UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Thesis Matcher

Aplicație de repartizare a studenților la profesori coordonatori de licență

propusă de

Andrei Dascălu

Sesiunea: februarie, 2023

Coordonator științific

Lect. Dr. Frăsinaru Cristian

UNIVERSITATEA ”ALEXANDRU-IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Thesis Matcher

Andrei Dascălu

Sesiunea: februarie, 2023

Coordonator științific

Lect. Dr. Frăsinaru Cristian

Avizat,
Îndrumător lucrare de licență,
Lect. Dr. Frăsinaru Cristian.
Data: ..... Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Dascălu Andrei** domiciliat în **România, jud. Iași, mun. Iași, strada Sf. Teodor nr. 14**, născut la data de **03 noiembrie 2000**, identificat prin CNP **5001103226752**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2022, 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 **Thesis Matcher** elaborată sub îndrumarea domnului **Lect. Dr. Frăsinaru Cristian**, pe care urmează să o susțin 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 introducerea conținutului ei 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 și 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: ..... Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Thesis Matcher**, 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.

Absolvent **Andrei Dascălu**

Data: ..... Semnătura: .....

Cuprins

Motivație

Lucrarea de licență reprezintă culminarea anilor de facultate, o dovadă a studentului de deprindere a unor noțiuni, de specializare într-un anumit domeniu și de capabilitatea a acestuia de a contribui cu soluții proprii la problemele curente sau viitoare ale societății. În consecință, alegerea unei tematici adecvate pentru lucrarea de licență este un pas de bază datorită unor motive pertinente. În primul rând, abordarea unei tematici cât mai aproape de domeniile sau subiectele de interes ale studentului rezultă într-o atenție mai mare și o implicare corespunzătoare ale acestuia în realizarea în sine a lucrării. În al doilea rând, există o legătură directă între eficiența și temeinicia realizări lucrării de licență și colaborarea dintre profesorul coordonator și student, fiind necesară o comunicare constantă și productivă prind feedback-ul în ambele sensuri, dar și prin resurse, materiale sau idei.

Profesorii coordonatori au un număr de locuri limitat, prin urmare este necesară o repartizare eficientă și mai ales optimă a studenților în funcție de preferințe, luând în același timp în calcul și punctele de vedere ale profesorilor. De asemenea, studenții ar trebui să aibă posibilitatea să cunoască dinainte tematicile propuse de profesori și eventualele condiții prealabile de realizare a respectivelor teme. O aplicație centralizată este așadar întru totul necesară optimizării și îmbunătățirii în general ale acestui proces organizatoric din cadrul facultății.

Introducere

0.0.1 Scopul documentului

Documentul de față are ca scop prezentarea problemei și a unei soluții propuse adecvate acesteia. În acest fel, se urmărește descompunerea problemei în subprobleme și detalierea acestora, ilustrarea unor soluții deja existente, analizarea aplicației web propuse, în mod sistematic, evidențiind atât detalii de abordare, arhitectură, tehnologii utilizate, implementare.

0.0.2 Scopul aplicației

Aplicația *Thesis Matcher* este o soluție web ce urmărește să rezolve problema alocării studenților din anii terminali la profesorii coordonatori.

Unul dintre principalele obiective este centralizarea întregului proces, de la repartizare până la alte colaborare și alte aspecte organizatorice, asigurând astfel o desfășurare bună și mai sigură. Fiind un proces anual, cu un număr semnificativ de părți implicate, este inevitabilă necesitatea de a automatiza într-o anumită măsură desfășurarea acestuia.

Un obiectiv de asemenea important este simplificarea obținerii informaților de interes, atât de către studenți, cât și de profesori. Studenții au posibilitatea să afle tematicile propuse de fiecare coordonator, eventual să propună o idee proprie. Participanții pot urmări în timp real situația locurilor disponibile, precum și alte statistici.

Cel de al treilea obiectiv este optimizarea repartizării prin implementarea ierarhizării de către student a opțiunilor alese sub formă de preferințe.

0.0.3 Modul de implementare

Soluția se prezintă sub formă de aplicație web ce se împarte în două ”sub-aplicații”.

Partea de *Front-End* este implementată în Angular. Partea de *Back-End* este implementată în Java Spring Boot. Această parte cuprinde atât comunicarea cu Front-End-ul prin intermediul request-urilor, dar și procesare sub forma implementării unui algoritm de *deouble-push* pentru rezolvarea unei probleme de *asymmetric assignment*. Acest lucru presupune o optimizare a soluției deoarece permite părților să efectueze alegeri cu același grad de importanță. Mai precis, în contextul de față, un student poate realiza o ordine a preferințelor profesorilor, în unele cazuri cu o anumită indiferență în ideea că studentul poate prefera atât o opțiune, cât și alta cu același grad.

Capitolul 1

Descrierea problemei

Lucrarea de față propune să rezolve o instanță a unei probleme des întâlnite în cotidian și descrise în literatra de specialitate, problema *Asignarii optime*. În cazul de față, există două mulțimi participante. În primiul rând profesorii, mai precis propunerile individuale ale acestora pentru proiectul final de licență, și studenții. Fiecare student stabilește o ierarhie a preferințelor (unde sunt ordonate propunerile alese de ei; precizări și detalieri ale acestora urmează a fi făcute).

De asemenea, profesorii au opțiunea de a încheia un acord cu un student sau mai mulți pentru realizarea unui anumit proiect fără a participa la etapa de stabilire a unei repartiții pe cele două mulțimi.

Într-un final, după efectuarea algoritmului, fiecărui student îi este atribuit în mod optim, cu alte cuvinte cu o satisfacere cât mai mare a preferințelor, un proiect pentru lucrarea sa de licență.

1.1 Scopul aplicației

În consecință, aplicația propusă are atât scopul de a eficientiza propcesul de repartizare a studenților la profesorii de licență (respectiv tezele propuse de aceștia), un proces complex și periodic specific oricărei facultăți, dar și scopul de a optimiza această repartiție, în ideea de a distribui proiectele studenților după abilitățile și preferințele lor, încurajând astfel o dedicare și elaborare cât mai adecvate din partea studenților în realizarea lucrării finale.

1.2 Scopul documentului

Documentul prezent descrie o soluție a problemei evidențiate anterior sub forma unei aplicații web ce utilizează o formă a algoritmului de *licitație* (*Auction algorithm*) adaptat contextului, ce utilizează o strategie *double-push* pentru pasul de licitare. Sunt ilustrate părțile componente ale aplicației și justificată alegerea anumitor implementări și tehnologii. De asemenea, acest document urmărește și familiarizarea eventualilor utilizatori cu aplicația.

1.3 Structura documentului

Pentru început, a fost realizată o descriere generală a aplicației subliniind câteva particularități în al doilea capitol, **Arhitectura aplicației. Detalii**. Capitolul ulterior, **Teorii și tehnologii utilizate**, explică alegerea anumitor framework-uri precum și avantajele acestora, plus necesitatea unui algoritm de rezolvare a problemei de *Stable matching* (un scurt istoric a fost realizate pentru familiarizarea cu aceste tehnologii și teorii). Urmează patru capitole care surprind în detaliu părțile componente ale aplicației, **Baza de date**, **Front-end-ul**, **Back-end-ul**, **Algoritmului de repartizare**. În final, capitolul **Indicații de utilizare** prezintă sugestii în legătură cu navigarea și folosirea aplicației, nu fără a sublinia în cele din urmă câteva **Concluzii** și posibile idei de dezvoltare și îmbunătățire a aplicației în viitor.

1.4 Contribuții

Aplicația **Thesis Matcher** este compusă după cum a fost precizat din două părți principale.

Front-end-ul realizat în Angular a fost adaptat de către mine pentru o cât mai facilă utilizare atât de către profesori, cât și de către studenți, urmărindu-se un aspect cât de cât minimalist și clar.

Back-end-ul în Spring Boot conține o parte de autentificare și autorizare simplă, dar eficientă și suficientă prin intermediul JWT (JSON Web Token).

O contribuție majoră o reprezintă însă algoritmul de determinare a soluției, componentă tot a back-end-ului. În cazul de față, a fost utilizat un algoritm de licitație (*Auction algorithm*) pentru o problemă de asignare asimetrică în ideea că cele două mulțimi

participante cel mai probabil nu sunt de aceeași dimensiune.

Capitolul 2

Arhitectura aplicației. Detalii

În primul rând, **Thesis Matcher** este o aplicație de tipul utilitar, scopul acesteia este unul de uz intern, în cadrul facultății. Prin urmare, traficul efectuat de către utilizatori este unul scăzut, lucru ce permite o arhitectură monolitică.

În al doilea rând, arhitectura urmează un model *MVC* (Model View Controller) caracteristic unei aplicații web de acest tip, unde clientul trimite anumite request-uri prin interacțiunea cu GUI-ul, iar serverul la rândul său transmite anumite informații. Acest model arhitectural permite o organizare și modularizare riguroasă a aplicației.

În al treilea rând, baza de date este una de tipul SQL deoarece este necesară stocarea unui număr relativ mare de relații între entități, reprezentate de către preferințele studenților. De aceea, a fost ales MySQL pentru modelarea relațiilor și datelor din aplicație.

Astfel, aplicația este pe trei niveluri (*3-tier architecture*). Un nivel al clientului, *Presentation Layer* ce facilitează utilizarea, un nivel de business, *Application Layer*, ce tratează request-uri, efectuează operații de autentificare și autorizare, precum și algoritmice, iar în final un nivel de prelucrare al datelor, *Data Layer*.

