03

Validando o restante dos campos

Transcrição

Se o nosso objetivo é adicionar diversas mensagens de erros, uma para cada erro que surgir para o array, teremos que acessar a função do `form.js` e não apenas alterar o conteúdo da `<ul>`. Será necessário criar uma `<li>` para cada erro e, depois, adicioná-la na `<ul>`.

Vamos analisar como está o `form.js`:

```
var botaoAdicionar = document.querySelector("#adicionar-paciente");
botaoAdicionar.addEventListener("click", function(event){
    event.preventDefault();

    var form = document.querySelector("#form-adiciona");
    var paciente = obtemPacienteDoFormulario(form);
    var pacienteTr = montaTr(paciente);

    var erros = validaPaciente(paciente);

    if (erros.length > 0){
        var mensagemErro = document.querySelector("#mensagem-erro");
        mensagemErro.textContent = erros;
        return;
    }
```

Como esta lógica pode ficar complicada, iremos extrair a parte de exibição das mensagens de erro para a função `exibeMensagensDeErro()`. Ela será responsável por pegar o array de erros e, para cada item, criaremos uma `<li>` a ser adicionada na `<ul>` de `mensagens-erro`. Observe que faremos uma pequena alteração no `id`, em `index.html`:

```
<section class="container">
    <h2 id="titulo-form">Adicionar novo paciente</h2>
    <ul id="mensagens-erro">

    </ul>
```

No arquivo `form.js`, se queremos exibir a mensagem de erro, antes teremos que criar a função `exibeMensagensDeErro(erros)`, recebendo o array de mensagens de erros. Vamos adicioná-la logo acima da função `obtemPacienteDoFormulario()`:

```
function exibeMensagensDeErro(erros){

}
```

Para cada item do array, selecionaremos a `ul`, que será guardada em uma variável:

```
function exibeMensagensDeErro(erros){
    var ul = document.querySelector("#mensagens-erro");
}
```

E para cada item do array, adicionaremos a tag `<li>`. Poderemos fazê-lo de diferentes formas, como usando o `for` tradicional:

```
function exibeMensagensDeErro(erros){
    var ul = document.querySelector("#mensagens-erro");
    for(var i = 0; i < erros.lenght ; i++){
        var erro = erros[i];
    }
}
```

Além do `for`, existe outro tipo de iteração, o `forEach()`, o qual não precisamos delimitar, e que passará por todos os elementos. Para cada item do array, será realizada uma ação.

```
function exibeMensagensDeErro(erros){
    var ul = document.querySelector("#mensagens-erro");
    erros.forEach(function(erro){
        var li = document.createElement("li");
        li.textContent = erro;
        ul.appendChild(li);
    });
}
<li></li>
```

Temos que falar que o texto da `<li>` será uma mensagem de erro, que está dentro do array de erros. Para termos acesso ao erro, recebemos o item de iteração dentro da `function()` e o `textContent` será o `erro`. Nós colocaremos `li` dentro `ul`, usando a função `appendChild()`.

```
function exibeMensagensDeErro(erros){
    var ul = document.querySelector("#mensagens-erro");
    erros.forEach(function(erro){
        var li = document.createElement("li");
        li.textContent = erro;
        ul.appendChild(li);
    });
}
```

A função que exibe a mensagem de erro é:

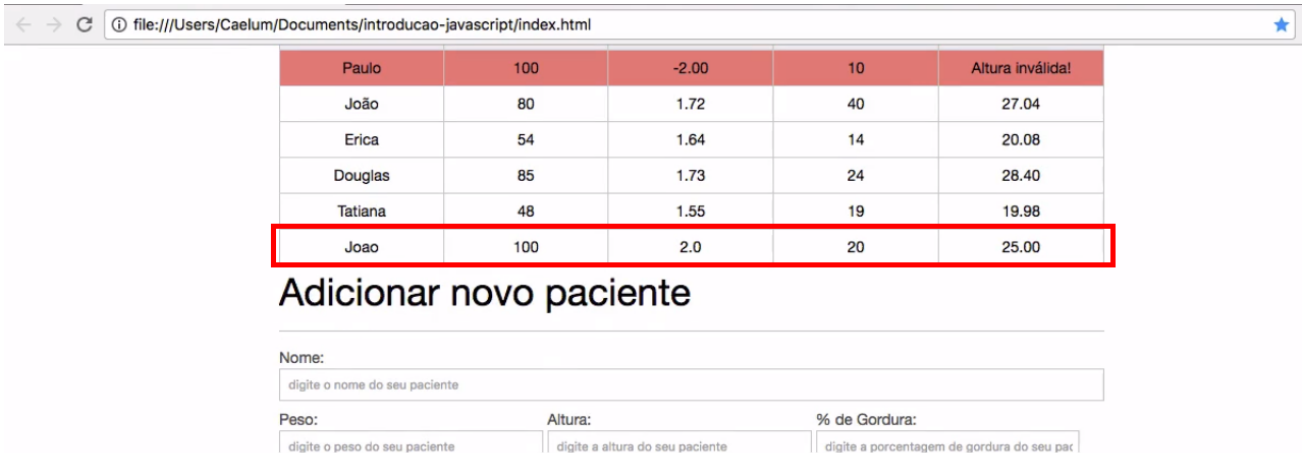
```
var erros = validaPacientes(paciente);
console.log(erros);
if(erros.length > 0){
    exibeMensagensDeErro(erros);
    return;
}
```

