1 Introdução ao Ajax

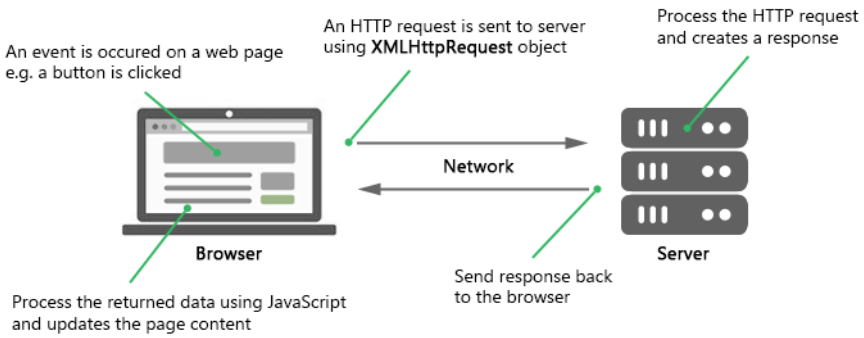
AJAX significa **Asynchronous JavaScript and XML**, é uma **técnica** para criar páginas web rápidas e dinâmicas. **AJAX permite que as páginas da Web sejam atualizadas de forma assíncrona, trocando pequenas quantidades de dados pelo servidor**. Isso significa que é possível atualizar partes de uma página da Web, sem recarregar a página inteira.

O conceito AJAX existe desde a década de 90. Porém, recebeu maior importância quando a Google o incorporou no Google Mail e Google Maps em 2004. Atualmente, é bastante utilizado em diversas aplicações web para melhorar o processo de comunicação com o servidor.

Podemos encontrar exemplos em sistemas de votação e avaliação, canais de chat, etc.

1.1 O que é uma chamada ajax?[1]

Uma **chamada Ajax** é uma requisição assíncrona iniciada pelo browser e processada pelo servidor em segundo plano.



[1] Consultado em: https://pt.wikipedia.org/wiki/XMLHttpRequest

1.2 O que é o XMLHTTPREQUEST?

XMLHttpRequest (XHR) é uma API disponível em linguagens de script para *browsers* tais como JavaScript. É utilizada para enviar requisições HTTP ou HTTPS diretamente ao servidor web e carregar os dados de resposta do servidor de volta ao script. Por outras palavras, o objeto XMLHttpRequest é usado para a troca de dados entre o browser e o servidor.

Enviar um pedido e recuperar a resposta

Antes de realizar a comunicação Ajax entre cliente e servidor, a primeira tarefa a fazer consiste em instanciar a classe XMLHttpRequest, conforme mostrado abaixo:

```
var pedido = new XMLHttpRequest();
```

A próxima etapa, envio do pedido ao servidor, consiste em aplicar os métodos **open()** e **send()** do objeto XMLHttpRequest.

Consultando a tabela na página seguinte, o método open() normalmente aceita dois parâmetros — o **método do pedido** HTTP a ser usado, como "GET", "POST" etc., e a **URL** para onde enviar o pedido. Vejamos os exemplos:

```
pedido.open('GET', 'historia.txt', true);
ou
pedido.open("POST", "adiciona-utilizador.php");
```

Geralmente, o método **GET** é usado para enviar uma pequena quantidade de dados ao servidor. Por outro lado, o método **POST** é usado para enviar grandes quantidades de dados, como os dados de um formulário.

No método GET, os dados são enviados como parâmetros de URL. No método POST, os dados são enviados ao servidor como parte do corpo da requisição HTTP. Os dados enviados pelo método POST não ficarão visíveis na URL.

Dica: O ficheiro pode ser de qualquer tipo, como .txt ou .xml, ou ficheiros de script do lado do servidor, como .php ou .aspx, que podem executar algumas ações no servidor (por

exemplo, inserir ou ler dados da base de dados) antes do envio da resposta de volta ao cliente (*browser*).

