COMPUTING SUBJECT:	Restful ASP.Net Core-services
TYPE:	Assignment
IDENTIFICATION:	RestService#4
COPYRIGHT:	*Peter Levinsky & Michael Claudius*
LEVEL:	Medium
TIME CONSUMPTION:	1½-2½ hours
EXTENT:	80 lines
OBJECTIVE:	Testing the Restful services
PRECONDITIONS:	Rest service theory. Http-concepts Computer Networks Ch. 2.2
COMMANDS:	

IDENTIFICATION: RestService#4 / 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. (this assignment)**
5. Consuming a REST service from a C# Console application.
6. Adding Support for CORS to the REST Service
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)

This Assignment: RestService#4

Purpose
The purpose of this assignment is to test your REST Service and published in the Azure-cloud.

Mission
You are to control, your Restful web services based on the ASP.Net Core services is working correctly. You will be able set up at test-project (unit test) for the REST-Service and to publish it in the cloud in this case Azure:

1. Component test (unit test) of your Controller

2. Integration Test of your REST-Service

3. Publish in Azure

Additional reading:

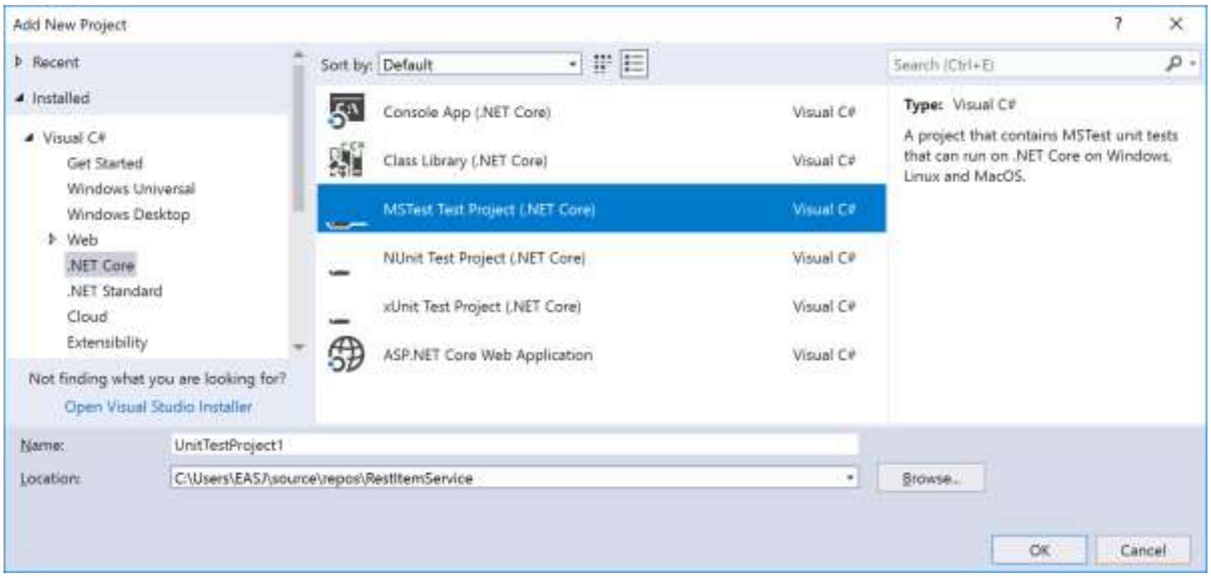
- **Unittest in C# / Visual Studio : https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019**
- **Walkthrough Test : https://docs.microsoft.com/en-us/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code?view=vs-2019**
- **Azure at Zealand : https://helpdesk.zealand.dk/hc/en-us/articles/205679095-Access-to-Azure-Dev-Tools-Microsoft-Imagine-**

Assignment 1: Component test (unit test) of your Controller

Before use of your REST-Service you should test the controller(s) in this case the '**ItemController**', by creating a unit test.

Here in .Net Core you are to create your test-project your-self, so follow these steps:

a. Add a new project to your solution. Right-click on the solution -> add new project – pick the MS-test (remember to choose among .Net Core applications):


Add New Project
Recent
Installed
Visual C#
Get Started
Windows Universal
Windows Desktop
Web
.NET Core
.NET Standard
Cloud
Extensibility
Not finding what you are looking for?
Open Visual Studio Installer
Sort by: Default
Search (Ctrl+E)
Console App (.NET Core) Visual C#
Class Library (.NET Core) Visual C#
MSTest Test Project (.NET Core) Visual C#
NUnit Test Project (.NET Core) Visual C#
xUnit Test Project (.NET Core) Visual C#
ASP.NET Core Web Application Visual C#
OK
Cancel
Browse...

Type: Visual C#

A project that contains MSTest unit tests that can run on .NET Core on Windows, Linux and MacOS.

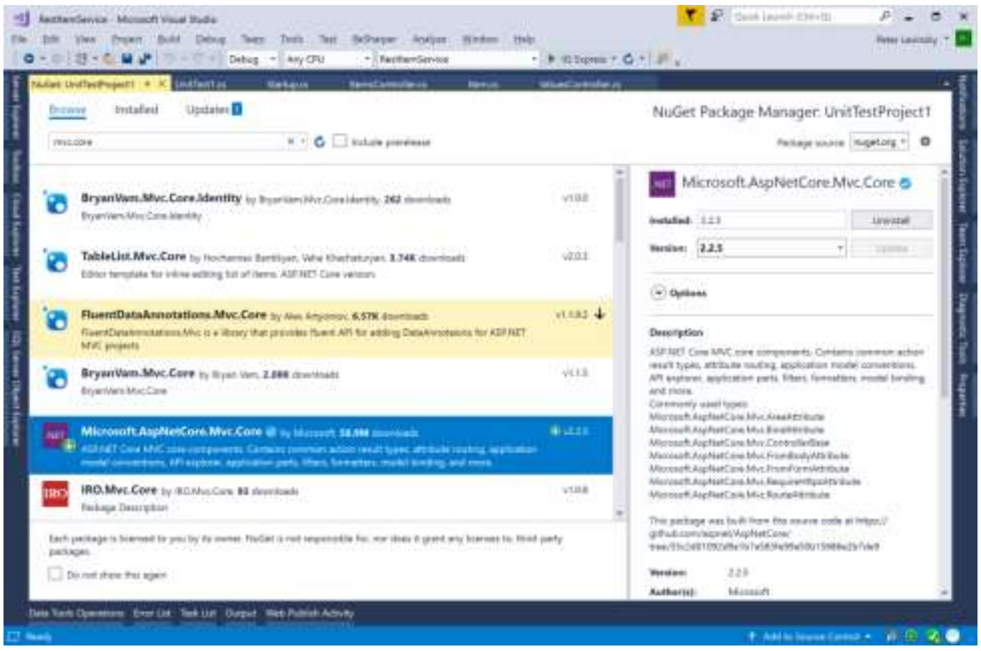
Name: UnitTestProject1

Location: C:\Users\EASJ\source\repos\RestItemService

(For 2019 version see picture at the end)

b. In the Test-project in 'dependencies' add a reference to your REST-Service project (i.e. to the controller to be tested). And add second reference to your '**ModelLib**'.

