Your Books EveryWhere

Analysis and Design Document

Student:Alexandru-Horea Anisorac

**Group:30238**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

The application that we have to implement is an application to search books and buy them into every client library.If another client has a book,than the others are being pushed to a queue until the client retrieves it.Clients can filter books after title,genre and author.These books have two more characteristics:release date and if it is available or note.

# Functional Requirements

The application will be designed and implemented in C# language and framework .NET.The Client should be able to log in,register,filter books and add them to their library.

The managers should manage the books and process the requests.

# Non-functional Requirements

First requirement is to implement the application with one architectural pattern and that is CQRS.Another requirement is to use one OOP language.And the last one we have to use Mediator design pattern.

2. Use-Case Model

*[Create the use-case diagrams and provide one use-case description (according to the format below).*

*Use-Case description format:*

*Use case: <use case goal>*

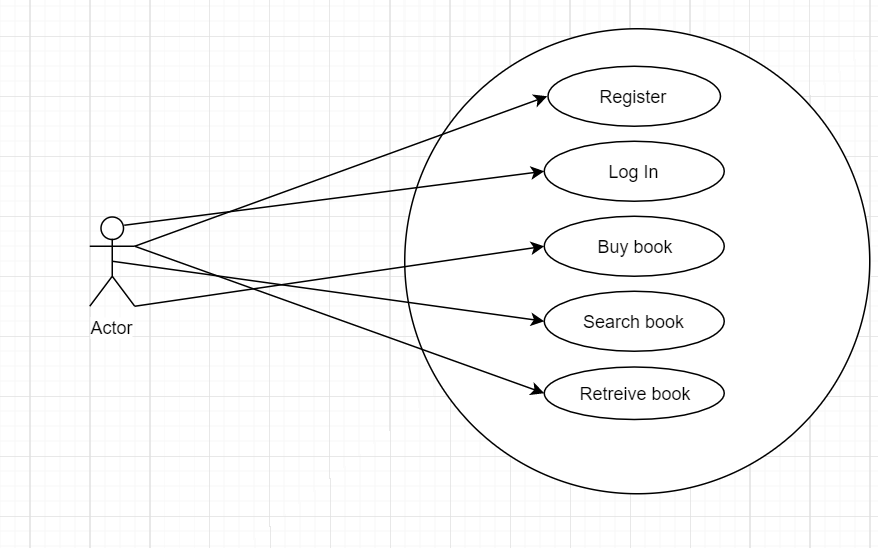
*Level: <one of: summary level, user-goal level, sub-function>*

*Primary actor: <a role name for the actor who initiates the use case>*

*Main success scenario: <the steps of the main success scenario from trigger to goal delivery>*

*Extensions: <alternate scenarios of success or failure>*

*]*



3. System Architectural Design

**3.1 Architectural Pattern Description**

CQRS, which means *Command* *Query Responsibility Segregation*, comes from CQS (*Command Query Separation*) introduced by Bertrand Meyer in *Object Oriented Software Construction*. Meyer states that every method should be either a *query*or a *command*.The difference between CQS and CQRS is that every CQRS object is divided in two objects: one for the query and one for the command.A command is defined as a method that changes state. On the contrary, a query only returns a value.The following schema shows a basic implementation of the CQRS pattern inside an application. All messages are sent through commands and events.

**3.1.1Commands**

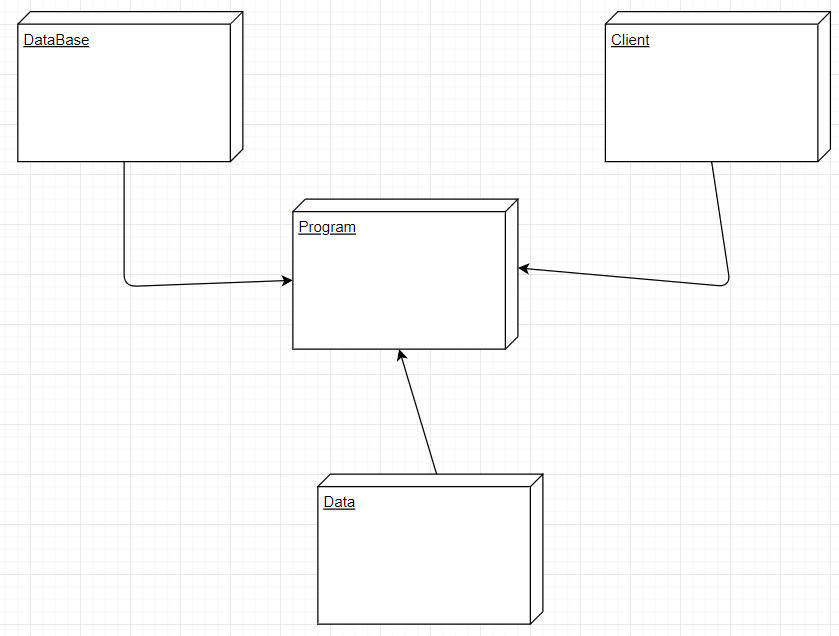
A command tells our application to do something. Its name always uses the indicative tense, like *TerminateBusiness* or *SendForgottenPasswordEmail*. It’s very important not to confine these names to *create, change, delete…* and to really focus on the use cases (see *CQRS Documents*at the end of this document for more information).A command captures the intent of the user. No content in the response is returned by the server, only queries are in charge of retrieving data.

