



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
CATEDRA CALCULATOARE

MOVIE HOUSE

-documentație-
-proiect nr.17-

Student:

Pop Ruxandra Maria
Anul 3, grupa 30236

1. Enunțul problemei

Scopul proiectului este dezvoltarea unei aplicații desktop care poate fi utilizată de o casă de producție filme. Aceasta aplicație va dispune de 2 tipuri de utilizatori: angajat și administrator, fiecare putând efectua diferite operații în funcție de rolul lui.

Obiectivul principal al acestei teme este familiarizarea cu şablonul arhitectural Client-Server, dar și cu numeroasele sabloane de proiectare alese de mine, printre care se numără Singleton, Strategy, Observer, Decorator, Builder, Proxy.

2. Instrumente utilizate

Înainte să ne apucăm de proiectarea și implementarea propriu zisă a aplicației, este necesar să cunoaștem ce cerințe trebuie acesta să indeplinească, pentru a descoperi ce trebuie să facă sistemul, cum trebuie să funcționeze, pentru a stii ce obiective dorim să atingem la sfârșitul implementării.

Astfel ca și prima etapă în realizarea temei, am construit diagrama de use case, care cuprinde actorii aplicației și operațiile pe care le pot face. Diagrama de use case a fost realizată cu ajutorul instrumentului software **starUML**, tot prin intermediul acestui instrument am proiectat și diagrama de clase, care respectă arhitectura Client-Server.

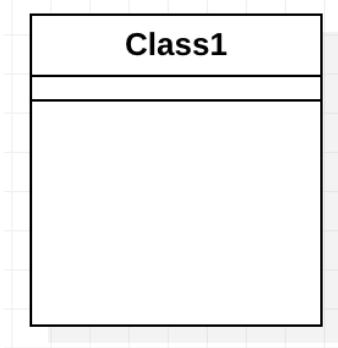


Fig2.1

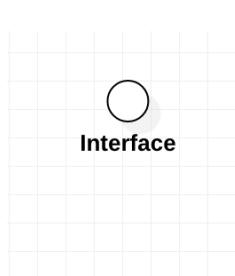


Fig2.2

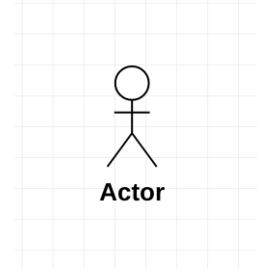


Fig2.3

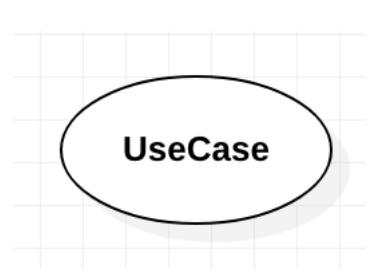


Fig2.4

După ce am realizat diagrama de use case ,am trecut la realizarea diagramelor de activității pentru fiecare caz de utilizare în parte,iar pe urma la realizarea diagrameelor de secvență.

Aplicația a fost implementată cu ajutorul mediulului de dezvoltare IDE IntelliJ IDEA 2021.2.2, iar ca limbaj de programare am ales să folosesc Java.

Java este un limbaj de programare orientat pe obiecte, a fost conceput de James Gosling la Sun Microsystems la începutul anilor 90 ,fiind lansat în 1995.

Am ales să folosesc acest limbaj ,pentru că mi se pare extrem de simplu să realizezi aplicații de tip GUI ,deoarece există framework-ul Java Swing ,care permite să realizezi interfețele intr-un mod mai prietenos,având o privire de asamblu asupra interfeței . Totodată , am vrut să îmi îmbunătățesc cunoștiințele despre acest limbaj de programare, deoarece în viitor îmi doresc să lucrez la proiecte de dimensiuni mai mari .

Pentru persistență am utilizat o bază de date relațională și anume **phpMyAdmin**, unde am creat o bază de date numită **moviehouse** ,unde am creat două tabele : **Movie** și **User**. Pentru a corespunde cu clasa **AbstractDAO** ,care se ocupă cu implemtarea query-lor.



Fig2.5 IntelliJ icon

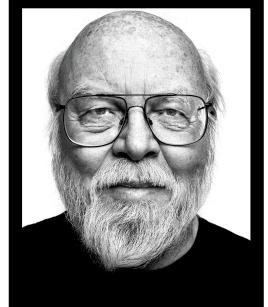


Fig2.6 James Gosling



3. Descrierea diagramelor UML

3.1 Diagrama de use case

Diagrama de use case prezintă o colecție de cazuri de utilizare și actori.

În cazul acestei teme :

- actorul este utilizatorul(administratorul/angajatul)
- iar cazurile de utilizare sunt:

Pentru administrator

În caz de succes:

- Username-ul și parola sunt corecte și preluate din baza de date USER
- Adaugă un nou utilizator completând toate field-urile necesare și apasă butonul Add
- Selectează utilizatorul a cărui date vrea să le modifice,completând field-urile necesare,toate detaliile despre utilizator înfără de username pot fi modificate.
- Totodată poate să steargă utilizatorul selectat prin apăsarea butonului Delete
- Poate să părasească contul prin apasarea butonului LogOut
- Poate să filtreze utilizatori după tipul lor(admin/angajat)
- Poate să aleagă în ce limbă să fie afișată interfața

În caz de nesucces:

- Username-ul sau/și parola nu sunt corecte, nu se găsesc în baza de date
- Nu selectează produsul pe care dorește să-l modifice
- Încearcă să modifice username-ul unui utilizator selectat
- Încearcă să adauge un nou utilizator care are același username ca un utilizator care există deja
- În toate situațiile se vor genera eroriile/avertizările necesare prin interfață grafică

Pentru angajat

În caz de succes:

- Username-ul și parola sunt corecte și preluate din baza de date User
- Adaugă un nou film completând toate field-urile necesare și apasă butonul Add,
- Selectează filmul a cărui date vrea să le modifice,completând field-urile necesare,toate detaliile despre film înfără de titlu pot fi modificate.
- Totodată poate să steargă filmul selectat prin apăsarea butonului Delete
- Poate să caute un film pe baza titlului acestuia

- Poate să filtreze filmele după numeroase criterii: tip film, categorie film, anul realizării
 - Poate să genereze statistici în legătură cu numărul de filme dintr-o anumită categorie/tip
 - Poate să salveze rapoarte cu informații despre filme în mai multe formate: csv, json, xml
 - Filmele vor fi afisate în ordinea categoriei și anul producției
 - Poate să aleagă în ce limbă să fie afișată interfața

În caz de nesucces:

- Username-ul și parola nu sunt corecte, nu se găsesc în baza de date
 - Încearcă să adauge un nou film care are același titlu ca un alt film care există deja, iar firmul nu este de tip serial. Dacă filmul are aceeași nume cu alt film, dar ambele filme sunt de tip serial, dar anul realizări este același.
 - Încearcă să modifice titlul unui film selectat
 - În toate situațiile se vor genera eroriile/avertizările necesare prin interfață grafică

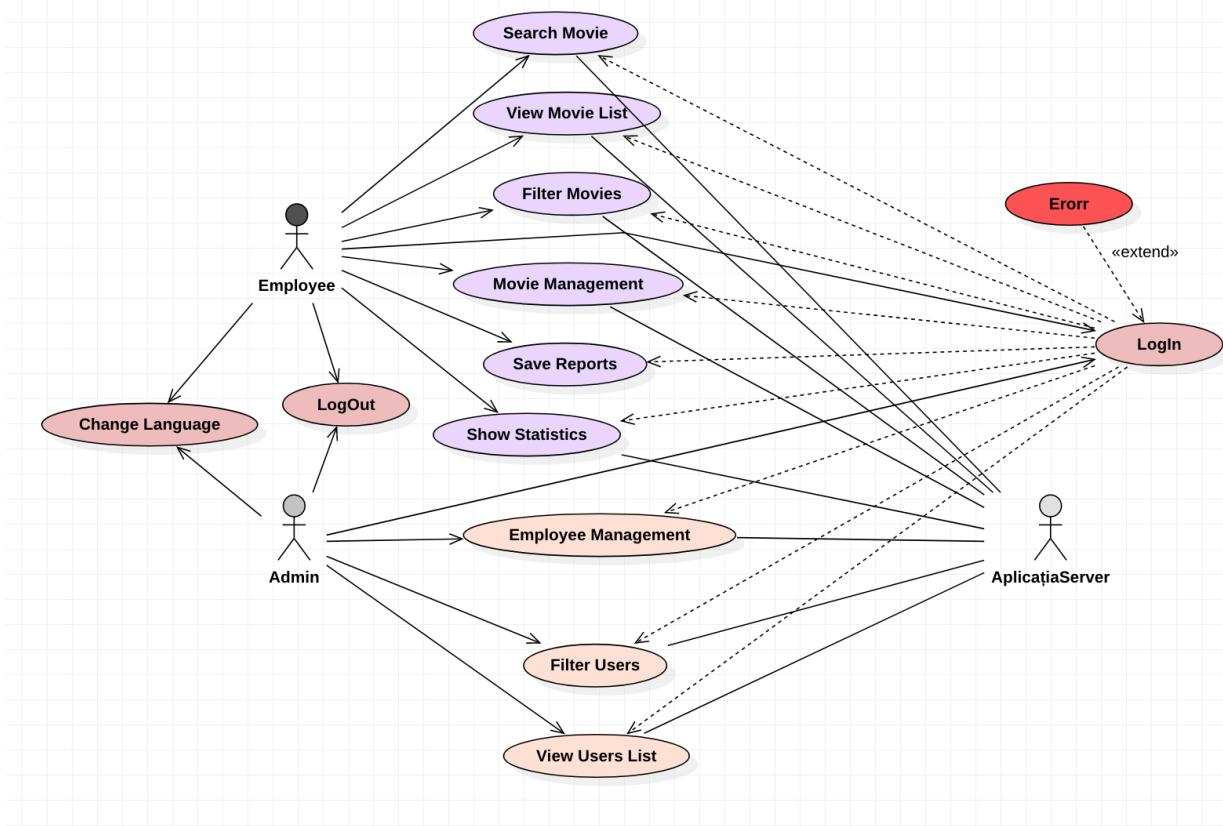


Fig. 3.1.1 Diagrama de use case pentru Client

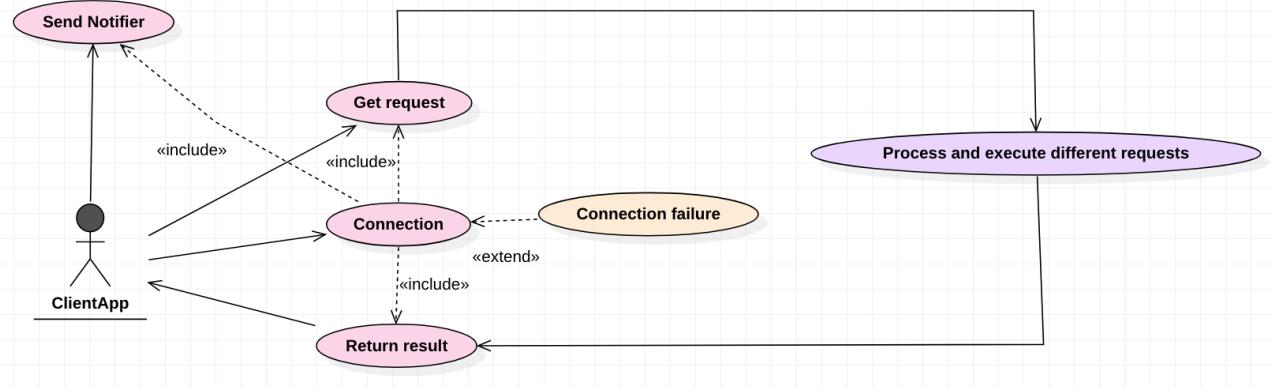


Fig. 3.1.2 Diagrama de use case pentru Server

Pentru a realiza comunicarea dintre client și server, avem nevoie de conexiunea ce se realizează după cum este descris în diagrama de mai sus. Clientul cere aplicației Server, date prin intermediul unor mesaje, iar aplicația Server trimite obiecte către client prin ObjectOutputStream.

3.2 Diagrama de clase

Reprezintă un set de clase, interfețe, colaborări și alte relații.

