Assignment 3

Student: Prodan Andreea

**Group: 30235**

Table of Contents

Contents

[1. Requirements Analysis 3](#_Toc513747676)

[1.1 Assignment Specification 3](#_Toc513747677)

[1.2 Functional Requirements 3](#_Toc513747678)

[1.3 Non-functional Requirements 3](#_Toc513747679)

[2. Use-Case Model 4](#_Toc513747680)

[3.System Architectural Design 5](#_Toc513747681)

[4. UML Sequence Diagrams 7](#_Toc513747682)

[5. Class Design 7](#_Toc513747683)

[6. Data Model 9](#_Toc513747684)

[7. System Testing 9](#_Toc513747685)

[8. Bibliography 9](#_Toc513747686)

# 1. Requirements Analysis

# Assignment Specification

The application represents 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.

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
* 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.

# Functional Requirements

**Interface requirements**

The imdb must have a number format.

The type of the introduced show must be “movie”, “theatre” or “sport event”.

The premium field must be “false”or “true”.

**Regulatory/Compliance Requirements**

When the agent selects a row from the table, the data from the row is put in the specific fields.

If the user enters a wrong data in a field, he/she will receive a warning message.

**Security Requirements**

The administrator can’t access the data about itself.

# Non-functional Requirements

**Access Security**

The application can be accesed just by authorized users, using their username and password.

**Accessibility**

Any person should be able to use the application after reading the user manual.

**Response time**

The first action, of displaying the data to the tables, must last less than 5 seconds. Any other action must last less than 1.5 seconds in 90 precent of cases and less than 5 seconds in rest.

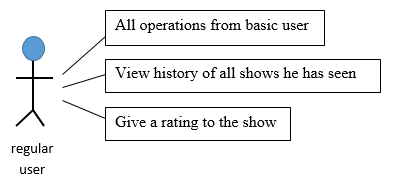
**Availability**

The application should be available allways between 6am and 6pm, the updates being applied just outside this hours.

**Reliability**

Probability of failure on demand (POFOD) shall be 0.0001 (1 out of 10000 plays) when an agent requests to see a specific data.

# 2. Use-Case Model

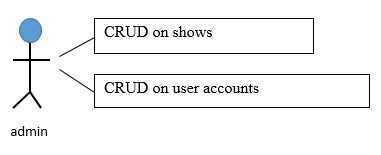


Use case: Select a row from the Show table

Level: user goal

Primary actor: regular user

Main success scenario: The user selects a row from the table. The details of the selected show are displayed in the table below.



Use case: Add a new show

Level: user goal

Primary actor: administrator

Main success scenario: The administrator introduces all the data correctly in the fields and presses the Add button. The show is added.

Extensions: One of the introduced data is incorrect. The adding operation isn’t performed and a warning message is displayed.

# 3.System Architectural Design

**3.1 Architectural Pattern Description**

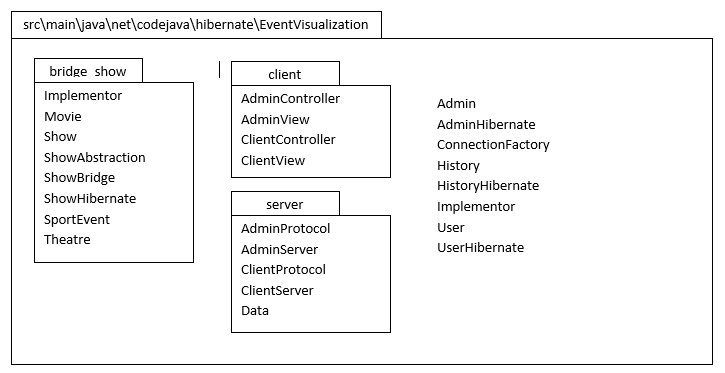
The MVC architectural pattern is used to organize the application. The architecture devides the application into three interconnected parts: view, model, controller. This is done to separate internal representations of information from the ways information is presented to and accepted from the user, allowing for future code reuse.