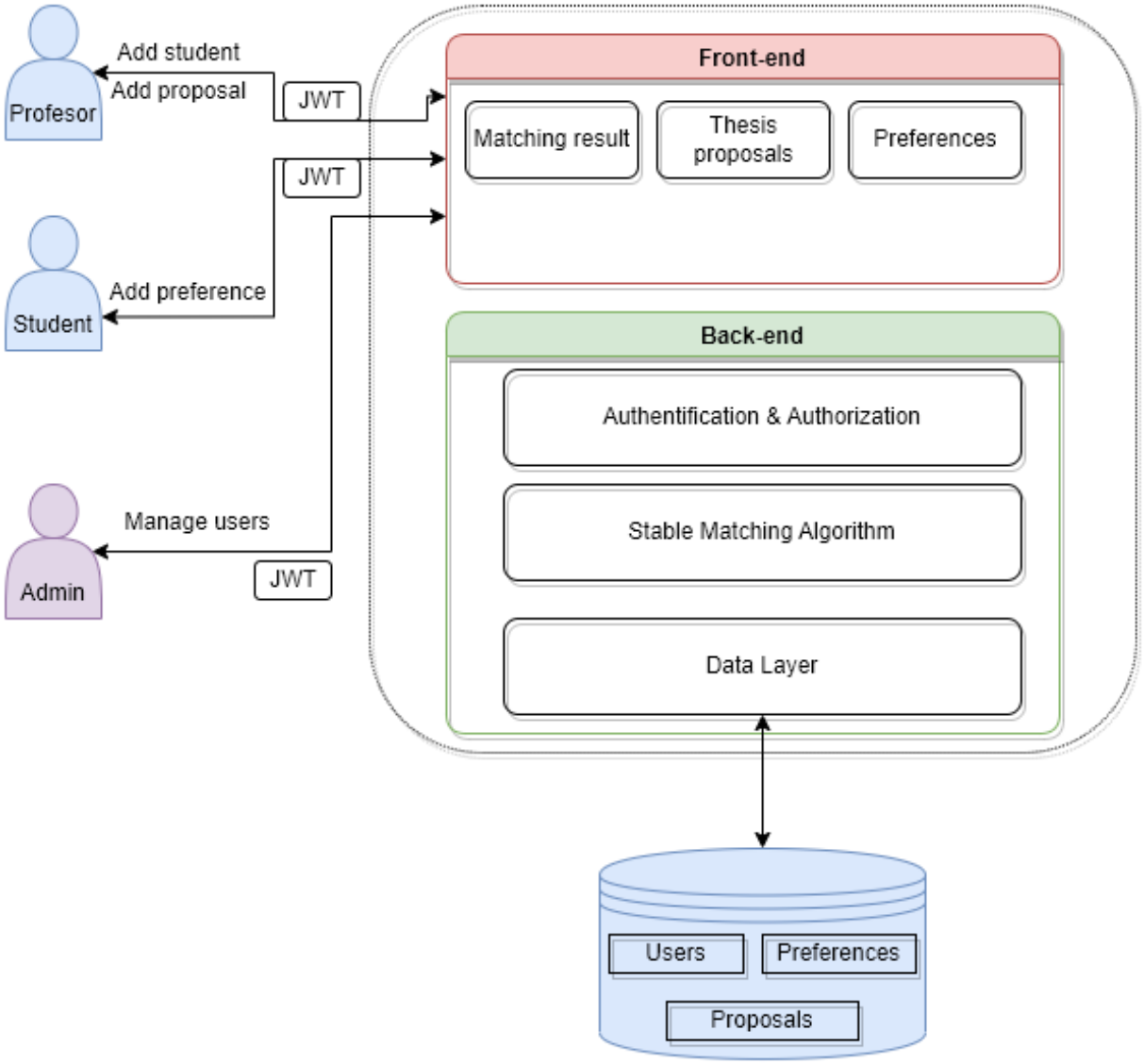

Figura 2.1: Arhitectura aplicației

Capitolul 3

Teorii și tehnologii utilizate

În acest capitol sunt prezentate în primul rând tehnologiile utilizate în implementarea aplicației. În cea de a doua parte este evidențiat în mare parte particularități ale algoritmicii utilizate în determinarea unei alocări optime specifice problemei, în alte cuvinte algoritmul de *stable matching* (*stable marriage*).

3.1 Angular

Angular este un framework de JavaScript scris în TypeScript și menținut de Google. Framework-ul a fost dezvoltat în principal pentru crearea aplicațiilor web *single-page*, într-o manieră ce ușurează mentenanța și dezvoltarea ulterioară.

3.1.1 Scurt istoric

În 2010, Miško Hevery, un angajat la Google la acel timp, a lansat un proiect cu numele *AngularJS* care a fost apreciat în mare măsură de comunitate. Între anii 2014-2015 a avut loc o reîmprospătare majoră a framework-ului însemnând de fapt o rescriere majoră a acestuia. Noua versiune avea să fie numită simplu Angular. Au urmat câțiva ani de tranziție deoarece multe proiecte deja în producție erau utilizau AngularJS și trebuiau refactorizate. În acest moment, Angular este cel mai folosit framework

de front-end, în specialde dezvoltatorii de la Google și de către start-up-uri. Exemple de companii recunoscute ce folosesc Angular sunt Microsoft, Gmail, PayPal, Forbes.

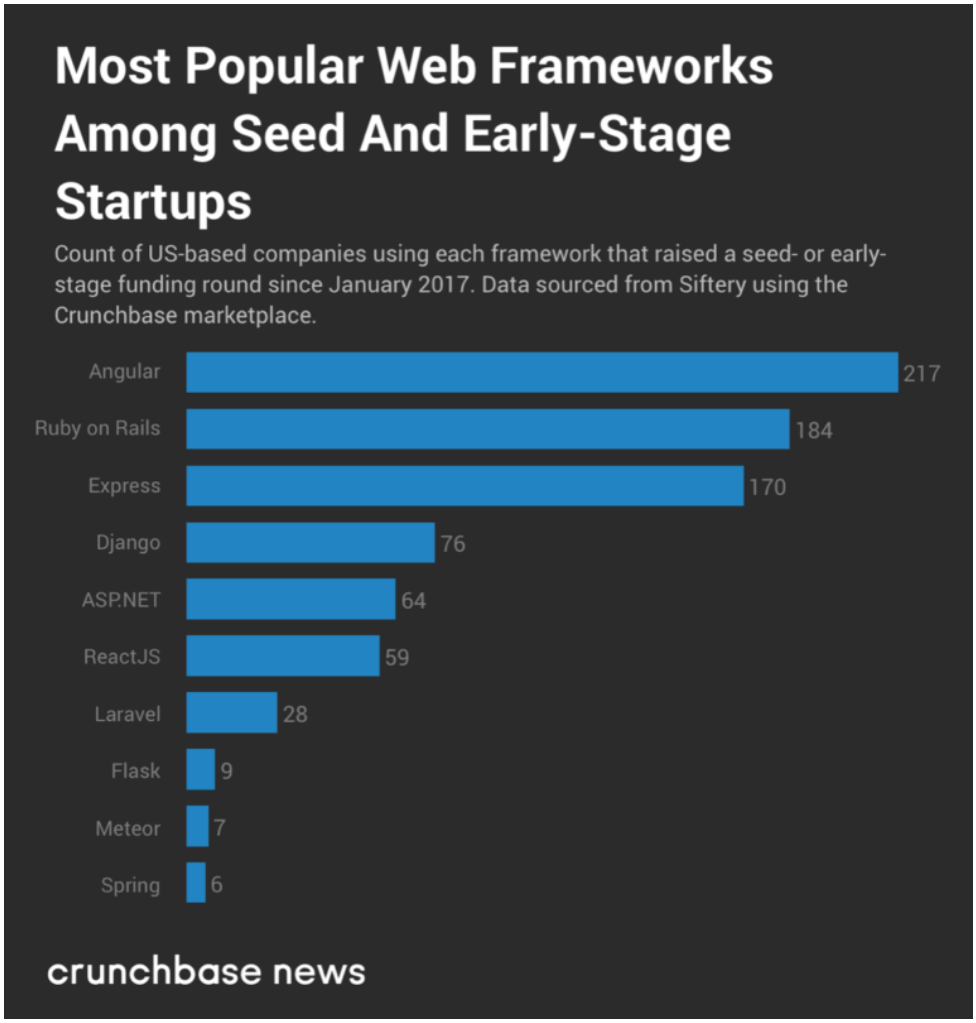

Figura 3.1: [2]

3.1.2 Particularități

Motivul alegerii Angular pe partea de front-end este reprezentat de o serie de caracteristici.

Un astfel de motiv este *arhitectura bazată pe componenta*, fiind astfel o formă de programare orientată pe obiect. Utilizatorul crează în mod uzual clase corespunzătoare componentelor ce conțin și un șablon HTML (*eng.* HTML template). Pentru simplificare, Angular oferă și opțiunea de ”injectare” a serviciilor *custom* sau *in-built* într-o componentă ce utilizează aceste funcționalități. În acest fel, utilizatorul poate reutiliza, înlocui, modifica componente în diverse locuri, obținâându-se astfel un UI (User Interface) modularizat.

Un alt motiv este modul de încărcare a paginii web. Angular folosește *lazy loading* ce permite încărcarea instantanee a website-urilor, prin afișarea doar a componentelor cerute și necesare utilizatorului, în timp ce celelalte sunt pregătite în fundal pentru alte eventualități.

Dependency injection reprezintă un al treilea motiv, un design pattern ce permite împărțirea lucrului între diferite servicii, distribuind în mod eficient sarcinile. Prin inițializarea dependențelor, Angular reușește să reducă în mod considerabil codul de tip *boilerplate* (fragmente similare de cod des utilizat între care există mici diferențe) și să extindă mai ușor o astfel de aplicație.

Framework-ul are trei tipuri de *dependency injections*:

1. Constructor injection
2. Setter injection
3. Interface injection

Din punct de vedere al arhitecturii, Angular este în totalitate un framework MVC (model-view-controller), după cum se observă în figura următoare.