Diagrama de clase a fost realizată ,respectând modelul arhitectural Client Server .Astfel fiecare pachet ,clasa ,are un rol prestabilit.

Arhitectura client / server este un model de calcul în care serverul găzduiește, livrează și gestionează majoritatea resurselor și serviciilor care urmează să fie consumate de client. Acest tip de arhitectură are unul sau mai multe computere client conectate la un server central printr-o rețea sau conexiune la internet. Acest sistem partajează resurse de calcul.

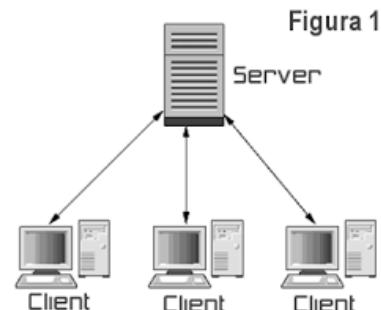


Fig 3.2.1 Arhitectura Client-Server

3.2.1 Diagrama de clase pentru Aplicația Client

Pentru realizarea aplicației Client am respectat modelul arhitectural Model-View-Controller. Aceasta arhitectură duce la izolarea părții logice de interfață proiectului. Ca și structura este alcătuit din 3 pachete:

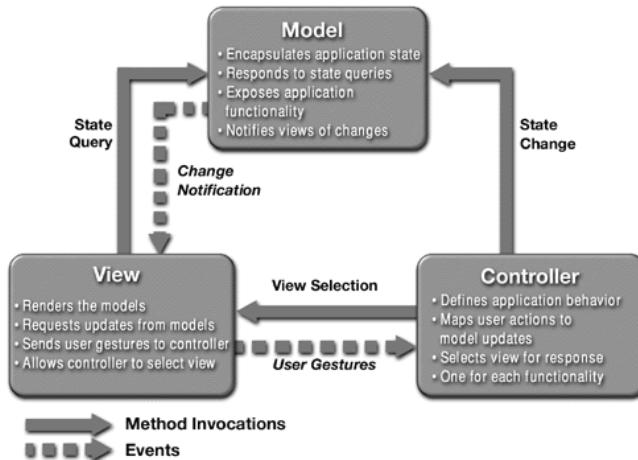


Fig 3.2.1.1 Modelul arhitectural MVC

- **Model** - modelul este responsabil cu gestionarea datelor din aplicație. Răspunde la cereri care vin din View, deasemenea și intrucțiunilor din Controller. Modelul reprezintă nucleul aplicației - aici se face și legătura cu baza de date.

Cuprinde următoarele clase:

- Language: reprezintă un dicționar care cuprinde cuvintele în cele 3 limbi disponibile :Română ,Engleză și Spaniolă.
- Model admin : reprezintă clasa care face legatura cu pattern-ul Observer, și cuprinde 2 atrbute necesare pentru un utilizator de tip admin: o instață de tip Language și una de tip UserBLL.
- Model user : reprezintă clasa care face legatura cu pattern-ul Observer, și cuprinde 2 atrbute necesare pentru un utilizator de tip angajat : o instață de tip Language și una de tip MovieBLL.

- Subject : reprezintă o clasa abstractă necesară pentru a implementa patternul Observer, care va fi moștenită de cele 2 modele declarate mai sus. Ea menține o listă de referințe cu observatori, astfel când apar modificări anunță toții observatorii.
- Observer : definește o interfață, care cuprinde o singură metodă care va fi invocată de către Subject pentru a notifica o schimbare.

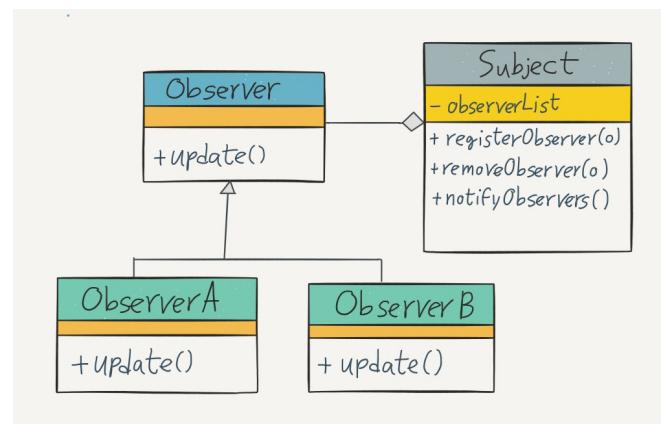


Fig 3.2.1.2 Modelul comportamental Observer

- UserBuilder și MovieBuilder sunt 2 clase folosite pentru implementarea sablonului creational Builder. Fieind o interfață pentru creare părțiilor unui obiect.

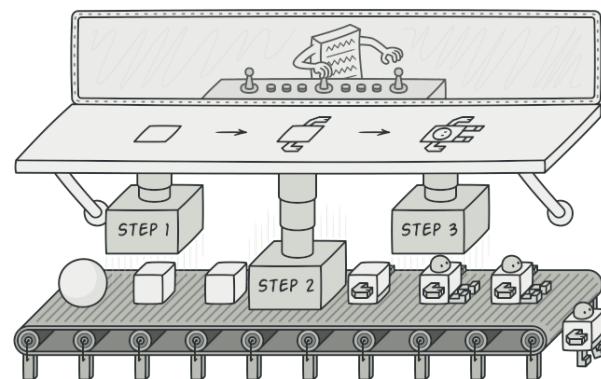


Fig 3.2.1.2. Modelul creational Builder

- **Controller** - este partea aplicației care se ocupă de interacțiunea cu utilizatorul. În controller se citesc datele introduse de utilizator, se trimit către model, se execută operațiile, după care se trimite răspunsul către view.

Cuprinde următoarele clase:

- AdministratorC: aici se găsesc mai multe clase, care implementează operațiile pe care le poate realiza administratorul aplicației(stergerea, adăugarea, actualizarea unui utilizator; schimbarea limbii de afișare, filtrarea utilizatorilor după tipul lor).
- UserC: aici se găsesc mai multe clase, care implementează operațiile pe care le poate realiza angajatul aplicației(stergerea, adăugarea, actualizarea unui film; schimbarea limbii de afișare, filtrarea filmelor după tipul, an, categorie; căutarea unui film după titlu; salvarea raportelor în cele 3 formate:xml, csv, json; generarea statisticilor).
- LoginC: aici se găsesc mai multe clase, care realizează ori schimbarea limbii de afișare a interfeței ori logarea utilizatorului prin username și parolă.

- **View** - este partea în care se afișează datele (înregistrările din baza de date), exprimarea ultimei forme a datelor: interfață grafică ce interacționează cu utilizatorul final.Rolul său este de a evidenția informația obținută până ce ea ajunge la controller.

Cuprinde următoarele clase:

- AdministratorGUI: aici se proiectează interfață pt administrator prin numeroasele atribute prezente, și se gasesc metodele care vor fi implementate în controllerul corespunzător
- UserGUI: aici se proiectează interfață pt angajat prin numeroasele atribute prezente, și se gasesc metodele care vor fi implementate în controllerul corespunzător
- LoginGUI: aici se proiectează interfață pt pagina de login prin numeroasele atribute prezente, și se gasesc metodele care vor fi implementate în controllerul corespunzător

Alte pachete prezente în aplicația Client sunt:

- **BLL**- reprezintă pachetul care face legatura dintre client și server. Contine un atribut de tip Client, prin intermediul căruia se transmit mesaje către Server, pentru a se realiza operațiile din baza de date.

Cuprinde următoarele clase:

- MovieBLL: care transmite mesaje către Server, pentru a se face operațiile necesare în baza de date Movie.
- UserBLL: care transmite mesaje către Server, pentru a se face operațiile necesare în baza de date User.

- **Adapter**- care reprezintă defapt design pattern-ul cu același nume. Cuprinde o singură clasă, StatisticAdapter, care e folosită pentru a converti o listă de filme, în chart-uri.
- **Proxy**- care reprezintă defapt design pattern-ul creational cu același nume. Cuprinde o interfață care conține o metodă de login, și o clasă ProxyLogin, care implementează interfața și are ca scop scrierea metodei de login care va fi folosită în LoginC.
- **Com**- cuprinde clasa Client care va face legatura cu Server-ul. Aici se întâlnesc metode care au ca scop, conectarea la Server, recepționarea de mesaje de la Server, și închiderea conexiunii cu Server-ul.
- **Strategy**- respectă arhitectura design-ului care are același nume.

Cuprinde următoarele clase:

- WriteStrategy: este interfața care va fi implementată de cei 3 algoritmi de scriere.
- WriteJSON: are ca scop scrierea filmelor într-un fisier de tip Json.
- WriteCSV: are ca scop scrierea filmelor într-un fisier de tip csv.
- WriteXML: are ca scop scrierea filmelor într-un fisier de tip xml.

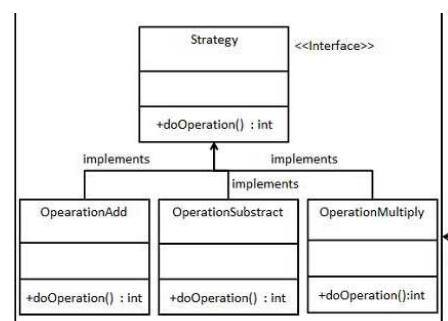


Fig 3.2.1.3. Modelul structural Strategy

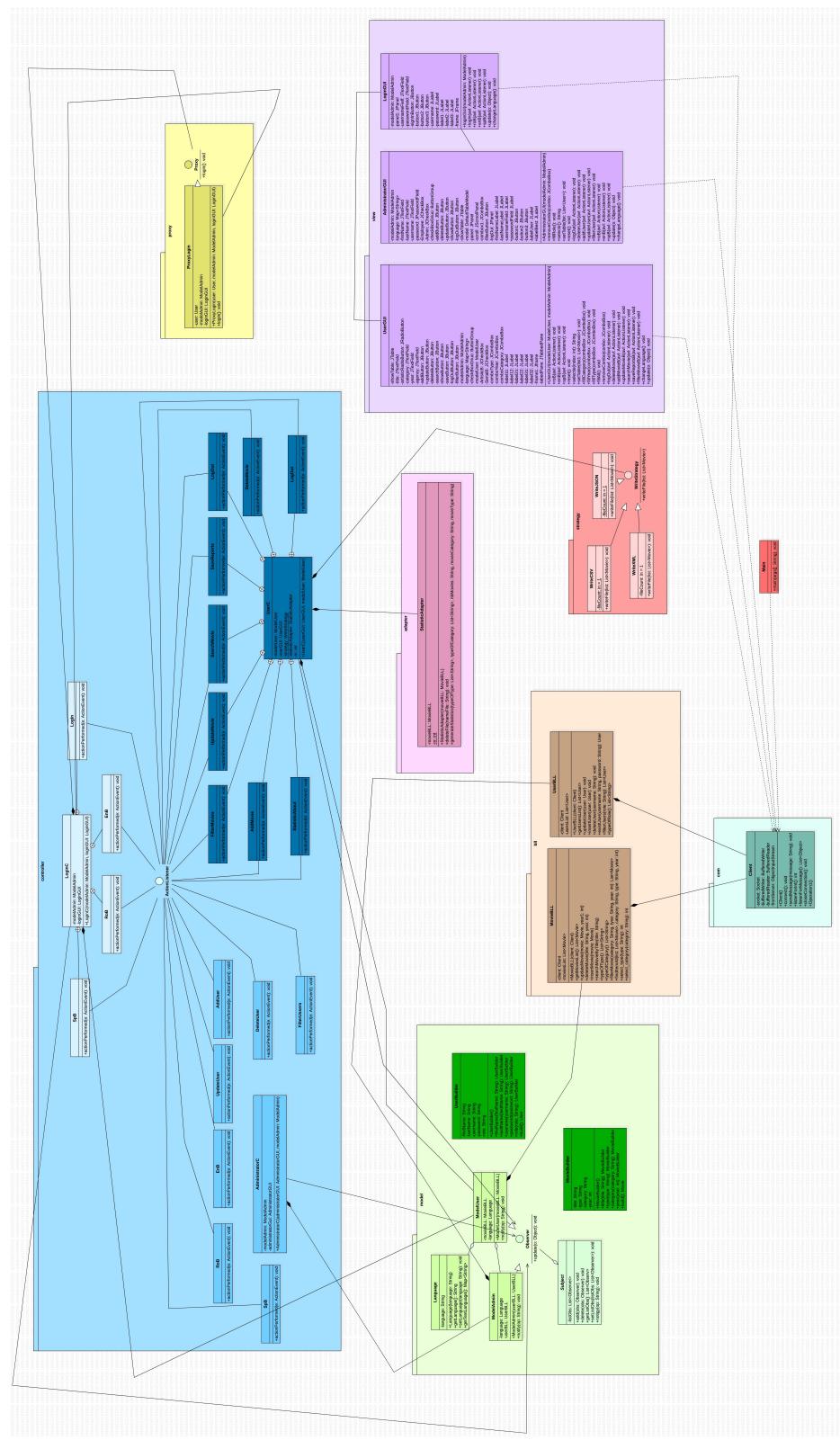


Fig 3.2.1.4. Diagrama de clase pentru aplicația Client

3.2.2 Diagrama de clase pentru Aplicația Server

Pentru realizarea aplicației Server, este nevoie să avem o conexiune la baza de date moviehousedb. Pentru a implementa această conexiune am folosit design pattern-ul



Fig. 3.2.2.1 Modelul structural Singleton

Singleton, deoarece este nevoie de o singură instanță pentru baza de date.

