| | |
|---|---|
| **COMPUTING SUBJECT:** | Restful ASP.Net Core-services |
| **TYPE:** | Assignment |
| **IDENTIFICATION:** | RestService#1 |
| **COPYRIGHT:** | *Peter Levinsky & Michael Claudius* |
| **LEVEL:** | Medium |
| **TIME CONSUMPTION:** | 1½-2 hours |
| **EXTENT:** | 120 lines |
| **OBJECTIVE:** | Restful services based on ASP.Net Core |
| **PRECONDITIONS:** | Rest service theory. Http-concepts<br>Computer Networks Ch. 2.2 |

**COMMANDS:**

**IDENTIFICATION:** RestService#1 / 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 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 to be more precise javascript) i.e. support CORS.

The whole group of assignments consist of six steps:

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

Purpose

The purpose of this assignment is to be able to provide a simple Restful ASP.Net Web Services with CRUD.

Mission

You are to make a restful web services based on the ASP.Net services by setting up a service (provider). The service supports simple CRUD by implementing the classic GET, POST, PUT and DELETE requests. This we shall do in two major steps:

1. Investigate a simple default REST Service
    a. Create the REST Service
    b. Run the REST Service
    c. Try the REST Service (Postman or Fiddler)
    d. Explore the REST Service

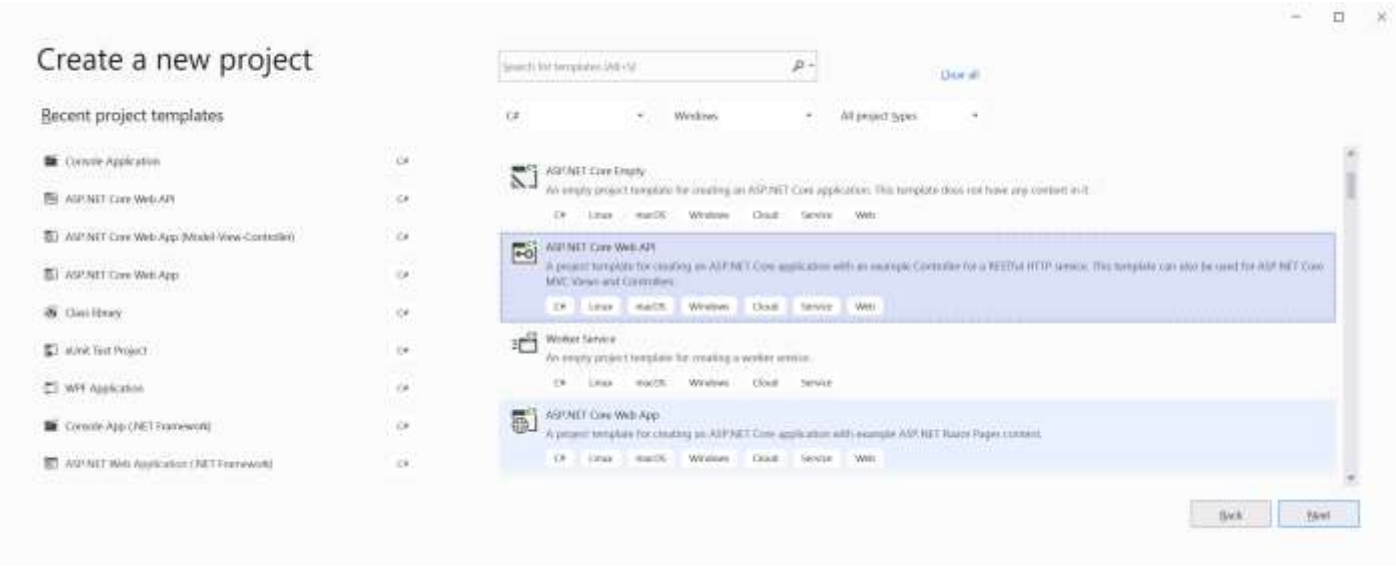
2. Create your own REST simple Service
    a. Create a model class
    b. Create a Manger
    c. Create a controller
    d. Run and try your REST Service (Postman or Fiddler)