O método **send()** é usado para enviar a requisição HTTP ao servidor. Este método é chamado após a configuração da requisição usando os métodos open() e, opcionalmente, setRequestHeader() para definir o método HTTP, a URL e quaisquer cabeçalhos adicionais necessários.

pedido.**send**();

ou

pedido.**send**(body);

Para mais detalhes consulte a seguinte tabela:

Métodos	**Descrição**
new XMLHttpRequest()	Cria um objeto XMLHttpRequest
open(*method, url, async, user, pass*)	open() é usado para inicializar uma requisição HTTP antes de enviá-la *method*: tipo de requisição: GET or POST *url*: servidor local (file) *async*: true (asynchronous) ou false (synchronous) *user:* username por opção *pass:* password por opção
send()	Envia o pedido (request) ao servidor (GET)
send(string)	Envia o pedido (request) ao servidor (POST)
abort()	Cancela o pedido (request) corrente
getAllResponseHeaders()	Devolve a informação do cabeçalho
setRequestHeader(*header, value*)	Adiciona um par label/valor ao header para ser enviado

A próxima etapa consiste em capturar (porque a resposta é assíncrona) a resposta vinda do servidor. Par tal, temos de compreender o significado de algumas propriedades, a propriedade ***readyState*** indica o estado atual da requisição; a propriedade ***onreadystatechange*** define a função/evento a ser executada quando o ***readyState*** muda de estado; a propriedade ***status*** é usada para obter o código de status HTTP da resposta da solicitação. O código de status HTTP é um número de três dígitos que é incluído na resposta do servidor para indicar o resultado da solicitação HTTP. A propriedade ***statusText*** é usada para obter a descrição textual associada ao código de status HTTP retornado na resposta da requisição.

Consulta a tabela para melhora compreensão das propriedades associadas ao objeto XMLHttpRequest.

Propriedades	Descrição
onreadystatechange	Define a função a ser chamada quando a propriedade ***readyState*** muda de estado
readyState	Possui o ***status*** do XMLHttpRequest. **0**: pedido não inicializado (estado **UNSENT**). Um objeto XMLHttpRequest foi criado, mas o método open() ainda não foi chamado. **1:** (estado **OPENED**): o método **open() foi chamado**. Neste ponto, pode especificar os detalhes da requisição, como o método HTTP a ser usado e a **URL**. **2** (estado **HEADERS_RECEIVED**): o método **send() foi chamado** e os cabeçalhos da resposta HTTP foram recebidos. As informações dos cabeçalhos estão disponíveis através dos métodos getAllResponseHeaders() e getResponseHeader(). **3**: (estado **LOADING**): os dados da resposta são recebidos. Pode aceder aos dados parciais usando a propriedade responseText ou responseXML, dependendo do tipo de dados esperados.

	4: (estado **DONE**): o conteúdo da resposta foi recebido ficando disponível. Neste ponto, pode verificar o código de status da resposta e processar os dados.
status	Devolve o número do estado do *request* **200 OK**: indica que a requisição foi bem sucedida. **201 Created**: indica que a requisição foi bem sucedida e resultou na criação de um novo recurso. **204 No Content**: indica que a requisição foi bem sucedida, mas não há conteúdo a ser devolvido na resposta. **400 Bad Request:** indica que a requisição do cliente é inválida. **401 Unauthorized**: indica que a requisição requer autenticação e as credenciais fornecidas são inválidas. **404 Not Found**: indica que o recurso solicitado não foi encontrado no servidor. **500 Internal Server Error**: indica que houve um erro no servidor ao processar a requisição.
responseText	Devolve os dados da resposta como string
responseXML	Devolve os dados da resposta como XML
statusText	Devolve o estado com o OK ou Not Found

A lista dos valores que a propriedade *status* pode assumir é muito maior.

É importante fixar: quando o **readyState for 4** e o **status for 200**, significa que a requisição foi concluída com sucesso e o conteúdo da resposta ficará disponível em xhr.responseText.