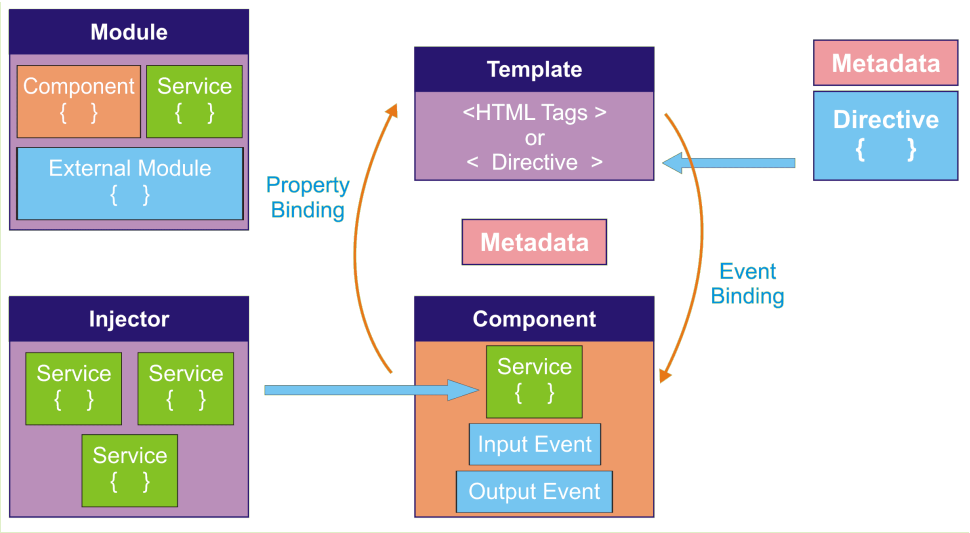

Figura 3.2: `https://www.ngdevelop.tech/wp-content/uploads/2017/12/Angular_Architecture.png`

3.2 Spring Boot

Spring Boot este un micro-framework open-source folosit pentru a crea aplicații Spring cu microservicii. Spre deosebire de alte framework-uri de Java, acesta oferă configurări XML flexibile, procesare în loturi puternică, tranzacții cu baza de date și o varietate de instrumente de dezvoltare.

3.2.1 Scurt istoric

Framework-ul *Spring* a fost creat în 2004 pentru a simplifica dezvoltarea programelor pe partea de server. În aprilie 2014 a fost lansat Spring boot 1.0.0 în urma unor cereri din partea programatarilor de a configura serviciile de web container într-un container spring din metoda principală. În decembrie 2016 a fost lansat Spring Boot 1.3 ulterior trecerii framework-ului Spring de la versiunea 4.1 la 4.2 și includea sprinjin pentru fisiere JAR complet executabile, noi utilitare spring-boot-dev și auto-configurare pentru tehnologii de caching.

3.2.2 Motivație

Spring Boot este bazat pe Java, unul dintre cele mai populare limbaje de programare. Framework-ul are o comunitate vastă de utilizatori cu diverse materiale și cursuri.

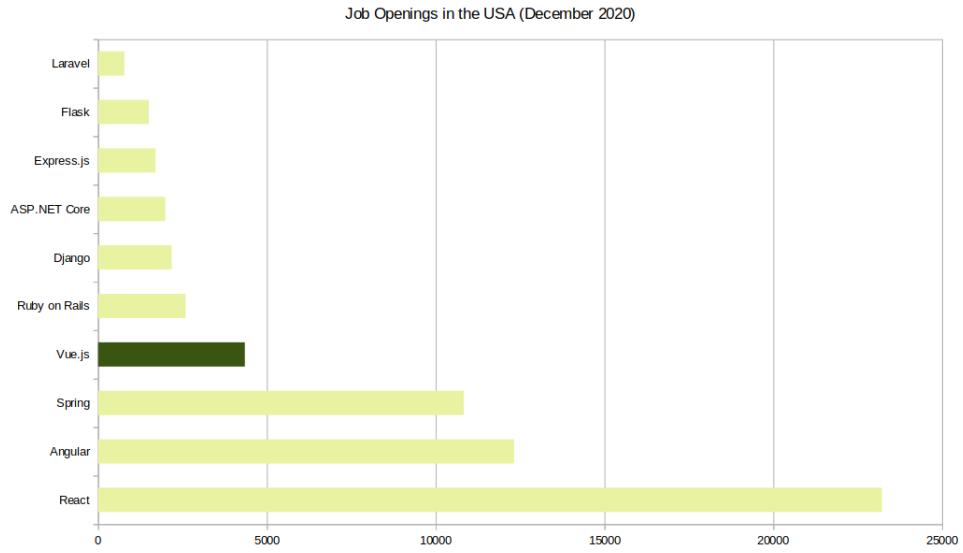

Figura 3.3: `https://miro.medium.com/max/1400/1*Y6A_rS5IdRDUG4KRIFP_fA.png`

Avantajele prinicipale sunt după cum urmează. Spring Boot este *multi-threaded*, util pentru operații repetitive și de durată. Facilitează crearea și testarea aplicațiilor Java oferind un setup default pentru unit și integration testing. Există de asemenea posibilitatea de a integra Spring Boot cu ecosistemul Spring ce include Spring Data, Spring Security, Spring ORM și Spring JDBC într-un mod simplificat.

3.2.3 Particularități

Principala particularitate a Spring Boot o reprezintă adnotările (*Spring Boot annotations*) utilizate în auto configurare. De exemplu, **@SpringBootApplication** marchează metoda principală a aplicației și este obligatorie. **@EnableAutoConfiguration** oferă oricărei clase pe care o marchează cu opțiunea de Automatic Configuration. **@ComponentScan** scanează la inițializare toate declarările de *beans* și pachete.

Un alt detaliu este utilizarea *Spring Starter Dependencies* ce facilitează gestionarea dependențelor unei aplicații în continuă dezvoltare.

Spring Boot Actuator oferă utilitare de producție aplicației. Actuator este utilizat în mare parte pentru a obține informații de funcționare despre o aplicație în rulare (metrice, info, dump, env, etc.), cu ajutorul HTTp endpoints și JMX beans. Ultima versiune, Spring Boot 2.x Actuator, suportă și modele CRUD.

3.2.4 Detalii

Versiunea de Java este 11.

Versiunea de Spring Boot este 2.7.6.

3.3 MySQL

3.3.1 Baza de date

3.4 Algoritmica

3.4.1 Problema stable matching (SMP)

Problema constă în găsirea unei potriviri stabile (stable match) între două mulțimi de aceași dimensiune de elemente, luând în considerare o ierarhie a preferințelor. Astfel, o soluție este o bijecție între elementele primei mulțimi și cele din cea de a doua. O potrivire nu este stabilă dacă:

1. Există un element α în mulțimea întâi care preferă un anumit element β din cea de a doua mulțime în detrimentul unui element din a doua mulțime deja cuplat.

2. β preferă α în detrimentul elementului cu care β deja este cuplat.

Cea mai întâlnită formă a problemei este *stable marriage problem*. Fie *n* bărbați și *n* femei, fiecare persoană având o ierarhizare a tuturor membrilor de gen opus în funcție de preferințe.

Prima soluție a fost prezentată în 1962 de către David Gale și Lloyd Shapley care au demonstrat că pentru orice număr de participanți, există o soluție pentru a face toate cuplurile stabile.

3.4.2 Algoritmul Gale-Shapley

Algoritmul implică un număr de iterații. În prima iterație, fiecare element din prima mulțime A încearcă cuplarea cu prima sa obțiune din mulțimea B. Elementele din B sunt cuplate inițial cu elementul pretendent pe care îl preferă cel mai mult.

În iterațiile ulterioare, fiecare element din A necuplat încă încearcă să se cupleze cu următoarea opțiune din preferințele (necuplată sau nu). Elementele din B

pot schimba partenerul curent cu cel propus în cazul în care este deja cuplată dacă îl preferă pe cel din urmă.

Procesul se repetă până când există o bijecție între cele două mulțimi.

3.4.3 Instanța problemei prezente

În cazul problemei distribuirii studenților la profesorii coordonatori de licență, fiecare student primind implicit o temă unică de licență, există atât asemănări cu *SMP* (Stable Marriage Problem), cât și o serie de deosebiri particulare care necesită o adaptare a algoritmului de rezolvare.

Asemănări

Există două mulțimi ce trebuie cuplate într-un mod stabil. Cuplajul este realizat ținând cont și în această instanță de o serie de preferințe. Ambele tipuri de participanți au o ierarhizare a preferințelor sale. Din punct de vedere al temelor de licență, cel mai probabil există o relație de bijecție între studenți și tematicile primite sau propuse și acceptate.

Deosebiri

O primă deosebire observată este că mulțimile nu au un număr egal de elemente. În mod natural, mulțimea studenților este semnificativ mai mare decât cea a profesorilor.

În al doilea rând, relația dintre cele două mulțimi nu este de bijecție, ci mai degrabă una injectivă. Fiecare student trebuie să aibă la finalul procesului de repartizare atribuit un profesor coordonator de licență. Cu alte cuvinte,

$$\forall\, stud,\ unde\ stud \in Studs,\ \exists\, prof \in Profs,$$

$$BTD(stud) = prof$$

unde *BTD* = Bachelor Thesis Distributor, funcție ce repartizează fiecare student unui profesor, *Studs* = mulțimea studenților, *Profs* = mulțimea profesorilor.

În consecință, este inevitabil ca un profesor să aibă sub coordonare mai mulți studenți, o altă particularitate fiind însă că un profesor are o limită prestabilite de astfel de studenți.

La finalul procesului de repartizare, studenții care nu au o preferință sunt repartizați aleatoriu sau după un anumit criteriu arbitrar profesorilor care mai au locuri libere.

3.5 Resurse externe

Datele necesare dezvoltării și testării aplicației printre care se numără studenții, profesorii, preferințele, propunerile (tematici generale sau proiecte particularizate) au fost generate cu ajutorul API-ului *Mockaroo*.

