Módulo VI: CORS

AUTOR: CHANQUIA, Luis Antonio

Índice

Introducción

La seguridad de los navegadores evita que una página web realice solicitudes AJAX a otros dominios diferentes al propio. Esta restricción se denomina política del mismo origen y evita que un sitio malintencionado lea datos confidenciales de otro sitio. Sin embargo, a veces es posible que deseemos permitir que otros sitios llamen a nuestra API web.

Cross Origin Resource Sharing (CORS) es un estándar W3C que permite a un servidor liberar la política del mismo origen. Con CORS, un servidor puede permitir explícitamente algunas solicitudes de origen cruzado mientras rechaza otras. (https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api)

Objetivos

1. Incorporar los conceptos básicos sobre CORS.

CORS

¿Qué es CORS?

CORS (Cross Origin Resource Sharing, o bien en español Intercambio de Recursos de Origen Cruzado) es un mecanismo que permite solicitar recursos restringidos desde una página web de un dominio determinado a otro recurso web de otro dominio. De esta manera, CORS define una manera en la que el navegador y el servidor puedan interactuar para determinar si la petición de origen cruzado es segura.

¿Cómo funciona CORS?

CORS trabaja agregando cabeceras HTTP que permiten a los servidores describir un conjunto de orígenes (dominios) que tienen permisos para obtener una información usando el navegador. Adicionalmente la especificación de estos, sugiere que los navegadores verifiquen la solicitud solicitando métodos soportados desde el servidor con un método de solicitud HTTP OPTIONS, luego con la aprobación del servidor se envía la verdadera solicitud con el método HTTP.

¿Qué es "mismo origen"?

Dos URL tienen el mismo origen si tienen esquemas, hosts y puertos idénticos.

Por ejemplo, estas dos URL tienen el mismo origen:

http://example.com/foo.html

http://example.com/bar.html

Y estas URL tienen orígenes diferentes a los dos anteriores:

http://example.net - Dominio diferente

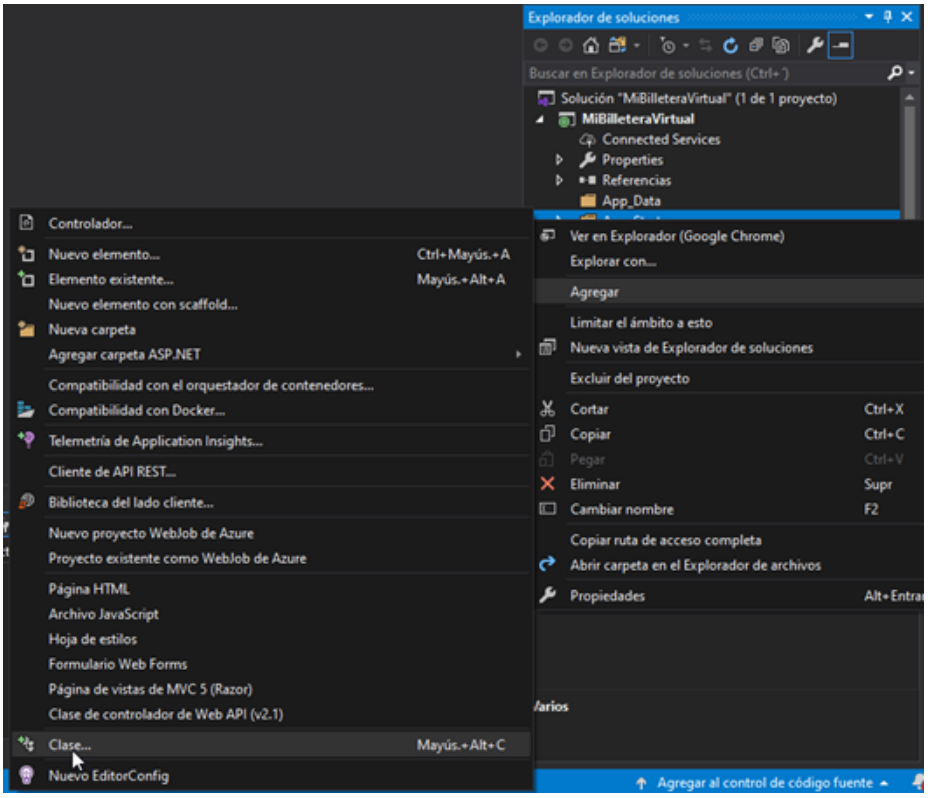
http://example.com:9000/foo.html - Puerto diferente

