Online show visualization

Analysis and Design Document

Student: Pali Anamaria

**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

Use Java/C# API 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

The premium user can perform the following operations:

* All operations from basic user
* Recommend a show to a friend or a group of friends who also have accounts on the site and are premium users (the recommendation will also appear as a notification on the friends page)
* 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.

In addition, when a new show is uploaded on the site and there are users interested in that show the application should inform all the interested users about that show by sending them an update about the show and let them know they can watch it.

# Non-functional Requirements

The application should be client-server and the data will be stored in a database.

Use an ORM (hibernate) for database operations.

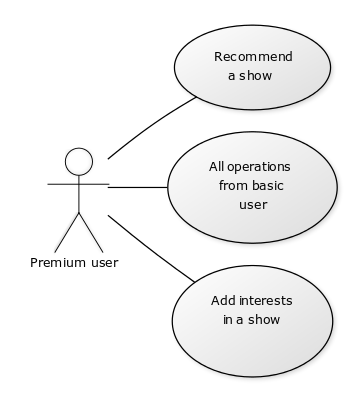
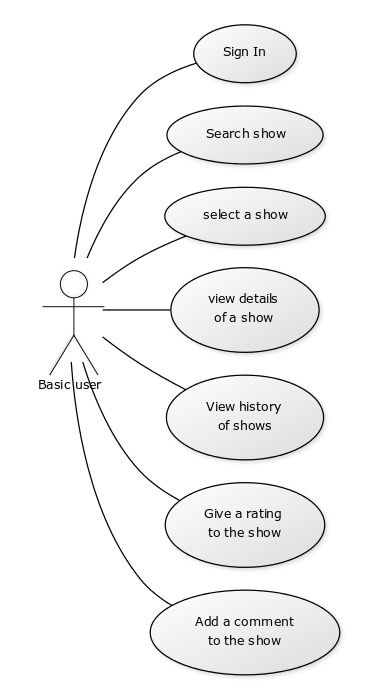
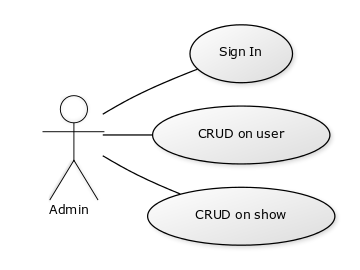
Use the Observer design pattern for notifying the users when the movies they are interested in have been uploaded.

Use the bridge design pattern to implement user relationship and show relationship. (or you can implement a different scenario).

All the inputs of the application will be validated against invalid data before submitting the data and saving it.

Use Spring (optional)

