COMPUTING SUBJECT:	Restful ASP.Net Core-services with CORS
TYPE:	Assignment
IDENTIFICATION:	RestCustomerService No. 2
COPYRIGHT:	*Michael Claudius & Peter Levinsky*
LEVEL:	Medium
TIME CONSUMPTION:	2 hours
EXTENT:	40 lines
OBJECTIVE:	Restful services based on ASP.Net Core
PRECONDITIONS:	Exercise RestCustomerService No. 1 is a must Rest service theory. Http-concepts Computer Networks Ch. 2.2
COMMANDS:	

IDENTIFICATION: RestCustomerService No. 2 /MICL&PELE

Purpose

The purpose of this assignment is to set up Unit test and to provide Cross Origin Resource Sharing (CORS) of a restful ASP.Net Core Web service. More specifically, to enable Cross Origin Requests in ASP.Net Core Web API 3.1, alternatively 2.1

Precondition

You must have done the RestCustomerService, as basic information and guidelines are given in this exercise.

Mission

You are to make and use restful web services based on the ASP.Net Core services by setting up a server (provider), test the services by use of Fiddler/Postman and create a client (consumer) using the services provided. On the way you will publish the service to the cloud (Azure). The service supports the classic GET, POST, PUT and DELETE requests. This we shall do in the following steps:

1. Create a project with auto generated service: api/values
2. Create a model class Customer for customer data
3. Create a controller CustomerController to provide REST services
4. Extend CustomerController with a list of customers
5. Create and provide a controller oriented service in CustomerController
6. Test the service using Browser/Fiddler/Postman
7. Create a client/consumer utilizing the service
8. More services and testing by Fiddler/Postman and client/consumer
9. Publish to Azure
10. Support simple Cross Origin Resource Sharing (CORS) using Azure
11. Support dedicated Cross Origin Resource Sharing (CORS) in the project
12. Set up a project for Unit test
13. Refactor the consumer code

This assignment holds steps 11 – 13, whereas steps 1-10 were done in the previous exercise RestCustomerService No. 1.

Domain description

Management and administration of customers utilizing web services for the classic operations:

Create (POST)
Read, i.e. Find one or more. (GET)
Update (PUT)
Delete (DELETE)

Reflecting standard Http requests.

When surfing on the net it is easy to find many descriptions more or less useful, and in more or less updated versions. Here are some:

Useful links for C#:

CORS
https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api (from the middle enable CORS)

https://docs.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-2.1
Probably the best description holds comparison of Middleware CORS and CORS in MVC

https://stackoverflow.com/questions/44379560/how-to-enable-cors-in-asp-net-core-webapi

