



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 Model-View-Controller, cu şablonul comportamental Observer și cu un şablon de proiectare creațional ales de noi, eu am ales să implementez Builder.

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 primă 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 MVP.

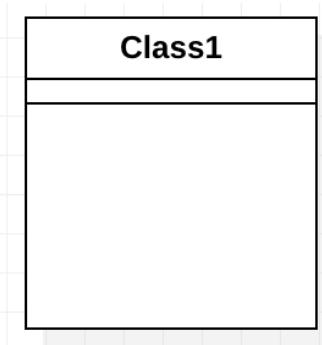


Fig2.1

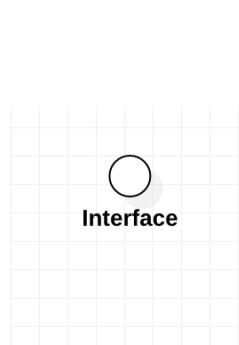


Fig2.2

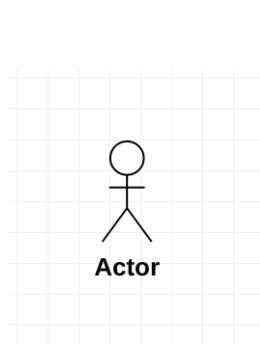


Fig2.3

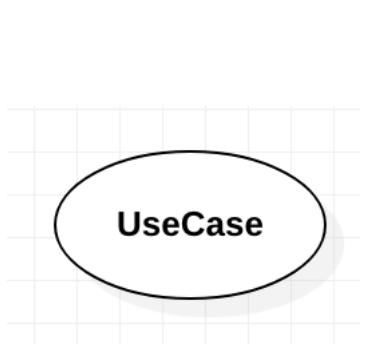


Fig2.4

După ce am realizat diagrama de use case ,am trecut la realizarea diagramelor de activitații pentru fiecare caz de utilizare în parte.

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.



Fig2.5 IntelliJ icon

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.

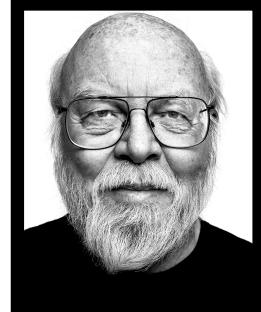


Fig2.6 James Gosling

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 : **movies** și **users**.



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 users
- 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 înafără 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 users
- 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 înafără 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

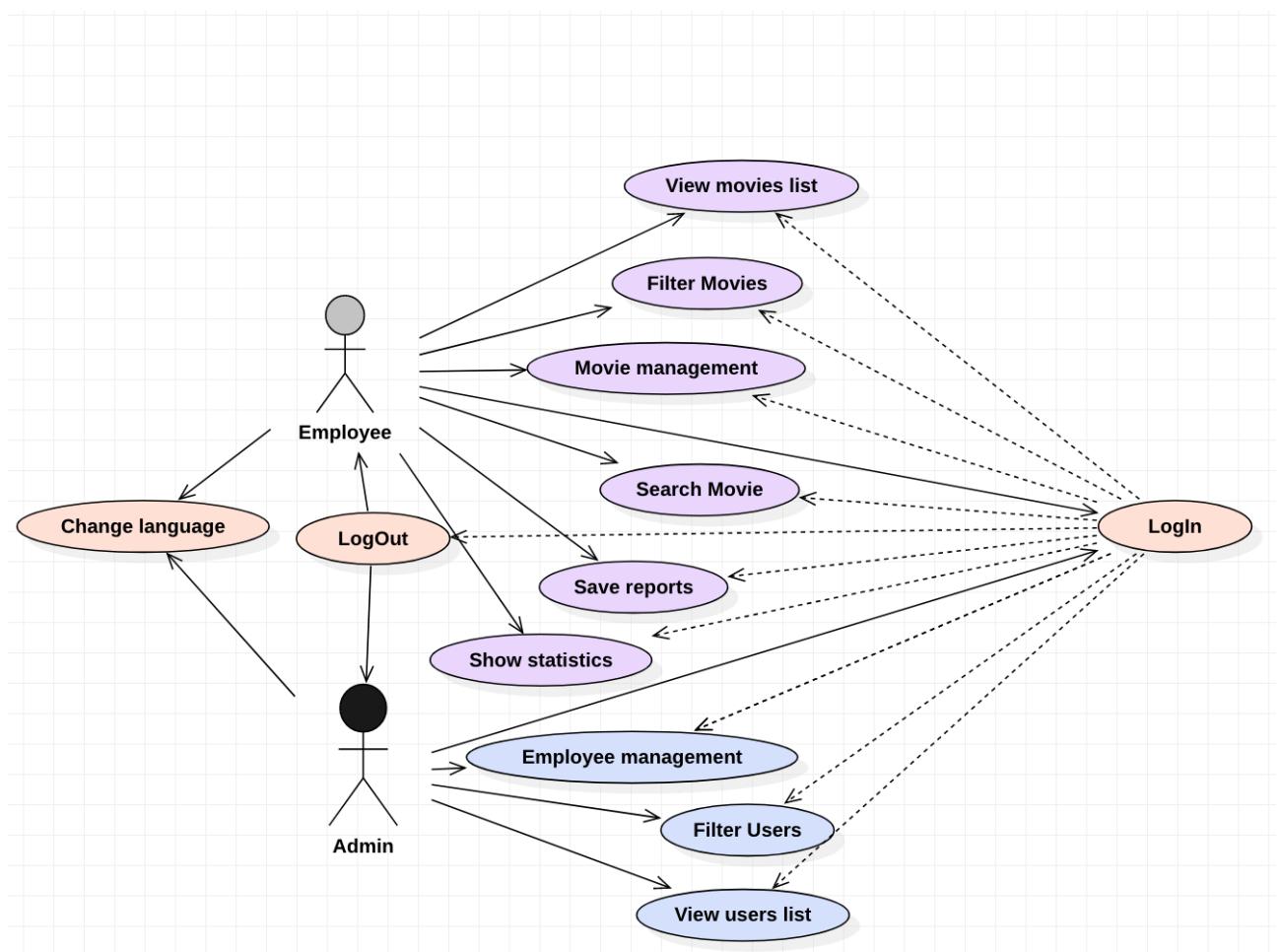


Fig 3.1 Diagrama de clase

3.2 Diagrama de clase

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

Diagrama de clase a fost realizată ,respectând modelul arhitectural Model-View-Controller și modelul comportamental Observer.Astfel fiecare pachet ,clasa ,are un rol prestabilit.

MVC este un model de arhitectură software care separă reprezentarea informațiilor din interacțiunea cu utilizatorul cu informațiile în sine.

Succesul modelului MVC se datorează izolării logicii de business față de considerentele interfeței cu utilizatorul, rezultând o **aplicație** unde aspectul vizual sau/și nivelele inferioare ale regulilor de business sunt mai ușor de modificat, fără a afecta alte nivale.

