Readers-Writers Application

Analysis and Design Document

Student: Albert Erika-Time

**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

The aim of this assignment was to design and implement a software that supports the operations of a news agency. The application needs to handle user request for several use cases (changing personal data, publishing articles, reading, deleting, editing them) by sending them to the server, which, based on the received information, produces an appropriate response that can be processed by the user (client). The user can be of two types – writer or reader. The writer is able to perform CRUD operations on his/her own account and read other articles, while the reader is only able to open one particular article and read it.

One important aspect of this assignment was to familiarize the developer with the Client-Server architectural pattern, as well as with the Observer design pattern, as the updates from one user’s operations propagate to all the others via the notification of the observers.

# Functional Requirements

1. Log In: The legitimate writer can log in to perform later operations.

2. Create profile information: The writer can add personal information of himself/herself.

3. Manage profile information: The writer can change any personal data.

4. Publish article: The writer is able to publish an article by entering its title, abstract and body.

5. Read own articles: By clicking on an article title, the writer can read it.

5. Edit own article: By clicking on an article title, the writer can edit its body.

6. Delete own article: By clicking on an article, the writer can delete it.

7. Read any article: Both the general reader and the writer can read any article but not make any changes to them.

# Non-functional Requirements

1 Availability: the system needs to be available as much as possible since both readers and writers need to use it frequently, the percentage of time when the system is down should be as small as possible

1. Performance: the application needs to be performant as well; has to respond in time to the user request – there may be time periods when the system will run in overload mode, as too many requests are arriving (e.g. many writers publishing at the same time and readers trying to open them)
2. Security: in case of this system we need to distinguish between writers and general users; twriters are authorized for editing their own articles or delete them, but readers should not have access this area. The application should be able to prevent reader access to this section but should allow writers to enter it.
3. Testability: the system is kind of complex and I need to design it in a way to be able to gradually test each component (by unit tests) but also the whole product when ready (user testing)
4. Usability: the whole application should be user friendly, not overcomplicated and should make it easy for the user to perform specific tasks. It should provide features that help avoiding errors when using it, efficient usage and t has to obtain user satisfaction.

2. Use-Case Model

Use case: Publish Article

Level: user-goal level

Primary actor: Writer

Main success scenario:

1. writer enters title
2. writers enters abstract
3. writer enters article body
4. writer clicks on publish button
5. article gets inserted in the database
6. all the active users are notified about the change

Extensions:

* writer doesn’t enter any data, initial data is inserted

A close up of a map

Description generated with high confidence

3. System Architectural Design

**3.1 Architectural Pattern Description**

For this assignment we had to use the Client-Server architectural pattern, where the clients (in our case the readers and the writers) are sending the requests to the server, that is receiving the message and is responding by sending back the requested data. To ease up the communication between the client and the server we used sockets that are the endpoints of this two-way communication. The messages are sent in form of a JSON string by serializing the data to be sent. Furthermore, I also tried to keep the layered architecture, mostly for the server, that accesses the data. This way the code is more organized and it is easier to maintain and understand.

*A close up of text on a white background

Description generated with very high confidence*

**3.2 Diagrams**

**A screenshot of a social media post

Description generated with very high confidence**

**A screenshot of a cell phone

Description generated with high confidence**

4. UML Sequence Diagrams

**Publish Article**

**A screenshot of a social media post

Description generated with very high confidence**

5. Class Design

* 1. **Design Patterns Description**

1. Observer Design Pattern: One requirement of the assignment was to use the observer design pattern. Once one modification is done to the article table (insertion, update, deletion), the observers (the threads on with which all the separate running clients are communicating) are notified.
2. Singleton Design Pattern: To be able to add observers to the article service, one single article object needs to be created. The same procedure needs to be applied on the connection creation.
3. Factory Method Design Pattern: Initially I was running my application on hibernate connection, then I modified it to jdbc connection. I still left the hibernate connection creating in my application, so later on I would be able to extend the application.
   1. **UML Class Diagram**

**A close up of a map

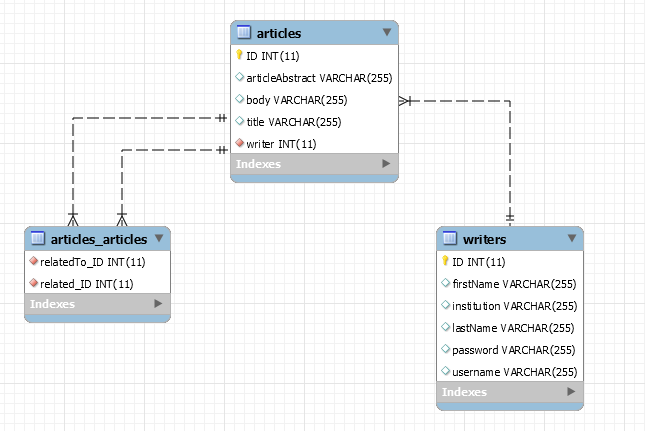
Description generated with very high confidence**

**A close up of a map

Description generated with very high confidence**

6. Data Model

In my application the data model consists of a table for the writers and one for the articles. Between the writers and the articles there is a one-to-many relationship (one writer can have many articles), while there is a many-to-many relationship of the article table with itself in case someone would like to add related articles to another one.



7. System Testing

The way I tested the developed application was User Acceptance testing. Basically, I gradually developed every feature. Once I reached the UI level with it, I connected it to the business logic and I tested it the way a normal user would do it. I was putting the application in real world situations, trying to generate an error in the software. When I met an error (exception) I was tracing it back right until I found what caused it and I corrected it.

I was following this above-mentioned procedure until I reached the final solution.

8. Bibliography

* <https://en.wikipedia.org>
* <https://stackoverflow.com>
* <https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html>
* <https://www.tutorialspoint.com/design_pattern/observer_pattern.html>