Assignment 3

Analysis and Design Document

Student: Boros Hanniel

**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 3

3. System Architectural Design 4

4. UML Sequence Diagrams 7

5. Class Design 8

6. Data Model 10

7. System Testing 10

8. Bibliography 11

1. Requirements Analysis

# Assignment Specification

Design and implement a Java application for the management of students in the CS Department at TUCN. The application should have two types of users (students and teacher/administrator user) which have to provide a username and a password in order to use the application.

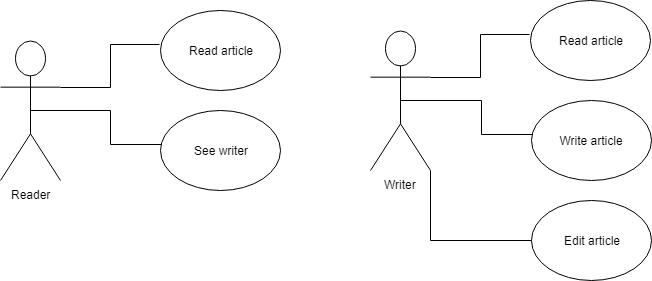
# Functional Requirements (for writer account)

* Authentication
* Edit profile
* Add new article
* Edit existing article

# Non-functional Requirements

* Usability
* Performance (response time < 1s)
* Security
* Data integrity
* Documentation

2. Use-Case Model



One use case description

Use case: Create a new article

Level: goal-function

Primary actor: writer <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>

Writer Log In option from menu

Writer enters necessary information (article title, article abstract, article body)

Writer presses Create button, finishing process.

Writer waits a short time while information is entered in the database. The article automatically shows up in the articles list. All readers are notified.

Extensions:

Alternate scenario for failure:

if connection to database fails for some particular reasons, or an Exception is encountered, the information may not be updated.

3. System Architectural Design

**3.1 Architectural Pattern Description**

**Client-Server Architectural Pattern**

The client/server architectural style describes distributed systems that involve a separate client and server system, and a connecting network. The simplest form of client/server system involves a server application that is accessed directly by multiple clients, referred to as a 2-Tier architectural style.

The current project contains a server part and a client part, respectively, as the DP consists in splitting the system into two categories of programs: clients and sever.

A client is a program or a process which connects to another program (the server) and lets it carry out a specific task. In particular it might require supplying some data from the server.

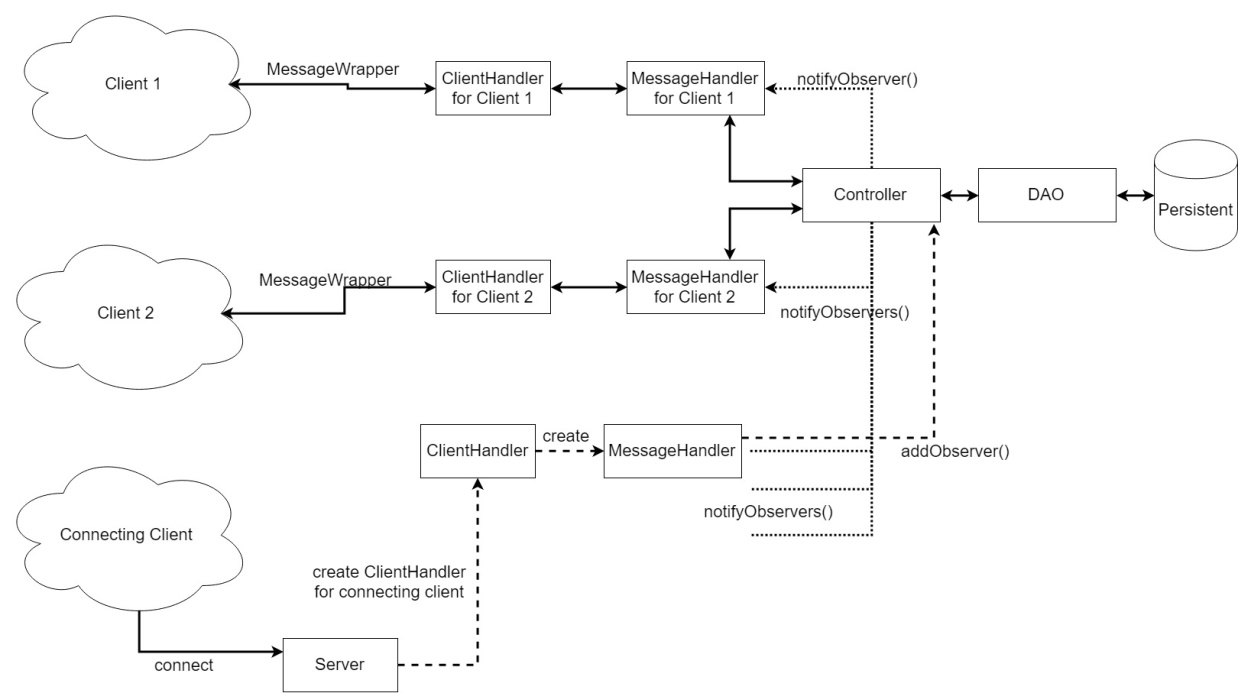
A server is a program or a process which provides services for clients. For example, it might supply some data or a result of processing data to clients.