https://www.nuget.org/packages/Microsoft.AspNetCore.Cors/

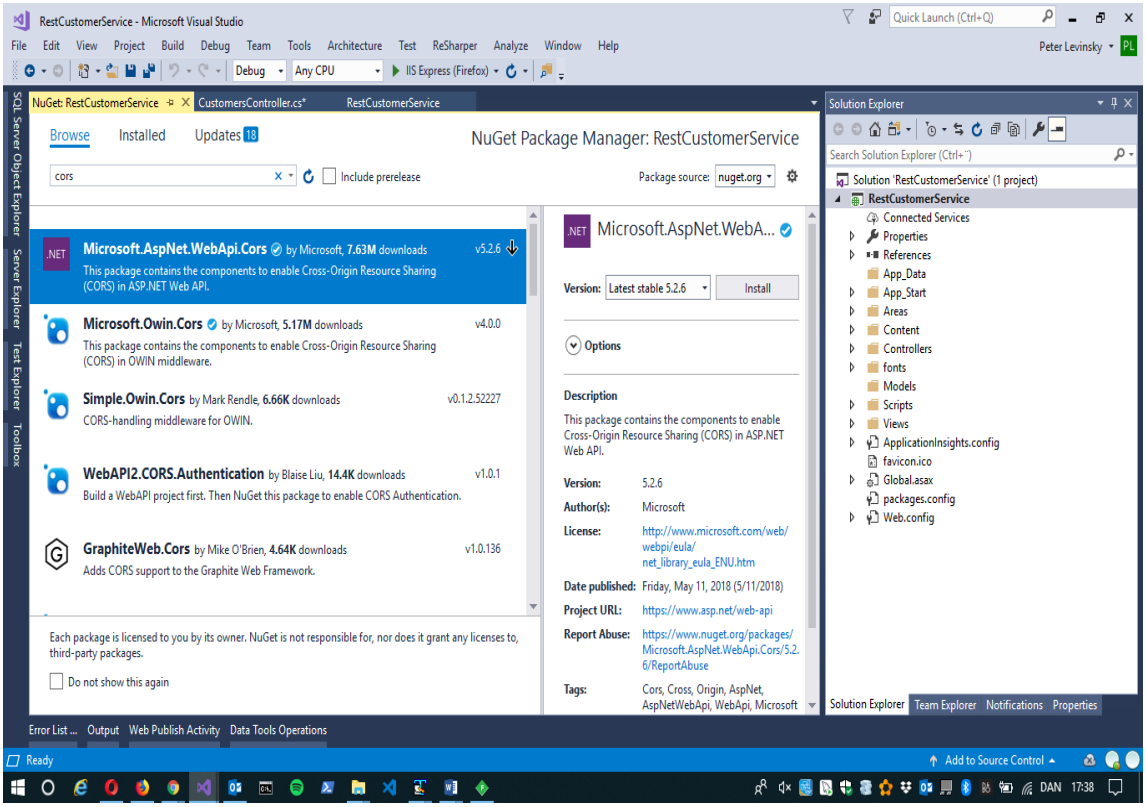
Test
https://code.msdn.microsoft.com/Unit-Testing-with-ASPNET-1374bc11/sourcecode?fileId=179451&pathId=1993051352

https://docs.microsoft.com/en-us/aspnet/web-api/overview/testing-and-debugging/unit-testing-with-aspnet-web-api

Assignment 11: Support Cross Origin Resource Sharing (CORS)

You must now extend your Rest Service to support CORS, so your Rest Service (API) can be consumed in scripting frontend pages, as Javascript/Typescript based application.
In the following we use the MVC approach; i.e. the policies can be different and specified for be for the whole application service, the controllers and the methods in each controller. In Appendix C another approach is described.

a. First you need to install NuGet-package, Microsoft.AspNetCore.Cors (2.1.1)
Remember to use the version (2.1.1 or 2.2.0) that fits the your Asp.Net version! Otherwise an error will be given!!
Maybe you can use Microsoft.AspNet.WebApi.Cors, originally made for EntityFramework, but I don't recommend that.
In your project open the NuGet manager and choose the package to be installed:



and yes … It takes a little time…..

In the following we use the MVC approach; i.e. the rules/policies are specific for the service, for each controller and each method.

a. In the solution (Solution Explorer) open the file Startup.cs.
   In the *ConfigureServices* method, add various policies like:

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

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


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

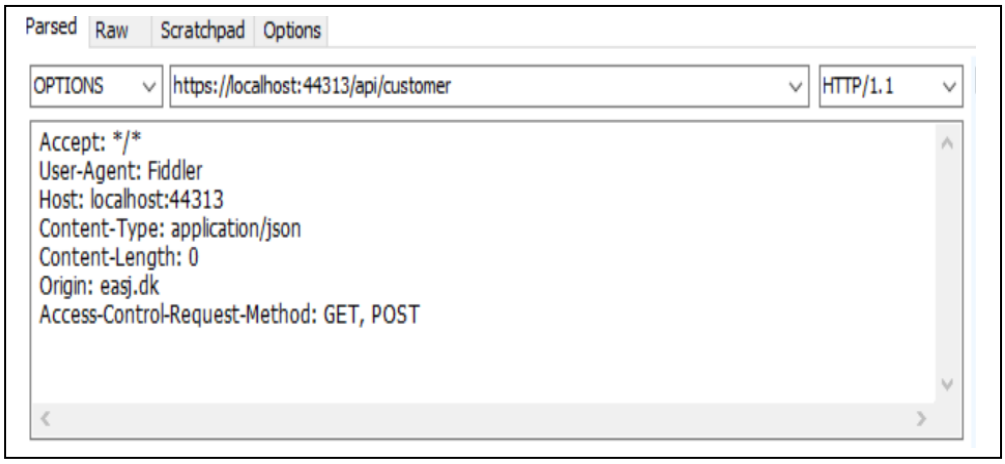
b. Still in the class, Startup.cs. In the *Configure*-method, before *app.UseAuthorization()* and after *app.UseRouting()* add the line:

```
app.UseCors("AllowAnyOriginGetPut"); // or another policy
```

