

AJAX =



?

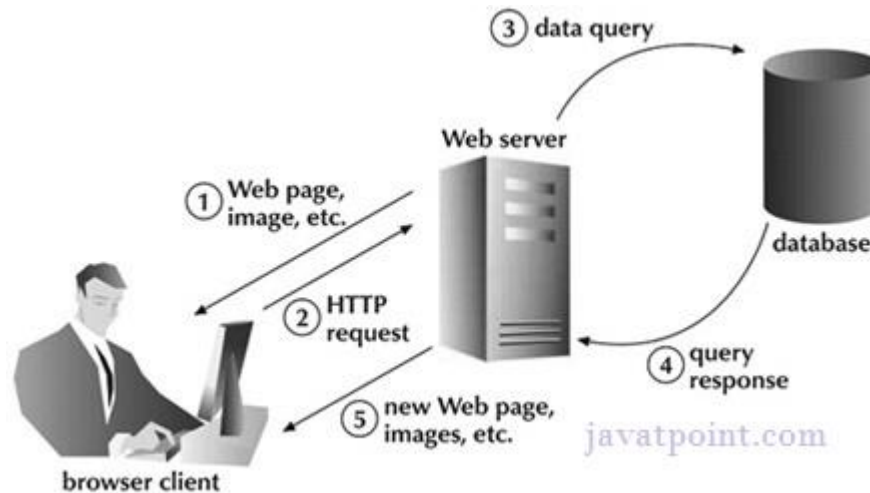
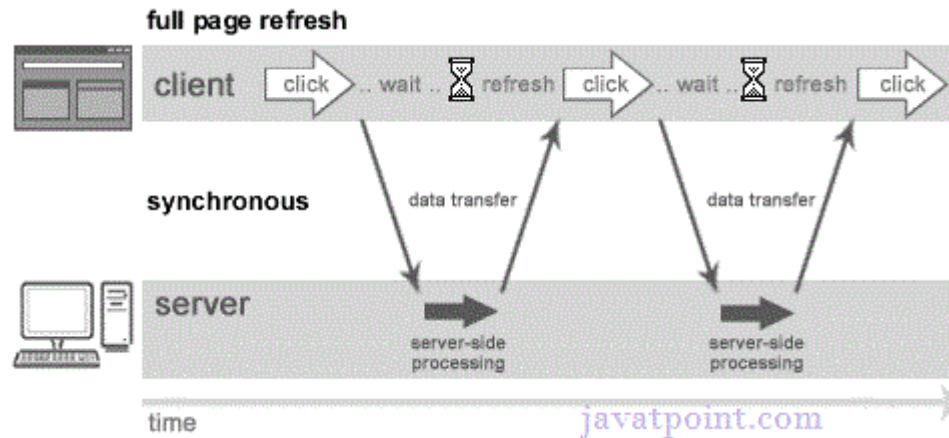
AJAX = Asynchronous Javascript and XML: Conjunto de tecnologias de desenvolvimento Web que actuam em conjunto para proporcionar aplicações Web com um alto nível de interatividade

Permite comunicações assíncronas com o servidor

Principais tecnologias usadas:

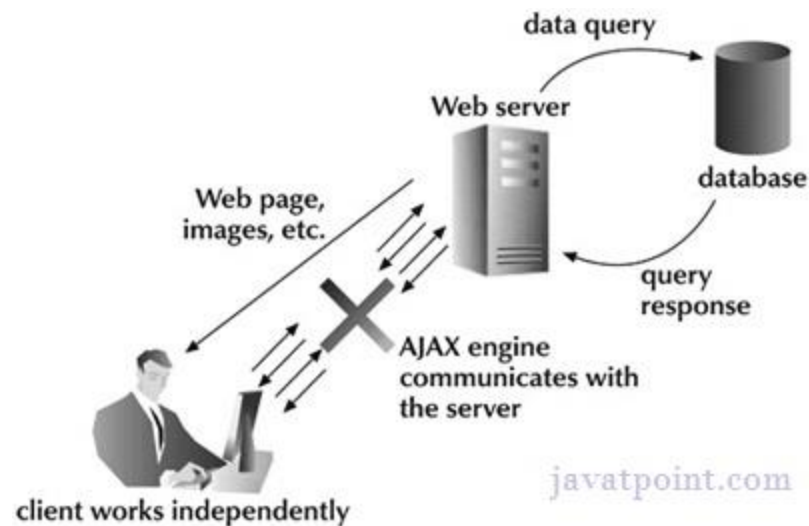
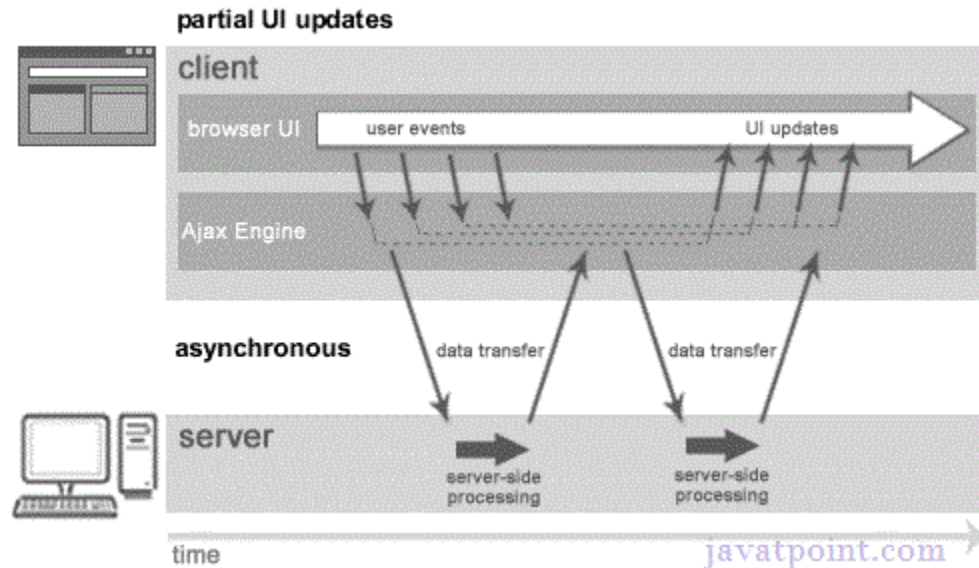
- XHTML / CSS
- DOM: Document Object Model
- JavaScript
- XMLHttpRequest (comunicação assíncrona com servidor)
- XML: Extensible Markup Language
- Fetch API
- JSON