A fundamental notion for exchanging information through the network is a socket.

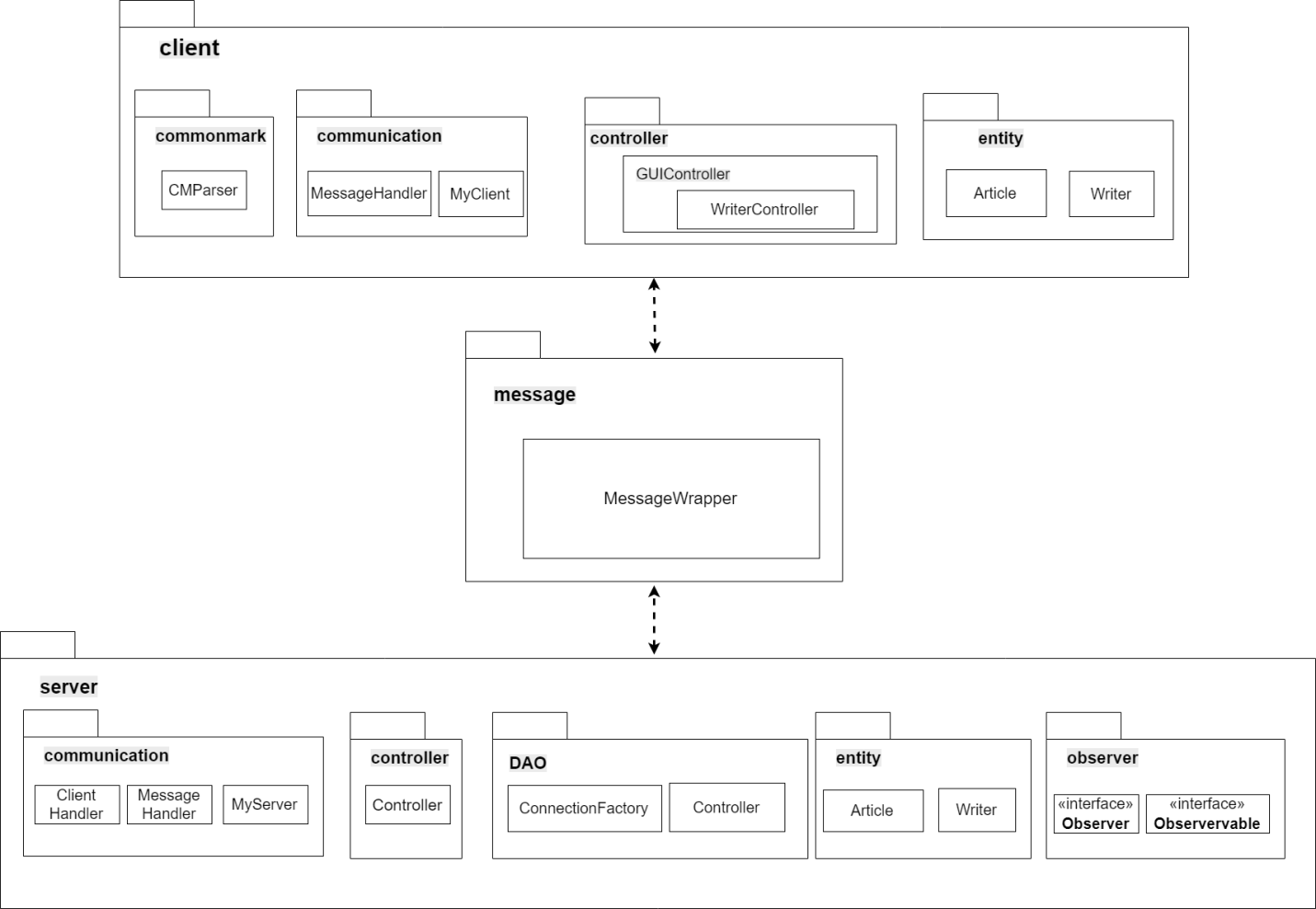
A socket is one end-point of a two-way communication link between two programs running on the network.