Domain description

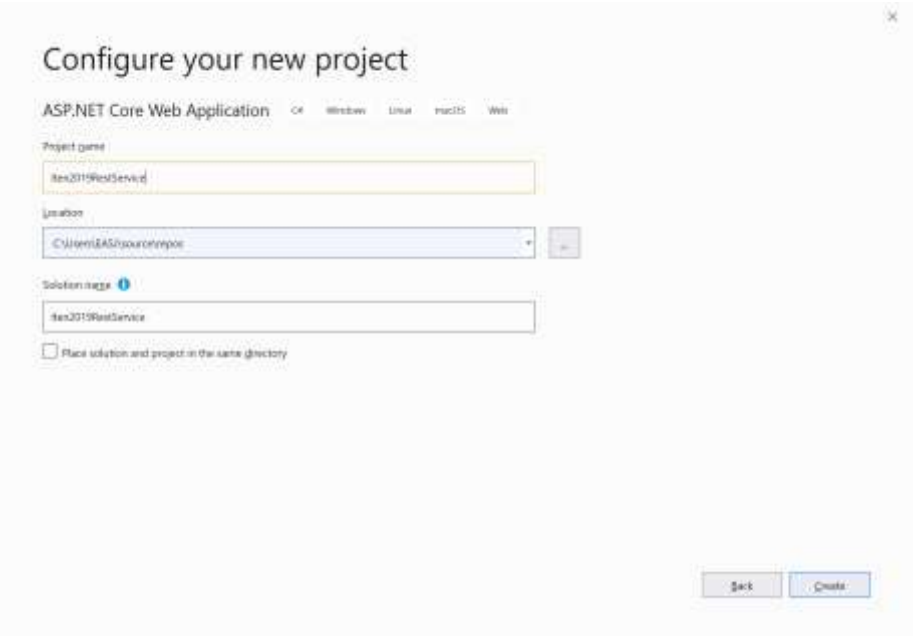
First, you shall just utilize the simple auto generated web service defined by WeatherForecastContolller. Next you are to create your own REST Service (Provider), which can manipulate (make CRUD) on model elements ‘Item’s.

# Assignment 1.a: Create The RESTful ASP.Net Core-service provider

You are to make a Rest Service provider '**RestItemService**'.
Start Visual Studio:File -> New -> Project.
Choose: Web -> ASP.NET Core Web API
Browse to a convenient location and give the name '**RestItemService**'.




Click OK.
Give the project and solution a name (here the same)



