Show Management

Analysis and Design Document

Student:Bratean Larisa

**Group:30235**

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

We want to design and implement a client-server application for managing online show visualization such as movies, theatre performances and sport events. The application has three types of users: the basic user, the premium user and an administrator.

# Functional Requirements

The basic user can perform the following operations:

* Search show, select a show and view details of a show
* View history of all shows he has seen
* Give a rating to the show
* Add a comment to the show
* Add interests in a show he wants to see when it will be uploaded on the site and receive notification from application that the show was uploaded so that he can watch it

The administrator can perform the following operations:

* CRUD on shows (for ex. movie information: name, description, actors, release date, imdb rating).
* CRUD on user accounts.

# Non-functional Requirements

* + 1. **Availability**

The system is available 100% for the users of the application, 24 hours a day, while the Java application is running.

* + 1. **Performance**

The application it s develop to work really fast, answer fast when the regular user or administrator tries to make an operation, no matter if they try to add, delete, view or update something.

* + 1. **Security**

The application it’s safe, because both the administrators and the regular users have to log in before entering it. They have to enter an username and a password, which is encrypted with the special character \*.

* + 1. **Usability**

The system allows users to access the application using Java, so it’s a desktop application. Nobody needs training to use it, because we run it and the interface it will open, and you just need to log in. The system it’s friendly and simple.

# 2. Use-Case Model

**2.1. User**

The user has the purpose to log in successfully in the application, and make operations based on shows dates. He can also see all the shows he watched.

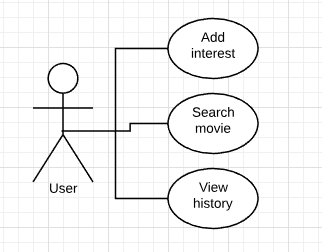
A successful scenario would be: the employee is trying to log in the application, and succeeds.

A success-failure scenario would be:

* Preconditions: the employee tries to log in to see some product
* Postconditions:

Success : the employee logs in the application, and see the products

Failure: the employee can’t log in the application, because he made a mistake on his password

****

**2.2. Admin**

The administrator has the purpose to log in the application, and make different operations based on the information about users and shows.

He has an important role for taking care of the application.

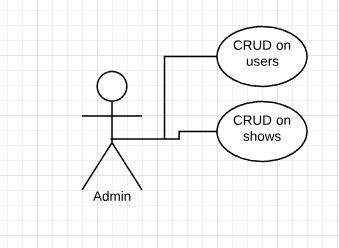
A successful scenario would be: the administrator logs in the application, and wants to delete a user. He writes the id of the show and a deletes the user saved in the database.

A success-failure scenario would be:

* Preconditions: the administrator tries to log to delete an user.
* Postconditions:

Success: the administrator will log in the application, and will delete an user from the application

Failure: the administrator has an error while he tries to log in, because he made a mistake with the password.



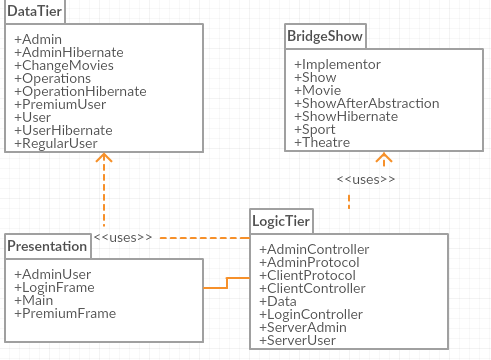
3. System Architectural Design

**3.1 Architectural Pattern Description**

 The typical client-server architecture consists of a server application running on one machine, and one or more clients -- often, but not always, operating on different machines across a network. The server is modeled as a perpetual process that waits for clients to connect (request a service), and then processes the client requests. Java threads are ideal to the implementation of a server application, because a new thread can be started for each connecting client. This prevents one client request from holding up others if and when it enters a wait state, or some time-consuming task.

Everytime a regular user or the admin it’s going to log in in the application, they will become the clients. After they start, we will start the sever also, and they will communicate. The client will ask for data, and the server will give it to the client.