**3.2 Diagrams**

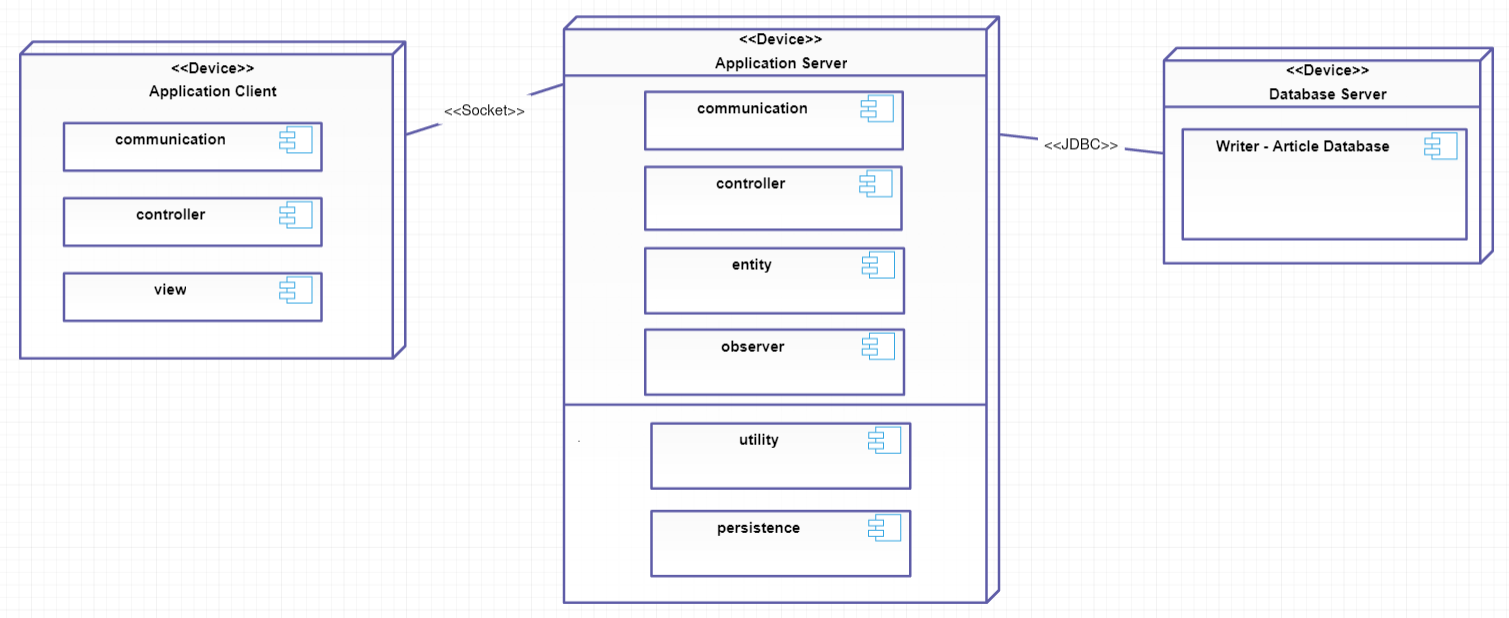
Client-Server Architecture Pattern of the solution



