

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

Aplicație bazată pe microservicii pentru regăsirea în scopuri educaționale a unor persoane cu interese comune

LUCRARE DE LICENȚĂ

Coordonator științific
Ș.I. dr. inf. Tiberius Dumitriu

Absolvent
Radu-Valentin Cornea

Iași, 2023

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII LUCRĂRII DE DIPLOMĂ

Subsemnatul CORNEA RADU-VALENTIN , legitimat cu CI seria ZC nr. 593105 , CNP
5000724046250 autorul lucrării

APLICAȚIE BAZATĂ PE MICROSERVICII PENTRU REGĂSIREA ÎN
SCOPURI EDUCAȚIONALE A UNOR PERSOANE CU INTERESE COMUNE

elaborată în vederea susținerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași, sesiunea IULIE 2023 a anului universitar 2022-2023, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 – Funcționarea Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a convențiilor internaționale privind drepturile de autor.

Data

Semnătura

Iași, 2023

Cuprins

Rezumat.....	1
Introducere	3
Cap. I. Tehnologiile și arhitectura aplicației.....	5
I.1. Tehnologiile utilizate	6
I.1.1. Backend	6
I.1.2. Frontend	6
I.1.3. Elementele de inteligență artificială	7
I.2. Arhitectura aplicației	8
I.2.1. Gateway.....	9
I.2.2. Identity Management	9
I.2.3. Profile Service	10
I.2.4. Algorithms Service	11
I.2.5. Setul de date	12
I.2.6. Web Application.....	13
Cap. II. Funcționalitatea și implementarea aplicației	14
II.1. Funcționalitatea aplicației.....	14
II.2. Implementarea aplicației.....	16
II.2.1. Gateway.....	17
II.2.2. Identity Management	20
II.2.3. Profile Service	24
II.2.4. Algorithms Service	25
Cap. III. Rezultatele aplicației.....	30
III.1. Experimente realizate.....	30
III.2. Rezultatele obținute	31
Concluzii	37
Bibliografie.....	38

Aplicație bazată pe microservicii pentru regăsirea în scopuri educaționale a unor persoane cu interese comune

Radu-Valentin Cornea

Rezumat

În zilele noastre, fenomenul prieteniiilor online ia din ce în ce mai mare amploare și, din păcate, acest fapt nu se datorează doar avansării tehnologiei. Bine-cunoscuta pandemie din 2019 a venit la pachet cu o mulțime de probleme: oamenii au început să sufere de singurătate și izolare socială, în special oamenii mai vârstnici sunt cei mai vulnerabili și care au nevoie să fie conectați cu ceilalți [1]. Această perioadă, deși a luat prin surprindere pe toată lumea, a făcut ca cei mai mulți oameni să găsească alternative online de a discuta și relaționa cu ceilalți. Astfel, oamenii au reușit să-și împărtășească gândurile și sentimentele, fără prejudecăți, și chiar să dezvolte prietenii frumoase și de lungă durată alături de persoane cu interese similare.

Totuși, realizarea acestor aplicații nu este ușoară. Există multe abordări de a recomanda persoane cu interese similare. Obiectivul principal al acestei lucrări constă în găsirea celor mai potrivite metode și tehnici de recomandare din sfera inteligenței artificiale, în funcție de preferințele pe care le au oamenii în comun cu ceilalți utilizatori, astfel încât aceștia să aibă cât mai multe în comun cu alte persoane. În cadrul aplicației, accentul cade pe grupuri de interes din sfera educațională, dar proiectul poate fi extins la orice alte domenii.

Tema a fost aleasă pe baza unor experiențe cu mai multe aplicații și pe baza unor observații cu privire la faptul că foarte puține dintre acestea îndeplineau cerința de a primi recomandări de utilizatori similari pe baza preferințelor. S-a constatat că utilizatorii găsiți de aplicație erau diferiți din punct de vedere al criteriilor introduse în aplicație. În felul acesta, proiectul își propune să realizeze și să testeze aceste metode de recomandări în conformitate cu ce au nevoie și și-ar dori ceilalți utilizatori, câștigându-le timp, încât aceștia să găsească persoanele potrivite mai rapid, să aibă o experiență plăcută și satisfăcătoare. În același timp, s-a luat în considerare integrarea conceptelor studiate în cadrul materiilor de programare web, statistică, învățare automată precum și a celor legate de sistemele distribuite.

Se consideră că îmbinarea microserviciilor cu inteligența artificială reprezintă o abordare promițătoare pentru a permite oamenilor să găsească alți utilizatori cu preferințe similare. Studiul redă câteva aspecte generale ale aplicațiilor bazate pe microservicii, modalități de securizare a aplicației, tehnologiile folosite, motivația realizării unei aplicații diferite de cele existente, arhitectura realizată și rezultatele obținute.

Microserviciile sunt utilizate cu scopul de a îmbunătăți procesul de dezvoltare, dar și de a modulariza aplicația încât microserviciile acesteia să poată eșua independent, să fie scalate pe orizontală (adăugând mai multe sisteme) sau verticală (adăugând mai multă putere de procesare RAM, CPU, GPU sistemelor existente, ori folosind containere virtuale). Fiecare dintre microserviciile utilizate are propriile pachete și clase. Un alt punct tare al aplicației îl reprezintă

persistența datelor, ele continuând să existe și după ce au loc erori, iar ori de câte ori se pornesc serverele, datele de interes sunt încărcate din bazele de date sau fișiere. Pe lângă acestea, s-a implementat și o interfață grafică, folosind și un cadru pentru dezvoltarea mai facilă a acestora (Thymeleaf).

Experimentele efectuate presupun aplicarea unor algoritmi de inteligență artificială, precum K-Nearest Neighbors (KNN) pentru a i se recomanda unui nou utilizator, pe baza preferințelor sale, un grup de persoane (cunoscute sau nu) ce au interese similare. Pentru aceasta se utilizează valori diferite ale parametrului K, precum și mai multe metrici (Jaccard, euclidiană, cosine etc.) cu scopul de a determina cea mai potrivită recomandare.

În primul capitol al lucrării se discută despre tehnologiile folosite și importanța acestora, dar și mai important este subcapitolul ce abordează noțiunile de arhitectură bazate pe microservicii și inteligență artificială. Tot în acest subcapitol se pătrunde în detalii cu privire la microserviciile utilizate: se discută despre motivația alegerii lor, unor baze de date, mecanisme de comunicare și se prezintă, unde este cazul, diagrame utile înțelegerii modului în care funcționează sistemul.

Capitolul al doilea prezintă funcționalitatea aplicației și implementarea acesteia. În primul subcapitol se discută despre interfața grafică a aplicației și elementele importante utilizatorilor, alături de figuri ce ilustrează Frontend-ul aplicației la nivel minimalist. Cel de al doilea subcapitol prezintă secvențe de cod legate de implementare ale aplicației.

Al treilea capitol prezintă rezultatele aplicației. Aici se discută despre experimentele realizate, rolul acestora, modul în care au fost realizate, precum și rezultatele obținute, interpretarea și semnificația acestora.

La final sunt prezentate concluzia cu privire la proiectul realizat, precum și a experimentelor și rezultatelor obținute și bibliografia ce cuprinde sursele de inspirație utilizate în cadrul realizării acestui document.

Pentru realizarea aplicației s-au utilizat: IDE-urile IntelliJ IDEA, PyCharm, Visual Studio Code pentru Python, Kotlin, JavaScript, HTML, CSS, încât să se ruleze cu succes serverele aplicației; baze de date MongoDB, MariaDB; mecanisme de securizare (JWT, criptare, decriptare) utilitare precum DBeaver, Compass, Postman; librăriile Python pentru lucrul cu texte, server-ul Flask și antrenarea modelelor de inteligență artificială (KNN). În Kotlin, sunt importate dependențele Maven pentru Spring, Spring Security, REST, Thymeleaf și bazele de date, dar se recomandă o instalare curată. Este posibil să fie necesară instalarea Java.

Introducere

Astăzi există o mulțime de aplicații de socializare și de conectare cu ceilalți: de la aplicații pentru angajați și angajatori (LinkedIn, Indeed), la aplicații agregatoare de știri (Reddit), de discuții cu cei apropiați (Facebook, WhatsApp, Instagram), până la aplicații pentru a cunoaște prieteni (PersonalityMatch, Connected2.me) și chiar și cu tente romantice (Tinder). Tot mai mulți oameni ajung să depindă de aceste tehnologii, putând discuta cu ceilalți oameni din întreaga lume.

Există o mulțime de abordări pentru a realiza recomandările de persoane, iar una dintre acestea este bazată pe similaritatea preferințelor deținute de utilizatori. Preferințele sunt doar niște grupuri de cuvinte care reprezintă trăsăturile definitorii utilizatorilor. De exemplu, în domeniul tehnologiei informației, preferințele unui utilizator ar putea fi reprezentate de: kotlin, python, inteligență artificială. Una dintre strategiile des întâlnite de recomandare a oamenilor este de a căuta utilizatorii care să aibă cât mai multe trăsături în comun (să fie cât mai similari).

Proiectul realizat constă într-o aplicație pentru interconectarea oamenilor, în funcție de preferințele lor, utilizând diverși algoritmi. Obiectivul proiectului este de a uni oamenii mai ușor în scop educațional în funcție de preferințele legate de tehnologii sau concepte teoretice, însă proiectul ar putea fi extins și în alte arii și domenii decât cele educaționale, în funcție și de nevoile aplicației și ale firmelor. În proiect, preferințele reprezintă cuvinte din domeniul tehnologiei informației. Atunci când un utilizator dorește să cunoască cei mai potriviți oameni, algoritmiul preia preferințele lui și ale celorlalți, apoi algoritmiul caută cei mai potriviți utilizatori țintei respective. Țintei i se asociază utilizatorii cu o similaritate mai mare de a avea preferințe comune. În final, el este cel care decide ce să facă cu aceste informații, de exemplu să înceapă o nouă conversație sau să se mai gândească cu cine să intre în discuție. Tot în cadrul proiectului s-au realizat statistici cu privire la algoritmiul utilizat (de exemplu acuratețea acestora), s-au integrat și microserviciile pentru gestionarea identităților utilizatorilor, a punctului de acces în aplicație (Gateway-ul) și s-a realizat și o interfață grafică cu utilizatorul la un nivel minimalist.

Unii dintre autori precum Xin Wei [2], Zeinab Shahbazi [3], Qian Zhang [4], au gândit sisteme de recomandări bazate pe metode ale inteligenței artificiale și pe psihologia educației, folosind resurse precum imagini, texte, clipuri și link-uri. În funcție de cum se interacționează pe site-uri cu resursele menționate. Studenții au fost clasificați în: studenți activi, studenți cu potențial și studenți inactivi. Toate aceste trei grupuri au fost împărțite pe baza modului în care interacționau cu paginile respective, urmând ca fiecare dintre ei să primească resurse educative potrivite [2]. Există foarte multe domenii de aplicare pentru algoritmiul de recomandare și pot fi folosiți nu doar în scopuri educative ci și divertisment, precum filmele [4].

În urma unei studieri ale aplicațiilor de pe piață, s-a constatat că, în ciuda faptului că există aplicații care încearcă să recomande persoane potrivite după anumite criterii, acestea nu realizează în totalitate obiectivul dorit. Multe dintre aplicațiile găsite pe Play Store includ opțiunea de alegere de preferințe (de exemplu, mâncare, hobby-uri, muzică), însă filtrele de utilizatori sunt inexistente, neputând primi utilizatori similari unei ținte, cel puțin din punctul de vedere al preferințelor [5]. Singura aplicație care s-a constatat că ar face o parte dintre aceste funcționalități de recomandare este Meetup, dar acolo au loc recomandări de evenimente, nu de persoane. Panion ar fi fost un exemplu bun, dar în prezent nu mai funcționează publicului larg.

Din acest motiv, s-a luat în calcul că o astfel de aplicație pentru recomandarea persoanelor în funcție de preferințele lor ar fi necesară. Ținta principală în cadrul aplicației reprezintă oamenii dornici să își îmbunătățească abilitățile, discutând și învățând cu alți utilizatori ce au preferințe similare lor. Obiectivul este acela de a găsi și de a filtra cât mai mulți utilizatori potriviți cu filtrele aplicate, dar și de a avea un produs funcțional, sigur și securizat, care să fie ușor de înțeles, plăcut și intuitiv de folosit.

O posibilitate de a rezolva această problemă de tip recomandări de persoane este dată de utilizarea microserviciilor și inteligenței artificiale. Arhitecturile bazate pe microservicii oferă scalabilitate, rulare independentă, performanțe adiționale, avantajul de a fi ușor de menținut, costuri reduse etc. [6], în timp ce inteligența artificială oferă posibilități multiple de modelare și antrenare a datelor după anumite seturi de date. Câțiva dintre acești algoritmi ce pot fi folosiți în acest sens sunt: K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Singular Value Decomposition (SVD), Random Forest, filtrul colaborativ, filtrul bazat pe conținut, abordări hibride [5, 6].

Ceea ce face acest proiect diferit de celelalte este faptul că oferă posibilitatea de a lăsa utilizatorii să decidă prin ce metode să descopere alți utilizatori, folosindu-se de mecanisme ale inteligenței artificiale, precum și de microservicii. Utilizatorii trebuie doar să aleagă strategia de căutare a celorlalți utilizatori, precum și un limitator procentual, ce reprezintă cei mai potriviți x% utilizatori, obținând astfel utilizatori cu interesele cele mai asemănătoare țintei în cauză. Toate aceste procesări au loc în background, utilizatorul nefiind nevoit să cunoască modul în care se obțin rezultatele. Procesările constau în antrenarea modelului și potrivirea acestuia cu noi utilizatori înregistrați, pe măsură ce rulează serverele.

Cap. I. Tehnologiile și arhitectura aplicației

În cadrul acestui capitol se discută despre tehnologiile folosite, importanța lor în cadrul aplicației, arhitectura folosită, modul în care a fost concepută aceasta, precum și motivațiile din spatele acestor alegeri. Cele mai importante specificații ale aplicației sunt redată în rândurile de mai jos.

Este necesară *existența unui sistem de autentificare și autorizare*. Cele mai multe dintre aplicațiile de socializare au o modalitate de a gestiona conturile utilizatorilor. În plus, în cazul în care unii și-ar dori să reintre în aplicație să discute cu alte persoane, trebuie să poată reveni în punctul rămas, iar o modalitate de a realiza aceasta este cea bazată pe sistemele de gestiune a identităților. În felul acesta, nu doar că utilizatorii vor putea reveni mai ușor în punctul rămas, ci și vor avea conturi proprii inaccesibile de terțe persoane.

Totuși este util ca un sistem de gestionare a identităților să folosească *mecanisme de securizare a aplicației*. Deseori se îmbină mai multe modalități, tocmai pentru a spori gradul de securizare. În această lucrare, s-a folosit codificarea parolelor, prin intermediul funcției puse la dispoziție de cadrul Spring Security, *BCryptPasswordEncoder*. Pe lângă aceasta, s-au folosit JWT-urile, unde se stochează claim-urile fiecăror dintre utilizatori (claim-urile cuprind detalii cu privire la autoritățile deținute de utilizatori). Pentru a preveni situațiile în care utilizatorii aleg să modifice acești tokeni JWT, la nivelul clientului sunt criptați și stocați într-un cookie, iar ori de câte ori ajung necesari la nivelul serverelor, aceștia sunt decriptați, obținându-se claim-urile de interes, nealterate de clienți, care vor decide mai departe dacă persoana este autentificată și autorizată.

Modularizarea serviciilor și claselor, face parte din bunele practici de programare. Ea este utilizată tocmai pentru sporirea procesului de dezvoltare, prin prisma faptului că, proiectul ajunge foarte bine organizat, știind exact unde trebuie realizate următoarele operații logice de creare, citire, actualizare sau ștergere. Există și varianta folosirii unui monolit, însă această abordare poate să fie împotriva dezvoltatorilor la un moment dat, tocmai din cauză că principiul separării preocupărilor ajunge încălcat (*engl. separation of concerns*). De aceea, deși poate fi justificată în anumite situații folosirea unui monolit, este preferată evitarea acestui model arhitectural al aplicației.

Un alt lucru important este realizarea unei *interfețe grafice cu utilizatorul, la nivelul paginilor web*. Din dorința de a oferi o experiență cât mai bună utilizatorilor finali, este necesară o interfață grafică, chiar și la nivel minimalist. În loc să fie nevoiți să realizeze cereri din terminal, este mult mai de preferat să poată introduce detaliile necesare la nivel de formulare, restul logicii de execuție a cererilor fiind realizate de servere.

Tot pentru a îmbunătăți experiența celorlalți oameni, este necesară *utilizarea bazelor de date*. În cazul în care sistemele întâmpină erori sau eșuează, trebuie reluată activitatea dintr-un anumit punct, realizând astfel persistența datelor. Pe lângă aceasta, atunci când se doresc testarea anumitor funcționalități din cadrul aplicațiilor, în special a celor care depind de anumite seturi de date, precum algoritmi de inteligență artificială, bazele date și sistemele de gestiune a fișierelor ajung esențiale.

