UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Aplicații ale bazelor de date de tip graf

propusă de

Marius-George Olaru

Sesiunea: *iulie, 2018*

Coordonator științific

Lect. Dr. Cristian Frăsinaru

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Aplicații ale bazelor de date de tip graf

Marius-George Olaru

Sesiunea: *iulie, 2018*

Coordonator științific

Lect. Dr. Cristian Frăsinaru

Declarație privind drepturile de autor

Prin prezență declar că Lucrarea de licenţă cu titlul "Aplicații ale bazelor de date de tip graf" este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau din străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imagini etc. preluate din proiecte open-source sau alte surse sunt utilizate cu respectarea drepturilor de auton și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iaşi, 3 iulie 2018 Absolvent Marius-George Olaru

(semnătură în original)

Declarație de consimțământ

Prin prezență declar că sunt de acord ca Lucrarea de licență cu titlul "Aplicații ale bazelor de date de tip graf", 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 sa 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, 3 iulie 2018

Absolvent Marius-George Olaru

_____

(semnătură în original)

Cuprins

Introducere

Sistemele de baze de date sunt o componentă esenţială a vieţii de zi cu zi în societatea modernă. În cursul unei zile, majoritatea persoanelor desfăşoară activităţi care implică interacţiunea cu o bază de date: depunerea sau extragerea unor sume de bani din bancă, rezervarea biletelor de tren sau avion etc. Bazele de date pot avea dimensiuni (număr de înregistrări) extrem de variate, de la câteva zeci de înregistrări (de exemplu, baza de date pentru o agendă cu numere de telefon) sau pot ajunge la zeci de milioane de înregistrări (de exemplu, baza de date pentru plata taxelor şi a impozitelor). În sensul cel mai larg, o bază de date (database) este o colecţie de date corelate din punct de vedere logic, care reflectă un anumit aspect al lumii reale. Criteriul principal de clasificare a bazelor de date îl reprezintă modelul conceptual utilizat în descrierea structurii datelor. Aparținând criteriului bazelor de date nerelaționale, Neo4J este o bază de date fondată pe teoria grafurilor, fiind o soluție optimizată pentru a modela și interoga volume mari de date strâns relaționate, reprezentabile prin structuri de tip graf.

Am ales să folosesc acest sistem de gestiune a bazei de date în contextul unei aplicații Web , care poate fi descrisă ca o rețea socială în miniatură. Principalul motiv pentru care am luat această decizie este acela că o reprezentare sub formă de graf a datelor se pliază foarte "intuitiv" în acest context, astfel încât entitățile din baza de date sunt reprezentate prin noduri, iar relațiile dintre acestea prin muchiile dintre noduri.

Pentru realizarea aplicației se vor folosi tehnologii actuale, ce se află atât în componenta de back-end a aplicației , cât și pe partea de client.

Scopul lucrării prezente este de a folosi pentru stocarea datelor soluția oferită de Neo4J (fiind cea mai cunoscută implementare de bază de date de tip graf), astfel studiind această abordare de reprezentare a datelor și de a putea compara cu alte abordări (spre exemplu bazele de date relaționale) din următoarele perspective: performanță , scalabilitate, flexibilitate, complexitate.

Deși folosirea bazelor de date relaționale se pretează pe majoritatea situațiilor, în acest caz, folosirea unei baze de date de tip graf, respectiv Neo4J, în contextul în care avem de memorat cantități mari de noduri și relații între acestea, și apoi de parcurs prin interogări, este o soluție extrem de potrivită.

Contribuții

În cadrul acestui proiect am avut contribuții pe următoarele planuri:

- Pe plan teoretic:
  - Definirea unei structuri a bazei de date , prin crearea nodurilor și a muchiilor dintre ele , care să pună cât mai mult în valoare folosirea acestui tip de reprezentare a datelor;
  - Studierea limbajului folosit pentru executarea scripturilor asupra bazei de date Neo4J, Cypher;
  - Compararea performanței realizate de baza de date pentru aducerea datelor în serviciu, apoi urmând să fie afișate clientului.
- Pe plan practic:
  - Identificarea cerințelor pentru a implementa o astfel de aplicație;
  - Analizarea soluțiilor din punct de vedere al implementării modelului propus;
  - Implementarea aplicației pe partea de client, pentru a demonstra contextul folosit, și anume cel al unei rețele sociale în miniatură;
  - Integrarea aplicației cu baza de date pusă la dispoziție de Neo4J, și implementarea scripturilor Cypher necesare funcționării corecte a modelului.

Contextul actual

În prezent, bazele de date tradiţionale sunt puse la încercare din ce în ce mai mult de noile tipuri de aplicaţii care le folosesc. Aceste tipuri de aplicaţii utilizează de regulă o cantitate mare de date complexe. Dacă în trecut pentru o mare perioadă de timp bazele de date relaţionale esenţiale pentru aplicaţii web erau MySQL, astăzi aceste baze de date relaţionale întâmpină multe dificultăţi în lucru cu cantităţi mari de date. Pe piaţa în care activează MySQL au pătruns furnizorii de soluţii de baze de date cloud. Aceste baze de date cloud poartă numele de NoSQL - Not only SQL şi sunt baze de date non relaţionale. Dezvoltarea NoSQL şi NewSQL ameninţă monopolul MySQL.

Diversele baze de date NoSQL existente azi pe piaţă prezintă diferite abordari. Ceea ce au în comun este faptul că nu sunt relaţionale. Principalul avantaj este acela că permit lucrul eficient cu date nestructurate precum e-mail, multimedia, procesoare de text. În prezent există multe companii care au dezvoltat propriile baze de date NoSQL. Cele mai populare sunt cele dezvoltate de către companiile mari Web, precum Amazon şi Google, din nevoia de a procesa cantităţi mari de date. Acestea au dezvoltat Dynamo şi Big Table ce stau la baza multor alte baze de date NoSQL existente acum pe piaţă.

Unul din motivele apariţiei NoSQL constă în nevoia aplicaţiilor web de a manipula cantităţi mari de date a pentru a putea rămane competitive. Cantitatea de informaţie digitală la nivel mondial este măsurată în exabytes. Conform unui studiu realizat de Universitatea Southern California cantitatea de date adaugată în 2006 a fost de 161 de exabytes. Doar un an mai tarziu, în 2007 capacitatea totală s-a ridicat la 295 de exabytes, reprezentând o creștere substanțială. Altfel spus, există o cantitate mare de informaţie în lume şi aceasta creşte exponenţial. De aici survine şi nevoia de baze de date web ce suportă cantităţi mari de date.

Conform unui studiu realizat de 451 Group Research intitulat MySQL vs. NoSQL and NewSQL între 2009 şi 2011 s-a înregistrat o scădere în utilizarea MySQL de la 82% la 73%. Studiul a fost efectuat asupra unui eşantion compus din 347 de utilizatori de baze de date opensource. 49% din respondenţi au abandonat soluţiile MySQL pentru a migra la soluţii NoSQL. Astfel se poate observa ameninţarea directă pe care o presupune NoSQL asupra MySQL.

Baze de date de tip graf

O bază de date de tip graf este o bază de date care folosește grafuri ca structura de stocare a datelor. Nodurile (entitățile) și muchiile (relațiile dintre acestea) sunt principalele proprietăți care definesc o astfel de modelare a bazei de date. Muchiile dintre noduri permit acestora să poată fi conectate direct, astfel încât în multe situații datele pot fi preluate cu o operație simplă.

1 Introducere în "Baze de date NoSQL"

Bazele de date nerelationale (NoSQL) au apărut din necesitatea unei simplificari a designului și a sclabilitatii pe orizontală și îmbunătățirii timpui de răspuns. În unele situații bazele de date nerelationale sunt mai rapide decât cele relaționale (mai ales la dimensiuni foarte mari ăla bazelor de date sau în aplicațiile Web real-time).

Tipuri de baze de date nerelationale:

1. Coloana : este asemănător cu modelul relațional dar datele sunt păstrate în memorie pe coloane și nu pe linii ceea ce permite anumite operații de compresie a datelor și îmbunătățește performanță în anumite situații de utilizare și spațiul de memorie folosit;

2. Document : conceptul general este de Document în care se păstrează codificat datele că și XML, JSON, etc. iar documentele au o cheie unică și pot fi asociate cu înregistrări iar ele se păstrează în colecții care se pot asocia cu tabele;

3. Cheie-valoare : se bazează pe conceptul de dicționar (map) prin care se asociază fiecărei cheie o anumită valoare;

4. Graf : se aplică pentru acele modele în care relațiile dintre elemente sunt multiple spre exemplu rețele de socializare, rute de transport;

5. Multi-model : înglobează mai multe modele de baze de date nerelationale intenționând să ofere și acele proprietăți pe care doar bazele de date relaționale le ofereau.

Există actualmente o serie de sisteme de gestiune care se deosebesc de cele clasic relaționale prin una sau ambele din caracteristicile de mai jos:

- folosirea modelului relațional al datelor;
- folosirea limbajului de interogare SQL.

Astfel de sisteme se caracterizeaza in plus prin unele din elementele de mai jos:

- sunt proiectate pentru medii distribuite sau paralele;
- sunt folosite pentru gestiunea documentelor și mai putin a datelor atomice;
- oferă garanții mai slabe de consistentă;
- unele sunt proiectate pentru a oferi consistentă la citire, specifică în sistemele clasice;
- în cazul arhitecturilor distribuite datele sunt prezente redundant în mai multe noduri. În felul acesta este realizată scalabilitatea și protecția împotriva căderilor accidentale ale unor noduri;

Bazele de date NoSQL oferă o scalabilitate nelimitată cu performanțe importante, împărțind nodurile în toată baza de date, indiferent cât de mare ar fi aceasta. Deși bazele de date tradiționale bazate pe RDBMS încă domină piața, bazele NoSQL își fac loc in general în medii care necesită o procesare rapidă și de mare viteză. Termenul NoSQL nu este unul nou, fiind utilizat încă de la sfârșitul anilor 1990, cu unele modele care stau la baza acestuia dezvoltate chiar mai devreme. NoSQL a intrat pe piață la un nivel redus la mijlocul anilor 2000, urmând ca deja în 2010 NoSQL sa fie utilizat pentru aplicații critice, necesitând deja garanții de performanță. În prezent, NoSQL este utilizat pentru gestionarea bazelor de date ale unora din cele mai mare magazine de date din lume pentru aplicații precum rețele sociale, analiza datelor de la senzori și analiza pieței de valori.

2 Scurt istoric Neo4J

Neo4J[1] reprezintă un sistem de management al unei baze de date de tip graf dezvoltat de compania Neo4J și este catalogat ca cel mai popular reprezentant al acestui tip de reprezentare a datelor. Acest sistem a pornit ca un concept în anul 2000 când s-a început dezvoltarea primului prototip Neo4J, ajungându-se că în 2002 să fie dezvoltată prima versiune de Neo4J, iar în 2003 acest sistem să fie lansat în producție, și să fie disponibil 24/7. Abia în 2010 este lansată versiunea 1.0 , pentru ca în 2016 să fie lansată versiunea 3.0. De asemenea, în 2017 Neo4J lansează "Graph Platform", un tool extrem de util pentru a vizualiza și a analiza date, și o să vedem în paginile următoare importanța acestuia.

Neo4J este implementat în Java și accesibil prin limbajul specific, dezvoltat de companie, Cypher.

Datorită eficienței integrării modelului pus la dispoziție de Neo4J, acesta a ajuns să fie utilizat de giganți din domeniu precum: Microsoft , eBay , IBM , LinkedIn etc.

[1] https://neo4j.com/

3 Modelarea datelor interconectate

Neo4j este o baza de date open-source fondată pe teoria grafurilor, fiind o soluție optimizată pentru a modela și interoga volume mari de date strâns relaționate, reprezentabile prin structuri de tip graf. Dinamismul,creșterea volumului datelor, precum și evolutia continuă a procesării informațiilora impus ieșirea din spațiul bazelor de date relaționale tradiționale și orientarea spre soluții NoSQL. O caracteristică unică a acestora este gradul ridicat de adaptibilitate la modelele reale de date.