Capitolul 4

Baza de date

În acest capitol este argumentată alegerea unui tip de baze de date SQL, mai exact MySQL, pentru a reține informații importante aplicației. A fost preferată o abordare clasică asupra unei baze de date relaționale în special datorită numărului mare de tranzacții ce necesită o asigurare mai mare a integrității datelor [5].

Aplicația Thesis Matchereste de tip utilitar, aceasta are scopul de a fi folosită într-un context restrâns, în cadrul facultății. Prin urmare numărul de utilizatori este unul relativ mic, reprezentat de către studenți și profesori. De asemenea, poate fi observată o relație strânsă între profesori și propunerile acestora, dar și între studenți și preferințele acestora.

Relațiile între entități sunt de mai multe tipuri, în special *One-to-one* și *One-to-many*. Acest lucru se poate observa în diagrama următoare a bazei de date utilizate.

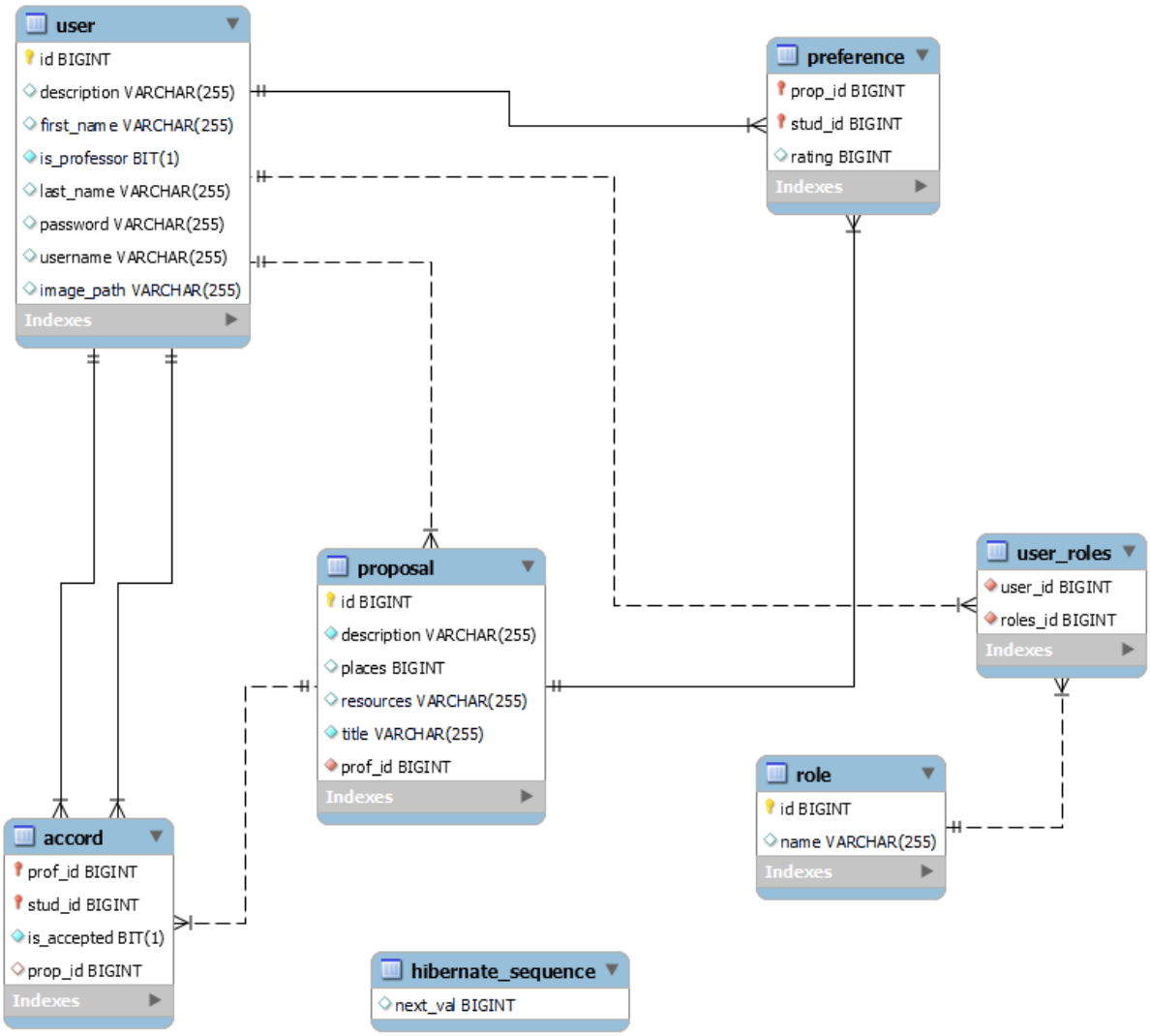

Figura 4.1: Baza de date

În următoarele subcapitole sunt detaliate modelele utilizate pentru a reprezenta aceste date.

4.1 Modelarea utilizatorilor

Utilizatorii au fost modelați într-un mod clasic. Aceștia au la rândul lor roluri în funcție de drepturile pe care le au ei în cadrul aplicației. Astfel, sunt identificate două astfel de roluri: **ROLE_USER** și **ROLE_ADMIN**. Pe lângă acest lucru, utilizatorii sunt ori profesori, ori studenți, particularitate ilustrată de câmpul boolean *is_professor* al entității **`user`**.

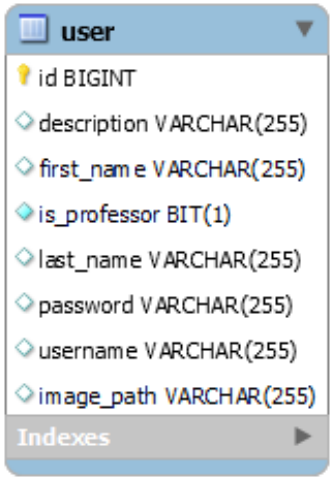

Figura 4.2: User

Fiecare utilizator este identificat unic printr-un `id` de tipul de date BIGINT, care este și cheie primară a entității. Ca în majoritatea cazurilor, aceștia au un nume de familie (`last_name`) și un prenume (`first_name`). Câmpul `username` este de tipul VARCHAR(255) și este reprezentat de un email valid, lucru asigurată pe partea de front-end. Câmpul `password` este tot de tipul VARCHAR(255) și reprezintă parola utilizatorului, însă criptată pe partea de back-end cu un algoritm clasic. Aceste două câmpuri sunt primite de către utilizator din partea unui administrator și sunt utilizate pentru autentificarea în aplicație. A fost preferată această abordare pentru a împiedica crearea de conturi din partea unor persoane din afara contextului.

4.1.1 ROLE_USER

Cei mai mulți utilizatori au rolul **ROLE_USER**, un rol default. Acest lucru le autorizează accesul la aplicație, odată ce aceștia sunt autentificați. Interfața diferă în funcție de tipul utilizatorului.

Tipuri

Profesorii pot crea și actualiza propuneri (en. proposals) pentru teza de licență. Aceste propuneri pot fi *project* sau *topic*, detalierea acestora urmând a fi făcută ulterior. De asemenea, aceștia pot încheia acorduri (en. accords) cu anumiți studenți pentru unele dintre propunerile lor. Acest lucru le permite studenților să nu mai participe la algoritmul de stable matching, fiind deja repartizați profesorilor respectivi.

Studenții pot crea și actualiza preferințe (en. preferences) într-o anumită ierarhie a lor. Așadar, categorisirea utilizatorilor în profesori și studenți are unicul scop de a diferenția interfața în funcție de funcționalitățile specifice tipurilor acestora.

4.1.2 ROLE_ADMIN

Există în plus un număr restrâns de utilizatori care au rolul de administrator, **ROLE_ADMIN**. Aceștia au dreptul de a gestiona conturile participanților, mai precis de a crea, actualiza și elimina utilizatorilor. De asemenea, administratorii au opțiunea de a impune anumite limite în cadrul aplicației cum ar fi o limită a propunerilor pentru profesori și o limită a preferințelor pentru elevi.

4.2 Preferințele

Preferințele sunt modelate simplu, reprezentând în sine o relație între entitățile studenți și propuneri.

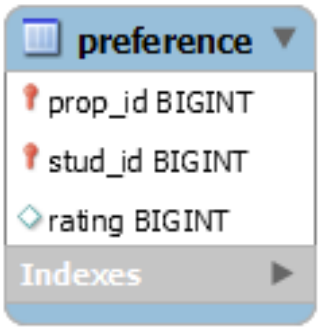

Figura 4.3: Preference

După cum se poate observa în tabela **`preference`**, acestea cuprind un `id_stud` ce face legătura cu tabela **`student`** și un `id_prop` ce face legătura cu tabela **`proposals`**.

În plus, există un câmp `rating` care este de tipul BIGINT și indică invers proporțional importanța preferinței, mai exact o preferință cu un `level` $= 0$ este pe primul loc în ierarhie. Este de asemenea important de precizat că mai multe preferințe ale unui student pot avea același rating, lucru necesar pentru un algoritm de *stable matching with ties*.

Aceste relații între studenți și tezele propuse vor constitui datele de intrare pentru algoritm, fiind astfel de o importanță semnificativă în cadrul aplicației.

4.3 Propunerile

În cazul propunerilor, la nivel teoretic acestea pot fi de două tipuri, proiect (en. project) sau temă (en. topic). Diferența constă în faptul că un proiect este o propunere mai concisă, cu o descriere exactă a lucrării așteptate, fie ea una de cercetare sau o aplicație propriu-zisă, iar un proiect nu poate fi ales și realizat decât de către o singură persoană. Un astfel de exemplu ar putea fi *„Realizare unei aplicații de reconoaștere facială utilizând metode de machine learning"*, cu precizările și restricțiile aferente.