**3.2 Diagrams**

****

5. Class Design

**5.1 Design Patterns Description**

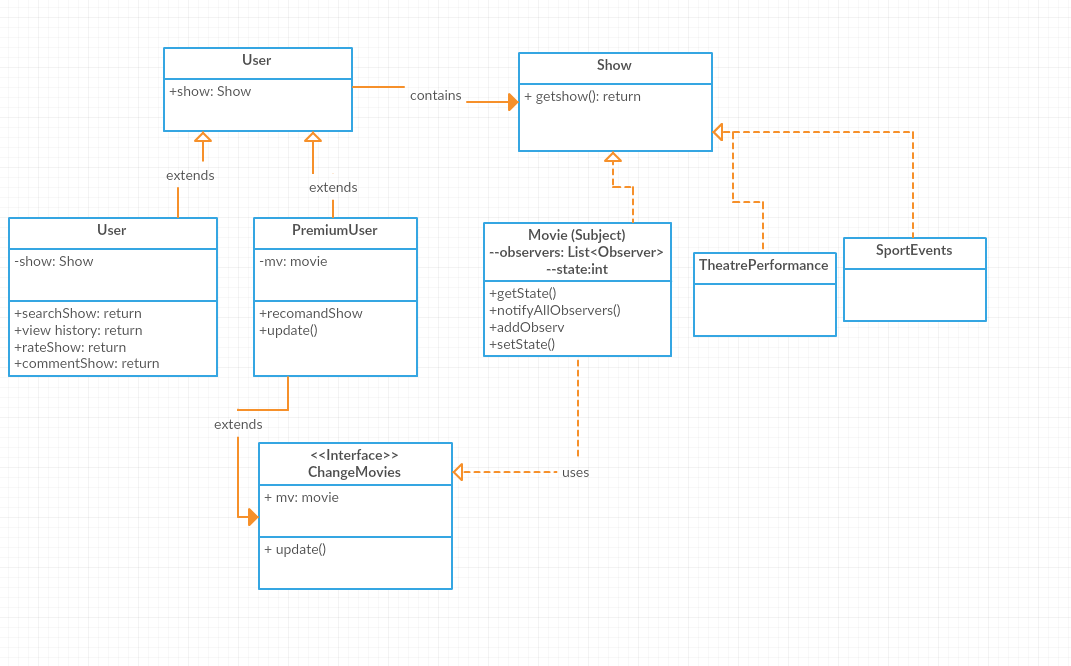
The design pattern that I used for this project is Bridge Design Pattern. Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under structural pattern as this pattern decouples implementation class and abstract class by providing a bridge structure between them.

This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.

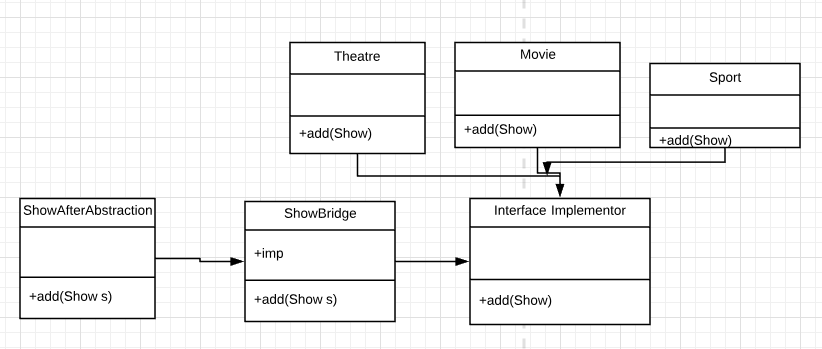
In my project,the abstraction is represented by ShowBridge, and I have the Interface Implementor, where I have the method addShow. I also have the concrete classes Movie, Theatre, and Sport that will implement that interface. Therefore by using the abstract class the implementation is hided from the user.In order to use one of the implementation, the user has to use the ShowAfterAbstraction class to which it has to pass the show he wants to add and a concrete type of implementor of the showo it wants to add.

