UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Aplicarea algoritmilor de filtrare colaborativă și bazată pe conținut în dezvoltarea aplicațiilor web

Conducător științific
Conf. dr. Guran Adriana

Absolvent
Chiţea Răzvan

2025

ABSTRACT

Lucrarea prezintă procesul de dezvoltare a unei platforme sociale moderne, cu accent pe integrarea unui sistem de recomandare bazat pe algoritmi de filtrare colaborativă și conținut semantic. În capitolul 1 sunt prezentate motivația alegerii temei, domeniul abordat, obiectivele lucrării și structura generală a proiectului. Capitolul 2 este dedicat analizei teoretice a algoritmilor de filtrare colaborativă, diferențiind între abordările bazate pe memorie și pe model, precum și descrierii detaliate a algoritmilor hibrizi. Capitolul 3 tratează aspectele tehnice ale dezvoltării aplicației web, incluzând arhitectura client-server, framework-urile utilizate (Next.js, Flask, MongoDB) și bunele practici de design UI/UX. Capitolul 4 conține studiul de caz – rețeaua socială PoemSociety – în care este integrat sistemul de recomandare personalizat, iar fiecare poezie este analizată din perspectiva limbii, temei și emoțiilor. Sunt descrise atât funcționalitățile aplicației, cât și implementările de front-end și back-end, împreună cu diagramele de utilizare și testarea aplicației.

Contribuția originală a lucrării constă în propunerea și implementarea unui sistem hibrid de recomandare adaptiv, care combină scorurile colaborative și cele bazate pe conținut, aplicat într-un context cultural specific – promovarea poeziei printr-o platformă socială. Algoritmul propus integrează analiza semantică automată a textelor și permite ajustarea preferințelor utilizatorului în timp real, oferind o experiență personalizată și scalabilă.

Cuprins

Capitolul 1

Introducere

În zilele noastre, Inteligenţa Artificială (AI) este regăsită în orice tip de aplicaţie. Sistemele de recomandare bazate pe filtrarea colaborativă se regăsesc în majoritatea platformelor folosite de oameni: platforme sociale, magazine online, motoare de căutare, platforme de streaming, platforme de publicitate digitală etc, acestea fiind modelele AI care influenţează zilnic contactul lor cu lumea virtuală.

"Soon after the invention of the World Wide Web, the recommender system emerged and related technologies have been extensively studied and applied by both academia and industry. Currently, recommender system has become one of the most successful web applications, serving billions of people in each day through recommending different kinds of contents, including news feeds, videos, e-commerce products, music, movies, books, games, friends, jobs etc." [4]

În următoarele pagini este descris atât conceptul algoritmilor de filtrare, cât şi implementarea *Algoritmului de filtrare colaborativă* într-o aplicaţie web. Studiul de caz realizat pentru a înţelege fenomenele din spatele acestui algoritm constă într-o platformă sociala pentru poezii.

1.1 Obiectivele lucrării de licenţă

Această lucrare are drept scop încapsularea principiilor esenţiale ale Algoritmului de filtrare colaborativă şi exemplificarea acestuia printr-o platformă socială de tip TikTok, Instagram, X, ... Proiectul evidenţiază tehnologiile moderne AI integrate într-o aplicaţie web construită pe baza unui framework, unde algoritmul de filtrare colaborativă va recomanda poezii utilizatorilor în funcţie de preferinţe şi interacţiuni cu alţi utilizatori şi alte postări.

Pentru realizarea unui studiu de caz concret, este necesară şi analiza comparativă a algoritmilor, cel standard realizat prin asocieri între utilizatori şi postări, şi cel hibrid unde "tag"-uri sunt asociate postărilor, filtrarea fiind realizată şi prin cea

de conţinut. În următoarele pagini vor fi explicate toate cele enumerate în detaliu, iar partea practică a lucrării va fi compusă dintr-un website realizat în Next.js - platformă socială pentru poezii, unde conţinutul va fi filtrat diferit pentru fiecare utilizator, iar poeziile vor fi analizate din punct de vedere al genului, al tematicii şi al conţinutului.

Astfel, obiectivul principal este de a înţelege cum funcţionează acest algoritm şi cum îl putem testa, utilizând tehnologiile moderne AI într-o aplicaţie web.

1.2 Prezentarea domeniului din care face parte proiectul

Proiectul se încadrează în domeniul informaticii aplicate, în special la intersecția dintre dezvoltarea aplicațiilor web și inteligența artificială.

Conceptul de recomandare de conţinut a fost dezvoltat prima oară în 1979 de către Elaine Rich [15]. Aceasta a proiectat un algoritm bazat pe stereotipuri pentru sugerarea de cărţi, tratând problema ca un bibliotecar: creează pe baza unor întrebări un stereotip al cititorului, apoi recomandă cărţi acestuia pe baza unor clase de preferinţă. *"A system, Grundy, is described that builds models of its users, with the aid of stereotypes, and then exploits those models to guide it in its task, suggesting novels that people may find interesting."*[15] Printre pionierii sistemelor de recomandare a fost şi Google, care în anii 2000 prezintă două soluţii de acest tip pentru serviciile oferite de ei:

- Google Ads (2000) - Acest serviciu foloseşte "targeting behavioral" şi contextual pentru publicitate. Astfel reclamele oferite de aceştia sunt recomandate utilizatorilor în funcţie de istoricul de căutare, comportamentul acestuia, preferinţele şi interacţiuni cu alte reclame.

- Google News (2002) - Primul sistem major de recomandare prin filtrare colaborativă a fost implementat în această aplicaţie. Utilizatorilor li se recomanda ştiri din categoriile accesate în trecut sau ştiri ce erau accesate de utilizatori cu aceleaşi preferinţe. Ulterior Google a adăugat şi filtrarea bazată pe conţinut, care asocia "tag"-uri şi cuvinte cheie articolelor de ştiri.

Primele platforme sociale care au introdus acest algoritm în structura lor sunt Friendster (2002), MySpace (2003) şi Facebook (2004). Aplicaţile acestea recomandă utilizatorilor prieteni noi pe baza preferinţelor, dar şi a conexiunilor dintre persoane deja existente, formând primele forme de reţele sociale. MySpace şi Facebook proiectează algoritmul şi pentru sugerarea de conţinut, oferind utilizatorilor o experienţă personalizată şi unică. În zilele noastre Friendster şi MySpace nu au

reuşit să menţină popularitatea pe care au avut-o, în schimb Facebook a devenit un gigant al industriei. Platforme precum Instagram, X şi TikTok sunt cele mai populare acum, acestea folosind algormitmul de filtrare colaborativă cu mici îmbunătăţiri pentru tipul de conţinut şi pentru comportamentul utilizatorului. Aceştia au dezvoltat nişte aplicaţii atât de complexe, încât utilizatorilor le sunt create nişte profiluri exacte de preferinţe. *Uneori, algoritmul creat de TikTok pare să cunoască utilizatorul mai bine decât el pe el însuşi, aceasta fiind perspectiva comună a oamenilor despre puterea platformei* [13]

Dezvoltarea Web este una din cele mai accesibile metode de a crea o aplicaţie, atât pentru utilizator, cât şi pentru programator. Totuşi popularitatea aplicaţilor mobile a scăzut utilizarea site-urilor web a unor platforme repetitiv accesate, spre exemplu: magazine online (Amazon, Emag), platforme sociale (X - vechi: Twitter, Instagram, Facebook), platforme de streaming (Netflix, Hulu). Aceste aplicaţii au încă versiune web, astfel dezvoltarea într-un framework, ca React, oferă programatorului avantajul de a refolosi codul pentru varianta de browser pentru cea mobilă şi vice-versa. Versiunea web fiind primul mediu prin care o reţea socială s-a popularizat şi tranziţia la mobil nefiind complicată cu ajutorul framework-ului, studiul de caz va fi construit ca aplicaţie pentru browser.

1.3 Structura lucrării şi a proiectului

”Procesele computaționale sunt entități abstracte care locuiesc în computere. Pe măsură ce evoluează, aceste procese manipulează alte entități abstracte numite date. Evoluția unui proces este ghidată de un set de reguli numit program. Oamenii creează programe pentru a direcționa procesele. În esență, invocăm spiritele computerului cu vrăjile noastre.”[1]. Esenţial pentru a construi o aplicaţie complet, sunt necesare teoretizarea şi structurarea acesteia în totalitate, determinând bazele produsului, programului şi algoritmilori creaţi. Fără o direcţie anume, în procesul dezvoltării unei aplicaţii se crează un conflict între programator şi liniile de cod, ce-l distanţează pe acesta de rezultat, distrugând orice tip de entuziasm şi voinţă.

Pentru a evidenţia obiectivul licenţei - aplicarea şi explicarea *Algoritmilor de filtrare colaborativă* - lucrarea este structurată în patru părţi: *Descrierea Algoritmilor*, *Dezvoltarea Aplicaţiilor Web şi Realizarea Proiectului*. Aceste componente sunt împărţite astfel încât înţelegerea şi aplicarea algoritmului să poată fi studiată, atât de profesionişti din domeniu, cât şi de persoane interesate de informatică aplicată sau din domenii conexe. Fiecare capitol introduce noţiuni şi concepte noi, acumulând definiţile necesare dezvoltarii algoritmului şi al aplicaţiei în mod eficient şi corect.

Structurarea proiectului are o importanţă majoră pentru informatician, deoarece urma acesteia poate urma o serie de paşi pentru realizarea programului. O aplicaţie

web este în general structurată astfel: o platformă pe care să se poată lucra cu elementele de design, unul sau mai multe servere pentru accesarea datelor de la utilizator la aplicaţie şi de la aplicaţie la baza de date şi o bază de date modernă, securizată şi intuitivă pentru stocarea tuturor datelor necesare. Urmarea unui model pus la punct şi construit logic, ajută programatorul să lucreze la aplicaţie rapid, iar dacă aceasta este realizată de o echipă, structurarea ajută şi la împărţirea sarcinilor către informaticieni şi designeri.

1.4 Motivarea alegerii temei

Într-o lume care a ajuns codependentă de internet, înţelegerea mecanismelor din spatele aplicaţilor este importantă pentru evitarea dependenţei şi a manipulării create de acestea.

Atât platformele sociale, cât şi website-urile de ştiri, deţin puterea de informare, respectiv dezinformare a populaţiei. Acestea par a fi simple aplicaţii de divertisment sau documentare, dar prin diferiţi algoritmi decizionali din spatele lor, recomandă conţinut pe placul utilizatorului. În teorie, sistemul de sugestie pe baza preferinţelor ar aduce un impact pozitiv lumii, dar prin toate modalitaţile de *data-tracking* şi *data-collecting* acesta începe să limiteze accesul la informaţie - *filter-bubble* sau să afişeze conţinut fals, ambele în scop manipulativ. Spre exemplu un utilizator caută "destinaţii Grecia", "Zbor ieftin Grecia" şi interacţionează cu postări despre "Athena", sistemul de recomandare începe să îi afişeze mai multe postari cu aceste tag-uri specifice: grecia, athena, avion, zbor etc şi mai multe ştiri cu respectivele tag-uri, fără a filtra fake-news-ul de realitate şi adaugă reclame cu oferte şi pachete de vacanţă în scop comercial. Astfel decizia de a călători în Atena poate părea a utilizatorului, dar sistemul l-a influenţat pe acesta creându-i ideea de cea mai bună variantă.

Studiul "The Influence of Social Media on the Travel Behavior of Greek Millennials (Gen Y)" realizat de Mary Constantoglou şi Nikolaos Trihas, analizează comportamentul a 261 de persoane născute între 1980-1994 în timpul vacanţei (Figura 1.1). [3]. Iar articolul publicat de Andreea Fortuna Șchiopu, Ana-Maria Nica, Ana Mihaela Pădurean, Mădălina-Lavinia Ţală, susţine că generaţia Z utilizează platforme sociale pentru vacanţe şi în timpul acestora mai mult ca generaţia mileniarilor. [17] Acest fapt ar trebui luat în considerare de cei ce au puterea de a manipula conţinutul recomandat prin diferiţi algoritmi şi să dezvolte metode de evitare a bulelor de filtrare şi de analiză a conţinutului fals sau neadecvat, pentru a crea o experienţă personalizată şi nelimitată a utilizatorilor.