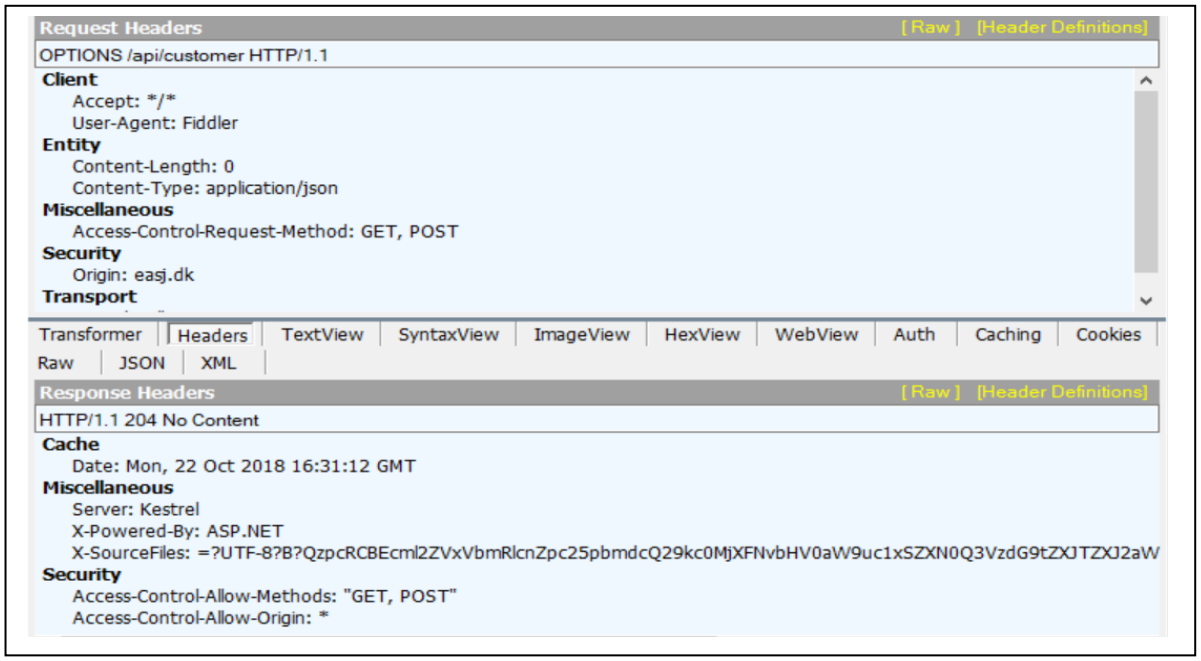
We are now ready to test the new project by setting up a Preflight Request using Fiddler/Postman.

c. Try to send a Preflight Request from Fiddler/Postman
   Be aware that you must:
   a. Click on Composer
   b. Choose OPTIONS
   c. Define the Content-Type: application/json
   d. Define Origin: easj.dk //or similar
   e. Define Access-Control-Request-Method: ANY or GET, POST, PUT //or more

It will look something like this:


Parsed
Raw
Scratchpad
Options
OPTIONS
https://localhost:44313/api/customer
HTTP/1.1
Accept: */*
User-Agent: Fiddler
Host: localhost:44313
Content-Type: application/json
Content-Length: 0
Origin: easj.dk
Access-Control-Request-Method: GET, POST