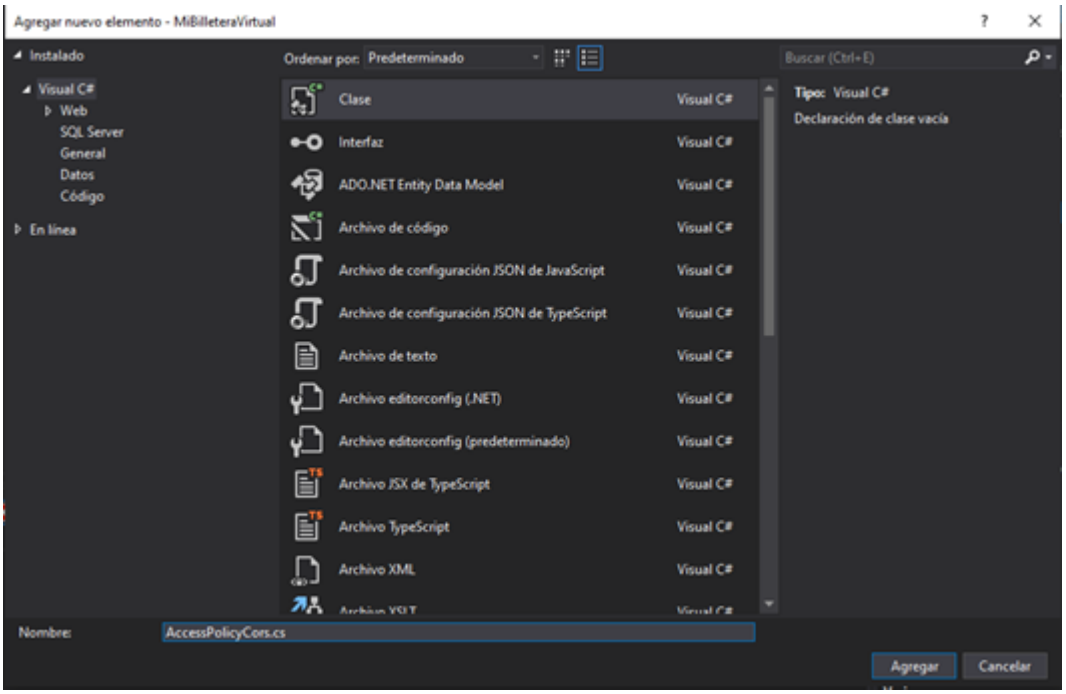
https://example.com/foo.html - Esquema diferente

