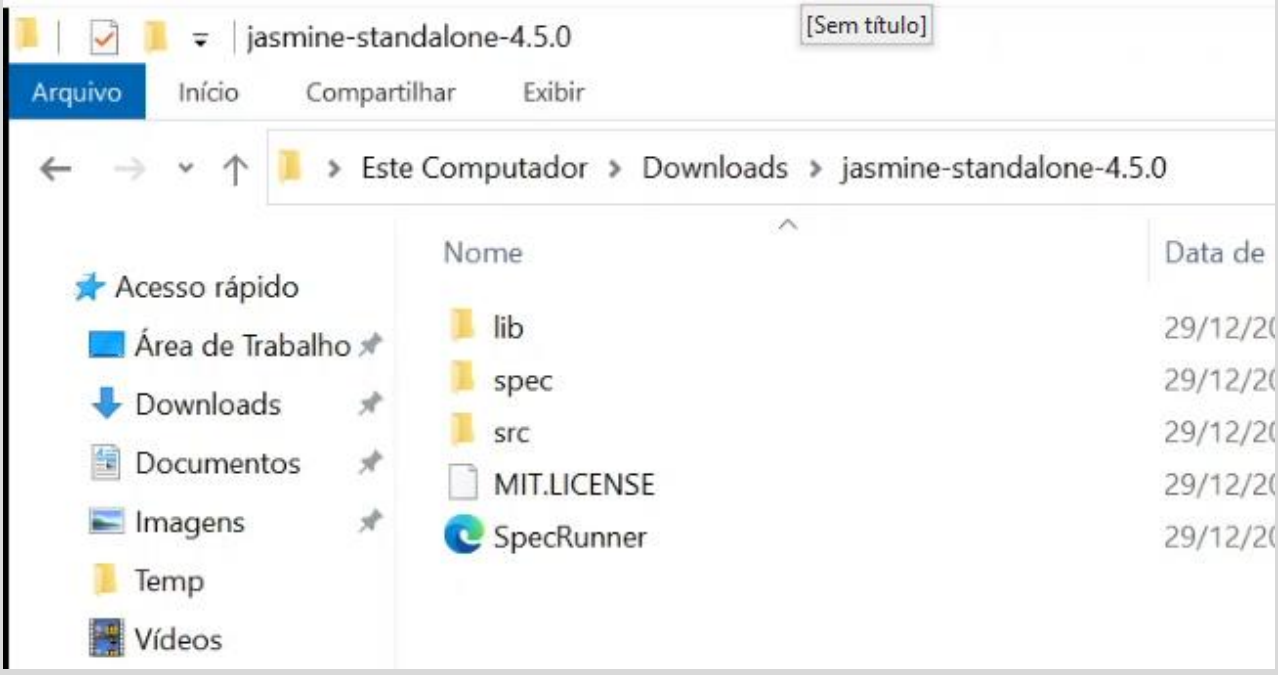
2. Na página seguinte, role a tela até encontrar **JASMINE STANDALONE**. Clique em **releases page**.



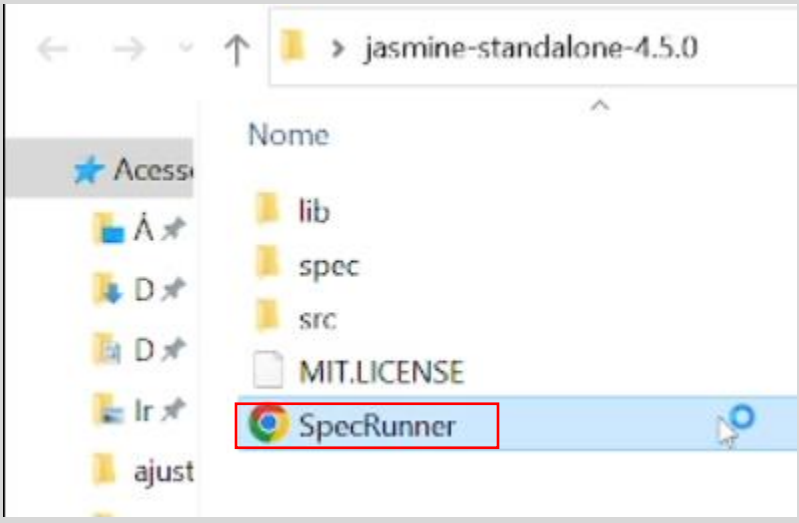
3. Encontre o arquivo compactado do **jasmine-standalone** e clique nele para fazer o download.