Prin tema aleasă, această lucrare evidenţiază mecanismele din spatele sistemelor de recomandare, pentru a înţelege fenomenele ce influenţează viaţa cotidiană, prin

platforme sociale, de ştiri, de reclame şi multe altele.

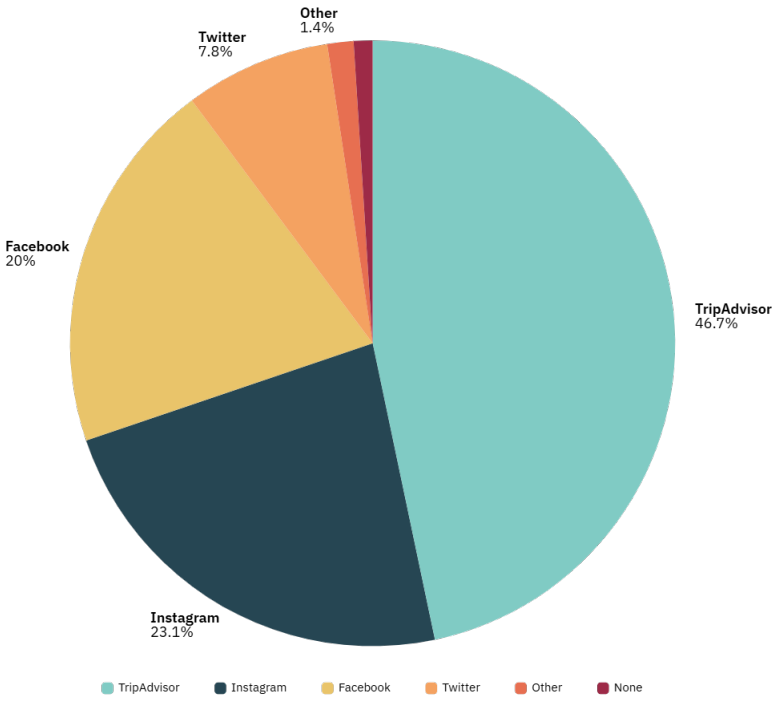

Figura 1.1: Utilizarea platformelor sociale în timpul vacanţei [3]

Capitolul 2

Algoritmi bazati pe filtrare colaborativă

Ce sunt de fapt algoritmii bazaţi pe filtrare colaborativă? Aceştia sunt nişte mecanisme de asociere a gusturilor utilizatorului în funcţie de amprenta sa digitală. Interacţiunea pozitivă cu un tip de conţinut asemănător duce la crearea unei preferinţe, pe care sistemul de recomandare o va prioritiza pe platforma folosită de utilizator. De asemenea, algoritmul ia în considerare şi feedback-ul negativ oferit de consumatorul de conţinut online, postările de acelaşi tip fiind omise sau chiar excluse din feed.

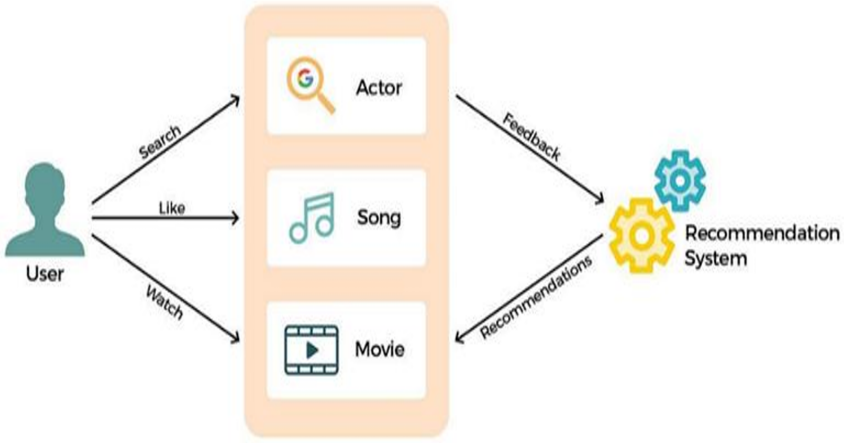

Figura 2.1: Fluxul comportamentului a unui utilizator [16]

În figura 2.1 se poate observa fluxul comportamentului utilizatorului. Acesta interacţionează cu iteme (căutări, like-uri, vizualizări, ...), platforma trimite acest feedback primit de la client către sistemul de recomandare, care în urma unei analize trimite conţinut filtrat potrivit preferinţelor utilizatorului înapoi. Exemplu: Utilizarea unei platforme sociale. Clientul îşi creează un profil unde specifică date semni-

ficative pentru algoritm: gen, locaţie, vârstă. Aceştia sunt primii factori analizaţi de sistemul de recomandare. Utilizatorul apoi începe să vizioneze un conţinut vast din mai multe domenii. Fiecare comportament poate fi analizat: timp de vizionare, like-uri, comentarii negative / pozitive, redirecţionări etc., iar din acestea algoritmul începe să creeze un sistem de notare. Procesul prin care recomandă mai departe conţinut este prin compararea activităţilor altor utilizatori cu preferinţe similare. În acelaşi timp şi postările noi sunt analizate şi asociate prin caracteristici la cele preferate de clienţi, pentru a menţine un flux continuu.

2.1 Noţiuni introductive

Potrivit studiului recent "Recommender Systems" [14], un sistem de recomandare este un tip de sistem de filtrare a informației, conceput pentru a prezice și sugera conţinut (produse, filme, muzică, articole, postări, etc) care ar putea interesa utilizatorul, bazându-se pe comportamentul și preferințele sale anterioare, dar și pe comportamentul altor utilizatori cu profiluri similare. Scopul principal al unui astfel de sistem este de a îmbunătăți experiența utilizatorului, de a crește gradul de implicare și de a facilita luarea deciziilor în contextul unui volum excesiv de opțiuni.

Matricea de ratinguri (Rating Matrix) este o reprezentare bidimensională a interacțiunilor dintre utilizatori și itemi. Fiecare linie corespunde unui utilizator, iar fiecare coloană unui item. Elementele din matrice conțin scoruri acordate de utilizatori pentru itemi, de regulă pe o scară de la 1 la 5. În realitate, această matrice este în general **sparsă**, deoarece majoritatea utilizatorilor nu evaluează fiecare item. O **matrice sparsă** este o matrice în care majoritatea elementelor sunt nule sau lipsă. În contextul sistemelor de recomandare, matricea de ratinguri este de obicei sparsă deoarece utilizatorii interacţionează de regulă cu o mică parte din totalul de conţinut.

Factorizarea matricei (Matrix Factorization – MF) este o tehnică matematică utilizată pentru a descompune matricea de ratinguri într-un produs al două matrici mai mici.

Factorii latenți reprezintă trăsături ascunse, invizibile direct în datele originale, care explică relația dintre utilizatori și itemi (produse, filme, poezii etc.) într-un sistem de recomandare.

MSE (Eroarea Pătratică Medie) este una dintre cele mai utilizate funcții de pierdere în algoritmii de învățare automată, în special pentru probleme de regresie și sisteme de recomandare. Se calculează ca media pătratelor diferențelor dintre valorile reale și cele prezise:

$$MSE = \frac{1}{|K|} \sum_{(u,i) \in K} \left(R_{ui} - \hat{R}_{ui} \right)^2$$

- R_{ui} este valoarea reală (ratingul oferit de utilizatorul u itemului i),
- $\hat{R}_{ui}$ este valoarea prezisă de algoritm,
- $|K|$ este numărul total de date reale (ratinguri observate);

SGD este o metodă de optimizare utilizată pentru a învăța valorile matricilor U și V din factorizare. Funcția de pierdere (loss) măsoară diferența între ratingurile reale și cele prezise. SGD actualizează treptat valorile embeddingurilor, în direcția minimizării acestei erori. Este eficient pe seturi mari de date și este utilizat frecvent în învățarea automată.

Norma Frobenius este o măsură a mărimii unei matrici, definită ca:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2},$$

unde a_{ij} sunt elementele matricii A. Este adesea utilizată ca termen de regularizare în algoritmii de factorizare matricială pentru a controla complexitatea modelului și a preveni overfitting-ul.

Exemplu: Fie matricea:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \|A\|_F = \sqrt{1^2 + 2^2 + 3^2 + 4^2} = \sqrt{30} \approx 5.48$$

2.2 Tipuri de filtrare colaborativă

Filtrarea colaborativă reprezintă una dintre cele mai utilizate metode în sistemele de recomandare și poate fi clasificată în două mari categorii: filtrare colaborativă bazată pe memorie și filtrare colaborativă bazată pe model. Această distincție este frecvent evidențiată în literatura de specialitate, inclusiv în revizuiri recente ale domeniului (“A comprehensive review of recommender systems: Transitioning from theory to practice”) [14].

2.2.1 Filtrare colaborativă bazată pe memorie (Memory-based CF)

Abordarea bazată pe memorie utilizează direct valorile existente în matricea de ratinguri. Aceasta nu necesită un model de învățare, ci se bazează pe calcularea similarităților între utilizatori sau itemi. Există două subtipuri comune:

- **User-based CF** — se pleacă de la ipoteza că utilizatorii cu preferințe similare în trecut vor avea preferințe similare în viitor. Similaritatea este adesea măsurată prin coeficientul de corelație Pearson sau măsura cosinusului.

- **Item-based CF** — presupune că dacă un utilizator a apreciat un anumit item, va aprecia și altele similare. Similaritatea este evaluată între itemi, nu între utilizatori.

Deși metodele bazate pe memorie sunt ușor de implementat și de înțeles, ele suferă de probleme de scalabilitate și performanță în prezența unor date sparse.

2.2.2 Filtrare colaborativă bazată pe model (Model-based CF)

Această abordare presupune învățarea unui model predictiv care captează relațiile subtile dintre utilizatori și itemi, de regulă prin utilizarea tehnicilor de factorizare matricială sau învățare automată. Printre cele mai cunoscute metode se numără:

- **Factorizarea matricii (Matrix Factorization)** — descrisă în detaliu anterior, reduce dimensiunea problemei și evidențiază factori latenti;
- **Metode bazate pe machine learning** — precum arbori de decizie, regresie sau modele ensemble;
- **Rețele neuronale** — care pot învăța reprezentări adânci și complexe ale preferințelor;
- **Factorizare tensorială** — utilizată în contexte multidimensionale (de ex., timp, locație).

Model-based CF este preferată în aplicațiile de mari dimensiuni datorită scalabilității și performanței sale ridicate. Conform Raza et al. (2024), aceste metode „au depășit abordările clasice prin capacitatea lor de a generaliza mai bine în condiții de sparsitate extremă și dinamică ridicată a datelor” [14].

2.3 Analiză teoretică a algoritmului

În continuare, vom analiza în detaliu principiile matematice care stau la baza filtrării colaborative.

Factorizarea matricilor are ca obiectiv reconstrucția aproximativă a valorilor lipsă din matricea de ratinguri. Astfel, putem prezice cum ar evalua un utilizator un item pe care nu l-a văzut. Procesul se bazează pe aproximarea valorilor necunoscute prin produsul scalar dintre vectorii asociați fiecărui utilizator și fiecărui item. Formula matematică este reprezentată astfel: $R \approx U \cdot V^T$, unde:

- U este matricea utilizatorilor,
- V^T este matricea transpusa a itemilor,

- R este matricea ratingurilor observate;

Pentru a realiza factorizarea matriciilor, se minimizează o **funcție de pierdere (loss function)**, care măsoară cât de bine predicțiile reconstruiesc valorile observate din matrice. Funcția de pierdere uzuală este eroarea pătratică medie (Mean Squared Error – MSE), calculată doar pe ratingurile existente, și este exprimată astfel:

$$\min_{U,V} \sum_{(u,i)\in K} \left(R_{ui} - U_u \cdot V_i^T\right)^2 + \lambda \left(\|U\|_F^2 + \|V\|_F^2\right)$$

Semnificaţii:

- R_{ui} — ratingul observat pentru utilizatorul u și itemul i,
- U_u — vectorul de factori latenti pentru utilizatorul u,
- V_i — vectorul de factori latenti pentru itemul i,
- V_i^T — transpusul vectorului V_i,
- K — mulțimea perechilor (u, i) pentru care există ratinguri observate,
- λ — coeficientul de regularizare care penalizează complexitatea modelului,
- $\|U\|_F^2$ și $\|V\|_F^2$ — normele Frobenius pătrate ale matricilor U și V.