# Antes do AJAX a comunicação cliente servidor era síncrona

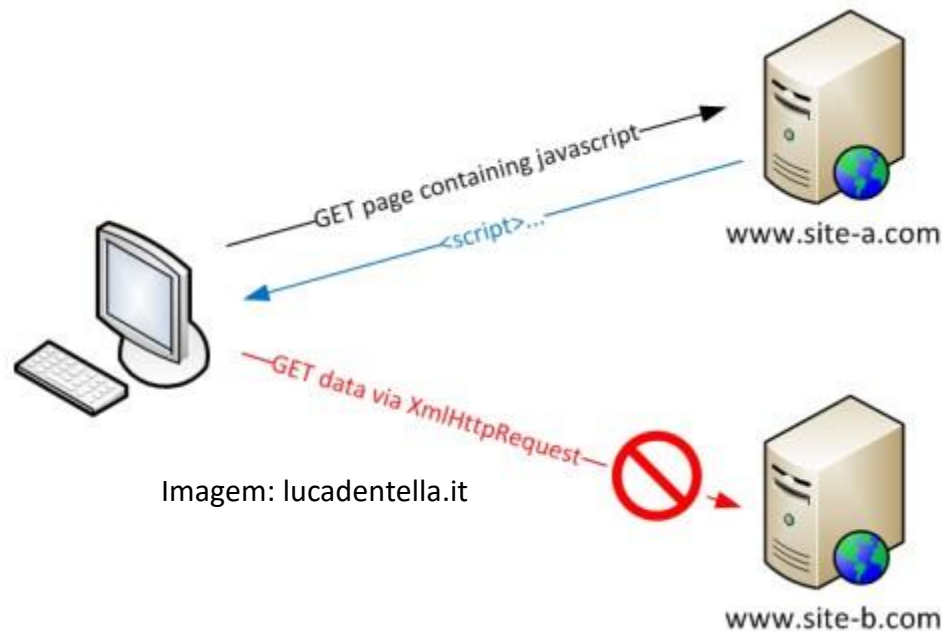


Imagens: javatpoint.com

# Com AJAX a comunicação passa a ser assíncrona



# Same Origin Policy – Proteção mas também limitação

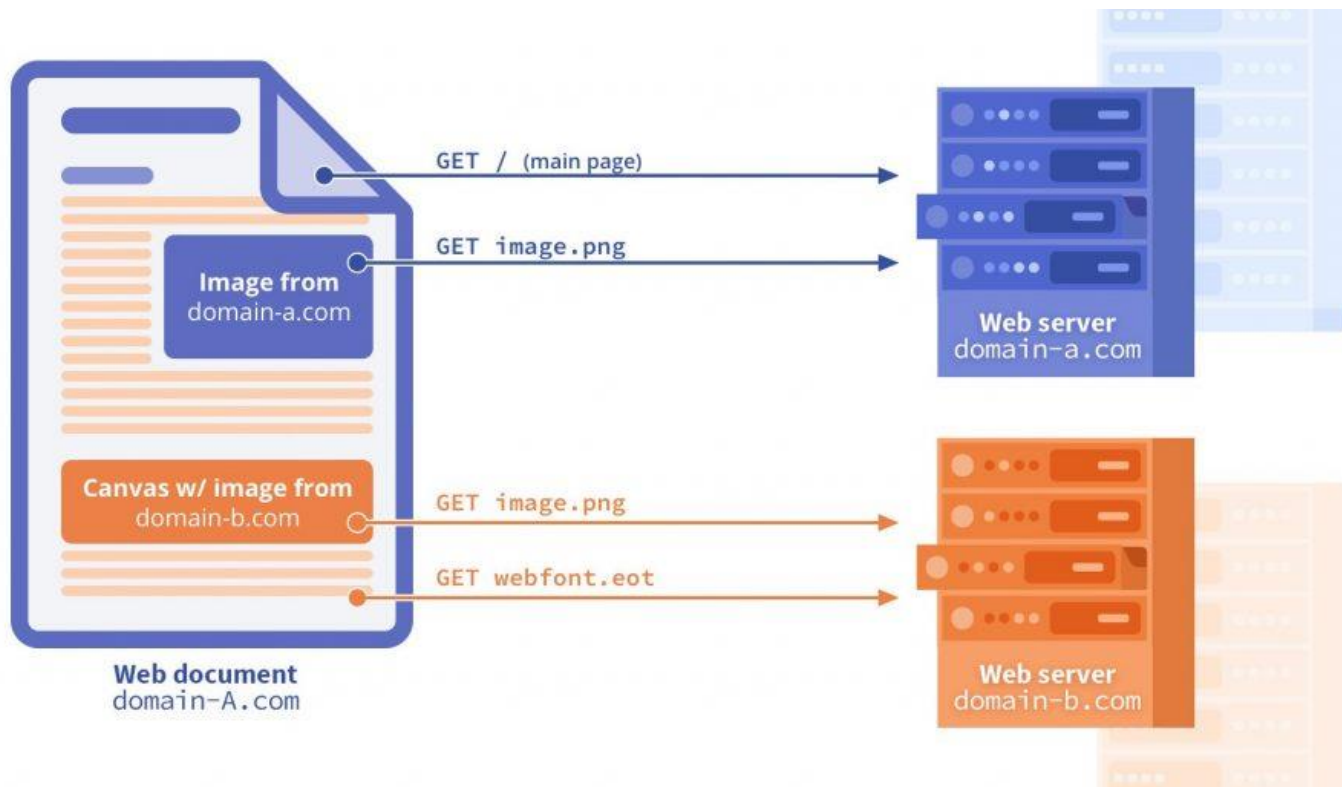


## ■ “Same Origin Policy “

- Política de segurança dos browsers que só permite comunicações assíncronas AJAX a partir de uma página, para o servidor de onde veio essa página (mesmo protocolo, mesmo domínio e mesmo port)
- Caso seja permitido fazer pedidos a qualquer servidor, isso pode criar vulnerabilidades de segurança
- Durante muito tempo, não era possível de todo aceder por AJAX a outros servidores. Contudo, atualmente, em determinadas condições é possível interagir com servidores de outros domínios

