Marcelosmbr
https://github.com/Marcelosmbrr

SUMÁRIO

1. O QUE É AJAX

Acrônimo de Asynchronous Javascript and XML, é uma metodologia de programação entre front-end e back-end de aplicações Web. Um dos principais objetivos é tornar as respostas das páginas Web mais rápidas pela troca de pequenas quantidades de informações com o servidor Web, nos bastidores.

Além disso, evita-se que a página Web inteira tenha que ser recarregada cada vez que alguma nova informação precisa ser consultada no servidor. Em geral, isso significa que páginas Web com recursos AJAX permitem maior interatividade, velocidade de processamento e usabilidade.

Em outras palavras, requisições AJAX permitem que a comunicação com o servidor seja efetuada para a atualização de dados em tempo real, sem que haja uma atualização de todo o conjunto de informações da aplicação. Assim, ao invés de, por exemplo, criar um outro documento HTML, independente do Index, com as informação deste, e mais as adicionais que deveriam ser mostradas após algum evento, como um click em um botão, uma requisição AJAX permite que apenas uma informação especifica seja atualizada na página, mantendo o seu restante estrutural intacto – desta forma, não há um recarregamento da página.

É utilizado para diversos fins, por motivos nítidos de melhor usabilidade, mas poderia citar aquelas aplicações especificas a que chamamos de SPA (Single Page Application). Este tipo, é uma aplicação Web de página única, isto é, em que todas as informações atualizadas são independentemente carregadas em uma mesma página. Um exemplo deste tipo de abordagem é a página do Outlook, em que há uma estrutura padrão da página, mas as informações apresentadas, e que podem ser acessadas, são atualizadas dentro da mesma estrutura, sem que haja um recarregamento inteiro do documento.

Outra aplicação específica que poderia ser citada, e que é realizada por meio do uso de requisições assíncronas, é o uso de **API's** para atualização de informações no lado do cliente, sem que haja recarregamento do documento. Neste caso o servidor comunicado envia os dados requisitados em formato XML ou JSON, que são acessados a partir da criação de um Objeto, com a linguagem JavaScript, da classe XMLHttpRequest. E, sendo uma comunicação cliente x servidor, este último pode ser local, e a API utilizada desenvolvida pelo próprio desenvolvedor da aplicação, ou pode ser um serviço criado e ofertado por outra entidade. Por exemplo, a

Google oferece diversas API's, e a depender do seu uso, expresso em número de requisições realizadas ao servidor, cobra um valor determinado.

2. TECNOLOGIAS UTILIZADAS

AJAX utiliza uma série de tecnologia combinadas, padronizadas e não padronizadas, para o desenvolvimento Web. Além do óbvio HTML e CSS, para criação da estrutura do documento, o AJAX utiliza principalmente, entre outras:

XML (Extensible Markup Languague): Formato que permite a criação de documentos estruturados com dados hierárquicos, sendo muito usado para interoperabilidade de sistemas na Web.

XSLT (Extensible Stylesheet Language for Transformation): Linguagem para transformação de documentos XML.

ECMAScript: Padrão de linguagem de script, cujas principais implementações são o JavaScript e o JScript.

XMLHttpRequest: Objeto que define uma API com funcionalidades para scripts do lado do cliente para comunicação entre um cliente e um servidor usando o protocolo HTTP.

DOM (Document Object Model): Para manipular a estrutura e o estilo de documentos Web dinamicamente.

JSON (Javascript Object Notation): Possui o mesmo fim que o formato XML, isto é, intercâmbio de dados entre sistemas. No entanto, é preferível o seu uso em muitos casos, por ser arquitetado em formato compatível com a linguagem JavaScript.

Uma das principais tecnologias nos bastidores do AJAX é o objeto **XMLHttpRequest**, que torna possível a característica mais atraente do AJAX: a sua natureza assíncrona. Esse objeto pode ser usado pelo JavaScript para transferir XML e outros dados textuais de um servidor Web para uma página Web, usando o protocolo HTTP.

