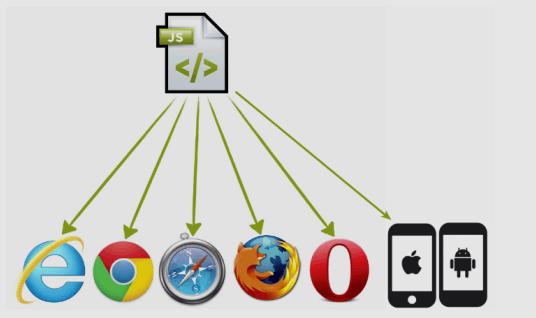
**Introdução**

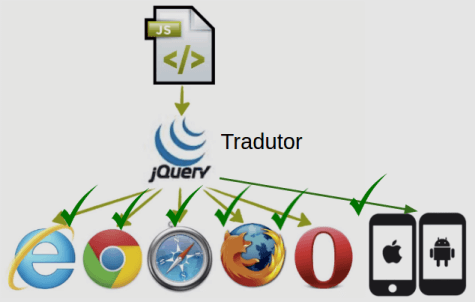
Construiremos uma aplicação “Plantier Typer”, um jogo que consegue medir a velocidade de digitação do usuário. O resultado é adicionado ao placar localizado mais abaixo, com o número de palavras escritas. Um dos objetivos é aprender algumas frases novas.

Hoje trabalhos como um variedades de navegadores que torna o desenvolvimento do JavaScript mais complexo, já que existem pequenas diferenças no seu funcionamento de cada navegador, fazendo com que um código seja funcional em um determinado navegador, mas com mau funcionamento em outro.



Neste cenário, devemos garantir que o JS rode nos mais diversos navegadores, aqui entra o famoso framework ***jQuery***.

O jQuery nos garante que suas funções funcionarão em diversos browsers, e nos dá a garantia que o código JS funcionará em um navegador específico. Ele age como um “tradutor”, tornando o código incompatível para certos navegadores em um código funcional:



*Vantagens*

- Compatibilidade entre navegadores.

- Produtividade. (Write less, do more).

Link oficial: <https://jquery.com/download/>

* *Qual é a diferença entre a versão comprimida e a não comprimida?*

Minificar é o processo de reduzir o tamanho de um arquivo fazendo certos processos como remover comentários, trocar nome de váriaveis por nomes menores, remover espaços entre outras técnicas.

A versão de *produção*, por ser minificada, tem um tamanho reduzido, sendo incrívelmente 30% do peso da versão de desenvolvimento. Utilizar a versão de produção quando estamos disponibilizando a versão final do nosso website para o usuário é de extrema importância, pois conseguimos onerar menos a banda de nossos usuários, algo que é bem interessante nos dias de hoje, já com o aumento dos acessos a website via mobile a maioria dos usuários depende de redes 3G e 4G, que são naturalmente mais limitadas.

Para desenvolver um website, é comum utilizarmos a versão de desenvolvimento, pois podemos ver como uma determinada função do jQuery foi implementada e fica mais fácil debuggar em caso de necessidade.

A versão de produção é minificada, já a de desenvolvimento vem em sua forma normal.

A versão slim do jQuery, não da suporte a AJAX e nem a animações.

* *Qual é a diferença entre .val() e .text()?*

Ambas as funções **.val()** e **.text()** podem manipular os valores de texto dos elementos, mas a **.val()** funciona em elementos de <input> que são campos aonde o usuário do site insere dados , como os campos de <input>(todos os tipos), <textarea> e <select>. Já a função **.text()** pega o **conteúdo de texto** de tags HTML que tem texto dentro, como as<h1>, <span> e <p>.

Ambas as funções podem **atribuir novos valores** a determinados elementos, ou apenas **pegar** os valores deles.

* *Evento focus*

É usado para identificar quando um usuário entra em um campo, não necessariamente digitando ou clicando.

* *Sobre as funções****.one()****e****.on()****do jQuery, marque as afirmativas verdadeiras:*

A função **.one()** funciona de modo semelhante a função **.on()**, ambas podem ser utilizadas em **qualquer** elemento, recebem **qualquer evento** como primeiro parâmetro e uma função anônima ou uma função nomeada como segundo parâmetro. A diferença entre elas é na hora de escutar os eventos, a função **.one()** escuta o evento apenas **uma única vez**, diferentemente da função **on()**, que como já vimos fica escutando o evento em um elemento do HTML por tempo ilimitado.

