<Assignment 3>

Student:Savan Catalin

**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.

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.

# Functional Requirements

*Datele despre utilizatori ,spectacole ,comentari sunt stocate intr.o baza de date .*

*Pentru logare trebuia sa te audentifici cu contul si parola contului tau .*

*Exista 3 tipuri de utilizator : Admin, User si USerPremium.*

*Datele sunt restrictionate in functie de utilizatorul aplicatiei.*

*Este necesar sa ai un cont in baza de date pentru a te loga.*

# Non-functional Requirements

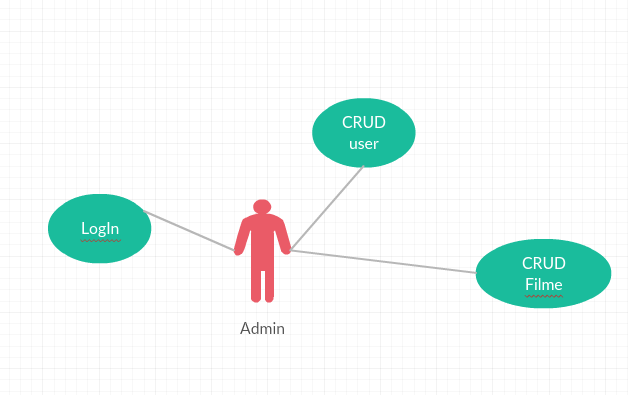
*-Prin interfara destul de explicita aplicatia trebuie sa fie destul de usor de utilizat butoanele si campurile tabelelor avand denumire destul de specifica.*

*-Timpul de raspuns al aplicatiei trebuie sa fie destul de scazut ( < 1 sec ) .*

2. Use-Case Model

*[Create the use-case diagrams and provide one use-case description (according to the format below).*

*Use-Case description format:*

**

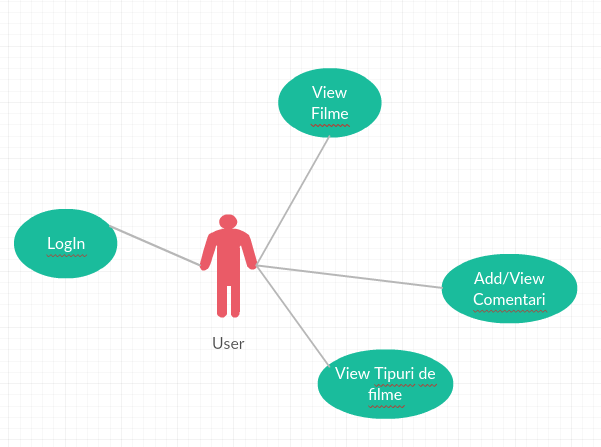
*Use case: Sterge Film*

*Level: user-goal level*

*Primary actor: Admin*

*Main success scenario: Selectarea butonului de stergere , din fata filmului droit*

*Extensions: in urma stergeri ar trebui sa apara dinou lista de filme fara filmul sters.*



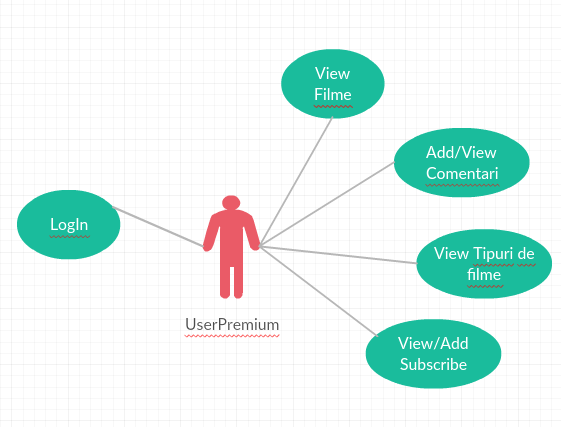
*Use case: Add comentariu*

*Level: user-goal level*

*Primary actor: User*

*Main success scenario: Introduce datele neceasre unui comentariu si anume filmul la care doreste sa posteze comentariul urmat de comentariu si raitinul dorit*

*Extensions: in urma adaugari ar trebui sa apara dinou lista de comentari.*



*Use case: Add Subscribe*

*Level: user-goal level*

*Primary actor: UserPremium*

*Main success scenario: Introduce tipul de film la care doreste sa fac subscribe .*

