<Shows>

Student: Pop David

**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 4

3. System Architectural Design 6

4. UML Sequence Diagrams 7

5. Class Design 8

6. Data Model 9

7. System Testing 10

8. Bibliography 10

1. Requirements Analysis

# Assignment Specification

The application aims to implement a management system for an online show service in a client-server structured style.

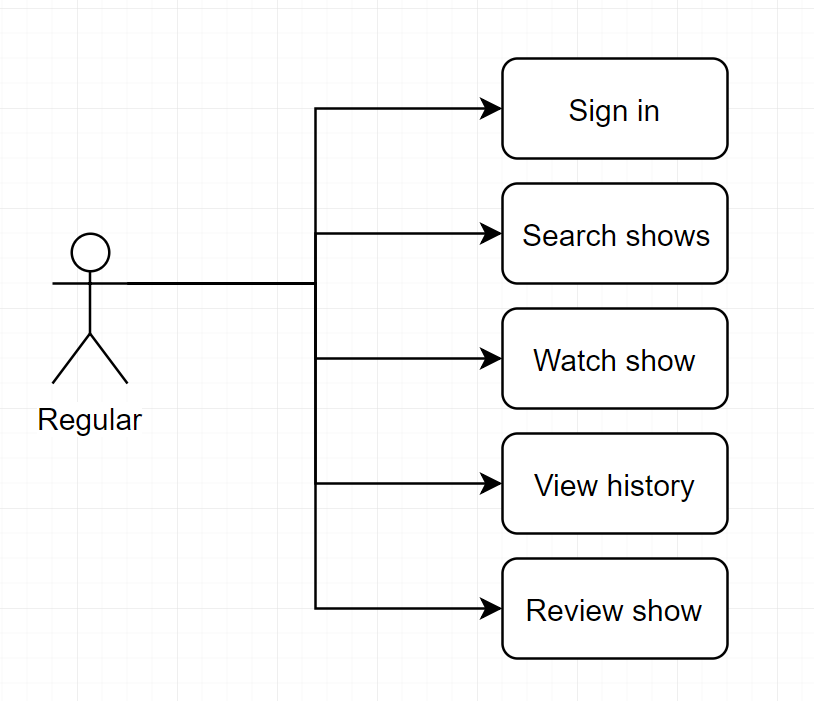
# Functional Requirements

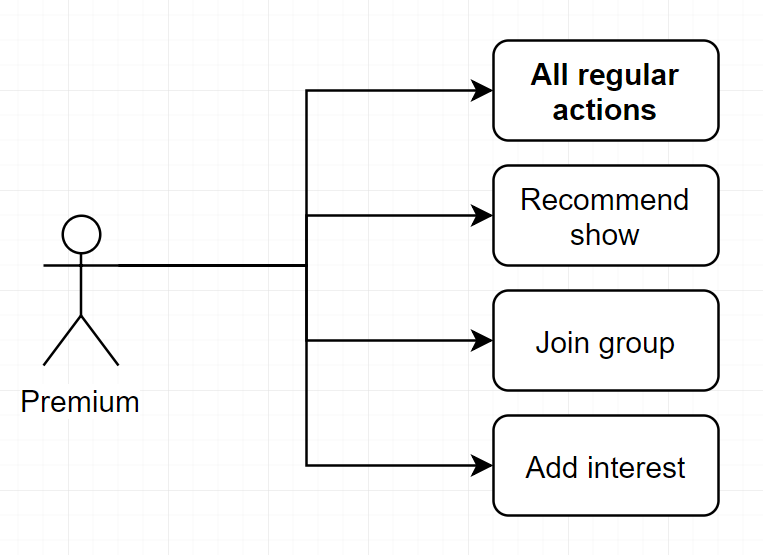
* All input data types are constrained by their respective container, and, optionally, additional logic
* A show’s actor list is an array of names, separated by a comma, and stored as a string
* After sign-in, the application will redirect the user to the corresponding interface ([regular/premium]/administrator)
* All objects stored in the database have been mapped to platform-specific objects using an ORM
* The client application must listen to certain messages sent by the server
* The application must not be accessible by an unauthorized user
* The application must be divided into a server application and a client application
* The server application and client application must accept and send JSON-encoded objects

