

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

# Gemixque

sistem de recomandări de jocuri video

propusă de

*Radu Damian*

Sesiunea: *iunie/iulie, 2022*

Coordonator științific

*Lect.dr. Cristian Frăsinaru*

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

# Gemixque

sistem de recomandări de jocuri video

*Radu Damian*

Sesiunea: *iunie/iulie, 2022*

Coordonator științific

*Lect.dr. Cristian Frăsinaru*

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și Prenume \_\_\_\_\_

Data \_\_\_\_\_ Semnătura \_\_\_\_\_

### **DECLARAȚIE privind originalitatea conținutului lucrării de licență**

Subsemnatul(a) \_\_\_\_\_  
domiciliul în \_\_\_\_\_  
născut(ă) la data de \_\_\_\_\_, identificat prin CNP \_\_\_\_\_,  
absolvent(a) al(a) Universității "Alexandru Ioan Cuza" din Iași,  
Facultatea de \_\_\_\_\_ specializarea \_\_\_\_\_,  
promoția \_\_\_\_\_, declar pe propria răspundere, cunoscând consecințele  
falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației  
Naționale nr. 1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul:

\_\_\_\_\_

elaborată sub îndrumarea dl. / d-na \_\_\_\_\_  
pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum  
conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată  
prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la in-  
troducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări  
științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei  
lucrări de licență, de diplomă sau de disertație în acest sens, declar pe proprie răspundere  
că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data azi, \_\_\_\_\_

Semnătură student \_\_\_\_\_

## DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul "*Gemixque*", codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea "Alexandru Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Radu Damian*

---

(semnătura în original)

## Mulțumiri

Sunt profund recunoscător pentru sprijinul necondiționat oferit de părinții și de bunica mea în vederea realizării acestei lucrări de licență.

Îi mulțumesc Monicăi pentru faptul că m-a ajutat să-mi gestionez trăirile și să privesc provocările pe care le-am întâmpinat cu o atitudine sănătoasă.

De asemenea, îi mulțumesc domnului prof. dr Frăsinaru Cristian, îndrumător al lucrării de licență, pentru sfaturile, resursele, și încrederea oferită de-a lungul perioadei în care am lucrat pentru această lucrare.

# Cuprins

1	Descrierea problemei . . . . .	4
2	Tehnologii și resurse folosite . . . . .	4
3	Scopuri și cerințe ale aplicației . . . . .	4
3.1	Scopul documentului . . . . .	4
3.2	Publicul țintă . . . . .	5
3.3	Scopul aplicației . . . . .	5
4	Analiză și proiectare . . . . .	5
4.1	Baza de date . . . . .	5
4.2	De ce NoSQL și nu SQL? . . . . .	6
4.3	Modelarea bazei de date . . . . .	8
4.4	Potențiale probleme . . . . .	11
4.5	Generarea datelor de intrare . . . . .	13
5	Implementare . . . . .	14
6	Manual de utilizare . . . . .	14
7	Concluzii . . . . .	14

## Introducere

Această lucrare de licență propune prezentarea implementării unui sistem de recomandare, sub forma unei aplicații web.

În următoarele capitole vor fi ilustrate următoarele: problema ce trebuie rezolvată, tehnologiile folosite, modelarea bazei de date, modul în care au fost generate datele de intrare ale problemei, server-ul de back-end, algoritmul implementat și server-ul de front-end.

## Motivație

Ideea de bază a lucrării de licență a venit oarecum din întâmplare. Când eram student în semestrul II din anul 2 de facultate, mă uitasem într-o zi obișnuită la acest site: <https://simkl.com>, pe care-l foloseam pentru a nu uita la ce episod am rămas dintr-un anumit serial. Acest site poate oferi și recomandări de filme/seriale în funcție de datele din profilul meu.

Fiind un amator de jocuri video pe PC, mi-a venit ideea de a face un astfel de site, însă care să recomande, bineînțeles, jocuri video.

O observație importantă este faptul că aș fi putut alege orice resursă ce poate fi recomandată (de exemplu: periute de dinți, telefoane, picturi etc.), așadar problema în speță poate fi privită într-un mod mai general. Motivul pentru care am ales jocurile video este dat de pasiunea mea pentru acestea.

De asemenea, eram curios în acea perioadă, printre altele, legat de cum aș putea implementa o soluție pentru acest sistem de recomandare pe care mi l-am propus să-l realizez.

Așadar, am avut inspirație, dar și noroc, deoarece această idee mi-a venit relativ repede, lucru ce a reprezentat un punct de pornire important în realizarea efectivă a acestei lucrări.

# 1 Descrierea problemei

Problema ce trebuie rezolvată se rezumă la recomandarea unor jocuri video (niciunul, unul, sau mai multe) unui utilizator. În ansamblu, datele de intrare ar putea fi reprezentate de o mulțime de jocuri video, o mulțime de utilizatori și o mulțime de recenzii.

Aceste recenzii sunt oferite de utilizator și asociate unui joc. În recenzie, utilizatorul oferă o notă (un scor, pe o scară de la 1 la 10) care indică cât de mult i-a plăcut/displăcut jocul respectiv. Un utilizator poate face maxim o recenzie per joc.