c. Add a Nuget package '**Microsoft.AspNetCore.Mvc.Core**' to your Test-project


NuGet Package Manager: UnitTestProject1

Browse | Installed | Updates 1

mvc.core   Include prerelease

Package source: nuget.org

BryanVam.Mvc.Core.Identity by BryanVam.Mvc.Core.Identity, 262 downloads v1.0.0
BryanVam.Mvc.Core.Identity

TableList.Mvc.Core by Nocharnas Bartikyan, Vahe Khachaturyan, 3.74K downloads v2.0.1
Editor template for inline editing list of items. ASP.NET Core version.

FluentDataAnnotations.Mvc.Core by Alex Anpomov, 6.57K downloads v1.1.8.2
FluentDataAnnotations.Mvc is a library that provides fluent API for adding DataAnnotations for ASP.NET MVC projects

BryanVam.Mvc.Core by Bryan Vam, 2.88K downloads v1.1.0
BryanVam.Mvc.Core

Microsoft.AspNetCore.Mvc.Core by Microsoft, 58.9M downloads v2.2.5
ASP.NET Core MVC core components. Contains common action result types, attribute routing, application model conventions, API explorer, application parts, filters, formatters, model binding, and more.

IRO.Mvc.Core by IRO.Mvc.Core, 88 downloads v1.0.8
Package Description

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

Microsoft.AspNetCore.Mvc.Core

Installed: 2.2.5   Uninstall

Version: 2.2.5   Update

Options

Description

ASP.NET Core MVC core components. Contains common action result types, attribute routing, application model conventions, API explorer, application parts, filters, formatters, model binding and more.
Commonly used types:
Microsoft.AspNetCore.Mvc.AreaAttribute
Microsoft.AspNetCore.Mvc.BindAttribute
Microsoft.AspNetCore.Mvc.ControllerBase
Microsoft.AspNetCore.Mvc.FromBodyAttribute
Microsoft.AspNetCore.Mvc.FromFormAttribute
Microsoft.AspNetCore.Mvc.RequireHttpsAttribute
Microsoft.AspNetCore.Mvc.RouteAttribute

This package was built from the source code at https://github.com/aspnet/AspNetCore/tree/[illegible]

Version: 2.2.5
Author(s): Microsoft

d. Now test all your methods (2 get (could be 4 with the two from exercise RESTService#2), 1 post, 1 put, 1 delete). Make use of Assert.
Remember to test for different parameters and return values.

e. Check your coverage of your test by at the 'Test-tab -> analyse code coverage -> all test'
What would be a good coverage?
If you have 100% coverage is your controller you are testing correct?

Assignment 2: Integration Test of your REST-Service

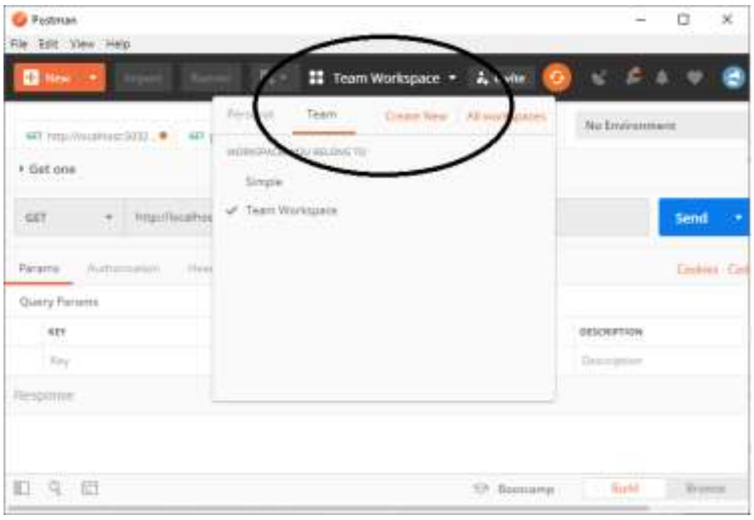
After testing that your controller functional are working correct, -- at least is tested ☺, you are to test your URI are well designed and working i.e. you are going to make an integration test. With this in mind you needed as minimum testing five entries (URI's):