# CORS: Cross-Origin Resource Sharing.

- CORS - Cross-Origin Resource Sharing
  - É uma política que permite aceder a servidores de outros domínios – requer determinados cabeçalhos HTTP da parte do servidor remoto
  - Com o CORS, é possível aceder a servidores remotos bloqueando ao mesmo tempo atividades maliciosas

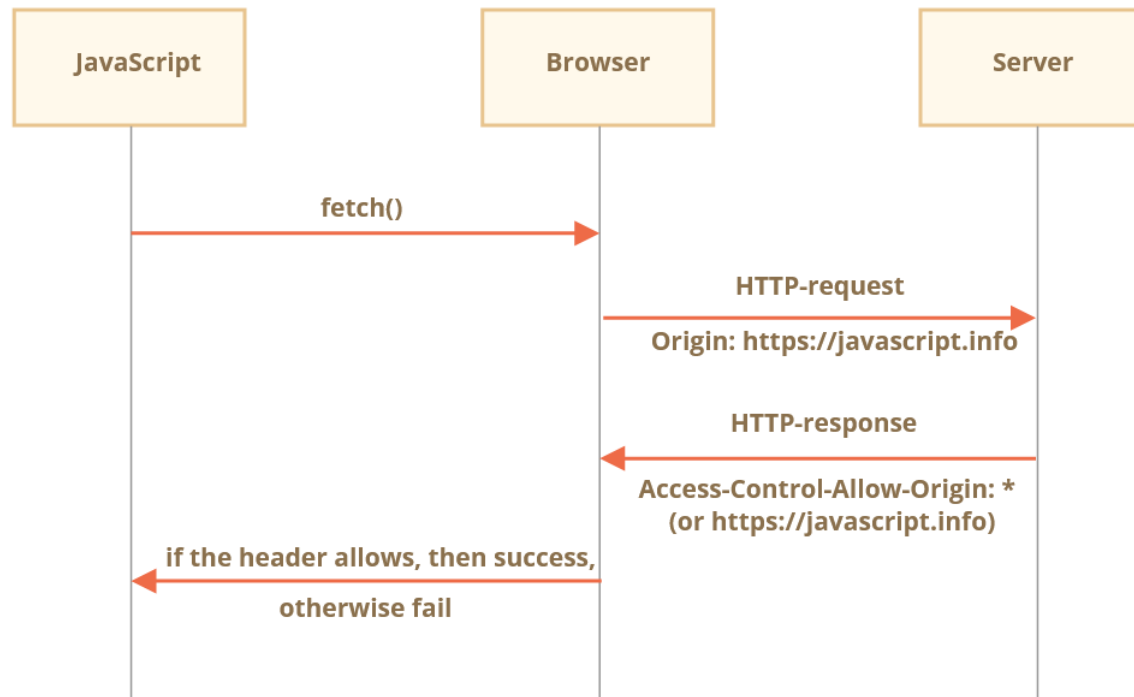


# CORS: pedidos seguros (safe requests)

- Um pedido é classificado como seguro, se satisfizer ao mesmo tempo estas duas condições
  - Tem que usar um método seguro: GET, POST ou HEAD
  - Os únicos cabeçalhos permitidos são:
    - **Accept**,
    - **Accept-Language**,
    - **Content-Language**,
    - **Content-Type** com o valor **application/x-www-form-urlencoded**, **multipart/form-data** ou **text/plain**.
    - **Origin**
- Todos os outros pedidos são considerados inseguros