## 2 Tehnologii și resurse folosite

Pentru aplicația web:

1. Baza de date: Neo4j.
2. Server de back-end:
  - Limbaj: Java versiunea 11
  - Framework: Spring Boot
  - REST API
3. Server de front-end:
  - Limbaj: Typescript
  - Framework: Angular

Pentru modulul de generare al datelor de intrare:

1. API extern folosit pentru a procura date despre jocuri video: IGDB API [1]
2. API extern folosit pentru procurarea de recenzii de pe Steam [2]
3. Java Faker - pentru generarea de date aleatoare [3]
4. Apache Commons CSV - pentru manipularea de fișiere CSV [4]

## 3 Scopuri și cerințe ale aplicației

### 3.1 Scopul documentului

Scopul acestui document este de a ilustra modul în care funcționează un sistem de recomandări de jocuri video sub forma unei rețele sociale.



## 3.2 Publicul țintă

Acest document este destinat atât cititorilor avizați(e.g. profesori universitari) pentru a afla de exemplu soluțiile utilizate în cadrul capitolului 4 **Analiză și proiectare**, sau modul de implementare a aplicației în capitolul 5 **Implementare**, cât și cititorilor neavizați(utilizatori obișnuiți), care se pot informa în legătură cu modul în care pot interacționa cu site-ul în capitolul 6 **Manual de utilizare**.

## 3.3 Scopul aplicației

Scopul acestei aplicații este de a oferi o soluție în a contracara cantitatea masivă de informații ce se regăsește pe internet [5] în ceea ce privește multitudinea de jocuri video, prin a dezvolta un sistem de recomandări care să faciliteze decizia unui utilizator legat de ce joc să aleagă.

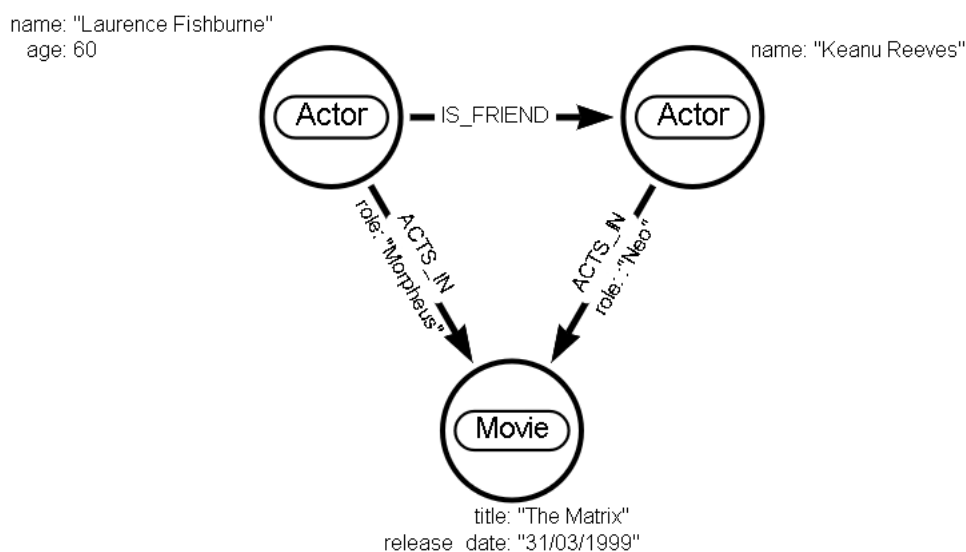
# 4 Analiză și proiectare

## 4.1 Baza de date

Tipul de stocare ales este cel oferit de Neo4j, în care se prezintă o abordare NoSQL de tip graf. Comparativ cu o bază de date relațională, în care datele sunt stocate prin intermediul unor înregistrări(tuple) în tabele, în Neo4j datele sunt stocate prin intermediul nodurilor și muchiilor.

În exemplul următor, vor fi ilustrate caracteristicile nodurilor și muchiilor în stocarea efectivă a datelor:

Figure 1

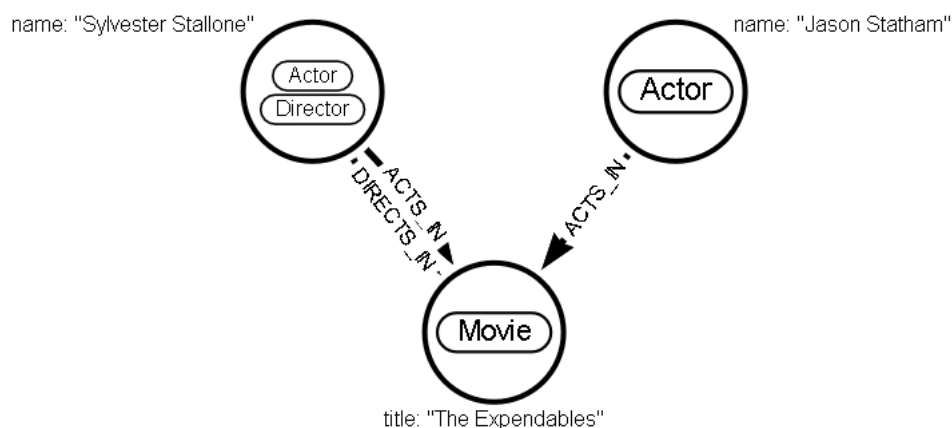


Un nod în neo4j are următoarele caracteristici [6]: reprezintă entități/obiecte, pot fi etichetate și pot avea proprietăți.