Package diagram

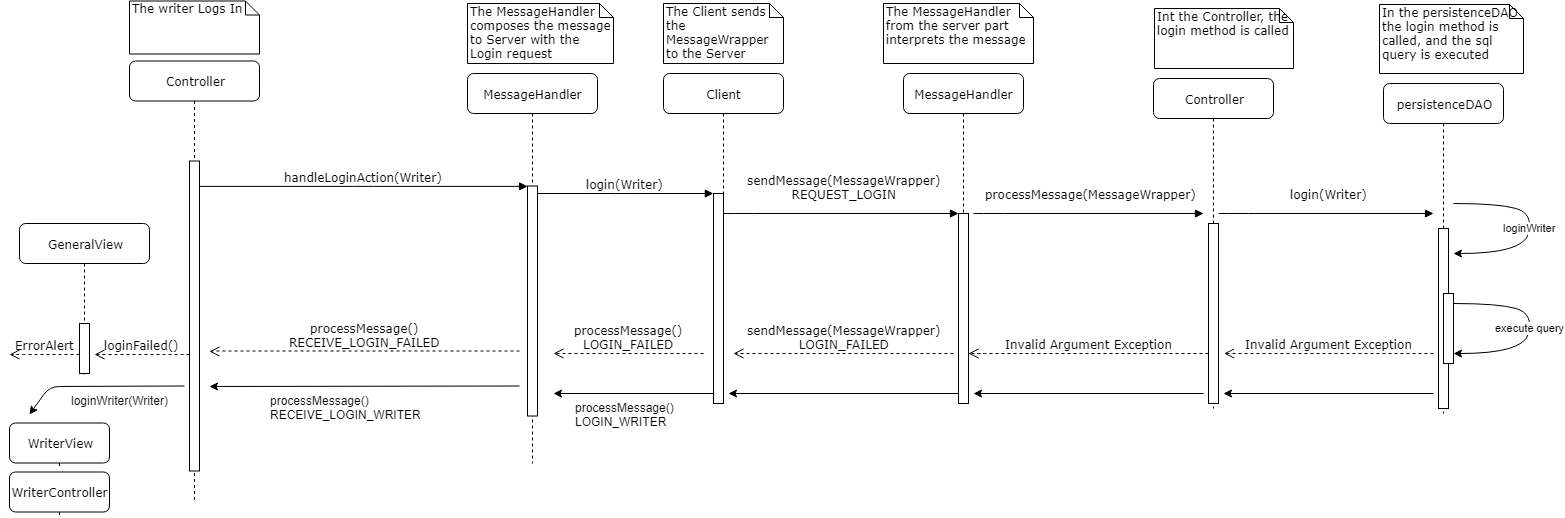


Deployment diagrams



4. UML Sequence Diagrams

Sequence diagram for a writer logging in.



5. Class Design

**5.1 Design Patterns Description**

**Observer Behavioral Design Pattern**

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically. The observer DP is composed of two main subjects: the observer and the observable. The observable has a list of observers and whenever the observable’s state is changed, all of its observers are notified.

The roles are fulfilled implementing specific interfaces: Observer for the observers and observable for the observables.

In this projects, the Observable is the Server-side controller which performs the core logic of the application. The observers are Clients, which are represented by the ClientHandler and it’s allocated MessageHandler classes. Whenever the Controller state’ is changed, all of the MessageHandler’s are notified, and they communicate the current changes to the client they are responsible of. In this way, every change in the application can be seen immediately, as it is performed automatically without being necessary a refresh option. In other words, all the clients are notified automatically when a change in the data (managed by the server) is performed.

**Singleton Creational Design Pattern**

This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

This pattern was used for the design of the classes: ConnectionFactory, PersistenceDAO, Controller (in the server).

