

COMPUTING SUBJECT:	Restful ASP.Net Core-services
TYPE:	Assignment
IDENTIFICATION:	RestService#6
COPYRIGHT:	<i>Peter Levinsky & Michael Claudius</i>
LEVEL:	Medium
TIME CONSUMPTION:	1½-2 hours
EXTENT:	120 lines
OBJECTIVE:	Restful services using a Database
PRECONDITIONS:	Rest service theory. Http-concepts Computer Networks Ch. 2.2
COMMANDS:	

IDENTIFICATION: RestService#6 / 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. [Testing a REST Service.](#)
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 (this assignment)**

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)

Helpful link:

SQL references: <https://www.w3schools.com/sql/default.asp>

DBContext: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-3.1&tabs=visual-studio#add-a-database-context-2>

Note (Anders Børjesson): REST controller using Entity Framework:
https://docs.google.com/document/d/e/2PACX-1vTIIIdWBWVYpZF4W9MEtH6vbtR19VwtILi9n_866-IK_LR6vIVhMk0FDQpMhPZOus8zrJfkImMXJTbtX/pub

This Assignment: RestService#6

Purpose

The purpose of this assignment is to refactor your REST Service so it can use a Database for persistence instead of a static list.

Mission

You are to refactor your implementation of the controllers to use Database. You will only work with one simple table to hold data i.e. no foreign key and no talk of 3th Normal Form.

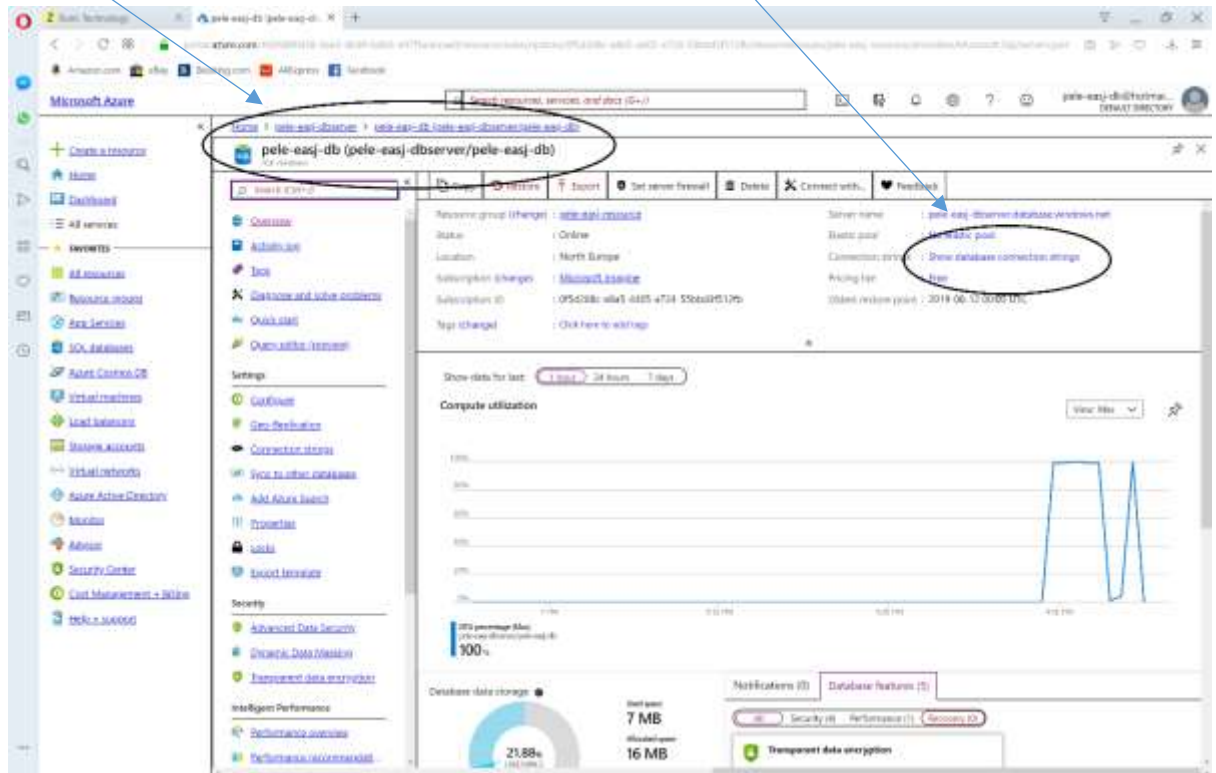
Assignment 1: Prepare Solution for Database persistency

