# Aplicație bazată pe microservicii pentru identificarea persoanelor cu interese similare

Radu-Valentin Cornea
UNIVERSITATEA TEHNICĂ
„Gheorghe Asachi" din IAȘI
FACULTATEA DE AUTOMATICĂ
ȘI CALCULATOARE
Iași, România
radu-valentin.cornea@student.tuiasi.ro
Coordonator științific:
Ș.l. dr. inf. Tiberius Dumitriu

***Abstract*—În această lucrare se propune prezentarea unor metode de recomandare a anumitor persoane, în funcție de preferințele pe care le au în comun cu ceilalți utilizatori, folosind metode din sfera inteligenței artificiale. Accentul cade pe grupuri de interes din sfera educațională, dar poate fi extins la orice alte domenii.**

**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. 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 detemina cea mai potrivită recomandare. Rezultatele obținute sunt prezentate sub formă de tabele și grafice care evidențiază avantajele acestei abordări.**

**Pe baza experiențelor anterioare obținute în urma recomandărilor, unui utilizator nou i se poate asocia rapid un grup de persoane cu interese similare.**

***Cuvinte cheie—Aplicații software, Aplicații educaționale, Inteligență artificală***

## I. INTRODUCERE

Proiectul constă într-o aplicație pentru interconectarea oamenilor, în funcție de preferințele lor, utilizând diverși algoritmi. Unii dintre algoritmii utilizați sunt mai simpli, iar alții folosesc inteligență artificială. Se dorește și recomandarea utilizatorilor în funcție de alte filtre precum distanța geografică, dar momentan recomandările se fac doar în funcție de preferințe. Scopul 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 folosit și în alte arii decât cele educaționale.

Totuși, există o mulțime de studii pe tema recomandărilor, unii dintre oameni gândindu-se deja la sisteme de recomandări bazate pe inteligență artificială și pe psihologia educației, tocmai pentru a recomanda studenților resurse de studiu, în funcție de cum interacționeaza cu imaginile de pe site-ul respectivei instituții și cu elementele video din acestea. Studenții au fost clasați 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 [1]. Există foarte multe domenii de aplicare pentru algoritmii de recomandare, și pot fi folosiți nu doar în scopuri educative, ci și divertisment, precum filmele [2].

În urma unui studiu de piață, s-a constatat că, în ciuda faptului că există aplicații care încearcă să recomande oameni 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. 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.

Pe baza acestui studiu de piață, s-a ajuns la concluzia că o astfel de aplicație pentru recomandarea persoanelor în funcție de preferințele lor ar fi necesară pieței. Obiectivul principal este acela de a găsi și de a filtra cât mai mulți utilizatori potriviți cu ținta în cauză, dar și de a avea un produs funcțional, sigur și securizat, care să fie ușor de folosit, plăcut și de înțeles de oricine.

O posibilitate de a rezolva această problemă este de a folosi microserviciile și inteligența artificială. Arhitecturile bazate pe microservicii oferă scalabilitate, rulare independentă, performanțe adiționale, avantajul de a fi ușor de menținut, costuri reduse și multe altele [3]. 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: KNN, SVM, SVD, Random Forest, filtrul colaborativ, filtrul bazat pe conținut, abordări hibride [4, 5].

Tocmai de aceea, în încercarea de a rezolva problema propusă, tehnologiile și conceptele teoretice folosite în cadrul aplicației pot fi clasificate astfel:

- Backend: Spring, Spring Security, JWTs, REST, Kotlin, MariaDB, MongoDB, criptare, decriptare;
- Frontend: Thymeleaf, JavaScript, CSS, HTML;
- AI: KNN, Python, sklearn, pandas;
- Aplicații software: IntelliJ IDEA, PyCharm, Visual Studio Code, Postman, MongoDB, Compass, DBeaver.