Pe de altă parte, o temă este o descriere pe larg a anumitor particularități ale unui proiect de cele mai multe ori dintr-un anumit domeniu de specialitate. La fel ca un proiect specific, o temă poate cuprinde atât lucrări general teoretice, cât și practice, studentul fiind nevoit să stabilească împreună cu profesorul său coordonator o teză concretă. Însă o propunere de acest fel poate avea un număr de locuri disponibile mai mare de 1, deoarece mai mulți studenți pot urma direcții diferite asupra tematicii propuse. Spre exemplu, profesorul poate propune o temă de „Algoritmi genetici și optimizarea acestora" care poate fi repartizată unor 3 studenți care stabilesc în mod distinct cu profesorul de licență lucrările lor.

Cu toate acestea, din motive de simplificare și optimizare, la nivelul de scehmei de baze de date, a preferată modelarea propunerilor într-o singură tabelă intitulată sugestiv **`proposal`**, diferența dintre cele două tipuri fiind făcută de câmpul `places`.

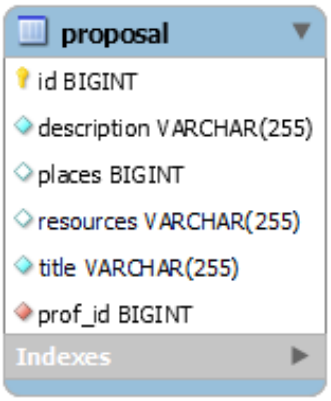

Figura 4.4: Proposal

În cazul proiectelor (projects) câmpul acesta rămâne `null`, spre deosebire de teme (topics). Fiecare propunere are un titlu (`title`) reprezentativ și o descriere (`description`) adecvată de tipul VARCHAR(255), precum și o listă de referințe pentru a ajuta studentul. Ultimul câmp este `id_prof` care indică autorul propunerii.

4.4 Acordurile

În cadrul aplicației Thesis Matcher, fiecare profesor poate stabili împreună cu un student un acord, reprezentat de tabela cu același nume **`accord`**, în ideea că acest student și propunerea aleasă nu mai sunt luate în calcul la rularea algoritmului de repartizare. Cu alte cuvinte, acest detaliu le permite studenților posibilitatea să realizeze un proiect preferat, adecvat abilităților și înclinațiilor sale, desigur cu aprobarea profesorului corespunzător tezei alese.

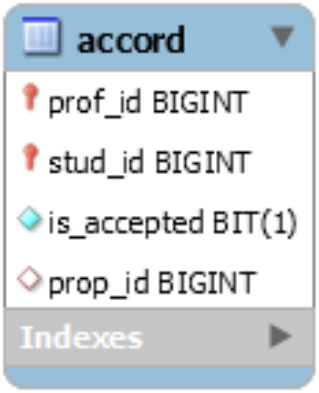

Figura 4.5: Accord

4.5 Lucrul pe partea de back-end cu baza de date

Partea de back-end a aplicației, fiind implementată cu ajutorul Spring Boot, interacționează cu baza de date prin intermediu Hibernate, alegere făcută luând în considerare o serie de avantaje.

4.5.1 Hibernate

Figura 4.6: `https://hibernate.org/images/hibernate-logo.svg`

Ce este Hibernate

Hibernate este un framework, mai precis un ORM (object relational mapping) creat pentru a facilita maparea modelelor orientate-obiect la baze de date relaționale pentru aplicații web [4]. În mod intern, acest ORM utilizează JDBC API pentru a interacționa cu baza de date [3].

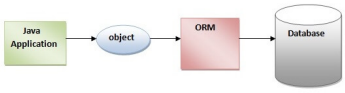

Figura 4.7: ORM

Hibernate este cu alte cuvinte o implementare a specificațiilor JPA (Java Persistence API) pentru persistența datelor. JPA este un set de reguli care oferă un standard definit și o anumită funcționalitate acestor tehnologii ORM [3].

Funcționalitatea Hibernate

Framework-ul precizat corelează clase din limbajul Java tabelelor din serverul de baze de date, dar și tipurile de date din Java cu cele din SQL pentru a asigura persistența obiectelor. Hibernate returnează astfel aplicației enitățile sub formă de obiecte Java și reduce astfel timpul de procesare a rezultatelor și codul necesar implementat de programator.

Avantaje ale Hibernate

Hibernate este în primul rând un framework *open source* și *lightweight*, lucru ce mi-a permis familiarizare cu acesta și ușoara configurare a lui.

În al doilea rând, Hibernate este un framework relativ rapid deoarece beneficiază de un cache intern, structurat pe două niveluri, primul nivel fiind folosit în mod normal [3].

De asemenea, tehnologia optată utilizează o versiune orientată obiect a SQL-ului, HQL (Hibernate Query Language), care permite generează query-uri independente de baza de date, fără a scrie în mod explicit query-uri în SQL [3].

În final, Hibernate oferă posibilitatea de a crea și popula tabelele din baza de date în mod automat, direct din cod.

Imaginea următoare ilustrează specificațiile din fișierul `application.properties` specific oricărui proiect Maven, în care sunt este configurată utilizarea framework-ului Hibernate. Acest lucru include date de conectare la baza de date, modul de conectare, dialectul SQL folosit etc.

```
spring.datasource.url=jdbc:mysql://localhost:3306/thesis_matcher?useSSL=false&serverTimezone=UTC
spring.datasource.username=[redacted]
spring.datasource.password=[redacted]
spring.datasource.platform=mysql
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.jpa.properties.hibernate.format_sql=true
```

Figura 4.8: application.properties

Pentru a putea folosi Hibernate într-un proiect de Spring Boot este necesară adăugarea dependenței următoare în fișierul `pom.xml`.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Figura 4.9: Dependența din pom.xml

4.6 Generarea datelor

În vederea generării unor date reprezentative problemei actuale, a fost creată o clasă separată numită `Populator` ce introduce înregistrări (records) în baza de date. De aceea au fost stabilite în primul rând o serie de parametri care reglează dimensiunile populațiilor și proporțiile dintre diferitele tipuri de entități. În continuare vor fi prezentați pe scurt acești parametri și configurarea lor, urmând a descrie maniera de generare propriu-zisă a utilizatorilor, propunerilor profesorilor și nu în ultimul rând, preferințelor studenților.

Configurarea parametrilor necesari generării datelor este realizată prin intermediul unei clase `PopulatingConfiguration`.

```
@Data
@RequiredArgsConstructor
public class PopulatingConfiguration {
    private final Integer userCount;
    private final Integer adminCount;
    private final Integer maxTopicPlaces;
    private final Integer topicProb;
    private final Integer professorProb;
    private final Integer preferenceCount;
    private final Double propsToStudsRatio;
}
```

Figura 4.10: Clasa de parametri de configurare

Aceasta conține parametri precum numărul de utilizatori (`userCount`), numărul de administratori (`adminCount`), probabilitatea ca un utilizator să fie profesor (`professorProb`) etc.

Primul pas este generarea utilizatorilor. Pentru această etapă a fost folosită clasa `Faker` în obținerea unor nume aleatorii. Parola tuturor a fost setată aceeași pentru simplificare, iar primi `adminCount` utilizatori primesc `ROLE_ADMIN` pe lângă `ROLE_USER`.

Cel de al doilea pas este generarea propunerilor pentru fiecare profesor ales dintre utilizatori conform unei probabilități. Se calculează întâi un număr de locuri în corelație cu numărul de studenți ce trebuie acoperit de fiecare profesor pentru a putea da șansa unei repartizări a tuturor studenților. De menționat că tipul propunerilor este tot ales aleatoriu.

Ultimul pas este crearea preferințelor și a acordurilor utilizatorilor.

În cazul de față, numărul total de utilizatori a fost ales **500**, cu o probabilitate de **10%** ca un utilizator să fie profesor. Fiecare student are **20** de preferințe, iar numărul total de propuneri este într-un raport de **1.2** cu numărul total de studenți.

Capitolul 5

Front-end

Pentru partea de front-end a aplicației a fost ales framework-ul de Typescript Angular datorită unor mai multe avantaje. O particularitate importantă și principală a acestui framework este faptul că utilizează componente pentru crearea unei aplicații *single-page*.

Astfel, posibilitatea de creare a componentelor permite reutilizarea acestora și economisirea codului, aplicația devenind una orientată-obiect datorită și faptului că aceste componente sunt implementate în Typescript, un limbaj *strongly typed* dezvoltat din Javascript.

Nu în ultimul rând, familiarizarea cu această tehnologie are loc relativ repede, iar documentația vastă alături de diversele cursuri și tutoriale permit o învățare adecvată și dezvoltarea unor soluții optime.

5.1 Interfața

Interfața este adaptată în funcție de tipul de utilizator autentificat. Există elemente comune în general, dar și anumite funcționalități specifice.

5.1.1 Pagina principală

Prima pagină a aplicației Thesis Matcher este cea de **Matching**, vizibilă oricărui utilizator, indiferent de tipul său sau dacă este autentificat sau nu.