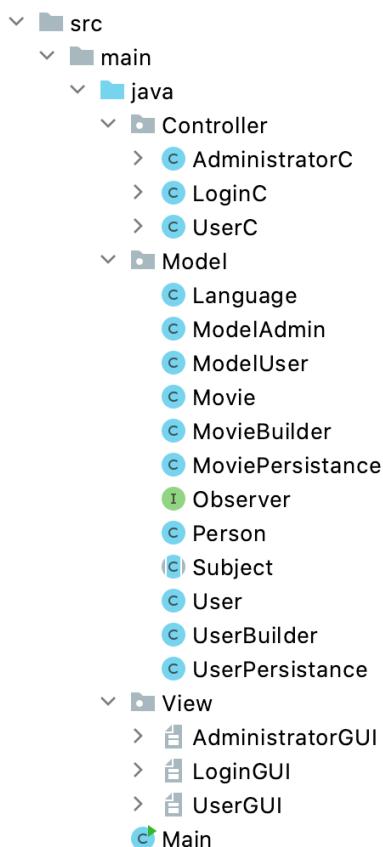


Fig 3.2.1 Pachetele cu clasele corespunzătoare

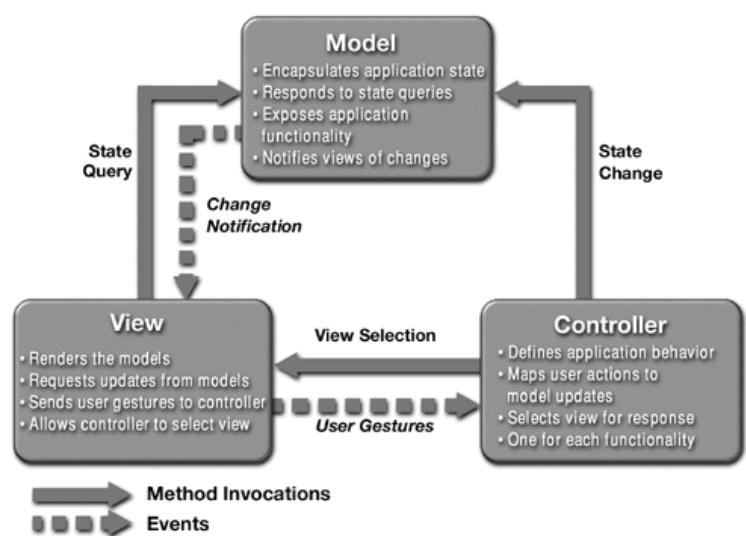


Fig 3.2.2 Modelul arhitectural MVC

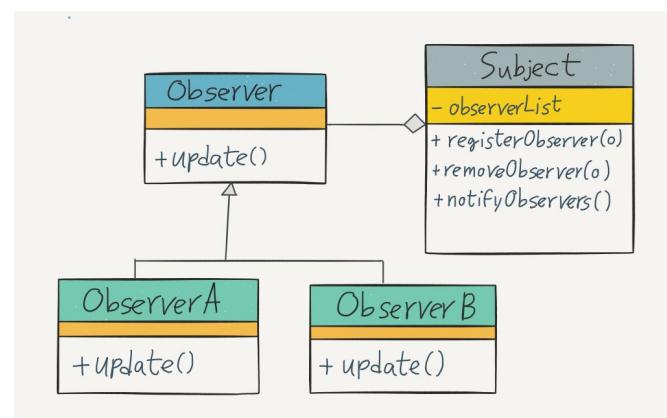


Fig 3.2.3 Modelul comportamental Observer

- **Model** - modelul este responsabil cu gestionarea datelor din aplicație. Răspunde la cereri care vin din View, deasemenea și instrucțiunilor 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 atribute necesare pentru un utilizator de tip admin: o instață de tip Language și una de tip UserPersistance.
- Model user : reprezintă clasa care face legatura cu pattern-ul Observer, și cuprinde 2 atribute necesare pentru un utilizator de tip angajat : o instață de tip Language și una de tip MoviePersistance.
- 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.
- Movie : cuprinde detalii despre film(titlu,categorie,tip,anul realizari).
- MoviePersistance : cuprinde o lista de filme asupra careia se vor realiza diferite metode care au ca scop scriere/citirea/modificarea unei baze de date .
- Person : cuprinde detalii despre o persoană din viață reală(nume și prenume).
- User : cuprinde detalii despre utilizator(username,password,role).
- UserPersistance : cuprinde o lista de utilizatori asupra careia se vor realiza diferite metode care au ca scop scriere/citirea/modificarea unui fisier de tip xml.
- UserBuilder și MovieBuilder sunt 2 clase folosite pentru implementarea sablonului creational Builder. Fieind o interfață pentru creare părților unui obiect.

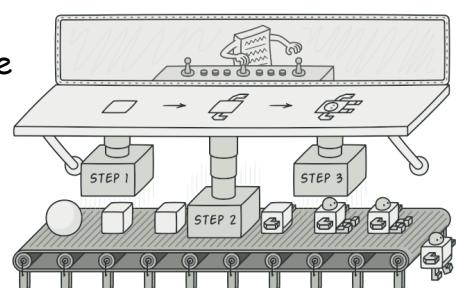


Fig 3.2.3 Modelul creational Builder

- **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 rapoartelor 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

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.

4. Descrierea aplicației

Primele filme au revoluționat viața culturală a lumii. Prin îmbunătățiri constante aduse camerei de filmat și tehniciilor 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 sa 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

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.