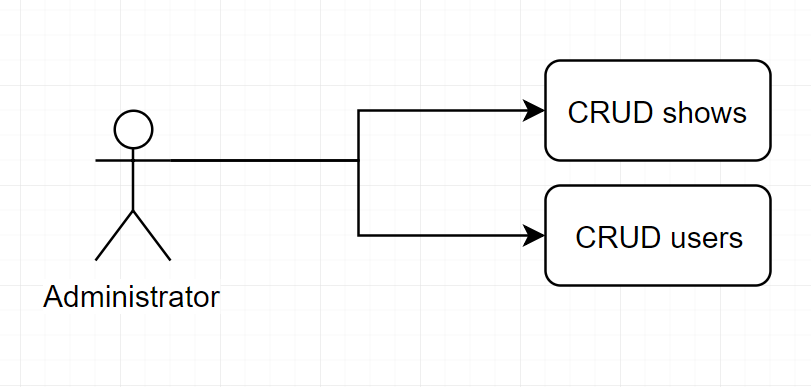
# Non-functional Requirements

* The application should be intuitive enough to be properly used with the knowledge contained in the manual
* The application should not take longer than 1 second to respond to a request

2. Use-Case Model







**Use case:** Recommend show to friend

**Level:** user goal

**Primary actor:** premium user

**Main success scenario:** The user selects a show in the corresponding tab, enters the friend’s username in the text field and presses the “Recommend” button

**Extensions:** none

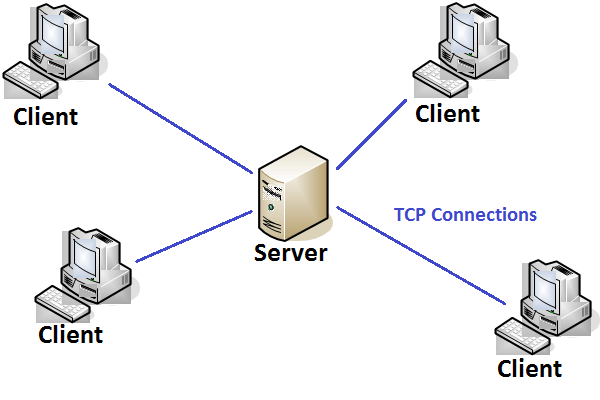
3. System Architectural Design

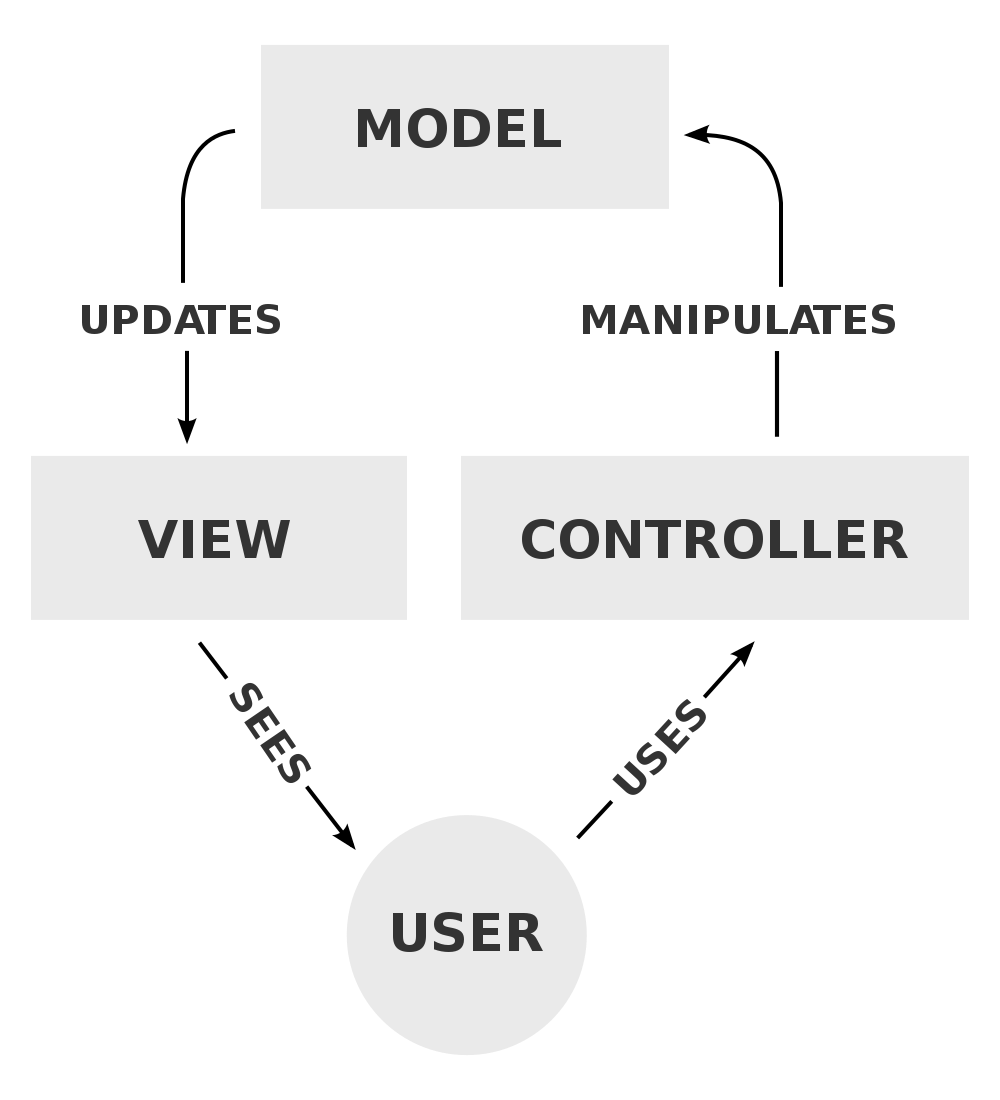
**3.1 Architectural Pattern Description**

