

COMPUTING SUBJECT:	Restful ASP.Net Core-services
TYPE:	Assignment
IDENTIFICATION:	RestCustomerService
COPYRIGHT:	<i>Michael Claudius & Peter Levinsky</i>
LEVEL:	Medium
TIME CONSUMPTION:	2-5 hours
EXTENT:	60 lines
OBJECTIVE:	Restful services based on ASP.Net Core
PRECONDITIONS:	Exercise RestCalculatorService is a must Rest service theory. Http-concepts Computer Networks Ch. 2.2
COMMANDS:	

IDENTIFICATION: RestCustomer /MICL&PELE

Purpose

The purpose of this assignment is to be able to provide and consume restful ASP.Net Core web services on objects of a specific class.

Precondition

You must have done the RestCalculatorService, as basic information and uidelines 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 Cross Origin Resource Sharing (CORS)
11. Set up a project for Unit test
12. Refactor the consumer code

This assignment holds all 12 steps.

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#:

Serializable Class

[https://msdn.microsoft.com/en-us/library/4abdf6k0\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/4abdf6k0(v=vs.110).aspx)

CRUD-Operations and routing

<https://docs.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/attribute-routing-in-web-api-2>

Building ASP web services

This is for ASP Framework but there are ideas about the programming

<https://docs.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/create-a-rest-api-with-attribute-routing>

Cors – Cross reference support

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

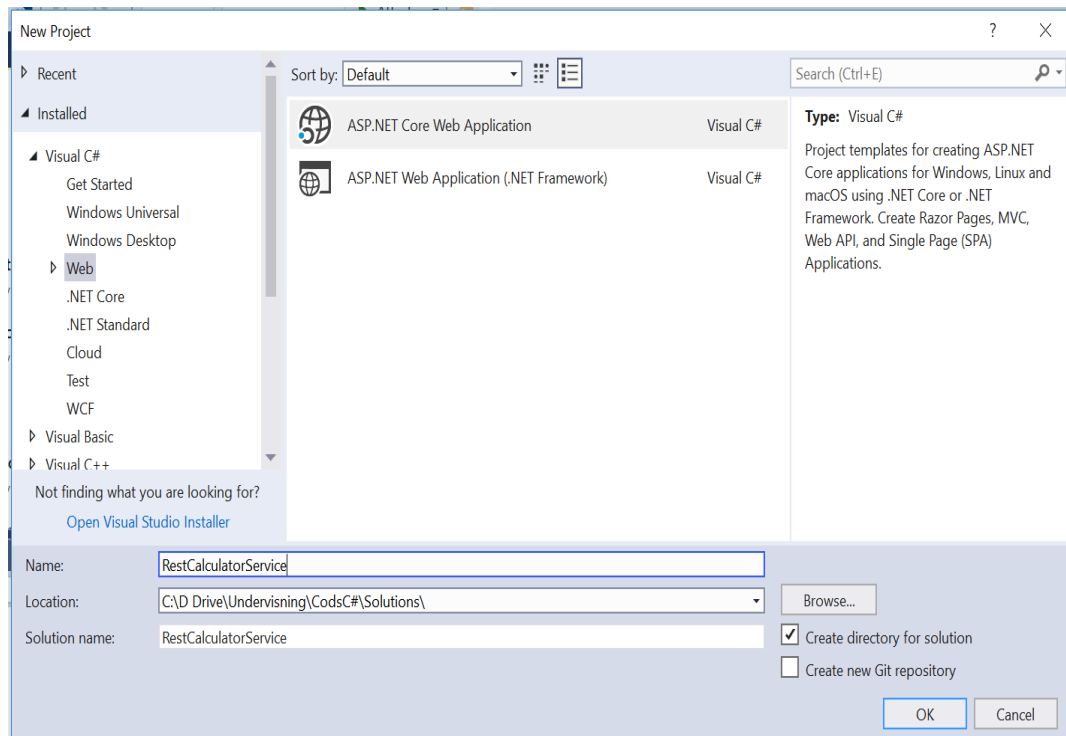
Assignment 1: Restful ASP.Net Framework-service provider

You are to make a Rest Service provider RestCustomerService.

Start Visual Studio:File -> New -> Project.

