Student: Șinca Mădălina
Group: 30433

Table of Contents

1. Requirements Analysis

1.1 Assignment Specification

1.1.1 Objective

The objective of this assignment is to allow students to become familiar with architectural patterns.

1.1.2 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.

1.1.3 Application Constraints

- 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.

1.1.4 Requirements

- Create the analysis and design document (see the template).
- Implement the application.

1.1.5 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:
  o how to install your application on a clean computer
  o how to access your application and with what users
  o images with all use cases and their scenarios implemented

1.2 Functional Requirements

The functional requirements for this project are:

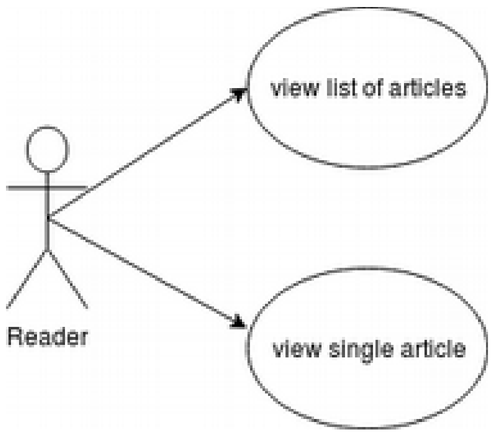
- Login for both Writers and Admins
- View articles
- View matches
- Update article list in real-time
- Implementing using client-server architecture

1.3 Non-functional Requirements

The non-functional requirements for this project are:

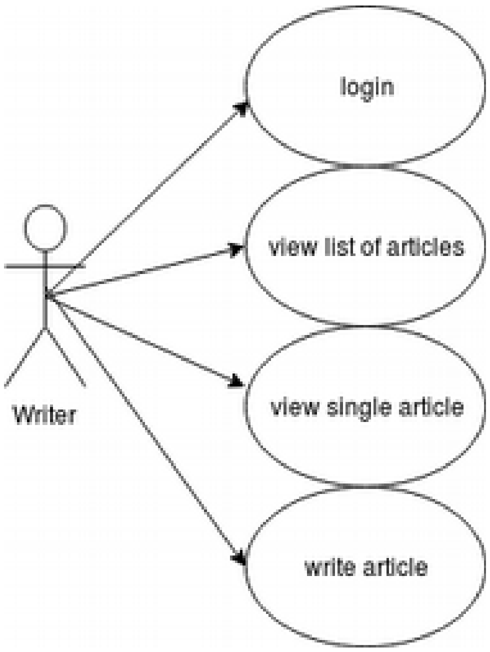
- Performance: a response time of maximum 2 seconds
- Availability: 16hours/day
- Security: the passwords will be encrypted when sent to the database
- Testability: hardest part to test is the ending of the match/game and update score, this is done by both printing out results and Junit tests
- Usability: the application will work on any computer that has java installed

2. Use-Case Model



Use case: View Articles
Level: reader-goal level
Primary actor: Reader
Main success scenario:

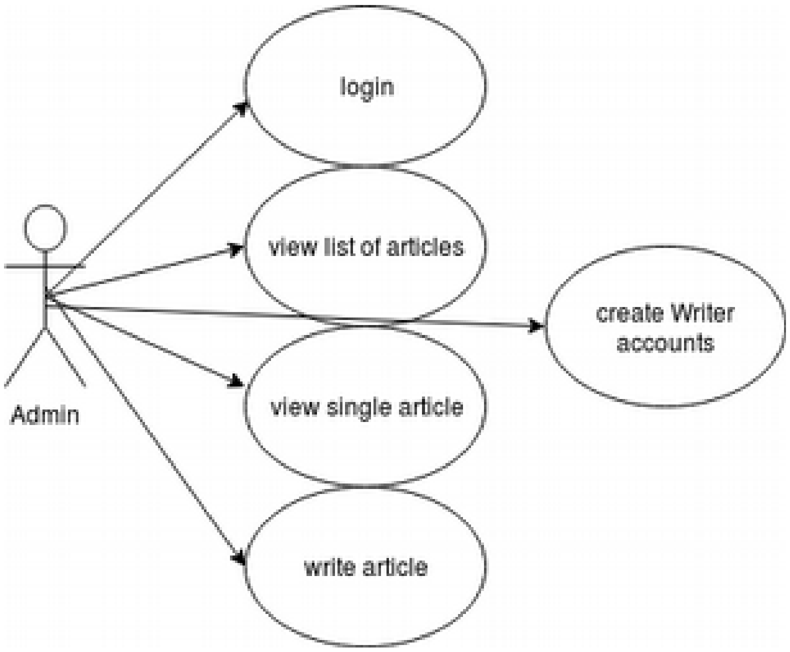
- open application
- user pane opens
- choose the view articles button and click it
- view the articles