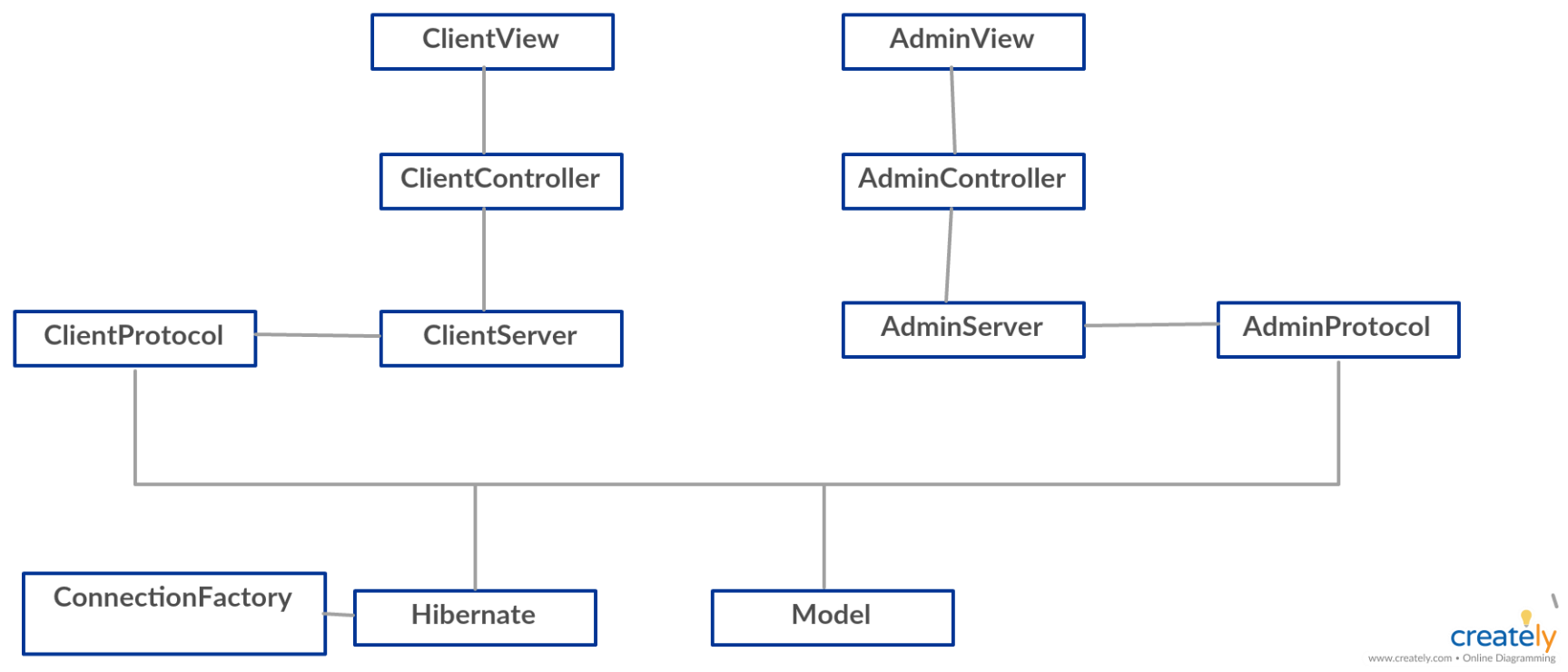
**3.2 Diagrams**

The model is responsible for managing the data of the application.

The view represents the presentation of the model and is what the user can see and access directly. When the user performes an action, the view signals the controller about it and doesn’t takes any other action, unless the controller tells so. The view, therefore, has no knowledge about how the data is processed or stored.

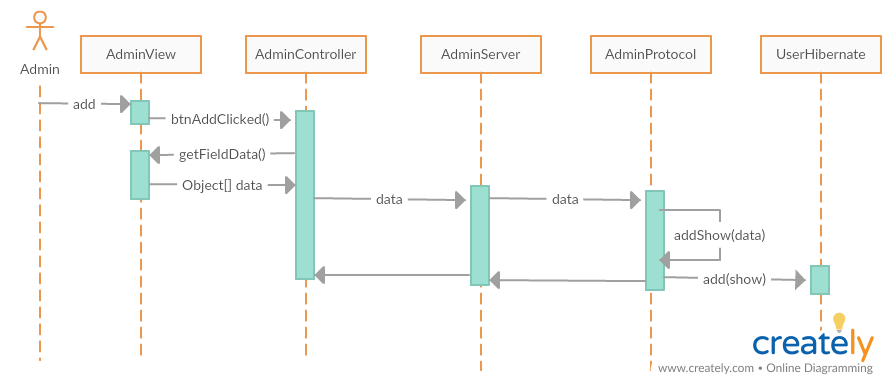
The controller is responsible for responding to the user input and perform interactions on the data model objects. The controller receives the actions performed by the user and input, optionally validates the input and then passes the input to the model. Also, it receives back information from the model, prepares the information in the specific format for the view (Object), so the view will have no knowledge about the model, and then passes it to the view. Therefore, the controller doesn’t care how the information is processed or the logical operations performed for achievieng a specific result, it does care about the steps which have to be followed and the results which should be obtained for each action performed by a user.