Algoritmii de factorizare a matricii sunt apreciați nu doar pentru performanța lor predictivă, ci și pentru eficiența computațională în scenarii reale, unde datele sunt masive și sparse. Se observă şi faptul că acest algoritm este eficient, atât din punct de vedere al timpului de execuţie, cât şi din cel al memoriei.

Evaluarea acestora:

Complexitatea spațială a modelului

Modelul de factorizare matricială încarcă în memorie:

- matricea utilizatorilor $U \in \mathbb{R}^{m\times k}$, care conține vectorii de factori latenți pentru fiecare dintre cei m utilizatori;
- matricea itemilor $V \in \mathbb{R}^{n\times k}$, care conține vectorii de factori latenți pentru fiecare dintre cei n itemi.

Prin urmare, complexitatea spațială totală este:

$$\mathcal{O}((m+n)\cdot k)$$

Această formulă exprimă numărul total de valori reale (float) necesare pentru a reprezenta embedding-urile tuturor utilizatorilor și itemilor, în funcție de dimensiunea latentă k.

Exemplu numeric:
Presupunem:

- $m = 100{,}000$ utilizatori,
- $n = 10{,}000$ itemi,
- $k = 50$ dimensiunea vectorilor latenți.

Atunci:

$$(100{,}000 + 10{,}000) \cdot 50 = 5{,}500{,}000 \text{ valori reale}$$

Dacă fiecare valoare este stocată ca un `float32` (4 octeți), rezultă o memorie totală de:

$$5{,}500{,}000 \cdot 4 = 22\,\text{MB}$$

Această valoare este remarcabil de mică în comparație cu modelele complexe de tip rețea neuronală adâncă sau metode bazate pe grafuri, care pot necesita sute de megabytes sau chiar gigabytes pentru a reprezenta aceleași informații. Astfel, factorizarea matricii rămâne o soluție extrem de eficientă din punct de vedere al memoriei, fiind adecvată și pentru implementări scalabile în aplicații reale.

Complexitatea temporală a antrenării modelului

Antrenarea unui model de factorizare matricială presupune optimizarea vectorilor de factori latenți din matricile $U \in \mathbb{R}^{m \times k}$ și $V \in \mathbb{R}^{n \times k}$, astfel încât să se minimizeze funcția de pierdere pe toate ratingurile observate.

Folosind o metodă de optimizare stocastică, precum **Stochastic Gradient Descent (SGD)**, complexitatea temporală per **iterare de antrenare** (o trecere completă prin toate ratingurile observate) este proporțională cu:

$$\mathcal{O}(|K| \cdot k)$$

unde:

- $|K|$ este numărul de ratinguri observate (dimensiunea setului de antrenare),
- k este dimensiunea spațiului latent.

Fiecare pereche utilizator-item este procesată individual, iar actualizarea gradientului are complexitatea $\mathcal{O}(k)$ deoarece implică doar vectorii latenți de lungime k corespunzători acelui utilizator și acelui item.

Exemplu numeric:
Presupunem:

- $|K| = 10^7$ ratinguri observate,

- $k = 50$,
- $T = 20$ iterări de antrenare (treceri complete prin setul de date).

Atunci complexitatea totală a antrenării este:

$$\mathcal{O}(T \cdot |K| \cdot k) = \mathcal{O}(20 \cdot 10^7 \cdot 50) = \mathcal{O}(10^9)$$

Adică aproximativ 1 miliard de operații, ceea ce este considerat eficient în comparație cu antrenarea rețelelor neuronale adânci, care implică parametri suplimentari, funcții de activare și propagare directă și inversă. Optimizarea poate fi paralelizată eficient pe GPU sau distribuită pe mai multe procesoare, datorită independenței relative dintre actualizările ratingurilor.

Concluzie: Modelul MF oferă o bună scalabilitate temporală, fiind aplicabil și în sisteme cu milioane de utilizatori și itemi, cu costuri computaționale rezonabile.

2.4 Algoritmi Hibrizi

Algoritmii hibrizi reprezintă o soluție versatilă în cadrul sistemelor de recomandare, fiind concepuți pentru a îmbina mai multe surse de informații și metode de predicție. Scopul acestora este de a depăși limitările specifice fiecărei abordări individuale (cum ar fi filtrarea colaborativă sau cea bazată pe conținut), obținând o acuratețe mai mare și o adaptabilitate sporită în contexte diverse.

2.4.1 Filtrare bazată pe conţinut

Filtrarea bazată pe conținut funcționează prin asocierea caracteristicilor itemilor cu preferințele exprimate anterior de utilizator. Într-un astfel de sistem, fiecare item este descris printr-un set de atribute (ex. limbă, categorie, etichete semantice, emoții etc.), iar istoricul utilizatorului este analizat pentru a extrage trăsături comune ale itemilor apreciați. Pe baza acestei analize, sistemul construiește un profil de preferințe care este ulterior folosit pentru a evalua alți itemi.

Scorul de compatibilitate între un item și un utilizator este obținut prin măsurarea gradului de suprapunere între atributele itemului și cele preferate de utilizator. Astfel, un item care corespunde mai multor criterii relevante (cum ar fi limba favorită sau emoțiile dominante) va primi un scor mai mare. Această etapă poate fi privită ca o componentă deterministă, bazată pe potriviri directe între profiluri.

2.4.2 Realizarea scorurilor de preferință

Pe lângă analiza conținutului, scorul total atribuit unui item se construiește și în funcție de istoricul de interacțiuni. Se definesc mai multe tipuri de acțiuni prin care un utilizator poate semnala interesul față de un item, precum vizualizarea, aprecierea, comentariul sau marcarea pentru utilizări ulterioare. Fiecărei acțiuni îi este asociată o pondere care reflectă importanța sa în procesul de recomandare.

În mod generic, scorul de preferință poate fi exprimat ca o funcție de tipul:

$$\text{Score}(u, i) = \sum_{a \in A} w_a \cdot f_a(u, i) + \sum_{c \in C} w_c \cdot g_c(u, i)$$

unde:

- A este mulțimea acțiunilor posibile (ex. like, comment, view, bookmark),
- $f_a(u, i)$ este funcția care măsoară frecvența sau intensitatea acțiunii a efectuate de utilizatorul u asupra itemului i,
- C este mulțimea caracteristicilor relevante (ex. limbă, temă, emoție),
- $g_c(u, i)$ este funcția care măsoară potrivirea caracteristicii c între item și preferințele utilizatorului,
- w_a și w_c sunt coeficienți de importanță, ajustabili.

2.4.3 Realizarea algoritmului hibrid

Algoritmul hibrid propus combină componentele colaborative și cele bazate pe conținut printr-un mecanism de agregare. Scorul final pentru fiecare item este calculat ca o medie ponderată între scorul obținut din filtrarea colaborativă și cel rezultat din analiza conținutului. Formula generală a agregării poate fi exprimată astfel:

$$\text{Score}_{\text{final}}(u, i) = \alpha \cdot \text{Score}_{\text{CF}}(u, i) + (1 - \alpha) \cdot \text{Score}_{\text{CB}}(u, i)$$

unde:

- $\text{Score}_{\text{CF}}(u, i)$ este scorul colaborativ, calculat pe baza preferințelor altor utilizatori similari,
- $\text{Score}_{\text{CB}}(u, i)$ este scorul bazat pe conținut, rezultat din potrivirea profilului utilizatorului cu atributele itemului,
- $\alpha \in [0, 1]$ este un coeficient de echilibru care stabilește importanța relativă a celor două componente.

În practică, valoarea lui α poate fi ajustată dinamic în funcție de disponibilitatea datelor. De exemplu, în cazul în care un utilizator este nou și nu are suficiente interacțiuni înregistrate, componenta colaborativă poate fi redusă, iar sistemul se va baza mai mult pe analiza conținutului. Pe măsură ce utilizatorul interacționează cu platforma, scorul colaborativ devine din ce în ce mai relevant.

Această metodă de combinare este justificată și în literatura de specialitate. Potrivit studiului lui Burke (2002)[2], agregarea este una dintre cele mai eficiente strategii de hibridizare, permițând sistemului să beneficieze simultan de punctele tari ale fiecărui algoritm. Alte studii ulterioare, precum Zhang et al. (2019)[20], evidențiază avantajele algoritmilor hibrizi în ceea ce privește precizia, diversitatea și robustețea recomandărilor în medii reale.

Prin integrarea acțiunilor explicite, a caracteristicilor semantice ale itemilor și a relațiilor colaborative dintre utilizatori, algoritmul hibrid devine un mecanism eficient și scalabil de personalizare, adaptat unei game variate de scenarii și profiluri comportamentale.

Capitolul 3

Dezvoltarea Aplicaţiilor Web

Aplicațiile web reprezintă software accesibil prin intermediul unui browser, fără a necesita instalare locală. Acestea au apărut în urma dezvoltarii World Wide Web-ului în 1991. WWW a fost o inovaţie propusă de Tim Berners-Lee, cercetător la CERN, în 1989. Cercetătorul a dorit să construiască un sistem de distribuţie a datelor şi al informaţiilor pentru calculatoarele legate într-o reţea.

Iniţial, paginile web erau statice, acestea utilizând doar HTML. Introducerea în 1995 a JavaScript şi a CSS a dus la crearea unor aplicaţii web dinamice şi interactive, acestea începând să funcţioneze şi să arate ca aplicaţiile uzuale de desktop direct dintr-un browser.

Această evoluție a condus la definirea conceptului de aplicație web, înțeles ca un software care funcționează prin intermediul internetului și permite interacțiunea directă cu utilizatorul, procesarea datelor și comunicarea cu servere aflate la distanță [10]. Astăzi, aplicațiile web sunt integrate într-o varietate de domenii, de la comerț, educație și divertisment, până la administrație publică, sănătate și sectorul bancar, devenind esențiale pentru funcționarea eficientă a multor activități cotidiene.

3.1 Noţiuni de bază

Pentru a înțelege modul în care sunt dezvoltate aplicațiile web moderne, este importantă prezentarea principalelor concepte fundamentale care stau la baza funcționării acestora.

Arhitectura client-server reprezintă modelul de bază al comunicării într-o aplicație web. În această arhitectură, clientul (de obicei un browser web) trimite cereri către un server, care procesează aceste cereri și returnează un răspuns. Acest model este esențial pentru funcționarea aplicațiilor distribuite, permițând ca logica aplicației și stocarea datelor să fie gestionate pe server, în timp ce utilizatorul interacționează printr-o interfață grafică în browser.