1. Read All (Get)
2. Read One (Get)
3. Create one (Post)
4. Update one (Put)
5. Delete one (Delete)

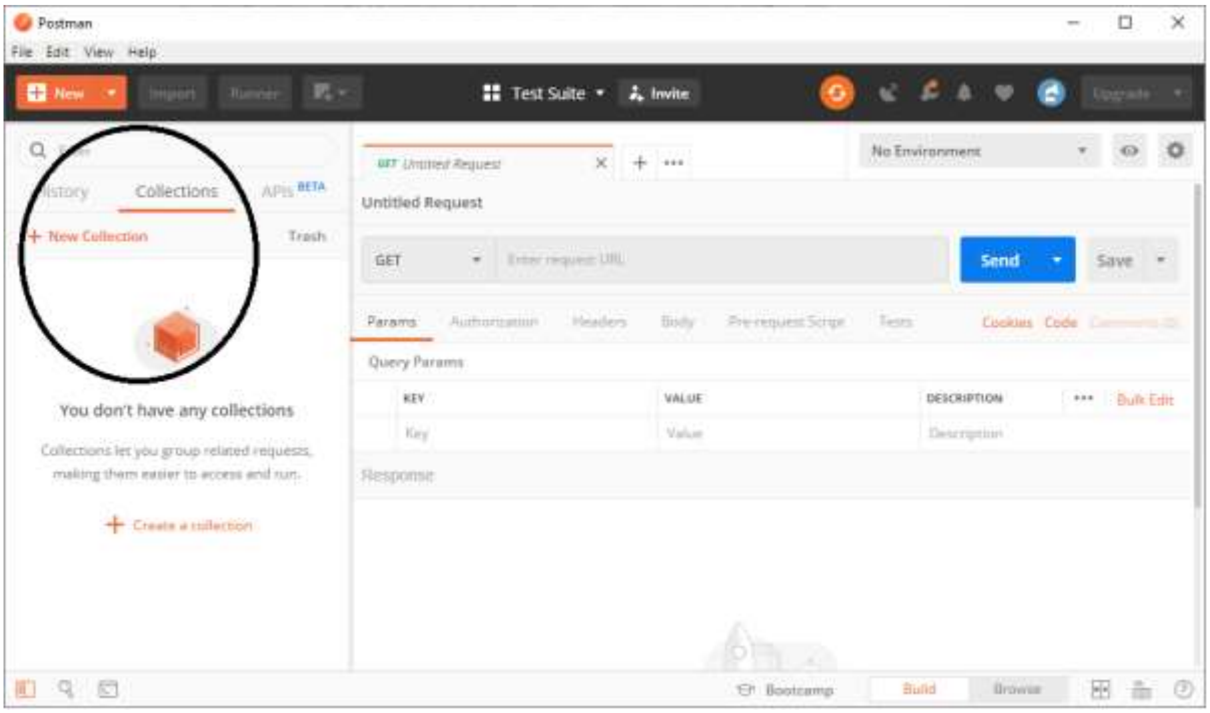
For this purpose you could have two approaches either make a new UnitTest-project, where you making HTTP-connection (see RESTService#5) OR you make a test-suite using Postman.

In this assignment you will go through steps to make a test suite in Postman.

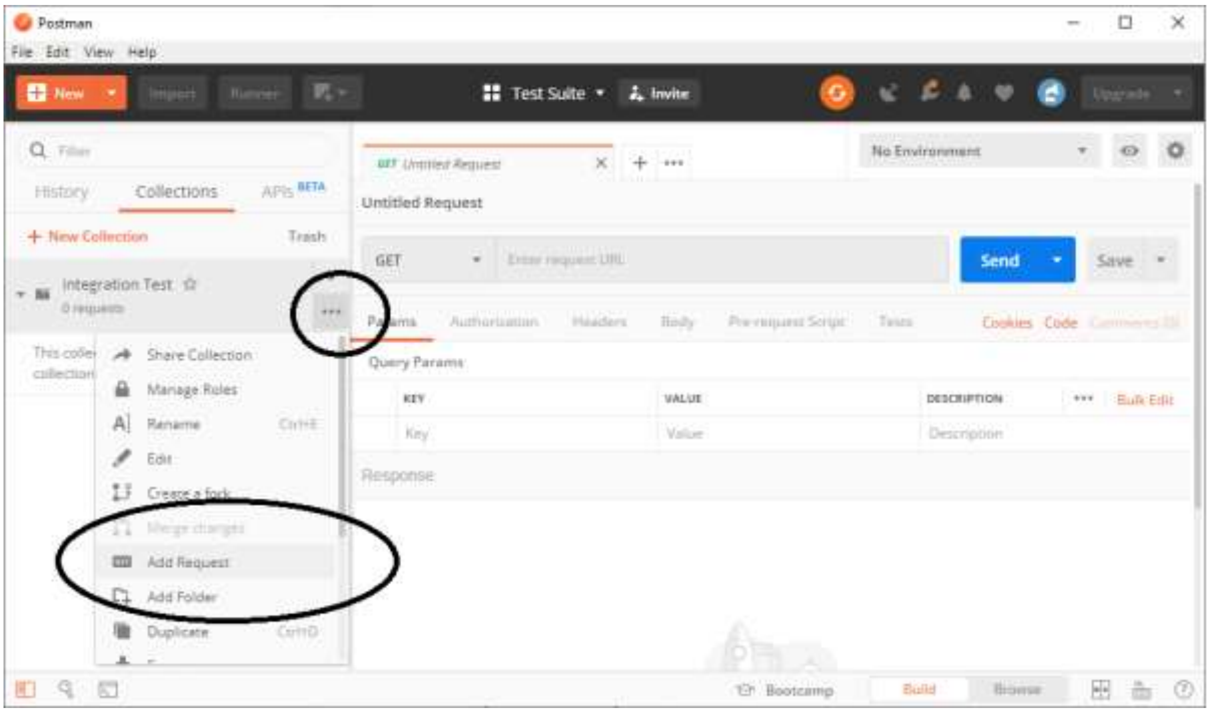
a. Open Postman and create a new workspace e.g. '**Test Suite**'


Postman
File Edit View Help
New
Team Workspace
Invite
Personal
Team
Create New
All workspaces
Simple
Team Workspace
No Environment
Get one
GET
Send
Params
Authorization
Query Params
KEY
DESCRIPTION
Response
Bootcamp
Build
Browse

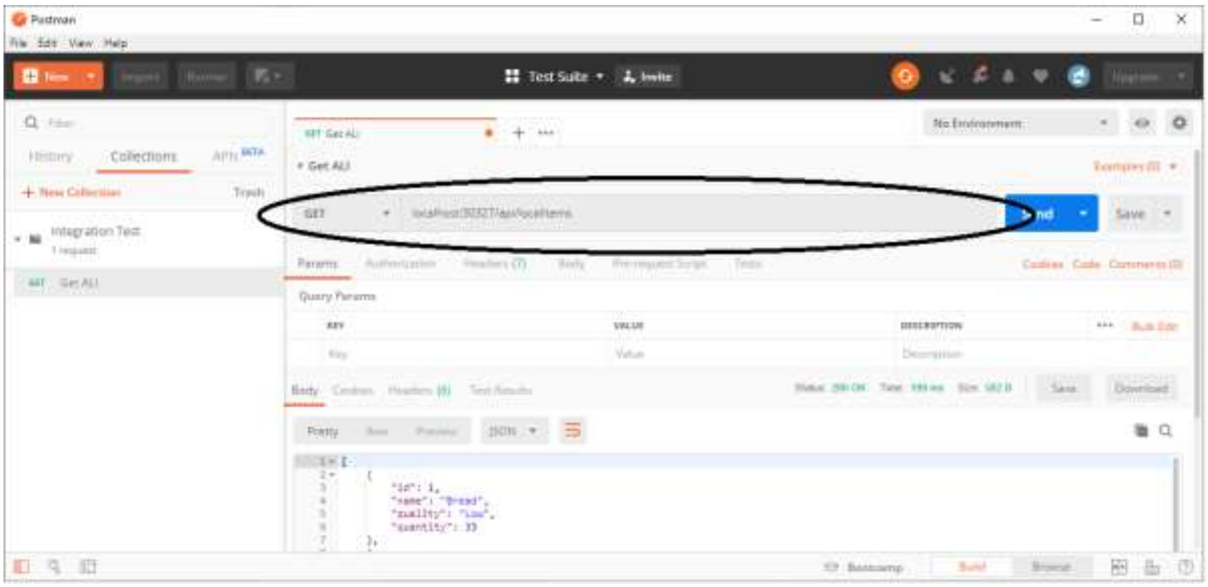
b. Create a new Collection e.g. ‘**Integration Test**’



