A picture containing text, book, statue, art

Description automatically generated

CONTENTS

**1 Problems with** **repository pattern**

[1. Fat](file:///E:\My\Drive\CQRS\My-CQRS.docx#_bookmark0) repository

[2. Performance](file:///E:\My\Drive\CQRS\My-CQRS.docx#_bookmark1) issues

[3. System](file:///E:\My\Drive\CQRS\My-CQRS.docx#_bookmark2) with different users and different roles   
on the same bounded context

**2 Advantages of CQRS pattern**

1. Single of responsibility

2. Separation of concern

[3. Independent](file:///E:\My\Drive\CQRS\My-CQRS.docx#_bookmark7) Scalability

4. Efficient Performance for database read operations.

5. Efficient with system that has different users and different roles on the same bounded context.

CONTENTS

**3 Types of CQRS pattern**

1. Single database

1.1 Single Database Single Project

1.2 Single Database Two Projects

2. Multiple databases

2.1 Same databases

2.2 different databases

**4 Implementation**

1. Single Database Single Project

2. Single Database Two Projects

3. Multiple Same Databases

3. Multiple Different databases

**Chapter 1**

**Problems with repository pattern**

**A picture containing drawing, illustration, art, cartoon

Description automatically generated**

**Repository pattern is one of the most popular pattern that used today to work with the database , there are also some other patterns to work with the database like CQRS pattern.**

**every pattern has its own advantages and disadvantages or in more accurate words every pattern is designed to solve specific problems or used with specific business or technical cases.**

**the architecture and development teams must be aware about this cases , when - why - how to use this pattern ? and also the impact for using this pattern for long term.**

**Overview of the project architecture**

The project follows the clean architecture and domain driven design principles , we will not focus on the clean architecture or domain driven design principles , but focus on the repository pattern and CQRS pattern from the view of clean architecture and domain driven design.

A screenshot of a computer

Description automatically generated

The aggregations are live on the Domain.Core

A screenshot of a computer

Description automatically generated

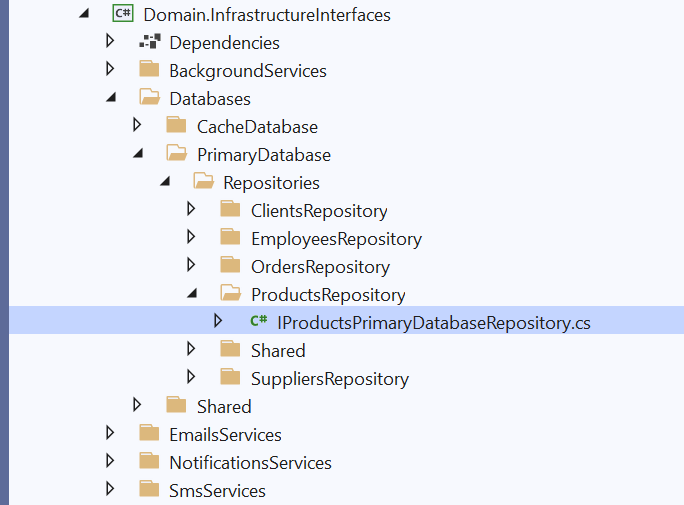
Let's take the product aggregation as example :

A screenshot of a computer

Description automatically generated with medium confidence

Product object is the aggregation root which control and manage the whole aggregation and other objects may be entities , value objects , Enums.

The product repository interface is live on :



And here the operations of products repository interface :

