TÓPICO 19 - CORS

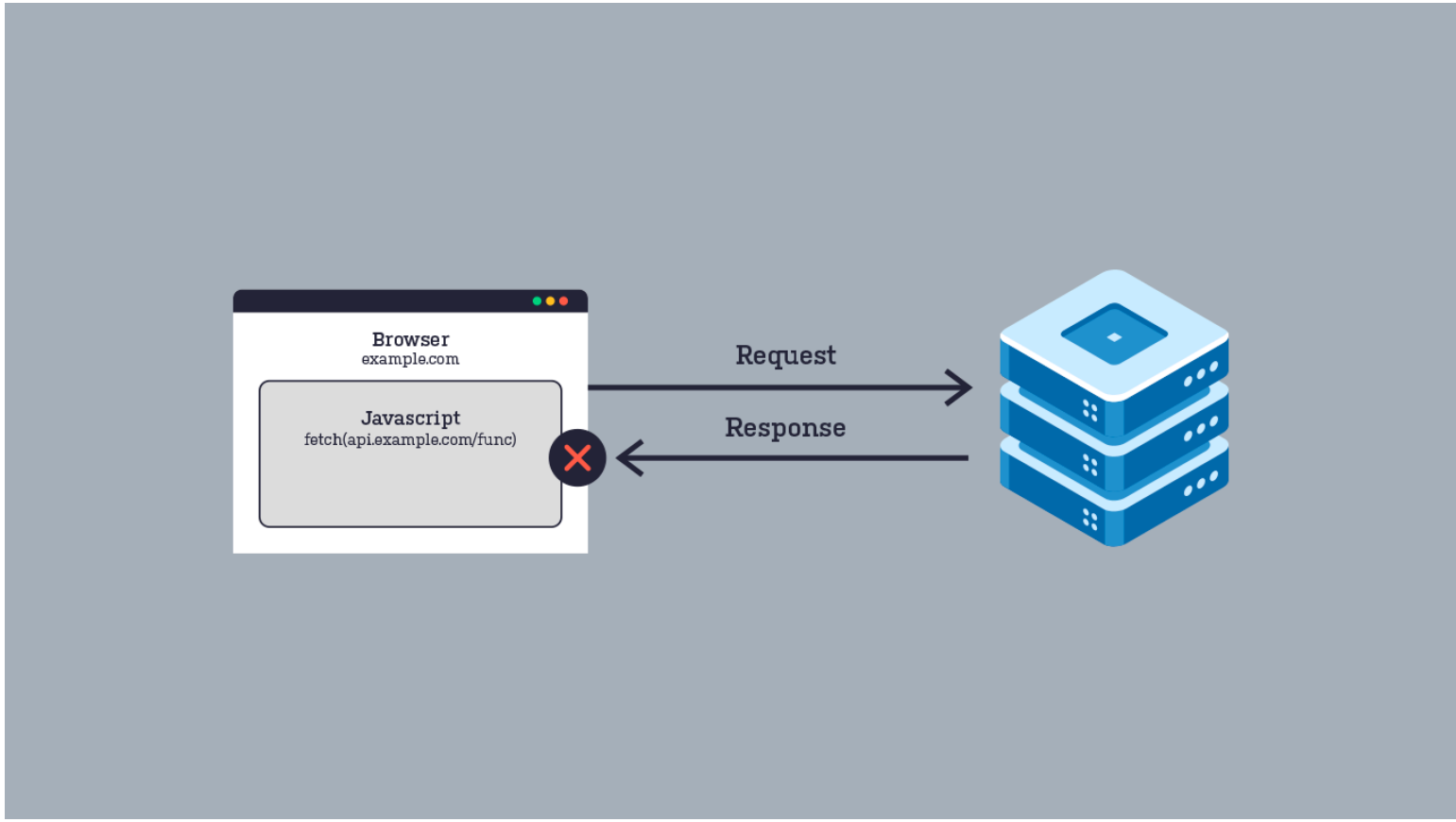
Disciplina de Backend - Professor Ramon Venson - SATC 2025

Same Origin Policy (SOP)

Por razões de segurança, o navegador web não permite que um script em uma página da web faça uma solicitação de rede para um recurso em um domínio diferente.

Isso pode acontecer mesmo em um ambiente de desenvolvimento local, onde o servidor e o cliente são executados de portas diferentes.

As políticas de SOP são aplicadas para domínio, protocolo e porta.