Atât în cazul bazelor de date relaționale cât și în cazul unor soluții NoSQL (non-graph), procesul de modelare/design trece prin două faze:

- definirea conceptelor , a entităților și a interacțiunii dintre ele - model logic/real;
- materializarea modelului logic într-un model fizic/abstract.

De cele mai multe ori modelul logic este foarte diferit de modelul fizic. În cadrul unei organizații software în prima fază poate participa orice echipă nu neapărat tehnică(management / sales) pentru o mai bună definire a cerințelor sau conceptelor.În cea de a doua fază însă are loc abstractizarea modelului logic în funcție de opțiunea de stocare. Astfel gradul de înțelegere al modelului logic scade odată cu creșterea complexității datelor.

Marele avantaj al bazelor de date de tip graf și implicit utilizarea Neo4j este că modelul logic este același cu modelul fizic. Având acest mod de reprezentare uniformă sau altfel spus, o reprezentare intuitivă, ce oferă un mare grad de flexibilitate, adaptibilitate și expresivitate în modelarea datelor reale.

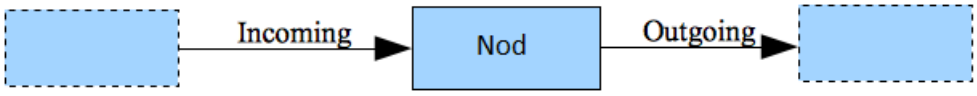
4 Reprezentarea datelor în Neo4J

În Neo4j datele sunt reprezentate prin noduri și relații. Atât nodurile cât și relațiile pot avea proprietăți. Relațiile au un rol foarte important în cadrul bazei de date de tip graf pentru că traversarea grafului și implicit manipularea datelor se realizează prin intermediul lor.

O relație implică întotdeauna două noduri, are o direcție și un tip identificat unic printr-un nume.

Relativ la un nod relațiile se pot clasifica în două tipuri:

- incoming
- outgoing



Atât proprietățile unui nod cât și cele ale unei relații pot fi indexate pentru îmbunătățirea performanței de traversare a grafului , similar cu indexarea coloanelor în bazele de date tradiționale.

Forțând o comparație cu bazele de date tradiționale, vă puteți imagina un nod ca o înregistrare dintr-un tabel, iar o relație ca o înregistrare dintr-un tabel de legătură sau o pereche de coloane din același tabel în cazul unei reprezentări tip denormalizat.

5 Limbajul de interogare Cypher

Neo4j are propriul limbaj de interogare a datelor organizate în structuri de graf. Este folosit conceptul de "Traversal" prin intermediul căruia se navighează în graf, se identifică drumurile și implicit se selectează nodurile pentru rezultatul unei interogări.

Limbajul Cypher[2] este un limbaj de interogare declarativ fiind foarte intuitiv și "human readable", putând fi înțeles cu ușurință chiar și de o persoană non-tehnică. Unele cuvinte cheie sunt inspirate din SQL cum ar fi: where, order by , limit, skip (echivalentul offset).

Limbajul este alcătuind din următoarele clauze :

- START - punctul de intrare în graf. Orice interogare în graf are cel puțin un nod de start;
- MATCH - șablonul pentru căutarea nodurilor și care este legat de nodul de start;
- WHERE - condițiile de filtrare a nodurilor / relațiilor;
- RETURN - rezultatul interogării;
- CREATE - creează noduri sau relații;
- DELETE - șterge noduri sau relații;
- SET - setează proprietăți noduri sau relații;
- FOREACH - update pe liste de noduri;
- WITH - împarte interogarea în mai multe părți distincte.

O formulare tipică Cypher arată relația dintre două noduri. Spre exemplu, pentru a crea un nou nod "University" avem urmatoarul query:

```
CREATE (university:University { id:1, name:"Alexandru Ioan Cuza"})
```

În acest exemplu, `university` este numele variabilei pe care am l-am atribuit nodului creat în baza de date. Am marcat nodul că fiind un obiect de tip `University`. Acest tip de obiect are un atribut numit `id` care este un număr unic asignat unei univeristati și de-asemenea un atribut `name` , numele universității. Putem folosi acest id unic pentru a interoga baza de date pentru a cere informații .

Acum, pentru a selecta universitatea din baza de date , putem executa următorul query:

```
MATCH  (u:University {id:1})
RETURN u;
```

Muchiile sunt parcurse folosind o sintaxă bazată pe săgeți. (n)–(m) descrie orice relație între două noduri , în timp ce (n)–>(m) descrie orice relație directata de la un nod la altul. Tipul relației, direcția și proprietățile relației pot fi adăugate opțional sintaxei, prin plasarea unor paranteze în mijlocul săgeții.

Presupunând că avem deja creat și un nod `Faculty` , putem adaugă relația de tip `contains` între nodul de tip `University` și cel de tip `Faculty` prin executarea următorului query:

[2]https://neo4j.com/developer/cypher-query-language/

```
MATCH  (u:University {id:1})
MATCH  (f:Faculty {id:1})
CREATE (u)-[:contains]->(f)
```

Mai jos este prezentat un exemplu de interogare scrisă în Cypher, căutând un nod `Person` cu proprietatea `name` având valoarea "John", și numele prietenilor prietenilor săi. Aici, primului nod îi este dată variabila numită j și prietenii prietenilor săi le este dată variabila fof, variabile care sunt variabile de context și care există doar în interiorul interogării.

```
MATCH (j:Person {name: 'John'})-[:FRIEND]->()-[:FRIEND]->(fof)
RETURN j.name, fof.name
```

6 Concluzii

Lumea reală este puternic interconectată, iar bazele de date de tip graf țintesc să mimeze perfect acele relații într-un mod cât de intuitiv posibil. Bazele de date de tip graf sunt extrem de utile în înțelegerea seturilor de date mari în diferite scenarii.

Acest tip de baze de date este situat cu siguranță pe o pantă ascendentă, iar conceptul de "big data"[3] cu atât mai mult.

[3]https://www.sas.com/en_us/insights/big-data/whatisbigdata.html

Rolul bazelor de date de tip graf în rețelele sociale

1 Rețele sociale

Rețelele sociale sunt printre cele mai populare siteuri Web și încă sunt în continuă creștere, și una foarte rapidă. Acestea pun la dispoziție mecanisme pentru stabilirea identităților, crearea de relații, și distribuirea informației. Graful rezultat oferă bazele comunicării, distribuirii și localizării conținutului. Graful social, denumit și rețea socială, are multiple proprietăți. Nodurile au diverse tipuri reprezentând entități precum oameni, fotografii, videoclipuri, în timp ce muchiile reprezintă relațiile dintre noduri, precum o relație de prietenie sau de un tag aplicat unei postări. Rețelele sociale sunt uriașe: numărul nodurilor și al muchiilor sunt în ordinul miliardelor. Sunt ușor scalabile datorită distribuției nodurilor; unele noduri au mai multe conexiuni că altele, acest fapt conducând la o "distribuție de putere" între gradele nodurilor.

Aplicațiile sociale emit interogări și tranzacții asupra unei rețele sociale. Volumul acestora de muncă include atât citirea cât și actualizarea interogărilor. În general, actualizările includ un "update" după cheia primară (spre exemplu, adăugarea unei noi persoane sau adăugarea unei muchii reprezentând o relație de prietenie între doi oameni, un tag a unui prieten într-o fotografie sau actualizarea unui status). Activitatea rețelei este dominată de operația de citire. Această variază de la o simplă operație de căutare până la operații complexe precum găsirea celui mai scurt drum între două noduri, sau căutarea nodurilor care satisfac anumite condiții de-a lungul unui drum în graf. Câteva dintre aceste interogări pot fi exprimate că interogări SQL, în timp ce altele (precum interogarile de accesibilitate) nu pot.

Pentru a administra datele unei rețele sociale, sistemul fie folosește un mecanism relațional, care vine cu însușiri favorabile precum indexare sau suport tranzacțional, fie folosește un mecanism sub formă de graf. În primul caz, interogarile de accesibilitate sunt rescrise, de exemplu prin limitarea lungimii drumului exprimat în SQL. În cazul al doilea, sistemele ce stochează și administrează grafuri nativ folosind noduri și muchii, pot suporta interogări mult mai expresive, care de-asemenea conduc la schimbări aplicate execuției interogarilor (schimbări legate de execuția interogării sau de modalități de optimizare).

Interogările reprezentative pentru rețelele sociale includ următoarele:

- actualizări asupra nodurilor individuale (spre exemplu , schimbarea statusului unui utilizator dat);
- inserări de muchii(spre exemplu, pentru a conecta utilizatori între ei);
- interogări aplicate asupra unui domeniu(spre exemplu, pentru a obține toate nodurile direct conectate unui utilizator);

- interogări de tip "shortest-path"(spre exemplu, găsirea relației de conexiune dintre doi utilizatori);
- interogări de accesibilitate(spre exemplu, găsirea tuturor angajaților care au acces la a trimite raport către CTO);

2 Consecințe ale folosirii bazelor de date de tip graf

Bazele de date de tip graf dispun de mai multă putere analitică decât bazele de date relaționale în contextul rețelelor sociale. În esență, rețelele sociale sunt tot grafuri. Astfel, bazele de date de tip graf reprezintă o potrivire excelență pentru modelarea, depozitarea și interogarea rețelei sociale. Aceasta, împreună cu relațiile sale pot fi reprezentate prin grafuri deoarece prezintă similarități, în sensul că, fiecare persoană din rețea va juca rolul unei entități individuale, și poate avea relații cu alte entități.

Dacă alegem să folosim baze de date relaționale în implementarea unei rețele sociale, vom avea de-a face cu mai multe situații:

1. Interogările aplicate rețelei sociale vor fi practic bazate pe relații. Relațiile într-o baza de date relațională sunt "rezolvate" prin operația `join` între tabele. Să presupunem următorul scenariu: "prietenii prietenului și paginile apreciate de aceștia", și un query care să rezolve această cerință. În SQL trebuie să scriem un număr considerabil de operații de tip `join` între multe tabele, care vor fi costisitoare la rulare. Acest lucru se datorează faptului că bazele de date relaționale nu stochează relații, acțiune ce se întâmplă în bazele de date de tip graf, și face ca un astfel de query necesar pentru scenariul de mai sus să fie foarte ușor de scris și eficient la rulare.

2. Performanța bazelor de date relaționale este slabă în relațiile "many to many". Într-un query simplu aplicat pe o rețea socială, poate fi necesară găsirea unei soluții parcurgând relații "many to many". Având în vedere că fiecare nod dintr-o baza de date de tip graf stochează relații care sunt preprocesate, nu mai este nevoie de operația join. Necesită, în schimb, o parcurgere a nodurilor, și verificarea fiecărui nod până la găsirea celui vizat.

3. Dimensiunea unei rețele sociale este scalabila. Asta înseamnă că poate avea oricând noi entități și relații, ceea ce va necesita schimbarea schemei unei baze de date relaționale, în timp ce bazele de date de tip graf permit o scalabilitate ridicată a schemei.

4. Mai multe entități din rețea au permisiuni de scriere și citire, astfel încât este nevoie de o concurență asupra citirii și scrierii. Bazele de date relaționale vor executa un control al concurenței, folosit pentru a asigura proprietățile ACID[1]. Problema este că, nu întotdeauna este nevoie de a asigura proprietățile ACID și controlul concurnetei. Spre exemplu, dacă 1000 de oameni ce urmăresc o persoană, iar acea persoană face o postare, nu mai este nevoie să se execute controlul concurenței dacă scriem în fiecare nod într-o bază de date de tip graf.

[1] https://en.wikipedia.org/wiki/ACID

5. Performanța unui query într-o baza de date de tip graf este de 1000 de ori mai mare decât într-o baza de date relațională (bazat pe o statistică realizată de IMDB și Neo4J) în structurile de tip graf.

6. Reprezentarea unei rețele sociale ca un set de tabele ar fi o operație complexă. Și, câteodată, ar fi nevoie de o reindexare dacă vreo schimbare apare în rețea, datorită faptului că mărimea rețelei nu este statică.