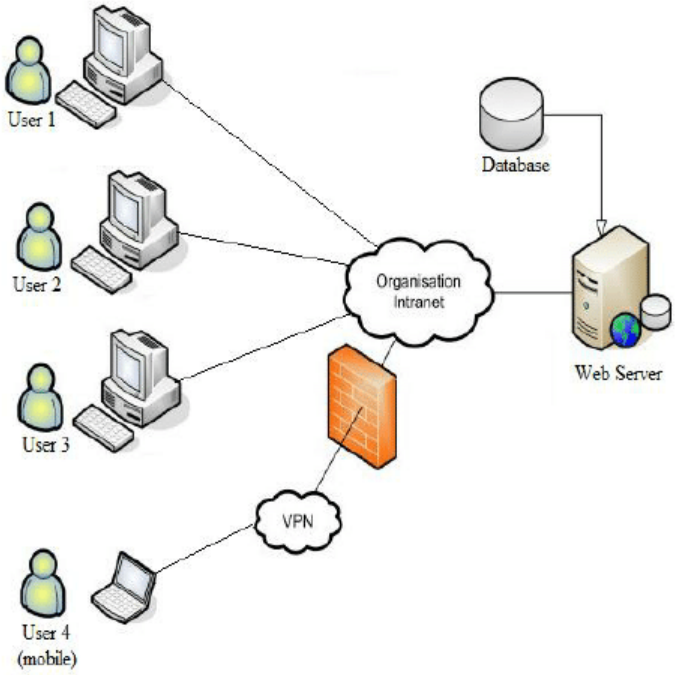

Figura 3.1: Arhitectura Client-Server

Protocolul HTTP (HyperText Transfer Protocol) este standardul utilizat pentru comunicarea între client și server. El definește metode precum `GET` (pentru obținerea de resurse), `POST` (pentru trimiterea de date către server), `PUT` (pentru actualizarea datelor existente) și `DELETE` (pentru ștergerea datelor). Variantele moderne ale acestui protocol folosesc **HTTPS**, o versiune securizată care criptează informațiile transmise, asigurând confidențialitatea și integritatea datelor.

HTML (HyperText Markup Language) este limbajul standard pentru structura unei pagini web. Acesta definește elementele vizuale precum titluri, paragrafe, liste sau formulare. **CSS (Cascading Style Sheets)** este folosit pentru a stiliza aceste elemente – de exemplu, pentru a schimba fontul, culoarea, poziționarea sau dimensiunea. În completare, **JavaScript** este limbajul care adaugă funcționalitate și interactivitate paginii, permițând răspunsuri la acțiunile utilizatorului și modificarea dinamică a conținutului fără a reîncărca pagina.

Delimitarea între front-end și back-end este o altă componentă esențială. Front-end-ul reprezintă partea vizibilă a aplicației – ceea ce utilizatorul vede și cu care interacționează. Este implementat în principal cu HTML, CSS și JavaScript. Back-end-ul se ocupă cu logica aplicației, procesarea datelor și interacțiunea cu baza de date. Acesta rulează pe server și poate fi scris în diverse limbaje, precum Node.js, Python, PHP sau Java.

3.2 Importanţa Regulilor de Design

În dezvoltarea aplicațiilor web moderne, design-ul nu se referă doar la estetică, ci și la modul în care utilizatorii interacționează cu interfața, cât de intuitivă este navigarea și cât de eficient este livrat conținutul. Respectarea unor reguli de design bine definite contribuie esențial la crearea unei experiențe de utilizare coerente, accesibile și plăcute [12].

Design-ul centrat pe utilizator (User-Centered Design – UCD) pune accent pe nevoile reale ale utilizatorilor, permițând dezvoltatorilor să creeze aplicații funcționale și ușor de folosit. Acest proces implică înțelegerea publicului țintă, prototiparea iterativă și testarea continuă. Conform lui Krug, un design bun „Nu-l face pe utilizator să gândească prea mult" [9].

Principiile de design vizual – precum contrastul, alinierea, proximitatea și repetarea – contribuie la organizarea logică a informației și la evidențierea elementelor importante de pe pagină. Acestea sunt derivate din teoria gestaltistă a percepției vizuale și sunt fundamentale pentru lizibilitate și claritate [11].

Regulile de accesibilitate (exemplu: WCAG – Web Content Accessibility Guidelines) asigură că aplicația poate fi utilizată de cât mai mulți oameni, inclusiv persoane cu dizabilități. Acestea presupun folosirea unor culori cu contrast ridicat, etichete clare pentru formulare, suport pentru navigare cu tastatura și compatibilitate cu cititoare de ecran [19].

Mobile-first design este o abordare modernă care pornește de la ideea că aplicația va fi utilizată în primul rând de pe dispozitive mobile. Această metodologie optimizează încărcarea, navigarea și interacțiunea pe ecrane mici, fiind esențială într-un context global în care peste 50

Reguli de consistență și stil sunt importante mai ales în aplicații mari. Un sistem de design (ex: Material Design, Bootstrap) oferă un set unitar de culori, fonturi, dimensiuni și comportamente, care garantează uniformitatea vizuală a întregii aplicații, reducând confuzia și timpul de dezvoltare.

Respectarea acestor principii contribuie direct la creșterea satisfacției utilizatorilor, reducerea ratei de abandon și îmbunătățirea performanțelor aplicației.

3.3 Diferențe între dezvoltarea web clasică și cea bazată pe framework

Pe parcursul timpului au apărut o multitudine de metode de a realiza o aplicaţie web, de la simplu cod scris intr-un fisier HTML şi rulat în browser, până la framework-uri unde codul este compilat pe server şi utilizatorii au acces direct la pagină prin

intermediul acestuia.

Folosirea unui framework pentru dezvoltarea unei aplicaţii web în locul dezvoltării doar cu HTML, CSS şi JS, are multiple avantaje, mai ales când se doreşte construirea unei aplicaţii mari. Această metoda oferă, în primul rând, o modalitate mai bună pentru structurarea şi organizarea proiectului, fiecare framework având un set de reguli clare de împărţire a modulelor, a componentelor şi a fişierelor folosite. Totodată un alt avantaj este evidenţiat de existenţa unor funcţionalităţi necesare oricărui website (routing, state management, formulare, validări, etc), astfel acestea nu trebuie reconstruite, ceea ce economiseşte timp în procesul dezvoltării. Scalabilitatea mai uşoară şi reutilizarea componentelor, permite programatorilor să dezvolte aplicaţii mari mai uşor şi evită repetarea liniilor de cod excesivă pentru nişte componente des întâlnite într-un site (butoane, chenare, chestionare, header, footer şi multe altele). Prin intermediul unui framework se pot implementa cu uşurinţă metode de testare şi metode de securitate, care sunt actualizate mereu la ultimele versiuni şi previn mai rapid erori, bug-uri şi leak-uri informaţionale ce pot exista în aplicaţie.

Astfel, utilizarea tehniciilor noi apărute odată cu framework-urile pentru aplicaţiile web sunt recomandate pentru proiecte complexe, economisind timp în proiectarea acestora şi oferind un *workflow* simplu şi comod programatorului. Totuşi metoda de dezvoltare web clasică este adecvată proiectelor mici, cum ar fi website-uri de prezentare, sau pentru persoanele la început de drum în programarea web.

Framework-uri populare:

- Front-end (Interfața utilizatorului):
  - **React.js** – este o bibliotecă JavaScript creată de Facebook şi folosită pentru dezvoltarea de aplicații **SPA** (Single Page Applications).
  - **Next.js** – este construit peste React, adaugă server-side rendering, static generation şi routing, permițând programatorului să creeze o aplicaţie web cu mai multe pagini.
  - **Angular** – este un framework complet dezvoltat de Google, folosit în aplicații enterprise.
  - **Vue.js** – este un framework progresiv, ușor de învățat și flexibil.
- Back-end (Logica server-ului):
  - **Node.js** – este un mediu de execuție pentru JavaScript, construit pe motorul V8 de la Google Chrome. Node.js permite rularea codului JavaScript pe server, fiind foarte popular datorită vitezei de execuție și a modelului său asincron bazat pe evenimente. Acesta este ideal pentru aplicații în

timp real, cum ar fi chat-uri sau jocuri online, și oferă un ecosistem vast prin managerul de pachete `npm`, ceea ce facilitează dezvoltarea rapidă și modulară a aplicațiilor [18].

- **Flask** – este un microframework pentru limbajul Python, cunoscut pentru simplitatea și flexibilitatea sa. Flask permite dezvoltatorilor să creeze rapid aplicații web și API-uri RESTful, oferind libertate completă în structurarea codului. Deși este minimalist, poate fi extins ușor prin extensii precum SQLAlchemy (pentru baze de date) sau Flask-Login (pentru autentificare). Este potrivit atât pentru proiecte mici cât și pentru aplicații mai complexe, dacă este bine organizat [5].
- **Spring Boot** – este un framework Java dezvoltat de echipa Spring, care simplifică procesul de creare a aplicațiilor enterprise prin eliminarea configurației complexe. Spring Boot vine cu suport integrat pentru conectarea la baze de date, securitate, autentificare și dezvoltarea de API-uri REST. Este ideal pentru aplicații robuste, scalabile, care necesită un nivel ridicat de organizare și performanță, fiind folosit pe scară largă în mediul corporativ [8].
- **Django** – este un framework complet pentru Python, bazat pe modelul MTV (Model-Template-View), care oferă o structură clară și un set de funcționalități integrate: panou de administrare, rutare automată, ORM (mapare obiect-relațională), autentificare, și protecție împotriva atacurilor web comune. Django respectă principiul DRY (Don't Repeat Yourself), facilitând dezvoltarea rapidă și sigură a aplicațiilor web complexe, într-un mod organizat și scalabil [6].

3.4 Framework: React - Next.js - Node.js

(- De ce acest mod de a lucra)

1. **React.js – Interfața aplicației (UI)**
   React este o bibliotecă JavaScript dezvoltată de Facebook, folosită pentru construirea interfețelor de utilizator dinamice. React se bazează pe conceptul de componente reutilizabile, ceea ce permite structurarea aplicației în elemente modulare, ușor de întreținut. Folosirea *Virtual DOM* optimizează performanța, actualizând doar elementele modificate, nu întreaga pagină.

2. **Next.js – Framework pentru React cu capabilități full-stack**
   Next.js este un framework bazat pe React care adaugă funcționalități importante pentru dezvoltarea aplicațiilor moderne:

- **Server Side Rendering (SSR)** – randare pe server pentru pagini optimizate SEO;
- **Static Site Generation (SSG)** – pagini statice generate la build-time;
- **Routing automat** – fiecare fișier din folderul `/pages` devine o rută web;
- **API Routes** – permite scrierea logicii de backend direct în aplicația Next.js, fără a avea un server separat.

(De ce Next.js/React şi nu Angular?)

Caracteristică	**Angular**	**Next.js**
Tip	Framework complet	Framework bazat pe React
Limbaj principal	TypeScript	JavaScript (cu suport pentru TypeScript)
Renderizare	Client-side rendering (CSR), cu suport SSR limitat	Server-side rendering (SSR), static & hybrid
Routing	Gestionat prin sistem propriu declarativ	Routing automat bazat pe fișiere în `/pages`
Arhitectură	Monolitică, all-in-one	Modular, extensibil cu librării externe
Learning curve	Mai ridicată (complexitate mai mare)	Mai accesibil (React + Next = curba mai lină)
Utilizare	Aplicații enterprise, structuri mari	Aplicații rapide, scalabile, SEO-friendly
Dezvoltat de	Google	Vercel (open-source)

Tabela 3.1: Comparație între Angular și Next.js

3. **Node.js – Mediu de execuție JavaScript pe server**
   Node.js permite rularea de cod JavaScript pe server, în afara browserului. Este rapid și eficient, datorită motorului V8. În contextul unei aplicații web, Node.js este folosit pentru:

   - scrierea de servere web;
   - gestionarea rutelor backend;
   - conectarea la baze de date;
   - trimiterea de răspunsuri către client.

3.5 RestAPI şi Baze de Date: Flask - MongoDB

Flask este utilizat aici ca framework minimalist pentru Python, care permite definirea rapidă a unor rute REST și a logicii asociate acestora. Structura clară a unui API REST presupune folosirea metodelor standard HTTP: `GET` (pentru citire), `POST` (pentru creare), `PUT/PATCH` (pentru actualizare) și `DELETE` (pentru ștergere). Flask permite definirea acestor rute într-un mod clar și modular, facilitând testarea și scalarea aplicației [5].

MongoDB este o bază de date NoSQL, orientată pe documente, în care datele sunt stocate sub formă de obiecte JSON (intern, în format BSON). Aceasta oferă flexibilitate ridicată în gestionarea structurii datelor, permițând adăugarea ușoară de câmpuri noi, fără a necesita un model rigid de tip tabelar, cum se întâmplă în bazele de date relaționale. MongoDB este ideală pentru aplicații moderne, cu date dinamice, precum rețele sociale, aplicații mobile sau platforme de conținut. De asemenea, se integrează ușor cu Python, folosind biblioteca `pymongo` sau framework-uri ODM precum `MongoEngine`.