Use case: Write Articles
Level: writer-goal level
Primary actor: Writer
Main success scenario:

- open application
- login
- writer pane opens
- choose the write articles button and click it

- write article in textbox
- publish article



Use case: Create Writer Account
Level: admin-goal level
Primary actor: Admin
Main success scenario:

- open application
- login as admin
- admin pane opens
- choose the create writer account button and click it
- enter credentials in textboxes
- create account

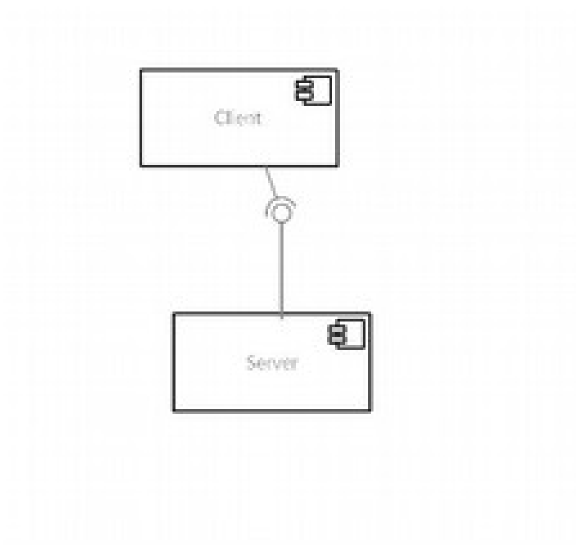
3. System Architectural Design

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.