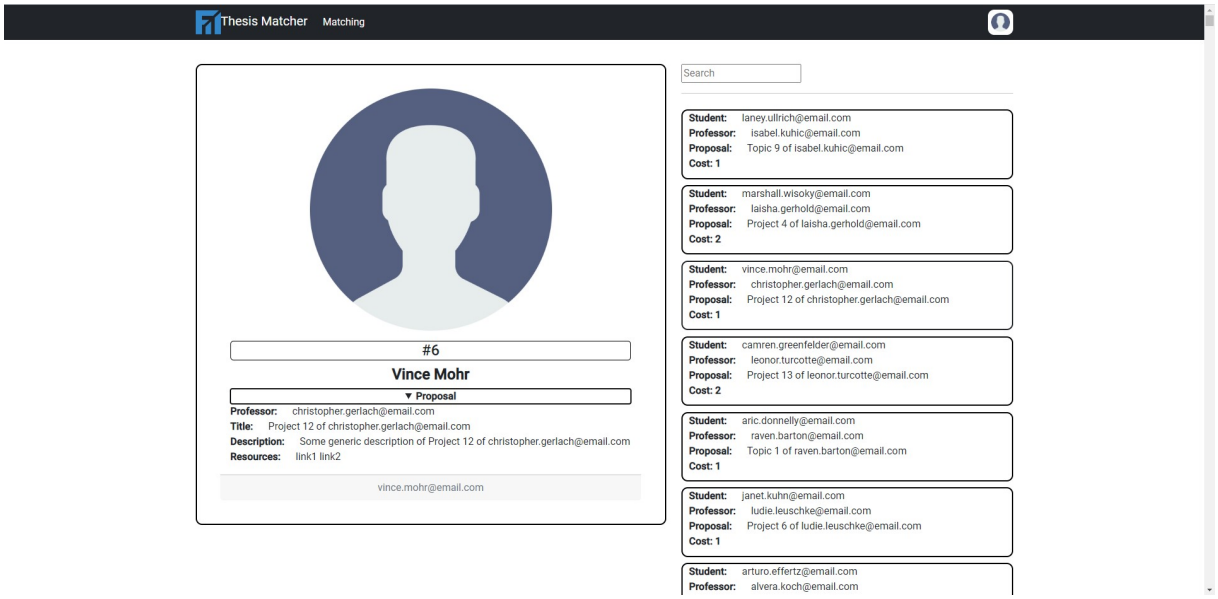

Figura 5.1: Matching (repartiția)

Aici se poate observa lista tuturor studenților și propunerea (proiectul sau tematica) asignată fiecăruia în funcție de preferințele sale. De asemenea, pentru fiecare asignare este înregistrat un *cost* care indică locul propunerii primite în ordinea preferințelor sale (de exemplu, o asignare cu costul 1 înseamnă că propunerea a fost pe primul loc în lista sa, o alta cu costul 2 a fost pe al doilea loc etc.). În plus, există o opțiune de a căuta prin rezultat după numele studentului, al profesorului sau titlul lucrării.

Totuși, opțiunile utilizatorului sunt mult restrânse, lucru ce se poate observa și din bara de navigare, el fiind nevoit să se autentifice.

Pagina de autentificare este simplă, utilizatorul fiind nevoit să introducă email-ul și parola unui cont existent după cum se poate obeserva în figura următoare.

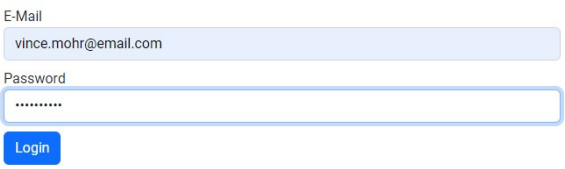

Figura 5.2: Pagina de autentificare

Odată autentificat, utilizatorul își poate schimba fotografia de profil, descrierea în care poate specifica domeniile de interes. În plus, acesta își poate schimba parola de la cont. Toate aceste setări pot fi efectuate din pagina **My Profile**.

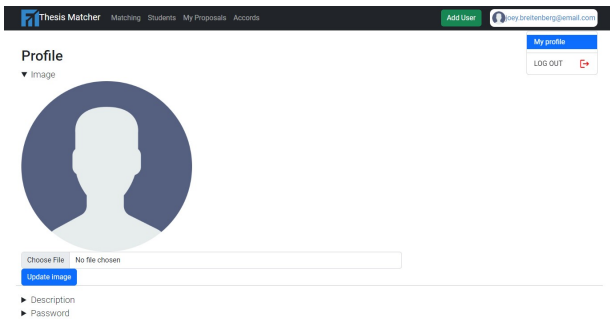

Figura 5.3: Profilul utilizatorului

5.1.2 Interfața studentului

Un utilizator de tip *student* poate accesa pagina **Professors** unde poate vedea detalii despre fiecare profesor, însă principalul scop al acestei pagini este de a accesa propunerile fiecărui profesor în parte.

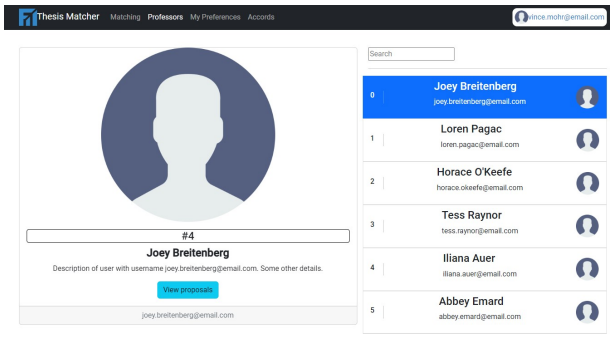

Figura 5.4: Profesorii

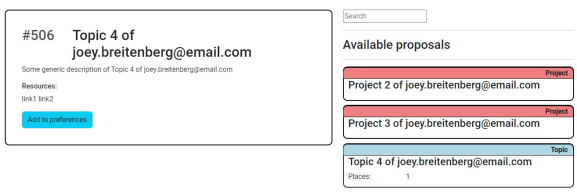

Figura 5.5: Propunerile unui profesor

Acest lucru permite adăugarea unei propuneri la lista sa de preferințe. El poate căuta în lista de propuneri a profesorului respectiv după titlu sau descriere. Pentru a marca și diferența opțiunile între ele după tip, proiectele au fost colorate cu roșu,

iar tematicile (topics) în albastru, acestea având specificat și un număr limită de locuri disponibile, spre deosebire de proiecte care au câte unul singur.

Pagina preferințelor (**Preferences**) conține toate propunerile apreciate de către student. Acesta poate elimina de exemplu un proiect din listă sau poate modifica rating-ul, un număr întreg între 1 și 100. După cum a fost deja menționat, două preferințe pot avea același rating, iar acest număr este utilizat strict pentru ordonarea preferințelor. Atunci când este adăugată o preferință, aceasta are inițializat rating-ul cu 1, fiind la finalul ierarhiei.

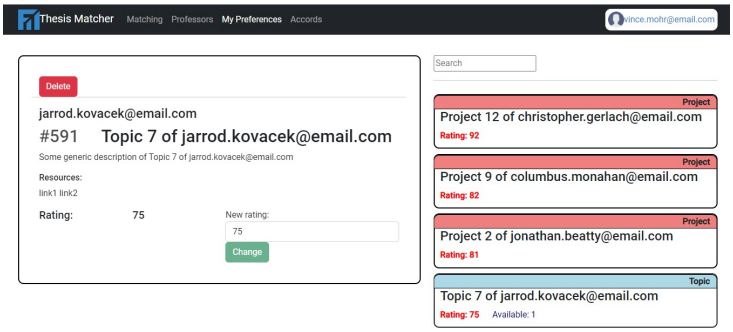

Figura 5.6: Preferințele

5.1.3 Interfața profesorului

Interfața unui profesor este în mare măsură similară. În pagina **Students** el poate vedea o listă de studenți și inițiia un acord pentru o anumită lucrare, adică să asigneze un proiect sau o tematică unui student.

Pagina **My proposals** cuprinde propunerile create de profesor. Acesta poate adăuga noi proiecte, modifica elemente deja existente sau să le elimine.

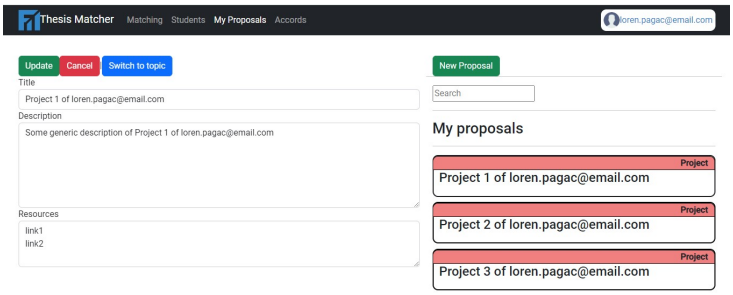

Figura 5.7: Propunerile

În secțiunea **Accords** există toate acordurile inițiate de către profesor. Pentru a fi

luat în calcul, un acord trebuie însa acceptat de către student. În acest fel, studentului i se asignează deja un proiect, iar în acest fel atât acesta, cât și propunerea, nu mai intră în pasul de determinare a unei repartizări. Acordurile acceptate sunt marcate cu verde, în caz contrar, cu roșu.

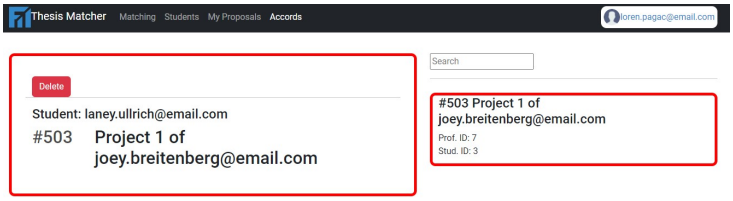

Figura 5.8: Acordurile

5.1.4 Administratorul

În cadrul acestei aplicații, fiind proiectată ca utilitar de uz intern, doar utilizatorii cu drepturi de administrator (ROLE_ADMIN) pot adăuga noi utilizatori. Prin urmare, doar aceștia au acces la pagina **Add User** de unde pot realiza acest lucru.

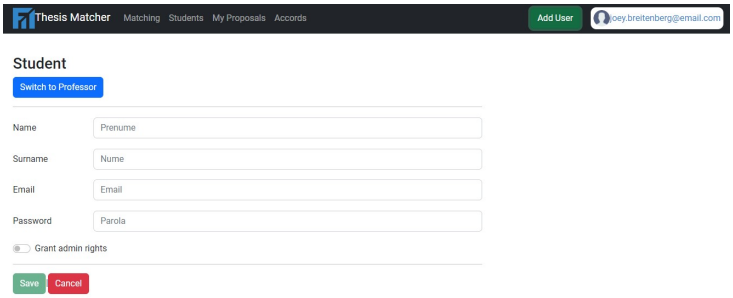

Figura 5.9: Adăugarea unui utilizator

5.2 Detalii de implementare

Înainte de a prezenta implementarea generală a părții de front-end, este necesară explicarea alcătuirii unei componente Angular.