Acest design pattern este implementat în pachetul connection.

- **Connection** - conține clasa ConnectionFactory care are ca scop realizarea conexiunii dintre baza de date și aplicația client.
- **DAO** - aici se găsesc metodele necesare manipulării bazei de date, adăugare în baza de date, stergere, actualizare... și totodată se găsesc și metodele care crează query-urile.

Cuprinde următoarele clase:

- **AbstractDAO**: care este o clasă abstractă după cum sugerează și numele, conține toate metodele de manipulare a bazei de date, îndiferent de tipul obiectului.
- **MovieDAO**: extinde clasa AbstractDAO, deci moșteneste toate metodele ei, și le convertește în metode care manipulează obiecte de tip Movie. Mai conține încă plus metode pentru selectarea categoriei, respectiv tipului.
- **UserDAO**: extinde clasa AbstractDAO, deci moșteneste toate metodele ei, și le convertește în metode care manipulează obiecte de tip User. Mai conține încă plus metode pentru selectarea categoriei, respectiv tipului.

- **Decorator** - corespunde design pattern-ului cu același nume. și e folosit pentru notificarea utilizatorului atunci când detaliile despre el sunt modificate.

Cuprinde următoarele clase:

- Notifier: este o interfață care conține metoda care urmează să fie implementată în funcție de tipul notificării (email sau sms).
- EmailDecorator: implementează metoda send din interfața Notifier astfel încât utilizatorul să fie notificat prin email.
- SMSDecorator: implementează metoda send din interfața Notifier astfel încât utilizatorul să fie notificat prin sms
- **Model** - cuprinde clasele care reprezintă tipul obiectelor pe care le manipulăm, movie și user.
- **Com** - cuprinde clasele care realizează conexiunea cu Clientul.

Cuprinde următoarele clase:

- Server: care defines crearea unui instanță de tip server ipAddress 127.0.0.1 și ascultă la portul 5001. Cuprinde metode de pornire server și de oprire.
- ClientThread: prin intermediul acestei clase Server-ul primește request de la Client sub forma de mesaje și realizează operațiile corespunzătoare asupra bazei de date, și trimitе mesaje sau obiecte înapoi spre Client.

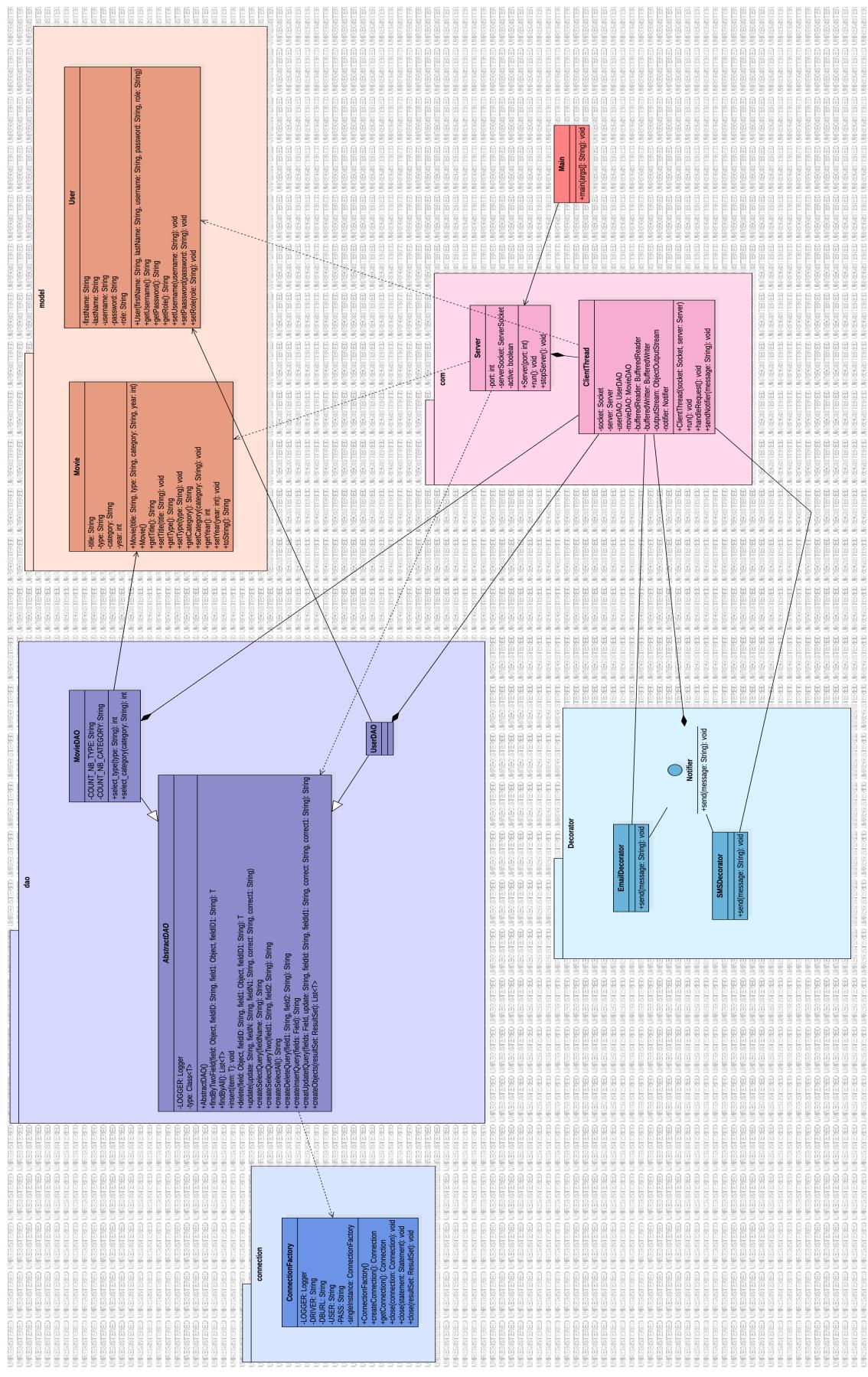


Fig 3.2.2. Diagrama de clase pentru aplicația Server

3.3 Baza de date

movies	
movieID	INT
title	VARCHAR
type	VARCHAR
category	VARCHAR
year	INT

users	
userID	INT
firstName	VARCHAR
lastName	VARCHAR
username	VARCHAR
password1	VARCHAR
role	VARCHAR

Fig 3.3.1 Baza de date moviehouse

Pentru a realiza baza de date am folosit **phpMyAdmin** unde am creat o baza de date intitulată **moviehouse** care cuprinde două tabele **users** și **movies**. Aceste tabele se află în relație directă cu clasele **user** și **movie** din pachetul **model**. Tabela **users** cuprinde câmpurile :**userID**, **firstName**,**lastName**,**username**,**password1**,**role**. Tabela **movies** cuprinde câmpurile :**movieID**,**title**,**type**,**category**,**year**. Între cele 2 tabele nu există relație ,deoarece nu au legatură una cu cealaltă.

3.4 Design pattern-lee folosite

- **Singleton**: un design pattern structural, restricționează instanțierea unei clase la o singură instanță "unică". S-a folosit pentru a crea o singură instanță la conexiunea cu baza de date .
- **Proxy**: un design pattern creational, permite să crearea unui înlocuitor sau un substituent pentru un alt obiect.. S-a folosit pentru implementarea metodei de **LogIn**.
- **Decorator**: un design pattern creational, permite adăugarea comportamentului unui obiect individual, dinamic, fără a afecta comportamentul altor obiecte din aceeași clasă. S-a folosit pentru a implementa metodele de notificare a utilizatorului când î-se modifica datele.
- **Strategy** : un design pattern comportamental, care permite implementarea fiecărui algoritm în clase separate și selectarea unuia în timpul execuției. S-a folosit pentru a implementa tipul de fisiere care pot fi scrise: **xml**, **json**, **csv**.

- **Observer** : un design pattern comportamental, menține o listă a dependentilor săi, numiți observatori, și îi notifică automat cu privire la orice schimbare de stare. S-a folosit pentru notificarea interfețelor grafice în momentul schimbării stării unui obiect.
- **Builder**: un design pattern creațional, este folosit pentru a separa construcția unui obiect de reprezentarea sa, astfel procesul de construcție putând genera și alte reprezentări. S-a folosit la nivelul obiectelor de tip User și Movie.
- **Adapter** : un design pattern structural, permite utilizarea interfeței unei clase existente ca altă interfață. S-a folosit pentru a converti o lista de filme în charturi.

4. Descrierea aplicației

Primele filme au revoluționat viața culturală a lumii. Prin îmbunătățiri constante aduse camerei de filmat și tehnicielor de proiecție, filmul a devenit cel mai important mediu de comunicare al secolului al XXI-lea, ajungând să modeleze lumea aşa cum nu a mai făcut-o nicio altă artă până la apariția sa.

Tehnologia ,în ultimi anii , a evoluat drastic,datorită apariției noului "Coronavirus" ,iar marile case de producție filme ,au fost nevoie să își mute activitatea în online ,pentru a nu pierde interacțiunea dintre administratori și angajații.

Suntem actorii unei lumi aflate în continuă ascensiune , tocmai datorită acestui fapt și noi trebuie să ne menținem potrivit așteptărilor . Așadar , pentru a eficientiza munca diferitelor case de productie de filme ,propun următoarea aplicație denumită "Movie House", pentru angajați și administrator, care are următoarele beneficii:

- Vizualizarea listei care conține filmele sortate după categorie și an.
- Filtrarea filmelor după anumite criterii: tip film,categorie film,anul realizării.
- Operații CRUD în ceea ce privește persistența filmelor.
- Căutarea unui film după titlu .
- Salvarea rapoarte/liste cu informații despre filme în mai multe formate.

- Operații CRUD pentru informațiile legate de utilizatorii aplicației.
- Vizualizarea unor statistici legate de numărul de filme dintr-o anumite categorie,sau de un anumit tip.
- Vizualizarea listei care conține utilizatori.
- Filtrarea utilizatorilor după tipul lor.
- Notificarea utilizatorilor atunci când detalile lor sunt schimbate.

Codul care stă în spatele funcționalității acestei aplicații este unul destul de simplu și basic, am încercat să scriu cât mai clar și simplu toate operațiile menționate mai sus, am împărțit aplicația în clase specifice, am respectat modelul arhitectural MVP, respectiv modelul SOLID, pentru a fi mai simplu și eficient, a modifica sau a adăuga alte funcționalități, fără să se producă modificări majore liniilor de cod existente, ci doar să adăugăm alte linii.

De asemenea, am încercat să creez o interfață unică, care să îmbine simplitatea cu stilul meu, și a ieșit ce puteți vedea mai jos, eu zic că am făcut o treabă destul de bună.:)

Pentru filtrările după anumite categorii am folosit expresiile lambda, doarece mi s-a părut o modalitatea mai ușoara și mai eficientă decât a face numeroase comparații între diferite filme.