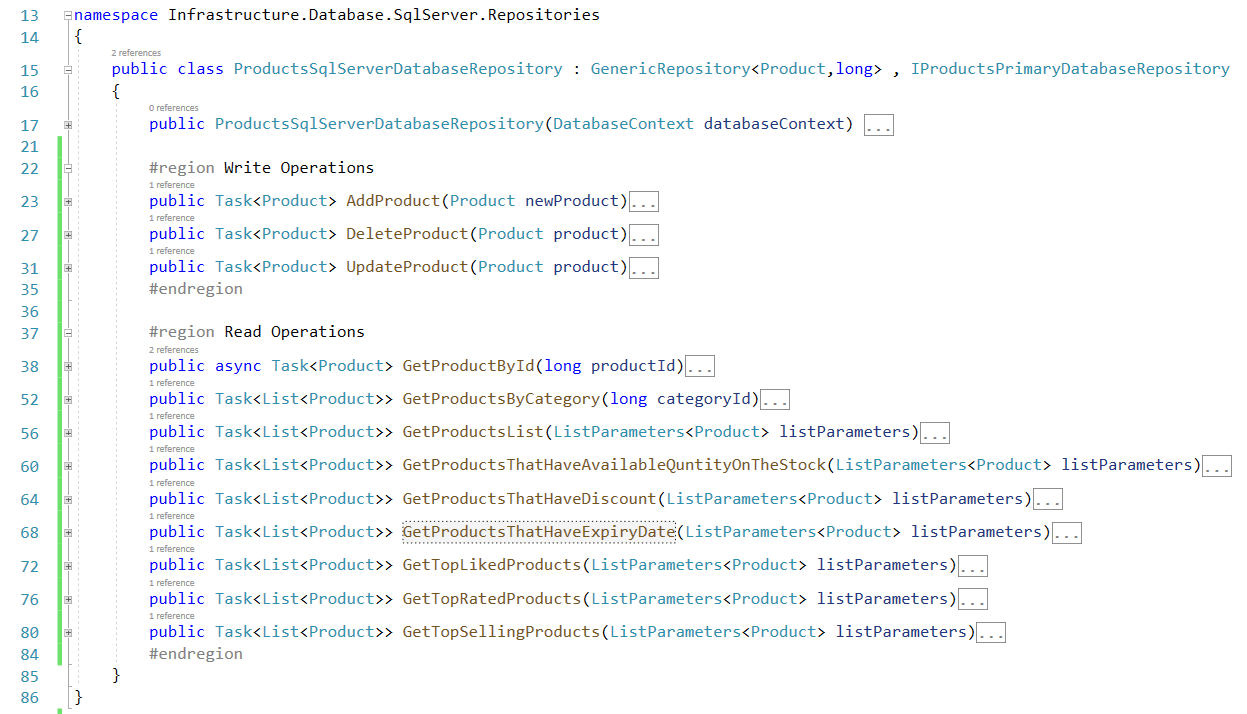
A screenshot of a computer code

Description automatically generated with low confidence

The actual implementation of the repository pattern is live on

A screenshot of a computer

Description automatically generated



**Repository pattern principles**

From Domain driven design, Repository pattern must apply two main principles:



**First principle**

**Repository pattern should be accessed via the aggregation root only.**

1. This mean repository must always **return or save** one aggregation root or list of aggregation roots ( **Product or list of Products** )
2. Repository pattern can not return normal entities directly in the aggregation like ( ProductAttribute ) , value objects like ( ProductTax ) , enums like ( ProductFileTypeEnum ).
3. Repository pattern can not return dto directly like

( ProductNameAndPriceDto )

A screenshot of a computer

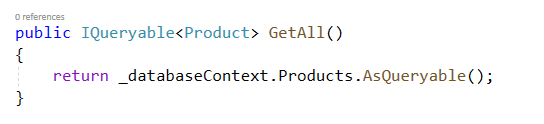
Description automatically generated with low confidence

1. Repository pattern should not return IQueryable , This is one of the most famous mistakes when using Repository pattern , because this break the main goal of repository pattern : abstract and isolate the logic or the implementation of data access layer outside the domain layer , here are example :

To get the products list which of golden category the products must satisfy the acceptance criteria :

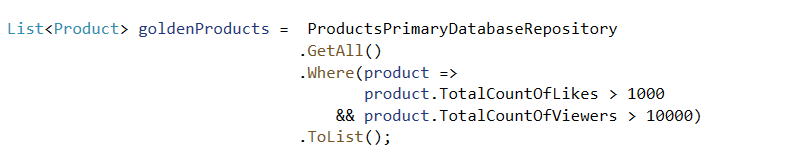
* Total number of likes > 1000
* Total number of viewers > 10,000

ProductsRepository.cs



You need to get golden products in two different services

Application Service 1



Application Service 2

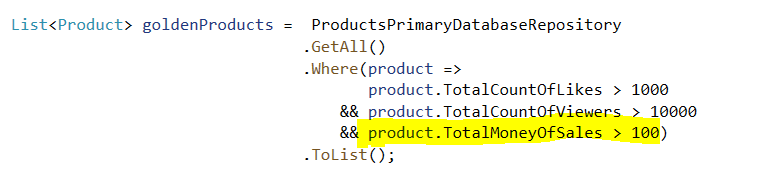
A screen shot of a computer code

Description automatically generated with low confidence

When new acceptance criteria is added or any change is occur , you must loop on all application services that get the golden products and apply this change , for example if new acceptance criteria is added to the golden product :

Total money of sales > 100

Application Service 1

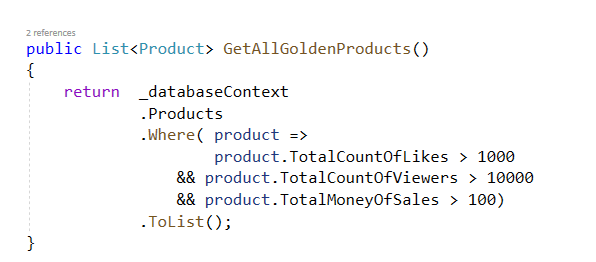


Application Service 2A picture containing text, screenshot, font, line

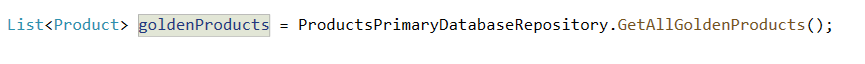
Description automatically generated

When repository pattern return IQueryable this cause a lot of duplications inside the application services , so repository pattern should not never return IQueryable , but return single or list of aggregation roots directly and put the logic or the implementation of database access inside the repository itself , so any change on the requirements will occur in single place in the repository and then will be reflected on the all application services directly.

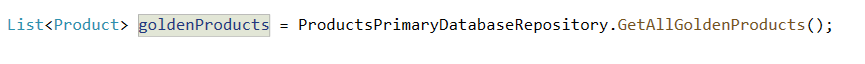
ProductsRepository.cs



Application Service 1



Application Service 2



1. Repository pattern can return native or primitive data types like numbers , Boolean

A screenshot of a computer code

Description automatically generated with low confidence

Important Note :

It usually prefers to use async/await with all operations of repository pattern or any network call , the previous examples are synchronous for just simplify.



**Second Principle :   
Repository pattern should load or save the whole aggregation together.**

From domain driven design view :

the aggregation is a group of objects that logically related and affect on each other , for example on order aggregation when user add new order item , the total price property in the order is recalculated to add the new order item price to the previous total price.

Order.cs :

A screen shot of a computer code

Description automatically generated with low confidence

From the code above :

* Order aggregation root is needed to load all order items to check duplication business rule.
* Any change on the order items list ( add – update – delete ) has an effect on total price on order.

So the whole aggregation objects is work as transactional concept.

This mean the repository can not load part only from the aggregation , the repository must load the whole aggregation.

for example on product aggregation:

A screenshot of a computer

Description automatically generated