Programare orientată pe obiecte

Tema - IMDB

Data postării: 06.11.2023

Deadline: 08.01.2024 ora 23:55

Ultima modificare: 07.11.2023 ora 11:00

Echipă temă: Carmen Odubășteanu, Alexandru Tudor, Eduard Radu, Claudiu Mogodeanu, Bogdan Sprîncenatu, Andrei Petrea

Colaboratori: Diana Nucuță

Facultatea de Automatică şi Calculatoare

Universitatea Națională de Știință și Tehnologie Politehnica București

Anul universitar 2023 - 2024

Seria CC

1 Obiective

În urma realizării acestei teme, studentul va fi capabil:

- să aplice corect principiile programării orientate pe obiecte studiate în cadrul cursului;
- să construiască o ierarhie de clase, pornind de la un scenariu propus;
- să utilizeze un design orientat-obiect;
- să trateze excepțiile ce pot interveni în timpul rulării unei aplicații;
- să transpună o problemă din viața reală într-o aplicație.

Internet Movie Database (IMDB) este o bază de date online care conține informații extinse despre filme, emisiuni TV, actori, regizori, producători și alte elemente legate de industria cinematografică și de televiziune. Este una dintre cele mai mari și mai populare resurse online pentru informații despre filme și seriale TV, oferind recenzii, ratinguri, date de lansare și multe altele.

Proiectul propune crearea unei astfel de baze de date folosind principiile programării orientate pe obiect și funcționalitățile puse la dispoziție de limbajul de programare JAVA, aplicând noțiunile studiate în cadrul acestui curs.

2 Arhitectura aplicației

2.1 CLASE

2.1.1 IMDB

Clasa care va reprezenta aplicația noastră. Aceasta va conține toate detaliile din sistem, extrase din fișierele json:

- Listă cu elemente de tip **Regular**, **Contributor** și **Admin**;
- Listă cu elemente de tip **Actor**;
- Listă cu elemente de tip **Request**;
- Listă cu elemente de tip **Movies** și **Series**;
- ```
  public void run();
  ```
  Metodă prin care
  - se vor încărca toate datele parsate din fișierele JSON oferite;
  - se va face autentificarea utilizatorului;
  - se pornește flow-ul aplicației în funcție de rolul utilizatorului.
- Alte metode pentru a gestiona alegerile utilizatorului.

2.1.2 Production

Clasă **abstractă** care implementează interfața `Comparable` și reprezintă producțiile cinematografice aflate în sistem. Aceasta va conține