1.3 Vamos passar à prática

O primeiro exemplo explora os conceitos apresentados anteriormente recorrendo a uma requisição GET.

1. Cria um ficheiro chamado **tuto1.html** na raiz do servidor e analisa o seguinte código:

```
<!DOCTYPE html>
<html lang="pt-pt  ">
<head>
<meta charset="utf-8">
<title>JavaScript Ajax Método GET</title>
<script>
function displayFullName() {
    // Create a XMLHttpRequest object
    var pedido = new XMLHttpRequest();
    pedido.open("GET", "greet.php?fname=John&lname=Clark");

    // Defining event listener for readystatechange event
    pedido.onreadystatechange = function() {
        // Check if the request is compete and was successful
        if(this.readyState === 4 && this.status === 200) {
            // Inserting the response from server into an HTML element
            document.getElementById("result").innerHTML =
this.responseText;
        }
    };

    // Sending the request to the server
    request.send();
}
</script>
</head>
<body>
    <div id="result">
        <p>O resultado do pedido ao servidor (resposta) é colocado nesta
DIV</p>
    </div>
```

```
    <button type="button" onclick="displayFullName()">Mosta Nome
Completo</button>
</body></html>
```

2. Guarda o ficheiro.

Quando a requisição é assíncrona, o método send() provoca uma resposta após o envio da requisição. Portanto, há a necessidade de verificar qual o estado da requisição antes de a processar usando a propriedade **readyState** do objeto XMLHttpRequest.
Como vimos na tabela anterior, o readyState representa um número inteiro que especifica o status da requisição HTTP. Além disso, a função que captura a resposta é o evento **onreadystatechange** que é chamado sempre que a propriedade readyState é alterada.

Observação: teoricamente, o evento readystatechange deve ser acionado sempre que a propriedade readyState for alterada. Porém, a maioria dos browsers não dispara esse evento quando readyState muda para 0 ou 1. No entanto, todos os browsers disparam o evento quando readyState muda para 4. Este é o motivo pela existência da instrução this.readyState === 4 na estrutura if acima definida.

3. Agora vamos apresentar o código a processar pelo servidor. Cria o ficheiro **greet.php**.

```
<?php
if(isset($_GET["fname"]) && isset($_GET["lname"])) {
    $fname = htmlspecialchars($_GET["fname"]);
    $lname = htmlspecialchars($_GET["lname"]);

    $fullname = $fname . " " . $lname;
    echo "Olá, o meu nome completo é $fullname.";
} else {
    echo "Este é o meu site.";
}
?>
```

4. Executa o ficheiro **tuto1.html** no browser.

Agora falta apresentar um exemplo que explora os conceitos apresentados anteriormente recorrendo a uma requisição POST. Analisa o seguinte código:

1. Cria um ficheiro com o nome **tuto2.html** na raiz do servidor.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JavaScript Ajax POST</title>
<script>
function postComment() {
    var request = new XMLHttpRequest();
    request.open("POST", "confirmation.php");

    request.onreadystatechange = function() {
        if(this.readyState === 4 && this.status === 200) {
            document.getElementById("result").innerHTML =
this.responseText;
        }
    };

    var myForm = document.getElementById("myForm");
    var formData = new FormData(myForm);

    request.send(formData);
}
</script>
</head>
<body>
    <form id="myForm">
        <label>Nome:</label>
        <div><input type="text" name="name"></div>
        <br>
```

```
        <label>Comentário:</label>

        <div><textarea name="comment"></textarea></div>
        <p><button type="button" onclick="postComment()">Enviar
Comentário</button></p>
    </form>
    <div id="result">
        <p>Aqui será apresentada a resposta do servidor</p>
    </div>
</body>
</html>
```

2. Guarda o ficheiro.
3. Cria o ficheiro com o nome confirmation.php para processar a requisição.