http://www.example.com/foo.html - Subdominio diferente

Configuración del proyecto MiBilleteraVirtual para permitir CORS

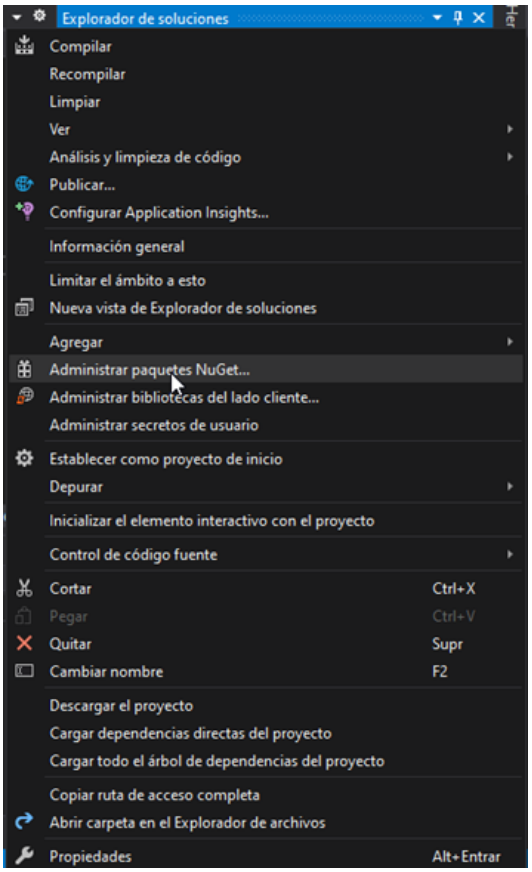
Primero, creamos una clase llamada AccessPolicyCors en la carpeta App_Start de nuestro proyecto, para esto, pulsamos clic derecho sobre la carpeta App_Start, vamos a Add y elegimos la última opción que es Class…


Explorador de soluciones
Buscar en Explorador de soluciones (Ctrl+ ')
Solución "MiBilleteraVirtual" (1 de 1 proyecto)
MiBilleteraVirtual
Connected Services
Properties
Referencias
App_Data
Controlador...
Nuevo elemento... Ctrl+Mayús.+A
Elemento existente... Mayús.+Alt+A
Nuevo elemento con scaffold...
Nueva carpeta
Agregar carpeta ASP.NET
Compatibilidad con el orquestador de contenedores...
Compatibilidad con Docker...
Telemetría de Application Insights...
Cliente de API REST...
Biblioteca del lado cliente...
Nuevo proyecto WebJob de Azure
Proyecto existente como WebJob de Azure
Página HTML
Archivo JavaScript
Hoja de estilos
Formulario Web Forms
Página de vistas de MVC 5 (Razor)
Clase de controlador de Web API (v2.1)
Clase... Mayús.+Alt+C
Nuevo EditorConfig
Ver en Explorador (Google Chrome)
Explorar con...
Agregar
Limitar el ámbito a esto
Nueva vista de Explorador de soluciones
Excluir del proyecto
Cortar Ctrl+X
Copiar Ctrl+C
Pegar Ctrl+V
Eliminar Supr
Cambiar nombre F2
Copiar ruta de acceso completa
Abrir carpeta en el Explorador de archivos
Propiedades
Agregar al control de código fuente