Aplicația a fost concepută să fie utilizată de orice categorie de persoană, atât de una care are cunoștiințe în tainele programării cât și de una care acum aude pentru prima dată cuvântul "programare". Prin urmare, folosesc o interfață User-Friendly pentru a permite utilizatorilor să se simtă mai familiarizați cu programul chiar înainte de a-l fi utilizat. Printr-o interfață utilizatorul poate înțelege mai bine cum funcționează programul, poate învăța mai repede modul de utilizare a acestei aplicații. Aceasta interfață cuprinde 3 frame-uri, unul principal pentru logare, iar altele două pentru angajat, respectiv administrator.

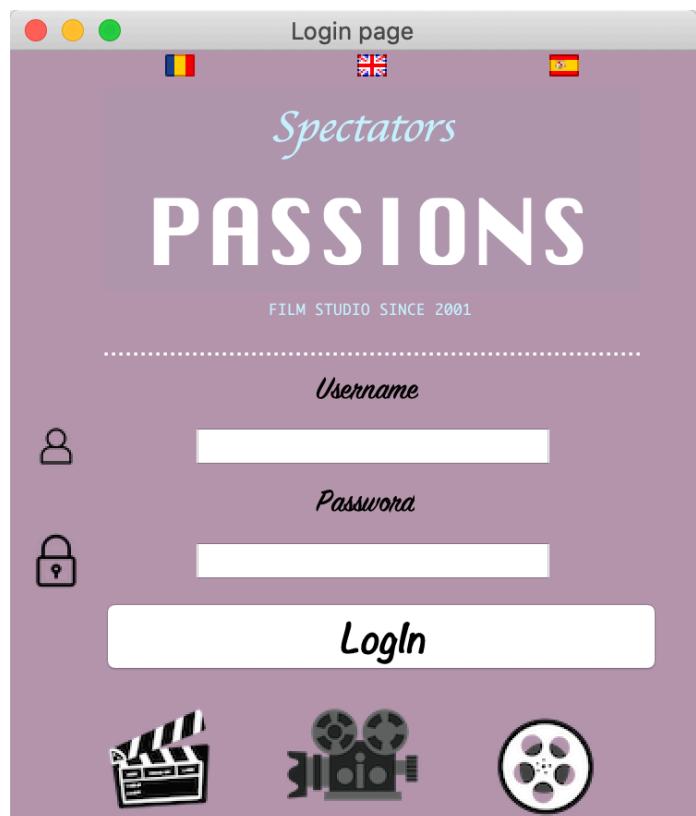


Fig4.1 Interfața pentru pagina de login

Admin page




USERS

-The best-

First Name	Last Name	Username	Password	Role
silvia	cotan	silvia	cotan	Admin
e	dd	ddddd	ddd	Admin
iulia	timis	iulia	timis	Employee

Fist Name

Last Name

Username

Password

Employee Admin





Add **Update** **Delete** **Filter**

Fig4.3 Interfața pentru pagina administratorului

Employee page





MOVIES

-Produced-

Title	Type	Category	Year
Love	Artistic	Dragoste	1990
TVD	Serial	Drama	2013
TVD	Serial	SF	8
Batman	Artistic	SF	2020
TVD	Serial	SS	7
TVD	Serial	SS	9
TVD	Serial	SS	10
TVD	Serial	SS	14
rr	Artistic	e	44444

Title

Category

Year

Artistic Serial

Filter **Search**

Add **Update** **Delete** **Save** **Statistic**

Username:iulia
Password:timis

Fig4.2 Interfața pentru pagina angajatului

4.1 Descrierea diagramelor de activități

4.1.1. Client

- **Show Users:** se preia lista de utilizatori de la server, sterge lista curentă și se creează alta prin parcursarea fiecărui utilizator și notarea detaliilor despre acesta.

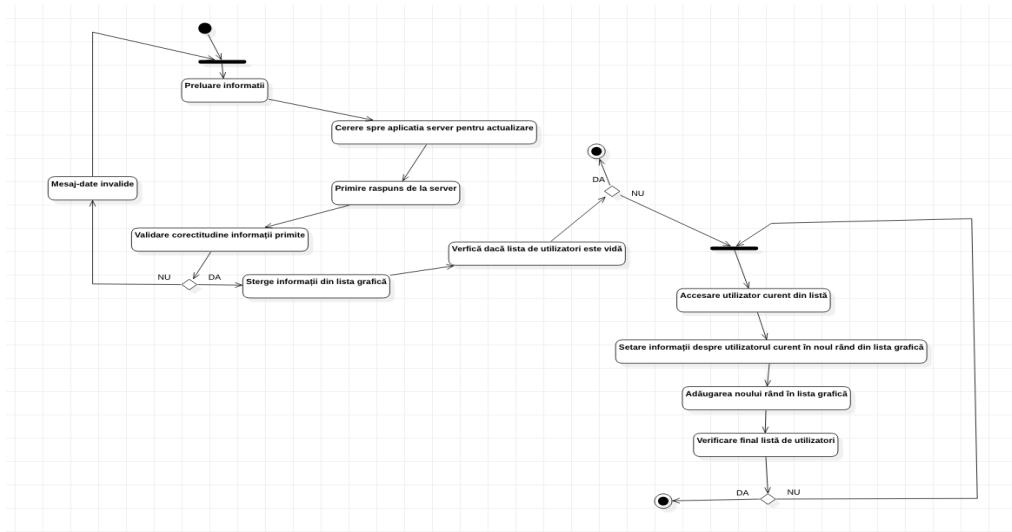


Fig.4.1.1.1 ActivityDiagram-Show Users.

- **Add User:** se preiau informațiile de la server și se afisează.

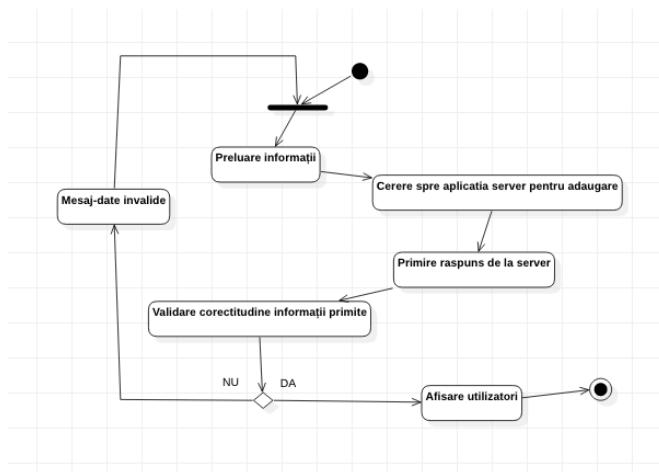


Fig.4.1.1.2 ActivityDiagram-Add User

- **Update User:** se preiau din interfață informațiile care urmează să fie actualizate pentru utilizatorul selectat, se transmit la Server după ce se preia răspunsul și se afisează lista de utilizatori.

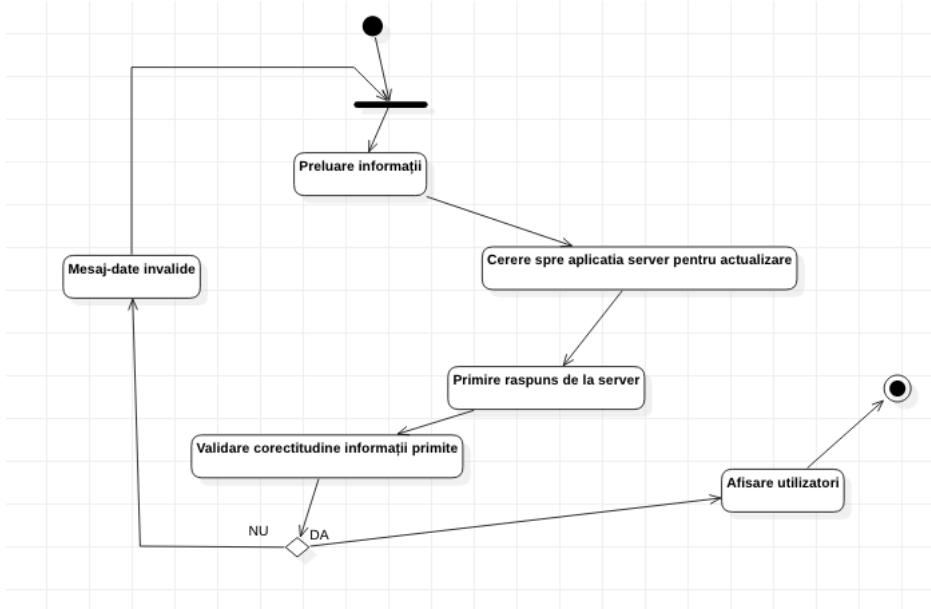


Fig.4.1.1.3 ActivityDiagram-Update User

- **Delete User:** se preia utilizatorul selectat după care se transmite cererea de stergere la server, se așteaptă răspunsul, și se afisează lista de utilizatori.

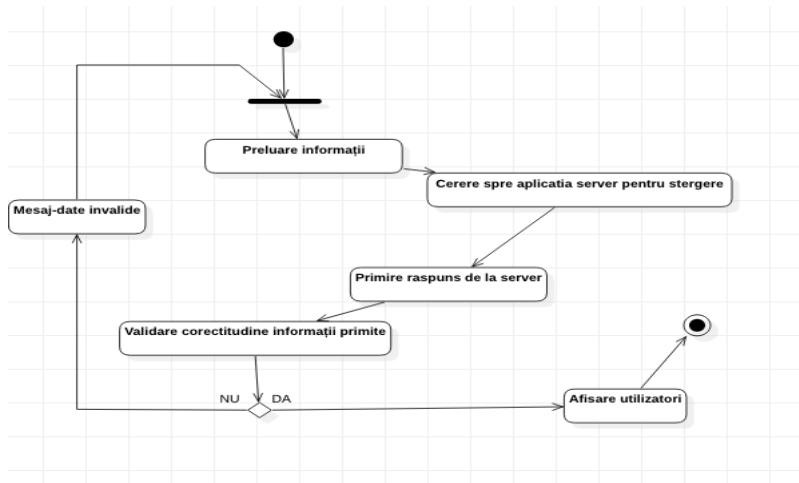


Fig.4.1.1.4 ActivityDiagram-Delete User

- **Filter Users:** se preia tipul utilizatorului după care se verifică dacă există, apoi se transmite cererea la server, se așteaptă răspuns, și se afisează lista obținută.

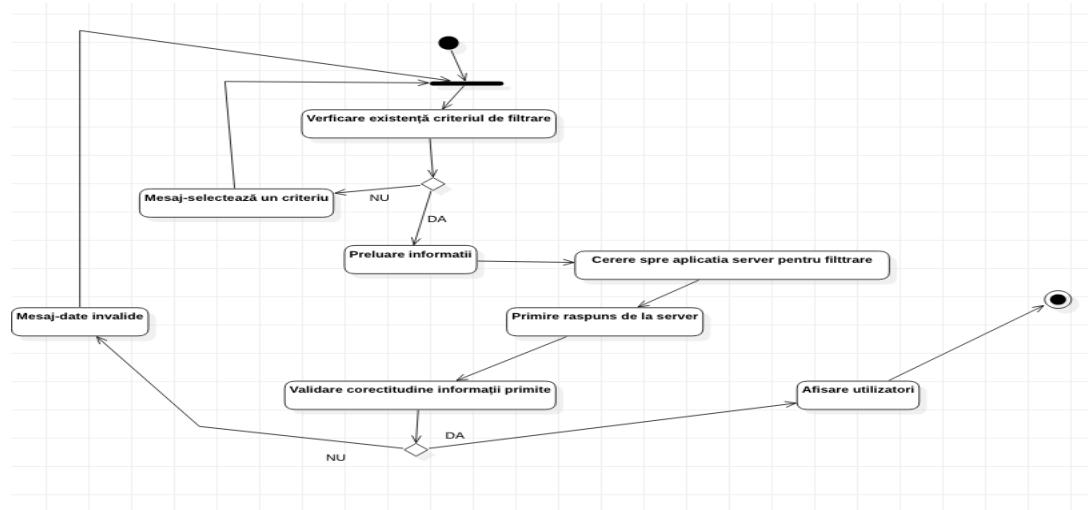


Fig.4.1.1.5 ActivityDiagram-Filter Users

- **Delete Movie:** se preia utilizatorul selectat după care se transmite cererea de stergere la server, se așteaptă răspunsul, și se afisează lista de filme.

