



ACADEMIA DE STUDII ECONOMICE DIN BUCUREȘTI

Facultatea de Cibernetică, Statistică și Informatică Economică

Specializarea: Informatică Economică

Lucrare de Licență

**- Aplicație Web de tip CMS pentru o redacție
de știri cu elemente de social media -**

Cadrul didactic coordonator:

Prof. univ. doctor Bâră Adela

Absolvent: Neacșu David-Andrei

Cuprins

Introducere.....	3
Capitolul 1 Descrierea problemei economice.....	4
1.1. Prezentarea domeniului abordat.....	4
1.2. Prezentarea activității care va fi informatizată.....	5
1.3. Analiza sistemelor existente pe piață.....	6
1.3.1. Intagram	6
1.3.2. Reddit	7
1.3.3. Analiza comparativă a funcționalităților	8
Capitolul 2 Analiza sistemului informatic.....	11
2.1. Specificarea cerințelor sistemului informatic.....	11
2.2. Analiza sistemului existent.....	13
Capitolul 3. Proiectarea sistemului informatic	24
3.1. Proiectarea noului sistem	24
3.2. Proiectarea schemei bazei de date.....	26
Baze de date de tip document.....	26
Definirea schemei bazei de date	28
Colecția User	29
Colecția Article	30
Relații între colecții	30
Capitolul 4. Implementarea sistemului informatic	32
4.1. Tehnologii informatice utilizate.....	32
4.1.1. Tehnologiile de bază ale dezvoltării web (HTML, CSS, Javascript).....	32
4.1.2. Instrumente utilizate.....	32
Visual Studio Code - IDE.....	32
Git - DVCS	33
GitHub.....	34
4.1.3. Frontend – frameworkuri utilizate.....	34
React.js	34
Bootstrap	35
Bootstrap-React.....	35
4.1.4. Backend – mediu de rulare și framework.....	35

Node.js	35
Express.js.....	36
4.1.5. MongoDB	37
MongoDB database – bază de date NoSQL	37
MongoDB Atlas - Database-as-a-Service (DBaaS)	37
MongoDB Compass - interfață grafică oficială	38
4.1.6. Cloudinary – Software-as-a-Service (SaaS).....	39
4.1.7. Utilizarea serviciilor de localizare (Geoapify - Geocoding).....	39
4.1.8. Utilizarea LLM (Large Language Models)	40
LangChain.....	40
Gemini API - modelul Gemini-2.0-flash	41
4.1.9. Render - Platform-as-a-Service (PaaS)	41
4.2. Implementarea aplicației.....	42
Structura aplicației și gestionarea variabilelor de mediu.....	42
Configurarea bazei de date și definirea schemelor	46
Conexiunea cu serviciile externe.....	47
Implementarea serverului	49
Mecanisme de autentificare, înregistrare și autorizare	51
Gestionarea fișierelor media pe infrastructură cloud.....	55
Utilizarea LLM în gestionarea conținutului	59
Utilizarea LLM în analiza datelor și implementarea de strategii de business	61
Agregarea datelor de interacțiune ale utilizatorilor pentru vizualizări grafice și analize LLM	63
Mecanisme de gestionare a listei de prieteni	67
Implementarea secțiunii de comentarii.....	68
Implementarea serviciilor de localizare	71
Mecanismul de creare și editare a articolelor.....	72
4.3. Prezentarea aplicației	72
Concluzie	88
Anexă.....	89
Bibliografie.....	91

Introducere

În contextul actual al digitalizării accelerate, ecosistemul media se confruntă cu schimbări profunde, generate de interacțiunea tot mai complexă dintre jurnalismul tradițional și rețelele sociale. Dorința crescută pentru obținerea de informație în manieră cât mai rapidă, interactivă și personalizată a determinat o reevaluare a modului în care conținutul jurnalistic este creat, distribuit și consumat. În acest peisaj dominat de algoritmi și platforme sociale, presa se confruntă cu provocări majore în menținerea relevanței, credibilității și vizibilității sale.

Lucrarea de față propune dezvoltarea unei aplicații web de tip CMS (Content Management System), destinată redacțiilor de știri, care integrează funcționalități consacrate specifice rețelelor de socializare în activitatea jurnalistică. Scopul este de a crea o adapta și inova felul în care activitatea media este realizat, utilizând noile modalități de interacțiune digitală. Oferind o platformă intuitivă, scalabilă și orientată spre utilizator, aplicația urmărește consolidarea relației redacție—cititor, care în noul mediu, cititorul poate fi referit drept utilizator.

Ideea implementării unei astfel de soluții a fost motivată de nevoia de îndepărtare de modul în care rețelele sociale promovează în prezent informația, adaptând-o algoritmic la criterii de vizibilitate și angajare, ci nu la nevoile unui public-țintă bine definit. Totodată, această inițiativă reflectă dorința de a crea un mediu informațional mai sigur, protejat de dezinformare și atacuri hibride, oferind o platformă în care conținutul jurnalistic să poată fi distribuit în mod transparent, etic și responsabil.

Aplicația dezvoltă o infrastructură completă, bazată pe tehnologii moderne, precum React.js, Node.js (cu Express.js), MongoDB și servicii de tip cloud, alături de componente de inteligență artificială, menite să optimizeze atât procesele redacționale, cât și analiza comportamentului utilizatorilor. Prin integrarea de elemente precum comentarii ierarhizate, liste de prieteni, distribuirea de articole, statistici interactive, vizualizări grafice avansate și generarea de rapoarte pe bază de modele de limbaj largi, platforma urmărește să redefinească modul în care informația este consumată, diseminată și valorificată în mediul online.

Lucrarea este structurată în patru capitole, începând cu analiza problemei economice și a soluțiilor existente, urmată de specificarea cerințelor și proiectarea sistemului informatic, culminând cu detalierea procesului de implementare și încheind cu prezentarea aplicației dezvoltate, publicate în mediul online. Prin această abordare, se evidențiază complexitatea dezvoltării unui produs software adaptat la nevoile reale ale pieței media digitale actuale.

Capitolul 1 Descrierea problemei economice

1.1. Prezentarea domeniului abordat

Într-o lume tot mai digitalizată și mai interconectată prin intermediul rețelelor de socializare, peisajul media a fost redefinit atât de volumul uriaș de informații și de diversitatea surselor care o propagă, cât și de algoritmi ce au schimbat modul în care utilizatorii accesează conținut media. Pe lângă redefinirea inevitabilă a mediei, rețelele de socializare au amplificat totodată fenomenul dezinformării, devenind din ce în ce mai dificil de combătut, afectând întregul ecosistem informațional. Astfel, jurnaliștii se confruntă cu provocări semnificative în realizarea activității profesionale, una dintre cele mai mari fiind metodele prin care să ajungă la publicul larg, necesitând identificarea unor noi soluții moderne, cerute de noua realitate în care ne regăsim.

Profesia de jurnalist presupune activitatea de colectare, verificare și transmitere a informațiilor, analizelor și opiniilor relevante despre evenimente de actualitate, persoane publice, realizări cu impact asupra societății sau de interes larg.

În trecut, presa scrisă și televiziunea reprezentau principalele surse de informare, având și un rol de formator de opinii, modelând percepția asupra realității pentru un public extins. Astăzi, în condițiile în care informația este abundentă și canalele de comunicare sunt tot mai numeroase și diverse, presa ajunge cu dificultate întâmpină la publicul larg.

Multe publicații optează să atragă și să fidelizeze anumite segmente ale societății filtrând informația de interes, oferind perspective specifice asupra realității, dar nealterate și respectând normele etice și integritatea informațiilor. Pe de altă parte, rețelele de socializare au devenit un mediu propice dezinformării, unde realitatea este adesea distorsionată, iar competiția nu se mai bazează pe calitatea informației, ci pe capacitatea de a capta cât mai mult atenția utilizatorilor, manipularea, informațiile false și întărirea preconcepțiilor fiind modalități prin care cei ce propagă conținutul reușesc să-și genereze un public.

Deși aceste platforme sunt adeseori folosite pentru dezinformare și captarea atenției, funcțiile pe care le oferă ele pot deveni instrumente valoroase pentru redacțiile de știri, prin care acestea se pot conecta mai ușor și direct cu publicul larg, într-o manieră modernă, eficientă, totodată facilitând primirea unui feedback sugestiv pentru conținutul publicat.

Astfel, îmi propun să dezvolt o aplicație web care să îmbine elementele tradiționale regăsite în platformele de știri existente cu funcționalități specifice rețelelor sociale, oferind o soluție modernă prin care redacțiile să interacționeze mai bine cu publicul lor, prin care să reușească să își crească audiența fără a renunța la normele etice și informarea corectă și prin care să genereze venituri suplimentare din activitatea realizată.

1.2. Prezentarea activității care va fi informatizată

Domeniile vizate de informatizarea propusă sunt jurnalismul și social media, două arii care, deși inițial distincte, au ajuns să se intersecteze tot mai frecvent în contextul digitalizării accelerate a societății. Aplicația propusă urmărește să răspundă nevoilor actuale ale redacțiilor de presă, care se confruntă cu dificultăți tot mai mari în a-și menține relevanța și vizibilitatea într-un ecosistem informațional dominat de viteza de circulație a informației, algoritmi de distribuție și o concurență acerbă pentru atenția utilizatorilor.

Aplicația vizează informatizarea procesului editorial specific activității jurnalistice: redactarea, editarea, publicarea și organizarea articolelor, gestionarea conturilor redacționale și optimizarea fluxului de lucru intern. Aceste activități vor fi susținute de o interfață web intuitivă, adaptată atât nevoilor redactorilor, cât și celor ale administratorilor platformei. Redactorii vor putea crea, modifica și șterge articole rapid și eficient, în timp ce administratorii vor avea control asupra conturilor redacționale, articolelor create și a spațiului de stocare.

Datele sunt gestionate prin intermediul unei baze de date MongoDB Atlas, un serviciu DBaaS performant care oferă funcționalități avansate de securitate, autentificare și criptare end-to-end. Fișierele media sunt stocate și livrate prin Cloudinary, o platformă dedicată gestionării conținutului media în cloud, optimizată pentru aplicații web și mobile. Întreaga aplicație este găzduită pe Render, o platformă cloud modernă care permite construirea și implementarea automată a aplicației la fiecare modificare din ramura Git asociată. Această infrastructură cloud integrată asigură flexibilitate, scalabilitate, securitate și eficiență în gestionarea aplicației.

Dincolo de componenta jurnalistică, aplicația înglobează și funcționalități specifice rețelelor sociale, cu scopul de a transforma experiența utilizatorilor, răspunzând nevoilor acestora în noul peisaj digital. Utilizatorii pot accesa și citi articole, pot oferi feedback conținutului prin aprecierea acestuia sau prin implicarea în discuții în secțiunea de comentarii, pot salva postările într-o pagină dedicată, interacționa cu alți utilizatori prin intermediul unei liste de prieteni și pot distribui acestora articolele care le-au stârnit interesul, activități specifice unei rețele de socializare moderne, consacrate.

Această abordare modernă a transmiterii de conținut jurnalistic prin integrarea elementelor specifice aplicațiilor de social media are scopul de a crea un nou model de comunicare digitală, în care conținutul de calitate, verificat și etic, poate fi promovat prin mecanisme moderne de distribuție și interacțiune. Astfel, aplicația oferă redacțiilor un instrument puternic pentru a-și extinde audiența și a construi o legătură directă cu cititorii, fără a face compromisuri privind acuratețea și integritatea informației. În același timp, utilizatorii beneficiază de o platformă unde pot consuma conținut jurnalistic într-un mod personalizat și pot lua parte la dezbateri constructive, contribuind la formarea unei comunități informate și implicate.

Prin informatizarea modului de realizare a conținutului jurnalistic, eliminând dependența de modul în care platformele curente depind de modul în care informația este promovată, prin integrarea funcționalităților sociale aplicația propusă adresează una dintre cele mai mari provocări ale presei contemporane: reconectarea cu publicul într-un spațiu digital suprasaturat de informații, dar adesea lipsit de relevanță și coerență.

1.3. Analiza sistemelor existente pe piață

În contextul dezvoltării unei aplicații web care integrează funcționalități jurnalistice și elemente specifice rețelelor sociale, este esențială analiza comparativă a unor soluții deja existente pe piață. Aceasta permite identificarea celor mai relevante modele funcționale și inspirarea din bune practici aplicate în produse consacrate. Trei dintre cele mai influente platforme care oferă funcționalități apropiate de cele propuse în proiect sunt *Instagram* și *Reddit*.

1.3.1. Instagram

Instagram reprezintă o platformă de socializare online, care are ca scop principal partajarea fotografiilor și videoclipurilor între utilizatori. Aplicația a fost lansată în 2010 de două persoane: Kevin Systrom, absolvent la Stanford, lucrase anterior la Google, respectiv Mike Krieger, de asemenea absolvent la Stanford, lucrase anterior la platforma socializare Meebo. Aceștia au adoptat un stil minimalist pentru prototipul lor, concentrându-se pe postarea de imagini, cu opțiunea de a adăuga diverse filtre, pe comentarii și caracteristici de „apreciere” a postărilor de către utilizatori. Cei doi dezvoltatori au decis, de asemenea, să lanseze o versiune web a aplicației pentru a putea fi folosită de către dispozitivele iOS, care aveau să valorifice capacitățile fotografice mult mai îmbunătățite ale dispozitivelor iPhone 4 de la acea vreme. Au reușit să finalizeze aplicația în doar câteva luni, postând primele fotografii ale platformei în

luna iulie a anului 2010. Pe 6 octombrie 2010 Instagram a putut fi descărcată și pe dispozitivele mobile ale Apple, iar notorietatea versiunii web a avut ca rezultat a înregistrarea 25 000 de utilizatori încă din prima zi, ca la mai puțin de trei luni mai târziu să atingă un număr imens de un milion de utilizatori. (Eldridge A. , 2025)

Succesul răsunător a Instagram a determinat Facebook, din anul 2021 cunoscută sub denumirea „Meta Platforms, Inc.”, să achiziționeze platforma, în luna aprilie a anului 2012, la 18 luni de la lansare, pentru suma de un miliard de dolari în numerar și acțiuni. (Eldridge A. , 2025)

În prezent, Instagram este una dintre cele mai mari platforme de social media din lume, depășind în anul 2022 două miliarde de utilizatori activi lunar. (Eldridge A. , 2025)

1.3.2. Reddit

Reddit este o platformă de socializare și un site web în stil forum unde conținutul este organizat în forumuri și promovat de membrii fiecărui forum prin vot. (Yasar, What is Reddit? How it works, history and pros and cons, 2025)

Platforma a fost lansată în anul 2005 de către Steve Huffman și Alexis Ohanian, ambii , absolvenți ai Universității din Virginia, unde Steve Huffman a studiat informatica, iar Alexis Ohanian a studiat istoria. Reddit a devenit oficial o companie publică în anul 2024, cu sediul în San Francisco. Platforma servește ca o rețea de comunități specifice în care utilizatorii pot interacționa cu alte persoane care împărtășesc interese, hobby-uri și pasiuni similare. Spre deosebire de alte platforme de social media care folosesc o abordare bazată pe urmăritori, Reddit optează pentru organizarea sub formă de colecții de forumuri tradiționale. Această structură unică permite utilizatorilor să găsească și să se conecteze cu comunități construite în jurul intereselor comune, pentru a promova un angajament mai profund în discuții. (Eldridge A. , 2025)

La început, Reddit a fost menit să fie „prima pagină a internetului” – și într-adevăr acesta a fost motto-ul său de ceva timp. Ideea a fost ca utilizatorii să trimită cele mai bune articole de la principalele instituții de știri, reviste și alte site-uri, astfel încât să nu fie nevoiți să viziteze fiecare publicație în parte. Huffman a creat un algoritm care a permis celui mai popular conținut să se ridice în top atât în funcție de perioada în care a fost creat și promovat și de popularitatea de care a avut parte. Pe Reddit, popularitatea unei postări scade în timp, cu excepția cazului în care utilizatorii continuă să o „voteze” – pentru a o marca drept importantă, interesantă sau

relevantă. Astfel platforma își propune ca un anumit conținut să fie promovat în funcție de criterii obiective, nu criterii de apreciere personală, fiind un punct esențial prin care Reddit diferă de alte rețele sociale și site-uri de agregare de știri. Comentarea postărilor a fost adăugată la câteva luni după lansare, iar capacitatea utilizatorilor de a crea noi forumuri cunoscute sub denumirea de „subreddit” a apărut în anul 2008. (Eldridge A. , 2025)

În 2006, Huffman și Ohanian au vândut compania lui Condé Nast și s-au retras din rolurile lor de conducere în 2009, însă după anumite probleme în cadrul companiei, Huffman a revenit în 2015 ca nou CEO al companiei. Intrarea oficială pe piața publică s-a realizat în martie 2024. (Eldridge A. , 2025)

1.3.3. Analiza comparativă a funcționalităților

Instagram este o platformă de referință în domeniul social media, cu un set extins de funcționalități care încurajează interacțiunea dintre utilizatori și distribuirea conținutului. Aplicația propusă împrumută o serie de funcționalități esențiale de la aceste platforme:

- Aplicația oferă un feed continuu asemenea fluxului de postări din *Facebook* și *Instagram*. Articolele vor fi afișate într-un flux cronologic cu încărcare continuă la fiecare 20 de carduri de vizualizare, utilizând serviciile oferite de API-ul „*Intersection Observer*”, permițând utilizatorilor să acceseze constant conținut nou fără a fi necesară reîncărcarea paginii. Astfel utilizatorii au parte de o experiență îmbunătățită față de abordarea clasică privind paginarea.
- Funcționalitatea care stă la baza fiecărei rețele de socializare, posibilitatea creării unei liste de prieteni cu care un utilizator poate interacționa, nu lipsește nici soluției informatice propuse, interacțiunile fiind totuși centrate mai degrabă în conținutului jurnalistic. Utilizatorii vor avea posibilitatea de a adăuga prieteni, de a distribui articolele către aceștia și de a aprecia postările – funcționalități inspirate direct din dinamica rețelelor sociale moderne. De asemenea, utilizatorii vor avea o pagină dedicată articolelor primite de către prietenii din listă

Prin aceste funcționalități, aplicația reușește să creeze un mediu familiar pentru utilizatori, facilitând adoptarea platformei și încurajând interacțiunea continuă cu conținutul jurnalistic.

Mergând către o altă rețea de socializare amplu utilizată, soluția informatică propusă se inspiră mai departe de la platforma *Reddit*, care se remarcă prin structura sa orientată către

discuții și dezbateri, model care a influențat direct modul în care aplicația propusă va gestiona sistemul de comentarii. Caracteristicile preluate din această platformă sunt adaptate pentru a oferi o experiență interactivă și coerentă în jurul fiecărui articol:

- Comentariile urmează o structură similară sistemului de „threaduri” din *Reddit*, implementată printr-un algoritm propriu, utilizând structuri de date complexe. Comentariile vor fi salvate sub formă de listă, fiecare comentariu incluzând informații precum ID-ul, utilizatorul, conținutul și comentariul căruia îi răspunde dacă este cazul, pentru a gestiona într-o manieră cât mai optimă spațiul de stocare disponibil. Mai departe lista este transformată la încărcarea paginii cu articolul accesat într-un dicționar care reține în plus o listă de răspunsuri directe pentru fiecare comentariu. Ulterior, pe baza dicționarului, se creează un arbore care generează recursiv secțiunea de comentarii, ceea ce permite o afișare structurată a discuțiilor.
- Păstrarea contextului în cazul ștergerii este esențială pentru îndeplinirea componentei comunicaționale a aplicației, astfel că în cazul în care un comentariu este șters, dar conține răspunsuri, aplicația va păstra structura discuției, înlocuind din comentariul eliminat numele utilizatorului și conținutul comentariului cu eticheta „șters”. Acest comportament este identic cu cel adoptat de *Reddit*, unde menținerea contextului conversațional este esențială pentru coerența dezbaterilor. Pentru eficientizarea utilizării memoriei, în situațiile în care un comentariu șters care generase un fir conversațional nu mai are răspunsuri asociate, aplicația va elimina complet acel element, contribuind astfel la eliberarea de memorie și optimizarea performanței sistemului.

Prin combinarea funcționalităților cheie preluate din platforme de social media, precum *Instagram* și *Reddit*, aplicația propusă oferă un echilibru între consumul de conținut jurnalistic și interacțiunea socială modernă. Acest model hibrid valorifică atât experiența utilizatorului cu rețelele sociale, cât și nevoia de structurare, moderare și contextualizare a informației, specifică domeniului jurnalistic. Astfel, soluția propusă se distinge pe piață ca o platformă complexă, care urmărește nu doar diseminarea de informații, ci și formarea unei comunități active și implicate în jurul conținutului publicat.

În ceea ce privește funcționalitatea jurnalistică, aplicația propusă este influențată semnificativ de modul în care este structurat și prezentat conținutul de către platformele digitale ale marilor redacțiilor, inspirația majoră provenind de la redacția britanică „BBC News”, una

dintre cele mai mari și respectate organizații media internaționale, dar și de la redacția autohtonă „Ziarul Libertatea”.

Similar soluțiilor existente, aplicația utilizează o interfață bazată pe carduri interactive pentru afișarea articolelor, oferind utilizatorilor o experiență de navigare structurată și intuitivă. Această metodă de prezentare permite o verificare rapidă a titlurilor, care sunt recomandate a fi cât mai detaliate și sugestive, iar imaginea de prezentare are rolul de a informa utilizatorul mai pe larg despre conținutul articolului respectiv, contribuind la o mai bună organizare vizuală a paginii, o transmitere eficientă a conținutului și la o experiență cât mai fluidă pentru utilizatorul modern.