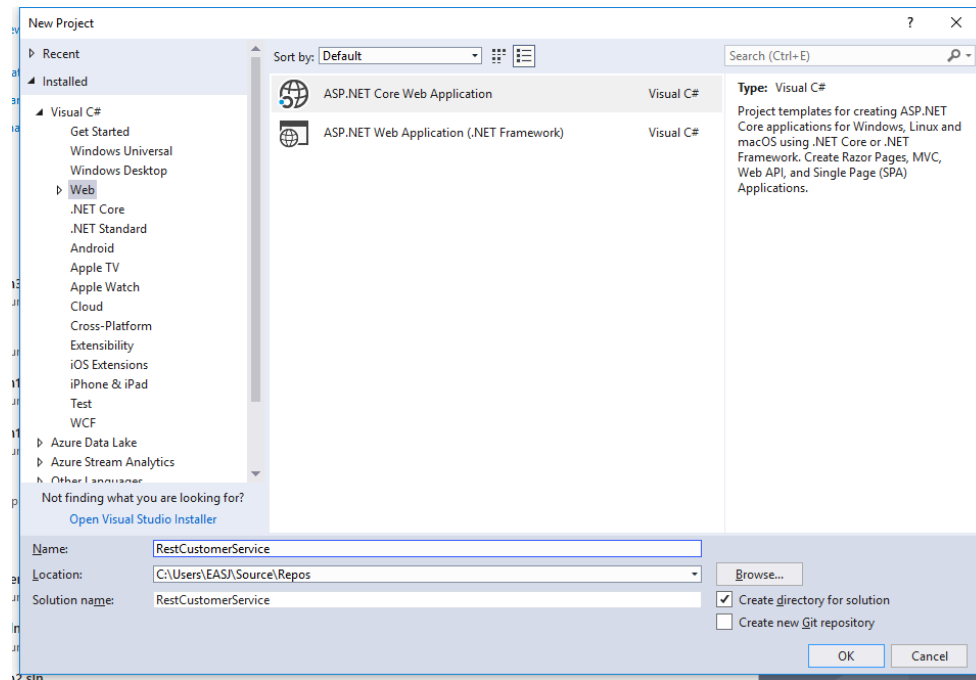
Choose: Web -> ASP.NET Core Web Application (**not .Net Framework**).

Browse to a convenient location and give the name RestCustomerService.



Click OK.

Now a project template with options are given.



Choose the API.

DON'T tick Docker support.

Tick HTTPS if you intend to use GoogleChrome or MicrosoftEdge as browser. And Fiddler as a tester-client. This has been tested by me (Michael Claudius) and it all works fine.

Don't tick HTTPS, if you intend to use Firefox as browser. And maybe, maybe not Postman, as it can have problems using https. But then of course https will **NOT** be supported later on.

Now you have to wait a while...

Execute the Application by viewing it in a local Browser. This will start the Azure emulator.

As you can see it takes the predefined URL:

<http://localhost:49972/api/values>

The port number (49972) will be different on your computer.

Now we are ready to extend the project with first a model class then a controller class.

Assignment 2: Model class Customer

We need a class, Customer, for customer objects. Therefore, to the project add (right click project, choose Add -> Folder) a folder named "Model" and in this folder add a public class, "Customer", with the data fields:

ID, unique identifier
 FirstName,
 LastName,
 Year, year of registration

with get/set method for all the fields; i.e. they are properties.
Make the constructors:

```
Customer(int id, string first, string last, int year)  
    Initializing all the data fields
```

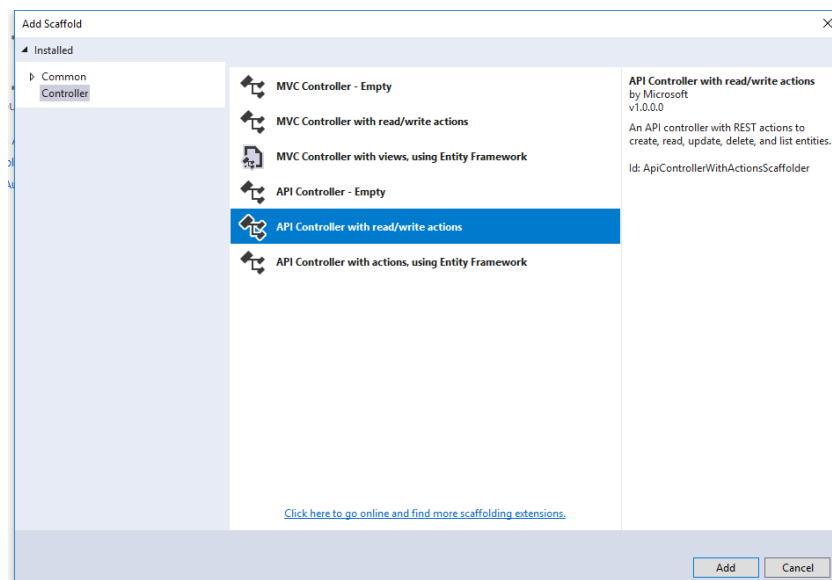
```
Customer() {}  
//empty constructor needed for JSON transfer. Serializable objects.
```