c. Next Step is to add the different request to this collection



d. Now you fill out this request as normal i.e. choose method, write URI, and for Post and Put set the body as a json-string e.g. first request 'Get ALL' like:



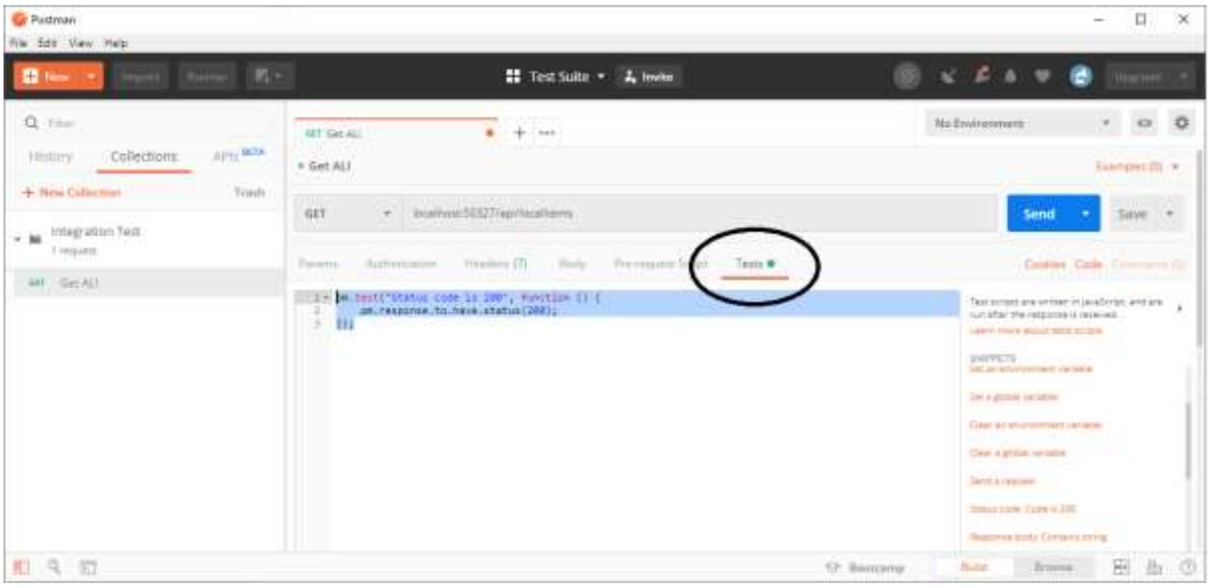
```
[
    {
        "id": 1,
        "name": "Bread",
        "quality": "Low",
        "quantity": 33
    },
```

e. Next step is set up the test conditions. This is done in the test-tab using a postman-scripting language.

References to the scripting language:
* https://learning.getpostman.com/docs/postman/scripts/postman_sandbox_api_reference/
* https://learning.getpostman.com/docs/postman/scripts/intro_to_scripts/

You typical need to test the status code:

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

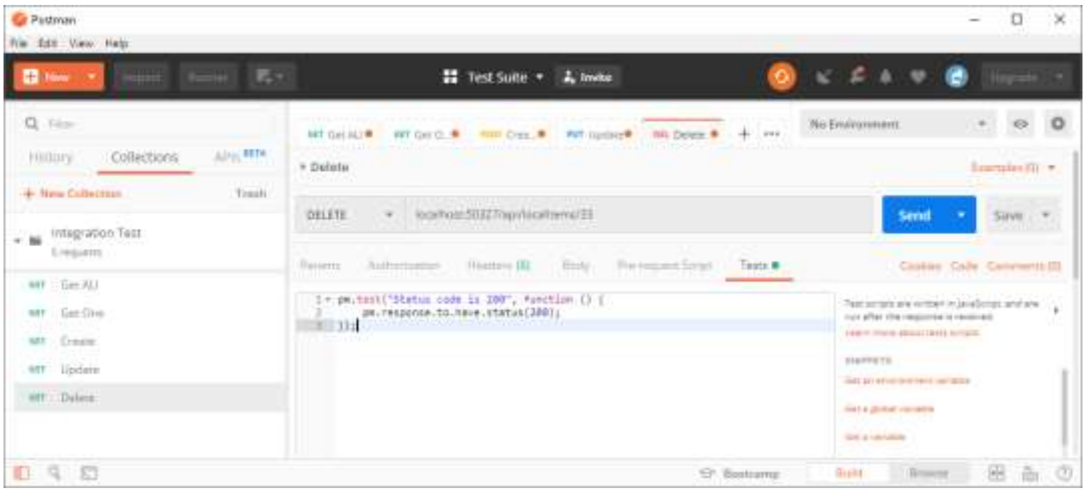


```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Test scripts are written in JavaScript, and are run after the response is received.

f. You can try this individual request by clicking ‘send’ if it is satisfied, then click ‘save’.

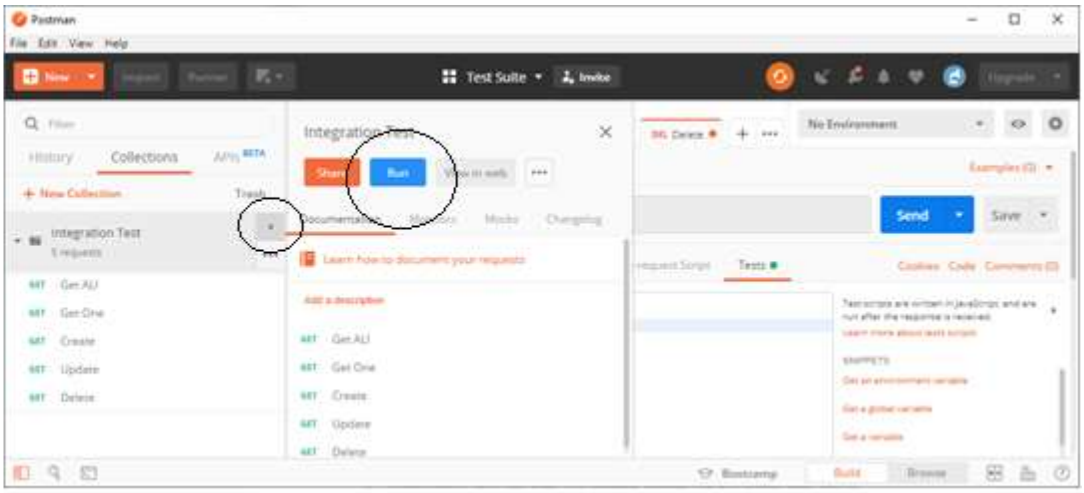
g. After all request have been added to the collection:



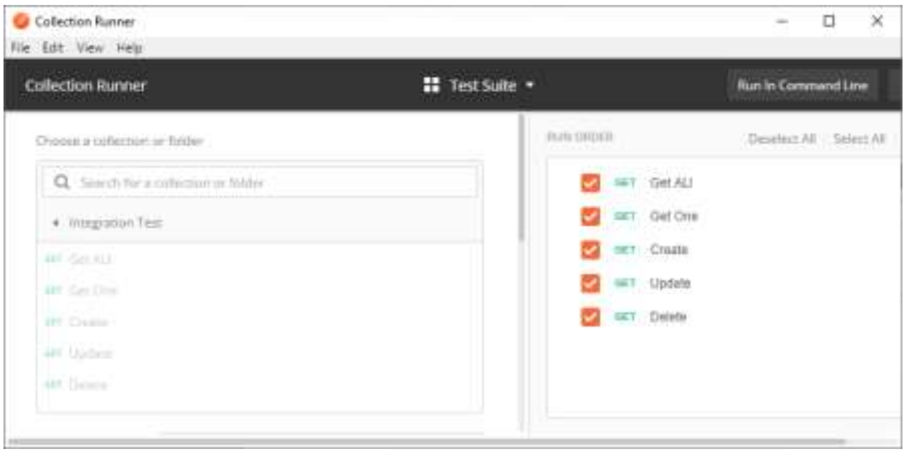
DELETE localhost:50327/api/localitems/33

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

h. You can run the whole collection i.e. the test suite by:



And your result should look like:



i. *Extra: Add more in deep test of the result – using the scripting language i.e. check the content of the result body e.g. 'pm.response.to.have.json;'*

Now you have tested your REST Service functional as well as an integration, so it is ready to be published to the cloud – at Zealand meaning Microsoft Cloud 'Azure'.

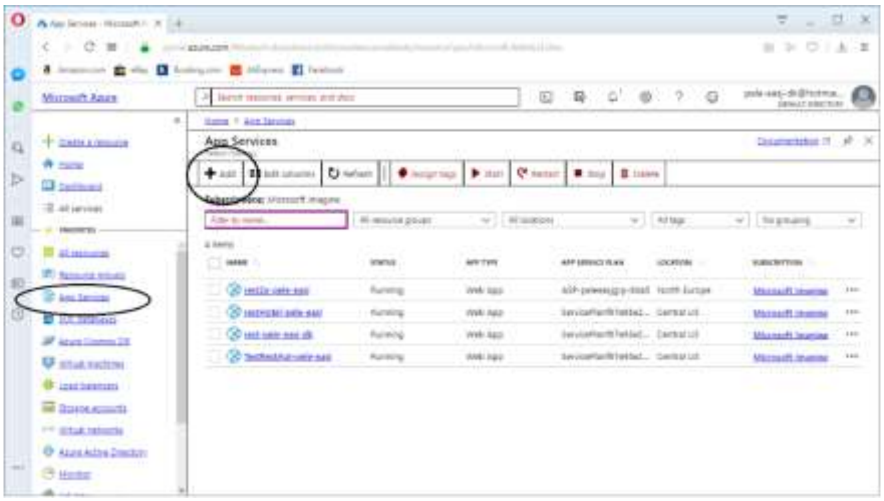
Assignment 3: Publish in Azure

To publish in Azure you have to create an app-service in Azure i.e. make a virtual machine with a web-server (MicroSoft IIS-server). Then to upload (publish) your REST-Service. You need to make a quick fix to support typescript accessing your REST service

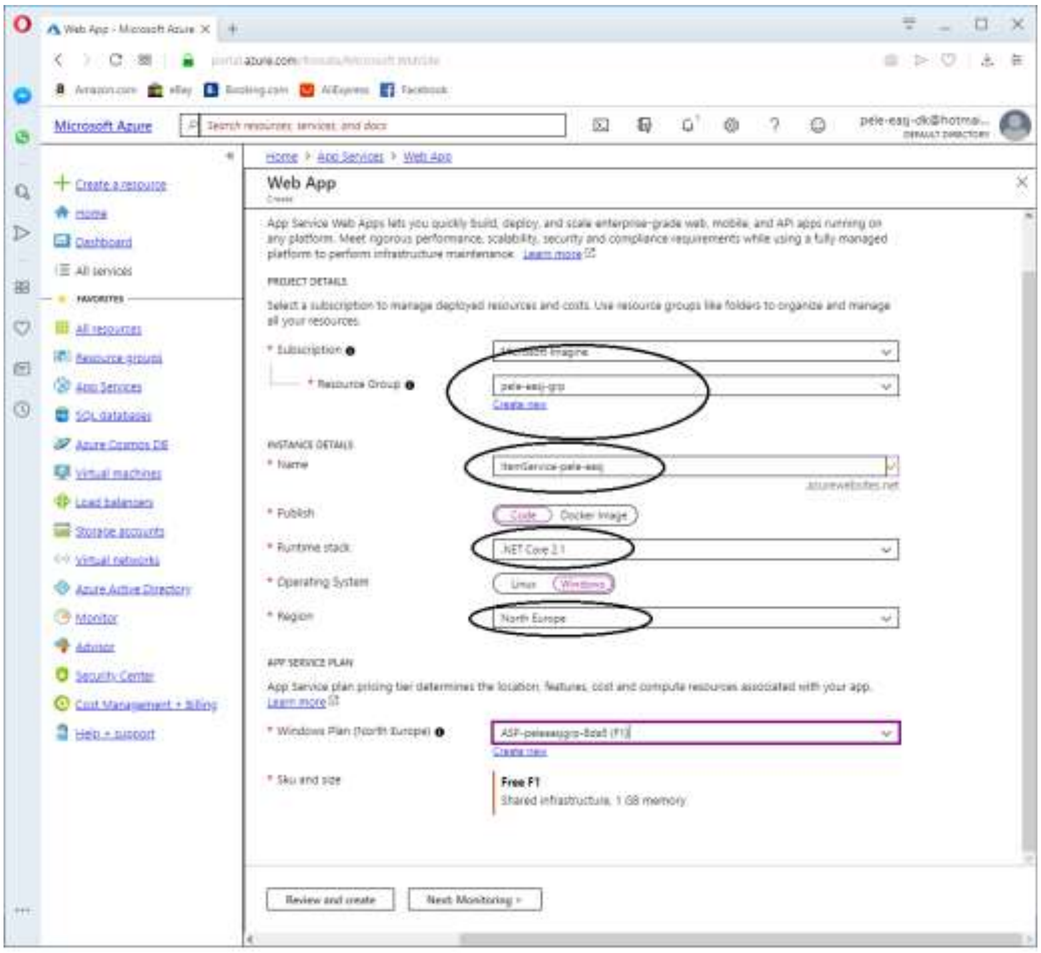