Observăm din figură cele trei entități: două noduri etichate cu 'Actor' și un nod etichetat cu 'Movie'. De asemenea, ambele noduri 'Actor' au proprietatea 'name', însă doar unul din ele are și proprietatea 'age'. Așadar, nu trebuie neapărat ca două noduri cu aceeași etichetă să aibă aceleași proprietăți.

Încă un lucru important de menționat este faptul că un nod poate avea mai multe etichete, așa cum se poate vedea din următorul exemplu:

Figure 2



Practic, din această figură se observă faptul că se poate modela cu ușurință situația în care un regizor joacă în propriul său film.

O altă noțiune importantă într-o bază de date de tip graf este cea de relație, care este asociată unei muchii. O relație trebuie să aibă un tip, un sens, și poate avea proprietăți. Din figurile anterioare s-au putut observa relațiile **ACTS\_IN**, **DIRECTS\_IN** sau **IS\_FRIEND**.

Un alt aspect important de precizat este faptul că între două entități pot exista mai multe relații, așa cum s-a putut vedea în figura anterioară. Așadar, se modelează practic un multigraf orientat.

## 4.2 De ce NoSQL și nu SQL?

Având în vedere faptul că în această aplicație este prezentă o rețea socială în care mai mulți utilizatori pot interacționa între ei și pot oferi recenzii jocurilor video, o bază de date de tip graf este o alegere inspirată.

În ceea ce privește limbajul utilizat pentru a efectua interogări pe baza de date, avem în vedere următorul studiu de caz:

Să presupunem că dorim să modelăm ceea ce se întâmplă în cadrul unei facultăți, pe scurt: gestionarea studenților, a notelor pe care le iau aceștia la cursuri, și a profesorilor. Exemplul ce urmează a fi ilustrat se bazează pe schema bazei de date ce a fost utilizată în cadrul materiei Baze de date din anul II semestrul I. [7]

Să propunem că dorim să efectuăm următoarea interogare: pentru un student, să aflăm numele profesorilor la cursurile în care studentul a luat nota 10.

O interogare în limbajul SQL ar putea arăta în felul următor:

```

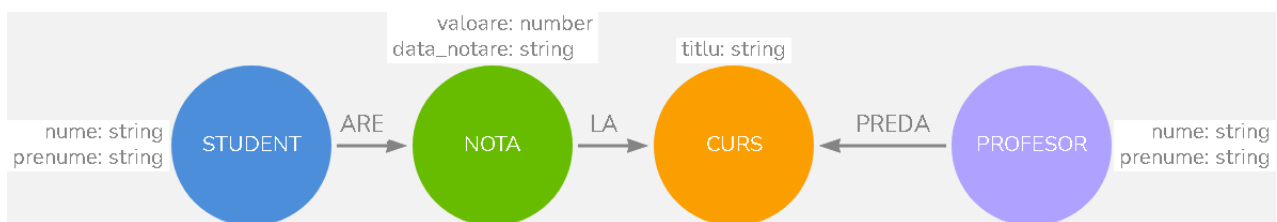
SELECT p.num, p.prenume FROM NOTE n
JOIN CURSURI c ON n.id_curs = c.id
JOIN DIDACTIC d ON d.id_curs = c.id
JOIN PROFESORI p ON p.id = d.id_profesor
WHERE VALOARE = 10 AND ID_STUDENT = 36;

```

Se poate observa așadar faptul că sunt necesare o serie de join-uri pentru a putea obține rezultatul dorit.

Pentru a putea compara această interogare cu cea care s-ar putea face în limbajul Cypher folosit în Neo4j, se va ilustra pe scurt cum s-ar putea modela schema bazei de date anterior menționată în una de tip graf. (nu în totalitate, ci doar de ceea ce avem nevoie pentru a evidenția interogarea)

