



UNIVERSITATEA DIN
BUCUREŞTI



FACULTATEA DE
MATEMATICA ŞI
INFORMATICA

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

TOWNTREKKER - APLICAȚIE DE SOCIALIZARE

Absolvent

Avian Silviu-Gabriel

Coordonator științific

Lect. Dr. Ruxandra Marinescu Ghemeaci

București, iunie 2023

Rezumat

Această lucrare prezintă procesul de dezvoltare al aplicației mobile **TownTrekker**, o rețea de socializare destinată explorării și descoperirii punctelor de interes din oraș, ce utilizează o varietate de noțiuni și elemente, cu scopul de a oferi utilizatorilor o experiență personalizată și captivantă, prin afișarea de conținut relevant în funcție de preferințele și interesele acestora. În analizarea datelor utilizatorilor sunt aplicati algoritmi de filtrare, pentru a pune la dispoziție recomandări personalizate în funcție de interacțiunile acestora în cadrul platformei. Totodată, aplicația utilizează un model personalizat de Inteligență Artificială, ce analizează recenziile utilizatorilor, furnizând astfel recomandări fiabile și relevante pentru ceilalți utilizatori ai rețelei sociale.

De asemenea, în implementarea acestei rețele, este utilizat algoritmul BFS pentru a extinde rețeaua de conexiuni între utilizatori și a facilita descoperirea de persoane noi sau grupuri cu interese similare. În plus, pentru a adăuga un element de surpriză și diversitate în experiența utilizatorilor, platforma utilizează selecția aleatoare a elementelor în anumite contexte, precum afișarea de recomandări ale unor locații sau sugestii de postări. Astfel, prin folosirea acestor funcționalități, se încurajează descoperirea de conținut nou și interacțiunea cu persoane și grupuri diverse în cadrul aplicației de socializare.

Abstract

This paper presents the development process of the mobile application **TownTrekker**, a social network designed to explore and discover points of interest in the city, which uses a variety of concepts and elements with the aim of providing users with a personalized and engaging experience, by displaying relevant content based on their preferences and interests. By analyzing user data, filtering algorithms are applied to provide personalized recommendations based on their interactions within the platform. At the same time, the application uses a personalized Artificial Intelligence model, which analyzes user reviews, thus providing reliable and relevant recommendations for the other users of the social network.

Also, in the implementation of this newtork, the BFS algorithm is used to expand the network of connections between users and facilitate the discovery of new people or groups with similar interests. Additionally, to add an element of surprise and diversity to the user experience, the platform uses random selection of elements in certain contexts, such as displaying location recommendations or post suggestions. Thus, by using these functionalities, the discovery of new content and interaction with different people and groups within the social media application is encouraged.

Cuprins

1 Introducere	5
1.1 Răspândirea platformelor de socializare	5
1.2 Obiective	5
1.3 Motivația personală	6
1.4 Structura lucrării	6
2 Preliminarii	8
2.1 Analizarea pieței de social media	8
2.2 Identificarea categoriilor de utilizatori	8
2.3 Metodologia de dezvoltare	13
2.3.1 Definirea cerințelor	13
2.3.2 Alegerea uneltelor pentru analizarea recenziilor	13
2.3.3 Alegerea limbajului de implementare pentru aplicație	14
2.3.4 Unelte folosite	15
2.3.5 Limitări	16
2.3.6 Controlul versiunilor și organizarea sarcinilor de lucru	17
2.4 Compararea cu aplicații similare	18
3 Prezentarea aplicației	19
3.1 Alegerea titlului	19
3.2 Generarea logo-ului	20
3.3 Arhitectura proiectului	22
3.4 Structura aplicației	23
3.4.1 Activitatea de Autentificare	23
3.4.2 Activitatea Principală	29
4 Fundamentele algoritmice	41
4.1 Analizarea recenziilor	41
4.1.1 Alegerea și împărțirea setului de date	41
4.1.2 Selectarea și testarea algoritmilor	42
4.1.3 Balansarea setului de date	47

4.1.4	Alte preprocesări ale datelor	48
4.1.5	Optimizarea hiperparametrilor	50
4.1.6	Exportarea modelului final	52
4.1.7	Testarea modelului în aplicație	54
4.2	Recomandări personalizate	56
4.2.1	Recomandări pe baza preferințelor de postări	57
4.2.2	Recomandări pe baza intereselor persoanelor urmărite	59
4.2.3	Recomandări aleatorii	61
4.3	Postări personalizate	61
4.3.1	Postări pe baza categoriilor de interes	62
4.3.2	Postări ale utilizatorilor apropiati	63
4.3.3	Postări aleatorii	65
5	Concluzii	66
	Bibliografie	68

Capitolul 1

Introducere

1.1 Răspândirea platformelor de socializare

În ultimii ani, am asistat la o creștere accelerată a rețelelor de socializare, care au devenit un element de bază al vieții noastre cotidiene, revoluționând felul în care interacționăm și ne conectăm cu cei din jurul nostru. Într-un context în care tehnologia și internetul au devenit mult mai accesibile, rețelele sociale precum Facebook, Instagram sau Twitter au devenit mediul preferat pentru exprimare personală, împărtășirea experiențelor și interacționarea cu prieteni sau membrii ai familiei.

Aceste aplicații oferă o serie de funcționalități, cum ar fi postarea de fotografii sau videoclipuri, trimiterea de mesaje, organizarea de evenimente sau promovarea unor afaceri. În acest mod, ele au facilitat conectarea oamenilor din întreaga lume, oferindu-le posibilitatea de a comunica în timp real. De asemenea, acestea au avut un impact semnificativ asupra culturii și comportamentului uman, generând tendințe și modalități noi de exprimare.

Pe măsură ce aceste platforme de socializare au câștigat popularitate și utilizatori, au apărut anumite întrebări cu privire la algoritmii și metodele folosite în dezvoltarea aplicațiilor de socializare. Deoarece implementările utilizate sunt adesea proprietare și nu sunt dezvăluite publicului, există astfel o lipsă de transparentă în ceea ce privește maniera în care acestea funcționează și cum influențează conținutul și experiența utilizatorilor.

1.2 Obiective

Obiectivul acestei lucrări constă în dezvoltarea unei platforme de socializare, cu denumirea **TownTrekker**, care să aibă funcționalități similare cu cele descrise anterior, dar cu îmbunătățiri semnificative în ceea ce privește funcționalitatea și experiența utilizatorului, prin aducerea unor caracteristici noi care să o diferențieze și să o facă mai atractivă.

În cadrul aplicației **TownTrekker**, utilizatorii pot să descopere punctele de interes aflate în oraș și să împărtășească experiențele lor, prin intermediul recenziilor sub formă de postări, pe care aceștia le pot distribui pe platformă. De asemenea, utilizatorii au acces la conținut (postări sau recomandări de locații) în funcție de preferințele acestora, prin folosirea elementelor algoritmice, existând o claritate în legătură cu scopul și modul în care acestea sunt utilizate în aplicație.

Contribuția personală constă în utilizarea eficientă a cunoștințelor din mai multe sfere de activitate: dezvoltarea de aplicații pentru platforma Android, implementarea și optimizarea modelelor de Inteligență Artificială pentru analizarea textului și utilizarea practică a algoritmilor pe grafuri sau pentru selecția aleatorie a elementelor.

Codul sursă pentru această aplicație este disponibil pe platforma **GitHub**, la adresa <https://github.com/Silviu1409/TownTrekker>.

1.3 Motivația personală

Am ales această temă deoarece am fost captivat de succesul pe care l-au avut aplicațiile de acest tip în ultimii ani, reușind să atragă o audiență extinsă. În același timp, piața telefoanelor inteligente s-a dezvoltat, odată cu avansul tehnologic; acestea au adus conectivitatea permanentă și mobilitatea în viața de zi cu zi, prin intermediul serviciilor pe care le oferă și a gamei largi de aplicații ce pot fi accesate, cu diferite scopuri, de la cele utilizate pentru navigarea pe internet până la cele de socializare.

Cercetarea pe această temă m-a făcut să descopăr și să dobândesc cunoștințe noi din acest domeniu, cum ar fi elaborarea unei interfețe grafice pentru o aplicație de acest tip, procesarea datelor și determinarea utilității algoritmilor utilizați. O provocare a fost constituită de alegerea tehnologiilor și a uneltelor ce corespund cerințelor, dintr-o varietate ce pot fi folosite, alături de asimilarea noțiunilor prezentate, întrucât a fost nevoie de o documentație amplă pentru a înțelege, analiza și pune în practică aceste aspecte teoretice.

1.4 Structura lucrării

Această lucrare este structurată în cinci capitole:

1. **Introducere:** este reprezentat de capitolul actual, ce a inclus o prezentare generală a tematicii aplicației, a obiectivelor lucrării și a motivației personale.
2. **Preliminarii:** oferă o imagine de ansamblu asupra pieței rețelelor de socializare, exemplifică funcționalitățile pe care le are o astfel de platformă și ilustrează resursele utilizate, justificând rolul acestora prin expunerea avantajelor, dar și a limitărilor pe care le au.

3. **Prezentarea aplicației:** această secțiune este mai amplă, întrucât conține o prezentare detaliată a structurii aplicației. Aspectele abordate implică elemente de interfață grafică, funcționalități prezente și optimizări aduse în procesul de implementare.
4. **Fundamentele algoritmice:** detaliază elementele algoritmice integrate în contextul aplicației, explicând importanța acestora și modul în care au fost puse în practică.
5. **Concluzii:** sintetizează informațiile cuprinse în acest document și aduce în discuție îmbunătățiri ce pot fi realizate ca direcții pentru dezvoltarea viitoare a aplicației.

Capitolul 2

Preliminarii

2.1 Analizarea pieței de social media

După cum am discutat în capitolul anterior, la secțiunea 1.1, rețelele sociale au devenit o parte integrantă a vieții noastre de zi cu zi și au dobândit o importanță semnificativă în societate, prin punerea la dispoziție a unui mediu în cadrul căruia utilizatorii pot împărtăși conținut și interacționa într-un mod amplu și accesibil.

Pentru a înțelege mai bine acest subiect, este necesară o analiză a acestei piețe, cu scopul de a dezvolta o strategie eficientă pentru implementarea unei aplicații de acest tip. Principalele caracteristici includ interacțiunea în timp real între utilizatori, posibilitatea de a crea și partaja conținut generat de către aceștia și accesibilitatea la o gamă largă de informații și fișiere multimedia. Utilizatorii pot publica și trimite texte, imagini, videoclipuri, sau alte tipuri de conținut, având opțiunea de a putea comunica prin comentarii, mesaje private sau prin reacții precum *like* sau *share*.

În sfera *social media* există o diversitate mare de rețele sociale, unde fiecare are propriile funcționalități și particularități, adresându-se diferitelor nevoi și preferințe ale utilizatorilor. Cele mai cunoscute exemple includ platforme precum **Facebook**, o rețea de socializare generală, în cadrul căreia utilizatorii au posibilitatea de a crea și urmări pagini și grupuri cu tematici diferite, **Instagram**, orientată către postarea de imagini și videoclipuri, **Twitter**, concentrat pe mesaje scurte și rapide sau **LinkedIn**, cu accent către mediul profesional.

2.2 Identificarea categoriilor de utilizatori

Utilizatorii reprezintă componenta centrală a rețelelor de socializare. Aceștia sunt indivizi sau grupurile care accesează aceste platforme, cu scopurile menționate anterior, iar publicul este unul extrem de diversificat și cuprinde milioane de oameni din întreaga lume. În acest sens, am consultat platforma **DataReportal**, ce oferă rapoarte și studii

anuale despre starea rețelelor sociale și a utilizatorilor acestora, la nivel global.

În cadrul raportului *Digital 2023 Global Overview Report* [14], publicat în luna Ianuarie a anului 2023 și realizat în parteneriat cu compania **Meltwater** și agenția **We Are Social** se regăsesc mai multe statistică cu privire la domeniul digital și, implicit *Social Media*, iar în continuarea acestei secțiuni voi include câteva date relevante, ilustrate în figurile 2.1 - 2.6.

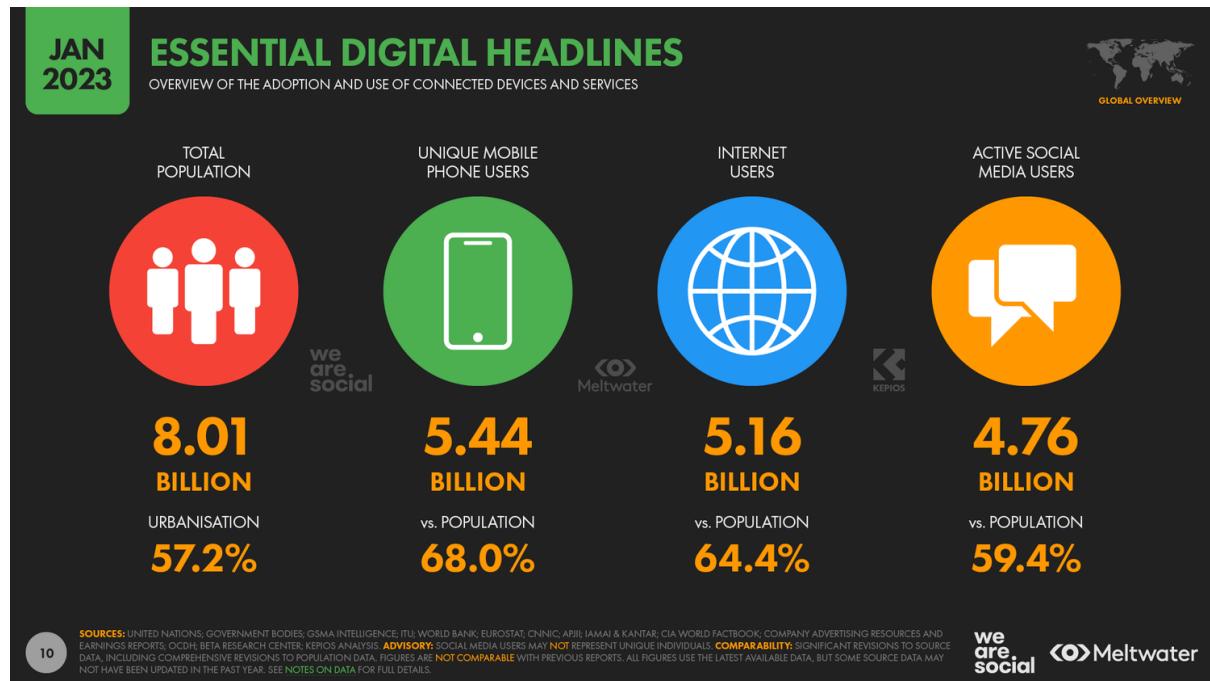


Figura 2.1: Utilizarea serviciilor și dispozitivelor mobile

Din figura 2.1, se observă următoarele aspecte:

- populația globală a ajuns la aproximativ **8.01 miliarde** de oameni
- numărul de utilizatori unici ai telefoanelor mobile a ajuns la aproximativ **5.44 miliarde**, ceea ce reprezintă **68%** din totalul populației globale, iar aceste cifre confirmă creșterea accelerată a pieței telefoanelor inteligente
- **5.16 miliarde** de persoane s-au conectat la internet; acest lucru reflectă avansul tehnologic, ce ne facilitează accesul la aceste resurse
- rețelele de tip *social media* adună aproximativ **4.76 miliarde** de utilizatori activi, echivalentul a **59.4%** din ansamblul populației mondiale, element ce confirmă utilizarea vastă a aplicațiilor din această categorie

Conform figurii 2.2, în rândul utilizatorilor cu vârste între **16 și 64 de ani**, rețelele de socializare se află pe locul 2 (**94.6%**), în topul website-urilor/aplicațiilor folosite, atestând interesul publicului pentru acest topic. Aici se regăsesc și aplicații de mesagerie (94.8%), motoare de căutare/site-uri web (81.8%) sau platforme de cumpărături (76%).

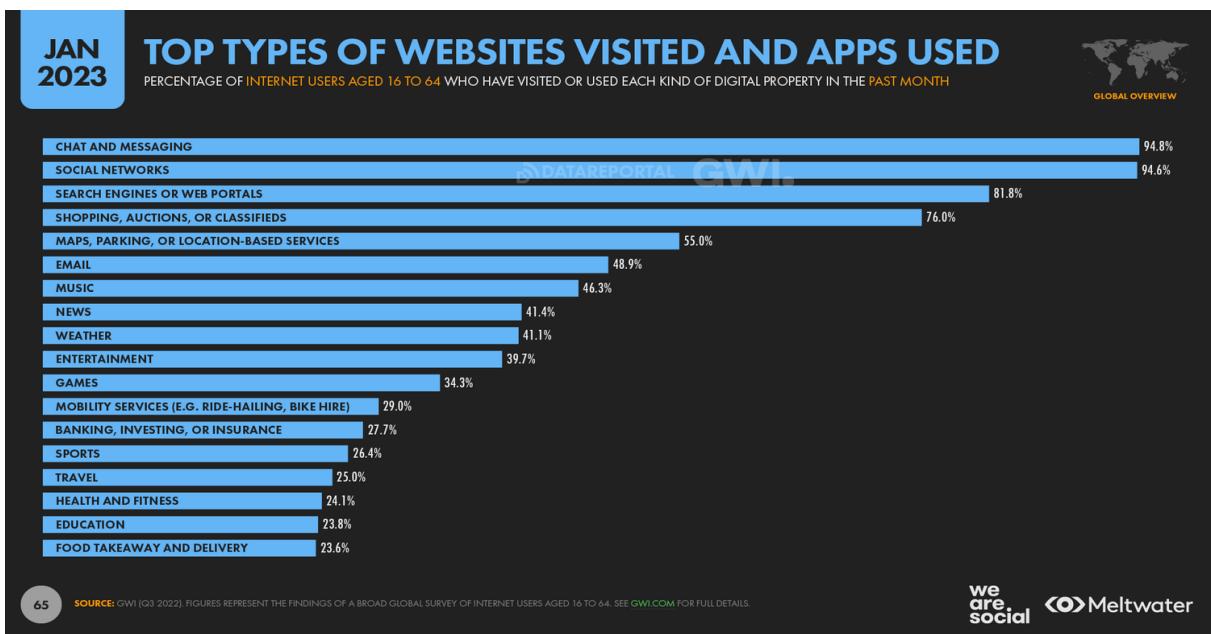


Figura 2.2: Tipuri de website-uri/aplicații accesate

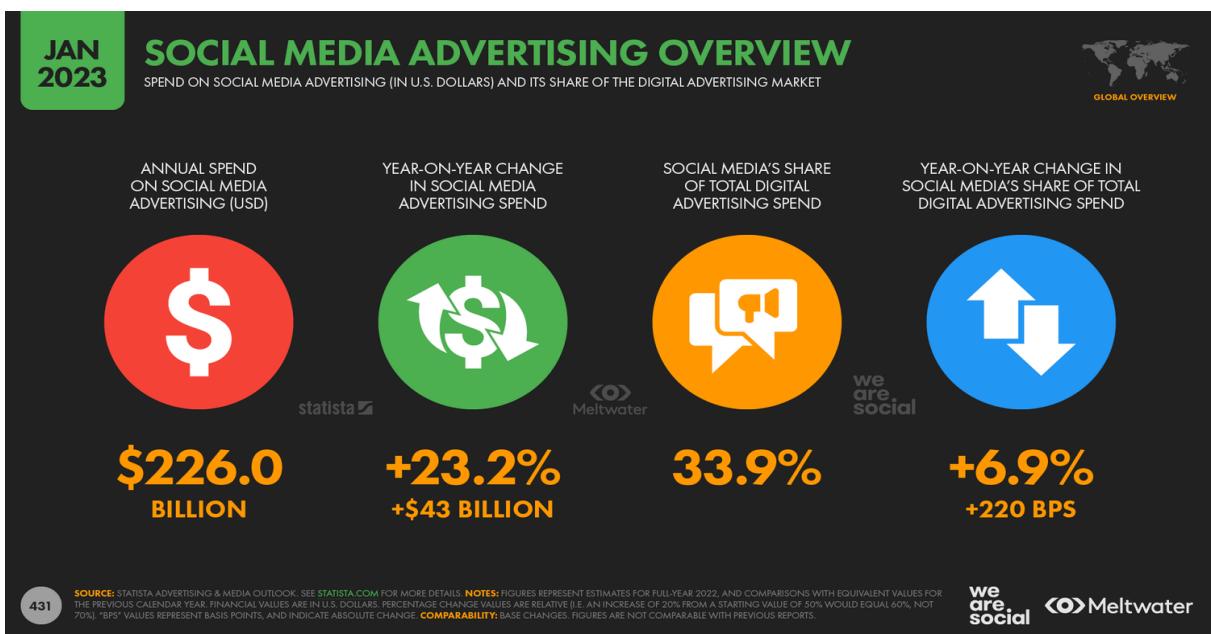


Figura 2.3: Publicitatea pe social media

Potrivit figurii 2.3, pe parcursul anului 2022, pentru publicitatea pe rețelele de socializare s-au cheltuit, estimativ, **226 miliarde de dolari**, reprezentând aproximativ **33.9%** din suma totală alocată pentru publicitatea integral digitală, cu o creștere semnificativă, de **23.2%** față de anul 2021. Datele de față evidențiază că aceste platforme reprezintă un mediu pentru a promova branduri, produse sau servicii, prin intermediul campaniilor de promovare, colaborarea cu *influencerii* sau prin organizarea de concursuri/evenimente online. Scopul promovării este de a atrage publicul țintă, prin conținut relevant și atractiv, aspect ce poate ajuta la construirea brandurilor și la creșterea vizibilității acestora.

