Assignment 3

Student: Raț Gabriel Cătălin

**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 5

4. Class Design 5

5. Data Model 8

6. System Testing 8

7. Bibliography 9

1. Requirements Analysis

# Assignment Specification

In this assignment we are required to design and implement an application that has as main usage case the management of articles in a news agency based on a client-server architecture. The main actors are the viewer and the admins who can log in using an account & password authentication method.

# Functional Requirements

*The application is required to provide the following functionalities for the:*

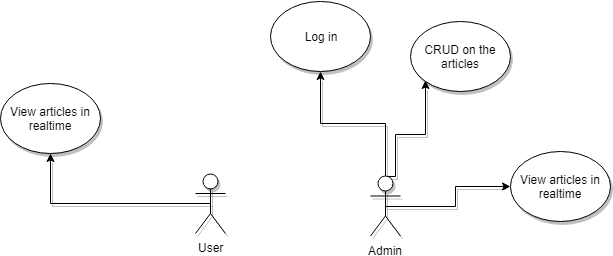
*-Viewer:*

* *Visualize the articles*

*-Admin:*

* *CRUD operations on any article* Non-functional Requirements
* *Log in/ log out*
* *Security – each type of user must have access only to its meant functionalities*
* *Usability – easy to understand and use user interface*
* *Response time – under 0.5 seconds*
* *Portability – perk of the java programming language*
* *Robustness – the system can cope with erroneous input because all the input data is validated*

2. Use-Case Model



*Use case: Admin creates an article*

*Level: user-goal level*

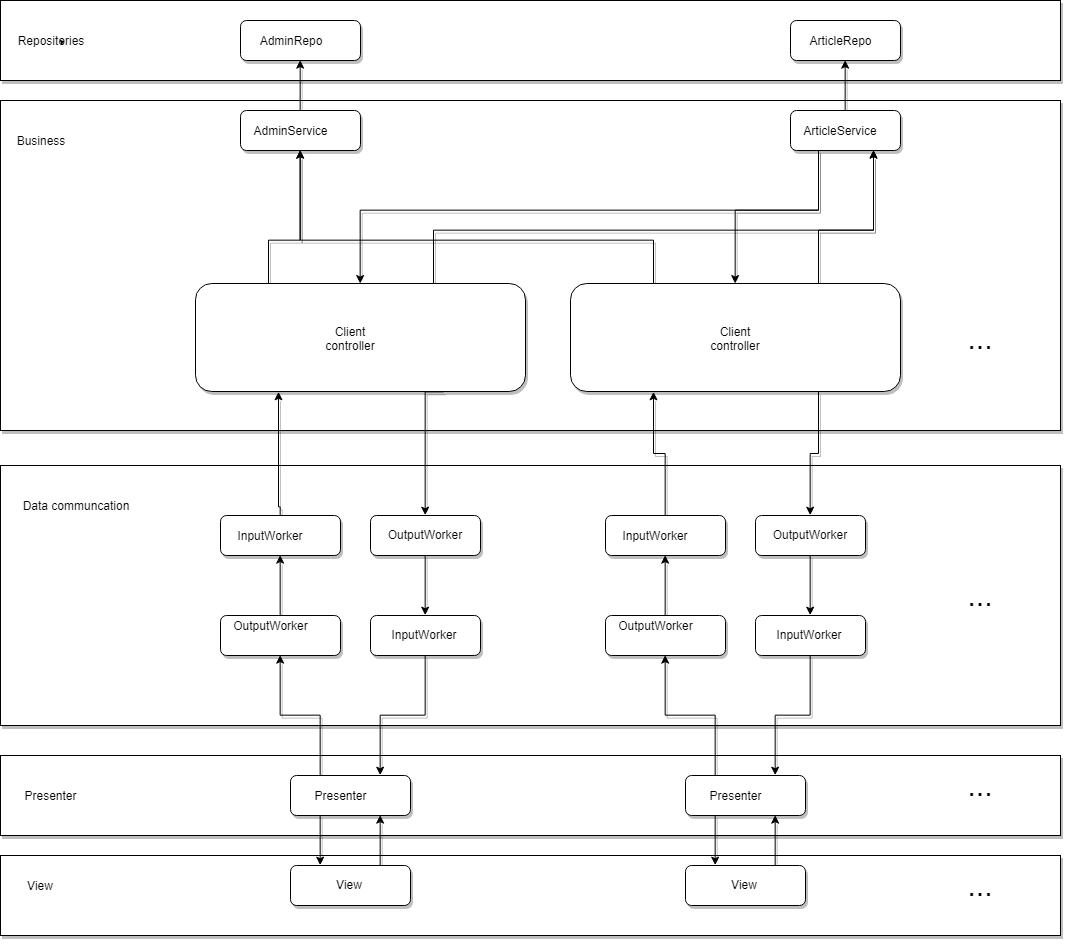
*Primary actor: admin*

*Main success scenario: After an admin logs in he/she presses the Add article button and a new window will appear that has fields for title, author, abstract and body of the article. After he/she is finished writing the article, if the Save button is pressed the article window will disappear and the article information will be added to the list of articles and all connected clients will pe notified of the change and their articles list will update.*