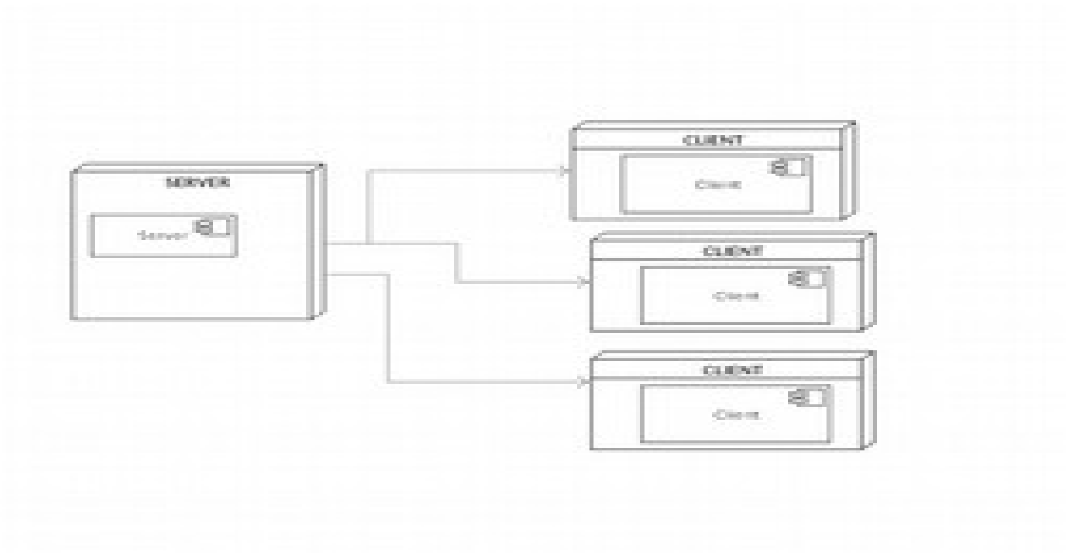
3.2 Diagrams

Component Diagram



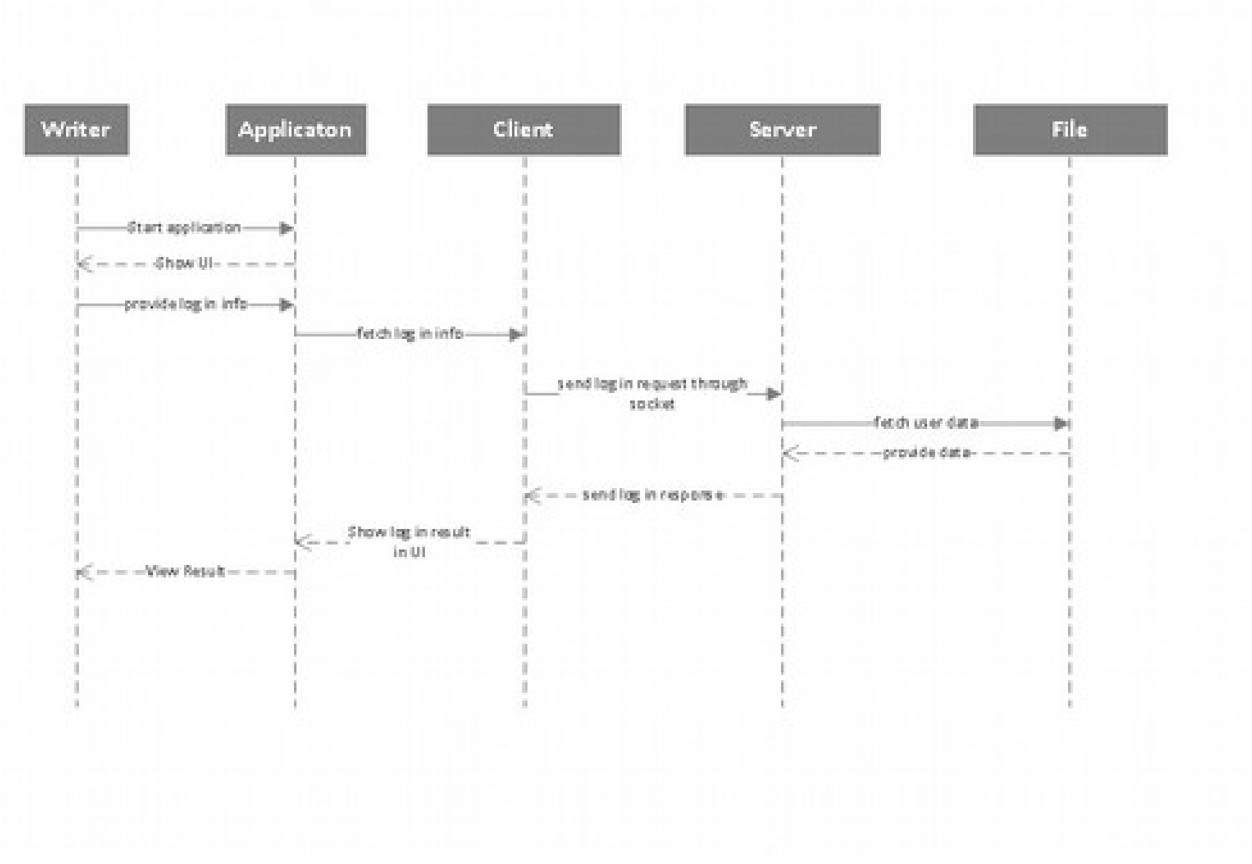
Deployment Diagram

4.