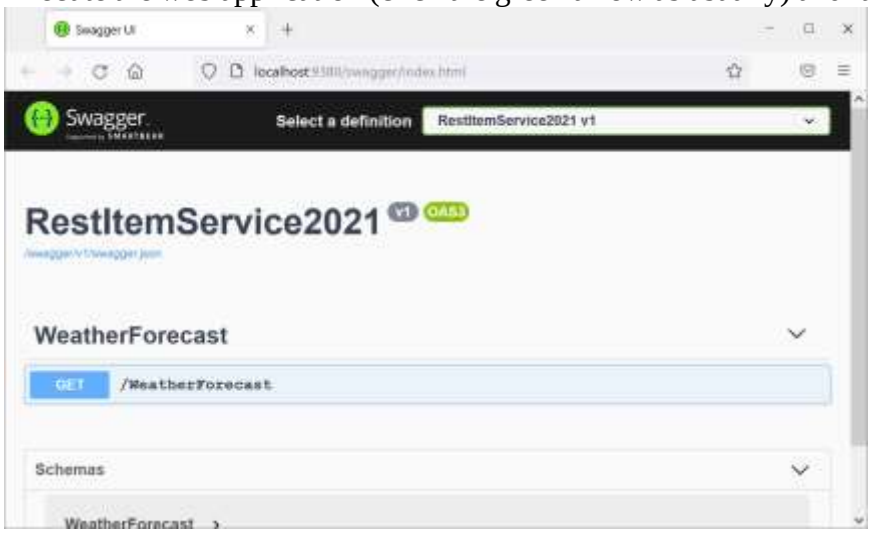
Configure type of web application


Untick checkbox for HTTPS

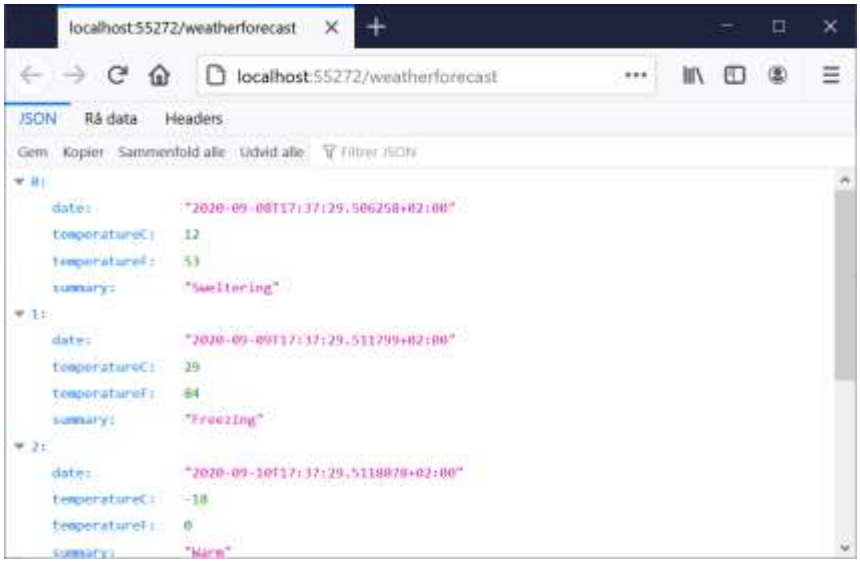
Now you have to wait a while…
Then we are ready to investigate the created project.

# Assignment 1.b:  Run the REST Service

Execute the web application (click the green arrow as usually) and it will open a local Browser.




Try with the URL http://localhost:55272/ WeatherForecast




Try to give the url:

http://localhost:55272/weatherforecast/2

and

http://localhost:55272/weatherforecast/7

What do you get? **(*Remember to use your own port number*)**

## Assignment 1.c: Fiddler/Postman

Have the Postman or Fiddler installed – see tools.htm

Try to invoke the methods from Fiddler/Postman using the same url's as before. If you are not familiar with fiddler (or postman) for details see 2.c below.

## Assignment 1.d: Exploring the REST Service i.e. the whole set up

Some questions arise:

Why does the project start with '…/swagger/index.html'?
Where are the Http-methods defined?
Why is it port 55272 (or more correct your port-number)?
How can it be executed in console mode on a different port?

To find the answers look in Solution Explorer:

Controller -> WeatherForecastController
Propeties -> LaunchSettings.json

Insert following code for the Get method body

```
[HttpGet("{id}")]
public int Get(int id)
{ return id; }
```

Execute again (with http://localhost:55272/weatherforecast/2 ). Any difference? What happens?