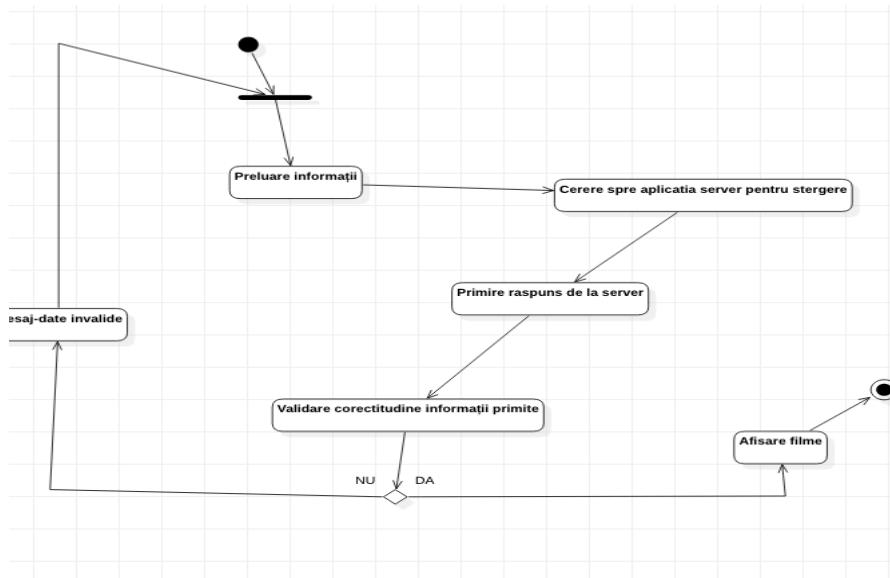


Fig.4.1.2.6 ActivityDiagram-Delete Movie

- **Show Statistics:** se apasă butonul pentru generarea statisticilor se transmite cerere spre server pentru a afla nr de filme, după care vor fi afisate în cele 3 ferestre disponibile

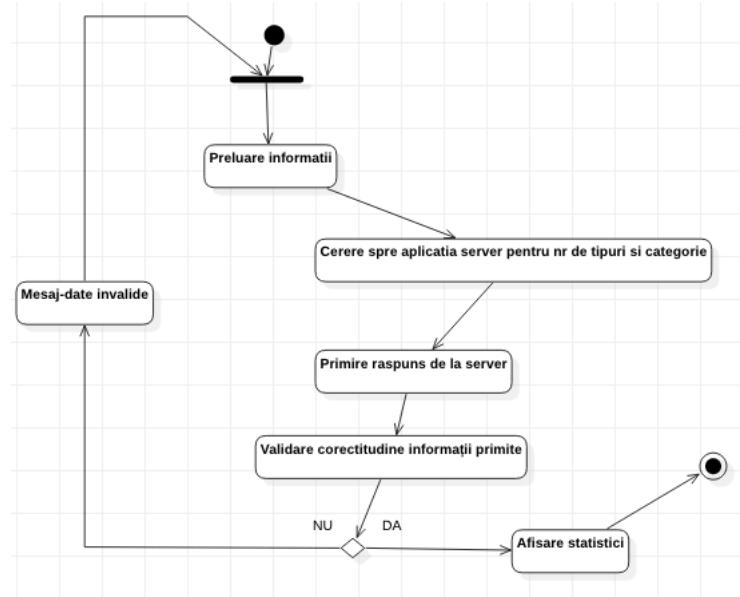


Fig.4.1.7 ActivityDiagram-Show Statistics

- **Search Movie:** se preia titlul după care se trimite cererea la server, serverul trimite lista ,după care se afisează o listă care conține toate filmele care au titlul respectiv

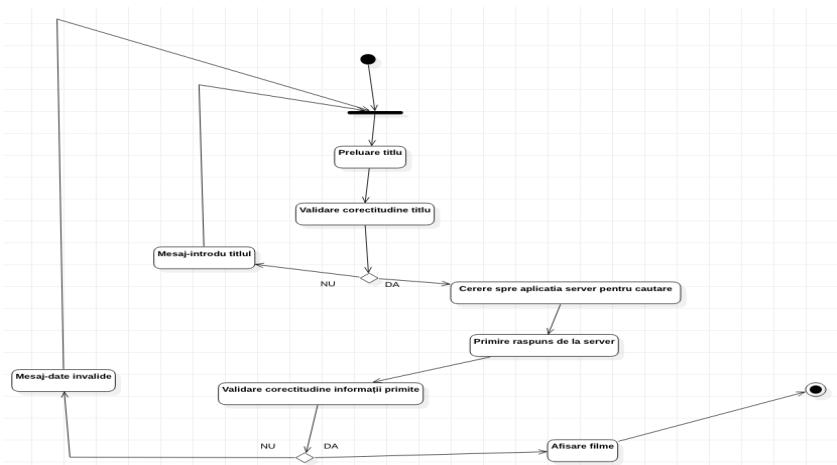


Fig.4.1.1.8 ActivityDiagram-Search Movies

- **Show Movies:** se trimite cerere spre server pentru a receptiona întreaga listă de filme, sterge lista curentă și se creează alta prin parcurgerea fiecărui film și notarea detaliilor despre acesta

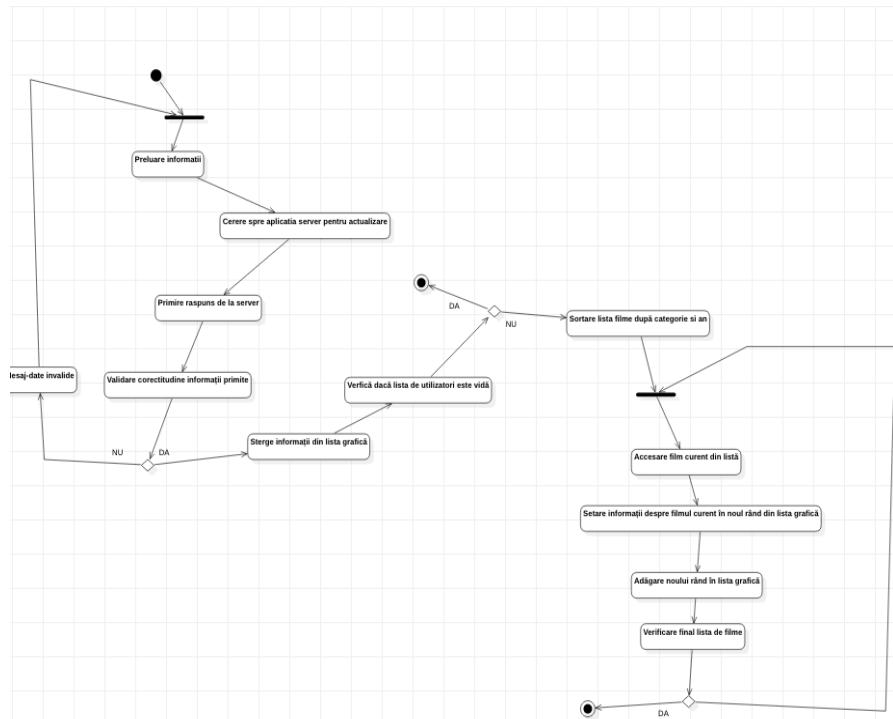


Fig.4.1.1.9 ActivityDiagram-Show Movies

- **Add Movie:** se preiau informațiile de la server și se afisează.

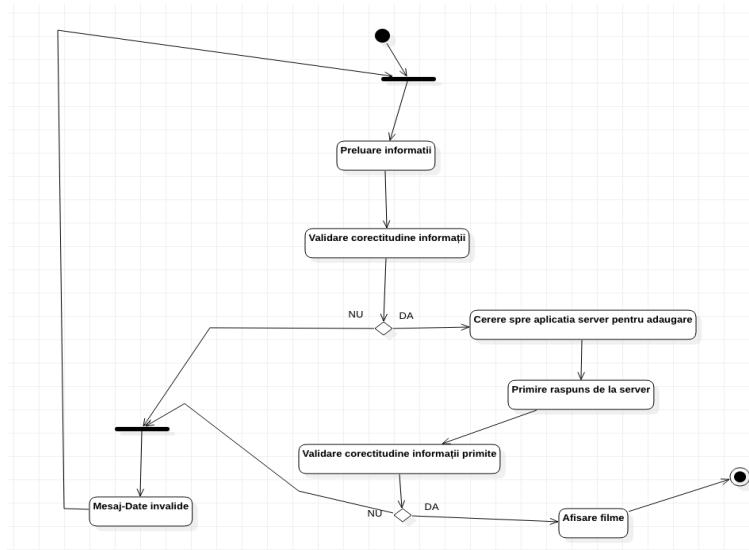


Fig.4.1.1.10 ActivityDiagram-Add Movie

- **Update Movie:** se preiau din interfață informațiile care urmează să fie actualizate pentru filmul selectat, se transmit la Server după ce se preia răspunsul și se afisează lista de filme.

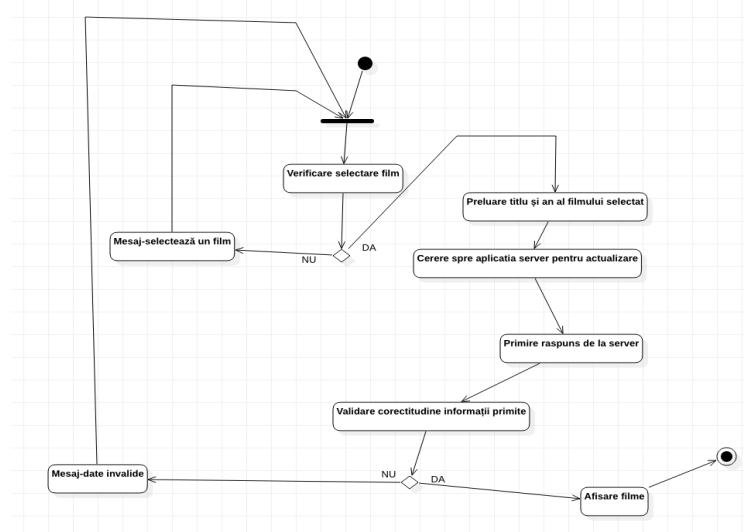


Fig.4.1.1.11 ActivityDiagram-Update Movie

- **Save Reports:** se apasă pe butonul pentru generarea de rapoarte după care se alege formatul în care să fie generat raportul

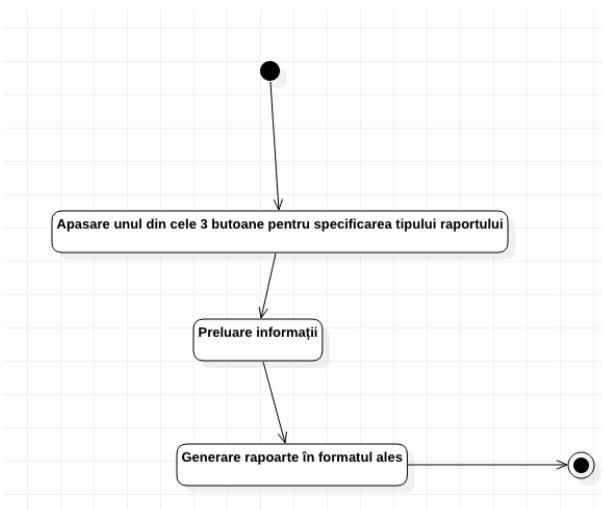


Fig.4.1.1.12 ActivityDiagram-Save Reports

- **Filter Movies:** se verifică dacă au fost selectate criterii de filtrare, apoi se transmite cererea la server, se așteaptă răspuns, și se afisează lista obținută.