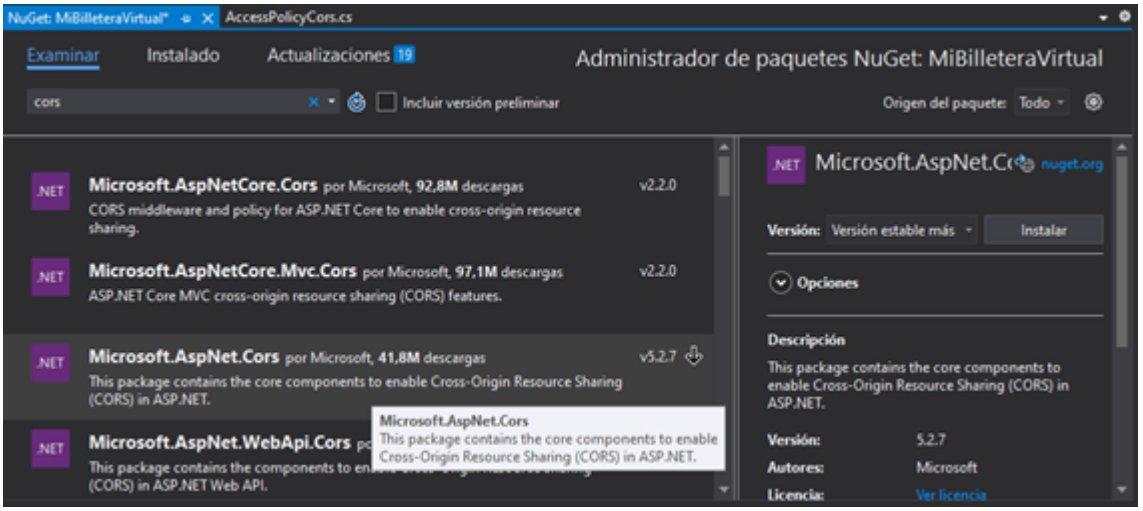

Agregar nuevo elemento - MiBilleteraVirtual
Instalado
Visual C#
Web
SQL Server
General
Datos
Código
En línea
Ordenar por: Predeterminado
Buscar (Ctrl+E)
Clase Visual C#
Interfaz Visual C#
ADO.NET Entity Data Model Visual C#
Archivo de código Visual C#
Archivo de configuración JSON de JavaScript Visual C#
Archivo de configuración JSON de TypeScript Visual C#
Archivo de texto Visual C#
Archivo editorconfig (.NET) Visual C#
Archivo editorconfig (predeterminado) Visual C#
Archivo JSX de TypeScript Visual C#
Archivo TypeScript Visual C#
Archivo XML Visual C#
Tipo: Visual C#
Declaración de clase vacía
Nombre: AccessPolicyCors.cs
Agregar
Cancelar

Instalar paquetes CORS

Una vez creada la clase instalamos los paquetes CORS. Para instalarlos hacemos clic derecho sobre nuestro proyecto y seleccionamos la opción Administrar Paquetes NuGet…



Vamos al campo Buscar y luego instalamos los siguientes paquetes.



Una vez descargados los paquetes, vamos a nuestra clase y agregamos los siguientes using.

```
using System.Web.Http.Cors;
using System.Threading.Tasks;
using System.Web.Cors;
using System.Net.Http;
using System.Threading;
```

Dentro de la clase que acabamos de crear vamos a hacer que herede de otras 2 clases: Attribute e ICorsPolicyProvider.

```
14      public class AccessPolicyCors : Attribute, ICorsPolicyProvider
```

Luego de haber heredado, crearemos un método privado y asíncrono llamado IsOriginFromCustomer. Tendrá el siguiente código.

```
43      private async Task<bool> IsOriginFromCustomer(string originRequested)
44      {
45          return true;
46      }
```

Este método nos será útil para más adelante, su función es validar si hemos recibido una petición.

El siguiente método que crearemos se llamará GetCorsPolicyAsync, será un método público, asíncrono y tendrá como tarea el CorsPolicy. Este método es propio de la interfaz ICorsPolicyProvider.

```
16      public async Task<CorsPolicy>
        0 referencias
17      GetCorsPolicyAsync(HttpRequestMessage request, CancellationToken cancellationToken)
```

Lo siguiente que haremos es traer el contexto de la solicitud y el origen de la misma.

```
19          var corsRequestContext = request.GetCorsRequestContext();
20          var originRequested = corsRequestContext.Origin;
```

Ahora usaremos nuestro método privado que creamos anteriormente, lo usaremos dentro de una condición “if”.

```
22          if (await IsOriginFromCustomer(originRequested))
23          {
```

Dentro del “if” crearemos un nuevo objeto CorsPolicy donde le diremos que acepte cualquier cabecera y cualquier método.

```
24              var policy = new CorsPolicy
25              {
26                  AllowAnyHeader = true,
27                  AllowAnyMethod = true
28              };
```

Y a continuación añadimos el origen de nuestra solicitud al origen de nuestro objeto “policy”.

```
32              policy.Origins.Add(originRequested);
```

Por último devolvemos el objeto “policy” y cerramos nuestra condición “if”.

```
39              return policy;
40          }
```