Un alt punct important al aplicației îl reprezintă *sistemul de recomandări*. Acesta trebuie să fie funcțional și corect. Fără el, aplicația nu ar mai pune la dispoziție strategiile pentru realizarea recomandărilor de persoane bazate pe inteligență artificială. Tot acest punct este folosit și pentru testarea și explorarea algoritmilor ce țin de învățarea automată.

1.1. Tehnologiile utilizate

Pentru dezvoltarea proiectului *aplicațiile software*: IntelliJ IDEA, PyCharm, Visual Studio Code, Postman, Compass, DBeaver au contribuit la realizarea proiectului final.

1.1.1. Backend

Dezvoltarea componentelor de *Backend* presupune operații ce au loc la nivel de server, ce țin de logică, creare a API-urilor, integrare a componentelor, autentificare și autorizare a utilizatorilor, securizare a aplicației și a bazelor de date utilizate, astfel încât sistemele să poată fi scalabile, întreținute în timp și să gestioneze volume mari de trafic, folosindu-se de Gateway, cozi de mesaje și servicii cloud [7]. În ceea ce privește Backend-ul aplicației, s-a ales limbajul Kotlin, iar framework-ul folosit este Spring Boot, împreună cu serviciile RESTful.

Kotlin permite dezvoltatorilor să fie mai productivi, prin scrierea de cod mai puțin decât în limbaje precum Java, și de a obține același rezultat ca acesta [8]. Ca o completare a limbajului Java, Kotlin vine cu o mulțime de elemente noi, precum cele legate de calculul funcțional (expresiile lambda, funcțiile de extensie), de programare orientată obiect (data classes, obiecte companion), și cele legate de calculul paralel (corutinele). Documentația pusă la dispoziție este foarte ușor de urmărit. Ce este iarăși util în contextul problemei este că limbajul Kotlin poate fi folosit atât pentru dezvoltarea aplicațiilor Android, cât și pentru aplicații Web, fiind o alternativă foarte bună pentru dezvoltarea serviciilor REST/RESTful.

Serviciile REST reprezintă un stil arhitectural, ce lucrează cu resurse și se folosește de verbe HTTP precum GET, PUT, POST, DELETE în scopul accesării acestora [9, 10]. Avantajele utilizării serviciilor REST sunt acelea că ușurează implementarea dezvoltatorilor, sunt și mai ușor de înțeles și urmărit. Dacă sunt implementate corect, pot fi foarte utile și în a scala aplicația pe orizontală, cu ajutorul load balancer-elor. Altă funcție pe care o au aceste servicii sunt flexibilitatea, putându-se realiza *servicii RESTful* în funcție și de cum doresc dezvoltatorii.

Acestea pot fi dezvoltate cu ajutorul lui *Spring Boot*, un framework pentru realizarea aplicațiilor web, pentru Backend [11]. Este folosit și pentru a facilita conexiuni cu bazele de date, în cazul de față, MariaDB și MongoDB. Spring Boot pune la dispoziție și suport pentru securitate, cu ajutorul lui Spring Security [12]. Prin intermediul lui Spring Security se vor folosi facilitățile pentru securizarea rutelor aplicației. Iar pe lângă aceasta, pentru a spori gradul de securitate al aplicației, s-au folosit și mecanisme de criptare, decriptare și JWT-urile [13] (standard RFC 7519, folosit pentru reprezentarea claim-urilor dintre două părți în siguranță [14]).

1.1.2. Frontend

Frontend-ul are ca scop crearea unui design plăcut, interactiv, ușor de folosit și accesibil utilizatorilor. Este o combinație între *HTML*, ce reprezintă scheletul și structura paginilor web, *CSS*, folosit pentru stilizarea paginilor, și *JavaScript*, utilizat pentru a defini comportamentul dinamic și interactivitatea [15]. Există un set de reguli despre care trebuie să se țină cont atunci când se creează Frontend, precum alegerea unei palete de culori aflate în contrast, semnificația culorilor să fie corectă și consecventă, iar acțiunile utilizatorilor să fie intuitive, fără să fie nevoiți să dea prea multe click-uri în cadrul paginilor [16].

Unul dintre framework-urile utilizate pentru interfața grafică este *Thymeleaf* [17]. Motivația este simplă: comunică foarte bine cu ecosistemul Spring Boot și oferă facilități pentru metode HTTP precum POST, la nivelul server-ului:

```
@PostMapping("/register")
fun register(@ModelAttribute userModel: UserDTO): ModelAndView {
    if(!userService.isUserRegisterValid(userModel))
```

```

        return ModelAndView("register-fail")
    return ModelAndView("register-success")
}

```

Iar la nivel de view, acesta ar putea fi un mod de a-l folosi:

```

<form th:action="@{/register}" th:object="${user}"
method="post">
<input id="register_username" type="text" placeholder="e.g. SuperUser23"
th:field="*{username}"/>

```

1.1.3. Elementele de inteligență artificială

Inteligența artificială este un domeniu larg, care se referă la utilizarea tehnologiilor pentru a construi mașini ce au capacitatea de a imita funcțiile cognitive asociate cu inteligența umană, cum ar fi capacitatea de a vedea, înțelege și răspunde la limbajul vorbit sau scris, analiza date, de a face recomandări și multe altele. Deși inteligența artificială este adesea gândită ca un sistem în sine, este un set de tehnologii implementate într-un sistem pentru a-i permite să raționeze, să învețe și să acționeze pentru a rezolva o problemă complexă [18].

Învățarea automată este un subset al inteligenței artificiale, care permite automat unui sistem să învețe și să se îmbunătățească prin experiență. Învățarea automată folosește algoritmi pentru a analiza cantități mari de date, pentru a învăța din informații și apoi pentru a lua decizii informate. Algoritmii de învățare automată îmbunătățesc performanța în timp pe măsură ce sunt antrenați și, deci, expuși la mai multe date [18].

Dezvoltarea aplicațiilor ce conțin elemente ale inteligenței artificiale poate fi mai provocatoare pentru începători, în special din cauză că dezvoltatorii sunt nevoiți să aibă fundamente teoretice ce țin de matematică și programare. În plus, există mai multe subramuri ale acestui domeniu care pot necesita cunoștințe legate de prelucrarea grafică, statistică, algebră, analiză matematică. De obicei, dezvoltatorii preferă utilizarea limbajului Python pentru acest tip de cerințe, tocmai din cauză că există foarte multe librării ce vin cu scopul de a le facilita procesul de dezvoltare, precum *scikitlearn*. Totuși, se pot folosi și alte limbaje, precum C#, Kotlin sau chiar C++ pentru performanțe extra, fiind un limbaj mai low-level.

În ceea ce privește acest domeniu, există o mulțime de algoritmi ce ar putea fi folosiți pentru antrenarea datelor, dar foarte important este să se și potrivească cu cerințele dezvoltatorilor. Există sisteme de recomandare bazate pe conținut, pe cunoștințe, personalizat, colaborativ, hibrid. Există și algoritmi de grupare precum K-Means, Expectation-Maximization; de clasificare (Arbori de decizie, metoda Bayes naivă, K-Nearest-Neighbours); se pot folosi chiar și rețele neuronale sau algoritmi precum Support Vector Machine, Singular Value Decomposition.

În cadrul aplicației, componentele ce țin de inteligență artificială și învățare automată presupun antrenarea unor diverși algoritmi pe baza unor seturi de date de intrare salvate și actualizate odată cu rularea întregii aplicații. Tot aici, pe baza rezultatelor obținute, se obțin statistici cu privire la rezultatele obținute de modulele astfel create. Pentru dezvoltarea microserviciului de algoritmi de inteligență artificială, s-au folosit Python, Flask, matplotlib și pandas.

Python este un limbaj de programare folosit pentru integrarea și crearea rapidă de sisteme, având o sintaxă intuitivă, fiind un limbaj ușor de interpretat și de urmărit [19]. *Pandas* [20] a fost folosit pentru a extrage datele dintr-un element HTML de tip tabel ce include utilizatorii aplicației. *matplotlib* [21] este utilizat pentru a genera diverse grafice pentru o mai bună vizualizare a datelor de ieșire a aplicației și a comparării rezultatelor obținute. *Flask* [22] este util pentru a realiza legătura cu server-ul Kotlin pentru Profilurile utilizatorilor.

I.2. Arhitectura aplicației

O posibilitate de a implementa soluția propusă, așa cum s-a discutat anterior, este folosirea microserviciilor împreună cu inteligența artificială. Componentele principale necesare sunt: Aplicația Web, Gateway-ul, serviciul pentru gestionarea identităților IDM (Identity Management), cel pentru controlul profilelor utilizatorilor (Profile Service) și unul pentru algoritmi de inteligență artificială (Algorithms Service). Alte servicii posibile a fi utile unei astfel de aplicații sunt cele pentru identificarea locației, comunicarea între utilizatori, și pentru suport-ul utilizatorului (în cazul în care acesta are nevoie de ajutor, să poată depune cereri).

S-au ales microserviciile pentru că oferă avantajul îmbunătățirii procesului de dezvoltare, prin împărțirea sistemului în componente mai mici, mai ușor de gestionat (*Figura I.1*). Microserviciile permit eșuarea independentă, o modularizare mai bună, scalabilitatea, alocarea mai eficientă de resurse, conducând astfel la o aplicație software îmbunătățită [23].

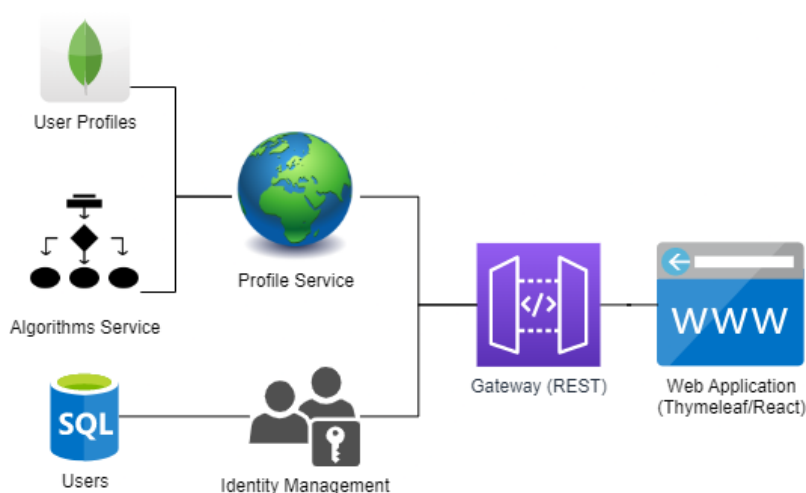


Figura I.1. Diagrama de servicii

Așa cum se poate remarca din figura de mai sus, Gateway-ul reprezintă liantul aplicației, el făcând legătura dintre Frontend (Aplicația Web) și Backend (IDM, Profile Services). Gateway-ul este folosit și pentru a securiza rutele aplicației. Aplicația Web reprezintă interfața grafică cu utilizatorul la nivelul paginilor web. Aici utilizatorii pot interacționa cu elemente de tip HTML, să se înregistreze, să performeze acțiuni de tip login, register, logout, să-și vizualizeze contul și să primească recomandările utilizatorilor bazate pe metoda aleasă, precum și un limitator procentual. Serviciul Identity Management este cel care se asigură de păstrarea identității utilizatorilor, precum și performarea operațiilor de autentificare și autorizare. Conturile utilizatorilor folosite pentru IDM sunt stocate într-o bază de date de tip relațional SQL. Pe lângă acest serviciu, mai există Profile Service, folosit pentru afișarea propriului cont, precum și extragerea utilizatorilor cei mai potriviți din punct de vedere al similarităților, utilizând o bază de date NoSQL de tip document, MongoDB, precum și Algorithms Service, folosit pentru extragerea utilizatorilor prin algoritmi de inteligență artificială testați. Acest serviciu din urmă cuprinde și module utile pentru testarea metodelor utilizate, măsurându-se gradele de similaritate dintre doi utilizatori, și acuratețea algoritmilor, precum și metode pentru afișarea datelor colectate în urma rulării acestor teste. Astfel, codul a fost despărțit pe module, clase și servicii, în unele situații fiind necesare șabloanele de proiectare și șabloanele arhitecturale.

1.2.1. Gateway

Modelul arhitectural Gateway se recomandă în cazul aplicațiilor mari și complexe bazate pe microservicii cu mai multe aplicații client. Modelul este similar șablonului de proiectare *fațadă*, dar face parte dintr-un sistem distribuit de proxy invers sau de rutare gateway pentru utilizare ca model de comunicare sincronă [24].

Gateway-ul este componenta ce leagă Backend-ul aplicației de Frontend, iar rolul lui este de a adăuga un strat ce conține componente de securitate, dar și de a redirecta cererile clienților către serverele potrivite. La acest nivel s-au creat Controllere, astfel încât să poată fi ușor create cererile către celelalte dintre microservicii, fiecare Controller fiind asociat câte unui dintre servicii. Securizarea aplicației din cadrul acestei componente constă în faptul că se realizează verificările tokenilor criptați JWT, din care se extrag claim-urile necesare și se caută autoritățile de interes, folosite mai departe pentru a decide dacă executantul acelei cereri este cine spune că este și are drepturile de acces ale acelei rute.

1.2.2. Identity Management

Serviciul IDM se ocupă cu identificarea utilizatorilor și autorităților acestora, dar și de crearea conturilor unor noi utilizatori. Scopul serviciului este de a oferi diferitor utilizatori accesul la unele rute pe care alții nu îl au. În spate există baza de date MariaDB folosită pentru a stoca aceste detalii legate de conturile utilizatorilor, dar și legate de tokenii emiși și starea lor. Această bază de date s-a ales pentru că este Open-Source, dar și pentru că oferă consistență datelor. În ceea ce privește obiectele folosite, s-a folosit șablonul arhitectural *Data Transfer Object (DTO)*. DTO reprezintă obiecte utilizate pentru a transfera datele dintre procese și este utilizat cu scopul de a reduce numărul de apelări și request-uri [25].

Pentru siguranța datelor utilizatorilor, s-au utilizat diverse mecanisme de protecție a datelor, precum JWT-urile, păstrate la nivelul clientului prin Cookies, în format criptat (tocmai pentru a se evita schimbarea câmpurilor din interiorul lor), ele fiind mai întâi decriptate la nivelul server-ului, apoi se validează formatul și semnătura acestora, urmând abia apoi să se realizeze validările pe câmpurile lor (de exemplu, există un câmp expiry), autorizarea realizându-se abia la sfârșit, în funcție de autoritatea pe care o deține utilizatorul respectiv (*Figura 1.2*).

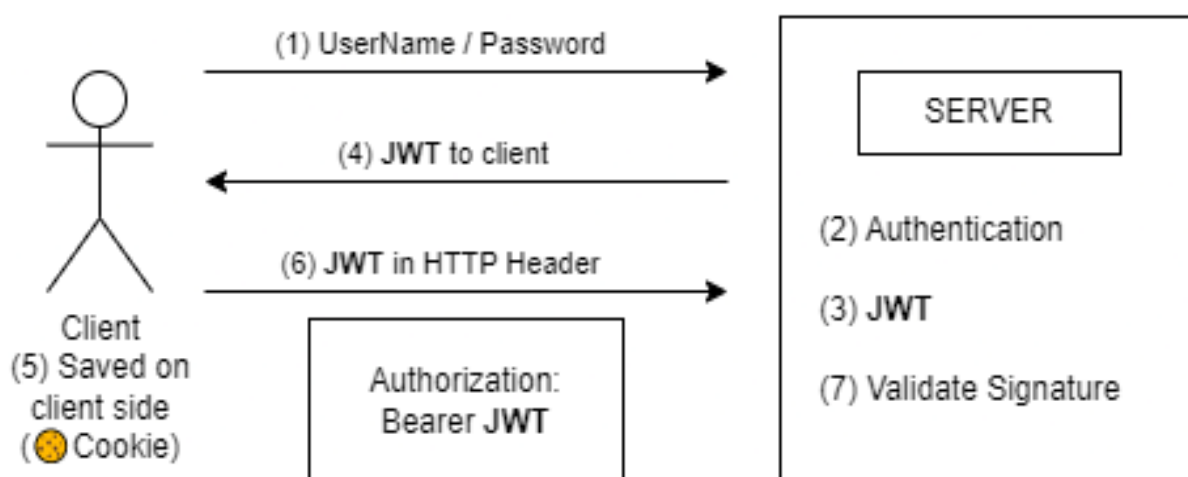


Figura 1.2. Interacțiunea dintre client și server folosind JWT

În *Figura 1.3* este prezentată modalitatea de validare a unui token JWT. Tokenii JWT au un format propriu, reprezentat de *Headers* (h), *Payload* (p) și *Signature* (s). *Headers* cuprind detalii

cu privire la algoritmul folosit (HMAC256), tipul token-ului folosit (JWT). *Payload* cuprinde detalii cu privire la claim-uri, iar claim-urile reprezintă detaliile de interes (uuid, id, username, password, authorities, issued at, not before, expires at). *Signature* reprezintă codificarea dintre *Header*, *Payload* și un *secret*. Sumar, *Signature* reprezintă rezultatul concatenării dintre următoarele elemente: *Header* codificat, punct, *Payload* codificat, secret. Pentru a fi valid token-ul, semnătura acestuia trebuie să fie identică cu rezultatul concatenat dintre elementele menționate.