Figure 3: Miniatură a schemei bazei de date



Așadar, având în vedere acest model, interogarea în Cypher ar putea arăta în felul următor:

```

MATCH (s:STUDENT)-[:ARE]->(n:NOTA {valoare: 10}),
(n)-[:LA]->(c:CURS)<-[:PREDA]-(p:PROFESOR)
WHERE id(s) = 0
RETURN p.num, p.prenume

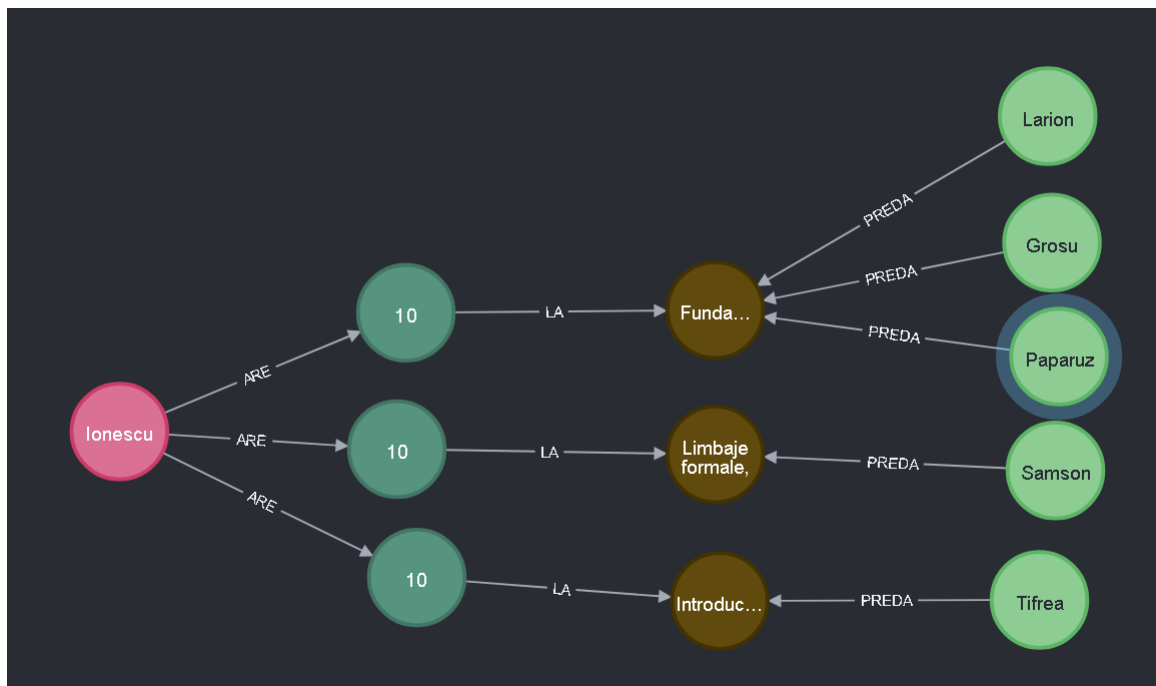
```

Un prim lucru interesant care s-ar putea observa este faptul că Cypher introduce prin sintaxa sa conceptul de ASCII Art [8], [9], care practic face posibilă o nouă interpretare a codului, una vizuală, adică se poate observa relativ ușor cum ceea ce este scris în primele două linii de cod seamănă destul de mult cu modelul din figura anterioară. Cele două linii s-ar fi putut scrie într-o singură linie fără nicio dificultate, însă am optat pentru această variantă pentru o mai bună lizibilitate a codului.

Pe de altă parte, o altă caracteristică importantă a bazelor de date de tip graf este că relațiile între entități au cea mai mare prioritate în modelarea datelor [10], acest lucru reflectându-se în modul cum a fost construită această interogare, în care este mult mai facil să interogăm niște date aflate la capătul unui lanț format din mai multe relații, așa cum se observă și în acest exemplu.

Pentru o mai bună înțelegere a modului în care sunt reprezentate efectiv datele, se poate vedea figura de mai jos:

Figure 4



Datele folosite pentru a exemplifica interogarea în Cypher

### 4.3 Modelarea bazei de date

Având în vedere noțiunile prezentate în capitolele anterioare, în acest capitol va fi descrisă schema bazei de date ce va fi folosită pentru a stoca datele necesare rezolvării problemei de a recomanda jocuri video.

Însă, înainte de a ilustra schema în ansamblu, fiecare entitate în parte va fi detaliată în rândurile următoare.

#### Utilizatorul

Această entitate reprezintă punctul central al aplicației, deoarece utilizatorul este cel care oferă recenzii(note) jocurilor și influențează într-un mod indirect recomandările unui alt utilizator, deci el este cel care, practic, inițiază acțiunea.

Nodul corespunzător acestei entități este cel din **Figura 5**.

Printre proprietățile alese pentru a reprezenta un utilizator, două dintre acestea reflectă activitatea lui în sistemul de recomandare, anume *average\_score* și *nr\_reviews\_made*, care semnifică numărul total de recenzii făcute de utilizator, respectiv scorul mediu al acestuia(media tuturor scorurilor oferite).

#### Recenzia

Recenzia reprezintă practic indicatorul calitativ pe care-l oferă un utilizator asupra unui joc. Așa cum se observă din **Figura 6**, o recenzie poate fi caracterizată prin proprietatea *score*, nota oferită de utilizator jocului, proprietatea *content*, care reprezintă un

text ce poate fi introdus de utilizator pentru a-și exprima, dacă dorește, opinia sa asupra jocului, și proprietatea *time*, adică momentul în care a făcut recenzia.

## Jocul

Jocul indică resursa ce va fi recomandată de către utilizator și este punctul de interes al aplicației. Faptul că această entitate poate avea un număr relativ semnificativ de proprietăți reprezintă o provocare justificată, iar abordarea mea pentru a reprezenta această entitate se poate observa în **Figura 7**.