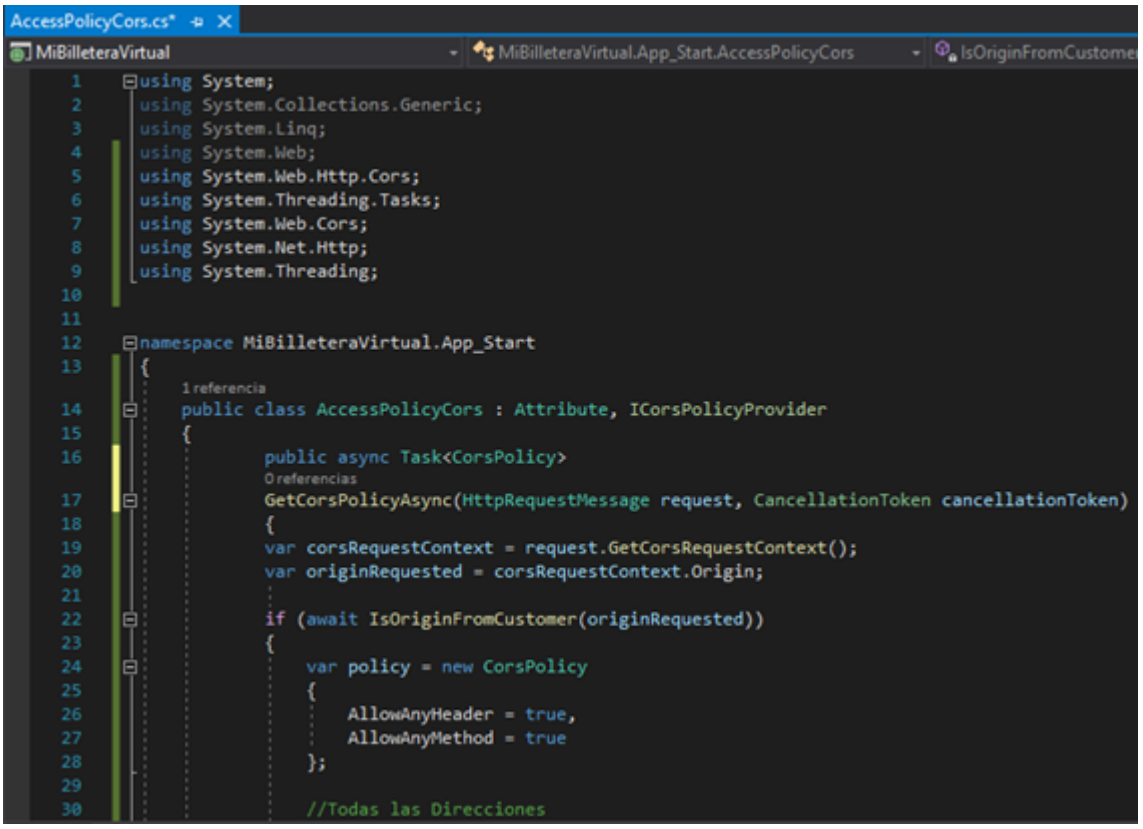
Si no cumple la condición, retornamos null.

```
41          return null;
```

