05

Capturando erros de requisição

Transcrição

Estamos conseguindo trazer os pacientes e adicioná-los à tabela com um simples clique no botão. Mas sabemos que, na web, nem todas as requisições dão certo.

Muitas vezes, ao acessarmos um site, podemos receber um erro `404`. Por exemplo, ao acessarmos a plataforma da Alura por meio de uma URL inexistente, seremos notificados sobre o tal erro. E este não é o único erro possível, também existem os erros `201`, `402`, `503`, entre outros, que podem ocorrer ao fazermos uma requisição AJAX na web.

O ideal seria que em casos em que fizéssemos uma requisição com o JavaScript, fosse possível detectar algum erro e avisar isso ao usuário.

Para testarmos se uma requisição falhou ou não, devemos verificar o seu código HTTP. Por exemplo, o código HTTP `404` indica que tivemos problema na requisição. O código para uma requisição perfeita, que indica que deu tudo certo, é `200`. Então, depois que a requisição for carregada, poderemos testar se o código é este, caso contrário, significa que houve algum erro.

Para sabermos qual o código da requisição, acessaremos a propriedade `status` do `XMLHttpRequest`.

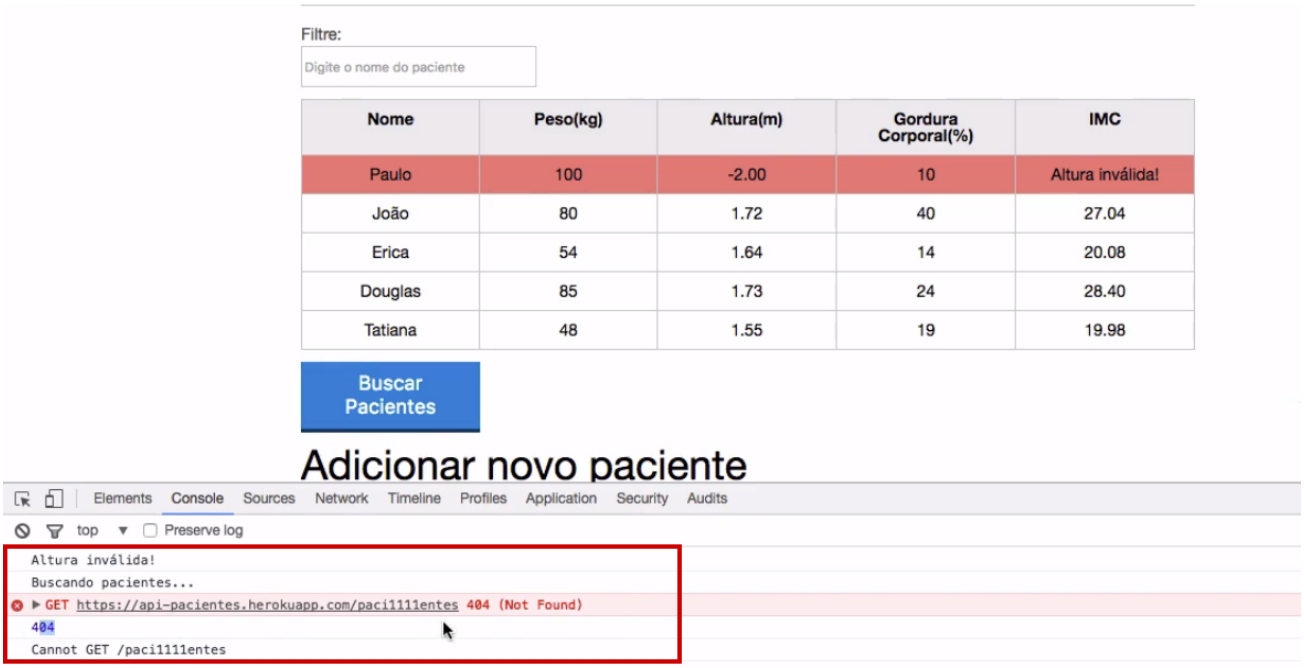
```
var botaoAdicionar = document.querySelector("#buscar-pacientes");

botaoAdicionar.addEventListener("click", function() {
    var xhr = new XMLHttpRequest();

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

    xhr.addEventListener("load", function() {

        if (xhr.status == 200) {
            var resposta = xhr.responseText;
            var pacientes = JSON.parse(resposta);

            pacientes.forEach(function(paciente) {
                adicionaPacienteNaTabela(paciente);
            });
        } else {
            console.log(xhr.status);
            console.log(xhr.responseText);
        }
    });

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

Criaremos um `if`, com o qual testaremos o `xhr.status` e, neste caso, carregaremos os dados da página. Em caso de erro, cairemos no `else` e exibiremos o erro no console. Mostraremos tanto o `xhr.status` como `xhr.responseText`.

No browser, a requisição continuará funcionando. Vamos testar o que aconteceria caso utilizássemos uma URL inexistente, como `https://api-pacientes.herokuapp.com/paci1111entes` .