## II. Implementare

### A. Concept

O posibilitate de a rezolva soluția propusă, așa cum s-a discutat în capitolul anterior, este de a folosi microserviciile împreună cu inteligența artificală. Arhitectura întregii aplicații ar putea fi împărțită astfel: pentru servicii, se pune accentul pe cum comunică între ele și cum sunt împărțite, încercând să se respecte cât mai bine principiile SOLID, atât pentru clase, cât și microservicii, și cea dea doua arhitectură este pentru inteligență artificială, în care importante sunt datele care intră, datele care se așteaptă drept ieșiri, dar și modul în care au fost prelucrate acestea, putând asemana arhitectura aceasta cu white box.

### B. Proiectarea aplicației

Alte părți de cod au o granularitate fină, având propriul proiect și server, urmărindu-se în acest fel și decuplarea aplicației dezvoltate. Aplicația se folosește de modelul client-server, fiind bazată pe servicii/microservicii. Câteva dintre serviciile implementate până acum cuprind: IDM (Identity Management), Profile, Algorithms și Gateway-ul ce leagă serviciile. Alte exemple de servicii utile aplicației, dar neimplementate încă, 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).

Pe lângă cele două arhitecturi menționate, au fost necesare și diagrame de servicii, diagrame UML și ER. Pentru o bună funcționalitate, dar și dezvoltare mai facilă a aplicației, părțile de cod au fost despărțite pe module și clase, în unele situații fiind necesare șabloanele de proiectare. Unul dintre acestea este chiar șablonul de proiectare strategie. Acesta 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: căutari directe pe utilizatori utilizând paradigma funcțională, iar celelalte strategii bazându-se pe algoritmul KNN și câteva metrici ale acestuia.

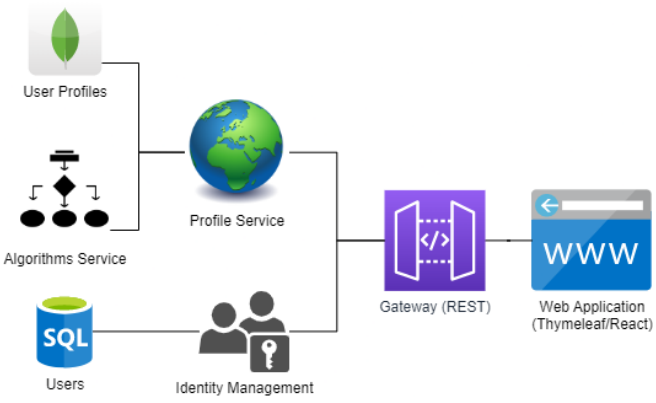


Fig. 1. Diagrama de servicii

### C. Modalități de securizare a aplicației

Pentru siguranța datelor utilizatorilor, s-au utilizat divere 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.

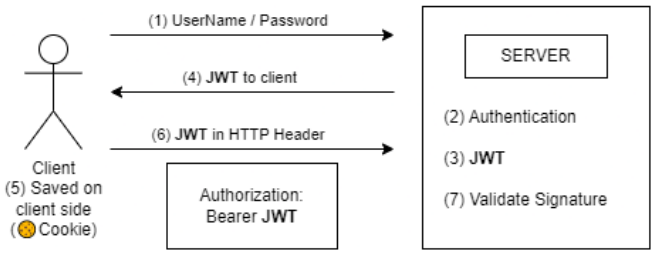


Fig. 2. Interacțiunea dintre client și server folosind JWT

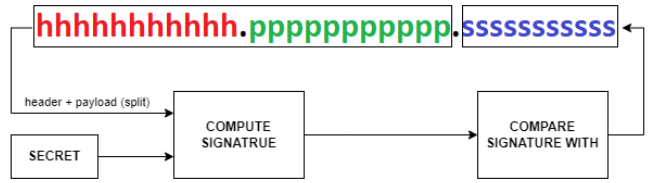


Fig. 3. Validarea semnăturii unui JWT

Pentru a persista parolele, acestea sunt ținute în baza de date criptate cu ajutorul funcției BCrypt, 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 folosit JWT-urile și Spring Security.