- Titlul producției - **String**;
- Listă cu numele regizorilor - **List<String>**;
- Listă cu numele actorilor - **List<String>**;
- Listă cu genurile în care se încadrează - **List<Genre>**;
  ```

- Listă cu evaluările primite de la utilizatori - **List<Rating>**;
- Descrierea subiectului filmului - **String**;
- Nota filmului, egală media aritmetică a tuturor evaluărilor primite de la utilizatori - **Double**;
- ```
  public abstract void displayInfo();
  ```
  Metodă pentru afișarea informațiilor specifice fiecărei subclase
- ```
  public int compareTo(Object o);
  ```
  Metodă necesară sortării filmelor și serialelor în funcție de titlu

2.1.3 Movie

Clasa extinde `Production` și conține durata filmului, anul lansării și, opțional, alte detalii pe care le considerați importante.

2.1.4 Episode

Clasă ce conține numele episodului, durata acestuia și,opțional, alte detalii pe care le considerați relevante.

2.1.5 Series

Clasa extinde `Production` și conține anul lansării, numărul de sezoane și un dicționar ce are drept cheie numele sezonului și drept valoare o listă ce conține obiecte de tip Episode - **private Map<String, List<Episode> >**.

Observații

- Dacă nu exista suficiente detalii pentru completarea tuturor câmpurilor (din Series și Movie), acestea vor primi valoarea null și nu vor fi afișate la cerere.

2.1.6 RequestsHolder

Clasă **statică** ce conține o listă cu toate cererile pe care **toată echipa de admini** trebuie să le rezolve, precum și metodele necesare pentru gestionarea acestei liste (adăugarea/ștergerea unei cereri). Fiind o clasă statică, membrii săi sunt accesibili direct prin numele clasei, fără a fi necesar să se creeze o intanță a clasei, iar modificările aduse asupra sa sunt vizibile în întreaga aplicație.

2.1.7 Request

Această clasă reprezintă o cerere care poate fi făcută de un utilizator. Clasa conține:

- Enum ce definește tipul de cerere - **private RequestType**;
- Data creării cererii - **private LocalDateTime**;
- Titlul producției sau numele actorului, în funcție de caz;
- Descrierea problemei;
- Username-ul utilizatorului care creează cererea;
- Username-ul utilizatorului care trebuie să rezolve cererea.

Observații

- Data va fi formatată folosind `DateTimeFormatter`.
- Username-ul utilizatorului care trebuie să rezolve cererea va fi completat automat cu username-ul celui care are dreptul de a face modificări asupra subiectului aflat în cerere. Astfel, dacă cererea este de tip DELETE_ACCOUNT sau OTHERS, aceasta va fi pusă în lista cererilor destinate **tuturor adminilor** (la username se va trece "ADMIN"). Dacă cererea este de tip ACTOR_ISSUE sau MOVIE_ISSUE, aceasta va fi pusă în lista de cereri a utilizatorului care a introdus producția/actorul respectiv în sistem.
- Un admin/contributor poate rezolva sau respinge o cerere. Dacă alege să rezolve cererea, utilizatorul care a creat-o va primi experiență, așa cum a fost descris mai sus. În ambele situații, creatorul cererii va fi notificat.
- Utilizatorul va trebui să aleagă tipul de cerere dintr-o listă (cu tipurile de cereri) pusă la dispoziție și să scrie motivul/descrierea acelei cereri. Celelalte date (numele producției/actorului, username-ul utilizatorului care a adăugat producția/actorul în sistem, etc.) vor fi preluate automat și nu introduse manual de către utilizator.

2.1.8 Rating

Clasa reprezintă evaluările pe care un utilizator le poate acorda producțiilor. Conține:

- Username-ul utilizatorului care a oferit rating;
- Nota oferită, număr întreg din intervalul [1, 10];
- Comentariile utilizatorului.

Observații

- Doar utilizatorii **Regular** pot evalua filmele și serialele.
- Un utilizator poate evalua o singură dată o producție. Totuși, poate șterge evaluarea anterioară și poate adăuga una nouă pentru aceeași producție, însă nu va primi puncte de experiență.

2.1.9 User

Clasă **abstractă** care poate lucra cu tipuri de date care sunt comparabile. Aceasta reprezintă utilizatorii din sistem și conține:

- Informații despre utilizator - **Information**;
- Tipul utilizatorului - **AccountType**;
- Username-ul contului - **String**;
- Experiența utilizatorului - **int**;
- Listă cu toate notificările utilizatorului - **List<String>**;
- O colecție sortată alfabetic, care reține producțiile și actorii preferați ai utilizatorului (obiecte de tip Actor, Movie, Series) - **SortedSet<T>**;
- Una sau mai multe metode pentru adăugarea elementelor de tip Movie, Series, Actor în lista de favorite;
- Una sau mai multe metode pentru ștergerea elementelor de tip Movie, Series, Actor din lista de favorite;
- O metodă pentru actualizarea experienței utilizatorului;
- O metodă pentru delogarea utilizatorului din sistem și revenire la pagina de logare.

Observații