*Extensions: in urma adaugari ar trebui sa apara dinou lista de subscribe.*

3. System Architectural Design

**3.1 Architectural Pattern Description**

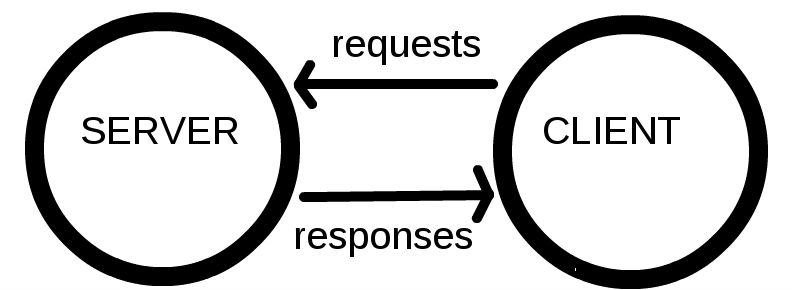
*Pentru organizarea proiectului am utilizat pattern-ul architectural MVC , prin aceasta arhitectura proiectul se imparte in 3 parti : Model( descrierea datelor si felul cum pot fi utilizate in proiect) , View( interfata de vizulaizare a datelor gata prelucrate) si Controller( logica de prelucrare a datelor) , acest pattern nu permite vizualizarea concreta a structuri datelor prelucrate ,de catre utilizator acesta vizualizand doar rezultatul de la finalul operatiilor.*

*Organizarea proiectului este un Client-Server partea de frontend ruland pe un anumit server in cazul nostrum 8181 , iar partea de client pe portul 8080 , clientul trimitand doar cerinte serverului care la randul lui in urma cerintei trimite spre interfata client datele cerute.*

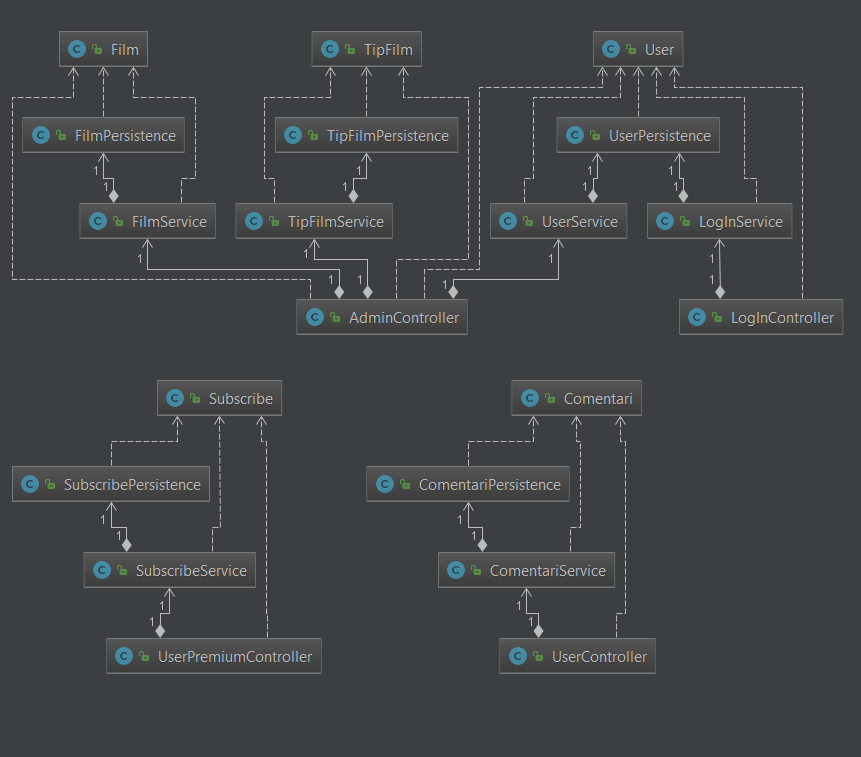
*Accesarea datelor se realizeaza cu ajutorul hibernate.*

**3.2 Diagrams**

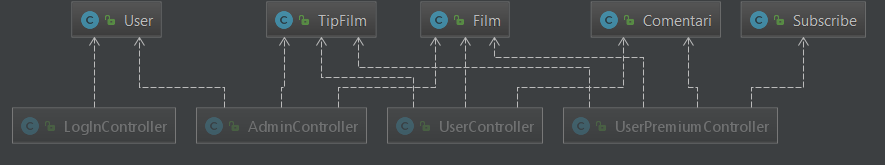
*[Create the system’s conceptual architecture; use architectural patterns and describe how they are applied. Create package, component and deployment diagrams]*