Pentru a nu suprasatura un nod cu multe proprietăți, am decis să împart acestea în funcție de categoria în care ar putea fi încadrate. Nodul principal **Game** are doar titlul, genurile și anul primei lansări, iar prin relația de tipul **HAS** se evidențiază nodurile care extind această entitate, și anume cel care conține proprietățile care indică elemente vizuale ale jocului (*Visuals*), cel care conține date suplimentare despre joc (*Details*) și cel care ține evidența legat de recenziile aplicate asupra jocului (*Average*), unde proprietatea *value* reprezintă scorul mediu al jocului (cât de bine este notat în medie de către utilizatorii sistemului), iar proprietatea *aggregated\_rating* reprezintă scorul obținut dintr-o sursă externă (e.g. Metacritic)

## Schema bazei de date

În **Figura 8**, având în vedere și aspectele menționate anterior legat de entitățile care formează în ansamblu schema bazei de date, se poate intui fluxul principal al aplicației. Utilizatorul introduce în profilul său ce jocuri video s-a jucat prin intermediul relației **PLAYS**, iar acțiunea prin care acesta efectuează o recenzie asupra jocului este reprezentată prin relația de tip **MAKES**. De asemenea, relația **ON** indică asupra cărei entități etichetată cu *Game* se face recenzia.

Se poate deduce faptul că relațiile între noduri sunt numite astfel încât să reprezinte acțiuni ce pot fi efectuate sau aplicate asupra unor entități, ceea ce poate reprezenta un procedeu de bună practică în modelarea bazei de date în acest context. [11]