# CORS: como é feito um pedido seguro

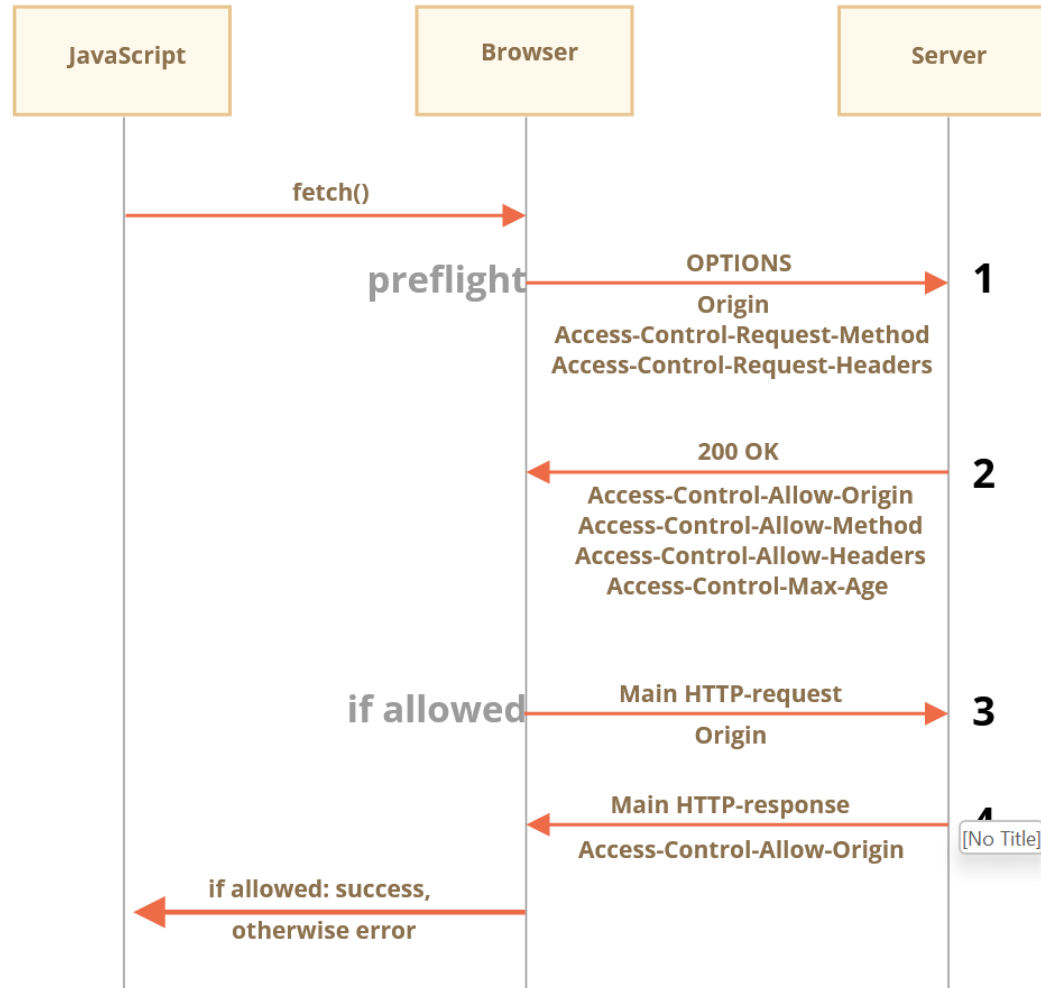
- Num pedido seguro, o browser acrescenta um cabeçalho Origin, com o domínio de onde a página foi descarregada
- O servidor analisa o conteúdo do cabeçalho Origin e se concordar em aceitar o pedido, responde com um cabeçalho Access-Control-Allow-Origin
- No fundo, o browser tem um papel de mediador de segurança