De ce alegem o baza de date de tip graf in detrimentul altora

Sunt câteva aspecte pe care trebuie să le luăm în considerare atunci când dorim să alegem o baza de date pentru o rețea socială:

- Tehnologia unei baze de date de tip graf în acest moment nu este atât de evoluată precum tehnologiile bazelor de date relaționale.
- Puterea analitică a unei baze de date de tip graf în contextul unei rețele sociale este mult mai ridicată decât a bazelor de date relaționale.
- Nu toate bazele de date de tip graf au capacitate și performanță semnificative pentru interogări analitice pe seturi mari de date.
- Folosește operații pe grafuri, ceea ce ne permite un nivel foarte ridicat de abstractizare în exprimarea unei interogări.

Considerând punctele enumerate mai sus, alegerea unei baze de date de tip graf pentru a stoca și a administra o rețea socială ar trebui să fie pliabilă pe măsura cerințelor. Spre exemplu, nu reprezintă o soluție bună o administrare completă a rețelei cu o baza de date de tip graf, deoarece nu urmărește proprietățile ACID ale tranzacțiilor precum o fac bazele de date relaționale.

Nu toate bazele de date de tip graf vor permite cu siguranță stocarea unei rețele sociale foarte mari (milioane de utilizatori). Dacă am alege să împărțim baza de date pentru a evita această problema, va conduce către alte probleme. Bazele de date relaționale sunt excelente în contextul datelor statice și predictibile tabelar. Dar rețelele sociale au tendința de a fi mai puțin statice și predictibile, astfel, reprezintă candidați ideali pentru bazele de date de tip graf.

3 Probleme şi soluții sugerate

Probleme în alegerea unei baze de date de tip graf în contextul unei rețele sociale

- Probleme teoretice precum "creșterea pătratică" , de exemplu: dacă pe Twitter, fiecare utilizator începe să îi urmărească pe toți ceilalți , încărcarea datelor va reprezenta o problema dificilă.
- Nu putem executa operații analitice asupra datelor într-o bază de date de tip graf precum găsirea unei sume sau a unei medii etc., cel puțin nu folosind comenzi de bază.
- Găsirea nodului de început, dacă acel nod nu este conectat la nodul de unde interogarea de căutare este generată reprezintă o problema.

- Dacă sunt milioane de noduri într-o rețea, va fi dificil să plasăm toate nodurile într-un singur sistem.

Soluții

- Dacă o astfel de problemă apare, putem stoca proprietățile în noduri dar putem evita stocarea relațiilor dintre noduri și putem crea noduri către care se poate ajunge de oriunde
- Putem implementa multe dintre acestea noi înșine
- Putem folosi într-o baza de date relațională pentru a stoca legături către toate nodurile din rețea. Putem căuta un nod folosind tabela relațională respectivă dar după ce am găsit nodul respectiv, putem folosi parcurgerea grafului pentru a găsi un nod particular.
- Putem împărți baza de date în mai multe componente și le putem plasa în sisteme diferite, chiar dacă nu este o operație atât de ușor de realizat.

4 Studiu de caz

Rețea socială formată din studenți și facultăți aparținând unei universități, la susținerea lucrării de licență

Această rețea socială se ocupă cu susținerea lucrării de licență a studenților aparținând mai multor facultăți. Studentul poate folosi laboratoare pentru partea de "research" a lucrării sale. Componentele rețelei sunt astfel: `Student`, `Faculty`, `Thesis`, `Laboratory`.

Câteva interogări ce ar putea apărea în această rețea socială sunt: "Lucrările de licență cu același titlu" , "Studenții care au același titlu al lucrării de licență" , "Studenți care au același subiect al lucrării de licență dar folosesc laboratoare diferite pentru research" etc. Pentru primele două exemple, nu reprezintă o problemă folosirea unei baze de date relaționale, dar interogările mai complexe , precum cea din al treila exemplu, am avea nevoie de tehnologii mai rapide și care folosesc mai puțin spațiu de stocare. Din acest motiv, bazele de date de tip graf reprezintă cea mai bună soluție pentru acest gen de situații. Reprezentăm fiecare componentă că un nod al grafului iar muchiile că relațiile ce conectează oricare două noduri, fiecare nod având anumite proprietăți în el. De exemplu, nodul `Student` are proprietăți precum nume , email, număr matricol.

Provocare

Rețeaua socială pe care am creat-o pentru studenți și facultăți poate la un moment dat să aibă mii de noduri, și la fel de bine este posibil să avem nevoie să executăm interogări complexe bazate pe relații dintre entități, asemenea interogări presupuse mai sus. Folosirea unor baze de date relaționale pot aduce situații de reindexare sau renormalizare. Astfel, alegerea unui model al bazei de date pentru rețeaua socială, care este flexibil și scalabil, și de-asemenea prezintă o performanță ridicată de execuție este cu adevărat o provocare, mai ales dacă dorim că în viitor să adăugăm noi implementări ale aplicației (precum chatting).

Soluție

Folosirea unei baze de date de tip graf poate accelera semnificativ viteza de execuție a interogărilor. De-asemenea , schema acesteia o face flexibilă și scalabila. Bazele de date de tip graf sunt construite să execute interogări complexe asupra relațiilor "many to many", și acordă o mai mare importanță asupra relațiilor dintre entități. Avantajele acestui tip de baze de date îl prezintă concurența mare, împreună cu latenta scăzută asupra citirii și scrierii, eficiența depozitării datelor, extensibilitatea, prezintă costuri foarte mici ale operațiilor.

5 Exemplul Facebook

Facebook este un serviciu de rețea socială online care a fost lansată în 2004. În Septembrie 2015, Facebook avea 1.55 miliarde de utilizatori activi lunar, dintre care 1.01 miliarde erau activi zilnic, în timp ce în Iunie 2018 erau 2.20 miliarde de utilizatori activi lunar, dintre care 1.15 miliarde activi zilnic. Facebook reprezintă cu adevărat o rețea socială impresionantă, mai ales dacă luăm în calcul și următoarele statistici (Iunie 2018): sunt create 5 noi profile în fiecare secundă, se realizează aproximativ 300 de milioane de încărcări de fotografii zilnic, la fiecare minut sunt adăugate 510.000 de comentarii și 293.000 de statusuri sunt actualizate, 50% dintre persoanele cu vârsta cuprinsă între 18 și 24 de ani deschid Facebook atunci când se trezesc, iar 42% dintre cei preocupați de partea de marketing declară că Facebook este cel puțin important, chiar critic pentru desfășurarea afacerii lor.

Funcționalitate

Facebook este o rețea socială pe care utilizatorii își pot înregistra un profil personal și se pot conecta cu alți utilizatori (aceste conexiuni fiind numite prietenii). Utilizatorii pot adaugă postări text, fotografii și videoclipuri, evenimente și invitații, își pot posta informații personale pe profil, și pot adăuga postări text sau media pe profilele prietenilor. Utilizatorii de-asemenea pot crea sau se pot alătura unor grupuri publice sau private unde de-asemenea sunt realizate postări. De-asemenea pot fi create și profile de business, care pot fi folosite similar unei pagini personale. Utilizatorii pot controla ce părți din profilul lor pot fi văzute de alții. Aceștia pot aprecia sau comenta postările la care au acces. Bineînțeles, utilizatorii pot comunica și în privat, iar paginile de business pot alege să creeze o reclamă doar unui anumit tip de utilizator.

Graf social

Graful social al rețelei Facebook prezintă o varietate de entități și conexiuni. Precum funcționalitatea aplicației este complexă, așa este și graful. Acesta poate fi vizualizat în figura 2.0.

figura 2.0

În această vizualizare, entitățile principale sunt utilizatorii, postările, comentariile, paginile, grupurile și evenimentele. În plus, există multiple tipuri de entități în graf, precum fotografii, videoclipuri, canale de comunicare, mesage, notificări, locuri, aplicații, reclame, și diferite tipuri de muchii precum tag, follow, checked-into, read, seen, și multe altele. Graful este orientat, și, în timp ce majoritatea muchiilor sunt unidirecționale, muchia friend este bidirecțională. Graful este rar, dar puternic clasterizat. Majoritatea utilizatorilor au mai puțin de 200 de prieteni, iar media prietenilor unui utilizator este de 99.

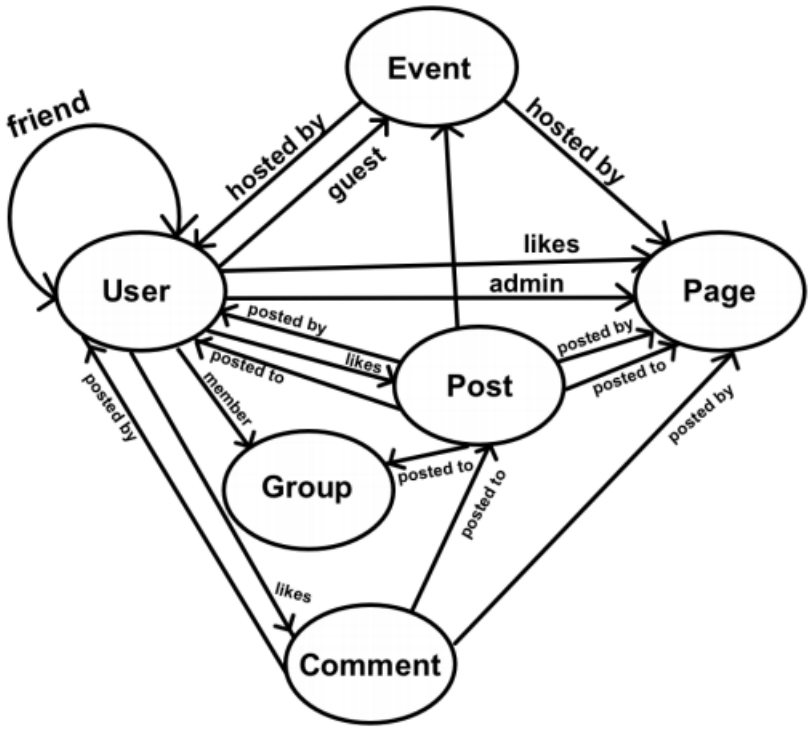


figura 2.1: Graful social Facebook (simplificat)

Arhitectură

Inițial, Facebook stoca toate datele în MySQL. În ziua de astăzi, datele pentru diferitele servicii din Facebook sunt tocate în mai multe soluții de stocare. Datele pentru aplicațiile care au nevoie de o latență scăzută a accesului la baza de date sunt stocate în RocksDB. Datele precum mesajele, prieteni apropiați, indexarea căutărilor sunt stocate în HBase.Totuși, nu toate datele au fost mutate din MySQL. Nodurile și muchiile care sunt adăugate în graful social sunt mai întâi scrise într-o instanța de MySQL, înainte de a fi replicate și adăugate difertelor servicii din Facebook.

Deși stochează datele majoritar în baze de date relaționale sau depozite cheie-valoare, Facebook a implementat un nivel superior de tip graf peste cel de stocare, pentru a pune la dispoziție graful social diferitelor servicii care cer informații.

6 Concluzii

Prin folosirea unei implementări de tip graf completă a aplicației ca metodă de stocare în locul unei baze de date relaționale, serviciile unei rețele sociale cu siguranță vor prezența o îmbunătățire în sensul dezvoltării și performanței. În serviciile unde modelul datelor nu este determinat complet, și pot interveni schimbări pe parcurs, acest tip de baze de date au avantajul de o schemă flexibilă, care poate conduce la costuri de administrare mai scăzute. În mare, bazele de date NoSQL, și în special cele bazate pe grafuri oferă o scalabilitate mai bună, împreună cu o flexibilitate și costuri de administrare mai avantajoase, în comparație cu ceea ce bazele de date relaționale sunt capabile să pună la dispoziție.

