CIT Assignment 4 – Web service and Data Layer

This assignment consists of two components. The first part involves the creation of a domain model and a data service. The data service serves as the intermediary layer responsible for communicating with the database, offering an interface to the rest of the system, and handling the transformation between the database model and the domain model. In the second part, scheduled for Friday, October 22th, we will add a RESTful web service layer on top of the data layer developed in the first part. Like the third assignment, you will receive test suites to ensure your code meets the required functionality. However, this assignment places a greater emphasis on the structure of your code and your design choices. We aim for clean and well-structured code[1].

Instructions for Submission

Please submit the following items:

1. A concise 1–2-page document containing the following information:
    - List of group members.
    - The URL to your GitHub repository (or a similar platform) where the source code is accessible.
    - A table or screenshot (from your testing environment) displaying the status of the tests in the two test suites for this assignment.
2. Upload the document to the Moodle platform under "Assignment 4" by no later than October 22 at 23:59.

Important

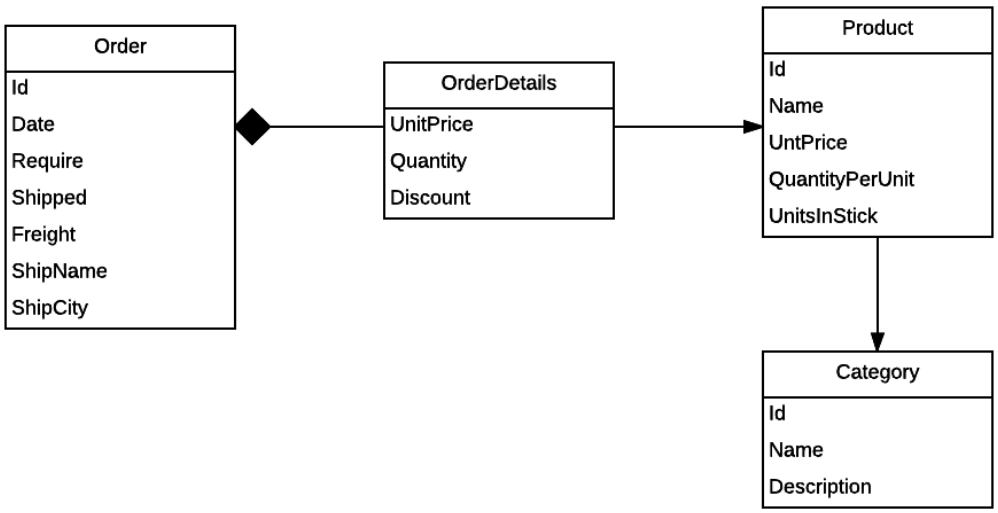
Submit a single group submission but ensure that the names of all participants in your group are clearly listed at the top of the file.

Part I – The Data Layer

In this assignment, the overall objective is to create a RESTful web service that interfaces with a small testing database called "Northwind". This database is a sample provided with earlier versions of Microsoft SQL Server, and you can find a diagram of the database here: https://northwinddatabase.codeplex.com/. The database for this assignment can be downloaded from Moodle.

For this assignment, our focus will be on the following tables from the Northwind database: "orders," "orderdetails," "products," and "categories." We will disregard all other tables. The data service, which will be developed in the first part, will expose the following domain model to the subsequent layers.

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



The mapping between the database model and the object-oriented model (the provided domain model) will be handled using Entity Framework Core (EF Core), an object-relational mapper that facilitates data movement between the data service and the database, bridging the gap between object-oriented and relational models.

The following are the requirements for the data service:

Order

1. Get a single order by ID: Retrieve the complete order, including all its attributes and the entire list of order details. Each order detail should include the product, which must also include the category.
2. Get orders by shipping name: Return a list of orders with their ID, date, shipping name, and city.
3. List all orders: Provide a list of orders with the same information as in requirement 2.

