COMPUTING SUBJECT:	Restful ASP.Net Core-services **For .NetCore 3.1**
TYPE:	Assignment
IDENTIFICATION:	RestService#6
COPYRIGHT:	*Peter Levinsky & Michael Claudius*
LEVEL:	Medium
TIME CONSUMPTION:	1½-2½ hours
EXTENT:	50-60 lines
OBJECTIVE:	Adding supporting CORS to the Restful services **version .Net Core 3.1**
PRECONDITIONS:	Rest service theory. Http-concepts Computer Networks Ch. 2.2
COMMANDS:	

IDENTIFICATION: RestService#6 / PELE with kindly respect and inspiration from MICL

Overall Purpose

The overall purpose for the group of 'RestService' assignments is to be able to provide and consume restful ASP.Net Core web services, to prepare the 'RestService' to be published in Azure, including testing the service and finally to setup the 'RestService' to be consumed from a browser (e.g. using Typescript) i.e. support CORS.

The whole group of assignments consist of 7 steps:

1. A simple REST Service with CRUD.
2. More advanced and complex URI's.
3. Adding help-pages to the REST Service (Swagger)
4. Testing a REST Service and publish in Azure.
5. Consuming a REST service from a C# Console application.
6. **Adding Support for CORS to the REST Service (this assignment)**
7. A REST Service using a database

Background Material:

The HTTP protocol: See Computer Network chap 2 pp. 111-136

Note of REST (Peter Levinsky): See NetHttpNote.pdf

Oswago Universitet: RESTful Service Best Practices: Recommendations for Creating Web Services: See http://cs.oswego.edu/~alex/teaching/csc435/RESTful.pdf

Usefull tools (Postman & Fiddler): See Tools.htm (tool #3 & tool #4)

Additional Literature

Note: https://www.moesif.com/blog/technical/cors/Authoritative-Guide-to-CORS-Cross-Origin-Resource-Sharing-for-REST-APIs/#

For setting up CORS in .NetCore in former version 2.1

see REST Service with CORS version 2.1

This Assignment: RestService#6

Purpose
The purpose of this assignment is to refactor your REST Service so it can manage call from script-pages in a browser in other words to support CORS.

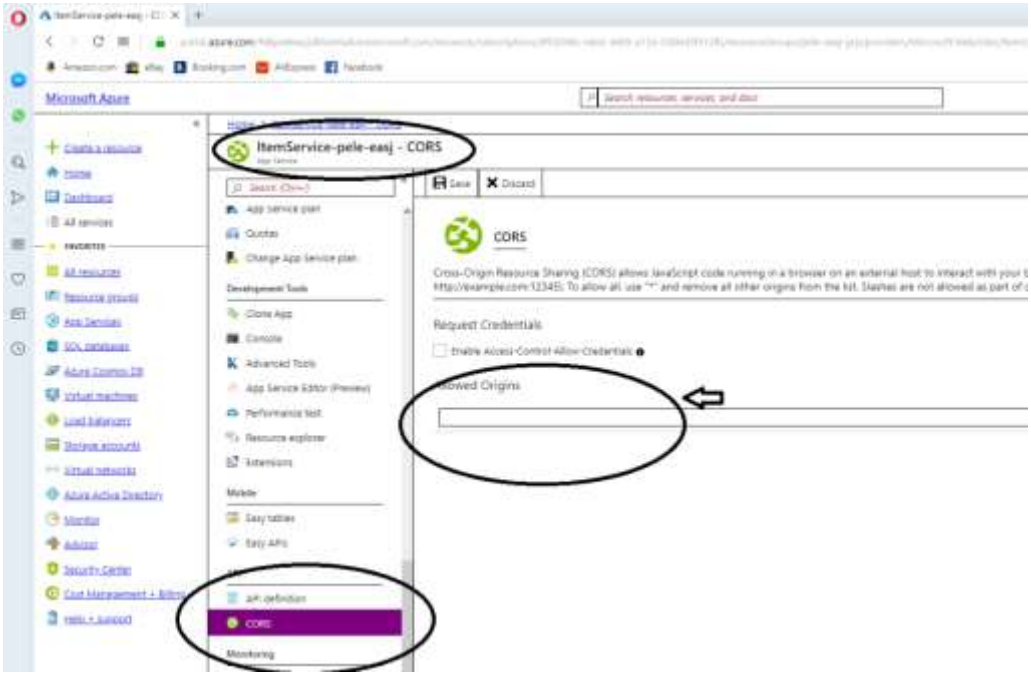
Mission
You are to design and implement CORS (Cross Origin Resource Sharing). There are three different way to design and implement CORS, they varying in the granularity of access control.