A função **.attr()**, como o próprio nome sugere é para alterar os atributos de elemento, como os atributos rows e col de uma textarea. Em tags HTML que não possuem o nome do atributo. É passado, o valor e true ou false. Exemplo: campo.attr("disabled", true);

A **.removeAttr()**, tem como objetivo remover atributos de elementos.

A função **.click()** é um atalho para a função **.on("click", ...)** . Ela tem o mesmo comportamento, apenas sendo um jeito mais curto e rápido de escrever a função. Os eventos mais comuns do jQuery possuem funções de atalho ( ou shorthand functions, como na documentação) , por exemplo as funções **.blur**, **.focus**, **.change**, **.dblclick**  etc. Todos eles equivalem a chamada da função **on()** passando-os como parâmetro, e existem no jQuery apenas para facilitar a vida do desenvolvedor.

*Manipulando o CSS via jQuery.*

* Para que usamos o CSS no desenvolvimento front-end?

HTML é usado para estruturar conteúdos da página, o CSS para formatar conteúdos estruturados. CSS é uma linguagem para estilos que define o layout de documentos HTML. Por exemplo, CSS controla fontes, cores, margens, linhas, alturas, larguras, imagens de fundo, posicionamentos e muito mais.

## A função .css() também pode ser usada para adicionar novas propriedades, por exemplo:

$("textarea").css("border","2px solid red");

## No entanto isso não é considerado boa prática e deve ser evitado. Por quê?

Para facilitar a manutenção devemos declarar as propriedades dentro de um arquivo CSS. Colocando as propriedades dentro do código JavaScript estamos violando a separação de responsabilidades. HTML é a estrutura da página, JavaScript a parte dinâmica e o CSS define o estilo. Boa prática então é definir os estilos previamente no CSS e usar apenas as classes no JavaScript.

*Para saber mais: Materialize e Bootstrap*

O Materialize é uma framework front-end que auxilia muito a estilizar páginas, evitando assim uma grande perda de tempo nessa tarefa. É útil para quem gostaria de seguir boas práticas no mundo CSS para por exemplo criar uma página responsiva sem entrar em detalhes. Link: <https://materializecss.com/>

O Materialize foi criado pela Google e segue o *Material Design* que é uma linguagem de design desenvolvida pela Google. Material Design é utilizado em vários produtos do Google e também é aplicado nos aplicativos Android.

Outro framework famoso nessa linha é o Bootstrap. Para ser correto o Bootstrap é o framework mais famoso para estilizar páginas. Link: <https://getbootstrap.com/>

Vamos obter um **erro clássico** de quando estamos trabalhando com jQuery. Tentamos utilizar uma função do **jQuery** em um **elemento do HTML**. Elementos comuns do HTML não tem acesso as funções do jQuery, precisamos empoderá-los com as funções do objeto jQuery, envolvendo-os com a função jQuery:

$(".botao-remover").click(event, function(){

$(this).remove();

})

Deste modo o this (elemento HTML que foi clicado), ganha acesso as várias funções do jQuery, como .text(),.val(), .css() e todas as outras funções do jQuery.

## Dado o seguinte HTML:

<p id="descricao">jQuery - write less, do more</p>

## Este código Javascript(com jQuery importado previamente):

$('#descricao').click(function() {

//AQUI

});

## Como podemos selecionar o texto do parágrafo no lugar de //AQUI com jQuery?

var texto = $(this).text();

Temos que lembrar que dentro de um evento do Javascript e do jQuery, a palavra reservada *this* sempre se refere ao elemento que sofreu o evento, neste caso ao elemento clicado. Como o elemento clicado é um elemento do HTML, ele se torna um objeto tradicional do Javascript dentro de nosso código. E como já sabemos, a função **.text()** é exclusiva de objetos jQuery, logo temos de convertê-lo antes.