În timp ce sunt prezinte avantaje considerabile pentru a folosi baze de date de tip graf pentru aplicațiile bazate pe grafuri, infrastructura curentă din jurul bazelor de date de tip graf nu apare să fie atât de robustă încât să se riște ca fiind o soluție principala de stocare a datelor pentru servicii sociale în rețea, precum Facebook. Folosirea bazelor de date NoSQL ca un nivel intermediar între stocarea relațională și aplicație oferă cam aceleași avantaje, păstrând în același timp avantajele stocării relaționale, dar implică costuri suplimentare de implementare și administrare.

Modelul propus

1 Experienţa utilizatorului

Experiența utilizatorului reprezintă modul cum se simte o persoană atunci când interacționează cu un produs software și este unul dintre cei mai importanți factori pentru succesul unei aplicații, mai ales pentru succesul unei rețele sociale. Practic, joacă un rol extrem de important în păstrarea utilizatorilor existenți și atragerea celor noi. Experiența utilizatorului este acoperită de mai mulți factori, unii factori fiind controlabili de designeri și dezvoltatori și alți factori care reprezintă preferințele utilizatorului. Acești factori includ uzabilitatea, accesibilitatea, performanța, designul, estetica, utilitatea, ergonomia, chiar și marketingul.

Experiența utilizatorului este o practică care stă la intersecția dintre știința comportamentală , dezvoltarea web , și cunoștiințele asupra domeniului specific. Este o abordare umană cu scopul de a înțelege cum pot oamenii să înțeleagă tehnologia și cum să le poți oferi cele mai bune experiențe web posibile. Considerând următoarele:

- 88% dintre utilizatorii online prezintă o slabă probabilitate de a mai reveni pe un site după o experiență nefericită;
- 94% din utilizatori își formează primă impresie legată de un siteanalizând designul.

Astfel, pentru a respectă cerințele legate de experiență utilizatorului pe aplicație, informația oferită acestuia trebuie să respecte următoarele:

- siteul web trebuie să livreze conținut specific conform cu ceea ce pretinde că trebuie să ofere;
- siteul trebuie să fie ușor navigabil;
- designul trebuie să fie atractiv, inteligent, și ușor de urmărit;
- utilizatorii nu ar trebui să aibă probleme în găsirea informațiilor cheie în timp ce navighează pe site;
- conținutul trebuie să fie accesibil oricui;
- siteul are nevoie de autoritate și de suficientă calitate astfel încât conținutul să fie credibil

2 Despre fundaţia Alumni

Fundația Alumni[1] a Universității "Alexandru Ioan Cuza"[2] din Iași reprezintă o comunitate online dedicată absolvenților universității, cu scopul ca aceștia să rămână în contact cu universitatea și cu foștii colegi. Aceasta a fost constituită pe baza hotărârii Senatului Universității "Alexandru Ioan Cuza" din Iași în ședința din data de 23.10.2006 și își propune să dezvolte și să consolideze comunitatea puternică a absolvenților, să îi ajute și, la rândul ei, să le ceară ajutorul în atingerea obiectivelor sale.

Nu există taxă de înscriere sau cotizaţie. Fundația Alumni UAIC oferă membrilor săi:

- Un site pentru toți alumni UAIC;
- Newsletter;
- Cluburi alumni;
- Sprijin în organizare de Cursuri festive și Reuniuni aniversare;
- Oferte de angajare și recrutare;
- Participări la întalniri, proiecte, simpozioane și evenimente.

Astfel, prin înscriere în comunitate, vei fi conectat la activitățile universității prin invitații la conferințe, seminarii sau evenimente speciale, păstrand legătura strânsă cu profesorii și colegii tăi şi devenind persoană resursă sau mentor pentru generațiile viitoare.

[1] http://www.alumni.uaic.ro/
[2] http://www.uaic.ro/en/

3 Descrierea aplicaţiei şi funcţionalităţi

Aplicația construită este o aplicație web, mai exact o rețea socială, având ca scopuri principale îmbunătățirea versiunii curente a siteului Alumni deja existent și punerea la dispoziție utilizatorilor săi noi funcționalități.

La accesarea siteului, aceasta este prima pagină care îi este arătată utilizatorului (figura 3.0).

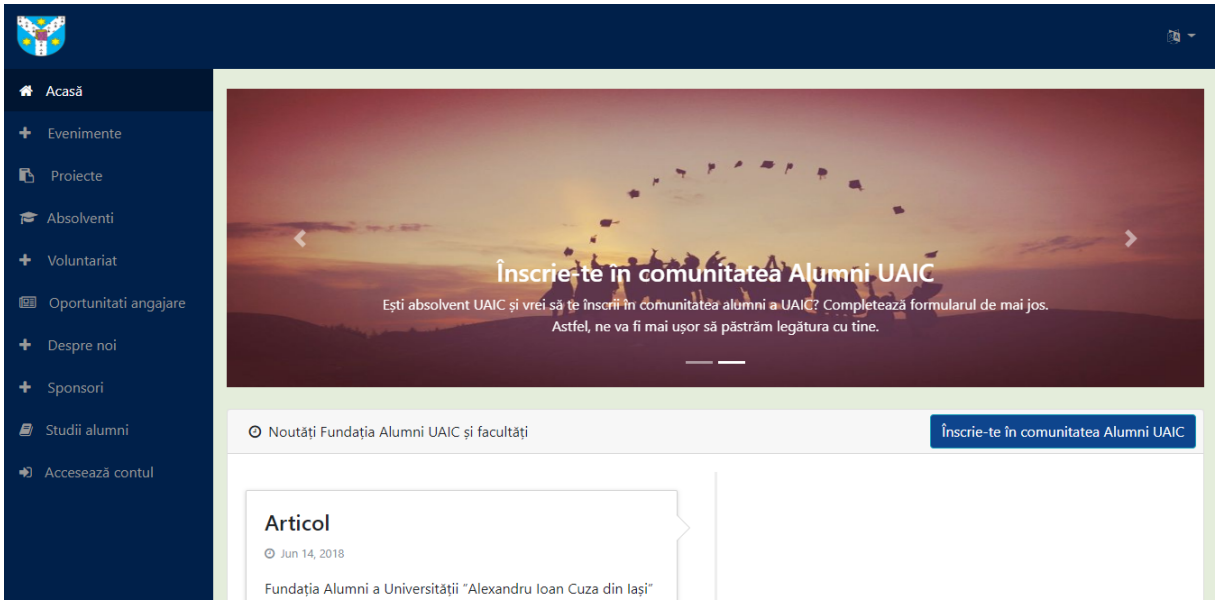


figura 3.0: Prima pagină a aplicatiei

Prima pagină este disponibilă oricui accesează linkul. Aceasta prezintă un meniu de navigare în stânga, care conține butoane ce informează utilizatorul la ce informații este restricționat, fără a avea un cont pe aplicație.

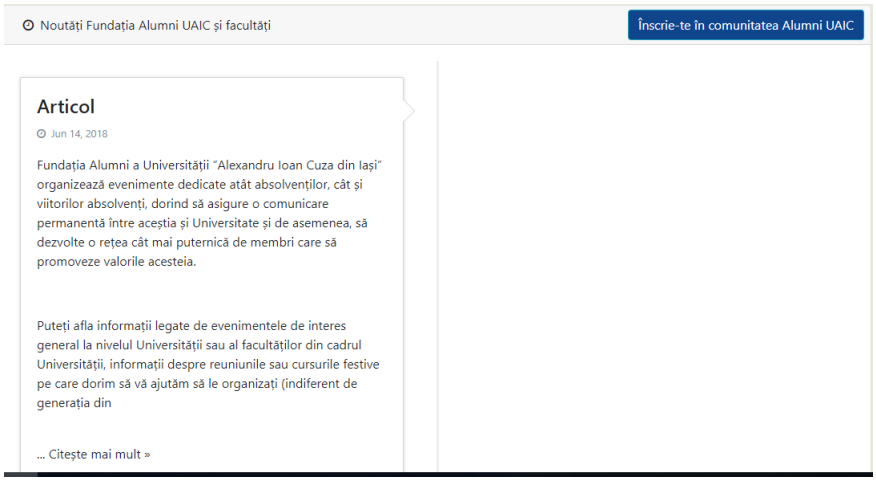


figura 3.1: Articole noi publicate

Pagina principală afișează utilizatorului articole ce pot fi publicate doar de un utilizator autorizat (administrator - figura 3.2), în ordinea celor mai noi publicate. Articolele pot fi modelate în orice fel (formatare text, fișiere media etc.), deoarece la publicare, conținutul articolului este randat în format HTML. Aceasta reprezintă principala modalitate prin care universitatea înștiințează vizitatorii paginii de noi evenimente sau noutăți.

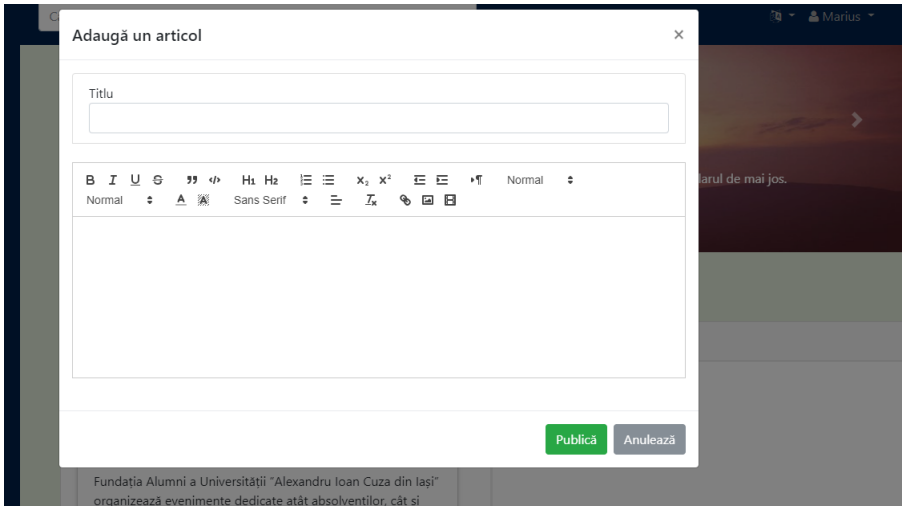


figura 3.2: Adăugare articol nou de către un administrator

Asemenea articolelor, pot fi create și evenimente pe care vizitatorii paginii le pot vizualiza și pot afla detalii. Acestea pot fi create tot de un cont autorizat, și de-asemenea pot fi modelate în orice fel (formatare text, fișiere media etc.), deoarece la publicare, conținutul articolului este randat în format HTML.

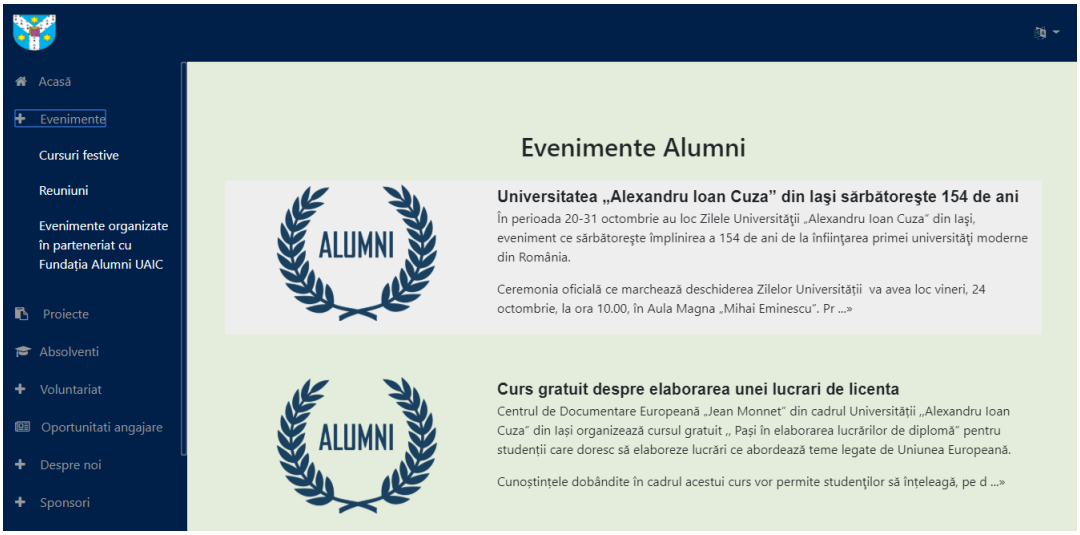


figura 3.3: Evenimente Alumni

La apăsarea butonului "Înscrie-te în comunitatea Alumni UAIC" utilizatorul este redirecționat către o pagină (figura 3.4) unde este prezent un formular, pe care acesta ar trebui să îl completeze cu informațiile necesare și la final să își creeze contul. Din acel moment, utilizatorul are un cont și poate beneficia de serviciile de socializare puse la dispoziție de aplicație.

