Usar Spring REST CORS es muy habitual hoy en día ya que la mayoría de peticiones a servicios REST que se realizan es utilizando algún tipo de tecnología Javascript y por lo tanto utilizando AJAX. Crear un servicio REST con Spring Framework es muy sencillo hoy en día ya que es suficiente con crear una clase anotada con @RestController . Vamos a verlo utilizando Spring Boot , para ello necesitaremos instalar las dependencias de Web.

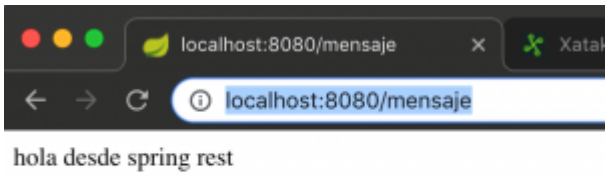
```
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-
test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>
```

Una vez instaladas estas dependencias el siguiente paso es construirnos un Controlador de tipo REST que Spring nos provee por defecto con la típica url de /mensaje.

```
package com.arquitecturajava.rest;

import org.springframework.web.bind.annotation.GetMapping;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HolaRESTController {

        @GetMapping("/mensaje")
        public String mensaje() {
                return "hola desde spring rest";
        }
}
```

Creado el controlador nos será suficiente con arrancar la aplicación de Spring Boot y solicitar la url.

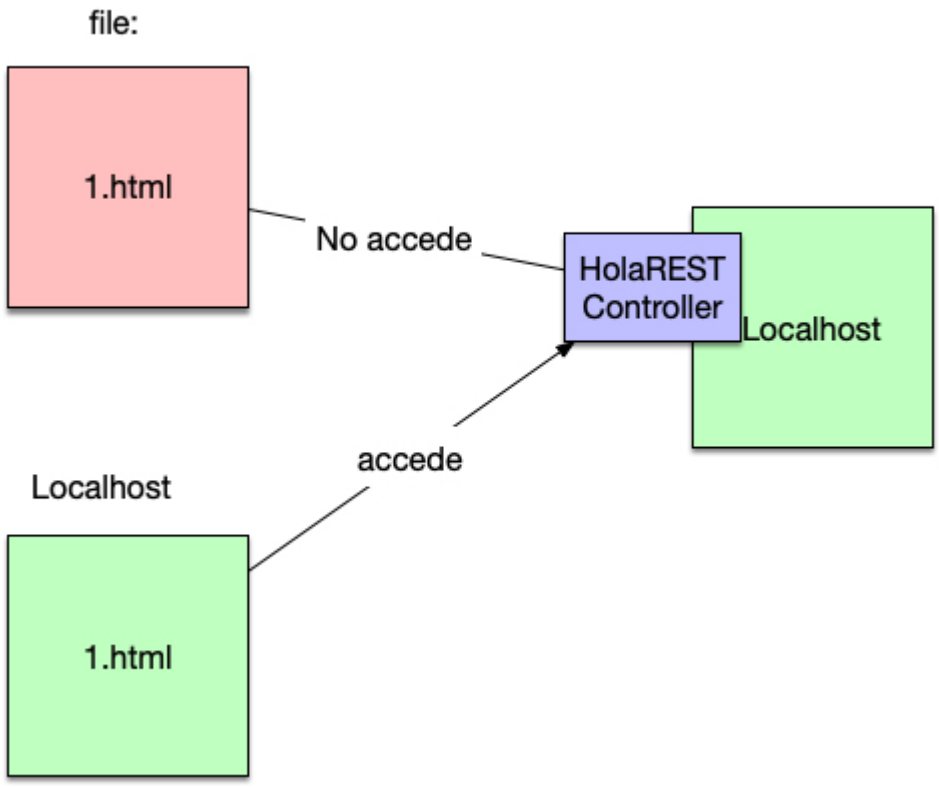


El servicio REST nos responde sin ningún problema . Ahora bien esto se debe a que realizamos una invocación directa y no vía JavaScript . Si intentamos solicitar esta url con JavaScript utilizando jQuery nos encontraremos con un problema de Cross Origin Resource Sharing (CORS) que nos impide el acceso.

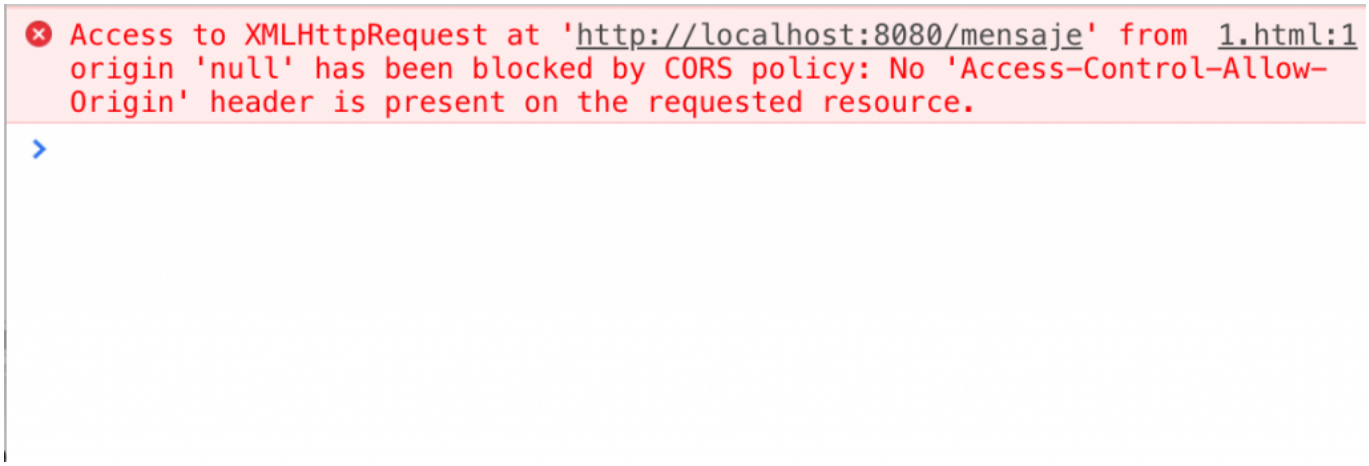
```
</pre>
<html>
```

```
<head>
<script type="text/javascript" src="jquery-3.3.1.min.js">
</script>
<script type="text/javascript">
$(document).ready(function() {

$.get("http://localhost:8080/mensaje",function(datos) {

console.log(datos);
})

})
</script>
</head>
<body>

</body>
</html>
<pre>
```