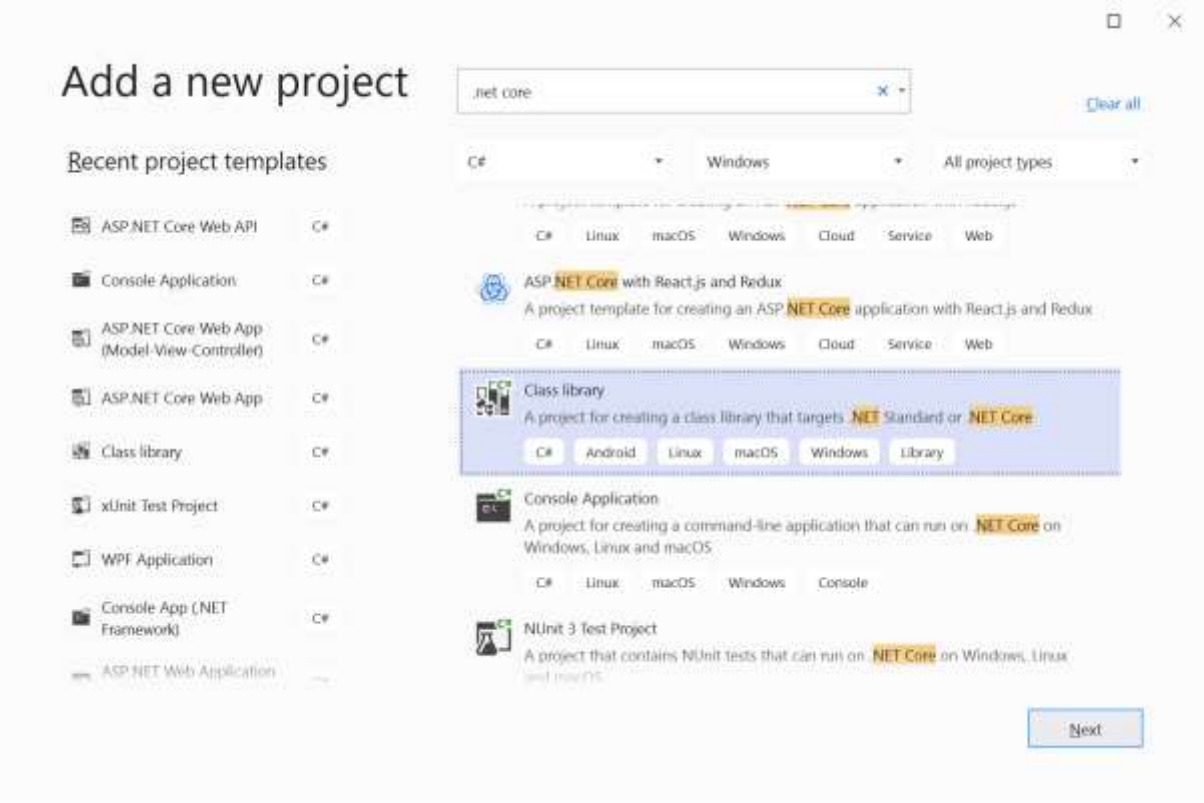
Finally, at the 'green arrow' change 'IISExpress' to 'RestItemService' and run the application again.
What could be the purpose of doing this?

*This ends the investigation of the auto generated service.*
*In the following assignments you shall create your own service.*

## Assignment 2.a: Create a model class Item

We need a model class '**Item**' before you can make your own REST service.

Therefore to the Solution (right click -> add new project) create a new project (Class Library) and named it '**ModelLib**'. Like this:




In this new project create a folder '**model**' (right click project, choose Add -> Folder) and in this folder add a class, "**Item**", with the properties:

- int Id;
- string Name;
- string ItemQuality;
- double AmountQuantity;

Remember to two constructors:

- Data() { … }
  //empty constructor needed for JSON transfer. Serializable objects.

- Item(int id, string name, string itemQuality, double quantity) { … }
  Intializing all the data fields

And make a ToString – method

Before you are finished, first make the class public and second **remember to compile the library i.e. build the project**. (If you are nice, delete the 'Class1' class in the library)

*Now you are ready to the next step.*

## Assignment 2.b: Creating a manager

a. In the solution (Solution Exploire); Create a folder e.g. named 'Managers'.

b. In this folder create an interface '**IManageItems**' with 5 methods:

```
IEnumerable<Item> Get();
Item Get(int id);
bool Create(Item value);
bool Update(int id, Item value);
Item Delete(int id);
```

To let your Interface referring to your model class (Item) from 2.a, you have to add a reference to refer to your library i.e. right-click at the Dependencies -> add reference choose from your solution 'ModelLib'.

c. Continue in this folder to create a class '**ManageItems**' which implements the interface **IManageItems**.

i) In this simple Manager, you are **not** using a database, but instead a **static list** of Item's as an instance field.

```
private static  List<Item> items = new List<Item>()
```

You can fill in some default values e.g. like:

```
private static readonly List<Item> items = new
List<Item>()
   {
       new Item(1,"Bread","Low",33),
       new Item(2,"Bread","Middle",21),
       new Item(3,"Beer","low",70.5),
       new Item(4,"Soda","High",21.4),
       new Item(5,"Milk","Low",55.8)
   };
```