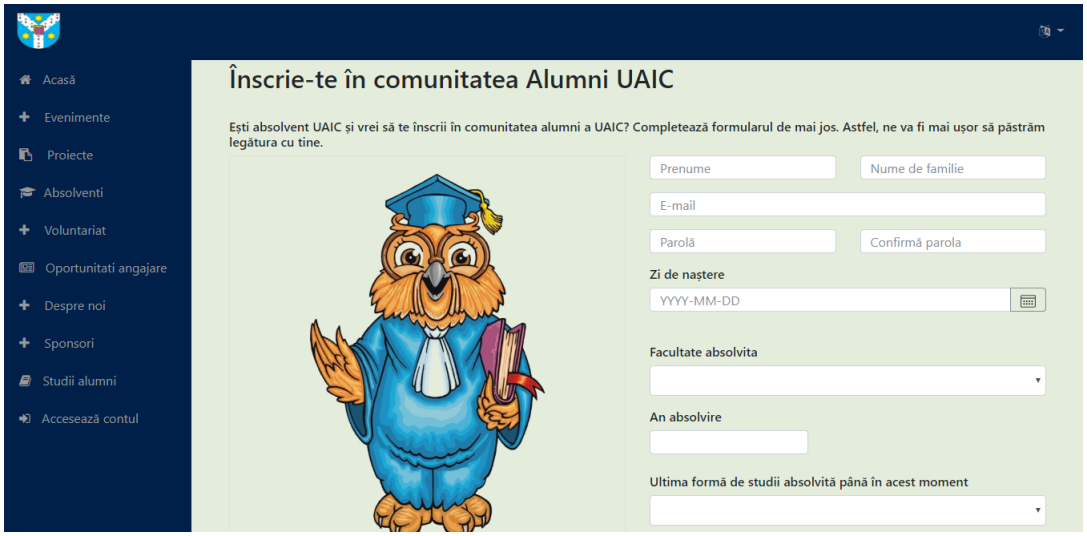


figura 3.4: Pagina afișată la apăsarea butonului de înscriere

Celelalte adrese din meniul de navigare oferă de-asemenea informații folositoare utilizatorului.

Prin apăsarea butonului "Accesează-ți contul" din meniul de navigare, utilizatorul este direcționat către o pagină, unde apare un header (figura 3.5), ce conține două câmpuri, email și parolă, cu ajutorul cărora completate corect cu, contul creat de utilizator, acestuia îi este afișată o pagină cu mai multe funcționalități ce țin de socializare.

figura 3.5: Header de autentificare

După autentificare, utilizatorului îi va fi afișată pagină de Cronologie (figura 3.6), unde acesta poate vedea postări ale altor utilizatori din comunitate, în ordinea postării lor.

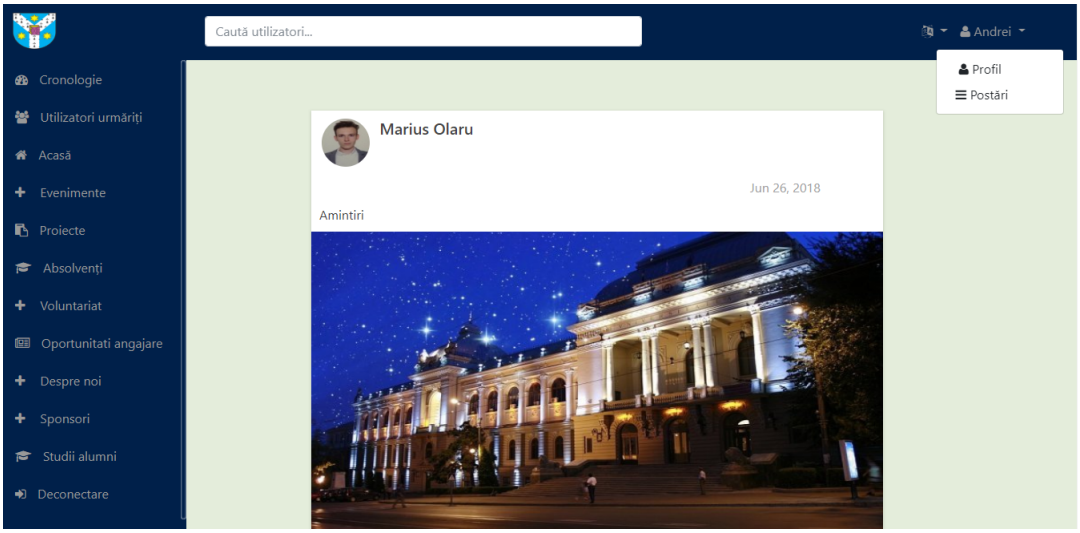


figura 3.6: Pagina de cronologie

Utilizatorul poate intra direct pe profilul unui utilizator apăsând pe numele lui afișat la o postare din cronologie, sau poate folosi funcția "Caută utilizatori" (figura 3.7) din topul paginii.

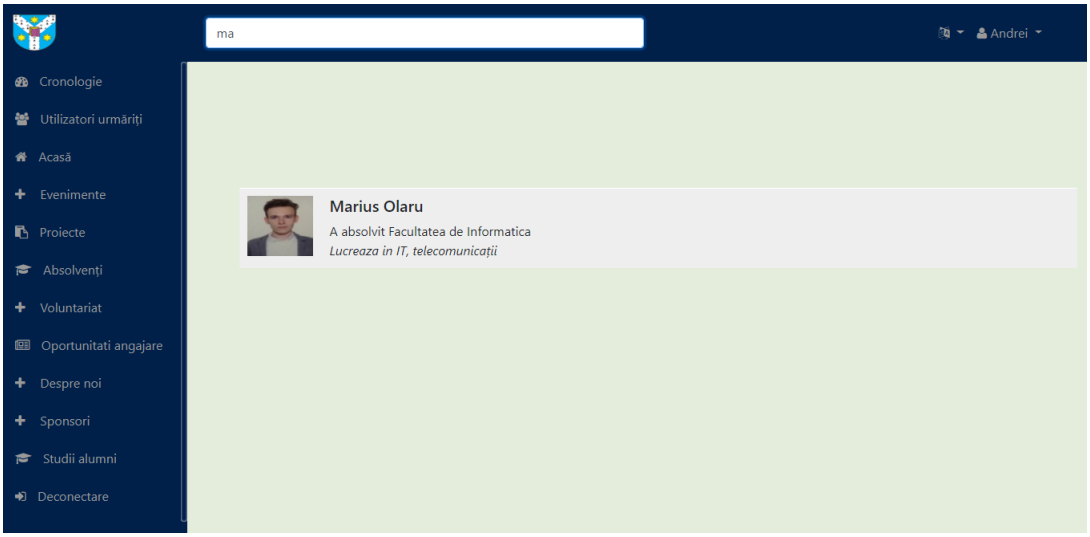


figura 3.7: Căutare utilizatori

Accesând adresa altui utilizator (figura 3.8), se poate vizualiza profilul acestuia, ce cuprinde detalii expuse de respectivul, precum: nume, prenume, poză de profil, adresă de email, domiciliu, facultate absolvită, loc de muncă, alte profile pe social media etc.

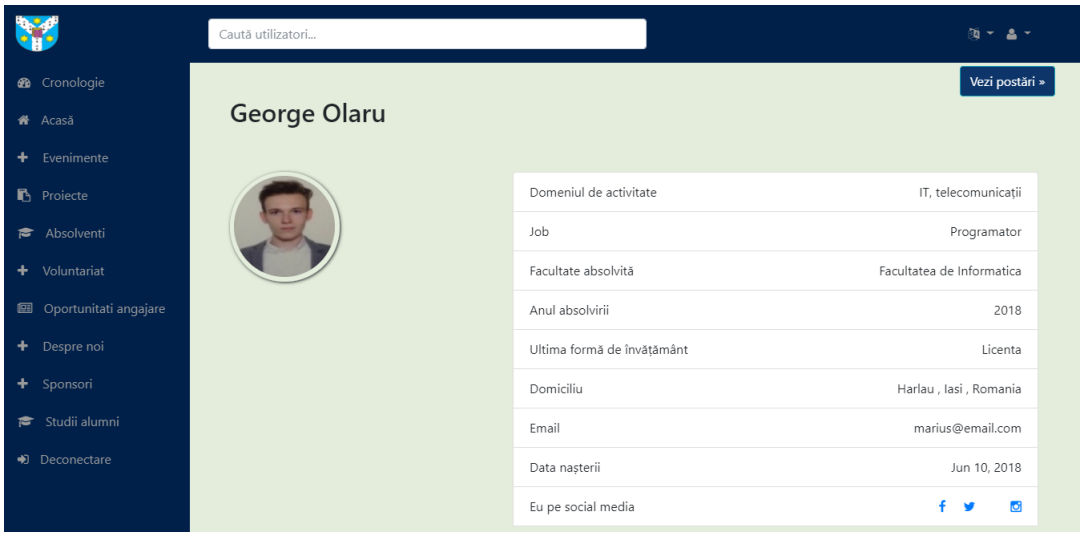


figura 3.8: Profilul unui utilizator

Apăsând pe butonul "Vezi postări", se pot vizualiza postări efectuate de utilizatorul vizitat (figura 3.9).

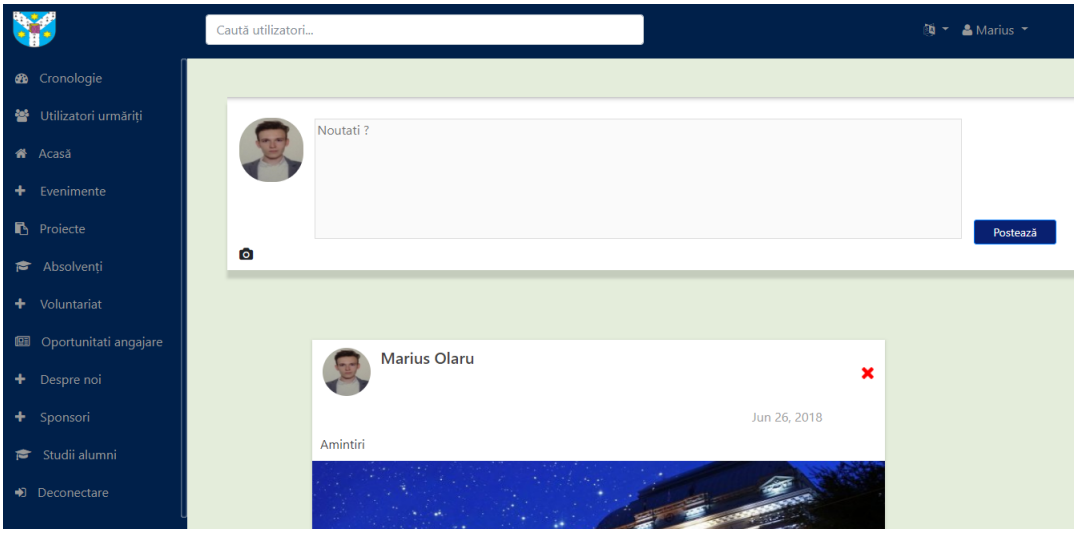


figura 3.9: Postările unui utilizator

Utilizatorul poate crea postări (figura 3.10) intrând pe pagina "Postări" din meniul situat în dreapta sus. Acesta poate adăuga un mesaj, opțional împreună cu o fotografie. Această acțiune, de a distribui amintiri legate de universitate, sau poze actuale, are rolul de a crea o stare plăcută celui care îi vizitează profilul, și găsește astfel poze realizate poate cu foarte mulți ani în urmă (poze cu alți colegi, poze cu universitatea, poze cu amfiteatre, laboratoare) sau chiar poze actuale ale utilizatorului.