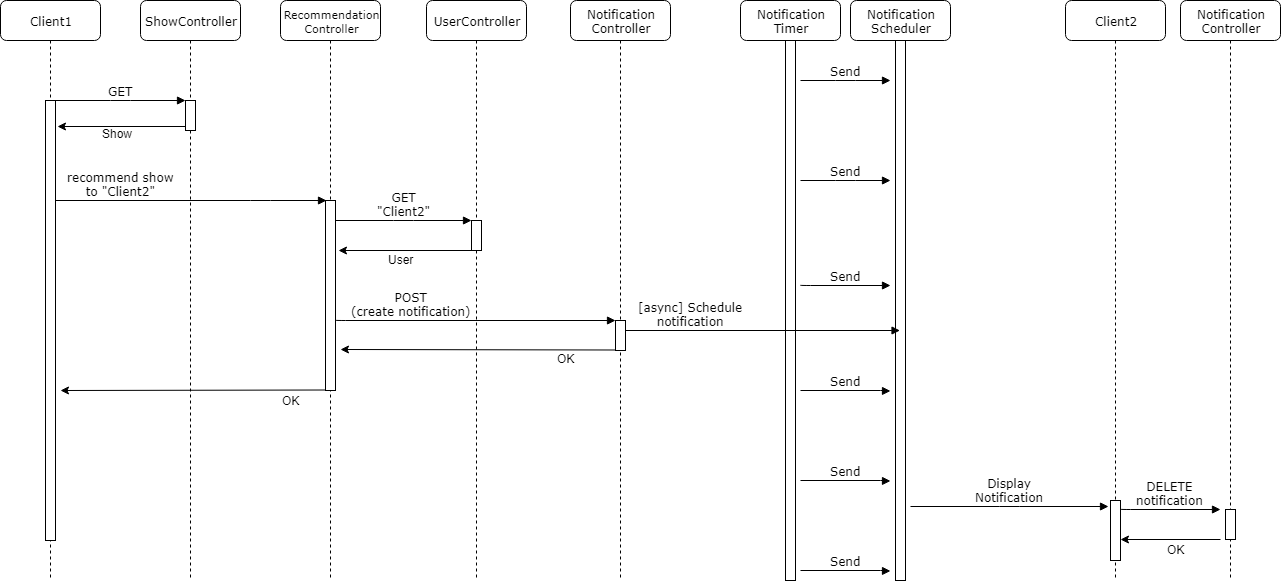
The main architectural pattern used in this application is the client-server pattern. The pattern states that there is one main server managing databases and validating most data, and exposing it’s services via a TCP tunnel, and the clients that act as terminals and user interfaces for the server. Multiple clients can be served at the same time.

A secondary architectural patten is the MVC pattern. “Controllers” are implemented server-side, “Models” are implemented in a core project and “Views” are implemented in the client-side. One notable aberration is that the “Views” have sizeable logic implemented.