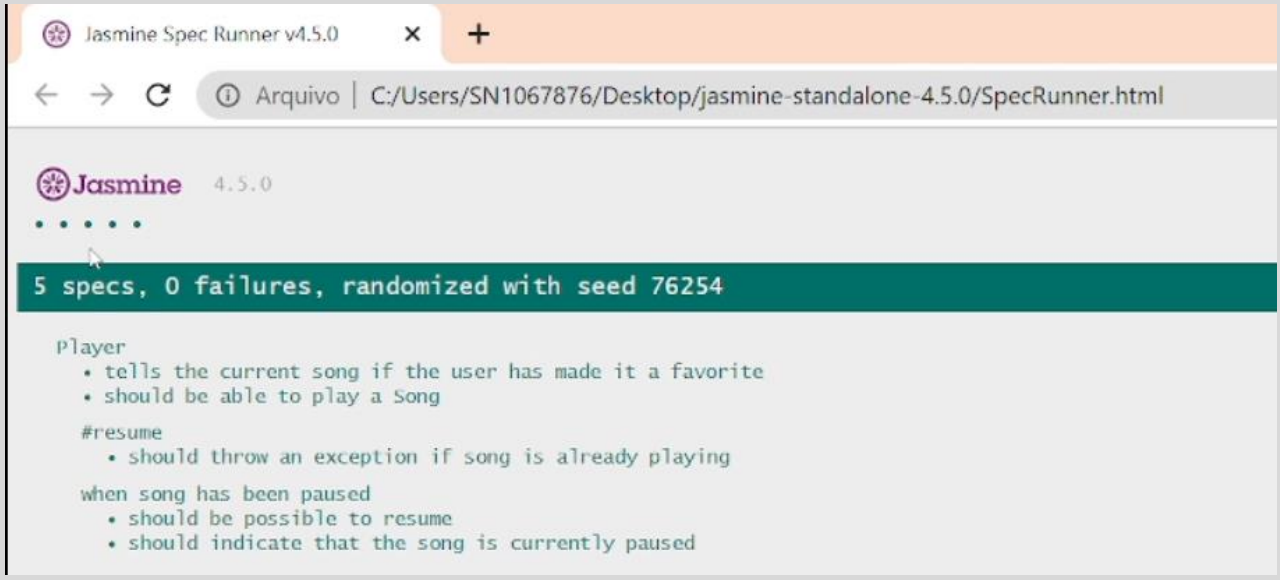
4. Descompacte o arquivo e abra a pasta dele.



