Assignment 3

Student: Ardelean Eugen-Richard

**Group: 30433**

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

***1. Objective***

The objective of this assignment is to allow students to become familiar with the client-server

architectural style and the Observer design pattern.

***2. Application Description***

Use Java/C# API to design and implement a client-server application for a news agency. The application has three types of users: the readers, the writers and an administrator.

The readers can view a list of articles, read an article and do not need to login in order the use the application.

The writers ​need to authenticate in order to ​create, update or delete articles.

The admin ​is the only one who can create writer accounts, but cannot create new admin accounts. So the admin accounts are preset by the application developer and cannot be altered.

An article has the following components:

* Title
* Abstract
* Author
* Body
* List of related articles

When reading an article the user should be able to see the title and and the abstract of the related articles. By clicking on the title of the related article, he will be taken to a page that displays the full article.

The application must support multiple concurrent users. If a writer posts a new article, the readers must see it in the list of articles in real time, without performing any refresh operation.

**Bonus points:** ​If you can include pictures in the articles you get 1 bonus point.

***3. Application Constraints and Technical Requirements***

* The application must be client-server.
* Use the Observer design pattern for updating the list of articles in real time
* For sending data from the client to the server use JSON serialization.
* When writing an article, show a list that supports multi-select for choosing the related articles.

***4. Deliverables***

In the same Github repository as Assignment 1 and Assignment 2, add a new folder with the following files:

* Analysis and design document.
* Implementation source files.
* Readme file that describes the installation process of the application and how to use it:
* how to install your application on a clean computer
* how to access your application and with what users
* images with all use cases and their scenarios implemented

# Functional Requirements

# The functional requirements for this project are:

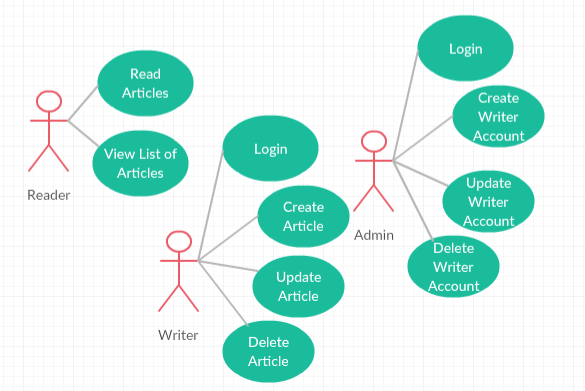
* The application must be implemented using Java/C# API.
* Performance: a response time of maximum 2 seconds
* Availability: 16hours/day
* Security: the passwords will be encrypted when sent to the database
* Usability: the application will work on any computer that has java installed

# Non-functional Requirements

# The non-functional requirements for this project are:

* Reader: view a list of articles, read an article and do not need to login in order the use the application.
* Writer: ​need to authenticate in order to ​create, update or delete articles.
* Admin: ​create writer accounts, but cannot create new admin accounts
* When reading an article the user should be able to see the title and and the abstract of the related articles. By clicking on the title of the related article, he will be taken to a page that displays the full article.
* The application must support multiple concurrent users. If a writer posts a new article, the readers must see it in the list of articles in real time, without performing any refresh operation.
* The application must be client-server.
* Use the Observer design pattern for updating the list of articles in real time
* JSON serialization for sending data from the client to the server
* When writing an article, show a list that supports multi-select for choosing the related articles.