1. GlobalWare, Quick, but not so configurable and UNSECURE in Azure (do NOT work when using localhost!)
2. MVC, Specific setup CORS for each URI.

Assignment 1: Quick solution in Azure

a. Open the Azure portal https://portal.azure.com/

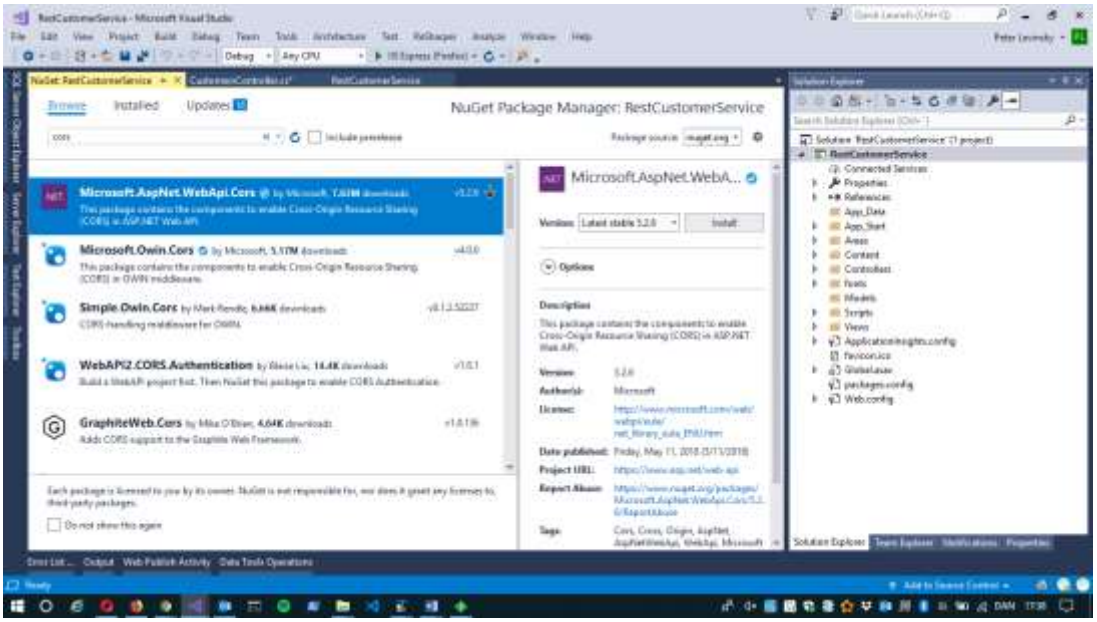
b. Open your APP Service that hold your REST Service, it will similar to this:



For Allowed origins insert '*****', meaning everything from anywhere.
Remember to save.
Now it's working in your Azure REST Service.