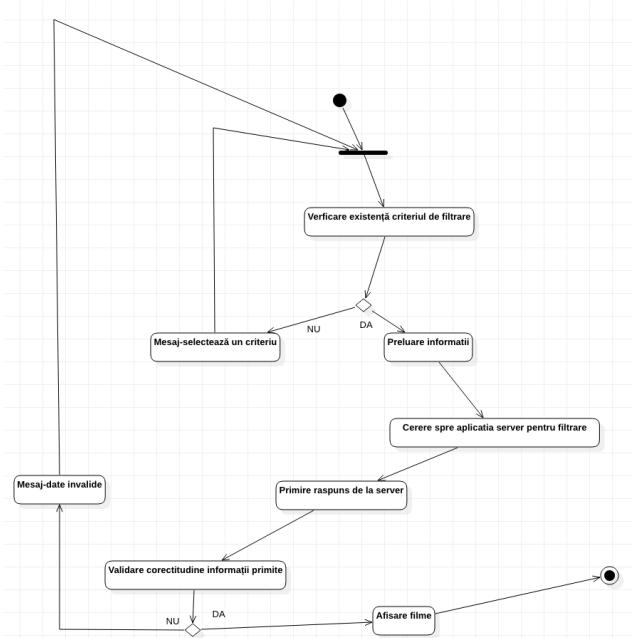


Fig.4.1.1.13 ActivityDiagram-Filter Movies

- **LogOut:** se apasă pe butonul de LogOut după care se închide fereastră curentă și se deschide o pagină de LogIn

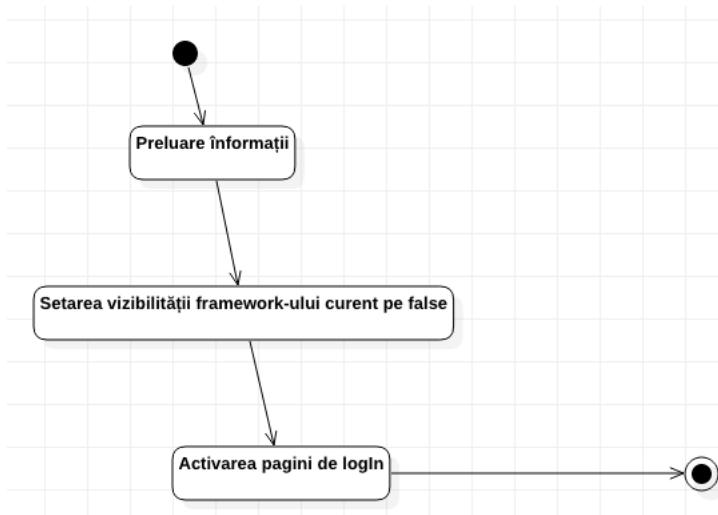


Fig.4.1.1.14 ActivityDiagram-LogOut

- LogIn:** se introduce numele de utilizator și parolă, după care se apasă pe un buton care determină inchiderea acestei interfețe și deschiderea interfeței pentru angajat/admin în funcție de tipul utilizatorului introdus

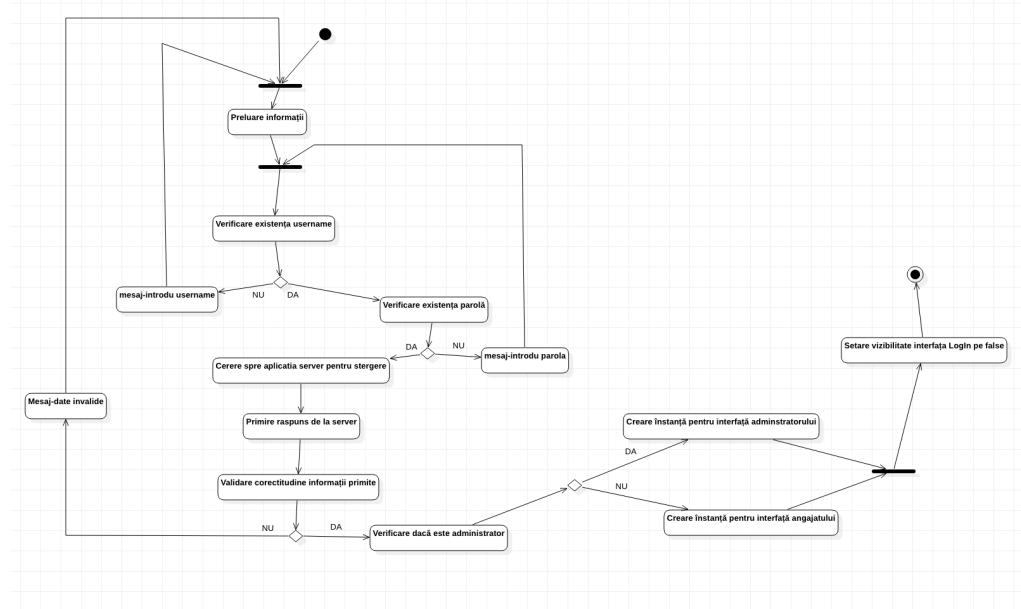


Fig.4.1.1.15 ActivityDiagram-Login

- Change Language:** se apasă pe unul din cele 3 butoane, care reprezintă cele 3 limbi disponibile, după care se notifică prin Observer ca limba a fost schimbată, astfel interfața va fi disponibilă în limba selectată

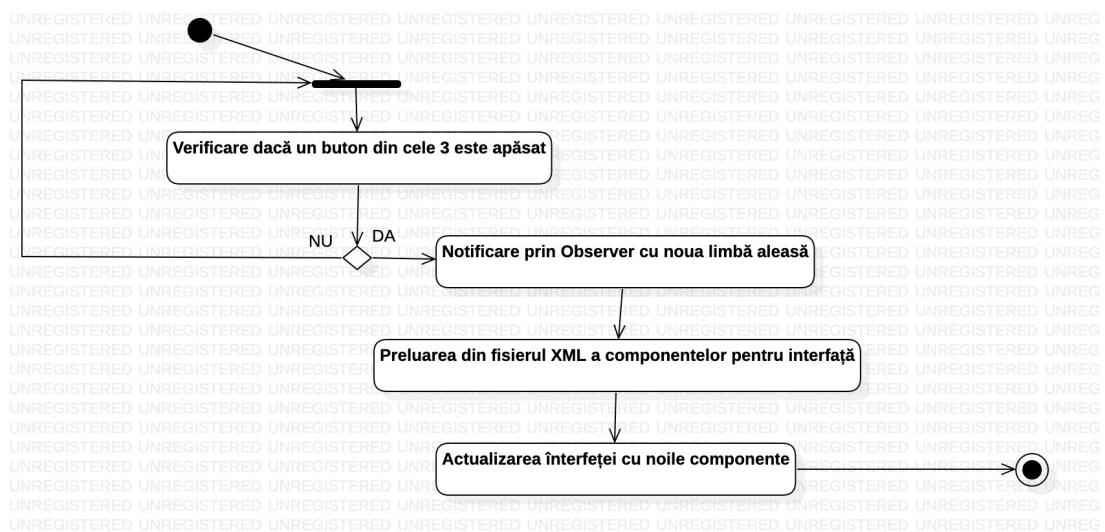


Fig.4.1.1.16 ActivityDiagram-Change language

4.1.2. Server

- Add:** se primește cererea de la client, se extrag informațiile corespunzătoare din baza de date, se verifică dacă datele sunt valide, se cauta în baza de date username-ul sau titlu (în funcție de caz), după care se crează o nouă instanță obiect, se stochează și se trimit răspuns la client.

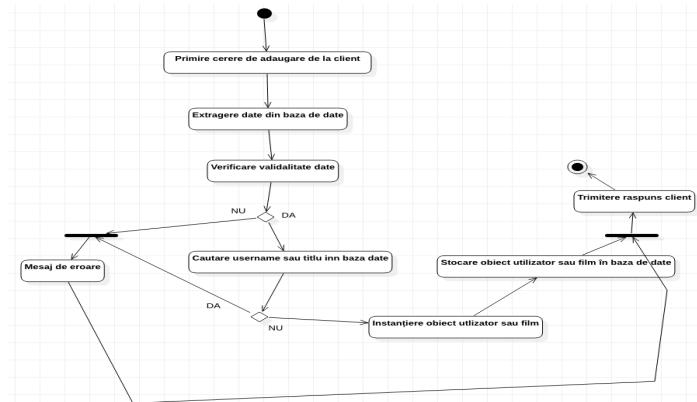


Fig.4.1.2.1 ActivityDiagram-Add

- Delete:** se primește cererea de la client, se extrag informațiile corespunzătoare din baza de date, se verifică dacă datele sunt valide, se cauta în baza de date username-ul sau titlu și anul (în funcție de caz), după care se sterge obiectul respectiv din baza de date, se trimit răspuns la client.

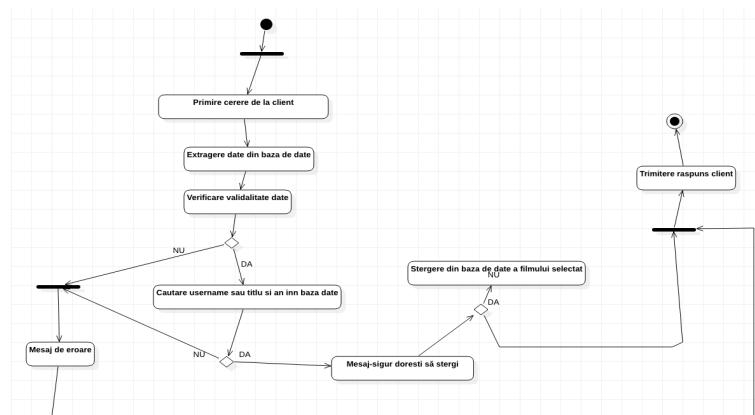


Fig.4.1.2.2 ActivityDiagram-Delete

- Update:** se primește cererea de la client, se extrag informațiile corespunzătoare din baza de date, se verifică dacă datele sunt valide, se cauță în baza de date username-ul sau titlu și anul (în funcție de caz), după care se preiau informațiile care trebuie actualizate, se verifică corectitudinea lor, se înstanțiază un nou obiect și se modifică în baza de date cel vechi, se trimit email și sms la utilizatorul care a fost modificat, și se trimit răspuns la client.

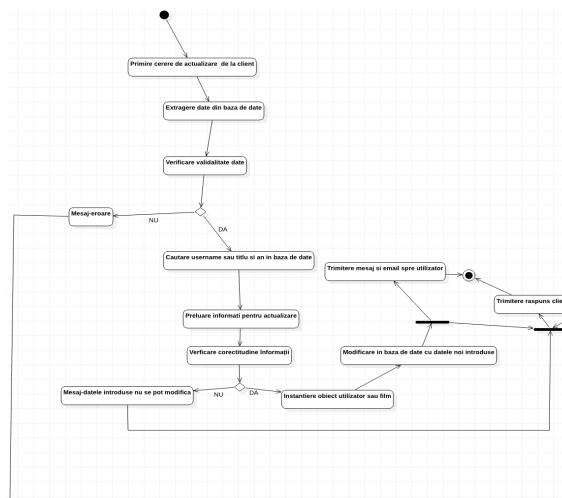


Fig.4.1.2.2 ActivityDiagram-Update

- Show:** se primește cererea de la client, se extrage lista corespunzătoare din baza de date, se verifică dacă datele sunt valide, se trimit răspuns la client.

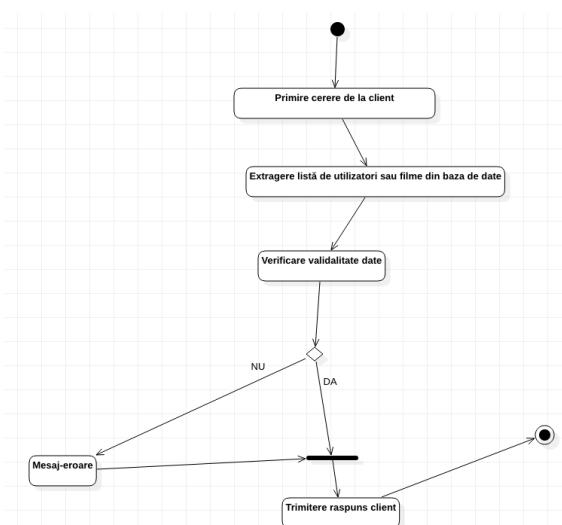


Fig.4.1.2.3 ActivityDiagram-Show

- Exist:** se primește cererea de la client, se extrag informațiile corespunzătoare din baza de date, se verifică dacă datele sunt valide, după care se caută în baza de date, dacă există utilizatorul cu parola și username-ul primit de la client, după care se trimite răspunsul corespunzător la client.

