Tarea 2

CENFOTEC

Johnny Rivera

Desarrollo de RESTful Web APIs con .NET Core

Trabajo de investigación

1. ¿En qué consiste CORS?
2. Incluir un ejemplo de solución en C#, JavaScript, Python, Go o algún otro lenguaje

Respuesta 1 investigado de la fuentes:

<https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS>

<https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/cross-origin-resource-sharing/>

El Intercambio de Recursos de Origen Cruzado ([CORS](https://developer.mozilla.org/en-US/docs/Glossary/CORS)) es un mecanismo que utiliza cabeceras [HTTP](https://developer.mozilla.org/es/docs/Glossary/HTTP) adicionales para permitir que un [user agent](https://developer.mozilla.org/en-US/docs/Glossary/user_agent) obtenga permiso para acceder a recursos seleccionados desde un servidor, en un origen distinto (dominio) al que pertenece. Un agente crea una petición HTTP de origen cruzado cuando solicita un recurso desde un dominio distinto, un protocolo o un puerto diferente al del documento que lo generó.

Un ejemplo de solicitud de origen cruzado: el código JavaScript frontend de una aplicación web que es localizada en http://domain-a.com utiliza [XMLHttpRequest](https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest) para cargar el recurso http://api.domain-b.com/data.json.

Por razones de seguridad, los exploradores restringen las solicitudes HTTP de origen cruzado iniciadas dentro de un script. Por ejemplo, [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) y la API [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) siguen la [política de mismo-origen](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy). Ésto significa que una aplicación que utilice esas APIs [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) sólo puede hacer solicitudes HTTP a su propio dominio, a menos que se utilicen cabeceras CORS.

El [W3C](http://www.w3.org/) [Grupo de Trabajo de Aplicaciones Web](http://www.w3.org/2008/webapps/) recomienda el nuevo mecanismo de [Intercambio de Recursos de Origen Cruzado](http://www.w3.org/TR/cors/) (CORS, por sus siglas en inglés). CORS da controles de acceso a dominios cruzados para servidores web y transferencia segura de datos en dominios cruzados entre navegadores y servidores Web. Los exploradores modernos utilizan CORS en un  contenedor API (como [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) o [Fetch](https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API)) para ayudar a mitigar los riesgos de solicitudes HTTP de origen cruzado.

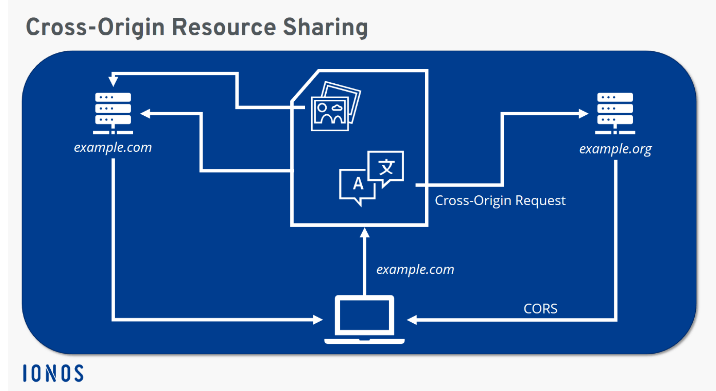
**CORS: qué es y cómo funciona el cross-origin resource sharing**

Cuando se abre una página web, cargar datos de servidores ajenos está, en teoría, estrictamente prohibido. Sin embargo, puede haber excepciones: si los administradores de ambas webs han acordado trabajar juntos, no hay por qué evitar el intercambio. En estos casos, el llamado cross-origin resource sharing (CORS) regula la colaboración

La same-origin policy (SOP o política de seguridad del mismo origen) prohíbe que se carguen datos de servidores ajenos al acceder a una página web. Todos los datos deben provenir de la misma fuente, es decir, corresponder al mismo servidor. Se trata de una medida de seguridad, ya que JavaScript y CSS podrían cargar, sin que el usuario lo supiese, contenido de otros servidores (y, con este, también contenido malicioso). Tales intentos son denominados “cross-origin requests”. Si, por el contrario, ambos administradores web saben del intercambio de contenido y lo aprueban, no tiene sentido impedir este proceso. El servidor solicitado (es decir, aquel del que se quiere cargar contenido) puede permitir entonces el acceso mediante cross-origin resource sharing, en castellano, intercambio de recursos de origen cruzado.

Este permiso se da, no obstante, únicamente a clientes concretos, es decir, el CORS no es un comodín para realizar cualquier cross-origin request. En lugar de eso, el segundo servidor permite al primero un acceso exclusivo mediante una [cabecera HTTP](https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/cabecera-http/). En dicha cabecera de la respuesta HTTP está indicado específicamente qué servidores pueden cargar datos y ponerlos a disposición del usuario. El acceso generalizado a todos los clientes se permite únicamente mediante una “wildcard” o certificado comodín. Esta solución, sin embargo, solo es conveniente para servidores cuyo contenido debe estar a disposición del público general, como es el caso, por ejemplo, de las tipografías web.

Si todo sale bien, el usuario no se percatará en absoluto del intercambio entre ambos servidores. Todos los navegadores actuales soportan el CORS, y el envío de solicitudes y respuestas sucede rápidamente al solicitar una página web sin que el usuario lo note.



## Configuración a nivel de método y a nivel de clase

Spring nos permite hacer la configuración a nivel de método y también a nivel global o de clase.

Para hacerla a nivel de método, que podría ser de un método del controlador o del controlador completo, Spring provee la anotación @CrossOrigin, que nos permite indicar que un determinado método, además de estar protegido por CORS, va a permitir el acceso de determinados orígenes, y si no permitimos el acceso a esa petición, nos mostrará un error.

@CrossOrigin(origins = http://localhost:9001

@GetMapping(“/producto”)

public ResponseEntity<?> obtenerTodos() {

//…

}

También se puede hacer la configuración a nivel global, lo podríamos hacer de diferentes maneras. Incluso podríamos bajar el nivel de abstracción, acercarnos más a Java y crear un filtro dónde hacer la configuración, aunque con un WebMvcConfigurer podríamos también hacer la configuración de CORS.

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurer() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping(“/producto/\*\*”)

.allowedOrigins(“http://localhost:9001”)

.allowedMethods(“GET”, “POST”, “PUT”, “DELETE”)

.maxAge(3600);

}

};

}