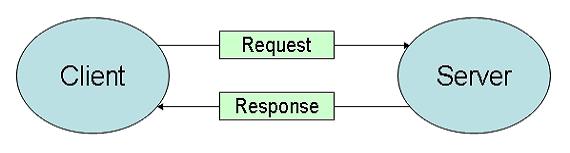
2. Use-Case Model



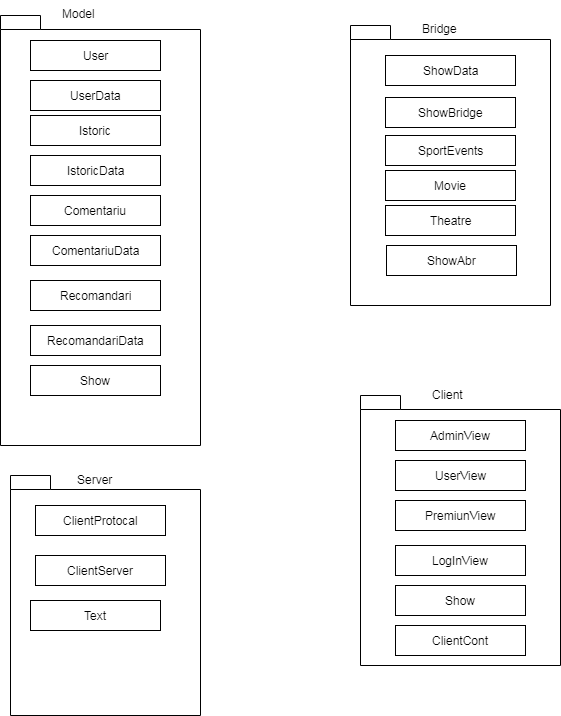
3. System Architectural Design

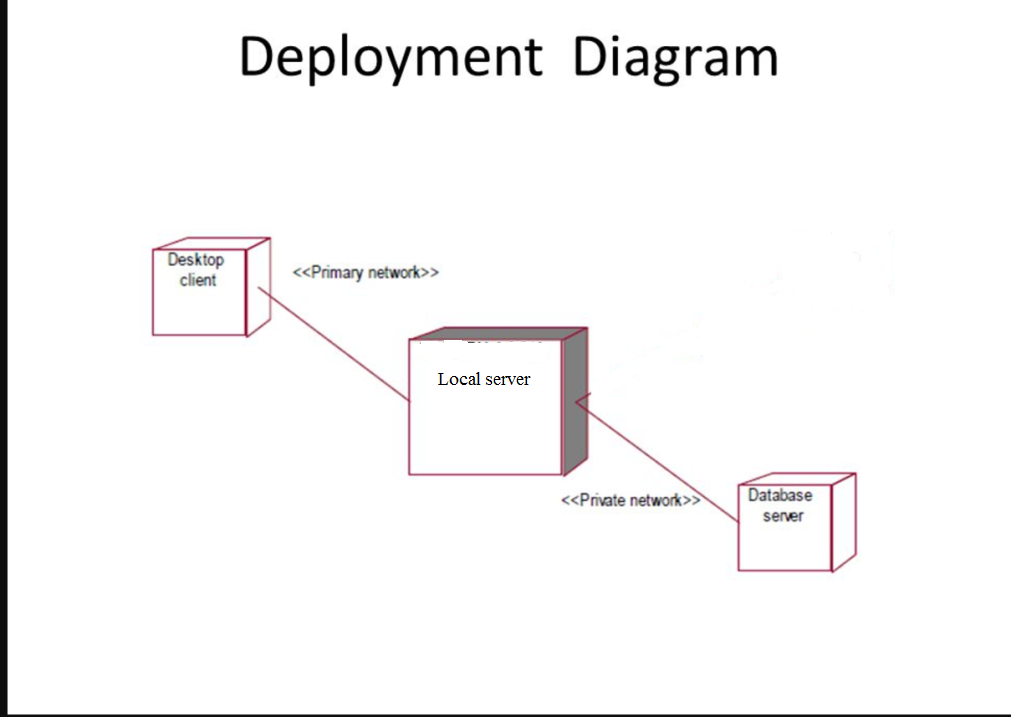
**3.1 Architectural Pattern Description**

I used 4 packages, one for the server part ,one for the operations with the database and one for the client part where are the GUI classes, in which I have the connection with the server and the requests, and one for the bridge pattern.

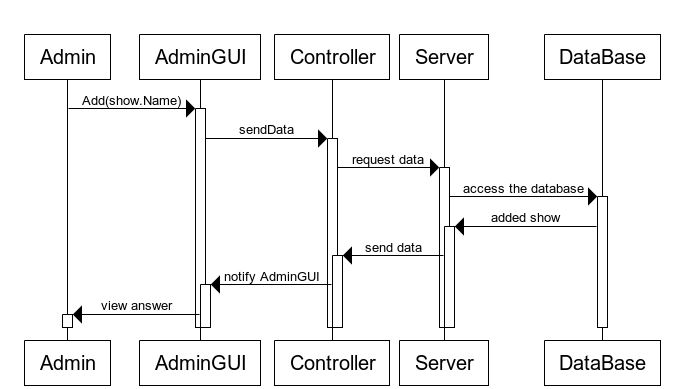


**3.2 Diagrams**

****



4. UML Sequence Diagrams



5. Class Design

**5.1 Design Patterns Description**

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

Observer pattern uses three actor classes. Subject, Observer and Client. Subject is an object having methods to attach and detach observers to a client object. We have created an abstract class Observer and a concrete class Subject that is extending class Observer.