Para converter um objeto tradicional do Javascript em um objeto jQuery, devemos fazer uso da função jQuery, passando o *this* para ela, deste modo:

var texto = $(this).text();

A função **.parent()** tem como objetivo **navegar pela árvore de elementos** do HTML. Ela por padrão retorna o **pai** do elemento em que foi chamada, logo no caso abaixo:

<div>

<h1>Olá</h1>

</div>

O pedaço de código $("h1").parent(); nos retorna a **div**, já que ela é o pai do h1. Podemos encadear várias chamadas a **.parent()** para buscar elementos mais acima, como no seguinte:

<section>

<div>

<h1>Olá</h1>

</div>

</section>

O código $("h1").parent().parent(); nos retornaria a **section**, que é como se fosse o avô do <h1>.

* *Como criar um novo elemento HTML (DOM) pelo jQuery?*

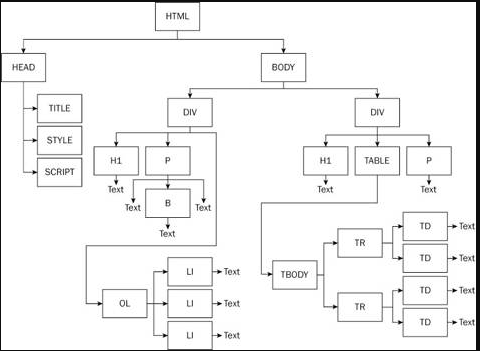
Para criarmos elementos do DOM com jQuery, devemos utilizar a própria função jQuery (**$**) , mas em vez de passarmos um *id*, *classe*, ou *nome de um elemento* para ela buscar, devemos passar uma tag HTML completa( com os sinais < e >), deste modo:

var itemDaLista = $("<li>");

Depois de criado um elemento na memória do navegador, podemos adicioná-lo a página com algumas das várias funções do jQuery, por exemplo com a **append().**

*O que é um Objeto DOM?*

O Document Object Model é utilizado pelo navegador web para representar a sua página web. Quando altera-se esse modelo com o uso do Javascript altera-se também a página web. É mais fácil trabalhar com DOM do que diretamente com o código HTML ou CSS. Com DOM pode-se mover itens dentro de uma página ou criar efeitos CSS sem precisar nem mesmo recarregar a página. Exemplo de uma árvore DOM:



A função **fadeOut**, que vai esmaecendo o elemento até o ponto dele desaparecer, mas ele continua no HTML.

Assim como existem o slideUp, slideDown e slideToggle, existem funções semelhantes que executam o *fade*, o fadeIn, fadeOut e fadeToggle, respectivamente.

Antes de realizarmos a animação, paramos a que estiver acontecendo através da função **stop** do jQuery. Essa função faz exatamente o que precisamos, a animação que estiver acontecendo no momento é interrompida, e uma próxima é iniciada. A função stop, controla melhor as nossas animações.

Se queremos achar em que posição (na página) o placar se encontra, basta selecionarmos-o e chamar a função offset:

$(".placar").offset();

Essa função nos retorna a distância em que o elemento está do **topo** e da **esquerda** da página. Se acessarmos o valor do topo (*top*), teremos o valor exato para onde queremos *scrollar* a página.

Para animar algo, utilizamos a função **animate** do jQuery. Ela recebe dois parâmetros, um **objeto** que contém as propriedades CSS a serem animadas e os seus valores, e o **tempo de duração** da animação.

Para conseguirmos fazer o *scroll* do body, utilizaremos a propriedade scrollTop, ela recebe um valor em pixels, que representa a posição da nossa página para onde desejamos *scrollar* a página.

A função **toogleClass** (não confundir com a função toggle) adiciona uma classe caso ela não exista no elemento. Se existir, ela remove a classe.

A função **toggle** é um atalho para as funções **hide** e **show**. Quando ela é chamada para um elemento visível, o elemento fica invisível. Quando é chamada para um elemento invisível, ela torna o elemento visível.

$('li').dblclick(function() {

var $this = $(this);

$this.fadeOut(function() {

$this.remove();

})

});

Para evitar de ter que fazer duas vezes $(this) Luciano decidiu guardar o this promovido para jQuery Object na variável $this. O prefixo $ deixa claro para ela que é um jQuery Object e não um this ordinário. Para que a remoção seja feito só depois do efeito de fadeOut ter terminado, ela passou uma função para a função fadeOut. Essa função é chamada só quando fadeOut termina. Sendo assim, na função ela pôde chamar $this.remove() com a certeza que o efeito de fadeOut já foi processado.