3. SOBRE O PROCESSO DE REQUISIÇÃO

Precisamos saber como funciona uma requisição no modelo de comunicação síncrona, para que a diferença da assíncrona se torne clara. Neste primeiro caso, ao fazer uma requisição, a aplicação cliente ficará em espera e não permitirá que os usuários continuem interagindo até receber uma resposta do servidor. Este servidor realizará uma série de tarefas e depois retornará

as informações solicitadas em outra página, produzindo um recarregamento completo da aplicação.

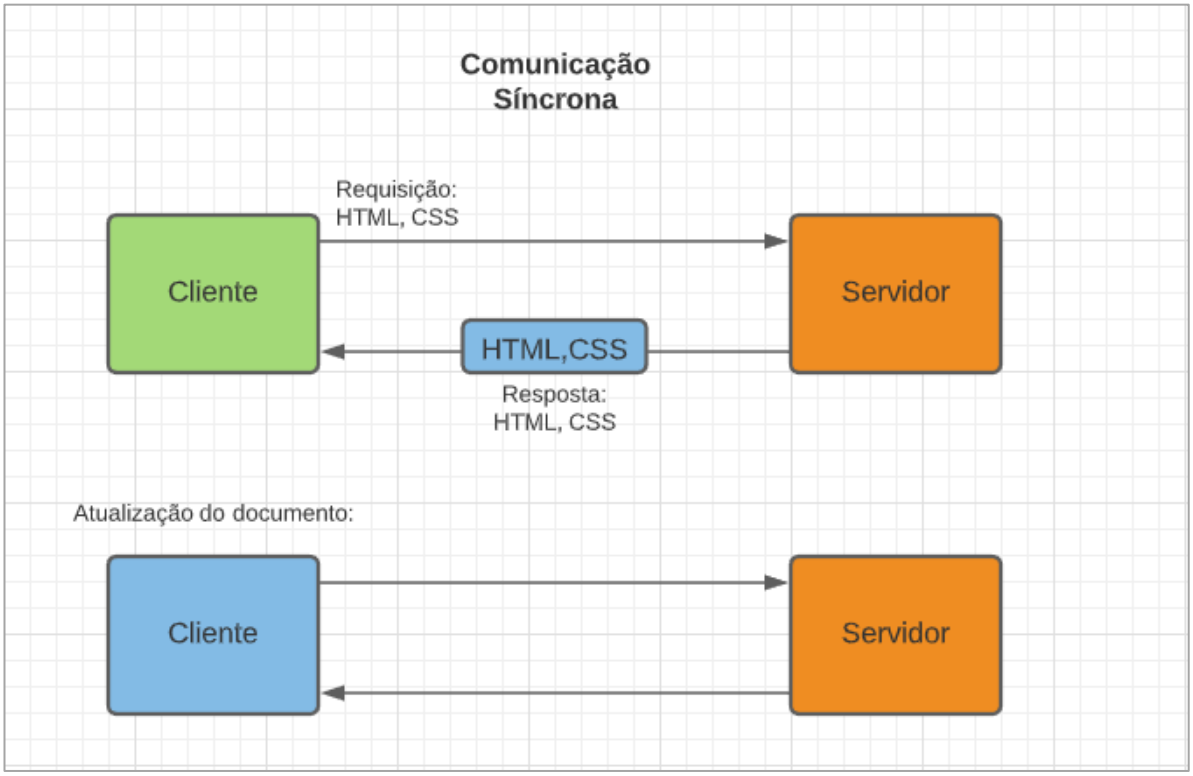


Figura 1 Fonte: Criado no Lucidchart

Para ajudar a melhorar a experiência do usuário, no AJAX, a requisição é chamada em segundo plano. O navegador gera uma chamada do JavaScript que irá ativar o **XMLHttpRequest**. Com isso, em segundo plano, o navegador cria uma requisição HTTP para o servidor. Este recebe a requisição, busca os dados e envia para o navegador. O navegador, ao receber a resposta de volta, utiliza o DOM para modificar a página atual de maneira que esta reflita a resposta que veio do servidor.