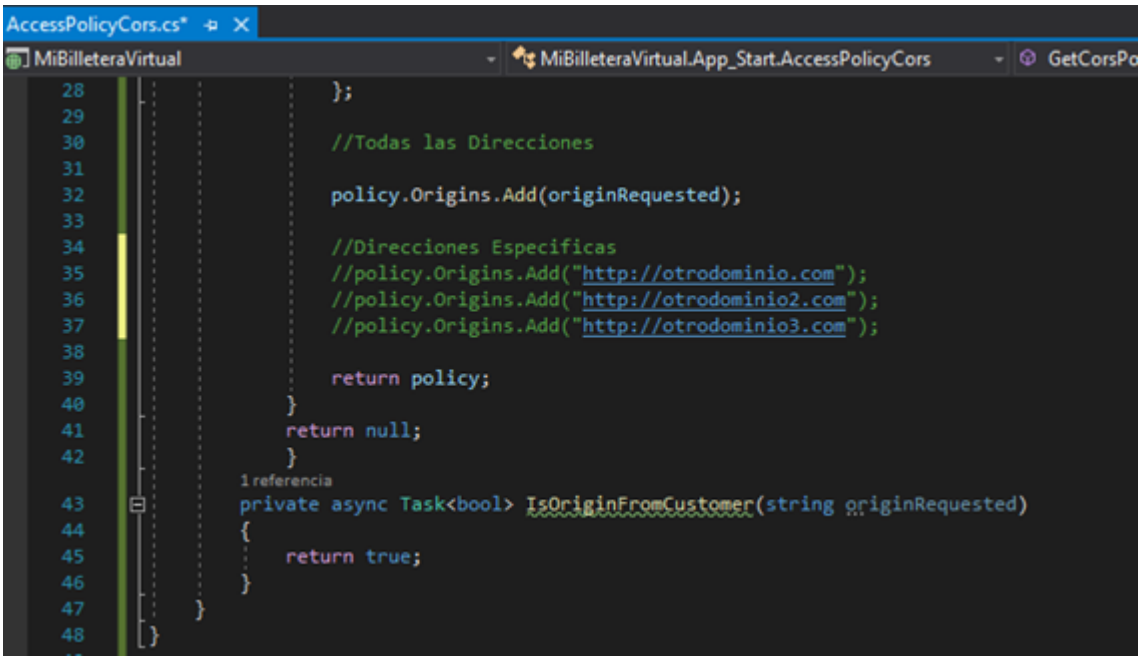
Si queremos permitir direcciones específicas agregamos las siguientes líneas según la dirección que necesitemos.

```
//Direcciones Especificas
policy.Origins.Add("http://otrodominio.com");
policy.Origins.Add("http://otrodominio2.com");
policy.Origins.Add("http://otrodominio3.com");
```

Versión final de la clase AccesPolicyCors



```
AccessPolicyCors.cs*
MiBilleteraVirtual
MiBilleteraVirtual.App_Start.AccessPolicyCors
IsOriginFromCustome
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Http.Cors;
6  using System.Threading.Tasks;
7  using System.Web.Cors;
8  using System.Net.Http;
9  using System.Threading;
10
11
12 namespace MiBilleteraVirtual.App_Start
13 {
       1 referencia
14     public class AccessPolicyCors : Attribute, ICorsPolicyProvider
15     {
16             public async Task<CorsPolicy>
               0 referencias
17             GetCorsPolicyAsync(HttpRequestMessage request, CancellationToken cancellationToken)
18             {
19             var corsRequestContext = request.GetCorsRequestContext();
20             var originRequested = corsRequestContext.Origin;
21
22             if (await IsOriginFromCustomer(originRequested))
23             {
24                 var policy = new CorsPolicy
25                 {
26                     AllowAnyHeader = true,
27                     AllowAnyMethod = true
28                 };
29
30                 //Todas las Direcciones
```