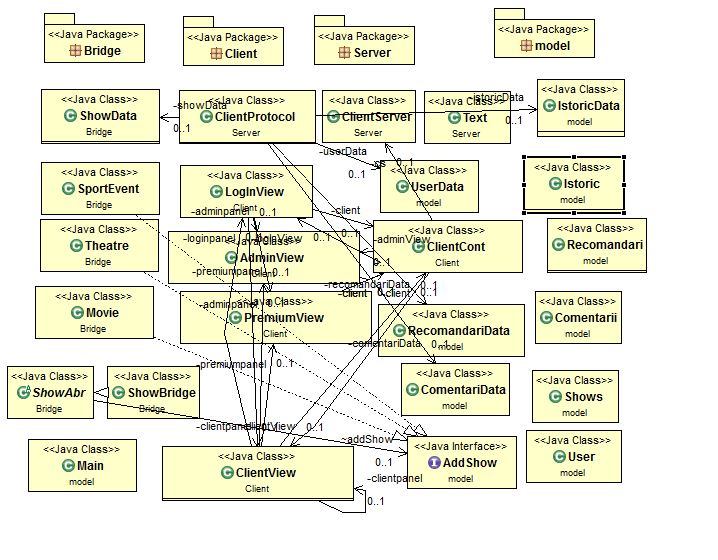
The bridge pattern allows the Abstraction and the Implementation to be developed independently and the client code can access only the Abstraction part without being concerned about the Implementation part.

The abstraction is an interface or abstract class and the implementor is also an interface or abstract class.

The abstraction contains a reference to the implementor. Children of the abstraction are referred to as refined abstractions, and children of the implementor are concrete implementors. Since we can change the reference to the implementor in the abstraction, we are able to change the abstraction’s implementor at run-time. Changes to the implementor do not affect client code.

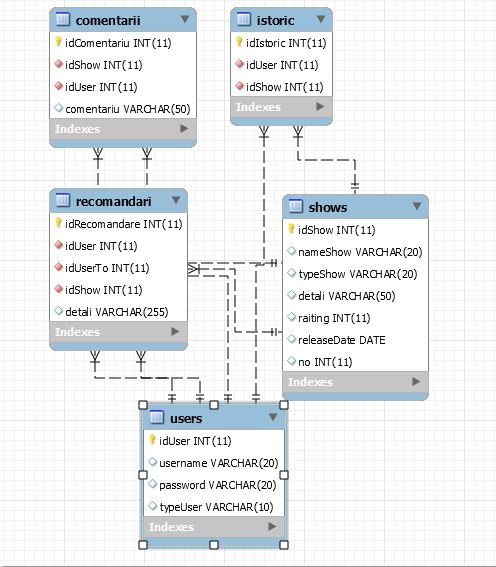
It increases the loose coupling between class abstraction and it’s implementation.

**5.2 UML Class Diagram**

**

6. Data Model

**Hibernate ORM** (Hibernate in short) is an [object-relational mapping](https://en.wikipedia.org/wiki/Object-relational_mapping) tool for the [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) programming language. It provides a [framework](https://en.wikipedia.org/wiki/Software_framework) for mapping an [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) domain model to a [relational database](https://en.wikipedia.org/wiki/Relational_database). Hibernate handles [object-relational impedance mismatch](https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch) problems by replacing direct, [persistent](https://en.wikipedia.org/wiki/Persistence_(computer_science)) database accesses with high-level object handling functions.

**

Hibernate's primary feature is mapping from Java classes to [database tables](https://en.wikipedia.org/wiki/Table_(database)), and mapping from Java data types to [SQL](https://en.wikipedia.org/wiki/SQL" \o "SQL)data types. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.

7. System Testing

The application was tested manually for all operation.

8. Bibliography

<http://download.oracle.com/javase/tutorial/networking/sockets/index.html>

<http://docs.oracle.com/javase/tutorial/uiswing/>

<http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>

<https://refactoring.guru/design-patterns/bridge>

<https://refactoring.guru/design-patterns/observer>

<https://www.tutorialspoint.com/hibernate/orm_overview.htm>

<https://examples.javacodegeeks.com/enterprise-java/hibernate/hibernate-annotations-example/>