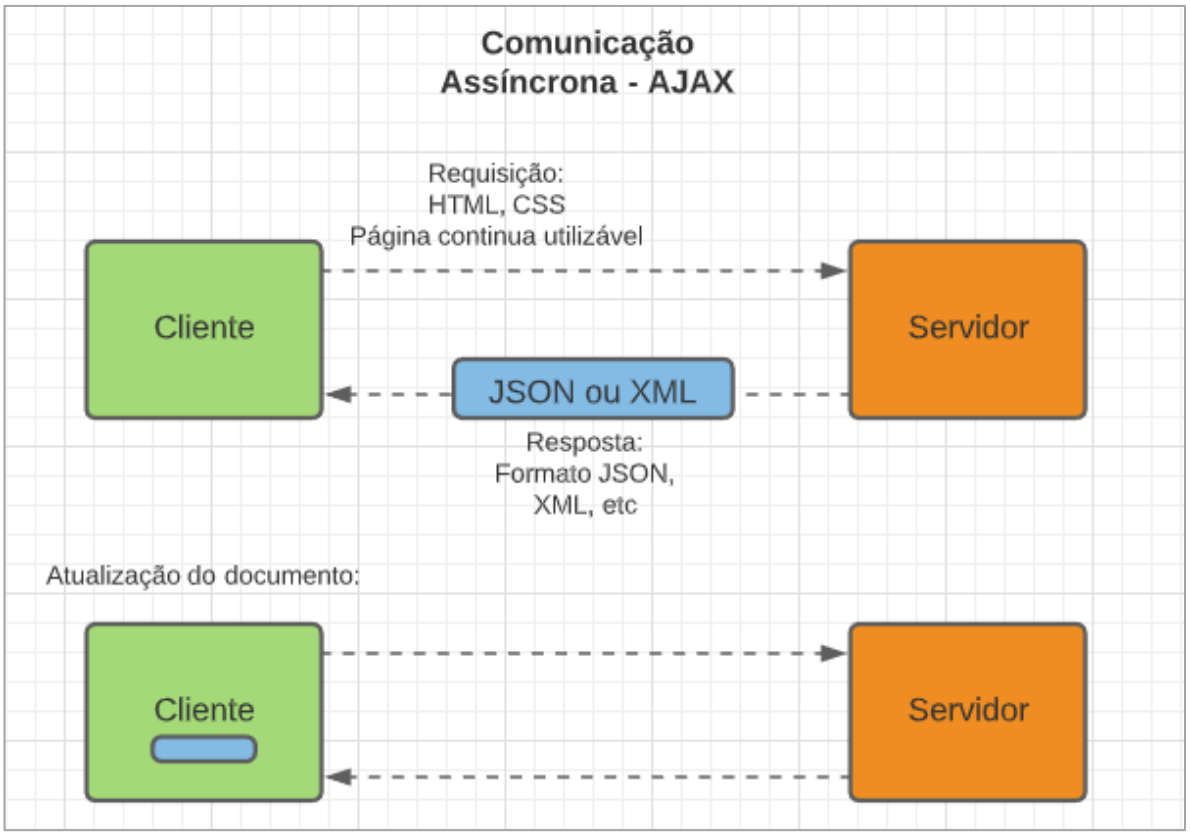


Figura 2 Fonte: Criado no Lucidchart

4. COMO REALIZAR UMA REQUISIÇÃO AJAX

4.1. REQUISIÇÃO HTTP

Para fazer uma requisição HTTP ao servidor usando JavaScript, você precisa de uma instância de uma classe que fornece essa funcionalidade. Este é o lugar onde o XMLHttpRequest entra.

