TUCN student management system

Analysis and Design Document

Student: Ács Dávid

**Group: 30432**

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 4

3. System Architectural Design 4

4. UML Sequence Diagrams 6

5. Class Design 8

6. Data Model 9

7. System Testing 10

8. Bibliography 11

1. Requirements Analysis

# Assignment Specification

The Article management system, where articles are stored and published. Users can view articles without logging in. Writers must login to add, modify or delete articles.

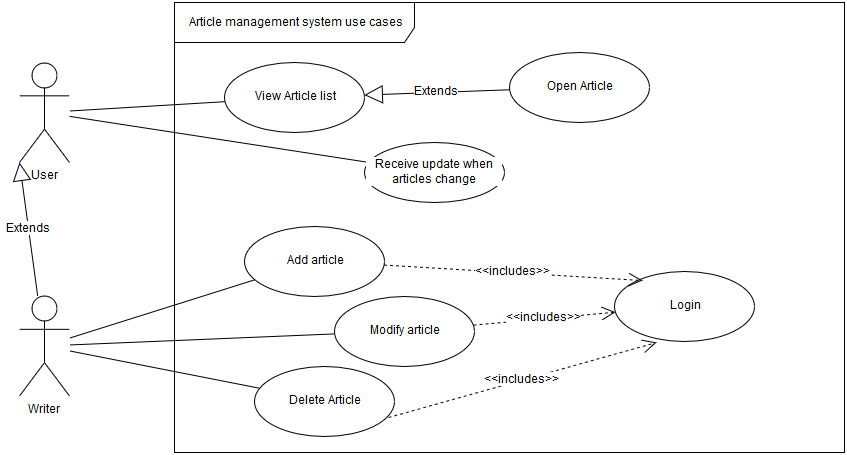
# Functional Requirements

* + 1. **Login:** Writers can log in if they provide the correct username, password combination.
    2. **Article reading:** Users can view and open (read) all the articles that are published by the corporation.
    3. **Writers manage articles:** Writers can add new articles, modify existing articles and delete articles.
    4. **Update in real time:** Once an article is modified, all the currently connected users must get an update, meaning it will

# Non-functional Requirements

* + 1. **Portability:** The system must be able to run on all major releases of the Windows operating
    2. **Response time:** The system must respond to user input in 1s in 99% of cases.
    3. **Maintainability:** Mean time to repair should be as low as 40 hours.
    4. **Readability:** The formatting of the code should reflect the logical structure of code.
    5. **Performance:** Performance (throughput) is not a priority.

2. Use-Case Model



Use-case diagram detailing the needed functionality of the system.

Use case: Add article.

Level: user-goal level.

Primary actor: Writer.

Main success scenario:

1. The writer completes the login process (prerequisite).
2. The writer writes the article and submits the article.
3. If the article is valid the article will appear in the list of articles.

Extensions: the articles may be invalid, and the writer will be presented with an error dialog detailing the source of the error. The system will not change its state in this case.

3. System Architectural Design

**3.1 Architectural Pattern Description**

The client-server architecture was chosen as the overall architecture of the system.

On the client side, the MVVM (model-view-viewmodel) architecture was chosen, to leverage the WPF functionality. This will allow us to make a clear separation of the responsibilities of the view

In MVVM view and the model are completely separated and the viewmodel serves as a mediator between these two.

The viewmodel provides a view of the model, thus introducing a level of indirection between them. The bindings are a crucial element in MVVM since they are the “glue” between the view and the viewmodel. This clear separation also allows the view to be designed by a completely different team than the developers.

On the server side a layered architecture was chosen to logically group the different technical concerns. This will provide modularity and allow us to swap out layers if changes were to be made in the environment or in the application itself.