- La crearea unui utilizator nou, adminul va introduce numele persoanei care deține contul și veți avea grijă ca username-ul să fie unic și să fie un șir de caractere care să conțină și numele utilizatorului (de exemplu, pentru utilizatorul Emily Wilson, a fost generat username-ul unic emily_wilson_789). De asemenea, se va genera automat și o parolă puternică pentru noul utilizator.
- Experiența unui utilizator se actualizează atunci când utilizatorul adaugă o recenzie unei producții, creează o cerere de tipul ACTOR_ISSUE sau MOVIE_ISSUE și aceasta primește o rezolvare (nu este respinsă), adaugă o producție în sistem sau adaugă un actor în sistem. Modul în care crește experiența este explicată mai jos (șablonul Strategy).
- Considerăm că utilizatorii de tip admin au experiență infinită. Prin urmare, experiența lor nu va fi modificată pe parcursul folosirii aplicației, indiferent de acțiunile pe care le face.
- Experiența contribuie la ratingurile oferite producțiilor. Vor fi afișate mai întâi ratingurile oferite de utilizatorii cu experiența mai mare.

2.1.10 Regular

Clasa reprezintă tipul de utilizator regular. Aceasta moștenește clasa `User`, implementează interfața `RequestsManager` și pe lângă funcționalitățile descrise în superclasă, oferă:

- Crearea/Ștergerea unei cereri;
- Adăugarea unei recenzii (element de tip `Rating`) pentru o producție (va nota cu puncte de la 1 la 10 și va adăuga un comentariu).

2.1.11 Staff

Clasă **abstractă** care extinde clasa `User` și implementează interfața `StaffInterface`. Aceasta reprezintă utilizatorii din sistem cu anumite privilegii și conține:

- O listă cu cererile pe care **doar** acest utilizator trebuie să le rezolve - List<Request>;
- O colectie sortată alfabetic, care reține producțiile și actorii pe care acest utilizator i-a adăugat în sistem - SortedSet<T>;
- Implementarea metodelor din interfață.

Clasa oferă, pe lângă funcționalitățile din superclasă, următoarele funcționalități:

- Adăugarea unei producții, unui actor în sistem;
- Ștergerea unei producții, unui actor din sistem;
- Actualizarea informațiilor despre producțiile/actorii adăugați de el în sistem;
- Rezolvarea cererilor primite de la utilizatori.

2.1.12 Contributor

Clasa reprezintă tipul de utilizator contributor. Aceasta extinde clasa `Staff`, implementează interfața `RequestsManager` și oferă, pe lângă funcționalitățile descrise în superclasă, posibilitatea de a crea/șterge o cerere.

2.1.13 Admin

Clasa reprezintă tipul de utilizator administrator. Aceasta extinde clasa `Staff` și oferă, pe lângă funcționalitățile descrise în superclasă, posibilitatea de a adăuga/șterge un utilizator din sistem (se vor șterge, de asemenea, toate detaliile care includ utilizatorul respectiv - recenziile oferite și cererile create).

În această clasă se vor **supradefini** metodele care implică modificarea elementelor care sunt în grija **întregii echipe de amini**.

Observații

- Pentru a crea instanțe ale Claselor Regular, Contributor și Admin, se va utiliza **șablonul** `Factory`.
- Trebuie să vă asigurați că aceste clase pot lucra cu tipuri de date care pot fi comparabile. **HINT: <T extends Comparable<T> >**.
- Un utilizator poate șterge doar producțiile/actorii adăugați de ei în sistem. În plus, adminii pot face acest lucru și pentru producțiile/actorii aflați sunt responsabilitatea întregii echipe de admini.
- Un utilizator Contributor nu are voie să creeze o cerere pentru o producție/actor adăugat de el însuși în sistem.
- La ștergerea unui utilizator Contributor, toate producțiile și actorii adăugați de el vor fi notate ca adăugate de echipa de admini (rămâne la decizia voastră cum faceți acest lucru). Astfel, când se va crea o cerere pentru o producție adăugată de admini, acea cerere va fi pusă în lista de cereri pentru toți adminii. De exemplu, admin1 a adăugat film1 în sistem, iar admin2 nu a adăugat niciun film. Film2 a fost adăugat de un contributor al cărui cont a fost șters. Film2 va fi pus într-o **listă comună** la care are acces atât admin1 cât și admin2. Deci, admin1 are în atribuții film1 și film2, iar admin2 are în atribuție doar film2.

2.1.14 Information

Clasă **internă** clasei `User`. Conține

- Credențialele utilizatorului - **private Credentials**;
- Informații personale - nume, țară, vârstă, gen (unul dintre caracterele - F, M, N).
- Data nașterii - **private LocalDateTime**;

2.1.15 Credentials

Clasa conține datele de autentificare ale utilizatorului: e-mail și parolă.

Observații

- Pentru a instanția un obiect de tip `Information`, se va folosi șablonul de proiectare `Builder`.
- Data va fi formatată folosind `DateTimeFormatter`.
- Clasa `Credentials`, va fi implementată folosind **principiul încapsulării**.

2.1.16 Actor

Clasa reprezintă actorii din sistem și conține numele actorului, o listă cu perechi de tipul Name:Type ce include numele și tipul filmului (Movie, Series) în care actorul respectiv a interpretat un rol și o biografie personală.

2.2 INTERFEȚE

2.2.1 RequestsManager

Conține metode pentru:

- crearea unei cereri

