News Client-Server application

Analysis and Design Document

Student: Stanciu Alexander

**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

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.

# Functional Requirements

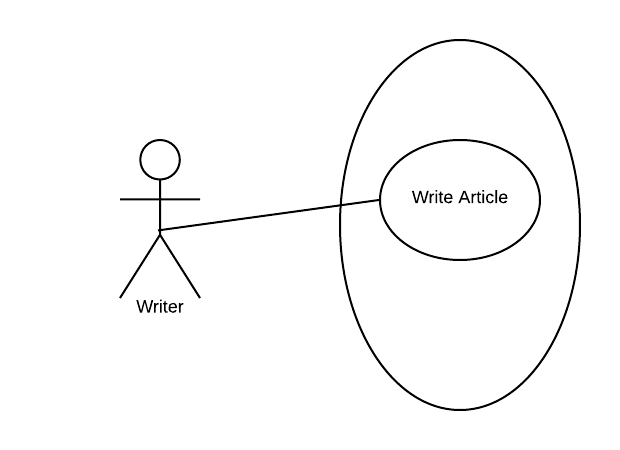
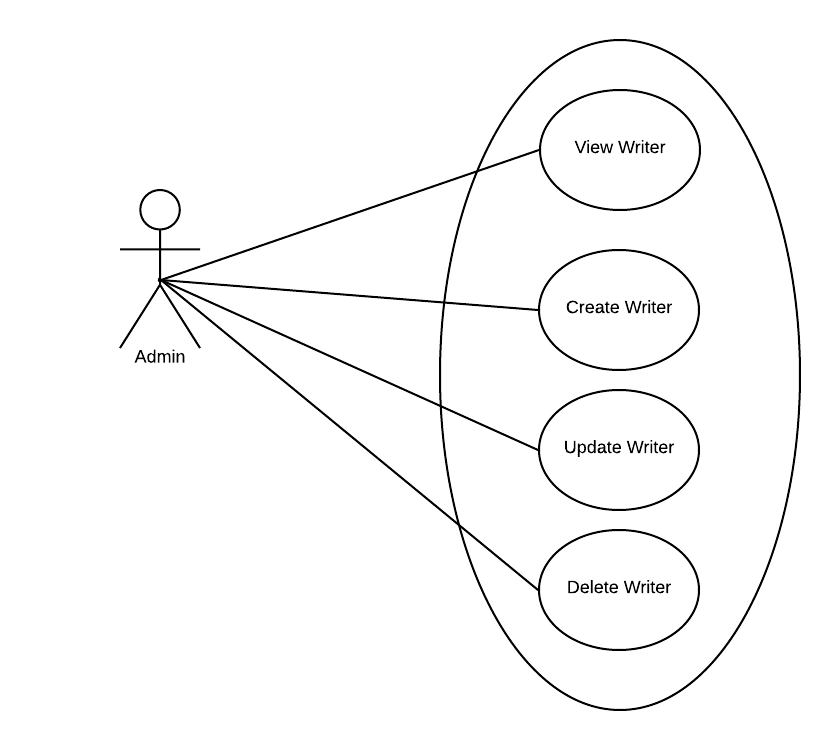
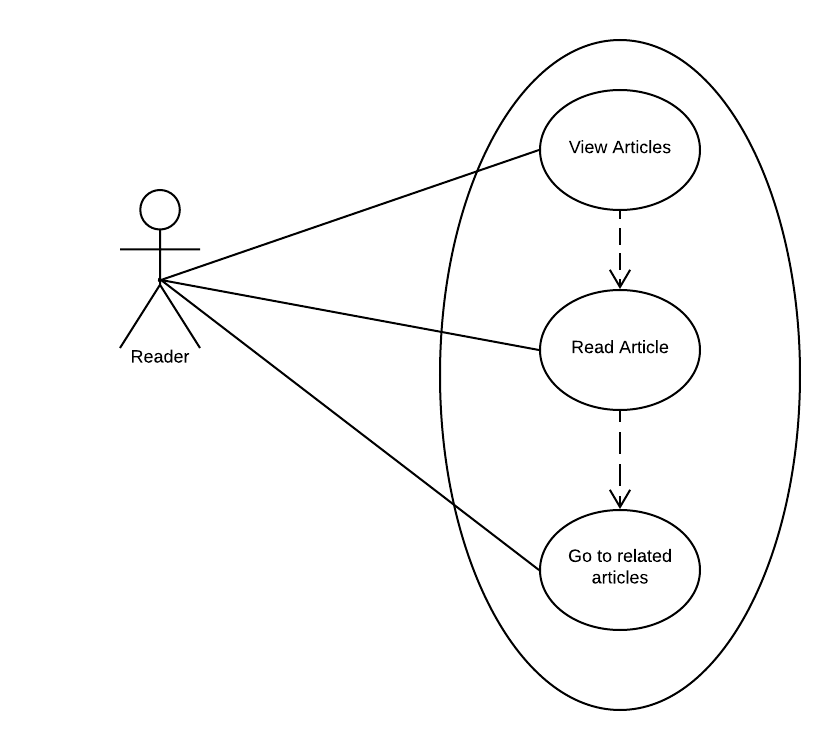
* Writers can write articles after they log in
* When writing an article, show a list that supports multi-select for choosing the related articles.
* Admin can perform crud on writers
* When viewing an article, you can go to the related articles as well.
* The application must be client-server.