- a. Open your REST-service project (more correctly solution). You properly have a reference to your Model Library (see assignment 1) otherwise create a reference to your Model Library
- b. *Alternative create a new Solution and add the model library reference to the project.*
- c. In Azure create a Table 'Item' with the properties:
 - int Id; // i.e. Id int not null primary key
 - string Name; // i.e. Name nvarchar(35) not null
 - string ItemQuality; // i.e. ItemQuality nvarchar(35) not null
 - double AmountQuantity; // i.e. Quantity float not null

You can make the Id generate automatically by using identity keyword

Assignment 2: Create a Utility Class for Database connections

- a. To implement these methods you need the connection string (get this string from database in Azure):



Show the connection strings and copy the one for ADO.Net, though you need to fill in your user name and password!!

Use a simple trick to prevent my connection string (including username + password) to be uploaded to a public GitHub repository.

Keep the connection string in a separate file (here called Secrets).

Mention the Secrets file in the .gitignore file in your projects.

Simply add a single line to the .gitignore file

Secrets.cs

Notice that the .gitignore file is not created until you add Git version control to your project. Now Git will ignore the Secrets file.

Make the class Secrets in the model folder.

```
public class MySecrets
{
    public const string ConnectionString =
        "Connection string including user ID + password.
        Can be obtained from portal.azure.com";
}
```

b. Install a Nuget Package to support SQL entity framework

Install the NuGet package '**Microsoft.EntityFrameworkCore.SqlServer**'

Create the connection between your REST and the Database.

In the model folder create a class '**ItemContext**', this class must extends the DbContext class like:

```
public class ItemContext : DbContext
{
    public CarContext(DbContextOptions<ItemContext> options) :
        base(options)
    {
    }

    public DbSet<Item> Items { get; set; }
}
```

If you have more tables in your database you can make more DbSet<T> objects in the Context class.

For further reading see: [Add to the TodoContext database context](#)

c. Register your Database Context in the Startup-file.

You are now to register your Database context in the startup -> ConfigureServices, just like you did at 2nd semester where you registered different singletons.

```
services.AddDbContext<ItemContext>(opt =>
    opt.UseSqlServer(MySecrets.ConnectionString));
```

- d. Create the '**ManageItemsDB**' class

In the folder (managers from exercise RESTservice#1) create a class '**ManageItemsDB**' which also implements the **IManageItem** interface.

Like:

```
private readonly ItemContext _context;

public ManageItemsDB(ItemContext context)
{
    _context = context;
}

public IEnumerable<Item> Get()
{
    return _context.Items.ToList();
}

public bool Create(Item value)
{
    try
    {
        // if automatic generated ID insert following line
        // value.Id = 0;
        _context.Items.Add(value);
        _context.SaveChanges();
        return true;
    }
    catch (Exception e)
    {
        return false;
    }
}

...
```

Assignment 3: Refactor the Controller class

- a. Refactor your **ItemsController** to use this **ManageItemsDB** class, by calling the appropriated methods.
- b. Support context injection by

```
public ItemsController(ItemContext context){
    mgr = new ManageItemsDB(context);
}
```

- a. Run your component unit test and your integration test
- b. If succeed publish the refactored REST service in Azure

*By now you are 'full flying' REST service implementer
and can do other REST services*