```
public void createRequest(Request r);
```

- ștergerea unei cereri

```
public void removeRequest(Request r);
```

2.2.2 StaffInterface

Conține metode pentru:

- adăugarea unei producții în sistem

```
public void addProductionSystem(Production p);
```

- adăugarea unui actor în sistem

```
public void addActorSystem(Actor a);
```

- ștergerea unei producții **care a fost adăugată de el** din sistem

```
public void removeProductionSystem(String name);
```

- ștergerea unui actor **care a fost adăugat de el** din sistem

```
public void removeActorSystem(String name);
```

- actualizarea detaliilor despre o producție **care a fost adăugată de el in sistem**

```
public void updateProduction(Production p);
```

- actualizarea detaliilor despre un actor **care a fost adăugat de el in sistem**

```
public void updateActor(Actor a);
```

- rezolvarea cererilor primite de utilizatori

2.3 ENUMERĂRI

2.3.1 AccountType

Cuprinde toate tipurile de utilizatori: **REGULAR**, **CONTRIBUTOR**, **ADMIN**.

2.3.2 Genre

Cuprinde toate genurile de filme: Action, Adventure, Comedy, Drama, Horror, SF, Fantasy, Romance, Mystery, Thriller, Crime, Biography, , War, etc. (puteți completa cu oricâte alte genuri).

2.3.3 RequestTypes

Cuprinde toate tipurile de cereri pe care utilizatorii le pot face:

- **DELETE_ACCOUNT** - cerere pentru ștergerea contului (făcută de regular/contributor pentru admini);
- **ACTOR_ISSUE** - cerere în legătură cu datele privind un actor (făcută de regular/contributor pentru contributorul/adminul care a adăugat actorul respectiv în sistem);
- **MOVIE_ISSUE** - cerere în legătură cu datele privind o producție (făcută de regular/contributor pentru contributorul/adminul care a adăugat producția respectivă în sistem);
- **OTHERS** - cereri care nu se încadrează în niciuna din cererile de mai sus (exemplu: data nașterii contului personal a fost completată greșit și necesită actualizare), destinate doar adminilor.

3 Excepții

Pentru ca aplicația să funcționeze corect în toate cazurile, va trebui să tratați anumite excepții care pot apărea în folosirea aplicației. Astfel, se vor arunca următoarele excepții, cu mesaje sugestive:

- `InvalidCommandException` - dacă utilizatorul introduce o comandă invalidă;
- `InformationIncompleteException` - dacă se încearcă realizarea unui obiect de tip Information fără credențiale sau fără nume;

Atenție!

Veți pierde puncte dacă în timpul utilizării aplicației apar excepții pe care nu le tratați. Sunteți liberi să adăugați orice alte tipuri de excepții.
În practică, un utilizator poate folosi greșit aplicația și se poate ajunge la un comportament neprevizibil sau erori ale aplicației. De exemplu, utilizatorul poate introduce un șir de caractere acolo unde se așteaptă un număr întreg. Mai departe, în aplicație, acel șir de caractere este folosit într-o ecuație matematică, ceea ce ar produce o eroare. Trebuie să vă asigurați că nu apar astfel de cazuri la folosirea aplicației **(programare defensivă)**.

4 Șabloane de proiectare

4.1 SINGLETON PATTERN

Singleton Pattern este unul dintre cele mai familiare modele de proiectare utilizate. Acesta se încadrează în categoria modelelor de proiectare creaționale și garantează că o clasă are doar o singură instanță și oferă un punct global de acces la acea instanță.

Va trebui să folosiți acest șablon pentru a restricționa numărul de instanțe ale clasei IMDB.

4.2 BUILDER PATTERN