Adicionaremos o `console.log()` para vermos o array de erros caso ocorra algum problema.

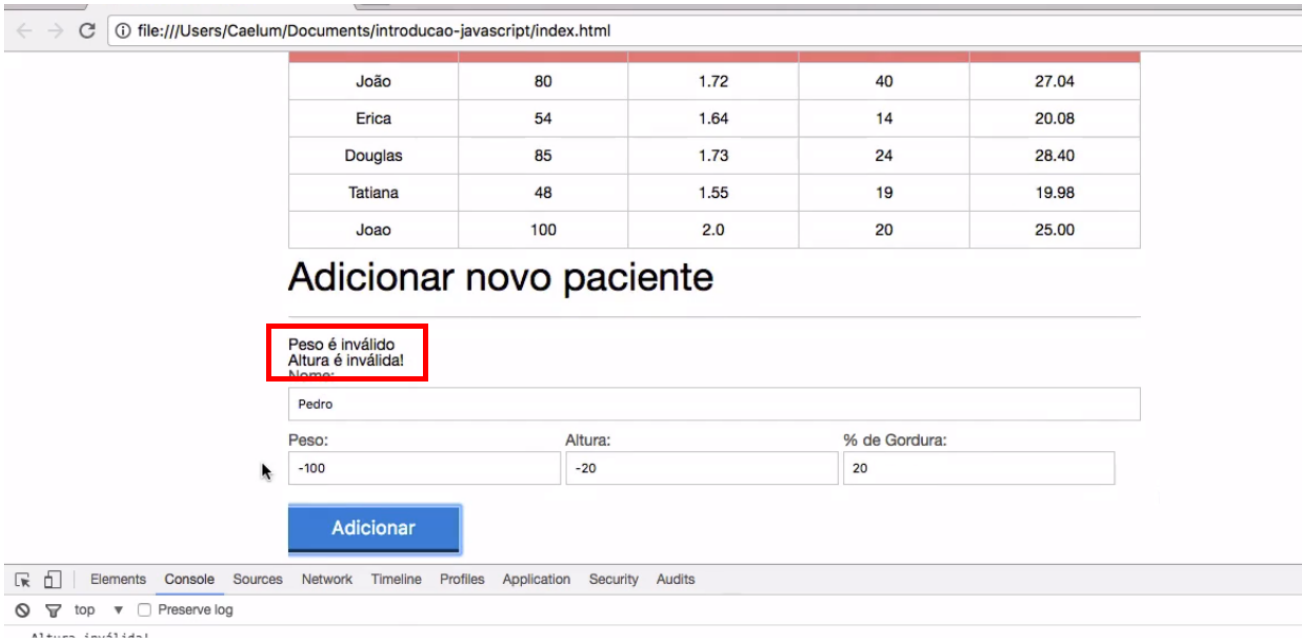
No `index.html`, verificaremos que havíamos adicionado uma tag `<ul>`:

```
<section class="container">
    <h2 id="titulo-form">Adicionar novo paciente</h2>
    <ul id="mensagens-erro">
    </ul>
//...
```

Em seguida, vamos preencher o formulário no browser. Se ocorrer algum erro, ele será exibido no console. Se cadastrarmos um paciente com os dados corretos, será impresso um array vazio.



Vamos testar adicionar outro paciente com dados inválidos, usando valores negativos.



Será impresso um array com as mensagens de erro no console, além disso, o novo paciente não foi adicionado na tabela, e duas mensagens foram exibidas acima do formulário. Se inspecionarmos essas mensagens, veremos que a `<ul>` está com `mensagens-erro` e duas tags `<li>`:

```
<h2 id="titulo-form">Adicionar novo paciente</h2>
<ul id="mensagens-erro">
    <li>Peso é inválido</li>
    <li>Altura é inválida!</li>
</ul>
```

A cor da fonte da mensagem não está vermelha pois mudamos o nome do `id`. Vamos ajustar isso no `index.css` e, agora, o id da `<ul>` é `mensagens-erro` e não mais `mensagem-erro`:

```
#mensagens-erro {
    color: red;
}
```

Adicionaremos um paciente com dados inválidos e as mensagens serão exibidas para o usuário com a fonte vermelha.