Primeiro, normalmente, cria-se uma function que será chamada após a ocorrência de um evento na aplicação, e que irá conter os procedimentos para a requisição do recurso do servidor. Além disso, como parâmetro, a function irá conter o recurso que será requisitado, podendo ser um link, de uma API ofertada por uma entidade externa, um arquivo local, de uma estrutura HTML, armazenada em um servidor local, ou qualquer outro recurso requisitável de qualquer servidor possível.

É importante ressaltar que se o servidor for local, e por conseguinte o recurso requisitado estiver na própria máquina da aplicação, esta deverá estar na pasta que o servidor local utiliza para buscar documentos. Por exemplo, se for utilizado o XAMPP, o diretório para os arquivos deve ser o “htdocs”, e se for o WAMPP, será o “www”.

Tendo isto em mente, o segundo passo seria a própria implementação dos procedimentos da requisição, no escopo da função criada segundo os requisitos citados.

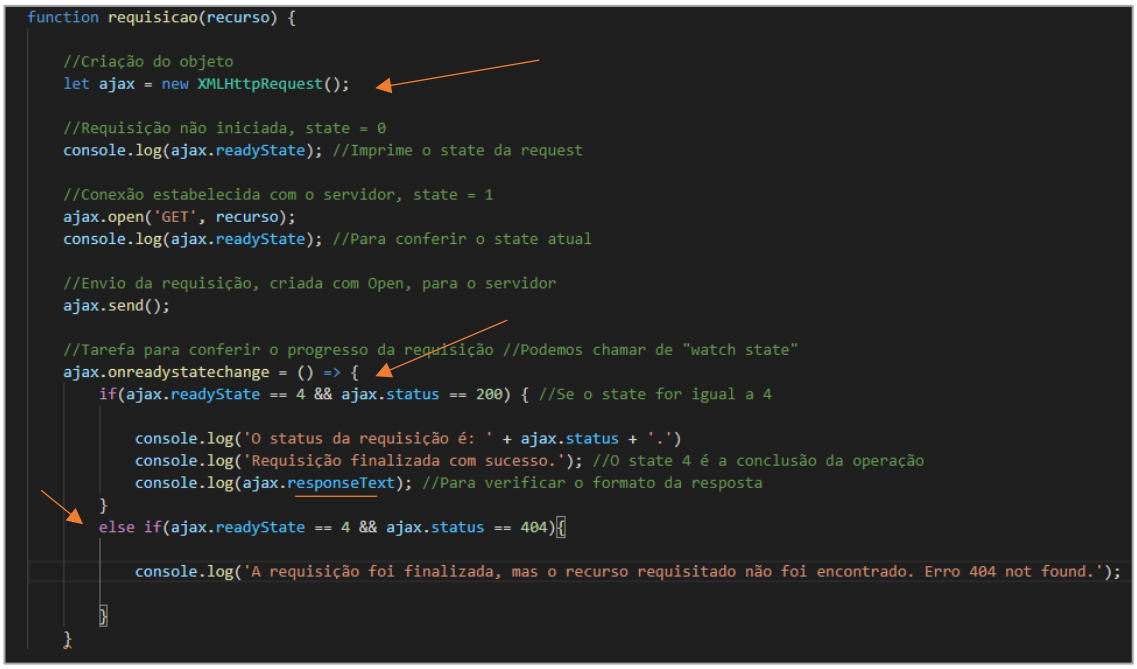
Assim, após realizado o primeiro passo, o segundo consiste em criarmos e instanciarmos o Objeto da classe XMLHttpRequest, acessarmos o método de classe "open()", para abrir a requisição ao servidor, com os parâmetros 'GET' e o próprio recurso, e, em seguida, o método "send()" para enviar a requisição. Podemos consultar o **state** da requisição chamando o método "readyState".