# CORS: como é feito um pedido inseguro

- Num pedido “inseguro”, é feito primeiro um ensaio do pedido , para verificar se o servidor aceita tal pedido
- Se o servidor aceitar, o pedido “a sério” é feito a seguir



A comunicação assíncrona com servidores Web pode ser feita através do objecto **XMLHttpRequest**

Este objecto possui os seguintes métodos:

- **open**("method","URL",async,"uname","pswd")
- **setRequestHeader**("label","value") (define um cabeçalho HTTP do pedido)
- **send**(content) (envia o pedido)
- **abort**() (cancela o pedido)
- **getAllResponseHeaders**() (obtém todos os cabeçalhos HTTP da resposta)
- **getResponseHeader**("headername") (obtém um determinado cabeçalho HTTP da resposta)

Este objecto possui as seguintes propriedades:

- **onreadystatechange** (event handler)
- **readyState** (estado da comunicação)
  - 0 = uninitialized
  - 1 = loading
  - 2 = loaded
  - 3 = interactive
  - 4 = complete
- **responseText** (resposta como texto)
- **responseXML** (resposta como XML)
- **status** (código de estado, exemplo: 404)
- **statusText** (texto de estado, exemplo: “not found”)

# XMLHttpRequest : exemplo em Javascript

```
xmlhttp=null;
xmlhttp=new XMLHttpRequest();
if (xmlhttp!=null)
{
    xmlhttp.onreadystatechange=state_Change; //callback

    xmlhttp.open("GET","/cotacoes_accoes.php?cotada=edp",true);

    xmlhttp.send(null);
}
else
{
    alert("Está na hora de instalar um browser que suporte XMLHTTP.");
}
```

# XMLHttpRequest : continuação do exemplo Javascript

```
function state_Change()  
{  
  if (xmlhttp.readyState==4)  
    {  
      // 4 = "loaded"  
      if (xmlhttp.status==200)  
        {  
          // 200 = "OK"  
          resposta=xmlhttp.responseXML;  
          //processar a resposta  
        }  
      else  
        {  
          alert("Problema a receber dados do servidor");  
        }  
    }  
}
```

# AJAX: felizmente em jQuery é tudo muito mais simples

Script a invocar

Parâmetros

Função que vai processar os dados quando eles chegarem

```
$.getJSON("/cotacoes_accoes.php", {cotada : edp}, function( data ) {  
  
    //aqui processam-se os dados  
  
});
```

- Os dados podem demorar desde alguns milissegundos até alguns segundos a chegar
- Nada garante que o servidor responda ou responda corretamente
- É preciso preparar o código para todas as eventualidades
- É fundamental definir um tempo máximo de espera pela resposta

# AJAX: alguns métodos jQuery

| <code>\$.ajax(options)</code>                            | Faz um pedido AJAX, permite várias opções     |
|----------------------------------------------------------|-----------------------------------------------|
| <code>\$.get(url,data,callback,type)</code>              | Carrega dados remotos usando o método GET     |
| <code>\$.post(url,data,callback,type)</code>             | Envia dados usando o método POST              |
| <code>\$.getJSON(url,data,callback)</code>               | Carrega dados remotos JSON (método GET)       |
| <code>\$.getScript(url,callback)</code>                  | Carrega e executa um script Javascript remoto |
| <code>\$(<i>selector</i>).load(URL,data,callback)</code> | Carrega dados remotos para um elemento        |

- Os dados podem chegar em vários formatos diferentes, desde texto simples até objetos JSON
- Pode ser necessário usar parsers para extrair determinadas partes desses dados

