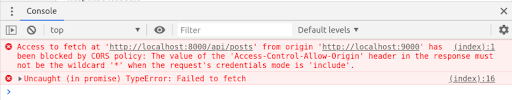
Quando desenvolvemos APIs e queremos que todos os seus recursos fiquem disponíveis a qualquer cliente HTTP, uma das coisas que vem à nossa cabeça é o CORS (*Cross-Origin Resource Sharing*), em português, “compartilhamento de recursos com origens diferentes”. Se ainda não aconteceu com você, fique tranquilo, é normal termos erros de CORS na hora de consumir e disponibilizar APIs.



Mas afinal, o que é CORS, o que causa os erros e como evitá-los em nossas APIs com Spring Boot?

**CORS**

O CORS é um mecanismo utilizado para adicionar cabeçalhos HTTP que informam aos navegadores para permitir que uma aplicação Web seja executada em uma origem e acesse recursos de outra origem diferente. Esse tipo de ação é chamada de *requisição cross-origin HTTP*. Na prática, então, ele informa aos navegadores se um determinado recurso pode ou não ser acessado.

Mas por que os erros acontecem? Chegou a hora de entender!

**Same-origin policy**

Por padrão, uma aplicação Front-end, escrita em JavaScript, só consegue acessar recursos localizados na mesma origem da solicitação. Isso acontece por conta da política de mesma origem (*same-origin policy*), que é um mecanismo de segurança dos Browsers que restringe a maneira de um documento ou script de uma origem interagir com recursos de outra origem. Essa política possui o objetivo de frear ataques maliciosos.

Duas URLs compartilham a mesma origem se o protocolo, porta (caso especificado) e host são os mesmos. Vamos comparar possíveis variações considerando a URL https://cursos.alura.com.br/category/programacao:

| **URL** | **Resultado** | **Motivo** |
| --- | --- | --- |
| <https://cursos.alura.com.br/category/front-end> | Mesma origem | Só o caminho difere |
| <http://cursos.alura.com.br/category/programacao> | Erro de CORS | Protocolo diferente (http) |
| <https://faculdade.alura.com.br:80/category/programacao> | Erro de CORS | Host diferente |

Agora, fica a dúvida: o que fazer quando precisamos consumir uma API com URL diferente sem termos problemas com o CORS? Como, por exemplo, quando queremos consumir uma API que roda na porta 8000 a partir de uma aplicação React rodando na porta 3000. Veja só!

Ao enviar uma requisição para uma API de origem diferente, a API precisa retornar um header chamado **Access-Control-Allow-Origin**. Dentro dele, é necessário informar as diferentes origens que serão permitidas para consumir a API, em nosso caso: Access-Control-Allow-Origin: http://localhost:3000.

É possível permitir o acesso de qualquer origem utilizando o símbolo \*(asterisco): Access-Control-Allow-Origin: \*. Mas isso não é uma medida recomendada, pois permite que origens desconhecidas acessem o servidor, a não ser que seja intencional, como no caso de uma API pública. Agora vamos ver como fazer isso no Spring Boot de maneira correta.

**Habilitando diferentes origens no Spring Boot**

Para configurar o CORS e habilitar uma origem específica para consumir a API, basta criar uma classe de configuração como a seguinte:

@Configuration

**public** **class** **CorsConfiguration** **implements** **WebMvcConfigurer** {

@Override

**public** void **addCorsMappings**(CorsRegistry registry) {

registry.**addMapping**("/\*\*")

.**allowedOrigins**("http://localhost:3000")

.**allowedMethods**("GET", "POST", "PUT", "DELETE", "OPTIONS", "HEAD", "TRACE", "CONNECT");

}

}

COPIAR CÓDIGO

[**http://localhost:3000**](http://localhost:3000/) seria o endereço da aplicação Front-end e **.allowedMethods** os métodos que serão permitidos para serem executados. Com isso, você poderá consumir a sua API sem problemas a partir de uma aplicação Front-end.