De asemenea, filtrarea articolelor în funcție de categorii constituie o funcționalitate esențială, regăsită în majoritatea platformelor informatice orientate spre diseminarea conținutului jurnalistic. Această opțiune permite utilizatorilor să acceseze rapid și eficient informațiile relevante, în funcție de propriile interese tematice, facilitând astfel personalizarea experienței de consum media și îmbunătățind navigabilitatea în cadrul platformei.

Aplicația propusă va implementa o structură bazată pe un set de categorii bine definit, concentrată pe domenii de interes major pentru publicul general, după cum urmează: Politică, Evenimente Externe, Financiar, Sport, Stil de viață și Tehnologie, acoperind o paletă largă de subiecte, menite să satisfacă răspundă intereselor cât mai multor utilizatori.

Pentru a spori accesibilitatea și deschiderea către un public internațional, s-a decis implementarea unei interfețe afișate în limba engleză, limba dominantă în mediul online global, ceea ce oferă premisele extinderii audienței și ale promovării conținutului jurnalistic dincolo de granițele regionale.

Integrarea acestor practici validate în mediul profesional asigură faptul că aplicația nu doar satisface cerințele tehnice, ci răspunde și așteptărilor din punct de vedere al ergonomiei și experienței de utilizare, fiind aliniată la standardele impuse prin exemplu de platformele de știri moderne.

Capitolul 2 Analiza sistemului informatic

2.1. Specificarea cerințelor sistemului informatic

Proiectarea unui sistem informatic presupune utilizarea resurselor financiare, umane și materiale într-un mod organizat, urmând o serie de etape bine definite, cum ar fi analiza, proiectarea, implementarea și testarea. Aceste activități sunt esențiale pentru a crea un sistem performant și adaptat nevoilor organizației.

Scopul principal al acestui proiect constă în dezvoltarea unui sistem informatic integrat care să sprijine activitatea unei redacții de știri prin intermediul unei aplicații web moderne. Aceasta îmbină funcționalități specifice unui sistem de tip CMS (Content Management System) cu elemente interactive caracteristice platformelor de social media, oferind astfel o soluție complexă, scalabilă și orientată spre nevoile actuale ale consumatorilor moderni de conținut digital.

Platforma este concepută să funcționeze pe infrastructuri de tip cloud, astfel utilizând serviciile: MongoDB Atlas pentru stocarea datelor structurale, Cloudinary pentru gestionarea eficientă a fișierelor media, Render pentru găzduirea completă a aplicației, respectiv ambele componente – backend și frontend.

Aplicația va integra două componente funcționale majore:

- Componenta redacțională, dedicată echipei editoriale, care va permite redactarea, editarea, clasificarea și publicarea articolelor, precum și gestionarea conturilor și a resurselor media;
- Componenta orientată către utilizator, destinată publicului cititor, care va putea accesa, aprecia, comenta, salva și distribui articole, precum și interacționa cu alți utilizatori prin adăugarea în liste de prieteni sau prin schimbul de conținut.

Prin această abordare, proiectul urmărește nu doar automatizarea și eficientizarea fluxului editorial, ci și consolidarea relației dintre redacție și cititor, printr-un mediu interactiv, sigur și ușor de utilizat. Platforma propusă se poziționează astfel ca un instrument digital adaptat noilor cerințe ale presei moderne, favorizând jurnalismul de calitate într-un ecosistem informațional dinamic.

Pe partea de redacție, aplicația va integra funcționalități esențiale care susțin întregul proces editorial. Redactorii vor avea la dispoziție o interfață intuitivă prin care pot crea articole,

le pot publica, actualiza sau șterge, asigurând astfel menținerea relevanței și actualității conținutului jurnalistic. Fiecare articol va fi asociat unei categorii specifice, indicată de redactor în momentul publicării, alături de etichete relevante pentru o mai bună organizare și filtrare și căutare de către utilizatorul final a conținutului jurnalistic. Articolele vor fi afișate cronologic într-un flux editorial continuu, cu funcționalitate de „încărcare infinită”, care va reflecta inclusiv modificările și actualizările operate ulterior. Administratorul va avea acces la toate articolele create, va putea gestiona conturile redactorilor și va putea elimina fișierele media care nu mai sunt utilizate, contribuind la optimizarea spațiului și a resurselor aplicației.

Pe lângă aceste caracteristici, platforma va oferi statistici utile, cum ar fi numărul de vizualizări sau interacțiuni pentru fiecare articol, feedbackul primit prin comentarii, aprecieri, salvări și distribuire, ajutând redacția să ia decizii informate privind crearea de conținut. Redactorii vor avea acces doar statisticile privind articolele proprii, în timp ce administratorul va avea acces la toate datele statistice. Prin utilizarea acestui sistem, redacția va putea centraliza și automatiza multe dintre procesele sale, economisind timp și resurse.

Pe partea de social media, elementele integrate în aplicație vor contribui semnificativ la crearea unei comunități active în jurul redacției prin facilități precum posibilitatea de a comenta, aprecia, primi recomandări de articole și distribui articole listei de prieteni. Utilizatorii vor fi implicați în mod direct în interacțiunea cu conținutul, ceea ce le va oferi un sentiment de apartenență la platformă, creând o comunitate.

Astfel, comentariile permit utilizatorilor să-și exprime opiniile, să dezbată subiectele abordate și să interacționeze atât cu alți utilizatori, cât și cu autorii articolelor. Aprecierile ajută la evidențierea articolelor populare, încurajând redacția să creeze mai mult conținut în linie cu preferințele publicului. Comentariile vor fi gestionate de către administrator, care ca reprezentant al redacției își rezervă dreptul de a le elimina în funcție de politicile de conținut ale redacției.

Pentru gestionarea datelor, s-a optat pentru o soluție de tip NoSQL, utilizând baza de date MongoDB, stocată în cloud prin intermediul serviciului MongoDB Atlas. Această alegere oferă flexibilitate în structurarea datelor și scalabilitate în funcție de necesitățile aplicației, asigurând în același timp atât un nivel ridicat de securitate, cât și performanță. Datele vor fi introduse și gestionate conform unor scheme predefinite, asigurând astfel consistența logică a structurii informaționale.

Fișierele media asociate articolelor vor fi stocate prin intermediul platformei Cloudinary, un serviciu cloud specializat în gestionarea și livrarea conținutului media. La momentul încărcării, Cloudinary generează automat adrese URL unice pentru fiecare fișier, aceste adrese fiind ulterior salvate în baza de date pentru a facilita redarea rapidă și eficientă a resurselor în aplicație.

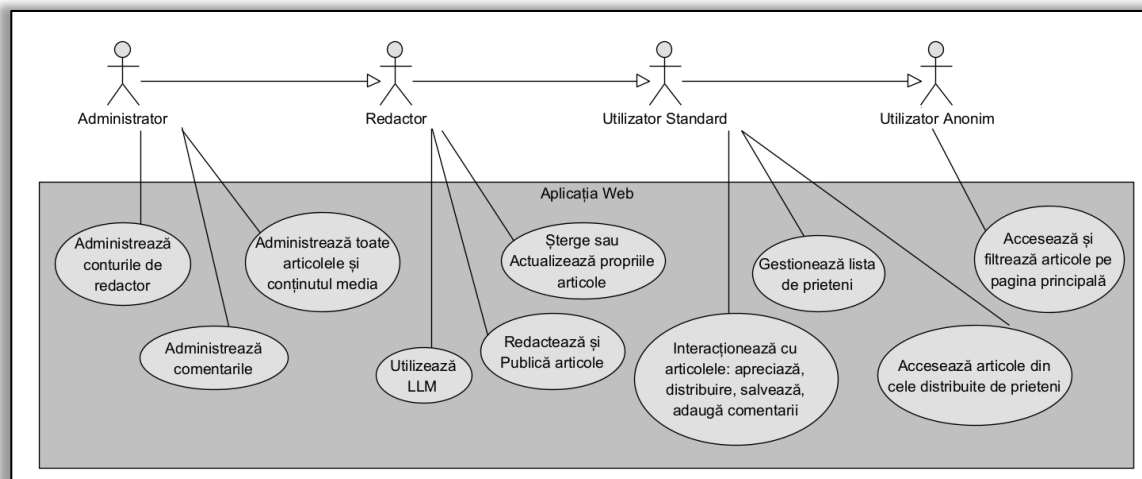
Aplicația web este găzduită pe platforma Render, care permite integrarea completă a backend-ului și frontend-ului în același mediu de execuție. Partea de interfață (frontend), realizată în React, este compilată direct în cadrul serverului și servită prin același punct de acces cu aplicația de backend. Acest mod de lansare asigură o coerență între cele două componente ale aplicației, simplificând procesul de dezvoltare, implementare și întreținere, și permițând utilizarea variabilelor de mediu într-o manieră centralizată.

2.2. Analiza sistemului existent

Diagramele UML ¹sunt un set de instrumente vizuale standardizate, utilizate pe scară largă în ingineria software și în alte domenii pentru a specifica, vizualiza, construi și documenta artefactele sistemelor software. Ele oferă o modalitate consistentă și comprehensivă de a modela diferite aspecte ale unui sistem, de la structura sa statică la comportamentul său dinamic (Visual Paradigm, 2025).

Diagramele de cazuri de utilizare sunt de obicei denumite diagrame de comportament și sunt utilizate pentru a descrie un set de acțiuni (cazuri de utilizare) pe care anumite sisteme ar trebui sau le pot realiza în colaborare cu unul sau mai mulți utilizatori externi ai sistemului (actori). Fiecare caz de utilizare ar trebui să ofere un rezultat observabil și valoros actorilor sau altor părți interesate ale sistemului (Fakhrouddinov, UML Use Case Diagrams, 2025).

¹ UML = Unified Modeling Language = Limbaj de Modelare Unificat



Figură 1 - Diagrama cazurilor de utilizare

Diagrama prezentată, intitulată "Diagrama cazurilor de utilizare", ilustrează arhitectura funcțională a unei aplicații web prin prisma interacțiunilor dintre actori și sistem. Această reprezentare vizuală, specifică metodologiei de modelare Unified Modeling Language (UML), deliniază comportamentele sistemului din perspectiva utilizatorului final și a altor entități externe.

Sistemul este structurat în jurul a patru roluri cheie, sau actori, fiecare având un set distinct de responsabilități și interacțiuni: utilizatorul anonim, utilizatorul standard, redactorul și administratorul aplicației.

Actorul reprezentat de utilizatorul anonim reprezintă acea entitatea care este limitată strict la utilizarea funcționalităților de bază ale aplicației, accesarea și căutarea de articole. Astfel acesta este singurul actor care nu are nevoie de un cont pentru a utiliza aplicația și este considerat vizitator și potențial utilizator standard.

Actorul reprezentat de utilizatorul standard reprezintă entitatea principală care interacționează cu conținutul și rețeaua socială a aplicației, prin crearea unui cont gratuit și promovarea de la utilizator anonim. Funcționalitățile asociate acestui rol includ:

- Accesarea și filtrarea articolelor pe pagina principală – capacitatea de a naviga și căuta personalizat vizualizarea conținutului disponibil.
- Accesarea articolelor distribuite de prieteni – o componentă specifică aplicațiilor de socializare care permite vizualizarea conținutului partajat de utilizatori din cadrul rețelei personale.

- Gestionarea listei de prieteni – administrarea conexiunilor sociale în cadrul aplicației, precum adăugarea sau eliminarea utilizatorilor din rețeaua personală.
- Interacțiunea cu paginile de articol printr-un set de acțiuni specifice care îmbunătățesc angajamentul utilizatorului cu conținutul, dar și cu alți utilizatori, precum "aprecierile" (liking), "distribuirile" (sharing), "salvarările" (saving) și "adăugarea comentariilor" (commenting).

Actorul reprezentat de redactorul sau autor este entitatea responsabilă de crearea și gestionarea conținutului jurnalistic propriu în cadrul platformei. Funcționalitățile sale specifice sunt:

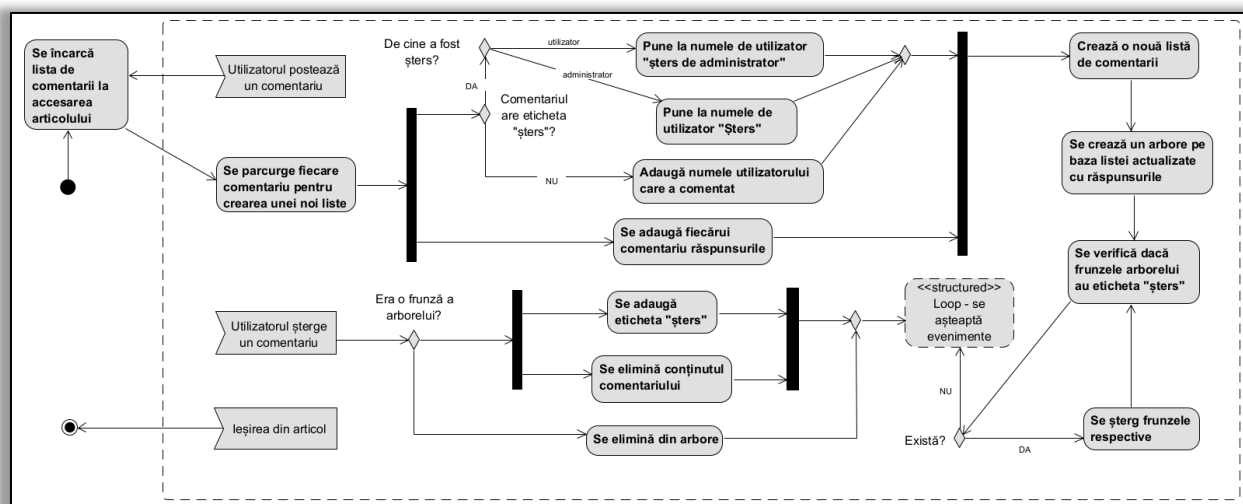
- Redactarea și publicarea articolelor – procesul de creare și lansare a conținutului nou.
- Ștergerea sau actualizarea propriilor articole – capacitatea de a menține și de a revizui conținutul personal.
- Acesta are un rol și dincolo de sfera aplicației, fiind cel care trebuie să se documenteze li informeze constant, dar și să participe la diverse evenimente de interes public, pentru a păstra conținutul cât mai relevant și precis.

Actorul reprezentat de administrator este entitatea care dispune de privilegii extinse de gestionare la nivelul întregului sistem, asigurând integritatea și buna funcționare a aplicației. Atribuțiile sale cuprind:

- Administrarea tuturor articolelor și a conținutului media: Controlul centralizat asupra publicării, modificării și ștergerii conținutului.
- Administrarea comentariilor: Monitorizarea și moderarea interacțiunilor utilizatorilor.
- Administrarea conturilor de redactor: Gestionarea accesului și a permisiunilor pentru actorii care contribuie cu conținut.

Această diagramă oferă o perspectivă generală asupra cerințelor funcționale ale aplicației web, asupra rolurilor și privilegiilor fiecărui actor și servește ca instrument esențial în faza de analiză și proiectare a sistemului, facilitând înțelegerea comportamentului așteptat al aplicației CMS².

² CMS = Content Management System = Sistem de administrare a conținutului



Figură 2 - Diagramă de activitate: Secțiunea de comentarii

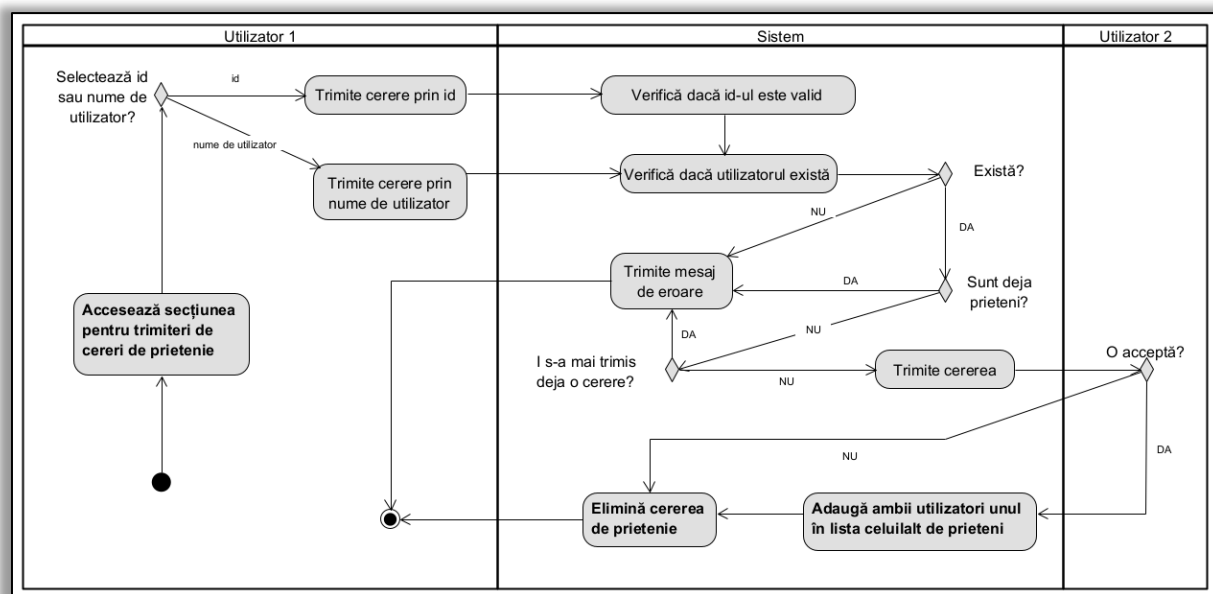
Diagrama de activitate este o diagramă de comportament UML care arată fluxul de control sau fluxul de obiecte cu accent pe secvența și condițiile fluxului. Acțiunile coordonate de modelele de activitate pot fi inițiate pentru că alte acțiuni se termină de executat, pentru că obiectele și datele devin disponibile sau pentru că apar unele evenimente externe fluxului. (Fakhroutdinov, Activity Diagrams, 2025)

Diagrama prezentată constituie o Diagramă UML de Activitate, care în contextul dat, detaliază etapele și deciziile implicate în gestionarea comentariilor în cadrul unei pagini de articol în cadrul aplicației.

Diagrama ilustrează comportamentul sistemului pe parcursul interacțiunii utilizatorului cu secțiunea de comentarii, cât timp acesta se află pe pagina articolului. Se optează pentru stocarea comentariilor în baza de date sub forma unei liste de obiecte, fiecare obiect conținând informații strict relevante pentru a optimiza gestionarea memoriei. Astfel, în contextul relațiilor dintre comentarii, această structură păstrează doar identificatorul comentariului căruia îi răspunde, dacă este cazul. Ulterior, în urma încărcării datelor în pagina de articol accesată de utilizator, se construiește o nouă listă care asociază fiecărui comentariu toate răspunsurile directe aferente, permițând ulterior generarea unei structuri arborescente care să reflecte ierarhia discuțiilor și pe baza căreia să se genereze secțiunea de comentarii.

Totodată, diagrama surprinde modul în care sistemul reacționează la adăugarea sau eliminarea comentariilor de către utilizatorul curent, actualizând structurile interne în funcție de evenimentele declanșate. În plus, este evidențiată implementarea unui mecanism de ștergere „logică”, inspirat de modelul utilizat de platforma „Reddit”, care păstrează o amprentă minimă

a comentariului șters în scopuri de consistență structurală. Diagrama subliniază și rolul administratorului în menținerea unui mediu comunicațional adecvat, acesta având responsabilitatea de a elimina comentariile cu conținut neadecvat, contribuind astfel la calitatea și relevanța discuțiilor.



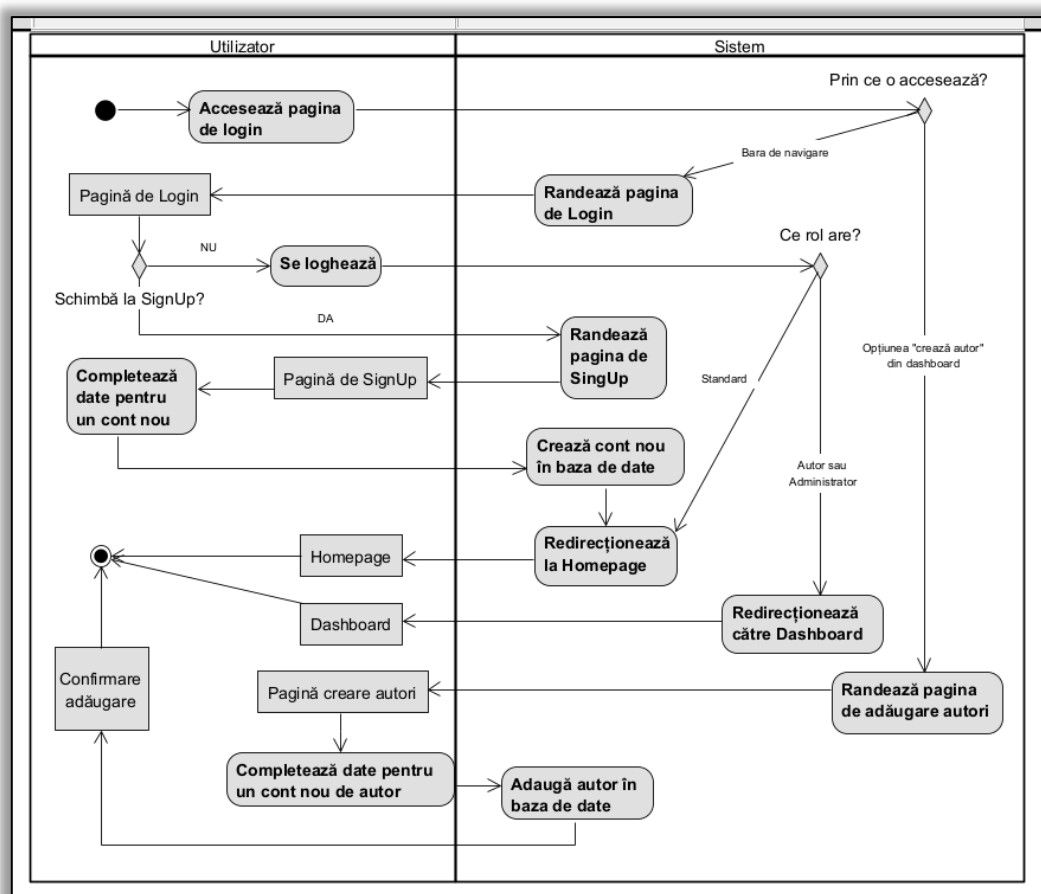
Figură 3 - Diagramă de activitate: Flux operațional cereri de prietenie

Diagrama de activitate ilustrată prezintă fluxul operațional aferent procesului de trimitere și acceptare a cererilor de prietenie între utilizatori în cadrul sistemului informatic. Diagrama este structurată în trei coloane corespunzătoare celor doi actori implicați – Utilizator 1 și Utilizator 2 – și modului central „Sistem”, care coordonează logica procesului.

