CQRS Design Pattern Overview

Irshad Faras Updated date Nov 02, 2020

10.7k 2 4

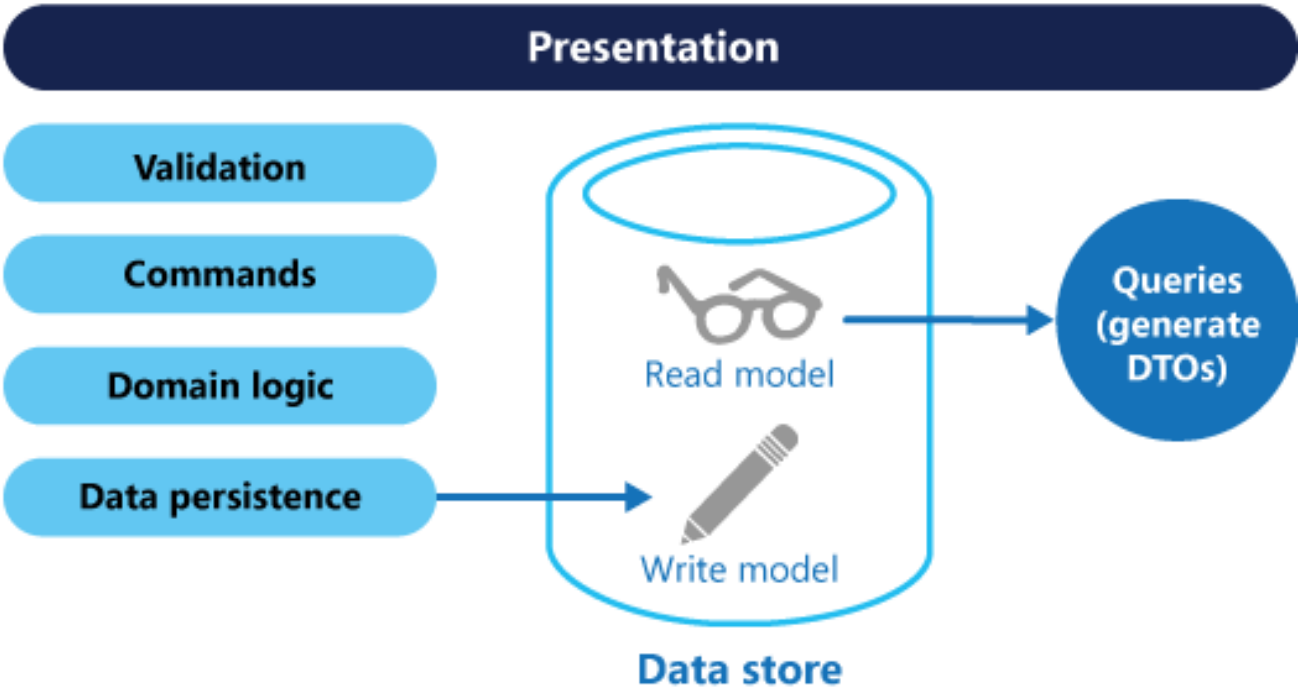
Introduction

Command Query Responsibility Segregation (CQRS) is an architectural pattern that separates reading and writing into two different models. It does responsibility segregation for the Command model & Query model.

So what is Command and Query? Let's see...

Command - Modifies data (e.g. Insert, Update & Delete actions)

Query - Never modifies data (e.g. Select action)



(Source: https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs)

In the above diagram, you can see that the read and write models have separate responsibilities. The write model (Command model) is responsible for validation, commands, complex domain logic, and data persistence. However, the read model (Query model) is responsible only for reading data, which makes it quite simple.

Advantages of the CQRS Pattern

Single Responsibility Principle

As you can see in the above diagram, there are separate models, read & write. It means a method should be either Command or Query. With this separation, you get the Single Responsibility Principle by design.

Independent Scaling

One of the major benefits of the CQRS pattern is Independent scaling. Let's assume you have a website which has more reads than write (something like StackOverflow.com). In such a case, you can scale your read models to get better performance.

Also, you can use two separate databases for the read & write models. With this approach, the write model will send a command to the write database (then it will asynchronously update the read database), and the read model will fetch data from the read database. Furthermore, you can denormalize a read database that will result in simple queries, less complex joins, quick response time, etc. You can even use NoSQL for the read database. Then, you can scale your models independently.

Separation of Concern

Since you have two separate models to read & write, you can keep validations, complex logic into the write model only and keep your read models simple to fetch the data. Similarly, your write model will be responsible for writing to the model.

Disadvantages of the CQRS Pattern

Although the CQRS pattern is exciting and easy to understand, in real scenarios it can be painful if not used correctly. Below are few downsides of this pattern:

- It adds unnecessary complexity if applications have simple CRUD operations, which can be achieved by traditional architectural styles.
- As we require separate models for read & write, code duplication is inevitable.
- In the case of two separate databases for read & write, the write database needs to update the read database that could result in Eventually Consistent Views.

When to use the CQRS Pattern

Below are some scenarios where the CQRS pattern fits perfectly:

- Large projects where high performance is required and independent scalability is required.
- In the applications where business logic is complex. In such a case, you can separate your reads from write to make it more simple.
- If you want parallel development where one team can work on the read models & other team works on write models.

Summary

In summary, the CQRS pattern adds value to the project when it is used with caution. Though it does not fit most of the requirements, it is good to know the design pattern for a developer. I tried my best to explain the concepts behind it in a simple manner. I hope this will help you get started with the concept.

CQRS CQRS design Pattern

Next Recommended Reading
Clean Architecture And CQRS Pattern