All this require you have an Azure account – see https://helpdesk.zealand.dk/hc/en-us/articles/205679095-Access-to-Azure-Dev-Tools-Microsoft-Imagine- for more information.

a. Go to your portal of Azure (https://portal.azure.com/) and log in.
You must create a new APP-service ‘**ItemService**’

Part 1



Part 2



Web App

Create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. Learn more

PROJECT DETAILS

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription: Microsoft Imagine
* Resource Group: pele-easj-grp

INSTANCE DETAILS

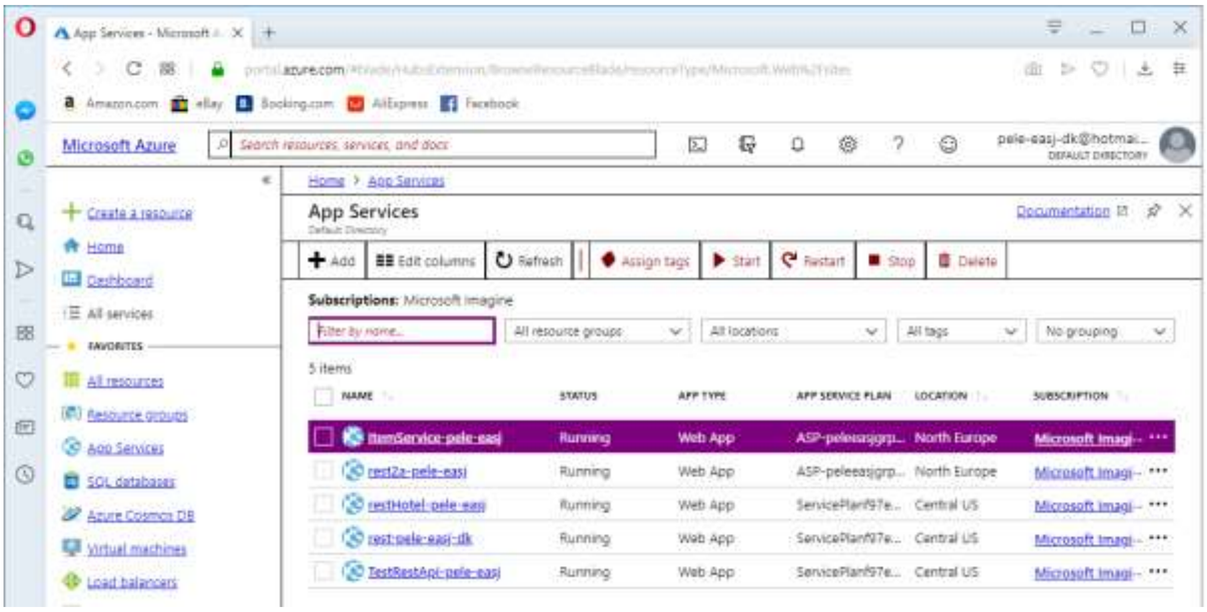
* Name: ItemService-pele-easj .azurewebsites.net
* Publish: [✓ Code] Docker Image
* Runtime stack: .NET Core 2.1
* Operating System: Linux [✓ Windows]
* Region: North Europe

APP SERVICE PLAN

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. Learn more

* Windows Plan (North Europe): ASP-peleeasjgrp-8da5 (F1)
* Sku and size: Free F1, Shared infrastructure, 1 GB memory

Part 3



App Services

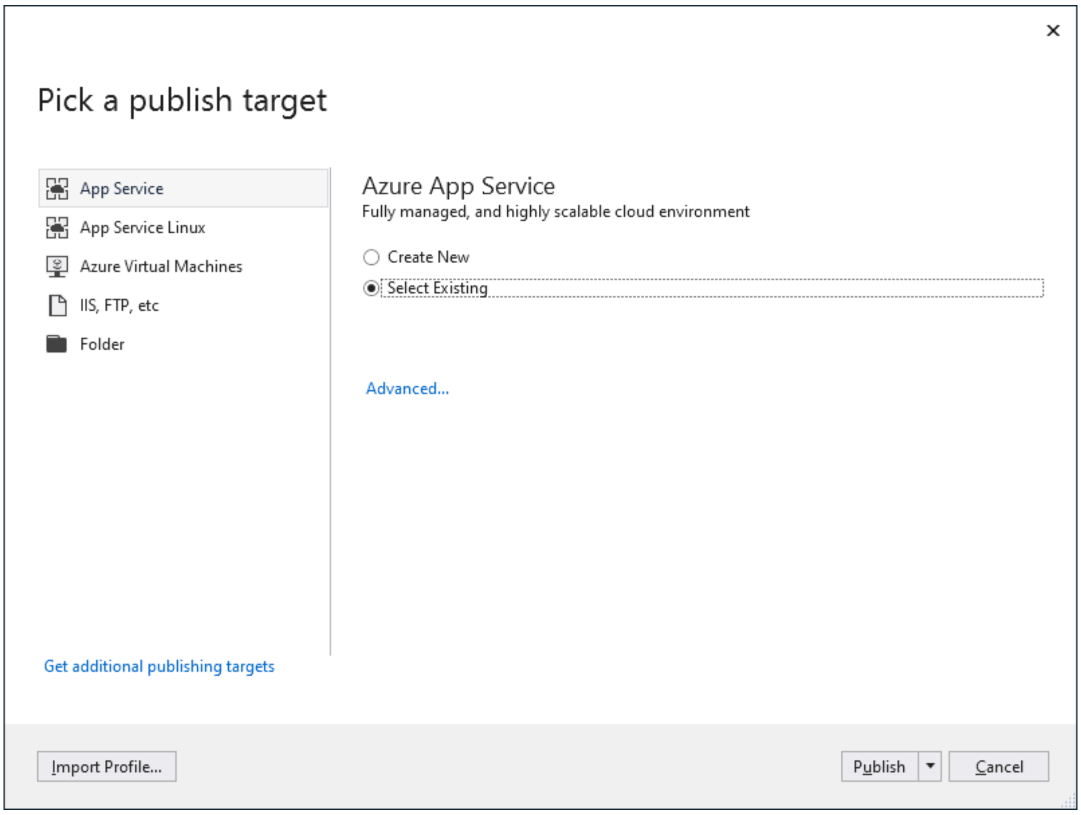
Subscriptions: Microsoft Imagine

5 items

NAME	STATUS	APP TYPE	APP SERVICE PLAN	LOCATION	SUBSCRIPTION
itemService-pele-easj	Running	Web App	ASP-peleeasjgrp...	North Europe	Microsoft Imagi...
rest2a-pele-easj	Running	Web App	ASP-peleeasjgrp...	North Europe	Microsoft Imagi...
restHotel-pele-easj	Running	Web App	ServicePlanf97e...	Central US	Microsoft Imagi...
rest-pele-easj-dk	Running	Web App	ServicePlanf97e...	Central US	Microsoft Imagi...