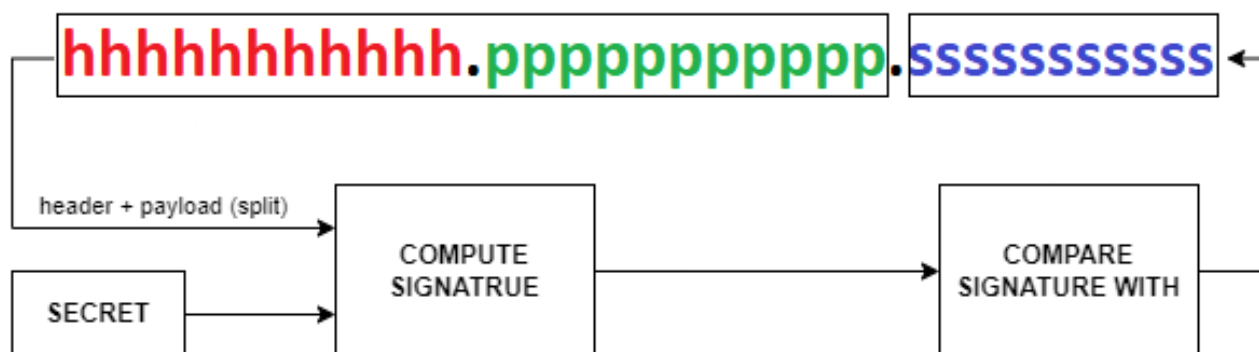


Figura I.3. Validarea semnăturii unui JWT

Pentru ca parolele să fie persistente, acestea sunt ținute în baza de date criptate cu ajutorul funcției *BCryptPasswordEncoder*, iar în cazul în care un client se autentifică, la verificare, parola introdusă este criptată și ea și comparată cu valoarea din baza de date. S-ar fi putut folosi alte metode pentru autentificare și autorizare, precum OAuth2, SAML, OpenID, dar în scop demonstrativ și pentru a asigura securizarea aplicației, s-au preferat JWT-urile, Spring Security și modalitățile clasice de codificare, criptare și decriptare a datelor.

Ecosistemul Spring pune la dispoziție mecanisme de securizare a aplicației, precum Spring Security. Acesta poate fi folosit pentru autentificare, dar și autorizare. Acesta se folosește de lanțuri de filtre (filter chains) și dispune de o sintaxă proprie, Spring Expression Language (SpEL), pentru autorizarea într-un mod facil al utilizatorilor în funcție de diverși parametri, precum *hasIpAddress*, *hasAuthority*, *principal* și mulți alții [26].

I.2.3. Profile Service

Serviciul Profile funcționează împreună cu IDM, dar este destinat utilizatorilor care deja sunt autentificați, sau care își crează cont. Un profil de utilizator este posibil ca, în viitor, să aibă altă structură și alte câmpuri. De aceea, acesta are în spate o bază de date MongoDB utilă pentru a gestiona fragmente de date. Inițial, dorința era de a separa detaliile din profilul utilizatorilor de cele pentru autentificare. Se recomandă citirea subcapitolului *Setul de date* pentru a înțelege mai bine cum comunică cele două microservicii.

În prezent, *Profile Service* se ocupă de profilele utilizatorilor, putând verifica detaliile cu privire la propriul cont, dar și de a primi recomandări în funcție de profilele celorlalți. Șablonul de proiectare *strategie* este unul de tip comportamental ce permite definirea unei familii de algoritmi, astfel încât aceștia să fie puși în clase separate și obiectele lor să devină interschimbabile [27]. În aplicația curentă a fost folosit pentru a decide modul în care utilizatorii vor fi recomandați unei ținte. În prezent, sunt patru strategii de a decide acest aspect: căutări directe în lista utilizatorilor prin aplicarea paradigmei funcționale, celelalte strategii bazându-se pe algoritmul KNN și metrici ale acestuia.

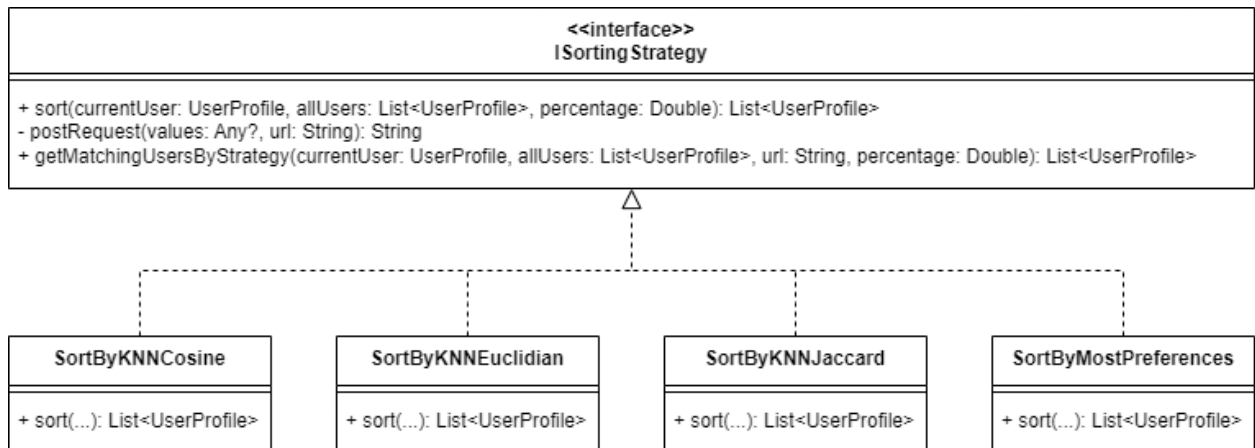


Figura I.4. Diagrama UML de clase pentru modelul strategie pentru sortări

Figura de mai sus pune în evidență structura claselor și a interfeței pentru șablonul de proiectare *Strategie* utilizat în cadrul aplicației. Așa cum se poate observa, există 4 implementări diferite ale aceleiași interfețe (*ISortingStrategy*). Metoda cea mai importantă este *sort*, folosită pentru a returna utilizatorii, unde se stabilește URI-ul către care să se facă apelul. În cadrul acestei metode, se apelează funcția *getMatchingUsersByStrategy*, care la rândul ei va adăuga în lanțul de apeluri și unul pentru *postRequest*, unde va avea loc obținerea efectivă a rezultatelor, pe baza serviciului ce ține de algoritmi. Similar, s-a folosit modelul acesta de proiectare și în cadrul microserviciului pentru algoritmi, diferența constând în faptul că se pune accent pe metricile utilizate.

I.2.4. Algorithms Service

Algorithms reprezintă punctul de acces pentru diverși algoritmi de inteligență artificială. Rolul lui este de a recomanda utilizatori în funcție de preferințele celui în cauză, și de preferințele celorlalți. Momentan, acesta se folosește de KNN și metricile cosine, euclidiană și Jaccard.

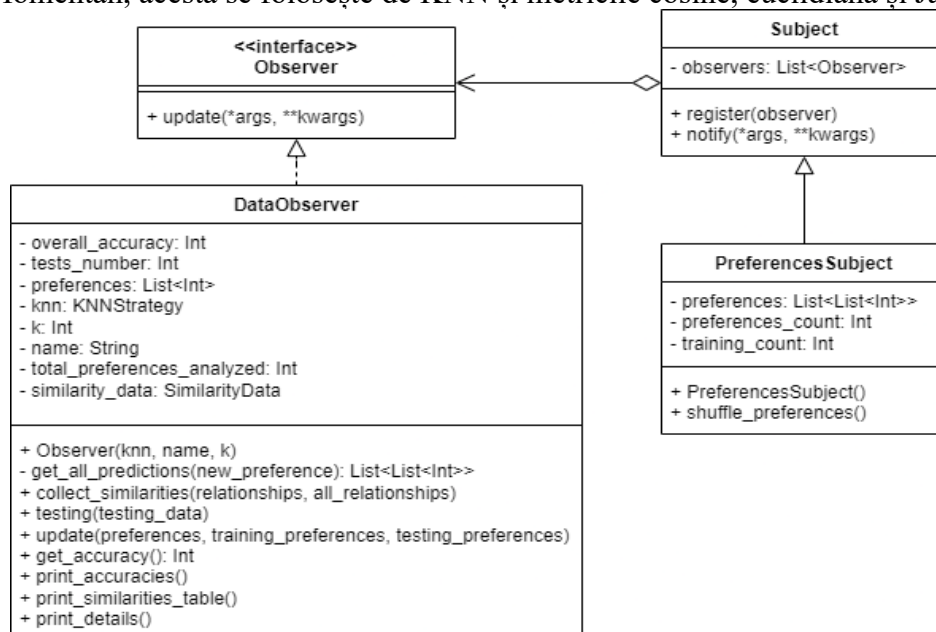


Figura I.5. Diagrama UML de clase pentru modelul observator, utilizate pentru colectarea datelor

Aici, pe lângă modelul *Strategie*, s-a folosit șablonul de proiectare *Observer*, de tip

comportamental, ce permite definirea unui mecanism de abonare pentru a notifica mai multe obiecte despre orice eveniment care se întâmplă obiectului pe care îl observă [28].

Există un subiect ce conține preferințele utilizatorilor, observatorii reprezintă cele trei variante ale lui KNN menționate, folosiți pentru a colecta datele. Ori de câte ori are loc operația de *shuffle* a listei de preferințe ale utilizatorilor din subiect, observatorii ce s-au abonat acestuia vor fi anunțați de schimbarea de preferințe, vor antrena din nou modelul KNN pe baza preferințelor de antrenare menționate de subiect, și vor testa acuratețea algoritmului, precum și măsura gradele de similaritate dintre țintă și restul de utilizatori tocmai antrenați. Acești observatori sunt folosiți în cele din urmă pentru a colecta datele și de a le transpune în fișiere. Pe baza acestor date se realizează în final și statisticile, precum și comparațiile dintre metricile utilizate, cu ajutorul librăriei matplotlib din Python.

Toate aceste servicii urmăresc respectarea principiilor SOLID, atât pentru clase, cât și microservicii, obținând astfel o granularitate fină. Fiecare are propriul proiect, server și bază de date sau interfață grafică unde e cazul, urmărindu-se decuplarea aplicației dezvoltate. Aplicația se folosește de *modelul client-server*, ceea ce înseamnă că există și un protocol de comunicare. Aici s-a preferat protocolul HTTP, tocmai pentru că procesul de dezvoltare devine facil, dar și din cauză că anumite cereri așteptau răspunsul pentru a finaliza operațiile. Cererile și răspunsurile dintre servicii se obțin prin REST API, fiecare dintre servicii folosindu-se de framework-ul Spring Boot, mai puțin Algorithms, unde s-a folosit Flask.

1.2.5. Setul de date

Bazele de date sunt folosite pentru a ține cont și memora detalii cu privire la utilizatori și tokenii folosiți. În cazul repornirii aplicației, e de așteptat ca utilizatorii și tokenii lor să rămână la fel ca ultima dată când a fost pornit server-ul. La nivelul serverelor au fost necesare crearea unor entități și tabele. Scopul folosirii acestor entități, alături de Spring JPA și Hibernate, precum și adnotările Spring Boot, este acela de a realiza într-un mod cât mai facil operații precum înregistrări de utilizatori noi. Bazele de date relaționale (SQL) sunt utile când se dorește ca datele să fie consistente, ceea ce se potrivește pentru persistarea datelor de autentificare ale utilizatorilor. Bazele de date nerelaționale (NoSQL) sunt utile că se dorește ca anumite câmpuri să fie opționale.

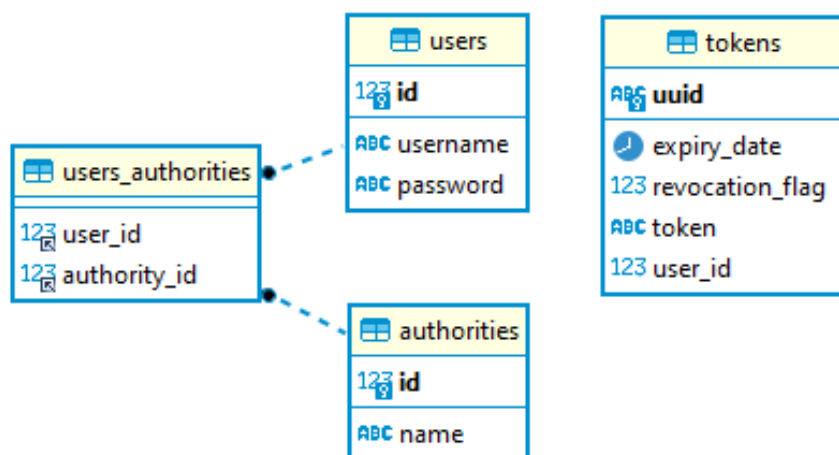


Figura 1.6. Diagrama Entity-Relation

Pe baza acestora, s-au realizat tabelele, care arată ca în figura alăturată (Figura 1.6). Fiecare utilizator are un username și parolă (câmpurile), dar și autorități și tokeni pe care i-a folosit (din celelalte tabele). Utilizatorul este folosit pentru autentificare, autoritățile pentru autorizare. Tokenii

parte dintre aceste detalii, dar sunt ținuti criptați aici, cât și la client.

Pentru a realiza acestea, au fost necesare dependențele Maven, configurarea Spring JPA Hibernate, Spring DataSource, precum și driver-ul MariaDB sau MongoDB, în fișierul de configurare în funcție de metoda de persistare aleasă. Apoi se pot crea interfețele de tip repositories cu adnotarea respectivă (Repository). Aici se vor crea funcții cu interogările SQL/NoSQL, folosind adnotarea Query. În cele din urmă, se realizează entitățile-tabel, cu rolul de a asocia diverse obiecte cu bazele de date. În cazul acestor entități trebuie incluse adnotările utilizate (Entity, Table). Aceste clase includ variabile ce asociază coloanele sau câmpurile din tabele.

Alte adnotări utilizate sunt Id, GeneratedValue, Column, Transient, ManyToMany, JoinTable. În spate, aceste adnotări ușurează procesul de creare al aplicației, dezvoltatorul nemaifiind obligat să includă secvențe de cod în plus.

Asocierea dintre utilizatorul simplu din IDM cu profilul din serviciul Profile este făcută prin stocarea câmpului idmId în interiorul profilului, atunci când este creat la înregistrare. Această asociere este importantă, pentru că, atunci când se creează un nou utilizator, este de așteptat ca și profilul lui să fie creat. Similar, atunci când utilizatorul vrea să își șteargă contul, trebuie să fie șters și contul lui din cealaltă bază de date. S-a adoptat acest model tocmai pentru a separa preocupările serviciilor.

```
_id: ObjectId('644a158836f4276994d6d770')
idmId: 84
firstName: "Victoria"
lastName: "Campbell"
email: "victoria.campbell@yahoo.com"
▼ preferences: Array
  0: "react"
  1: "javascript"
  2: "css"
  3: "html"
  4: "node.js"
_class: "com.project.profile.data.entities.UserProfile"
```

Figura I.7. Structura profilului utilizatorului

Așa cum se poate remarca în figura alăturată, câmpul idmId este cel care face legătura cu user_id din baza de date relațională MariaDB. Fiind o bază de date orientată document, MongoDB oferă facilitatea de a avea câmpurile parțial completate, pentru posibile situații în viitor în care utilizatorii vor fi nevoiți să includă alte câmpuri decât cele vizibile mai sus.

I.2.6. Web Application

Componenta *Aplicație Web* reprezintă punctul de acces de la nivelul clientului. Rolul componentei este de a oferi utilizatorului o experiență pozitivă pe site. S-a folosit Thymeleaf pentru a construi paginile web, precum și JavaScript pentru a realiza cereri către server, CSS pentru stilizarea paginilor și HTML pentru scheletul acestora. Mai multe detalii cu privire la interfața grafică pot fi găsite în subcapitolul *Funcționalitatea aplicației*.

Cap. II. Funcționalitatea și implementarea aplicației

În cazul utilizatorilor finali, există mai multe acțiuni ce pot fi realizate de aceștia. Pe lângă operațiile ce țin de gestiunea contului (register, login, logout), utilizatorii pot să-și vizualizeze contul atunci când sunt autentificați în aplicație, să manevreze pagina principală, și, cel mai important, să caute ceilalți utilizatori folosindu-se de criteriile de filtrare bazate pe algoritmi puși la dispoziție de aplicație. După această discuție despre funcționalitatea aplicației, în subcapitolul următor se va pătrunde în mai multe detalii cu privire la implementarea propriu-zisă a aplicației.

II.1. Funcționalitatea aplicației

La nivel minimalist a fost realizată și o interfață grafică cu utilizatorul. Aceasta ar putea fi dezvoltată și mai bine cu ajutorul unor framework-uri precum Angular și React, însă proiectul are ca puncte tari microserviciile și elementele de inteligență artificială. S-a decis că o interfață grafică, în acest stadiu, este frumos să existe, chiar dacă nu este necesară. Așadar, în scop demonstrativ, cu ajutorul Thymeleaf, HTML, CSS, JavaScript, a fost realizată și o interfață grafică la nivelul paginilor Web.