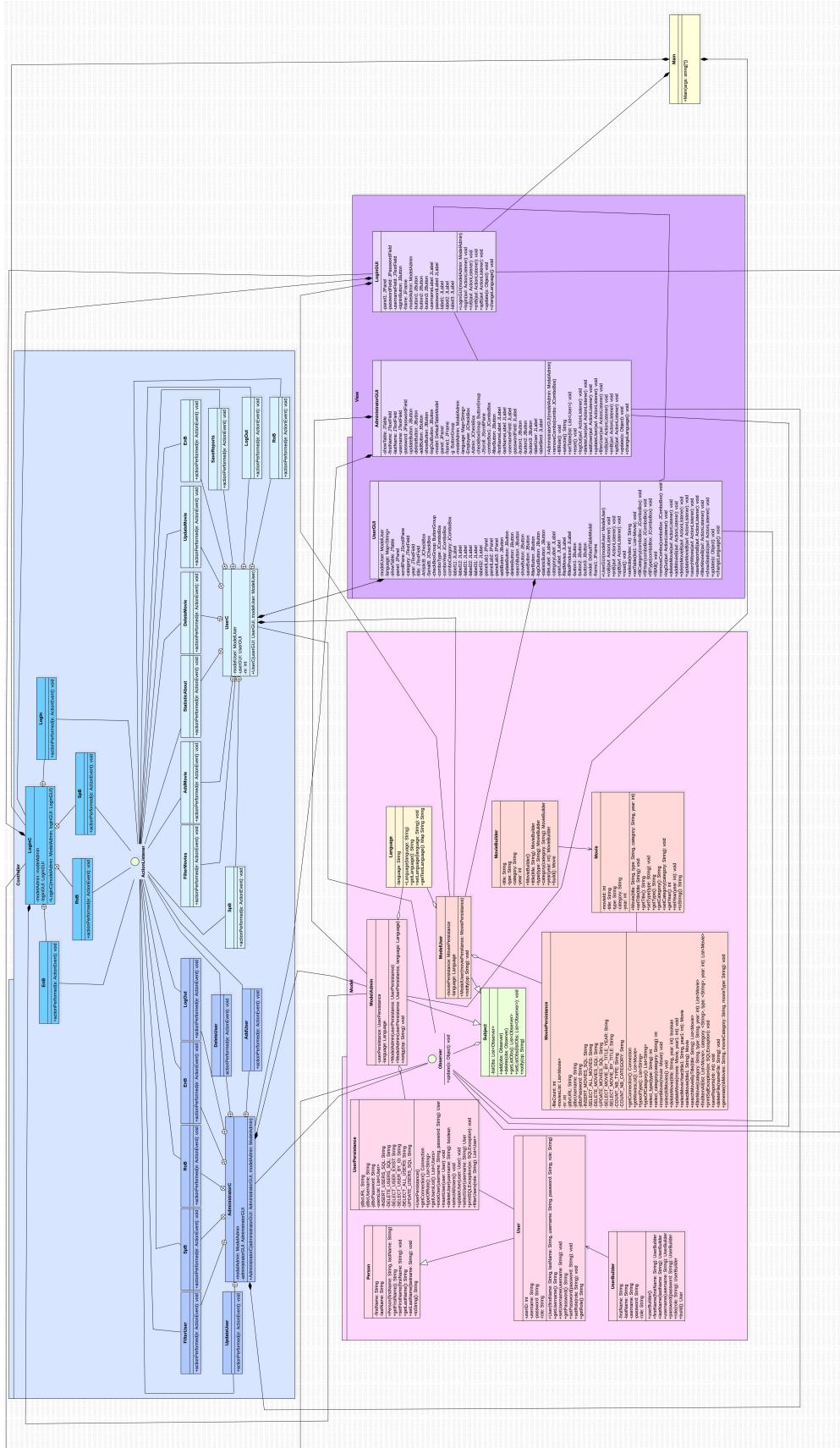


Fig 4.1 Diagrama de clase

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.