UML Sequence Diagrams

The following diagram is a Sequence Diagram for when a writer wants to log-in into a new session:



5. Class Design

5.1 Design Patterns Description

The three-tier architecture is an architecture in which the functional process logic, data access and user interface are developed and maintained as independent modules on separate platforms. Three-tier architecture is a software design pattern and a well-established software architecture.

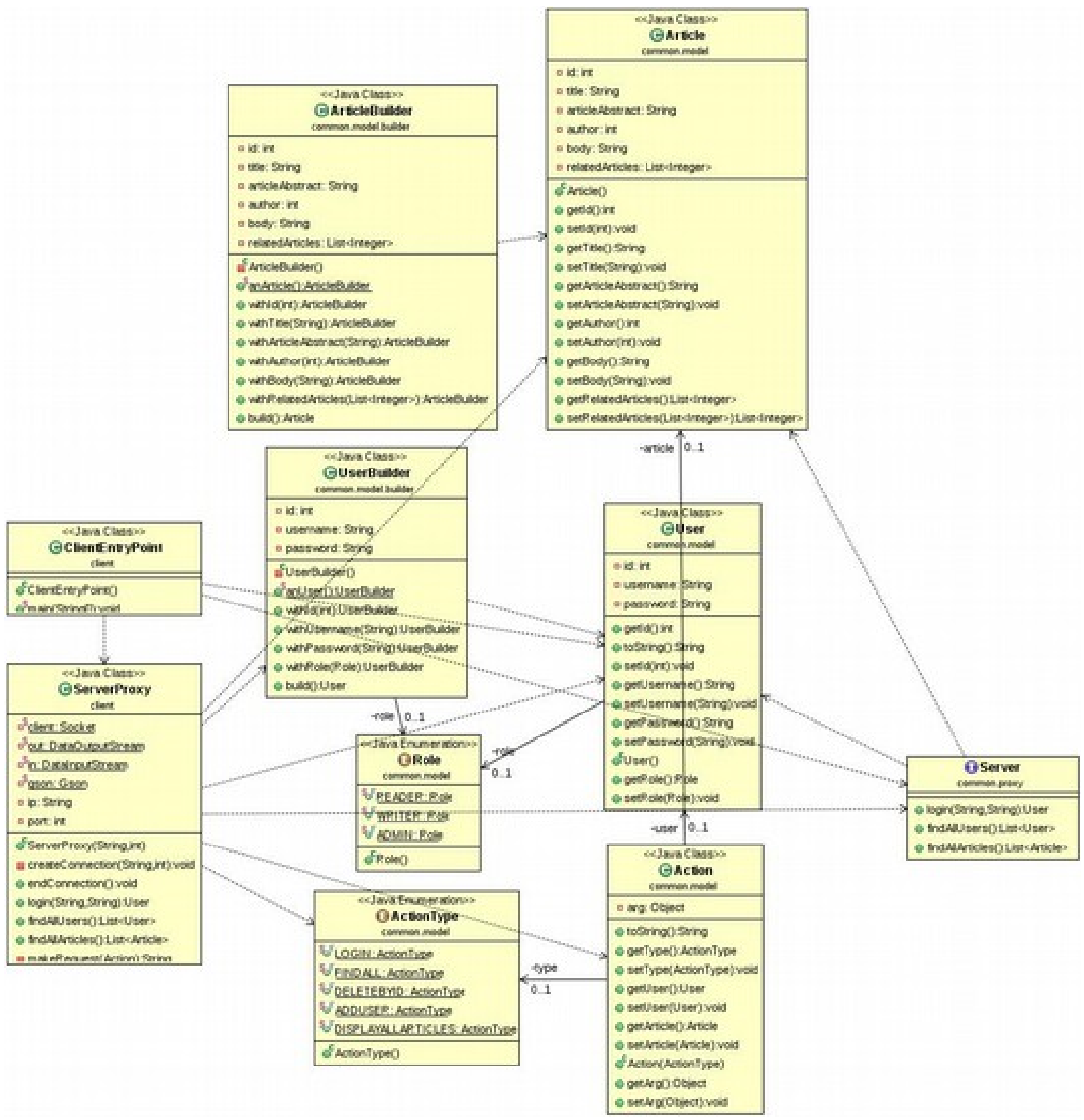
Data Access Object Pattern or DAO pattern is used to separate low level data accessing operations from high level business services. Following are the participants in Data Access Object Pattern.