Componenetele sunt elementele de bază ale interfeței unei aplicații Angular, o astfel de aplicație conține un arbore de componente [1]. Acestea sunt de fapt din puct de vedere tehnic un tip de directive, întotdeauna asociate cu un template.

O componentă este alcătuită dintr-un fișier Typescript ce modelează componenta prin intermediul unei clase, un *template* (șablon) ce descrie cum este construită (*rendered*) aceasta și eventual un fișiser CSS care indică detalii de stilizare.

5.2.1 Lifecylce hooks

Fiecare componentă are un ciclu de viață (lifecycle) care începe odată cu instanțierea și afișarea acesteia, continuă cu detectarea schimbărilor în view și în proprietățile instanței și se termină cu distrugerea componentei și eliminarea acesteia din DOM [**?**].

Există posibilitatea de a răspunde la astfel de evenimente din ciclul de viață al unei componente prin intermediul unor interfețe precum OnInit sau OnDestroy.

```
ngOnInit(): void {
  this.subscription = this.userService.usersChanged.subscribe(
    (users: User[]) => {
      this.users = users;
    }
  );
  this.users = this.userService.getUsers();
}

ngOnDestroy() {
  this.subscription.unsubscribe();
}
```

Figura 5.10: Metodele ngOnInit și ngOnDestroy în componenta UserList

Se poate observa cum în metoda `ngOnInit` a componentei `UserList` sunt efectuate operații necesare instanțierii și afișării listei de utilizatori. Utilizând serviciul `userService` injectat în constructor, se crează un obiect de tipul `Subscription` pentru a obține un tablou de utilizatori. În metoda `ngOnDestroy`, acest obiect este înlăturat pentru a optimiza folosirea memoriei.

5.2.2 Accesarea unei rute

Angular este utilizat în principal pentru crearea de aplicații *single-page*, adică toate funcționalitățile există într-o singură pagină HTML. Browser-ul încarcă doar părțile (componentele) necesare utilizatorului, fără a încărca o nouă pagină. De aceea, utilizatorul navighează prin intermediul rutelor predeterminate [**?**]. Acestea permit afișarea unor view-uri specifice în funcție de calea URL. Pentru a dispune de această funcționalitate, trebuie importat modulul `RouterModule` din pachetul `@angular/router`.

Rutele sunt stabilite în acest caz în modulul `AppRoutingModule`, iar definiția unei rute este din punct de vedere tehnic un obiect JavaScript [**?**]. Fiecare are măcar proprietățile `path` ce indică calea URL și `component`, numele componentei afișate. Definirea este realizată în manieră ierarhică, fiecare rută putând avea o listă de descendenți. Spre exemplu, în imaginea următoare se poate vedea rutele necesare vizualizării propunerilor, a detaliilor unui anumit element, adăugarea și modificarea unei propuneri.

```
{
  path: 'proposals',
  component: ProposalsComponent,
  canActivate: [AuthGuard, AuthProfessorGuard],
  children: [
    {
      path: '',
      component: ProposalsStartComponent,
      resolve: [ProposalResolverService],
    },
    { path: 'new', component: ProposalEditComponent },
    {
      path: ':index',
      component: ProposalDetailComponent,
      resolve: [ProposalResolverService],
    },
    {
      path: ':index/edit',
      component: ProposalEditComponent,
      resolve: [ProposalResolverService],
    },
  ],
},
```

Figura 5.11: Rutele pentru propuneri

5.2.3 Serviciile

În cadrul acestei aplicații, serviciile sunt o parte esențială în special în comunicarea dintre componente și transmiterea și primirea de informații către/de la back-end. Fiecare serviciu este adnotat cu `Injectable` pentru a permite injectarea acestora în alte obiecte.

În figura următoare este prezentată parțial implementarea serviciului de autentificare.

Figura 5.12: Serviciul de autentificare

Acesta utilizează la rândul său serviciile `HttpClient` pentru comunicarea cu back-end-ul prin request-uri HTML și `Router` pentru navigare. Astfel, în metoda `login` este efectuat un request `POST`, cu *body* de tipul `x-www-form-urlencoded` (specificat în *headers*) conținând câmpurile *username* și *password* cu valorile aferente. Rezultatul acestui request este un obiect de tipul *AuthResponseData* ce conține în special câmpul *accessToken*. Acest token este un șir de caracter reprezentând un JWT (JSON Web Token) și este unic fiecărui utilizator și necesar pentru efectuarea cu succes a oricărui request ulterior.

Un alt detaliu este reprezentat de variabila `userData`, de tipul *BehaviourSubject¡UserData¿*, care permite emiterea unor evenimente pentru a indica autentificarea și dezautentificarea utilizatorului.

Resolver

Un categorie particulară a serviciilor este cea de **Resolvers**. Un astfel de serviciu implementează interfața *Resolve* și este utilizat pentru operații premergătoare încărcării unei componente, iar specificarea acestui are loc în `AppRoutingModule`. În cazul propunerilor, prin intermediu serviciului `ProposalService` este obținută o listă completă a propunerilor profesorului autentificat.

```
@Injectable({
  providedIn: 'root',
})
export class ProposalResolverService implements Resolve<Proposal[]> {
  constructor(private proposalService: ProposalService) {}

  resolve(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Proposal[] | Observable<Proposal[]> | Promise<Proposal[]> {
    return this.proposalService.fetchUserProposals();
  }
}
```

Figura 5.13: Resolver pentru propuneri

Guard

O altă categorie este cea a gărzilor sau **Guards** în engleză, care autorizează accesul utilizatorilor la diversele rute în conformitate cu drepturile acestora, de student și profesor sau administator. În `AuthGuard` este detaliată verificarea dacă utlizatorul este autentificat. În cazul favorabil, acestuia îi este permis accesul la ruta respectivă prin returnarea valorii *true*, altfel este returnat un obiect de tipul textitUrlTree pentru a îl redirecționa spre autentificare.

```
@Injectable({
  providedIn: 'root',
})
export class AuthGuard implements CanActivate {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ):
    | Observable<boolean | UrlTree>
    | Promise<boolean | UrlTree>
    | boolean
    | UrlTree {
    return this.authService.userData.pipe(
      map((userDate) => {
        const isAuth = !!userDate;
        if (isAuth) {
          return true;
        }
        return this.router.createUrlTree(['/auth']);
      })
    );
  }
}
```

Figura 5.14: Guard pentru autentificare

5.2.4 Formularele

Obținerea, modificarea și ștergerea datelor nu ar fi posibilă fără formularele utilizate. În cadrul acestei aplicații, pe parte de front-end a fost aleasă metoda *template-driven* de implementare a formularelor (forms). Un astfel de tip folosește *two-way data*

binding pentru a actualiza modelul de date din componentă în timp ce au loc modificări și vice-versa [**?**].

```
<form #authForm="ngForm" (ngSubmit)="onSubmit(authForm)" *ngIf="!isLoading">
  <div class="form-group">
    <label for="email">E-Mail</label>
    <input type="email" id="email" class="form-control" ngModel name="email"
      required email>
  </div>
  <div class="form-group">
    <label for="password">Password</label>
    <input type="password" id="password" class="form-control" ngModel
      name="password" required minlength="6">
  </div>
  <div class="form-group">
    <button class="btn btn-primary" type="submit"
      [disabled]="!authForm.valid">Login</button>
  </div>
</form>
```

Figura 5.15: Formular de autentificare

Se poate observa cum formularul de autentificare este compus din mai multe input-uri, cel pentru email și cel pentru parolă, incluse în div-uri marcate de clasa *form-group*. Fiecare input are o serie de directive specifice ce asigură restricții necesare pentru o bună experiență a utlizatorului. De exemplu, directiva `required` indică obligativitatea completării input-ului, iar `email` verifică forma corectă a email-ului.

Procesarea datelor are loc pe în codul de Typescript odată ce formularul este trimis, în acel moment fiind apelată metoda `OnSubmit` ce primește ca argument un obiect de tipul *Form*.

5.3 Precizări

Versiunea de Angular utilizată este 13.

În primul rând, pentru stilizarea șabloanelor (templates) componentelor a fost folosit Bootstrap v5.2. Acest framework a permis facilitarea poziționării elementelor HTML, modificarea stilului și dimensiunilor acestora.

Căile URL necesare request-urilor către back-end au fost construite utilizând enum *ApiPaths* definit în fișiserul `environment.ts`.

Pentru modelarea ușoară a obiectelor JSON în comunicarea cu back-end au fost create clase *model* precum *ProposalModel*, *PreferenceModel* etc.

Capitolul 6

Back-end

Amet venenatis urna cursus eget. Quam vulputate dignissim suspendisse in est ante. Proin nibh nisl condimentum id. Egestas maecenas pharetra convallis posuere morbi. Risus viverra adipiscing at in. Vulputate eu scelerisque felis imperdiet. Cras adipiscing enim eu turpis egestas pretium aenean pharetra. In aliquam sem fringilla ut morbi tincidunt augue. Montes nascetur ridiculus mus mauris. Viverra accumsan in nisl nisi scelerisque eu ultrices vitae. In nibh mauris cursus mattis molestie a iaculis. Interdum consectetur libero id faucibus nisl tincidunt eget. Gravida in fermentum et sollicitudin ac orci. Suscipit adipiscing bibendum est ultricies. Etiam non quam lacus suspendisse. Leo urna molestie at elementum eu facilisis sed odio morbi. Egestas congue quisque egestas diam in arcu cursus. Amet consectetur adipiscing elit ut aliquam purus.