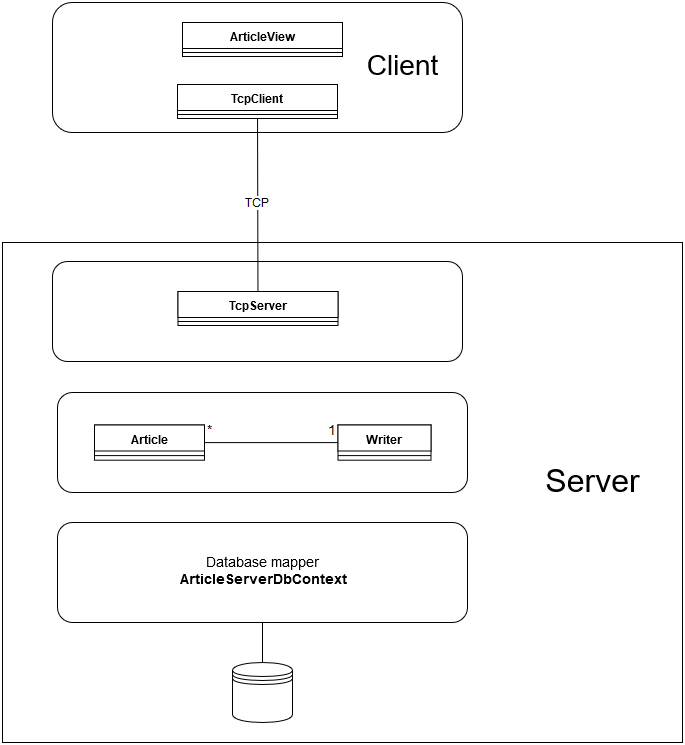
The connection between the client side and the server side is made thought sockets.

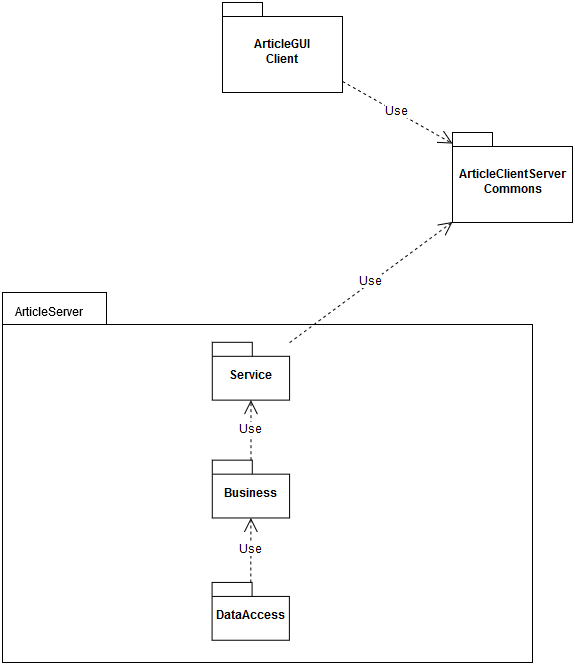
* 1. **Diagrams**

As you can see, the presentation layer contains the Views and Presenters, while the Domain Model is exposed to the client side through the Services.

For the business logic the **Domain Model** was chosen.

For mapping the Domain Model to the database Entity Framework will be used, with the Code-First approach.