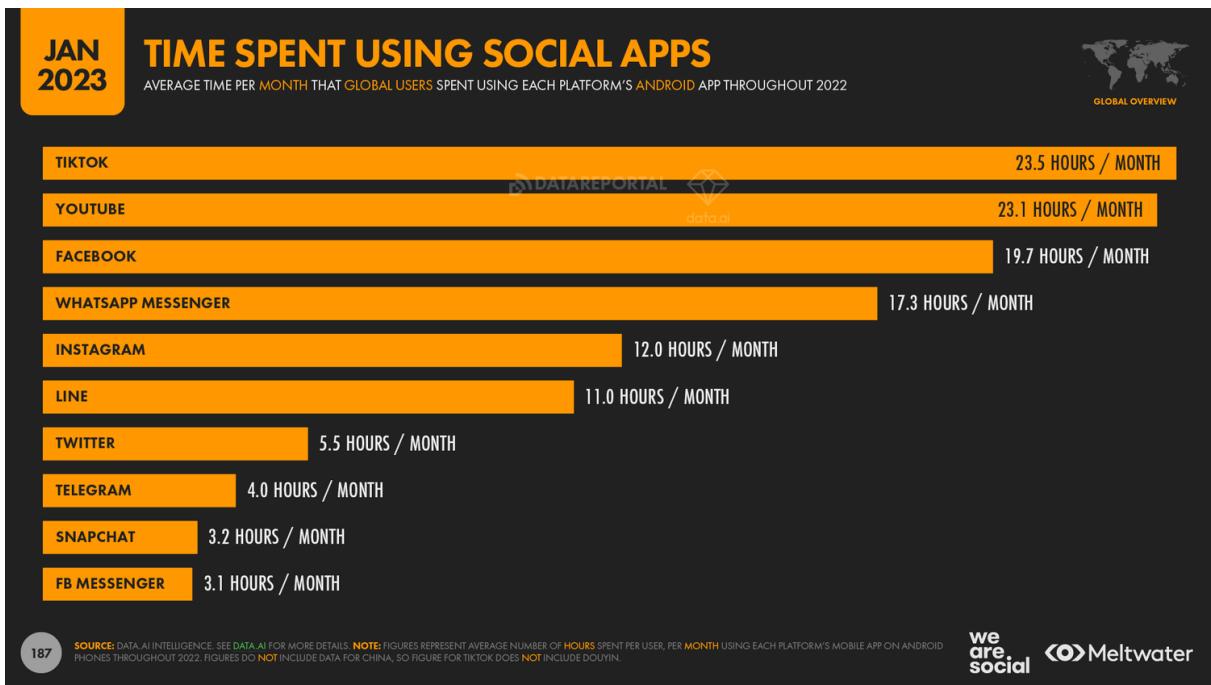


Figura 2.4: Timpul petrecut pe platformele de social media

Acest grafic, conținut în figura 2.4, ilustrează timpul mediu lunar petrecut de către utilizatori pe aplicațiile **Android** ale unor dintre cele mai populare rețele de socializare, pe parcursul anului 2022. În top se află **TikTok** și **YouTube**, fiecare cu peste 23 de ore pe lună, iar diferența considerabilă față de alte platforme, aflate pe ultimele locuri este dată de mai mulți factori: calitatea și relevanța conținutului disponibil, interacțiunile sociale, caracteristicile și funcționalitățile oferite sau factorii demografici și contextuali.

În cele două tabele, prezente mai jos, în figura 2.5, sunt exprimate, procentual, preferințele pentru mai multe platforme de socializare, pe mai multe categorii de vârstă, în funcție de gen (feminin - stânga și masculin - dreapta). Aceste preferințe nu diferă semnificativ în baza genului, dar diferă conform grupelor de vârstă, astfel:

- În rândul utilizatorilor tineri (16-24 de ani), *Instagram* este lider detașat, încrucișat acest public este cunoscut pentru accentul său pe conținutul vizual și interacțiunea rapidă.
- În ceea ce privește persoanele adulte de vârstă mijlocie (45-54 de ani și, respectiv, 55-64 de ani), cele două platforme care concurează sunt *WhatsApp* și *Facebook*; acestea sunt platforme de socializare tradiționale, ce oferă funcționalități precum mesagerie instantanee, grupuri, evenimente și conexiunea cu prietenii și familia.
- În cadrul tinerilor adulți (25-34 de ani și 35-44 de ani) apare un echilibru între *WhatsApp*, *Instagram*, *Facebook* și *WeChat*; această balanță este legată de poziția intermediară între cele două categorii menționate anterior, unde preferințele variază în raport cu interesele, cerințele profesionale sau alegerile personale.

JAN
2023

FAVOURITE SOCIAL MEDIA PLATFORMS

PERCENTAGE OF ACTIVE SOCIAL MEDIA USERS WHO SAY THAT EACH OPTION IS THEIR "FAVOURITE" SOCIAL MEDIA PLATFORM



FAVOURITE SOCIAL MEDIA PLATFORMS AMONGST FEMALE INTERNET USERS

SOCIAL PLATFORM	AGE 16-24	AGE 25-34	AGE 35-44	AGE 45-54	AGE 55-64
WHATSAPP	12.3%	13.3%	15.4%	16.4%	20.3%
INSTAGRAM	23.1%	17.6%	13.2%	10.6%	7.6%
FACEBOOK	6.9%	13.5%	14.7%	16.9%	18.9%
WECHAT	8.0%	13.4%	15.0%	13.1%	11.7%
TIKTOK	12.0%	7.5%	5.5%	4.6%	3.2%
DOUYIN	4.7%	6.8%	7.6%	6.3%	3.6%
TWITTER	5.0%	2.6%	2.1%	2.1%	1.9%
FB MESSENGER	2.1%	2.5%	2.7%	2.9%	3.3%
TELEGRAM	2.0%	1.6%	1.6%	1.9%	1.8%
LINE	1.0%	1.4%	2.2%	3.4%	4.6%

FAVOURITE SOCIAL MEDIA PLATFORMS AMONGST MALE INTERNET USERS

SOCIAL PLATFORM	AGE 16-24	AGE 25-34	AGE 35-44	AGE 45-54	AGE 55-64
WHATSAPP	15.5%	15.4%	17.1%	18.5%	19.5%
INSTAGRAM	21.3%	14.6%	9.4%	7.0%	4.9%
FACEBOOK	10.5%	15.7%	17.1%	16.8%	18.4%
WECHAT	8.4%	12.1%	13.8%	14.1%	15.0%
TIKTOK	7.7%	5.1%	4.4%	4.1%	2.2%
DOUYIN	4.1%	6.0%	6.7%	5.3%	4.7%
TWITTER	4.2%	3.9%	3.8%	3.8%	3.5%
FB MESSENGER	2.1%	2.8%	2.8%	2.6%	2.7%
TELEGRAM	3.0%	2.8%	2.3%	2.4%	2.3%
LINE	0.8%	1.3%	1.9%	2.9%	3.7%

SOURCE: GWI (Q3 2022) SEE GWI.COM FOR FULL DETAILS. NOTES: ONLY INCLUDES INTERNET USERS WHO HAVE USED AT LEAST ONE SOCIAL MEDIA PLATFORM IN THE PAST MONTH. SURVEY RESPONDENTS COULD CHOOSE FROM OTHER OPTIONS NOT SHOWN ON THIS CHART. SO VALUES MAY NOT SUM TO 100%. SOURCE IS NOT AVAILABLE AS AN ANSWER FOR THIS QUESTION IN GWI'S SURVEY. WE REPORT GWI'S VALUES FOR TIKTOK IN CHINA SEPARATELY AS DOUYIN, AS PER BYTDANCE'S CORPORATE REPORTING. COMPARABILITY: VALUES NOW REPRESENT SHARE OF ACTIVE SOCIAL MEDIA USERS ONLY, RATHER THAN SHARE OF ALL INTERNET USERS. VERSIONS OF THIS CHART THAT FEATURED IN OUR PREVIOUS REPORTS DID NOT INCLUDE DATA FOR CHINA, SO VALUES ARE NOT COMPARABLE.

we
are.
social

Meltwater

Figura 2.5: Platformele preferate, în funcție de gen, pe categorii de vârstă

Conform figurii 2.6, în luna noiembrie a anului 2022, **Android** a fost sistemul de operare utilizat majoritar pentru accesarea paginilor web, urmat la o diferență semnificativă de **iOS**. Totodată, creșterea popularității pentru **Android** și scăderea în cazul **iOS** accentuează preferința utilizatorilor pentru prima platformă.

JAN
2023

SHARE OF MOBILE WEB TRAFFIC BY MOBILE OS

PERCENTAGE OF WEB PAGE REQUESTS ORIGINATING FROM MOBILE HANDSETS RUNNING EACH MOBILE OPERATING SYSTEM IN NOVEMBER 2022



SHARE OF MOBILE WEB TRAFFIC ORIGINATING FROM ANDROID DEVICES



71.96%

YEAR-ON-YEAR CHANGE
+1.7% (+122 BPS)

SHARE OF MOBILE WEB TRAFFIC ORIGINATING FROM APPLE IOS DEVICES



27.48%

YEAR-ON-YEAR CHANGE
-3.7% (-106 BPS)

SHARE OF MOBILE WEB TRAFFIC ORIGINATING FROM SAMSUNG OS DEVICES



0.34%

YEAR-ON-YEAR CHANGE
-10.5% (-4 BPS)

SHARE OF MOBILE WEB TRAFFIC ORIGINATING FROM KAI OS DEVICES



0.07%

YEAR-ON-YEAR CHANGE
-50.0% (-7 BPS)

SHARE OF MOBILE WEB TRAFFIC ORIGINATING FROM OTHER OS DEVICES



0.15%

YEAR-ON-YEAR CHANGE
-25.0% (-5 BPS)

337

SOURCE: STATCOUNTER. NOTES: FIGURES REPRESENT THE NUMBER OF WEB PAGES SERVED TO BROWSERS ON MOBILE PHONES RUNNING EACH OPERATING SYSTEM COMPARED WITH THE TOTAL NUMBER OF WEB PAGES SERVED TO MOBILE BROWSERS IN NOVEMBER 2022. FIGURES FOR SAMSUNG OS REFER ONLY TO THOSE DEVICES RUNNING OPERATING SYSTEMS DEVELOPED BY SAMSUNG (E.G. BADA AND TIZEN), AND DO NOT INCLUDE SAMSUNG DEVICES RUNNING ANDROID. PERCENTAGE CHANGE VALUES REPRESENT RELATIVE CHANGE (I.E. AN INCREASE OF 20% FROM A STARTING VALUE OF 50% WOULD EQUAL 60%, NOT 70%). "BPS" VALUES REPRESENT BASIS POINTS, AND INDICATE THE ABSOLUTE CHANGE. FIGURES MAY NOT SUM TO 100% DUE TO ROUNDING.

we
are.
social

Meltwater

Figura 2.6: Cota de trafic mobil, în funcție de sistemul de operare

2.3 Metodologia de dezvoltare

Această secțiune are ca scop prezentarea abordării generale și a procesului de dezvoltare utilizat pentru realizarea aplicației, ce acoperă aspecte cheie precum definirea cerințelor, uneltele folosite sau organizarea sarcinilor de lucru. În cadrul acestor subsecțiuni voi specifica nevoile și funcționalitățile de bază ale aplicației într-un mod concis, pentru a ghida dezvoltarea și implementarea cu succes a produsului final.

2.3.1 Definirea cerințelor

Aplicația **TownTrekker** este concepută pentru a se adresa tuturor categoriilor de vârstă, având ca scop să funcționeze drept o platformă de socializare generală, în cadrul căreia utilizatorii să aibă posibilitatea de a-și crea un cont și să se bucure de numeroasele funcționalități captivante. Unul dintre aspectele cheie ale aplicației este capacitatea de a descoperi locații noi din împrejurimi. Utilizatorii se pot inspira din postările altor persoane, care distribuie experiențele și descoperirile lor. De asemenea, platforma oferă recomandări personalizate pentru locații, în funcție de preferințele utilizatorilor, ajutându-i să descopere locuri interesante și autentice. Pentru o experiență și mai captivantă, aceștia pot explora o hartă interactivă personalizată, care le permite să vizualizeze și să navigheze prin diferite locații, să găsească informații relevante și să planifice călătorii. Toate aceste funcționalități vin în ajutorul utilizatorilor, pentru a le îmbunătăți experiența interacțiunii cu rețea de socializare.

2.3.2 Alegerea uneltelor pentru analizarea recenziilor

În cadrul aplicației, voi folosi un model de Inteligență Artificială, inclus în secțiunea 4.1, ce are ca scop analizarea recenziilor utilizatorilor și împărtirea acestora în mai multe categorii. Am optat pentru alegerea limbajului de programare **Python3** [21] și a platformei **Kaggle** [13], deoarece această combinație oferă numeroase avantaje și resurse esențiale pentru dezvoltarea și experimentarea cu algoritmi de Inteligență Artificială.

Python3 pune la dispoziție o sintaxă simplă și elegantă, ce facilitează scrierea și citirea codului. Este un limbaj popular în acest domeniu datorită ecosistemului vast și bogat în librării de specialitate, precum **scikit-learn**, **pandas** sau **spaCy**, biblioteci pe care urmează să le folosesc în procesul de implementare.

În ceea ce privește **Kaggle**, aceasta este o platformă valoioasă pentru pasionații din sfera *data science*, deoarece furnizează o colecție bogată de seturi de date și diverse competiții în acest domeniu, oferind oportunități de învățare și colaborare în comunitatea globală. De asemenea, ea permite utilizatorilor să testeze și să-și îmbunătățească abilitățile și implementările, cu ajutorul *kernel-urilor* interactive pe care poate fi rulat cod scris în **Python3**.

2.3.3 Alegerea limbajului de implementare pentru aplicație

Așa cum am menționat la finalul secțiunii anterioare, sistemul de operare cel mai utilizat pe platforma mobilă pentru accesarea site-urilor online a fost **Android**, fiind motivul principal pentru care l-am ales în vederea dezvoltării aplicației **TownTrekker**. În acest sens, am efectuat o comparație detaliată între două dintre cele mai folosite limbaje de programare pentru această platformă.

Java [18]

Prima opțiune studiată a fost **Java**. Este un limbaj de programare orientat pe obiecte, care oferă o sintaxă clară, ușor de înțeles și, pentru o lungă perioadă, a fost unul dintre principalele limbaje de programare utilizate în dezvoltarea de aplicații **Android**. Factorii pentru care nu am ales această platformă sunt:

- Sintaxa lungă, ce poate necesita mai mult cod pentru a realiza aceeași funcționalități, comparativ cu alte limbaje de programare.
- Posibilitatea apariției erorilor legate de gestionarea memoriei; deși Java are un sistem de gestionare a memoriei bazat pe colectarea de gunoi (garbage collection), acesta poate duce la probleme de performanță și consum de memorie (scurgeri de memorie sau utilizare ineficientă) în anumite cazuri.
- Scăderea în relevanță, apărută o dată cu introducerea limbajului **Kotlin**.

Kotlin [12]

Este un limbaj de programare modern și concis, care rulează pe platforma Java Virtual Machine (JVM) și, din anul 2017, este acceptat oficial de către **Google** ca limbaj pentru dezvoltarea aplicațiilor Android. Am optat pentru această variantă deoarece surclasază **Java** în ceea ce privește aspectele prezentate anterior, astfel:

- Sintaxa este mai scurtă și mai sugestivă, reducând astfel cantitatea de cod necesară.
- Kotlin beneficiază de sistemul de colectare a gunoiului al Java, dar, cu toate acestea, Kotlin aduce câteva îmbunătățiri și facilități care pot ajuta la evitarea unor erori comune legate de gestionarea memoriei, cum ar fi introducerea sistemului de tipuri *nullable* și *non-nullable*, interferența tipului sau definirea de funcții de extensie.
- A câștigat rapid popularitate în comunitatea dezvoltatorilor Android și este considerat de mulți un înlocuitor potrivit pentru Java în acest sens.
- În plus, Kotlin aduce avantaje precum siguranța tipurilor și interoperabilitatea cu Java, permitând dezvoltatorilor să folosească ambele limbaje în același proiect.

2.3.4 Unelte folosite

Am utilizat **Android Studio** [6] ca mediu de dezvoltare integrat (IDE), deoarece este o unealtă open-source, specializată în dezvoltarea aplicațiilor Android, folosind limbajul de programare Kotlin. Este dezvoltat de către **Google** și **JetBrains**, oferind o serie de funcționalități și instrumente ce facilitează procesul de modelare și optimizare:

- Un editor de cod bogat în funcționalități, ce pune la dispoziție completare automată, evidențierea sintaxei, refactorizări și altele; aceste facilități ajută dezvoltatorii să scrie cod mai rapid și într-un mod eficient.
- Un emulator integrat, ce permite dezvoltatorilor să testeze și să ruleze aplicațiile pe mai multe versiuni și configurații de dispozitive Android.
- O serie de instrumente de depanare, cum ar fi debugger-ul, care permite urmărirea execuției aplicației pas cu pas, identificarea și rezolvarea erorilor sau a problemelor de performanță.
- Facilitatea integrării aplicațiilor cu servicii furnizate de către Google, precum Firebase, Google Maps, Google Play Services și altele. Aceasta oferă instrumente și şabloane ce servesc pentru adăugarea de funcționalități, din care fac parte stocarea datelor în cloud, autentificarea sau accesul la o hartă.

Firebase [7]

Este o platformă de dezvoltare a aplicațiilor, extinsă și întreținută de către Google. Aceasta este proiectată pentru a ajuta creatorii să construiască aplicații de înaltă calitate, scalabile și ușor de gestionat. Firebase pune la dispoziție o suită de servicii și instrumente care acoperă o gamă largă de nevoi, iar dintre aceste unelte urmează să utilizez:

1. **Authentication:** oferă servicii puternice de autentificare, permitând dezvoltatorilor să adauge funcționalitatea de autentificare și autorizare în aplicațiile lor, folosind metode precum autentificarea prin e-mail, conturi sociale (de exemplu Google, Twitter, GitHub) sau număr de telefon.
2. **Cloud Firestore:** este o bază de date flexibilă și accesibilă, ce permite stocarea și gestionarea datelor aplicației într-un mod structurat și eficient; de asemenea, furnizează suport pentru consultări complexe, sincronizare în timp real și se integrează perfect cu celealte servicii Firebase.
3. **Storage:** reprezintă o soluție de stocare în cloud, ce permite salvarea și gestionarea conținutului media într-un mod sigur și eficient, fiind proiectat pentru a oferi acces rapid la acestea în interiorul platformei.

Google Maps Platform [8]

Reprezintă o serie de servicii și instrumente furnizate de către Google, care permit integrarea de funcționalități avansate de hartă și locație în cadrul aplicațiilor. Este una dintre cele mai populare și fiabile soluții de cartografiere disponibile pe piață, fiind utilizată de milioane de dezvoltatori și utilizatori din întreaga lume. Dintre toate resursele puse la dispoziție de către acest mediu de dezvoltare, voi detalia elementele de interes, utilizate în contextul aplicației:

- **Maps SDK for Android:** este o bibliotecă destinată platformei Android, ce permite afișarea de hărți interactive în cadrul aplicației, oferind utilizatorilor posibilitatea de a explora și naviga într-o selecție largă de locații din întreaga lume. Aspectul hărților poate fi personalizat și se pot adăuga marcaje, poligoane, linii sau alte elemente, pentru a evidenția punctele de interes sau a afișa informații suplimentare. Utilizatorii pot efectua gesturi de glisare, zoom, rotație sau căuta locații, pentru a putea descoperi elementele de pe hartă într-un mod simplu și interactiv.
- **Places API:** facilitează accesul la o bază de date extinsă de locații și puncte de interes la nivel global, precum restaurante, hoteluri, magazine, puncte de interes turistic sau zone de recreere. Utilizatorul poate căuta obiective pe baza unor criterii cum ar fi denumirea, adresa sau tipul de locație și are posibilitatea de a găsi informații detaliate, relevante despre aceste locații, spre exemplu numele, adresa, coordonatele geografice, site-ul web sau orele de funcționare.

2.3.5 Limitări

Această secțiune se axează pe descrierea restricțiilor și a constrângerilor tehnice pe care le au unelele prezentate anterior, deși ele vin cu o succesiune de beneficii.

Android Studio necesită o configurație hardware adecvată pentru a putea funcționa într-un mod eficient, însă folosește o cantitate semnificativă de memorie RAM, iar dacă sistemul nu îndeplinește aceste cerințe minime, performanța și timpul de răspuns pot fi afectate. În plus, timpul de construcție al aplicațiilor **Android** poate dura mai mult timp pentru proiectele mai mari, ce conțin mai mult cod. Toți acești factori pot influența negativ timpul de dezvoltare necesar pentru implementarea aplicației.

Firebase are anumite restricții ale planului gratuit, cum ar fi limite privind cantitatea de date stocate, traficul de rețea sau numărul de cereri către anumite API-uri. Aceste limite pot necesita o actualizare la un plan plătit, pentru a putea profita de resurse suplimentare. De asemenea, bazele de date în timp real au anumite restricții referitoare la dimensiunea datelor, viteza de sincronizare și complexitatea interogărilor, cu posibilitatea de a afecta performanța și scalabilitatea aplicației.

Google Maps Platform poate fi supusă limitărilor de utilizare, spre exemplu numărul maxim de cereri de locații într-un anumit interval de timp. Similar cu Firebase, utilizarea serviciilor acestei platforme poate duce la costuri suplimentare, în funcție de volumul cererilor efectuate pentru fiecare API în parte. Un alt aspect este reprezentat de limitările de accesibilitate în cazul unor țări, regiuni geografice sau pentru anumite tipuri de conturi.

2.3.6 Controlul versiunilor și organizarea sarcinilor de lucru

Controlul versiunilor reprezintă o metodă esențială în dezvoltarea software, ce permite gestionarea și urmărirea modificărilor aduse codului sursă pe parcursul timpului. În acest sens, am creat un repositoriu **Git**, care este stocat pe un server remote, **GitHub**, ce oferă un mediu centralizat pentru partajarea și vizualizarea codului.

În plus, față de capacitatea de a putea salva codul sursă, platforma **GitHub** oferă soluții pentru o organizare mai bună a proiectului. Folosind secțiunea *Projects* din cadrul interfeței, am putut să grupez funcționalitățile prezente în contextul aplicației în mai multe categorii, reprezentând câte un *Project*.