Comunicarea între utilizatori am hotărât să fie realizată prin email sau prin rețelele de social media, dacă utilizatorul și-a expus pe pagina de profil aceste informații.

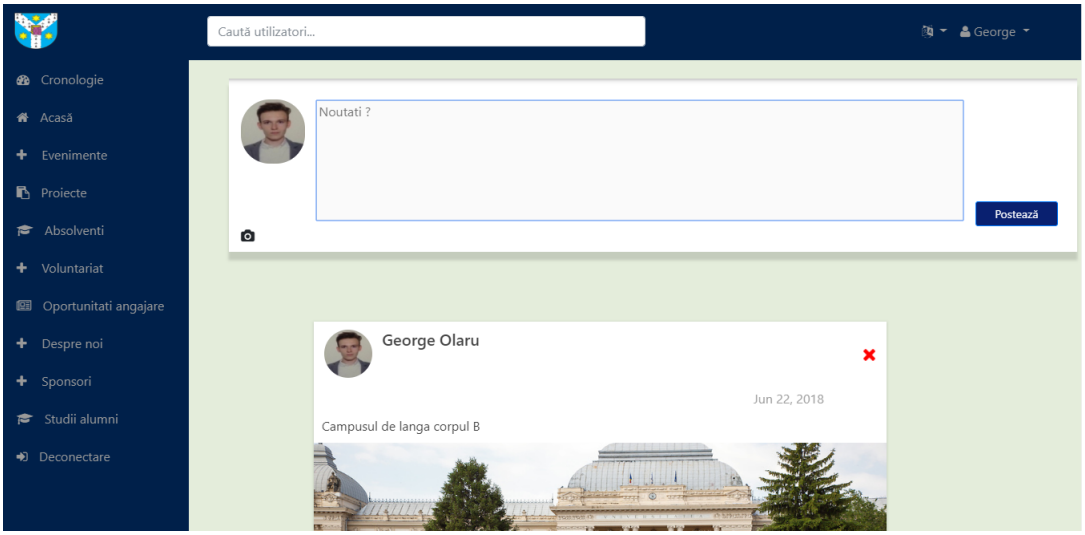


figura 3.10: Crearea postărilor

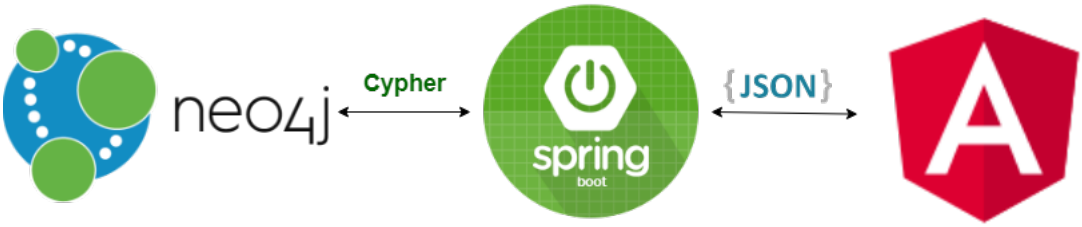
4 Obiectivele aplicaţiei

În primul rând, pagina Alumni este o pagină care pune la dispoziție celor ce o accesează informații legate de universitate, informații ce sunt publicate de un utilizator autorizat. Acestea pot fi articole, evenimente, proiecte, oportunități de angajare etc. Pentru ca un vizitator al paginii să poate avea acces către partea de socializare a aplicației, acesta trebuie să completeze un formular cu datele necesare și să își creeze un cont. În acest moment intervine și obiectivul principal al aplicației, acela de a crea o comunitate de foști studenți ai Universității "Alexandru Ioan Cuza", pentru că aceștia să distribuie amintiri legate specifice facultății și să se poată regasi chiar și după o lungă perioada de timp. Utilizatorii au profile unde își pot afișa diferite date și o secțiune în care pot posta diferite mesaje sau imagini.

În concluzie, putem spune că, pagina Alumni cu partea de rețea de socializare integrată este ca o agendă a studenților Universității "Alexandru Ioan Cuza" din Iași.

Arhitectura aplicaţiei şi implementare

1 Arhitectura aplicaţiei



Cererea din clientul Angular este trimisă prin Http către serviciul Spring Boot, unde este procesată, și se execută toată logică, care în majoritatea cazurilor cuprinde interacțiune cu baza de date Neo4J, prin limbajul specific Cypher. Apoi, rezultatul ce trebuie returnat de serviciu este împachetat într-un JSON[1] și este trimis către clientul Angular, unde este despachetat și modelat pentru a fi afișat utilizatorului.

2 Tehnologii folosite

2.1 Spring Boot

Există mai multe modalități stabile și de încredere pentru a construi aplicații web în ecosistemul Java, și având în vedere popularitatea, cu siguranță frameworkul Spring este soluția primară.

Spring oferă o cale puternică pentru a construi o aplicație web, având suport pentru mecanismul de dependency injection, administrarea tranzacțiilor, securitatea aplicației, suport pentru REST API, framework MVC și multe altele.

În mod tradițional, aplicațiile Spring întotdeauna au necesitat o configurație semnificativă și, din acest motiv, pot presupune situații foarte complexe pe parcursul dezvoltării. Acesta este locul unde intervine Spring Boot. Spring Boot își propune să realizeze dezvoltarea aplicațiilor web folosind Spring mult mai rapid și mai ușor. Acest framework se bazează pe principiul "convenție deasupra configurației".

Spring Boot vine cu multe funcționalități, printre care cele mai importante:

- module de start pentru a simplifica configurațiile pentru dependințe (figura 4.1)

[1] https://en.wikipedia.org/wiki/JSON

- auto-configurare
- Tomcat, Jetty sau Undertow, incorporate
- aplicații Spring independente
- nu necesită configurație XML

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.2.RELEASE</version>
    <relativePath/>
</parent>
```

figura 4.1: Spring Boot starter

Prin această modalitate, este nevoie doar să fie specificată versiunea dependinței o singură dată în părinte. Valorea este apoi folosită pentru a determina versiunile pentru majoritatea celorlalte dependințe - precum proiecte Spring sau librării. Avantajul acestei abordări este acela că elimină potențiale erori cauzate de versiuni incompatibile de librării. Când este nevoie de un update asupra versiunii Boot, trebuie doar o singură schimbare, iar restul proiectului va fi actualizat implicit.

Maven

Maven este un instrument de automatizare folosit la operațiunea de build a unui proiect, și este folosit în principal pentru proiecte Java. Maven adresează două aspecte: mai întâi, descrie cum un software este construit, apoi, îi descrie dependințele. Spre deosebire de instrumentele anterioare precum Apache Ant[2], folosește convenții pentru procedura de build, și este nevoie doar ca excepțiile să fie scrise. Fișierul XML descrie construirea proiectului, dependințele sale asupra altor module și componente, ordinea executării și plug-inuri necesare.

Maven descarcă dinamic librării Java și plung-inuri Maven din unul sau mai multe depozite precum `Maven 2 Central Repository`, și le depozitează într-un cache local. Acest cache local, ce cuprinde artefacte descărcate poate de-asemenea să fie actualizat cu artefacte create de proiecte locale. Depozitele publice pot de-asemenea să fie actualizate.

Proiectul Maven este hostat de Apache Software Foundation.

Spring IoC container

"Inversion of Control", termen cunoscut și ca "dependency injection"[3], este un proces prin care obiectele își definesc propriile dependințe, care reprezintă celelalte obiecte cu care interacționează, doar prin argumente date constructorului, argumente către o metodă de tip "factory", sau proprietăți care sunt setate unei instanțe a unui obiect după ce este construit sau returnat dintr-o metodă de tip "factory". Containerul apoi injectează toate acele dependințe și creează ceea ce se numește un `bean`[4]. Acest proces este fundamental invers, prin urmare denumirea "Inversion of Control".

Pachetele `org.springframework.beans` și `org.springframework.context` stau la baza containerului IoC al frameworkului Spring. Interfața `BeanFactory`

[2]https://ant.apache.org/
[3]https://en.wikipedia.org/wiki/Dependency_injection
[4]https://docs.spring.io/spring-javaconfig/docs/1.0.0.M4/reference/html/ch02s02.html

pune la dispoziție un mecanism avansat de configurație capabil să administreze orice tip de obiect. ApplicationContext este o sub-interfață a BeanFactory. Permite o integrare mai ușoară cu programarea bazată pe aspecte (AOP[5]) din Spring, manipularea mesajelor din resure (pentru internaționalizare) etc.

In Spring, obiectele care ofera stabilitatea aplicatiei si care sunt administrare de containerul IoC Spring sunt numite `beans`. Un `bean` este un obiect care este instaniat, asamblat, si administrat de un container IoC Spring. Altfel, un `bean` este unul dintre multe alte obiecte din aplicatiei. `Beanurile`, si dependintele lor, sunt reflectate in configuration metadata, folosită de container.

Următoarea diagramă este o vizualizare a modului cum lucrează Spring. Clasele aplicației sunt combinate cu metadate de configurare, astfel încât după ce `ApplicationContext` este creat și inițializat, va fi pus la dispoziție un sistem configurat total și executabil.

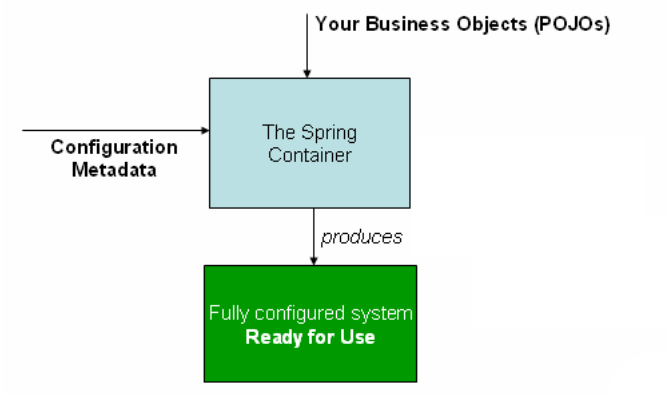


figura 4.2: Diagrama container Spring

Spring Data Neo4J

Pentru a putea beneficia la maxim de maparea obiectelor și de alte tipuri de suport care vin cu Spring Data, Neo4J pune la dispoziție utilizatorilor Spring Data Neo4J. Asocierea dintre Neo4J și Spring Data este strâns legată de frameworkul Spring și oferă Object-Graph Mapping (OGM).

Este bazată pe Neo4J-OGM, un "Java plain Object-Graph-Mapper" și se integrează în infrastructura Spring Data, incluzant Spring Data repository și suport pentru "object-mapping annotation". Spring Data Neo4J este de-asemenea suportat de Spring Boot.

Chiar dacă Spring Data Neo4J exista de mult timp, APIurile Neo4J și modul de a le folosi au evoluat foarte rapid, de la o baza de date integrată Java la o soluție de tip server cu interacțiuni Cypher.

Pentru a folosit Spring Data Neo4J, doar trebuie adăugată dependința de Spring Data Neo4J, și adnotări asupra claselor/câmpurilor (figura 4.3) pentru a beneficia de "object-graph mapping".

[5]https://docs.spring.io/spring/docs/4.3.15.RELEASE/spring-framework-reference/html/aop.html

```java
@NodeEntity
@Getter
@Setter
public class User {

    @GraphId
    public Long id;

    private String firstName;

    private String lastName;

    @Email
    private String email;

    @JsonIgnore
    private String password;

    private Date birthday;

    @Relationship(type ="belongs" , direction = Relationship.OUTGOING)
    private Faculty graduatedFaculty;
```

figura 4.3: Spring Data annotation

De-asemenea, se poate defini un "Spring Data Repository" pentru nivelul de persistență al datelor (figura 4.4).