A verificação está funcionando corretamente e o paciente não foi adicionado na tabela.

Nosso código está todo organizado, isolamos as responsabilidades em diferentes funções, tornando simples a ação de adicionar uma nova validação. Atualmente, se não preenchermos o campo "Nome", o paciente será adicionado na tabela do mesmo jeito.

Como validamos o peso e a altura, vale a pena validarmos o nome. Vamos acessar a função `validaPaciente()` e adicionar um novo `if`. Para verificar se um campo está em branco, podemos analisar o seu tamanho (`length`), se ele for igual `0`, significa que o campo não foi preenchido. Por exemplo, o nome do paciente:

```
function validaPaciente(paciente) {

    var erros = [];

    if (paciente.nome.length == 0) {
        erros.push("O nome não pode ser em branco");
    }

    if (!validaPeso(paciente.peso)) {
        erros.push("Peso é inválido");
    }

    if (!validaAltura(paciente.altura)) {
        erros.push("Altura é inválida");
    }

    return erros;
}
```

Adicionaremos a mensagem `O nome não pode ser em branco` no caso de `length` ser igual a `0`. Faremos validações semelhantes nos demais campos:

```
function validaPaciente(paciente) {

    var erros = [];

    if (paciente.nome.length == 0){
        erros.push("O nome não pode ser em branco");
    }

    if (paciente.gordura.length == 0){
        erros.push("A gordura não pode ser em branco");
    }

    if (paciente.peso.length == 0){
```

```
            erros.push("O peso não pode ser em branco");
        }

        if (paciente.altura.length == 0){
            erros.push("A altura não pode ser em branco");
        }

        if (!validaPeso(paciente.peso)){
            erros.push("Peso é inválido");
        }

        if (!validaAltura(paciente.altura)){
            erros.push("Altura é inválida");
        }

        return erros;
    }
```

Em seguida, testaremos o envio do formulário com os campos todos em branco.

Veremos quatro mensagens de erros, e agora não conseguiremos mais adicionar pacientes com algum campo não preenchido.

Limpando as mensagens de erro

Ao tentarmos adicionar um paciente com algum dado inválido, a mensagem é impressa. Depois da correção e dos dados serem enviados, o paciente será adicionado à tabela, mas a mensagem continuará na página, em acúmulo.

Nós queremos limpar a lista de mensagens (`ul`), antes que as mensagens de erro sejam exibidas. Para esvaziar a `ul` , removeremos **todo o conteúdo HTML**. Para isto, utilizaremos a propriedade `innerHTML` , que nos permite controlar o HTML interno de um elemento. Passaremos uma string vazia para a propriedade:

```
function exibeMensagensDeErro(erros) {
    var ul = document.querySelector("#mensagens-erro");
    ul.innerHTML = "";

    erros.forEach(function(erro) {
        var li = document.createElement("li");
```

```
            li.textContent = erro;
            ul.appendChild(li);
        });
    }
```

Deste modo, sempre que as mensagens de erro forem exibidas, as anteriores serão apagadas, de forma a validarmos o formulário.

Faremos um pequeno ajuste para os casos em que adicionamos um paciente na tabela. Vamos limpar a `ul` no `form.js`. Na variável `mensagensErro`, usaremos `document.querySelector()`. Após a adição, vamos limpar as mensagens:

```
var tabela = document.querySelector("#tabela-pacientes");

tabela.appendChild(pacienteTr);

form.reset();

var mensagensErro = document.querySelector("#mensagens-erro");
mensagensErro.innerHTML = "";

});
```

Agora, quando adicionamos um paciente com sucesso, as mensagens de erro são limpas da tela.

Vimos nessa aula como validar um formulário, mostramos também a importância de organizarmos nosso código à medida em que ele vai crescendo. Demonstramos ser útil saber trabalhar com arrays, como fizemos no caso de exibirmos mais de uma mensagem de erro. É importante conhecermos as estruturas básicas do JavaScript - incluindo strings e arrays - e saber quando utilizar cada uma delas. Mostramos como fazer a validação com uma tag `<ul>` em vez de `<span>`. Nós estamos agrupando todo o conhecimento adquirido e seguiremos melhorando nossa aplicação!