Procesul debutează cu inițializarea acțiunii de trimitere a unei cereri de prietenie de către Utilizatorul 1, prin accesarea secțiunii dedicate, unde are posibilitatea de a selecta una dintre cele două modalități de identificare a destinatarului – prin ID sau nume de utilizator. În funcție de opțiunea selectată, sistemul validează inputul primit. În cazul utilizării ID-ului, se verifică în plus și validitatea acestuia, apoi se verifică existența destinatarului în baza de date.

Diagrama prezintă și cum sistemul ar trebui să gestioneze erorile și trimiterea de mesaje de eroare către utilizator. În situația în care identificarea este validă, sistemul continuă verificările pentru a determina dacă utilizatorii sunt deja prieteni sau dacă o cerere a fost deja transmisă anterior. Dacă există o cerere în așteptare, sistemul anunță utilizatorul care a emis noua cerere, prevenindu-se astfel redundanța.

În absența unui conflict logic, cererea este trimisă către Utilizatorul 2, care este invitat să o accepte. Acceptarea cererii conduce la actualizarea listelor de prieteni pentru ambii utilizatori, aceștia fiind reciproc adăugați în rețelele proprii, iar în caz negativ, cererea este eliminată, tot din considerente legate de optimizarea stocării.



Figură 4 - Diagramă de activitate: Redare contextuală Login

Diagrama de activitate prezentată modelează procesul de autentificare, înregistrare și gestionare a conturilor de autor într-un sistem informatic, accentuând redarea contextuală a interfețelor utilizatorului, în funcție de traseul de navigare și de privilegiile de acces, paradigmă de dezvoltare specifică utilizării React.

Procesul este inițiat de utilizator prin accesarea paginii de login, după care sistemul determină contextul de acces — prin bara de navigare sau din panoul de control al administratorului. Dacă utilizatorul accesează pagina din bara de navigare, sistemul redă interfața de autentificare standard. În caz contrar, accesarea opțiunii „crează autor” determină sistemul să redea interfața modificată, asociată procesului de creare a unui cont de autor.

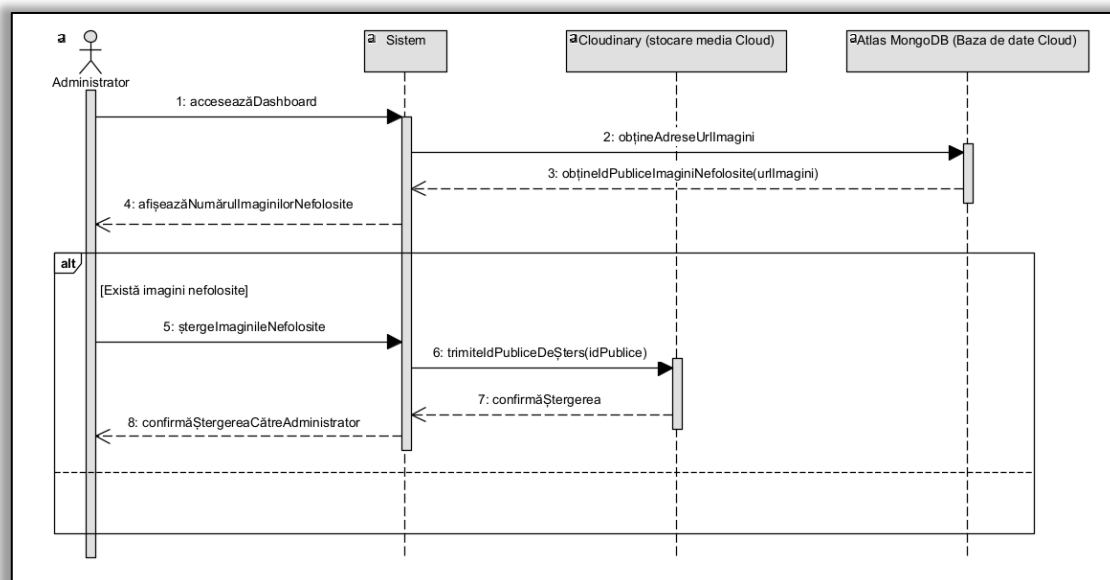
După afișarea paginii de „Login”, dacă aceasta a fost redată în forma standard, utilizatorul are opțiunea de a se autentifica sau de a comuta către procesul de înregistrare, schimbând structura formularului și funcționalitatea sa. În cazul în care se alege înregistrarea, sistemul redă pagina de „Sign Up” și preia datele introduse pentru crearea unui cont nou.

Ulterior, contul este înregistrat în baza de date, iar utilizatorul este redirecționat către pagina principală. Dacă s-a optat pentru autentificare, utilizatorul este redirecționat în continuare către aceeași pagină menționată anterior, dacă acesta este utilizator standard, în caz contrar fiind redirecționat către fie către panoul de control specific rolului (autor sau administrator), caz în care fluxul se consideră finalizat.

Dacă utilizatorul era autentificat anterior și avea rol de administrator, din diagramă reiese faptul că se continuă cu fluxul de creare de conturi noi pentru autori. După completarea datelor și înregistrarea unui cont, sistemul transmite un mesaj de confirmare a operațiunii către administrator, determinând alt caz de finalizare a fluxului operațional.

Prin această diagramă se evidențiază un comportament adaptiv al sistemului, care redă aceeași interfață în contexte funcționale distincte, în funcție de rolul utilizatorului și de punctul de acces. Astfel, se obține o reutilizare eficientă a componentelor vizuale, alături de o logică robustă de control al fluxului și al permisiunilor.

O diagramă de secvență este un tip de diagramă UML care ilustrează modul în care obiectele sau componentele unui sistem interacționează între ele într-o anumită ordine temporală, pentru a realiza o funcționalitate specifică. Este o reprezentare vizuală a fluxului de mesaje schimbate între entități, cu accent pe ordinea în care aceste mesaje sunt trimise și primite (Fakhrouddinov, UML Sequence Diagrams, 2025). Diagrama de secvență este cel mai comun tip de diagramă de interacțiune, care se concentrează pe schimbul de mesaje între un număr de linii de viață. Diagrama secvenței descrie o interacțiune concentrându-se pe secvența de mesaje care sunt schimbate, împreună cu specificațiile de apariție corespunzătoare pe liniile de viață (Object Management Group, 2017, p. 595).

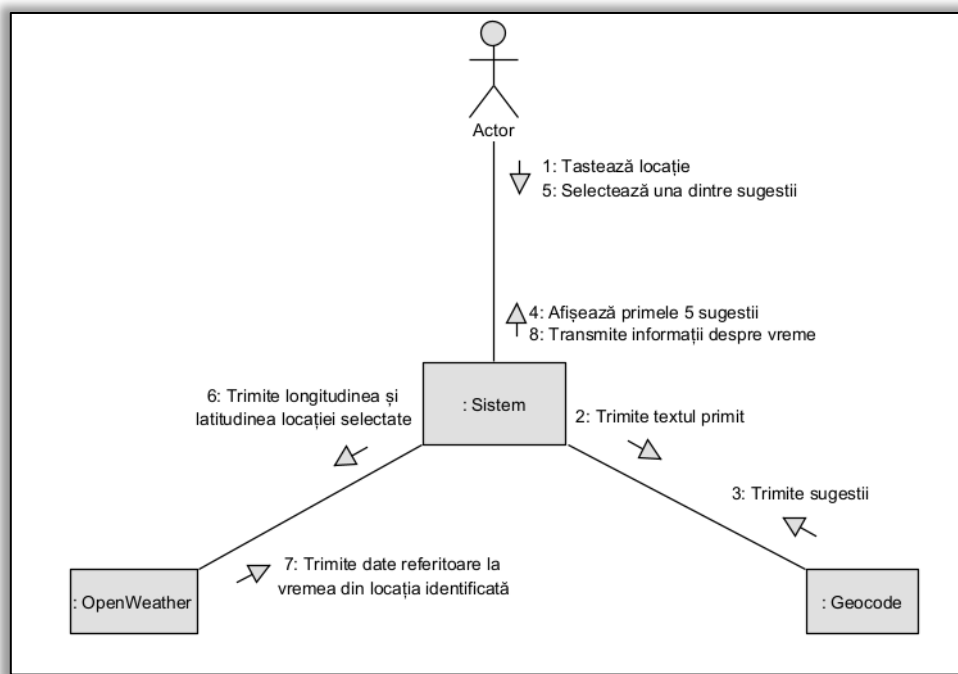


Figură 5 - Diagramă de secvență: Sistem autonom de șterge a imaginilor nefolosite

Diagrama de secvență prezentată modelează procesul de gestionare și curățare a resurselor de imagine rămase nefolosite în cadrul aplicației, evidențiind interacțiunile dintre administrator, sistem și serviciile de stocare cloud pentru bază de date (Atlas MongoDB) și fișierele media (Cloudinary).

Procesul este inițiat de Administrator prin accesarea tabloului de control al aplicației, ulterior, sistemul interoghează Atlas MongoDB pentru a obține adresele URL ale imaginilor, apoi determină, pe baza acestora, ID-urile publice ale imaginilor nefolosite. Această operațiune pregătitoare culminează cu afișarea numărului de imagini nefolosite către Administrator.

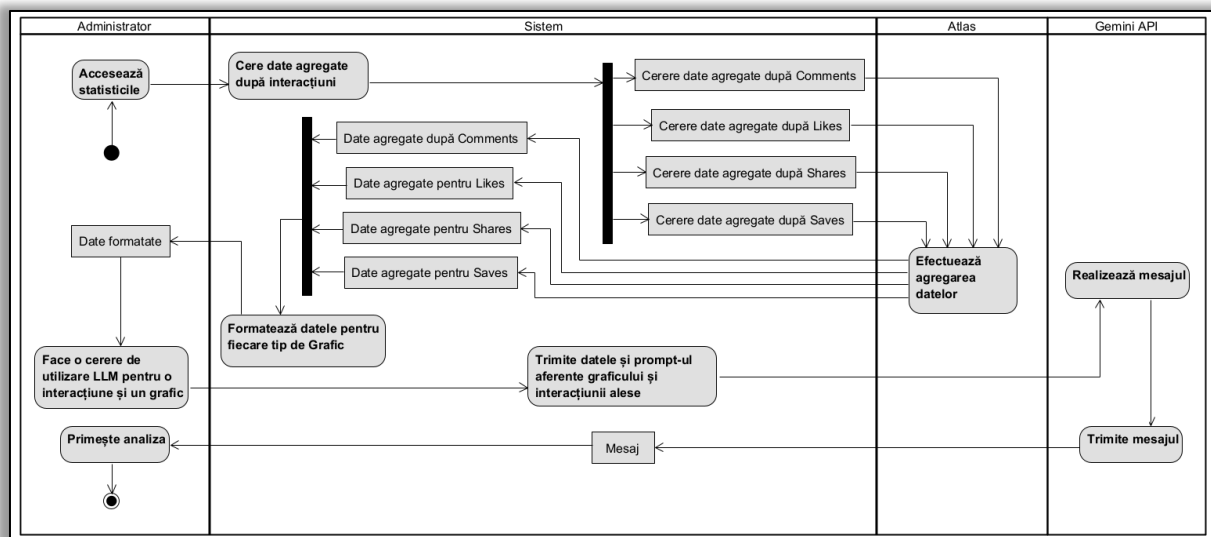
Un punct de decizie este ilustrat prin blocul alternativ "alt" care determină ca atunci când există imagini nefolosite administratorul să poată iniția procesul de ștergere. În acest scenariu, Sistemul transmite solicitarea de ștergere a ID-urilor publice către Cloudinary, serviciul de stocare media în cloud. După primirea confirmării ștergerii de la Cloudinary, sistemul notifică administratorul cu privire la succesul operațiunii. Această abordare structurată asigură o gestionare eficientă a spațiului de stocare și o interacțiune clară între utilizator și infrastructura de backend.



Figură 6 - Diagramă de comunicare: Comunicare Geocode și OpenWeather

Diagrama de comunicare este un tip de diagramă de interacțiune UML care arată interacțiunile dintre obiecte și/sau părți (reprezentate ca linii de viață) folosind mesaje secvențiate într-un aranjament liber. Diagrama de comunicare corespunde unei diagrame de secvență simplă (adică pot fi convertite una la alta), fără mecanisme de structurare, cum ar fi utilizări de interacțiune și fragmente combinate (Fakhroutdinov, UML Communication Diagrams Overview, 2025).

Diagrama de comunicare prezintă în contextul actual comunicarea dintre utilizator cu sistemul și dintre sistem cu API-urile Geocode și OpenWeather, ilustrând ordinea mesajelor prin care un utilizator poate obține informații despre vreme în pagina principală, fiind una dintre utilitățile și implementările API-ului de la Geoapify.



Figură 7 - Diagramă activitate: Prelucrare date pentru analiză prin LLM

Diagrama de activitate prezintă procesul prin care datele sunt prelucrate pentru a fi furnizate unui model de limbaj larg, care generează o analiză pe baza acestora.

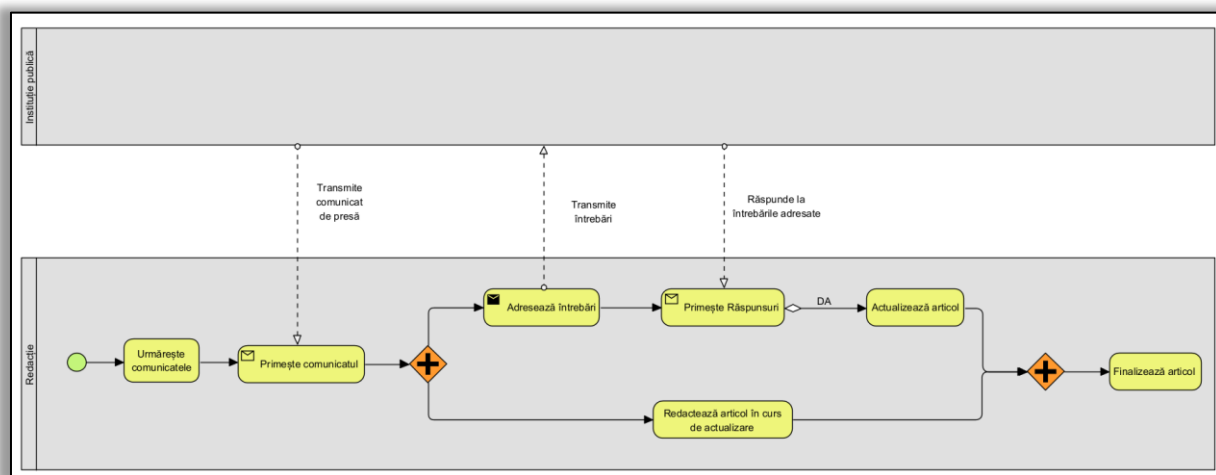
Astfel, când un administrator accesează pagina de vizualizare a statisticilor, sistemul inițiază un proces complex de extragere, prelucrare și analiză a datelor pentru a oferi perspective detaliate asupra interacțiunilor social media. Acest flux continuă cu sistemul care trimite cereri de agregare a datelor către serviciul de baze de date Cloud Atlas. Aceste cereri sunt formulate pentru a colecta informații specifice despre diverse tipuri de interacțiuni social media (cum ar fi aprecieri, salvări, distribuiri și comentarii). Cloud Atlas procesează aceste cereri și returnează sistemului datele agregate.

Ulterior, sistemul preia aceste date agregate și le prelucrează pentru a se potrivi unor tipuri specifice de grafice (de exemplu: diagrame bar, pie, scatter plot sau radar). Această prelucrare implică, de exemplu, gruparea datelor pe categorii de vârstă, gen sau număr de prieteni, în funcție de cerințele fiecărui tip de vizualizare. După formatare, datele sunt transmise utilizatorului în interfața de vizualizare.

Ultima etapă a acestui proces este momentul în care utilizatorul poate solicita o analiză a datelor primite formate. Sistemul preia aceste date și le trimite împreună cu un prompt

contextualizat către modelul lingvistic de mari dimensiuni Gemini. Gemini, pe baza instrucțiunilor primite, generează o analiză detaliată a datelor.

Fluxul se încheie atunci când utilizatorul primește această analiză finală, oferind informații acționabile despre tendințele și performanța interacțiunilor social media din cadrul sistemului informatic.



Figură 8 - Diagramă BPMN

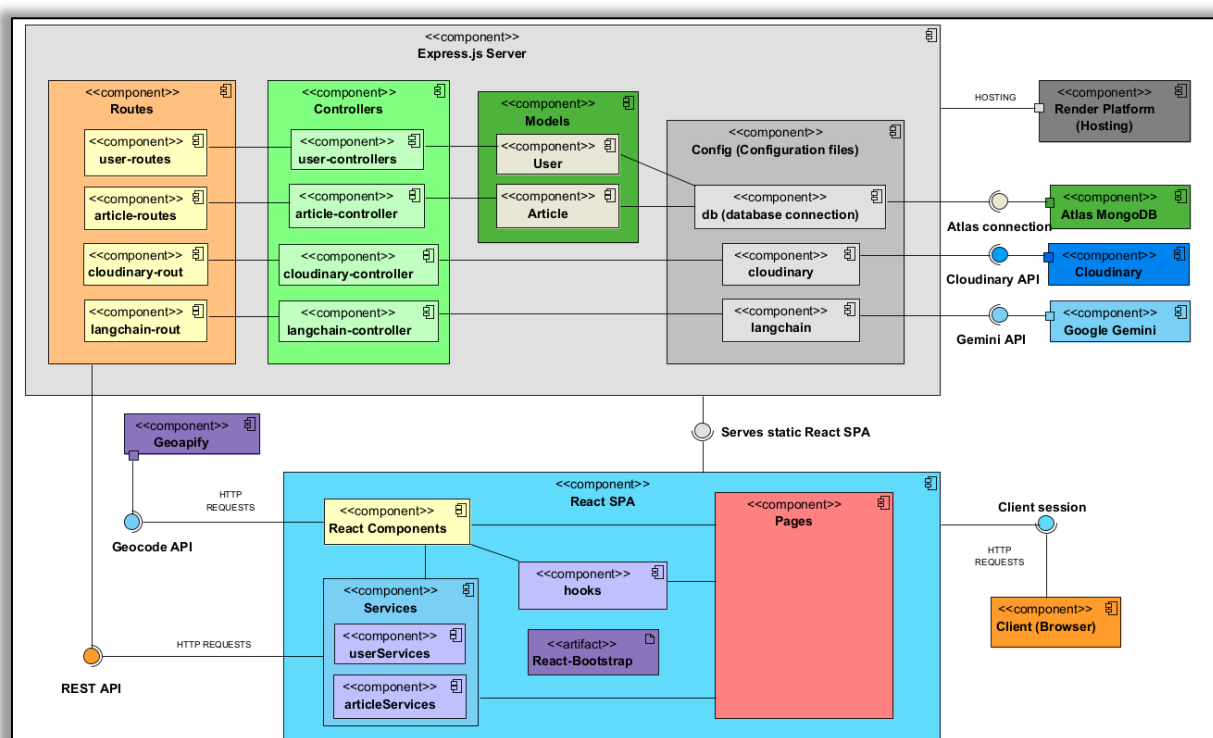
„Business Process Model and Notation” (BPMN) este o reprezentare grafică pentru proiectarea și modelarea vizuală a proceselor de afaceri. Este un standard pentru modelarea proceselor de afaceri și oferă o notație grafică pentru specificarea proceselor într-o diagramă a proceselor de afaceri (BPD) (Visual Paradigm, 2024).

Diagrama BPMN descrie, în contextul actual, fluxul de lucru dintre Instituțiile publice și redacție în contextul redactării unui articol bazat pe comunicate de presă. Același proces este urmat și în cazul altor surse de informare.

Capitolul 3. Proiectarea sistemului informatic

3.1. Proiectarea noului sistem

Diagrama componentelor arată componentele, interfețele furnizate și necesare, porturile și relațiile dintre ele. Acest tip de diagrame este folosit în Dezvoltarea bazată pe componente (CBD³) pentru a descrie sisteme cu arhitectură orientată pe servicii (SOA⁴). Dezvoltarea bazată pe componente se bazează pe ipoteza că componentele construite anterior ar putea fi reutilizate și că componentele ar putea fi înlocuite cu alte componente „echivalente” sau „conforme”, dacă este necesar. Artefactele care implementează componenta sunt menite să fie capabile să fie implementate și reinstalate independent, de exemplu pentru a actualiza un sistem existent (Fakhroutdinov, UML Component Diagrams, 2025).



Figură 9 - Diagramă de deployment

În cazul de față, diagrama de componente descrie arhitectura unei aplicații web full-stack, organizată în două subsisteme principale: frontend-ul de tip SPA ⁵(Single Page

³ CBD = Component-Based Development

⁴ SOA = Service-Oriented Architecture

⁵ SPA = Single Page Application (aplicație pe o singură pagină) - O aplicație SPA este o aplicație web sau un site web care interacționează cu utilizatorul prin rescrierea dinamică a paginii web curente cu date noi de pe serverul web, în loc de metoda implicită de încărcare a paginilor noi. Scopul este de a obține tranziții mai rapide care fac site-ul web să se simtă mai mult ca o aplicație nativă. (Flanagan, 2006, p. 497)

Application), construit cu React, și backend-ul bazat pe framework-ul Express.js. Procesul începe cu interacțiunea Clientului (Browser), care trimite cereri HTTP către aplicație.