```java
public interface PostRepository extends GraphRepository<Post> {

}
```

figura 4.4: Exemplu Spring Data Neo4J Repository

Interfața extinde `GraphRepository<T>`, și pune la dispoziție implementări pentru metodele de tip CRUD[6], dar și metode formate după convenții de nume (figura 4.5). Prin aceste convenții, Spring Data Neo4J creează dinamic implementări pentru aceste metode, fără a fi nevoie ca programatorul să ofere el o implementare. De-asemenea, oferă și alte implementări, spre exemplu pentru paginare sau sortare.

```java
public interface PostRepository extends GraphRepository<Post> {

    List<Post> getPostsByUserId(Long userId);

    List<Post> getPostsByPostingDateGreaterThanEqual(Date date);

}
```

figura 4.5: Exemplu de metode generate dinamic

În exemplul de mai sus (figura 4.5), metodele `List<Post> getPostsByUserId(Long userId)` și `List<Post> getPostsByPostingDateGreaterThanEqual(Date`

[6]https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

`date)` sunt scrise doar în PostRepository, și nu prezintă o implementare în altă parte.

Spring Data Neo4J, folosind convenții de nume, reușește să traducă numele metodelor în interogări Cypher oricând sunt apelate.

Având ca exemplu o entitate de tip Person, în figura 4.6 este prezentată diagrama standard de clase pentru interacțiunea cu baza de date Neo4J. PersonRepository este folosit pentru a obține instanțe din baza de date, iar PersonController pentru a le expune către web.

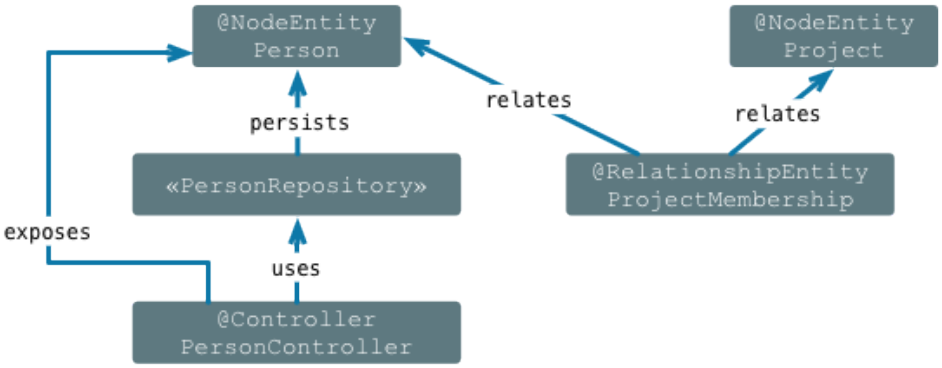


figura 4.6: Diagrama de clase Spring Data Neo4J

Spring RESTful API

`REpresentational State Transfer` (REST) reprezintă un stil arhitectural care specifică constrângeri, precum o interfață uniformă, care aplicată unui serviciu web induce anumite proprietăți, precum performanță sau scalabilitate, ceea ce le permite serviciilor să funcționeze cel mai bine pe web. În athitectura REST, datele și funcționalitățile sunt considerate resurse și sunt accesate folosind `Uniform Resource Identifiers`(URIs), însemnând linkuri tipice din web. Resursele sunt acționate prin utilizarea unui set de operațiuni simple, bine definite. Stilul arhitectural REST constrânge o arhitectură de tip client/server și este proiectat să folosească o comunicare de tip stateless (fără stare), de obicei HTTP. În arhitectura REST, clienții și serverele își schimbă reprezentări de resurse folosind o interfață și un protocol standardizate.

Spring oferă un modul care facilitează crearea de servicii REST, reducând cantitatea de cod scris pentru a expune un astfel de serviciu. Este nevoie doar de adnotarea `@RestController` (figura 4.7) iar Spring, la scanarea proiectului, își va da seama că are de-a face cu un serviciu REST ce trebuie expus.

```
@RestController
@RequestMapping(value = "users")
public class UserController {
```

figura 4.7: Adnotarea RestController aplicată unei clase

Următoarele principii încurajează aplicațiile RESTful să fie simple, rapide și preferabil de folosit:

- Identificarea resurselor prin `URI`: Un serviciu web RESTful expune un set de resurse care identifică obiectivele interacțiunii venite de la clienți. Resursele sunt identificate prin URIs, care pun la dispoziție un spațiu global de adresare pentru resurse și pentru servicii.

- Interfață unfiroma: Resursele sunt manipulate folosind un set de operații: `create`, `read`, `update`, `delete`, `patch` etc. : `PUT`, `GET`, `POST`, `DELETE`, `PATCH`. PUT creează o nouă resursă , care poate fi apoi eliminată folosind DELETE. GET obține starea curentă a resurselor. POST transferă o nouă stare într-o resursă. PATCH se folosește pentru o actualizare parțială a unei resurse.

- Mesaje auto-descriptive: Resursele sunt decuplate de reprezentarea lor astfel încât conținutul lor poate fi accesat într-o varietate de formate, precum HTML, XML, plain text, PDF, JPEG, JSON, și altele. Metadata resurselor este disponibilă și folosită, spre exemplu, pentru controlul cachingului, detectarea erorilor de transmisie, pentru negocierea formatului reprezentării unei resurse, pentru autentificare sau controlul accesului.

- Interacțiuni de tip "statefull" prin hyperlinkuri: Fiecare interacțiune cu o resursă este stateless (fără stare). Interacțiunile statefull sunt bazate pe conceptul de transfer de stare explicit. Există multe tehnici pentru a schimba stări, precum rescrierea unui `URI` sau prin cookieuri. Starea poate fi chiar împachetată într-un mesaj de răspuns.

2.2 Neo4J

Pentru a prezenta schema bazei de date, și cum se modifică aceasta când apare o nouă entitate în sistem, vom folosi fereastra pusă la dispoziție de Neo4J.

Entitățile inițiale din baza de date (Universitatea, ce cuprinde Facultățile prin relația `contains`, domeniile de activitate și ultimele forme de studiu):

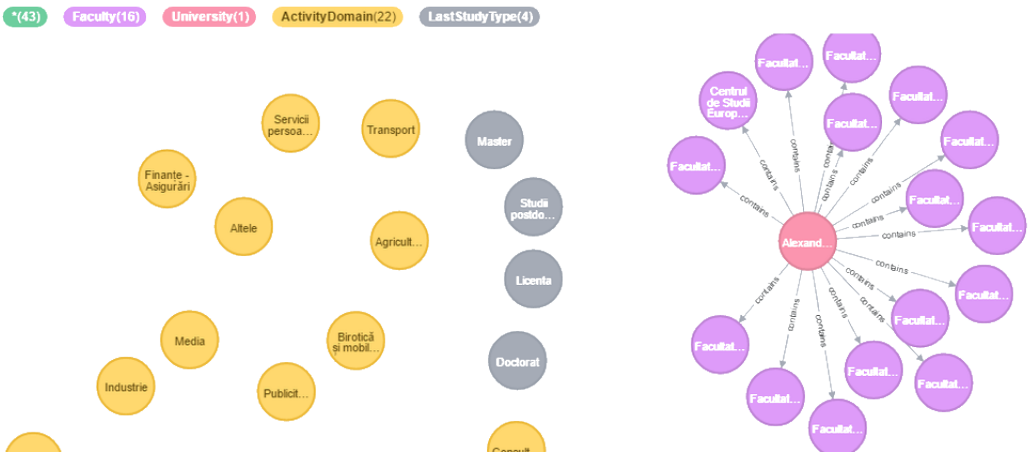


figura 4.8

Când un utilizator își va crea un cont, (spre exemplu: a fost student la Facultatea de Informatică, ultima formă de învățământ este "Licență" și lucrează în IT/Telecomunicații) baza de date se va adapta astfel:

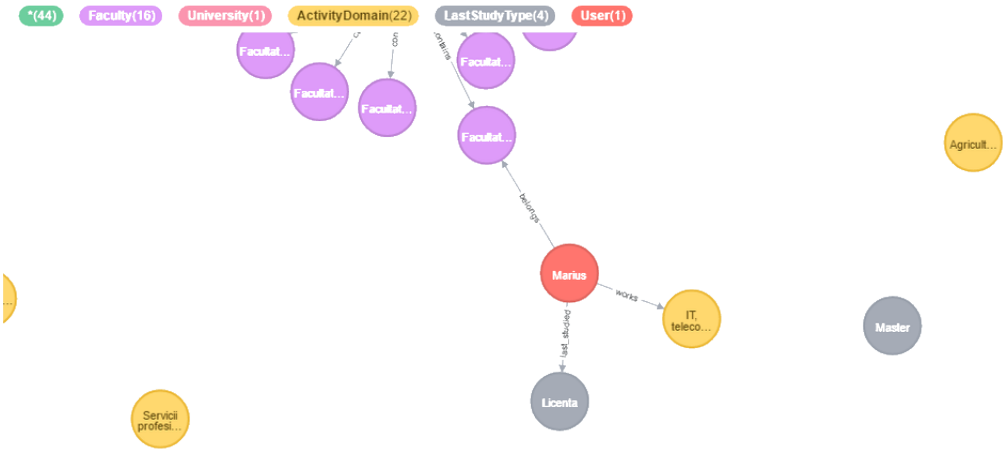


figura 4.9

Pe măsură ce lumea își creează un cont, graful bazei de date devine destul de "aglomerat":

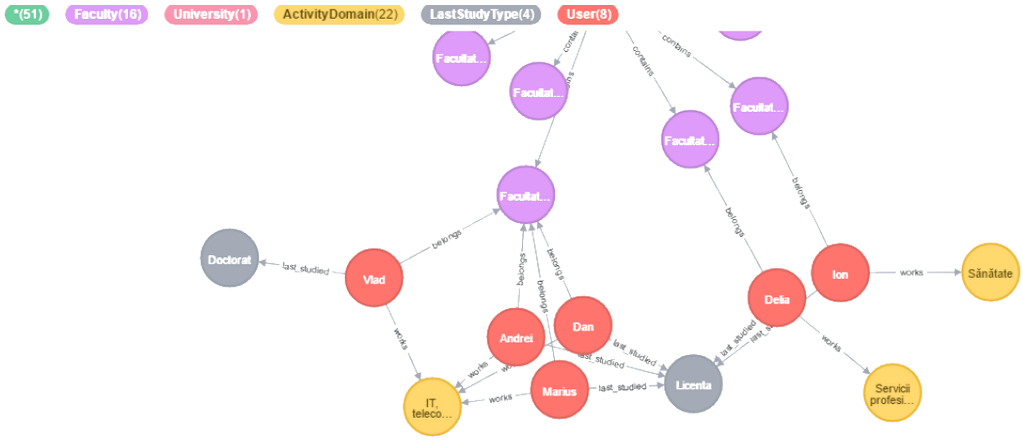


figura 4.10

Iar dacă un utilizator realizează postări, și urmărește și alte persoane pe aplicație, graful devine puțin mai complicat (având in vedere și numărul mic de utilizatori, postări și relații din exemple -figura 4.11-). Aici intervin cu adevărat avantajele bazelor de date de tip graf, unde, în primul rând, schema bazei de date încă a rămas intuitivă și ușor de citit, dar și din punct de vedere al performanței aducerii datelor pentru a le afișa utilizatorului.

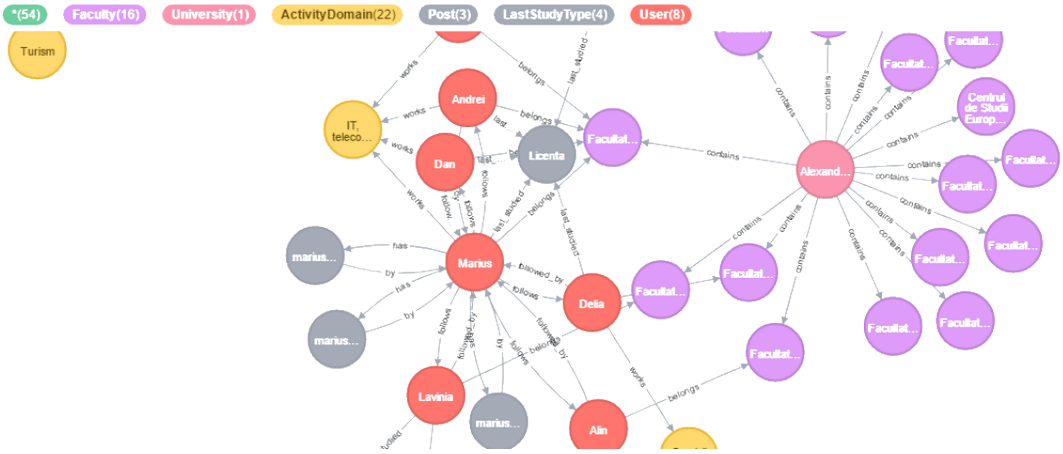


figura 4.11

În timp, nodurile devin din ce în ce mai conectate (figura 4.12) .

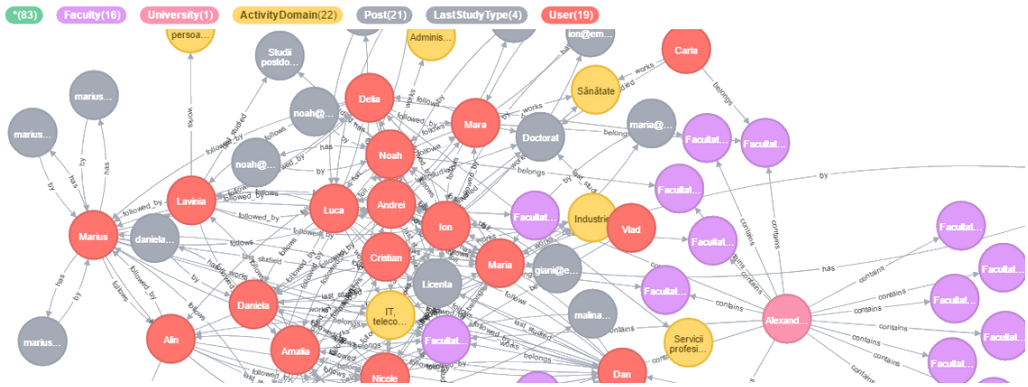


figura 4.12

Avantajele bazelor de date de tip graf apar în acest context mai ales atunci când oferim utilizatorului recomandări de utilizatori pe care i-ar putea cunoaște, după abordarea "pe cine urmăresc cei pe care el îi urmărește", unde de altfel este și problema bazelor de date relaționale din punct de vedere al performanței.

2.3 Angular

Angular este un framework JavaScript ce ajută developerii în construirea aplicațiilor. Librăria pune la dispoziție un număr de funcționalități ce fac trivială implementarea unor cerințe complexe a unor aplicații moderne, precum data binding , rutare sau animații.

Angular de-asemenea pune la dispoziție o serie de convenții pentru cum ar trebui abordată dezvoltarea unei aplicații, ceea ce poate fi extrem de benefic pe termen lung pentru echipele mari care trebuie să lucreze împreună pe o baza de cod unică.

Angular are mai multe versiuni, cea mai nouă fiind versiunea 6. La momentul creerei aplicației, am folosit versiunea 5.

Angular este scris în TypeScript, iar TypeScript este limbajul folosit pentru a scrie cod. TypeScript este un superset al JavaScript, inseamnând că toată sintaxa JavaScript este o sintaxă TypeScript validă. TypeScript a fost creat și este menținut

de Microsoft, și a crescut în popularitate în ultimii ani; prin urmare, nu prezintă un risc includerea TypeScript în dezvoltarea unei aplicații.

Arhitectura Angular

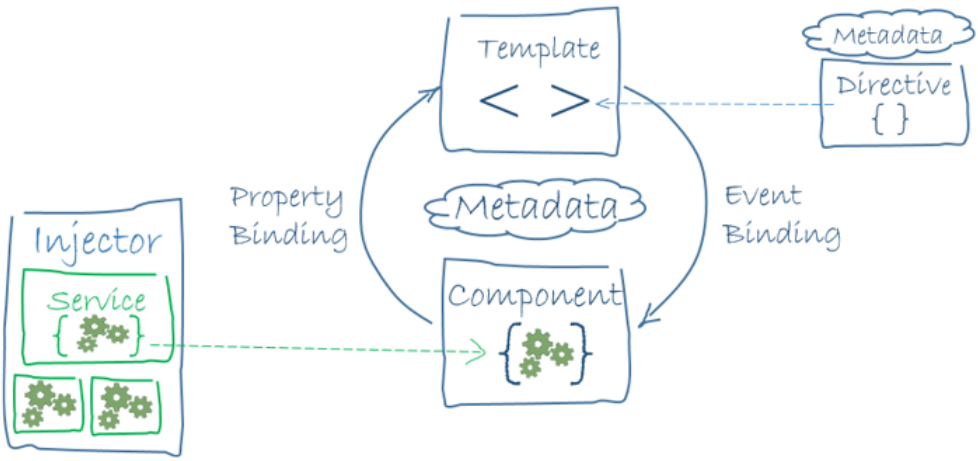


figura 4.8: Arhitectura Angular

Angular propune un model format din mai multe clase de resurse:

- Componente - O componentă controlează o bucată din ecran. Angular crează, actualizează și distruge componente în funcție de interacțiunea utilizatorului.
- Module - O aplicație Angular are cel puțin un modul, anume modulul rădăcină (root). Un modul are responsabilitatea gestiunii conținutului sau.
- Templateuri - Modul cum este vizualizată o componentă HTML este definit cu ajutorul unui template. Templateul este un conținut HTML ce este îmbunătățit folosind proprietățile frameworkului Angular.
- Servicii - Serviciile oferă funcționalități. Acestea pot avea multiple implementări, de la apeluri HTTP la apeluri de APIuri.

Exemplu de comunicare între serviciul Spring Boot și clientul Angular

Spre exemplu, avem următorul endpoint expus în serviciul Spring Boot, ce returnează toate postările ce trebuie să îi apară unui utilizator când accesează tabul Cronologie:

```
@GetMapping(value = "/chronology/{id}")
public @ResponseBody ResponseEntity<List<CronologyPostDTO>> getChronologyPosts(@PathVariable("id") Long userId){
    return new ResponseEntity<>(postService.getCronologyPosts(userId) , HttpStatus.OK);
}
```

În clientul Angular, acest endpoint este apelat în felul următor:

```
getCronologyPosts(userId : number): Observable<any> {
  return this.httpClient.get('http://localhost:8080/posts/chronology/' + userId);
}
```

Ng-Bootstrap

Ng-Bootstrap este un repository open-source care pune la dispoziție o serie de widgeturi native Angular ce sunt construite folosind Bootstrap 4 CSS. Bootstrap este unul dintre cele mai populare frameworkuri CSS folosit pentru a stiliza și a moderniza aplicații, ce folosește HTML, CSS și librării JavaScript. Pentru a folosi frameworkurile Bootstrap în aplicația Angular, cu ceva timp în urmă ar fi trebuit să folosim bootstrap CSS împreună cu fișiere JavaScript, dar "Ng-Bootstrap" oferă o abordare de a folosi acest framework fără a include nici un fișier JavaScript în aplicație.

"Ng-Bootstrap" pune la dispoziție un modul care poate fi încărcat în aplicații Angular și poate fi folosit în aceeași modalitate precum componentele Angular sau componentele personalizate.

Concluzii finale

Scopul aplicației a fost folosirea unei baze de date de tip graf în contextul unei aplicații Web de tip rețea socială, pentru studierea comportamentului acesteia din mai multe perspective: a modelării, a extensibilitatii, a performanței.În final, se poate trage concluzia că, folosirea acestui tip de baze de date a s-a dovedit, conform așteptărilor, o alegere foarte bună, astfel încât acestea au răspuns foarte bine din toate perspectivele enumerate mai sus. Schema bazei de date inițial intuitivă și ușor de modelat, apoi se păstrează ușor citibilă. Pe parcurs, această schemă se adaptează și răspunde foarte bine la interacțiuni. Performanța aducerii datelor din baza de date, în client, prin intermediul serviciului a fost impresionantă, mai ales în situațiile de multiple relații între entități (precum sunt multiplele operații join între tabelele SQL). De asemenea, un alt scop a fost folosirea unor suite de tehnologii cât mai noi și de actualitate în dezvoltarea aplicației iar Angular și Spring Boot se încadrează în această categorie.

Aplicația poate fi extinsă cu o multitudine de elemente (având și avantajul dezvoltării folosind Spring Boot și Angular, aplicația este foarte ușor extensibilă):

- sugestii de utilizatori folosind APIuri puse la dispoziție de rețele sociale precum LinkedIn sau Facebook, spre exemplu recomandarea de persoane care au trecută aceeași facultate în descrierea unui profil de LinkedIn sau Facebook, sau chiar colegi de generație;
- implementarea unei funcționalități ce presupune grupuri de utilizatori;
- implementarea unui sistem de mesagerie ;
- un sistem de evenimente ale universității, unde utilizatorii se pot înscrie și pot primi notificări prin Email sau chiar prin SMS.
- implementarea unui job ce se ocupă de partea de promovare a aplicației, prin găsirea de emailuri unde poate trimite o invitație pe aplicație

Bibliografie

1. Charles Kadushin, *"Understanding Social Networks: Theories, Concepts, and Findings"*
2. Ian Robinson, *Jim Webber and Emil Eifrem - "Graph databases"*
3. Ajitesh Shukla, *"Building Web Apps with Spring and Angular"*
4. Michael Hunger, Oliver Gierke, Vince Bickers, Adam George, Luanne Misquitta, Michal Bachman, Mark Angrish *"Good Relationships"*, *The Spring Data Neo4j Guide Book*.
5. "https://neo4j.com/"
6. "https://angular.io/guide/quickstart"
7. "https://angular.io/guide/architecture-services"
8. "https://ng-bootstrap.github.io/#/home"
9. "https://www.safaribooksonline.com/library/view/graph-databases/9781449356255/ch01.html"
10. "https://neo4j.com/developer/graph-db-vs-rdbms/"
11. "https://pdfs.semanticscholar.org/f511/7084ca43e888fb3e17ab0f0e684cced0f8fd.pdf"
12. "http://www.baeldung.com/spring-data-neo4j-intro"
13. "https://www.todaysoftmag.ro/article/384/neo4j-graph-database-modelarea-datelor-interconectate"
14. "https://core.ac.uk/download/pdf/154676517.pdf"

Ghid de instalare

Pentru a putea rula aplicația, este nevoie de următoarele:

- Serverul de baze de date Neo4J (https://neo4j.com/download/)
- IntelliJ IDEA Ultimate (https://www.jetbrains.com/idea/download/#section=windows)

Pentru a putea folosi angular-cli, ar trebui instalate Node.js (https://nodejs.org/en/) si npm (https://docs.npmjs.com/getting-started/what-is-npm)

angular-cli se poate instala folosind npm, prin scrierea următoarei comenzi în linia de comandă: `npm install -g @angular/cli`

După ce au fost descărcate cele menționate mai sus și proiectul de pe repository (https://github.com/mariusolaru/Licenta), trebuie urmați următorii pași:

- crearea unui server de baze de date în Neo4J, numele și parola trebuind să fie aceleași cu cele definite în fișierul `application.properties` din serviciul Spring Boot, pentru a se putea conecta serviciul la baza de date
- deschiderea proiectului Spring Boot cu IntelliJ IDEA, și importarea dependintelor scrise în fișierul `pom.xml`
- în linia de comandă, trebuie executată comanda `npm install` în directorul Alumni, această comandă știind să rezolve toate dependințele din fișierul JSON asociat proiectului

Apoi, după ce s-a pornit serverul bazei de date din interfața pusă la dispoziție de Neo4J, și apoi serviciul Spring Boot, trebuie executată în linia de comandă, în directorul Alumni, comanda `ng serve --aot`.

Aplicația va porni la adresa `http://localhost:4200`.