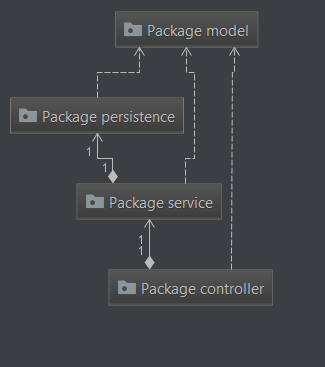
*Server:*

**

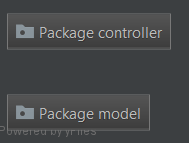
*Client:*

**

*Server:*

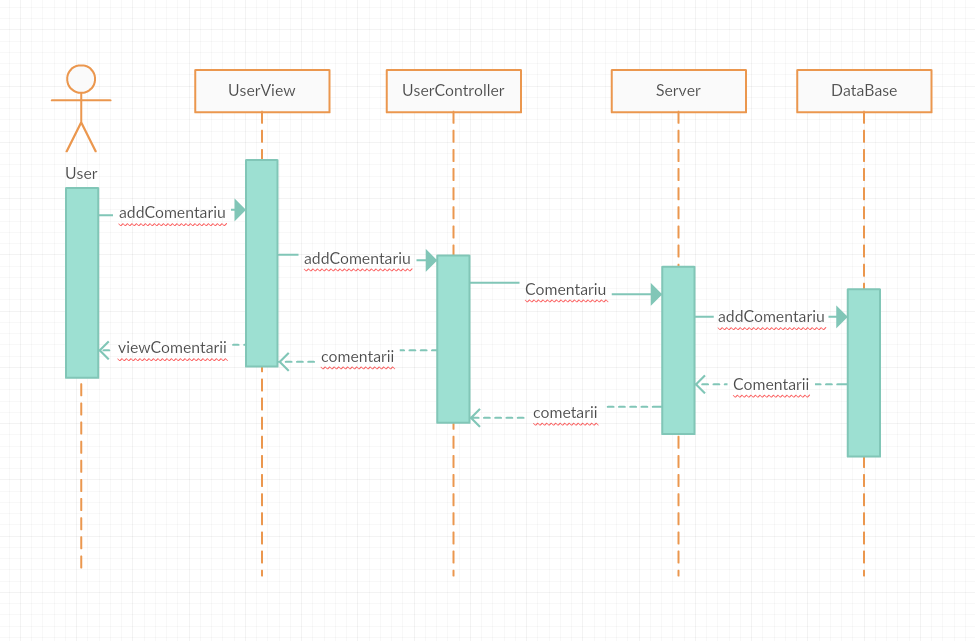
**

*Client:*

**

4. UML Sequence Diagrams