Frontend-ul React SPA constituie interfața utilizator principală, fiind responsabil pentru gestionarea interacțiunii cu utilizatorul. Acesta este compus din următoarele componente:

- **Componente UI (React Components)** – elemente reutilizabile care formează interfața vizuală a aplicației.
- **Pagini (Pages)** – definesc rutele și structura navigabilă a aplicației.
- **Servicii (Services)** – împărțite logic în `userServices` și `articleServices`, aceste module gestionează comunicarea cu backend-ul prin apeluri HTTP.
- **„Hooks”**⁶ personalizate (Custom Hooks) – funcții specializate React, utilizate pentru extragerea și reutilizarea logicii comune.
- **React-Bootstrap** – bibliotecă UI utilizată pentru dezvoltarea de interfețe grafice într-un mod eficient și standardizat. Este inclusă în diagramă ca artefact.

Interfața React SPA trimite cereri HTTP către backend-ul Express.js, care este compus din mai multe module organizate pe principii MVC (Model-View-Controller):

- **Rute (Routes)** – definesc punctele finale ale API-ului RESTful;
- **Controlere (Controllers)** – implementează logica de procesare asociată fiecărei;
- **Modele (Models)** – definesc structura datelor persistente;
- **Fișiere de configurare (Config)** – centralizează setările aplicației, incluzând conexiunea la baza de date MongoDB Atlas, integrarea serviciului Cloudinary și autentificarea în API-ul Gemini, utilizând Langchain.

Cadrul Model-View-Controller (MVC) este un model arhitectural/design care separă o aplicație în trei componente logice principale Model, View și Controller. Fiecare componentă arhitecturală este construită pentru a gestiona aspecte specifice de dezvoltare ale unei aplicații. Astfel, izolează logica de afaceri și stratul de prezentare unul de celălalt. A fost folosit în mod tradițional pentru interfețele grafice de utilizator (GUI) pentru desktop. În zilele noastre, MVC

⁶ Hooks (Cârlig) = Cârligele sunt funcții care permit conectarea la funcțiile React și ale ciclului de viață din componentele funcției. Cârligele nu funcționează în cadrul claselor, deci permit utilizarea React fără nevoia de a crea clase. (Meta Platforms, Inc., 2025)

este unul dintre cele mai frecvent utilizate cadre de dezvoltare web standard din industrie pentru a crea proiecte scalabile și extensibile. De asemenea, este folosit pentru proiectarea aplicațiilor mobile. Acest standard a fost creat de Trygve Reenskaug cu scopul principal de a rezolva problema utilizatorilor care controlează un set mare și complex de date prin împărțirea unei aplicații mari în secțiuni specifice, care au toate propriul scop (GeeksForGeeks, 2024).

Persistența datelor este asigurată prin conectarea serverului Express.js la o bază de date externă MongoDB Atlas. Totodată, sistemul backend integrează servicii externe pentru a extinde funcționalitățile aplicației:

- **Cloudinary API** – utilizat pentru gestionarea resurselor media (imagini).
- **Google Gemini API** (prin Langchain) – integrează funcționalități de inteligență artificială (generare de text, procesare semantică).
- **Geocodify API** – un serviciu de geocodificare accesat din frontend pentru localizarea geografică a informațiilor.

Infrastructura de găzduire este asigurată prin platforma Render, care oferă atât găzduirea aplicației Express.js, cât și servirea fișierelor statice ale aplicației React către clientul final (browser). Această arhitectură facilitează o separare clară a responsabilităților între partea de prezentare (frontend) și cea de procesare (backend), contribuind astfel la scalabilitatea, mentenabilitatea și modularitatea sistemului. Interacțiunea eficientă între componente, alături de integrarea cu servicii externe, susține dezvoltarea unei aplicații moderne, robuste și extensibile.

3.2. Proiectarea schemei bazei de date

Baze de date de tip document

Bazele de date de tip document sunt clasificate în categoria bazelor de date non-relaționale, cunoscute și sub denumirea generică de NoSQL. În contrast cu bazele de date relaționale tradiționale, care organizează datele în structuri rigide de tip tabelar, utilizând rânduri și coloane fixe, bazele de date document adoptă un model flexibil, bazat pe documente. Acest tip de bază de date reprezintă una dintre cele mai populare alternative la modelul relațional, fiind utilizat frecvent în dezvoltarea de aplicații moderne, distribuite și scalabile (MongoDB, Inc., 2025).

Un document reprezintă unitatea de bază a stocării în cadrul acestor sisteme și poate fi interpretat ca o înregistrare individuală ce conține informații despre un anumit obiect, împreună

cu metadatele aferente. Structura internă a unui document este formată din perechi de tip cheie-valoare, unde valorile pot varia ca tip și complexitate. Astfel, un document poate conține șiruri de caractere, valori numerice, date calendaristice, liste (matrice) sau chiar obiecte imbricate. Din punct de vedere al formatului, documentele sunt de regulă stocate sub forma unor fișiere JSON, BSON (Binary JSON) sau XML, fiecare dintre acestea permițând o exprimare clară și ierarhică a datelor (MongoDB, Inc., 2025).

Documentele sunt grupate în colecții, care funcționează ca entități logice similare tabelelor din bazele de date relaționale. Deși documentele dintr-o colecție partajează, în general, un tipar structural comun, schema acestora nu este impusă în mod rigid. Această caracteristică oferă o schemă flexibilă, permițând variații semnificative între documente, fapt ce facilitează adaptarea modelului de date la cerințe dinamice. Cu toate acestea, unele sisteme de gestiune a bazelor de date de tip document oferă mecanisme opționale de validare a schemei, prin care se poate impune o structură prestabilită atunci când aplicația o solicită (MongoDB, Inc., 2025).

De asemenea, bazele de date document pun la dispoziția utilizatorilor un API specializat sau un limbaj de interogare care permite executarea operațiilor fundamentale de tip CRUD⁷:

- **Crearea** presupune adăugarea unui nou document în colecție, fiecare document fiind asociat cu un identificator unic, care permite referințierea sa ulterioară.
- **Citirea** documentelor se realizează fie prin utilizarea identificatorului unic, fie prin interogări bazate pe valorile câmpurilor. Pentru optimizarea performanței în această etapă, pot fi definiți indecși asupra câmpurilor frecvent utilizate în interogări.
- **Actualizarea** constă în modificarea conținutului unui document existent, fie în întregime, fie parțial, prin suprascrierea anumitor câmpuri.
- **Ștergerea** permite eliminarea unui document din colecție, operație care poate fi efectuată pe baza identificatorului unic sau a unor criterii de filtrare.

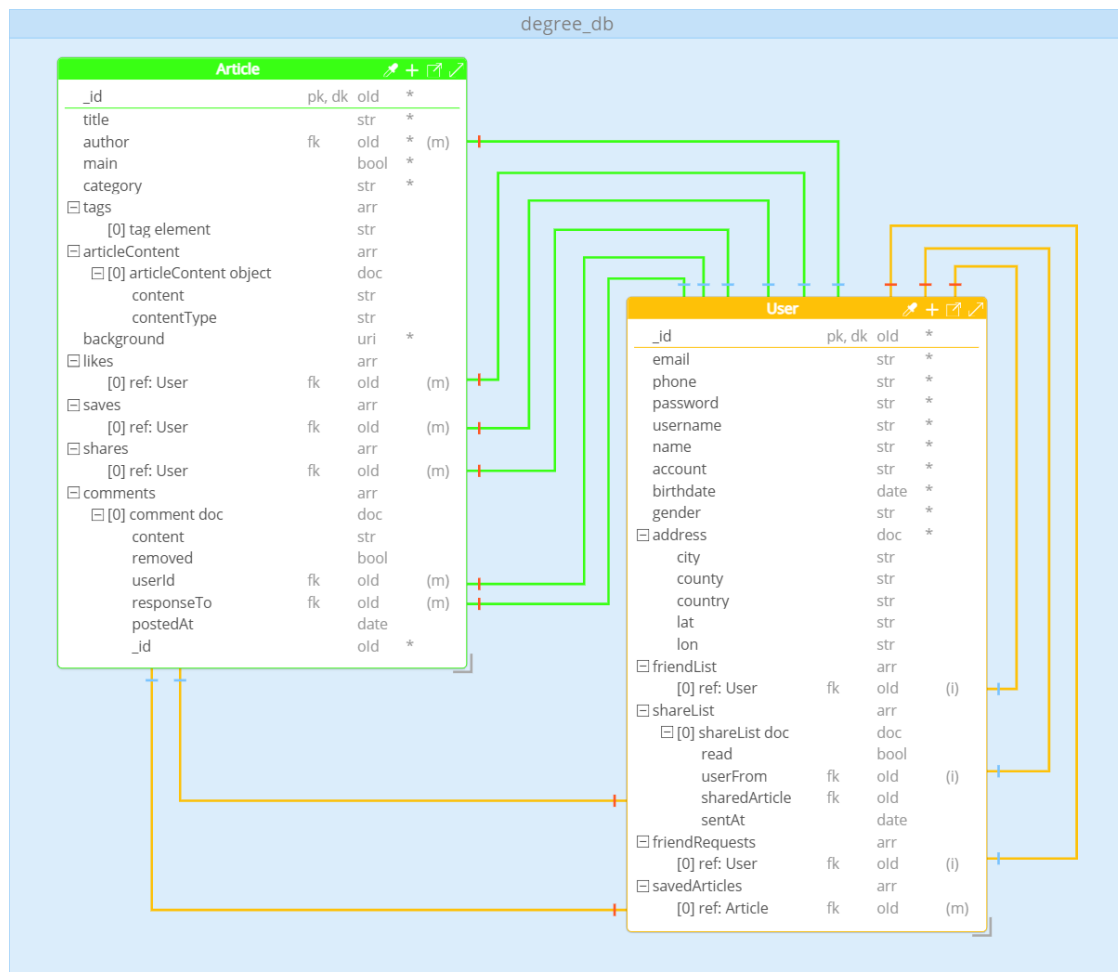
Prin aceste caracteristici, bazele de date de tip document oferă un cadru extrem de flexibil și performant pentru gestionarea datelor semi-structurate, fiind adecvate pentru aplicații cu cerințe dinamice și cu volume de date nestructurate sau variabile, fiind o opțiune ideală pentru aplicația propusă în cadrul lucrării.

⁷ CRUD (Create, Read, Update, Delete) = creare, citire, actualizare și ștergere a documentelor.

Definirea schemei bazei de date

O schemă MongoDB definește structura documentelor dintr-o colecție. Specifică câmpurile, tipurile acestora, regulile de validare, valorile implicite și alte constrângeri. Tipurile de date definite pot include primitive, cum ar fi șiruri și numere, precum și tipuri structurale, cum ar fi obiecte și matrice, care pot fi combinate pentru a crea scheme care reprezintă tipuri de obiecte personalizate. Atlas App Services pune la dispoziție schemele BSON, care extind standardul de schema JSON, pentru a defini modelul de date al aplicației și pentru a oferi posibilitatea validării documentelor ori de câte ori sunt create, modificate sau șterse (MongoDB, Inc., 2024).

Un model este un înveliș în jurul schemei care ne permite să interacționăm cu baza de date MongoDB (operații CRUD precum crearea, citirea, actualizarea, ștergerea), care oferă o interfață pentru interogarea și manipularea datelor (GeeksforGeeks, Sanchhaya Education Private Limited, 2025).



Figură 10 - Diagrama relațiilor dintre entități

Diagrama de relație entitate prezintă schema bazei de date MongoDB, denumită `degree_db`, pentru o soluția informatică dezvoltată, centrată pe gestionarea articolelor și a utilizatorilor. Astfel, aceasta evidențiază două colecții principale: `Article` și `User`, împreună cu relațiile dintre ele.

Diagrama este organizată conform unui model semi-relational, specific bazelor de date NoSQL de tip documentar (cum este MongoDB). Se utilizează referințe (ref) pentru a simula relații între documente, similar relațiilor între tabele în modelul relațional.

Ambele scheme conțin atributul fundamental `_id`, cheia primară a documentului, este de tip `ObjectId` și este autogenerat de baza de date la inserarea unui nou document.

Colecția User

Această colecție modelează informațiile despre utilizatori. Câmpurile esențiale sunt:

- **email, phone, username, name:** Atribute de identificare a unui utilizator.
- **password:** Atribut utilizat pentru autentificare, reținut sub formă de hash
- **account:** Atribut central în gestionarea conturilor. Acesta desemnează rolul fiecărui utilizator, astfel că stabilește privilegiile fiecărui tip de cont, exemplificate în diagrama cazurilor de utilizare.
- **birthdate, gender:** Informații suplimentare de profil.
- **address:** Obiect imbricat, care conține detalii precum orașul, județul, țara, și coordonate geografice (latitudinea și longitudinea).
- **friendList:** Listă de referințe către alți utilizatori, modelând o rețea socială.
- **shareList:** Listă de obiecte care conțin informații legate de distribuirile de articolele între utilizatori. Câmpurile obiectelor sunt:
 - **userFrom:** Referință către utilizatorul care a partajat.
 - **sharedArticle:** Referință către articolul partajat.
 - **read, sentAt:** Stare (dacă a fost sau nu accesat) și data la care a fost trimis.
- **friendRequests:** Cereri de prietenie în curs, reținute sub formă de listă de referințe către utilizatorii care au realizat o cerere către utilizatorul curent.
- **savedArticles:** Listă de articole salvate de utilizator, prin referințe către `Article`.

Colecția Article

Această colecție modelează articolele din platformă. Câmpurile principale includ:

- **title, category, tags:** Atribute de identificare și clasificare. Primele două atribute sunt de tip șir de caractere, spre deosebire de atributul „tags” care reprezintă o listă care reține șiruri de caractere, pentru o mai bună filtrare a conținutului.
- **author:** Reține o referință către documentul „User” către un utilizator al cărui cont este fie de tip „author”, fie de tip „admin”
- **main:** Este de tip boolean și reține dacă un atricol ar trebui afișat în formă principală sau secundară, adică ce tip de card ar trebui să utilizeze.
- **articleContent:** Obiect complex, de tip listă de obiecte, care desemnează structura și conținutul articolului, ordinea elementelor și conține următoarele atribute:
 - **content:** Conținutul propriu-zis al fiecărui element.
 - **contentType:** Tipul elementului (subtitlu, paragraf, imagine). Pentru imagini reține adresele publice generate de clodinary ale respectivelor elemente media
- **background:** Reține un URI către imaginea care va fi afișată în cardul de prezentare și accesare al articolului, generat tot de clodinary
- **likes, saves, shares:** Liste de referințe către utilizatorii care au efectuat acțiunile respective.
- **comments:** Listă de comentarii, fiecare având:
 - **content:** Textul comentariului.
 - **removed:** Stare booleană care indică dacă a fost șters.
 - **userId:** Referință către autorul comentariului.
 - **responseTo:** Referință către alt comentariu (pentru a permite fire de discuții).
 - **postedAt:** Data publicării.

Relații între colecții

Diagrama prezintă multiple relații între User și Article, implementate prin referințe (ref) și liste de obiecte:

- Un articol poate fi apreciat, salvat sau partajat de mai mulți utilizatori.
- Un utilizator poate salva sau partaja mai multe articole.
- Comentariile sunt anonimizate structural în cadrul articolului, dar includ referințe către utilizatorul autor.

S-a optat pentru această abordare pentru a oferi o bună echilibrare între denormalizare⁸ (favorabilă performanței de citire) și utilizarea referințelor (care permit menținerea integrității și flexibilitate în interogări).

Astfel, definirea celor două scheme și a relațiilor dintre acestea oferă bazei de date concepute caracteristici mai avansate, care includ:

- **Model hibrid între „embedded” și „referenced documents”:** Comentariile sunt embedded – stocare direct în interiorul documentului Article, nu într-un document Comments separat – dar cu referințe interne.
- **Design orientat spre social media:** Prezența listei de prieteni, a listei de cereri de prietenie, dar și acțiuni de partajare indică o componentă socială robustă.
- **Scalabilitate și modularitate:** Separarea entităților majore (User, Article) permite o gestionare eficientă a datelor în contexte de volum mare, preferându-se utilizarea frecventă de referințe.

Așadar, schema degree_db este bine structurată pentru o aplicație cu o profundă componentă socială care implică interacțiuni între utilizatori. Designul urmează principiile de modelare NoSQL, cum ar fi utilizarea documentelor imbricate acolo unde are sens (de exemplu: comentarii), dar și menținerea referențelor pentru entități reutilizabile (de exemplu: utilizatorii). Această schemă asigură flexibilitate, performanță și coerență în gestionarea datelor.

⁸ „Denormalizarea este procesul de încercare de a optimiza performanța de citire a unei baze de date prin adăugarea de date redundante sau prin gruparea datelor.” (Elmasri, 2008)

Capitolul 4. Implementarea sistemului informatic

4.1. Tehnologii informatice utilizate

4.1.1. Tehnologiile de bază ale dezvoltării web (HTML, CSS, Javascript)

HTML (HyperText Markup Language) este un limbaj de marcare utilizat pentru a structura conținutul web în cadrul documentelor accesibile prin internet. Acesta definește ierarhia și semantica elementelor de pe o pagină web, permițând browserelor să interpreteze și să afișeze conținutul într-un mod standardizat. (W3C, 2025)

CSS (Cascading Style Sheets) este un limbaj de stilizare conceput pentru a separa prezentarea vizuală a unui document de structura sa și de conținut, permițând definirea unor reguli de stil proprii și reutilizabile, astfel contribuind la menținerea unui cod curat, modular și mult mai ușor de întreținut. Împreună cu Javascript și HTML este un pilon al dezvoltării web moderne. (W3C, 2025)

JavaScript (JS) este un limbaj de programare dinamic, interpretat sau compilat just-in-time, folosit în principal pentru dezvoltarea aplicațiilor web, dar și în alte medii. Acesta susține mai multe paradigme de programare — orientată pe obiect, funcțională și imperativă — și se remarcă prin caracteristici precum funcții de ordin înalt, introspecție a obiectelor și evaluare dinamică a codului. (Mozilla Developer Network, 2025)

JavaScript este în primul rând un limbaj la nivelul clientului, astfel că executarea codului se realizează de către un browser web, precum ar fi Google Chrome sau Firefox, nu de către un server. Limbajul a devenit cu adevărat omniprezent, cu peste 98% dintre site-urile web raportând că au utilizat JavaScript în 2022. Cu toate acestea, de la lansarea în anul 2009 a Node.js, un mediu de server open-source, JavaScript a devenit un limbaj de popular și pe partea de server, depășind limitarea anterioară de a rula doar într-un browser. (Munro, 2025)

4.1.2. Instrumente utilizate

Visual Studio Code - IDE

Pentru dezvoltarea aplicației propuse a fost ales Visual Studio Code (VS Code), un editor de cod sursă modern, dezvoltat de Microsoft și lansat pentru prima dată în aprilie 2015. Acesta este un editor gratuit, open-source, compatibil cu principalele sisteme de operare – Windows, macOS și Linux – și oferă în același timp performanță, extensibilitate și dă dovadă de ușurință în utilizare

Visual Studio Code este construit pe Electron, un framework care permite dezvoltarea de aplicații desktop utilizând tehnologii web precum HTML, CSS și JavaScript. Motorul de redare utilizat este Chromium, iar partea de server este asigurată de Node.js, ceea ce face din VS Code un instrument nativ în ecosistemul JavaScript (Johnson, 2019).

Alegerea acestui editor este justificată de compatibilitatea sa deplină cu tehnologiile utilizate în dezvoltarea soluției informatice propuse, în special biblioteca React pentru front-end și platforma Node.js pentru back-end. Datorită suportului nativ pentru JavaScript, precum și a integrării facile cu Git și cu diverse medii de dezvoltare, VS Code oferă un mediu de lucru coerent și eficient pentru dezvoltatorii web. În plus, editorului pune la dispoziție o gamă vastă de extensii care pot îmbunătăți semnificativ productivitatea, precum ESLint, Prettier, Live Server, sau extensii dedicate React și Node.js. Funcționalitățile precum IntelliSense (completarea automată a codului), debugging-ul integrat și terminalul încorporat contribuie la eficientizarea procesului de dezvoltare și testare a aplicațiilor.

Visual Studio Code se remarcă și prin oferirea unei experiențe personalizate, oferind posibilitatea modificării interfeței și a funcționalităților, prin tematici vizuale sau prin configurarea fișierelor settings.json și keybindings.json. Astfel oferă dezvoltatorilor un grad ridicat de adaptabilitate, ceea ce poate încuraja creativitatea și adoptarea celor mai bune practici în programare și reprezintă în un mediu optim pentru dezvoltarea unei aplicații web moderne (Microsoft , 2025).

Git - DVCS

Git reprezintă un sistem de control al versiunilor distribuite (DVCS – *Distributed Version Control System*), dezvoltat în anul 2005 de către Linus Torvalds, creatorul nucleului Linux. Proiectat inițial pentru a satisface cerințele de performanță și integritate ale comunității Linux, Git a fost conceput ca o alternativă open-source la sistemele existente la acea vreme. Torvalds a declarat că a creat Git pentru că „nu existau soluții disponibile care să îndeplinească cerințele de performanță și integritate ale codului”.

Git a fost dezvoltat într-un timp record, având funcționalități de bază implementate într-o singură săptămână. Sistemul permite gestionarea eficientă a modificărilor codului sursă prin crearea de *ramuri* (branches), facilitând lucrul colaborativ fără a produce conflicte sau pierderi de date. Prin intermediul mecanismului de commit și merge, dezvoltatorii pot adăuga modificări în mod controlat într-un depozit comun, păstrând în același timp un istoric clar al tuturor schimbărilor.