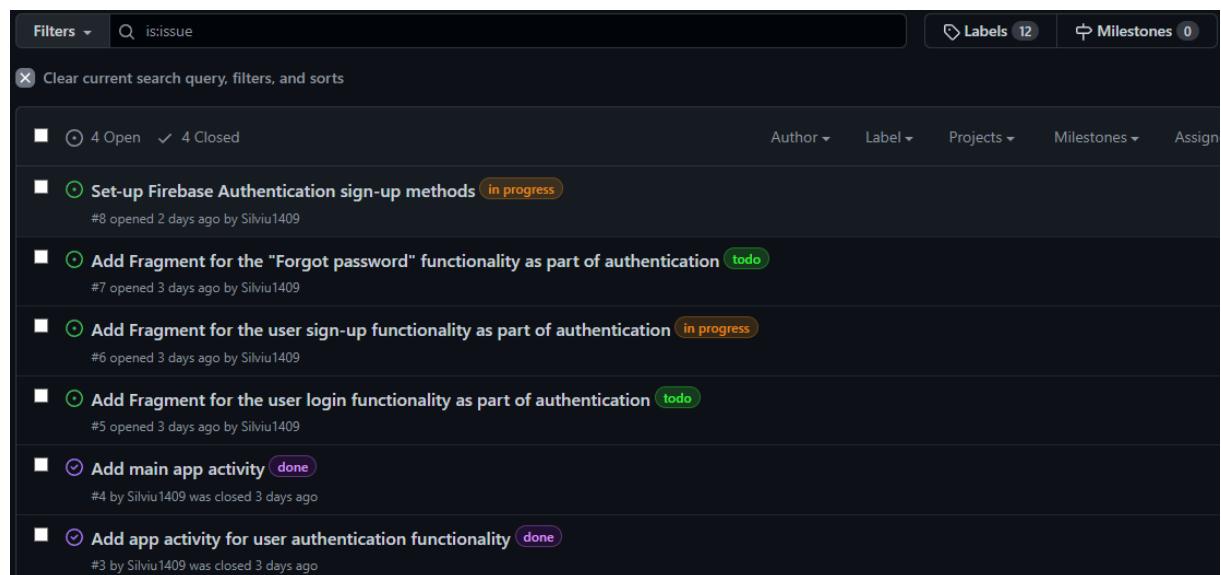


Figura 2.7: Utilizarea GitHub Projects

Fiecare *Project* are la rândul lui mai multe *task-uri*, sub formă de *issues*, care pot să fie *Open* sau *Closed*, în funcție de finalitatea activității. După cum se poate observa și în figura 2.7, acestea pot să aibă atribuite etichete, ce pot fi personalizate în funcție de nevoi. Spre exemplu, pentru a putea ține evidența sarcinilor de lucru ce trebuie implementate, am creat 3 etichete noi: *todo* (pentru *task-urile* neîncepute, dar care urmează să fie implementate), *in progress* (pentru *task-urile* în curs de implementare, la care se lucrează în momentul de față) și *done* (pentru *task-urile* îndeplinite).

2.4 Compararea cu aplicații similare

Pe piață există unele aplicații care se concentrează asupra aceleiași tematici, iar printre cele mai populare se numără TripAdvisor, Yelp și Foursquare. În cele ce urmează, voi explica similaritățile și factorii distinctivi față de aceste platforme.

- Asemănător cu aplicațiile enumerate mai devreme, utilizatorii pot vedea recenzii și evaluări ale altor utilizatori pentru punctele de interes, primind recomandări personalizate în funcție de preferințele acestora.
- Unele funcționalități avansate ale Foursquare sunt disponibile doar în interiorul versiunii plătite, spre deosebire de aplicația pe care o dezvoltă, ce nu va avea o astfel de variantă.
- În cazul acestor platforme, utilizatorii pot aprecia pe o scară de la 1 la 5 experiența avută într-o anumită locație. Opiniile și evaluările acestora pot fi subiective și pot varia în funcție de preferințele personale, iar acest lucru poate afecta încrederea altor utilizatori în aceste recenzii furnizate, încărcând informațiile furnizate să fie incorecte, inexacte sau neactualizate. Pentru a nu întâmpina această problemă, în cadrul rețelei mele de socializare, utilizatorii pot descrie experiențele avute în astfel de locații sub forma unei recenzii, ce apare în cadrul unei postări, dar nu pot acorda note care să influențeze sistemul de notare al acelor puncte de interes. Recenzia este analizată de către modelul de Inteligență Artificială (descriș în secțiunea 4.1 a acestei lucrări), iar pe baza evaluării textului de către acesta se stabilește un scor, ce va reprezenta tipul de recenzie (negativă, neutră sau pozitivă).
- În contextul acestor aplicații, anunțurile și reclamele intruzive prezente pe platformă pot afecta designul și experiența utilizatorului, prin încetinirea aplicației, irelevanța pentru utilizator sau riscul de clicuri accidentale. Această componentă este conținută într-o secțiune specială a aplicației, unde locațiile pot să fie promovate, iar apoi recomandate utilizatorilor, dacă acestea intră în sfera lor de interes.

Capitolul 3

Prezentarea aplicației

În acest capitol voi prezenta în detaliu aplicația **TownTrekker**, prin includerea raționamentului din spatele alegerii acestui nume, arhitectura sistemului și funcționalitățile cheie ale acestei platforme de *social media*.

3.1 Alegerea titlului

Un nume relevant pentru o astfel de rețea de socializare este important, pentru a face ca aplicația să fie ușor de identificat și reținut de către utilizatori. În plus, o denumire potrivită tematicii aplicației poate comunica în mod clar valorile și mesajul pe care platforma vrea să le transmită, dar de asemenea facilitează comunicarea și promovarea acesteia pe piață.

Plecând de la tema aplicației, ce constă într-o platformă în cadrul căreia utilizatorii pot explora și descoperi puncte de interes dintr-un oraș, am ajuns la denumirea **TownTrekker**, formată din:

- „**Town**” (oraș): pentru a evidenția faptul că aplicația se concentrează pe explorarea și descoperirea punctelor de interes dintr-un oraș.
- „**Trekker**” (explorator): sugerează că aplicația încurajează utilizatorii să exploreze și să călătorească prin oraș, descoperind locuri și experiențe noi. Aceasta poate inspira utilizatorii să devină exploratori urbani și să descopere aspecte mai puțin cunoscute ale orașului lor.

Astfel, numele „TownTrekker” combină elementele de explorare urbană și descooperire a unui oraș, subliniind natura aventuroasă și informativă a aplicației, și sugerează că utilizatorii pot naviga și exploră cu ușurință diferite atracții și destinații dintr-un oraș.

3.2 Generarea logo-ului

Logo-ul aplicației a fost generat folosind **Playground AI** [20], o platformă online ce oferă utilizatorilor acces la modele de Inteligență Artificială pre-construite, ce pot fi personalizate, pentru a crea elemente precum imagini, logo-uri sau artă. Acestea pot fi generate folosind o descriere în cuvinte, sub forma unui *prompt*, sau plecând de la o altă imagine. Opțiunile de personalizare diferă în funcție de planul de subscriptie ales pe platformă. Spre exemplu, în versiunea gratuită, utilizatorul poate folosi doar algoritmii *Stable Diffusion v1.5* sau *Stable Diffusion v2.1*, iar în versiunea plătită acesta poate utiliza și *DALL-E 2*, această variantă oferind astfel o gamă mai largă de funcționalități pe care clientul le poate utiliza.

Pentru a genera acest logo, am creat o descriere pentru aplicație, pe care am simplificat-o în mai multe cuvinte cheie, ca parte a preprocesării datelor, cu scopul de a îmbunătăți performanța modelului folosit, a reduce viteza de procesare și pentru o mai bună clasificare a textului. Am testat doar modelul *Stable Diffusion v1.5* [22], întrucât doar acesta poate fi folosit pe platformă pentru a genera un logo, iar acest model are variabile ce influențează rezultatul final:

- **Image Dimensions** - reprezintă dimensiunile (lungime și lățime) ale imaginii pe care vrem să o generăm, iar acestea au valori predefinite
- **Prompt Guidance** - poate avea valori de la 0 la 30 și indică modelului cât de mult să se apropie de detaliile menționate în prompt
- **Quality & Details** - poate avea valori de la 10 la 150 și reprezintă numărul de pași pe care modelul îi efectuează; durata pentru generarea imaginii crește o dată cu un număr mai mare de pași, dar poate produce un rezultat de o calitate mai mare
- **Seed** - poate fi o valoare setată manual (aceeași valoare a seed-ului va produce aceeași imagine) sau o secvență pseudo-aleatoare (va rezulta în variații noi ale imaginii)
- **Sampler** - acesta este utilizat pentru a simula evoluția procesului stocastic în cadrul modelului; include o listă de algoritmi ce pot fi folosiți pentru a produce rezultate diferite, printre care *plms*, *k_euler*, *k_dpm_2* sau *k_lms*

După cum se poate observa în figura 3.1, am obținut o diversitate mare de varianțe ale imaginilor, prin ajustarea valorilor pentru hiperparametri menționați anterior, cu scopul de a ajunge la un logo minimalist, dar cât mai reprezentativ pentru aplicație.

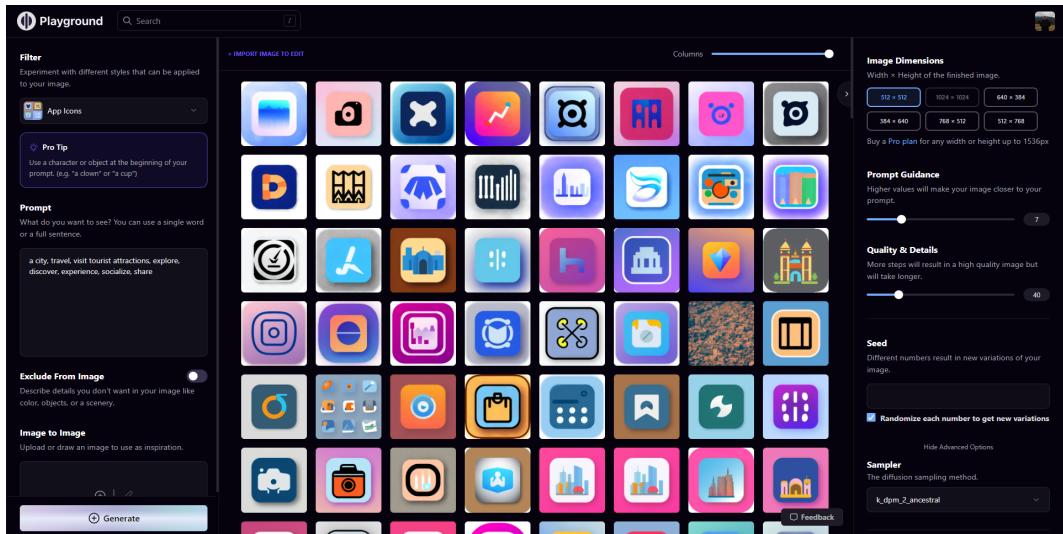


Figura 3.1: Interfața grafică a Playground AI, alături de câteva logo-uri generate

În cele din urmă, am ajuns la logo-ul ilustrat în figura 3.2, ce poate fi replicat pe platformă, folosind *prompt-ul* „a city, travel, visit tourist attractions, explore, discover, experience, socialize, share” și setând următoarele valori pentru parametrii:

- **Image Dimensions** - 512 x 512; vreau ca acest logo să aibă raportul de 1:1, iar aceasta a fost singura dimensiune pe care am putut să o aleg cu acest raport
- **Prompt Guidance** - 6; platforma recomandă valori între 7 și 10, însă o valoare mai mare poate crește timpul de execuție, deoarece modelul trebuie să țină cont de toate detaliile din *prompt*, iar din acest motiv am ales o valoare mai mică pentru un timp de execuție rezonabil, dar și pentru o varietate mai mare a rezultatelor
- **Quality & Details** - 40; platforma recomandă valoarea de 50, deoarece pe măsură ce această variabilă crește, cu atât timpul de execuție este mai mare (proporțional cu numărul de pași efectuați de model)
- **Seed** - 164144139; am încercat valori alese pseudo-aleatoriu, iar în cele din urmă, logo-ul final îl folosește pe acesta
- **Sampler** - k_dpm_2

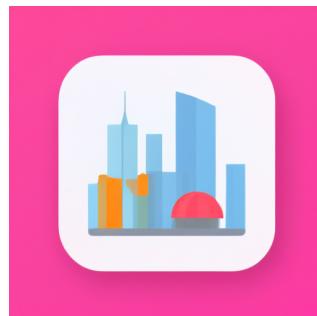


Figura 3.2: Logo-ul aplicației

3.3 Arhitectura proiectului

În acest proiect am utilizat o variație a arhitecturii **MVC**(Model-View-Controller) [26], pentru a organiza și structura componentele aplicației.

Model

Modelul reprezintă partea care se ocupă de gestionarea datelor și logica de business. Aici sunt incluse clasele și structurile de date care gestionează starea aplicației, manipulează și salvează informațiile necesare.

În cazul de față, din această categorie fac parte clasele Kotlin care definesc structura datelor utilizate în aplicație, spre exemplu atributele conținute în cadrul unei postări sau informațiile reținute de către aplicație pentru un utilizator.

View

View(Vizualizarea) reprezintă componența care gestionează afișarea interfeței utilizatorului, ce include elementele grafice, aspectul și interacțiunea cu utilizatorul. Vizualizarea primește datele din *Model* și le afișează utilizatorului într-un mod adecvat. În plus, aceasta poate interacționa cu utilizatorul pentru a prelua informații sau a notifica evenimente către Controller.

În contextul aplicației, din **View** fac parte fișierele de tip *XML*(Extensible Markup Language), ce conțin și definesc componente interfeței utilizatorului, cum ar fi aspectul vizual, vizualizările și elementele de interfață (spre exemplu butoane, câmpuri de text sau imagini).

Controller

Clasele de control (Controllers) acționează ca intermediar între Model și View, fiind responsabile pentru gestionarea interacțiunii dintre aceste două componente. Controllerul primește acțiunile utilizatorului din View și le prelucrează, efectuând operațiile necesare în Model, iar apoi transmite rezultatele către View pentru a actualiza interfața utilizatorului.

În acest proiect, un **Controller** poate conține metode prin care sunt gestionate acțiuni ale utilizatorului, printre care se numără crearea unui cont, logarea sau crearea unei postări noi.

Astfel, arhitectura **MVC** asigură o separare clară a responsabilităților între gestionarea datelor, afișarea interfeței utilizatorului și logica de control. Aceasta facilitează dezvoltarea, testarea și întreținerea aplicației, permitând unui dezvoltator sau unei echipe de dezvoltare să lucreze eficient, în paralel, pe diferite componente ale aplicației.

3.4 Structura aplicației

O aplicație **Android** este structurată în jurul conceptului de componente, care reprezintă unități independente de funcționalitate. Astfel, sunt utilizate două componente principale în cadrul arhitecturii unei aplicații Android: activități (Activities) și fragmente (Fragments).

Activities

Activitățile reprezintă ecranele individuale ale aplicației Android și se ocupă de interacțiunea directă cu utilizatorul, coordonând afișarea și interacțiunea cu elementele de interfață vizuală (View). O activitate poate avea un ciclu de viață specific, care include etape precum crearea, pornirea, intreruperea și distrugerea. În arhitectura **MVC**, activitățile au adesea un rol de controlor (Controller), gestionând logica aplicației și interacțiunea între model și vizualizare.

Fragments

Fragmentele sunt elemente folosite pentru a crea interfețe complexe și reutilizabile, ce permit împărțirea ecranului în părți independente. Acest aspect facilitează gestionarea ecranelor și interacțiunea între diferite componente ale aplicației, fie că este vorba despre activități sau fragmente.

Activitățile și fragmentele lucrează împreună pentru a crea experiențe de utilizare bogate și complexe în aplicațiile Android. Prin intermediul acestor componente, aplicația poate interacționa cu utilizatorul, afișa date, stabili conexiuni cu servicii externe și gestionează fluxul de navigare între diferite ecrane.

În contextul acestei aplicații sunt prezente două activități cheie:

- **Activitatea de Autentificare**, ce este responsabilă pentru gestionarea procesului de autentificare în aplicație. Scopul său principal este de a permite utilizatorilor să își creeze un cont pe platformă și să se conecteze pentru a accesa funcționalitățile și resursele aplicației.
- **Activitatea Principală** reprezintă ecranul principal al aplicației, oferă acces la funcționalitățile de bază și este afișată după ce utilizatorul s-a autentificat cu succes.

3.4.1 Activitatea de Autentificare

Așa cum am subliniat anterior, **Activitatea de Autentificare** se ocupă de coordonarea operațiunii de autentificare și reprezintă ecranul inițial al aplicației. După cum se poate observa în figura 3.3, această activitate este responsabilă de gestionarea și afișarea a 4 fragmente distincte, ce sunt interconectate: Splash (Ecranul de pornire), Logare,

Autentificare și Resetare. De asemenea, se poate constata că această activitate, prin fragmentele sale, face legătura cu **Activitatea Principală**.

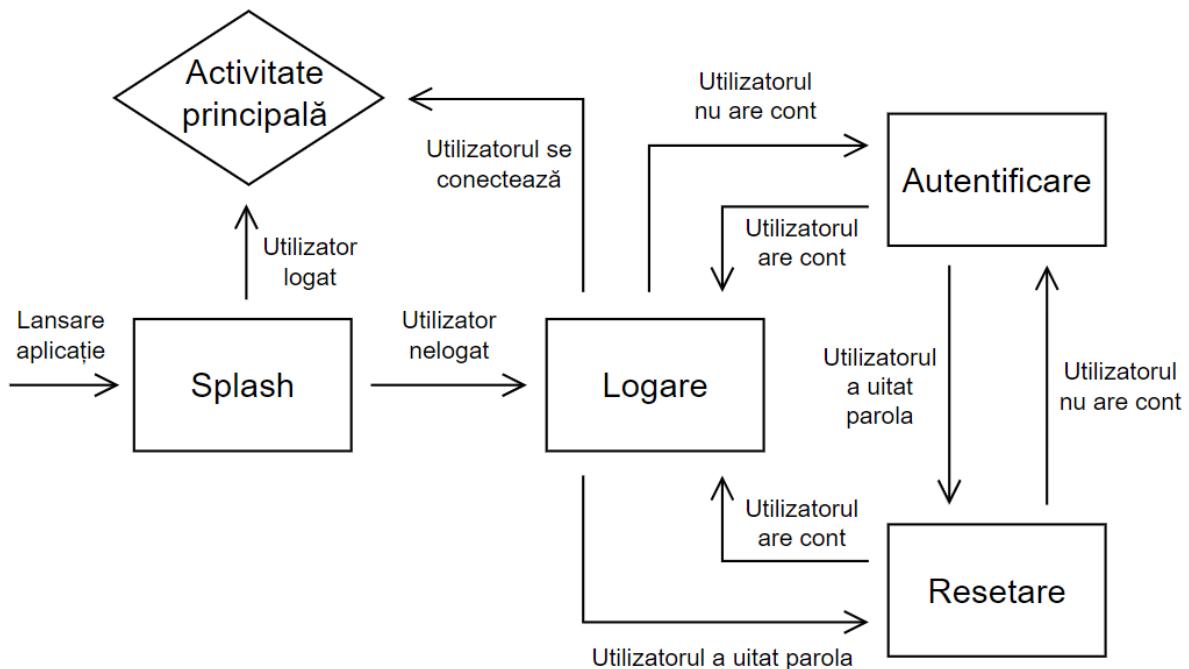


Figura 3.3: Diagramă de navigare pentru activitatea de autentificare

Splash

Plecând de la figura 3.3, fragmentul **Splash**(Ecranul de pornire) este prima interfață pe care utilizatorul o vede atunci când lansează aplicația. În cadrul acestui fragment este prezentă o animație personalizată, ce este construită pe baza logo-ului aplicației (figura 3.2), unde la începutul tranzitiei ecranul este gol, neavând nicio informație vizuală, dar pe parcursul afișării acestuia, logo-ul apare treptat, făcând astfel tranzitia plăcută.

Pe durata afișării acestei animații, în cadrul activității părinte (Activitatea de Autentificare) verific dacă utilizatorul s-a conectat anterior pe platformă și a rămas logat într-o sesiune anterioară. Acest lucru este posibil, întrucât **Firebase Authentication**, unealtă folosită pentru identificarea utilizatorilor pe platformă, utilizează un sistem de autentificare bazat pe *token-uri* pentru a valida și autentifica utilizatorii. În procesul de verificare a existenței unui utilizator curent, această unealtă verifică dacă în sesiunea actuală a aplicației există un astfel de token valid, existând 2 cazuri.

Tokenul de autentificare este valid

În această situație verific dacă datele despre utilizatorul respectiv sunt salvate local, sub formă de *SharedPreferences* (un fișier XML, în cadrul căruia datele sunt reprezentate într-un format de tip cheie-valoare), fișier creat cu scopul de a reduce numărul de apeluri

redundante către baza de date.

Dacă datele sunt prezente local, la finalul animației se face tranziția către Activitatea Principală. În caz contrar se apelează baza de date ce conține datele despre utilizatorul conectat, iar în cazul în care am preluat datele cu succes se realizează aceeași tranziție și datele despre utilizator sunt trimise ca parametru, cu scopul de a le salva ulterior într-un fișier de tip *SharedPreferences*.

Tokenul de autentificare este invalid sau expirat

În acest caz, utilizatorul este redirecționat automat către ecranul de logare, unde poate introduce credențialele pentru a se loga în contul său. Pentru cazurile în care utilizatorul a uitat parola contului sau nu are un cont creat, acesta are posibilitatea de a naviga către paginile ce oferă aceste funcționalități.

Logare

Fragmentul de logare are o interfață ce este concepută într-un mod ușor de înțeles, având un aspect simplu și intuitiv. În cadrul acestei pagini, utilizatorii se pot conecta pe platformă prin 2 modalități.

Autentificarea cu contul Google

Această funcționalitate este reprezentată în interfață prin intermediul unui buton dedicat, ilustrat în figura 3.4, ce permite utilizatorilor să acceseze rapid și ușor aplicația, folosind datele de acces ale contului Google deja existent. În plus, este o metodă eficientă de conectare, deoarece nu necesită introducerea manuală a unor date noi de identificare. Dacă există un profil asociat cu contul Google, utilizatorul este direcționat către activitatea principală, iar datele asociate cu contul sunt transmise drept parametru.