Scopul acestui design pattern este de a facilita crearea obiectelor complexe, pas cu pas, în special atunci când obiectul final are o mulțime de atribute opționale sau configurabile. Face parte din categoria șabloanelor de proiectare creaționale și se concentrează pe modul în care obiectele sunt create, asigurându-se că procesul de instanțiere este flexibil, eficient și controlat.

Acest șablon realizează separarea construcției de obiecte complexe de reprezentarea lor astfel încât același proces să poată crea diferite reprezentări. Builder-ul creează părți ale obiectului complex de fiecare dată când este apelat și reține toate stările intermediare.

În aplicația voastră, va trebui să utilizați acest șablon pentru a instanția un obiect de tip Information.

4.3 FACTORY PATTERN

Factory Pattern face parte din categoria șabloanelor de proiectare creaționale și oferă o metodă unificată pentru crearea unor obiecte dintr-o familie de clase conexe, lăsând subclasele să decidă care clasă concretă să instanțieze. Veți folosi acest șablon pentru a instanția utilizatorii din sistem - **regular**, **contributor**, **admin**.

Atenție!

Fabrica se va numi **obligatoriu** `UserFactory`.
Funcția de creare va stabili tipul de utilizator folosind enumul `AccountType`.

4.4 OBSERVER PATTERN

Șablonul de proiectare Observer face parte din categoria celor comportamentale și se concentrează pe gestionarea notificărilor și actualizărilor între diferite părți ale unei aplicații.

În această aplicație, va trebui să folosiți acest design pattern pentru a implementa funcționalitatea de notificări, așa cum este descris mai jos. Notificările vor fi șiruri de caractere care includ detalii sugestive despre subiectul notificării.

Un utilizator **regular** trebuie să primească notificări atunci când:

- O cerere pe care a creat-o a fost rezolvată sau respinsă (valabil și pentru un utilizator **contributor**);
- O producție pe care a evaluat-o primește o altă recenzie.

Un utilizator **admin** sau **contributor** trebuie să primească notificări atunci când:

- A primit o cerere destinată lui, pe care trebuie să o rezolve;
- O producție pe care a adăugat-o în sistem primește o recenzie.

Observații

- Asigurați-vă că notificările sunt generate corespunzător și că utilizatorii primesc informații actualizate despre evenimentele care îi privesc.
- Sistemul de notificări ar trebui să fie flexibil și să permită adăugarea ulterioară a altor tipuri de notificări sau evenimente care necesită notificări.
- **HINT:** observatorii vor fi utilizatorii, iar subiectele vor fi cererile și ratingurile.

4.5 STRATEGY PATTERN

Strategy pattern face parte din categoria șabloanelor comportamentale și este folosit pentru a defini o familie de algoritmi, încapsulându-i și făcându-i interschimbabili între ei. Acest design pattern este recomandat dacă este nevoie de un tip de strategie/algoritm cu mai multe implementări posibile și se dorește să se aleagă în mod dinamic implementarea potrivită.

Trebuie să utilizați acest șablon de proiectare pentru a implementa modalități diferite prin care utilizatorii pot crește experiența, în funcție de acțiunile pe care le desfășoară. Asigurați-vă că atunci când un utilizator efectuează una dintre acțiunile care contribuie la creșterea experienței, se utilizează strategia corespunzătoare pentru a actualiza experiența utilizatorului. Experiența utilizatorului evoluează în urma unor acțiuni specifice:

- adăugarea unei recenzii pentru o producție;
- crearea unei cereri de tipul ACTOR_ISSUE, MOVIE_ISSUE care primește o rezolvare (nu este respinsă);
- adăugarea unei noi producții sau a unui actor în sistem.

Veți defini interfața `ExperienceStrategy` care să conțină metoda ce va returna experiența câștigată în urma unei acțiuni.

