Assignment A3

Student: Adrian Timis

**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 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

This application is meant to provide an online environment where readers can read articles posted by writers without the need of logging in order to view them. The only people who can post articles are the writers, who are given access to the platform by the system administrator. That means that there is no section for creating accounts, but rather you are given access by a person. The authentication of the writers must be secure, since only them should be able to post articles, and no other people.

# Functional Requirements

Functional requirements for this application include:

* Safe and secure authentication only for the writers who were given access to the platform
* Live update of the list of articles on the platform, so that the readers do not need to refresh the page in order to check whether a new article has been posted
* The application must be client-server
* The application must use the Observer design pattern in order to notify the users when a new article has been posted

# Non-functional Requirements

The non-functional requirements include:

* Performance – the user should not wait too much for a new article to load
* Security – since only the writers who have been given access should be able to post articles
* Capacity – the platform should hold without problems a large number of articles

2. Use-Case Model

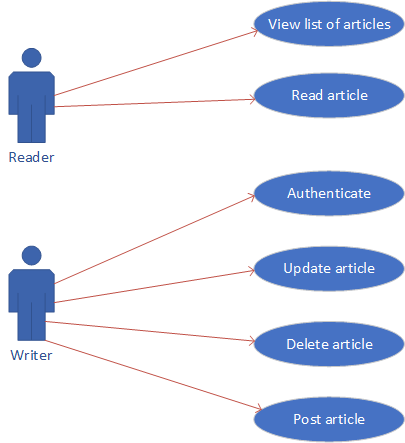
*Use case: Provide app functionality*

*Level: Summary level*

*Primary actor: Reader*

*Main success scenario: access platform, read posted articles*

*Extensions: Read article or do not read an article*



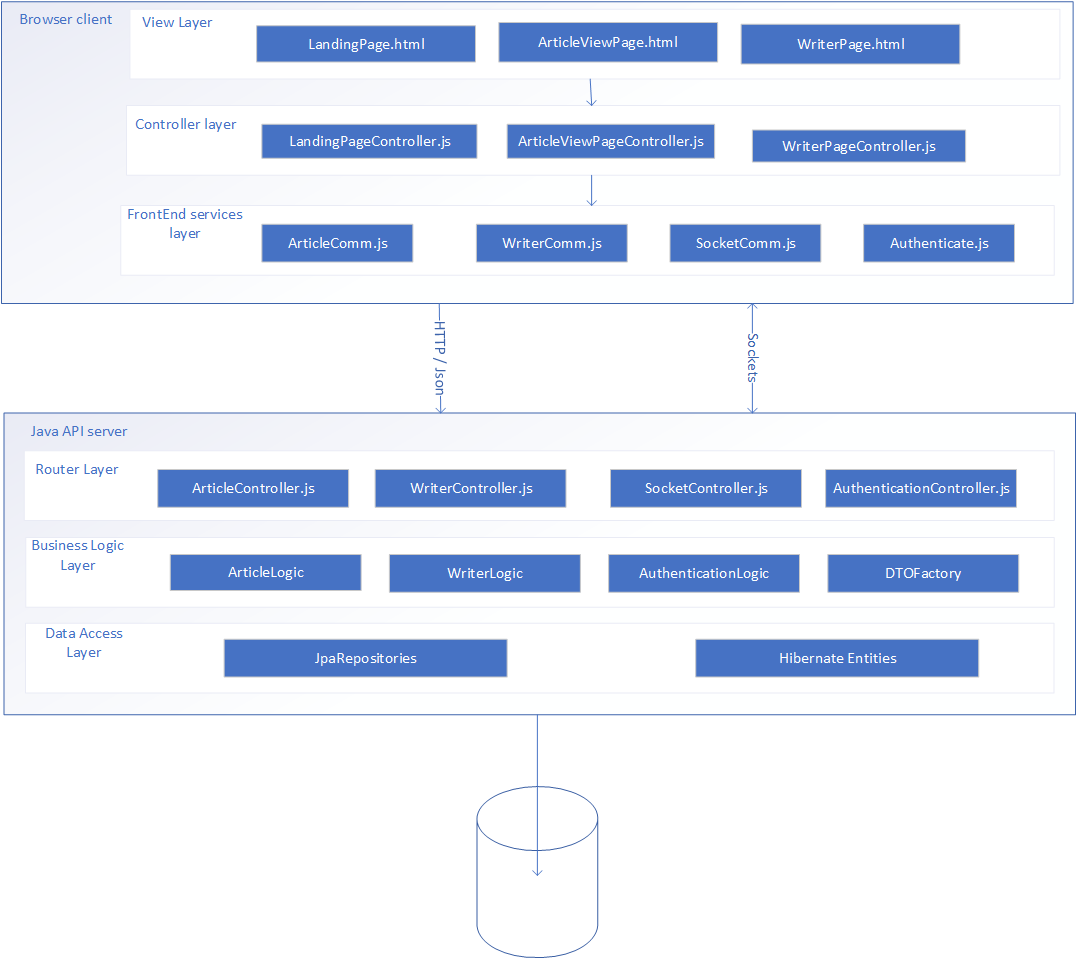
3. System Architectural Design

**3.1 Architectural Pattern Description**

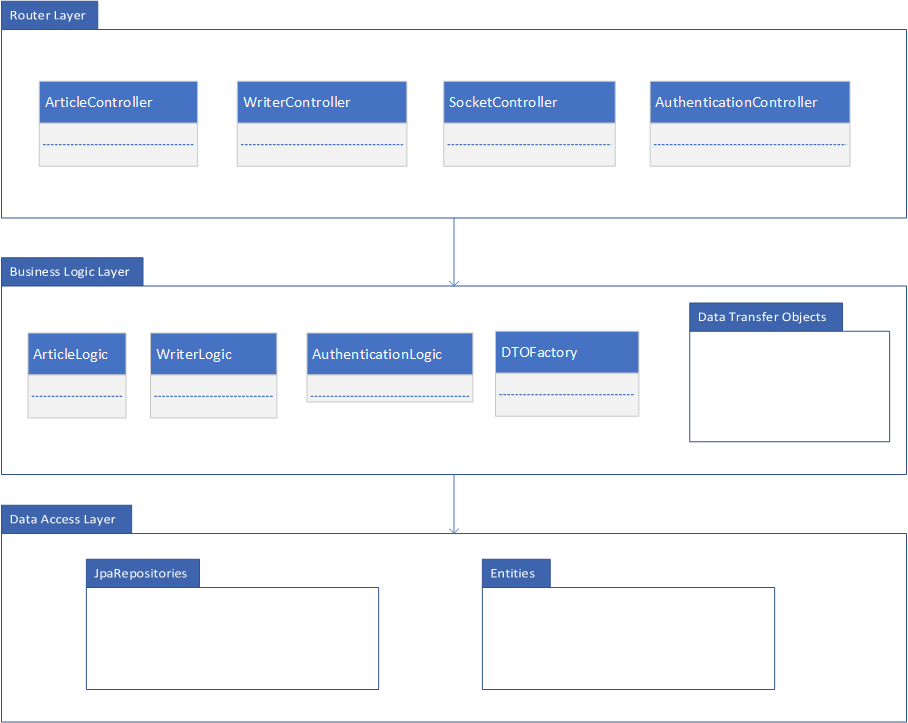
The client-server architecture is a distributed application structure that distributes tasks between the providers of a resource or service, which are called servers, and service requesters, which are called clients. In this case, the server will be Java API, and the client will be an angular single page application. Since the clients need to be updated automatically when an article has been posted, the communication will be done using sockets. Some http requests may also exist, for retrieving the entire article, and for the writers which will post articles on the platform.

The server side of the platform will be structured using a layered architecture, in order to easily distinct between the classes role’s.