În prezent, Git este cel mai utilizat sistem de control al versiunilor, fiind adoptat atât de dezvoltatori individuali, cât și de companii precum Microsoft, Google sau Netflix. Conform sondajului Stack Overflow din 2023, 93% dintre dezvoltatorii chestionați utilizează Git în activitatea lor profesională.

GitHub

Pe baza acestui sistem, au fost dezvoltate platforme de colaborare precum GitHub, lansat în 2008 și achiziționat ulterior de Microsoft. Este o platformă cloud care oferă instrumente avansate pentru găzduirea, gestionarea și colaborarea pe proiecte software, care a revoluționat dezvoltarea software prin integrarea funcționalităților Git cu servicii suplimentare precum integrarea continuă (CI), livrarea continuă (CD), urmărirea erorilor, gestionarea proiectelor și colaborarea între echipe. (Matthias, 2024)

4.1.3. Frontend – frameworkuri utilizate

React.js

React este o bibliotecă JavaScript utilizată pentru dezvoltarea interfețelor de utilizator (UI), care se remarcă prin trei caracteristici principale care facilitează construirea de aplicații web interactive și scalabile:

- React permite crearea de interfețe interactive într-un mod simplu și eficient, deoarece redă doar componentele necesare atunci când datele se schimbă. Această abordare declarativă face codul mai previzibil, ușor de înțeles și de depanat, contribuind astfel la o dezvoltare mai rapidă și mai sigură.
- React încurajează crearea de componente izolate, reutilizabile, care își gestionează propriile stări, optându-se pentru combinarea acestora pentru a obținerea de interfețe complexe. Deoarece logica componentelor este scrisă în JavaScript, datele pot fi transmise ușor prin aplicație, iar starea aplicației este menținută în „Document Object Model (DOM)”.
- React nu impune restricții asupra tehnologiilor utilizate în cadrul aplicației, astfel că permite dezvoltarea de noi funcționalități fără a fi necesară rescrierea codului existent. De asemenea, React poate fi utilizat atât pentru randarea aplicațiilor pe server, o modalitate fiind utilizarea Node.js, cât și pentru dezvoltarea de aplicații mobile prin React Native.

(Meta Platforms, Inc., 2025)

Bootstrap

Creat inițial la mijlocul anilor 2010 de către Mark Otto și de Jacob Thornton, cu scopul de a avea un cadru de proiectare coerent și uniform în toate aplicațiile interne ale companiei Twitter, Bootstrap a devenit rapid unul dintre cele mai populare framework-uri front-end și proiecte „open source”.

Prima lansare oficială ca proiect deschis a fost pe 19 august 2011, de atunci framework-ul trecând prin peste douăzeci de modificări, inclusiv două rescrieri major, din care au rezultat Bootstrap 2 și 3, unde Bootstrap 2 a adăugat funcționalitate receptivă întregului framework ca foaie de stil opțională, ca ulterior versiunea a 3-a să ofere funcționalitatea receptivă în mod implicit, concentrându-se în special pe aplicațiile mobile.

Versiunea a 4-a a venit cu o migrare către Sass și a trecut la flexbox-ul CSS, proiectul propunându-și participe la dezvoltarea comunității de dezvoltare web prin promovarea unor proprietăți CSS mai noi, cu mai puține dependențe și care să utilizeze tehnologiile noi în pe care le ofereau browserele mai moderne.

Cea mai recentă versiune, Bootstrap 5, se concentrează pe îmbunătățirea bazei de cod a versiunii a 4-a, aducând cât mai puține modificări majore posibile, conform contributorilor la proiect. Versiunea aceasta a adus îmbunătățiri funcțiilor și componentelor existente, browsere mai vechi nu mai primesc suport, s-a renunțat la jQuery și s-au adoptat tehnologii noi, cum ar fi proprietățile personalizate CSS (Mark & Thornton, 2025).

Bootstrap-React

Bootstrap-React este una dintre cele mai vechi biblioteci React și a înlocuiește Bootstrap JavaScript. Biblioteca își propune să rămână compatibilă cu ecosistemul imens de interfață al Bootstrap și oferă mai mult control asupra formei și funcției fiecărei componente utilizând modelul de funcționare a componentelor React, fiecare componentă fiind construită de la zero ca o componentă React separată. Astfel acestea devenind accesibile implicit și conferă o experiență superioară de dezvoltare. (React Bootstrap, 2025)

4.1.4. Backend – mediu de rulare și framework

Node.js

Node.js este un mediu de rulare JavaScript asincron, bazat pe evenimente, destinat pentru construirea aplicațiilor de rețea scalabile, putând să gestioneze concurent mai multe conexiuni, datorită acestui model de rulare, care se distinge de modelele tradiționale de

concurență care folosesc firele de execuție ale sistemului de operare (OS threads). În plus, Node.js rulează pe motorul V8 JavaScript (dezvoltat de Google pentru Chrome) în afara browserului, ceea ce îi conferă o performanță ridicată (OpenJS Foundation, 2025).

Rețelele bazate pe fire de execuție sunt relativ ineficiente și foarte greu de utilizat. Mai mult decât atât, utilizatorii Node.js nu trebuie să se îngrijoreze de posibile blocaje ale procesului (deadlocks), deoarece nu există blocaje (locks). Atunci când Node.js efectuează o operațiune I/O, acesta nu firul de execuție, acțiune care ar duce la irosirea resurselor CPU, ci în schimb așteaptă răspunsul de la operațiune și continuă execuția când la primirea răspunsului. Astfel, Node.js poate gestiona mii de conexiuni simultane cu un singur server, evitând problemele asociate gestionării concurenței pe fire multiple.

Aproape nicio funcție din Node.js nu efectuează operații I/O direct, iar procesul nu va fi blocat decât atunci când operațiile I/O sunt realizate folosind metodele sincronizate din biblioteca standard a Node.js. Datorită acestui fapt, dezvoltarea sistemelor scalabile în Node.js devine o opțiune foarte fezabilă. (OpenJS Foundation, 2025)

Express.js

Express.js este un framework web minimalist și flexibil, construit pentru mediul de execuție Node.js, care facilitează dezvoltarea de aplicații web și a interfețelor de programare a aplicațiilor (API-uri) într-un mod eficient și modular. Creat pentru a simplifica complexitatea dezvoltării pe server cu JavaScript, Express.js oferă noi funcționalități care permit manipularea cererilor HTTP, definirea rutelor, gestionarea răspunsurilor și extinderea funcționalității prin mecanisme de tip middleware. (OpenJS Foundation, 2025)

Funcționând ca o interfață între aplicațiile scrise în JavaScript și protocolul HTTP⁹, Express.js oferă premisele necesare unei structurări clare a logicii aplicației, respectând și bazându-se pe principiile arhitecturale ale REST¹⁰. De asemenea, oferă suport extensiv pentru procesarea cererilor asincrone și integrarea cu baze de date, motoare de template sau alte biblioteci externe. În comparație cu utilizarea directă a modulelor native ale Node.js, Express permite o dezvoltare mai rapidă, mai organizată și scalabilă a aplicațiilor, reprezentând un instrument esențial în ecosistemul de dezvoltare backend al JavaScript, fiind frecvent utilizat în

⁹ Hypertext Transfer Protocol (HTTP) = un protocol la nivelul aplicației utilizat pentru transmiterea documentelor hipermedia. Acest protocol funcționează pe principiul cerere-răspuns (request-response), în care clientul (de obicei browserul) inițiază o cerere, iar serverul returnează un răspuns corespunzător. (Mozilla Developer Network, 2025)

¹⁰ Representational State Transfer = stil arhitectural pentru proiectarea serviciilor web (IBM, 2025)

aplicații web moderne datorită ușurinței de utilizare, flexibilității și compatibilității cu alte tehnologii web contemporane.

4.1.5. MongoDB

MongoDB database – bază de date NoSQL

MongoDB este un sistem de gestiune a bazelor de date NoSQL, orientat pe documente, dezvoltat inițial în 2007 de compania „10gen”, redenumită ulterior „MongoDB ,Inc.”. Față de modelele relaționale tradiționale, MongoDB utilizează o arhitectură flexibilă, bazată pe documente BSON (Binary JSON), permițând astfel stocarea de date semi-structurate într-un format natural pentru aplicațiile moderne, facilă în special aplicațiilor web, forma de stocare a datelor fiind profund similară cu a obiectelor din Javascript. (MongoDB, Inc., 2025)

Modelul de date al MongoDB este unul non-relațional care permite încorporarea de documente complexe în colecții fără a necesita o schemă fixă, oferind dezvoltatorilor o flexibilitate sporită în definirea și extinderea structurii datelor. Fiecare document este un obiect similar cu un formatul unui fișier de tip JSON, ce poate conține tipuri de date diverse, inclusiv liste și subdocumente imbricate. Această abordare face MongoDB o alegere potrivită în mod deosebit pentru aplicația propusă în această lucrare, MongoDB oferind și suport complet pentru operațiunile CRUD (Create, Read, Update, Delete), interogări expresive și agregări complexe, printr-un limbaj propriu de interogare bazat pe documente.

Spre deosebire de bazele de date relaționale, în care datele sunt normalizate și legate prin chei externe, MongoDB favorizează denormalizarea și încorporarea datelor în interiorul documentelor pentru a reduce complexitatea interogărilor și a crește performanța. Această caracteristică contribuie semnificativ la scalabilitatea orizontală, MongoDB fiind proiectat nativ pentru distribuirea datelor pe mai multe noduri. Denormalizarea a fost un principiu care a fost adoptat pe larg în aplicația propusă, gestionând într-un document multiple date și favorizând interogările de mare viteză, integrând de exemplu comentariile aferente unui articol sau id-urile utilizatorilor care au apreciat articolul în schema care definește în sine articolul respectiv.

MongoDB Atlas - Database-as-a-Service (DBaaS)

În plus, platforma poate fi utilizată atât local, cât și în cloud, prin serviciul MongoDB Atlas – o platformă de tip Database-as-a-Service (DBaaS), dezvoltată și oferită de compania MongoDB Inc., care permite gestionarea automată și scalabilă a bazelor de date MongoDB în

cloud. Lansată ca o extensie naturală a sistemului de baze de date NoSQL MongoDB, Atlas permite dezvoltatorilor să creeze, să ruleze și să extindă aplicații moderne fără a fi necesară gestionarea manuală a infrastructurii bazei de date. Platforma este disponibilă în principalele medii cloud – Amazon Web Services (AWS), Google Cloud Platform (GCP) și Microsoft Azure – și oferă un set robust de funcționalități orientate spre securitate, scalabilitate și performanță. (MongoDB, Inc., 2024)

Utilizarea MongoDB Atlas începe prin crearea unui *cluster*, care reprezintă o instanță MongoDB gestionată automat, cu suport pentru replicare, backup și scalare orizontală (*sharding*). Atlas oferă o interfață intuitivă prin care utilizatorii pot configura și monitoriza clusterelor, seta reguli de acces și autentificare.

Precum toate serviciile cloud, Atlas include automatizarea sarcinilor de întreținere, monitorizare în timp real și backup automat. Astfel, timpul dedicat administrării bazei de date este redus semnificativ, iar dezvoltatorii se pot concentra pe logica de business a aplicației. Prin caracterul său complet gestionat, MongoDB Atlas permite scalarea aplicațiilor de la prototipuri la nivel enterprise, fiind utilizat de companii precum eBay, Adobe, și Cisco.

De asemenea, suportul nativ pentru modele de date documentare (de formă *JSON*), a căror structură se poate considera semi-structurată, precum cea a articolelor jurnalistice, dar și a altor elemente ale soluției informatice propuse, fac Atlas și baza de date MongoDB o alegere perfectă pentru dezvoltarea aplicației web care reprezintă obiectul acestei lucrări. (MongoDB, Inc., 2025)

MongoDB Compass - interfață grafică oficială

În cadrul dezvoltării aplicației propuse, MongoDB Compass a fost o aplicație esențială pentru vizualizarea și modificarea datelor. MongoDB Compass este interfața grafică oficială (GUI) dezvoltată de MongoDB Inc. pentru gestionarea vizuală a bazelor de date MongoDB. Concepută pentru a facilita interacțiunea cu bazele de date, Compass oferă dezvoltatorilor, administratorilor și analiștilor un mediu intuitiv în care pot inspecta, vizualiza, interoga și analiza datele, fără a fi necesară utilizarea liniei de comandă.

Principalul avantaj al MongoDB Compass constă în capacitatea sa de a oferi o reprezentare vizuală a structurii și conținutului bazelor de date. Utilizatorii pot examina documente individuale, pot construi interogări complexe folosind un editor grafic și pot vizualiza automat indexurile, performanța acestora și gradul de utilizare. Compass permite, de

asemenea, modificarea structurii documentelor, adăugarea sau ștergerea câmpurilor, gestionarea colecțiilor și a bazelor de date, precum și validarea schemelor – toate printr-o singură interfață prietenoasă. Este un instrument ideal pentru explorarea datelor, mai ales în fazele inițiale de dezvoltare a aplicațiilor sau pentru învățarea modului de lucru cu bazele de date MongoDB.

Un avantaj suplimentar pe care l-a oferit Compass în timpul dezvoltării aplicației web a reprezentat faptul că suportă conexiuni către instanțe găzduite în MongoDB Atlas, serviciu folosit pentru stocarea datelor ale soluției propuse.

4.1.6. Cloudinary – Software-as-a-Service (SaaS)

Cloudinary este o platformă Software-as-a-Service (SaaS) specializată în gestionarea, transformarea și livrarea conținutului vizual, precum imagini și videoclipuri, prin intermediul unei suite de API¹¹-uri scalabile și automatizate. Aceasta oferă o soluție completă pentru întregul ciclu de viață al fișierelor media, incluzând funcționalități de încărcare, transformare, optimizare, livrare și analiză.

Cloudinary este conceput pentru a răspunde cerințelor aplicațiilor moderne web și mobile, în care viteza de livrare a conținutului este un factor esențial pentru experiența utilizatorului. Distribuția conținutului se realizează prin intermediul unei rețele globale de livrare de conținut (CDN), asigurând acces rapid și fiabil din orice locație geografică. De asemenea, Cloudinary oferă un sistem complet de gestionare a fișierelor și metadatelor asociate acestora, alături de funcționalități avansate de securitate, cum ar fi semnăturile de autentificare și controlul accesului la resurse (Cloudinary, 2025).

Pentru soluția informatică dezvoltată în cadrul acestei lucrări s-a optat pentru integrarea platformei Cloudinary cu Node.js, Prin intermediul SDK¹²-ului oficial pentru Node.js, care simplifică interacțiunea cu API-ul platformei și pune la dispoziția o serie de funcționalități esențiale pentru manipularea eficientă a fișierelor media (Cloudinary, 2025).

4.1.7. Utilizarea serviciilor de localizare (Geoapify - Geocoding)

În cadrul aplicației realizate în această lucrare, a fost utilizată platforma **Geoapify**, care oferă o suită extensivă de servicii geospațiale. Geoapify este o platformă modernă concepută

¹¹ API = Application Programming Interface reprezintă un set de definiții de sub-programe, protocoale și unelte pentru programarea de aplicații și software (Wikipedia, 2023)

¹² SDK = Software Development Kit (Pachet de dezvoltare software)

pentru a sprijini dezvoltarea de aplicații bazate pe localizare, furnizând o gamă variată de API-uri, inclusiv pentru geocodare, generare de hărți, rutare și analiză spațială. Acestea sunt adaptate atât nevoilor comerciale, cât și celor academice sau de cercetare (Geoapify, 2025).

Pentru realizarea aplicației, a fost utilizat în mod special Geoapify Geocoding API, un instrument esențial pentru transformarea adreselor poștale în coordonate geografice (latitudine și longitudine) (Geoapify, 2025). Această funcționalitate este indispensabilă pentru integrarea coerentă a datelor spațiale în cadrul aplicațiilor informatice, permițând localizarea exactă a entităților și facilitând analizele geospațiale.

În contextul lucrării de față, serviciul de geocodare a fost utilizat în principal pentru susținerea analizelor statistice asupra utilizatorilor, corelând datele demografice și comportamentale cu profilul geografic al acestora. Astfel, au fost prelucrate informații relevante privind țara, localitatea și județul de proveniență, în scopul identificării unor tipare de comportament și distribuții teritoriale semnificative.

4.1.8. Utilizarea LLM (Large Language Models)

LangChain

LangChain reprezintă o bibliotecă open-source concepută pentru a facilita dezvoltarea de aplicații care utilizează modele de limbaj de mari dimensiuni (LLM – *Large Language Models*). Aceasta oferă o serie de componente modulare și abstractizate ce permit integrarea facilă a modelelor de limbaj în fluxuri de lucru complexe, îmbinând astfel procesarea naturală a limbajului cu surse externe de date, instrumente și logica aplicațiilor. LangChain se bazează pe conceptul de chains – lanțuri de operațiuni – care leagă mai multe apeluri LLM într-o succesiune logică. Aceste lanțuri pot include preprocesarea prompturilor, validarea datelor, interacțiunea cu baze de date sau API-uri, precum și post-procesarea răspunsurilor generate. (LangChain, Inc., 2025)

Implementarea LangChain în limbajul JavaScript conferă o flexibilitate crescută în dezvoltarea aplicațiilor web, aspect ce o recomandă drept o alegere potrivită pentru integrarea unor funcționalități de bază utilizând modele de limbaj de mari dimensiuni (LLM). Printre funcționalitățile implementate în cadrul aplicației dezvoltate se numără generarea automată de etichete relevante, pe baza conținutului unui articol, precum și generarea unui titlu sugestiv adaptat tematicii și stilului textului. Aceste capacități sunt susținute prin utilizarea de

prompturi predefinite, care ghidează modelul în generarea unui răspuns personalizat, adaptat conținutului oferit de utilizator în timpul redactării sau modificării unui articol.

Gemini API - modelul Gemini-2.0-flash

Prelucrarea conținutului textual are loc în componenta client a aplicației, asigurând astfel o interacțiune directă și rapidă între interfața utilizator și serviciile oferite de modelul LLM. Pentru implementarea logicii de generare, a fost utilizat modelul „gemini-2.0-flash”, o variantă optimizată a modelelor de limbaj din seria Gemini, pus la dispoziție prin intermediul Gemini API, dezvoltat de Google DeepMind. Acest model este recunoscut pentru performanța sa în generarea de text coerent și contextualizat, precum și pentru viteza ridicată de răspuns, ceea ce îl face adecvat pentru aplicații interactive în timp real. Integrarea acestuia prin API a permis realizarea unui flux de procesare optimizat, în care LangChain gestionează logica aplicației și prompturile, iar Gemini răspunde cu outputul generat în mod dinamic.

O caracteristică distinctivă a modelului constă în latența redusă a răspunsului, ceea ce îl face ideal pentru aplicații interactive în care experiența utilizatorului depinde de rapiditatea feedback-ului. În plus, modelul păstrează o bună acuratețe în generarea de conținut, având capacitatea de a interpreta prompturi complexe și de a livra răspunsuri adaptate atât contextului textual, cât și intenției utilizatorului. (Google, 2025)

În cadrul lucrării de față, s-a optat pentru o arhitectură hibridă, ce îmbină capacitățile oferite de LangChain cu performanțele modelului Gemini, evidențiază potențialul integrării LLM în aplicații moderne orientate către conținut, sprijinind astfel atât eficiența procesului de creare, cât și calitatea semantică a produsului final.

4.1.9. Render - Platform-as-a-Service (PaaS)

Render este o platformă modernă de tip Platform-as-a-Service (PaaS) destinată dezvoltatorilor, care facilitează implementarea, scalarea și gestionarea aplicațiilor web, serviciilor de fundal și bazelor de date într-un mod automatizat și eficient. Prin integrarea cu sisteme de control al versiunilor precum GitHub, GitLab sau Bitbucket, Render permite actualizări continue ale aplicațiilor, eliminând necesitatea intervențiilor manuale în procesul de implementare (Render, 2025).

Astfel, având în vedere că aplicația web dezvoltată în cadrul acestei lucrări utilizează sistemul de control al versiunilor oferit de GitHub, integrarea cu platforma *Render* reprezintă o soluție tehnică eficientă pentru publicarea aplicației în mediul online. Platforma permite setarea

variabilelor de mediu necesare funcționării serviciilor externe, precum autentificarea, conexiunile la bazele de date sau configurarea serviciilor terțe. În plus, posibilitatea de a defini comenzi personalizate pentru lansarea aplicației a permis construirea și servirea statică a componentei React direct din cadrul serverului Node.js, simplificând astfel procesul de livrare și reducând complexitatea arhitecturii.

Distribuirea aplicației printr-un singur serviciu facilitează gestionarea versiunilor și automatizarea procesului de *deployment*, transformând actualizarea aplicației într-o operațiune rapidă și lipsită de efort suplimentar. Totodată, conexiunea stabilă și eficientă cu serviciile cloud utilizate – *MongoDB Atlas* pentru stocarea datelor și *Cloudinary* pentru gestionarea resurselor media – evidențiază capacitatea platformei Render de a integra aplicații moderne cu infrastructuri distribuite, garantând astfel scalabilitate, securitate și performanță în execuția aplicației.