* *Como faremos para gerar essa frase aleatória?*

Vamos buscar a frase em um servidor, que estará rodando na nossa própria máquina, e nesse servidor terão diversas frases. Ou seja, faremos uma **requisição** para o servidor, mas de modo **assíncrono** (sem atrapalhar o fluxo da nossa aplicação), utilizando **AJAX**.

Utilizaremos o servidor Node.js. O [Node.js](https://nodejs.org/) é um ambiente JavaScript multiplataforma disponível para Linux, Mac e Windows. Link: <https://nodejs.org/en/>

O jQuery disponibiliza diversas funções para fazermos requisições AJAX, mas nós utilizaremos aqui a **$.get**, por quê? Dentre os tipos de requisições HTTP, como POST, DELETE, PUT, fazemos um GET quando acessamos o servidor no navegador, através da URL <http://localhost:3000/frases>.

* O que é AJAX

O AJAX, como a própria sigla( Asynchronous Javascript and XML) diz é assíncrono, ou seja, ele envia a requisição de modo paralelo ao nosso Javascript, sem travá-lo esperando a resposta.

* *Quais as vantagens de utilizarmos requisições AJAX ?*

O AJAX nos permite carregar apenas uma parte do conteúdo da página, fazendo com que o site do usuário fique mais leve , e por consequência mais rápido na hora de carregar. Exemplo: é muito comum encontrarmos sites que abusam desta técnica, conhecida como Infinte Scrolling, ou scroll infinito em português claro. O Facebook, carrega apenas uma parte das postagens de seu mural, e quando o usuário vai se aproximando do fim da página ele carrega mais postagens assincronamente e os adiciona embaixo das já existentes! Um uso muito esperto do AJAX, assim o usuário não precisa carregar todas as postagens de todos os seus amigos antes de abrir a página!

Com o AJAX também conseguimos enviar dados para o servidor sem depender de um form do HTML.

Com requisições AJAX conseguimos trazer novos dados para o usuário sem que ele precise trocar de página. Ou seja, conseguimos atualizar os dados da página com novas informações mesmo após o usuário já ter carregado a página.

FALSO: Requisições AJAX são mais rápidas que requisições normais.

--

Quando usamos **$.get** precisamos passar como primeiro parâmetro o endereço do que desejamos consumir via Ajax. O segundo parâmetro é aquela função que será chamada automaticamente por $.get assim que os dados retornarem do servidor. Esse retorno pode demorar um ou mais segundos, nunca sabemos quando ela será terminada, por isso dizemos que $.get executa um código assíncrono. O servidor retorna um array.

Quais dos códigos abaixo faz uma requisição AJAX, utilizando um GET, para o endereço http://localhost:3000/frases e imprime corretamente os dados deste endereço no console?

Quando a função **$.get** vai retornar dados para que sejam utilizados pelo usuário, é preciso colocar nos argumentos da função que é chamada ao sucesso da requisição uma variável que irá conter os dados recebidos. Normalmente colocamos um nome semântico, como dados ou data, para indicar que aquele é o resultado obtido da requisição AJAX:

// Como função anônima

$.get("http://localhost:3000/frases",function(dados){

console.log(dados);

});

// Como função nomeada

$.get("http://localhost:3000/frases", imprimeConsole);

function imprimeConsole(dados){

console.log(dados);

}

## Mensagem quando AJAX falha

A nossa mensagem deve aparecer quando a requisição AJAX realmente falha. Vamos enfileirar a chamada da função **fail** na função .get. **fail** recebe uma função anônima com o código que é

Executado com um erro acontece:

function fraseAleatoria() {

$.get("http://localhost:3000/frases", trocaFraseAleatoria)

.fail(function(){

$("#erro").toggle();

setTimeout(function(){

$("#erro").toggle();

},1500);

});

}

O jQuery oferece uma função chamada **always** que podemos chamar depois do fail. Ela é executada sempre, independentemente de erro ou não.