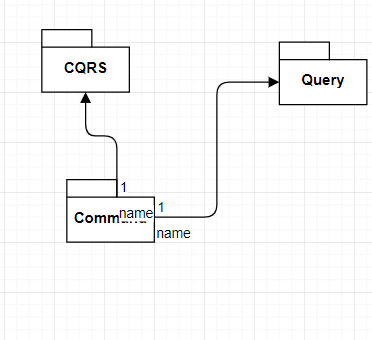
**3.1.2Queries**

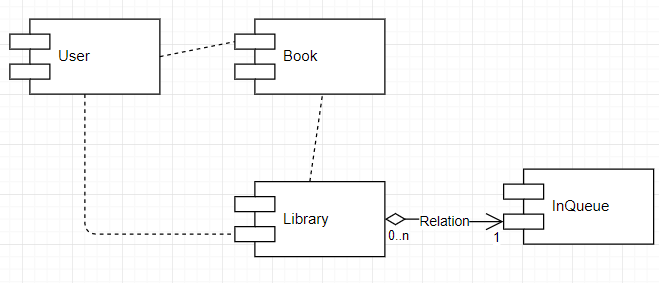
Using different *data stores*in our application for the command and query parts seems to be a very interesting idea. As Udi Dahan explains very well in his article *Clarified CQRS*, we could create a user interface oriented database, which would reflect what we need to display to our user. We would gain in both performance and speed.  
Dissociating our data stores (one for the modification of data and one for reading) does not imply the use of relational databases for both of them for instance. Therefore, it would be more thoughtful to use a database that can read our queries fastly.If we separate our data sources, how can we still make them synchronized? Indeed, our “read” data store is not supposed to be aware that a command has been sent! This is where events come into play.