Click on *Execute* and hopefully the Response will look something like this:

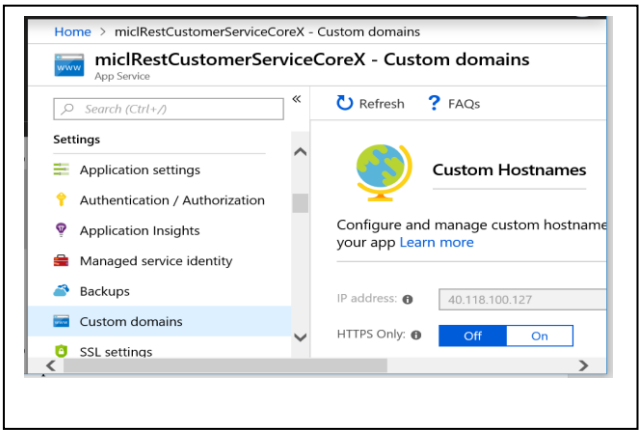


Notice the **Security**: Access-Controls- answers. This means the service is ready for communication with the client (Javascript/Typescript/Fiddler etc.)

d. Now try to invoke the methods GET, POST etc. from Fiddler.

e. Publish your service in Azure in a **new** Web-App and also set up a Preflight Request similar as the one for Localhost.
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…

f. Go to your Azure Portal
   a. Open your Web-App project
   b. Find *Custom domains* in the left scroll-bar
   c. Set *Https-Only* to OFF
   d. Click *Refresh*



g. Now set up a Preflight Request similar as the one for Localhost.
Finally try to invoke the methods GET, POST, PUT etc. from Fiddler.

NEARLY DONE !!

The next questions are for the fast students.
For even more detailed CORS setup, one can specify policy for the controller suppressing the service policy. Furthermore, one can specify the policy for each method, suppressing the controller- policy.

a. In the controller, *CustomerController*, specify the policy you want on the controller itself, suppressing the service policy; like:

```
[Route("[controller]")]
[EnableCors("AllowAnyOrigin")]
[ApiController]
```

