04

Tratando a resposta da requisição

Transcrição

Os dados da requisição são exibidos no console, nós só temos que transformá-lo para um formato mais palpável, e depois, adicioná-los na tabela. Até o momento, as informações vêm no formato texto, todo concatenado. Poderemos averiguar qual o formato dos dados, adicionando a variável `resposta`.

```
botaoAdicionar.addEventListener("click", function(){
    console.log("Buscando pacientes...");

    var xhr = new XMLHttpRequest();

    xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

    xhr.addEventListener("load", function() {
        var resposta = xhr.responseText;
        console.log(typeof resposta);

    });
```

No console, será retornado que o conteúdo é uma string, com uma estrutura bem parecida com um array do JavaScript...

O formato de dados JSON

Os dados possuem uma estrutura parecida com o objeto do JavaScript porque eles estão no formato `JSON` (sigla de *JavaScript Object Notation*), um formato de dados parecidos com os objetos do JavaScript que podemos transportar pela web.

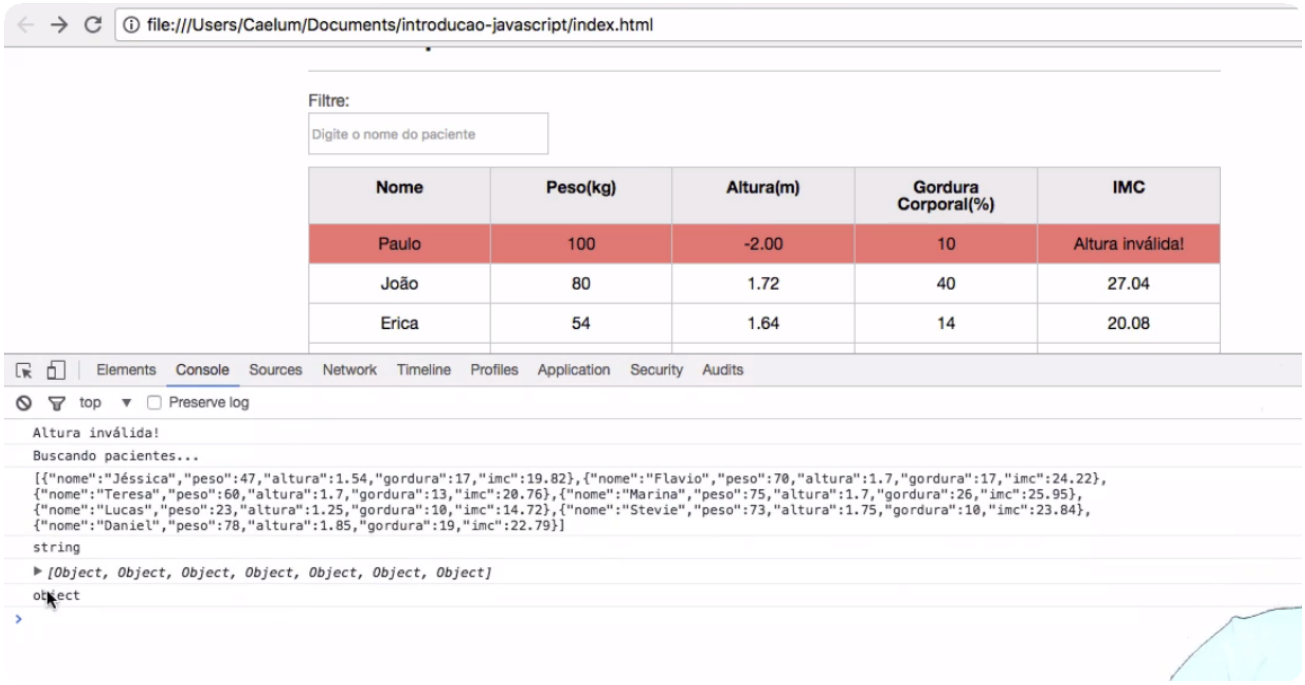
A semelhança é tanta que podemos facilmente converter JSON (o texto da resposta da requisição) em objetos do JavaScript com os quais estamos mais acostumados a utilizar, como array ou mesmo uma string. Queremos que ele seja transformado em um array de objetos, mais útil para o JS.

Para conseguirmos transformar a resposta, que é um texto (uma string), em um array de pacientes, usaremos um "transformador", mais precisamente um **parseador** de JSON para objetos do JavaScript. Para realizarmos esta tarefa usaremos o método `parse()`. Assim, receberemos o texto em JSON, que depois será *parseado*. Em seguida, será retornado um objeto JavaScript. Como nossa resposta se parece com um objeto, o método entenderá isso e nos retornará um **array do objetos**:

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

botaoAdicionar.addEventListener("click", function(){
    console.log("Buscando pacientes...");

    var xhr = new XMLHttpRequest();

    xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");
```

```
    xhr.addEventListener("load", function(){
        var resposta = xhr.responseText;
        console.log(resposta);
        console.log(typeof resposta);

        var pacientes = JSON.parse(resposta);
        console.log(pacientes);
        console.log(typeof pacientes);
    });

    xhr.send();
});
```

No console, veremos que já temos um array, sendo que cada item do array é um objeto que representa o paciente, basta iterar por eles e adicionar cada um na tabela.



Implementamos essa adição de pacientes à tabela, no arquivo `form.js`, porém, o código está preso ao escutador do evento `click` do botão "Adicionar":

```
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) {
        exibeMensagensDeErro(erros);
        return;
    }

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

```
    tabela.appendChild(pacienteTr);

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

});
```