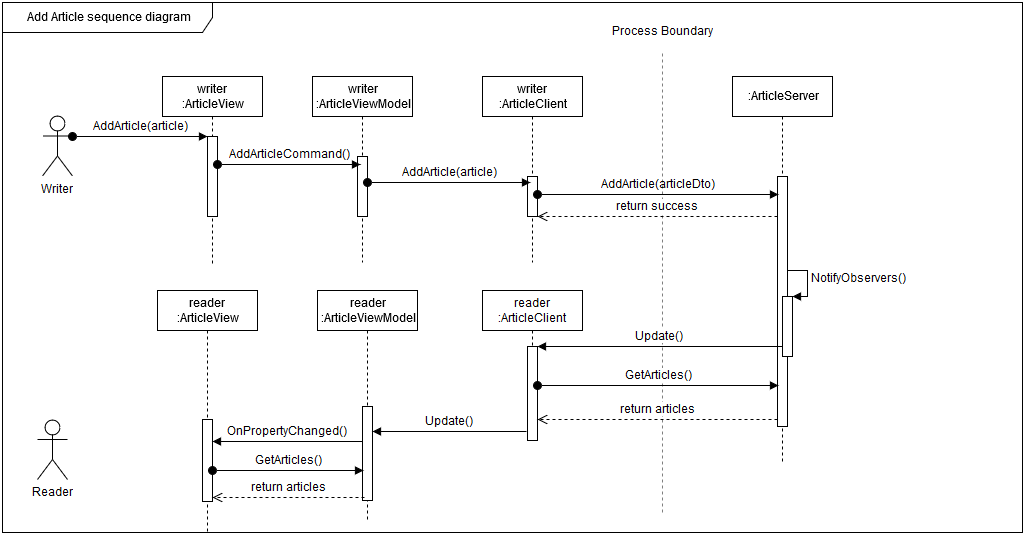


The component diagram is really like that of a layered system in the server. I extracted a third package (ArticleClientServerCommons), to avoid circular dependencies.



Since the server is run on a single tier, the deployment diagram consists of two servers: The main server where the application will run and the Database server where the database will run.  
We may observe that the server is capable of servicing multiple clients at the same time, and that the connection between the client and the server is made through TCP.

4. UML Sequence Diagrams

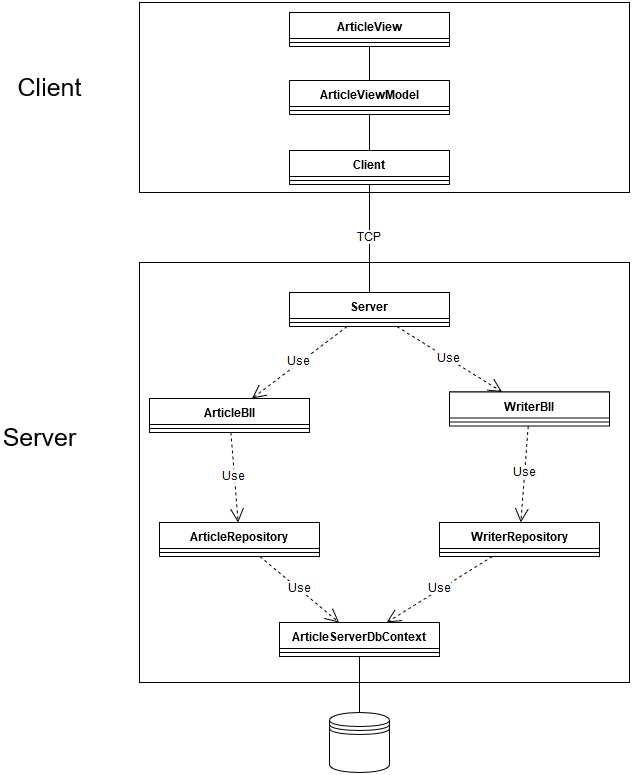


5. Class Design

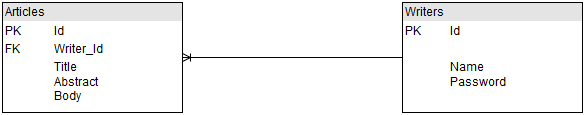
**5.1 Design Patterns Description**

Repository design pattern for the database. Command pattern may be used to transfer commands between the client and the server.

**5.2 UML Class Diagram**



6. Data Model



The data model (entity relationship model) is presented above. Please note that this database design was not designed by me, it was created by entity framework, using the code first approach.

7. System Testing

Unit testing and system testing will be performed as the system is implemented and once the system has been completely built. The unit testing method chosen is mocking the objects.

8. Bibliography

Inversion of Control with Microsoft Unity:

<https://msdn.microsoft.com/en-us/library/dn178470(v=pandp.30).aspx>

synchronous socket client example:

<https://docs.microsoft.com/en-us/dotnet/framework/network-programming/synchronous-client-socket-example>

Async socket server example:

<https://docs.microsoft.com/en-us/dotnet/framework/network-programming/asynchronous-server-socket-example>

Relay command in WPF:

<https://stackoverflow.com/questions/22285866/why-relaycommand>