5. Descompacte o arquivo salvo, abra a pasta e clique duas vezes em **SpecRunner**.

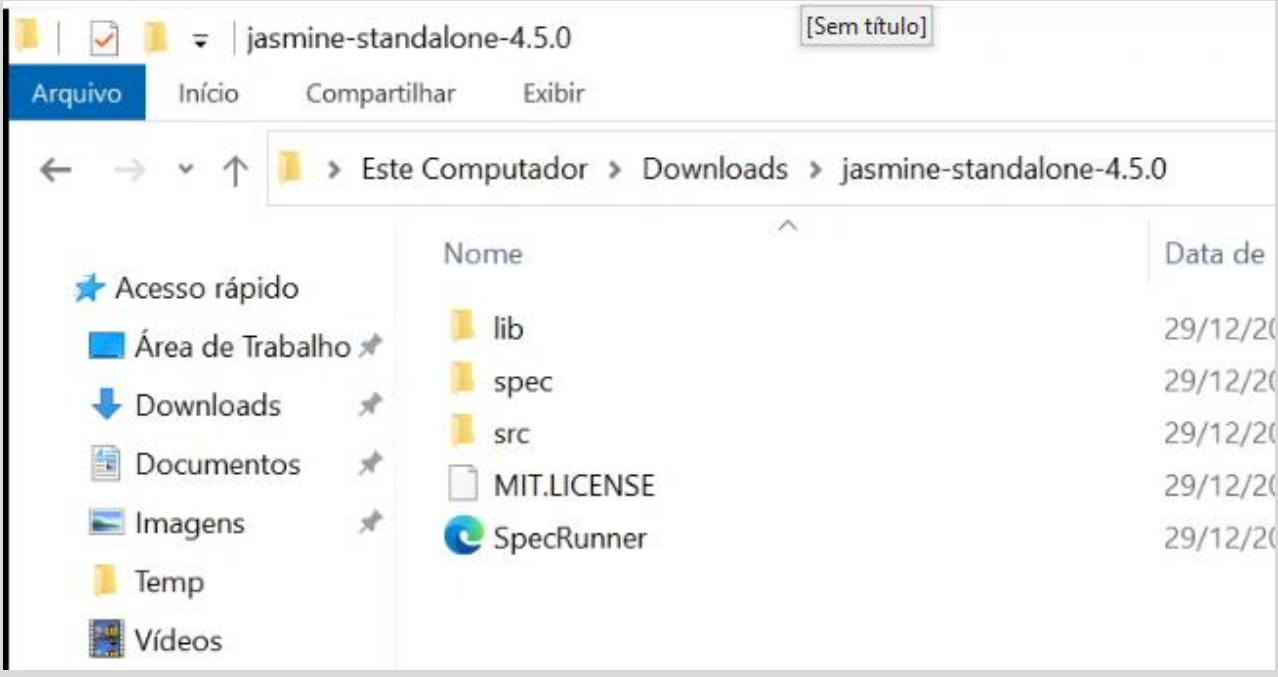


6. O Jasmine será executado no navegador.

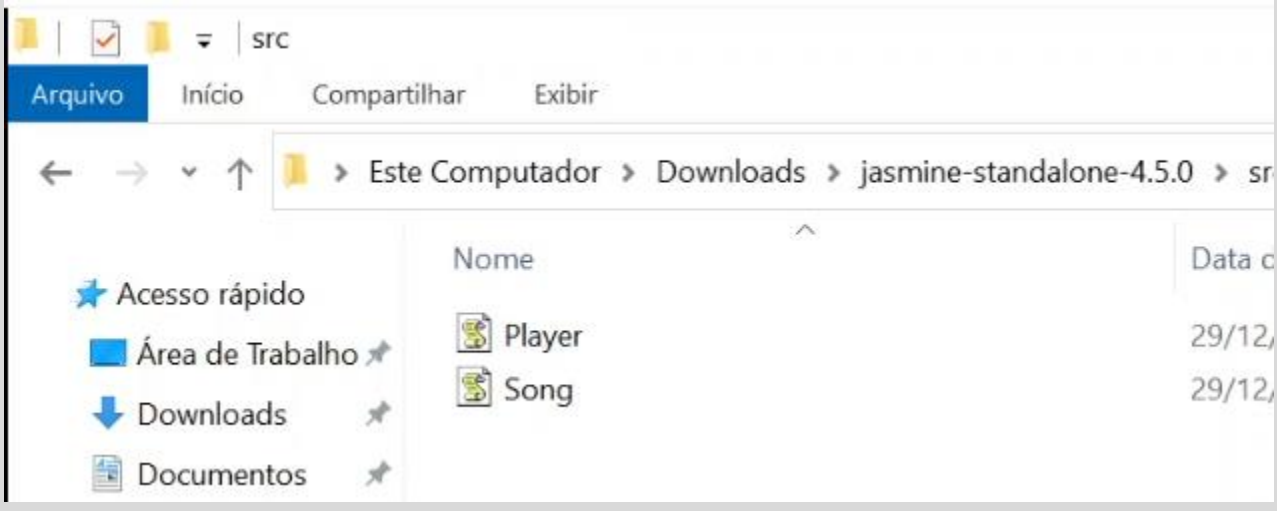


Uso

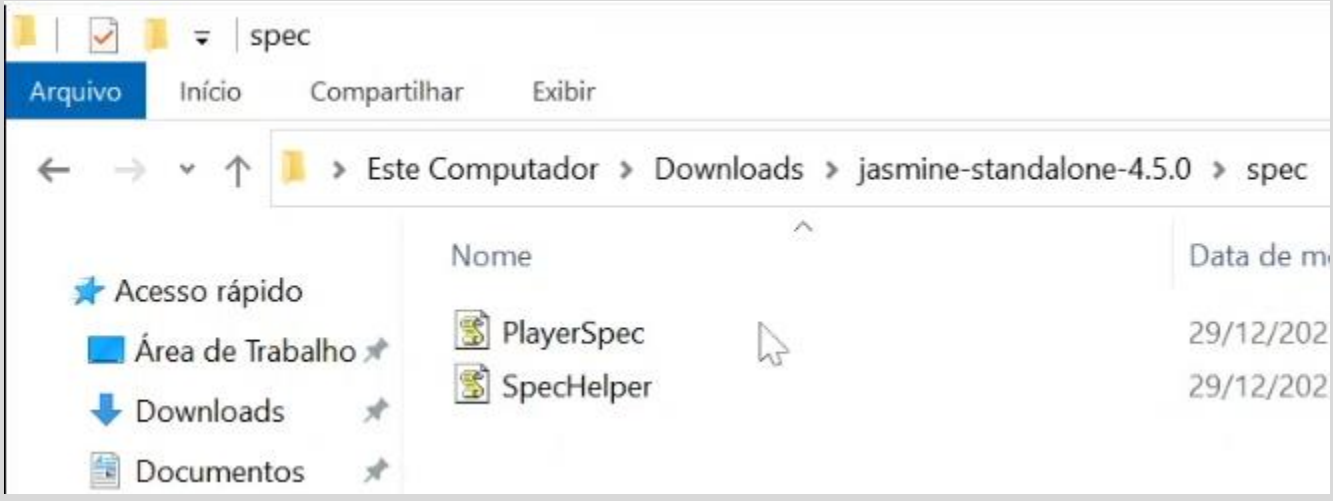
1. Abra a pasta **jasmine-standalone**.



Na pasta **src** estão os códigos Javascript que vamos testar. Dentro dela, há duas classes Javascript de exemplo, **Player** e **Song**.



Na pasta **spec** estão os testes propriamente ditos. Por padrão, nomeie os testes com o mesmo nome do arquivo testado, acrescentando "spec" no final. Observe o exemplo **PlayerSpec.js**.