# Non-functional Requirements

* 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.

2. Use-Case Model

Use-case Diagram:



Use case: Write Article

Level: writer-goal level

Primary actor: writer

Main success scenario: Introduce a valid article in the server data storage and update the views of everyone connected to the server

Extensions: Introducing anything else than a valid email-password combination results in an error message.

3. System Architectural Design

**3.1 Architectural Pattern Description**

Three-tier architecture is a client–server software architecture pattern in which the user interface (presentation), functional process logic ("business rules"), computer data storage and data access are developed and maintained as independent modules, most often on separate platforms. It was developed by John J. Donovan in Open Environment Corporation (OEC), a tools company he founded in Cambridge, Massachusetts.

**Three-tier architecture:**

**Presentation tier**

This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing and shopping cart contents. It communicates with other tiers by which it puts out the results to the browser/client tier and all other tiers in the network. In simple terms, it is a layer which users can access directly (such as a web page, or an operating system's GUI).

**Application tier (business logic, logic tier, or middle tier)**

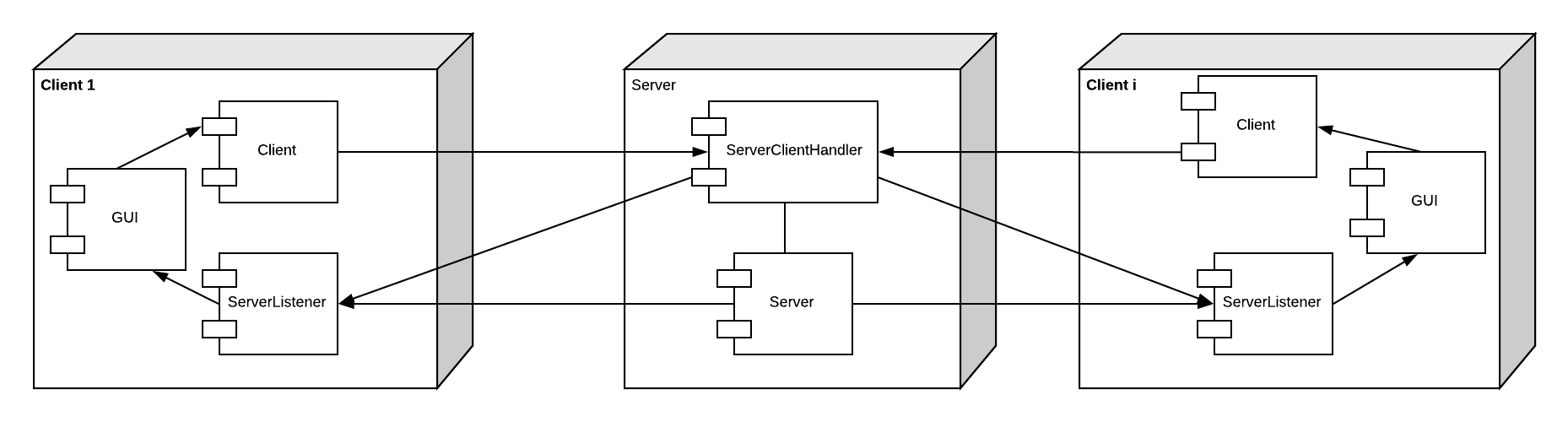
The logical tier is pulled out from the presentation tier and, as its own layer, it controls an application’s functionality by performing detailed processing.