```
<?php
if($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = htmlspecialchars(trim($_POST["name"]));
    $comment = htmlspecialchars(trim($_POST["comment"]));

    if(!empty($name) && !empty($comment)) {
        echo "<p>Olá, <b>$name</b>. O teu comentário foi recebido com
sucesso.<p>";
        echo "<p>Escreveste: <b>$comment</b></p>";
    } else {
        echo "<p>Preenche os campos do formulário!</p>";
    }
} else {
    echo "<p>Algo correu mal. Tenta de novo.</p>";
}
?>
```

4. Executa o ficheiro **tuto2.html** no browser.

Por motivos de segurança, os *browsers* não permitem a realização de requisições Ajax entre domínios. Isto significa que só pode fazer requisições Ajax para URLs do mesmo domínio da página original, por exemplo, se a tua aplicação for executada no domínio "mysite.com", não poderá fazer requisições Ajax no domínio "othersite.com" ou qualquer outro domínio. No entanto, pode carregar imagens, folhas de estilo, ficheiros JS e outros recursos de qualquer domínio.

Se não pretender usar o objeto **FormData** para enviar os dados do formulário ao servidor, mas pretender enviar os dados ao servidor recorrendo a uma *query string*, ou seja **request.send(key1=value1&key2=value2)**, precisa definir explicitamente o cabeçalho da requisição utilizando o método **setRequestHeader()**, assim:

```
request.setRequestHeader("Tipo de conteúdo", "aplicativo/x-www-form-urlencoded");
```

O método setRequestHeader() deve ser chamado depois de chamar o método open() e antes de chamar método send().

Alguns cabeçalhos de requisição mais comuns são:

application/x-www-form-urlencoded	multipart/form-data
application/json	application/xml
text/plain	text/html

Para consolidar os teus conhecimentos, estuda os exemplos entregues pelo teu professor e aplica-os nos teus projetos.

1.4 Estudo dos Detalhes na Mudança de Estado

Considera os 3 ficheiros como ponto de partida, index.html, app.js e data.txt

1. Cria o ficheiro index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ajax</title>
</head>
<body>
    <div class="container">Olá</div>
    <button>Clique</button>
    <script src="app.js"></script>
</body>
</html>
```

2. Cria o ficheiro app.js

```
const btn = document.querySelector('button');
const main = document.querySelector('.container');
const url = 'data.txt';
btn.onclick = reqData;
function reqData(){
    console.log('clique')
    const xhr = new XMLHttpRequest();
    console.log("FR "+xhr.readyState);
}
```

3. Cria o ficheiro data.txt

```
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
```

4. Executa o ficheiro HTML no browser.

Clica no botão, na consola podes visualizar.

```
clique
FR 0
```

A execução do método open ainda não foi executada.

5. Acrescenta o código ao ficheiro app.js, a função anónima

```
…
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  console.log("On "+xhr.readyState)
}
console.log("FR "+xhr.readyState);
xhr.open('GET', url);
```

6. Executa o código e observa o resultado na consola.

```
clique
FR 0
On 1
```

Houve uma mudança de estado, passou de 0 para 1, após a execução do método open(). Repare que foi o método onreadystatechange que capturou a mudança de estado, logo a conexão com o servidor foi estabelecida.

Se executa-se o método send(), o evento *onreadystatechange* faria a mudança dos 4 estados, passava de 1 para 2, 3 e 4. No entanto, desejamos capturar as mudanças isoladamente.

7. Altera o código da função anónima no ficheiro app.js

```
console.log('clique')
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
        console.log("On "+xhr.readyState)
}
console.log("FR "+xhr.readyState);
xhr.onprogress = function(){
        console.log("Progress "+xhr.readyState)
}
xhr.open('GET', url);
xhr.send()
```

8. Executa o código e observa o resultado na consola.

```
clique
FR 0
On 1
On 2
On 3
Progress 3
On 4
```

Qual a conclusão destas mudanças de estado?