Assignment 2: MVC, Specific setup CORS for each URI

a. First you need a NuGet-package installed; at your project open the NuGet manager and choose '**Microsoft.AspNetCore.Cors**' version 2.2.0 to be installed:



and yes … It do take some time ☹

b. In the solution (Solution Explore); Open the file '**Startup.cs**'.
In the '**ConfigureServices**' method, add the line

```
services.AddCors(options =>
{
   options.AddPolicy("AllowSpecificOrigin",
      builder => builder.WithOrigins("http://zealand.dk")).
                         AllowAnyMethod().
                         AllowAnyHeader();

   options.AddPolicy("AllowAnyOrigin",
       builder => builder.AllowAnyOrigin()).
                        AllowAnyMethod().
                        AllowAnyHeader();

   options.AddPolicy("AllowAnyOriginGetPut",
        builder => builder.AllowAnyOrigin().
                         WithMethods("GET", "PUT")).
                         AllowAnyHeader();
     });
```

c. Still in the Startup.cs – class in the '**Configure**'-method:
Add the lines after '**app.UseRouting()**'

app.UseCors("AllowAnyOriginGetPut"); // one of the other policy names

And before '**app.UseAuthorization()**'

d. Now extend this for those services i.e. methods you will have to support CORS.

e. Publish your Rest Service in Azure and try with some of your Typescript applications. ***(if you have solved assignment 1, then go back to Azure and remove the '*')***

f. Check that your REST service is correctly configured for CORS using Postman or Fiddler. Compose a **simple request** (i.e. GET) with a header-field :

```
Origin: { your location e.g. http://easj.dk }
```

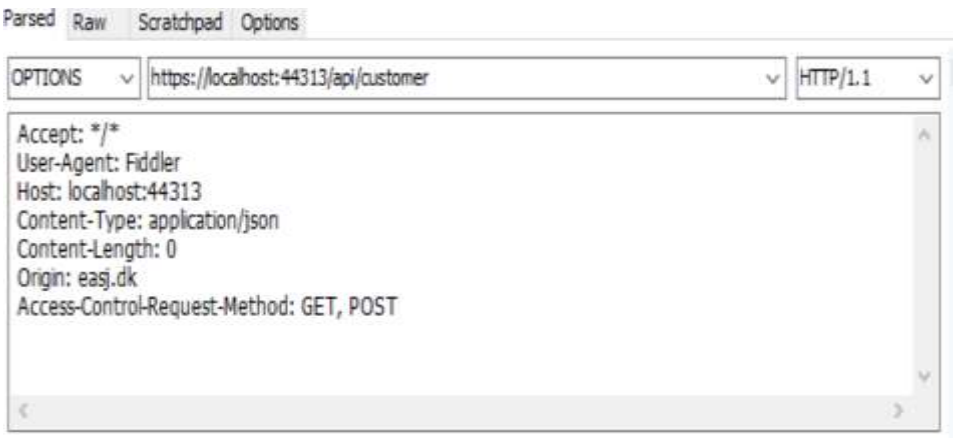
It should return a header field:

```
Access-Control-Allow-Origin: {your location e.g. easj.dk}
```

Or check for more complex request (i.e. PUT, POST, and DELETE) by a **preflighted request** using an '**OPTION'** request.

```
Origin: {your location e.g. http://easj.dk}
Access-Control-Request-Method: PUT, GET
Access-Control-Request-Headers: Authorization, Content-
Type
```