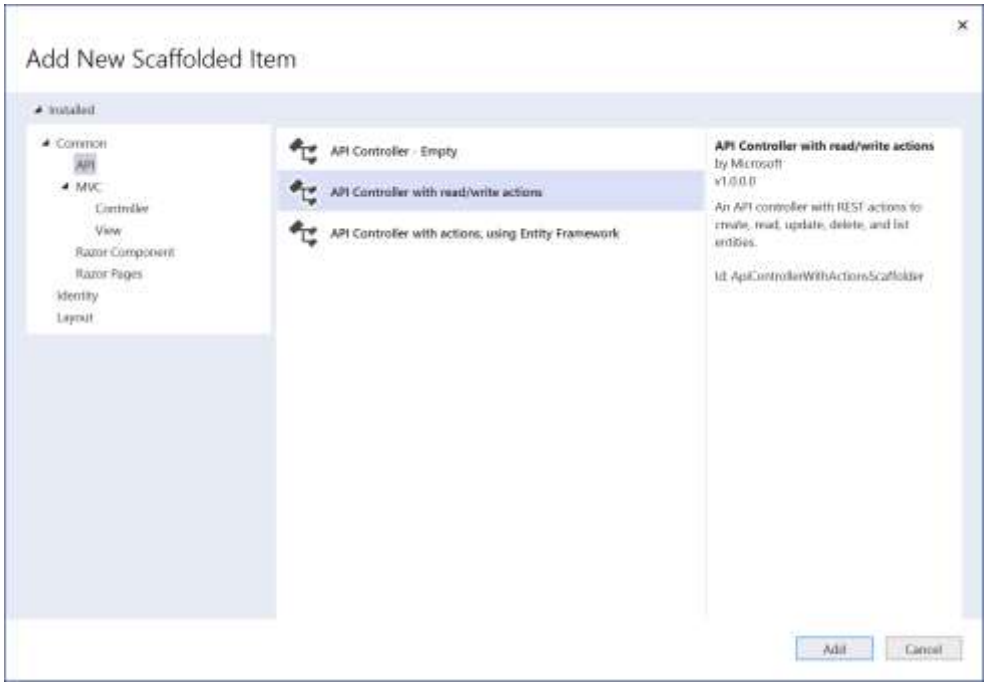
ii) Now you can modify the body of your five manager methods using this list.

```
public IEnumerable<Item> Get(){
   return new List<Item>(items);}

public Item Get(int id) {
  return items.Find(i => i.Id == id); }
```

```
public bool Create(Item value) {
  items.Add(value);
  return true;
 }

public bool Update(int id, Item value) {
    Item item = Get(id);
    if (item != null)
    {
         item.Id = value.Id;
         item.Name= value.Name;
         item.ItemQuality = value.ItemQuality;
         item.AmountQuantity = value.AmountQuantity;

         return true;
    }
    return false;
}

public Item Delete(int id) {
   Item item = Get(id);
   items.Remove(item);
   return item;
}
```

## Assignment 2.c: Creating a controller (making the REST API operation)

In the solution in the controller folder, add (i.e. Right click) a new controller named '**ItemsController**'. Choose 'API Controller **with read/write actions**'.



Now you have a template controller (the url is: http://localhost:44343/api/Items with your port number).

Modify the five signatures in the '**ItemsController**' to: (Modification marked with bold)

- public IEnumerable<**Item**> Get()
- public **Item** Get(int id)
- public **bool** Post([FromBody] **Item** value)
- public **bool** Put(int id, [FromBody] **Item** value)
- public **Item** Delete(int id)

In this simple CRUD REST Service you going to use the manager from 2.b. Meaning make an instance field of the manager like:

```
private readonly IManageItems mgr = new ManageItems();
```

then in each controller method make use of the manager e.g. like:

```
public IEnumerable<Item> Get(){
   return mgr.Get();
}
```

Now you can compile (build) and run your REST Service.