**Data tier**

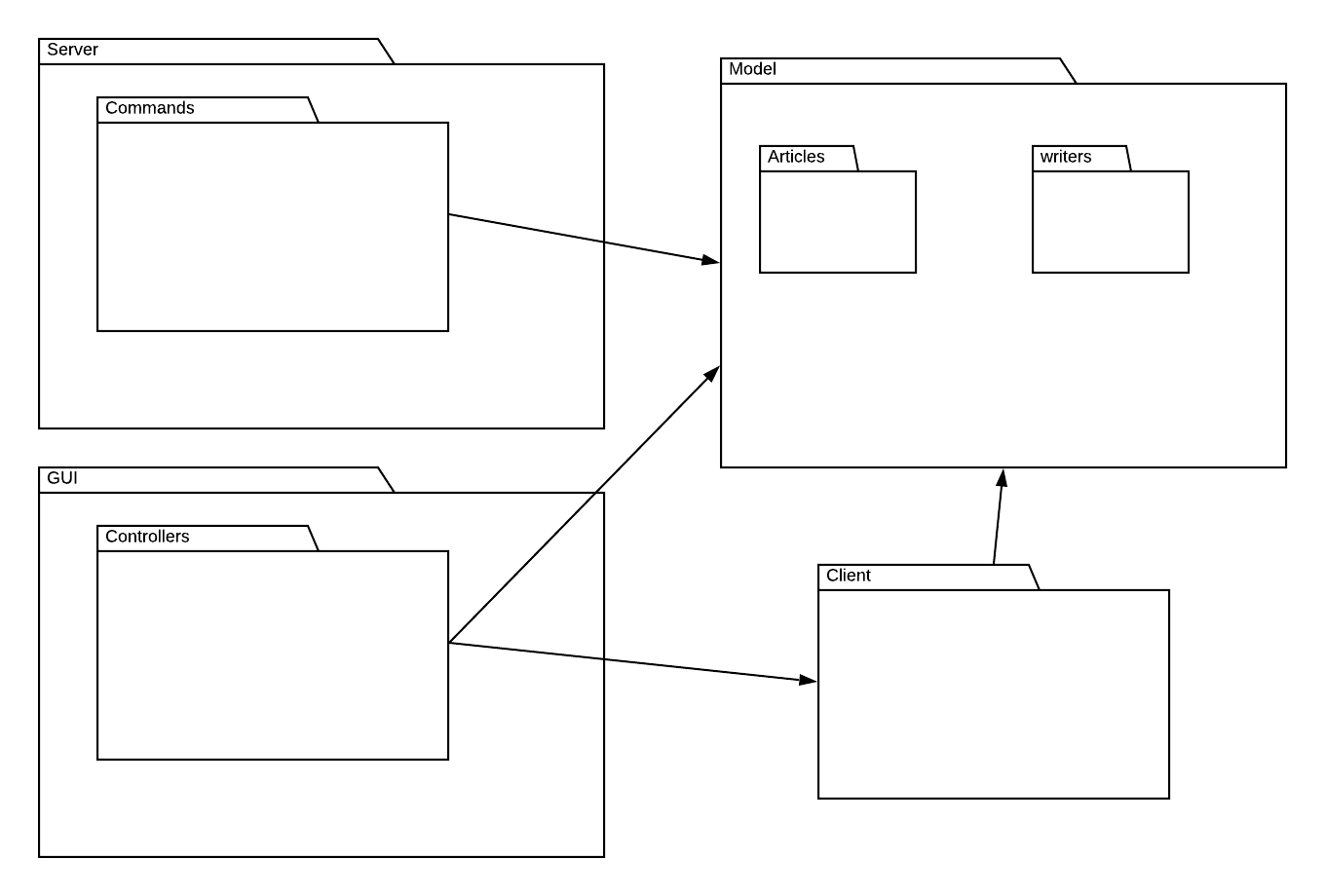
The data tier includes the data persistence mechanisms (database servers, file shares, etc.) and the data access layer that encapsulates the persistence mechanisms and exposes the data. The data access layer should provide an API to the application tier that exposes methods of managing the stored data without exposing or creating dependencies on the data storage mechanisms. Avoiding dependencies on the storage mechanisms allows for updates or changes without the application tier clients being affected by or even aware of the change. As with the separation of any tier, there are costs for implementation and often costs to performance in exchange for improved scalability and maintainability.