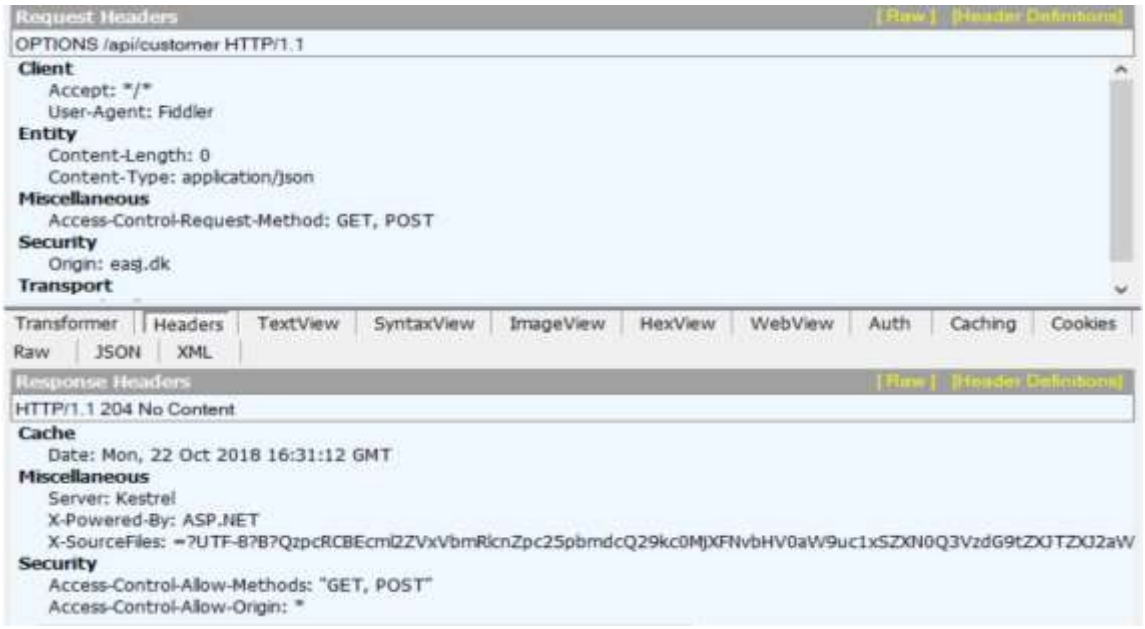
E.g. (Fiddler):



The server (with your CORS REST service) should return:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: {your location e.g. easj.dk}
Access-Control-Allow-Methods: POST, PUT, DELETE, GET, …
Access-Control-Allow-Headers: Authorization, Content-Type
```

E.g. (Fiddler):



What happen if you request access to POST or DELETE ??

f. For even more detailed CORS setup: In the controller, **ItemsController**, specify the policy you want on the controller itself, like:

```
[Route("api/[controller]")]
 [ApiController]
```

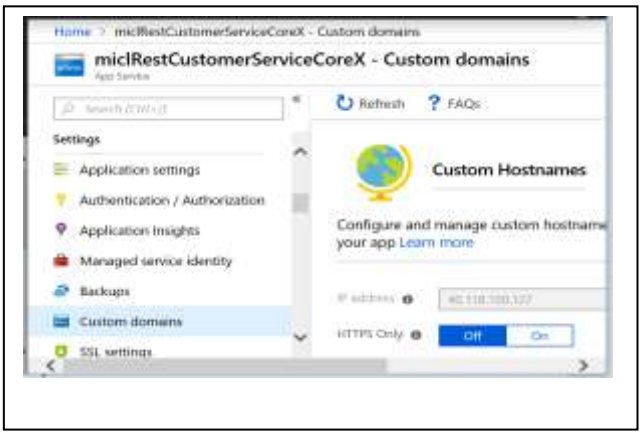
Still the controller, specify the policy for the methods, suppressing the controller-policy.

```
[HttpDelete("{id}")]
// no policy i.e. inherits the controller policy

[HttpPost]
[EnableCors("AllowSpecificOrigin")]

[HttpGet]
[DisableCors]  //disable the controller policy
```

g. After published in Azure, check your new configuration using Fiddler or Postman like in previous assignment.

h. **If you miss to uncheck the https when creating the REST service in assignment 1 do this and the following bullet.** *Unfortunately you probably get a 301/502 error security error.*
Why?
The issue is that if your project was created it was configured for Https and Fiddler uses Http-scheme for Azure. Read on…

i. Go to your Azure Portal
    1. Open your Web-App project
    2. Find *Custom domains* in the left scroll-bar
    3. Set *Https-Only* to OFF
    4. Click *Refresh*



Congratulations your REST service can now be used from e.g. a typescript application, the last step is to provide persistence in your REST service through a Database instead of a static-list.