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

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

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

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

[HttpPost]
[EnableCors("AllowSpecifOrigin")]
```

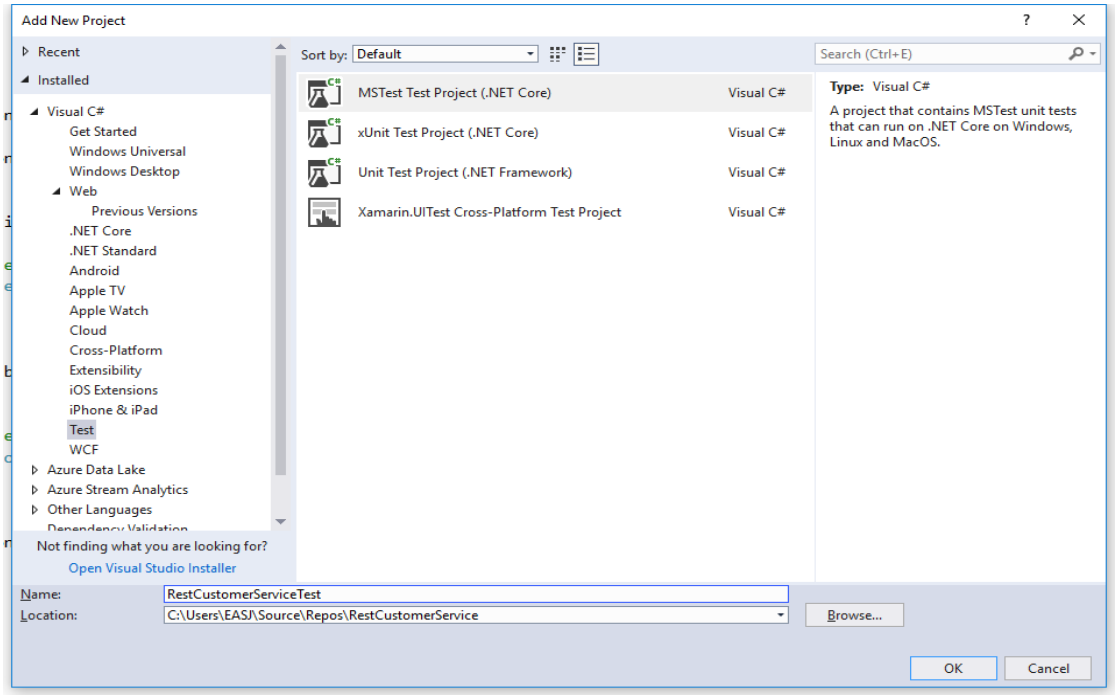
c. Publish again in Azure, and check your new configuration using Fiddler or Postman like in previous assignment.

Assignment 12: Testing your REST service

To be sure that your REST service are working correctly, you make a component test (Unit test) of the controller.

When testing you first create a test-project, then implement the test methods and finally run the test.

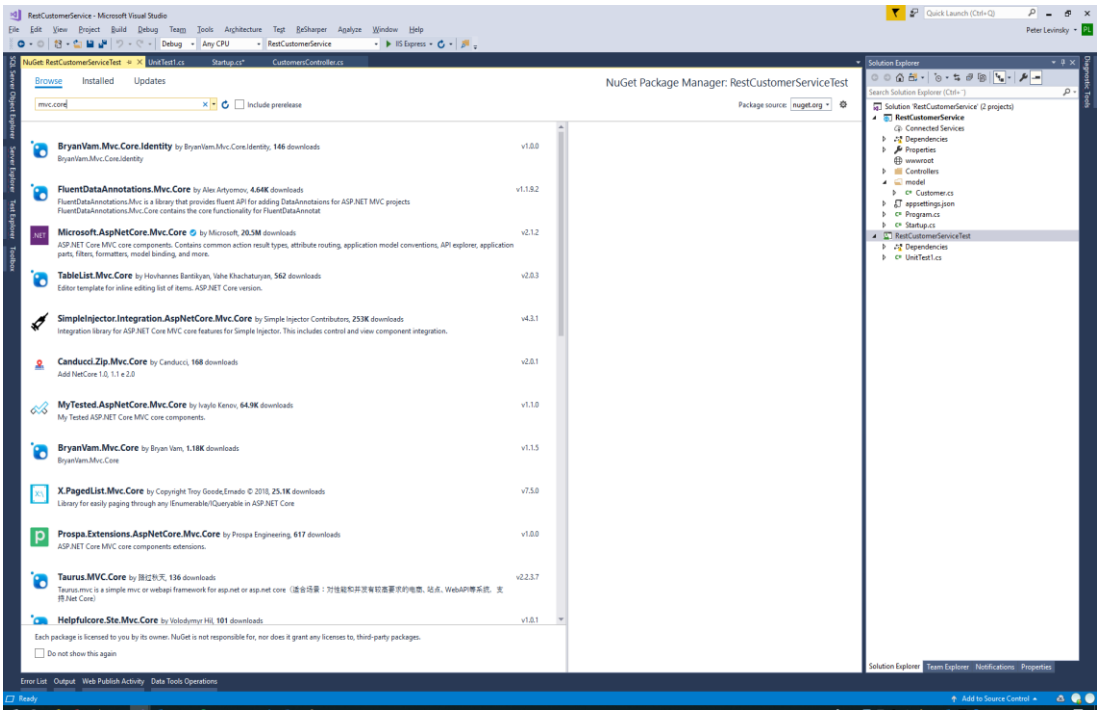
a. In your solution you have to create a new project for testing purpose.
In your solution right-click to add a new project.
Choose Test and then pick MSTest test project (.Net core) and give it a name e.g.: RestCustomerServiceTest.



Before implementing the test methods, you need two steps:

1. Let your Test-project refer to your Rest-project i.e. at the dependencies right-click and add a reference to the rest-project in your ‘Projects’.

2. To your Test-project add a NuGet package, browse for ‘mvc.core’ choose Microsoft.AspNetCore.Mvc.Core.