**3.2 Diagrams**

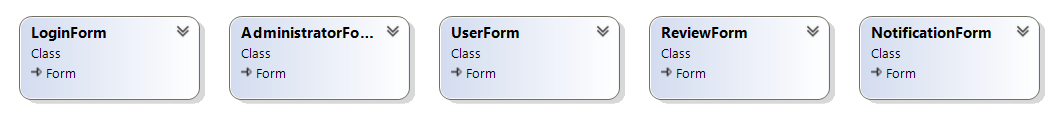


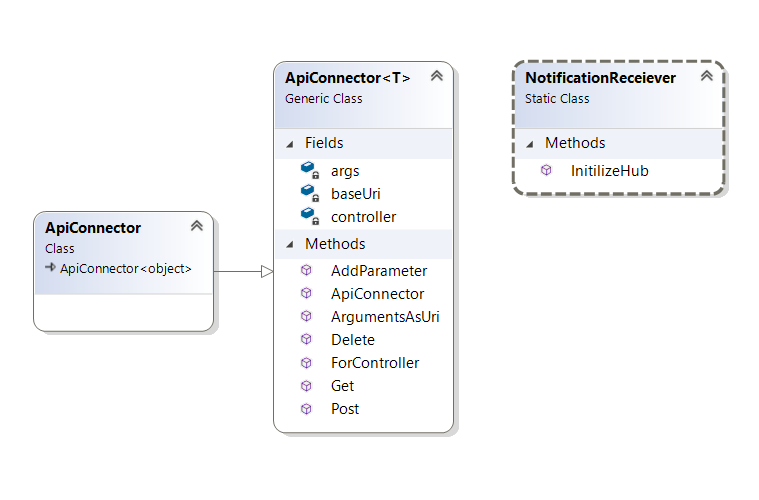


4. UML Sequence Diagrams

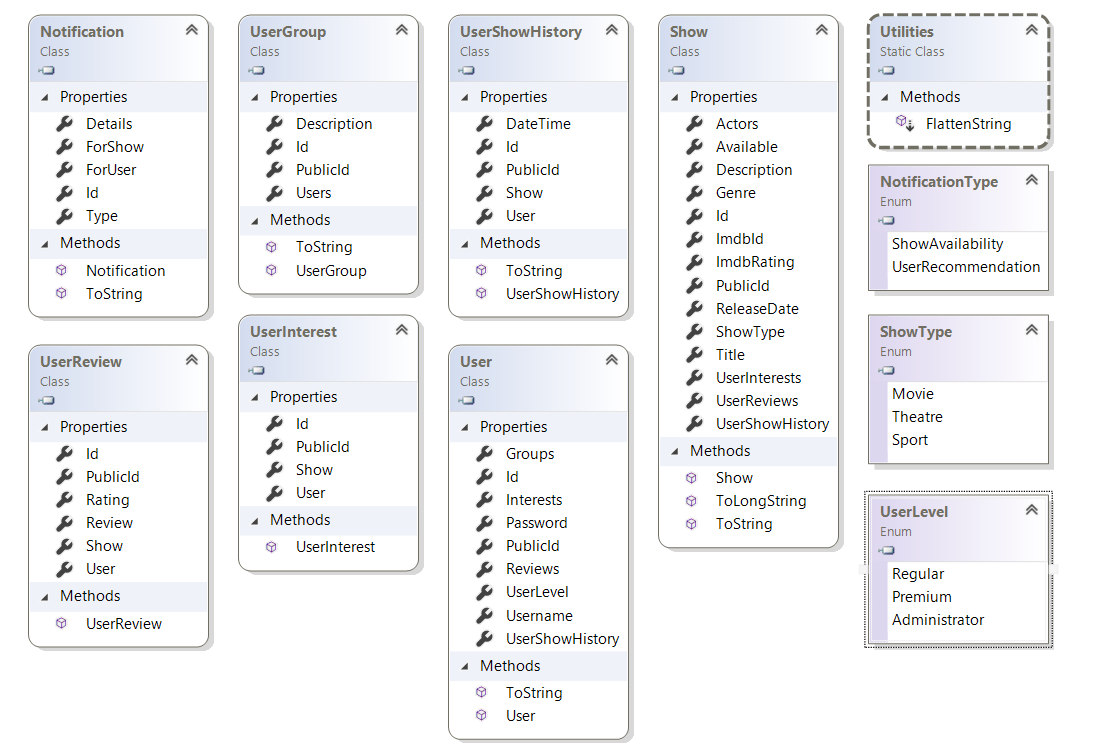
5. Class Design

**Client-side application:**

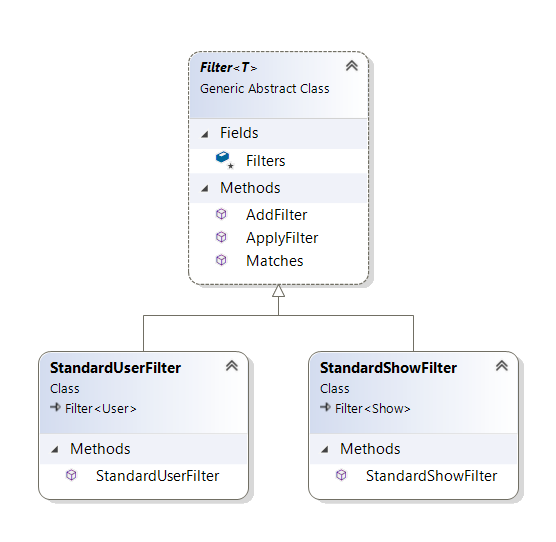
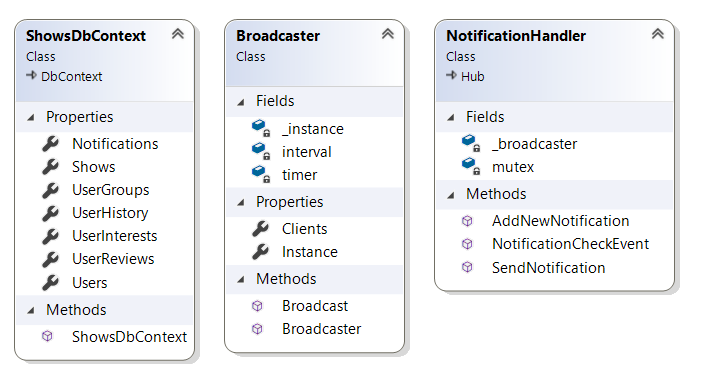
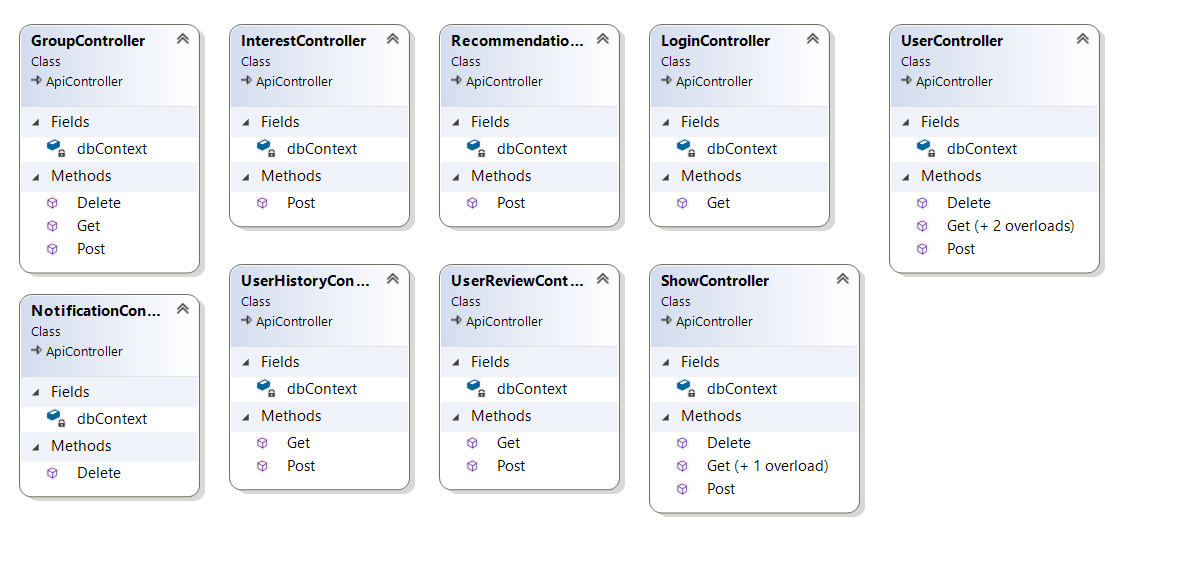




**Core library:**



**Server-side application:**



**5.1 Design Patterns Description**

The ORM data pattern is used to bind database tables with objects. As such, any relevant data retrieval and saving is done transparently in the controller level.

The observer design pattern was used to notify users of changes. Unsent notifications are stored until the recipient user is available.

The filter and bridge design patterns were used to search users and shows. The filter design pattern implies the dynamic creation of criteria which filter a collection of objects. The bridge design pattern decouples abstraction from implementation.

**5.2 UML Class Diagram**

*[Create the UML Class Diagram and highlight and motivate how the design patterns are used.]*

6. Data Model



7. System Testing

Client application testing was done manually, server application testing was done using Postman.

8. Bibliography