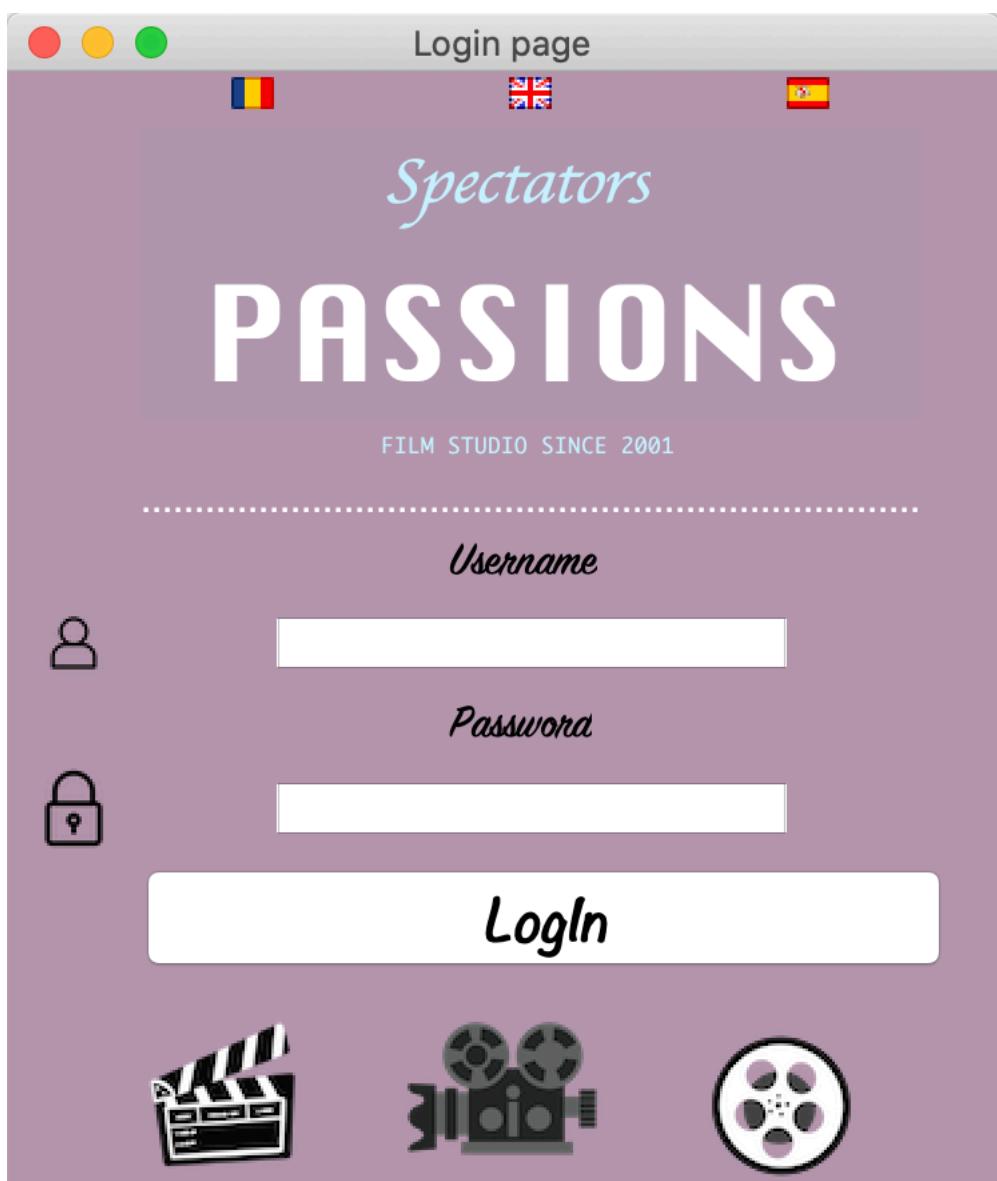


Fig4.1 Interfața pentru pagina de login

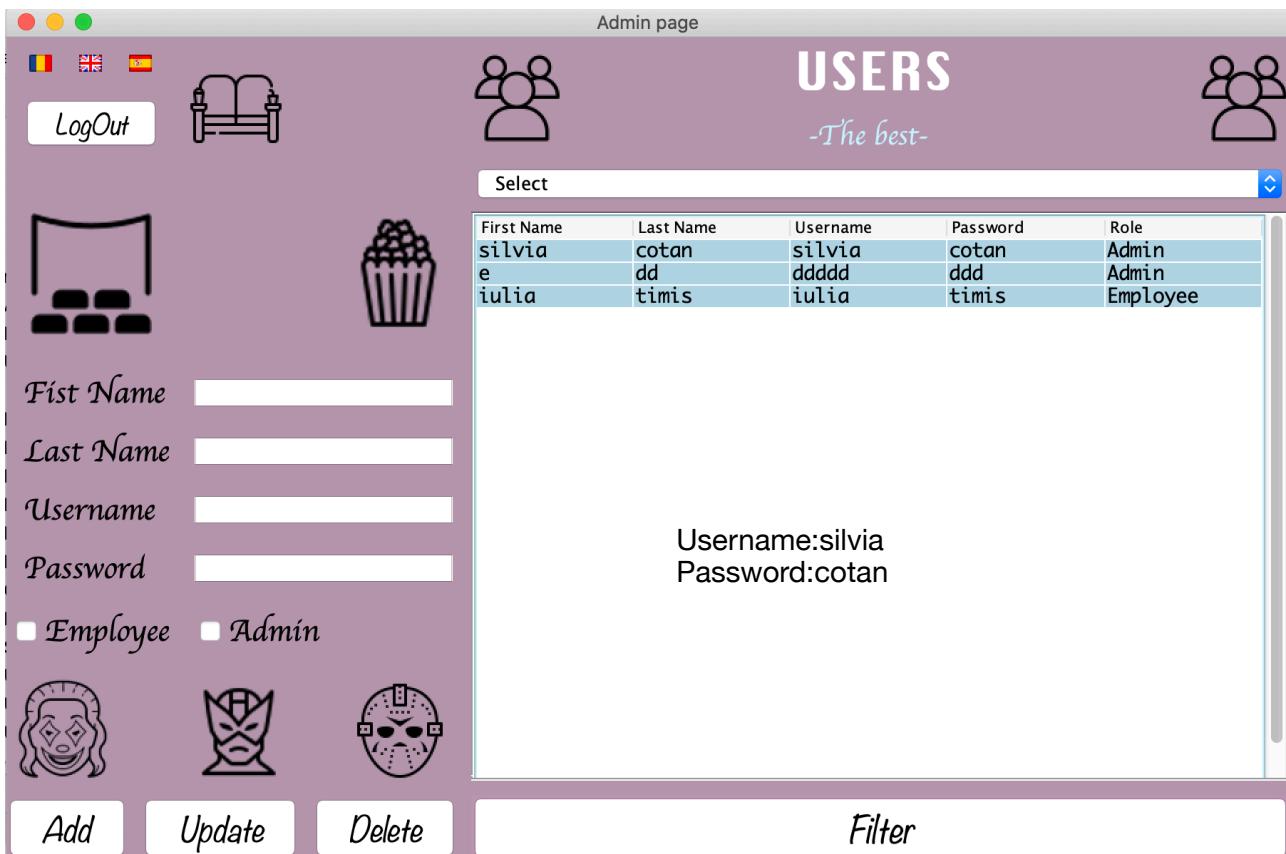


Fig4.3 Interfață pentru pagina administratorului



Fig4.2 Interfață pentru pagina angajatului

4.1 Descrierea diagramele de activități

- Show Users:** se sterge lista curentă și se creează alta prin parcurgerea fiecărui utilizator și notarea detaliilor despre acesta

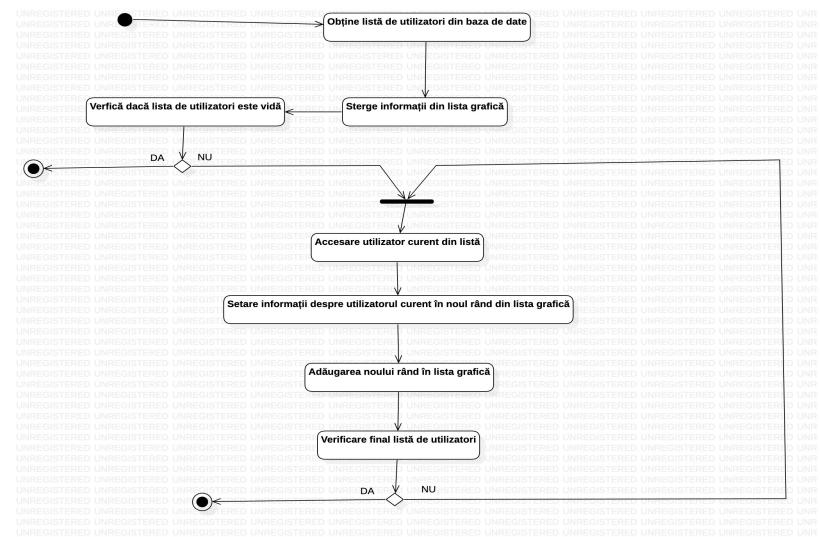


Fig.4.1.1 ActivityDiagram-Show Users.

- Add User:** se tau informațiile din interfață și se adăuga utilizatorul cu acele date

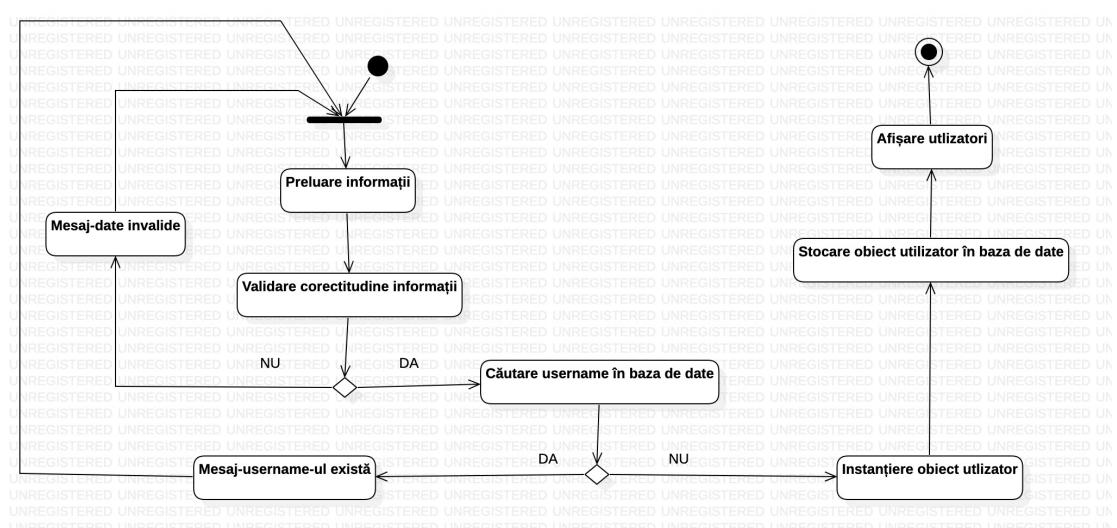


Fig.4.1.2 ActivityDiagram-Ass User

- Update User:** se preiau din interfață informațiile care urmează să fie actualizate pentru utilizatorul selectat