# 4. UML Sequence Diagrams

Adding a new show



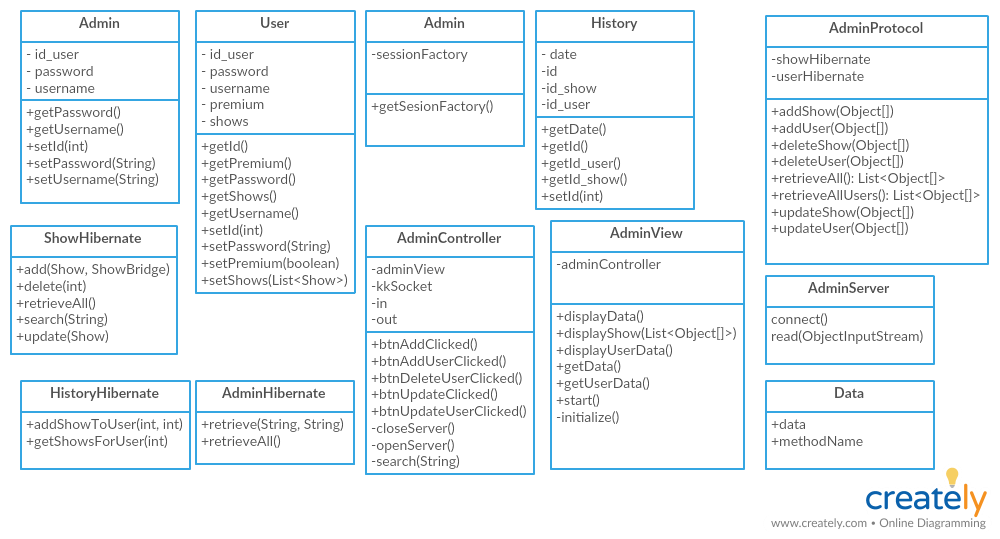
# 

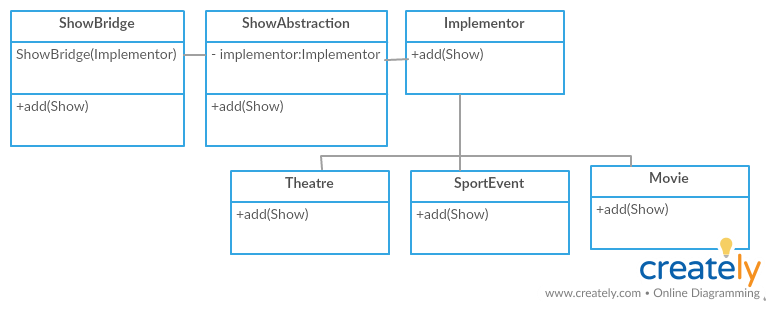
# 5. Class Design

**5.1 Design Patterns Description**

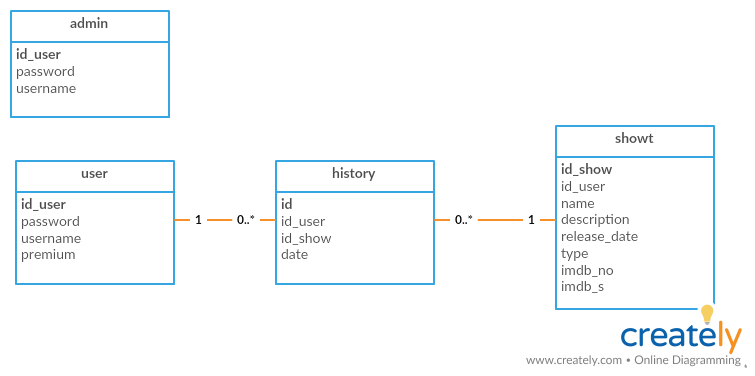
Bridge pattern is used to separate to decouple an abstraction from its implementation so that the two can vary independently. The abstraction is represented by ShowAbstraction. The implementation is composed from the interface Implementor and the concrete clases Movie and, Theatre and Sport, which realise the implementation of the 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 BridgeShow class to which it has to pass the show it wants to add and a concrete type of implementor of the show it wants to add.

**5.2 UML Class Diagram**





# 6. Data Model

**

# 

# 7. System Testing

Testing was made directly on the application, verifying the validations for each field.

# 8. Bibliography

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

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