Order Details

4. Get details for a specific order ID: Retrieve the order details, including the product.
5. Get details for a specific product ID: Obtain the complete list of details, including the product and the order. Order the details by Order Id.

Product

6. Get a single product by ID: Return Id, Name, UnitPrice, QuantityPerUnit and UnitsInStock from product plus the name of the category.
7. Get a list of products containing a substring: Search for products with names matching the given substring and return a list containing only the product names and category names. [Hint: This requires a new object with only these attributes]
8. Get products by category ID: Retrieve a list of products belonging to the given category, including information as described in requirement 6.

Category

9. Get category by ID: Return the category if found; otherwise, return null.
10. Get all categories: Provide a list of categories, including their ID, name, and description.
11. Add a new category: Add a new category to the system, accepting name and description as arguments. The system should generate a new ID and return the newly created category.
12. Update category: Accept ID, name, and description as arguments and update the category with the provided information. If the category is found, update it, and return true; otherwise, return false.
13. Delete category: Accept the category's ID as an argument. Return true if the category is successfully deleted; otherwise, return false.

Important Note: It might be essential to create multiple distinct objects (classes) to facilitate the retrieval of the necessary information. These objects are commonly known as Data Transfer Objects (DTOs) [2] and form a crucial part of the abstractions required to reduce dependencies between the various layers.

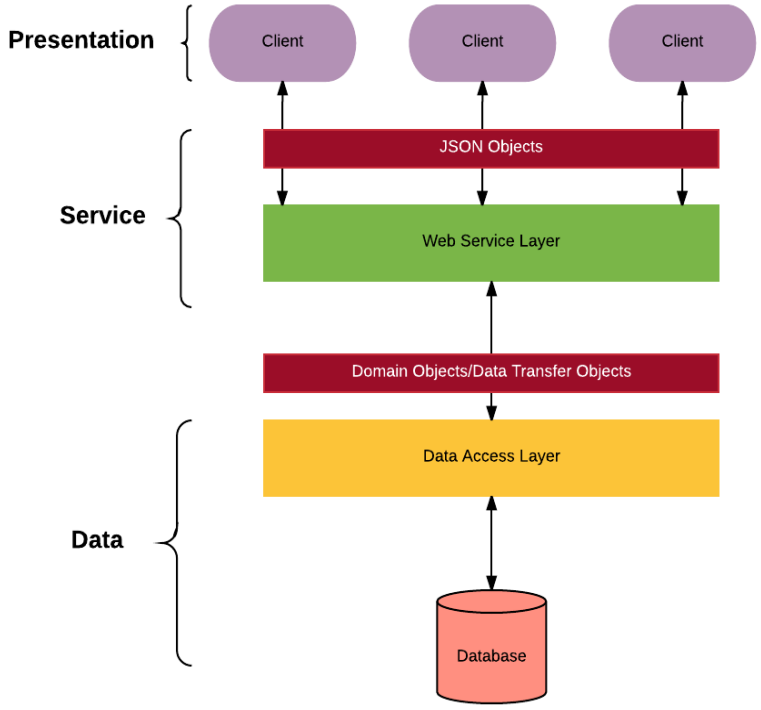
The data service should be implemented as a library that can be referenced by other projects. You can refer to an example in the solution with the test suite, where a dummy data service project is included. You can find the test suite for the first part here:

https://github.com/bulskov/CIT_2024_Assignment4PartI

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

Part II – The Web Service Layer

In this part, our objective is to introduce a web service that builds upon the DataLayer, allowing us to make specific portions of our data accessible to clients. The diagram below illustrates the layers we aim to establish. The data delivered to clients will be in the form of JSON objects. To achieve this, we must separate the interface presented to clients from the internal representation, ensuring this decoupling exists both within our service and within the underlying database.



The DataLayer, established in the first part, accomplates this by converting the database model into an object-oriented domain model. To ensure usability, the interface presented to front end clients, often referred to as the "outer-facing contract", must be user-friendly, consistent, and maintaining stability over time. It is crucial to separate this outer-facing contract from the internal model. Moreover, we must avoid exposing internal intricacies or delivering information that the user did not request. To achieve this, we commonly employ specialized objects known as Data Transfer Objects (DTOs).

