02

Lidando com HTML e Interfaces

Até agora, você deve estar se pensando: mas e a interface? Nos meus Javascripts, eu lido o tempo todo com os elementos HTML para pegar valores, preencher valores, e etc. Como faço para testá-los?

Esse é um dos maiores problemas quando pensamos em testabilidade. Independente de ser Javascript ou não, sempre que você escreve código que mistura regra de negócio com infraestrutura, a dificuldade de testar aumenta. Se nosso código acessasse um banco de dados, ou usasse alguma API maluca do Java, o problema seria o mesmo. A grande sacada aqui é então, não aprender a como testar, mas sim como escrever código que facilite o teste. **Você verá que quanto mais entender de teste, mais você precisará escrever código independente de infraestrutura.**

Veja, por exemplo, o trecho de código abaixo:

```
var peso = $("#peso").val();
var altura = $("#altura").val();

var imc = peso / (altura * altura);

$("#resultado").val("O IMC é " + imc);
```

O código acima é bastante comum em qualquer aplicação web. Generalize-o: ele pega dados da interface, invoca alguma regra de negócio e manipula de novo elementos da interface para exibir os dados. **A pergunta é: como testar esse método?** Afinal, precisamos ter elementos HTML para que o código funcione. E depois, como garantir que o comportamento foi como esperado?

Não há uma resposta fácil pra isso. Sempre que seu teste depende de muitas coisas, ele é mais difícil. A solução? Fazer seu código (e por consequência, o teste dele) depender de menos coisas. Para Javascript, em particular, é isso que aqueles frameworks MVC/MVVM, como AngularJS, fazem. Eles separam código que lida com a view do código que tem a regra de negócio. Se você já programou server-side e usou algum framework MVC (Spring MVC, Rails, etc), sabe que é assim também.

Aqui a mesma coisa. A ideia é isolar todo o código de interface possível. No exemplo que demos, todo o código que faz uso de JQuery ali, deve estar em outro lugar. Afinal, é ele que atrapalha o teste. Veja que o problema não é o JQuery, mas sim a manipulação de elementos.

Para facilitar o entendimento, aqui não usaremos nenhum framework como AngularJS, e faremos tudo na mão, para que você entenda o conceito. Precisamos isolar o processo de acessar a interface. Veja só o código abaixo:

```
function TelaDePacientes() {


    var clazz = {
        pegaPaciente : function() {
            return new Paciente(
                $("#nome").val(),
                $("#idade").val(),
                $("#peso").val(),
                $("#altura").val()
            );
```

```
        },

        exibeIMC : function(imc) {
            $("#resultado").val("O IMC é " + imc);
        }
    };

    return clazz;
}
```

Nessa classe, encapsulamos todo o processo de manipulação daquela interface. Agora, precisamos conectar os dois códigos: o código da nossa entidade `Paciente`, com o código que manipula a tela:

```
var ui = new TelaDePacientes();

var paciente = ui.pegaPaciente();
var imc = paciente.imc();

ui.exibeIMC(imc);
```

Ótimo. Chegamos onde queríamos. Agora vem a discussão filosófica.

Testar o método `imc()` na classe `Paciente` é fundamental. Afinal, é uma regra de negócio importante. E isso já testamos aqui nesse curso, sem grandes dificuldades.

A pergunta é: e o resto? Os outros códigos talvez não mereçam um teste de unidade. Eles são código de "integração", ou seja, eles integram camadas diferentes. Escrever um teste de unidade, com Jasmine, para eles não traria grandes benefícios. Mas precisamos garantir que ele funcione. Como fazer? Nesses casos, a sugestão é escrever um teste de sistema pra essa tela. Ou seja, um teste com Selenium, onde o browser abre, e cliques são simulados.

Mesmo que esse curso seja um curso de Javascript, vamos mostrar um pequeno trecho desse teste, em Java, usando Selenium. Veja o código abaixo:

```
public void buscaNoGoogle() {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.google.com.br");

    driver.findElement(By.name("q")).sendKeys("Alura");

    System.out.println(driver.getPageSource());

    driver.close();
}
```

No código acima, entramos na página do Google, encontramos a caixa de texto (que, no caso do Google, chama-se `q`), preenchemos, e depois exibimos o código-fonte. Esse código vai testar o site do Google funcionando como um todo, com todos seus javascripts, e lógicas de negócio.

As pessoas se confundem muito com isso. Quando elas aprendem a escrever testes, elas querem testar todo o sistema de maneira igual. Mas nosso código é diferente, e precisa ser testado de maneira diferente. Toda regra de negócio deve ser testada de maneira isolada, com Jasmine. Todo código de integração merece ser testado de maneira integrada.

Neste curso, não faremos esse tipo de teste. Se estiver interessado, faça o curso de Selenium, da nossa formação de testes. Mas leve daqui a ideia de separar ao máximo as responsabilidades. Isso facilitará a escrita dos testes das suas regras de negócio. É assim que fazemos em qualquer linguagem!