```
public int calculateExperience();
```

Observații

- Pe lângă cele specificate în această secțiune, mai puteți să alegeți și alte șabloane de proiectare pe care să le integrați în dezvoltarea acestei aplicații. În această situație, veți detalia în **README** motivul pentru care ați ales aceste tipuri și cum le-ați folosit.
- Punctajul o să fie acordat în functție de dificultatea integrării/implementării șabloanelor propuse!

5 Flow-ul aplicației

Aplicația trebuie să urmărească pașii de mai jos:

1. Utilizatorul alege în ce mod va folosi aplicația - terminal sau interfață grafică.
2. Utilizatorul trebuie să completeze credențialele pentru a se autentifica în cont. Dacă autentificarea eșuează (credențialele nu corespund), va fi pus să introducă alte credențiale până când datele sunt introduse corect.
3. După autentificare se vor afișa opțiunile utilizatorului în funcție de tipul acestuia - regular, contributor, admin. În tabelul de mai jos, simbolul **X** reprezintă faptul că utilizatorul de pe coloana corespunzătoare **poate** face acțiunea de la linia corespunzătoare.
4. Aplicația **trebuie să execute** opțiunea aleasă de utilizator, așa cum a fost descris anterior în cerință.
5. După îndeplinirea unei opțiuni, se revine la pasul 3 (credențialele sunt deja completate, se afișează din nou opțiunile).
6. Dacă utilizatorul alege opțiunea de **Delogare**, aplicația îi oferă posibilitatea fie să se autentifice din nou **(se trece la pasul 2)**, fie să închidă aplicația.

Opțiune	Regular	Contributor	Admin
Vizualizarea detaliilor tuturor producțiilor din sistem	X	X	X
Vizualizarea detaliilor tuturor actorilor din sistem	X	X	X
Vizualizarea notificărilor primite	X	X	X
Căutarea unui anumit film/serial/actor	X	X	X
Adăugarea/Ștergerea unei producții/actor din lista de favorite	X	X	X
Crearea/Retragerea unei cereri	X	X	
Adăugarea/Ștergerea unei producții/actor din sistem		X	X
Vizualizarea și rezolvarea cererilor primite		X	X
Actualizarea informațiilor despre producții/actori		X	X
Adăugarea/Ștergerea unei recenzii pentru o producție	X		
Adăugarea/Ștergerea unui utilizator din sistem			X
Delogare	X	X	X

Observații

- La vizualizarea detaliilor tuturor producțiilor din sistem, utilizatorul poate filtra rezultatele în funcție de gen sau numărul de evaluări primite).
- Pentru vizualizarea detaliilor pentru toți actorii din sistem, utilizatorul poate sorta rezultatele în funcție de nume.
- Rezolvarea unei cereri se va face manual de către utilizator. După ce acesta primește și analizează cererea, merge în sistem acolo unde trebuie adusă modificarea (de exemplu, la un anumit film pentru a îi schimba anul lansării). Apoi, va merge din nou la cereri și va marca cererea respectivă ca fiind rezolvată.
- Pentru ștergerea unui utilizator/film/actor din sistem, utilizatorul va primi o listă cu producțiile/actorii/utilizatorii pe care acesta îi poate șterge din sistem. Din această listă alege un element pe care dorește să îl șteargă, iar acțiunea se va face automat (utilizatorul doar trebuie să aleagă ce dorește să șteargă).
- Pentru crearea unei cereri, utilizatorului va putea să aleagă din mai multe liste tipul de cerere și dacă este cazul producția/actorul (listele vor conține opțiunile din cerință, astfel se asigură faptul că cererile sunt conforme).

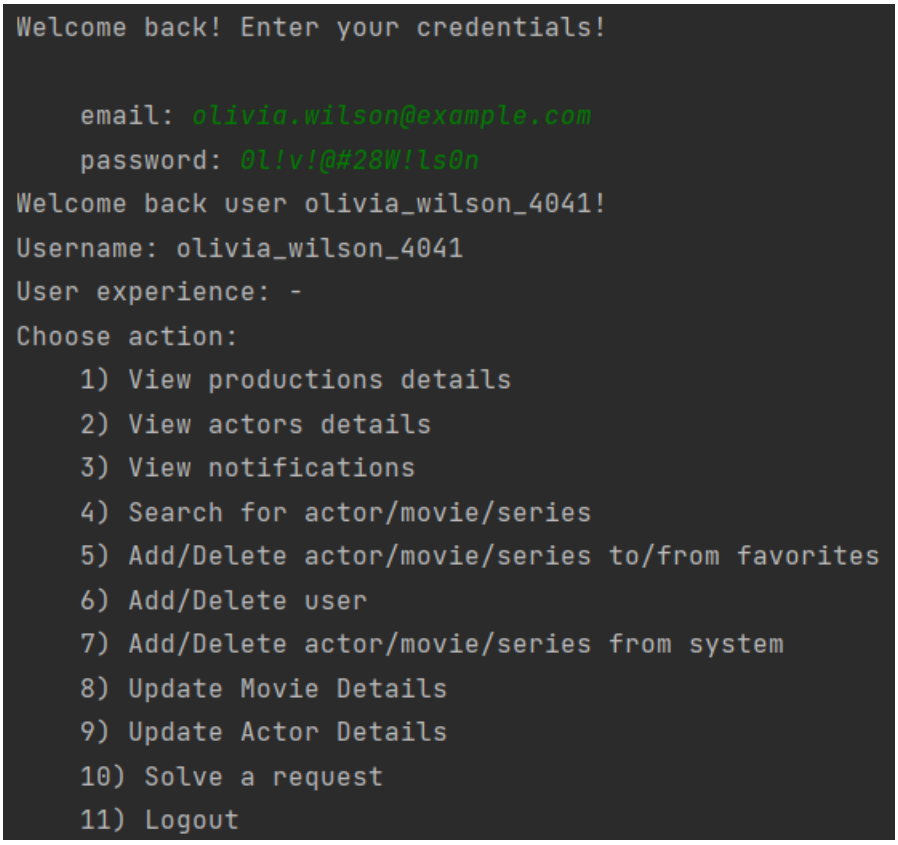

Figure 1: Exemplu de afișare în terminal a opțiunilor pentru utilizatorul de tip admin.

6 Interfața grafică

Va trebui să realizați o interfață grafică folosind pachetul `SWING`. Această interfața trebuie să cuprindă **obligatoriu** următoarele pagini:

- Autentificare
  - Utilizatorul introduce credențialele și intră în pagina principală.