În arhitectura propusă, Flask gestionează rutele API și interacțiunea cu baza de date MongoDB, oferind datele către front-end prin răspunsuri JSON. Această separare permite o dezvoltare paralelă a celor două componente (UI și server), îmbunătățind scalabilitatea, mentenabilitatea și testabilitatea aplicației.

Capitolul 4

PoemSociety - Rețea socială dedicată poeziilor şi scriitorilor

4.1 Analiza cerinţei

4.1.1 Descrierea problemei

În trecut, oamenii obişnuiau să petreacă mai mult timp citind, dar odată cu apariţia platformelor de streaming şi reţelelor sociale, aceştia au început să piardă interesul pentru literatură. Observăm în statistica din figura 4.1, cât a scăzut interesul în Germania pentru citit din 2019 până în 2024.

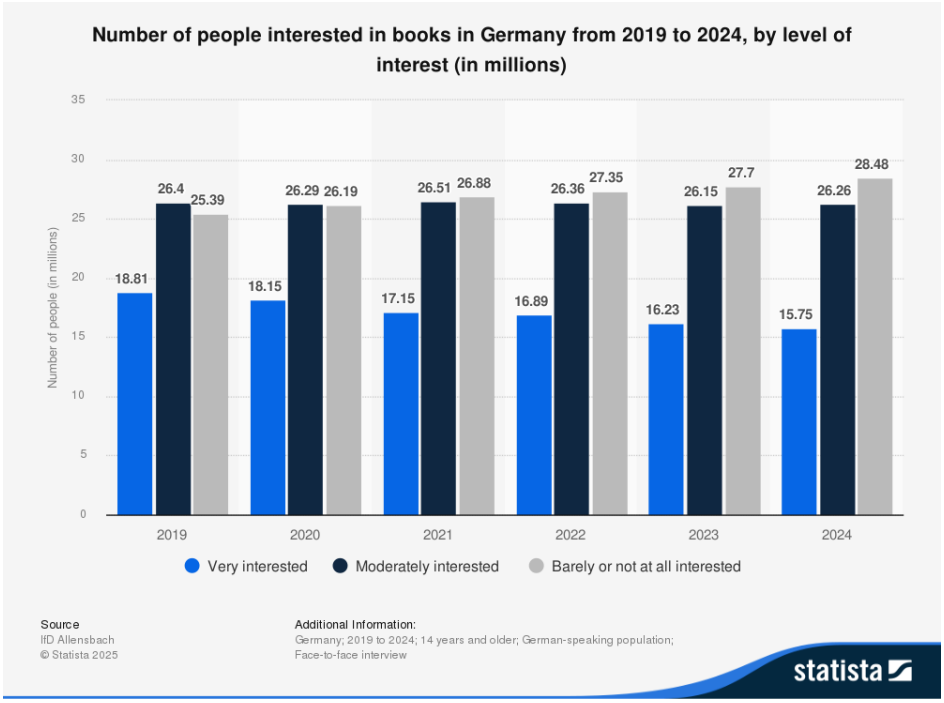

Figura 4.1: Interesul pentru cărți în Germania între 2019–2024 [7]

Totuşi nimeni nu-i poate forţa să acorde din nou interes literaturii, dar se poate încerca adaptarea acesteia în rutina oamenilor. Totodată autorii au fost afectaţi de lipsa interesului asupra operelor de artă, fiind voci neauzite ce vor să transmită lumii din nou emoţii şi simţiri diferite de cele trăite în filme şi negăsite în scroll-uri interminabile pe reţelele sociale.

4.1.2 Soluţia adusă de algoritmul propus

Cum este menţionat anterior, un mod de a reintroduce o parte a literaturii în vieţile oamenilor este prin adaptarea acesteia în noile forme ale lumii. Apariţia e-Book-urilor poate fi considerată o abordare de o readuce în rutina umană, dar în zilele noastre persoanele, în special copii şi adolescenţii, stau ataşaţi de reţele sociale precum TikTok şi Instagram, care oferă un feed personalizat cu conţinut pe placul acestora. Asftel o parte a literaturii poate fi integrată într-un mod asemănător, prin construcţia unei platforme sociale cu poezii, unde diferiţi poeţi, la începutul drumului sau deja cunoscuţi, pot publica scrierile lor, iar sistemul de recomandare (asemanator cu cel regasit pe TikTok sau Instagram Reels) sugerează clienţilor conţinut pe gustul lor. Aşadar o astfel de platformă poate creşte interesul oamenilor asupra literaturii şi ajută scriitorii să se facă din nou auziţi.

4.2 Aplicaţie – PoemSociety

Aplicaţia **PoemSociety**, prin care este implementată soluţia propusă, foloseşte tehnicile descrise în capitolele anterioare. Dezvoltarea aplicaţiilor web reprezintă interfaţa principală de interacţiune dintre utilizator şi sistemele de pe partea de back-end – sistemul de recomandare şi modulul de analiză AI a postărilor (poeme / poezii). În continuare, este descrisă construcţia aplicaţiei din punct de vedere teoretic şi practic, atât pentru partea de front-end, cât şi pentru cea de back-end.

4.3 Funcţionalităţi

O reţea socială are în general 3 funcţionalităţi de bază: like, follow şi post. Pe platforma PoemSociety sunt regăsite următoarele:

- **Logare**: Permite autentificarea utilizatorilor existenți prin introducerea unui nume de utilizator și a unei parole. După logare, sunt activate funcționalitățile rezervate utilizatorilor înregistrați.
- **Delogare**: Permite utilizatorilor activ să se deconecteze de la platformă şi să deactiveze funcționalitățile rezervate utilizatorilor înregistrați.

- **Înregistrare**: Oferă posibilitatea creării unui cont nou prin completarea unui formular cu date de bază (email, nume, prenume, username, parolă, gen, data naşterii). După înregistrare, utilizatorul se poate autentifica.
- **Apreciere**: Utilizatorul poate marca un poem ca „plăcut" printr-un click pe butonul de like. Aprecierea este salvată în baza de date și este luată în considerare în sistemul de recomandare.
- **Comentariu**: Utilizatorii autentificați pot adăuga comentarii la poezii, exprimându-și opiniile sau reacțiile. Comentariile sunt afișate cronologic sub fiecare poem. Comentariul afectează preferinţele utilizatorului de asemenea.
- **Salvare**: Prin opțiunea de bookmark, un utilizator poate salva un poem preferat într-o listă personală, accesibilă ulterior în secțiunea „Bookmarks".
- **Urmărire**: Permite unui utilizator să urmărească activitatea altui utilizator. Poeziile postate de utilizatorii urmăriți apar în feed-ul personalizat.
- **Căutare – Utilizator sau Poem**: Bara de căutare permite filtrarea poemelor după cuvinte cheie, autori sau titluri, precum și identificarea altor utilizatori după username.
- **Vizualizare Poem și Feed**: Utilizatorii pot parcurge poemele publicate fie sub formă de feed continuu personalizat după preferinţe, fie individual, accesând pagini dedicate fiecărui poem.
- **Publicare Poem**: Prin intermediul unui editor dedicat, utilizatorii pot posta creații proprii. Formularul include titlul, conținutul, autor şi data publicării poeziei.
- **Ștergere Poem**: Utilizatorii autentificați își pot șterge propriile poezii din profil. Operația este finală și elimină conținutul din toate secțiunile aplicației.
- **Contor vizualizări**: Fiecare accesare a unui poem este înregistrată. Numărul total de vizualizări este afișat public și poate influența recomandările sau popularitatea poeziei.

Fiecare dintre acestea are o logică programată, atât pe partea de front-end, cât şi pe cea de back-end. Interfaţa este adaptată pentru fiecare funcţionalitate şi se modifică dinamic, fără refresh, odată cu acţiunile utilizatorului. Pe partea de back-end, toate activităţile utilizatorului sunt analizate AI, prelucrate şi înregistrate în baza de data dacă este cazul, prin intermediul unui API şi a fetch-urilor din front-end.

4.3.1 Implementare Front-End

Conceput pe baza framework-ului **Next.js** (dezvoltat peste React), front-end-ul a fost realizat cu uşurinţă prin crearea de componente `.jsx`, care seamănă cu tag-urile clasice HTML şi pot fi stilizate cu CSS, respectiv controlate dinamic prin JavaScript.

Interfaţă Grafică – Interfaţa Utilizatorului / Experienţa Utilizatorului (UI/UX)

Pentru a oferi o experienţă UI/UX de calitate, s-a acordat atenţie atât aspectelor vizuale – precum prezentarea, cromatica, designul minimalist – cât şi construirii unui mediu în care fiecare funcţionalitate este clară şi uşor accesibilă. Paleta de culori (vezi Figura 4.2) a fost aleasă astfel încât să păstreze atenţia utilizatorului asupra conţinutului poetic, dar în acelaşi timp să exprime vitalitate şi coerenţă vizuală.

Interfaţa este aerisită şi intuitivă, având secţiuni clar delimitate şi un flux de navigare eficient, în care orice funcţionalitate importantă poate fi accesată în maximum două acţiuni. Designul reflectă tematica aplicaţiei – poezia – prin fundaluri neutre, spaţii generoase şi carduri elegante pentru postări, toate contribuind la crearea unui spaţiu emoţional potrivit lecturii şi reflecţiei.

La interacţiunea cu postările (like, comment, bookmark, search etc.), utilizatorul primeşte feedback vizual imediat, prin animaţii discrete şi modificări contextuale ale conţinutului. Toţi aceşti factori contribuie la o experienţă plăcută, coerentă şi modernă de utilizare a aplicaţiei.

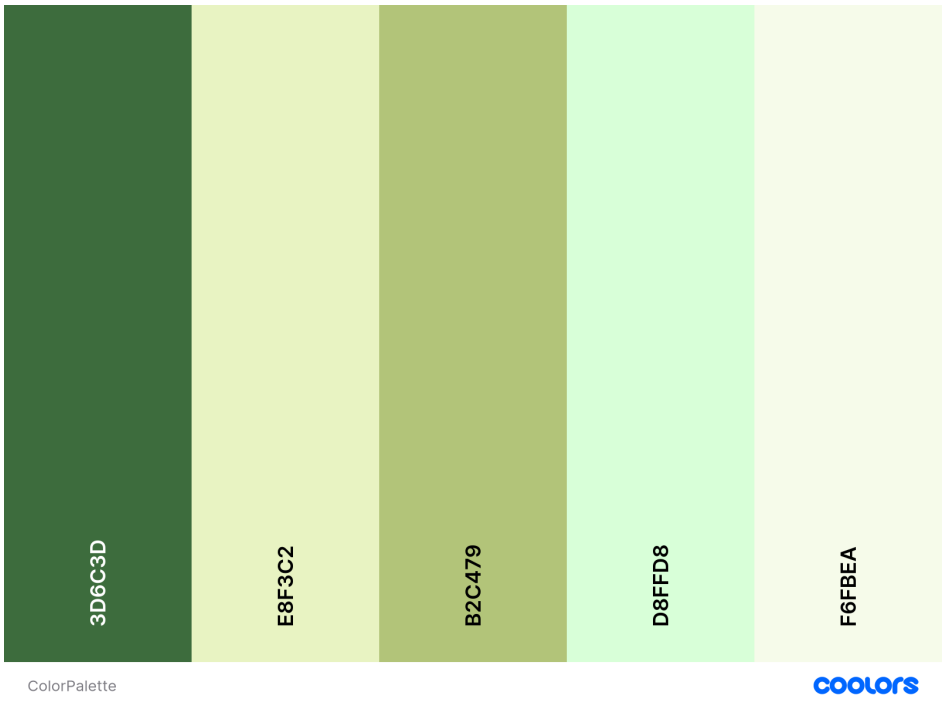

Figura 4.2: Paleta de culori din aplicaţie

Pagina principală introduce utilizatorul direct în conţinutul aplicaţiei, fiind structurată în trei secţiuni esenţiale (vezi Figura 4.3):

- **Header-ul**, care conţine logo-ul aplicaţiei şi butonul de logare;
- **Bara de navigare**, ce oferă acces rapid către toate funcţionalităţile importante;
- **Fereastra de afişare a poeziilor**, care prezintă postări individuale şi permite interacţiuni sau navigarea către următoarea poezie.