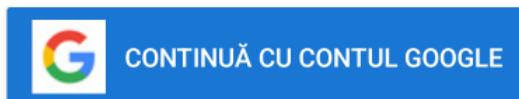


Figura 3.4: Logarea cu contul Google

De asemenea, prin această modalitate, utilizatorii au posibilitatea de a-și crea un cont nou, în cazul în care nu au deja unul. În această situație se generează un astfel de profil pentru platformă, iar apoi utilizatorul este trimis spre activitatea principală.

Conectarea folosind email și parolă

Această opțiune permite utilizatorilor să acceseze aplicația folosind adresa de email și parola asociate contului lor existent, oferind un nivel suplimentar de flexibilitate și

permite utilizatorilor să acceseze platforma, fără să depindă de existența unui cont Google. De asemenea, conectarea în acest mod permite utilizatorilor să își modifice parola, funcționalitate ce nu este disponibilă în cazul celeilalte metode de autentificare.

The image shows a user interface for logging in. At the top is a field labeled "Email" with a horizontal line underneath. Below it is a field labeled "Parolă" (Password) with a horizontal line underneath. To the right of the password field is a small circular icon containing an eye symbol, which typically indicates a password visibility toggle. Below these fields is a blue rectangular button with white text that reads "LOGHEAZĂ-TE".

Figura 3.5: Logarea cu email și parolă

Operațiunea de față este prezentată vizual conform figurii 3.5 și este constituită din două câmpuri de text în care utilizatorii pot introduce adresa de email, respectiv parola (câmp ce are și opțiunea de ascundere/afișare parolă) asociată contului lor deja existent.

Totodată, există și un buton de logare ce permite utilizatorilor să inițieze procesul de conectare în aplicație, o dată ce introduc corect datele de acces. Dacă acestea sunt valide, utilizatorii sunt ghidați către activitatea principală, unde pot folosi funcționalitățile aplicației, iar în caz contrar aceștia sunt informați cu privire la completarea datelor în mod eronat și sunt îndrumați să recompleteze câmpurile cu datele corecte.

[Nu ai cont? Înregistrează-te aici.](#)

[Ai uitat parola?](#)

Figura 3.6: Legătura dintre Logare și celelalte fragmente

Pe lângă aceste elemente, fragmentul este interconectat cu celelalte două fragmente (**Autentificare și Resetare**) prin intermediul a două *TextView-uri* (câmpuri de text) interactive, în concordanță cu reprezentarea din figura 3.6, iar în urma apăsării acestor butoane text, utilizatorul este redirecționat către fragmentele specifice.

Autentificare

Acest fragment îndeplinește un rol esențial în contextul aplicației, oferind utilizatorilor posibilitatea de a-și crea conturi noi pe platformă, folosind adresa de email și alegând o parolă.

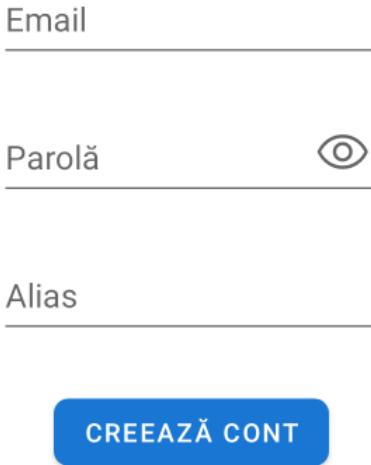


Figura 3.7: Crearea unui cont cu email și parolă

În concordanță cu figura 3.7, interfața vizuală este asemănătoare cu cea a paginii de logare, prezentată anterior. Sunt prezente 3 câmpuri de text, în care utilizatorul trebuie să introducă pe rând adresa de e-mail, parola și un alias (numele pe care ceilalți utilizatori ai aplicației o să îl vadă pe platformă).

Conform *Firebase Authentication*, la momentul de față o parolă trebuie să fie formată din minim 6 caractere pentru a putea fi considerată eligibilă, iar pentru alias am setat manual o limită de 15 caractere.

După introducerea datelor în câmpurile de text enumerate anterior, utilizatorul poate să își creeze un cont pe această platformă. În procesul de implementare, am creat o clasă denumită **User**, ce conține următoarele attribute:

- **uid** - reprezintă id-ul (cheia) asociată utilizatorului; este generată automat în momentul în care se creează un cont nou, are valoare unică și o voi folosi în baza de date pentru a diferenția documentele (înregistrările) ce conțin date despre utilizatori
- **email** - semnifică adresa de email, introdusă de utilizator
- **alias** - este alias-ul adăugat de către user
- **urmaritori** - inițializată cu o listă vidă, acest atribut reprezintă o listă cu id-urile utilizatorilor de pe platformă care urmăresc acest user
- **urmăreste** - similar cu atributul anterior, este inițializată în același mod, dar semnifică o listă a cheilor utilizatorilor pe care acest user îi urmărește pe aplicație
- **bio** - pornește de la un sir de caractere nul, ce ulterior poate fi modificat și reprezintă o scurtă descriere despre utilizator, ce poate fi vizualizată de către alții useri ai platformei; aceștia pot include informații relevante despre ei însiși, precum hobby-uri sau interese

- **parola** - este o variabilă opțională, ce are o valoare atribuită în cazul în care contul a fost creat folosind un email și o parolă, reprezentând parola dată de user

Similar cu ecranul de logare, acest fragment este conectat cu celelalte două componente (**Logare** și **Resetare**) prin două *TextView-uri* interactive (figura 3.8), iar prin apăsarea lor, user-ul este ghidat către fragmentele menționate anterior.

Ai deja cont? Loghează-te aici.

Ai uitat parola?

Figura 3.8: Legătura dintre Autentificare și celelalte fragmente

Resetare

În contextul acestui ecran, utilizatorii au abilitatea de a-și reseta parola asociată contului lor, dacă acesta este creat folosind email și parolă. În acord cu figura 3.9, în interfață grafică este prezent un câmp de text, unde utilizatorul trebuie să introducă adresa de email asociată contului acestuia. De asemenea, este prezent un buton, iar prin apăsarea acestuia se verifică dacă acel email există pe platformă, caz în care aplicația trimite un email către adresa specificată, ce conține instrucțiunile pentru procesul de resetare a parolei.

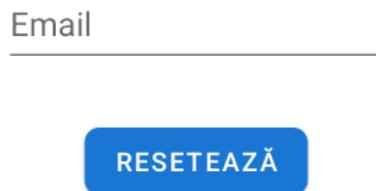


Figura 3.9: Resetarea parolei unui cont deja existent

În același fel cu celelalte două componente prezentate anterior, fragmentul curent este conectat cu acestea cu ajutorul a două *TextView-uri* interactive (figura 3.10), iar în urma apăsării acestora utilizatorul este transferat către fragmentele descrise în prealabil.

Nu ai cont? Înregistrează-te aici.

Ai deja cont? Loghează-te aici.

Figura 3.10: Legătura dintre Autentificare și celelalte fragmente

3.4.2 Activitatea Principală

În acord cu descrierea făcută la începutul acestei subsecțiuni, **Activitatea Principală** reprezintă ecranul central al aplicației, oferind acces la funcționalitățile de bază ale aplicației. În contextul acestei activități, utilizatorii pot vizualiza postări, descoperi locații noi folosind harta interactivă, primi recomandări personalizate de locații sau personaliza profilul.

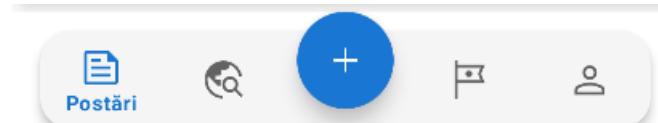


Figura 3.11: Bara de navigație a activității principale

Navigația către fragmentele (paginile) incluse în această activitate se face conform unei bare de navigație, ilustrată în figura 3.11. Aceasta conține cele 4 fragmente principale, reprezentate sub forma unor pictograme reprezentative, având prezent și textul asociat cu titlul paginii, în momentul în care utilizatorul selectează acel fragment.

Ordinea, de la stânga la dreapta a fragmentelor incluse este următoarea: **Postări**, **Explorează**, **Descoperă** și **Cont**. De asemenea, în centrul barei de navigație se află un buton prin care utilizatorii pot adăuga o postare nouă.

Structura postărilor

În cadrul acestei aplicații, postările făcute de către utilizatori au o structură bine definită, ce facilitează organizarea și prezentarea conținutului într-un mod eficient și atractiv. În procesul de implementare, structura unei postări este definită în cadrul unei clase de date de tip Kotlin, cu numele **Postare**, ce conține următoarele câmpuri:

- **id** - este cheia (identificatorul) postării, și este generată automat de către **Firebase Firestore Database**, în momentul în care postarea este adăugată în baza de date
- **user** - identificatorul utilizatorului care a făcut acea postare
- **numeUser** - alias-ul utilizatorului
- **numeLocatie** - numele locației căreia i se face recenzia în cadrul postării
- **adresaLocatie** - adresa la care se află acea locație
- **tipLocatie** - tipul locației date (spre exemplu *restaurant, bar, zoo*)
- **categorieLocatie** - categoria din care face parte locația, bazată pe tip (mai multe tipuri formează o categorie)

- **aprecieri** - numărul de aprecieri pe care îl strânge postarea
- **descriere** - componenta de text, ce reprezintă recenzia pe care utilizatorul o face locației
- **comentarii** - o listă, ce conține comentariile acestei postări
- **media** - o valoare de tip *Boolean* (adevărat sau fals), ce spune dacă utilizatorul a încărcat sau nu fișiere media în momentul în care acesta a făcut postarea
- **iconUser** - are același tip ca atributul menționat anterior și precizează dacă utilizatorul are asociată o poză de profil a contului
- **scorRecenzie** - scorul rezultat în urma analizei textului din *descriere*
- **tiprecenzie** - tipul de recenzie, calculat pe baza atributului *scorRecenzie*
- **listaMedia** - o listă ce poate să fie vidă (dacă atributul *media* este fals) și conține lista referințelor din Firebase Storage pentru fișierele media din postare

Clasificarea locațiilor

Tipurile locațiilor pentru această platformă sunt preluate folosind **Google Places API**, în momentul selectării locației dorite, iar aceste tipuri sunt definite în documentația oficială a platformei **Google Maps** [10]. Din această colecție am selectat câteva tipuri relevante, pe care apoi să le grupez în categorii [9], pentru o clasificare cât mai relevantă.

În figura 3.12 sunt reprezentate grafic tipurile locațiilor alese, ce sunt grupate, pe culori, în categorii mai mari, astfel:

- **food and drink** - cuprinde o varietate de locuri unde oamenii pot mâncă și/sau bea, iar această categorie include baruri, cafenele, cluburi de noapte, restaurante și brutării/patiserii
- **retail** - conține o mulțime de spații comerciale care sunt concepute și utilizate pentru vânzarea de bunuri și servicii către clienți, iar din această categorie fac parte locații de tipul magazinelor fizice, centre comerciale, supermarket-uri, farmacii și altele
- **services** - conține diverse spații și facilități specializate în furnizarea diferitelor servicii către public, iar la momentul de față include facilități și spații de cazare (tipul *lodging*)
- **entertainment** - cuprinde locurile concepute și destinate pentru a oferi distracție și divertisment publicului, inclusiv atracțiile turistice, parcurile de distracții, teatrele, muzeele și cinematografele

- **outdoor** - include spații și locații unde se pot realiza activități în aer liber, printre care se enumera parcurile, grădinile zoo, stadioanele și spațiile pentru camping
- **other** - este formată din locațiile care nu se încadrează în categoriile anterioare

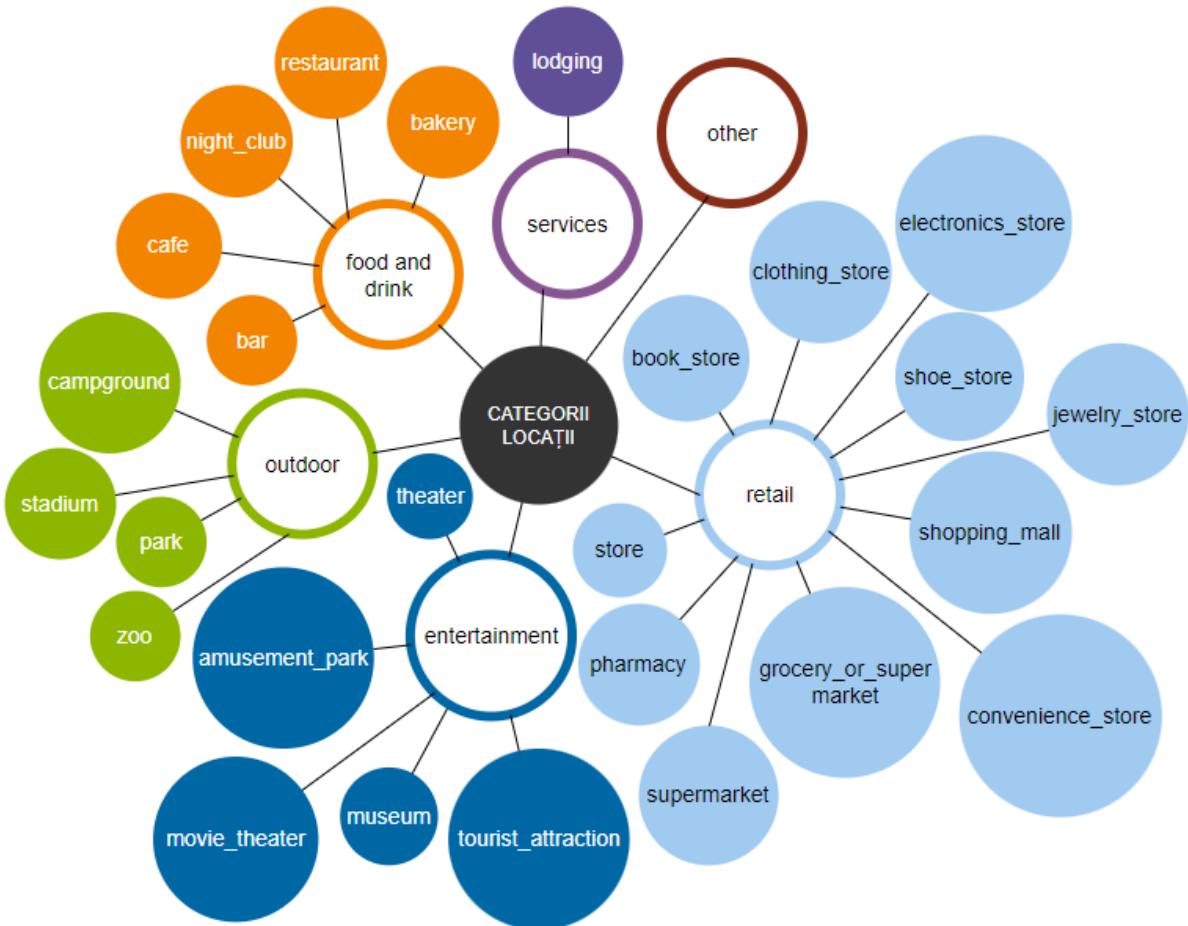


Figura 3.12: Categoriile de locații și tipurile asociate acestora

Acste categorii vor fi importante în ceea ce privește oferirea de conținut (postări și recomandări) personalizate, în funcție de categoriile cu care utilizatorul interacționează pe platformă, elemente ce vor fi dezvoltate ulterior în subsecțiunile 4.2 și 4.3, din cadrul acestei lucrări.

Adăugarea unei postări

În momentul în care utilizatorul apasă butonul destinat adăugării unei postări noi, se va deschide automat un dialog, ilustrat în figura 3.13, prin intermediul căruia utilizatorul poate furniza informațiile relevante pentru postare și aceasta va fi procesată în consecință.

Aduagă o postare

Locație

 Adaugă o locație

Descriere

Adaugă o descriere

ADAUGĂ IMAGINE/VIDEO

ANULEAZĂ POSTEAZĂ

Figura 3.13: Dialogul de adăugare a unei postări noi

Conform acestei figuri, utilizatorul are la dispoziție o bară de căutare personalizată, prin care poate căuta locația pentru care vrea să realizeze postarea, și implicit recenzia. Aceasta are sugestia de text „Adaugă o locație” și folosește în mod intern **Google Places API**, din cadrul platformei **Google Maps**. Pe măsură ce utilizatorii completează numele locației pe care vor să o caute, bara de căutare vine cu câteva recomandări pentru auto-completarea numelui locației, bazat pe textul prezent la momentul de față. Astfel, utilizatorii pot găsi într-un mod mai ușor punctul de interes căutat.

După selectarea locației dorite, folosind **Google Places API** pot prelua numele, adresa și tipul locației respective, cu scopul de a le salva în câmpurile asociate acestora din clasa **Postare**, iar în funcție de tipul returnat, locația va fi inclusă într-o categorie, din cele precizate anterior.

De asemenea, utilizatorul poate adăuga o descriere, prin intermediul câmpului de text cu sugestia „Adaugă o descriere” și reprezintă recenzia pentru locația selectată anterior. Această evaluare este apoi preluată și analizată conform pașilor din subsecțiunea 4.1, cu scopul de a obține scorul și tipul de recenzie, pentru a le adăuga postării.

În plus, utilizatorii pot adăuga conținut media, reprezentat de poze și videoclipuri, iar pentru această funcționalitate am setat manual o limită de 5 fișiere, cu dimensiuni de maxim 10 MB fiecare, cu scopul de a îmbunătăți performanța aplicației (prin preluarea unui număr mai mic de fișiere, cu dimensiuni mai mici, timpul necesar procesării acestora scade considerabil), dar și pentru o gestionare eficientă a spațiului oferit de către **Firebase Storage**.

În partea de jos a dialogului există următoarele două butoane:

- **Anulează** - prin apăsarea acestuia, utilizatorul va închide dialogul de față
- **Postează** - o dată cu selectarea acestui buton, se va crea o postare nouă, folosind datele furnizate de către utilizator, iar această postare va fi la rândul ei stocată în baza de date (Firebase Cloud Firestore), împreună cu fișierele ce vor fi încărcate în spațiul de stocare (Firebase Storage). În situația în care utilizatorul nu a selectat o locație (sau nu a adăugat o descriere sau introdus cel puțin un fișier media), acesta este notificat, printr-un mesaj ce apare pe ecran, prin care este îndrumat să completeze acele câmpuri

Fragmentul Postări

Acesta este pagina în cadrul căreia utilizatorul poate vedea postările celorlalți utilizatori de pe platformă. Postările sunt afișate sub forma unor *containere*, în cadrul unui **RecyclerView**, o componentă ce are capacitatea de a afișa elementele acesteia într-un mod eficient, întrucât aceasta le reciclează și reutilizează, pe măsură ce utilizatorul derulează sau interacționează cu lista de postări.

```
if (postare.id !in mainActivityContext.postariVizualizate  
    && postare.user != mainActivityContext.getUser()!!.uid) {  
  
    val timer = object : CountDownTimer( millisInFuture: 7500, countDownInterval: 1000 ) {  
        override fun onTick(millisUntilFinished: Long) {}  
  
        override fun onFinish() {  
            if(mainActivityContext.postariVizualizate.add(postare.id)) {  
                mainActivityContext.fisierPostariVizualizate.edit().remove( key: "vizualizate" ).apply()  
                mainActivityContext.fisierPostariVizualizate.edit().putStringSet( key: "vizualizate", ma  
            }  
        }  
    }.start()  
  
    holder.itemView.addOnAttachStateChangeListener(object : View.OnAttachStateChangeListener {  
        override fun onViewAttachedToWindow(v: View) {}  
  
        override fun onViewDetachedFromWindow(v: View) { timer?.cancel() }  
    })  
}
```

Figura 3.14: Verificarea unei postări care a fost deja afișată

Pentru a nu afișa postări pe care utilizatorul curent al aplicației le-a vizualizat deja, am proiectat o modalitate (ilustrată în figura 3.14) de a analiza postările afișate în această pagină. Pentru situațiile în care acea postare nu este făcută de către utilizatorul curent și nu se află în lista postărilor care au fost vizualizate de către acesta, am creat un cronometru ce verifică dacă postarea apare pe ecranul utilizatorului pentru o perioadă

de minim 7.5 secunde. În caz pozitiv, adaug identifierul asociat postării într-un set, pe care apoi îl salvez local, sub forma unui fișier de tip *SharedPreferences*, cu scopul persistenței datelor.

Dacă utilizatorul glisează prin postări și acestea nu mai apar pe ecran, cronometrul se resetează, pentru a nu lua în calcul postările pentru care cronometrul a fost inițializat, dar utilizatorul nu a apucat să le parcurgă conținutul. În plus, adaug postarea în lista celor vizualizate și pentru situațiile în care utilizatorul apreciază postarea sau deschide lista de comentarii asociată postării.

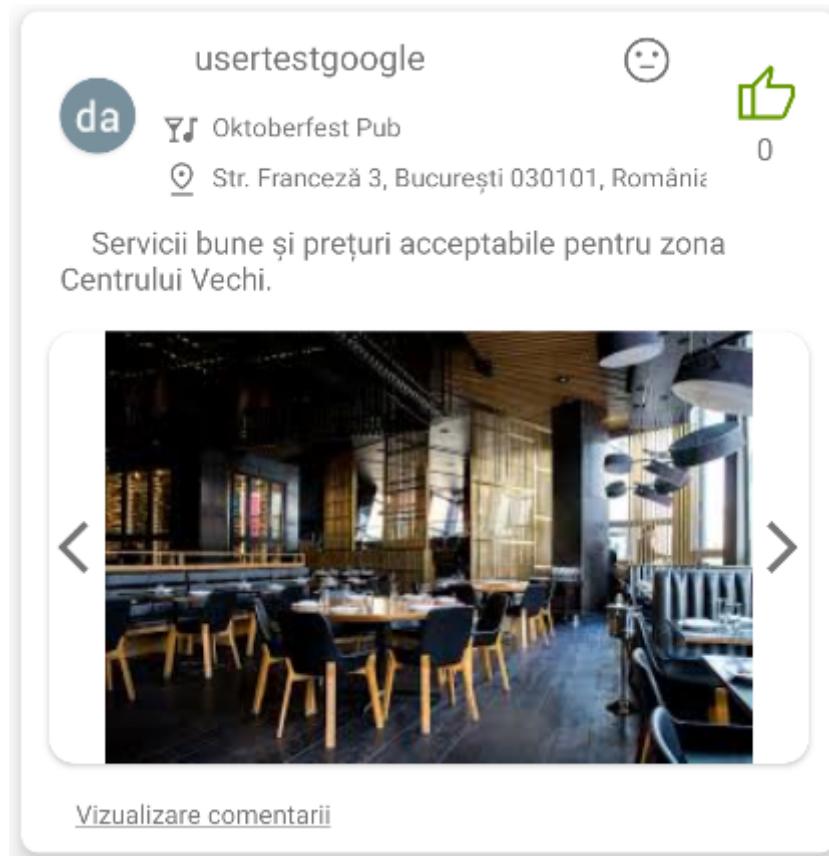


Figura 3.15: Exemplu de postare

Figura 3.15 conține o ilustrare a unei postări aflate pe platformă. Pentru cazul dat, se poate observa că această postare este făcută de către utilizatorul cu alias-ul *userstestgoogle*, pentru locația cu numele *Oktoberfest Pub*, de tipul *bar*, din categoria *food and drink*. De asemenea, postarea are 0 aprecieri, recenzia făcută de către utilizator a fost interpretată ca fiind una neutră (din pictograma cu sentiment neutră), iar acestei postări au fost atașate și fișiere media, ce pot fi accesate de către cititor. În plus, față de aceste aspecte vizuale, utilizatorul care vede această postare poate aprecia postarea (prin selecțarea butonului de apreciere), verifică profilul utilizatorului care a făcut postarea (prin apăsarea imaginii/pictogramei sale de profil) sau vizualiza comentariile asociate postării (prin apăsarea textului cu conținutul „Vizualizare comentarii”).