4.2. Implementarea aplicației

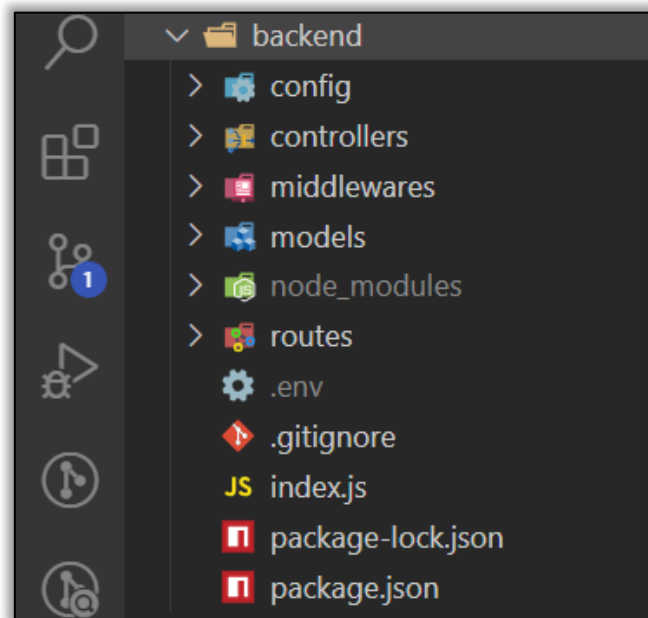
Structura aplicației și gestionarea variabilelor de mediu

Pentru implementarea aplicației s-a adoptat o arhitectură specifică soluțiilor informatice web moderne de tip REST API, bazată pe separarea clară între partea de backend și frontend.

Directorul **backend** conține logica de server și este organizat pe module funcționale, după cum urmează:

- **config/**: Conține fișierele de configurare necesare pentru conectarea aplicației la baza de date MongoDB și la servicii externe precum Cloudinary (pentru gestionarea fișierelor media) și Langchain (pentru procesarea limbajului natural și integrarea AI).
- **models/**: Include definițiile schemelor de date utilizate de MongoDB, construite cu ajutorul bibliotecii Mongoose. Acestea definesc structura documentelor salvate în baza de date.
- **controllers/**: Acest director conține funcțiile de control care implementează operațiunile CRUD prin intermediul Mongoose, acționând ca un intermediar între rutare și modelul de date.
- **routes/**: Definește rutele HTTP ale aplicației, mapând cererile primite de la client către funcțiile corespunzătoare din directorul controllers.

- **.env:** Fișier de configurare care stochează variabile de mediu și chei sensibile, cum ar fi stringul de conexiune la baza de date sau cheile API, pentru a asigura securitatea și flexibilitatea aplicației.
- **index.js:** Punctul de intrare al aplicației backend, în care este configurat și pornit serverul Express, precum și middleware-urile necesare.

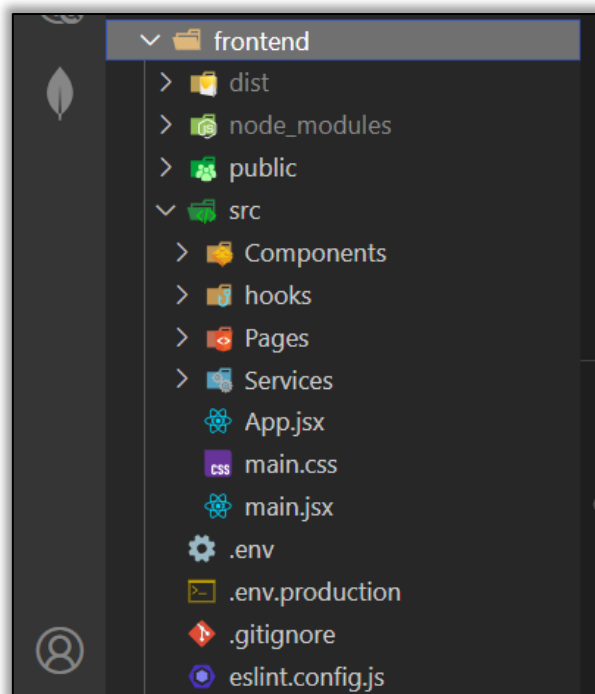


Figură 11 - Structură folder backend

Partea de interfață, dezvoltată cu React, este localizată în directorul **frontend**, care la rândul său conține subdirectorul **src/** – principalul loc al codului sursă. Structura acestuia este următoarea:

- **Components/:** Găzduiește componentele reutilizabile din interfața grafică, fiecare responsabilă pentru o parte specifică a UI-ului (de exemplu: *AuthProvider*, *GeoapifyAutocomplete*, *CommentSection*).
- **Pages/:** Include componentele care reprezintă paginile aplicației, fiecare având în componență anumite componente individuale (de exemplu: *ArticleEditorPage*, *DashboardPage*, *LoginPage*).
- **Services/:** Aici se află funcțiile care realizează cereri HTTP către backend, facilitând comunicarea dintre client și server, organizate în fișierele individuale *articleService* și *userService*.

- **hooks/:** Conține hook-uri personalizate (custom hooks) care extind funcționalitatea React într-un mod reutilizabil și modular.
- **main.jsx** și **App.jsx:** Fișierele de inițializare ale aplicației React, unde sunt definite rutele, layout-ul principal și componentele globale.
- **main.css:** Fișierul de stil global al aplicației.
- **.env.production:** Conține variabilele de mediu utilizate în mediul de producție, inclusiv URL-ul aplicației implementate public.
- **eslint.config.js:** În cadrul frontend-ului, aplicația utilizează Vite ca bundler modern pentru dezvoltare rapidă și optimizări de performanță în producție. Fișierul *eslint.config.js* definește regulile de analiză statică a codului JavaScript și React, contribuind la menținerea unui stil de programare unitar și la identificarea timpurie a posibilelor erori.



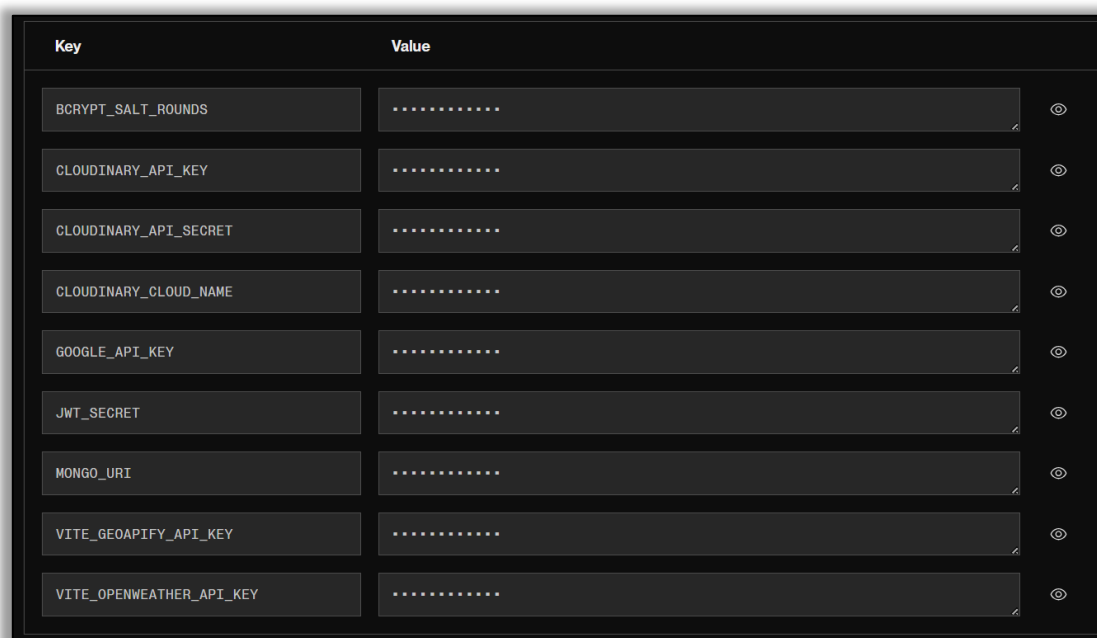
Figură 12 - Structură folder frontend

Această structură modulară și bine organizată permite o dezvoltare scalabilă, clară și ușor de întreținut, facilitând o gestionare eficientă a codului și contribuie la logica pentru delimitarea responsabilităților în cadrul aplicației.

Pe partea de gestionare securizată și eficientă a variabilelor de mediu, aceasta se realizează prin intermediul platformei Render, care depășește rolul de simplă platformă de găzduire, oferind o abordare centralizată a gestionării datelor confidențiale, crucială pentru securitatea aplicației și pentru configurarea flexibilă a diverselor sale medii (dezvoltare, testare, producție).

Render oferă o secțiune dedicată unde variabilele de mediu sunt înregistrate și stocate în siguranță. Prin intermediul acestei secțiuni, dezvoltatorii pot:

- Defini și actualiza variabilele de mediu direct din interfața Render.
- Asigura că aceste variabile sunt disponibile automat pentru aplicație la momentul rulării, fără intervenție manuală la fiecare deployment.
- Controla accesul la informațiile sensibile, îmbunătățind postura de securitate a aplicației.
- Genera chei de acces complexe, pentru îmbunătățirea securizării aplicației (de exemplu pentru *JWT_SECRET*)



Figură 13 - Interfață Render pentru variabile de mediu

Această facilitate contribuie la un deployment mai sigur și mai standardizat, permițând aplicației să acceseze configurațiile necesare într-un mod controlat și scalabil, esențial pentru un mediu de producție robust.

Configurarea bazei de date și definirea schemelor

Configurarea bazei de date este realizată prin intermediul bibliotecii *mongoose*, care facilitează interacțiunea cu o bază de date MongoDB în mod orientat obiect. Conexiunea este stabilită asincron prin apelul funcției *mongoose.connect*, utilizând o variabilă de mediu *MONGO_URI* ce conține stringul¹³ de conexiune către serviciul MongoDB Atlas – o soluție cloud pentru baze de date non-relaționale. Opțiunea *autoIndex* permite generarea automată a indexurilor definite în schemele de date. În cazul unei conexiuni reușite, aplicația confirmă acest lucru printr-un mesaj informativ în consolă, în timp ce eventualele erori sunt tratate prin afișarea mesajului de eroare și oprirea execuției aplicației.

```
import mongoose from 'mongoose';

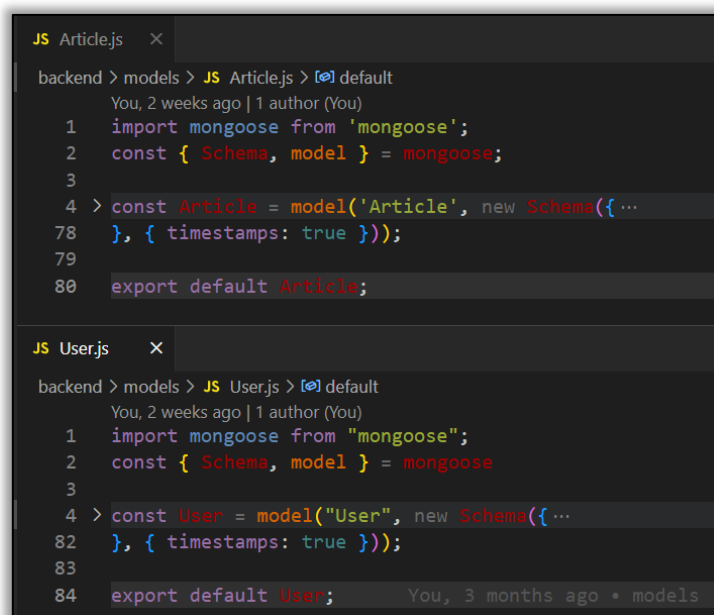
const connectDB = async () => {
  try {
    const conn = await mongoose.connect(process.env.MONGO_URI, {autoIndex: true});
    console.log(`MongoDB Connected: ${conn.connection.host}`);
    return conn;
  } catch (err) {
    console.error(`MongoDB Connection Error: ${err.message}`);
    process.exit(1);
  }
}

export default connectDB;
```

Figură 14 - Conexiune bază de date mongodb

Pentru definirea structurii datelor utilizate în aplicație, s-a utilizat, de asemenea, biblioteca *mongoose*, care oferă un sistem de modelare a obiectelor pentru MongoDB. Atât schema pentru utilizatori (*User*) cât și cea pentru articole (*Article*) au fost create folosind constructorul *Schema* din *mongoose*, facilitând validarea datelor, definirea tipurilor și aplicarea de restricții. Modelele rezultate sunt ulterior utilizate pentru a interacționa direct cu colecțiile corespunzătoare din baza de date, permițând efectuarea operațiilor CRUD într-un mod coerent și tipizat.

¹³ String = string este un tip de date folosit pentru a reprezenta secvențe de caractere, adesea text. Poate fi o constantă literală, o variabilă care conține text sau un tip compus.



```
JS Article.js ×
backend > models > JS Article.js > default
You, 2 weeks ago | 1 author (You)
1 import mongoose from 'mongoose';
2 const { Schema, model } = mongoose;
3
4 > const Article = model('Article', new Schema({ ...
78 }, { timestamps: true }));
79
80 export default Article;

JS User.js ×
backend > models > JS User.js > default
You, 2 weeks ago | 1 author (You)
1 import mongoose from "mongoose";
2 const { Schema, model } = mongoose
3
4 > const User = model("User", new Schema({ ...
82 }, { timestamps: true }));
83
84 export default User; You, 3 months ago • models
```

Figură 15 - Scheme mongoose

Conexiunea cu serviciile externe

Pentru extinderea funcționalităților aplicației, au fost configurate două servicii externe esențiale:

- **Langchain** (pentru procesare de limbaj natural)
- **Cloudinary** (pentru gestionarea fișierelor media).

Integrarea cu platforma *Langchain* se realizează prin pachetul *@langchain/google-genai*, utilizând modelul „*gemini-2.0-flash*” furnizat de către Google. Configurația este încărcată din variabile de mediu, iar cheia de acces *GOOGLE_API_KEY* este utilizată pentru autentificare. Parametrul *temperature* este setat la 0.5, ceea ce determină un comportament echilibrat al modelului față de unul complet determinist sau mult prea creativ, astfel fiind util în generarea de răspunsuri previzibile și controlate, dar cu un limbaj uman, creativ, potrivit activității unei redacții de știri, aspect necesar pentru aplicația dezvoltată în cadrul lucrării.


```
import { ChatGoogleGenerativeAI } from "@langchain/google-genai";
import { config } from "dotenv";

config();
const model = new ChatGoogleGenerativeAI({
  model: "gemini-2.0-flash",
  temperature: 0.5,
  googleApiKey: process.env.GOOGLE_API_KEY,
});

export default model;
```

Figură 16 - Configurare langchain/google-genai

Pentru o mai bună gestionare a fișierelor media, aplicația utilizează serviciul *Cloudinary* în combinație cu biblioteca *multer* și adaptorul *multer-storage-cloudinary*. Configurarea este realizată cu ajutorul variabilelor de mediu care conțin datele de autentificare (numele contului, cheia API și secretul API). Fișierele încărcate sunt salvate într-un folder predefinit („images”), cu limitarea formatelor permise (jpg, jpeg, png) pentru un mai mare control al conținutului media și cu acces public pentru vizualizare, imaginile încărcate fiind parte a articolelor redactate și accesate de public. Funcția *uploadMulter* este utilizată pentru încărcarea multiplă de imagini și este folosită în *endpoint-urile* ¹⁴relevante din aplicație.

```
dotenv.config();

const connectCloudinary = async () => cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});
```

Figură 17 - Configurare Cloudinary

¹⁴ Endpoint = este locația specifică, de obicei o adresă URL, unde un API primește solicitări de date sau funcționalități. Acționează ca punct de intrare pentru aplicațiile client pentru a interacționa cu serverul API (IBM Inc., 2024).

Implementarea serverului

Configurarea serverului a fost realizată prin intermediul framework-ului **Express.js**, unul dintre cele mai utilizate instrumente pentru dezvoltarea aplicațiilor web în ecosistemul **Node.js**. În prima etapă, sunt încărcate variabilele de mediu cu ajutorul pachetului **dotenv**, pentru a asigura accesul securizat la datele sensibile de configurare. Ulterior, sunt stabilite conexiunile asincrone la baza de date MongoDB și la serviciul Cloudinary, prin intermediul funcțiilor **connectDB()** și **connectCloudinary()**.

Serverul este inițializat pe portul specificat în fișierul **.env**, iar în lipsa acestuia, pe portul implicit **5000**. Configurația middleware-urilor include:

- **cookieParser()** – pentru gestionarea cookie-urilor în cadrul cererilor HTTP;
- **express.urlencoded()** și **express.json()** – pentru parsarea datelor trimise prin formulare sau în format JSON;
- **cors()** – configurat cu opțiunile din obiectul **corsOptions**, care definește comportamentul Cross-Origin Resource Sharing. Acesta controlează accesul resurselor backend de către aplicații client din alte domenii, în acest caz fiind permise doar originile: aplicația publicată pe Render (<https://newswebapp-qvqp.onrender.com>) și mediul de dezvoltare local (<http://localhost:5173>).

```
const app = express();

app.use(cookieParser());
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

const corsOptions = {
  origin: ["https://newswebapp-qvqp.onrender.com", "http://localhost:5173"],
  methods: ["GET", "POST", "PUT", "PATCH", "DELETE"],
  credentials: true,
  allowedHeaders: ["Content-Type"]
};

app.use(cors(corsOptions));
```

Figură 18 - Server: configurare middleware

Printre opțiunile CORS se numără și:

- **methods** – specifică metodele HTTP permise (GET, POST, PUT, PATCH, DELETE);
- **credentials** – setat „true” permite trimiterea de credențiale (cookie-uri, anteturi de autentificare etc.) în cererile cross-origin;

- **allowedHeaders** – definește anteturile permise, în acest caz Content-Type, esențial pentru trimiterea datelor în format JSON sau formulare.

Această configurație este importantă pentru asigurarea securității aplicației, permițând doar accesul controlat din partea aplicațiilor client de încredere.

Pentru a asigura o structură clară și modulară a aplicației, rutele backend sunt organizate pe baza principiilor REST și sunt asociate entităților principale: utilizatori, articole, fișiere media și interacțiuni cu modelul de inteligență artificială. Fiecare set de rute este gestionat de un router dedicat, definit în fișiere separate și importat în serverul principal. Rutele dedicate folosesc, de asemenea, metode dedicate din folderul controllers.

```
app.use('/article-api', articlesRouter);
app.use('/user-api', usersRouter);
app.use('/cloudinary-api', cloudinarydRouter);
app.use('/langchain-api', langchainRouter);
```

Figură 19 - Server: Routere express dedicate

Pentru servirea interfeței frontend generate de React (build-ul Vite), serverul folosește `express.static()` pentru a expune directorul „dist”, iar toate cererile neidentificate explicit prin rute sunt redirecționate către `index.html`, pentru a permite funcționarea routing-ului de tip client-side. Această abordare permite integrarea completă a aplicației într-un singur server, atât pentru API cât și pentru interfața web.

```
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

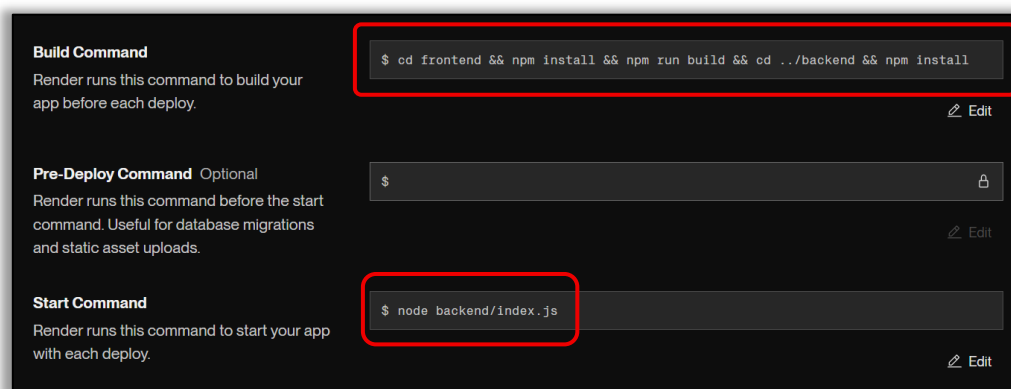
app.use(express.static(path.join(__dirname, "../frontend/dist")));

app.get("*", (req, res) => {
  res.sendFile(path.join(__dirname, "../frontend/dist/index.html"));
});
```

Figură 20 - Server: servirea interfeței React SPA

În plus, la nivelul platformei de găzduire **Render**, configurația pentru procesul de build și de pornire a aplicației integrează atât frontend-ul, cât și backend-ul într-o singură aplicație. Comanda de build parcurge pașii de instalare a dependențelor și generare a build-ului pentru

frontend, apoi instalează dependențele backend-ului. Comanda de start pornește aplicația Node.js, utilizând fișierul principal de configurare al serverului.



Figură 21 - Render: comenzi pentru deployment

Mecanisme de autentificare, înregistrare și autorizare

Crearea unui nou utilizator în aplicație implică un proces de prelucrare securizată a datelor de autentificare, urmat de stocarea acestora în baza de date. La apelarea rutei *POST /user*, serverul primește un obiect JSON ce conține datele de înregistrare ale utilizatorului, precum și o parolă setată de acesta, transmisă în format clar (plain text).

Pentru a proteja aceste informații sensibile, funcția `createUser` utilizează pachetul **bcrypt** pentru criptarea parolei. Procesul presupune generarea de *salt criptografic*¹⁵, urmată de *hash-uirea*¹⁶ parolei cu acest salt. În cadrul procesului de creare a unui utilizator nou, securitatea parolei este consolidată prin utilizarea unui număr variabil de runde de criptare, configurabil printr-o variabilă de mediu **BCRYPT_SALT_ROUNDS** sau cu valoarea implicită 10, permițând ajustarea nivelului de complexitate al procesului de hash-uire fără a modifica direct codul sursă, ceea ce contribuie la flexibilitatea și întreținerea pe termen lung a aplicației. În noul obiect creat care desemnează noul utilizator, parola primită în format clar este înlocuită cu parola criptată înainte ca noul utilizator să fie salvat în baza de date MongoDB.

¹⁵ În criptografie, „sarea”(salt) sunt date aleatorii adăugate la o parolă înainte de a fi hash-uită. Acest lucru face imposibil ca un atacator să obțină parole din hash-urile lor folosind tabele de parole precalculate și hash-urile corespunzătoare (Mozilla Developer Network, 2025).

