RAWDATA Assignment 4 – Web service and Data Layer

This assignment is in two parts. The first part is about creating a domain model and as data service. The data service is the layer that communicates with the database and provide an interface to the rest of the system. The data service also takes care of the transformation between the database model and the domain model. In the second part (Tuesday 20/3) a restful webservice is added in a layer on top of the data layer from part 1. Like the 3rd assignment test suites will be given to verify that your code has the necessary functionality, but the structure of your code and your design choices will play a bigger part in this assignment. We want clean code[1].

How and when to hand in

A 1-2 pages document defining the members of the group, the URL to a GitHub repository where the source code can be found, and a table (or a screen dump from your testing environment) showing the status of running all tests in the test suite attached to this assignment. Upload the document to Moodle "Assignment 4" no later than Marts 21 at 23.55.

Important

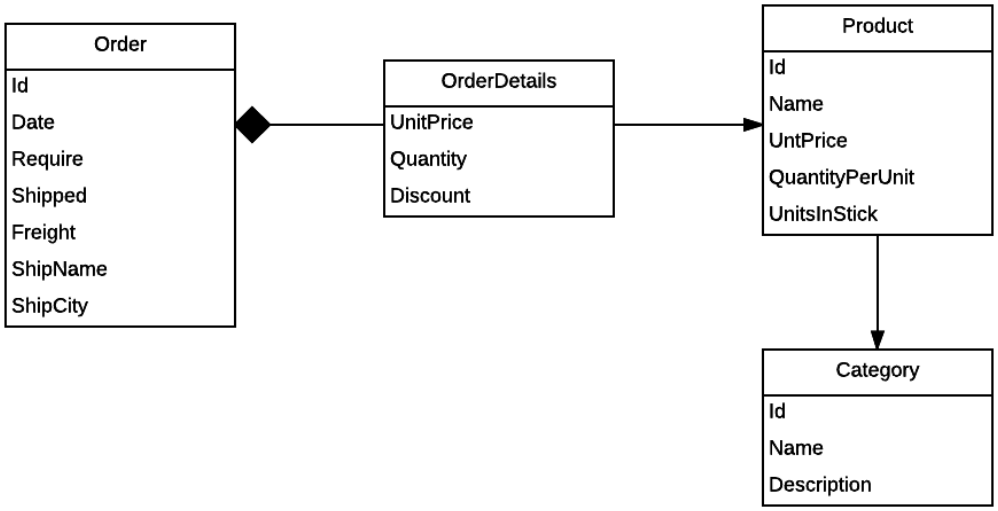
Hand in one submission from your group (from one of the members), but DO REMEMBER to write ALL NAMES of participants in your group in the top of the file.

Part I – The Data Layer

In this assignment the goal is to provide a restful webservice over a small testing database called Northwind, a database sample provided with earlier versions of Microsoft Sql Server. A diagram of the database can be found here https://northwinddatabase.codeplex.com/. The database used in this assignment can be downloaded from Moodle.

We will focus on the orders, orderdetails, products and categories tables from the Northwind database, and ignore all other tables. The data service, to be created in this part, uses the following domain model

[1] https://blog.goyello.com/2013/01/21/top-9-principles-clean-code/



when exposing data to the next layers. The mapping between the database model and the object-oriented model (the domain model) should be done by the Entity Framework Core (EF Core). EF Core is object-relational mapper that can help when moving data back and forth between the data service and the database, i.e. the object-oriented and relational models.

The following is a description of the requirements to the data service:

Order

1. Get a single order by ID
   Return the complete order, i.e. all attributes of the order, the complete list of order details. Each order detail should include the product which must include the category.
2. Get order by shipping name
   Return a list of orders with id, date, ship name and city.
3. List all orders
   Return a list of orders with the same information as in 2.

Order Details

4. Get the details for a specific order ID
   Return the order details with product name, unit price, quantity.
5. Get the details for a specific product ID
   Return the complete list of details, with order date, unit price, quantity

Product

6. Get a single product by ID
   Return the complete product with name, unit price and category name.
7. Get a list of products that contains a substring
   Search for products where the name matches the given substring. Return a list of product name and category name.
8. Get products by category ID

Return the list of products with the given category. Return the same information as in 6.

Category

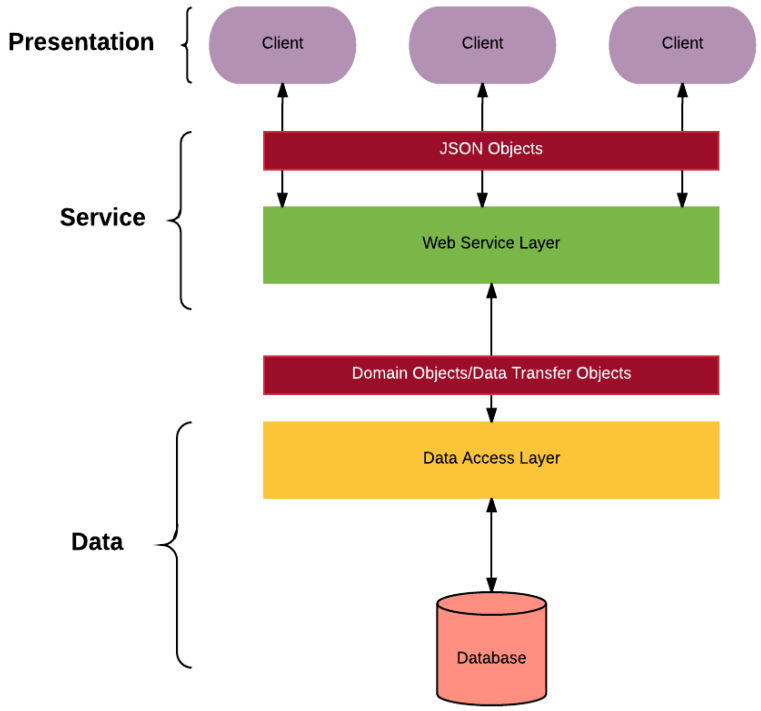
9. Get Category by ID
   Return the category if found otherwise return null.
10. Get all categories
    Return the list of categories with id, name and description.
11. Add new category
    Add a new category to the system. The method takes name and description as arguments. The system must provide a new ID and return the newly created category.
12. Update category
    Take as argument id, name and description and update name and description. If the category is found, update the category and return true, otherwise return false.
13. Delete category
    Take id as argument. Return true if the category is deleted, otherwise return false.