Try from your bowser

http://localhost:44343/api/Items
or
http://localhost:44343/api/Items/3

Can you call POST from your browser without using swagger?
What about PUT or DELETE?

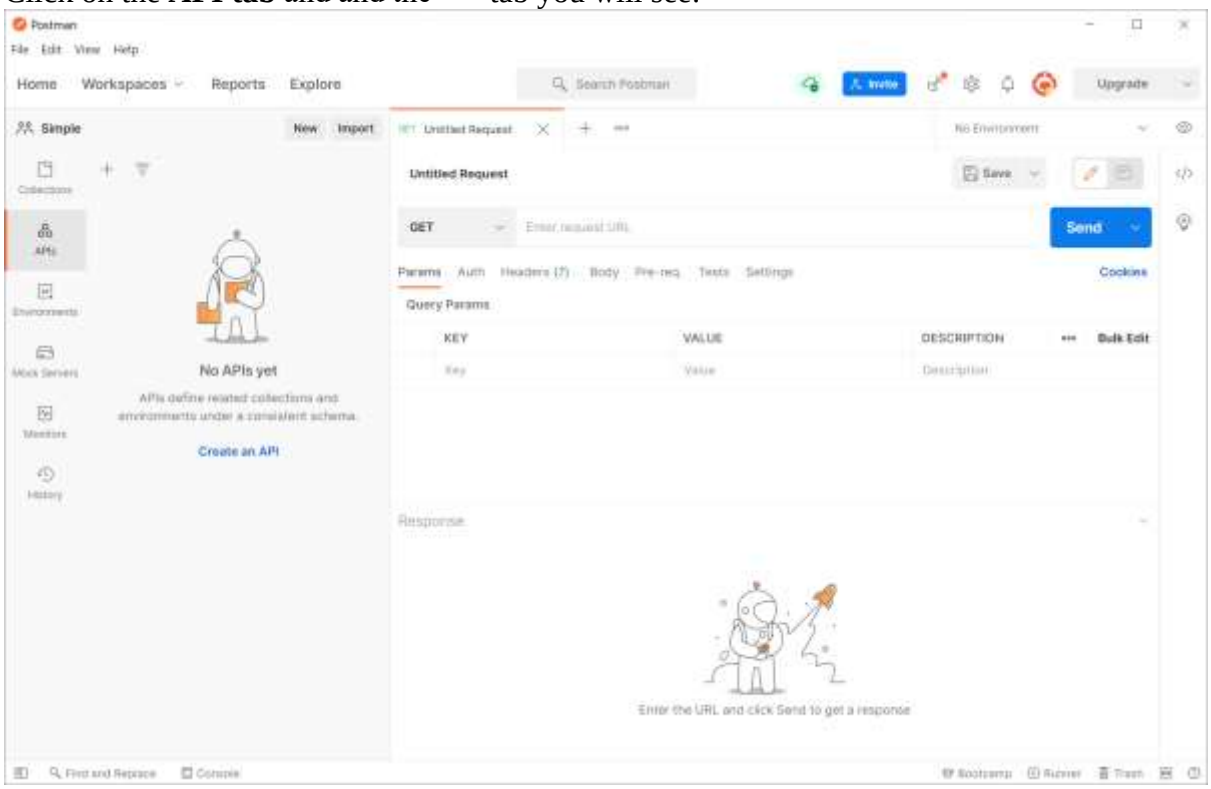
Try out the POST,PUT, DELETE with the swagger interface.

# Assignment 2.c:  Run and try your REST Service (Postman or Fiddler)

Have Postman or Fiddler installed. (See previous tool exercises).
To be able to try all methods in your REST Service you need tools like Postman, Fiddler  or by the build in swagger API help page.

Open e.g. Postman.

Click on the **API tab** and and the '+'-tab you will see:




Write the URL to your REST Service e.g. http://localhost:44343/api/Items click send. To see the result below.

