01

Same origin Policy e CORS

Transcrição

Agora que aprendemos bastante sobre AJAX é hora de conversar sobre um pequeno detalhe que é preciso se preocupar quando trabalhamos com AJAX.

Lembrando que nossa aplicação Alura Typer está hospedada no servidor que levantamos na linha de comando. Ou seja, na hora de acessar http://localhost:3000/principal.html (http://localhost:3000/principal.html) o servidor deveria estar em pé para servir a página ao navegador

Também já mostrei para vocês que existe um outro servidor na rede que também está rodando a aplicação. Essa aplicação é a mesma, o mesmo código rodando, no entanto em uma máquina diferente e por isso é necessário trocar `localhost` pelo IP de maquina, no nosso caso `192.168.0.83`

Acessando as frases com outro IP

Se a aplicação nessa outra máquina possui as mesmas funcionalidades, também podemos acessar `/frases` para ver todas as frases disponíveis. Não há mistério e funcionou. Ao acesssar http://192.168.0.83/frases (http://192.168.0.83/frases) aparece a lista de frases dessa aplicação.

Vamos pensar que devemos puxar as frases do outro servidor que não é o local. Isso é algo muito comum na web. Em outras palavras, queremos pegar dados de mais de um servidor! A nossa primeira ideia é simples: No arquivo `frase.js`, na função `fraseAleatoria` vamos usar o endereço do outro servidor:

```
//frase.js, na função fraseAleatoria
$.get("http://192.168.0.83/frases". trocaFraseAleatoria)
//resto omitido
```

Como tem no outro servidor a mesma aplicação com mesmo código, o esperado é que a requisição funcione! Para nossa surpresa ela falha! O que funciona em uma aplicação não funciona na outra! Como assim?

Entendendo a Same Origin Policy

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

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 (http://enable-cors.org/server_expressjs.html)

O que aprendemos?

- Um dos erros mais famosos quando utilizamos AJAX
- O que é o Same Origin Policy
- A proteção do Javascript contra scripts maliciosos
- O que é o CORS
- CORS é uma configuração no servidor