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). Un exemplu de filtru folosit în cadrul aplicației pentru serviciul IDM (modificat puțin pentru lizibilitate), se poate regăsi mai jos:

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

    return http

        .authorizeHttpRequests()
            .requestMatchers("/invalidate-
token").access(WebExpressionAuthorizationManager("
hasIpAddress('localhost')"))
            .requestMatchers("/check-
username/**").permitAll()
            .requestMatchers("/css/**", "/js/**",
"/images/**").permitAll()
            .anyRequest().authenticated()
            .and()

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

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

        .build()
}
```

Așa cum se observă în secvența de cod, Spring Security se folosește de Spring Expression Language (SpEL), pentru autorizarea într-un mod facil al utilizatorilor în funcție de diverși parametri, precum hasIpAddress, hasAuthority, principal și mulți alții [6].

### D. Date

Pentru a realiza cele propuse, 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.

Pentru început, au fost necesare setările proprietăților aplicației:

```
spring.datasource.url=jdbc:mariadb://localhost:330
6/security
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-
name=org.mariadb.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
```

Apoi, se pot crea repositories, similare exemplelor de mai jos, în care se pot remarca sintaxa JPQL, dar și Query-uri pentru MongoDB:

```
interface TokenRepository : JpaRepository<Token,
String> {
    @Query("select t from Token t where t._uuid =
:uuid")
    fun findTokenByUuid(uuid: String): Token

    @Query("select t from Token t where t.token =
:token")
    fun findTokenByToken(token: String): Token
}

interface ProfileRepository :
MongoRepository<UserProfile, String>{

    @Query(
        value = "{ 'idmId' : { \$ne: ?0 }, " +
                "'\$or' : [ " +
                "{ 'firstName' : { \$regex: ?1,
\$options: 'i' } }, " +
                "{ 'lastName' : { \$regex: ?1,
\$options: 'i' } } " +
                "] }",
        // projection parameter to exclude fields
       fields = "{ 'email' : 0, 'idmId' : 0, 'id'
: 0 }")
    fun findUsersNotMyIDAndMatchingName(userID:
Int, search: String): Optional<List<UserProfile>>
}
```

Mai jos se află entitatea utilizator, folosită pentru asocia utilizatorii din baza de date MariaDB.

```
@Entity
@Table(name = "users")
class User {
    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    @Column(name = "id")
    private var _id = 0
    private var username: String = ""
    private var password: String = ""
    @Transient
    private var passwordConfirm: String = ""

    @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()
}
```

Există și o entitate pentru token, în care se vor ține cont detaliile cu privire la conținutul din payload. Câmpul revocationFlag reprezintă validitatea token-ului respectiv.

```
@Entity
@Table(name = "tokens")
class Token() {

    @Id
    @Column(name = "uuid", nullable = false,
unique = true)
    private var _uuid: String = ""
    private var token: String = ""
    private var userId: Int = 0
    private var expiryDate: Date = Date()
    private var revocationFlag: Boolean = false

    constructor(uuid: String, token: String,
userId: Int, expiryDate: Date, revocationFlag:
Boolean) : this() { ... }
}
```

Mai există și o entitate tabelă pentru autorități, folosită de utilizatori. Rolul ei este de a permite accesul rutelor în funcție de valorile pe care re are utilizatorul în câmpul acesta.

```
@Entity
@Table(name = "authorities")
class Authority {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    @Column(name = "id")
    private var _id = 0
    private var name: String = ""

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

Asocierea dintre utilizatorul simplu din IDM cu profilul acestuia este facută prin stocarea câmpului idmId chiar în interiorul profilului, atunci când este creat la înregistrare.

```
_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"
```

Fig. 4. Structura profilului utilizatorului

### E. Interfață grafică

Unul dintre framework-urile utilizate pentru interfața grafică este Thymeleaf. Motivația este simpla: 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(!userValidatorService.isUserRegisterValid(userM
odel))
        return ModelAndView("register-fail")
    return ModelAndView("register-success")
}
```

Iar la niviel 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}"/>
```