Si cargamos esta página veremos rapidamente un error en la consola que nos restringe el acceso debido a que estamos realizando una petición ajax desde JavaScript y estas peticiones por defecto están limitadas a ficheros JavaScript que nos descarguemos desde el mismo servidor.

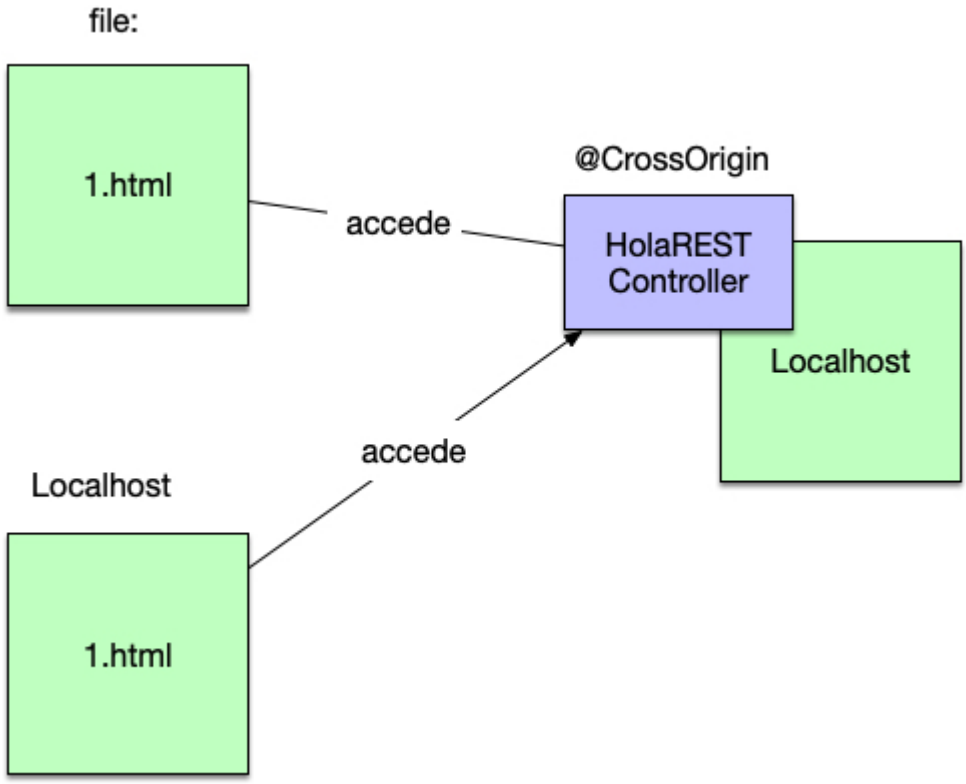


Si nosotros este fichero le cargamos desde un navegador directamente con file:// nos aparecerá el siguiente error:



Spring REST CORS

El recurso esta por defecto bloqueado para peticiones que no hagan desde localhost , caso de nuestra petición que se realiza desde un fichero directamente . Para solventar este problema es suficiente con modificar el servicio de Spring y añadir una cabecera @CrossOrigin para que nos permite el acceso desde otras ubicaciones.

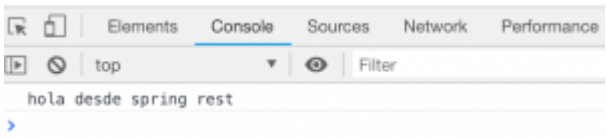


En este caso vamos a ser generalistas y permitir acceder al recurso desde cualquier lugar.

```
package com.arquitecturajava.rest;
```

```
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
@CrossOrigin(origins = "*", methods=
{RequestMethod.GET,RequestMethod.POST})
public class HolaRESTController {

        @GetMapping("/mensaje")
        public String mensaje() {
                return "hola desde spring rest";
        }
}
```

Volvemos a cargar el servidor y ahora si podremos acceder al mensaje almacenado en la url:



Recordemos que siempre que tengamos recursos REST y queramos acceder a ellos debemos usar Spring REST CORS y abrir el acceso remoto sino por defecto los datos de nuestros servicios no estarán accesibles.

Otros artículos relacionados :

1. ¿Que es CORS ?
2. Spring REST Client con RestTemplates
3. Swagger documentando nuestro API REST
4. Spring REST

Spring REST CORS y su configuración