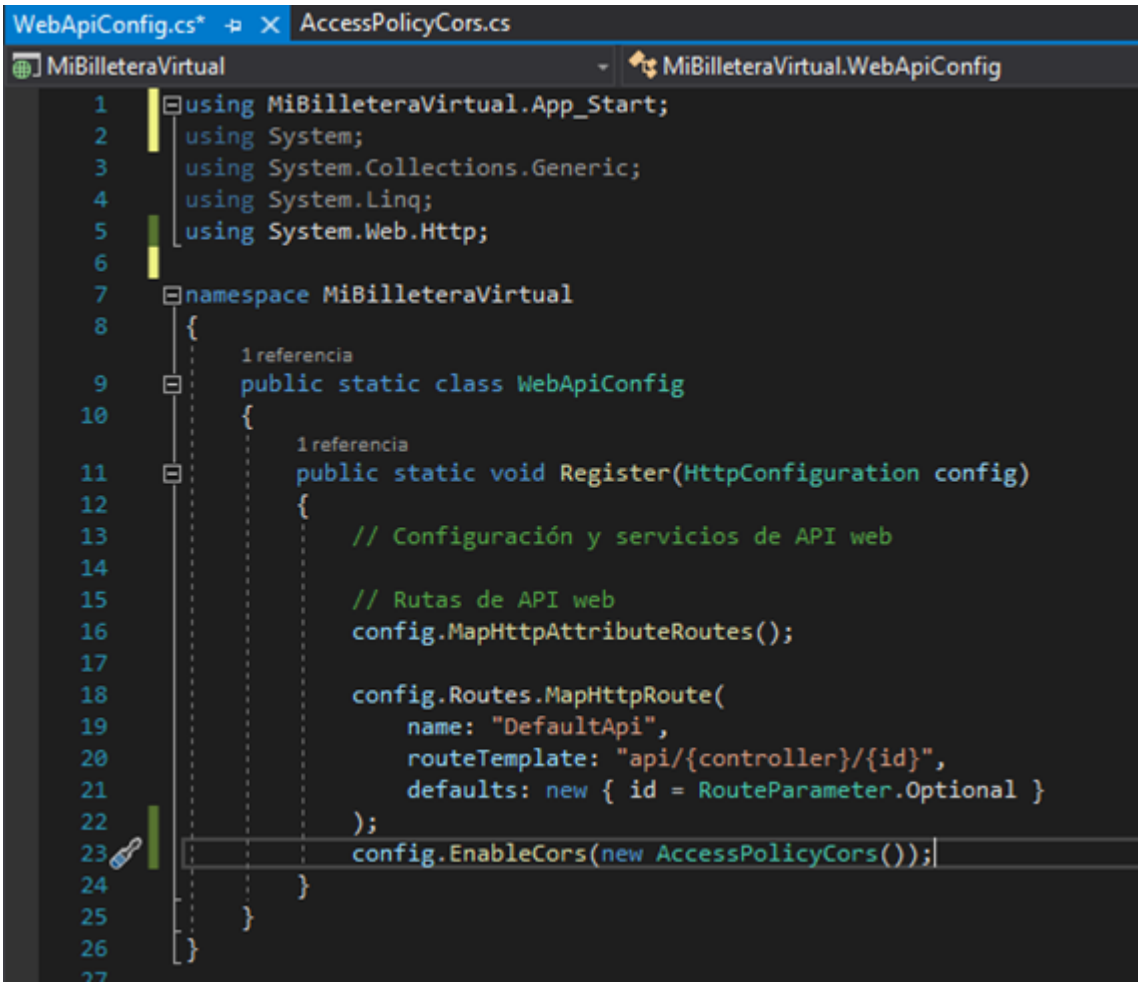


```
AccessPolicyCors.cs*
MiBilleteraVirtual
MiBilleteraVirtual.App_Start.AccessPolicyCors
GetCorsPo
28                 };
29
30                 //Todas las Direcciones
31
32                 policy.Origins.Add(originRequested);
33
34                 //Direcciones Especificas
35                 //policy.Origins.Add("http://otrodominio.com");
36                 //policy.Origins.Add("http://otrodominio2.com");
37                 //policy.Origins.Add("http://otrodominio3.com");
38
39                 return policy;
40             }
41             return null;
42             }
           1 referencia
43         private async Task<bool> IsOriginFromCustomer(string originRequested)
44         {
45             return true;
46         }
47     }
48 }
```

Nos dirigimos al archivo WebApiConfig que está en la carpeta App_Start que es la misma carpeta de nuestra clase y agregamos la siguiente línea.

```
config.EnableCors(new AccessPolicyCors());
```

El archivo quedará similar a la siguiente imagen.



Lo anterior, habilita el CORS a través de la clase que hemos creado. Con esto ya tenemos configurada nuestra API para recibir peticiones de otros orígenes.

Referencias

https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api

https://csharp.hotexamples.com/es/examples/System.Web.Cors/CorsPolicy/-/php-corspolicy-class-examples.html

https://techclub.tajamar.es/habilitar-cors-en-web-api/