Verificarea profilului unui utilizator

Acest ecran apare în momentul în care un utilizator interacționează cu poza de profil a utilizatorului care a făcut o postare pe platformă. Un exemplu al acestei pagini este ilustrat în figura 3.16, în care sunt prezentate mai multe detalii, precum postările făcute, persoanele urmărite de acel utilizator sau persoanele care urmăresc acel utilizator (toate acestea sunt prezentate inițial sub forma numărului acestora, dar lista lor completă poate fi accesată prin apăsarea căsuțelor corespunzătoare). În plus, apare și bio-ul utilizatorului respectiv, iar cel care vizualizează profilul are posibilitatea de a adăuga (sau elimina) acel utilizator din lista persoanelor pe care le urmăresc pe platformă.



Figura 3.16: Vizualizarea detaliilor unui utilizator

Vizualizarea comentariilor unei postări

Comentariile unei postări sunt reprezentate în același mod, în cadrul unui ecran nou, folosind un dialog în care se află lista cu comentarii (exemplificată în figura 3.17) și un câmp de text prin care utilizatorul poate adăuga un comentariu nou (figura 3.18).



Figura 3.17: Comentariile unei postări

Figura 3.18: Funcționalitatea de adăugare a unui comentariu nou

Un comentariu conține numele utilizatorului care l-a adăugat și textul ce conține comentariul în sine, iar în cazul funcționalității de adăugare a unui comentariu nou, am limitat numărul de caractere la 100, pentru a spori relevanța textului prezent în acestea.

Fragmentul Explorează

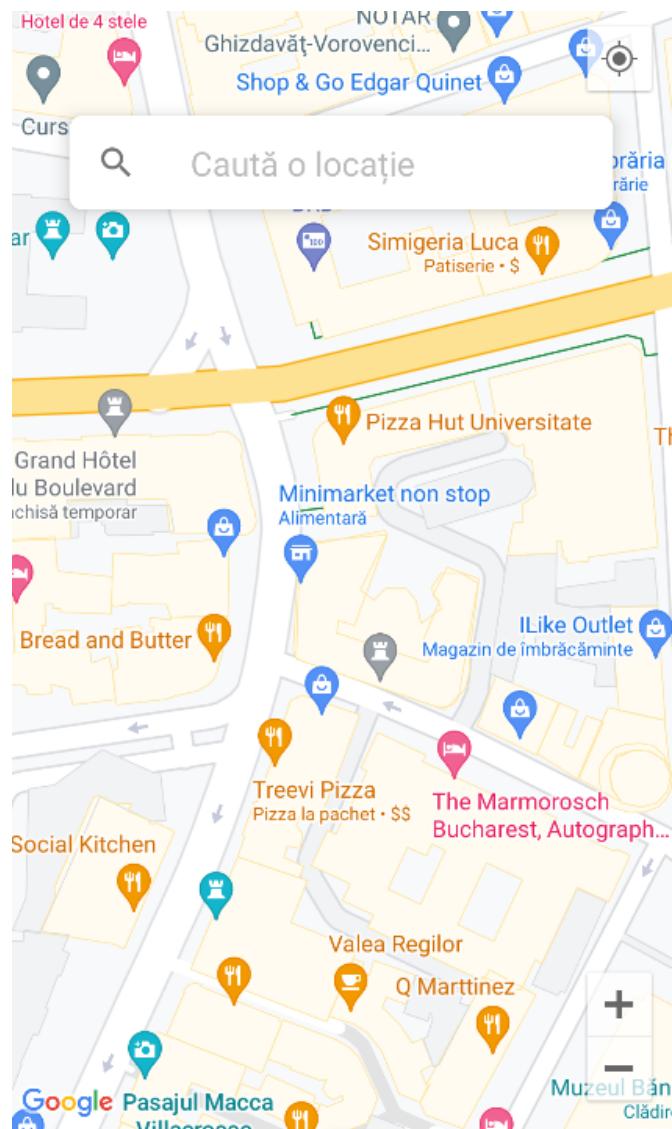


Figura 3.19: Pagina de explorare

Acest fragment este format dintr-o hartă interactivă, personalizată, în care utilizatorul poate explora împrejurimile, cu scopul de a găsi puncte de interes. În concordanță cu ilustrarea din figura 3.19, harta este modificată, folosind platforma **Google Maps**,

prin creșterea densității pentru punctele de interes (utilizatorul poate astfel să vadă mai multe locații, comparativ cu o hartă normală), eliminarea locațiilor din categoriile care nu aparțin celor menționate anterior (pentru creșterea relevanței elementelor apărute) și prin simplificarea afișării pentru clădiri (în mod normal ele sunt afișate ca elemente tridimensionale, dar pentru a optimiza navigarea am decis să păstreze afișarea bidimensională).

De asemenea, am păstrat trei funcționalități de bază prezente în cadrul unei hărți prezente în aplicația **Google Maps**:

- **Centrarea hărții în funcție de locația curentă** - dacă utilizatorul permite aplicației să acceseze locația curentă a dispozitivului de pe care este rulată, acesta poate vedea harta într-un mod optim, în funcție de coordonatele sale geografice
- **Plasarea unui marker** - utilizatorul poate plasa un maraj oriunde pe hartă, iar apoi prin selectarea acestuia poate deschide coordonatele geografice asociate marcajului folosind aplicația **Google Maps** (cu scopul de vizualizare sau de navigație)
- **Zoom in/Zoom out** - utilizatorul poate mări sau micșora elementele de pe hartă, fie prin utilizarea butoanelor cu simbolurile „+” și „-“ sau folosind gesturi tactile
- **Funcționalitatea de busolă** - indică direcția nordului pe hartă și apare după ce utilizatorul rotește harta, plasând două degete pe hartă și rotind în sensul acelor de ceasornic (sau invers); apăsarea acestui buton resetează orientarea hărții, prin alinierea nordului hărții cu nordul geografic, cu scopul de a ajuta utilizatorul să se orienteze ușor în spațiu

În plus, utilizatorul poate căuta o locație pe hartă, în funcție de nume sau adresă, similar cu câmpul de căutare descris anterior în cadrul subsecțiunii legate de **Adăugarea unei postări**. După selectarea locației, poziția hărții va fi centrată în funcție de aceasta, iar, în plus, va fi adăugat un maraj, pentru a putea vizualiza mai multe detalii asociate acelei destinații folosind aplicația **Google Maps**.

Fragmentul Descoperă

Ecranul **Descoperă** funcționează drept panou publicitar, fiind o pagină în cadrul căreia apar diferite recomandări ale unor locații, în funcție de postările cu care utilizatorii interacționează, postările pe care le fac persoanele pe care utilizatorul îi urmărește sau aleatoriu (prezentat în secțiunea 4.2). Aceste recomandări sunt adăugate manual în baza de date de către administrator, au o structură bine definită, similară cu cea a unei postări și conține următoarele atrbute:

- **id** - cheia generată automat de **Firebase Cloud Firestore** (baza de date)
- **logo** - link-ul către logo-ul locației (dacă locația are un logo)

- **nume** - numele locației
- **adresă** - adresa la care se află locația
- **rating** - evaluarea locației pe platforma **Google Maps** (dacă există o evaluare)
- **tip** - tipul locației (conform **Google Places API**, similar cu tipul locației, în cazul postărilor)
- **categorie** - categoria din care face parte acea locație, pe baza tipului
- **descriere** - un text de promovare, ce are scopul de a atrage utilizatorii să viziteze acea locație fizică (de exemplu afaceri locale, restaurante, baruri, obiective turistice)
- **geoLocatie** - coordonatele geografice la care se află locația din această recomandare

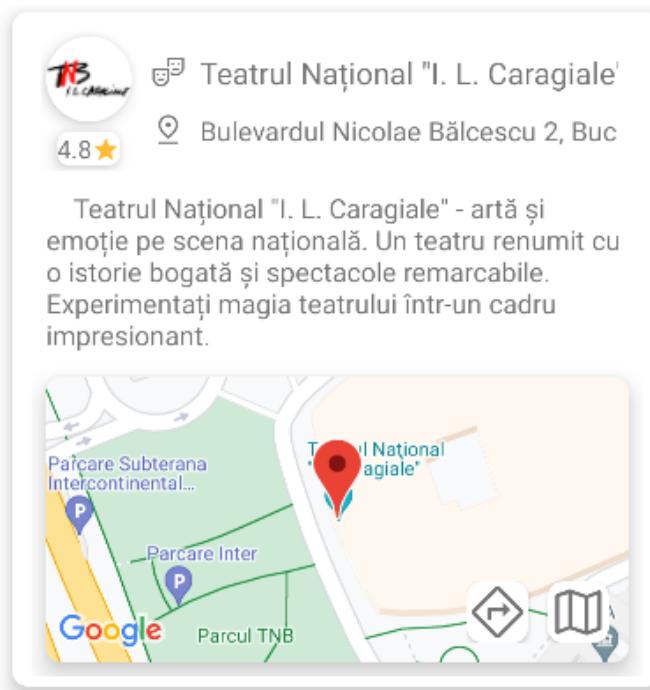


Figura 3.20: Exemplu de recomandare

Un exemplu de recomandare este ilustrat în figura 3.20, din care se poate observa că locația este *Teatrul Național „I.L. Caragiale”*, ce are o evaluare, în medie, de 4.8 stele pe platforma **Google Maps**. Textul de promovare descrie concret specificul acestei locații, îndemnând cititorul să o viziteze, iar harta prezintă câteva elemente din împrejurimi și un marcaj pus la coordonatele geografice ale locației. De asemenea, am adăugat două butoane personalizate, ce au rolul de a deschide locația în aplicația **Google Maps**, pentru navigație (butonul din stânga) sau vizualizare (butonul din dreapta).

Pagina **Descoperă**, descrisă anterior, conține aceste recomandări într-o componentă de tip **RecyclerView**, similar cu postările din pagina **Postări**, cu scopul de a

îmbunătăți performanța aplicației, prin afișarea elementelor acesteia într-un mod eficient. De asemenea, pentru a optimiza numărul de apeluri făcute către baza de date ce conține aceste recomandări, salvez elementele local, folosind **SharedPreferences** (similar cu salvarea datelor utilizatorului curent), urmând să actualizez zilnic lista de recomandări.

Astfel, baza de date nu este apelată de fiecare dată când utilizatorul deschide activitatea principală (ce conține acest fragment), ci este apelată zilnic, întrucât unele locații pot pune în descriere elemente precum ofertele zilnice, ce necesită o astfel de actualizare.

Fragmentul Cont



Figura 3.21: Exemplu de profil al unui utilizator

Acest fragment reprezintă o pagină în cadrul căreia utilizatorul curent al platformei poate să vadă detaliile personale și să editeze o parte dintre acestea. Conform figurii 3.21, în cadrul căreia este ilustrat un exemplu pentru această pagină, se poate observa faptul că utilizatorul poate să își editeze alias-ul (primul câmp de text din pagină), bio-ul (al doilea câmp de text) și parola (ultimul câmp de text). De asemenea, utilizatorul poate să își schimbe poza de profil cu o imagine din spațiul local, prin apăsarea pictogramei ce conține poza de profil curentă.

În cazul de față, utilizatorul și-a creat contul folosind email și parolă, dar în cazul în care acesta a creat contul utilizând contul **Google**, ultimul câmp de text nu este vizibil, deoarece acesta nu își poate modifica parola în mod direct. Toate câmpurile ce pot fi editate sunt marcate, la dreapta cu o pictogramă ce are forma unui creion, pentru o distincție mai ușoară față de câmpurile ce nu pot fi editate (spre exemplu adresa de email, ce este doar afișată pe ecran).

În plus, față de aceste elemente, utilizatorul poate vedea, similar cu ecranul prezentat în subsecțiunea **Verificarea profilului unui utilizator**, propriile postări, lista cu persoanele pe care le urmărește și lista cu persoanele care urmăresc acel utilizator pe platformă. Spre deosebire de postările obișnuite de pe platformă, în momentul vizualizării propriilor postări, utilizatorul nu le poate aprecia (acest lucru ar însemna aprecierea propriilor postări), iar alias-ul și logo-ul nu mai sunt prezente (deoarece acestea sunt deja prezente în pagina contului).

Utilizatorul are și opțiunea de a ieși din cont, selectând butonul aflat la finalul paginii. Prin apăsarea acestui buton, utilizatorul este scos din cont, detaliile salvate local despre acesta sunt șterse (pentru nu a păstra date sensibile despre acesta), iar utilizatorul este condus către **Activitatea de Autentificare** (și se omite ecranul de start), ajungând în cele din urmă la ecranul de logare, unde acesta se poate reconecta sau conecta cu un alt cont pe platformă.

Capitolul 4

Fundamentele algoritmice

Acest capitol are obiectivul de a prezenta algoritmii implementați în cadrul aplicației și modul în care aceștia contribuie la îmbunătățirea experienței utilizatorilor și la eficiența platformei.

4.1 Analizarea recenziilor

Pentru a analiza recenziile utilizatorilor acordate punctelor de interes, am decis să examinez textul (descrierea experienței) pe care aceștia o publică în cadrul unei postări, cu scopul de a evalua dacă reacția lor a fost una pozitivă, neutră sau negativă. În acest sens, am ales să creez un model de Inteligență Artificială care să interpreteze textul, iar apoi să ofere un răspuns al evaluării acestuia, sub forma unui scor, pe baza căruia pot încadra recenziile în categoriile menționate anterior.

4.1.1 Alegerea și împărțirea setului de date

Primul pas a fost căutarea și alegerea unui set de date relevant, cu elemente asemănătoare cu potențialele recenzii ale utilizatorilor platformei mele, cu care să pot antrena modelul pe care va urma să-l creez. Astfel, am ajuns la setul de date **ro_sent** [4], prezent pe platforma **Hugging Face** [11]. Acest set de date este conceput pentru analiza sentimentelor textelor din limba română și conține recenziile pentru produse și filme, evaluări în cadrul cărora există cuvinte ce se pot regăsi și în cazul opinioilor utilizatorilor.

Inițial, datele vin împărțite în aproximativ 62% date de antrenare și 38% date de test, dar am redistribuit aceste date în 80% pentru antrenare și 20% pentru test, pentru o capacitate de generalizare îmbunătățită a modelului și pentru reducerea supraantrenării, prin varietatea mai mare a exemplelor, elemente ce contribuie la îmbunătățirea performanței modelului ales.

4.1.2 Selectarea și testarea algoritmilor

După preluarea și împărțirea datelor, am testat mai mulți algoritmi din librăria **scikit-learn** [24], iar în funcție de performanța inițială a acestora a urmat să selectez modelele cu care să continui procesul de testare.

În procesul de implementare, pentru a ajunge la rezultate mai bune, am utilizat clasa **TfidfVectorizer** [25], prezentă în aceeași librărie. Aceasta este folosită pentru a transforma un set de texte într-o reprezentare numerică vectorială pe baza tehnicii **TF-IDF** [23, Scoring, term weighting, and the vector space model, pp. 100–123] (Term Frequency-Inverse Document Frequency) și este compusă din:

- **Term Frequency(TF)**: măsoară cât de des apare un cuvânt într-un document, fiind calculat drept raportul dintre numărul de apariții ale unui cuvânt într-un document și numărul total de cuvinte din acel document.
- **Inverse Document Frequency(IDF)**: măsoară importanța unui cuvânt în cadrul unei colecții de documente, fiind invers proporțional cu frecvența cuvântului în acea colecție; de exemplu, cuvintele care apar rar în colecție au o valoare **IDF** mai mare.

Pentru a obține scorul **TF-IDF** pentru un cuvânt într-un document, se înmulțește valoarea **TF** cu valoarea **IDF** a cuvântului, în cadrul colecției. Acest scor reprezintă importanța relativă a cuvântului în documentul respectiv în comparație cu celelalte documente din colecție, iar aceste scoruri sunt utilizate pentru a reprezenta vectorial documentele și termenii dintr-o colecție de texte. Aceasta este o metodă comună pentru extragerea caracteristicilor din texte și este utilizată într-o varietate de aplicații de prelucrare a limbajului natural și învățare automată.

TfidfVectorizer preia un set de texte și aplică transformarea **TF-IDF** asupra acestora, rezultând un vector de caracteristici pentru fiecare text, ce poate fi utilizat ulterior pentru a antrena modele de învățare automată sau pentru a extrage informații relevante din texte. În implementarea sa, TfidfVectorizer utilizează calculul termenilor de frecvență și *inverse document frequency*, aplicând și alte tehnici precum *tokenizarea*, eliminarea *stop words* și eliminarea caracterelor non-alfanumerice.

Am ales inițial 5 algoritmi de învățare automată supervizată pentru clasificarea datelor, iar pentru a eficientiza procesul total de testare al tuturor modelelor, am ales clasificatori care nu necesită o perioadă lungă pentru a fi antrenați și am inclus 3 metrii pentru măsurarea performanței:

- **Cross Validation Score**: este o metrică de măsurare a performanței în timpul etapei de antrenare și implică împărțirea setului de date în bucăți mai mici numite **fold-uri** și antrenarea modelului pe o parte din fold-uri, în timp ce se evaluatează performanța pe fold-urile rămase. Acest proces este repetat de mai multe ori și se

calculează scorul mediu al performanței pentru a obține o estimare mai robustă a performanței modelului. Această metrică este utilă pentru a estima performanța generală a modelului și pentru a evita problemele de overfitting sau underfitting.

- **Train Accuracy:** este o metrică ce măsoară precizia modelului pe datele de antrenare, comparând predicțiile făcute de model pe datele de antrenare cu etichetele reale cunoscute și calculează proporția de predicții corecte. Aceasta poate fi utilă pentru a evalua cât de bine învață modelul și se adaptează la datele de antrenare, deși un scor foarte mare poate indica o suprapunere excesivă (overfitting) a modelului pe datele de antrenare și o capacitate scăzută de generalizare pentru date noi.
- **Test Accuracy:** este o metrică ce măsoară precizia modelului pe datele de testare, iar, similar cu *Train Accuracy*, compara predicțiile făcute de model și calculează proporția de predicții corecte. Aceasta este o măsură a performanței pe datele noi și ilustrează capacitatea modelului de a generaliza cunoștințele pe date din afara setului de antrenare.

În comparația performanței clasificatoarelor testate, am decis să utilizez metrica **Cross Validation Score** pentru acuratețea la validare, deoarece pot observa cel mai bine capacitatea de generalizare a modelului și pot identifica problemele de *underfitting* și *overfitting*. De asemenea, în cadrul acestui proces, am împărțit setul de date de antrenare în 5 fold-uri, ceea ce înseamnă că modelul va fi antrenat și testat de tot atâtea ori.

Multinomial Naive Bayes

Multinomial Naive Bayes este un algoritm utilizat în special pentru clasificarea textului și este eficient în gestionarea datelor, reprezentate prin frecvență de termeni. Modelul este o variantă a algoritmului **Naive Bayes** [23, Text classification and Naive Bayes, pp. 234–265], ce se bazează pe **Teorema Bayes**, care folosește asumția naive de independentă între caracteristici pentru a determina probabilitatea unei anumite clase.

Acest model calculează probabilitățile a priori pentru fiecare clasă în funcție de frecvența lor în datele de antrenare, iar apoi estimatează probabilitățile condiționate pentru fiecare caracteristică (cuvânt) dată de o anumită clasă. Atunci când trebuie să facă o predicție pentru date noi, acest model calculează probabilitățile a posteriori (ce reflectă cât de probabil este ca un exemplu să aparțină acelei clase) pentru fiecare clasă, pe baza probabilităților a priori și a probabilităților condiționate: cu cât un cuvânt apare mai des într-o categorie de recenzii, cu atât va avea o probabilitate mai mare pentru acea clasă.