* *Por que é bom para o usuário implementarmos a feature do spinner ?*

A principal ideia de implementar um spinner como um elemento visual após uma requisição AJAX é informar ao usuário que o pedido dele está sendo processado, e exibir visualmente um ícone clássico que simboliza isto. É uma questão de melhorar a UX(User eXperience) do usuário na aplicação, algo que é muito importante na construção de qualquer sistema hoje em dia.

**Consultando uma frase específica**

*var parametros = {id: 5};*

Usamos id porque o servidor espera receber id. Quem define o nome do parâmetro é quem criou o servidor.

$.get('http://localhost:3000/frases', parametros, function(retorno) {

console.log(retorno); // imprime a frase!

});

Nesta altura do campeonato você já deve ter compreendido que para consumirmos dados de um servidor precisamos conhecer esse servidor. Geralmente quem construiu o servidor gera uma documentação que lista todos os endereços possíveis, quais verbos do HTTP usar e seus parâmetros. Se você é a pessoa que criou o servidor, fica mais fácil ainda saber os endereços e parâmetros.

Para conseguirmos enviar dados via AJAX, com jQuery, é preciso passar os dados como uma **String** ou um **Objeto** JavaScript simples, como o segundo parâmetro da função **$.get()**. Também é preciso receber como parâmetro da função de sucesso uma variável que irá conter o retorno do GET.

Vamos então implementar um POST para salvar os dados toda vez que o usuário clicar no botão sincroniza e um GET para carregar o placar quando o usuário entrar na página. O verbo/método post é usado quando desejamos enviar os dados para o servidor. Já o get quando queremos ler um dado.

## **Percorrendo as linhas de modo inteligente: a função $.each()**

Agora que temos todas as linhas, precisamos percorrer cada linha obtendo o nome de usuário e de palavras. Poderíamos utilizar um for tradicional do Javascript para percorrer o array linhas, porém vamos utilizar um recurso do jQuery que é a função each():

function sincronizaPlacar(){

var placar = [];

var linhas = $("tbody>tr");

linhas.each(function(){

});

}

A função each() executa a ação da função passada por parâmetro para cada item do array em que ela foi chamada, no caso, o array linhas.

Para buscar o <td> que é o primeiro filho do <tr> da iteração que estamos, podemos utilizar uma função conhecida nossa do jQuery, a função .find(), que faz as buscas nos elemento filhos. Porém não podemos utilizar o comando this.find(), pois o this neste caso é um objeto do Javascript que representa o elemento do HTML da linha, o <tr>. Precisamos empoderar este objeto Javascript e transformá-lo em um objeto jQuery atráves da função jQuery, assim ela terá acesso a função .find():

function sincronizaPlacar(){

var placar = [];

var linhas = $("tbody>tr");

linhas.each(function(){

var usuario = $(this).find();

var palavras = $(this).find();

});

}

## **Fazendo um POST com Ajax: enviando o placar**

Precisamos realizar um AJAX,para que a página do usuário não recarregue, e tem que ser um POST para que os dados sejam gravados definitivamente.

Para realizar um post AJAX com jQuery , podemos utilizar a função $.post:

function sincronizaPlacar(){

var placar = [];

var linhas = $("tbody>tr");

linhas.each(function(){

var usuario = $(this).find("td:nth-child(1)").text();

var palavras = $(this).find("td:nth-child(2)").text();

var score = {

usuario: usuario,

pontos: palavras

};

placar.push(score);

$.post("http://localhost:3000/placar");

});