```
botaoAdicionar.addEventListener("click", function() {
    var xhr = new XMLHttpRequest();

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

Com isso, o erro e a mensagem de resposta serão exibidos no console como gostaríamos, mas seria interessante deixar mais claro ao usuário que um erro ocorreu.



Ele mostra o status `404` e nos informa que a URL não foi encontrada. Ele imprimiu `Cannot GET /paciente1111entes` , ou seja, ele não conseguiu pegar a referida URL. Esta é uma forma para que as ações do código só sejam executadas quando as requisições funcionarem, caso contrário, avisaremos no console, ou alertaremos o usuário de algum outro modo, como por exemplo, exibir uma mensagem de erro.

Para isto, adicionaremos um `span` abaixo da tag `<table>` no arquivo `index.html` :

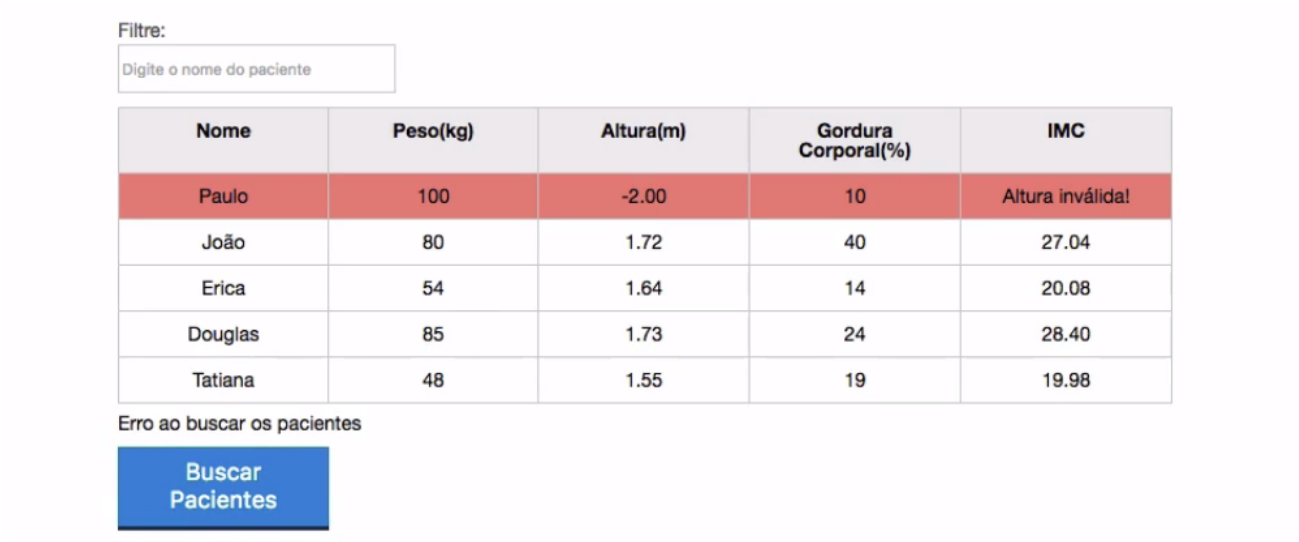
```
<main>
    <section class="container">
        <h2>Meus pacientes</h2>
        <label for="filtrar-tabela">Filtre:</label>
        <input type="text" name="filtro" id="filtrar-tabela" placeholder="Digite o nome do paci
        <table>
            <!-- conteúdo da tabela omitido -->
        </table>
        <span id="erro-ajax" class="invisivel">Erro ao buscar os pacientes</span>
        <button id="buscar-pacientes" class="botao bto-principal">Buscar Pacientes</button>
    </section>
</main>
```

Incluímos no `span` a classe `invisivel` , e por padrão ele não aparecerá na página. E em `buscar-pacientes.js` criaremos a variável `erro-ajax` , com a qual selecionaremos o `span` . Em caso de erro, removeremos a classe

`invisivel` , deixando a mensagem visível para o usuário mesmo para aqueles que estiverem com o console fechado:

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

botaoAdicionar.addEventListener("click", function() {
    var xhr = new XMLHttpRequest();

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

    xhr.addEventListener("load", function() {
        var erroAjax = document.querySelector("#erro-ajax");

        if (xhr.status == 200) {
            var resposta = xhr.responseText;
            var pacientes = JSON.parse(resposta);

            pacientes.forEach(function(paciente) {
                adicionaPacienteNaTabela(paciente);
            });
        } else {
            erroAjax.classList.remove("invisivel");
        }
    });

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

Se alterarmos a URL para uma que não exista, a mensagem "Erro ao buscar os pacientes" é apresentada ao usuário.