Assignment 3: REST API operation by creating a controller

You are to create a controller where the operation contracts must be defined as REST API routes and methods similar to ValuesController and CalculatorController.

In the solution in the controller folder, add (i.e. Right click) a new controller named “CustomerController”.

Choose ‘Web API Controller **with read/write actions**’.



Click Add and you can see the new controller.

Now we are ready to create customers and add services on them.

Assignment 4: Extend CustomerController with a list of customers

In *CustomerController* declare a static list holding three customers:

```
private static List<Customer> cList = new List<Customer>()
```

How to add customers so the list is always initialized with three customers?
Maybe you also like this:

```
public static int nextId = 0;
```

But what should that be used for...?

Assignment 5: Define the GET service GetCustomers

In *CustomerController* modify the first Get method (it's the one returning values) to support a Rest API GET request, that returns a list of all customers:

```
// GET: api/Customer
[HttpGet]
public IEnumerable<string> Get()
{
    return <your list> //cList
}
```

And implement the method to let it return your customer-list.

What is the full route to the Get operation?

Why did we choose to change HttpGet?

Assignment 6: Testing application in Browser/ and Fiddler/Postman

Execute the Application by viewing it in a local Browser. This will start the Azure emulator.
From the browser call the customer:

<http://localhost:44343/api/customer>

<http://localhost:44343/api/customer/1>

All fine?!

Not yet but we are close!!

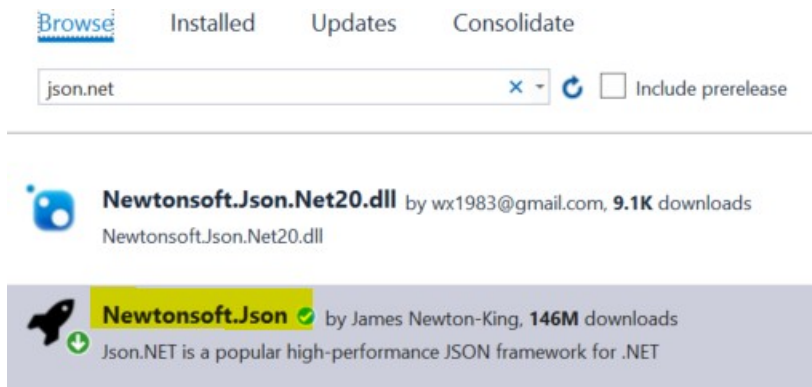
Try also to invoke the method from Fiddler/Postman.

Before adding more services, you will write a consumer program.

Assignment 7: Consumer: RestCustomerConsumer

Create a simple Console Application project "RestCustomerConsumer". Add a Customer class to the project, a class similar to the Customer class you used in the provider.

In order to serialize/deserialize objects, you must from NuGet install the package Newtonsoft.Json.



Now to consume the “Get” service, you in Program class (i.e. **Not inside main**) make a very special method:

```
public static async Task<IList<Customer>> GetCustomersAsync()
{
    using (HttpClient client = new HttpClient())
    {
        string content = await client.GetStringAsync(CustomersUri);
        IList<Customer> cList =
        JsonConvert.DeserializeObject<IList<Customer>>(content);
        return cList;
    }
}
```

Where the CustomerUri is the URI pointing to your service and method (api/customer).

- In *main* show how to use the method and print the list of customers.
Execute the program
- Carefully explain the code line-by-line what goes on.

Assignment 8: More services and usage by client/consumer

You must now extend the service (i.e. your controller) with more methods.