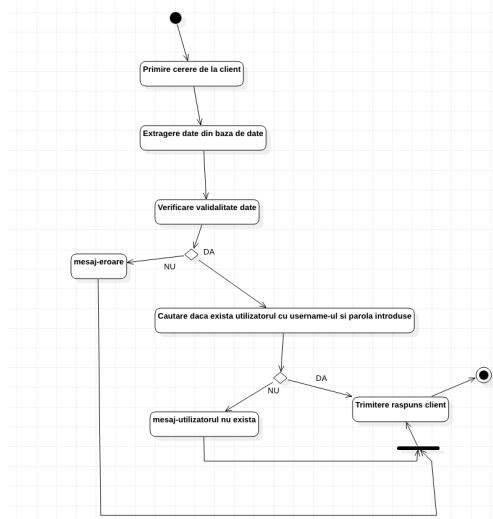


Fig.4.1.2.4 ActivityDiagram-Exist

- Select:** se primește cererea de la client, se extrag informațiile corespunzătoare din baza de date, se verifică dacă datele sunt valide, după ce se calculează căte filme sunt după categorie și tip și se trimite răspuns la client.

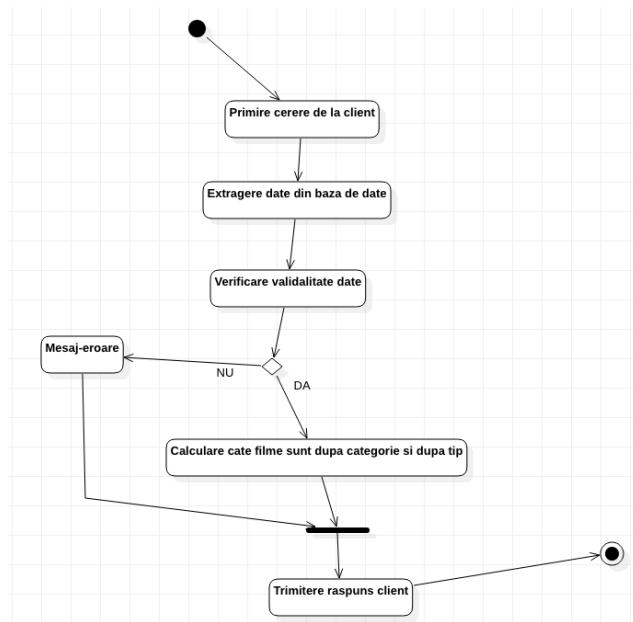


Fig.4.1.2.5 ActivityDiagram-Select

4.2 Diagrame de secvențe

- LogIn:

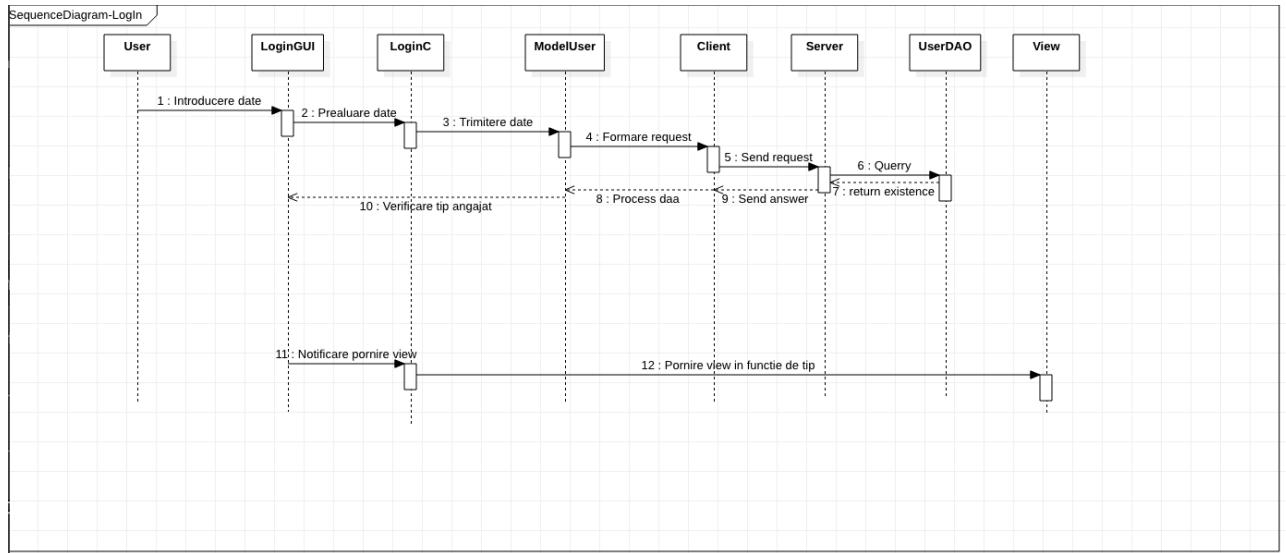


Fig.4.2.1 SequenceDiagram-LogIn

- Add

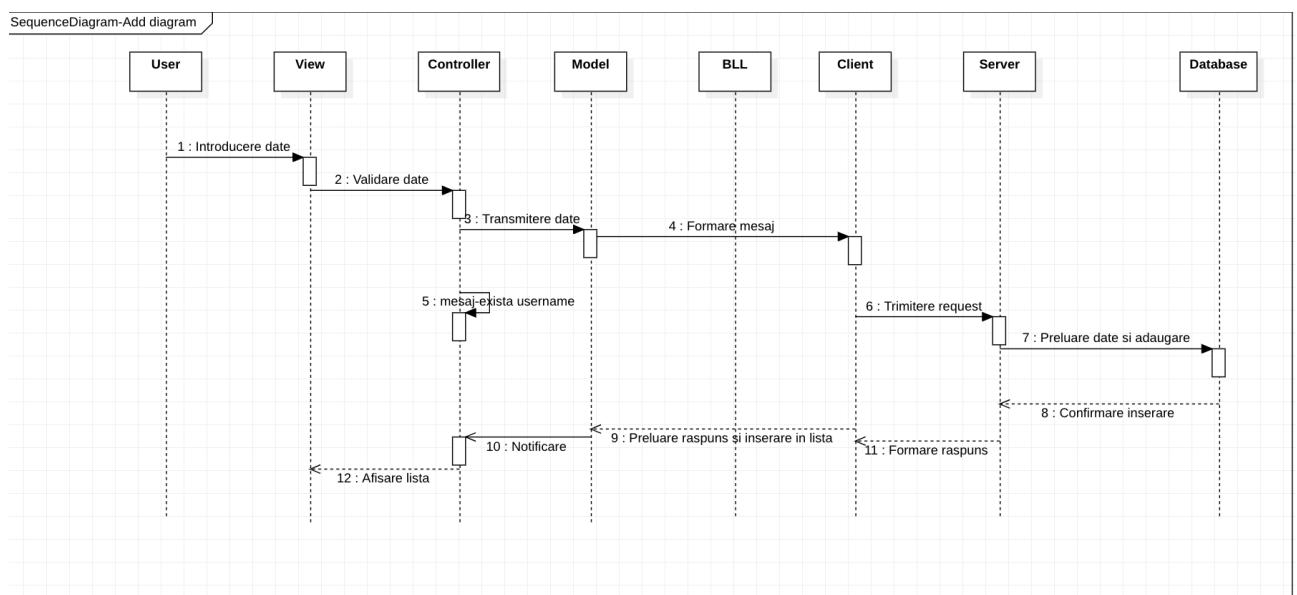


Fig.4.2.2 SequenceDiagram-Add

- ## Delete

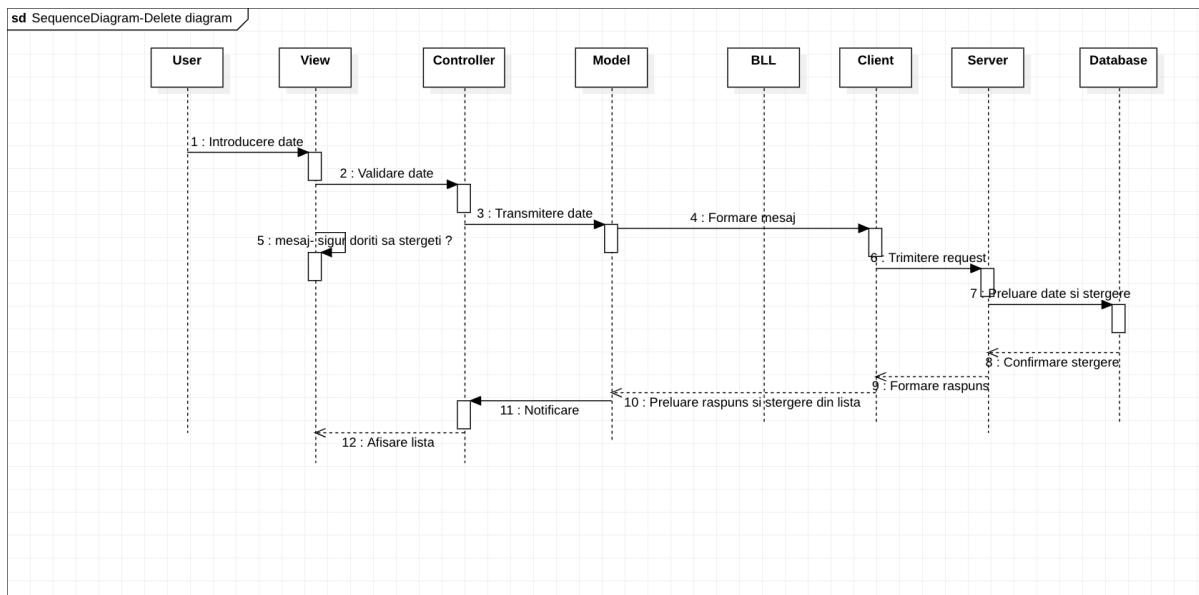


Fig.4.2.3 SequenceDiagram-Delete

- ## Update

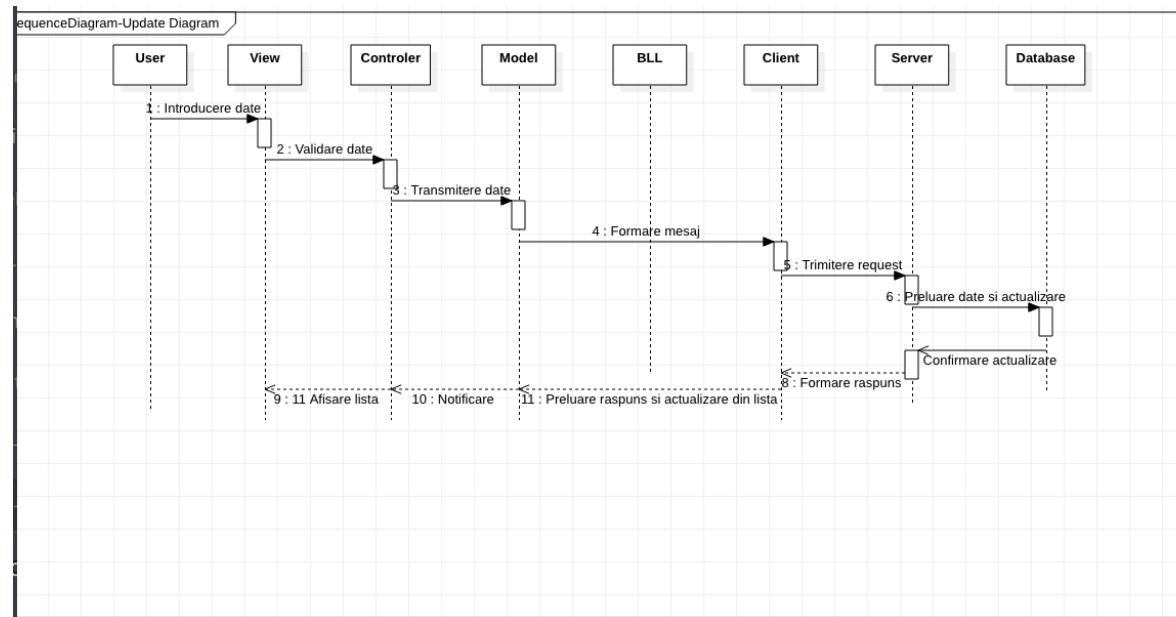


Fig.4.2.4 SequenceDiagram-Update

- ## Statistics

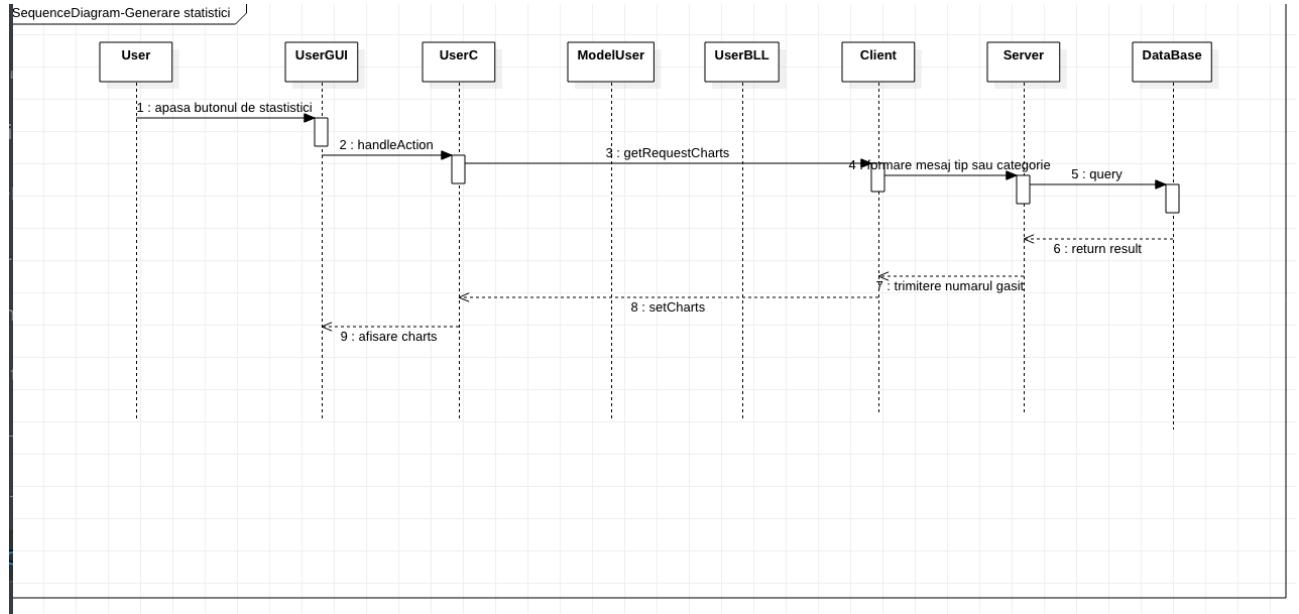


Fig.4.2.5 SequenceDiagram-Statistics

- ## Change Language

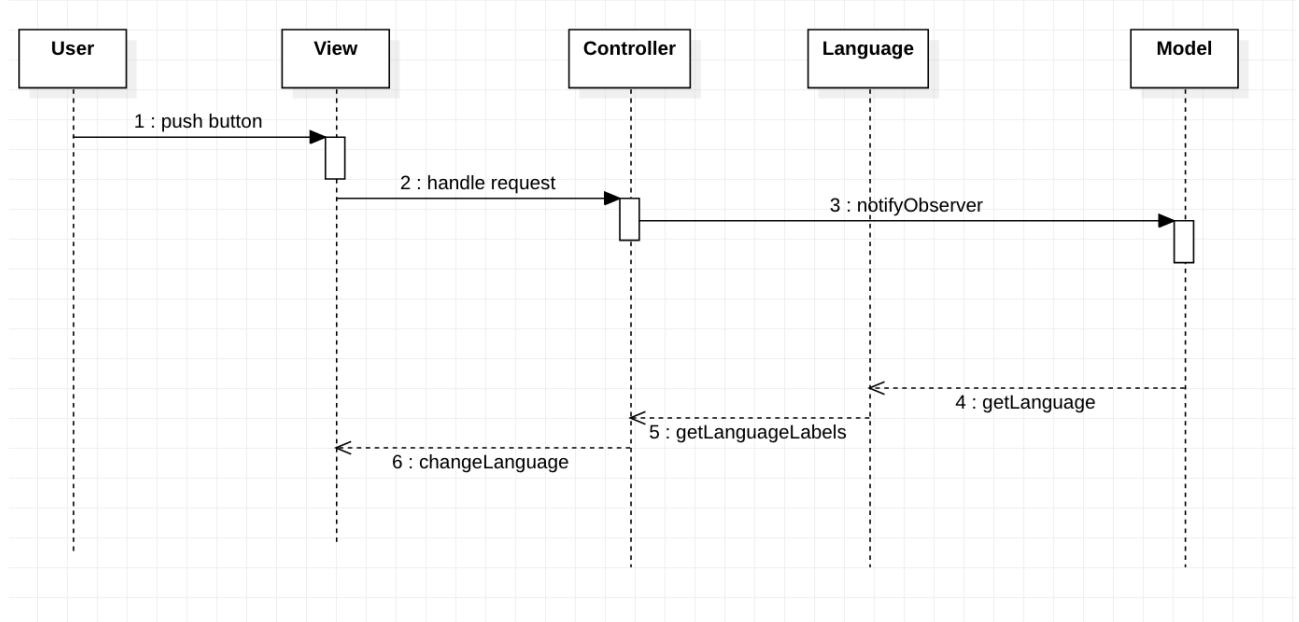


Fig.4.2.6 SequenceDiagram-Change Language

- ## LogOut

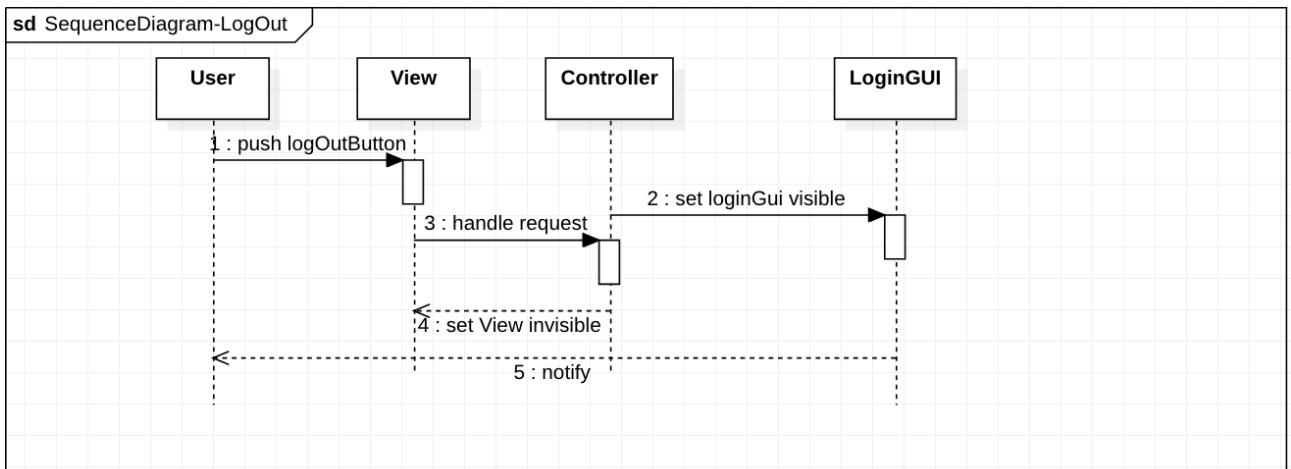


Fig.4.2.7 SequenceDiagram-LogOut

- ## Filter

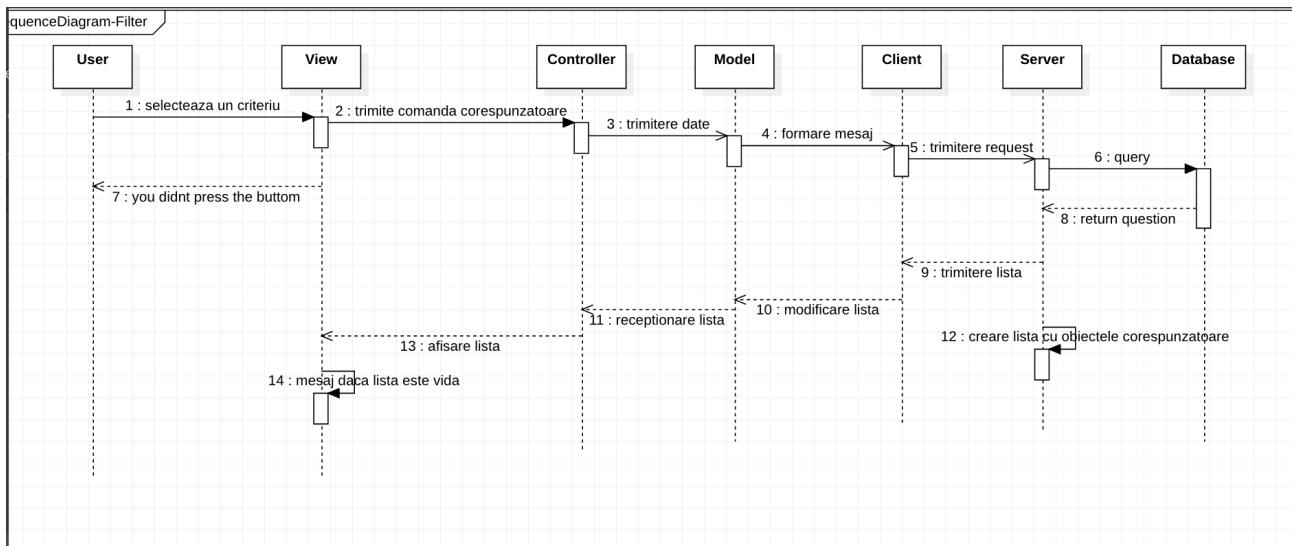


Fig.4.2.8 SequenceDiagram-Filter

- ## Search

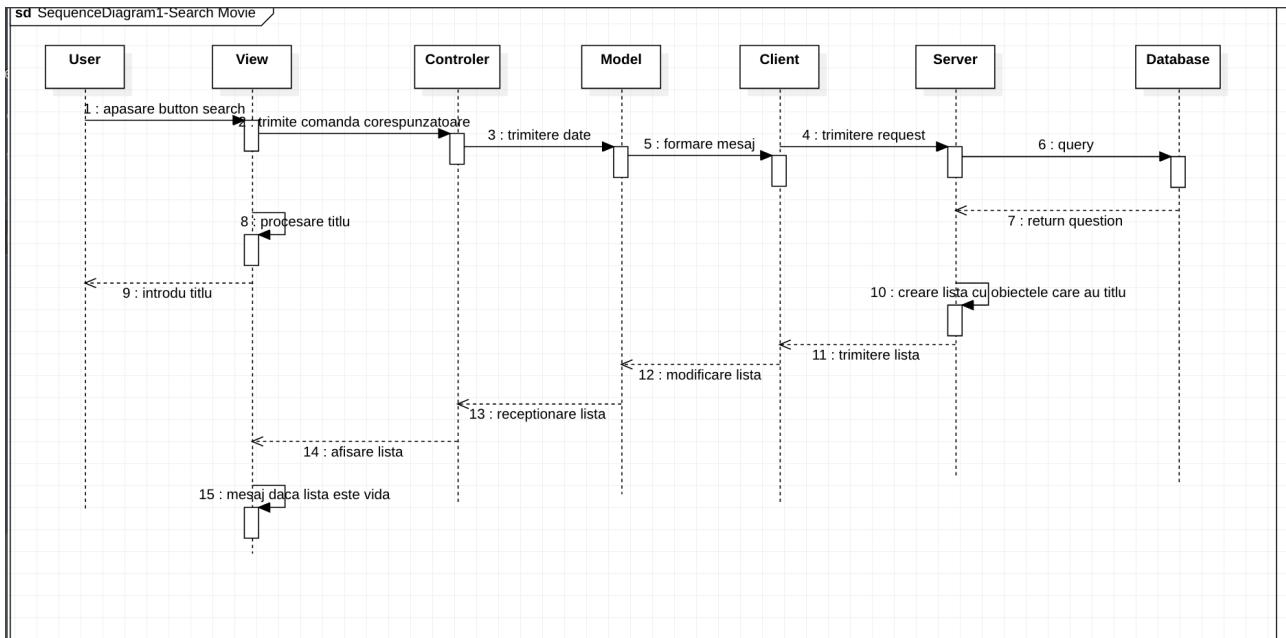


Fig.4.2.9 SequenceDiagram-Search