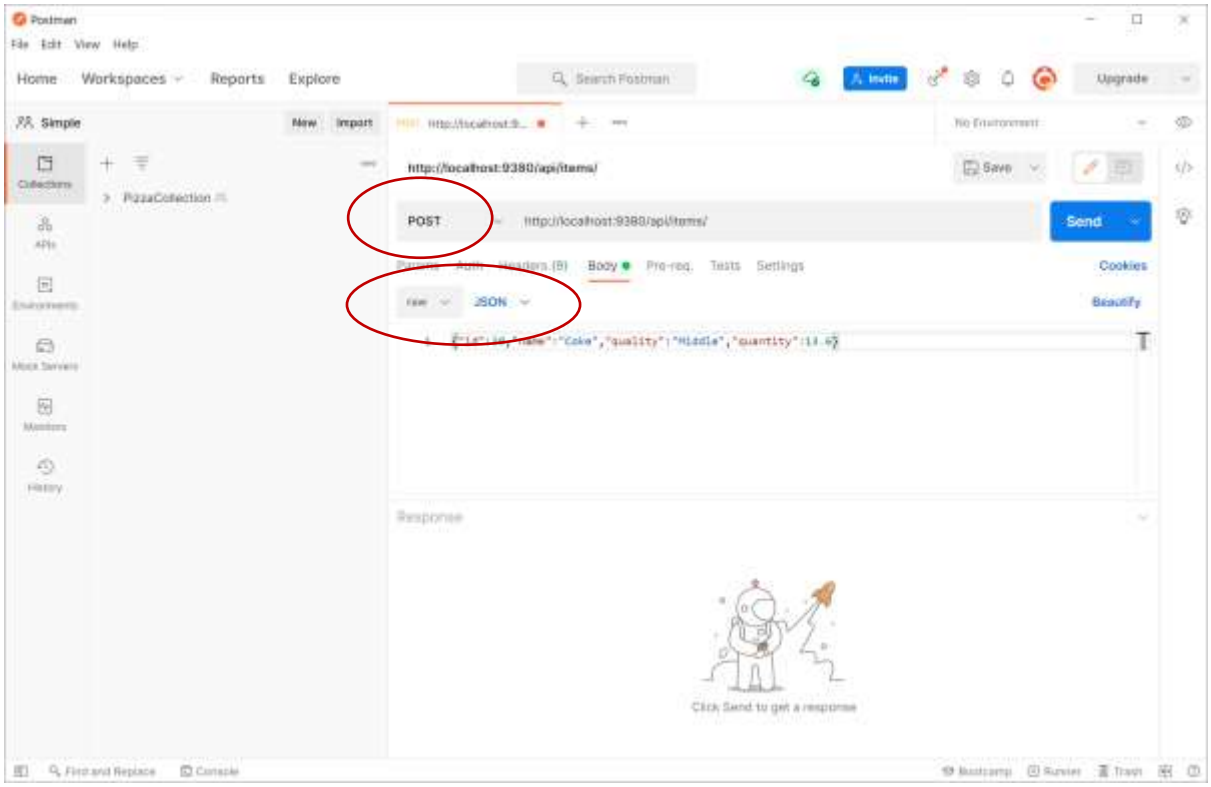
Now try with http://localhost:44343/api/Items/3 again look at the result.

All this is performing the GET method; next step is to try the other methods:

<u>POST:</u>

- First pick the POST method
- Request body must hold the data as a raw + changes text to Json-string e.g. {"id":30,"name":"Coke"," ItemQuality":"Middle"," AmountQuantity ":13.6}

It should look like:




- Send the REST Call
- Lookup the result – what do you see?
- Try to get …./Items/30 – what do you get?

<u>PUT:</u>

- First pick the PUT method
- Change the URL to …../api/Items/30
- Request body must hold the data as a Json-string (likewise post) e.g. {"id":30,"name":"**Coca** Coke"," ItemQuality":"Middle"," AmountQuantity":13.6}

I.e. change the name

- Send the REST call
- Try to get …./Items/30 – what do you get?

<u>DELETE:</u>

- First pick the DELETE method
- Change the URL to …../api/Items/30
- Send the REST call
- Try to get …./Items/30 – what do you get?

*That was all – You have nicely finished a simple crud REST-Service and you are ready to do more advanced REST services.*