In *CustomerController* define more operations handling:

- Customer Get(int id)
Return the customer information with the specified id.
- DeleteCustomer(int id)
Delete (DELETE) the customer with the specified id.
- InsertCustomer(Customer c)
Insert (POST) the customer object in the list
- UpdateCustomer(int id, Customer c)
Update (PUT) a specified customer. Retrieves a specified customer, replace the old customer object in the list a new customer object, c.

- a. For each method, show how to use it from Fiddler/Postman.
- b. For each method, show how to use it in the consumer RestCustomerClient.
Rather similar to the Async-methods (get and post) you already made.

Notice: Remember for each method you must carefully think about:

- Which HTTP method/verb to use?
- What should the URI (Route) look like? Any parameters to the URI, like {id}?
- Do you want to specify the route explicitly like: `[Route("customer/{id:int}")]` for each method.
- Return type: true/false, customer object, id etc..

And write down your arguments.

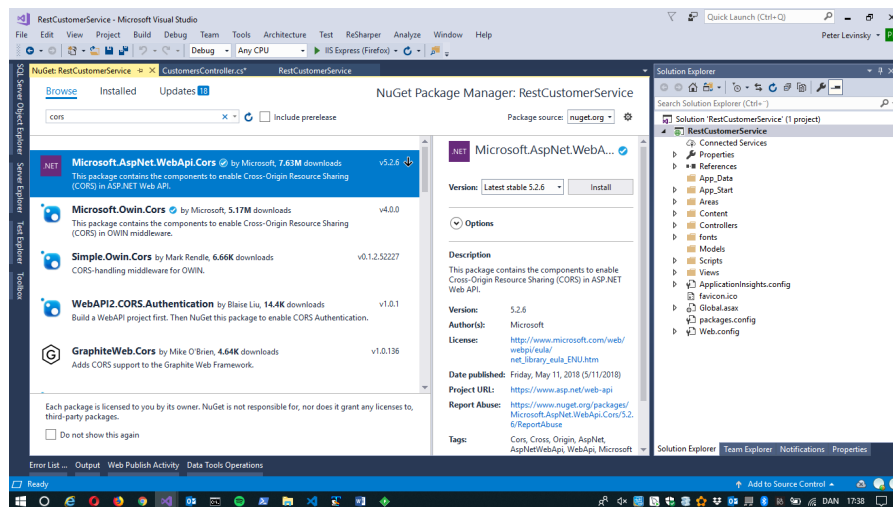
Assignment 9: Publish in Azure

- a. Publish your service in Microsoft Azure.
- b. Use a browser to show the API and the methods.
- c. Use Postman or Fiddler to show requests and responses.
- d. Show how to use the Azure service instead of the local URI in your consumer program.

Assignment 10: 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 a Typescript based application.

- a. First you need a NuGet-package installed, so at your project open the NuGet manager and choose WebApi.Cors to be installed:



and yes ... It do take some time ☹

- b. In the solution (Solution Explorer); Open the file Startup.cs.

In the ConfigureServices method, add the line

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

- c. Still in the Startup.cs – class in the Configure-method:

Add the lines:

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

- 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 it with some of your Typescript applications.

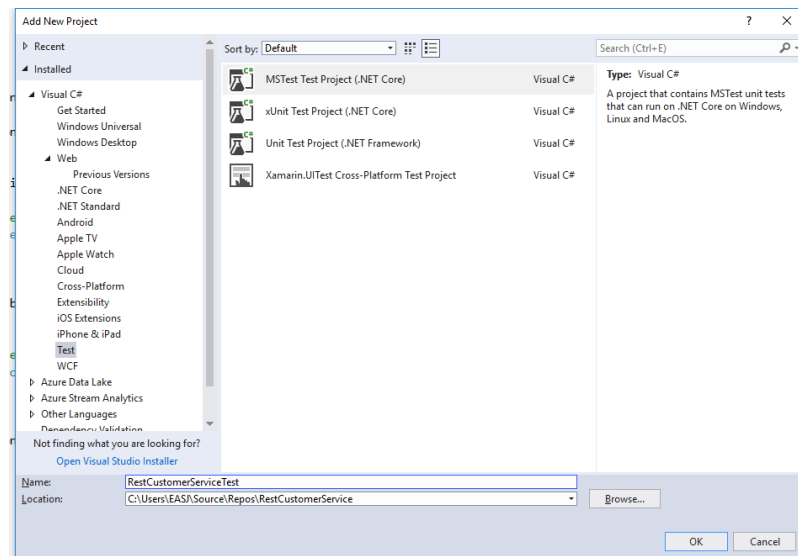
Assignment 11: Testing your REST service

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.