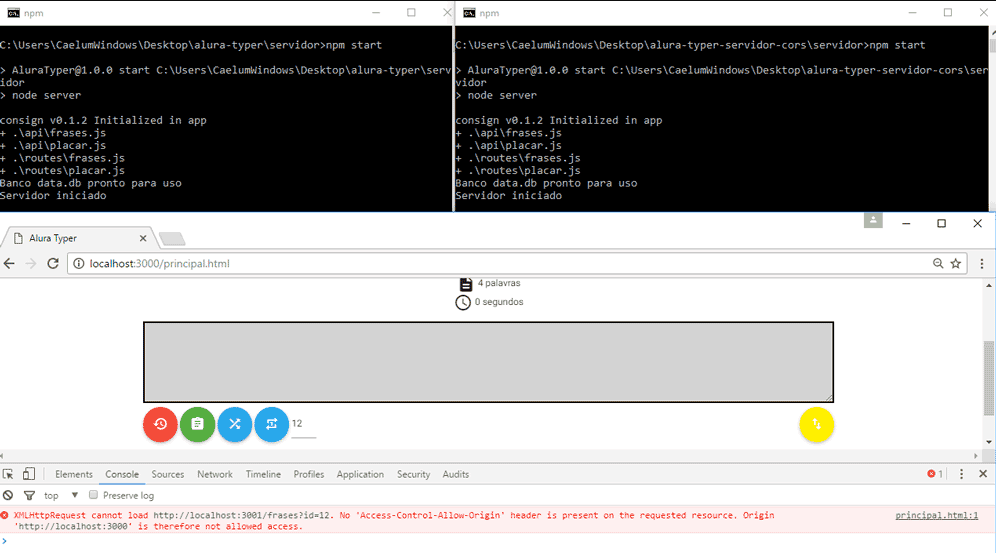
}

Passamos o endereço para qual queremos fazer o POST, mas também precisamos passsar os dados que queremos enviar, neste caso o nosso placar, e uma função para ele executar após o sucesso da requisição.

## **Carregando o placar junto da página**

Relembrando as funções: $.get, .each(), .append(), $(function(){})

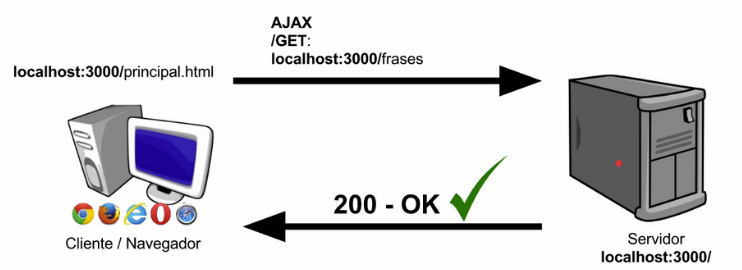
## Analisando o erro abaixo:

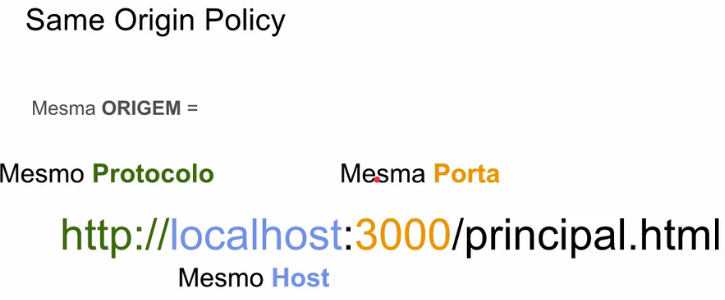
****

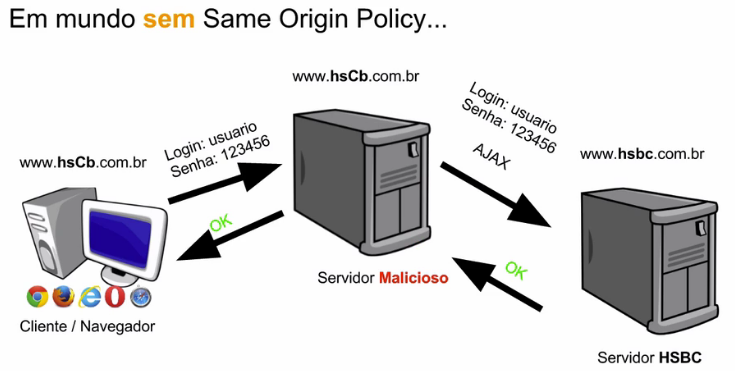
Ao analisar o console percebemos que o navegador simplesmente não permitiu a requisição para outro servidor. Isto é pois o navegador possui uma proteção que é chamada de Same Origin Policy.