*[Create a sequence diagram for a relevant scenario.]*

**

5. Class Design

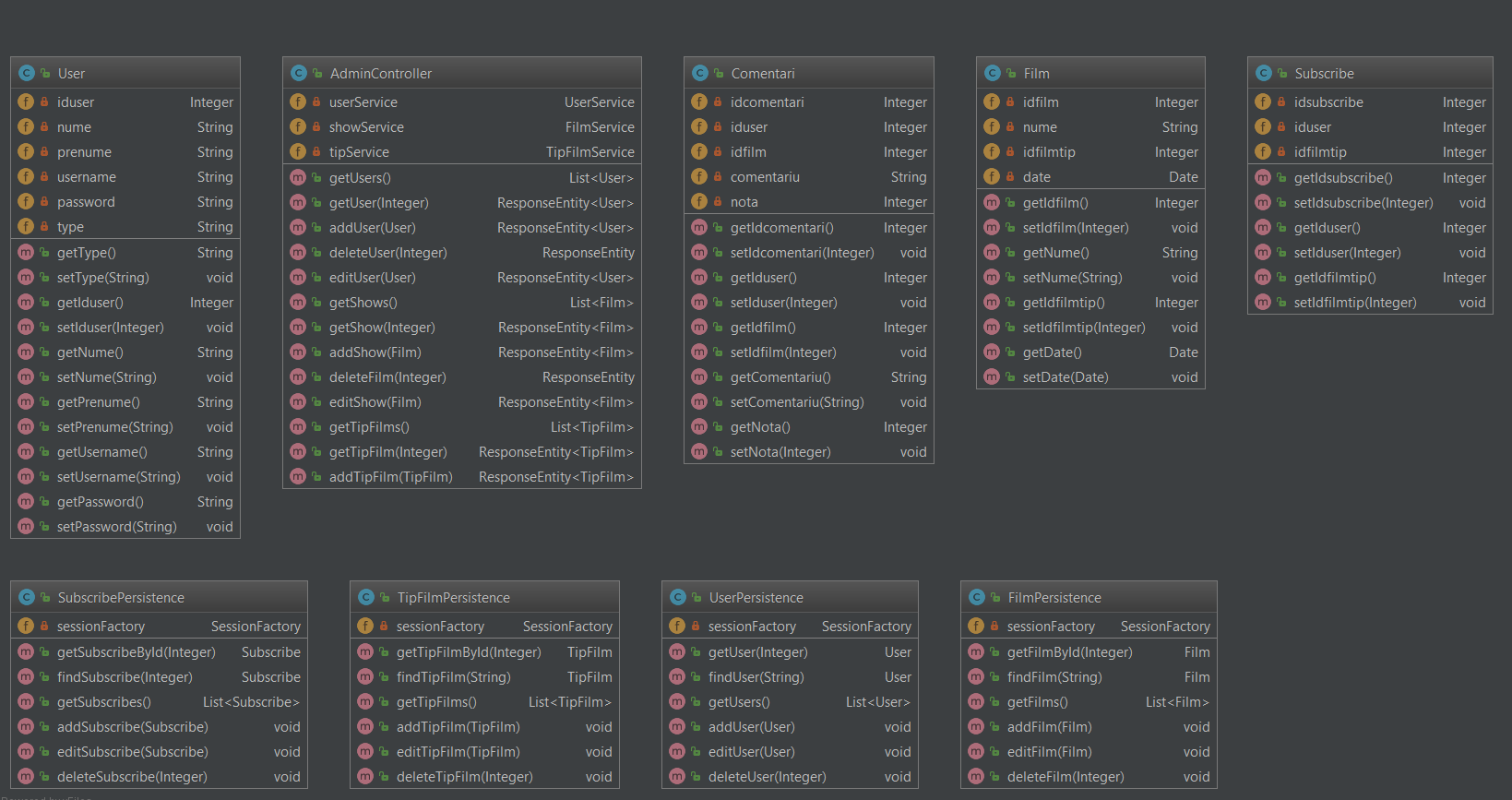
**5.1 Design Patterns Description**

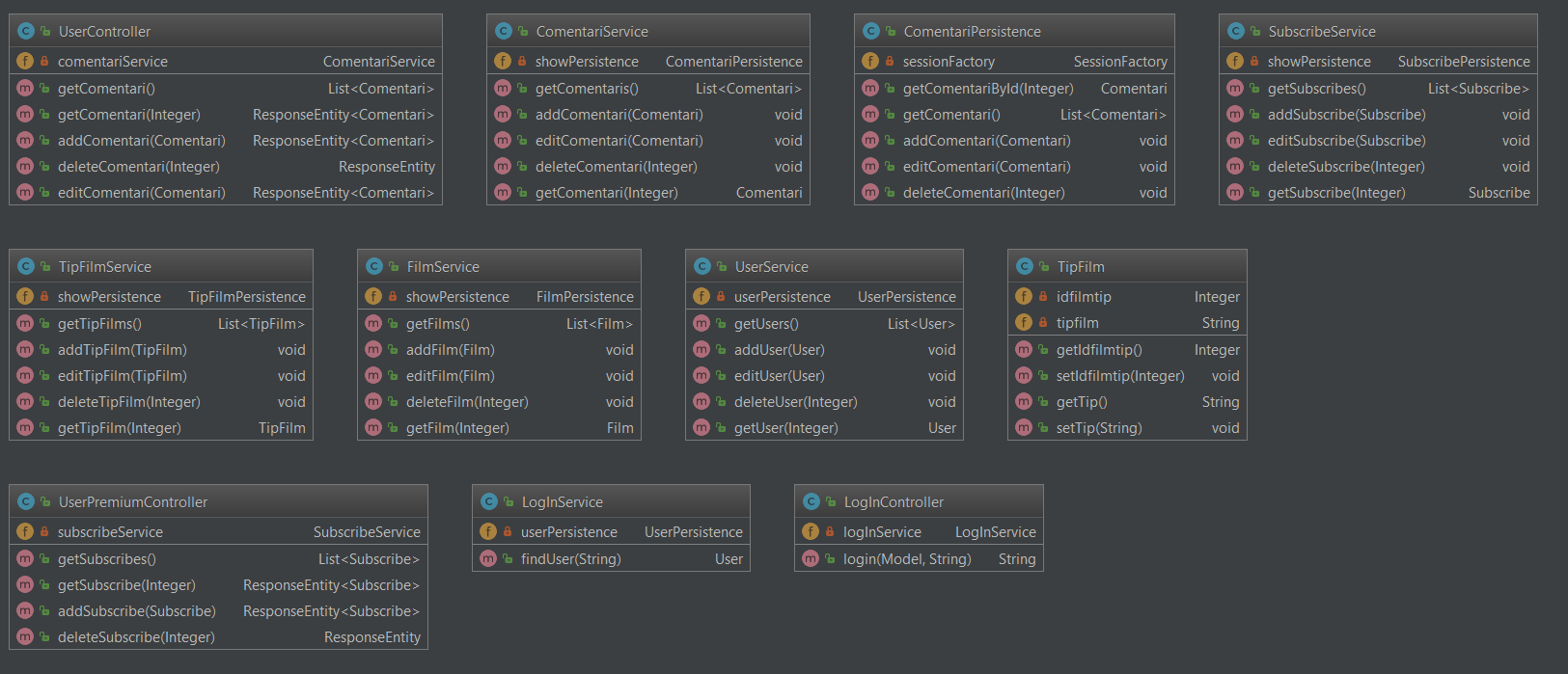