3. System Architectural Design

**3.1 Architectural Pattern Description**

* *Layered Architectural Pattern*



*Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface logic, whereas a business layer would be responsible for executing specific rules associated with the request.*

*The communication layer consists of multiple two way connections from the server to multiple users trough sockets.*

*Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a business request. For example, the presentation layer doesn’t need to know or worry about how to get customer data. It only needs to display that information on a screen format.*

*Similarly, the business layer doesn’t need to be concerned about how to format the customer data for display on a screen or even where the customer data is coming from. it only needs to get the data from the persistence layer, perform business logic against the data (e.g. calculate values or aggregate data or validate), and pass that information up to the presentation layer.*

* *Model-View-Presenter pattern*

*Like MVC, MVP is based on three components: the model, the view, and the presenter.*

*Model*

*The model represents the logic of the view. This can also be the business logic. However, all functionality must be accessible via the model in order to operate the view. The model is controlled solely by the presenter. The model itself knows neither the view nor the presenter.*

*View*

*The view contains no controlling logic and is solely responsible for the representation and the inputs and outputs. It gives neither access to the functionality of the presenter nor to the model. All control of the view is done by the presenter.*

*Presenter*

*The presenter is the link between model and view. It controls the logical processes between the other two layers and ensures that the view can fulfill its functionality.*

**3.2 Diagrams**

4. Class Design

**4.1 Design Patterns Description**

* *The builder*

*Is a design pattern from the area of software development and belongs to the category of the generation pattern ( Creational Patterns ). It separates the construction of complex objects from their representations , allowing the same design processes to be reused .*

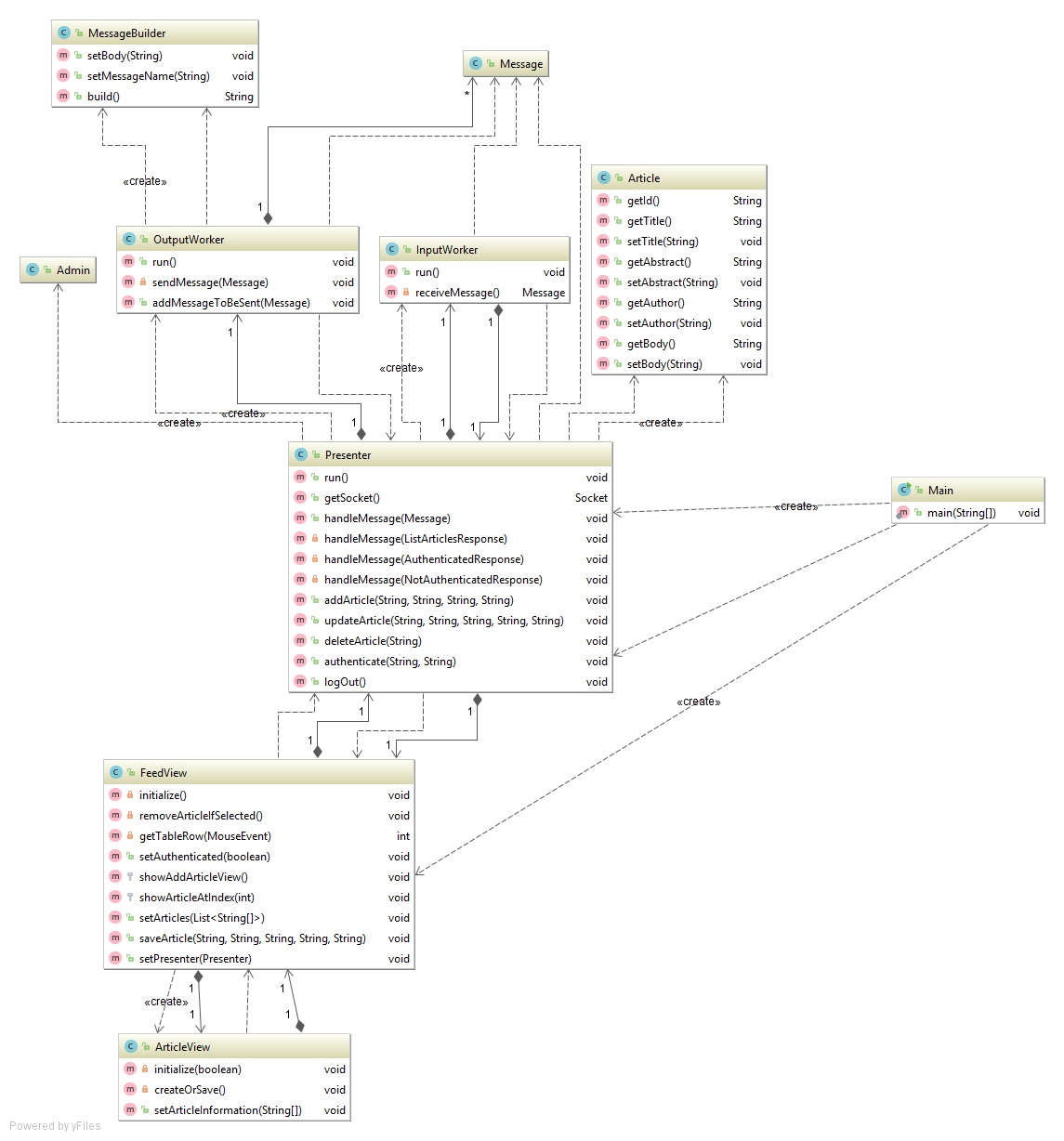
* *Dependency injection*

*Java Dependency Injection design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable. We can implement dependency injection in java to move the dependency resolution from compile-time to runtime.*