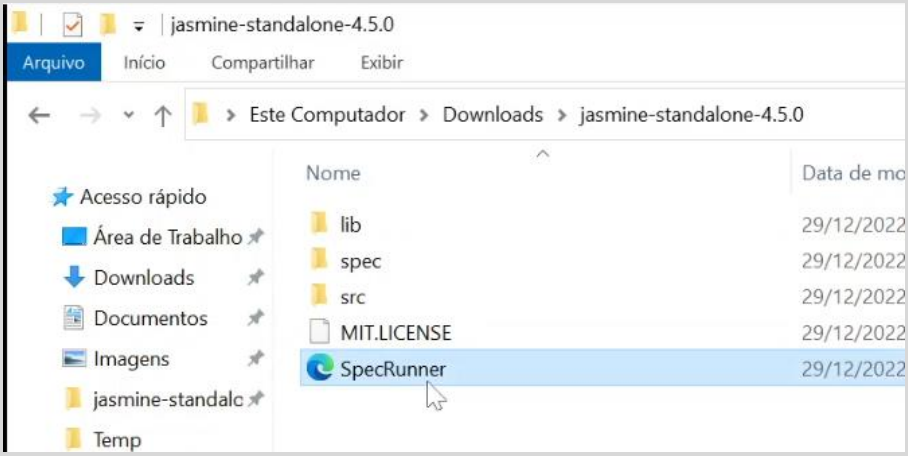


2. O html SpecRunner mostra o resultado do teste. Abra o arquivo e encontre o trecho a seguir.

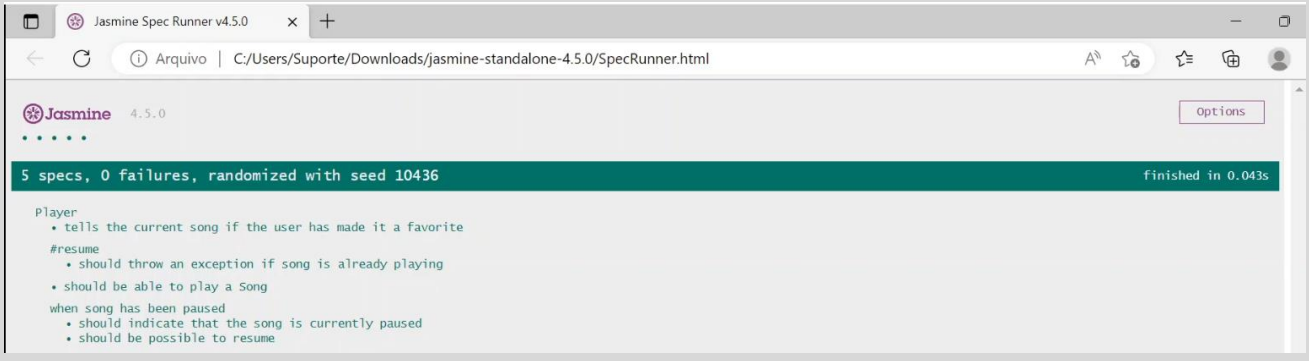
```
<!-- include source files here... -->
<script src="src/Player.js"></script>
<script src="src/Song.js"></script>
<script src="src/Soma.js"></script>

<!-- include spec files here... -->
<script src="spec/SpecHelper.js"></script>
<script src="spec/PlayerSpec.js"></script>
<script src="spec/SomaSpec.js"></script>
```