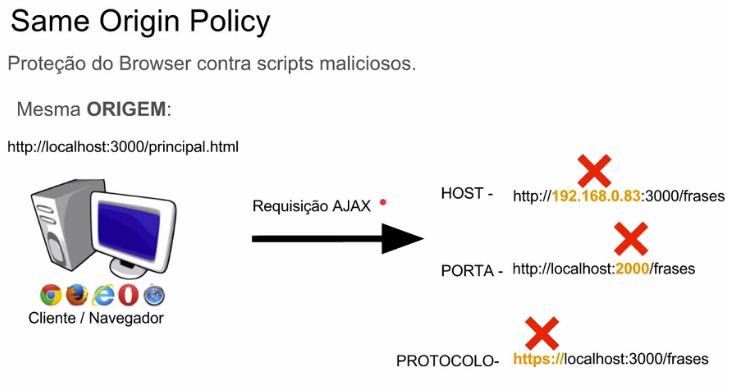
Same Origin significa que por padrão o navegador não permite chamar um outro servidor que não é da mesma origem. Ou seja, se a aplicação foi carregada pelo localhost:3000, o navegador só permite requisições para localhost na porta 3000. Até o protocolo importa, por exemplo o navegador não permite mudar a origem do http para https.

Isso tudo existe para evitar fraudes e é uma forma de impedir que o usuário use um site que na verdade não representa a origem.









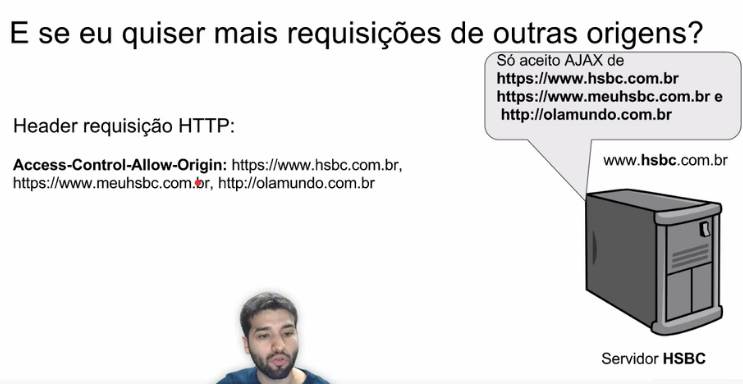
## 

## CORS - Cross-Origin Resource Sharing

É caracterizado por uma informação especial no header de resposta do servidor.

Veja o processo de habilitar ou não CORS é de responsabilidade do programador back-end. Sendo assim, o programador front-end pode solicitar sua habilitação, mas claro, quando o desenvolvedor back-end ver sentido nisso.





Voltando para nossa aplicação, então é impossível carregar as frases do outro servidor? Não é, mas nesse caso é preciso configurar as outras origens no servidor. Aqui o outro servidor (não a origem que já funciona com AJAX) adiciona um cabeçalho na resposta HTTP e baseado nessa resposta o navegador permitir uma requisição para outra origem.

O cabeçalho é bem simples e faz parte do protocolo HTTP:

Access-Control-Allow-Origin: http://localhost:3000, http://192.168.0.83:3000

Essa forma de permitir chamar uma outra origem também é chamado de Cross-Origin Resource Sharing ou CORS.

Repare então que isso não é uma configuração do jQuery ou AJAX em geral. Isso é algo que o servidor precisa se preocupar e adicionar na resposta HTTP. Como nosso foco aqui é jQuery não vamos dar muitos detalhes pois depende muito da linguagem e framework utilizado no lado do servidor.

No nosso caso escrevemos a aplicação servidora em node.js (JavaScript) usando o framework Express. Para habilitar CORS com Express basta utilizar:

app.use(function(req, res, next) {

res.header("Access-Control-Allow-Origin", "http://localhost:3000, http://192.168.0.83:3000");

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

next();

});

Mais infos sobre Express no link abaixo e nos nossos cursos sobre node.js!

<http://enable-cors.org/server_expressjs.html>

 Existem funcionalidades que outros desenvolvedores criaram, úteis para o dia a dia, oferecendo pequenas bibliotecas ou plugins para acrescentar valor e melhorar ainda mais o jQuery.

## **Aplicando carrossel com slick**

Existem milhares de plugins, até vários com o mesmo propósito. Nesse curso escolhemos o plugin slick:

<http://kenwheeler.github.io/slick/>

É bem com nas páginas de bibliotecas externas, ter a opção *usage, dowload ou get started* que é como usar o plugin disponibilizado.

Parei com 3 minutos e 25 segundos.