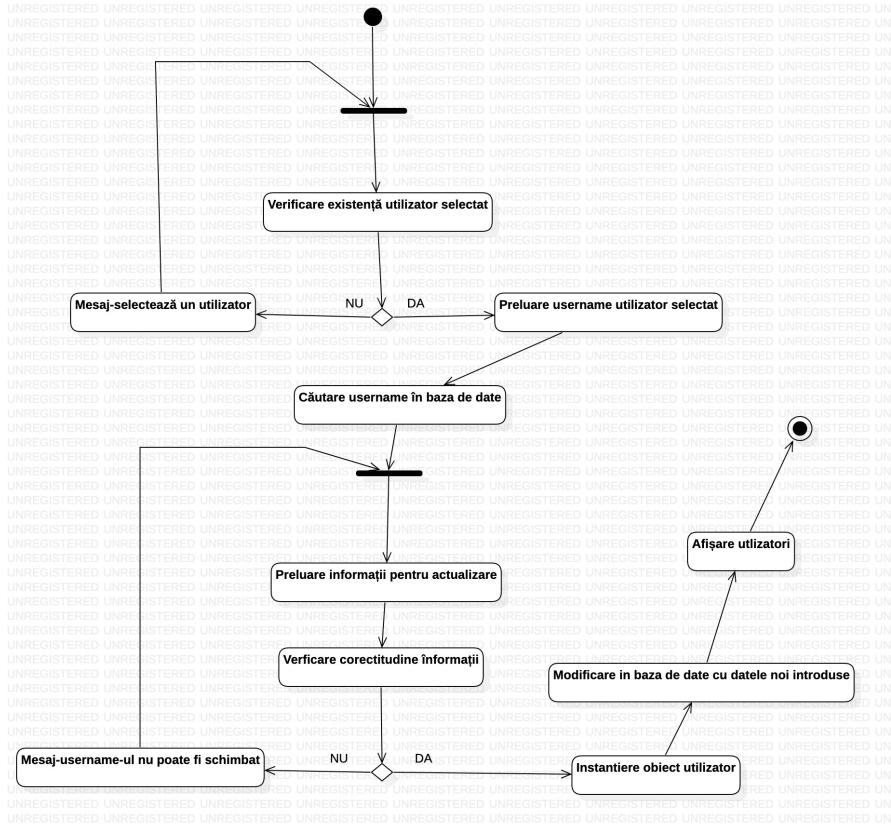


Fig.4.1.3 ActivityDiagram-Update User

- Delete User:** se preia utilizatorul selectat după care se vor sterge detaliile despre acest utilizator din baza de date

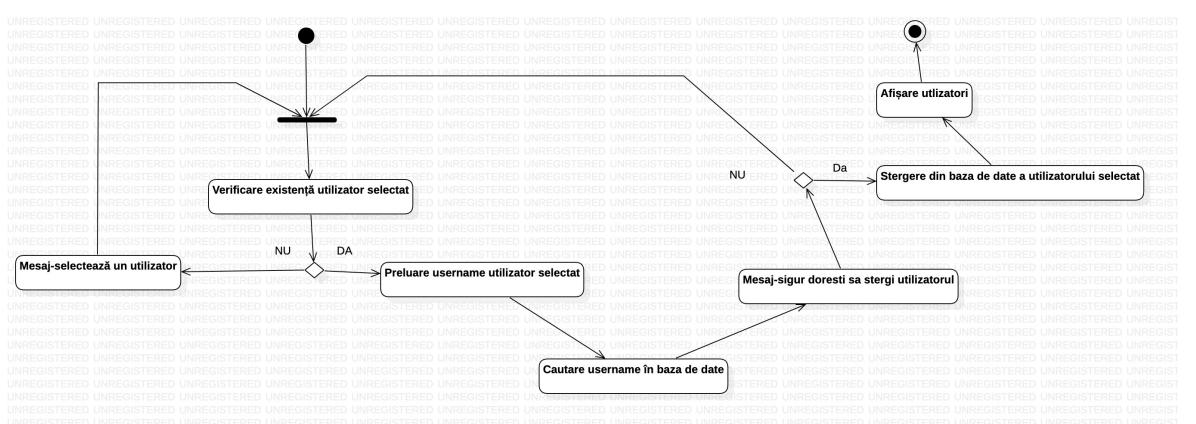


Fig.4.1.4 ActivityDiagram-Delete User

- **Filter Users:** se preia tipul utilizatorului după care se crează o lista cu utilizatori care indeplinesc criteriul selectat

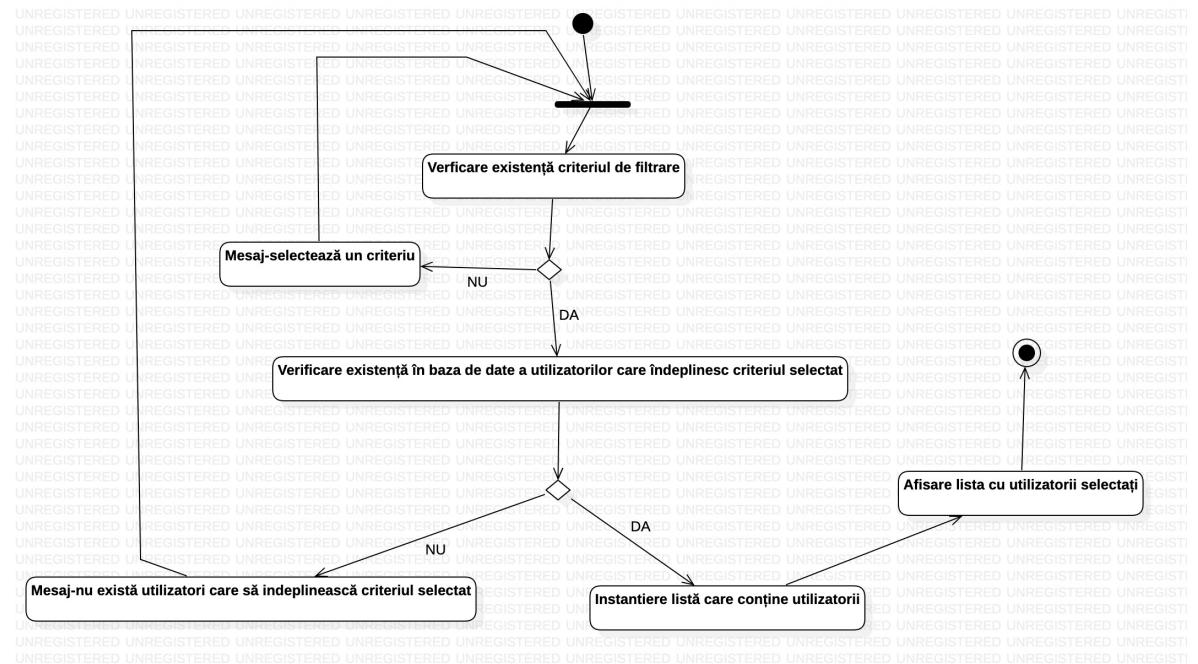


Fig.4.1.5 ActivityDiagram-Filter Users

- **Delete Movie:** se preia filmul selectat după care se vor sterge detaliiile despre acest utilizator din baza de date

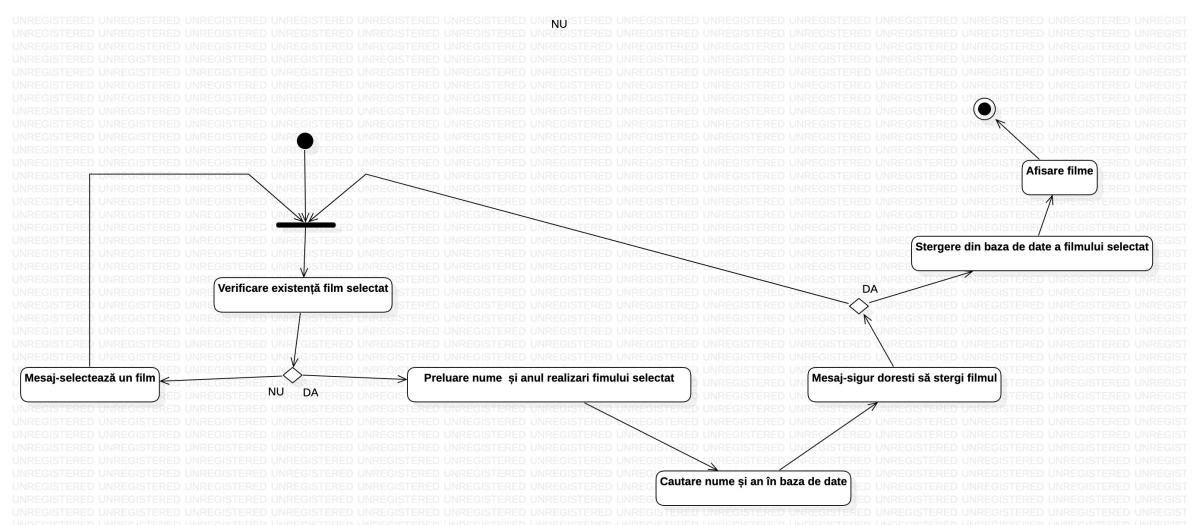


Fig.4.1.6 ActivityDiagram-Delete Movie

- **Show Statistics:** se apasă butonul pentru generarea statisticilor după care vor fi afisate în cele 3 ferestre disponibile