O primeiro bloco refere-se aos arquivos-fonte da pasta src, ou seja, os arquivos que serão testados. Já o segundo bloco refere-se aos arquivos de teste da pasta spec, ou seja, os testes propriamente ditos. Faça as alterações necessárias e salve o arquivo.

3. Retorne à pasta no explorador de arquivos e execute o SpecRunner.html com um duplo clique.



O resultado do teste deve ser como a imagem a seguir, em verde.



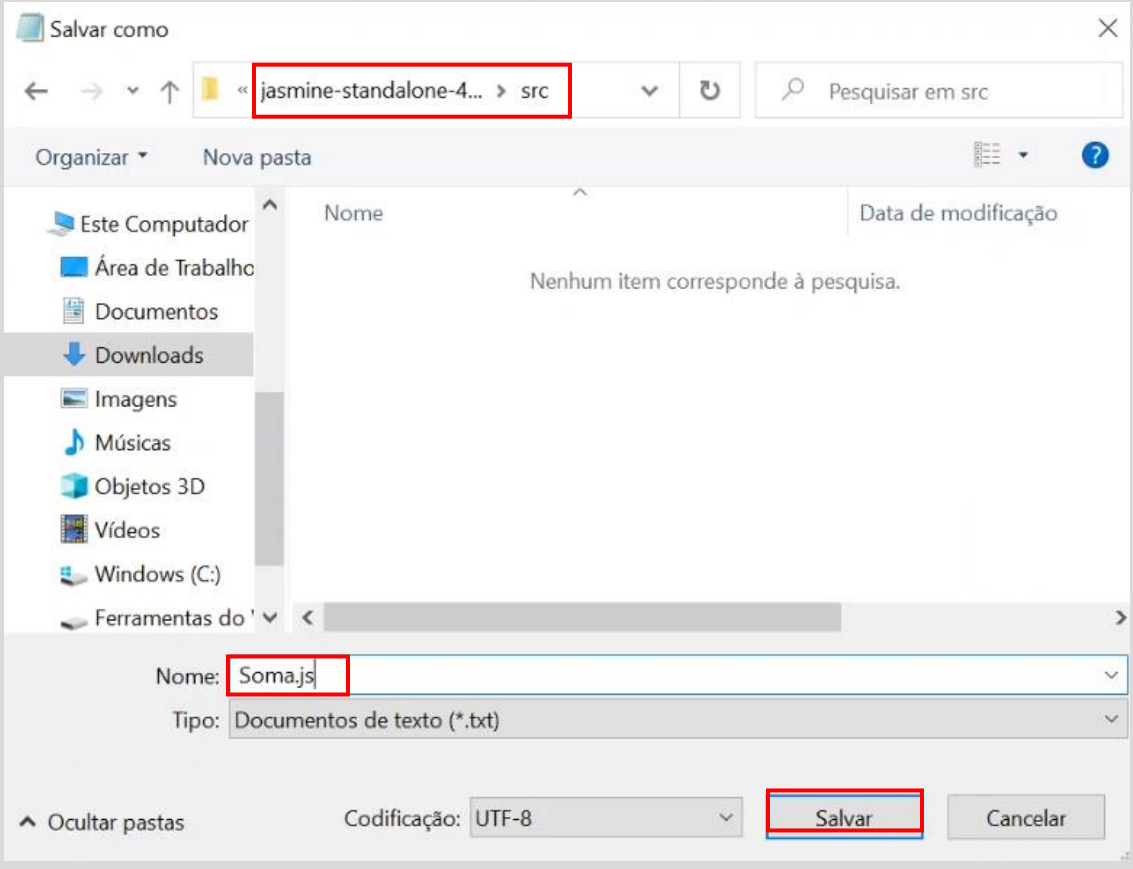
Quando o teste encontra erros, o resultado fica destacado em vermelho.