Inicialmente, o estado estava a 0 (pedido não inicializado). Dentro do evento onreadystatechange passou do estado 1 (conexão ao servidor foi estabelecida, executado o método open), depois passou para o estado 2 (o servidor recebeu o pedido, o método send foi executado), depois passou para o estado 3 (o servidor está a processar a requisição. Neste momento, o evento onprogress foi executado) e por fim, o estado mudou para 4 (o cliente recebeu a resposta do servidor).

9. Altera o código da função anónima no ficheiro app.js

```
console.log('clique')
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
        console.log("On "+xhr.readyState)
}
console.log("FR "+xhr.readyState);
xhr.onprogress = function(){
        console.log("Progress "+xhr.readyState)
}
xhr.onload = function(){
        console.log("Done "+xhr.readyState)
        console.log(xhr.responseText)
}
xhr.open('GET', url);
xhr.send()
```

10. Executa o código e observa o resultado na consola.

Repara que o evento onload executou quando o cliente recebeu a resposta da requisição (estado 4).

```
clique
FR 0
On 1
On 2
On 3
Progress 3
On 4
Done 4
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
```

11. Altera o código da função anónima no ficheiro app.js

```
console.log('clique')
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
        console.log("On "+xhr.readyState)
        console.log(xhr.responseText)
}
console.log("FR "+xhr.readyState);
xhr.onprogress = function(){
        console.log("Progress "+xhr.readyState)
        console.log(xhr.responseText)
}
xhr.onload = function(){
        console.log("Done "+xhr.readyState)
        console.log(xhr.responseText)
}
    xhr.open('GET', url);
    xhr.send()
```

12. Executa o código e observa o resultado na consola.

```
clique
FR 0
On 1
<empty string>
On 2
<empty string>
On 3
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
Progress 3
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
On 4
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
Done 4
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
```

13. Altera o código da função anónima no ficheiro app.js

```
console.log('clique')
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
        console.log("On "+xhr.readyState)
        console.log(xhr.responseText)
```

```
            console.log(this.status)
            if(xhr.readyState == 4){
                console.log(`Done ${xhr.responseText}`)
            }
        }
console.log("FR "+xhr.readyState);
xhr.onprogress = function(){
            console.log("Progress
"+xhr.readyState)
            console.log(xhr.responseText)
}
xhr.onload = function(){
            console.log("Done "+xhr.readyState)
            console.log(xhr.responseText)
 }
 xhr.open('GET', url);
 xhr.send()
```

```
clique
FR 0
On 1
<empty string>
0
On 2
<empty string>
200
On 3
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
200
Progress 3
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
On 4
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
200
Done2 <h1>Adelino Amaral</h1>
<div>Olá Turma</div>
Done 4
<h1>Adelino Amaral</h1>
<div>Olá Turma</div>
```

14. Executa o código e observa o resultado na consola.

15. Altera apenas o código do botão no ficheiro app.js

```
btn.onclick = () => {
    const xhr = new XMLHttpRequest();
    xhr.onload = () => main.innerHTML = xhr.responseText
    xhr.open('GET', url);
    xhr.send()
};
```

16. Executa o código e observa o resultado na consola.

2 Bibliografia

AJAX Introduction. [em linh a] Consultado em 31-01-2023. Disponível em: https://www.w3schools.com/js/js_ajax_intro.asp

Javascript AJAX. [em linha] Consultado em 31-01-2023. Disponível em: https://www.tutorialrepublic.com/javascript-tutorial/javascript-ajax.php

AJAX Tutorial. [em linha] Consultado em 31-01-2023. Disponível em: https://www.devmedia.com.br/ajax-tutorial/24797

ReadyStates JavaScript Code XMLHttpRequest for AJAX 0-4 for xHR object. Consultado em 4-02-2023. Disponível em: https://www.youtube.com/watch?v=6VJDqUWxjLI