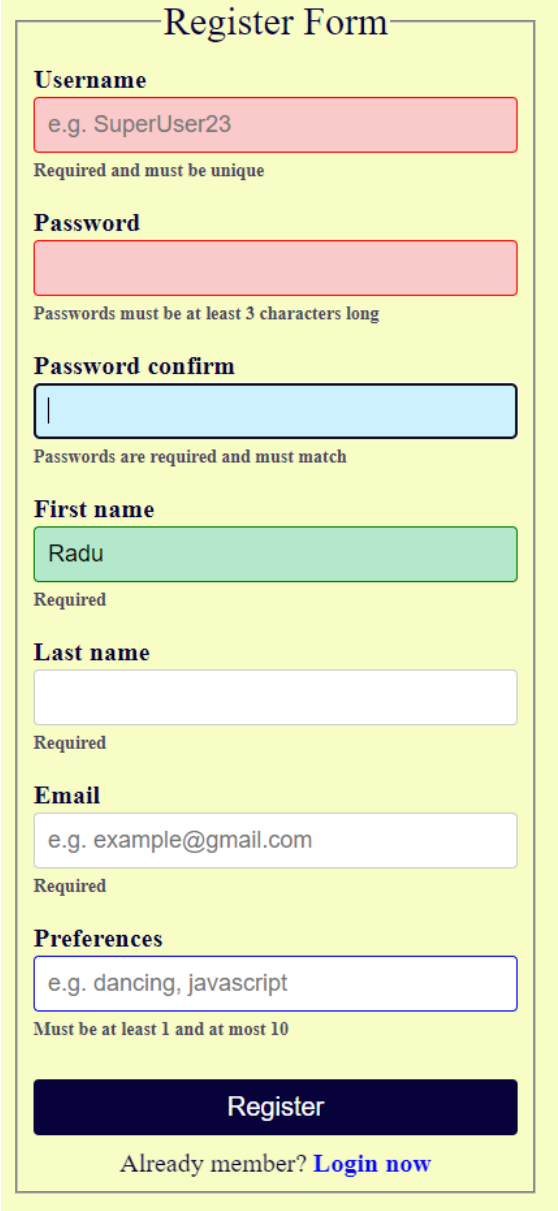


Fig. 5. Exemplu de pagină realizată cu Thymeleaf

## III. Experimente și rezultate

Până în momentul de față, s-a realizat o comparare între câteva dintre metricile algoritmului KNN, acestea fiind Jaccard, euclidiană și cosine. Dar pe lângă aceste comparări, s-au întocmit și niște rapoarte care ar sugera diverse situații de utilizare mai potrivite acelor metrici.

Prin aceste experimente se urmăresc: compararea metricilor cosine, euclidiană și Jaccard; compararea k-urilor 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.

### A. *Prerecuzită*

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

- Preferințele utilizatorilor, ca listă de liste de numere;
- Pe baza preferințelor, se vor obține, în mod stohastic, preferințe de antrenare (70%), precum și preferințe pentru potrivire (30%), astfel încât toate preferințele să se regăsească în datele reunite;
- Metrica vrută a fi utilizată;
- Valoarea lui k pentru a utiliza modeul KNN;

### B. *Modelul KNN*

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

Pentru compararea corectă a rezultatelor, trebuie ca aceste comparații să fie făcute pentru aceleași date de intrare. De exemplu, dacă se dorește compararea celor trei metrici, 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 principiu, modelul KNN are urmatorul algoritm:

```
knn = KNNCosine(training_preferences)
knn.train(k)
indices =
knn.fit_indices(fitting_preferences)
```

În urma rulării acestei secvențe de cod se vor obține vecinii cei mai apropiați utilizatorului cu preferințele alese. Intrând mai adânc în detalii, la nivelul constructorului se binarizează preferințele și se ține cont de aceste detalii, astfel încât să poată fi comparate corect folosind mai multe metrici. Atunci când se apelează funcția train(k), se antrenează modelul folosind metrica aleasă și preferințele binarizate. Indicii utilizatorilor cu preferințele apropiate se obțin prin binarizarea vectorului de preferințe nou inclus (kneighbors([target_user_binary])). Similar se pot obține și distanțele utilizatorilor. Dar ce este de interes în final este vectorul de preferințe similare țintei, care se poate obține prelucrând aceste date menționate.