# Dados AJAX: XML versus JSON

## XML

```
<?xml version="1.0"?>
<book id="123">
  <title>Firewalls</title>
  <author>Alguem</author>
  <published>
    <by>FCA</by>
    <year>2007</year>
  </published>
</book>
```

## JSON

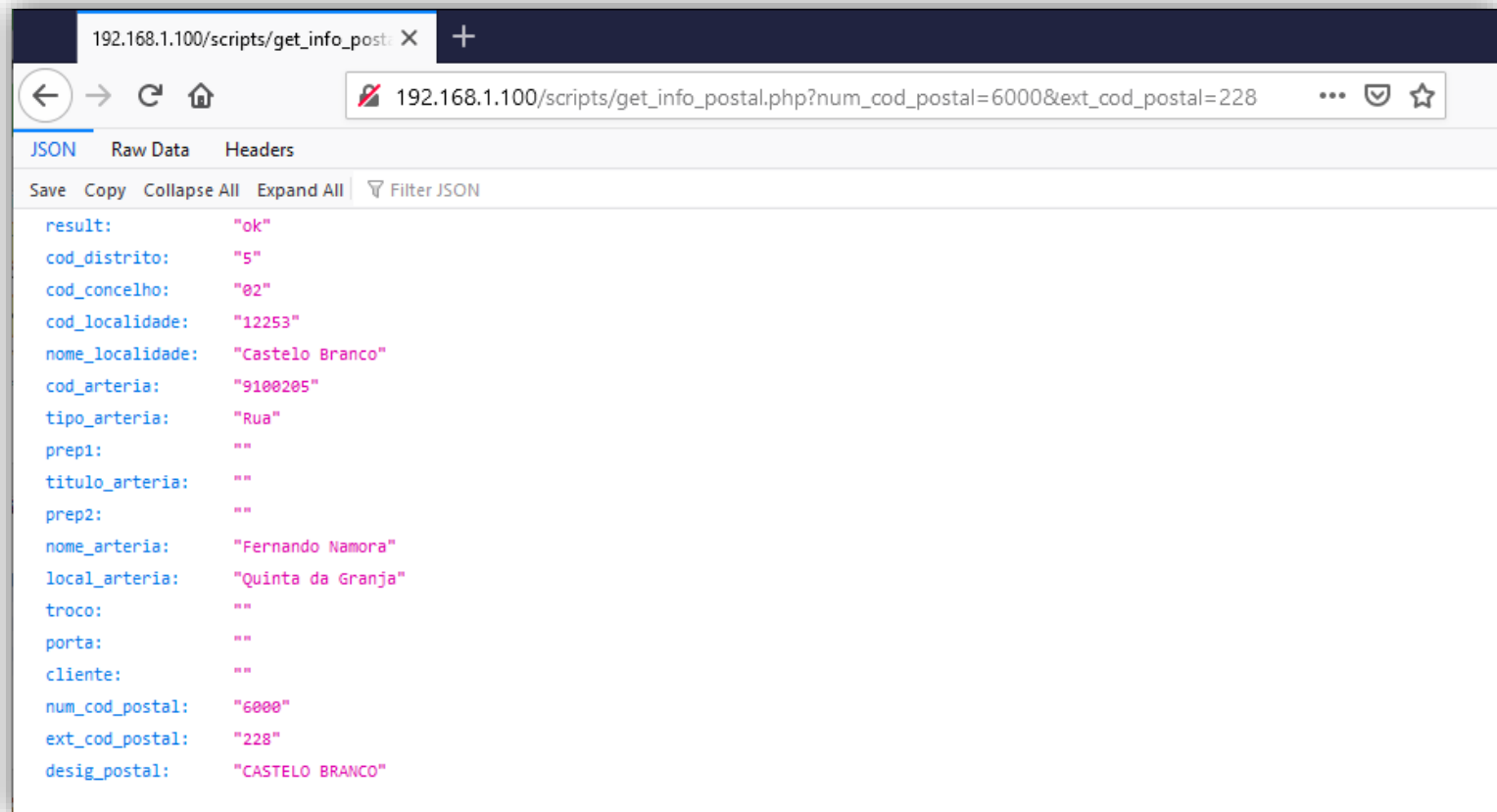
```
{
  "id": 123,
  "title": "Firewalls",
  "author": "Alguem",
  "published": {
    "by": "FCA",
    "year": 2007
  }
}
```