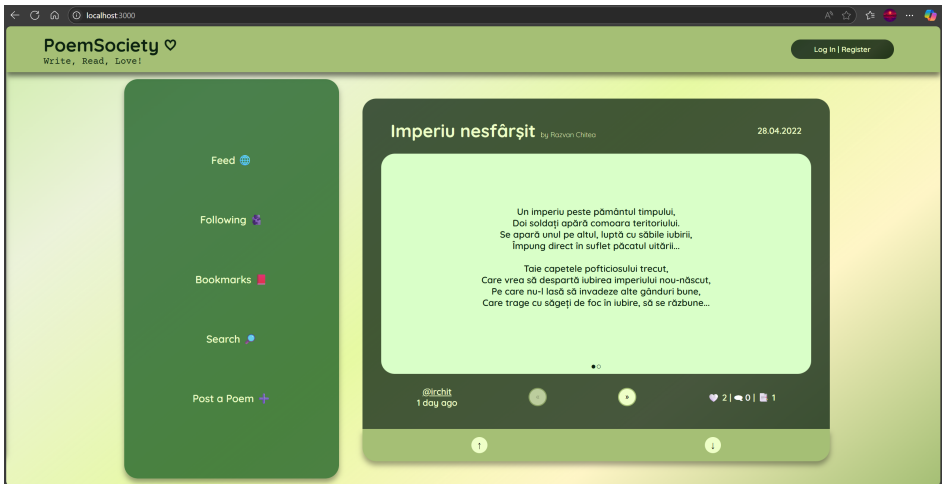

Figura 4.3: Pagina principală – PoemSociety

Celelalte pagini – `/login`, `/new_post`, `/poem/[id]`, `/[username]` (vezi Figura 4.4), `/bookmarks`, `/following`, `/search` – păstrează acelaşi design coerent şi accesibil, centrat pe conţinutul poetic.

Figura 4.4: Pagina utilizatorului autentificat – irchit

Cascading Style Sheets (CSS)

Un design modern şi atractiv necesită CSS. Utilizarea style sheet-urilor în această aplicaţie a fost necesară pentru dezvoltarea unei interfeţe grafice pe placul utilizatorului, oferind prin acest intermediu un contrast bun între text şi fundal, spaţiere consistentă şi dinamică paginii prin animaţii. Fiecare pagină are asociat câte un fişier specific, dar poate reutiliza şi clasele din celălalte CSS-uri. Spre exemplu butoanele de navigare ↑ şi ↓ folosesc în toate paginile unde apar aceiaşi clasă de stilizare (vezi Figura 4.5). De asemenea, a fost definit și un fișier global pentru elementele comune (font, fundal, reset CSS etc.).

```css
.button_next_page{
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    width: 4vh;
    height: 4vh;
    background-color: rgb(151, 255, 191);
    border: 2px solid;
    border-color: var(--olivine);
    border-radius: 100%;
    background-color: var(--olivine-light);
    transition: all 0.3s ease-in-out;
    cursor: pointer;
    user-select: none;
}

.button_next_page:hover{
    background-color: rgb(112, 165, 133);
    border-color: #ffffff10;

}

.button_next_page:disabled:hover{
    opacity: 0.7;
}

.button_next_page:disabled{
    opacity: 0.7;
    pointer-events: none;
    cursor: not-allowed;
}
```

Figura 4.5: Clasă CSS pentru `button next page`

Cu ajutorul CSS-ului, interfaţa este responsive, folosind dimensiuni relative la înălţimea şi lăţimea ferestrei (vh şi vw) pentru fiecare clasă şi font-size. De asemenea animaţii pentru evidenţierea link-urilor şi a butoanelor este realizată prin `.classname:hover` şi oferă dinamică acestora. Alte animaţii au fost realizate şi prin javascript împreună cu proprietăţile `transition` şi `transform`, spre exemplu pe pagina `/login` în momentul trecerii de la cardul de logare la cel de înregistrare şau vice-versa.

Next.js – Componente şi JavaScript (JS)

Aplicația este construită în framework-ul Next.js, care permite organizarea logică a paginilor și componentelor printr-un sistem de directoare. Fiecare pagină are un fișier `page.js`, iar componentele vizuale reutilizabile sunt organizate în folde-

rul `_components`, grupate în funcție de funcționalitate (exemplu: `Feed`, `Login`, `PostAPoem`, `UserPage`). Acest mod de structurare permite o separare clară între logică, interfață și funcții auxiliare (vezi Figura 4.6).

Spre exemplu, componenta `Post.js` este refolosită în mai multe instanţe: `Posts.js`, `poem/[id]/page.js` şi `PostsFollowing.js`. Cu ajutorul datelor transmise către diferite componente, aceasta a putut primi o poezie diferită la fiecare creare, indiferent din ce pagină sau altă componentă a fost generată. Astfel, arhitectura de tip componentă a facilitat reutilizarea logicii şi afişarea dinamică a conţinutului poetic, fără a fi nevoie de duplicarea codului sursă.

Figura 4.6: Structura Front-End Next.js – PoemSociety

Un alt avantaj al Next.js-ului este că permite folosirea celor două hook-uri importante din React: `useState` și `useEffect`. `useState` este utilizat pentru a păstra starea internă a componentelor – de exemplu, titlul și conținutul poeziei introduse de utilizator. `useEffect` este folosit pentru a efectua acțiuni atunci când componenta se montează, cum ar fi inițializarea datelor, verificarea autentificării sau trimiterea cererilor API.

Pagina `/new_post`, unde utilizatorul poate publica un poem nou, are o astfel de implementare clară şi evidentă, unde ambele hook-uri sunt folosite pentru obţinerea utilizatorului (din cookies), salvarea stării formularului şi pentru trimiterea datelor prin `fetch` în momentul publicării (vezi Figura 4.7). Metoda `fetch` este utilizată pentru a trimite o cerere de tip `POST` către serverul Flask, unde datele sunt procesate şi salvate. Conţinutul este convertit în format `JSON`, iar în caz de succes, utilizatorul este redirecţionat automat.

Acest mod de interacţiune asigură o legătură directă între client şi server, fără a fi nevoie de reîncărcarea completă a paginii, contribuind astfel la fluiditatea experienţei. Combinarea componentelor JSX cu logica JavaScript și interacțiunea asincronă cu back-end-ul oferă aplicaţiei un comportament modern, dinamic şi scalabil.

```
    const [title, setTitle] = useState("");
    const [author, setAuthor] = useState("");
    const [date, setDate] = useState(null);
    const [content, setContent] = useState("");
    const [userActive, setUser] = useState(null);

    const handleSubmit = async () => {
        if (title.length <= 0 && author.length <= 0 && !date && content <=
0){
            alert("Please fill out Poem content...");
            return;
        }
        const user = userActive;
        if (!user){
            alert("Must be logged in... redirect to login...");
            window.location.href = "/login";
        }
        const new_post = JSON.stringify({
            title: title,
            author: author,
            created_at: formatDateToDDMMYYYY(date),
            content: content.replace(/\n\n/g, '§§').replace(/\n/g, '§'),
            user: user.username,
            posted_at: new Date().toISOString()
        })

        try {
        const res = await fetch("http://localhost:5000/poems", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: new_post
        });

        if (!res.ok) {
            const error = await res.json();
            alert("Error posting poem: " + (error.message || error.error));
            return;
        }
        window.location.href = "/" + user.username;

    } catch (err) {
        alert("Unexpected error: " + err.message);
    }
    }

    useEffect(() => {
        const user = getUserFromCookie();
        setUser(user);
        if (!user) {
            alert("Must be logged in... redirect to login...");
            window.location.href = "/login";
        }
    }, []);
```

Figura 4.7: `Editor.js` – fragment de cod

4.3.2 Implemetare Back-End

În următoarele secţiuni este descrisă partea de back-end a aplicaţiei, cum funcţionează şi în ce este construită. Baza de Date este realizată în MongoDB, RestAPI-ul folosit este construit în FlaskAPI şi Python, iar Sistemul de recomandare hibrid bazat pe conţinut şi pe filtrare colaborativă dintre utilizatori împreună cu algoritmii de rating sunt dezvoltate tot în Python.

Baza de Date

Structura bazei de date a fost proiectată în stil document-oriented, folosind MongoDB. Fiecare colecție (ex: Utilizator, Poezie, Likes, Comments) este echivalentă cu o entitate logică a aplicației, iar relațiile sunt gestionate prin referințe (user_id, poem_id etc.). Este implementat un model flexibil care permite extinderea facilă cu noi atribute, fiind ideal pentru date semi-structurate precum analize de emoții sau preferințe lingvistice. De asemenea, formatul în care sunt salvate datele în MongoDB este ideal pentru lucrul cu python (back-end) şi javascript (front-end), deoarece parsarea datelor de tip JSON este imediată în aceste două limbaje.

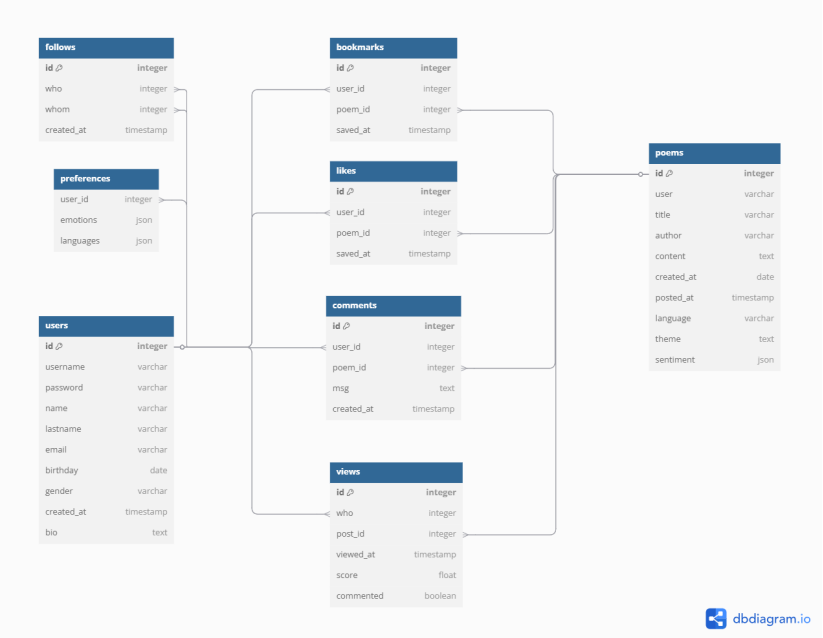

Figura 4.8: Structura Bazei de Date - PoemSociety

Analiza AI a Postărilor şi Sistemul de Recomandare

Analiza AI a Postărilor. Pentru a sprijini personalizarea interacțiunii utilizatorilor cu aplicația, fiecare poezie publicată este supusă unui proces automat de analiză semantică şi afectivă. Acest proces implică mai multe etape succesive:

- detectarea limbii dominante folosind probabilități lexicale pe baza unui set predefinit de vocabular specific limbii;
- extragerea celor mai probabile emoții prezente în text, utilizând o clasificare multi-label alimentată de un model AI antrenat pe seturi multilingve;
- identificarea temei poeziei printr-o analiză a frecvenței și co-apariției cuvintelor semnificative (keywords);

Pentru fiecare poezie, sunt returnate: limba principală, un vector cu scorurile celor mai probabile 3 emoții și un fragment recurent identificat ca temă. Aceste atribute sunt salvate în baza de date, în câmpul dedicat analizei (`analysis`). Astfel, aplicația poate folosi aceste rezultate ulterior în sistemul de recomandare, în procesul de afișare contextualizată sau în explorarea preferințelor utilizatorului.