*[Describe briefly the used design patterns.]*

**5.2 UML Class Diagram**

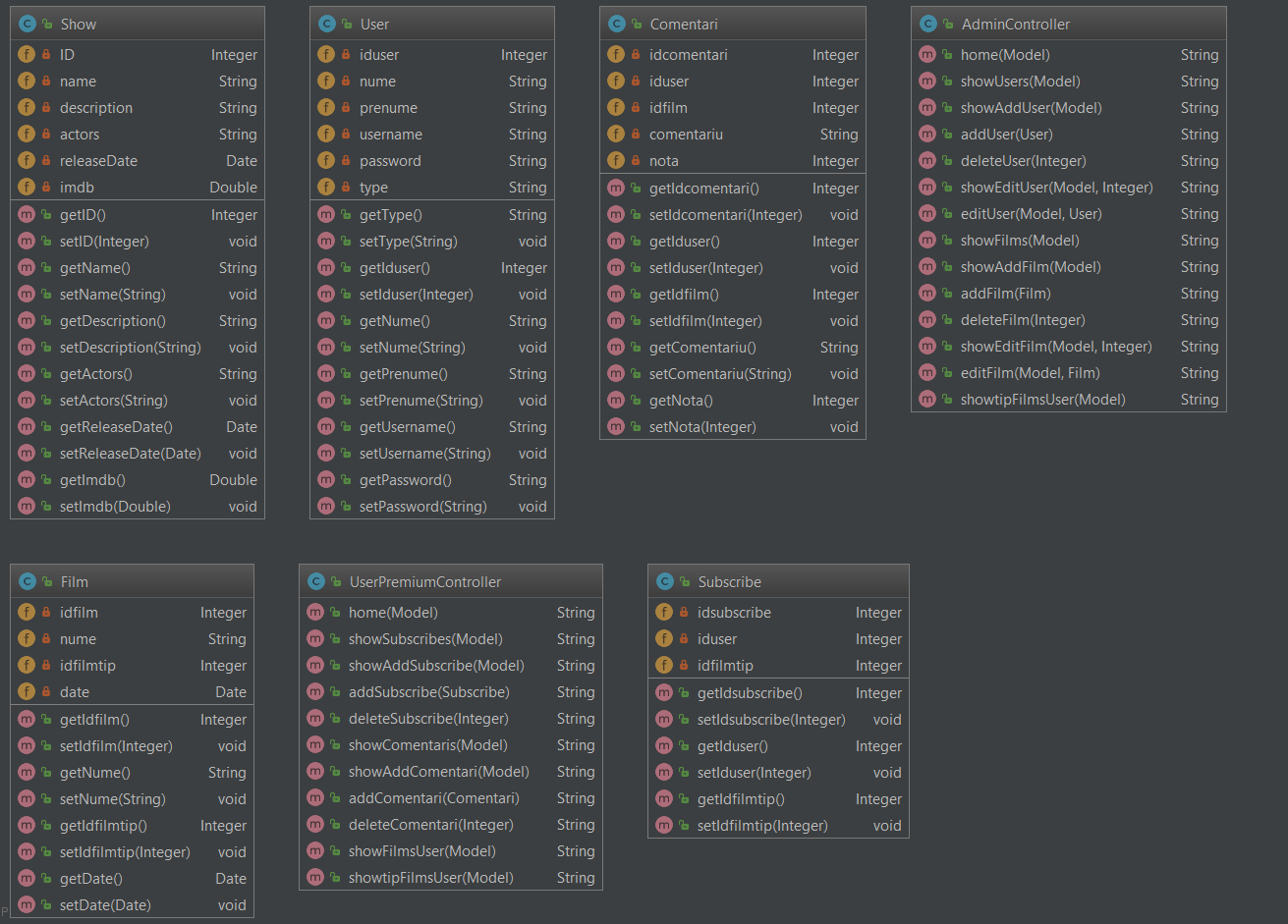
*[Create the UML Class Diagram and highlight and motivate how the design patterns are used.]*