6.1 Autentificare

Eros donec ac odio tempor. Facilisi morbi tempus iaculis urna id volutpat. Faucibus in ornare quam viverra orci sagittis eu. Amet tellus cras adipiscing enim eu turpis egestas. Integer feugiat scelerisque varius morbi. Platea dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim. Bibendum arcu vitae elementum curabitur. Eu nisl nunc mi ipsum faucibus. Id aliquet lectus proin nibh nisl condimentum id venenatis a. Cras adipiscing enim eu turpis egestas pretium. Quisque non tellus orci ac auctor augue mauris augue. Malesuada pellentesque elit eget gravida cum. Ut lectus arcu bibendum at. Massa id neque aliquam vestibulum morbi blandit. Posuere ac ut consequat semper viverra nam. Viverra adipiscing at in tellus integer feugiat

scelerisque varius morbi. Morbi enim nunc faucibus a pellentesque sit amet porttitor eget. Eu feugiat pretium nibh ipsum consequat nisl vel. Nisl purus in mollis nunc sed.

6.2 Abordare

Elementum sagittis vitae et leo duis ut diam quam nulla. Purus sit amet volutpat consequat mauris nunc. Tincidunt augue interdum velit euismod in pellentesque massa. Nunc sed augue lacus viverra vitae congue. Porttitor leo a diam sollicitudin. Faucibus pulvinar elementum integer enim. Adipiscing bibendum est ultricies integer quis auctor elit. Blandit aliquam etiam erat velit scelerisque in. A iaculis at erat pellentesque adipiscing commodo elit at. Erat nam at lectus urna duis. Consequat ac felis donec et. Fermentum posuere urna nec tincidunt praesent semper feugiat nibh sed. Proin gravida hendrerit lectus a. Pretium viverra suspendisse potenti nullam ac tortor vitae purus. Arcu cursus euismod quis viverra nibh cras pulvinar mattis. Gravida arcu ac tortor dignissim convallis aenean. Quam nulla porttitor massa id neque aliquam vestibulum morbi. Sed viverra ipsum nunc aliquet. Quis enim lobortis scelerisque fermentum dui faucibus in.

Capitolul 7

Algoritmul de repartizare

Amet venenatis urna cursus eget. Quam vulputate dignissim suspendisse in est ante. Proin nibh nisl condimentum id. Egestas maecenas pharetra convallis posuere morbi. Risus viverra adipiscing at in. Vulputate eu scelerisque felis imperdiet. Cras adipiscing enim eu turpis egestas pretium aenean pharetra. In aliquam sem fringilla ut morbi tincidunt augue. Montes nascetur ridiculus mus mauris. Viverra accumsan in nisl nisi scelerisque eu ultrices vitae. In nibh mauris cursus mattis molestie a iaculis. Interdum consectetur libero id faucibus nisl tincidunt eget. Gravida in fermentum et sollicitudin ac orci. Suscipit adipiscing bibendum est ultricies. Etiam non quam lacus suspendisse. Leo urna molestie at elementum eu facilisis sed odio morbi. Egestas congue quisque egestas diam in arcu cursus. Amet consectetur adipiscing elit ut aliquam purus.

7.1 Descrierea particularităților

Eros donec ac odio tempor. Facilisi morbi tempus iaculis urna id volutpat. Faucibus in ornare quam viverra orci sagittis eu. Amet tellus cras adipiscing enim eu turpis egestas. Integer feugiat scelerisque varius morbi. Platea dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim. Bibendum arcu vitae elementum curabitur. Eu nisl nunc mi ipsum faucibus. Id aliquet lectus proin nibh nisl condimentum id venenatis a. Cras adipiscing enim eu turpis egestas pretium. Quisque non tellus orci ac auctor augue mauris augue. Malesuada pellentesque elit eget gravida cum. Ut lectus arcu bibendum at. Massa id neque aliquam vestibulum morbi blandit. Posuere ac ut consequat semper viverra nam. Viverra adipiscing at in tellus integer feugiat

scelerisque varius morbi. Morbi enim nunc faucibus a pellentesque sit amet porttitor eget. Eu feugiat pretium nibh ipsum consequat nisl vel. Nisl purus in mollis nunc sed.

7.2 Implementare

Elementum sagittis vitae et leo duis ut diam quam nulla. Purus sit amet volutpat consequat mauris nunc. Tincidunt augue interdum velit euismod in pellentesque massa. Nunc sed augue lacus viverra vitae congue. Porttitor leo a diam sollicitudin. Faucibus pulvinar elementum integer enim. Adipiscing bibendum est ultricies integer quis auctor elit. Blandit aliquam etiam erat velit scelerisque in. A iaculis at erat pellentesque adipiscing commodo elit at. Erat nam at lectus urna duis. Consequat ac felis donec et. Fermentum posuere urna nec tincidunt praesent semper feugiat nibh sed. Proin gravida hendrerit lectus a. Pretium viverra suspendisse potenti nullam ac tortor vitae purus. Arcu cursus euismod quis viverra nibh cras pulvinar mattis. Gravida arcu ac tortor dignissim convallis aenean. Quam nulla porttitor massa id neque aliquam vestibulum morbi. Sed viverra ipsum nunc aliquet. Quis enim lobortis scelerisque fermentum dui faucibus in.

Capitolul 8

Indicații de utilizare

Amet venenatis urna cursus eget. Quam vulputate dignissim suspendisse in est ante. Proin nibh nisl condimentum id. Egestas maecenas pharetra convallis posuere morbi. Risus viverra adipiscing at in. Vulputate eu scelerisque felis imperdiet. Cras adipiscing enim eu turpis egestas pretium aenean pharetra. In aliquam sem fringilla ut morbi tincidunt augue. Montes nascetur ridiculus mus mauris. Viverra accumsan in nisl nisi scelerisque eu ultrices vitae. In nibh mauris cursus mattis molestie a iaculis. Interdum consectetur libero id faucibus nisl tincidunt eget. Gravida in fermentum et sollicitudin ac orci. Suscipit adipiscing bibendum est ultricies. Etiam non quam lacus suspendisse. Leo urna molestie at elementum eu facilisis sed odio morbi. Egestas congue quisque egestas diam in arcu cursus. Amet consectetur adipiscing elit ut aliquam purus.

Eros donec ac odio tempor. Facilisi morbi tempus iaculis urna id volutpat. Faucibus in ornare quam viverra orci sagittis eu. Amet tellus cras adipiscing enim eu turpis egestas. Integer feugiat scelerisque varius morbi. Platea dictumst vestibulum rhoncus est pellentesque elit ullamcorper dignissim. Bibendum arcu vitae elementum curabitur. Eu nisl nunc mi ipsum faucibus. Id aliquet lectus proin nibh nisl condimentum id venenatis a. Cras adipiscing enim eu turpis egestas pretium. Quisque non tellus orci ac auctor augue mauris augue. Malesuada pellentesque elit eget gravida cum. Ut lectus arcu bibendum at. Massa id neque aliquam vestibulum morbi blandit. Posuere ac ut consequat semper viverra nam. Viverra adipiscing at in tellus integer feugiat scelerisque varius morbi. Morbi enim nunc faucibus a pellentesque sit amet porttitor eget. Eu feugiat pretium nibh ipsum consequat nisl vel. Nisl purus in mollis nunc sed.

Elementum sagittis vitae et leo duis ut diam quam nulla. Purus sit amet volut-

pat consequat mauris nunc. Tincidunt augue interdum velit euismod in pellentesque massa. Nunc sed augue lacus viverra vitae congue. Porttitor leo a diam sollicitudin. Faucibus pulvinar elementum integer enim. Adipiscing bibendum est ultricies integer quis auctor elit. Blandit aliquam etiam erat velit scelerisque in. A iaculis at erat pellentesque adipiscing commodo elit at. Erat nam at lectus urna duis. Consequat ac felis donec et. Fermentum posuere urna nec tincidunt praesent semper feugiat nibh sed. Proin gravida hendrerit lectus a. Pretium viverra suspendisse potenti nullam ac tortor vitae purus. Arcu cursus euismod quis viverra nibh cras pulvinar mattis. Gravida arcu ac tortor dignissim convallis aenean. Quam nulla porttitor massa id neque aliquam vestibulum morbi. Sed viverra ipsum nunc aliquet. Quis enim lobortis scelerisque fermentum dui faucibus in.

Concluzii

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Nunc mattis enim ut tellus elementum sagittis vitae et. Placerat in egestas erat imperdiet sed euismod. Urna id volutpat lacus laoreet non curabitur gravida. Blandit turpis cursus in hac habitasse platea. Eget nunc lobortis mattis aliquam faucibus. Est pellentesque elit ullamcorper dignissim cras tincidunt lobortis feugiat. Viverra maecenas accumsan lacus vel facilisis volutpat est. Non odio euismod lacinia at quis risus sed vulputate odio. Consequat ac felis donec et odio pellentesque diam volutpat commodo. Etiam sit amet nisl purus in. Tortor condimentum lacinia quis vel eros donec. Phasellus egestas tellus rutrum tellus pellentesque eu tincidunt. Aliquam id diam maecenas ultricies mi eget mauris pharetra. Enim eu turpis egestas pretium.

Bibliografie

[1] Angular - component. https://angular.io/api/core/Component. (Accessed on 02/07/2023).

[2] How to use angular development in 2022. https://www.moveoapps.com/blog/how-to-use-angular-development/.

[3] Learn hibernate tutorial - javatpoint. https://www.javatpoint.com/hibernate-tutorial. (Accessed on 01/11/2023).

[4] What is hibernate? definition from theserverside. https://www.theserverside.com/definition/Hibernate. (Accessed on 01/11/2023).

[5] When to use sql vs. nosql. https://integrant.com/blog/when-to-use-sql-vs-nosql/. (Accessed on 01/10/2023).