Figure 5: Utilizatorul

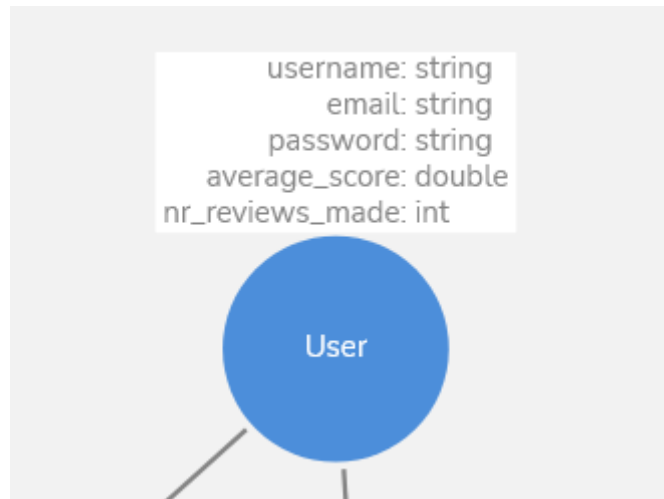


Figure 6: Recenzia

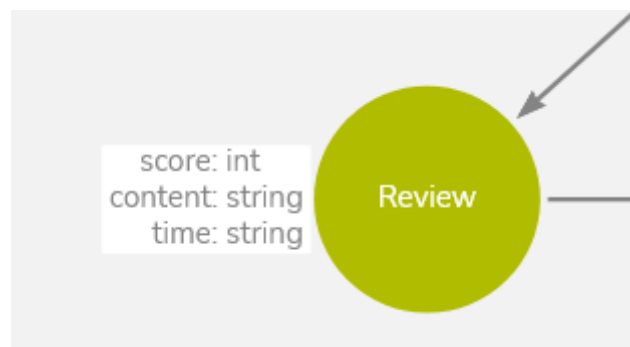


Figure 7: Jocul

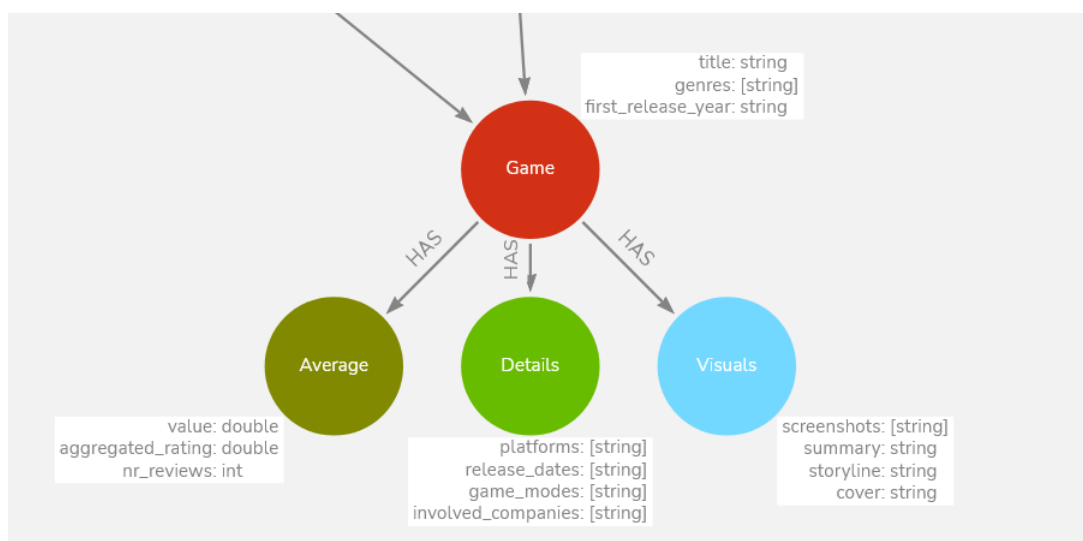
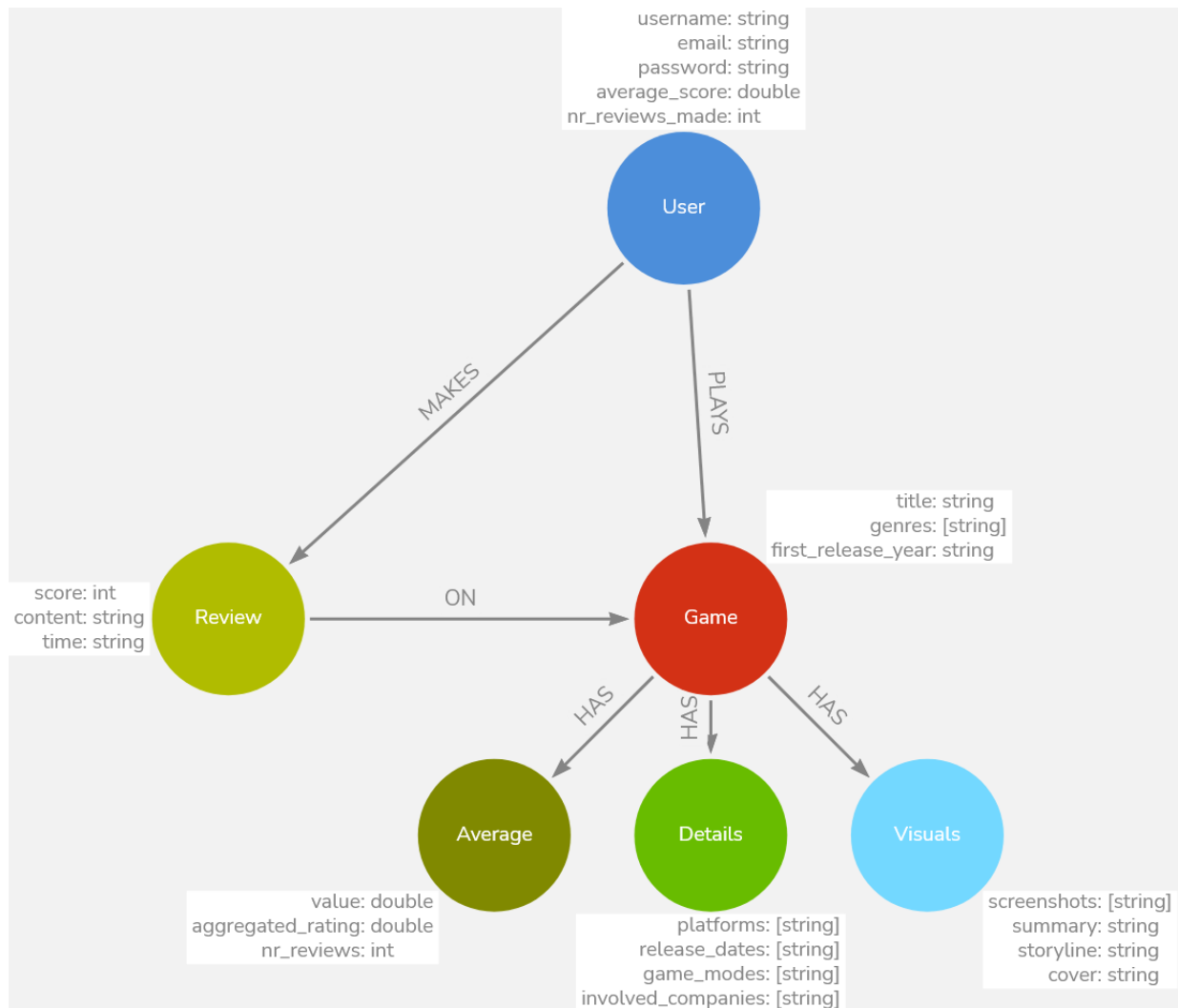


Figure 8: Schema bazei de date



Entitățile etichetate cu User, Game și Review mai au o proprietate în plus, și anume cea de id, care reprezintă un identificator unic pentru entitățile respective(uuid)