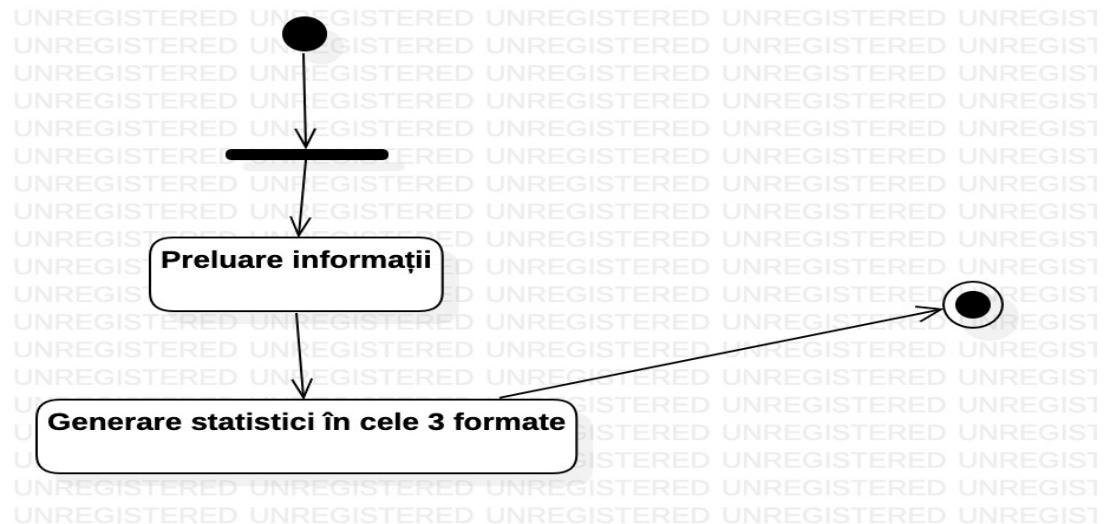


Fig.4.1.7 ActivityDiagram-Show Statistics

- **Show Movies:** se sterge lista curentă și se creează alta prin parcurgerea fiecărui film și notarea detaliilor despre acesta

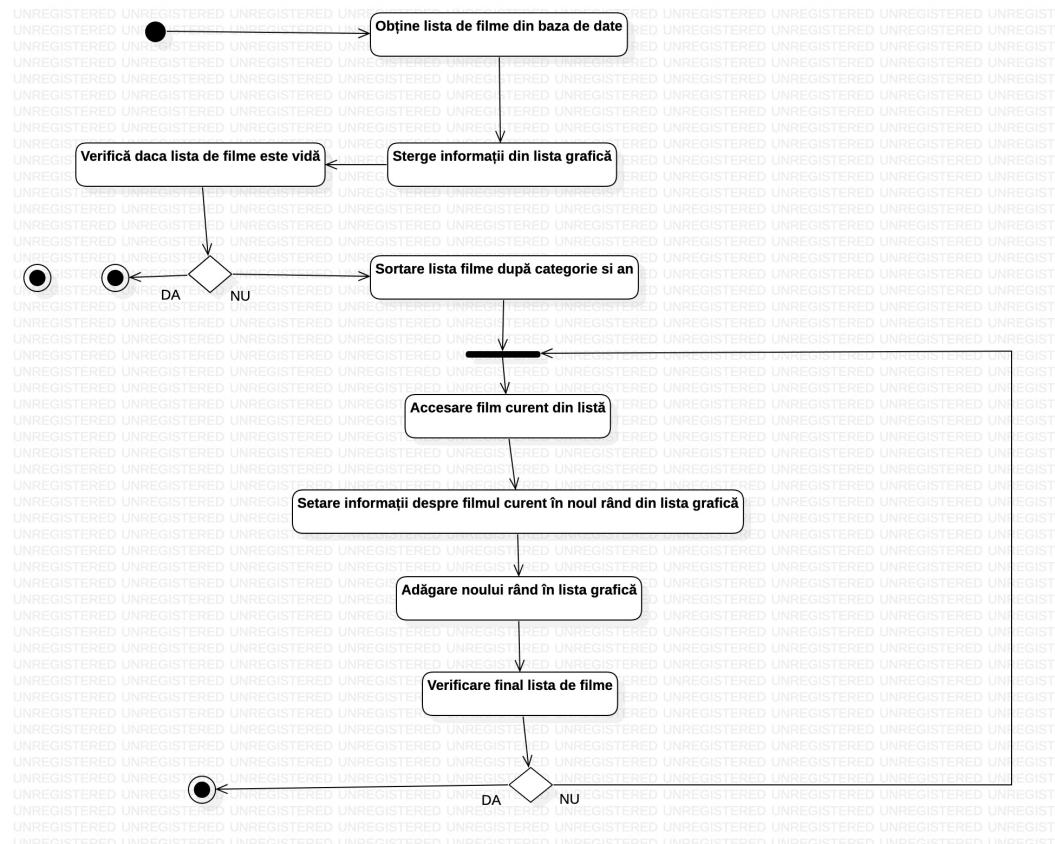


Fig.4.1.8 ActivityDiagram-Show Movies

- **Search Movie:** se preia titlul după care se caută filme, după care se afisează o listă care conține toate filmele care au titlul respectiv

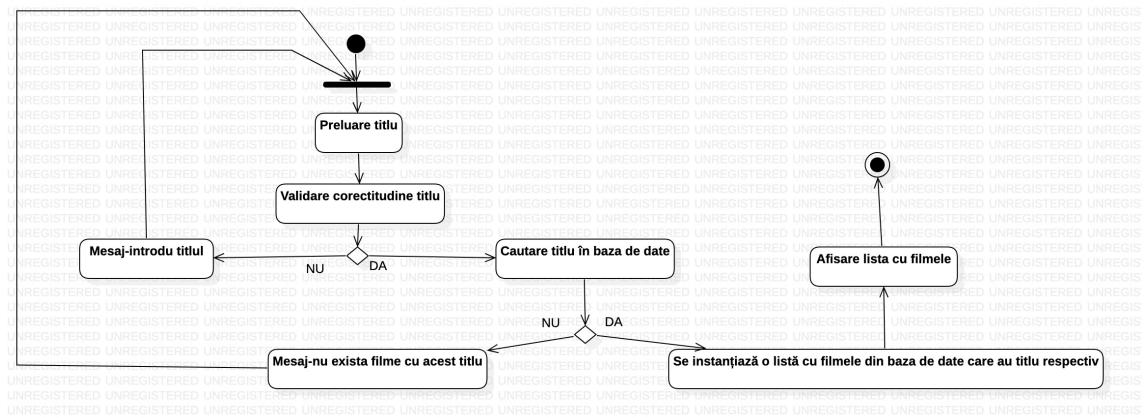


Fig.4.1.9 ActivityDiagram-Show Movies

- **Add Movie:** se tau informațiile din interfață și se adăuga filmul cu acele date