- **Data Access Object Interface -** This interface defines the standard operations to be performed on a model object(s).
- **Data Access Object concrete class -** This class implements above interface. This class is responsible to get data from a data source which can be database / xml or any other storage mechanism.
- **Model Object or Value Object -** This object is simple POJO containing get/set methods to store data retrieved using DAO class.

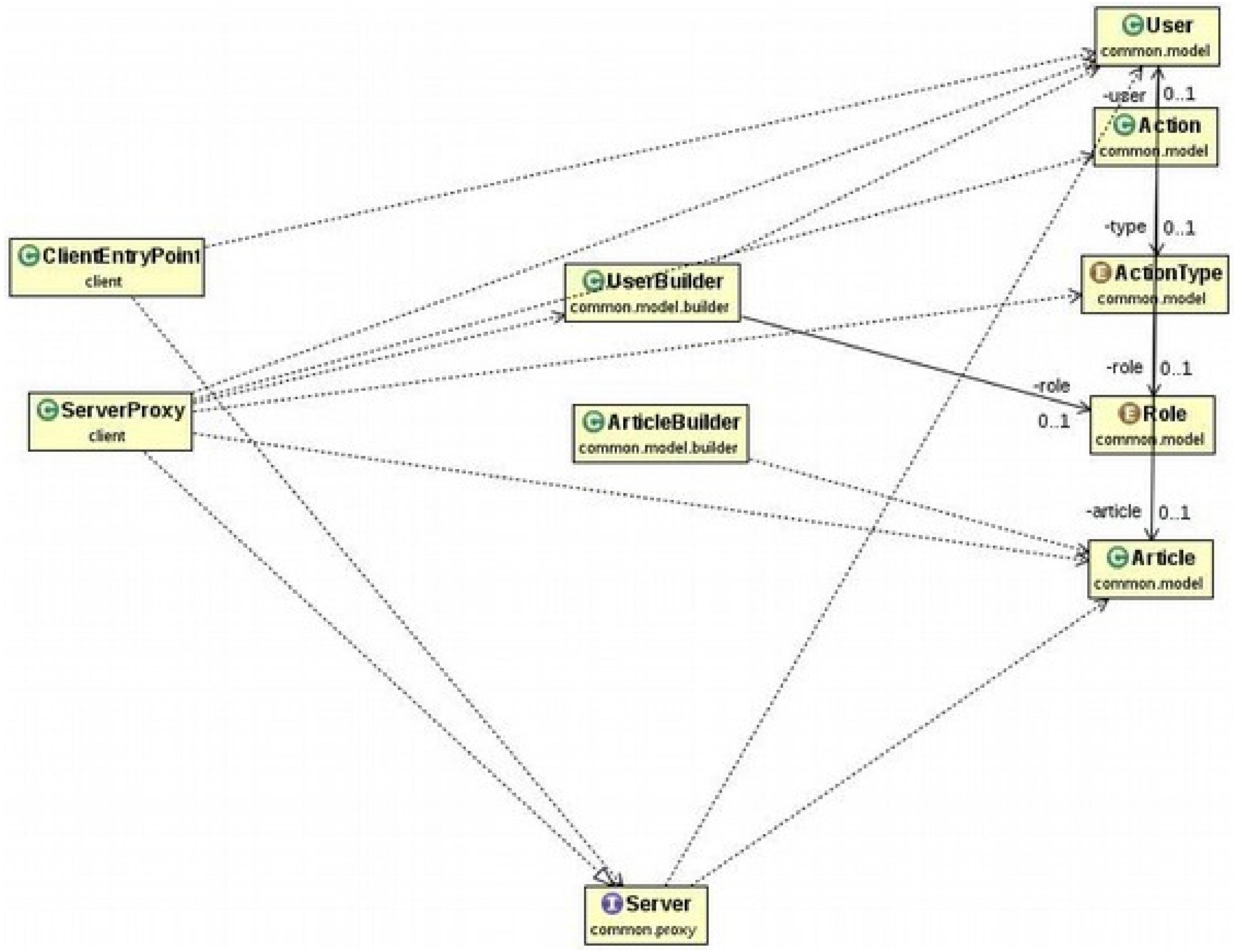
A Domain Model Design Pattern is one that creates a web of interconnected objects, where each object represents some meaningful individual, whether as large as a corporation or as small as a single line on an order form.[3]