### C. *Cosine*

```
cos_sim = np.dot(u, v) /
(np.linalg.norm(u) * np.linalg.norm(v))
```

### D. *Euclidiană*

```
dist = np.linalg.norm(u - v)
```

### E. *Jaccard*

```
intersection = len(u.intersection(v))
union = len(u.union(v))
jaccard_sim = intersection / union
```

### F. *Tabele*

Pentru o bună înregistrare și urmările a rezultatelor, înaintea fiecăror tabele s-au specificat si valorile adiționale, care s-ar repeta, precum k, o anumită metrică și altele, dar și alte informații utile în a înțelege cum au fost realizate.

Pentru început, se întocmesc tabele pentru a vedea dacă utilizatorii chiar s-ar potrivi așa cum s-ar aștepta.

TABLE I. EVALUAREA METRICILOR, CU K=3

| **Preferințele țintei** | **Metricile evaluate** | | |
|---|---|---|---|
| | ***Cosine*** | ***Euclidiană*** | ***Jaccard*** |
| [5, 19, 32] | [[19, 99, 81, 4, 5], [58, 19, 4, 5, 6, 66], [142, 19, 4, 5, 6, 14]] | [[19, 99, 4], [19, 145, 146], [21]] | [[19, 99, 81, 4, 5], [58, 19, 4, 5, 6, 66], [142, 19, 4, 5, 6, 14]] |
| [16] | [[16], [13, 5, 14, 15, 6, 4, 16, 17], [18, 19, 6, 4, 5, 14, 20, 21, 17, 16]] | [[16], [113], [21]] | [[16], [13, 5, 14, 15, 6, 4, 16, 17], [18, 19, 6, 4, 5, 14, 20, 21, 17, 16]] |
| [141, 5, 19, 55, 13] | [[142, 19, 4, 5, 6, 15], [142, 19, 4, 5, 6, 14], [142, 19, 4, 5, 6, 66, 67]] | [[18], [8], [19, 99, 4]] | [[142, 19, 4, 5, 6, 14], [142, 19, 4, 5, 6, 15], [142, 19, 4, 5, 6, 66, 67]] |

TABLE II. EVALUAREA METRICILOR, CU K=5

| **Preferințele țintei** | **Metricile evaluate** | | |
|---|---|---|---|
| | ***Cosine*** | ***Euclidiană*** | ***Jaccard*** |
| [5, 19, 32] | [[19, 99, 81, 4, 5], [58, 19, 4, 5, 6, 14], [58, 19, 4, 5, 6, 66], [142, 19, 4, 5, 6, 14], [58, 19, 4, 5, 6, 15]] | [[18], [19, 145, 146], [8], [19, 29, 212], [19, 29, 212]] | [[19, 99, 81, 4, 5], [142, 19, 4, 5, 6, 15], [58, 19, 4, 5, 6, 15], [142, 19, 4, 5, 6, 14], [58, 19, 4, 5, 6, 14]] |
| [16] | [[16], [13, 5, 14, 15, 6, 4, 16, 17], [18, 19, 6, 4, 5, 14, 20, 21, 17, 16], [116, 117, 80, 118, 119], [159, 160]] | [[16], [113], [18], [21], [161, 162]] | [[16], [13, 5, 14, 15, 6, 4, 16, 17], [18, 19, 6, 4, 5, 14, 20, 21, 17, 16], [58, 19, 8, 31, 29, 50, 28], [205, 206, 207, 208]] |
| [141, 5, 19, 55, 13] | [[142, 19, 4, 5, 6, 15], [142, 19, 4, 5, 6, 14], [142, 19, 4, 5, 6, 66, 67], [13, 5, 14, 15, 6, 4, 16, 17], [19, 8]] | [[19, 8], [8], [215, 19, 209], [19, 29, 212], [19, 29, 212]] | [[142, 19, 4, 5, 6, 15], [142, 19, 4, 5, 6, 14], [142, 19, 4, 5, 6, 66, 67], [13, 5, 14, 15, 6, 4, 16, 17], [19, 8]] |