TestRestApi-pele-easj	Running	Web App	ServicePlanf97e...	Central US	Microsoft Imagi...

Now you are ready for the next step.

b. In Visual Studio open the Solution Explore.
Right-click at the project -> choose publish -> choose select existing



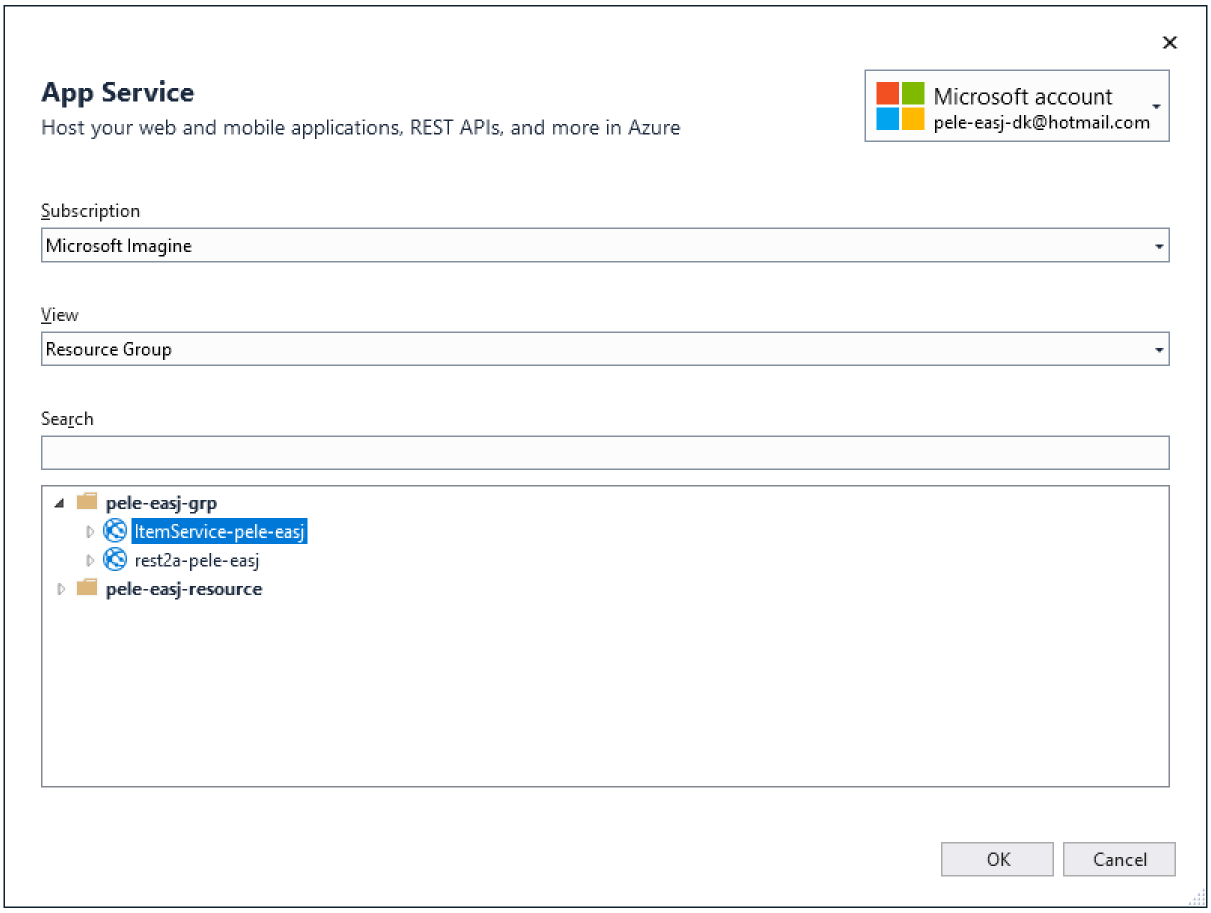
Pick a publish target

Azure App Service

Fully managed, and highly scalable cloud environment

Create New [✓ Select Existing]

Choose your APP-Service e.g. ItemService (in example 'ItemService-pele-easj'):


App Service

Host your web and mobile applications, REST APIs, and more in Azure

Microsoft account
pele-easj-dk@hotmail.com

Subscription: Microsoft Imagine

View: Resource Group

Search:

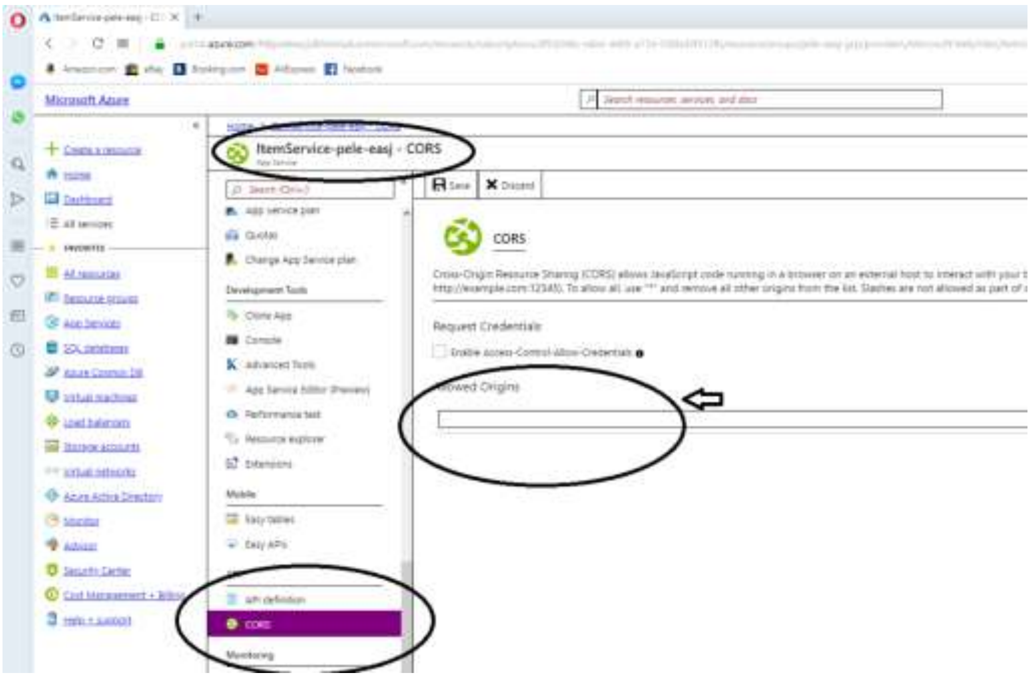
- pele-easj-grp
  - ItemService-pele-easj
  - rest2a-pele-easj
- pele-easj-resource

So that was it. A browser window will open with your RestService running in Azure with a URL-name like: http://itemservice-pele-easj.azurewebsites.net/index.html .

c. <u>Quick solution in Azure</u>

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

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



Cross-Origin Resource Sharing (CORS) allows JavaScript code running in a browser on an external host to interact with your
http://example.com:12345). To allow all, use "*" and remove all other origins from the list. Slashes are not allowed as part of