¹⁶ Hashing = este procesul de transformare a oricărei chei date sau a unui șir de caractere într-o altă valoare. Aceasta este de obicei reprezentată de o valoare sau o cheie mai scurtă, cu lungime fixă, care reprezintă și ușurează găsirea sau folosirea șirului original. Funcția hash generează noi valori conform unui algoritm de hash matematic, cunoscut ca valoare hash sau pur și simplu hash (Yasar, What is hashing?, 2024).

Autentificarea utilizatorilor în cadrul aplicației este implementată prin intermediul unui mecanism bazat pe *JSON Web Tokens (JWT)*, care asigură un mod securizat de gestionare a sesiunilor fără a stoca informații de autentificare pe server.

La apelarea rutei *POST/login*, serverul primește din partea clientului un nume de utilizator și o parolă, care sunt validate prin funcția asincronă *loginUser*. Aceasta caută un utilizator în baza de date MongoDB, iar în cazul în care acesta este găsit, parola transmisă este comparată cu versiunea criptată stocată, utilizând algoritmul *bcrypt*. Dacă autentificarea este reușită, se generează un token *JWT* (cu o valabilitate de o oră) ce conține datele utilizatorului excluzând parola, semnat cu cheia secretă stocată în variabila de mediu *JWT_SECRET*.

```
const loginUser = async (username, password, forRefresh = false) => {
  try {
    const user = await User.findOne({ username }).exec();
    if (!user) return { error: true, message: "User doesn't exist" };

    // Compare password with hashed version
    if (!forRefresh) {
      const validPassword = await bcrypt.compare(password, user.password);
      if (!validPassword) return { error: true, message: "Invalid password" };
    }
    else if (password !== user.password) return { error: true, message: "Invalid password" };

    const userObj = user.toObject();
    delete userObj.password;

    // Generate JWT token
    const token = jwt.sign(
      { user: userObj }, process.env.JWT_SECRET, { expiresIn: "1h" }
    );

    if (!token) return { error: true, message: "Error generating token" };
    return { token };
  } catch (err) {
    console.error("Error with login:", err);
    return { error: true, message: "Internal Server Error" };
  }
};
```

Figură 22 - Funcția *loginUser*

Tokenul este trimis clientului sub forma unui cookie, care pentru a spori securitatea este setat cu opțiunile:

- **httpOnly** – care este setat drept „true” (adevărat), însemnând că împiedică accesul scripturilor din browser la cookie, reducând riscul de atacuri XSS;
- **sameSite** – setat drept "Lax", însemnând că trimiterea cookie-ului este permisă doar în cadrul unor cereri de navigare normale (evitând trimiterea în cereri cross-site nesigure);
- **maxAge** – stabilește durata de viață a tokenului la o oră (60 x 60 x 1000 secunde).

```

usersRouter.route('/login').post(async (req, res) => {
  const result = await loginUser(req.body.username, req.body.password);

  if (result.error) {
    return res.status(400).json({ message: result.message });
  }

  res.cookie("token", result.token, {
    httpOnly: true,
    sameSite: "Lax",
    maxAge: 60 * 60 * 1000
  });

  return res.status(200).json(result);
})

```

Figură 23 - Server: cookies

În cazul în care numele de utilizator nu este găsit, parola este incorectă sau apare o eroare în generarea tokenului, serverul returnează mesaje de eroare corespunzătoare. Această abordare asigură o autentificare sigură și scalabilă, specifică pentru aplicații cu arhitectură REST.

De asemenea, pentru a separa procesul de înregistrare și cel de autentificare, în vederea respectării principiilor de bună practică în proiectarea aplicațiilor web moderne, autentificarea nu este realizată automat, ci este necesară o cerere separată de logare pentru generarea și trimiterea tokenului JWT.

Pentru protejarea rutelor care necesită autentificare, aplicația utilizează un middleware de autorizare, denumit *authMiddleware*, bazat pe token-ul generat la autentificare. Acesta are rolul de a verifica prezența și validitatea unui token transmis de client prin intermediul cookie-urilor HTTP, token stocat în variabila *req.cookies.token*. De asemenea, aplicația permite primirea token-ului și prin intermediul antetului (*headers*) utilizând schema de autentificare „*Bearer authentication*”¹⁷.

¹⁷ Bearer Authorization este o schemă de autentificare HTTP utilizată în mod obișnuit cu OAuth 2.0. În această abordare, clientul include un jeton de acces în antetul „Autorizare” utilizând schema „Purtător”, acordând permisiunea de a accesa resursele protejate. Serverul validează jetonul pentru autorizare. Este o metodă utilizată pe scară largă pentru securizarea accesului API, în special în scenariile care implică aplicații terțe (Apidog, Inc., 2024).

```
const authMiddleware = (req, res, next) => {
  const token = req.headers.authorization?.startsWith("Bearer ") ?
    req.headers.authorization.split(" ")[1] : req.cookies.token;
  if (!token) return res.status(403).json({ message: "Token required" });

  jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
    if (err) return res.status(403).json({ message: "Invalid token" });
    req.user = decoded.user;
    next();
  });
};
```

Figură 24 - funcție middleware pentru autentificare

Dacă tokenul este absent, serverul răspunde cu codul de stare HTTP 403 „Forbidden”, semnalând că accesul la resursa respectivă este restricționat. În caz contrar, tokenul este verificat cu ajutorul metodei *jwt.verify*, folosind cheia secretă stocată în variabila de mediu *JWT_SECRET*. În situația în care tokenul este invalid sau expirat, se returnează același cod de eroare. Dacă validarea are succes, payload-ul tokenului (datele salvate de în token sub formă criptată) este atașat obiectului *req.user*, permițând astfel accesul controlat la informațiile utilizatorului în cadrul rutelor protejate. Această tehnică are rolul de a asigura un nivel ridicat de securitate în cadrul soluției informatice, permițând filtrarea cererilor neautorizate și menținerea confidențialității datelor utilizatorilor autentificați.

Pe lângă validarea autentificării, middleware-ul contribuie și la controlul accesului pe baza rolului utilizatorului, ceea ce permite implementarea unei politici de autorizare mai complexe. Un exemplu relevant este ruta „/author/:id” care utilizează obiectul *req.user* pentru a identifica dacă utilizatorul care realizează cererea pe ruta respectivă este într-adevăr autor al redacției sau administrator, verificând valoarea atributului „account”.

```

usersRouter.route('/author/:id').get(async (req, res) => {
  const result = await getUserById(req.params.id)
  if(result.account !== "author" && result.account !== "admin"){
    return res.status(400).json({ message: "User is not an author" });
  }

  if (result.error) {
    return res.status(400).json({ message: result.message });
  }

  return res.status(200).json(result.name);
})

```

Figură 25 - Autentificare RBAC

Dacă acesta nu este setat ca "author" sau "admin", cererea este respinsă cu un răspuns HTTP de tip 400 Bad Request, însoțit de un mesaj informativ. Astfel, doar utilizatorii care dețin privilegiile respective de acces pot utiliza anumite funcționalități, cum ar fi editarea sau publicarea de conținut într-un articol jurnalistic.

Această abordare contribuie la securitatea aplicației, separarea clară a responsabilităților și prevenirea accesului neautorizat la funcționalități considerate sensibile, toate acestea în conformitate cu principiile de tip „Role-Based Access Control” (RBAC¹⁸).

Gestionarea fișierelor media pe infrastructură cloud

Aplicația web prezintă integrează un mecanism robust de gestionare a resurselor media, prin utilizarea platformei Cloudinary pentru stocarea conținutului media și a bazei de date Cloud Atlas pentru gestionarea referințelor la acest conținut. În cadrul schemei Article din baza de date Cloud Atlas, sunt persistate adresele publice (URL-uri) ale imaginilor încărcate pe Cloudinary. Aceste adrese sunt stocate în elementul *articleContent* atunci când *contentType* este definit ca „Image”, precum și în atributul background pentru imaginile utilizate în cadrul cardurilor de acces către articole

¹⁸ Controlul accesului bazat pe roluri (RBAC) este un model pentru autorizarea accesului utilizatorului final la sisteme, aplicații și date pe baza rolului predefinit al utilizatorului. De exemplu, un analist de securitate poate configura un firewall, dar nu poate vizualiza datele clienților, în timp ce un reprezentant de vânzări poate vedea conturile clienților, dar nu poate atinge setările paravanului de protecție. Într-un sistem RBAC, un administrator atribuie fiecărui utilizator individual unul sau mai multe roluri. Fiecare rol nou reprezintă un set de permisiuni sau privilegii pentru utilizator (Lindemulder & Kosinski, 2024).

Publicarea resurselor media în cadrul aplicației este gestionată printr-o integrare eficientă a librăriei *Multer* cu serviciul *Cloudinary*. Această abordare permite un control granular asupra stocării și accesibilității imaginilor.

```
const storage = new CloudinaryStorage({
  cloudinary,
  params: {
    folder: 'images',
    allowed_formats: ['jpg', 'jpeg', 'png'],
    resource_type: 'image',
    access_mode: 'public',
  }
});

const uploadMulter = multer({ storage }).array("images");
```

Figură 26 - configurare cloudinary storage și middleware multer

Astfel, se creează un obiect de stocare, *CloudinaryStorage*, care este instanțiat și configurat pentru a interacționa cu serviciul Cloudinary. Parametrii de configurare definesc modul în care imaginile sunt procesate și stocate:

- **folder: 'images':** Specifică directorul de pe Cloudinary unde vor fi încărcate toate imaginile. Aceasta contribuie la organizarea resurselor în cadrul platformei.
- **allowed_formats: ['jpg', 'jpeg', 'png']:** Restricționează tipurile de fișiere acceptate pentru încărcare, permițând doar formate de imagine comune, cum ar fi JPEG și PNG.
- **resource_type: 'image':** Definește tipul resursei care va fi încărcată, asigurând că Cloudinary tratează fișierele ca imagini.
- **access_mode: 'public':** Setează modul de accesibilitate al imaginilor încărcate ca fiind public, permițând accesul la acestea prin URL-uri generate.

Obiectul de stocare configurat este apoi utilizat pentru a inițializa middleware-ul *Multer*, în care se specifică faptul că aplicația va gestiona încărcarea de fișiere multiple sub. Această configurație permite procesarea eficientă a mai multor imagini simultan, simplificând fluxul de lucru pentru încărcarea conținutului.

Suplimentar, aplicația implementează un sistem de optimizare a utilizării spațiului de stocare. La accesarea panoului de administrare „*Dashboard*”, aplicația inițiază o interogare a bazei de date pentru a prelua toate adresele publice stocate. Ulterior, aceste adrese sunt transmise către API-ul Cloudinary, care realizează o validare a existenței fizice a resurselor. În urma acestei validări, o metodă dedicată este apelată pentru a elimina din Cloudinary toate imaginile care nu mai sunt referențiate în baza de date, contribuind astfel la eficientizarea consumului de memorie și la menținerea integrității datelor.

Procesul inițiază cu o interogare a colecției *Article*, selecționând exclusiv câmpurile care participă la stocarea adreselor imaginilor utilizate și la identificarea articolelor (*_id*, *articleContent* și *background*). Această abordare selectivă optimizează performanța, limitând volumul de date recuperate la informațiile strict necesare pentru identificarea URL-urilor.

```
const getAllImageUrls = async () => {
  try {
    const allArticles = await Article.find({}).select("_id articleContent background");

    const contentImageUrls = allArticles.flatMap((article) =>
      article.articleContent
        .filter((contentBlock) => contentBlock.contentType === "Image")
        .map((contentBlock) => contentBlock.content)
    );

    const backgroundUrls = allArticles.flatMap((article) => article.background);

    const imageUrls = [...contentImageUrls, ...backgroundUrls];

    return imageUrls
  } catch (err) {
    console.error(`getAllImageUrls Error: ${err.message}`);
    return { error: true, message: "Internal Server Error" };
  }
};
```

Figură 27 - Funcție pentru obținerea de url-urilor

Astfel, imaginile integrate în conținutul articolelor sunt obținute prin filtrarea blocurilor de conținut unde *contentType* este specificat ca de tip "Image", iar imaginile de fundal utilizate pentru cardurile de acces ale articolelor din pagina principală sunt obținute prin parcurgerea atributului *background* al fiecărui articol, extrăgându-se adresele URL aferente pentru ambele cazuri. Aceste liste de URL-uri sunt apoi agregate într-o singură colecție, iar orice anomalii apărute în timpul execuției sunt gestionate prin mecanisme de captură a excepțiilor, asigurând robustețea operației.

```
const getUnusedImagesPublicIds = async (uploadedImageUrls = []) => {
  const result = await cloudinary.search
    .expression(`folder:images`)
    .sort_by('public_id')
    .max_results(100)
    .execute();

  const allPublicIds = result.resources.map(resource => resource.public_id);

  if (uploadedImageUrls.length === 0) {
    return { unusedPublicIds: allPublicIds };
  }

  const uploadedIds = new Set(uploadedImageUrls.map(url => extractPublicId(url)));

  const unusedPublicIds = allPublicIds.filter(id => !uploadedIds.has(id));
  return { unusedPublicIds };
};
```

Figură 28 - Funcție pentru obținerea id-urilor publice nefolosite

Funcția *getUnusedImagesPublicIds* are rolul de a identifica resursele media neutilizate în cadrul sistemului. Aceasta primește ca parametru o listă de adrese URL (*uploadedImageUrls*) care conține adresele imaginilor stocate în baza de date MongoDB. Procesul inițial implică o interogare către serviciul Cloudinary, unde se solicită un număr maxim de 100 de identificatori publici ai imaginilor stocate în directorul „*images*”. Această operațiune are ca rezultat o listă a tuturor identificatorilor publici din Cloudinary.

Dacă lista *uploadedImageUrls* conține adrese, funcția extrage identificatorii publici din fiecare URL și îi stochează într-un set de date pentru o căutare eficientă. În cele din urmă, se realizează o filtrare a identificatorilor publici din Cloudinary, reținându-se doar aceia care nu se regăsesc în setul de identificatori proveniți din baza de date. În situația în care lista de adrese URL din baza de date (*uploadedImageUrls*) este goală, se consideră că toate imaginile prezente în Cloudinary sunt neutilizate, iar funcția va returna toți identificatorii publici obținuți anterior.

Această abordare permite identificarea precisă a imaginilor care sunt stocate în Cloudinary, dar care nu mai au o referință validă în baza de date, facilitând astfel procesul de optimizare și curățare a resurselor printr-un mecanism autonom.

În final, ștergerea se realizează folosind metodele oferite de API-ul de la cloudinary (*cloudinary.api.delete_resources(urls)*) în mod eficient și sigur.

Utilizarea LLM în gestionarea conținutului

Utilizarea modelului lingvistic de mari dimensiuni (LLM) Gemini-2.0-Flash, integrat prin intermediul framework-ului LangChain, a fost un element central în procesul de creare a articolelor și de gestionare a firelor de discuție din paginile articolelor.

Unul dintre beneficiile majore ale acestei integrări este capacitatea de a genera etichete (tag-uri) pe baza conținutului textual al unui articol. Această funcționalitate este implementată printr-o funcție dedicată, care preia textul articolului prelucrat în frontend și îl transmite API-ului LangChain. Funcția utilizează un șablon de sistem care instruește LLM să genereze între 5 și 10 etichete relevante, returnate ca o listă de cuvinte separate prin virgulă. Textul articolului este apoi inserat într-un șablon de prompt predefinit, care este ulterior invocat pentru a obține un răspuns de la modelul Gemini-2.0-Flash. Pe baza aceluiași mecanism se obțin și titluri adecvate conținutului redactat în cadrul unui articol pregătit pentru publicare.

```
const generateTags = async (articleText) => {
  const systemTemplate = `You are an assistant that generates 5 to 10 tags for the following article content,
  return only the words separated by a comma (,)`;

  const promptTemplate = ChatPromptTemplate.fromMessages([
    ["system", systemTemplate],
    ["user", "{article}"],
  ]);

  const promptValue = await promptTemplate.invoke({ article: articleText });
  const response = await model.invoke(promptValue);

  return response.content;
}
```

Figură 29 - Funcție generare tag-uri cu LLM

Un mecanism esențial pentru menținerea unui mediu online sigur și respectuos este filtrarea comentariilor cu conținut nepotrivit. Această funcționalitate utilizează capacitățile unui model lingvistic de mari dimensiuni (LLM) pentru a identifica și izola contribuțiile care încalcă standardele de decență.

API-ul article este responsabil pentru interogarea comentariilor din baza de date. Pentru a optimiza performanța și a gestiona volume mari de date, implementează un mecanism de paginare. Aceasta interoghează colecția de articole, selectând doar câmpurile relevante pentru comentarii. Comentariile sunt apoi iterate, extrase din fiecare articol și consolidate într-o listă unică. Fiecare intrare include identificatorii unici ai articolului și comentariului, ID-ul utilizatorului, conținutul textual și data creării. Pentru o prezentare coerentă, lista finală este sortată cronologic.

Mecanismul de filtrarea a comentariilor cu conținut nepotrivit, utilizează capabilitățile unui model lingvistic de mari dimensiuni (LLM) pentru a identifica și izola postările de comentarii care încalcă standardele de comunicare adecvată.

```
const getInappropriateComments = async (commentList) => {
  const systemTemplate = `You are an assistant that evaluates if the content
of a comment is inappropriate for a list of comments.
Return only the numeric id of the inappropriate comments.
the response should be in this format: x,y,z, ... where x, y, z are of ObjectId from mongodb format`;

  const promptTemplate = ChatPromptTemplate.fromMessages([
    ["system", systemTemplate],
    ["user", "{comments}"],
  ]);
```

Figură 30 - Funcție filtrare comentarii cu conținut nepotrivit cu LLM

Astfel, funcția *getInappropriateComments* primește o listă de comentarii și are rolul de a determina care dintre acestea conțin elemente inadecvate. Procesul implică interacțiunea cu un LLM, căruia îi este furnizat un șablon de sistem ce îl instruește să acționeze ca un asistent de evaluare a conținutului. Scopul său este de a returna exclusiv **identificatorii numerici (de tip ObjectId MongoDB)** ai comentariilor considerate nepotrivite, sub un format prestabilit de tip csv¹⁹.

Pentru a gestiona eficient volume mari de comentarii, funcția procesează lista de intrare în segmente de câte 10 comentarii. Fiecare segment este concatenat într-un șir de caractere, care include atât ID-ul, cât și conținutul fiecărui comentariu, apoi este trimis către LLM printr-un șablonul de prompt. Răspunsul modelului, care conține ID-urile comentariilor nepotrivite, este parcurs, iar ID-urile extrase sunt adăugate unei liste finale de identificatori. Acest proces iterativ asigură că toate comentariile sunt evaluate, indiferent de lungimea inițială a listei.

¹⁹ CSV = Comma separated value (valori separate prin virgulă)

```

let commentListSegment = "";
let invokeAtTen = 0;
let finalResponse = [];
for(const comment of commentList){
  commentListSegment += comment._id + ") " + comment.content + "\n";
  invokeAtTen++;
  if(invokeAtTen === 10){
    const promptValue = await promptTemplate.invoke({ comments: commentListSegment.trim() });
    const response = await model.invoke(promptValue);

    const ids = response.content.split(",").map(id => id.trim()).filter(id => id !== "");

    finalResponse.push(...ids);

    commentListSegment = "";
    invokeAtTen = 0;
  }
}

```

Figură 31 - Algoritm generare comentarii pe baza LLM

La final, funcția filtrează lista originală de comentarii, reținând doar acelea ale căror ID-uri se regăsesc în lista de răspunsuri generate de LLM. Astfel, sunt identificate și gestionate eficient comentariile care necesită intervenție, contribuind la menținerea calității conținutului generat de utilizatori. Pe baza acelei lista, administratorul poate elimina

Utilizarea LLM în analiza datelor și implementarea de strategii de business

Pe partea de analiză a datelor, aplicația implementează un mecanism avansat de analiză, esențial pentru a înțelege comportamentul utilizatorilor și a optimiza elementele de social media integrate. Acest proces se desfășoară prin intermediul unei interfețe dedicate pentru modelul lingvistic de mari dimensiuni, cu scopul de a genera analize descriptive, adaptate diverselor tipuri de vizualizări grafice, presetate, utilizate de aplicație.

Fluxul de analiză începe odată ce o cerere este inițiată către un endpoint specific de rutare, care primește atât datele necesare analizei, cât și parametrii ce definesc tipul de grafic dorit și interacțiunea socială ce urmează a fi investigată.

Un aspect crucial al acestui proces este validarea riguroasă a datelor de intrare. Sistemul verifică temeinic prezența și validitatea datelor, a tipurilor de interacțiune (cum ar fi aprecieri, salvări, distribuiri, comentarii sau o combinație a acestora) și a tipurilor de grafice solicitate (ploturi de dispersie, bare, radar sau pie, utilizând grafice oferite de pachetul react *nivo*). O regulă particulară impune ca, în cazul analizei tuturor tipurilor de interacțiune simultan, doar graficul de tip "radar" să fie permis, având în vedere capacitatea sa superioară de a reprezenta

multidimensional datele. Dacă se solicită o analiză "all" a interacțiunilor, sistemul convertește automat această cerință într-o listă explicită a tuturor tipurilor de interacțiuni considerate. Această convenție are rolul de a asigura o realizarea de cereri HTTP uniforme și stricte.