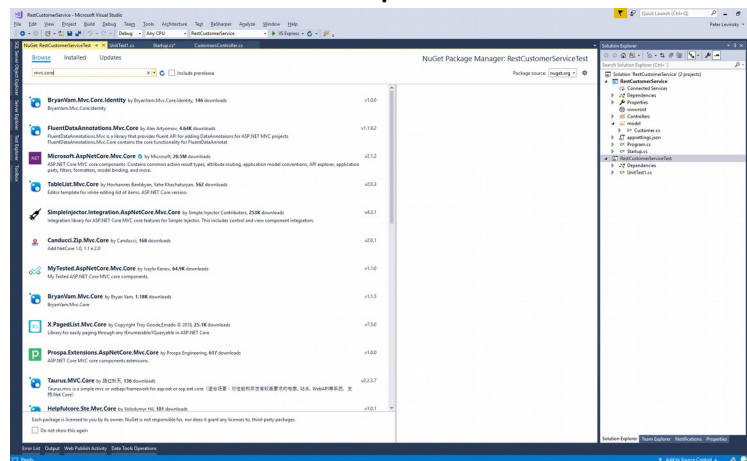
- a. Create a new project for testing purpose.
In your solution right-click to add a new project.

Choose Test and then pick MSTest test project (.Net core) and give it a name e.g.: RestCustomerServiceTest.



Before the 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. Let your Test-project add a NuGet package, browse for 'mvc.core' choose AspDotNet.Mvc.Core



- b. Implement your Test Methods by instantiate an object of your CustomerController and call some method and assert the result.
- c. Run your Test by right-click and run Test.

That's all folks.

Assignment 12: 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?!

Appendix A: Running from Fiddler/Postman

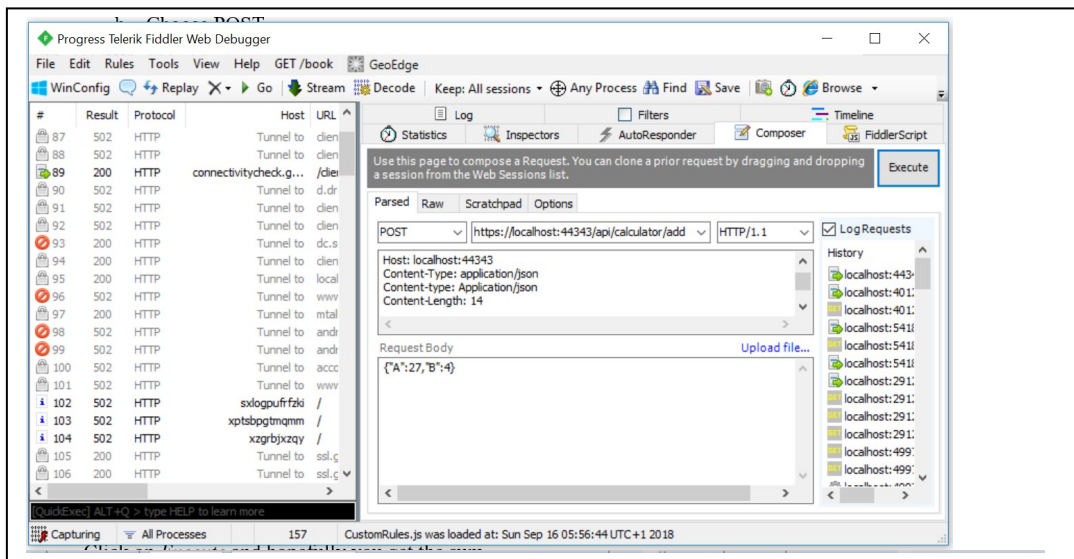
This is an example for using a service `add` on two integers. It will be similar for services on customers.

Try to invoke the method from Fiddler/Postman

Be aware that you must:

- Click on Composer
- Choose POST
- Define the Content-Type: `application/json`
- 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.