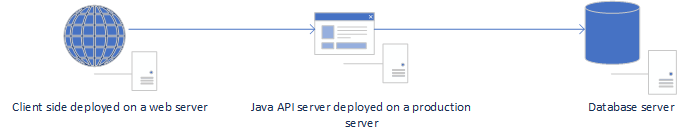
**3.2 Diagrams**

**

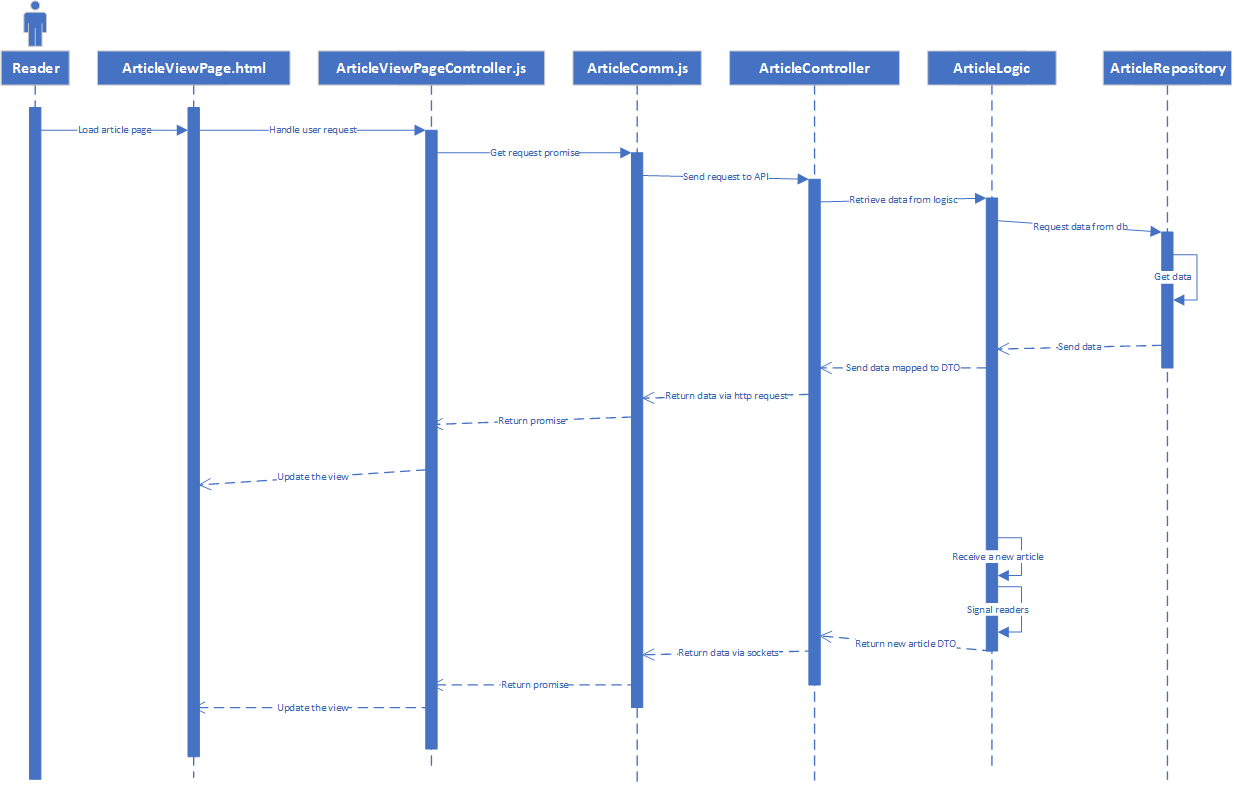
Package diagram:

**

Deployment diagram:



4. UML Sequence Diagrams



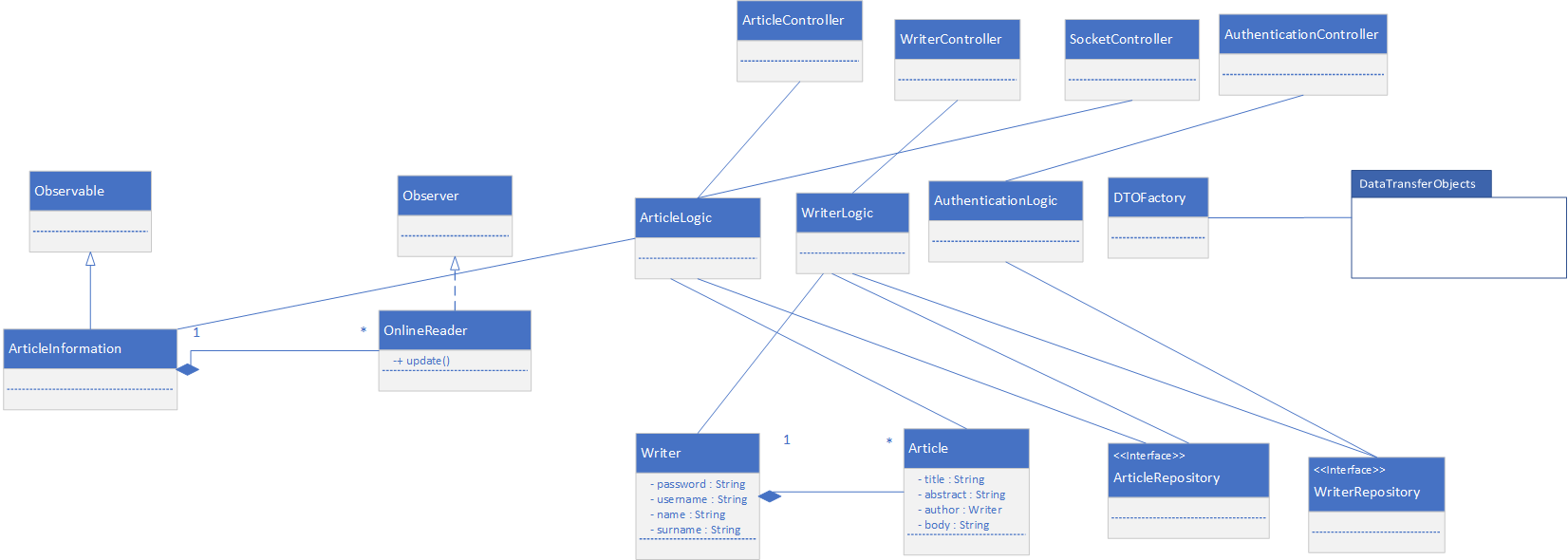
5. Class Design

**5.1 Design Patterns Description**

In this application the design patterns used are the factory design pattern and the observer. The factory design pattern is used to create data transfer objects, and to remove the need of creating those objects in the logic. This means that when new fields are added to the data transfer objects, or we want to change in some way the manner of creating them, we have to do this simply from one class. Also, this presents a supplement level of abstraction, as the logic classes do not know how the objects are created, they just receive the required object.

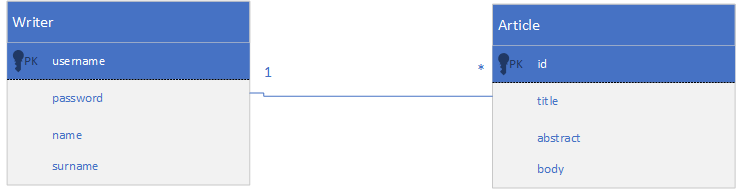
The second design patter is the observable design pattern. This is used to keep track of the readers who are currently on the platform, and to be able to update them with live data as soon as other writers upload a new article. The design pattern means that we have a class which holds information about the readers, that information at the moment will probably include just information regarding the socket information about each reader. This class will implement the Observer interface, which means that we will have to present an implementation of the public void update() method, which will send the new article to the respective reader via sockets. There is another class, Observable, which is the object that the readers ‘observe’. This class will hold information about the articles, and especially about the new ones. This class will extend the Observable class, which already provides implementations for methods of adding observers, and updating all of them.

**5.2 UML Class Diagram**



6. Data Model

The data model does not consist of many classes, since the vast part of the application consists of the socket communication.



7. System Testing

The system testing will be done mainly using unit testing and validation testing. As for methods, data-flow will be the preferred method.

8. Bibliography

* <https://msdn.microsoft.com/en-us/library/ee658109.aspx>
* <https://www.tutorialspoint.com/uml>
* <https://herbertograca.com/2017/08/03/layered-architecture/>
* <https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>