## 4.4 Potențiale probleme

### Constrângeri

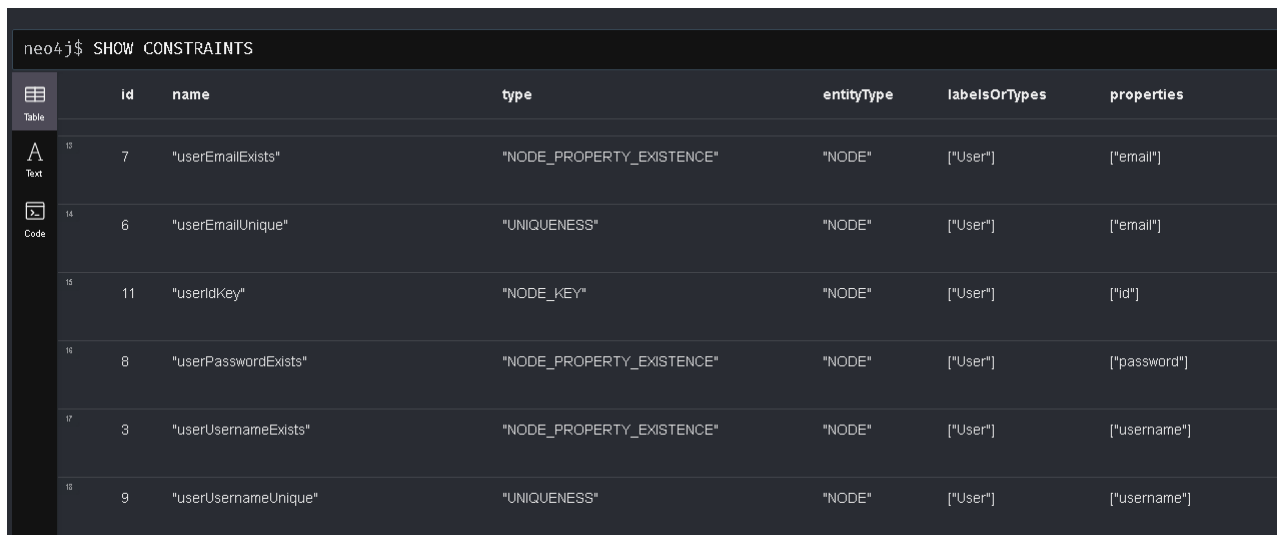
Am menționat atunci când am introdus capitolul de Baze de date faptul că două noduri care au aceeași etichetă nu trebuie să aibă neapărat aceleași proprietăți. Acest aspect face ca, în mod implicit, ca manipularea datelor să fie mult mai tolerabilă(cu mai puține constrângeri) în comparație cu o bază de date de tip SQL, unde acolo de exemplu trebuie să specificăm pentru o tabelă ce coloane să aibă și de ce tipuri să fie acestea.

Totuși, ce ar fi dacă am vrea să stabilim niște constrângeri? Poate am vrea să spunem de exemplu că o entitate etichetată cu User trebuie să aibă neapărat proprietățile *username*, *email* și *password*, sau am vrea să setăm pentru o entitate cheia sa, ca să o putem identifica. Pentru a rezolva acest lucru, Neo4j introduce conceptul de constrângeri (*constraints*) [12]

Pentru a lucra facil cu baza noastră de date, Neo4j oferă o aplicație desktop ce vine cu un panou de comandă care oferă multe utilități. Putem executa de exemplu interogări în Cypher și să vizualizăm în mod interactiv graful obținut, cum se poate observa și în **Figura 4**.

În figura de mai jos, am realizat o captură în acest panou de comandă în care am scris comanda `SHOW CONSTRAINTS` unde pot vedea ce constrângeri sunt setate.

Figure 9



	id	name	type	entityType	labelsOrTypes	properties
12	7	"userEmailExists"	"NODE_PROPERTY_EXISTENCE"	"NODE"	["User"]	["email"]
14	6	"userEmailUnique"	"UNIQUENESS"	"NODE"	["User"]	["email"]
15	11	"userIdKey"	"NODE_KEY"	"NODE"	["User"]	["id"]
16	8	"userPasswordExists"	"NODE_PROPERTY_EXISTENCE"	"NODE"	["User"]	["password"]
17	3	"usernameExists"	"NODE_PROPERTY_EXISTENCE"	"NODE"	["User"]	["username"]
18	9	"usernameUnique"	"UNIQUENESS"	"NODE"	["User"]	["username"]

Se pot observa constrângerile pe care le-am setat pentru entitatea User

### Modul în care se actualizează scorul mediu al utilizatorului și al nodului *Average* asociat nodului *Game*

Am menționat atunci când am prezentat schema bazei de date de faptul că am dori să ținem cont de scorul mediu al utilizatorului și al jocului, și de numărul de recenzii al acestor entități. O variantă naivă pentru a rezolva această problemă este să calculăm valorile acestor proprietăți în serverul de back-end, parcurgând toate recenziile asociate unui utilizator/joc și calculând media aritmetică a scorurilor acestora de fiecare dată când se face o nouă recenzie de exemplu. Dacă avem relativ puține date, nu ar fi o problemă, însă atunci când numărul de recenzii, de jocuri sau de utilizatori crește semnificativ, atunci trebuie abordată o altă soluție pentru a eficientiza acest proces.

Neo4j introduce un alt concept, prin intermediul librăriei APOC [13], cel de declanșatoare (*triggers*). [14] Declanșatoarele sunt folosite pentru a executa cod în Cypher înainte sau după ce anumite date au fost modificate.

Așadar, am putea folosi un declanșator pentru a actualiza numărul de recenzii și scorul mediu pentru entitățile *User* și *Game* după ce o recenzie a fost creată.

Pentru a înțelege codul ce va fi prezentat ulterior, voi introduce un mic artificiu matematic pentru a înțelege cum se actualizează media scorurilor în acest caz.

Să presupunem că după  $k$  recenzii efectuate, scorul mediu este  $a_k$  și cea de a  $k + 1$  recenzie are scorul  $s$ .