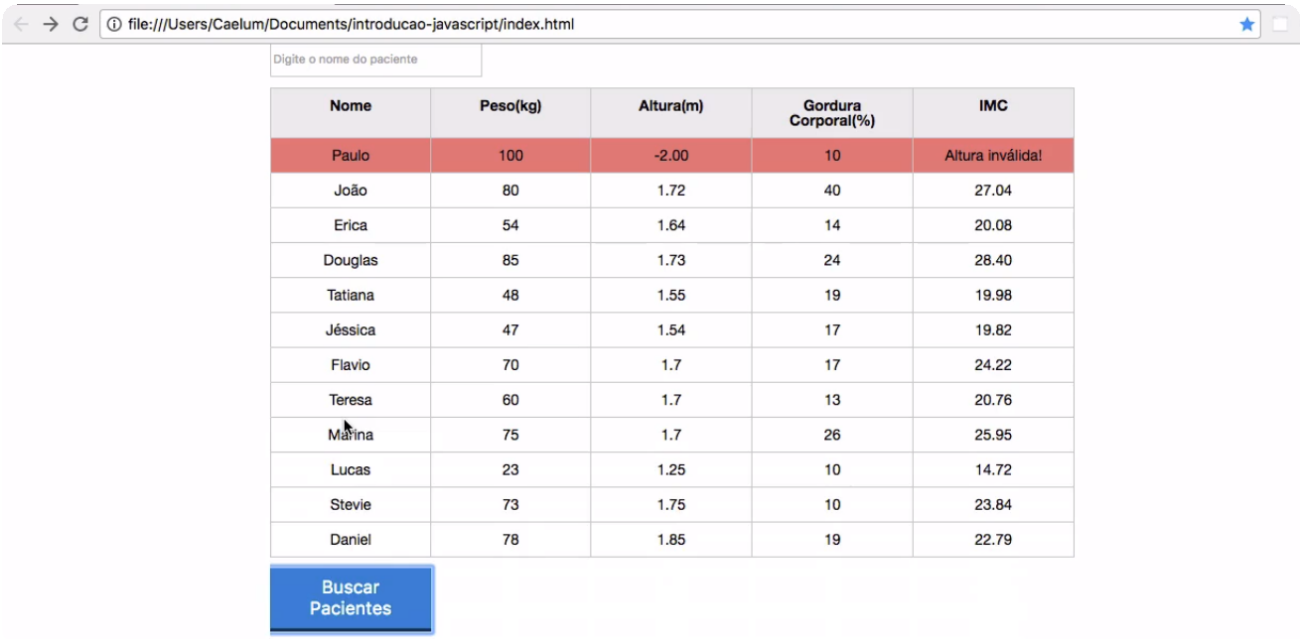


No caso da requisição ser bem sucedida e a requisição ocorrer normalmente, nós ocultaremos a mensagem novamente, adicionando a classe `invisivel` e movendo a variável `erroAjax` para cima do `if` :

```
var xhr = new XMLHttpRequest();

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

xhr.addEventListener("load", function() {
    var erroAjax = document.querySelector("#erro-ajax");
    if (xhr.status == 200) {
```

```
        erroAjax.classList.add("invisivel");
        var resposta = xhr.responseText;
        var pacientes = JSON.parse(resposta);

        pacientes.forEach(function(paciente) {
            adicionaPacienteNaTabela(paciente);
        });
    } else {
        erroAjax.classList.remove("invisivel");
    }
});

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

Ao testarmos, verificaremos que tudo ocorre conforme esperado. No caso de erro, visualizaremos a mensagem e, se ajustarmos o erro da URL, deixaremos de ver a mensagem de erro.



Nesta aula aprendemos o suficiente para fazermos requisições para outros servidores, usamos o método `open()` para abrir a requisição e configurarmos o método `GET` para o seu envio. O `send()` é o método que efetivamente envia a requisição, após o qual devemos escutar a resposta para sabermos quando ela retornar no `responseText`. Em seguida, ela será "parseada" com o `JSON.parse`, se a resposta for recebida no formato JSON - texto com "cara" de JavaScript.

Abordamos vários assuntos sobre a linguagem, sobre como utilizar as funções que já conhecemos para fazermos uma requisição e como integrar dois sistemas.