Sistemul de Recomandare. Pentru a oferi o experienţă personalizată fiecărui utilizator, sistemul de recomandare hibrid implementat combină două abordări: filtrarea colaborativă și recomandarea bazată pe conținut.

Componenta colaborativă analizează comportamentul altor utilizatori similari pentru a sugera conținut nou. Similaritatea dintre doi utilizatori este calculată pe baza vectorilor de interacțiune (like, view, comment, bookmark), folosind **cosinusul unghiului dintre vectori**:

$$\text{Sim}(u, v) = \frac{\vec{r}_u \cdot \vec{r}_v}{\|\vec{r}_u\| \cdot \|\vec{r}_v\|}$$

unde $\vec{r}_u$ și $\vec{r}_v$ sunt vectorii de scoruri pentru utilizatorii u și v, iar $\cdot$ reprezintă produsul scalar. De asemenea actiunea de urmărire a unui alt utilizator afectează filtrarea colaborativă adaugând un factor în plus în calcularea similarităţii între doi utilizatori.

Componenta bazată pe conținut analizează caracteristicile fiecărei poezii (emoții, limbă, temă) și le compară cu preferințele explicite ale utilizatorului, stocate în câmpul `preferences`. Pentru a genera o listă de recomandări, sistemul calculează un scor final pentru fiecare poezie pe baza următoarei formule ponderate:

$$\text{Score}_{poem} = \alpha \cdot \text{Sim}_{content} + \beta \cdot \text{Sim}_{collab}$$

unde α și β sunt coeficienți de ajustare (ex: $\alpha = 0.6$, $\beta = 0.4$), $\text{Sim}_{content}$ este scorul de potrivire dintre emoțiile/language-ul poeziei și preferințele utilizatorului, iar Sim_{collab} este media similarităților cu alți utilizatori care au apreciat poezia respectivă.

Pentru ajustarea fină a scorului final, au fost implementați algoritmi suplimentari care amplifică preferințele (ex: `score emotion modifier.py`, `score language modifier.py`), aplicând ponderi dinamice în funcţie de interacțiunile anterioare

ale utilizatorului.

Formula utilizată pentru a actualiza scorul unei emoții după interacțiunea cu un poem este următoarea:

$$\begin{aligned} \text{delta} &= b \cdot \text{sign}(w) \cdot (1 - s) \cdot i \cdot |w| \\ \text{new\_score} &= s + \text{delta} \\ \text{score\_final} &= \min\left(\max\left(\text{new\_score}, \text{min\_score}\right), \text{max\_score}\right) \end{aligned}$$

unde:

- s – scorul actual al emoției pentru utilizator (valoare între 0.01 și 1)
- i – intensitatea emoției respective în poem
- w – scorul interacțiunii utilizatorului cu poemul (poate fi negativ sau pozitiv)
- b – factor de învățare (denumit *blend factor*), controlează viteza de actualizare, implicit $b = 0.1$
- $\text{sign}(w)$ – semnul valorii w: +1 dacă $w > 0$, 1 dacă $w < 0$
- min_score – scorul minim permis pentru orice emoție (implicit 0.01)
- max_score – scorul maxim permis pentru orice emoție (implicit 1)

Funcția `growth_formula` calculează o valoare crescută a scorului în funcție de o curbă logistică discretizată şi este folosită pentru cresterea scorului de preferinţă lingvistică:

$$\begin{aligned} \text{Calcul k:} \quad & k = \texttt{calculate\_k}(x, x_0, r) + 1 \\ \text{Rezultatul final:} \quad & x_{\text{new}} = 1 - (1 - x_0) \cdot e^{-r \cdot k} \end{aligned}$$

unde:

- x – scorul curent (valoare între 0 și 1)
- x_0 – scorul inițial de referință (default 0.01)
- r – rata de învățare sau creștere (default 0.01)
- k – număr de pași de învățare, calculat cu `calculate_k`

`calculate_k` această funcție determină numărul de pași (k) necesari pentru a obține o anumită valoare x, plecând de la valoarea inițială x_0:

$$k = -\frac{1}{r} \cdot \ln\left(\frac{1-x}{1-x_0}\right)$$

cu condițiile:

- $0 \leq x < 1$
- $0 \leq x_0 < 1$
- $r \neq 0$

unde:

- x – scorul curent
- x_0 – scorul inițial
- r – rata de învățare

Formula scade scorul actual doar dacă $k > 1$, conform relației:

$$\begin{aligned} k &= \texttt{calculate\_k}(x, x_0, r) \\ \text{if } k \leq 1: &\quad x_{\text{new}} = x_0 \\ \text{else}: &\quad x_{\text{new}} = 1 - (1 - x_0) \cdot e^{-r \cdot (k-1)} \end{aligned}$$

unde:

- x – scorul curent
- x_0 – scorul minim permis
- r – rata de descreștere
- k – numărul de pași înapoi, calculat cu `calculate_k`

Această abordare hibridă asigură un echilibru între personalizarea individuală și tendințele comunității, contribuind astfel la o experiență mai relevantă și interactivă în cadrul platformei PoemSociety.

Server FlaskAPI

Serverul back-end al aplicaţiei PoemSociety a fost dezvoltat utilizând framework-ul **Flask**, cunoscut pentru simplitatea şi eficienţa sa în construirea de aplicaţii web şi API-uri REST. Aplicaţia rulează pe un server Flask care primeşte cereri de la interfaţa Next.js, procesează aceste cereri şi trimite răspunsuri în format `JSON`, folosind pachetul `Flask-CORS` pentru a permite comunicarea cross-origin între clientul front-end şi serverul back-end.

Principalele funcţionalităţi ale aplicaţiei sunt implementate ca rute HTTP, fiecare corespunzând unei operaţii logice: autentificare, înregistrare, publicare poezii, like, comment, bookmark, vizualizare sau recomandare. Conectarea la baza de date MongoDB se realizează prin pachetul `PyMongo`, iar colecţiile sunt iniţializate o singură dată la pornirea serverului pentru eficienţă.

Fiecare endpoint tratează datele primite în format JSON, le validează, şi în funcţie de acţiune, actualizează colecţiile MongoDB corespunzătoare (`Users`, `Poems`, `Likes`, `Comments`, `Views` etc.).

Pe lângă funcţiile clasice de tip CRUD, serverul realizează şi operaţii complexe precum:

- integrarea analizei semantice AI pentru fiecare poem nou (`/poems POST`), prin apelarea modulului `analyze_poem()` care extrage emoțiile și limba dominantă;
- calcularea şi actualizarea scorurilor preferinţelor utilizatorilor în funcţie de interacţiunile efectuate (like, comment, bookmark, view);
- furnizarea de recomandări personalizate prin endpoint-ul `/recommandation/<id>`, unde este apelat algoritmul de tip hibrid (`hybrid_recommendations()`), care combină scorurile bazate pe conținut cu cele de tip colaborativ.

La nivel tehnic, fiecare interacţiune a utilizatorului este înregistrată cu un timestamp și un scor, iar pe baza acestora sunt actualizate preferințele în câmpurile preferences.emotions și preferences.languages. De exemplu, un like aplicat unei poezii generează o creștere a scorului limbii în care este scrisă poezia folosind funcţia `growth_formula()`, în timp ce un dislike sau o eliminare a bookmark-ului duce la o descreștere calculată prin `decrease_formula()` (vezi Figura 4.9).

De asemenea, endpoint-urile includ validări şi verificări de redundanţă pentru a evita introducerea multiplă a aceloraşi date (ex: verificare existenţă user, existenţă poem, dublare interacţiuni etc.). Pentru fiecare comentariu adăugat, aplicaţia verifică dacă a fost deja comentată poezia respectivă de către utilizator, ajustând în mod corespunzător scorul de vizualizare în funcţie de starea acţiunii anterioare.

```python
@app.route('/likes/toggle', methods=['POST'])
def toggle_like():
    data = request.json
    user_id = data.get("user_id")
    poem_id = data.get("poem_id")
    print(f"Toggle like for user {user_id} and poem {poem_id}")

    if not user_id or not poem_id:
        return jsonify({"error": "Missing user_id or poem_id"}), 400

    existing = likes_col.find_one({
        "user_id": user_id,
        "poem_id": poem_id
    })

    if existing:
        likes_col.delete_one({"_id": existing["_id"]})
        views_col.update_one(
            {"who": user_id, "post_id": poem_id},
            {"$inc": {"score": -0.35}},
            upsert=True
        )

        poem = poems_col.find_one({"_id": poem_id})
        language = poem["analysis"]["language"]
        user_score = users_col.find_one({"_id": user_id})["preferences"]["languages"][language]
        updated_score = decrease_formula(user_score, 0.01)

        users_col.update_one(
            {"_id": user_id},
            {"$set": {f"preferences.languages.{language}": updated_score}}
        )

        current_user_emotions = users_col.find_one({"_id": user_id})["preferences"]["emotions"]
        new_user_emotions = update_emotions(current_user_emotions, poem, -0.40)

        users_col.update_one({"_id": user_id}, {
            "$set": {"preferences.emotions": new_user_emotions}
        })


        return jsonify({"message": "Like removed", "liked": False}), 200
    else:
        likes_col.insert_one({
            "user_id": user_id,
            "poem_id": poem_id,
            "saved_at": datetime.utcnow().isoformat() + "Z"
        })
        views_col.update_one(
            {"who": user_id, "post_id": poem_id},
            {"$inc": {"score": 0.35}},
            upsert=True
        )

        poem = poems_col.find_one({"_id": poem_id})
        language = poem["analysis"]["language"]
        user_score = users_col.find_one({"_id": user_id})["preferences"]["languages"][language]
        updated_score = growth_formula(user_score, 0.01)

        users_col.update_one(
            {"_id": user_id},
            {"$set": {f"preferences.languages.{language}": updated_score}}
        )
        current_user_emotions = users_col.find_one({"_id": user_id})["preferences"]["emotions"]
        new_user_emotions = update_emotions(current_user_emotions, poem, 0.35)

        users_col.update_one({"_id": user_id}, {
            "$set": {"preferences.emotions": new_user_emotions}
        })


        return jsonify({"message": "Like added", "liked": True}), 201
```

Figura 4.9: Funcţia `toggle_like()`

În concluzie, serverul FlaskAPI acţionează ca intermediar principal între interfaţa grafică şi logica de analiză/recomandare, păstrând o arhitectură modulară şi extensibilă. Această arhitectură permite scalarea aplicaţiei şi integrarea de funcţionalităţi suplimentare, cum ar fi autentificarea bazată pe token sau contorizarea în timp real a interacţiunilor.

4.4 Diagrame de Utilizare şi de Secvenţă

Majoritatea funcţionalităţilor menţionate mai sus sunt destinate unui utilizator logat, dar nu este obligatoriu. Înafară de acţiunile de apreciere, comentariu, salvare, urmărire sau postare restul funcţionează. Totuşi există un mic discomfort pentru accesarea paginii web fără a fi logat, algoritmul de recomandare nefiind funcţional. În următoarea diagram de utilizare (4.10) se regăsesc iar toate funcţiile de care beneficiază un utilizator.