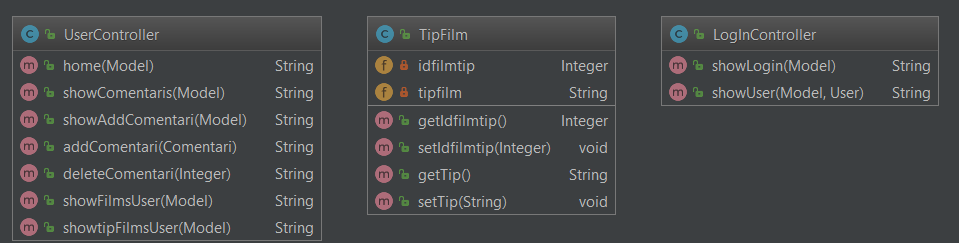
*Server:*

**

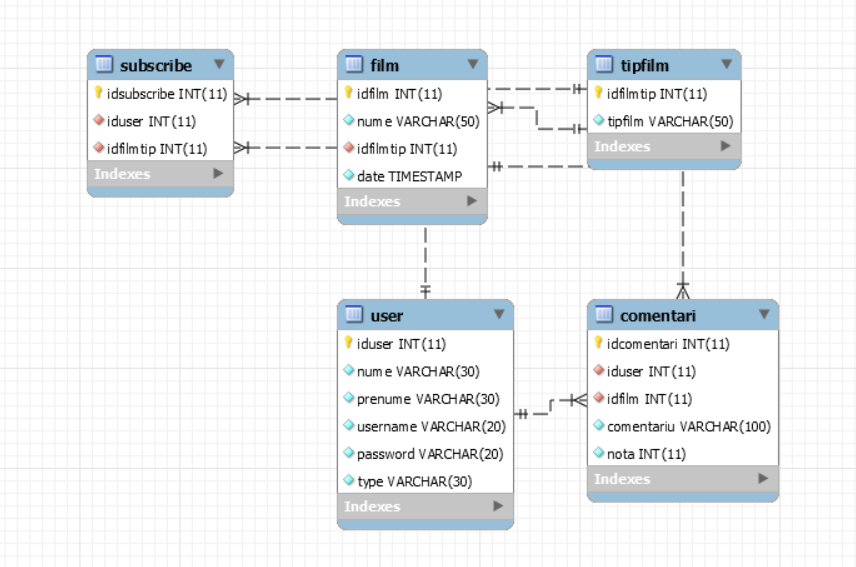


Client:





6. Data Model

**

7. System Testing

*Testarea s-a realizat prin verificare datelor primate de client si datele afisare de server , functionalitatea paginilor pentru fiecare utilizator , si nu in ultimul rand operatiile de add/delete/update.*

8. Bibliography