In the first part of this assignment, a data service was created, but it became apparent that for certain methods, the domain model contained excessive information. Therefore, the introduction of DTOs is necessary to better align with the desired output.

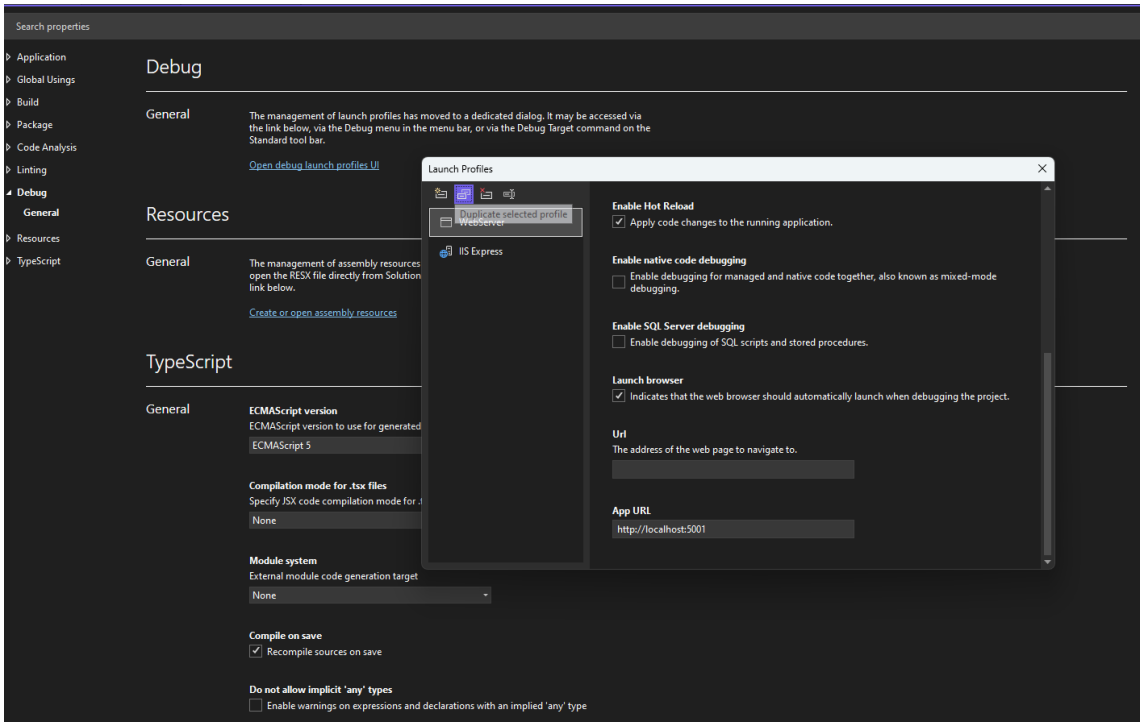
For this part, our goal is to expose requirements 6-13 from the data service, which pertain to methods involving products and categories. We intend to implement DTOs in such a way that the three methods related to products return a list of objects that precisely match the desired output. To accomplish this, adjustments of your data service may be necessary.

Create two endpoints, `/api/products` and `/api/categories,` along with the following interface:

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

Implement the MVC (Model-View-Controller) pattern and establish one controller for each path.

Ensure that your web service operates on port 5001. By default, when launching your service from the command line, it uses port 5000. To modify this setting, navigate to the properties of your web service project, and select "Debug." In the "General" section select "Open debug launch profiles UI". Choose "Webserver," and set the "App URL" at the bottom to http://localhost:5001.



The test project for part II can be found here:

Henrik Bulskov

https://github.com/bulskov/CIT_2024_Assignment4PartII

Henrik Bulskov