Teste de soma

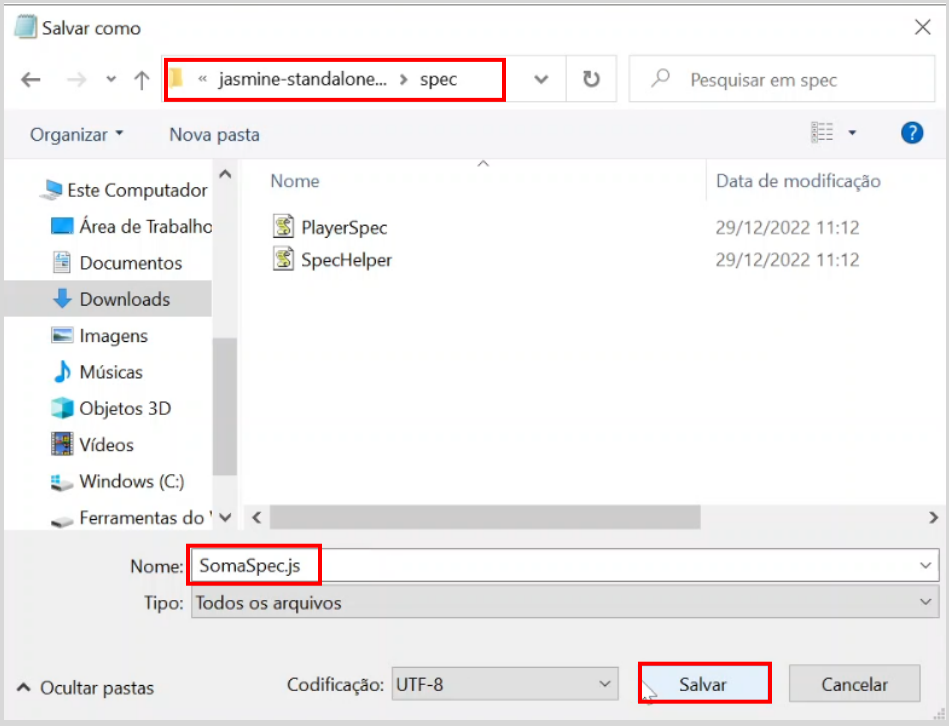
1. Abra o bloco de notas ou editor de código e digite a seguinte função de soma:

```
function soma(a,b)
{
    return a+b;
}
```

2. Salve como **Soma.js** dentro da pasta **src**.



3. Salve um arquivo em branco como **SomaSpec.js** dentro da pasta **spec**.



4. Abra o **SpecRunner.html** e ajuste os arquivos-fonte e arquivos **spec** conforme o seguinte:

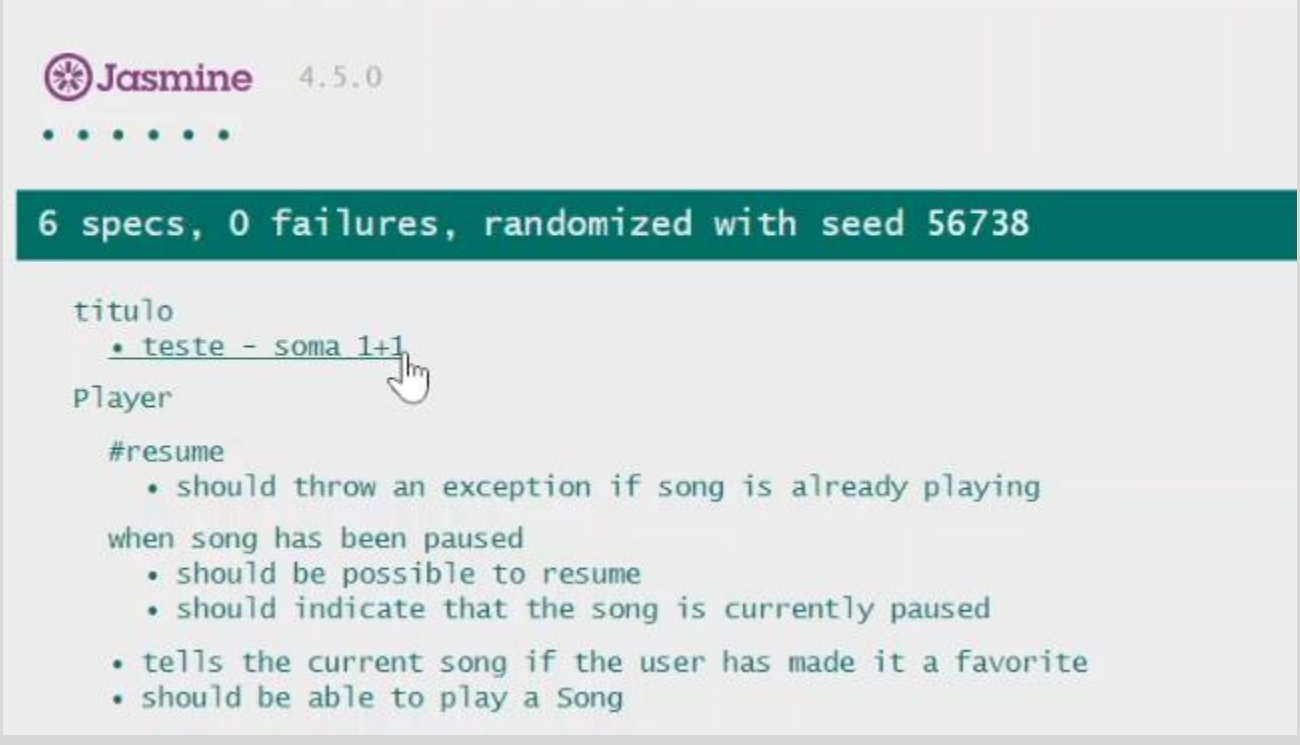
```
<!-- include source files here... -->
<script src="src/Player.js"></script>
<script src="src/Song.js"></script>
<script src="src/Soma.js"></script>

<!-- include spec files here... -->
<script src="spec/SpecHelper.js"></script>
<script src="spec/PlayerSpec.js"></script>
<script src="spec/SomaSpec.js"></script>
```