Exemplo de SOP

Imagine que você vai realizar uma requisição do navegador, usando a API Fetch, para um servidor que está hospedado em `app.dev`.

Para que essa requisição seja realizada, o navegador precisa ter certeza que você está usando a página `app.dev` e não uma página de um site malicioso.

Casos de SOP

A política de SOP é aplicada em três casos:

- **Requisições de navegação**: quando o usuário navega para uma URL diferente da URL atual.
- **Requisições de script**: quando um script em execução em uma página da web faz uma solicitação de rede para um recurso em um domínio diferente.

A política de SOP não será aplicada para esses casos:

- **Requisições de Medias**: quando uma página da web faz uma solicitação de rede para uma media (imagem, vídeo, *stylesheets*, *links*...) em um domínio diferente.
- **Requisições de formulário**: o SOP não restringe requisições, e sim o acesso à resposta. Logo, formulários não são afetados pela política de SOP.
- **Outros clientes HTTP**: Um cliente HTTP como Insominia, Postman, cURL, etc não possui a política de SOP.
- **Extensões de Navegador**: Extensões possuem geralmente permissões elevadas e não possuem requisições restritas.

Implicações do SOP

Sem essa restrição, um script malicioso poderia fazer uma solicitação de rede para qualquer site que o usuário visitou, por exemplo, para roubar informações confidenciais ou realizar ações maliciosas. Como nessa requisição:

```
fetch('https://api.app.dev/usuario', { method: 'GET'}); // Retorna dados do usuario
```

Essa requisição poderia ser feita de uma página de terceiros e funcionaria desde que o usuário estivesse devidamente autenticado.

Como essa limitação acontece com os navegadores, há uma forma de garantir que o servidor aceite requisições de outros domínios.

Isso é feito através do uso de um mecanismo chamado ***CORS (Cross-Origin Resource Sharing)***.

CORS

O CORS (Cross-Origin Resource Sharing) é um mecanismo implementado no **SERVIDOR**, que informa ao navegador que ele pode permitir solicitações de origens diferentes.

Isso é uma forma de permitir, por exemplo, que uma página da web em `www.app.dev` faça uma requisição para um servidor em `api.app.dev` .

Funcionamento

O CORS é implementado por meio da adição de cabeçalhos HTTP adicionais ao servidor:

- `Access-Control-Allow-Origin`, que especifica quais origens são permitidas para fazer solicitações ao servidor.
- `Access-Control-Allow-Methods`, que especifica quais métodos HTTP são permitidos para solicitações de uma origem específica.
- `Access-Control-Allow-Credentials`, que especifica se as credenciais (cookies, cabeçalhos de autenticação) podem ser incluídas nas solicitações CORS.

Esses cabeçalhos esclarecem ao navegador quais origens e métodos são permitidos para fazer solicitações ao servidor.


Allowed Origins
https://www.mywebsite.com
CORS
Browser
Server

Implementando CORS no Spring Boot

Para implementar o CORS no Spring Boot, podemos usar duas abordagens muito simples:

- Anotando o controlador com a anotação `@CrossOrigin`
- Configurando o CORS em uma classe de configuração da aplicação.

Configuração de Controlador

Para configurar o CORS em um `Controller`, você pode usar a anotação `@CrossOrigin`:

```
@CrossOrigin(origins = "http://app.dev:8080")
@RestController
public class MyController {
    ...
}
```

Esse exemplo estabelece que o controlador aceita solicitações de origem `http://app.dev:8080`.

Configuração de Aplicação

Se você deseja configurar o CORS para toda a aplicação, você pode definir uma classe de configuração `WebConfig`:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
            .allowedOrigins("http://app.dev:8080")
            .allowedMethods("GET", "POST", "PUT", "DELETE")
            .allowCredentials(true);
    }
}
```

Essa classe deve estar presente dentro da estrutura de pacotes da aplicação. Por exemplo, se a aplicação está em `com.example.app`, a classe de configuração pode ser inserida em `com.example.app.config`.

Riscos do CORS

É comum que o CORS seja configurado como uma configuração global em ambientes de desenvolvimento (`@CrossOrigin("*")`), permitindo solicitações de qualquer origem.

No entanto, isso pode ser um risco, pois permite que qualquer origem faça solicitações ao servidor, permitindo ataques como *cross-site scripting (XSS)* ou *cross-site request forgery (CSRF)*.

Material de Apoio

- CORS - MDN Web Docs
- CORS request not HTTP
- Usando Fetch
- Ataque XSS
- Ataque CSRF