- Comparar XML com JSON é como comparar um SUV com uma bicicleta
- De uma forma geral:
  - Para estruturas simples, o JSON pode ser usado sem problemas
  - Para estruturas complexas ou com probabilidade de evoluírem para estruturas complexas, o XML é mais adequado



# Exercício

- Neste exercício vai criar uma página Web com uma secção para o utilizador digitar uma morada.
- Quando o utilizador escreve o código postal, a página deve obter dinamicamente através de AJAX o nome da localidade, da cidade e da rua, a partir de um servidor e da API fornecida pelo docente
- Exemplo de pedido e resposta:



```
192.168.1.100/scripts/get_info_postal.php?num_cod_postal=6000&ext_cod_postal=228

{
  "result": "ok",
  "cod_distrito": "5",
  "cod_concelho": "02",
  "cod_localidade": "12253",
  "nome_localidade": "Castelo Branco",
  "cod_arteria": "9100205",
  "tipo_arteria": "Rua",
  "prep1": "",
  "titulo_arteria": "",
  "prep2": "",
  "nome_arteria": "Fernando Namora",
  "local_arteria": "Quinta da Granja",
  "troco": "",
  "porta": "",
  "cliente": "",
  "num_cod_postal": "6000",
  "ext_cod_postal": "228",
  "desig_postal": "CASTELO BRANCO"
}
```

# Exercício

- Ver enunciado no moodle

# A API Fetch

- A API Fetch é uma alternativa moderna ao objecto XMLHttpRequest
- A grande diferença é que a API Fetch usa o conceito de Promises (promessas) em vez de callbacks complicadas
- De uma forma simples, uma promessa é uma operação assíncrona que ainda não foi completada

```
fetch(url)
.then(function(response) {

    //código caso o pedido tenha sido respondido com sucesso

})
.catch(function(error) {

    //código em caso de erro

});
```

# A API Fetch: opções de fetch

- É possível personalizar o pedido HTTP, com opções

```
let reqHeader = new Headers();

reqHeader.append('Content-Type', 'text/json');

let initObject = {
  method: 'GET', headers: reqHeader,
};

fetch('um URL qualquer', initObject)
  .then(function (response) {
    //.....
  })
  .catch(function (err) {
    //.....
  });
```

# A API Fetch – receber dados JSON

- Para receber diretamente dados JSON, pode ser usada esta forma

```
fetch(url)
.then(function(response) {
  return response.json()
})
.then(jsonData) {

  //dados JSON caso o pedido tenha sido respondido com sucesso

})
.catch(function(error) {

  //código em caso de erro

});
```

# Exercício

- Neste exercício, vai usar esta API REST: <https://restcountries.com/>
- A API devolve dados sobre Países
- O objetivo deste exercício é criar uma página Web com uma Caixa de texto, onde o utilizador escreve o nome de um País.
- A seguir, a página deve mostrar a bandeira e o nome da Capital desse país
- Deve ser usada a API Fetch

# Exercício

- Neste exercício, vai usar esta API REST: <https://weatherstack.com/>
- A API devolve dados sobre meteorologia
- O objetivo deste exercício é complementar o exercício anterior, acrescentando a previsão meteorológica para a capital do País
- Nota: para usar esta API de meteorologia, vai precisar de uma API Key. Neste exercício, pode embeber a API key no Código, mas convém ter a noção que em produção, **as API Keys nunca devem ser embebidas no Código, mas sim em variáveis de ambiente.**