TABLE III. EVALUAREA METRICILOR, CU K=7

| **Preferințele țintei** | **Metricile evaluate** | | |
|---|---|---|---|
| | ***Cosine*** | ***Euclidiană*** | ***Jaccard*** |
| [5, 19, 32] | [[19, 99, 81, 4, 5], [142, 19, 4, 5, 6, 14], [58, 19, 4, 5, 6, 15], [58, 19, 4, 5, 6, 14], [58, 19, 4, 5, 6, 66], [142, 19, 4, 5, 6, 66, 67], [142, 58, 19, 4, 5, 6, 66]] | [[19, 8], [215, 19, 209], [6, 19, 146], [113], [16], [21], [19, 99, 81, 4, 5]] | [[19, 99, 81, 4, 5], [142, 19, 4, 5, 6, 14], [58, 19, 4, 5, 6, 15], [58, 19, 4, 5, 6, 66], [58, 19, 4, 5, 6, 14], [19, 8], [142, 58, 19, 4, 5, 6, 66]] |
| [16] | [[16], [18, 19, 6, 4, 5, 14, 20, 21, 17, 16], [21, 20, 98, 14, 133], [196, 197, 198, 199], [113], [164, 165], [17, 6, 25]] | [[16], [8], [18], [113], [156, 157], [147, 151], [148, 163]] | [[16], [18, 19, 6, 4, 5, 14, 20, 21, 17, 16], [19, 70, 90, 91, 213], [6, 14, 66, 80, 103], [19, 29, 212], [157, 156], [205, 206, 207, 208]] |
| [141, 5, 19, 55, 13] | [[19, 99, 81, 4, 5], [58, 19, 4, 5, 6, 14], [58, 19, 4, 5, 6, 66], [142, 19, 4, 5, 6, 14], [142, 19, 4, 5, 6, 15], [142, 58, 19, 4, 5, 6, 14], [142, 58, 19, 4, 5, 6, 66]] | [[19, 8], [113], [19, 209, 211], [215, 19, 209], [19, 210, 216], [19, 29, 210], [6, 19, 146]] | [[19, 99, 81, 4, 5], [58, 19, 4, 5, 6, 14], [142, 19, 4, 5, 6, 15], [142, 19, 4, 5, 6, 14], [58, 19, 4, 5, 6, 66], [142, 58, 19, 4, 5, 6, 14], [142, 58, 19, 4, 5, 6, 66]] |

Pe baza celor trei tabele se poate constata că cele mai bune metrici 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 metrici, dar și dintre valori diferite ale lui k. Pentru o bună înțelegere a rezultatelor, trebuie înțeles și ceea ce se colectează și cum anume se colectează. Datele colectate sunt legate de numărul de utilizatori ale căror preferințe se încadrează într-o anumită categorie față de preferințele utilizatorului țintă. Preferințele vecine țintei au fost încadrate în categorii astfel:

```
if similitude < 0.1: self.very_low += 1
elif 0.1 <= similitude < 0.5: self.low += 1
elif 0.5 <= similitude < 0.7: self.moderate += 1
elif 0.7 <= similitude < 0.8: self.high += 1
else: self.very_high += 1
```

Similitudinea reprezintă gradul de asemănare dintre ținta curentă și vecinul ei la acea iterație. Dacă e mai mare, înseamnă că sunt mai potriviți preferențial cei doi utilizatori.

TABLE IV. REZULTATE PENTRU FIECARE K ȘI METRICĂ

| **k** | **Metrica** | **100 teste, pentru fiecare k și metrică** | | | | | |
|---|---|---|---|---|---|---|---|
| | | ***Very High*** | ***High*** | ***Moderate*** | ***Low*** | ***Very Low*** | ***Total*** |
| 3 | Cosine | 1816 | 1236 | 3930 | 5511 | 1907 | 14400 |
| 3 | Euclidiană | 380 | 609 | 7867 | 5544 | 0 | 14400 |
| 3 | Jaccard | 792 | 900 | 2000 | 8685 | 2023 | 14400 |
| 5 | Cosine | 1863 | 1984 | 5554 | 10116 | 4483 | 24000 |
| 5 | Euclidiană | 380 | 609 | 12235 | 10776 | 0 | 24000 |
| 5 | Jaccard | 792 | 947 | 3111 | 14346 | 4804 | 24000 |
| 7 | Cosine | 1867 | 2234 | 7033 | 14856 | 7610 | 33600 |
| 7 | Euclidiană | 380 | 609 | 15472 | 17139 | 0 | 33600 |
| 7 | Jaccard | 792 | 951 | 3704 | 19874 | 8279 | 33600 |
| 11 | Cosine | 1867 | 2252 | 8620 | 25452 | 14609 | 52800 |
| 11 | Euclidiană | 380 | 609 | 20105 | 31706 | 0 | 52800 |
| 11 | Jaccard | 792 | 951 | 3814 | 31005 | 16238 | 52800 |
| 110 | Cosine | 1867 | 2252 | 9086 | 89176 | 425619 | 528000 |
| 110 | Euclidiană | 380 | 609 | 30917 | 446624 | 494770 | 528000 |
| 110 | Jaccard | 792 | 951 | 3814 | 74353 | 448090 | 528000 |