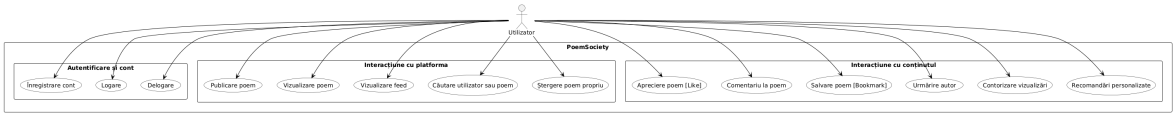

Figura 4.10: Diagrama de utilizare

Următoarele două diagrame de secvenţă evidenţiază fluxul informaţiei prin aplicaţie în urma acţiunilor de accesare feed (4.11) şi publicare poezie (4.12) de către utilizator:

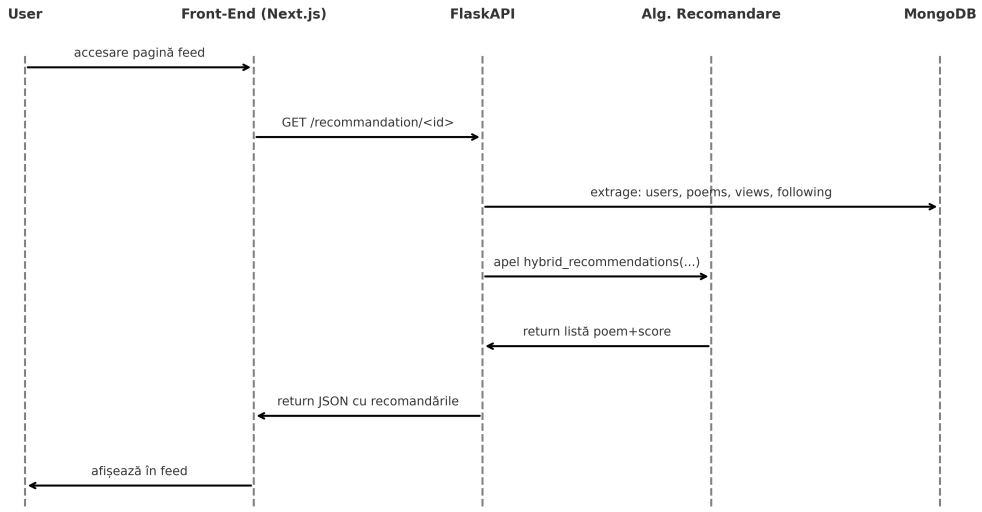

Figura 4.11: Diagrama de secvenţă - accesare feed

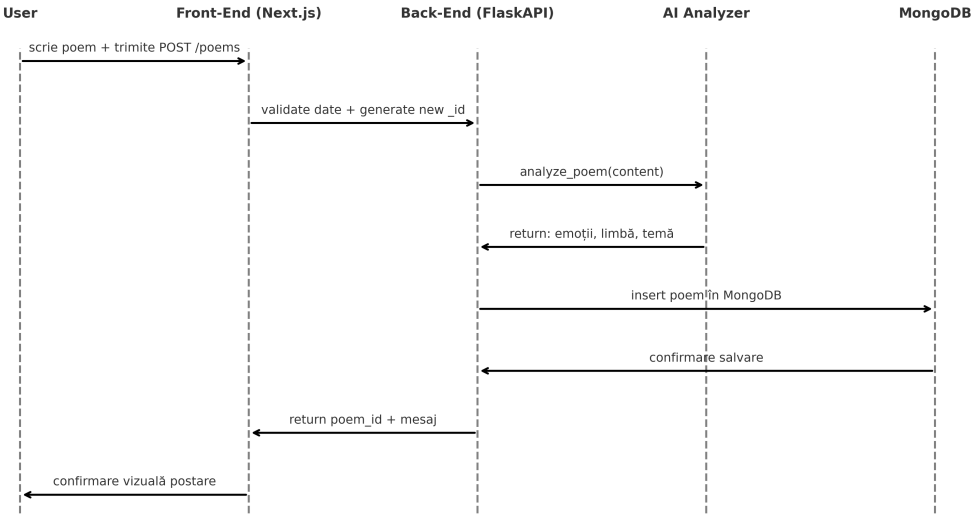

Figura 4.12: Diagrama de secvenţă - publicare postare

4.5 Testare Aplicatie - Manual de Utilizare

Pentru a rula corect aplicaţia PoemSociety trebuie luat în considerare că este o aplicaţie ce necesită şi mentenanţă a server-ului unde se află API, baza de date şi algoritmii AI şi de recomandare, dar şi a server-ului unde este front-end-ul publicat. Ambele părţi ale website-ului pot fi ţinute pe un server comun, oferind acces la date rapid, dar pus în practică real o platformă socială ar trebui distribuită pe mai multe servere, pentru a garanta acces instant a milioane de utilizatori.

Pentru testare set-up-ul iniţial este simplu. Sub ipoteza că persoana care doreşte să testeze aplicaţia are numai script-urile, atunci acesta trebuie să îşi instaleze următoarele: Node.js, Python, MongoDB, iar apoi prin comenzile npm şi pip, pachetele necesare pentru a rula Next.js şi FlaskAPI + MongoDB. Acesta poate crea două proiecte noi, goale, instalează package-urile, copiază fişierele pentru front-end in `./src/app/` şi cele pt API şi AI într-un folder separat. Ideal ar fi ca serverul API să fie pornit în momentul accesarii website-ului aşa ca persoana va da start serverului de API întâi, apoi celui pentru Next.js astfel:

>>$API-Path$: py poem-api.py
* Restarting with stat
Device set to use cpu
Device set to use cpu
Device set to use cpu
Device set to use cpu

Device set to use cpu
Device set to use cpu
 * Debugger is active!
 * Debugger PIN: pin
>>$API-Path$:cd $Next-Path$
>>$Next-Path$: npm run dev
- Local: http://localhost:3000
- Network: http://192.168.11.143:3000
 Starting...
 Ready in 1079ms

Apoi orice utilizator care are acces la `192.168.11.143:3000` poate vizualiza aplicaţia, iar persoana care a pornit serverele poate urmarii activitatea pe acestea.

Cum se accesează site-ul? În funcţie de unde este hostat (local/server) este necesar domeniul acestuia. După opţinerea domeniului utilizatorul poate naviga pe site după bunul plac, accesând toate subpaginile acestuia.

Capitolul 5

Concluzie

Lucrarea a investigat aplicarea algoritmilor de filtrare colaborativă și hibridă în contextul unei aplicații web sociale, demonstrând modul în care tehnologiile moderne pot fi utilizate pentru a genera experiențe digitale personalizate. Prin integrarea componentelor teoretice din domeniul sistemelor de recomandare cu un cadru de dezvoltare web contemporan, proiectul a oferit o soluție funcțională care combină analiza datelor, interfața prietenoasă și adaptabilitatea la comportamentele utilizatorilor.

Din perspectivă tehnică, abordarea s-a bazat pe o arhitectură scalabilă și flexibilă, formată din Next.js pentru front-end, Flask pentru serviciile API și MongoDB ca sistem de stocare NoSQL. Această structură a permis o separare clară a responsabilităților între interfață și logică de server, facilitând dezvoltarea modulară și integrarea facilă a algoritmilor de recomandare. Sistemul propus este ușor extensibil, fapt care îl face potrivit nu doar pentru aplicații culturale, ci și pentru platforme comerciale sau educaționale ce doresc personalizare în timp real.

Componenta algoritmică a fost dublată de o serie de mecanisme de ajustare adaptivă a preferințelor, în funcție de interacțiunile utilizatorului. Algoritmul hibrid a combinat scorurile generate de comportamentele similare ale altor utilizatori cu analiza conținutului poetic, rezultând într-un sistem de sugestii cu relevanță ridicată, chiar și în contexte cu date sparse sau utilizatori noi. S-a evidențiat astfel valoarea modelelor hibride în depășirea limitărilor sistemelor clasice și în generarea unei experiențe coerente și dinamice.

Studiul de caz – platforma PoemSociety – aduce o notă distinctă lucrării, prin încercarea de a reintroduce literatura într-un mediu digital care altfel privilegiază vizualul rapid și consumul instant. Poezia, formă subtilă de exprimare a emoției umane, este repusă în circulație printr-o interfață interactivă, personalizabilă și deschisă. Utilizatorii sunt invitați nu doar să citească, ci să descopere versuri care le reflectă stările, limbajul, temele preferate – toate selectate de un algoritm care „învață” gusturile și le rafinează cu fiecare interacțiune.

Astfel, lucrarea demonstrează că algoritmii de recomandare nu sunt doar instrumente comerciale, ci pot deveni punți între tehnologie și cultură, între cifră și metaforă. Într-un peisaj digital dominat de zgomot informațional, PoemSociety oferă o pauză lirică, ghidată de logică algoritmică și empatie calculată. Această fuziune dintre AI și artă nu doar ilustrează capacitatea tehnologiei de a adapta conținutul la individ, ci și redeschide o cale prin care poezia poate redeveni parte din rutina digitală a omului modern.

În perspectivă, sistemul poate fi extins prin integrarea unui motor de învățare profundă, a unei aplicații mobile și a unor mecanisme de filtrare a conținutului inadecvat. De asemenea, se pot adăuga instrumente de analiză statistică pentru autori și cititori, transformând platforma într-un spațiu de creație, interacțiune și reflecție augmentată.

Bibliografie

[1] Harold Abelson și Gerald Jay Sussman, *Structure and interpretation of computer programs*, The MIT Press, 1996.

[2] Robin Burke, "Hybrid Recommender Systems: Survey and Experiments", în *User Modeling and User-Adapted Interaction* 12.4 (2002), pp. 331–370, DOI: 10.1023/A:1021240730564.

[3] Mary Constantoglou și Nikolaos Trihas, "The influence of social media on the travel behavior of Greek Millennials (Gen Y)", în *Tour. Hosp. Manag* 8 (2020), pp. 10–18.

[4] Zhenhua Dong et al., "A Brief History of Recommender Systems", în *arXiv preprint arXiv:2209.01860* (2022), URL: https://export.arxiv.org/pdf/2209.01860v1.pdf.

[5] Miguel Grinberg, *Flask Web Development: Developing Web Applications with Python*, a 2-a ed., O'Reilly Media, 2018, ISBN: 9781491991732.

[6] Adrian Holovaty și Jacob Kaplan-Moss, *The Definitive Guide to Django: Web Development Done Right*, a 2-a ed., Apress, 2009, ISBN: 9781430219361.

[7] IfD Allensbach, *Number of people interested in books in Germany from 2019 to 2024, by level of interest (in millions)*, https://www.statista.com/statistics/382357/interest-in-books-germany/, 2024.

[8] Rod Johnson et al., *Professional Java Development with the Spring Framework*, Wrox Press, 2005, ISBN: 9780764574832.

[9] Steve Krug, *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability*, a 3-a ed., New Riders, 2014, ISBN: 9780321965516.

[10] Tao Li et al., "A Survey on Web Application Testing: A Decade of Evolution", în *arXiv preprint arXiv:2412.10476* (2024).

[11] William Lidwell, Kritina Holden și Jill Butler, *Universal Principles of Design*, a 2-a ed., Rockport Publishers, 2010, ISBN: 9781592535873.

[12] Donald A. Norman, *The Design of Everyday Things*, Revised and Expanded, Basic Books, 2013, ISBN: 9780465050659.

[13] Jake Pitre, "TikTok, creation, and the algorithm", în *The Velvet Light Trap* 91.1 (2023), pp. 71–74.

[14] Shaina Raza et al., "A comprehensive review of recommender systems: Transitioning from theory to practice", în *arXiv preprint arXiv:2407.13699* (2024).

[15] Elaine Rich, "User modeling via stereotypes", în *Cognitive science* 3.4 (1979), pp. 329–354.

[16] Sonika Sharma et al., "Hybrid Movie Recommendation System", în *International Journal of Engineering Research Technology (IJERT)* Vol. 14 (Apr. 2025).

[17] Andreea Fortuna Șchiopu et al., "Generation Z vs. Generation Y: different from or similar? A comparison of centennials and millennials regarding the use of social media for travel purposes", în *Cactus Tourism Journal* 5.1 (2023), pp. 20–35.

[18] Stefan Tilkov și Steve Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs", în *IEEE Internet Computing* 14.6 (2010), pp. 80–83.

[19] W3C, *Web Content Accessibility Guidelines (WCAG) 2.2*, `https://www.w3.org/TR/WCAG22/`, Accesat în 2025, 2023.

[20] Shuai Zhang et al., "Deep Learning based Recommender System: A Survey and New Perspectives", în *ACM Computing Surveys* 52.1 (2019), pp. 1–38, DOI: `10.1145/3285029`.