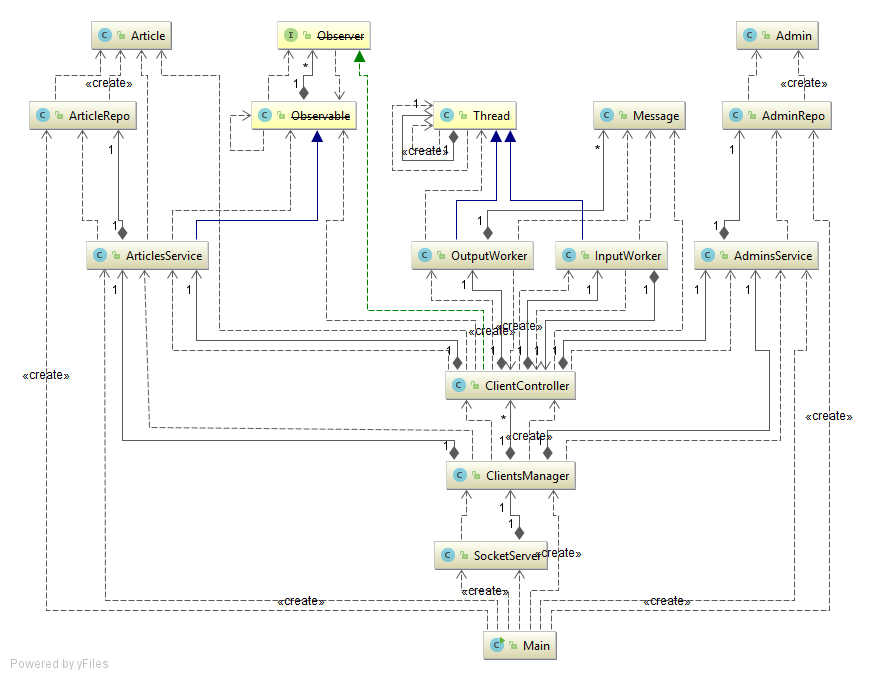
*This can also help when unit testing. In this exact case I will be injection the view and the data access in the controller. This way I will be able to unit test the controller.*

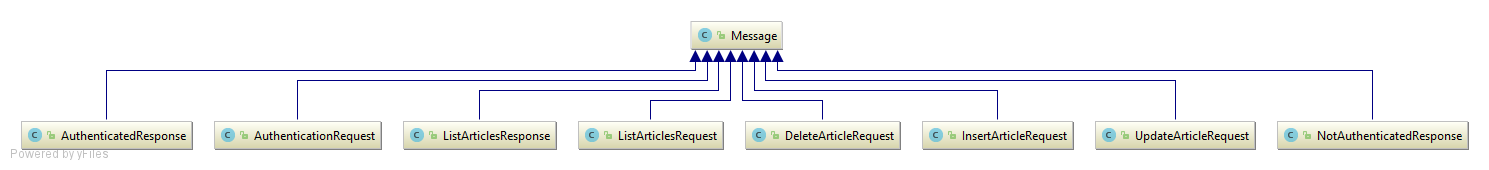
**4.2 UML Class Diagram**

5.2.1 Class diagram for client



5.2.2 Class diagram for Server



5.2.3 Class diagram for message classes

5. Data Model

6. System Testing

* Unit testing

The unit test is particularly important because it considers the test object in isolation and thus precludes interactions with other components. Therefore, occurring error effects can be clearly attributed to the tested software module, which may severely limit the search for the defect as the cause of the error.

The aim of the unit test is to check whether the software component fulfills the functional and non-functional requirements specified in the specification.

Functional requirements are defined as those requirements that specify the input or output behavior about the correctness of the delivered results.

By contrast, non-functional requirements include, for example, performance aspects or memory consumption during runtime.

* Integration testing

Integration tests check the cooperation of multiple system parts of increasing complexity from individual modules to subsystems up to the overall system.

The goal of the integration test is to check the interaction of different parts of a system. In doing so, both the correct interaction, such as the exchange of data through messages or shared memory, access to databases or the use of functionality through calls for interface functions beyond individual parts, must be checked, as well as the non-occurrence of undesired effects. Errors typically detected include incorrect use of interfaces, unauthorized parameter values, but also shared resource or race-conditional blocks due to unordered changing of shared data to inconsistent data states. Integration tests are performed because of the predominantly internal system focus of the creator of the software.

7. Bibliography

<https://en.wikipedia.org/wiki/Non-functional_requirement>

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter>

<https://sourcemaking.com/design_patterns/observer>