Conform rezultatelor obținute în figura 4.1, acest model, fără optimizări și preprocesări anterioare a produs o acuratețe de aproximativ 82.33%.

```

pipe = make_pipeline(TfidfVectorizer(stop_words=None) , MultinomialNB())
print("Medie cross validation " + str(cross_val_score(pipe, text_train.sentence , label_train, cv=5).mean()))
pipe.fit(text_train.sentence , label_train)
print("Acuratete train " + str(pipe.score(text_train.sentence , label_train)))
label_pred = pipe.predict(text_test.sentence)
print("Acuratete test " + str(accuracy_score(label_pred, label_test)))

```

```

Medie cross validation 0.8233722092871909
Acuratete train 0.8996372430471584
Acuratete test 0.8243523316062176

```

Figura 4.1: Modelul Multinomial Naive Bayes

Logistic Regression [28] este un algoritm utilizat pentru probleme de clasificare binară sau multi-clasă, ce estimează probabilitățile de apartenență a unui exemplu la o anumită clasă, funcționând astfel:

- **Asociază probabilități**, prin folosirea unei funcții logistice (sigmoide) pentru a mări valorile caracteristicilor unui exemplu la probabilități între 0 și 1.
- **Calculează ponderile**: pe durata antrenării, Logistic Regression găsește coeficienții (ponderile) asociate fiecărei caracteristici, ce reprezintă contribuția relativă a caracteristicilor la predicție.
- **Calculează probabilitățile**: Pentru a prezice probabilitatea de apartenență a unui exemplu la o clasă, se combină liniar coeficienții și valorile caracteristicilor exemplului, iar rezultatul este trecut prin funcția logistică.
- **Aplică pragul**: În cazul clasificării binare (situația de față), se aplică un prag (de regulă 0.5) la probabilitatea estimată, iar dacă probabilitatea este mai mare decât pragul, exemplul este clasificat în clasa pozitivă, în caz contrar fiind clasificat în cea negativă.
- **Ajustează parametrii** în timpul antrenării, pentru a estima probabilitățile cât mai apropiate de etichetele de clasă observate în datele de antrenare.

```

pipe = make_pipeline(TfidfVectorizer(stop_words=None) , LogisticRegression())
print("Medie cross validation " + str(cross_val_score(pipe, text_train.sentence , label_train, cv=5).mean()))
pipe.fit(text_train.sentence , label_train)
print("Acuratete train " + str(pipe.score(text_train.sentence , label_train)))
label_pred = pipe.predict(text_test.sentence)
print("Acuratete test " + str(accuracy_score(label_pred, label_test)))

```

```

Medie cross validation 0.8732511321726488
Acuratete train 0.9252893418552427
Acuratete test 0.8725388601036269

```

Figura 4.2: Modelul Logistic Regression

În conformitate cu figura 4.2, rezultatele obținute în urma utilizării acestui model au fost mai bune, comparativ cu cel precedent, având o acuratețe de aproximativ 87.32%.

KNeighborsClassifier este o clasă din librăria *scikit-learn*, ce implementează algoritmul **K-Nearest Neighbors** [15] pentru clasificare. Această clasă este utilizată pentru a crea un model de clasificare bazat pe principiul **K-NN**, unde un punct nou este clasificat pe baza claselor majoritare ale vecinilor săi cei mai apropiati, funcționând astfel:

1. Înainte de a face predicții, algoritmul K-NN primește date de antrenament sub formă de caracteristici (atribute) și etichetele (clasele) corespunzătoare, ce vor fi folosite pentru a învăța modelul.
2. Pentru a clasifica un punct de date nou, KNeighborsClassifier calculează distanța (similaritatea) față de celelalte puncte de antrenament. Distanța poate fi măsurată în diverse moduri, precum distanța euclidiană sau distanța Manhattan.
3. Clasificatorul selectează cei mai apropiati K vecini ai punctului nou în funcție de distanță calculată anterior, unde K reprezintă un parametru specificat de utilizator și determină câți vecini vor fi luați în considerare în cadrul clasificării.
4. După ce sunt selectați cei mai apropiati K vecini, KNeighborsClassifier determină clasa majoritară dintre vecini. Acest lucru se poate realiza prin votare majoritară, adică clasa care apare cel mai des printre vecini va fi considerată clasa prezisă pentru punctul de date nou.
5. După ce este determinată clasa majoritară, KNeighborsClassifier atribuie această clasă punctului nou și îl consideră clasificat în consecință.

În schimb, este un algoritm de învățare lenăș (lazy learning), prin lipsa etapei de antrenare explicită, deoarece modelul stochează datele de antrenare în memorie și nu construiește un model intern.

```
pipe = make_pipeline(TfidfVectorizer(stop_words=None) , KNeighborsClassifier())
print("Medie cross validation " + str(cross_val_score(pipe, text_train.sentence , label_train, cv=5).mean()))
pipe.fit(text_train.sentence , label_train)
print("Acuratețe train " + str(pipe.score(text_train.sentence , label_train)))
label_pred = pipe.predict(text_test.sentence)
print("Acuratețe test " + str(accuracy_score(label_pred, label_test)))
```

Medie cross validation 0.6050268819911171
Acuratețe train 0.6240283295906028
Acuratețe test 0.6043177892918825

Figura 4.3: Modelul K-Nearest Neighbors

Observând figura 4.3, rezultatele obținute pentru acest model au fost mai slabe, cu o acuratețe de aproximativ 60.5%.

Random Forest Classifier este un algoritm de învățare supervizată, utilizat în cazul problemelor de clasificare și de regresie, bazat pe conceptul de imbinare a mai multor arbori de decizie într-o colecție denumită **Random Forest** [2], ce funcționează astfel:

- După primirea datelor de antrenare (sub formă de attribute și etichete/clase corespunzătoare), Random Forest Classifier creează un ansamblu de arbori de decizie, fiecare fiind construit pe baza unui eșantion de date de antrenament, selectat aleatoriu. Aceasta implică selecția aleatoare a unui subset de date și a caracteristicilor asociate acestora.
- Fiecare arbore de decizie este construit într-un mod similar cu unul standard. Se alege o caracteristică a fiecărui nod al arborelui pentru a separa datele în funcție de acea caracteristică, astfel încât să se obțină partiții cât mai omogene în ceea ce privește clasele de etichete.
- Pentru a face o predicție pentru un punct de date nou, Random Forest Classifier colectează predicțiile individuale ale fiecărui arbore de decizie din ansamblu, unde fiecare arbore votează pentru clasa sa atribuită punctului de date nou, iar clasa finală este determinată prin votare majoritară.

```

pipe = make_pipeline(TfidfVectorizer(stop_words=None) , RandomForestClassifier())
print("Medie cross validation " + str(cross_val_score(pipe, text_train.sentence , label_train, cv=5).mean()))
pipe.fit(text_train.sentence , label_train)
print("Acuratete train " + str(pipe.score(text_train.sentence , label_train)))
label_pred = pipe.predict(text_test.sentence)
print("Acuratete test " + str(accuracy_score(label_pred, label_test)))

```

Medie cross validation 0.8252721857542603
Acuratete train 0.9991794783209536
Acuratete test 0.8227979274611399

Figura 4.4: Modelul Random Forest Classifier

Pe baza codului din figura 4.4, acuratetea produsă, folosind acest model, a fost de aproximativ 82.52%.

XGBClassifier, sau Extreme Gradient Boosting Classifier este o clasă din biblioteca *XGBoost*, care implementează un algoritm de învățare automată numit *Gradient Boosting* [5] pentru clasificare, ce funcționează astfel:

- Setul de antrenare va fi folosit pentru a construi un ansamblu de modele slabe.
- Modelul XGBClassifier este inițializat cu un estimator de bază, care poate fi, spre exemplu, un arbore de decizie simplu, ce va fi primul model slab din ansamblu.
- Apoi, clasificatorul antrenează iterativ o serie de modele slabe, adăugându-le la ansamblu, unde fiecare este construit pentru a corecta erorile făcute de modelele anterioare.
- XGBClassifier utilizează ponderi și calculează o medie ponderată a predicțiilor acestor modele slabe, cu scopul de a obține o predicție finală mai precisă, iar în cele din urmă, să returneze clasa prezisă pentru punctul de date nou.

Acest clasificator are mai multe aplicații, cum ar fi analizarea textului, recunoaștere vocală/facială sau finanțe și utilizează tehnici precum regularizarea, pruning-ul și selecția caracteristicilor.

```
pipe = make_pipeline(TfidfVectorizer(stop_words=None), XGBClassifier())
print("Medie cross validation " + str(cross_val_score(pipe, text_train.sentence, label_train, cv=5).mean()))
pipe.fit(text_train.sentence, label_train)
print("Acuratețe train " + str(pipe.score(text_train.sentence, label_train)))
label_pred = pipe.predict(text_test.sentence)
print("Acuratețe test " + str(accuracy_score(label_pred, label_test)))
```

Medie cross validation 0.8504058871113009
Acuratețe train 0.9385472447745725
Acuratețe test 0.8526770293609672

Figura 4.5: Modelul XGBClassifier

Deductibil din figura 4.5, acuratețea rezultată, folosind acest model, a fost de aproximativ 85.04%.

4.1.3 Balansarea setului de date

Am observat faptul că setul de date nu este unul balansat, cu același număr de exemple pentru ambele clase (recenzii pozitive și negative). Acest lucru poate afecta performanța modelului, prin predicții dezechilibrate, performanță scăzută pentru clasa minoritară și *overfitting* în cazul clasei majoritare. Pentru a observa dacă obțin o performanță mai bună prin echilibrarea datelor, am încercat două abordări.

```
# undersampling

b_text_train, b_text_test, b_label_train, b_label_test = train_test_split(
    total_df.drop('label', axis=1), total_df.label, test_size=0.2, random_state=42)

total_df_balansat = pd.concat([b_text_train, b_label_train], axis=1)

negativ_sample = total_df_balansat[total_df_balansat['label'] == 0]
pozitiv_sample = total_df_balansat[total_df_balansat['label'] == 1]

# voi alege aleator din recenziile pozitive, cu scopul de a balansa numărul de sample-uri
# random_state setat pentru a obține același rezultat la fiecare rulare
pozitiv_sample = pozitiv_sample.sample(n=len(negativ_sample), random_state=42)

total_df_balansat = pd.concat([pozitiv_sample, negativ_sample], axis=0, ignore_index=True)

# pentru a păstra efectul aleator, voi amesteca datele concatenate
total_df_balansat = total_df_balansat.sample(frac=1, random_state=42)

b_text_train = total_df_balansat['sentence']
b_label_train = total_df_balansat['label']

# acum distribuția este echilibrată, 11675 recenzi din ambele categorii
total_df_balansat['label'].value_counts()
```

0	9326
1	9326
Name: label, dtype: int64	

Figura 4.6: Undersampling

În primă instanță, am încercat **random undersampling-ul**, ce presupune reducerea numărului de exemple din clasa majoritară (în cazul meu clasa cu recenziile pozitive), prin selecția de date în mod aleatoriu. Conform codului ilustrat în figura 4.6, prima dată am împărțit datele în același mod, 80% pentru antrenare și restul de 20% pentru testare, iar apoi am concatenat textul și etichetele datelor de antrenare în același cadru de date. După aceea, am echilibrat numărul de date din cele două clase, prin selectarea aleatoare de texte din clasa majoritară și am retestat pentru aceleași modele.

De asemenea, am testat și **random oversampling-ul**. Acesta implică creșterea numărului de exemple din clasa minoritară (recenziile negative), prin adăugarea de exemple sintetice, generate aleatoriu, din acea clasă. În acest caz, la retestare, am creat un pipeline, în care am inclus și **SMOTE** (Synthetic Minority Over-sampling Technique), ce funcționează prin generarea de exemple sintetice pentru clasa minoritară prin interpolarea între vecinii săi, în spațiul caracteristic.

După cum se poate observa în tabela 4.1, prezintă mai jos, singurele modele pentru care oversampling-ul a ajutat la creșterea acurateții la validare au fost **Multinomial Naive Bayes** și **Random Forest Classifier**, dar rezultatele au fost mai slabe față de alte modele pe care nu am aplicat acest procedeu.

Model	Set de date inițial	Undersampling	Oversampling
Multinomial Naive Bayes	0.823372	0.796590	0.845266
Logistic Regression	0.873251	0.868057	0.862454
K-Nearest Neighbors	0.605026	0.506219	0.458844
Random Forest Classifier	0.825574	0.796857	0.826826
XGBClassifier	0.850405	0.842268	0.849153

Tabela 4.1: Rezultatele, la validare, obținute în urma balansării setului de date

În continuare am decis să selectez 2 dintre aceste modele, pe baza rezultatelor obținute în faza inițială. Astfel, am ales **Logistic Regression** și **XGBClassifier**, deoarece au produs cele mai bune rezultate (acuratețe la validare de peste 85%). Nu voi aplica undersampling sau oversampling, întrucât pentru aceste modele, rezultatele obținute au fost mai slabe.

4.1.4 Alte procesări ale datelor

În această subsecțiune voi testa diferite procesări pe cele două modele prezentate anterior, cu scopul de a obține un model cu acuratețe cât mai mari.

Clasa **TfidfVectorizer** aplică deja câteva procesări, iar printre cele mai importante se regăsesc transformarea cuvintelor la litere mici, tokenizarea (prin transformarea textului în n-grame) sau normalizarea L2, ce facilitează comparația și calculul similarității între texte. De asemenea, această clasă pune la dispoziție și eliminarea de *stop words*,

acele cuvinte comune care adesea nu aduc o semnificație în plus, în contextul prelucrării de limbaj natural. Această funcționalitate este valabilă pentru cuvinte din lexiconul englez, motiv pentru care am setat parametrul **stop_words** al acestei clase cu valoarea **False**, pentru a nu elimina cuvinte din limba română ce puteau fi interpretate accidental ca fiind din limba engleză.

Întrucât modelul este orientat către recenzii în limba română, pentru a realiza acest lucru trebuie făcută o funcție separată, ce conține *stop words* din limba română. Setul de date nu conține diacritice, deci eliminarea acestora nu este necesară în contextul dat. O altă preprocesare ce poate fi utilă este eliminarea semnelor de punctuație și a altor caractere speciale, deoarece, de obicei, acestea nu aduc o semnificație importantă în analiza textului. Alte două metode de preprocesare sunt lematizarea, prin aducerea cuvintelor la forma lor de bază, cu scopul de a trata cuvintele similare ca fiind identice și stematizarea, ce presupune trunchierea cuvintelor, prin eliminarea prefixelor și sufixelor, cu o menire similară lematizării.

Preprocesări	Logistic Regression	XGBClassifier
Fără preprocesări	0.873251	0.846346
char_spec	0.872560	0.846130
stop_words	0.871912	0.846260
stemming	0.874244	0.847814
lemming	0.873423	0.844575
char_spec + stop_words	0.871912	0.846260
char_spec + stemming	0.875237	0.849585
char_spec + lemming	0.873639	0.846735
stop_words + stemming	0.874287	0.849239
stop_words + lemming	0.873078	0.845007
stemming + lemming	0.872257	0.847469
char_spec + stop_words + stemming	0.874330	0.849239
char_spec + stop_words + lemming	0.872603	0.846519
stop_words + stemming + lemming	0.872257	0.848246
char_spec + stemming + lemming	0.872732	0.847382
char_spec + stop_words + stemming + lemming	0.872171	0.848635

Tabela 4.2: Rezultatele, la validare, obținute în urma preprocesărilor

În tabela 4.2, se pot observa rezultatele pentru cele 4 tipuri de preprocesări menționate anterior, unde:

- **char_spec** reprezintă eliminarea caracterelor speciale (non-alfanumerice și normalizare spații)
- **stop_words** reprezintă eliminarea de *stop words*
- **stemming** și **lemming** sunt operațiile de *stematizare*, respectiv *lematizare*

Pentru ambele modele testate, am setat parametrul `random_state` cu valoarea 42, pentru ca modelul să aibă aceeași secvență de numere pseudo-aleatoare în fiecare rulare, cu scopul de a obține rezultate comparabile, care să nu depindă de acest factor.

Pentru ambele modele, am obținut rezultate mai bune în cazul folosirii combinației dintre `char_spec` și `stemming`, cu o îmbunătățire de aproximativ 0.2% pentru **Logistic Regression** și aproximativ 0.32% în cazul **XGBClassifier**. În cele din urmă, am ales să optimizez primul model, aplicând aceste preprocesări, întrucât am obținut cea mai mare acuratețe la validare (87.52%).

4.1.5 Optimizarea hiperparametrilor

Pentru acest model, am decis să modific o serie de hiperparametri, cu scopul de a îmbunătăți acuratețea, la validare pe 5 fold-uri. Am modificat parametrii și am făcut validarea folosind clasa **GridSearchCV**, destinată căutării acestor combinații optime.

Prima dată am ales să modific parametrul `ngram_range` din cadrul clasei **TfidfVectorizer**. Aceasta definește numărul de n-grame care vor fi considerate în procesul de tokenizare, fiind reprezentat de tuplul (`min_n`, `max_n`), unde `min_n` și `max_n` reprezintă valoarea minimă, respectiv maximă pentru un n-gram, iar la testare am încercat mai multe combinații între unigrame și bigrame.

Pentru modelul **Linear Regression** am testat diferite valori pentru parametrii:

- **C**; controlează forța de regularizare, prin inversarea acesteia (o valoare mai mică înseamnă o forță de regularizare mai mare)
- **penalty**; reprezintă tipul de regularizare aplicată ($L1$, $L2$ sau `elasticnet`, ce combină ambele regularizări)
- **max_iter**; specifică numărul maxim de iterații pe care le va efectua algoritmul pentru a converge la soluție

```
pipe = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words=None)),
    ('logreg', LogisticRegression(random_state = 42))
])
parametrii = {'tfidf__ngram_range' : [(1, 1), (1, 2), (2, 2)],
              'logreg__C' : [1, 2, 3],
              'logreg__penalty' : ['l1', 'l2', 'elasticnet'],
              'logreg__max_iter' : [100, 150, 200]
}
grid_search_pipeline = GridSearchCV(pipe, parametrii, cv=5, scoring='accuracy')
grid_search_pipeline.fit(text_train.sentence , label_train)
print(grid_search_pipeline.best_params_, grid_search_pipeline.best_score_)

{'logreg__C': 3, 'logreg__max_iter': 100, 'logreg__penalty': 'l2', 'tfidf__ngram_range': (1, 2)} 0.8956210288179569
```

Figura 4.7: Rezultate inițiale pentru setarea hiperparametrilor

După cum se poate observa în figura 4.7, singurii hiperparametri, față de cei impliciti, care au dus la rezultate mai bune (acuratețe la validare de aproximativ 89.56%) au fost *ngram_range* și *C*. În continuare, am ales că setez valoarea pentru primul parametru cu (1, 2), pentru ca modelul să aleagă unigrame și bigrame, și să testez pentru valori mai mari ale celui de-al doilea parametru, cu obiectivul de a obține rezultate și mai bune. De asemenea, am dublat numărul maxim de iterații (implicit are valoarea 100, iar eu am setat 200), pentru a evita eventualele cazuri în care modelul nu converge către soluție.

```

pipe = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words=None, ngram_range=(1,2))),
    ('logreg', LogisticRegression(random_state = 42, max_iter=200))
])
parametrii = {'logreg__C' : list(range(3,20))}
grid_search_pipeline = GridSearchCV(pipe, parametrii, cv=5, scoring='accuracy')
grid_search_pipeline.fit(text_train.sentence , label_train)
print(grid_search_pipeline.best_params_, grid_search_pipeline.best_score_)

{'logreg__C': 18} 0.9008895429129143

```

Figura 4.8: Rezultate finale pentru setarea hiperparametrilor

Conform figurii 4.8, am ajuns la un model cu o acuratețe de 90.08% **la validare**, setând valoarea parametrului *C* cu valoarea 18. Preprocesările și setarea hiperparametrilor au îmbunătățit performanța modelului inițial (cu o acuratețe de 87.32%), înregistrând o creștere de 2.76%, ce face în final ca modelul să ofere rezultate cât mai relevante. Pe baza figurii 4.9, acest model are o acuratețe de 89.72% **la testare** și am afișat matricea de confuzie pentru setul de testare. În această matrice sunt comparate predicțiile făcute de model cu etichetele reale ale datelor, fiind afișate câte date au fost clasificate corect și câte au fost clasificate greșit:

- **True Negative:** numărul de cazuri în care algoritmul a prezis corect o recenzie negativă; este reprezentat de linia 1, coloana 1
- **True Positive:** numărul de cazuri în care algoritmul a prezis corect o recenzie pozitivă; este reprezentat de linia 2, coloana 2
- **False Negative:** numărul de cazuri în care algoritmul a prezis greșit o recenzie negativă, ca fiind pozitivă; este reprezentat de linia 2, coloana 1
- **False Positive:** numărul de cazuri în care algoritmul a prezis greșit o recenzie pozitivă, ca fiind negativă;; este reprezentat de linia 1, coloana 2

```
# model final

pipe = make_pipeline(TfidfVectorizer(stop_words=None, ngram_range=(1,2)), LogisticRegression(random_state = 42, max_iter=200, C=18))
pipe.fit(text_train.sentence, label_train)
print("Acuratete train " + str(pipe.score(text_train.sentence, label_train)))
label_pred = pipe.predict(text_test.sentence)
print("Acuratete test " + str(accuracy_score(label_pred, label_test)))

matrice_confuzie(label_pred, label_test)
```

Acuratete train 0.9990067369148384
Acuratete test 0.8972366148531952

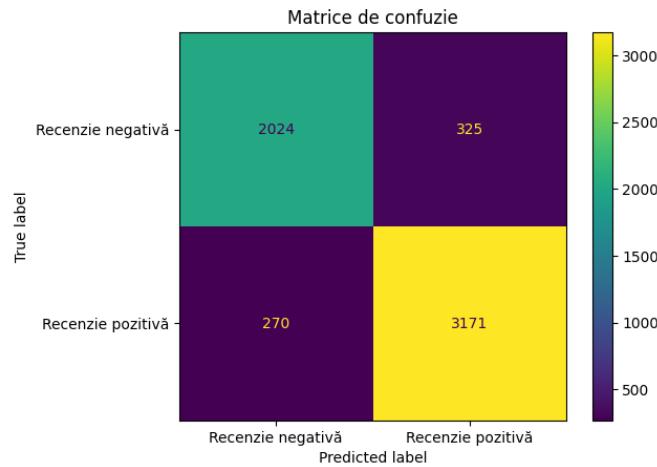


Figura 4.9: Modelul final

4.1.6 Exportarea modelului final

Pentru exportarea acestui model final am folosit funcția `convert_sklearn` din librăria `skl2onnx`, cu scopul de a converti modelul în formatul **ONNX** [17] (Open Neural Network Exchange). Am ales acest format deoarece este independent de limbaj și platformă, iar astfel pot folosi modelul în cadrul aplicației mele. Conform codului din figura 4.10, am salvat acest model într-un fișier pe care l-am descărcat local, urmând să îl import și să îl integrez în evaluarea recenziilor utilizatorilor.

```
# exportarea modelului final

initial_type = [('recenzie', StringTensorType([None, 1]))]

converted_model = convert_sklearn(pipe, initial_types=initial_type)
with open("model_ml.onnx", "wb") as f:
    f.write(converted_model.SerializeToString())
```