I also user the Observer Design Pattern, and everytime the show will change, the user will be updated.

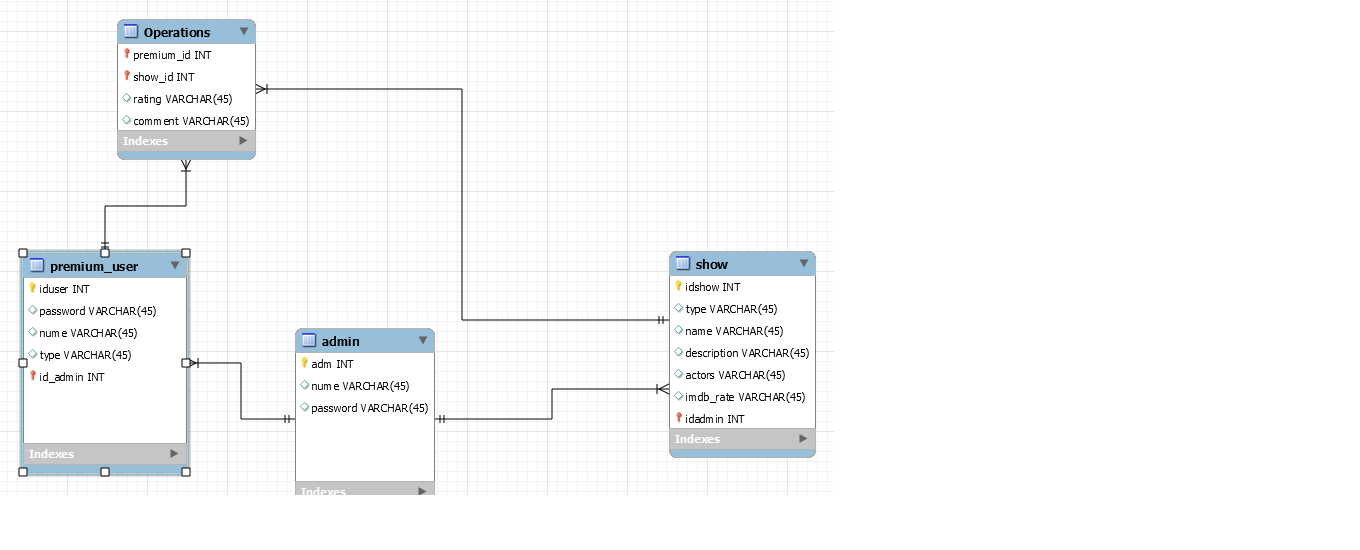
**5.2 UML Class Diagram**



Ilustrating more the bridge design pattern:



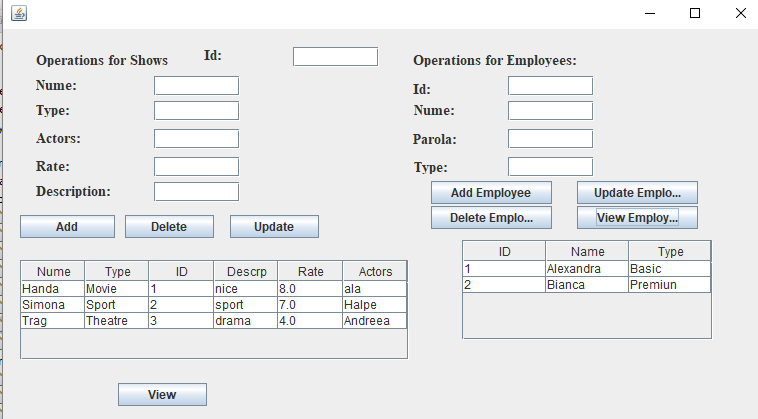
6. Data Model

**

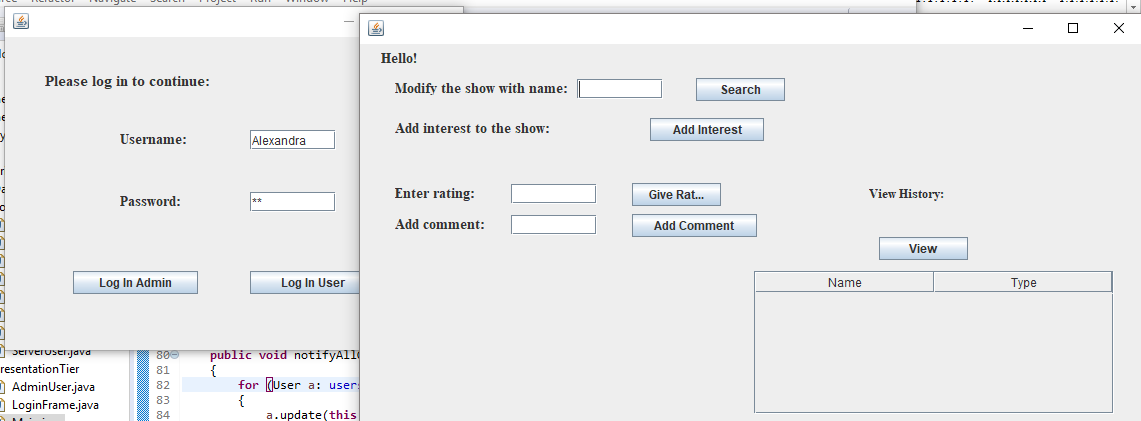
7. System Testing

I made more scenarios:

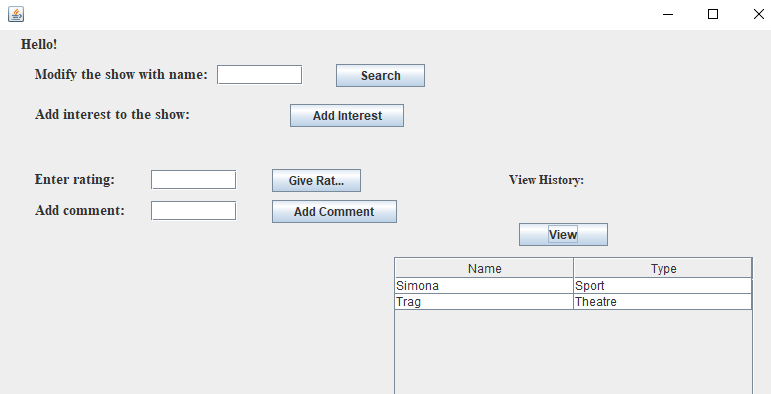
7.1 View all all shows and all users



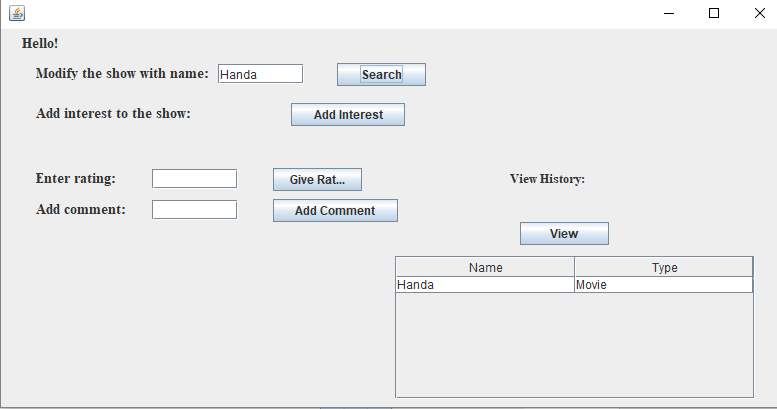
7.2 Login User



7.3 View history movies



7.4. Search a show by name



8. Bibliography

<https://www.tutorialspoint.com/design_pattern/bridge_pattern.htm>

<https://docs.oracle.com/javase/tutorial/networking/sockets/clientServer.html>

https://www.javaworld.com/article/2077234/java-se/generic-client-server-classes.html