IF you later are making a test on a Rest-project utilizing a database you *might* also need System.Data.SQLClient.
Maybe not be necessary IF you have clicked and accepted all Updates in NuGet.



b. Implement your Test Methods by instantiate an object of your CustomerController and call some methods and assert the result.

c. Run your Test by right-click and run Test.

Assignment 13: Refactor the consumer code

Refactoring is about making the code either smarter (faster, better overview, library usage) or downsizing the number of code lines!

Take a look at your consumer code.

Its messy and a lot of stuff in one sequence…. A more structured approach and program is beneficial.

Can you do something about it?!

CONGRATULATIONS YOU NOW HAVE A PROFFESIONAL RESTFUL WEB SERVICE

Now you and others can later utilize your rest service from Typescript/Javascript etc..

Appendix A: Running from Fiddler/Postman

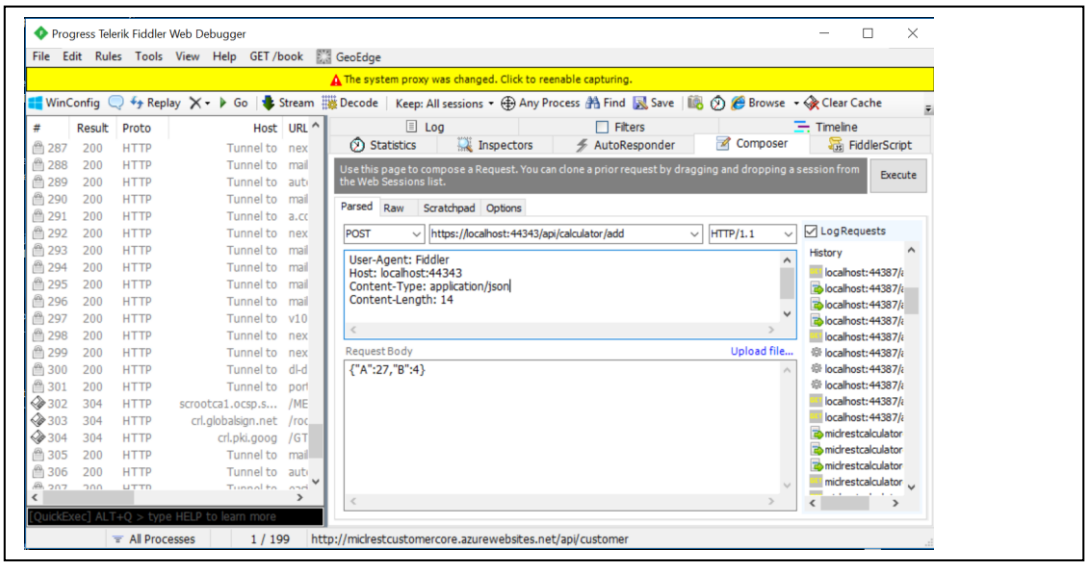
This a is an example for using a service add on two integers. It will be similar for services on customers.

Try to invoke the method from Fiddler/Postman

Be aware that you must:

f. Click on Composer
g. Choose POST
h. Define the Content-Type: application/json
i. Request body must hold the Customer as a Json-string

It will look something like this:

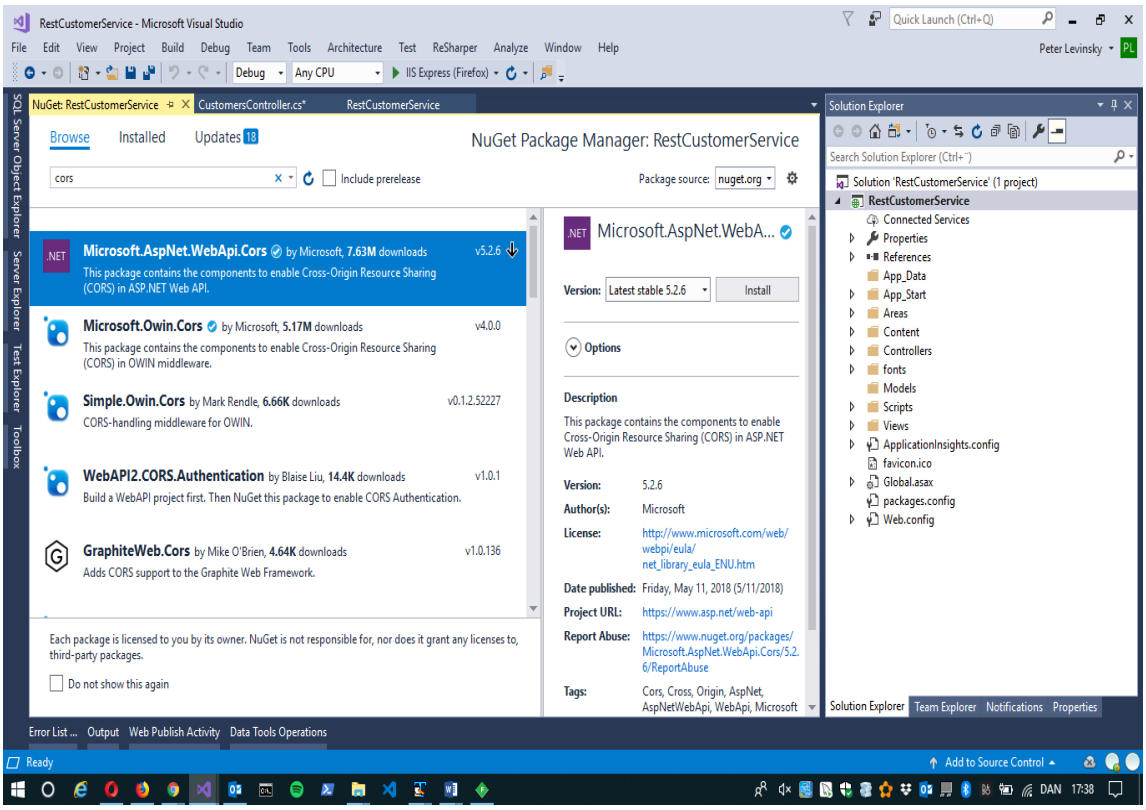


Click on *Execute* and hopefully you get the sum.

Appendix B: MVC CORS (Needed for ASP.Net 3.1)

This section describes how to extend your Rest Service to support CORS, so your Rest Service (API) can be consumed in scripting frontend pages, as Javascript/Typescript based application. In the following we use the MVC approach; i.e. the rules/policies are specific for the service, for each controller and each method.

d. First you need to install NuGet-package, Microsoft.AspNet.Core.Cors.
   Remember to use the version (2.1.1 or 2.2.0) that fits the your Asp.Net version! Otherwise an error will be given!!
   In your project open the NuGet manager and choose the package to be installed:



and yes … It takes a little time…..

In the following we use the MVC approach; i.e. the rules/policies are specific for the service, for each controller and each method.

e. In the solution (Solution Explorer) open the file Startup.cs.
In the *ConfigureServices* method, add various policies like:

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

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


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

f. Still in the class, Startup.cs. In the *Configure*-method, before *app.UseAuthorization()* and after *app.UseRouting()* add the line:

```
app.UseCors("AllowAnyOriginGetPut"); // or another policy
```

g. In the controller, *CustomerController*, specify the policy you want on the controller itself, suppressing the project policy; like:

```
[Route("[controller]")]
[EnableCors("AllowAnyOrigin")]
[ApiController]
```

h. 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("AllowSpecifOrigin")]
```

We are now ready to test the new project by setting up a Preflight Request using Fiddler/Postman.

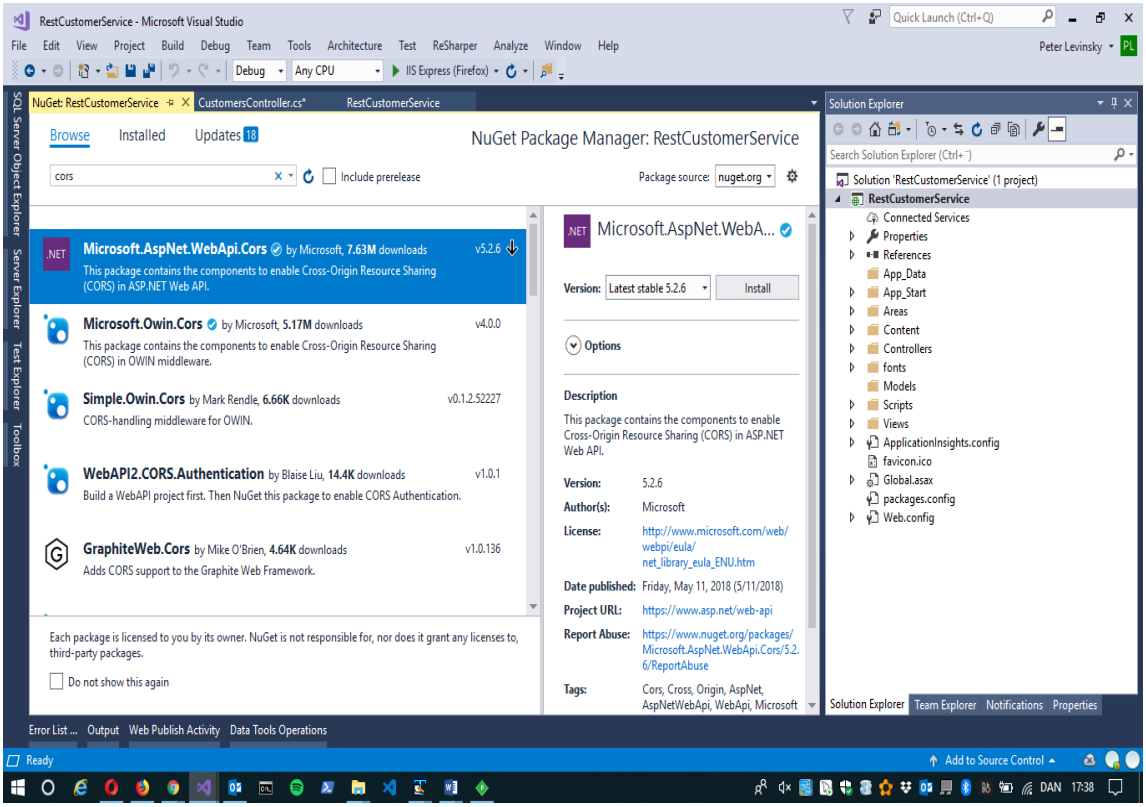
i. Just follow the Preflight guideline from assignment 11 d-h.
Check if the different policies actually work…?!!

Investigate more by yourself.

Appendix C: Middleware CORS (ASP.Net 2.1)

This section describes how to extend your Rest Service to support CORS, so your Rest Service (API) can be consumed in scripting frontend pages, as Javascript/Typescript based application. In the following we use the Middleware approach; i.e. the rules/policies will be valid for the whole application service (all controllers and all methods follow the rules).

a. First you need to install NuGet-package, Microsoft.AspNet.Core.Cors.
Remember to use the version (2.1.1 or 2.2.0) that fits the your Asp.Net version! Otherwise an error will be given!!
In your project open the NuGet manager and choose the package to be installed:



and yes … It takes a little time…..

In the following we use the Middleware approach; i.e. the rules/policies will be valid for the whole application service (all controllers and all methods follow the rules).

a. In the solution (Solution Explorer) open the file Startup.cs.
   In the *ConfigureServices* method, add the line

```
services.AddCors();
```

b. Still in the class, Startup.cs.
   In the Configure-method, before the *app.UseMvc()* call, add the lines:

```
app.UseCors(
    options =>
       {options.AllowAnyOrigin().AllowAnyMethod();
  // allow everything from anywhere
            });
```

We are now ready to test the new project by setting up a Preflight Request using Fiddler/Postman.

T