5. Abra o arquivo SomaSpec.js e digite o seguinte código:

```
describe("Teste do arquivo Soma", ()=>
{
    it('teste - soma 1+1', ()=>{
        expect(soma(1,1)).toEqual(2);
    });
});
```

Nesse teste, definimos que a função soma será testada e enviaremos o valor 1 como parâmetro. Esperamos, então, que a soma 1 + 1 seja igual a 2. Lembre-se de salvar as alterações.

6. Execute novamente o SpecRunner.html ou atualize a página no navegador. O resultado será como o da imagem a seguir:



7. Abra o arquivo SomaSpec.js e altere o valor **2** para **3**:

```
describe("Teste do arquivo Soma", ()=>
{
    it('teste - soma 1+1', ()=>{
        expect(soma(1,1)).toEqual(3);
    });
});
```

Nesse teste, vamos verificar se soma 1 + 1 é igual a 3.

Lembre-se de salvar as alterações.

8. Execute novamente o SpecRunner.html ou atualize a página no navegador. O resultado será como o da imagem a seguir:

O teste encontrou uma falha em **teste – soma 1+1**. A falha foi esperar que **2 seja igual a 3**.

9. Abra o arquivo SomaSpec.js e ajuste conforme o seguinte:

```
describe("Teste do arquivo Soma", ()=>
{
    it('teste - soma 1+1', ()=>{
        expect(soma(1,1)).toEqual(2);
    });
    it('teste - soma 1+2', ()=>{
       expect(soma(1,2)).toEqual(2);
     });
});
```

Nesse teste, corrigimos a primeira soma e acrescentamos outra. A segunda soma é 1 + 2 e esperamos que o resultado seja 2. Salve as alterações.

10. Execute novamente o SpecRunner.html ou atualize a página no navegador. O resultado será como o da imagem a seguir:

O teste encontrou uma falha em **teste – soma 1+2**. A falha foi esperar que **3 seja igual a 2**.

11. Clique em **Spec List** para conferir o mesmo resultado de outra maneira.

Nessa visualização, apenas os títulos são listados. Os que estão em verde não retornaram falha e o em vermelho retornou falha.