Mai întâi, se rulează testele pentru fiecare dintre date, obținând rezultate pentru k x metricile, întocmai tabelului anterior. După acesta, pe baza tabelului astfel obținut, se agregă 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.

TABLE V. REZULTATE AGREGATE CU K=[3, 5, 7, 11, 110]

| Metrica | 100 teste, k=[3, 5, 7, 11, 110] | | | | | |
|---|---|---|---|---|---|---|
| | *Very High* | *High* | *Moderate* | *Low* | *Very Low* | *Total* |
| Cosine | 9280 | 9958 | 34223 | 145111 | 454228 | 652800 |
| Euclidiană | 1900 | 3045 | 86596 | 511789 | 49470 | 652800 |
| Jaccard | 3960 | 4700 | 16443 | 148263 | 479434 | 652800 |

În urma colectării datelor obținute pe baza celor 100 de teste și ale acelorași date de intrate, s-au obținut și următoarele detalii cu privire la modelul KNN și valorile k ale acestuia, adunându-se valorile fiecărei metrici pentru fiecare k.

TABLE VI. REZULTATE AGREGATE PENTRU FIECARE METRICĂ

| k | 100 teste, pentru fiecare metrică | | | | | |
|---|---|---|---|---|---|---|
| | *Very High* | *High* | *Moderate* | *Low* | *Very Low* | *Total* |
| 3 | 2988 | 2745 | 13797 | 19740 | 3930 | 43200 |
| 5 | 3035 | 3540 | 20900 | 35238 | 9287 | 72000 |
| 7 | 3039 | 3794 | 26209 | 51869 | 15889 | 100800 |
| 11 | 3039 | 3812 | 32539 | 88163 | 30847 | 158400 |
| 110 | 3039 | 3812 | 43817 | 610153 | 923179 | 1584000 |

Până în momentul de față, s-a realizat o comparare între câteva dintre metricile algoritmului KNN, acestea fiind Jaccard, euclidiană și cosine. Dar pe lângă aceste comparări, s-au întocmit și niște rapoarte care ar sugera diverse situații de utilizare mai potrivite acelor metrici.

## G. Grafice

Pe baza acestor tabele s-au întocmit câteva grafice pentru formarea unei idei mai bune de ansamblu asupra modului în care funcționează aceste metrici alese pentru a fi testate.

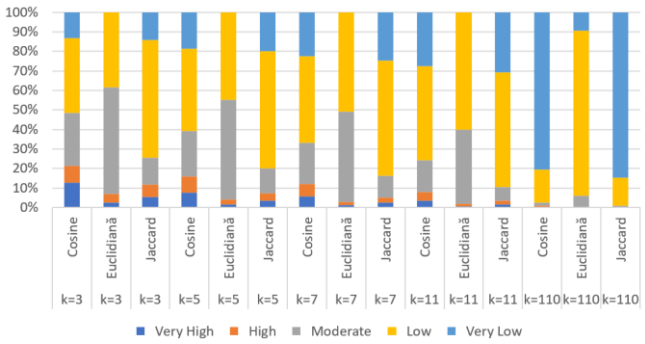


Fig. 6. Compararea celor valorilor tuturor metricilor și tuturor valorilor lui k, procentual