Pentru a reconstitui suma anterioară, aceasta reprezintă pur și simplu produsul dintre  $k$  și  $a_k$ . Așadar, putem calcula noua medie, prin formula:

$$a_{k+1} = \frac{k \cdot a_k + s}{k + 1}$$

Prin urmare, fiind introdusă această formulă, se poate prezenta codul ce va fi executat în cadrul declanșatorului, în rândurile următoare:

```
UNWIND \ $createdRelationships AS r
  MATCH (review:Review)-[r:ON]->(game:Game)
  MATCH (game)-[:HAS]->(average:Average)
  MATCH (u:User)-[:MAKES]->(review)-[:ON]->(game)
  SET average.value = toFloat(
    average.nr_reviews * average.value + toInteger(review.score)
  ) / (average.nr_reviews + 1),
    average.nr_reviews = average.nr_reviews + 1,
  u.average_score = toFloat(
    u.nr_reviews_made * u.average_score + toInteger(review.score)
  ) / (u.nr_reviews_made + 1),
  u.nr_reviews_made = u.nr_reviews_made + 1
```

A se ignora simbolul \ înainte de createdRelationships

Așadar, de fiecare dată după ce se creează o relație de tipul **ON** între nodurile *Review* și *Game*, atunci vom interoga nodurile *Average* și *User* pe care dorim să le actualizăm folosind formula anterioară.

## 4.5 Generarea datelor de intrare

Neo4j oferă funcționalitatea de a importa date prin intermediul fișierelor CSV, iar prin intermediul limbajului Cypher, pot fi create noduri și relații cu aceste date pentru a crea schema de baze de date.

Din intenția de a-mi modela baza de date după bunul meu plac, nu am vrut să optez pentru a lua un set de date deja gata făcut, ci am implementat un submodul în limbajul Java(separat de aplicația web propriu-zisă) care extrage date ori din API-uri externe(menționate la capitolul **Tehnologii și resurse folosite**, ori din fișiere salvate local, și generează fișiere CSV pentru a le putea importa apoi în baza de date.

### Generarea utilizatorilor

Pentru a-mi genera utilizatorii fictivi ai aplicației, am apelat la librăria Java Faker. Aceasta m-a ajutat ca să generez nume de utilizator, parole și adrese de e-mail. În ceea ce privește parolele, am generat prin intermediul acestei librării un șir de caractere *random* format din 8 caractere, ce conține cifre și litere. De asemenea, pe această parolă am aplicat o funcție hash, cu scopul de a fi folosită atunci când un utilizator se loghează în aplicație, pentru a avea un strat de securitate în acest caz. Am ales să utilizez algoritmul PBKDF2 în convertirea parolei deoarece este o variantă sigură. [15]

Așadar, cu valorile generate, am creat două fișiere CSV în care rețin 500 de utilizatori(un fișier cu parolele originale, celălalt cu parolele convertite prin funcția hash)

O observație importantă este că atât pentru utilizator, cât și pentru celelalte entități, am generat și un identificator unic, prin intermediul utilitarului UUID oferit de Java.

## **5 Implementare**

## **6 Manual de utilizare**

## **7 Concluzii**

# Bibliografie

- [1] *IGDB API*  
<https://www.igdb.com/api>  
<https://github.com/husnjak/IGDB-API-JVM>
- [2] *Steamworks Documentation - Reviews*  
<https://partner.steamgames.com/doc/store/getreviews>
- [3] *Java Faker*  
<https://github.com/DiUS/java-faker>
- [4] *Apache Commons CSV*  
<https://commons.apache.org/proper/commons-csv/>
- [5] *Recommendation systems: Principles, methods and evaluation*, 2015.  
F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh  
<https://www.sciencedirect.com/science/article/pii/S1110866515000341>
- [6] *What is a graph database?*  
<https://neo4j.com/developer/graph-database/>
- [7] *Baze de date - Laborator 1*  
A se vedea secțiunea 'Schema bazei de date - studiu de caz'  
[https://profs.info.uaic.ro/~bd/wiki/index.php/Laborator\\_1](https://profs.info.uaic.ro/~bd/wiki/index.php/Laborator_1)
- [8] *Neo4j Training - Querying with Cypher In Neo4j 4.x*  
<https://neo4j.com/graphacademy/training-querying-40/01-querying40-introduction-to-cypher/>
- [9] *ASCII art*  
[https://en.wikipedia.org/wiki/ASCII\\_art](https://en.wikipedia.org/wiki/ASCII_art)
- [10] *Neo4j Training - Overview of Neo4j 4.x - Neo4j is a Graph Database*  
<https://neo4j.com/graphacademy/training-overview-40/01-overview40-neo4j-graph-database/>

[11] *Graph Modeling Guidelines*

<https://neo4j.com/developer/guide-data-modeling/>

[12] *Constraints*

<https://neo4j.com/docs/cypher-manual/current/constraints/>

[13] *APOC Documentation / Introduction*

<https://neo4j.com/labs/apoc/4.1/introduction/>

[14] *Triggers*

<https://neo4j.com/labs/apoc/4.1/background-operations/triggers/>

[15] *Hashing a Password in Java*

<https://www.baeldung.com/java-password-hashing>