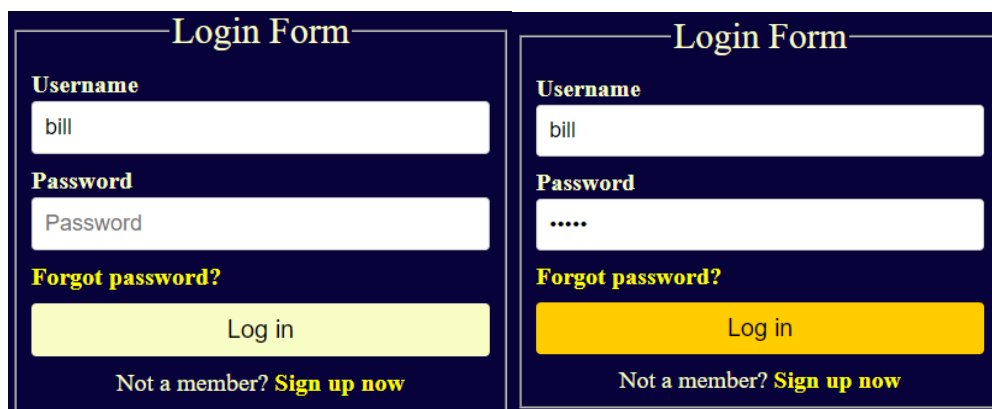


Figura II.1 Pagina pentru autentificare, cu un buton interactiv pe măsură ce se completează câmpurile

Așa cum se observă în Figura II.2, utilizatorii pot primi feedback cu paginile pe măsură ce completează din câmpuri. Similar s-a procedat și în cazul paginii de înregistrare. Pe lângă aceasta, modul de vizualizare este ținut la nivelul clientului, într-un cookie *darkThemeCookie*, ce poate lua valorile *light* și *dark*. Acest cookie este folosit la nivelul întregilor pagini din cadrul serviciului IDM, modul de vizualizare continuând să rămână așa cum a fost schimbat de către client. Aceste schimbări se produc prin apăsarea unor elemente de tip imagine sugestive precum cele de mai jos:



Figura II.2 Elemente folosite pentru comutarea modului de vizualizare, în dark mode (stânga) și light mode (dreapta)

Register Form

Username

 Required and must be unique

Password

 Passwords must be at least 3 characters long

Password confirm

 Passwords are required and must match

First name

 Required

Last name

 Required

Email

 Required

Preferences

 Must be at least 1 and at most 10

Register

Already member? [Login now](#)

Figura II.3 Pagina de înregistrare, în light mode (stânga) și dark mode (dreapta)

În figura de mai sus este prezentată pagina de înregistrare, realizată cu Thymeleaf. Așa cum se poate remarca, la nivelul clientului au fost realizate validări ale acestor câmpuri de înregistrare, toate acestea trebuind să fie completate atunci când un utilizator se înregistrează. Atunci când sunt corecte, vor fi marcate cu fundal verde, iar atunci când prezintă o selecție curentă, sunt marcate cu fundal albastru deschis. În caz contrar, acestea vor fi marcate de un fundal roșu sugestiv, care vor marca o invaliditate. Pe lângă un câmp necompletat, în cazul câtorva dintre acestea au fost necesare impunerea de constrângeri de tip regex (la username, caracterele să fie alfanumerice doar și cu ‘_’, ‘.’), precum și validări de lungime a caracterelor (username, de exemplu, trebuie să aibă măcar două caractere). Dar ce este foarte important la acest nivel este faptul că, pe măsură ce utilizatorii completează câmpurile username și email, se realizează cereri la Backend prin tehnica *debounce*, cereri prin care se verifică dacă câmpurile respective sunt valide (nu sunt luate de alți utilizatori). Această tehnică este folosită pentru a preveni funcționalități din a rula de prea multe ori, în cazul de față, request-uri prea multe atunci când utilizatorii completează câmpurile respective. Astfel, în loc să se realizeze request-uri pentru fiecare caracter apăsător, mulțumită tehnicii *debounce*, durează 0.25 secunde pentru a se trimite un request. Această durată se resetează dacă utilizatorul apasă alte taste mai rapid, iar în caz contrar, se va trata cererea respectivă.

Foarte relevant aplicației reprezintă punctul de acces *discovery*, din care se pot seta parametrii pentru strategia de căutare a utilizatorilor. Există un element de tip selecție și un input

de tip range, utilizat în procesul de votare pentru metricile algoritmului KNN, iar în cele din urmă se află și un buton *Find* pentru completarea cererii:

Figura II.4 Pagina Discovery

First Name	Last Name	Email	Preferences
Victoria	Campbell	victoria.campbell@yahoo.com	<ul style="list-style-type: none"> react javascript css html node.js
Robert	Williams	robert23@mail.com	<ul style="list-style-type: none"> c python html css javascript angular
Emily	Davis	emily04@mail.com	<ul style="list-style-type: none"> c++ python html css javascript react

Figura II.5 Rezultatele obținute (dreapta) unui utilizator cu anumite preferințe (stânga), pe baza parametrilor din Figura II.4

Se poate observa că, preferințele utilizatorilor din dreapta sunt chiar similare celui din stânga, deci se consideră că recomandarea a avut loc cu succes. Pentru mai multe detalii cu privire la testarea aplicației și a rezultatelor obținute, se recomandă parcurgerea capitoului legat de *Rezultatele aplicației*.

II.2. Implementarea aplicației

Aplicația a fost implementată folosind arhitectura de tip microservicii, tocmai pentru organizarea mai facilă a modulelor și claselor aplicației. Fiecare dintre microservicii include resurse necesare de tip imagini, CSS, JavaScript, HTML, baze de date, fișiere JSON pentru a stoca datele, după caz. S-a căutat ca aplicația să fie cât mai modularizată, să se respecte principii de

programare și protocoale folosite cât mai mult. În subcapitolele următoare se povestește despre fiecare dintre microservicii cum au fost implementate, despre rolul lor și motivația pentru care s-a ales acea implementare. În unele situații, se vor include și secvențe de cod pentru a înțelege mai bine despre ce este vorba.

II.2.1. Gateway

Gateway-ul, fiind un șablon arhitectural, este utilizat în aplicație pentru a realiza legătura dintre celelalte microservicii. Tot aici se sporește gradul de securitate al aplicației și, pe lângă acestea, se adaugă și o interfață la nivel minimalist.

Pentru început, Gateway-ul rulează pe adresa <http://localhost:8000/>. Dependentele Maven cele mai utilizate sunt Thymeleaf, REST, Spring Boot. Proiectul este împărțit pe pachete, printre care se pot considera *business*, *controllers*, *templates* și *css*.

În directorul *templates* sunt localizate fișierele HTML. Fișierul *error.html* este folosit pentru a ilustra în pagină anumite erori sau avertismente, pe când *user-list.html* este utilizat cu scopul de a include, la nivelul paginii și pe baza rezultatelor obținute cu ajutorul serviciului pentru algoritmi, lista de utilizatori afișată mai plăcut. Inițial, aceștia erau afișați în text brut, ca JSON, dar acum sunt afișați mai natural utilizatorilor finali. Pentru aceasta, a fost necesară folosirea lui Thymeleaf, pentru a crea o legătură între utilizatorii din Backend și Frontend:

```
<tr th:each="user : ${users}">
  <td th:text="${user.firstName}"></td>
  <td th:text="${user.lastName}"></td>
  <td th:text="${user.email}"></td>
  <td>
    <ul>
      <li th:each="preference : ${user.preferences}" th:text="${preference}"></li>
    </ul>
  </td>
</tr>
```

Mai există și paginile *discovery.html*, *index.html* și *profile.html*, însă acestea nu diferă cu mult față de pagina menționată mai sus, din punctul de vedere al implementării, dar ca scop, acestea reprezintă tab-urile din cadrul unui navbar, cu nume sugestive.

Pe lângă aceste pagini, s-au utilizat fișiere CSS, localizate în directorul *css*. Foarte pe scurt, fiecare dintre paginile HTML menționate mai sus se folosește de câte un fișier CSS, cu nume identic și extensie diferită. Tot ce trebuie făcut pentru a utiliza aceste fișiere este să fie incluse în paginile HTML, ca mai jos:

```
<head>
  <title>User List</title>
  <link rel="stylesheet" th:href="@{/css/user-list.css}" />
</head>
```

Acum că s-au prezentat detaliile cu privire la Frontend-ul aplicației și la modul în care acesta își face apariția la nivelul paginilor Web, se poate trece la aplicația propriu-zisă și la funcționalitățile de tip business-logic.

În primul rând, fiind o aplicație de tip Spring Boot, este necesar să se adnoteze aplicația, încât să se poată ceda controlul framework-ului (are loc *Dependency Inversion* aici). În fișierul *GatewayApplication.kt* au fost incluse următoarele, astfel încât să se realizeze asocierea menționată:

```
@SpringBootApplication
open class GatewayApplication
```

```
fun main(args: Array<String>) {
    runApplication<GatewayApplication>(*args)
}
```

În cadrul pachetului *business*, au fost create alte două subpachete, respectiv *interfaces* și *services*, tocmai din dorința de a respecta principiile SOLID, aici fiind vorba despre *Interface Segregation*, ce afirmă că este mai bine să existe mai multe interfețe specifice decât una generală. De exemplu, serviciul *AuthorizationService* implementează interfața *IAuthorizationService* ce are ca metodă funcția *authorize*. Similar, și celelalte servicii implementează interfețe, după aceeași convenție de notare ('I' la început, pentru a marca o interfață).

Serviciul *JSONOperationsService* este folosit de *TokenService* și *AuthorizationService*. În primul rând, este necesară adnotarea Spring *@Service*, la nivelul serviciului. Acest serviciu are ca scop obținerea câmpurilor din cadrul unor String-uri (acestea ar reprezenta un format JSON, care trebuie totuși verificat dacă este valid sau nu, returnând mesaje corespunzătoare). Mai relevant în cadrul acestui serviciu sunt funcțiile cu nume sugestive *getAuthoritiesFromJSONString* și *getIdFromJSONString*. În cazul amândurora dintre aceste funcții, se creează mai întâi un element de tip *Gson*, apoi, pe baza acestuia, un *JsonObject*, din care se pot extrage elementele de interes (în cazul Gateway-ului, sunt ID-ul și autoritățile). În cazul unor excepții, se returnează un ID invalid (-1) și o listă de autorități goală. Mai jos se poate regăsi modul de obținere al acestor detalii, în cazul autorităților:

```
val gson: Gson = GsonBuilder().setPrettyPrinting().create()
val jsonObject: JsonObject = gson.fromJson(stringToTransform, JsonObject::class.java)

val jsonElement: JsonElement = jsonObject.getAsJsonArray("authorities")
val authoritiesArray: Array<String> = gson.fromJson(jsonElement, Array<String>::class.java)

val authoritiesList: List<String> = authoritiesArray.asList()
return authoritiesList
```

Un alt serviciu de care depind *TokenService* și *AuthorizationService* este *UrlCallerService*. Așa cum indică și numele, este folosit pentru a realiza cereri între celelalte servicii. Atunci când se dorește apelarea celorlalte servicii, trebuie inclus și un token, care trebuie validat și verificat. Răspunsul unui astfel de apel se obține similar codului de mai jos:

```
val connection = URL(url).openConnection() as HttpURLConnection
connection.requestMethod = "GET"
connection.setRequestProperty("Cookie", "cookieAuthorizationToken=$token")

return getResponseBody(connection)
```

Funcția *getResponseBody* presupune returnarea unei resurse cerute, atunci când serviciul funcționează și poate fi apelat, iar resursa cerută există și se poate obține, pe baza unei conexiuni de tip *HttpURLConnection*.

Mai departe, *TokenService* are ca scop primordial obținerea ID-ului unui utilizator, bazat pe un token criptat. Pentru aceasta, în funcția *getUserIdFromEncryptedToken*, se realizează un request către <http://localhost:8001/verify-token>, folosind *IUrlCallerService*, tocmai în scopul transforării din token-ul criptat într-unul decriptat. Apoi, folosind structura token-ului, se poate extrage din Payload ID-ul utilizatorului. În cazul în care apare vreun neprevăzut, similar serviciului anterior, se returnează ID-ul -1.

În cele din urmă, *AuthorizationService*, are o singură funcție importantă, aceea fiind de a

autoriza utilizatorul. Aici există mai multe scenarii ce trebuiau luate în calcul, în fiecare dintre acestea considerându-se și returnarea unui status HTTP elocvent. Prima situație este cea în care utilizatorul reușește să ajungă în acest endpoint fără să aibă un token, caz în care trebuie redirectat către <http://localhost:8000/login>, cu statusul *FOUND*. Apoi, s-a luat în considerare situația în care formatul token-ului nu este respectat și, de exemplu, prima parte din token nu este „*Bearer:*”, caz în care s-a returnat codul de eroare *NOT_ACCEPTABLE*. Mai există apoi situația bună, în care utilizatorul are autoritatea necesară aplicației, primind ca status *OK*. În cazul în care acesta nu a reușit să primească acces, va primi în schimb statusul *METHOD_NOT_ALLOWED*, iar în cazul în care apar totuși alte erori ce țin de server, se returnează statusul *BAD_REQUEST*. Din dorința de a respecta cât mai mult standardul REST s-a încercat returnarea statusurilor cât mai potrivite, în funcție și de ceea ce s-a întâmplat. Similar trebuie dezvoltate și celelalte servicii, astfel încât să se returneze codurile HTTP potrivite. Acest standard ajută la o mai bună înțelegere a erorilor întâmpinate și a modului în care au avut loc.

Acum se va discuta câte ceva și despre clasele ce reprezintă câte un controller. Există un controller *HomeController*. Aici se prezintă, foarte succint, modalitatea în care sunt realizate și celelalte endpoint-uri ale aplicației. În primul rând, trebuie ca acestea să poată fi accesate, apoi, la nivelul acestei rute se stabilesc autoritățile de acces permise (aici fiind permiși cei ce au drepturile de *read* și *write*). Pe baza acestora, se decide dacă utilizatorul, pe baza unui token prezentat când accesează ruta, are voie (i se va returna pagina) sau nu (va fi redirectat) în cadrul paginii. Aceasta este o modalitate de a realiza cele menționate, iar celelalte servicii procedează în mod similar:

```
@GetMapping("/home", "/", "index")
fun home (
    @RequestHeader(value = "Authorization", required = false, defaultValue =
    "") bearerJws: String,
    @CookieValue(value = "cookieAuthorizationToken", defaultValue = "") cook-
    ieJws: String,
): Any {

    val allowedAuthorities = listOf("read", "write")
    val response = authorizationService.authorize(allowedAuthorities, bear-
    erJws, cookieJws)

    return if (response.statusCode.is2xxSuccessful) {
        ModelAndView("index")
    } else {
        RedirectView("/logout")
    }
}
```

Controller-ul *Discovery* expune două rute, mai precis <http://localhost:8000/discovery>, din care utilizatorul va putea să-și aleagă parametrii pentru algoritmi de căutare a celorlalți utilizatori în funcție de preferințele lor, și cea de a doua rută, pentru a primi utilizatorii găsiți, <http://localhost:8000/discovery/users?strategy=<strategy>&percentage=<percentage>>. Mai întâi se realizează autorizarea, apoi se face un apel către serviciul pentru profiluri, localizat pe portul 8002, de unde se vor obține mai întâi utilizatorii, iar apoi va avea loc conversația lor încât să aibă loc o afișare prietenoasă cu utilizatorii. Această transformare se realizează prin intermediul unui *ObjectMapper*.

ProfilesController expune un singur endpoint din cadrul serviciului pentru profiluri, în scop demonstrativ. Este vorba despre operația de vizualizare a contului propriu, accesându-se <http://localhost:8002/profile/<userId>>, și, la sfârșit, realizându-se operația de transformare a

obiectului, încât să fie mai ușor de urmărit afișarea utilizatorului.

Cu toate că serviciul IDM există deja, s-a considerat facilitarea accesului utilizatorilor, încât aceștia, în cazul în care încearcă să se autentifice pe portul greșit, să fie totuși redirecțați în locul potrivit. Cu asta se ocupă *IDMController*. Se realizează redirecțări atunci când nu este autentificat (login) sau deja este (home). Atunci când părăsește contul, are loc, pe lângă o redirecțare și o invalidare a token-ului, încât acesta să nu mai poată fi folosit ulterior. Iar în cazul în care utilizatorul vrea să se înregistreze, ajunge redirecțat către serviciul IDM, de unde va putea realiza operațiunile necesare.

Deoarece Gateway-ul reprezintă liantul dintre servicii, fără acesta, multe dintre celelalte servicii nu ar mai funcționa corespunzător. Singurul microserviciu ce va putea să-și continue ciclul de viață fără ca Gateway-ul să fie pornit este IDM.

II.2.2. Identity Management

Microserviciul pentru gestiunea identităților utilizatorilor este unul dintre cele mai complexe ale acestei aplicații, deoarece aici se realizează securizarea datelor. Acesta rulează pe portul 8001, folosește o conexiune cu bazele de date prin *Spring DataSource* și *Hibernate* și utilizează secrete.