Uma requisição AJAX possui **states** e **status**, que apesar de semelhantes sintaticamente, são diferentes. **Existem 5 states**, começando a partir do 0, que são os estados do procedimento de comunicação do cliente com o servidor. São eles:

- 0) Requisição não inicializada;
- 1) Conexão estabelecida com o servidor;
- 2) Requisição recebida pelo servidor;
- 3) Processamento da requisição;
- 4) Requisição finalizada e resposta enviada, do servidor para o cliente;

Mesmo que um recurso não seja encontrado no servidor, e que a finalidade da comunicação não seja satisfeita, os 5 states são realizados. O que varia, a depender da realização ou não da finalidade da requisição, é o **status. Existem códigos de status** que informam sobre o andamento de uma requisição, e sua conclusão. O conhecido "**erro 404 not found**" é um desses códigos, e que aparece para o cliente quando todos os states são realizados, mas nenhum recurso é enviado como resposta, pois o servidor constatou que não existe, em seus arquivos, o documento que fora requisitado. Existem diversos códigos, e podem ser estudados na internet. **Pesquise por "Códigos de status de respostas HTTP".**

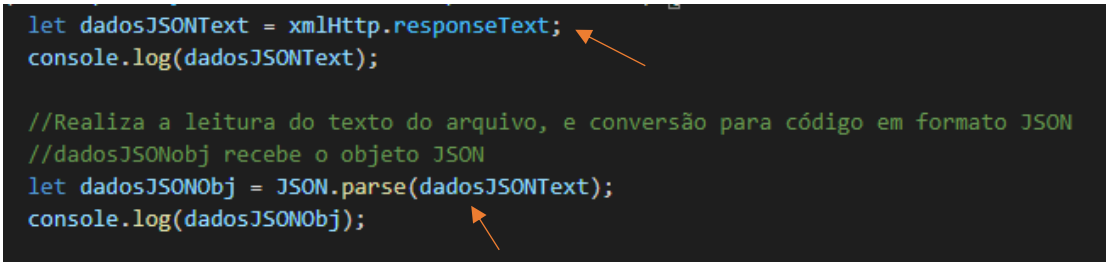
Seguindo com o procedimento, devemos então implementar o código para verificação do state da requisição e o status, por meio de condicionais. Para isso, utilizamos o método "onreadystatechange()", que é chamado sempre que o atributo "readystate" é modificado. Atribuímos a ele uma function que será chamada sempre que o método for chamado, e nesta implementamos as condicionais. **Se o state == 4 e o status == 200**, o procedimento foi concluído e o recurso foi enviado, se não, **se o state == 4 e o status == 404**, o procedimento foi concluído, mas o servidor não encontrou o recurso solicitado. A resposta do servidor, caso o status for igual a 200, é recuperada com o método "responseText".



```
function requisicao(recurso) {

    //Criação do objeto
    let ajax = new XMLHttpRequest();

    //Requisição não iniciada, state = 0
    console.log(ajax.readyState); //Imprime o state da request

    //Conexão estabelecida com o servidor, state = 1
    ajax.open('GET', recurso);
    console.log(ajax.readyState); //Para conferir o state atual

    //Envio da requisição, criada com Open, para o servidor
    ajax.send();

    //Tarefa para conferir o progresso da requisição //Podemos chamar de "watch state"
    ajax.onreadystatechange = () => {
        if(ajax.readyState == 4 && ajax.status == 200) { //Se o state for igual a 4

            console.log('O status da requisição é: ' + ajax.status + '.')
            console.log('Requisição finalizada com sucesso.'); //O state 4 é a conclusão da operação
            console.log(ajax.responseText); //Para verificar o formato da resposta
        }
        else if(ajax.readyState == 4 && ajax.status == 404){

            console.log('A requisição foi finalizada, mas o recurso requisitado não foi encontrado. Erro 404 not found.');

        }
    }
```