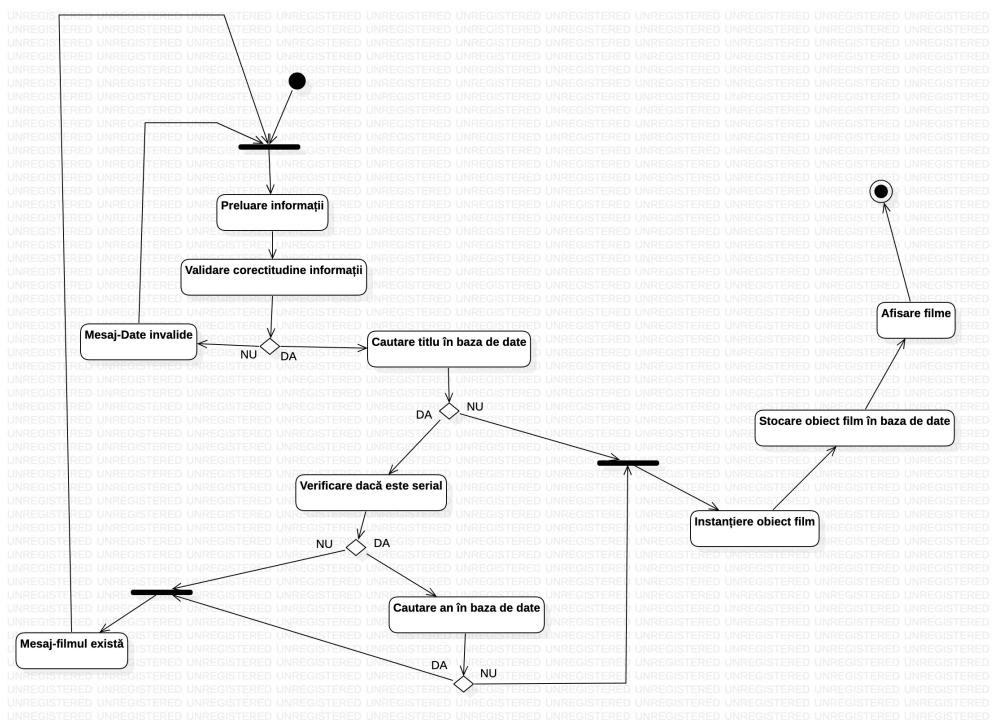


Fig.4.1.10 ActivityDiagram-Add Movie

- Update Movie:** se preiau din interfață informațiile care urmează să fie actualizate pentru filmul selectat

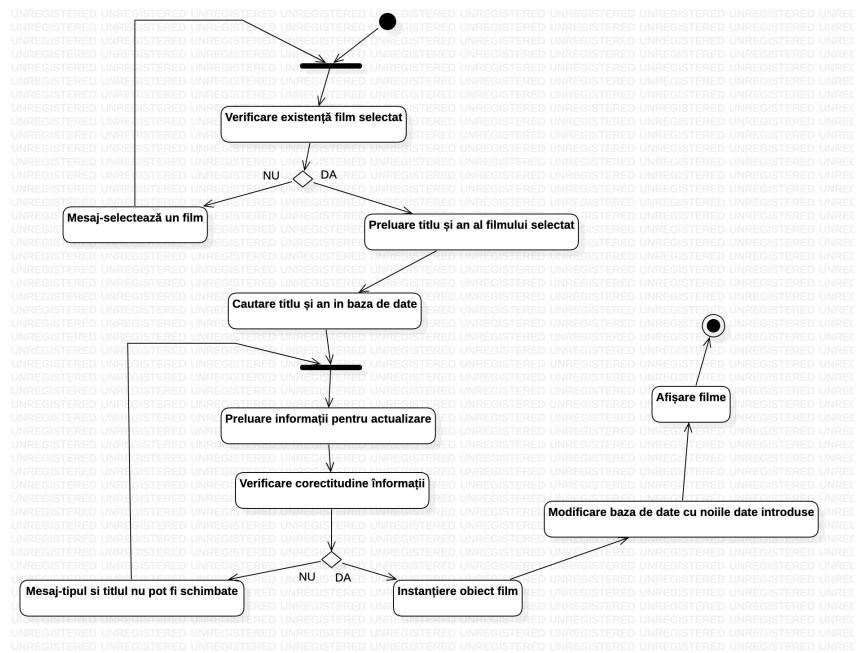


Fig.4.1.11 ActivityDiagram-Update Movie

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

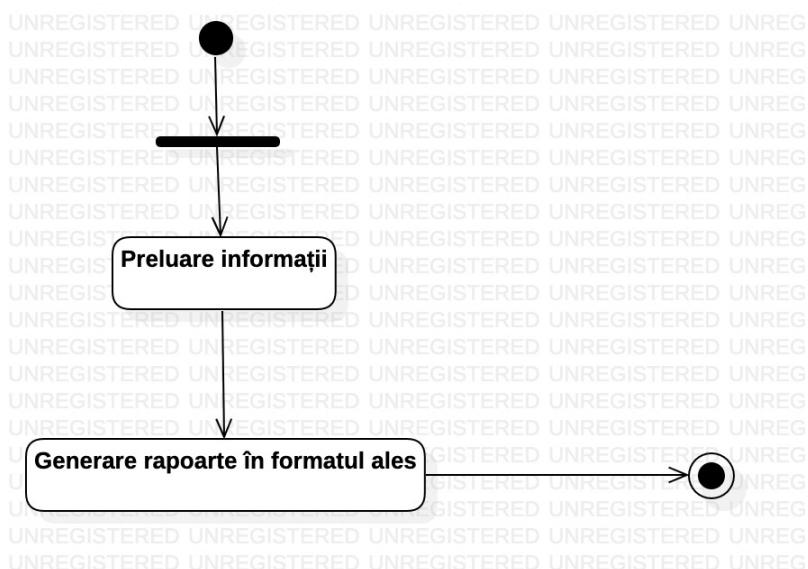


Fig.4.1.12 ActivityDiagram-Save Reports

- **Filter Movies:** se preiau criteriile selectate după care se crează o listă cu filmele care indeplinesc criteriile selectate