Similar *Gateway-ului*, aplicația server a fost împărțită pe pachete. În cadrul directorului cu resurse, există diverse fișiere HTML folosite pentru a randa paginile. Similar, s-a folosit framework-ul Thymeleaf. Singura diferență față de Gateway, este că au fost necesare, pe lângă importarea fișierelor CSS, și fișierele JavaScript și imagini. În unele cazuri, erau esențiale și formularele. Toate acestea se realizează conform sintaxei HTML de mai jos:

```
<script type="module" th:src="@{/js/Cookie.js}"></script>

<form th:action="@{/login}" method="post">
```

Au fost necesare template-uri (fișierele HTML) pentru login, logout, register și operații de succes, insucces. În cadrul fișierelor pentru funcționalitatea paginilor Web de la nivelul clientului, s-au despărțit fișierele încât acestea să poată fi reutilizate unde este necesar. De exemplu, *login.js* importă *Theme* și *Cookie* din *Theme.js* și *Cookie.js*. Similar, și celelalte fișiere pentru logout și înregistrare importă cele două fișiere. Fișierele acestea de tip JavaScript s-au folosit pentru a menține, la nivelul clientului, cookie-uri, de a randa paginile și pentru a crea pagini cât mai interactive cu utilizatorii.

Mai multe teme adaugă complexitate aplicației, însă vin la pachet și cu o experiență mai plăcută utilizatorilor, prin faptul că li se oferă accesibilitate. Există o temă *light* și una *dark*. Acestea pot fi comutate apăsând butoanele simbolice (soare și lună). Atunci când utilizatorii comută, background-ul, textul și butoanele își schimbă tematica. Fiecare pagină are elemente specifice care își modifică tema prin intermediul cookie-ului *darkThemeCookie*, mai precis, se folosesc *eventListeners*.

Fișierul *login.js* se asigură că utilizatorii nu lasă câmpuri necompletate atunci când se autentifică în cadrul aplicației. În schimb, fișierul *register.js* face verificările câmpurilor utilizatorilor. Pe lângă un format ce trebuie respectat, de exemplu să nu fie goale, să aibă un anumit număr de caractere sau să respecte un regex, s-a folosit tehnica *debouncing*, pentru a preveni request-uri prea multe din cadrul aplicației. Această tehnică presupune existența unei durate. De exemplu, 0.25 secunde. Atunci când utilizatorul își tastează mail-ul, se pornește un "timer", care verifică dacă s-a ajuns la 0.25 secunde. Ori de câte ori tastează, timer-ul se resetează. Atunci când timer-ul a ajuns la 0.25 de secunde, se realizează un request către server-ul Backend pentru o verificare în plus a posibilității de utilizare a mail-ului respectiv. Mai jos a fost inclusă o secvență

de cod, pentru a înțelege mai bine cum funcționează această tehnică:

```
static checkEmailBackend(email) {
  if (!email) return Promise.resolve(false);
  if (Register.debounceTimer) clearTimeout(Register.debounceTimer);

  // Return a Promise that resolves with a boolean value
  return new Promise((resolve, reject) => {
    Register.debounceTimer = setTimeout(() => {
      fetch(`http://localhost:8002/profile/check-email/${email}`)
        .then(response => response.json())
        .then(data => {
          resolve(data["available"]);
        })
        .catch(error => {
          console.error(error);
          reject(error);
        });
    }, 250);
  });
}
```

În principiu, <http://localhost:8002/profile/check-email/email> este folosită pentru a verifica dacă anumite email-uri sunt sau nu utilizate de alți utilizatori. Mai multe detalii despre operațiunile pentru profilurile utilizatorilor se găsesc mai jos, în *Profile Service*. Mai există totuși un fișier important, anume *preferences.js*, ce are ca scop introducerea preferințelor de către utilizatori atunci când se înregistrează. Momentan, utilizatorii pot doar să adauge preferințele, fără a modifica ordinea lor, fiind nevoiți să reia formularele dacă greșesc, însă acestea pot fi implementate similar funcționalității de adăugare. Atunci când utilizatorii apasă ‘,’ sau enter, se adaugă un nou element.

Trecând de la Frontend la Backend, trebuie menționate faptul că sunt necesare anumite configurări, în special cele referitoare la Spring Security și utilizatorii aplicației. Pentru început, sunt necesare lanțurile de filtre (filter chains). Este nevoie de adnotările *@Configuration* și *@EnableWebSecurity* (odată ce pornește server-ul, va realiza aceste configurări înainte). Lanțul de filtre utilizat în cadrul aplicației este:

```
@Bean
@Throws(Exception::class)
fun securityFilterChain(http: HttpSecurity): SecurityFilterChain {

    return http

        .authorizeHttpRequests()
            .requestMatchers("/invalidate-token").access(WebExpressionAuthorizationManager("hasIpAddress('localhost')"))
            .requestMatchers("/verify-token").access(WebExpressionAuthorizationManager("hasIpAddress('localhost')"))

            .requestMatchers("/register-temporary").permitAll()
            .requestMatchers("/check-username/**").permitAll()
            .requestMatchers("/css/**", "/js/**", "/images/**").permitAll()

            .requestMatchers("/register").permitAll()
            .requestMatchers("/**").hasAnyAuthority("read", "write")
}
```

```

        .anyRequest().authenticated()
        .and()

        .formLogin()
        .loginPage("/login")
        .defaultSuccessUrl("/")
        .permitAll()
        .and()

        .logout()
        .logoutSuccessUrl("/login")
        .permitAll()
        .and()

        .build()
    }

```

Așa cum se poate remarca, *.requestMatchers* este folosit pentru a decide care dintre cereri să fie validate și modul în care să aibă loc autorizarea lor. Foarte important de precizat este că aceste lanțuri de filtre se execută de la specific la general, dezvoltatorii fiind nevoiți mai întâi să includă resursele ce trebuie validate (*.requestMatchers*) și abia apoi cazurile generale (*.anyRequest*). În unele cazuri, este necesar ca apelurile să fie făcute de către localhost (*.access(WebExpressionAuthorizationManager("hasIpAddress('localhost')"))*), în altele să aibă acces oricine (*.permitAll*), să aibă o anumită autoritate dintr-o listă de autorități (*.hasAnyAuthority*) sau doar să fie autentificat (*.authenticated*). Spring Security pune la dispoziție formulare simple, dar dacă se doresc implementări proprii, trebuie folosite *.formLogin* și *.logout*. Separările dintre aceste filtre se realizează prin *.and*, iar la sfârșit trebuie doar să aibă loc *.build*. Pe lângă aceste filter chains, s-a creat și un *@Bean* pentru codificarea parolelor, unde s-a utilizat *BCryptPasswordEncoder*. Clasele *SecurityAuthority* și *SecurityUser* reprezintă niște wrappers pentru *GrantedAuthority*, respectiv *UserDetails*. Acestea sunt utilizate pentru a ajuta Spring Security să înțeleagă că, prin intermediul acestor obiecte vor avea loc verificările de autorități.

Pentru o dezvoltare mai facilă a aplicației, s-a utilizat conceptul de ORM (Object Relational Mapping), prin intermediul Spring Data JPA, un standard specific aplicațiilor Enterprise ce oferă posibilitatea de a mapa numele claselor-entități la nume de tabele SQL. În felul acesta, se pot asocia ușor și rapid obiectele cu tabelele. Astfel, utilizând adnotările specifice (*@Entity*, *@Table*) s-au creat trei entități principale: *User*, *Authority*, *Token*. Toate aceste entități includ date din tabelele pe care le asociază. Alte adnotări importante sunt *@Id*, *@GeneratedValue*, pentru a decide cum sunt folosite id-urile din tabele (de exemplu, în funcție de identitate, cu *strategy = GenerationType.IDENTITY*). *@Column* contribuie la asocierea unor coloane din tabel cu alte nume de attribute decât cel din tabel, precum și de a decide dacă, de exemplu, sunt unice sau pot avea valori nule. *@Transient* este utilizat pentru a exclude un câmp (de exemplu, parola de confirmare). În cazul aplicației, tabelele *users* și *authorities* au tabela comună *users_authorities*. O modalitate de a realiza această mapare bidirecțională este de a utiliza adnotarea *@ManyToMany*:

```

// User Entity
@ManyToMany(fetch = FetchType.EAGER, cascade = [CascadeType.ALL])
@JoinTable(
    name = "users_authorities",
    joinColumns = [JoinColumn(name = "user_id")],
    inverseJoinColumns = [JoinColumn(name = "authority_id")]
)
private var authorities: Set<Authority> = setOf()

```

```
// Authority Entity
@ManyToMany(mappedBy = "authorities")
private var users: Set<User> = setOf()
```

Cascade reprezintă modul în care entitățile depind între ele și ce se întâmplă atunci când unele dintre ele dispar, de exemplu, să fie ștearsă și cealaltă entitate. JPA pune la dispoziție operații precum PERSIST, MERGE, REMOVE, REFRESH, DETACH și ALL. *Fetch* dispune de două strategii: EAGER (“obține-l încât să-l ai la tine”) și LAZY (“obține-l când ai nevoie de el”) [29].

În ceea ce privește persistența datelor, utilizatorii, autoritățile, precum și tokenii au propriul lor repository (depozit), folosind *JpaRepository*. Aici se pot performa operații pentru bazele de date, utilizând adnotarea *@Query* și incluzând operația dorită. De exemplu, dacă se dorește găsirea unei autorități după numele acesteia, o modalitate de a realiza aceasta folosind sintaxa JPQL este:

```
@Query("select a from Authority a where a.name = :name") // JPQL syntax
fun findAuthorityByName(name: String): Authority
```

În ceea ce privește serviciile aplicației, a fost necesară implementarea interfeței *UserDetailsService*, ce reprezintă contractul Spring Security, prin care Spring înțelege cum sunt transmise datele de utilizator. În felul acesta, nu va mai fi nevoie de implementarea clasică ce implica folosirea unui utilizator cu o parolă temporară, ci aceștia se vor încărca odată cu pornirea server-ului, putând folosi propriile nume de utilizatori și parole.

Securizarea aplicației se realizează folosind *CryptographyService*. Operația de criptare constă în crearea unui cifru bazat pe un secret și folosirea algoritmului de criptare simetrică AES (Advanced Encryption Standard), cu ajutorul căruia se obține valoarea criptată. Similar se procedează și pentru decriptare, obținându-se în final valoarea decriptată.

Cu ajutorul *Auth0* și a serviciului menționat anterior, *TokenService* se ocupă de generarea criptată a tokenilor, și a validității acestora. Validarea tokenilor constă într-un set de pași complecși. Trebuie să aibă formatul corect (adică să fie în total trei elemente în urma operației de despărțire după virgulă), să aibă semnătura validată, să fie în termenul valabil de utilizare (după câmpul *expiry*), să fie exclus de pe lista neagră (în tabela *tokens* există un câmp *revocation_flag*), să nu fie modificat de utilizator (în urma găsirii aceluiși token în baza de date după uuid și a comparării cu token-ul în cauză) și să nu apară alte excepții în cadrul aplicației când se verifică validitatea token-ului. Dacă s-a trecut de toate acestea, abia acum se poate considera că token-ul este valid. Pe lângă acestea, există și un o funcție pentru invalidarea tokenilor, util pentru situațiile în care utilizatorii performează operația de logout aceștia nemaiputând utiliza același token pe care îl aveau înainte.

Atunci când se dorește verificarea anumitor câmpuri, este de bună practică să existe validări atât la nivel de Frontend, cât și Backend. În scop demonstrativ, pentru un utilizator care se înregistrează, s-au realizat câteva astfel de validări la nivelul serviciului *UserValidatorService*. Tot aici trebuie să se verifice dacă un nume de utilizator mai este valid, realizând request-uri către serviciul pentru Profile-uri. Pentru aceasta, s-a folosit un DTO pentru User, astfel încât acest obiect să poată fi folosit drept liantul dintre servere din cadrul operației de înregistrare a unui nou utilizator. *UserDTO* cuprinde atât detaliile referitoare la contul pentru *IDM*, cât și a câmpurilor utilizate de *Profiles*, în special lista *preferences*.

Unul dintre controllerele implementate este *TokenController*. El este utilizat doar pentru verificarea tokenilor, precum și invalidarea acestora. *UserValidationController* are ca scop verificarea în timp reală a valabilității câmpului *username*, dar similar se procedează și pentru câmpul *email*. Cel mai important în schimb este *IdentityManagementController*. Atunci când utilizatorul este deja autentificat, s-a considerat că ar putea fi redirectat către pagina principală din Gateway. S-au realizat rute pentru login, logout și pentru înregistrare, unde trebuie verificată mai întâi valabilitatea utilizatorului (prin intermediul serviciului *Profiles*) și, la sfârșit, să se anunțe și

serviciul *Algorithms* de existența noului utilizator.

Așadar, fără de acest microserviciu, niciunul dintre celelalte nu ar rula cu succes. Acesta este chiar necesar aplicației, și nu doar mulțumită operațiunilor de securizare utilizate, ci și a dependențelor celorlalte servicii.

II.2.3. *Profile Service*

În cadrul acestui microserviciu, ce se află pe portul 8002, s-a utilizat o bază de date de tip MongoDB, cu ajutorul lui Spring Data. Pe lângă acestea, a fost necesară și utilizarea lui Spring Security, încât anumite dintre rutele aplicației să poată fi accesate doar de către celelalte servere.

ProfilesController utilizează *ProfileRepository* pentru operații de verificare de mail, creare a unui profil nou și vizualizarea acestuia. În cadrul acestor rute s-a ținut cont ca statusurile HTTP returnate să fie corecte. De exemplu, FORBIDDEN are loc atunci când un utilizator are un ID invalid; NOT_FOUND când este de negăsit; FOUND când a fost găsită cu succes resursa; CREATED când utilizatorul a fost creat cu succes; CONFLICT când un utilizator cu acel mail există deja; INTERNAL_SERVER_ERROR când are loc o eroare neașteptată; OK când verificarea unui mail s-a finalizat cu succes.

Mai există și *RecommendationController*. Aici s-a utilizat șablonul de proiectare *Strategie*. Pe baza variabilei de tip *cale* din cadrul cererii (*userId*) și a parametrilor cererii (*strategy* și *percentage*), se va realiza recomandarea utilizatorilor. Se alege strategia, apoi se realizează cererea către serviciul *Algorithms*, prin intermediul căruia se obțin utilizatorii recomandați în urma algoritmului selectat. Logica aceasta este inclusă și în secvența de mai jos, unde *allUsers* reprezintă toate profilurile utilizatorilor, iar url semnifică link-ul către care să se realizeze cererea (acesta cuprinde și strategia) de la nivelul serviciului *Algorithms*:

```
val targetPreferences = currentUser.getPreferences()
val targetIndex = allUsers.indexOf(currentUser)
val knnPercentage = percentage

val jsonMap = mutableMapOf<String, Any?>()
jsonMap.put("target_preferences", targetPreferences)
jsonMap.put("target_index", targetIndex)
jsonMap.put("knn_percentage", knnPercentage)

// receive response
val response = postRequest(jsonMap, url) // [4 5 6 0 1]

val resultingUsers = response
    .removePrefix("[")
    .removeSuffix("]")
    .split(" ")
    .map { allUsers[it.toInt()] }

return resultingUsers
```

În cazul în care acest microserviciu ar avea probleme sau nu ar mai putea fi utilizat, singurele ce ar fi afectate sunt doar IDM și *Algorithms*. Utilizatorul nu își va mai putea vizualiza detaliile cu privire la cont, să primească recomandările de utilizatori și nici nu se vor mai putea înregistra alți noi utilizatori. Fără acest serviciu, ar mai fi posibilă doar autentificarea, operația de logout și manevrarea câtorva pagini din Gateway ce nu depind de serviciul acesta.

II.2.4. Algorithms Service

Inițial, server-ul folosit pentru algoritmi de recomandare utiliza librăria scikitlearn, dar acum implementările algoritmului KNN sunt realizate doar cu ajutorul funcțiilor matematice, paradigmei orientate obiect și facilităților oferite de Python pentru lucrul mai simplu cu listele. Mai întâi, se discută despre modul în care server-ul acesta comunică cu serviciul pentru profilurile utilizatorilor și pentru ce anume este folosit, urmând ca apoi să se discute despre experimentele realizate și rezultatele obținute, tot cu ajutorul lui Python.

Pentru început, trebuie pornit server-ul acesta, din fișierul *main.py*. Odată ce se pornește server-ul, se vor citi preferințele utilizatorilor ca string-uri din fișierul *knn/stored_preferences.txt*. Acest fișier cuprinde doar o listă de liste de cuvinte, ce reprezintă preferințele utilizatorilor, în ordinea indexului acestora. Apoi, pe baza acelor string-uri, se realizează operația de mapare (transformare) în numere a acestor string-uri, obținând o listă de liste de valori de tip *int*. În acest fișier există trei obiecte de tip KNN, fiecare reprezentând câte o variantă de implementare diferită a algoritmului, conform șablonului de proiectare *Strategie*. Urmează ca toți acești algoritmi ce trebuie testați să realizeze antrenarea datelor, în funcție de listele numerice obținute. Antrenarea (fit) modelului KNN constă în: obținerea punctelor, unui set de preferințe unice și în final obținerea punctelor binarizate (cu ajutorul preferințelor numerice și a numărului total de preferințe unice, aflat prin intermediul setului de preferințe unice). După toate aceste inițializări, server-ul pornește pe port-ul 5000, cu adresa *localhost*. Mai departe, server-ul expune două rute principale, apelate doar de către celelalte microservicii:

```
@app.route("/algorithms/register", methods=["POST"])
@app.route("/algorithms/knn/<strategy>", methods=["POST"])
```

În cazul primei rute, atunci când un utilizator se înregistrează în aplicație, apar noi seturi de preferințe, și este posibil ca acele preferințe să conțină preferințe noi pe care implementările algoritmului KNN încă nu le cunoaște, din cauză că nu a fost antrenat să folosească și acele preferințe.