4.2. RESPOSTA DO SERVIDOR E UMA VISÃO SOBRE API

Normalmente, como em uso de API's, a resposta enviada do servidor externo, como uma estrutura HTML, é em formato XML ou JSON textual, isto é, com conteúdo organizado segundo sua extensão, mas em tipo string. Assim, é necessário utilizar o algoritmo **Parser** para realizar a leitura do documento, e sua conversão para JSON que, como dito anteriormente, é preferível, por ser codificado em JavaScript. **Veja um exemplo abaixo, de conversão de um texto JSON, enviado pelo servidor, para um Objeto JSON:**



```
let dadosJSONText = xmlHttp.responseText;
console.log(dadosJSONText);

//Realiza a leitura do texto do arquivo, e conversão para código em formato JSON
//dadosJSONobj recebe o objeto JSON
let dadosJSONObj = JSON.parse(dadosJSONText);
console.log(dadosJSONObj);
```

No caso acima, primeiro a variável "dadosJSONText" recebeu a estrutura em formato de texto JSON, e então esta foi impressa no console para visualização do seu conteúdo. Logo em seguida, a mesma variável foi passada como parâmetro para a função pré-definida JSON.Parser(), que realiza a leitura do seu conteúdo e a conversão para um Objeto JSON, isto é, um Objeto em código Javascript, cujos atributos puderam vir a ser acessados, no restante da aplicação.

Para deixar claro, esta aplicação em específico é uma requisição de dados de uma API. Quando digitado um CEP em um input, o valor é recuperado com JS, por .value, e adicionado

ao link de requisição HTTP disponibilizado pela entidade criadora da API. Neste link, há um segmento em que, na documentação, é dito que deve estar o valor requisitado. O que fazemos, então, é atribuir o link à url, e concatenar com o valor recuperado do input e enviado, a partir de um evento DOM, como parâmetro na chamada da function da requisição AJAX. O evento DOM utilizado neste caso, foi o "onfocus", e o parâmetro da chamada da função foi (this.value).

Em termos práticos, se o link é "htttp://blablalbla/city_name=city&blalbla...", devemos escrever url = "htttp://blablalbla/city_name="+variável+"&blalbla...". Esta url será a requisição feita ao servidor e enviada como segundo parâmetro do método "open()", como já demonstrado. **No geral, é assim que se utilizam API's.**

Caso o servidor for local, e o recurso requisitado, por exemplo, for uma estrutura HTML, ao invés de um link, e de todo este procedimento descrito acima, o parâmetro da função principal será simplesmente o caminho do arquivo requisitado. A resposta da requisição poderá ser, assim, se for o caso, diretamente o conteúdo do documento requisitado no próprio servidor, ao invés de um JSON ou de um XML. E como será adicionado ao local apropriado da DOM do Index? Simples, a partir do Javascript terá que ser criado um novo elemento para a DOM, ou selecionado um já existente, a partir dos seletores, e atribuído ao seu valor ".value", o atributo "responseText" do Objeto XMLHttpRequest.

Veja abaixo um exemplo em que com Innerhtml, o conteúdo da resposta do servidor é atribuído a um elemento da árvore DOM, de id igual a "conteúdo":

```
console.log('O status da requisição é: ' + ajax.status + '.')
console.log('Requisição finalizada com sucesso.'); //O state 4 é a conclusão da operação
document.getElementById('conteudo').innerHTML = ajax.responseText; //Resposta do servidor para a requisição
document.getElementById('conteudo').style.textAlign = 'center';
```

5. FONTES

https://www.hostinger.com.br/tutoriais/o-que-e-ajax/

https://www.devmedia.com.br/o-que-e-o-ajax/6702

https://www.w3schools.com/xml/ajax_xmlhttprequest_response.asp

https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status

https://developer.mozilla.org/pt-BR/docs/Web/API/XMLHttpRequest/onreadystatechange

https://www.w3schools.com/js/js_json_parse.asp