Request Credentials

Enable Access-Control-Allow-Credentials

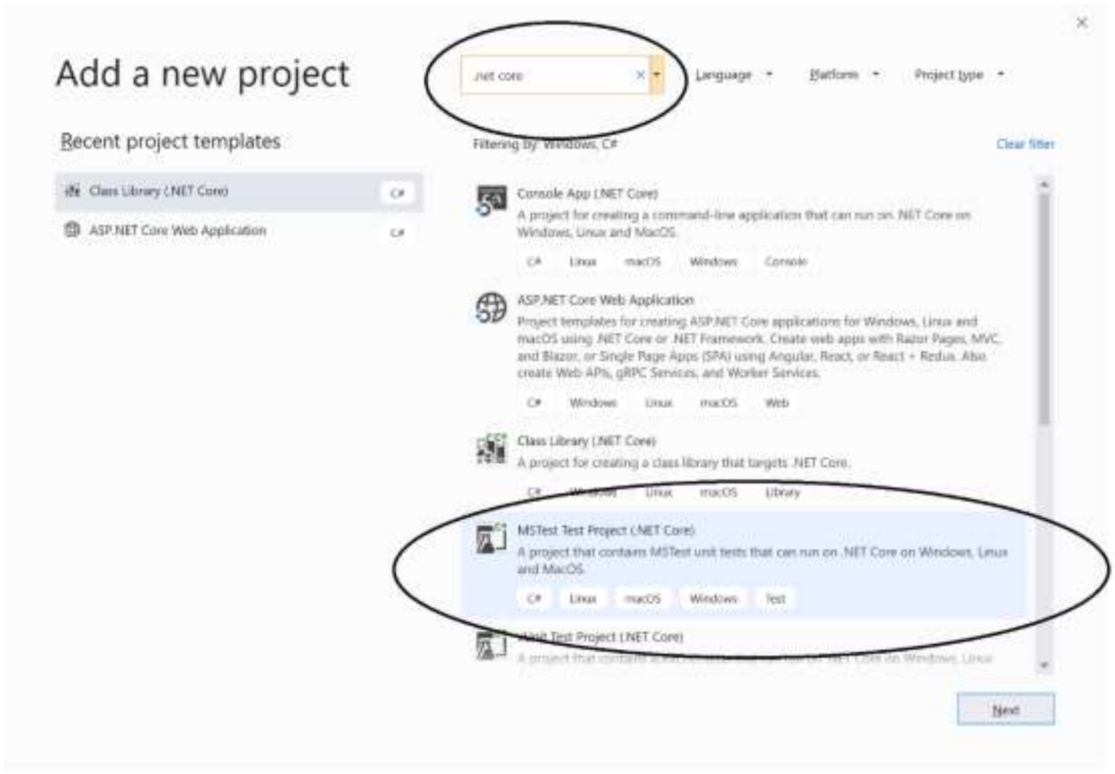
Allowed Origins

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

That was all for now creating the REST service – next step is to consume the rest service (very similar to what you did at 2 semester)

Visual Studio 2019 version

Create Test project:
Remember only **.net core**
Choose MSTest


Add a new project

Recent project templates

Class Library (.NET Core) C#

ASP.NET Core Web Application C#

.net core

Language Platform Project type

Filtering by: Windows, C#

Clear filter

Console App (.NET Core)
A project for creating a command-line application that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Console

ASP.NET Core Web Application
Project templates for creating ASP.NET Core applications for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, and Blazor, or Single Page Apps (SPA) using Angular, React, or React + Redux. Also create Web APIs, gRPC Services, and Worker Services.
C# Windows Linux macOS Web

Class Library (.NET Core)
A project for creating a class library that targets .NET Core.
C# Windows Linux macOS Library

MSTest Test Project (.NET Core)
A project that contains MSTest unit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Test

xUnit Test Project (.NET Core)
A project that contains xUnit tests that can run on .NET Core on Windows, Linux

Next