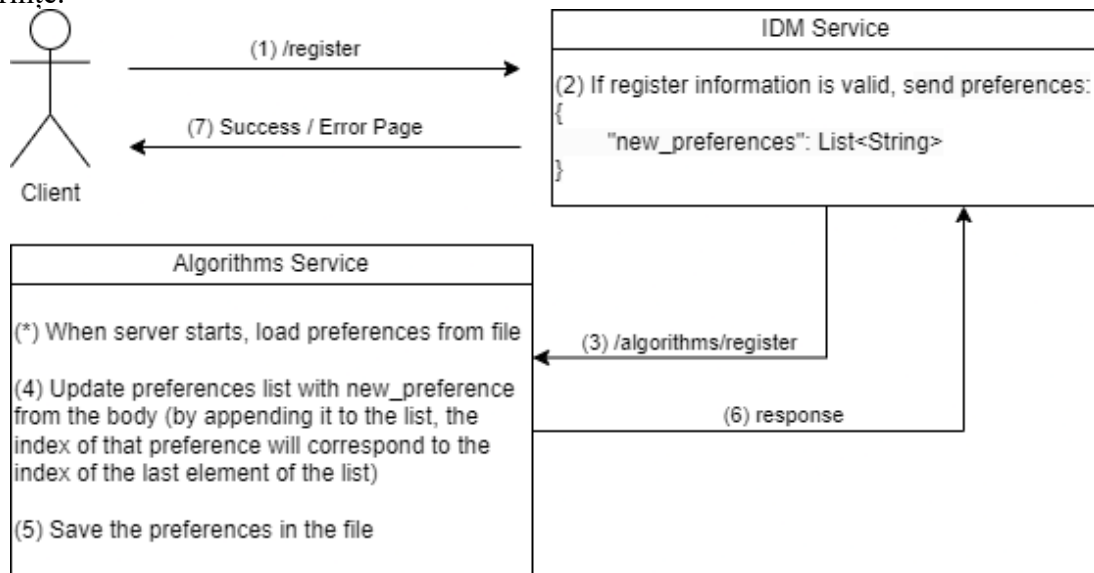


Figura II.6. Interacțiunea dintre client și servere atunci când are loc înregistrarea

Ori de câte ori cineva se înregistrează folosind serviciul IDM, se realizează un callback către <http://localhost:5000/algorithms/register>, punând în body detaliile despre preferințele utilizatorului nou creat (Figura II.6). Odată ajuns la *Algorithms*, se decodifică datele din body. Aceste date reprezintă un set nou de preferințe ca string-uri, ce trebuie transformate în *int*-uri. Apoi,

algoritmii trebuie reantrenați pe baza noii preferințe adăugate. Tot odată, aceasta trebuie să fie salvată și în fișier, ca ori de câte ori se pornește server-ul, aceasta să se regăsească în *knn/stored_preferences.txt*.

Cea de a doua rută este cea mai importantă. În funcție de strategia aleasă din *<strategy>*, și a body-ului din care se extrag preferințele țintei, indexul acesteia și procentajul de vot, se decide algoritmul ce va fi folosit și modalitatea în care acesta va funcționa. Mai întâi, pe baza preferințelor transformate în date numerice și a indexului țintei, se obțin *predicțiile*. Contează foarte mult variantele de implementare ale lui KNN, deoarece, în unele cazuri, predicțiile se realizează pe baza similarităților, în alte situații pe baza distanțelor. Similaritățile mari dintre utilizatori conduc la asemănări mari dintre aceștia, dar distanțele mari dintre ei conduc la deosebiri mai mari. Totuși, se obțin mai întâi relațiile dintre punctul țintă și celelalte puncte. Această operație constă în verificarea unei situații posibile de reantrenare (noi preferințe), binarizarea preferințelor țintei și construirea relațiilor față de celelalte puncte, ignorându-se punctul cu indexul țintei (pentru a nu primi recomandare pe sine însuși), și adăugându-se într-o listă relația dintre punctul iterat curent și țintă, precum și indexul iterației curente. Relația dintre punctul iterat curent și țintă se calculează pe baza metricii utilizate, bazate pe punctele binarizate. Mai jos a fost inclusă secvența de cod pentru a sumariza cele spuse anterior.

```
# get relationships for that prediction
relationships = []
for index, binarized_point in enumerate(self.binarized_points):
    if index == target_index: continue # to skip getting the same user
    relationship = self.metric(binarized_point, binarized_target_preferences)
    relationships.append([relationship, index])
```

Metrica reprezintă doar distanța sau similaritatea dintre două puncte binarizate. Spre exemplu, în cazul metricii euclidiene, aceasta se realizează astfel:

```
def metric(self, u, v):
    return np.sqrt(np.sum((np.array(u) - np.array(v)) ** 2))
```

În cele din urmă, pe baza predicțiilor obținute se mai realizează o filtrare, și anume cea pentru procentul de votare, folosind limitatorul procentual din cadrul body-ului din request. Practic, prin acel limitator procentual se păstrează doar cei mai similari utilizatori cu ținta, excluzându-l pe sine.

```
def get_best_x_percentage_best_predictions(target_preference, k_predicted_preferences, percentage=0.7):
    intersections = []
    for index, preference in enumerate(k_predicted_preferences):
        intersection = list(set(preference) & set(target_preference))
        intersections.append((index, intersection))

    total_procentually = ceil(int(percentage * len(intersections)))
    intersections.sort(key=lambda x: -len(x[1]))
    intersections = intersections[:total_procentually]

    # !!! careful at indices, those are indexed from 0 to n!!! meaning that
    # you'll have to make sure
    # the indices are consistent too when working with the values!!!
    intersections = list(map(lambda x: k_predicted_preferences[x[0]], intersections))
    return intersections
```

În principiu, este nevoie de o listă pentru intersecții, în care se adaugă tuple cu structura (*index, intersecție*). Apoi, pe baza limitatorului procentual, se decide câți dintre utilizatorii preziși

se vor păstra (cu *total_procentually*). În final, de interes sunt indecșii, de aceea se realizează o transformare a intersecțiilor.

Astfel, în urma celor menționate, se obțin utilizatorii cei mai similari unei ținte. Valoarea lui *k* este setată în aplicație la valoarea 11, aceasta putând fi modificată de aici momentan. Valoarea lui *k* a fost luată prin încercări, din 2 în 2, pentru valori impare. În situațiile în care există un număr par de clase, există posibilitatea ca, pe baza unui număr de valori *k* cei mai apropiați să apară cazul de egalitate, dar asta doar atunci când valorile lui *k* sunt pare. De aceea, se preferă utilizarea unei valori impare pentru *k*.

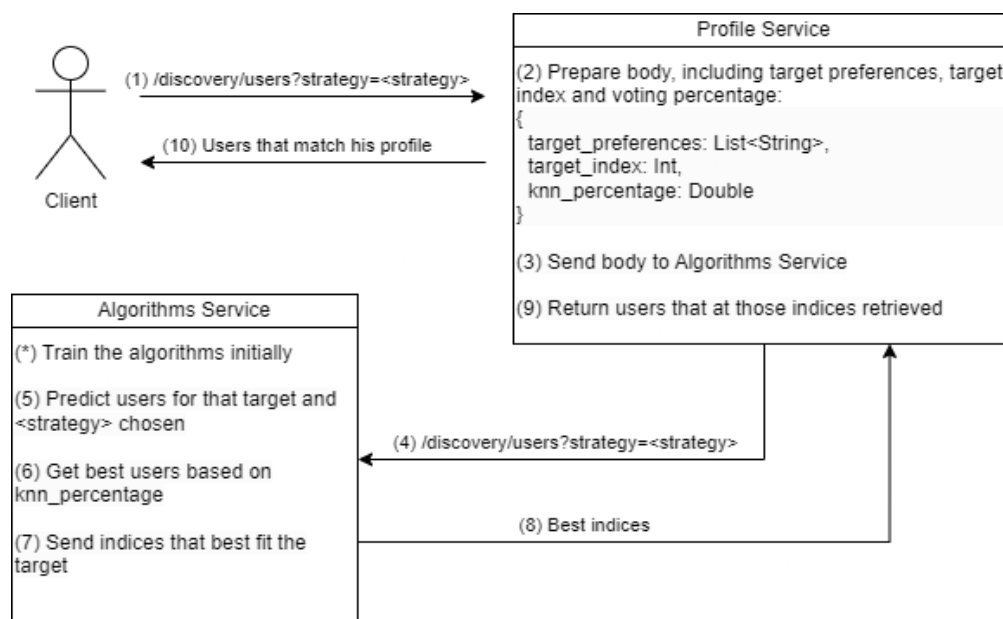


Figura II.7. Interacțiunea dintre client și servere folosind algoritmul KNN

Pentru a urmări mai ușor arhitectura aplicației în ceea ce privește algoritmul KNN, s-a realizat diagrama de mai sus, începând cu utilizatorul și preferințele lui, trecând prin lanțul de servicii în care se realizează procesările datelor, iar la sfârșit obținându-se utilizatorii potriviți preferințial țintei.

În ceea ce privește experimentele și rezultatele obținute, implementările acestora au fost concepute tot în cadrul pachetelor din cadrul serviciului Algorithms, având legătură directă cu preferințele și algoritmi implementați. Pentru colectarea datelor, s-a utilizat șablonul de proiectare *Observer*. Subiectul este cel responsabil de informarea observatorilor cu privire la schimbările efectuate, aici reprezentând lista preferințelor. Atunci când se inițializează obiectul, se obțin preferințele din fișierul *stored_preferences.txt*, iar apoi acestea sunt transformate în date numerice, pe baza cărora se stabilesc datele de antrenare și de testare.

```

class PreferencesSubject(Subject):
    def __init__(self):
        super().__init__()
        with open("stored_preferences.txt", "r+") as file:
            text = file.read()
            preferences_string = ast.literal_eval(text)
            dictionary_of_indices, dictionary_of_preferences = map_preferences(preferences_string) # 0: '.net'; '.net': 0
            self.preferences = [get_preferences_as_numeric(x, dictionary_of_preferences) for x in preferences_string]

            self.preferences_count = len(self.preferences)
  
```

```

self.training_count = int(0.7 * self.preferences_count)

def shuffle_preferences(self):
    random.shuffle(self.preferences)
    training_preferences = self.preferences[:self.training_count]
    testing_preferences = self.preferences[self.training_count:]
    self.notify(self.preferences, training_preferences, testing_preferences)

```

Observatorii sunt cei care se ocupă de colectarea datelor efectivă. S-au colectat date cu privire la acuratețea algoritmilor, precum și similaritatea dintre utilizatori pe diferite grupuri. Atunci când are loc operația de amestecare a preferințelor, observatorii reantrenează algoritmi și realizează testele pe baza seturilor de date primite de *Subiect*.

```

def testing(self, testing_data):
    test_set_accuracy = 0

    for new_preference in testing_data:
        self.total_preferences_analyzed += 1

        predictions = self.knn.predict(new_preference)
        relationships = list(map(lambda x: x[0], predictions))
        indices = list(map(lambda x: x[1], predictions))

        all_predictions = self.__get_all_predictions(new_preference)
        all_relationships = list(map(lambda x: x[0], all_predictions))

        # similarities
        self.collect_similarities(relationships, all_relationships)

        # accuracy modifications
        statistics = Statistics(self.preferences, relationships, indices,
new_preference)
        accuracy = statistics.accuracy()
        test_set_accuracy += accuracy

    self.overall_accuracy += test_set_accuracy / len(testing_data)

```

Datele cu privire la similaritățile utilizatorilor sunt colectate prin intermediul clasei *SimilarityData*. Acesta se ocupă cu încadrarea utilizatorilor în grupuri, astfel:

```

class SimilarityData(object):
    def __init__(self):
        self.very_low = 0 # [0-30)%
        self.low = 0 # [30-50)%
        self.moderate = 0 # [50-70)%
        self.high = 0 # [70-80)%
        self.very_high = 0 # [80-100]%

    def collect_data(self, relationships):
        for relationship in relationships:
            if relationship < 0.3: self.very_low += 1
            elif 0.3 <= relationship < 0.5: self.low += 1
            elif 0.5 <= relationship < 0.7: self.moderate += 1
            elif 0.7 <= relationship < 0.8: self.high += 1
            else: self.very_high += 1

```

Iar datele legate de acuratețea algoritmilor se realizează cu ajutorul clasei create *Statistics*.

Modul de funcționare este simplu: dacă există măcar o preferință în comun, atunci se poate considera că există o similaritate între cei doi utilizatori comparați:

```
def accuracy(self):
    total = len(self.indices)
    total_intersected = 0

    new_preference_set = set(self.new_preference)
    for index in self.indices:
        found_preference = set(self.preferences[index])
        intersected = True if len((new_preference_set & found_preference)) != 0 else False

        if intersected is True:
            total_intersected += 1

    return total_intersected / total
```

Apoi, pe baza observatorilor creați, s-a realizat agregarea datelor. Agregarea constă în colectarea datelor obținute de toți observatorii într-un singur loc. Astfel, au fost realizate trei tipuri de agregări: după k, după metrice și după ambele variabile. La sfârșit, acestea sunt salvate și în fișiere JSON, ce vor fi utilizate pentru afișarea grafică a rezultatelor, de exemplu, în cazul agregării în funcție de k:

```
def save_to_files(self):
    method = f"k={self.k}"
    very_low = self.similarity_data.very_low
    low = self.similarity_data.low
    moderate = self.similarity_data.moderate
    high = self.similarity_data.high
    very_high = self.similarity_data.very_high
    total = very_low + low + moderate + high + very_high

    data_to_save = {
        "method": method,
        "total": total,
        "very_low": very_low,
        "low": low,
        "moderate": moderate,
        "high": high,
        "very_high": very_high
    }

    with open("collected_data/results_aggregated_k.json", "r+") as file:
        collected_data = json.load(file)

    collected_data.append(data_to_save)

    with open("collected_data/results_aggregated_k.json", "w") as file:
        json.dump(collected_data, file, indent=4)
```

Utilizând librăria matplotlib din Python, s-au realizat grafice cu privire la datele agregate. Pe lângă acestea, s-a testat și algoritmul atunci când se creează noi utilizatori, tocmai pentru a vedea dacă într-adevăr algoritmul este și corect.

Fără acest serviciu, aplicația nu ar mai putea realiza recomandările de utilizatori. Însă, cu toate acestea, utilizatorii vor putea să-și vizualizeze contul și să realizeze operații de tip autentificare și autorizare în cadrul aplicației. Serviciul acesta, așadar, este necesar pentru punerea în evidență a temei abordate în proiect, dar fără acesta, aplicația poate rula în continuare.

Cap. III. Rezultatele aplicației

Algoritmul KNN este cel mai adesea folosit în problemele de clasificare și regresie, dar poate fi utilizat și ca măsură a similarității, precum și în problemele de compresie a datelor, finanțe, recunoaștere a formelor și în sistemele de recomandări, fiind util aplicației curente. Avantajele abordării KNN constau în implementarea ușoară, adaptabilitatea rapidă și numărul mic de parametri necesari (valoarea lui k și o metrică). În schimb, nu se recomandă utilizarea algoritmului acesta din cauză că pot apărea fenomenele de suprapotrivire și scalare slabă atunci când sunt foarte multe date de luat în calcul, devenind astfel mai lent pe măsură ce crește numărul de variabile [30].

În cazul aplicației realizate, KNN a fost utilizat drept măsură a similarității, cu scopul de a realiza recomandări. Astfel, se obțin utilizatorii cu preferințele cele mai similare. Gradul de similaritate poate fi considerat mai degrabă o valoare continuă decât discretă (în cazul etichetelor din cadrul clasificării clasice), iar aceasta se poate obține cu ajutorul unor metrici: Jaccard, euclidiană și cosine.

Prin aceste experimente se urmăresc: compararea metricilor cosine, euclidiană și Jaccard; determinarea valorii k cea mai potrivită pentru KNN (3, 5, 7, 11, lungimea datelor de antrenare); dar și urmărirea potrivirii utilizatorilor în funcție de o țintă cu preferințe oarecare, încât modelul să fie unul nu doar funcțional, ci și corect. Pe baza experimentelor obținute în urma recomandărilor, unui utilizator nou i se poate asocia rapid un grup de persoane cu interese similare.

III.1. Experimente realizate

În cadrul experimentelor realizate, s-a dorit să se verifice corectitudinea, acuratețea și gradul de similaritate dintre utilizatori, în funcție de valorile lui k și de metricile utilizate.

Corectitudinea algoritmului se poate verifica manual, creând mai multe preferințe de test și a verifica dacă preferințele recomandate de algoritm s-ar potrivi cu cele inițiale. Dacă se potrivesc, înseamnă că algoritmul KNN este corect utilizat pentru acea valoare a lui K și metrică.

