COMPUTING SUBJECT:	Restful ASP.Net Core-services
TYPE:	Assignment
IDENTIFICATION:	RestService#5
COPYRIGHT:	*Peter Levinsky & Michael Claudius*
LEVEL:	Medium
TIME CONSUMPTION:	1-1½ hours
EXTENT:	100-130 lines
OBJECTIVE:	Consuming the Restful services
PRECONDITIONS:	Rest service theory. Http-concepts Computer Networks Ch. 2.2
COMMANDS:	

IDENTIFICATION: RestService#5 / 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. (this assignment)**
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#5

Purpose
The purpose of this assignment is to consume the newly published REST service in azure.

Mission
You are to design and implement a C# console application to consume your REST Service. Remember your RestService are running on a web-server thereby you need to access the Service through the HTTP-protocol.

Assignment 1: C# console application

Set up the console application:

a. You can either create a new solution in Visual Studio or a new project in your already existing solution. But in either case create a **.Net Core console application** e.g. named ‘**ConsumeRest**’.

b. To be able to use the model class Item make a reference to your **ModelLib** from the first assignment RestService1
In the same workflow install the NuGet package ‘**Newtonsoft.Json’** to be used to serialise/deserialise json.

c. In the project make a ‘**Worker**’ class with a ‘**Start’** Method, make an instance of the Worker class in the Main method and call the ‘**Start’** method.

   Finish the Main method by adding *Console.ReadLine();* so you can see the result before the program close.

Do the work in the Console application:

d. First you are going to consume get all

   Make a method in the Worker class:

```
public async Task<IList<Item>> GetAllItemsAsync()
{
      using (HttpClient client = new HttpClient())
     {
          string content = await client.GetStringAsync(URI);
          IList<Item> cList =
              JsonConvert.DeserializeObject<IList<Item>>(content);
         return cList;
     }
 }
```

   Where URI is the path to your REST Service
   e.g. ’ *http://itemservice-pele-easj.azurewebsites.net/api/localItems*’

   Make a call in the start method of this method and print out the list of Items.

e. Create e new method to get one item with the signature:

```
public async Task<Item> GetOneItemsAsync(int id)
```

Print out one item in the start method.

f. What happen id you requested an id that do not exists?

g. To make the consumer act on the status code you need to call the more basic Get method on the HttpClient like:

```
using (HttpClient client = new HttpClient())
{
   HttpResponseMessage resp = await client.GetAsync(URI+id);

   String content = await resp.Content.ReadAsStringAsync();
   if (resp.IsSuccessStatusCode) // all 2xx status codes
   {
      Item cItem = JsonConvert.DeserializeObject<Item>(content);
      return cItem;
   }
   // else
   throw new KeyNotFoundException
                    ($"Status code={resp.StatusCode} Message={content}");
 }
```

And then you need to make try – catch on the Start method around the call of 'GetOneItemsAsync'

h. Create separated methods for PUT, POST and DELETE.

For PUT and POST you need to send a json-encoded string of an Item-object. Hint:

```
String jsonStr = JsonConvert.SerializeObject(newItem);
StringContent content =
     new StringContent(jsonStr, Encoding.UTF8, "application/json");
```

So now you can create a REST Service provider and a REST Service Consumer That's good.