Figura 4.10: Exportarea modelului final

În mediul de dezvoltare al aplicației, am salvat modelul creat anterior ca resursă a proiectului. De asemenea, înainte de a evalua o recenzie, am avut nevoie de o modalitate de a determina dacă textul este scris în limba română, întrucât modelul de Inteligență Artificială este construit pentru recenziile în această limbă. Pentru a determina acest

lucru, am folosit librăria **Lingua** [19], ce pune la dispoziție soluții pentru analizarea și manipularea textului în diferite limbi, printre care se regăsește și limba română.

```
private val detectorLimba = LanguageDetectorBuilder
    .fromLanguages(Language.ENGLISH, Language.SPANISH, Language.RUSSIAN, Language.GERMAN, Language.FRENCH,
        Language.JAPANESE, Language.PORTUGUESE, Language.TURKISH, Language.ITALIAN, Language.ROMANIAN)
    .build()

// funcție care detectează dacă textul este scris în Română
new*
private fun detecteazaLimbaRomana(input: String, detectorLimba: LanguageDetector): Boolean {
    val limbaDetectata = detectorLimba.computeLanguageConfidenceValues(text = input).firstKey()

    return limbaDetectata.toString() == "ROMANIAN"
}

val descriereInRomana = detecteazaLimbaRomana(descriereRef.text.toString(),
    mainActivityContext.getDetectorLimba())
```

Figura 4.11: Detectarea limbii în care este scrisă o recenzie

Pe baza codului din cadrul figurii 4.11, am inițializat detectorul cu primele 9 cele mai folosite limbi [27], conform **W3Techs**, în cadrul primelor 10 milioane de website-uri, la data de 9 Iunie 2023, la care am adăugat și limba română. Nu am folosit toate limbile posibile, deoarece timpul de așteptare și resursele necesare cresc pe măsură ce sunt adăugate elemente în plus, fiind cauzele pentru care am făcut acest compromis, cu scopul de a avea un timp de așteptare rezonabil în aplicație. După aceea, am creat o funcție în care textul dat este analizat, prin stabilirea probabilităților ca acesta să fie redactat în fiecare din limbile incluse mai devreme în detector. Apoi, selectez limba care are şansele cele mai mari (rezultatele sunt sortate descrescător, după probabilitate) și verific dacă este limba română. În funcție de rezultatul funcției, voi cunoaște răspunsul la întrebarea pusă anterior și dacă pot aplica modelul pe text.

```
private fun realizeazaPredictie(input : String, ortSession: OrtSession,
                                ortEnvironment: OrtEnvironment): Pair<Int, Double> {
    val numeInput = ortSession.inputNames.first()
    val date = listOf(input).toTypedArray()

    val inputTensor = OnnxTensor.createTensor(ortEnvironment, date, longArrayOf(date.size.toLong(), 1))
    val outputTensors = ortSession.run(mapOf(numeInput to inputTensor), ortSession.outputNames)

    val label = (outputTensors[0].value as LongArray).first().toInt()

    @Suppress("unchecked_cast")
    val probabilitati = (outputTensors[1].value as List<OnnxMap>)[0].value as Map<Int, Double>

    val scorRecenzie = probabilitati[probabilitati.keys.last()]!!.toDouble()

    return Pair(label, scorRecenzie)
}
```

Figura 4.12: Realizarea unei predicții în Kotlin, folosind modelul

Am aplicat preprocesările echivalente pe textul pentru care vreau să fac predicția, am încărcat modelul, iar apoi, conform cu figura 4.12, în cadrul funcției în care realizez predicția, am creat un tensor de tip **OnnxTensor**. Tensorul are rolul de a face predicția pe text, ce constă în *label*, clasa de recenzii din care face parte (0 - negative sau 1 - pozitive) și probabilitatea predicției, ce în practică va deveni un scor în intervalul [0, 1], care va indica ulterior tipul de recenzie ([0, 0.3] - „negativ”, [0.3, 0.7] - „neutru” și [0.7, 1] - „pozitiv”) și constituie metrica folosită în evaluarea recenziilor scrise în limba română.

4.1.7 Testarea modelului în aplicație

În această subsecțiune voi testa acest model pe un exemplu concret, sub forma unei recenzi, ilustrată în figura 4.13, preluată de pe platforma **Google Maps**. După cum se poate observa, recenzia este evaluată cu o stea, dintr-un maxim de 5, deci putem concluziona că această recenzie este una negativă.

★ ★ ★ ★ acum 9 luni
Aluatul prost frământat și necopt era aproape topit în balta lăsată de grămada de legume, neîngrijit ornata peste pizza aproape fără nici un gust ... Am cerut furculița și cuțit ca să pot culege pizza din cutia de la Carrefour Orhideea, unde angajații stăteau pe Instagram și se uitau la mine de parca îi sechestrăm din priviri. Cu alte cuvinte ... o pizza mai proastă n-am servit niciodată!

Figura 4.13: Recenzia aleasă pentru testare

Am preluat textul din această recenzie și am testat pe modelul de Inteligentă Artificială construit anterior, în ambele medii de dezvoltare (Kaggle, cu Python3 - mediul în care a fost implementat modelul și Android, cu Kotlin - mediul în care este construită aplicația), cu scopul de a obține un rezultat corect, iar acel rezultat să coincidă. În acest mod testează dacă modelul furnizează răspunsuri relevante pentru acest context al recenziilor și, de asemenea, testează dacă integritatea modelului a fost păstrată în procesul de conversie și de exportare.

```

val ortEnvironment = OrtEnvironment.getEnvironment()
val ortSession = createORTSession(ortEnvironment)

var textPreprocesat = descriereRef.text.toString()
textPreprocesat = eliminaDiacritice(textPreprocesat)
textPreprocesat = eliminaCaractereSpeciale(textPreprocesat)
textPreprocesat = stematizare(textPreprocesat)

val output = realizeazaPredictie(textPreprocesat, ortSession, ortEnvironment)

Log.d(mainActivityContext.getTag(), msg: "Output: ${output.first}, Scor: ${output.second}")

2023-05-26 14:40:39.807 28497-28497 test D Label: 0, Scor: 0.1247628927230835

```

Figura 4.14: Testarea modelului în Kotlin

În conformitate cu figurile 4.14 și 4.15, unde am exemplificat testarea modelului în **Kotlin**, respectiv **Python**, în ambele cazuri am obținut același rezultat pentru predicție, dar probabilitățile variază ușor, fiind o diferență mică, estimativ de 0.3%, deși modelul testat în Kotlin derivă din modelul implementat în Python, am aplicat preprocesări echivalente și am testat pentru același text (în Python explicit, iar în Kotlin prin testarea la adăugarea unei postări, unde textul se află în variabila *descriereRef*, vizibilă în figura 4.15). Aceste diferențe mici pot apărea din cauza versiunilor diferite ale unor instrumente folosite de către Python și ONNX, unde logica din spatele predicției variază ușor.

```

exemplu = ["Aluatul prost frământat și necopt era aproape topit în balta"

prop = ""
for litera in exemplu[0]:
    match litera.lower():
        case "ă" | "â":
            prop += "a"
        case "î":
            prop += "i"
        case "ş":
            prop += "s"
        case "Ń":
            prop += "t"
        case _:
            prop += litera
exemplu[0] = prop

# aplic aceleași preprocesări pe text
exemplu = elimina_char_speciale(exemplu)
exemplu = [stematizare(exemplu[0])]

# fac o predicție și arăt și scorul pe o scară de la 0 la 1 a recenziei
predictie = pipe.predict(exemplu)[0]
probabilitate = pipe.predict_proba(exemplu)[0][1]

print("Label:", predictie, ", Scor:", probabilitate)

```

Label: 0 , Scor: 0.1277132349728606

Figura 4.15: Testarea modelului în Python

În cadrul bazei de date, pentru fiecare postare va fi salvat scorul recenziei, alături de tipul de recenzie, determinat pe baza scorului, conform imaginilor din figura 4.16. În funcție de acestea, ceilalți utilizatori pot vedea sentimentele transmise în acea recenzie, în cadrul aplicației.

scorRecenzie: 0.1247628927230835 tipRecenzie: "negativ"	scorRecenzie: 0.6654117107391357 tipRecenzie: "neutră"	scorRecenzie: 0.9894194602966309 tipRecenzie: "pozitiv"
(a) Recenzie negativă	(b) Recenzie neutră	(c) Recenzie pozitivă

Figura 4.16: Reprezentarea scorului și a tipului de recenzie în baza de date

4.2 Recomandări personalizate

În conformitate cu ceea ce am precizat în cadrul subsecțiunii 3.4.2, unde a fost prezentată activitatea principală din contextul acestei aplicații, în fragmentul cu numele **Descoperă** sunt prezente recomandări ale unor locații, din mai multe categorii și subcategorii, menționate anterior.

Scopul meu este să ofer recomandări adaptate în mod individual pe acea pagină, în funcție de preferințele fiecărui utilizator. Pentru a analiza ce preferințe au utilizatorii de pe platformă, mă voi raporta la ultimele postări cu care aceștia au interacționat și la interesele individuale ale utilizatorilor pe care aceștia îi urmăresc. De asemenea, vreau ca o parte din recomandări să fie aleatorii, pentru a încuraja diversitatea locațiilor, a susține explorarea locurilor noi și pentru evitarea rutinei, prin aducerea unor recomandări diferite față de cele la care utilizatorul se așteaptă.

Astfel, am ales să implementez un sistem de recomandări cu ponderi, folosind cele trei elemente de recomandare descrise anterior, astfel:

- **Recomandări pe baza preferințelor de postări ale utilizatorului:** aceasta mi se pare cea mai importantă categorie, deoarece analizează ponderile specifice pentru fiecare categorie de postări, calculate pe baza unui sistem de scor (pe care urmează să îl dezvolt ulterior), motiv pentru acesteia îi este atribuită o pondere de 50% în lista de recomandări.
- **Recomandări pe baza intereselor individuale ale utilizatorilor pe care aceștia îi urmăresc:** în această categorie se încadrează postările făcute de către persoanele pe care un utilizator le urmărește, iar pentru acestea se calculează procentual câte aparțin fiecărei categorii și în funcție de aceste rezultate sunt determinate interesele generale ale tuturor utilizatorilor din acea listă; pentru această categorie se va aloca ponderea de 30%.
- **Recomandări aleatorii:** categoria aceasta are rolul de a aduce o diversitate în lista de recomandări, prin selectarea aleatoare a unor recomandări din multime și adăugarea acestora la lista personalizată. Pentru această categorie este alocată ponderea rămasă, de 20%.

Recomandările reprezintă o listă finită de elemente, iar pentru a crește relevanța în timp real, dar și pentru a eficientiza procesul de selecție a postărilor, am decis să aplic aceste ponderi pe incremente ce conțin 10-20 de elemente din listă, pe care apoi să le adaug la lista finală personalizată. În acest fel, timpul de încărcare al paginii va fi vizibil mai scurt și se va putea observa mai ușor utilitatea acestui sistem de ponderi.

De asemenea, a trebuit să tratez cazurile în care lista de recomandări rămâne fără elemente pe care să le recomande (spre exemplu trebuie să aleagă o locație din categoria

„food and drink”, dar toate locațiile de acel tip au fost deja recomandate), utilizatorul nu a interacționat cu nicio postare sau utilizatorul nu urmărește pe niciun alt utilizator. În situațiile de față a fost necesar să reatribui ponderile menționate anterior, astfel:

- Dacă nu se poate alege niciun element pe baza preferințelor de postări (nu mai există recomandări de acel tip sau nu există astfel de preferințe), se dublează ponderile de la celelalte două categorii (și devin 60%, respectiv 40%).
- În situația în care nu există elemente pe baza postărilor persoanelor urmărite (nu mai sunt recomandări de acel tip sau utilizatorul nu urmărește niciun alt utilizator), categoriei menționate anterior îi va fi alocată ponderea de 70%, iar pentru postările aleatorii va fi atribuit procentul de 30%.
- În situația în care nu pot fi alese recomandări din niciuna dintre cele două categorii precizate anterior, toate recomandările selectate în continuare vor fi aleatorii.

4.2.1 Recomandări pe baza preferințelor de postări

Așa cum am menționat anterior, în această categorie sunt încadrate postările cu care utilizatorul a interacționat, ce vor fi analizate pe baza unui sistem de scor.

Postările sunt analizate în funcție de categoria din care fac parte, iar pe baza acestor categorii voi analiza modul în care utilizatorul interacționează cu postările, astfel:

- Dacă utilizatorul **apreciază** o postare, voi adăuga **2 unități** la sistemul de scor și voi reține local într-un dicționar acea postare, folosind-mă de cheia acesteia din baza de date, iar pentru fiecare cheie voi asocia scorul acelei postări și data la care s-a interacționat prima dată cu acea postare (pentru a putea urmări ordinea cronologică a interacțiunilor). În situația opusă, în care utilizatorul **elimină aprecierea** unei postări, voi ajusta scorul prin eliminarea celor 2 unități adăugate anterior.
- De asemenea, la acest sistem de scor contribuie și **comentariile** pe care utilizatorul le adaugă acelei postări, ce au o pondere de **o unitate**, în momentul în care utilizatorul publică acel comentariu.

Acest sistem de scor este actualizat constant, în momentul în care utilizatorul interacționează cu o astfel de postare. De asemenea, pentru a păstra relevanța rezultatelor, am decis să analizez activitatea ultimelor 10 postări, dată fiind dimensiunea momentană mică a platformei. Această valoare poate fi modificată ușor în cod și adaptată în funcție de dimensiunea aplicației, ce poate suferi schimbări în timp.

În cadrul figurii 4.17 am exemplificat, sub forma a două tabele, cum se face conversia sistemului de scor (tabela din stânga) într-un sistem de ponderi (tabela din dreapta).

Aceste ponderi vor reprezenta probabilitățile cu care recomandările din aceste categorii vor fi alese, în lista de recomandări personalizate.

Categorie	Scor (ultimele 10 postări)
food and drink	6
retail	3
services	5
entertainment	7
outdoor	4
other	0

Categorie	Ponderi
food and drink	0.24
retail	0.12
services	0.2
entertainment	0.28
outdoor	0.16

Figura 4.17: Exemplificarea transformării sistemului de scor în ponderi

În cele din urmă, folosind ponderea alocată acestei categorii, înmulțită cu 10 (numărul minim de recomandări selectat per total pentru o iterație) va rezulta numărul de recomandări ce va fi ales aleatoriu, din această categorie.

La fiecare pas, se va face o **selectie aleatorie ponderată** cu utilizarea probabilității cumulate, proces ce utilizează **funcții de distribuție cumulativă** [1] (CDF), iar pentru realizarea unei predicții de acest tip preiau probabilitățile asociate fiecărui element din setul de date (calculate la pasul anterior, din figura 4.17).

Conform implementării prezentate în figura 4.18, în primă instanță generez un număr aleatoriu cu zecimale, între 0 și 1, denumit *valoareAleatoare*. În cazul generării valorii pentru această variabilă, distribuția este uniformă (fiecare valoare din acest interval are o probabilitate egală de a fi selectată), pentru a garanta că selecția se face în mod imparțial.

```
private fun preiaCategoriaAleator(categoriiProb: HashMap<String, Double>): String {
    val valoareAleatoare = Random.nextDouble()

    var probCumulativa = 0.0
    for ((categorie, prob) in categoriiProb) {
        probCumulativa += prob
        if (valoareAleatoare <= probCumulativa) {
            return categorie
        }
    }

    return categoriiProb.keys.last()
}
```

Figura 4.18: Alegerea aleatorie a categoriei

După aceea, inițializez probabilitatea cumulativă, ce pornește de la valoarea 0, iar apoi o calculez pe măsură ce iterez prin categoriile de postări ce pot fi alese. Înținând

cont că aceste categorii și ponderile lor sunt stocate într-un element de tip dicționar (*HashMap*), ordinea în care acestea sunt parcurse este aleatorie, deci intervalele date de probabilități cumulative vor dифeri de la o predicție la alta. În procesul de calculare a probabilității cumulative la pasul respectiv, verific dacă valoarea generată anterior se află între două probabilități cumulative, caz în care returnez categoria asociată acesteia.

Spre exemplu, pentru cazul ilustrat în figura 4.17, dacă pentru variabila *valoareAleatoare* este generată aleatoriu valoarea 0.5, iar în urma iterării dicționarului cu ponderi rezultă următoarele probabilități cumulative:

1. **food and drink** -> $0 + 0.24 = 0.24$
2. **outdoor** -> $0.24 + 0.16 = 0.4$
3. **retail** -> $0.4 + 0.12 = 0.52$
4. **entertainment** -> $0.52 + 0.28 = 0.8$
5. **services** -> $0.8 + 0.2 = 1$

Categoria returnată este **retail**, deoarece 0.5 se află în intervalul $(0.4, 0.52]$. Astfel, știu că trebuie aleasă o recomandare din această categorie. Pentru a face acest lucru, selectez aleatoriu (cu distribuție uniformă) un element din lista de recomandări rămase (care nu au fost deja adăugate), pe care îl adaug la lista personalizată de recomandări și îl elimin din lista de recomandări rămase.

După selectarea și adăugarea unei recomandări pe baza acestor criterii, verific dacă mai există astfel de recomandări pentru categoriile de interes, iar apoi actualizez dicționarul de ponderi pentru categoriile de interes (dacă o categorie a fost eliminată), reajustând ponderile.

4.2.2 Recomandări pe baza intereselor persoanelor urmărite

În cadrul acestui sistem de recomandări este folosită lista ce conține persoanele pe care utilizatorul curent le urmărește pe platformă (listă salvată în baza de date, ce conține cheile utilizatorilor urmăriți), pentru care aplic următoarele operații:

- Filtrez postările preluate, păstrându-le doar pe acele făcute de către persoane pe care utilizatorul le urmărește.
- Folosind lista nouă rezultată la pasul anterior, grupez elementele din acea listă, în funcție de categoriile din care fac parte, proces din care rezultă un element de tip dicționar, ce va conține postările, grupate după categorii.
- Aplic o funcție de transformare pentru fiecare cheie a dicționarului obținut la pasul anterior, prin preluarea numărului de postări făcute.

Pașii descriși anterior sunt exemplificați în figura 4.19, pentru utilizatorii urmăriți pe platformă de către utilizatorul cu alias-ul **usertestgoogle**. Toți cei 4 utilizatori au pe platformă câte 5 postări din categorii diferite, iar pentru fiecare dintre aceștia am inclus câte postări sunt din fiecare categorie, sub formă de tabele. Elementele din aceste tabele (reținute în cod sub formă de dicționare) sunt combinate în cadrul unui dicționar, ce va conține numărul de postări din fiecare categorie. Similar cu exemplul din figura 4.17, fac o conversie a valorilor în ponderi, pe care să le folosesc în procesul de selecție aleatorie a unei categorii din acel dicționar.

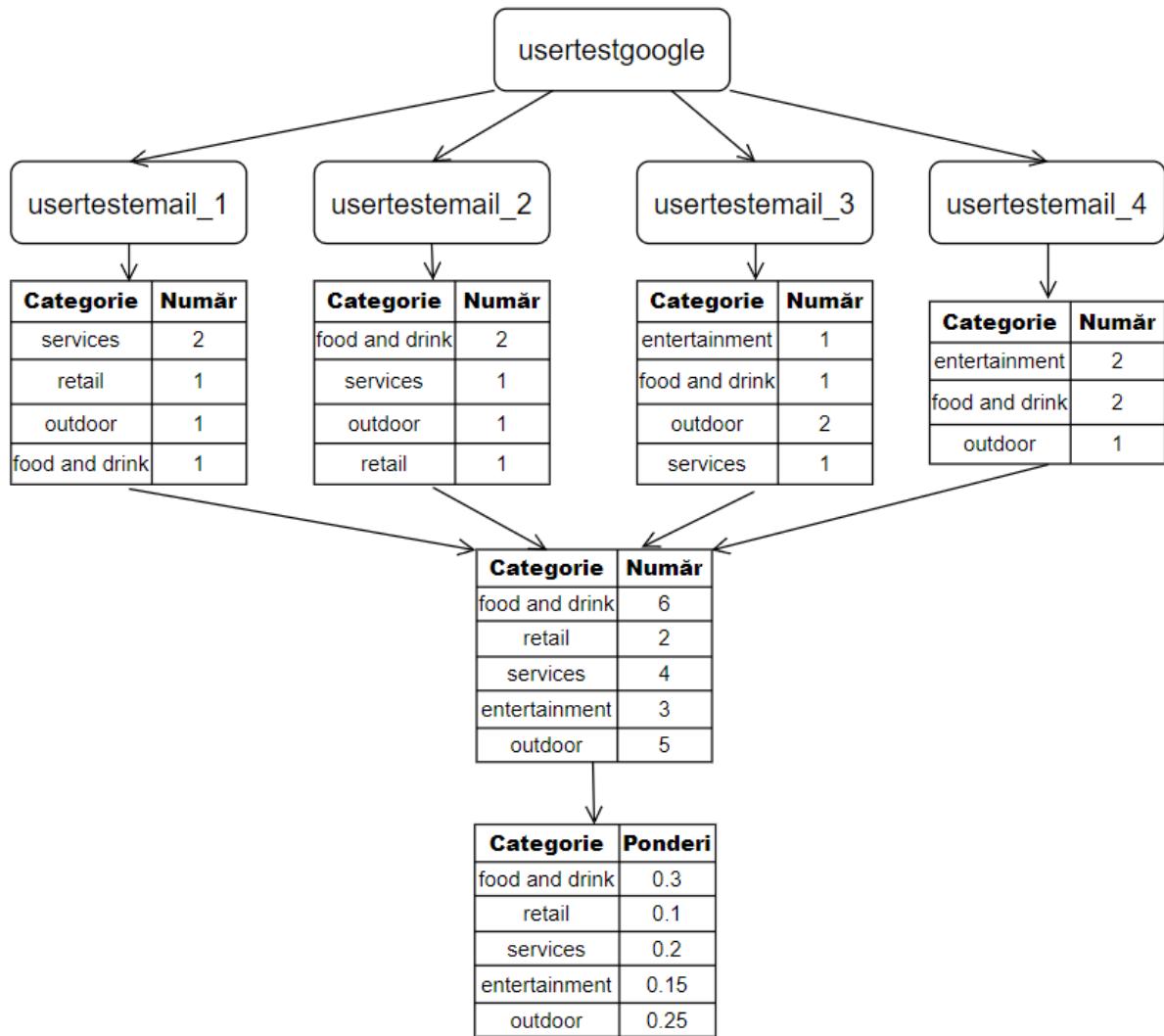


Figura 4.19: Preluarea preferințelor utilizatorilor urmăriți pe platformă

Lista cu persoanele urmărite de către un utilizator este actualizată constant, în momentul în care acesta decide să adauge/elimine persoane din lista sa de persoane pe care le urmărește pe platformă.