Acuratețea este dată de prezența măcar a unei similarități dintre două entități. Atunci când se compară două seturi de preferințe iar intersecția dintre ele este un rezultat nul, înseamnă că nu sunt similare deloc. În caz contrar, acuratețea va crește. Atunci când acuratețea metricii depășește un anumit prag, se poate considera că algoritmul este potrivit aplicației și corect utilizat.

Gradul de similaritate indică cât de apropiate sunt preferințele dintre utilizatori, în urma rulării testelor. Se urmăresc obținerea a cât mai mulți utilizatori care să aibă preferințe cât mai multe în comun și, pe baza acestui fapt, s-au evidențiat diferențele dintre metricile utilizate.

Pentru a testa modelul KNN și metricile menționate, precum și efectivitatea acestora, au fost necesare următoarele preprocesări:

- Transformarea preferințelor utilizatorilor în date numerice (preferințele, inițial reprezintă string-uri; în funcție de toate preferințele utilizatorilor, s-a realizat operația de mapare a acestora în numere întregi pozitive, de exemplu, ["c++", "java", "html"] se transformă în [0, 1, 2], reprezentând indexul mapării);

- Pe baza preferințelor, seturile de date se vor împărți în două subseturi: de antrenare (70%), și de testare (30%), astfel încât toate preferințele să se regăsească în datele reunite;

- Metrica dorită a fi utilizată;

- Valoarea lui k pentru a utiliza modelul KNN.

În urma antrenării modelului, este așteptată obținerea de utilizatori cu preferințe similare țintei, ținând cont de datele de intrare alese la început.

Pentru compararea celor trei metrici se utilizează același set de date de antrenare respectiv testare în calcularea acurateței. Atunci trebuie ales același k , iar preferințele de antrenare și potrivire trebuie să fie aceleași. Similar se procedează și când se urmărește compararea valorilor

lui k.

În urma rulării, se vor obține vecinii cei mai apropiați utilizatorului cu preferințele alese. Intrând mai adânc în detalii, mai întâi se realizează antrenarea algoritmului, prin funcția `fit(points)`. Această funcție se ocupă de binarizarea tuturor vectorilor folosiți ca date de antrenare. Binarizarea vectorilor are rolul de a simplifica din calcule și îmbunătăți performanța. Prin binarizare, valorile continue sunt transformate în valori binare de 0 (absența unei preferințe) și 1 (prezența preferinței). Apoi, atunci când pe baza unei ținte se doresc utilizatorii similari, trebuie apelată funcția `predict(new_point, target_index)`. În cadrul funcției acesteia, se binarizează preferințele țintei, apoi se compară cu cele antrenate de algoritm (utilizând o metrică), obținându-se relațiile dintre țintă și ceilalți utilizatori. Relațiile reprezintă în cazul unor metrici (cosinus și Jaccard) similaritatea, iar în cazul altora (euclidian) distanța. Tocmai de aceea, pentru că se dorește obținerea celor mai apropiate preferințe, trebuie să aibă loc și o sortare a acestora în funcție de apropierea lor. Astfel, se obțin cei mai buni k utilizatori din punct de vedere al preferințelor. La sfârșit, pe baza unui procentaj de vot, se pot obține, de exemplu, cei mai buni 70% dintre utilizatori, criteriul de selecție fiind numărul cât mai mare de preferințe în comun.

Similaritatea cosine este folosită în sisteme de recomandări. Matematic, ea reprezintă cosinusul unghiului dintre doi vectori și e folosită drept metrică de evaluare a distanței dintre două puncte din plan. Cu cât distanța dintre cele două puncte este mai mare, cu atât similaritatea dintre puncte este mai mică [31]. Formula similarității cosinusului se regăsește mai jos, cu A, B vectorii comparați [32]:

$$\text{cosine similarity} = Sc(A, B) = \frac{A \cdot B}{||A|| ||B||} \quad (1)$$

$$\text{cosine similarity} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}} \quad (2)$$

Distanța euclidiană reprezintă lungimea unui segment dintre două puncte. Cu cât distanța dintre cele două puncte este mai mare, cu atât similaritatea este mai mică [33]. Mai jos se află formula de calcul a acestei distanțe, A și B fiind cei doi vectori în plan.

$$\text{euclidian distance} = d(A, B) = ||A - B||_2 \quad (3)$$

$$\text{euclidian distance} = \sqrt{\sum_{i=1}^n (A_i - B_i)^2} \quad (4)$$

Similaritatea Jaccard este o metrică de proximitate pentru a calcula asemănarea dintre două obiecte, cum ar fi două documente text. Matematic, se definește drept cardinalul intersecțiilor preferințelor supra cardinalul reuniunii celor două mulțimi. Cu cât rezultatul fracției tinde către o valoare supraunitară, cu atât cele două seturi de date sunt mai similare [34]. Formulele de calcul pentru metrica Jaccard sunt acestea:

$$\text{Jaccard similarity} = J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (5)$$

$$\text{Jaccard similarity} = J(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (6)$$

III.2. Rezultatele obținute

În tabelul de mai jos se găsesc trei vectori țintă pentru care se dorește găsirea seturilor de preferințe similare (în cadrul procesării are loc maparea în format continuu), alături vecinii cei mai apropiați, obținuți pentru fiecare metrică în parte.

Tabelul III.1 Evaluarea Metricilor, cu K=5, percentage=80%

Preferință țintă	Metricile evaluate		
	Cosine	Euclidiană	Jaccard
['python', 'c++', 'java', 'sql']	[['c++', 'java', 'python', 'algorithms'], ['c++', 'python', 'sql', 'artificial intelligence', 'machine learning', 'data science', 'web development'], ['java', 'c++', 'python', 'algorithms', 'data structures', 'software architecture', 'testing'], ['python', 'java', 'c++', 'artificial intelligence', 'machine learning', 'data science', 'devops']]	[['c++', 'java', 'python', 'algorithms'], ['python', 'sql'], ['java', 'c++', 'rust', 'algorithms'], ['java']]	[['c++', 'java', 'python', 'algorithms'], ['c++', 'python', 'sql', 'artificial intelligence', 'machine learning', 'data science', 'web development'], ['java', 'c++', 'python', 'algorithms', 'data structures', 'software architecture', 'testing'], ['python', 'java', 'c++', 'artificial intelligence', 'machine learning', 'data science', 'devops']]
['python']	[['python', 'sql'], ['python', 'django', 'html'], ['python', 'assembly', 'compilers'], ['javascript', 'python', 'compilers']]	[['python', 'sql'], ['python', 'django', 'html'], ['cooking'], ['java']]	[['python', 'sql'], ['python', 'django', 'html'], ['python', 'assembly', 'compilers'], ['javascript', 'python', 'compilers']]
['react', 'css', '.net', 'c#']	[['microsoft', '.net', 'c#', 'html', 'css', 'javascript', 'windows', 'sql'], ['java', 'c#', 'react'], ['react', 'javascript', 'css', 'html', 'node.js'], ['c#', '.net', 'asp.net', 'sql server', 'azure']]	[['java', 'c#', 'react'], ['react', 'javascript', 'css', 'html', 'node.js'], ['c#', '.net', 'asp.net', 'sql server', 'azure'], ['cooking']]	[['microsoft', '.net', 'c#', 'html', 'css', 'javascript', 'windows', 'sql'], ['java', 'c#', 'react'], ['react', 'javascript', 'css', 'html', 'node.js'], ['c#', '.net', 'asp.net', 'sql server', 'azure']]

Similar primului tabel, a fost întocmit încă unul cu valoarea k=3, tocmai pentru a se putea observa existența diferențelor nu doar între metrici, ci și a valorilor lui k date (Tabelul III.2).

Tabelul III.2 Evaluarea Metricilor, cu K=3, percentage=80%

Preferință țintă	Metricile evaluate		
	Cosine	Euclidiană	Jaccard
['python', 'c++', 'java', 'sql']	[['c++', 'java', 'python', 'algorithms'], ['c++', 'python', 'sql', 'artificial intelligence', 'machine learning', 'data science', 'web development']]	[['c++', 'java', 'python', 'algorithms'], ['python', 'sql']]	[['c++', 'java', 'python', 'algorithms'], ['c++', 'python', 'sql', 'artificial intelligence', 'machine learning', 'data science', 'web development']]
['python']	[['python', 'sql'], ['python', 'django', 'html']]	[['python', 'sql'], ['python', 'django', 'html']]	[['python', 'sql'], ['python', 'django', 'html']]
['react', 'css', '.net', 'c#']	[['microsoft', '.net', 'c#', 'html', 'css', 'javascript', 'windows', 'sql'], ['java', 'c#', 'react']]	[['java', 'c#', 'react'], ['react', 'javascript', 'css', 'html', 'node.js']]	[['microsoft', '.net', 'c#', 'html', 'css', 'javascript', 'windows', 'sql'], ['java', 'c#', 'react']]

Pe baza celor două tabele se poate constata că cele mai bune metrice dintre cele trei testate pentru potrivirea utilizatorilor sunt cosine și Jaccard. Motivația este simplă: în metrica euclidiană, fiind necesară conversia în vectori binarizați, multe dintre preferințe vor fi setate pe valoarea 0 și doar foarte puține vor avea 1. Matematic vorbind, se caută preferințe aflate la distanțe euclidiene cât mai mici de preferința țintă, dar asta nu garantează că vor fi preferințe dintre cele pe care le deține utilizatorul. Totuși, din punct de vedere al distanței euclidiene, acei utilizatori sunt similari.

S-au rulat 100 de teste prin care se dorește compararea rezultatelor obținute dintre cele 3 metrice, dar și dintre valori diferite ale lui k . Datele colectate sunt legate de numărul de utilizatori ale căror preferințe se încadrează într-o anumită categorie față de preferințele utilizatorului țintă. În total, sunt 5 categorii în care se pot încadra preferințele (foarte în comun, în comun, în moderație, puțin în comun, foarte puțin în comun). Inițial, toate aceste grupuri pornesc de la 0 indivizi. Iterând prin vectorul cu vectori de preferințe obținuți pe baza algoritmului KNN, în funcție de valoarea elementului curent (preferința vecină), se compară gradul de similitudine cu preferința țintă și adaugă profilul în categoria corectă. Fiecare grup crește în număr în funcție de gradele de similitudine. Acestea au fost încadrate astfel:

$$\begin{aligned} \text{very low} &= \begin{cases} \text{very low} + 1, & 0 \leq \text{similitude} < 0.3 \\ \text{very low}, & \text{otherwise} \end{cases} \\ \text{low} &= \begin{cases} \text{low} + 1, & 0.3 \leq \text{similitude} < 0.5 \\ \text{low}, & \text{otherwise} \end{cases} \\ \text{moderate} &= \begin{cases} \text{moderate} + 1, & 0.5 \leq \text{similitude} < 0.7 \\ \text{moderate}, & \text{otherwise} \end{cases} \\ \text{high} &= \begin{cases} \text{high} + 1, & 0.7 \leq \text{similitude} < 0.8 \\ \text{high}, & \text{otherwise} \end{cases} \\ \text{very high} &= \begin{cases} \text{very high} + 1, & 0.8 \leq \text{similitude} < 1.0 \\ \text{very high}, & \text{otherwise} \end{cases} \end{aligned}$$

Similitudinea reprezintă gradul de asemănare dintre ținta curentă și vecinul ei la acea iterație. Dacă e mai mare, înseamnă că cei doi utilizatori sunt mai potriviți preferențial. Pentru acest algoritm, similitudinea este necesară pentru a putea încadra utilizatori în aceste categorii.

Se rulează testele pentru fiecare dintre date, obținând rezultate pentru k ori câte metrice sunt, conform tabelelor următoare (Tabelul III.3, Tabelul III.4, Tabelul III.V):

Tabelul III.3 Rezultate cosine, pentru valori diverse ale parametrului k

k	100 teste, cosine, pentru fiecare k					
	Very High	High	Moderate	Low	Very Low	Accuracy
3	85251	63335	172472	235433	185859	87.52%
5	88578	98926	250744	374383	424619	82.25%
7	88805	111681	326114	500941	704609	78.52%
11	88805	112846	406352	763358	1350589	73.65%
110	88805	112846	433344	1177764	25406741	20.26%

În urma rulării aceluiași test și încadrării similitudinale, au fost realizate statisticile de mai sus. Important din acesta este valoarea coloanei Very High și a coloanei High, tocmai pentru că acestea sunt valorile ce indică cei mai potriviți utilizatori.

Așa cum se observă în tabel, valorile acestor coloane rămân aproximativ identice (se stabilizează) de la anumite valori ale lui k . Este foarte important și numărul de utilizatori ce sunt luați în considerare. Aplicația, având doar 161 de utilizatori pentru care să ruleze algoritmul, este evident că valori mai mici ale lui k ar fi mai potrivite. Este posibil ca pentru o aplicație cu mulți utilizatori să aibă nevoie de valori mai mari ale lui k .

Tabelul III.4 Rezultate euclidiană, pentru valori diverse ale parametrului k

<i>k</i>	100 teste, euclidiană, pentru fiecare k					
	<i>Very High</i>	<i>High</i>	<i>Moderate</i>	<i>Low</i>	<i>Very Low</i>	<i>Accuracy</i>
3	16948	29335	286058	396713	13296	66.82%
5	16948	29335	398166	753967	38834	60.55%
7	16948	29335	436860	1166052	82955	56.19%
11	16948	29335	450245	1981987	243435	49.57%
110	16948	29335	451174	5729120	20992923	20.18%

Similar tabelului anterior, a fost realizată o astfel de observație și pentru metrica euclidiană. Spre deosebire de metrica de tip cosine, acesta înregistrează valori ale acurateței mai slabe.

Tabelul III.5 Rezultate Jaccard, pentru valori diverse ale parametrului k

<i>k</i>	100 teste, Jaccard, pentru fiecare k					
	<i>Very High</i>	<i>High</i>	<i>Moderate</i>	<i>Low</i>	<i>Very Low</i>	<i>Accuracy</i>
3	35768	45581	101100	165150	394751	87.52%
5	35768	48908	154520	229711	768343	82.25%
7	35768	49135	186185	289027	1172035	78.52%
11	35768	49135	191854	378311	2066882	73.65%
110	35768	49135	191854	413468	26529275	20.26%

Iar în cazul metricii Jaccard s-au obținut valori ale acurateței. Similar celorlalte tabele, de la anumite valori ale lui k se înregistrează același număr de utilizatori cu preferințe în comun. Aceasta poate însemna că, acea valoare a lui k este cea mai potrivită algoritmului pentru acel număr de utilizatori.

De notat ar fi și faptul că, atât Jaccard cât și metrica euclidiană au aceleași valori ale acurateței, cu toate că grupurile de utilizatori sunt diferit distribuite. Fenomenul acesta poate apărea atunci când datele sunt binarizate și atunci când sunt împrăștiate (*engl. sparse*), de exemplu conțin foarte multe valori de zero, iar în cazul aplicației se întâmplă ambele. Mai precis, este vorba despre metrica similarității cosine, aplicată în cazul vectorilor binarizați. De fapt și de drept, metrica de tip cosine este pentru a compara vectori de valori *reale* (iar metrica Jaccard pentru vectori cu valori binare). Atunci când se compară vectori binarizați, numărătorul din cadrul formulei se simplifică, produsul punctat devenind numărul de elemente din mulțimea intersecției dintre cei doi vectori comparați. Deoarece evaluarea constă în verificarea existenței unei valori (mai mari decât zero) și din cauză că au același numărător, acuratețea este identică pentru ambele metrici [35].

După acesta, pe baza tabelelor astfel obținute, se pot agrega datele, asociându-se cu metricile respective (k va lua valori de la [3, 5, 7, 11, 110] și se vor aduna pentru metrica respectivă toți utilizatorii). Următoarele două tabele sunt realizate cu scopul de a compara metricile (*Tabelul III.6*), precum și valorile lui k alese (*Tabelul III.7*).

Tabelul III.6 Rezultate agregate cu k=[3, 5, 7, 11, 110]

<i>Metrica</i>	100 teste, k=[3, 5, 7, 11, 110]				
	<i>Very High</i>	<i>High</i>	<i>Moderate</i>	<i>Low</i>	<i>Very Low</i>
Cosine	440244	499634	1589026	3051879	28072417
Euclidiană	84740	146675	2022503	10027839	21371443
Jaccard	178840	241894	825513	1475667	30931286

Pe baza celor 100 de teste, s-au obținut și următoarele detalii cu privire la modelul KNN și

valorile k ale acestuia, adunându-se valorile fiecărei metrici pentru fiecare k.

Tabelul III.7 Rezultate agregate pentru fiecare metrică

<i>k</i>	<i>100 teste, pentru fiecare metrică</i>					
	<i>Very High</i>	<i>High</i>	<i>Moderate</i>	<i>Low</i>	<i>Very Low</i>	<i>Accuracy</i>
3	137967	138251	559630	797296	593906	80.62%
5	141294	177169	803430	1358061	1231796	75.01%
7	141521	190151	949159	1956020	1959599	71.07%
11	141521	191316	1048451	3123656	3660906	65.62%
110	141521	191316	1076372	7320352	72928939	20.23%

Scăderea acurateții pe măsură ce crește numărul de preferințe comparate nu indică decât faptul că există o suprapotrivire. Aceasta poate fi rezolvată fie reducând valorile lui k, fie adăugând mai mulți utilizatori (de ordinul sutelor sau chiar miilor) în aplicație care să aibă preferințe relativ comune celorlalți.