Nós podemos reaproveitar o código criando a função `adicionaPacienteNaTabela`. Essa função receberá um `paciente`, e depois a tag `tr` será montada. O próximo passo será buscar a tabela e adicionar o `pacienteTr` nela.

```
function adicionaPacienteNaTabela(paciente) {
    var pacienteTr = montaTr(paciente);
    var tabela = document.querySelector("#tabela-pacientes");
    tabela.appendChild(pacienteTr);
}
```

Nós já temos uma função, em seguida iremos chamá-la acima do `form.reset()`, passando o `paciente`:

```
 adicionaPacienteNaTabela(paciente);

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

A função de adicionar pacientes no `form.js` já fazia tudo isso antes:

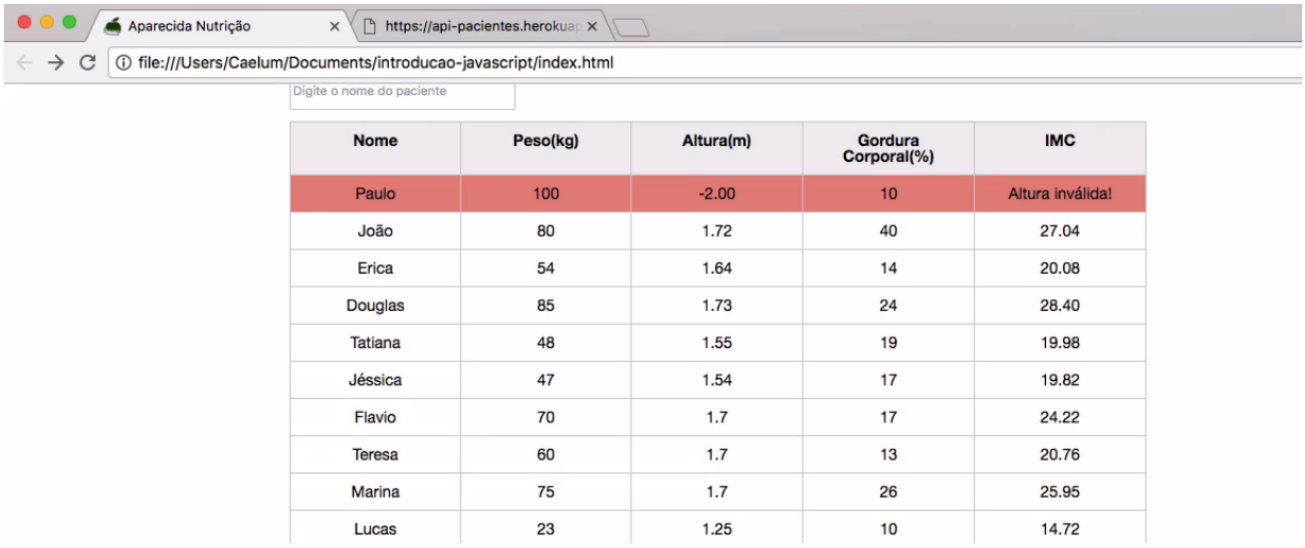
```
var botaoAdicionar = document.querySelector("#adicionar-paciente");
botaoAdicionar.addEventListener("click", function(event){
    event.preventDefault();

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

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

    adicionaPacienteNaTabela(paciente);

    form.reset();

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

});
```

Isto é, a função já montava um paciente, verificava os erros, e o adicionava na tabela. Nós poderemos chamar a função `adicionaPacienteNaTabela()` - com todas as ações mencionadas -, ou podemos aproveitá-la para adicionarmos os pacientes recebidos. No arquivo `buscar-pacientes.js`, adicionaremos `adicionaPacienteNaTabela()`, que por enquanto só inclui um paciente. No entanto, teremos um array com vários deles. Logo, iremos iterar pelo array usando o `forEach()`, e adicionaremos cada paciente contido nele.

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

botaoAdicionar.addEventListener("click", function(){
    console.log("Buscando pacientes...");

    var xhr = new XMLHttpRequest();

    xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

    xhr.addEventListener("load", function() {
        var resposta = xhr.responseText;

        var pacientes = JSON.parse(resposta);

        pacientes.forEach(function(paciente) {
            adicionaPacienteNaTabela(paciente);
        });
    });

    xhr.send();
});
```

Passamos o `paciente` como parâmetro da função anônima, o qual será adicionado na tabela em seguida. Como se tornaram desnecessários, removemos os `console.log()` s.

Vamos verificar se está tudo funcionando no browser. Ao clicarmos em "Buscar Pacientes", a tabela será preenchida com todos os pacientes que estavam no servidor externo.



Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	-2.00	10	Altura inválida!
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	48	1.55	19	19.98
Jéssica	47	1.54	17	19.82
Flavio	70	1.7	17	24.22
Teresa	60	1.7	13	20.76
Marina	75	1.7	26	25.95
Lucas	23	1.25	10	14.72

Tudo está funcionando como estava, incluindo a filtragem. Estamos conseguindo acessar o outro servidor, trazer os pacientes de lá e disponibilizá-los na tabela. A técnica utilizada nessa aula é conhecida como **AJAX**, essa maneira de fazer uma requisição de forma **assíncrona** com JavaScript.

É uma requisição assíncrona porque não está parando o fluxo do código, ou seja, no momento em que a requisição é feita, a execução continua normalmente. Durante esse processo de busca de pacientes no servidor externo, é possível excluir e adicionar pacientes.

Mais adiante, veremos o que acontecerá se houver algum problema na requisição.