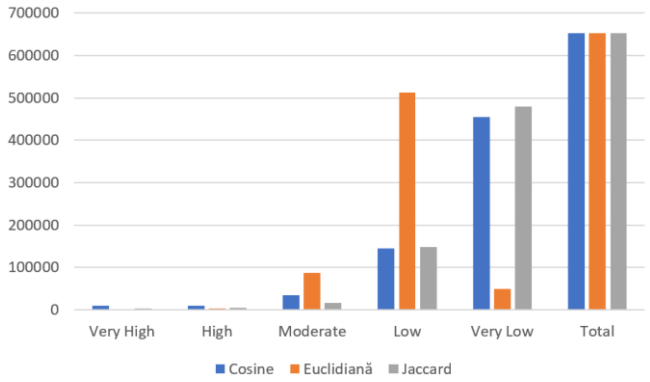


Fig. 7. Compararea celor trei metrici și a încadrării preferințelor lor

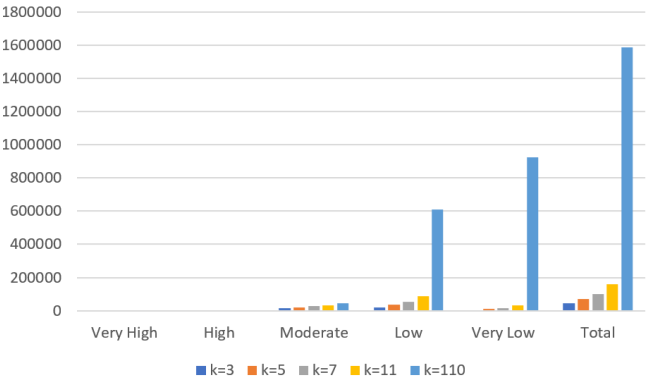


Fig. 8. Compararea celor 5 valori ale lui k și a preferințelor lor

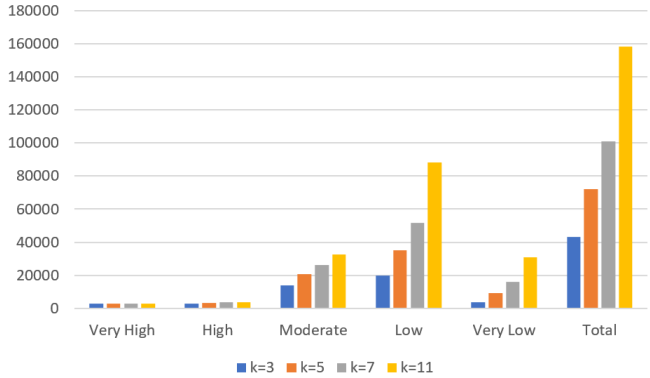


Fig. 9. Compararea celor 4 valori fără k=110 și a preferințelor lor

Pe baza acestor rapoarte, s-a constatat că metrica euclidiană este pesimistă în ceea ce privește recomandările utilizatorilor, și nu doar din cauză că s-ar realiza și recomandări de preferințe ce nu sunt neapărat comune cu ținta, ci și din cauză că sunt recomandați relativ puțini oameni cu preferințe în comun mai multe, fapt datorându-se tocmai distanței matematice dintre vectorii binarizați. De aceea, cei mai mulți dintre ei se încadrează la cei cu preferințe moderat potrivite.

În timp ce metrica de tip cosine este ceva mai optimistă, dar totuși potrivită în sensul corectitudinii, iar metrica Jaccard indică tendințe mai preferențiale, dar orișicât, și aceasta este corectă, recomandând doar utilizatori ce au preferințe în comun atunci când se permite acest fapt.

## IV. Concluzii

Ce și de ce? Ce aduc în plus?

## References


[1] https://www.frontiersin.org/articles/10.3389/fpsyg.2021.767837/full

[2] https://www.sciencedirect.com/science/article/pii/S0950584921000793?via%3Dihub

[3] https://www.mdpi.com/2227-7390/10/7/1192

[4] https://link.springer.com/article/10.1007/s40747-020-00212-w

[5] https://link.springer.com/article/10.1007/s40747-020-00212-w

[6] https://docs.spring.io/spring-security/reference/servlet/authorization/expression-based.html

[7]