- Pagina principală
  - Conține recomandări de filme și seriale. Modul în care apar aceste recomandări vă aparține (trebuie oferită posibilitatea ca utilizatorul să filtreze filmele în funcție de gen, numărul de evaluări, etc.).
  - Permite căutarea unui anumit film/serial/actor.
  - Permite navigarea la pagina de actori (unde se vor afișa actorii în ordine alfabetică).
  - La selectarea unui film/serial/actor, se vor afișa toate detaliile specifice. Odată cu afișarea acestor detalii, utilizatorul poate opta pentru a adăuga recenzie pentru un film sau pentru adăuga un film/actor în lista de favorite.
- Meniu
  - Din pagina principală se poate accesa pagina de meniu.
  - Oferă utilizatorului posibilitatea alegerii uneia dintre acțiunile prezentate în tabel.

Observații

- Se vor puncta creativitatea și design-ul, precum și folosirea de imagini sugestive (de exemplu, pentru actori și filme).
- Sunteți liberi să adăugați orice funcționalități suplimentare pe care le considerați utile din punct de vedere al interfeței grafice.
- Se va acorda bonus pentru o interfață grafică intuitivă și complexă, frumos realizată, care pune la dispoziție toate operațiile implementate de arhitectură.
- Puteți utiliza biblioteci suplimentare pentru a crea un aspect mai estetic, de exemplu pentru manipularea imaginilor. Orice lucru nou pe care îl utilizați trebuie menționat în README (ce ați folosit și în ce mod).

7 Bonus

Puteți obține bonus pentru orice funcționalitate adusă în plus aplicației. Mai jos aveți câteva propuneri pentru a obține un bonus:

- Salvarea modificărilor, efectuate asupra sistemului, în fișiere JSON, la închiderea aplicației.
- Posibilitatea ca utilizatorii să adauge recenzii și pentru actori.
- Mai multe criterii de a filtra producțiile/actorii (minim 5 diferite față de cele din enunț).
- Posibilitatea ca utilizatorii să vizioneze un trailer pentru fiecare producție.

8 Fișiere de intrare

Pentru testarea aplicației voastre, veți folosi fișierele JSON puse la dispoziție.

- **accounts.json** - detaliile despre utilizatorii din sistem
- **actors.json** - detaliile despre actorii din sistem
- **production.json** - detaliile despre producțiile existente în sistem
- **requests.json** - detaliile despre cererile existente în sistem

Fiecare actor din actors.json apare în exact o listă „actorsContribution" din accounts.json. Analog pentru fiecare film din production.json. Actorii din listele „actors" (din production.json) care nu apar în actors.json vor fi de asemenea instanțiați ca obiecte de tip Actor și vor fi trecuți sub responsabilitatea echipei de admini. Datele lor vor conține numele, în lista de producții va fi adăugat filmul în lista căruia acest actor a fost găsit, iar biografia va fi goală.

Pentru manipularea datelor în format JSON, recomandăm folosirea bibliotecii json-simple, care poate fi descărcată accesând acest **link**.

9 Observații

Atenție!

- **Tipizați** orice colecție folosită în cadrul implementării.
- Respectați specificațiile detaliate în enunțul temei și folosiți indicațiile menționate.
- Pe lângă clasele, atributele și metodele specificate în enunț, puteți adăuga altele dacă acest lucru îl considerați util și potrivit, în raport cu principiile programării orientate pe obiecte.
- Puteți adăuga orice alte detalii și funcționalități pentru aplicație.
- Puteți aduce modificări asupra fișierelor de input (de exemplu, pentru adăugarea imaginilor bonus), dar fără a modifica detaliile deja existente.