**3.2 Diagrams**

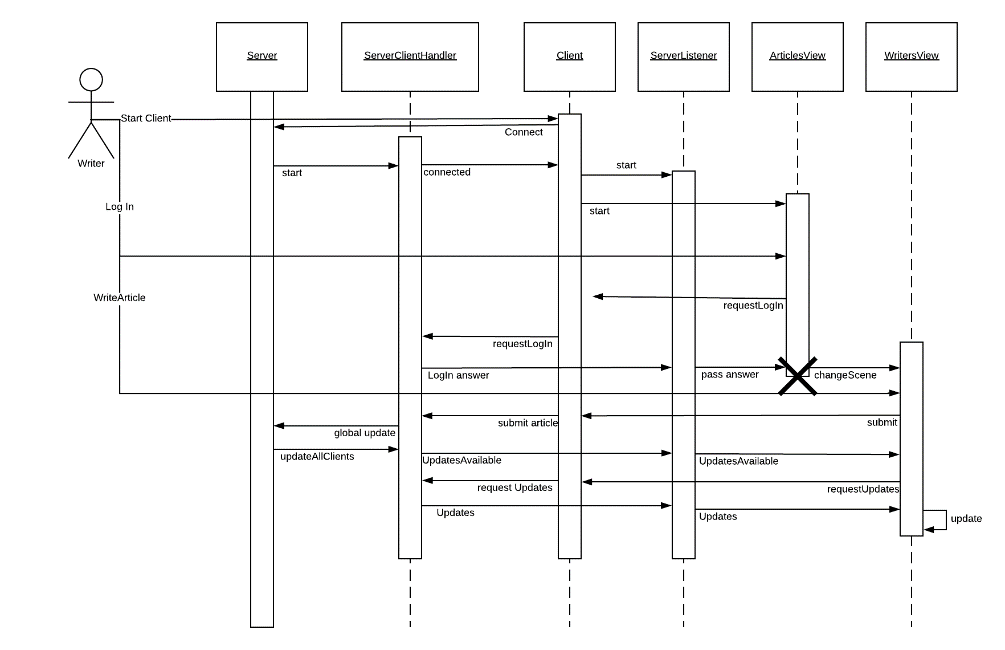
Deployment diagram:



Package diagram:



4. UML Sequence Diagrams

**

5. Class Design

**5.1 Design Patterns Description**

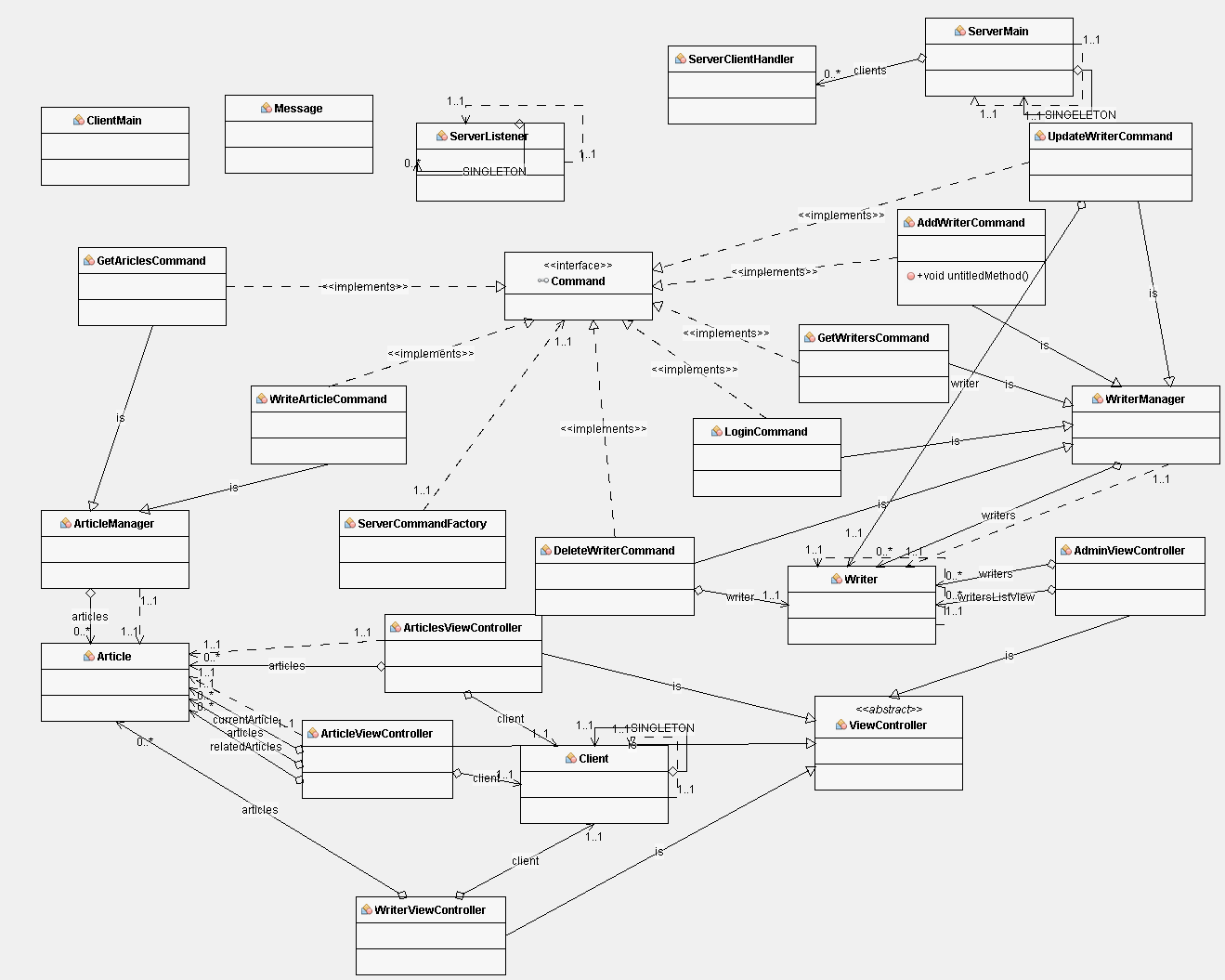
**Mediator pattern**

In software engineering, the mediator pattern defines an object that encapsulates how a set of objects interact. This pattern is considered to be a behavioral pattern due to the way it can alter the program's running behavior.

Usually a program is made up of a large number of classes. Logic and computation are distributed among these classes. However, as more classes are added to a program, especially during maintenance and/or refactoring, the problem of communication between these classes may become more complex. This makes the program harder to read and maintain. Furthermore, it can become difficult to change the program, since any change may affect code in several other classes.

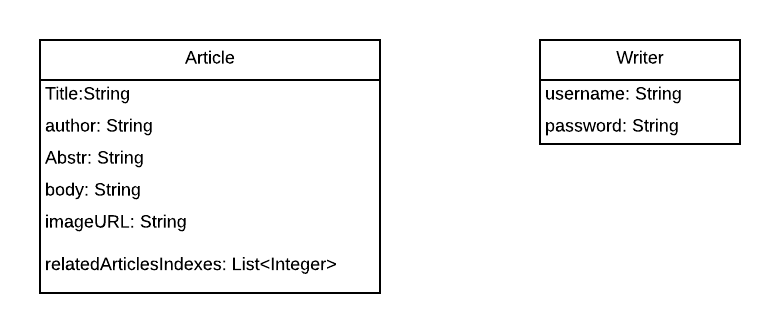
With the mediator pattern, communication between objects is encapsulated within a mediator object. Objects no longer communicate directly with each other, but instead communicate through the mediator. This reduces the dependencies between communicating objects, thereby reducing coupling.

**5.2 UML Class Diagram**



6. Data Model

Database diagram



7. System Testing

Sadly, all the testing was done manually. On the other hand, all the tests were successful.

Tests:

* Increment score works correctly and doesn`t update when it shouldn`t.
* Creates work correctly, but there is no check for valid email
* Scene transitions work correctly between scenes
* End match and end game are detected correctly

8. Bibliography

1. <https://en.wikipedia.org/wiki/Multitier_architecture>
2. <https://en.wikipedia.org/wiki/Mediator_pattern>
3. <https://stackoverflow.com/>
4. https://github.com/buzea/SoftwareDesign2018