5.2 UML Class Diagram

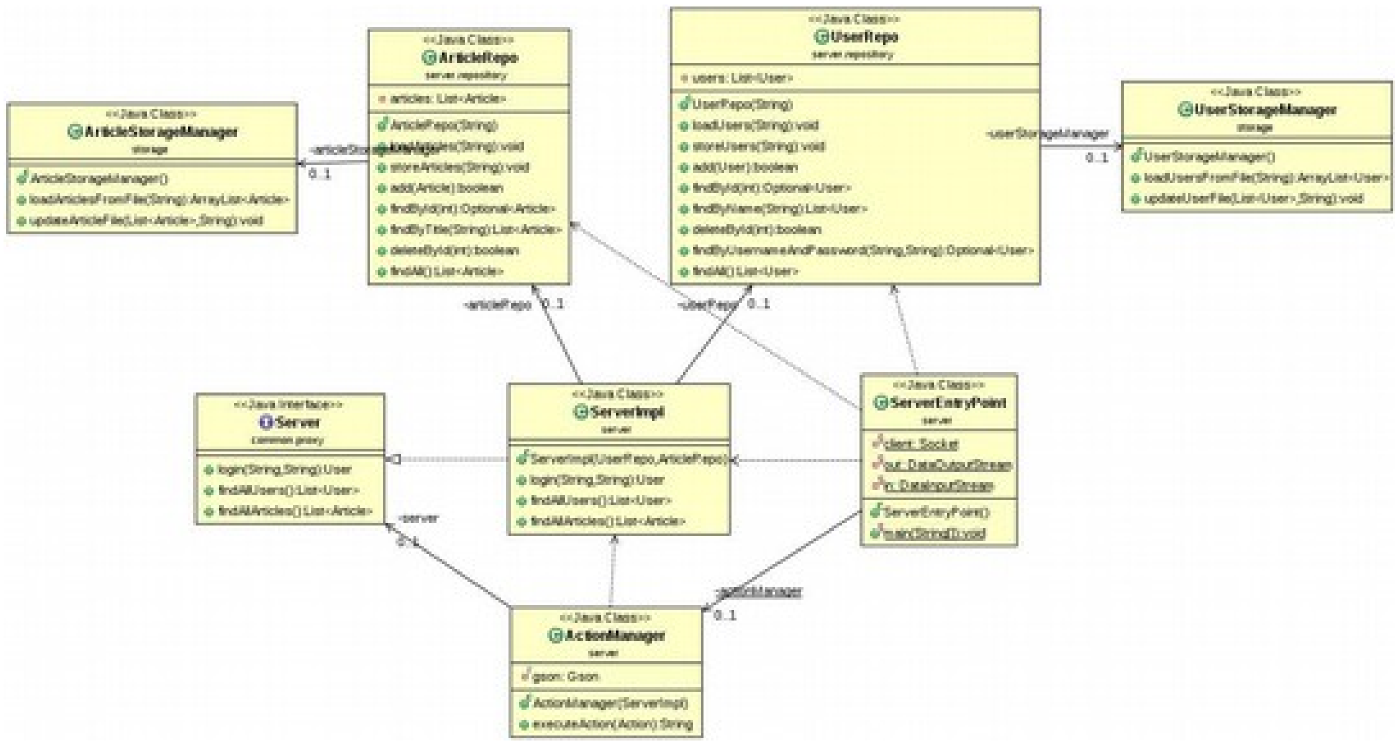
Client side



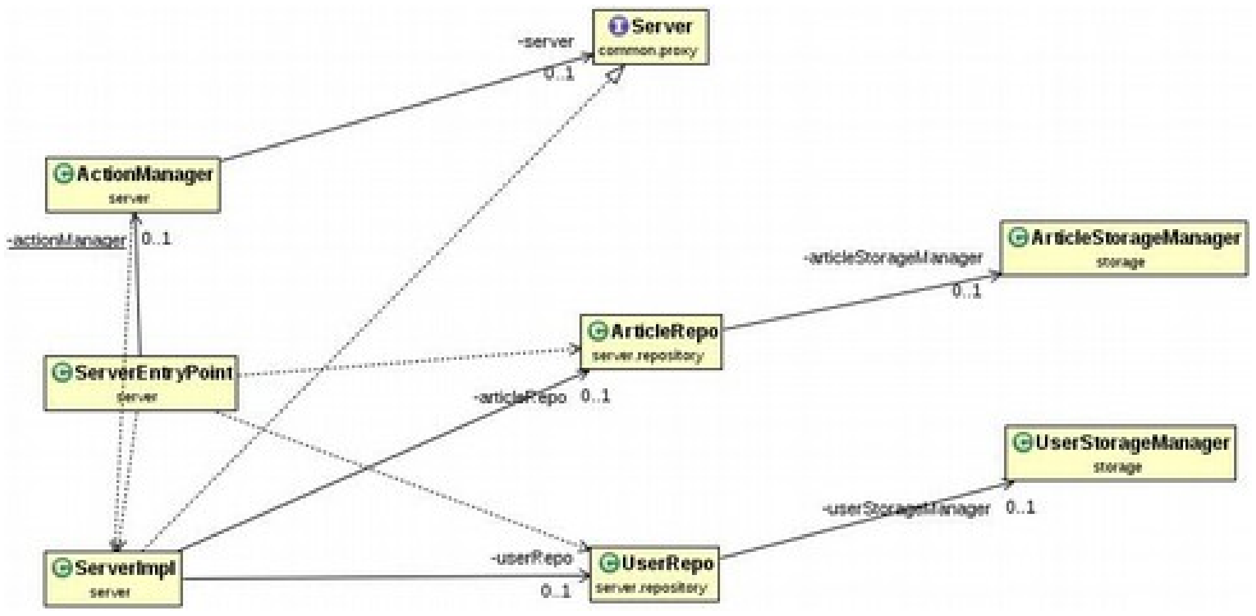
simplified version


User
common.model
-user
0..1
Action
common.model
-type
0..1
ActionType
common.model
-role
0..1
Role
common.model
-role
0..1
-article
0..1
Article
common.model
ClientEntryPoint
client
UserBuilder
common.model.builder
ServerProxy
client
ArticleBuilder
common.model.builder
Server
common.proxy

server side



simplified version



6. Data Model

For this project, the data was modeled in the following way:
reader: id, password, name
Admin: id, password, name
writer: id, password, name
Article: id, name, body, abstract, title

All models have id to be able to look for instances more easily, the id is the primary key in the database.
Admin and Writer have username and password for login.
Articles have a title, abstract, body, list of related articles, etc.

.

7. System Testing

Testing was done using flags placed during development and constantly checking the Json file for updates. The System.out.println() method was used to check various functionalitites.

8. Bibliography

[1] - https://docs.microsoft.com/en-us/dotnet/standard/events/observer-design-pattern
[2] - https://en.wikipedia.org/wiki/Client–server_model