**Relatório Técnico: CORS (Cross-Origin Resource Sharing)**

Gabriel Vieira Zangarelli – 3ºDS

**1. Introdução**

O desenvolvimento de aplicações web modernas frequentemente envolve a comunicação entre clientes (navegadores) e servidores hospedados em domínios distintos. Essa comunicação entre origens diferentes (cross-origin) é, por padrão, restringida pelos navegadores como uma medida de segurança. Para permitir essas interações legítimas e seguras entre origens diferentes, utiliza-se o mecanismo chamado **CORS (Cross-Origin Resource Sharing)**.

**2. O que é CORS?**

CORS é um protocolo de segurança implementado pelos navegadores que permite ou bloqueia requisições feitas de um domínio diferente daquele onde o servidor está hospedado. Ele é definido por um conjunto de **cabeçalhos HTTP** que informam ao navegador se uma determinada requisição pode ser executada ou não.

Exemplo de cenário onde o CORS é necessário:

* Frontend hospedado em *https://meusite.com*
* API hospedada em *https://api.terceiro.com*

Por padrão, o navegador bloqueará a requisição da aplicação do ***meusite.com*** para a API ***api.terceiro.com***, a menos que o servidor da API permita explicitamente essa origem.

**3. Para que o CORS é utilizado na API**

No contexto de uma API, o CORS é utilizado para:

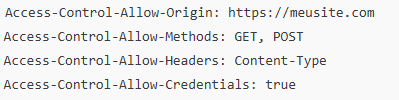
* **Controlar o acesso à API**: Definindo quais origens podem ou não consumir os recursos da API.
* **Prevenir ataques de segurança**, como o Cross-Site Request Forgery (CSRF), ao restringir solicitações não autorizadas.
* **Permitir integrações com clientes web**: Facilita o consumo da API por aplicativos hospedados em domínios diferentes.

**4. Como o CORS funciona**

Quando o navegador detecta uma requisição de origem cruzada, ele adiciona um cabeçalho ***ORIGIN*** à requisição. O servidor da API responde com um ou mais cabeçalhos CORS, como:

* ***Access-Control-Allow-Origin***: Define quais origens são permitidas. Pode ser um domínio específico ou \* (para permitir qualquer origem).
* ***Access-Control-Allow-Methods***: Informa os métodos HTTP permitidos (GET, POST, PUT, DELETE, etc.).
* ***Access-Control-Allow-Headers***: Lista de cabeçalhos personalizados que podem ser usados na requisição.
* ***Access-Control-Allow-Credentials***: Indica se cookies e credenciais podem ser incluídos na requisição.

**Exemplo de resposta CORS:**

****

**5. Pré-Requisições (Preflight)**

Para métodos HTTP diferentes de GET/POST simples ou quando se usam cabeçalhos personalizados, o navegador envia uma "preflight request" usando o método OPTIONS. Essa requisição verifica se o servidor permite a operação antes de enviar a requisição principal.

**6. Configuração do CORS**

A configuração do CORS varia conforme a tecnologia usada no backend. Exemplos:



* **Spring Boot (Java)**:

**7. Considerações de Segurança**

* Evite o uso de ***Access-Control-Allow-Origin:*** em APIs que exigem autenticação.
* Configure apenas as origens confiáveis.
* Teste as configurações de CORS com ferramentas apropriadas (como Postman ou navegadores com devtools).

**8. Conclusão**

O CORS é um mecanismo fundamental para o desenvolvimento seguro e funcional de aplicações web que interagem com APIs em diferentes domínios. Uma configuração adequada garante não apenas a funcionalidade esperada, mas também protege contra ataques e acessos indevidos.