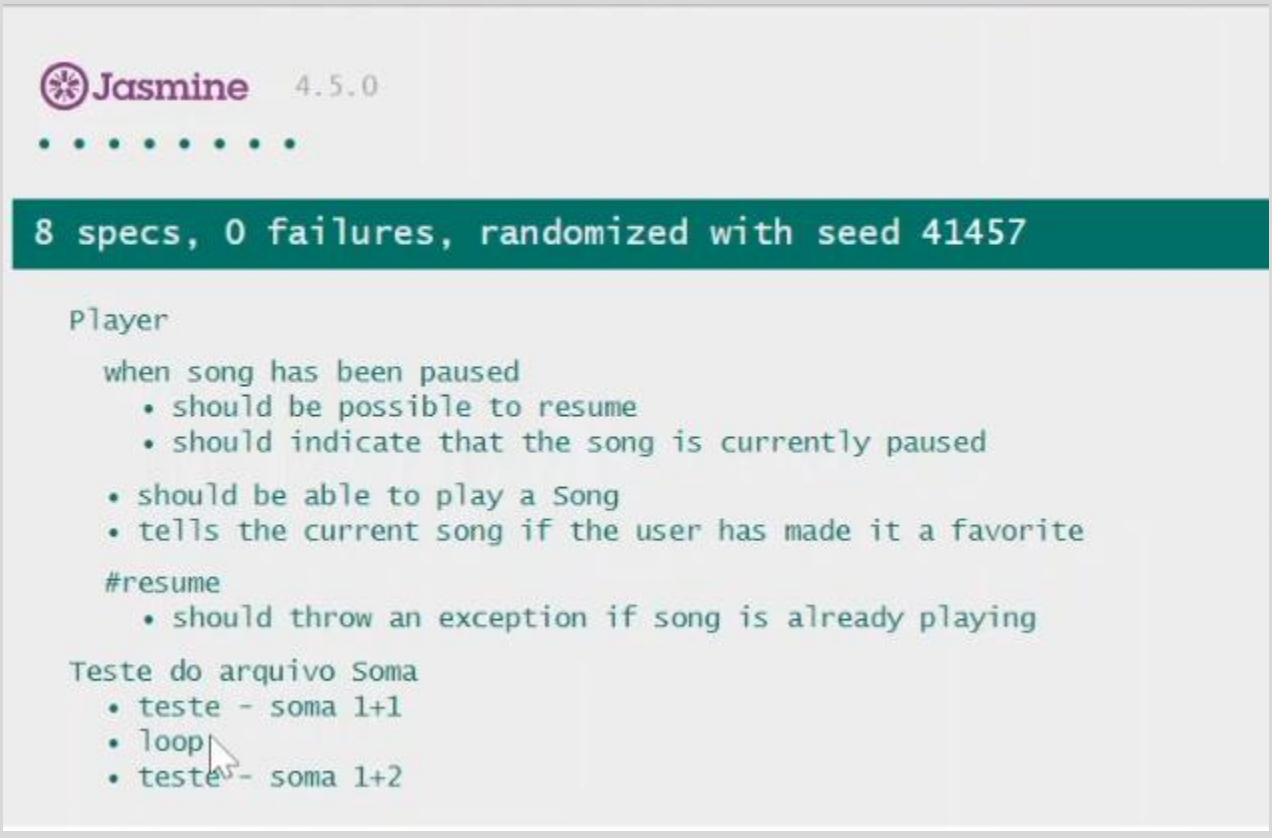
12. Abra o arquivo SomaSpec.js e ajuste conforme o seguinte:

```
describe("Teste do arquivo Soma", ()=>
{
     it('teste - soma 1+1', ()=>{
          expect(soma(1,1)).toEqual(2);
     });

     it('teste - soma 1+2', ()=>{
          expect(soma(1,2)).toEqual(3);
     });

     var n1=[1,2,3];
     it('loop', ()=>{
          for(var i=0;i<3;i++)
               expect(soma(n1[i],1)).toEqual(n1[i]+1);
     });
});
```

Nesse código, ajustamos o valor esperado para soma 1+2 para 3.

Além disso, criamos a lista n1 com os valores 1, 2 e 3. Criamos um laço de repetição e o nomeamos como **loop**. Dentro do loop, definimos a variável i iniciando em 0 e, enquanto i < 3, o valor de i aumenta 1. No teste, esperamos que a soma de cada valor da lista com 1 seja somado a n1[i]+1.

13. Execute novamente o SpecRunner.html ou atualize a página no navegador. O resultado será como o da imagem a seguir:



O laço criado não retornou erro.

Saiba mais

Esse tipo de programação é do tipo «paradigma de programação orientado a aspectos». Nesse tipo de paradigma, um programa analisa um aspecto pré-determinado de outro programa.

O objetivo desse tipo é separar componentes e aspectos de forma concisa, usando mecanismos de abstração e composição e, assim, permitindo a análise, teste ou validação de um código ou programa.

Além dos testes unitários, são considerados exemplos de programação orientada a aspectos: logs, cache, indexação de dados, entre outros.