The data service should be implemented as a library that can be referenced by other projects. You can see example on this in the solution with the test suite where a dummy data service project is included. You find the test suite for the first part here:

https://github.com/bulskov/RAWDATA2018F_Assignment4TestSuite

Part II – The Web Service Layer

In this part we want to add a web service on top of the Data Layer so we can expose selected parts of our data to clients. Below is a figure showing the layers we want to implement. The output to clients is by JSON objects, and we need to decouple the interface provided to clients from the internal representation, both in our service and in the underlying database.



The data layer created in the first part does this by mapping the data model into an object-oriented domain model. The interface provided to the clients, the outer facing contract, must be intuitive, consistent and stable, to be useful. To get these properties decupling from the internal model is important. Furthermore, we do not want to reveal internal details, or returning information which the user did not request. Specific objects designed to hold the requested information is often used, the so-called Data Transfer Objects (DTO)[2]. In the first part of this assignment we created a data service, but for some methods, the domain model actually had too much information, and DTOs should be introduced to better fit the requested output.

We want to expose requirements 6-13 from the data service (i.e. methods on products and categories). We want to introduce DTOs, such that the three methods on products, return list of objects that match the output. You need to make these changes to your data service.

Provide two paths, `/api/products` and `/api/categories` and the following interface:

[2] https://en.wikipedia.org/wiki/Data_transfer_object

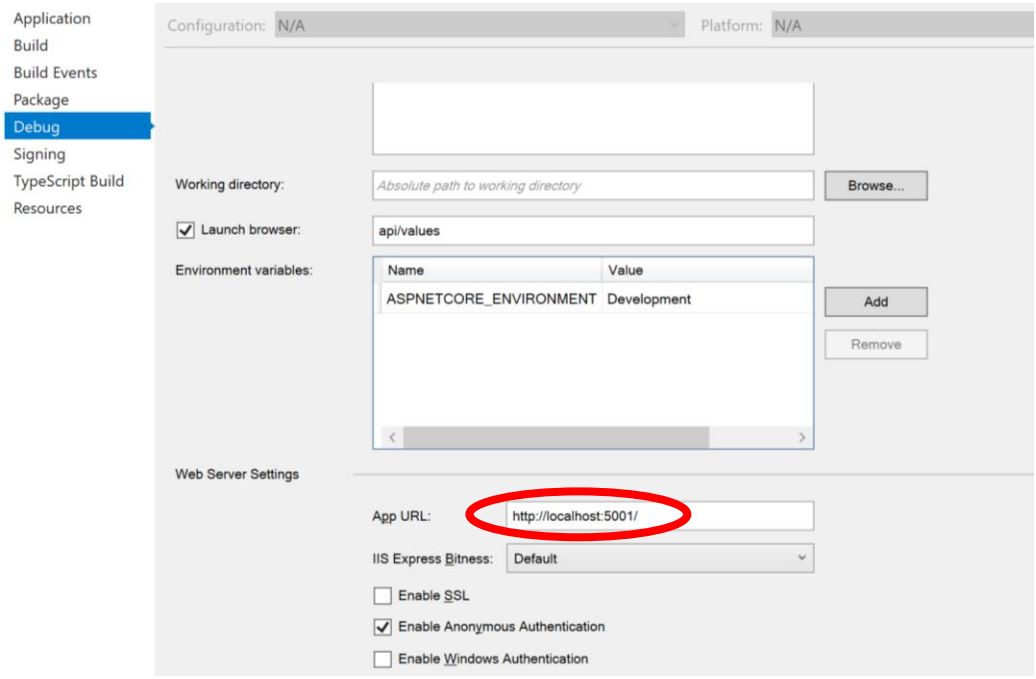
Path	HTTP	Description	Status	
			Valid	Invalid
/api/products/<id>	GET	Single product	OK	Not Found
/api/products/category/<id>	GET	List of products	OK	Not Found
/api/products/name/<substring>	GET	List of products	OK	Not Found
/api/categories	GET	List of categories	OK	
/api/categories/<id>	GET	Single category	OK	Not Found
/api/categories	POST	Create category	Created	
/api/categories/<id>	PUT	Update category	Ok	Not Found
/api/categories/<id>	DELETE	Delete category	Ok	Not Found

Use the MVC pattern and create one controller to each path.

Your web service must run on port 5001. If you start your service from the command line the default port is 5000. To change this, add the `UseUrl` to `BuildWebHost` in your `progam.cs` file, as shown below.

```
public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseUrls("http://localhost:5001/")
        .Build();
```

If you start your service from within Visual Studio, you must also set the App URL in the preferences of you Web Service project. Right click the project and select "preferences", select the "Debug" tab. In the bottom of the settings (you may need to scroll down) you find the "App URL". Change it like shown in the figure below.



NB: You need both settings, as the application will be started differently when debugging or not.

The test project for part II can be found here:

https://github.com/bulskov/RAWDATA2018F_Assignment4Part2TestSuite