**Data Mapper Design Pattern**

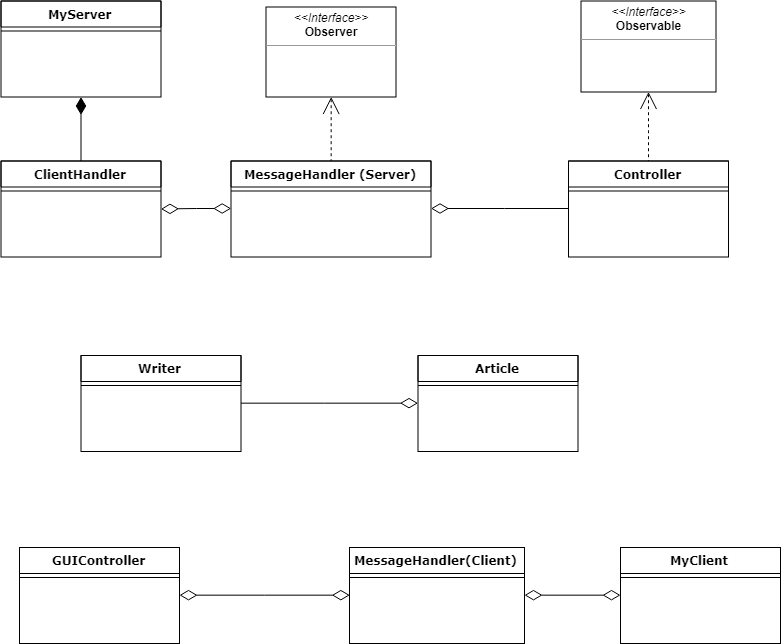
A Data Mapper is a Data Access Layer that performs bidirectional transfer of data between a persistent data store (often a relational database) and an in-memory data representation (the domain layer). The goal of the pattern is to keep the in-memory representation and the persistent data store independent of each other and the data mapper itself. The layer is composed of one or more mappers (or Data Access Objects), performing the data transfer.

This pattern was used for the design of the Data Access Objects.

**5.2 UML Class Diagram**

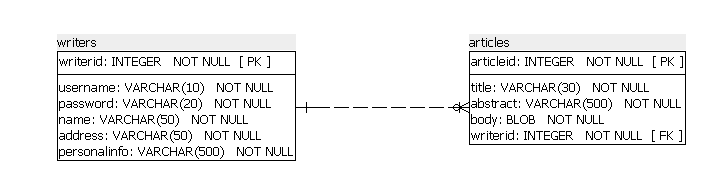
Sinces the UML Diagram for the whole system would be of great proportions, I decided to present the class diagrams of the relevant classes of each module and layer, simplifying it where I had the possibility.

As well, the general use-relationship is not presented below.



**6. Data Model IR diagram**

Below I present the Data IR diagram. This was used to represent and model the data of the application. There are four tables: three of them for direct data storage (client and other information as well as courses) and the other one for relationships (it models a many-to-many relationship by a one-to-many ⬄ many-to-one relationship).



7. System Testing

I will use unit testing in order to test my application.

I tried to make an implementation which will be favorable and sustain testing.

Mocking is primarily used in unit testing. An object under test may have dependencies on other (complex) objects. To isolate the behavior of the object one wants to test, it replaces the other objects by mocks that simulate the behavior of the real objects. This is useful if the real objects are impractical to incorporate into the unit test.

In short, mocking is creating objects that simulate the behavior of real objects.

8. Bibliography

Here are mentioned some resources which were helpful for writing this documentation and as well gave me a better understanding of the concepts and guided me in the implementation process:

**Information and knowledge:**

Microsoft Application Architecture Guide, 2nd Edition

Software Architecture Patterns, by Mark Richards

<https://www.tutorialspoint.com>

<https://en.wikipedia.org>

<https://stackoverflow.com>

<http://www.java2s.com>

<http://www.mkyong.com>

**Tools helpful for the documentation:**

<https://www.draw.io/>

<https://www.smartdraw.com>

<https://creately.com>