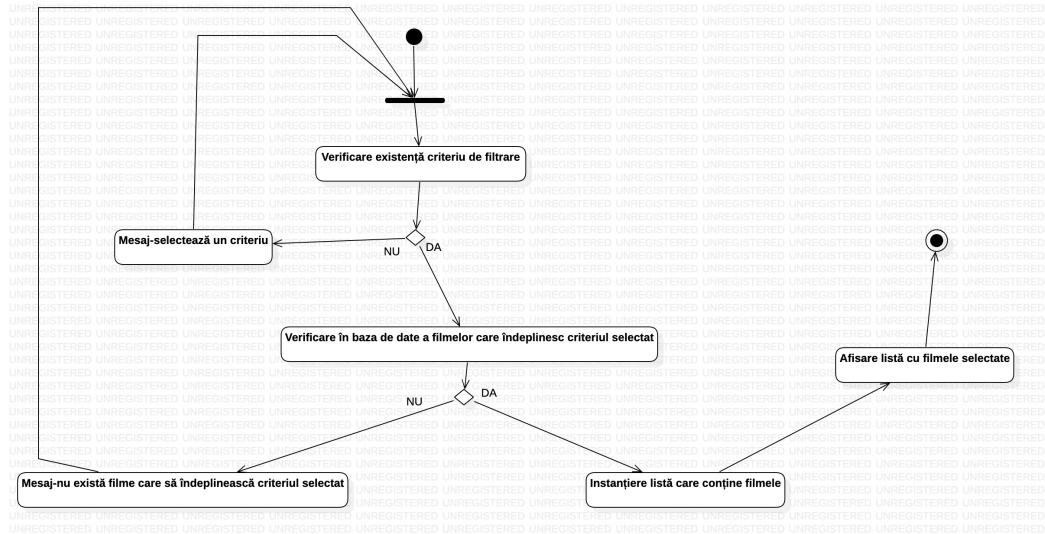


Fig.4.1.13 ActivityDiagram-Filter Movies

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

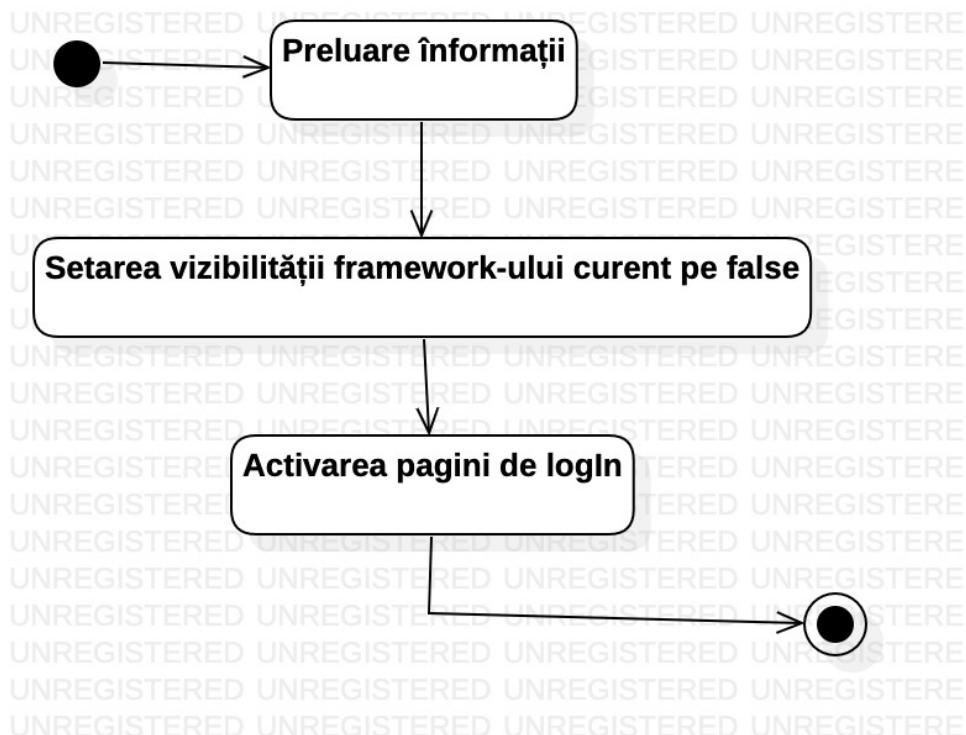


Fig.4.1.14 ActivityDiagram-LogOut

- **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

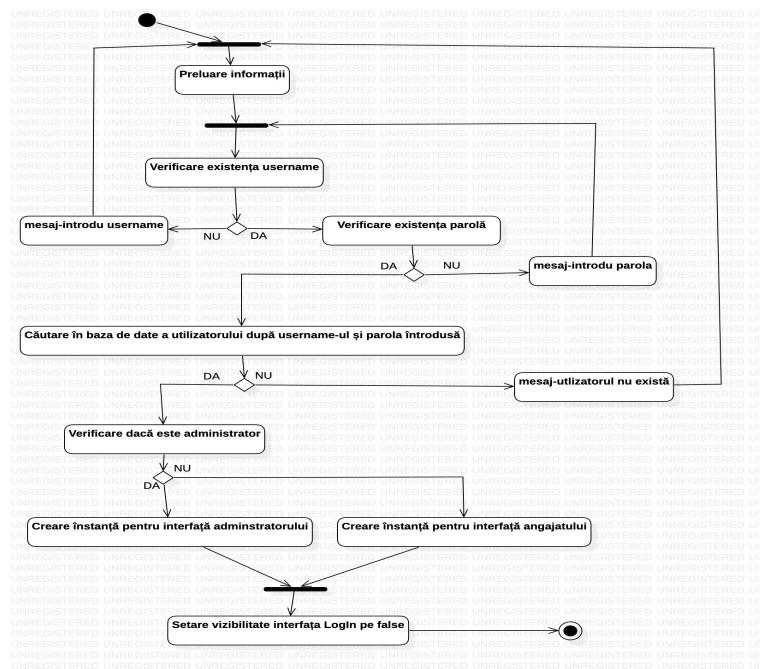


Fig.4.1.15 ActivityDiagram-LogIn

- **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

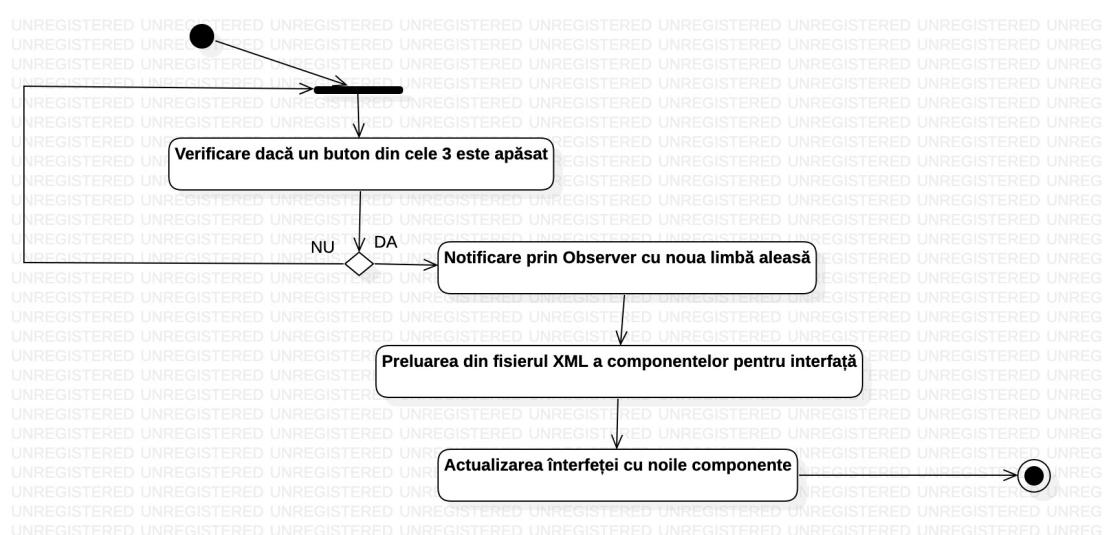


Fig.4.1.16 ActivityDiagram-Change language