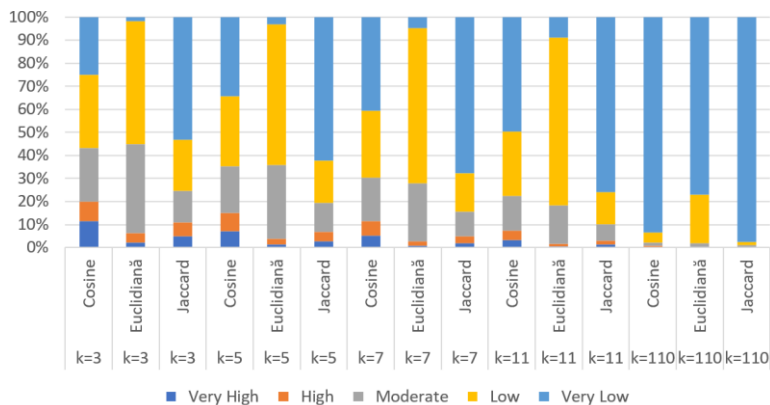


Figura III.1. Compararea valorilor tuturor metricilor și tuturor valorilor lui k, procentual

Graficul de mai sus încearcă să rezume cele discutate anterior, punând în evidență atât metricile folosite, valorile lui k utilizate cât și încadrarea procentuală, pentru fiecare categorie.

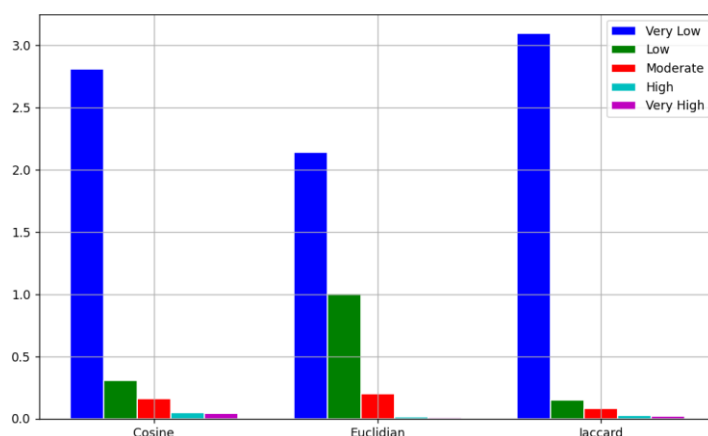


Figura III.2. Compararea celor trei metrici și a încadrării preferințelor lor

În cadrul graficului din *Figura III.2* se poate remarca faptul că, dintre cele trei metrici utilizate, varianta euclidiană este cel mai puțin potrivită pentru acest tip de problemă. În schimb, metricile Jaccard și cosine sunt mai optimiste și se potrivesc mai bine tipului acesta de problemă.

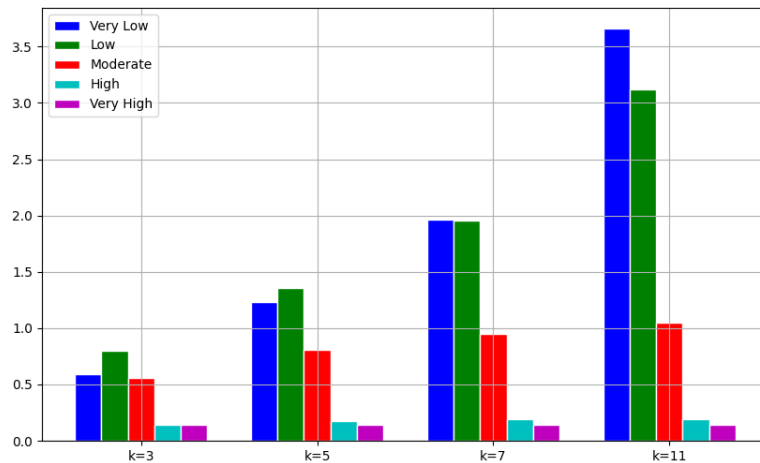


Figura III.3. Compararea valorilor lui $k=[3, 5, 7, 11]$ și a preferințelor lor

Figura III.3 are rolul de a evidenția stabilizarea efectivelor. De la valorile $k=7$ și $k=11$ se poate remarca faptul că numărul de valori din cadrul grupurilor *High* și *Very High* sunt aproape identice, ceea ce înseamnă că, dacă valoarea lui k va crește și mai mult, numărul de indivizi din cadrul acestor grupuri nu va mai crește. Acest fapt este util identificării valorilor k potrivite.

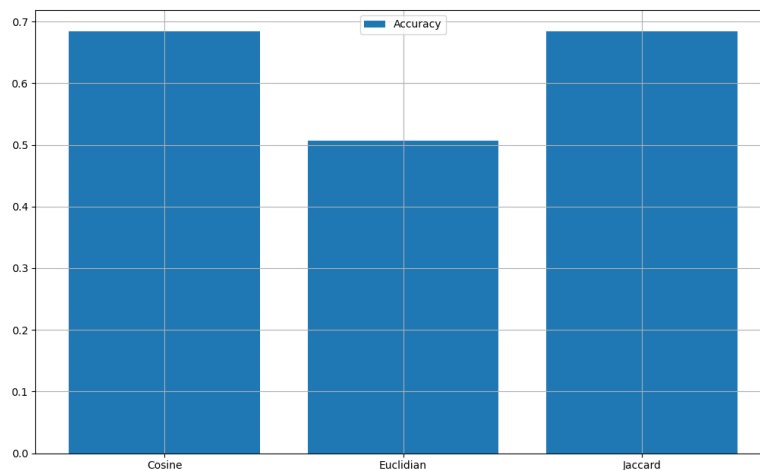


Figura III.4. Compararea valorilor de acuratețe ale celor trei metrici

Așa cum s-a discutat anterior, acuratețea identică pentru metricile cosine și Jaccard se datorează binarizării vectorilor și numărului mare de valori de 0, numitorul din cadrul formulei metricii cosine simplificându-se la numitorul din formula metricii Jaccard pentru similaritate.

În urma acestor experimente, s-a constatat că metrica euclidiană este mai puțin potrivită pentru această problemă, atât din cauză că se recomandă preferințe ce nu sunt neapărat comune cu ținta, cât și din cauză că sunt recomandați relativ puțini oameni cu un număr de preferințe în comun mai mare. Acestea se datorează tocmai distanței matematice euclidiene dintre vectorii binarizați. De aceea, cei mai mulți dintre ei se încadrează la cei cu preferințe moderat și puțin potrivite. În ceea ce privește metrica de tip cosine, aceasta, deși are cel mai mare număr de utilizatori încadrați la similarități *High* și *Very High*, trebuie ținut cont și de faptul că, în mod normal, această metrică se utilizează pentru vectori cu valori *reale*, și nu binare, cum e în cazul metricii Jaccard.

Concluzii

În prezent, tot mai multe sisteme necesită cunoștințe din ce în ce mai largi pentru domeniile ce țin de inteligență artificială și sisteme distribuite, iar crearea aplicațiilor de tipul recomandărilor de persoane nu este o excepție. Tocmai pentru a înțelege mai multe despre aceste domenii, prin acest proiect s-au propus următoarele obiective: realizarea unui sistem de autentificare și autorizare, utilizarea mecanismelor de securizare a aplicației, modularizarea serviciilor și claselor, implementarea unei interfețe grafice cu utilizatorul, utilizarea bazelor de date, și cel mai important, realizarea unui sistem de recomandări în scop educativ, ce ar putea fi extins și către alte domenii.

Proiectul meu constă în recomandarea persoanelor în funcție de preferințele lor în comun cu alți oameni. Am utilizat arhitectura bazată pe microservicii tocmai pentru a separa responsabilitățile serviciilor, obținând astfel o aplicație mai organizată, cu un grad sporit de securitate (alături de criptare, decriptare, JWT și Spring Security), ce poate fi scalată și eșua în mod independent. Am folosit și elemente de inteligență artificială, precum algoritmul KNN cu metricile cosinelor, euclidiană și Jaccard. Există suficiente variante pentru a recomanda utilizatorii în funcție de similaritatea preferințelor deținute de aceștia. Am ales să implementez algoritmul KNN personal tocmai din cauză că voiam să înțeleg mai bine cum s-ar implementa fără librării auxiliare, precum și din cauză că voiam să experimentez metricile menționate și să văd care dintre acestea s-ar potrivi mai bine pentru problema mea. Pe deasupra, am realizat și o interfață grafică cu utilizatorul și am reușit să stochez datele necesare, atât cu privire la utilizatori, profiluri, tokeni (cu MariaDB și MongoDB), cât și cele referitoare la datele despre metricile testate pentru KNN.

Pe lângă acestea, pot considera un succes și experimentele realizate din cadrul algoritmului KNN. În comparație cu metrica euclidiană, cosinelor este mult mai optimistă, înregistrând mult mai multe preferințe potrivite cu cele ale țintei, fapt datorat de binarizarea vectorilor. În cazul metricii Jaccard, s-au remarcat tendințe preferențiale mai mari decât în cazul metricii cosinelor. Spre deosebire de metrica euclidiană, Jaccard reușește să obțină utilizatori potriviți țintei, din punct de vedere al preferințelor. De asemenea, s-a constatat că cele mai potrivite valori ale lui k pentru această problemă sunt cuprinse între 5 și 11. Trebuie avut grijă la valorile lui k , deoarece, atunci când sunt prea mici, pot produce suprapotriviri, obținând utilizatori cu preferințe eronate. În schimb, dacă sunt prea mari, pot apărea subpotriviri, ceea ce va spori timpul de calcul, iar numărul de utilizatori încadrați în categoriile de similari și foarte similari se va stabili de la un anumit k .

Deși realizarea acestui tip de aplicații este dificilă, consider că, prin această lucrare, am atins obiectivele dorite, implementând specificațiile propuse mai sus. În momentul de față, abordarea KNN reușește să rezolve problema aceasta corect și cu succes. Mai departe, în scop explorativ, aplicația ar putea fi extinsă cu mai multe implementări și strategii de a realiza recomandările de utilizatori utilizând alți algoritmi decât KNN, ar putea avea un chat în timp real cu ceilalți utilizatori și să realizeze conexiunile cu aceștia. De asemenea, ar putea exista un sistem pentru raportare, unul pentru asistență a utilizatorilor și chiar și de notificare a acestora, iar tot în scopul îmbunătățirii experienței lor, aplicația ar putea să-și îmbunătățească și interfața, fiind în final mai reactivă, atractivă și ușor de folosit.

Bibliografie

- [1] Dalilah Anna, „Know the benefits of Online friendships”, <https://wearerestless.org/2022/04/28/what-can-we-gain-from-online-friendships/>, ultima accesare: iunie 2023.
- [2] Xin Wei, Shiyun Sun, Dan Wu, Liang Zhou, „Personalized Online Learning Resource Recommendation Based on Artificial Intelligence and Educational Psychology, Frontiers in Psychology”, Sec. Educational Psychology Volume 12, 2021, <https://doi.org/10.3389/fpsyg.2021.767837>.
- [3] Zeinab Shahbazi, Yung-Cheol Byun, „Agent-Based Recommendation in E-Learning Environment Using Knowledge Discovery and Machine Learning Approaches”, Mathematics 2022, 10(7), 1192, <https://doi.org/10.3390/math10071192>.
- [4] Qian Zhang, Jie Lu, Yaochu Jin, „Artificial intelligence in recommender systems”, Complex Intell. Syst. 7, 439–457 (2021), <https://doi.org/10.1007/s40747-020-00212-w>.
- [5] William Park, „Online dating might not help you to find the one. But the data from dating apps offers some tantalising insights.”, <https://www.bbc.com/future/article/20191112-how-dating-app-algorithms-predict-romantic-desire>.
- [6] Florian Auer, Valentina Lenarduzzi, Michael Felderer, Davide Taibi, „From monolithic systems to Microservices: An assessment framework, Information and Software Technology”, Volume 137, 2021, <https://doi.org/10.1016/j.infsof.2021.106600>.
- [7] roadmap.sh contributors, „Backend Developer”, 2023, <https://roadmap.sh/backend>, ultima accesare: iunie 2023.
- [8] Nicolae Sfetcu, „Limbajul de programare Kotlin pentru Android”, <https://www.telework.ro/ro/limbajul-de-programare-kotlin-pentru-android>, ultima accesare: iunie 2023.
- [9] Yiğit Kemal Erinc, „The Benefits of Going RESTful – What is REST and Why You Should Learn About It”, <https://www.freecodecamp.org/news/benefits-of-rest/>, ultima accesare: iunie 2023.
- [10] MDN contributors, „HTTP”, <https://developer.mozilla.org/en-US/docs/Web/HTTP>, ultima accesare: iunie 2023.
- [11] Spring contributors, „Spring Boot”, <https://spring.io/projects/spring-boot>, ultima accesare: iunie 2023.
- [12] Spring contributors, „Spring Security”, <https://spring.io/projects/spring-security>, ultima accesare: iunie 2023.
- [13] Auth0, „JSON Web Tokens”, <https://jwt.io/>, ultima accesare: iunie 2023.
- [14] M. Jones, Microsoft, J. Bradley ș.a., „JSON Web Token (JWT)”, <https://datatracker.ietf.org/doc/html/rfc7519>, ultima accesare: iunie 2023.
- [15] roadmap.sh contributors, „Frontend Developer”, 2023, <https://roadmap.sh/frontend>, ultima accesare: iunie 2023.
- [16] Cameron Chapman, „Breaking Down the Principles of Design (with Infographic)”, <https://www.toptal.com/designers/gui/principles-of-design-infographic>, ultima accesare: iunie 2023.
- [17] baeldung, „Introduction to Using Thymeleaf in Spring”, <https://www.baeldung.com/thymeleaf-in-spring-mvc>, ultima accesare: iunie 2023.
- [18] Google Cloud, „Artificial intelligence (AI) vs. machine learning (ML)”, <https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning>,
- [19] python, „Python”, <https://www.python.org/>, ultima accesare: iunie 2023. ultima accesare: iunie 2023.
- [20] pandas contributors, „pandas”, <https://pypi.org/project/pandas/>, ultima accesare: iunie 2023.
- [21] Matplotlib contributors, „Matplotlib”, <https://matplotlib.org/>, ultima accesare: iunie 2023.

- [22] fullstackpython, „Full Stack Python”, <https://www.fullstackpython.com/flask.html>, ultima accesare: iunie 2023.
- [23] Medina Iglesias, Juan Antonio, „Hands-On Microservices with Kotlin: Build reactive and cloud-native microservices with Kotlin using Spring 5 and Spring Boot 2.0”, Packt Publishing, Anglia, 2018.
- [24] Mehmet Ozkaya, „API Gateway Pattern”, <https://medium.com/design-microservices-architecture-with-patterns/api-gateway-pattern-8ed0ddfce9df>, ultima accesare: iunie 2023.
- [25] baeldung, „The DTO Pattern (Data Transfer Object)”, <https://www.baeldung.com/java-dto-pattern>, ultima accesare: iunie 2023.
- [26] Spring docs contributors, Expression-Based Access Control, <https://docs.spring.io/spring-security/reference/servlet/authorization/expression-based.html>, ultima accesare: aprilie 2023
- [27] refactoring.guru, „Strategy”, <https://refactoring.guru/design-patterns/strategy>, ultima accesare: iunie 2023.
- [28] refactoring.guru, „Observer”, <https://refactoring.guru/design-patterns/observer>, ultima accesare: iunie 2023.
- [29] Thorben Janssen, „Entity Mappings: Introduction to JPA FetchType”, <https://thorben-janssen.com/entity-mappings-introduction-jpa-fetchtypes/>, ultima accesare: iunie 2023.
- [30] IBM, „What is the k-nearest neighbors algorithm?”, <https://www.ibm.com/topics/knn>, ultima accesare: iunie 2023.
- [31] Darshan M., „What is cosine similarity and how is it used in machine learning?”, Mystery Vault, <https://analyticsindiamag.com/cosine-similarity-in-machine-learning/#:~:text=Cosine%20similarity%20is%20used%20as%20a%20metric%20in,find%20the%20similarity%20of%20texts%20in%20the%20document.>, ultima accesare: iunie 2023
- [32] Selva Prabhakaran, „Cosine Similarity – Understanding the math and how it works (with python codes)”, https://www.machinelearningplus.com/nlp/cosine-similarity/?utm_content=cmp-true, ultima accesare: iunie 2023.
- [33] Nikos kalikis, „Text Similarity: Euclidian Distance VS Cosine Similarity !!!”, <https://nikoskalikis.medium.com/text-similarity-euclidian-distance-vs-cosine-similarity-3a1167f686a>, ultima accesare: iunie 2023.
- [34] Fatih Karabiber, „Jaccard Similarity”, <https://www.learndatasci.com/glossary/jaccard-similarity/>, ultima accesare: iunie 2023.
- [35] Data Science Stack Exchange, „Applications and differences for Jaccard similarity and Cosine Similarity”, <https://datascience.stackexchange.com/questions/5121/applications-and-differences-for-jaccard-similarity-and-cosine-similarity>, ultima accesare: iunie 2023.