- În programarea orientată pe obiecte, **încapsularea** este un principiu fundamental, folosit pentru ascunderea detaliilor și protecția datelor. Deși este considerată o practică profesionistă în dezvoltarea software-ului, nu impunem ca toate clasele arhitecturii să fie implementate folosind acest principiu.

10 Testare

Vă punem la dispoziție un **checker** care va fi folosit pentru a verifica dacă extragerea datelor din fișierele json a fost efectuată corect, precum și dacă progarmul este corect din punct de vedere al implementării design pattern-urilor. Acesta nu va verifica și dacă programul funcționează conform specificațiilor din cerință. Pentru folosirea checker-ului, trebuie să aveți în vedere următoarele observații:

- Rezolvați tema (crearea claselor) în directorul **src/main/java/org.example** din scheletul de checker pus la dispoziție.
- În schelet există un config file - **test/resources/checker_config.json**. Înlocuiți exemplele cu numele complete ale claselor (create de voi) care implementează pattern-urile. Pentru a obține numele complet al unei clase, puteți folosi funcția `NUME_CLASA.class.getCanonicalName();`.
- Pentru rularea checker-ului se va folosi comanda **./gradlew test**. Dacă folosiți IntelliJ, se poate rula direct prin click dreapta pe folderul test și selectarea opțiunii **Run Tests** în TemaPOO2023.

Observații

- **Pentru a folosi checker-ul, trebuie să folosiți Java versiunile 17 sau 19.**
- Checker-ul nu funcționează dacă nu este finalizată întreaga ierarhie de fișiere, împreună cu funcțiile de bază. De asemenea, pentru funcționarea corectă a checker-ului, trebuie să vă asigurați că respectați constrângerile din cerință care țin de numele claselor (spre exemplu, pentru factory pattern numele clasei trebuie să fie UserFactory).
- Un rezultat negativ al checkerului nu înseamnă obligatoriu că pattern-urile nu sunt corecte. În schimb, un rezultat pozitiv garantează că pattern-urile sunt implementate corect (din punct de vedere structural), dar **nu** și că respectă funcționalitatea impusă în cerință.
- Dacă checker-ul nu rulează din cauza unor erori de compilare (**doar** în cazul în care ierarhia este completă și nu aveți erori detectate de IDE), adresați-vă pe forum, în secțiunea checkerului.

11 Punctaj

Cerința	Puncte
Implementarea integrală a arhitecturii și funcționalităților propuse și testarea acestora	120
Design Patterns	50
Interfață grafică	80
Total 1 - 2.5p din nota finală	**250**
Bonus	50
Total 2 - 3p din nota finală	**300**

Atenție!

- Tema va valora **2.5 puncte** din nota finală a disciplinei POO, iar pentru implementarea bonusurilor se poate ajunge la **3 puncte** din nota finală!
- Tema este **individuală**! Toate soluțiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.
- Tema se va prezenta în ultima săptămână din semestru!
- **Se vor puncta DOAR funcționalitățile care pot fi testate.** Acest lucru înseamnă că, în momentul prezentării, **trebuie să oferiți** posibilitatea **testării** tuturor funcționalităților pe care le-ați implementat.
- Tema se va încărca pe site-ul de cursuri până la termenul specificat în pagina de titlu. Se va trimite o arhivă **.zip** ce va avea un nume de forma **grupa_Nume_Prenume.zip** (ex. 326CC_Popescu_Andreea.zip) și care va conține următoarele:
  - un folder **SURSE** ce conține doar sursele Java;
  - un folder **PROIECT** ce conține proiectul în mediul ales de voi (de exemplu NetBeans, Eclipse, Intellij IDEA);
  - un fișier **README.pdf** în care veți specifica numele, grupa, gradul de dificultate al temei, timpul alocat rezolvării și veți detalia modul de implementare, specificând și alte obervații, dacă este cazul.
- Lipsa fișierului README sau nerespectarea formatului impus pentru arhivă duce la o **depunctare** de **10 puncte**.