Folosind dicționarul de ponderi, rezultat în exemplul din figura 4.19, se va alege aleatoriu (ponderat) o categorie de recomandare, iar apoi va fi selectat aleatoriu (cu distribuție uniformă) o recomandare din acea multime. Pașii pentru returnarea unei

acelei categorii sunt identici cu cei de la categoria prezentată anterior, în secțiunea 4.2.1, iar reajustarea ponderilor pentru dicționarul de categorii se face în aceeași manieră, prin verificarea existenței recomandărilor în dicționar și recalcularea conformă a ponderilor asociate acestora.

4.2.3 Recomandări aleatorii

Acest sistem de selecție aleatorie este unul prin distribuție uniformă și presupune că fiecare recomandare are șanse egale de a fi selectată. Lista din care se face această selecție este dată de recomandările care nu au fost deja făcute și sunt incluse toate categoriile din care acestea pot face parte. Procesul de selecție este unul simplu: se generează aleatoriu (uniform) un indice cuprins între valorile 0 și lungimea listei rămase de recomandări, iar în funcție de acesta, se va alege recomandarea asociată indexului, din listă. Recomandarea aleasă va fi eliminată din lista cu elemente rămase, dar va fi adăugată în lista personalizată, ce conține recomandările filtrate, în funcție de toate cele trei criterii prezentate în cadrul acestui subcapitol.

De asemenea, a fost necesar să verific dacă o recomandare aleasă aleatoriu face parte din categoriile anterioare (4.2.1 și/sau 4.2.2), pentru a putea verifica dacă în lista de recomandări neselectate există elemente din acea categorie. Astfel, am reactualizat ponderile pentru celelalte două metode de selecție, în același mod.

4.3 Postări personalizate

În acord cu prezentarea structurii aplicației principale (subsecțiunea 3.4.2), în cadrul ecranului cu numele **Postări** sunt incluse postările celorlalți utilizatori de pe platformă.

Având drept punct de plecare ideile de filtrare a recomandărilor prezentate în subsecțiunea 4.2, am decis să reutilizez și să extind aceste concepte, în vederea creării unui sistem personalizat de recomandare a postărilor pentru utilizatorul curent al aplicației, pe care îl voi structura astfel:

- **Postări pe baza categoriilor de interes ale utilizatorului:** similar cu *recomandările pe baza preferințelor de postări ale utilizatorului*, categorie pe care am descris-o în subsecțiunea 4.2, aceasta analizează categoriile de postări cu care utilizatorul a interacționat pe platformă, sub forma sistemului de scor, pe baza căruia se vor afișa postările pentru locații din categorii relevante; această categorie rămâne cea mai importantă și astfel îi voi aloca o pondere de 50%, în sistemul de selecție.
- **Postări ale utilizatorilor apropiati:** în acest criteriu se regăsesc postările făcute de către utilizatorii care se află într-un grup apropiat utilizatorului curent, iar pentru acest criteriu voi atribui ponderea de 30%.

- **Postări aleatorii:** această categorie este similară cu *recomandările aleatorii* și vine cu scopul de a aduce o varietate în rândul postărilor recomandate, prin selectarea aleatorie a acestora, în momentul construirii listei personalizate; această categorie va avea ponderea disponibilă, de 20%.

Pentru a păstra relevanța postărilor rezultate din acest proces de filtrare, voi lucra prin aceeași modalitate ca și în cazul recomandărilor personalizate, aplicând aceste ponderi pe incremente ce au 10-20 de elemente din lista de postări ce conține elemente care nu au fost selectate. Acest lucru asigură un timp de încărcare mai scurt, deoarece elementele noi selectate la o iterație vor fi adăugate la lista ce conține postările, în schimbul atribuirii tuturor postărilor după finalizarea procesului de filtrare, aplicat pentru toate acestea.

Asemănător cu recomandările personalizate, a fost necesară tratarea situațiilor în care unele dintre aceste criterii enumerate anterior nu au putut fi aplicate, prin reatribuirea ponderilor pentru fiecare categorie, în următoarele moduri:

- În situația în care nu se poate alege nicio postare pe baza categoriilor de interes ale utilizatorului (nu mai există postări de acel tip sau utilizatorul nu are categorii de interes), se dublează ponderile celorlalte două criterii.
- Atunci când utilizatorul nu are persoane în sfera de interes a acestuia sau nu mai există postări făcute de către acestia, care să fie recomandate utilizatorului curent, atribui ponderea de 70% primei categorii, respectiv 30% postărilor aleatorii.
- În cazul în care nu se poate alege nicio postare, folosind cele două criterii de selecție prezentate anterior, toate postările alese în continuare vor fi aleatorii.

4.3.1 Postări pe baza categoriilor de interes

Sistemul de scor pe baza căruia se analizează categoriile de interes ale utilizatorului curent este cel descris în cadrul subsecțiunii 4.2.1. Am decis să continui cu acest sistem de verificare a activității utilizatorului deoarece, în urma testărilor făcute pentru recomandările personalizate, am observat că rezultatele obținute au fost relevante postărilor cu care am interacționat, dovedind astfel utilitatea acestui sistem de scor în cadrul aplicației.

În situația de față, selecția postărilor se face în aceeași manieră, prin raportare la ponderile rezultate din clasificarea categoriilor de postări. Pașii pentru **selecția aleatorie ponderată** a unei categorii locații incluse într-o postare sunt identici cu cei descriși în interiorul secțiunii 4.2.1, prin utilizarea probabilității cumulative.

După alegerea unei postări prin această metodă, voi verifica dacă pot să mai fie alese alte postări cu locații din aceste categorii de interes și voi actualiza corespunzător coeficienții asociați acestor categorii.

4.3.2 Postări ale utilizatorilor aproiați

Pentru a determina modalitatea în care aleg utilizatorii aproiați ai utilizatorului curent al platformei de socializare, am consultat mai multe articole din acest domeniu, dar „Finding and evaluating community structure in networks” [16] s-a dovedit a fi cel mai important în contextul dat.

Acest articol dezvoltă metodele și tehniciile utilizate pentru identificarea și evaluarea structurilor comunităților în rețele complexe, ce sunt reprezentări grafice ale interacțiunilor și relațiilor între elemente, cum ar fi utilizatorii în rețelele sociale, proteinele în rețelele biochimice sau paginile web în rețelele de internet. În cadrul capitolului III este introdusă **parcurgerea în lățime** (BFS) drept soluție pentru găsirea celor mai scurte drumuri de la un punct dat dintr-o rețea către celelalte, iar în procesul de dezvoltare am optat pentru această abordare din mai multe motive, toate evidențiind relevanța și eficacitatea acesteia în cadrul aplicației.

Algoritmul **Breadth-First Search** (BFS), prezentat în cartea *Introduction to Algorithms* [3] este un algoritm de căutare în grafuri care se bazează pe explorarea în lățime a nivelurilor adiacente ale unui nod sursă. În cazul de față, nodurile sunt reprezentate de către utilizatorii aplicației, iar direcția muchiei indică faptul că un utilizator urmărește un alt utilizator pe platformă, exprimând astfel interacțiunea dintre aceștia. Această reprezentare sub formă de graf a utilizatorilor și a relațiilor dintre aceștia permite algoritmului de căutare în lățime să identifice în mod eficient utilizatorii aflați la o anumită distanță față de un utilizator dat, ca un grup de utilizatori aproiați.

De asemenea, prin explorarea în lățime a grafului pot obține mai întâi informații despre cei mai aproiați utilizatori (cei pe care utilizatorul curent îi urmărește pe platformă), aflați pe cel mai apropiat nivel, iar apoi să extind căutarea pe niveluri mai îndepărtate. În acest fel, grupurile de utilizatori sunt construite într-un mod structurat și coerent, iar acest algoritm poate fi adaptat în funcție de dimensiunea platformei, prin modificarea nivelului maxim până la care se face căutarea.

Complexitatea acestui algoritm poate fi exprimată în funcție de numărul de noduri (utilizatori), notat cu n , și de numărul de muchii (relații de prietenie), notat cu m , din rețeaua socială:

- **Complexitatea spațială**, determinată de cantitatea de memorie necesară pentru a putea stoca într-o coadă nodurile din graf ce urmează să fie parcurse și pentru a reține într-un set multimea nodurilor ce au fost deja parcurse, este $O(n + n) = O(2n)$, ce se reduce la $O(n)$, în cel mai rău caz (spre exemplu adâncimea grafului este 1, iar toate nodurile trebuie reținute în coadă și în set).
- **Complexitatea temporală** indică timpul necesar pentru executarea algoritmului BFS, ce crește proporțional cu numărul de noduri și muchii din rețea. În cel mai

rău caz, complexitatea temporală este $O(m + n)$, situație ce poate să apară în cazul în care rețeaua are o adâncime mare.

În comparație, un alt algoritm de parcursere a grafurilor, **DFS**, explorează în adâncime unul dintre vecinii unui nod sursă înainte de a se întoarce și a explora și ceilalți vecini, dar acest lucru poate duce la parcurserea unor drumuri lungi înainte de a găsi grupurile de utilizatori apropiati. Astfel, prin folosirea algoritmului BFS pentru găsirea grupurilor de utilizatori apropiati, există avantajul unei complexități liniare, dar și a unei explorări structurate a rețelei sociale.

În cadrul aplicației am decis să implementez acest algoritm pentru o adâncime maximă de **2 niveluri**, urmând pașii următori, exemplificăți și în figura 4.20, pentru un exemplu concret, din rețeaua socială.

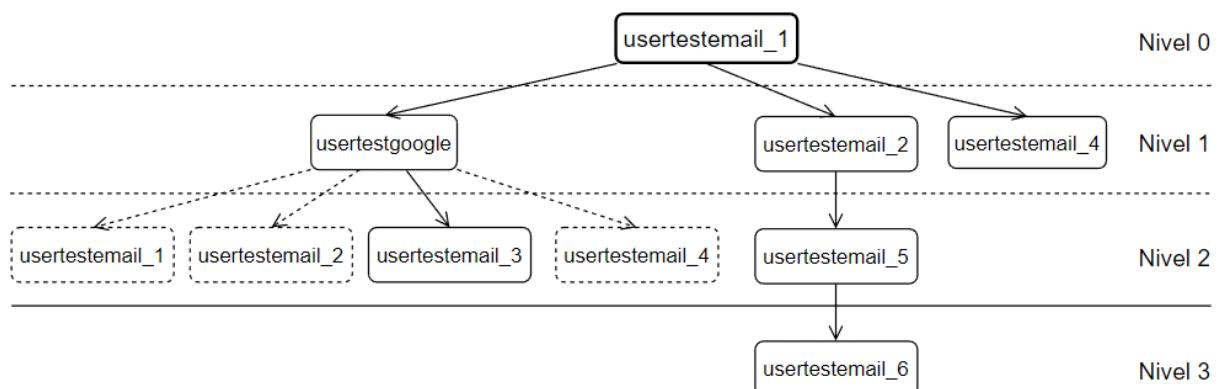


Figura 4.20: Utilizarea algoritmului BFS în identificarea utilizatorilor din sfera de interes

1. Pornesc de la un nod de start (în cazul de față utilizatorul **usertestemail_1**) și creez o coadă goală, ce va reține nodurile ce urmează să fie parcurse și un set ce va memora nodurile vizitate.
2. Adaug nodul de start în coadă și îl marchez ca fiind vizitat în setul nodurilor vizitate, iar acest nod se află pe **nivelul 0**.
3. Cât timp coada nu este goală, extrag primul nod din coadă, verific toate nodurile vecine nevizitate ale acestuia, iar apoi adaug nodurile vecine în coadă și le marchez ca vizitate în setul nodurilor vizitate. În exemplul dat, prima dată extrag nodul de start (utilizatorul curent al platformei), iar apoi adaug toți vecinii acestuia (utilizatorii pe care acesta îi urmărește), situați pe nivelul următor al arborelui, **nivelul 1**. Apoi, în momentul parcurgerii vecinilor utilizatorului **usertestgoogle**, utilizatorii ilustrați sub forma chenarelor punctate nu vor fi adăugați în coadă, deoarece aceștia au fost deja vizitați. La final, nodurile aflate la o adâncime mai mare față de **nivelul 2** nu vor fi luate în considerare, întrucât am stabilit anterior utilizarea acestei parcurgeri pentru maxim 2 niveluri.

În urma execuției acestui algoritm va rezulta un set, ce conține toate nodurile vizitate, iar pentru exemplul din figura 4.20 acest set va fi format din toți utilizatorii reprezentați în chenarele nepunctate (aflate pe nivelurile 0, 1 și 2). Din acest set voi elmina nodul de start (utilizatorul pentru care aplic acest algoritm), deoarece în pagina de postări nu vreau să includ și postările făcute de către el însuși. După aceea, utilizând acest set rezultat, voi prelua postările făcute de către utilizatorii din acel grup.

Selectia unei postări din această mulțime se va face în mod aleatoriu, cu distribuție uniformă (fiecare postare are aceeași probabilitate de a fi selectată), iar după fiecare pas voi verifica dacă au fost selectate toate postările care aparțin acestui criteriu de selecție.

4.3.3 Postări aleatorii

Elementele din această categorie sunt alese în aceeași manieră, prin distribuție uniformă, iar mulțimea din care se face selecția este reprezentată de poate postările care nu au fost deja alese. Procesul de selecție constă în generarea unui indice aleatoriu din cadrul listei de postări neselectate, iar pe baza acestuia voi adăuga elementul corespunzător la lista personalizată de postări, afișată în pagină.

După acest proces, voi verifica dacă postarea selectată aleatoriu aparține din mulțimea postărilor din categoriile de interes (secțiunea 4.3.1), respectiv postările utilizatorilor apropiati (secțiunea 4.3.2), cu scopul de a reajusta ponderile folosite la aceste criterii de selecție.

Capitolul 5

Concluzii

Aplicația **TownTrekker**, prezentată în capituloarele precedente, utilizează o varietate de unelte și îmbină cunoștințe din mai multe domenii, precum algoritmi pe grafuri, probabilități sau elemente de Inteligență Artificială. Prin această abordare complexă, platformă reușește să ofere o experiență personalizată utilizatorilor săi și pune la dispoziție o multitudine de funcționalități, specifice aplicațiilor de acest tip.

Dezvoltări ulterioare

În această secțiune voi include câteva optimizări și modificări ce pot fi aduse aplicației, pe viitor, cu scopul de îmbunătățiri experiența utilizatorului.

Preferințele utilizatorilor

În cadrul acestei aplicații, preferințele utilizatorului curent (ce constau în postările cu care acesta a interacționat și pe baza cărora se face sistemul de scor, descris anterior) sunt salvate local, pe dispozitivul de pe care acesta a utilizat platforma.

În momentul în care utilizatorul dorește să acceseze platforma de pe alt dispozitiv sau reinstalează aplicația, această listă de preferințe nu va fi disponibilă, iar în acest sens, pentru a păstra lista de preferințe, este necesară o soluție de stocare a acestora, eficientă din punct de vedere al spațiului ocupat și al numărului de modificări efectuate.

Analizarea recenziilor

Recenziile făcute de către utilizatorii aplicației sunt analizate folosind modelul de Inteligență Artificială explicitat în secțiunea 4.1, dacă textul a fost redactat în limba română. Astfel, recenziile făcute în alte limbi nu vor fi evaluate de către model, limitând aplicabilitatea modelului implementat.

Pentru a rezolva această problemă, o abordare posibilă o constituie traducerea textului respectiv, folosind aceeași librărie (*Lingua*) care verifică limba în care este redactat textul, în implementarea actuală. Totuși, utilizarea acestei metode (sau a altor abordări) poate avea unele dezavantaje, ce implică o acuratețe limitată, interpretare contextuală dificilă sau pierderea înțelesurilor textului dat în procesul de traducere și astfel este nevoie de o soluție care să acopere cât mai multe cerințe.

Categoriile de locații

La momentul de față, în aplicație sunt incluse locații de mai multe tipuri, ce sunt considerate de interes, dar această selecție poate duce la limitarea diversității de locații prezente pe platformă sau la reducerea atracției acestei rețele sociale pentru diferite grupuri de utilizatori.

O soluție pentru rezolvarea acestei deficiențe o constituie analizarea intereselor pe care le au utilizatorii platformei pe viitor, iar pe baza acestora să fie adăugate tipuri și categorii noi de locații, iar în acest fel platforma va răspunde la cerințele utilizatorilor, ce se schimbă în timp.

Recomandarea conținutului

Modalitatea în care conținutul (sub formă de postări sau recomandări de locații) este sugerat utilizatorilor se face raportat la activitatea acestora pe platformă. Spre exemplu, grupul de utilizatori apropiati unui alt utilizator al platformei este determinat folosind algoritmul BFS, pentru o adâncime maximă fixată, iar numărul maxim de postări anterioare cu care acesta a interacționat (pe baza căruia se calculează sistemul de scor al categoriilor de interes) are, de asemenea, o valoare fixată.

În timp, dimensiunea rețelei sociale suferă modificări și este necesar ca aceste variabile să fie ajustate, pentru a furniza date relevante. O modalitate de a cunoaște ce îmbunătățiri trebuie aduse platformei este colectarea reacțiilor de la utilizatorii platformei, pentru o identificare a problemelor și erorilor pe care le are aplicația și pentru a înțelege mai ușor nevoile acestora.

Bibliografie

- [1] Joseph K Blitzstein și Jessica Hwang, „Introduction to probability”, în Crc Press, 2019, cap. Cumulative distribution functions, pp. 120–123.
- [2] Leo Breiman, „Random forests”, în *Machine learning* 45 (2001), pp. 5–32.
- [3] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest și Clifford Stein, „Introduction to algorithms”, în MIT press, 2022, cap. Elementary Graph Algorithms, Breadth-first search, pp. 554–563.
- [4] Stefan Daniel Dumitrescu, Andrei-Marius Avram și Sampo Pyysalo, „The birth of Romanian BERT”, în *arXiv preprint arXiv:2009.08712* (2020).
- [5] Jerome H Friedman, „Greedy function approximation: a gradient boosting machine”, în *Annals of statistics* (2001), pp. 1189–1232.
- [6] Google, *Android Studio Documentation*, 2023, URL: <https://developer.android.com/docs> (accesat în 14.6.2023).
- [7] Google, *Developer documentation for Firebase*, 2023, URL: <https://firebase.google.com/docs> (accesat în 14.6.2023).
- [8] Google Maps Platform, *Google Maps Platform Documentation*, 2023, URL: <https://developers.google.com/maps/documentation> (accesat în 14.6.2023).
- [9] Google Maps Platform, *Place Icons*, 2023, URL: <https://developers.google.com/maps/documentation/places/android-sdk/icons> (accesat în 14.6.2023).
- [10] Google Maps Platform, *Place Types*, 2023, URL: https://developers.google.com/maps/documentation/places/android-sdk/supported_types (accesat în 14.6.2023).
- [11] Hugging Face, *ro_sent*, Hugging Face Datasets, 2023, URL: https://huggingface.co/datasets/ro_sent (accesat în 14.6.2023).
- [12] JetBrains, *Kotlin docs*, 2023, URL: <https://kotlinlang.org/docs/home.html> (accesat în 14.6.2023).
- [13] Kaggle, *How To Use Kaggle*, 2023, URL: <https://www.kaggle.com/docs/> (accesat în 14.6.2023).

- [14] Simon Kemp, *Digital 2023: Global Overview Report*, <https://datareportal.com/reports/digital-2023-global-overview-report>, Ian. 2023, accesat: 14.6.2023.
- [15] Oliver Kramer și Oliver Kramer, „K-nearest neighbors”, în *Dimensionality reduction with unsupervised nearest neighbors* (2013), pp. 13–23.
- [16] Mark EJ Newman și Michelle Girvan, „Finding and evaluating community structure in networks”, în *Physical review E* 69.2 (2004), p. 026113.
- [17] ONNX Contributors, *ONNX documentation*, 2023, URL: <https://onnx.ai/onnx/> (accesat în 14.6.2023).
- [18] Oracle, *Oracle Java SE Documentation*, 2023, URL: <https://docs.oracle.com/en/java/javase/index.html> (accesat în 14.6.2023).
- [19] Peter M. Stahl, *lingua*, GitHub repository, 2023, URL: <https://github.com/pemistahl/lingua> (accesat în 14.6.2023).
- [20] Playground AI, *Playground AI*, 2023, URL: <https://playgroundai.com> (accesat în 14.6.2023).
- [21] Python Software Foundation, *Python 3 Documentation*, 2023, URL: <https://docs.python.org/3/> (accesat în 14.6.2023).
- [22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser și Björn Ommer, „High-Resolution Image Synthesis With Latent Diffusion Models”, în *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Iun. 2022, pp. 10684–10695.
- [23] Hinrich Schutze, Christopher D Manning și Prabhakar Raghavan, *Introduction to information retrieval*, Cambridge University Press, 2008.
- [24] scikit-learn, *scikit-learn: Machine Learning in Python*, 2023, URL: <https://scikit-learn.org/stable/> (accesat în 14.6.2023).
- [25] scikit-learn, *TfidfVectorizer*, 2023, URL: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer (accesat în 14.6.2023).
- [26] Karina Sokolova, Marc Lemercier și Ludovic Garcia, „Android passive MVC: a novel architecture model for the android application development”, în *International Conference on Pervasive Patterns and Applications*, 2013, pp. 7–12.
- [27] W3Techs, *Usage statistics of content languages for websites*, 2023, URL: https://w3techs.com/technologies/overview/content_language (accesat în 14.6.2023).
- [28] Hsiang-Fu Yu, Fang-Lan Huang și Chih-Jen Lin, „Dual coordinate descent methods for logistic regression and maximum entropy models”, în *Machine Learning* 85 (2011), pp. 41–75.