2. Use-Case Model



Use case: Write Article

Level: user-goal level

Primary actor: User

Main success scenario:

* open application
* enter user login information, click login button
* introduce the necessary information for writing an article
* clicks the write button

Extensions: when the written information is erroneous, an error message is shown

Use case: Create Writer Account

Level: user-goal level

Primary actor: Admin

Main success scenario:

* open application
* enter admin login information, click login button
* introduce the necessary information for creating a writer account
* clicks the create button

Extensions: when the written information is erroneous, an error message is shown

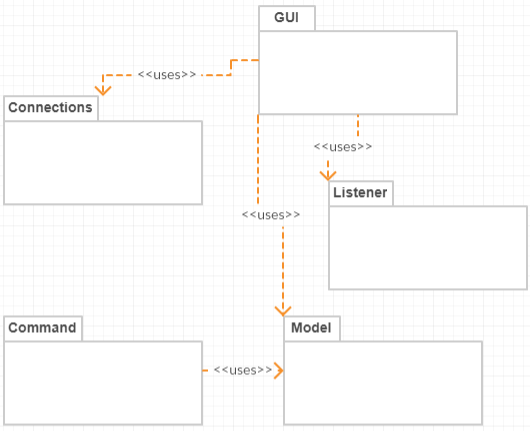
3. System Architectural Design

**3.1 Architectural Pattern Description**

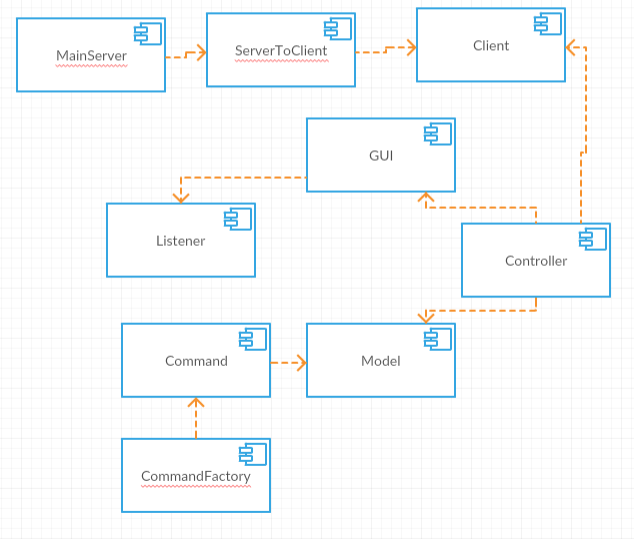
The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests. [1]