**3.2 Diagrams**

*[Create the system’s conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]*

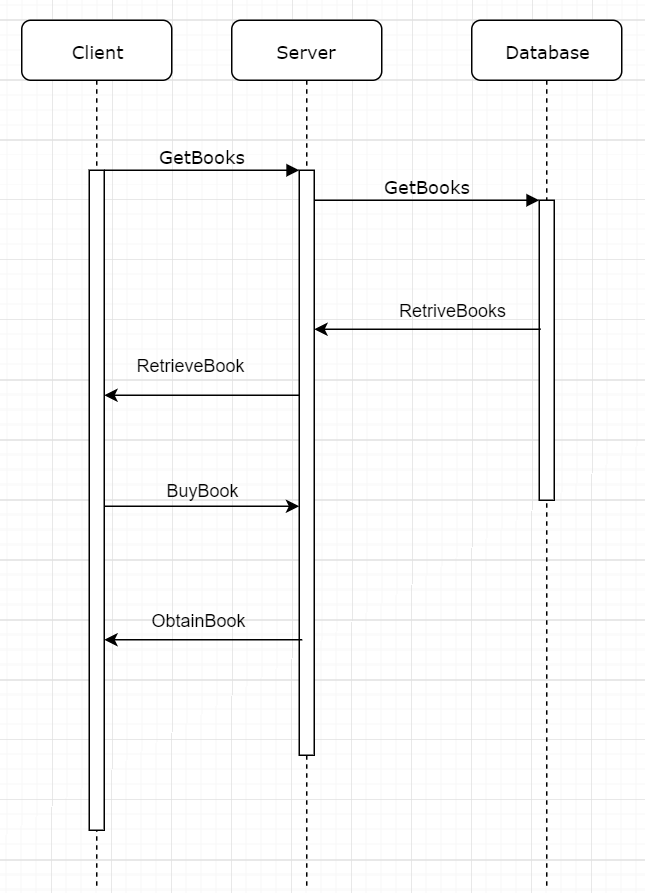






4. UML Sequence Diagrams

*[Create a sequence diagram for a relevant scenario.]*



5. Class Design

**5.1 Design Patterns Description**

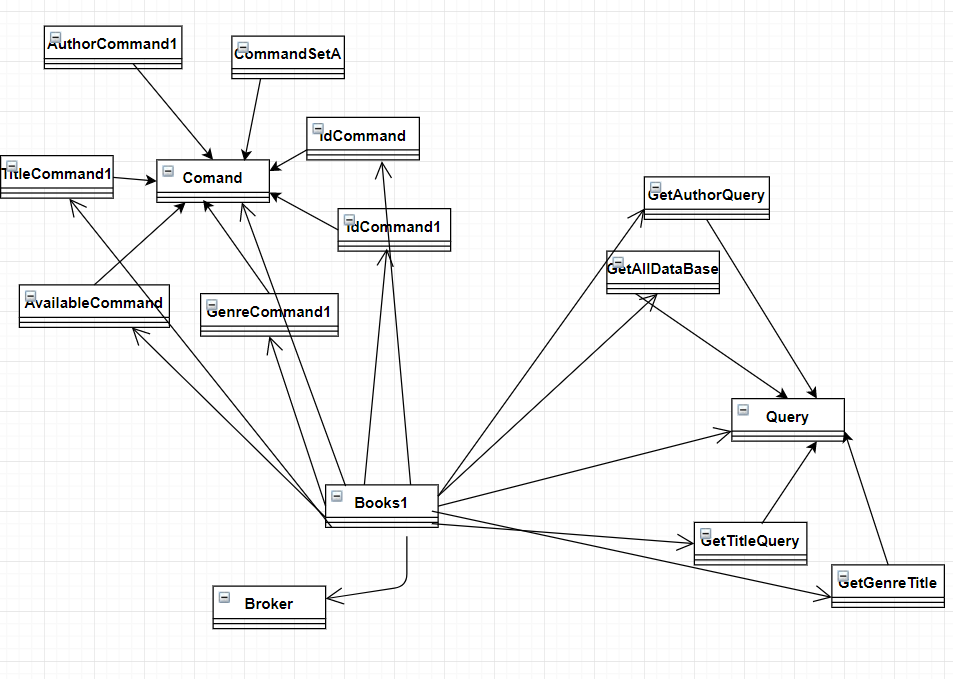
In [software engineering](https://en.wikipedia.org/wiki/Software_engineering), the mediator pattern defines an object that encapsulates how a set of objects interact. This pattern is considered to be a [behavioral pattern](https://en.wikipedia.org/wiki/Behavioral_pattern) due to the way it can alter the program's running behavior.

Usually a program is made up of a large number of [classes](https://en.wikipedia.org/wiki/Class_(computer_science)). [Logic](https://en.wikipedia.org/wiki/Logic) and [computation](https://en.wikipedia.org/wiki/Computation) are distributed among these classes. However, as more classes are added to a program, especially during [maintenance](https://en.wikipedia.org/wiki/Software_maintenance) and/or [refactoring](https://en.wikipedia.org/wiki/Refactoring), the problem of [communication](https://en.wikipedia.org/wiki/Communication) between these classes may become more complex. This makes the program harder to read and [maintain](https://en.wikipedia.org/wiki/Software_maintenance). Furthermore, it can become difficult to change the program, since any change may affect code in several other classes.

With the mediator pattern, communication between objects is encapsulated within a mediator object. Objects no longer communicate directly with each other, but instead communicate through the mediator. This reduces the dependencies between communicating objects, thereby reducing [coupling](https://en.wikipedia.org/wiki/Coupling_(computer_programming)).

**5.2 UML Class Diagram**

*[Create the UML Class Diagram and highlight and motivate how the design patterns are used.]*



6. Data Model

Data that we use in these application were :Users and Books.We create two independent classes which we control all the data.

7. System Testing

The system was tested with by taking all cases of using and finding bugs to prevent them and passed all tests.

8. Bibliography

[1] <https://www.youtube.com/watch?v=Q0Bz-O67_nI>

[2] <https://microservices.io/patterns/data/cqrs.html>

[3] <https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>

[4] <https://www.youtube.com/watch?v=uTCKzPg0Uak>