**3.2 Diagrams**

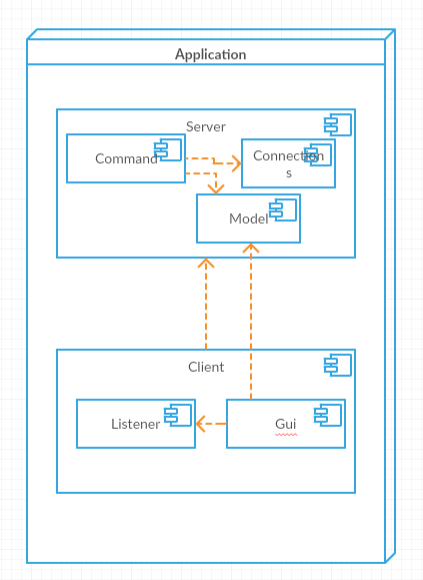
Package Diagram



Component Diagram

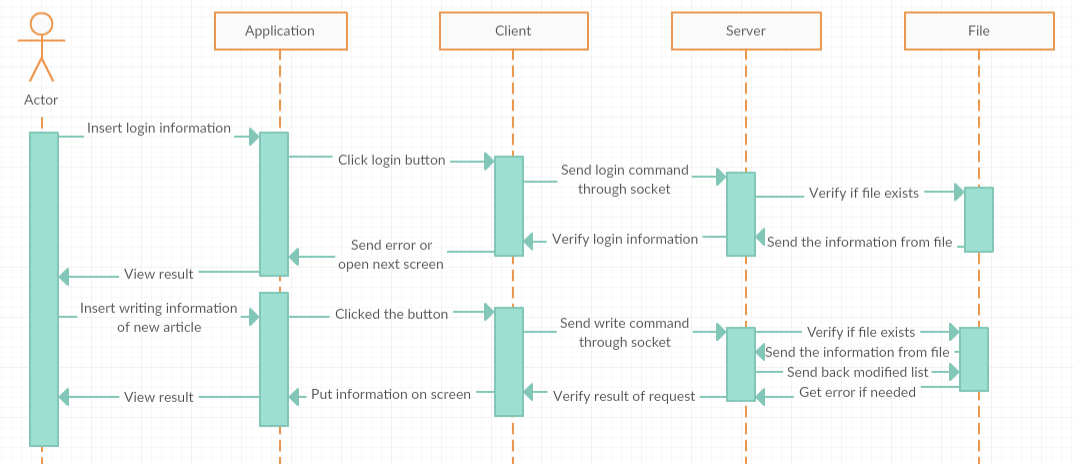


Deployment Diagram



4. UML Sequence Diagrams

The following diagram is a Sequence Diagram for the situation in which you can see the login and writing an article:



5. Class Design

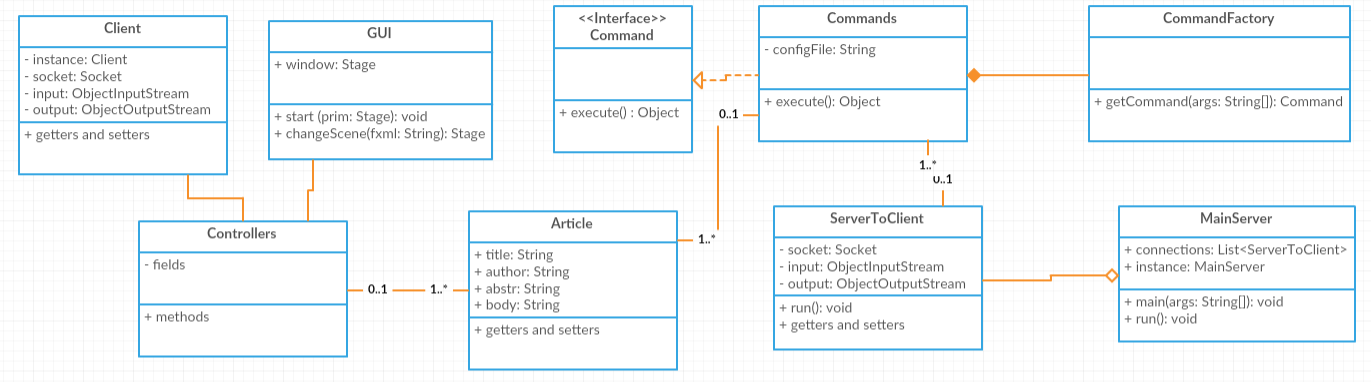
**5.1 Design Patterns Description**

The Observable and Observer objects play an important role in implementing Model-View-Controller architecture in Java. They are typically used in a system where one object needs to notify another about the occurrences of some important changes. Observable is a class and Observer is an interface. They are found in the java.util package as a part of the Java Core Utility Framework. However, the Observable entity found in the JavaFX is an interface and is a part of the javafx.beans package. The basic idea is the same, yet the implementation varies. This article shall try to explain the concepts behind the paradigm and how they are used in Java code with simple examples. [2]

In object-oriented programming, the **command pattern** is a behavioral design **pattern** in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time. This information includes the method name, the object that owns the method and values for the method parameters. [3]

A request is wrapped under an object as command and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command.

**5.2 UML Class Diagram**



6. Data Model

The data model used in this project has the following format:

Writer: username, password

Admin: username, password

Article: author, title, abstract, body, list of related articles

The data model is saved into a json file to which the information is sent and from which it is received.

7. System Testing

At the beginning the testing was done using printing functions (System.out.println()) to see that the data sent and received was correct. Later the testing was done using unit testing (JUnit) which involves testing parts (units) of the code to verify that they work as expected.

8. Bibliography

[1] Client-server: <https://en.wikipedia.org/wiki/Client%E2%80%93server_model>

[2] Observer: <https://www.developer.com/java/data/understanding-java-observable-and-javafx-observable.html>

[3] Command: <https://en.wikipedia.org/wiki/Command_pattern>