Odată ce validările sunt finalizate, datele sunt pregătite pentru procesare, fiind transformate într-un format JSON structurat. Acest set de date este apoi transmis unui șablonului de sistem predefinit, care configurează LLM-ul să funcționeze ca un inginer de date specializat. Acest șablon oferă context detaliat modelului, informându-l despre tipul specific de grafic, contextul aplicației (o agenție de știri care experimentează cu elemente sociale) și interacțiunea analizată. Pentru anumite tipuri de grafice, cum ar fi ploturile de dispersie, șablonul clarifică și semnificația axelor, ghidând LLM-ul să interpreteze corect relațiile dintre variabile.

```
const analyzeDataForChartType = async (chartType, data, interaction = "likes, shares, comments") => {
  const dataJson = JSON.stringify(data, null, 2);
  const mention = chartType === "scatter-plot" ? `The x is referring to the number of friends a user has,
  while the y is referring to the count of ${interaction} a user has done.` : "";
  const systemTemplate = `You are a data engineer that analyzes data coming from a Nivo chart (the chart type is ${chartType}).
  The context is that you analyze data for a web application for a news agency which is testing social media elements in their app.
  The social media element you analyze is: ${interaction}.
  The data is in JSON format.
  ${mention}
  Give me only the analysis, limit the text to 600 characters, but at least 350.
  By analysis I accept also simple observation of the data.
  If you can give a suggestion/recomandation to the news agency.
  the text should not have any * for bold or any other formatting`;

  const promptTemplate = ChatPromptTemplate.fromMessages([
    ["system", systemTemplate],
    ["user", `${dataJson}`],
  ]);

  const promptValue = await promptTemplate.invoke({ dataJson });
  const response = await model.invoke(promptValue);

  return response.content;
}
```

Figură 32 - Funcție cu prompt pentru analiză grafice cu LLM

Modelul este instruit să genereze o analiză concisă și relevantă, limitată la un număr specific de caractere și fără formatare adiționale. Această analiză poate include observații simple asupra datelor și, dacă este posibil, sugestii sau recomandări practice pentru agenția de știri. În cele din urmă, răspunsul generat de LLM, reprezentând analiza datelor, este returnat, oferind informații acționabile pentru optimizarea strategiei de social media.

Agregarea datelor de interacțiune ale utilizatorilor pentru vizualizări grafice și analize LLM

Pentru a facilita analize detaliate ale comportamentului utilizatorilor, sistemul implementează un mecanism sofisticat de agregare a datelor, capabil să extragă informații relevante despre interacțiunile acestora cu articolele. Această abordare permite obținerea unor seturi de date consolidate, care ulterior pot fi filtrate și analizate conform cerințelor specifice ale vizualizărilor dorite.

Pentru a facilita analize detaliate ale comportamentului utilizatorilor, sistemul implementează un mecanism sofisticat de **agregare a datelor**, capabil să extragă informații relevante despre interacțiunile acestora cu articolele. Această abordare permite obținerea unor seturi de date consolidate, care ulterior pot fi filtrate și analizate conform cerințelor specifice ale vizualizărilor dorite.

Procesul este realizat de o funcție din API-ul pentru articole, care interacționează cu cel pentru utilizatori, funcție care este responsabilă pentru construirea și executarea unei *pipeline de agregare MongoDB*. Această funcție primește un parametru, care poate reprezenta una dintre valorile: *saves*, *likes*, *shares*, *comments*, care dictează tipul de interacțiune analizat.

Sistemul definește o mapare internă pentru a asocia fiecare tip de interacțiune cu calea sa corespunzătoare în structura bazei de date. Ulterior, pipeline-ul de agregare este construit dinamic. Pentru interacțiunile precum salvări, aprecieri sau distribuiri, datele sunt descompuse (*unwind*) direct din câmpul corespunzător. În cazul comentariilor, pipeline-ul gestionează în mod specific structura acestora, descompunând colecția de comentarii și extrăgând *userId*-ul fiecărui comentariu. Acest pas are rolul de a lega interacțiunile din cadrul articolelor de utilizatorii care le-au realizat.

Următorul stadiu al pipeline-ului implică o operațiune de căutare (*\$lookup*), care realizează o joncțiune între datele de interacțiune agregate și colecția „*users*”, în vederea atașării informațiilor despre utilizatori, anume vârsta, genul, adresa și numărul de prieteni, împreună cu categoria articolului asupra căruia s-a realizat interacțiunea. În faza finală de proiecție (*\$project*), sunt selectate și transformate doar câmpurile relevante pentru analiză, menționate anterior.


```

pipeline.push(
{
  $lookup: {
    from: 'users',
    localField: 'userId',
    foreignField: '_id',
    as: 'user'
  }
},
{ $unwind: '$user' },
{
  $project: {
    id: '$_id',
    article_category: '$category',
    user_age: {
      $dateDiff: {
        startDate: '$user.birthdate',
        endDate: '$$NOW',
        unit: 'year'
      }
    },
    user_gender: '$user.gender',
    user_address: '$user.address',
    friend_count: { $size: '$user.friendList' },
    _id: 0
  }
}
);

```

Figură 33 - Pipeline agregare date

Prin executarea acestui pipeline de agregare, sistemul obține un set de date curat și structurat, gata pentru a fi utilizat în diverse analize și vizualizări. Orice eroare ce apare pe parcursul acestui proces este gestionată, semnalând problemele interne ale serverului. Acest mecanism avansat de obținere a datelor agregate stă la baza capacității aplicației de a oferi o înțelegere profundă a angajamentului utilizatorilor.

După etapa de obținere și agregare a datelor brute, sistemul inițiază un proces distinct de pregătire și structurare a acestor date, adaptat fiecărui tip de vizualizare grafică. Această etapă este crucială pentru ca datele să poată fi reprezentate corect și eficient pe diferite tipuri de diagrame.

Pentru diagramele de tip **bar**, procesul de pregătire a datelor se realizează prin gruparea interacțiunilor în funcție de criterii predefinite, cum ar fi vârsta, genul sau o combinație a acestora. Datele brute de interacțiune, care conțin informații despre utilizator (vârstă, gen) și articole (categorie), sunt transformate într-un format potrivit pentru reprezentarea vizuală.

Interacțiunile sunt agregate în funcție de criteriul de filtrare selectat, acesta fiind vârsta, genul sau o combinație a acestora. În cazul selectării criteriului vârstă sau filtrării combinate, sistemul definește categorii de vârstă și grupează utilizatorii în funcție de grupa de vârstă corespunzătoare, în caz contrar utilizează direct informațiile legate de gen, acestea definind deja categorii. După selectarea criteriului de filtrare și tip de interacțiune, sistemul număra câte

interacțiuni de un anumit tip au avut loc pentru fiecare grupă de utilizatori, pentru fiecare categorie de articol.

Acest mecanism dinamic de pregătire a datelor garantează că fiecare grafic primește informațiile într-un format optim, permițând vizualizarea clară a tendințelor și a distribuției interacțiunilor utilizatorilor în funcție de diversele attribute demografice și contextuale. Odată pregătite, aceste date sunt apoi utilizate pentru a genera vizualizarea grafică și, ulterior, pentru a alimenta procesul de analiză asistată de LLM.

Similar, diagrama de tip **pie** necesită o prelucrare specifică a datelor agregate pentru a le reprezenta corect proporțiile. Acest proces se concentrează pe gruparea interacțiunilor utilizatorilor în funcție de categorii demografice: vârsta sau genul. Formatarea se realizează pe baza aceluiași mecanism ca pentru diagrama de tip bar, exceptând filtrarea pe categorii de articole și combinarea criteriilor de filtrare.

În continuarea procesului de vizualizare a datelor, pentru diagrama de tip **scatter plot**, structurare specifică a datelor agregate are rolul de a realiza reprezentarea corectă a relațiilor dintre variabile. Acest mecanism de pregătire se concentrează pe gruparea numărului de interacțiuni ale unui utilizator, în funcție de categoria articolului, cu numărul de prieteni ai utilizatorului.

```
useEffect(() => {
  const grouped = {};
  for (const item of rawData) {
    const category = item.article_category;
    const friendCount = item.friend_count ?? 0;
    if (!grouped[category]) grouped[category] = {};
    if (!grouped[category][friendCount]) grouped[category][friendCount] = 0;
    grouped[category][friendCount] += 1;
  }
  const formatted = Object.entries(grouped).map(([category, pointsObj]) => ({
    id: category,
    data: Object.entries(pointsObj).map(([friendCount, count]) => ({
      x: Number(friendCount),
      y: count
    }))
  }));
  setData(formatted);
}, [rawData]);
```

Figură 34 - *useEffect* formatare date pentru grafice

Diagramele de tip **radar** sunt alese pentru a vizualiza performanța multidimensională a articolelor, permițând o comparație simultană a tuturor tipurilor de interacțiuni de tip social media din aplicație pentru fiecare categorie.

Procesul de pregătire a datelor pentru o diagramă radar începe prin primirea seturilor de date brute pentru fiecare tip de interacțiune: aprecieri, salvări, distribuiri și comentarii. Sistemul utilizează setul predefinit de categorii de articole, pentru fiecare, sistemul realizând o contorizare a numărului de interacțiuni de fiecare tip. Acest lucru se realizează prin filtrarea datelor brute pentru fiecare tip de interacțiune, reținând doar înregistrările care corespund categoriei de articol curente. Dacă o categorie nu înregistrează nicio interacțiune de un anumit tip, valoarea corespunzătoare este setată la zero.

```
useEffect(()=>{
  const categorySet = new Set(['politics', 'extern', 'finance', 'sports', 'tech', 'lifestyle']);

  const dataTemp = Array.from(categorySet).map(category => ({
    category,
    likes: rawDataLikes.filter(d => d.article_category === category).length || 0,
    saves: rawDataSaves.filter(d => d.article_category === category).length || 0,
    shares: rawDataShares.filter(d => d.article_category === category).length || 0,
    comments: rawDataComments.filter(d => d.article_category === category).length || 0
  }));

  setData(dataTemp);
}, [rawDataLikes, rawDataSaves, rawDataShares, rawDataComments]);
```

Figură 35 - useEffect combinare date pentru grafic de tip Radar

Rezultatul final este o structură de date adaptată pentru graficul radar, unde fiecare obiect reprezintă o categorie de articol, iar atributele sale sunt numărul total de aprecieri, salvări, distribuiri și comentarii pentru acea categorie. Această reprezentare permite vizualizarea rapidă a modului în care diferite categorii de articole performează în raport cu multiple tipuri de interacțiuni sociale, oferind informații valoroase pentru strategia de conținut.

După ce datele sunt agregate și pregătite pentru fiecare tip de vizualizare, ele sunt transmise către un modulul de analiză pe bază de model lingvistic de mari dimensiuni (LLM). Pe baza interpretării datelor, LLM-ul generează o analiză textuală concisă, care depășește simpla descriere a datelor, oferind observații sintetice și (când este posibil) sugestii practice pentru agenția de știri.

Prin această integrare, sistemul transformă datele vizuale în informații strategice, permițând redacției să ia decizii informate pentru a optimiza performanța elementelor de social media și a conținutului publicat.

Mecanisme de gestionare a listei de prieteni

Sistemul implementează un mecanism robust pentru gestionarea cererilor de prietenie, esențial pentru funcționalitățile de rețea socială. Procesul este definit de o funcția asincronă, care inițiază o serie de validări stricte înainte de a înregistra o nouă cerere. Mecanisme similare sunt prezente și în funcțiile care se ocupă de adăugarea sau ștergerea unui utilizator din lista de prieteni a altui utilizator.

```
if (id === friend) {
  return { error: true, message: "This is your account" };
}

const queryKey = method === "id" ? "_id" : "username";
const userFriend = await User.findOne({ [queryKey]: friend }).select('friendRequests');

if (!userFriend) {
  return { error: true, show: true, message: "Friend not found, wrong id or username!" };
}

const isAlreadyFriend = user.friendList?.some(fId => fId.toString() === userFriend._id.toString());
if (isAlreadyFriend) {
  return { error: true, show: true, message: "You are already friend with this user!" };
}

const requestAlreadySent = userFriend.friendRequests?.some(reqId => reqId.toString() === user._id.toString());
if (requestAlreadySent) {
  return { error: true, show: true, message: "Friend request already sent!" };
}
```

Figură 36 - Validări cereri de prietenie

Înainte de a procesa o cerere de prietenie, sistemul efectuează o serie de validări necesare pentru asigurarea integrității datelor și esențiale pentru implementarea logicii operațiunii.

Astfel, inițial se verifică validitatea identificadorului utilizatorului căruia i se trimite cererea, în funcție de alegerea utilizatorului care efectuează cererea – prin *id* sau prin *username*. Dacă metoda de identificare este prin ID, se asigură că acesta respectă formatul ObjectId specific MongoDB.

Ulterior, sistemul localizează în baza de date utilizatorul care a inițiat cererea, extrăgând câmpurile *_id* și *friendList*. Dacă utilizatorul curent nu este găsit, operațiunea este oprită, semnalând o eroare, iar prevenirea autodirecționării cererilor se realizează prin verificări suplimentare. Astfel, dacă utilizatorul încearcă să trimită o cerere de prietenie către propriul cont, această acțiune este blocată, prevenind scenarii ilogice.

În continuare, sistemul încearcă să identifice utilizatorul căruia i se adresează cererea, fie după *id*, fie după *username*, apoi se verifică dacă cei doi utilizatori sunt deja prieteni, prin

examinarea listei de prieteni a utilizatorului care inițiază cererea. Dacă o relație de prietenie existentă este detectată, cererea este respinsă.

În final, sistemul verifică dacă o cerere de prietenie de la utilizatorul curent către utilizatorul vizat a fost deja trimisă, cu scopul de a preveni înregistrarea cererilor multiple și inutile procesului.

După parcurgerea cu succes a tuturor validărilor, sistemul procedează la înregistrarea cererii de prietenie. Aceasta se realizează prin actualizarea documentului utilizatorului vizat, adăugând id-ul utilizatorului inițiator în câmpul *friendRequests*. De asemenea, utilizarea operatorului *\$addToSet* asigură că id-ul este adăugat doar dacă nu este deja prezent, prevenind duplicatele la nivel de bază de date.

În cazul oricărei erori neașteptate pe parcursul acestui proces, un mesaj de eroare internă este returnat. Acest flux asigură o gestionare eficientă și sigură a cererilor de prietenie în cadrul aplicației, iar în cazul operațiilor similare de acceptare a unei cereri de prietenie sau a ștergerii unui utilizator din lista de prieteni, se procedează în aceeași manieră.

Implementarea secțiunii de comentarii

La nivelul bazei de date, fiecare articol include un câmp *comments*, care reprezintă o colecție de obiecte, care definesc lista de comentarii pentru fiecare pagină unde un articol este postat. Din raționamente de eficientizare a stocării, fiecare comentariu reține id-ul comentariului căruia îi răspunde, urmând ulterior a se construi o listă de arbori pe baza acestor id-uri, necesară construirii secțiunii de comentarii, care este realizată în manieră recursivă. În plus, pentru eficientizarea interogării bazei de date lista s-a optat pentru definirea listei de comentarii drept un câmp al fiecărui articol și nu o colecție separată, care ar necesita căutări suplimentare.

Transformare a listei liniare de comentarii într-o structură de tip listă de arbori necesită o maparea inițială a comentariilor. Astfel, toate comentariile recuperate sunt transformate într-o *mapă*, unde cheia este ID-ul comentariului. Fiecare intrare în această mapă include datele originale ale comentariului, la care se adaugă un câmp *replies*, inițial vid. Această mapă servește drept structură intermediară, facilitând accesul rapid la comentarii după ID-ul lor.

```
useEffect(() => {
  if (!commentList || commentList.length === 0) return;

  const tempMap = {};
  commentList.forEach((comment) => {
    tempMap[comment._id] = { ...comment, replies: [] };
  });

  setCommentMap(tempMap);
}, [commentList]);
```

Figură 37 - creare mapă comentarii

În plus, această structură este utilizată pentru a identifica mai ușor dacă un comentariu este o frunză a viitorului arbore, prin simpla verificare a lungimii listei de răspunsuri, fiind zero în cazul în care un comentariu este frunză (*commentMap[comment._id]?.replies.length === 0*).

Ulterior, construcția listei de structuri arborescente se realizează prin iterarea prin mapa de comentarii, unde pentru fiecare comentariu se verifică dacă acesta reprezintă un răspuns la un alt comentariu, realizându-se două verificări importante:

1. Dacă un comentariu are un răspuns și comentariul părinte există în mapă, rezultă în adăugarea comentariului curent la lista *replies* a comentariului părinte, din mapă.
2. Dacă un comentariu nu are un răspuns (este un comentariu principal) sau comentariul părinte nu este găsit, rezultă în adăugarea comentariului respectiv în lista de arbori, drept una dintre rădăcini.

```
useEffect(() => {
  const tree = [];

  Object.values(commentMap).forEach((comment) => {
    if (comment.responseTo && commentMap[comment.responseTo]) {
      commentMap[comment.responseTo].replies.push(comment);
    } else {
      tree.push(comment);
    }
  });

  setCommentTree(tree);
}, [commentMap]);
```

Figură 38 - Creare listă de arbori cu comentariile

Odată ce structura arborescentă a comentariilor este construită, sistemul utilizează un mecanism recursiv pentru a afișa această ierarhie în interfața utilizatorului. Acest lucru permite o vizualizare intuitivă a conversațiilor, cu răspunsurile imbricate sub comentariile la care se referă.

Funcția *createCommentSection* responsabilă de generarea dinamică a structurii vizuale a comentariilor, operează pe principiul recursivității, parcurgând nodurile arborelui de comentarii și redând fiecare element. Pentru fiecare comentariu (nod), funcția generează un bloc vizual, iar utilizarea unei componente de tip "Collapse" permite expansiunea și restrângerea dinamică a secțiunilor de răspunsuri. Această funcționalitate, controlată de o mapă (openMap) care reține starea de deschidere/închidere a fiecărui comentariu, îmbunătățește lizibilitatea și navigarea, mai ales în cazul discuțiilor extinse.

Construirea secțiunii ține cont de două aspecte:

- Dacă un comentariu are răspunsuri (`comment.replies.length > 0`), funcția *createCommentSection* este apelată recursiv pentru a reda aceste răspunsuri, pe un nivel mai înalt („*depth*” - care este utilizat și pentru a ajusta identarea fiecărui nivel al ierarhiei, accentuând structura de tip arbore).
- Dacă un comentariu nu are răspunsuri, se merge mai departe cu următorul nod de la nivelul 0, adică următoarea rădăcină de arbore.

```
const createCommentSection = (nodes, depth = 0) => {
  if (!nodes || nodes.length === 0) return null;
  return nodes.map((comment) => {
    const nodeId = comment._id;
    return (
      <div key={nodeId}>
        <Row> ...
        </Row>
        <Row> ...
        </Row>
        <Collapse in={openMap[nodeId] || false}>
          <div id={nodeId} className="mt-2">
            {comment.replies.length > 0 && createCommentSection(comment.replies, depth + 1)}
          </div>
        </Collapse>
      </div>
    );
  });
};
```

Figură 39 - Funcție recursivă: secțiune comentarii

În plus, sistemul integrează un mecanism de curățare automată a comentariilor marcate ca „removed” (eliminate), care nu mai au răspunsuri asociate, constituind frunze ale arborelui conversațional. Comentariile intermediare sunt păstrate în mod deliberat, întrucât menținerea continuității și coerenței firului discuției reprezintă o prioritate în gestionarea conținutului conversațional.

Prin acest mecanism, sistemul asigură o reprezentare vizuală clară și interactivă a discuțiilor, unde utilizatorii pot explora cu ușurință comentariile principale și răspunsurile aferente, contribuind la o experiență de utilizare fluidă și organizată, dar și la eficientizarea stocării și interogărilor în baza de date.

Implementarea serviciilor de localizare

Aplicația integrează servicii de localizare în interfața clientului, utilizând API-ul *Geocoding Autocomplete* furnizat de către serviciul *Geoapify*. Acest serviciu facilitează introducerea adreselor de către utilizatori, oferind sugestii predictive pe măsură ce aceștia tastează, îmbunătățind semnificativ experiența utilizatorului prin validarea și simplificarea procesului de introducere a datelor geografice.

```
const handleInputChange = async (value) => {
  setQuery(value);
  if (skipFetchRef.current) { skipFetchRef.current = false; return; }
  if (value.length < 3) { setSuggestions([]); return; }
  try {
    const res = await fetch(
      `https://api.geoapify.com/v1/geocode/autocomplete?text=${encodeURIComponent(
        value
      )}&filter=countrycode:ro&type=city&lang=ro&limit=5&apiKey=${import.meta.env.VITE_GEOAPIFY_API_KEY}`
    );
    const data = await res.json();
    setSuggestions(data.features || []);
  } catch (error) {
    console.error("Geoapify error:", error);
    setSuggestions([]);
  }
};
```

Figură 40 - Implementare api geocode autocomplete

Interogarea se realizează pe baza unei valori introdusă de către utilizator, care este actualizată intern pentru a reflecta interogarea curentă, prin intermediul parametrului *value*, iar prevenirea trimerii redundante de solicitări se realizează printr-un mecanism intern, folosind hook-ul *useRef* al React (prin intermediul variabilei *skipFetchRef*) pentru a controla momentul declanșării cererilor către API. De asemenea, pentru a optimiza utilizarea resurselor API și a evita cererile inutile, sugestiile sunt generate doar dacă lungimea textului introdus depășește

două caractere. Sub acest prag, lista de sugestii este resetată. La îndeplinirea condițiilor prealabile, sistemul inițiază o cerere asincronă către API-ul Geoapify Geocoding Autocomplete.

Acest mecanism fluid și eficient de autocompletare contribuie la o experiență de utilizare superioară, permițând utilizatorilor să introducă rapid și precis informații de localizare, îmbunătățind accesibilitatea și acuratețea datelor colectate.

Mecanismul de creare și editare a articolelor

Soluția informatică oferă un modul dedicat și interactiv pentru crearea și editarea articolelor, conceput pentru a oferi redactorilor un control granular asupra structurii și conținutului. Acest modul facilitează atât compoziția articolelor noi, cât și modificarea celor existente, printr-o interfață vizuală intuitivă.

Editorul de articole este construit ca o pagină web dinamică, care gestionează starea internă a articolului în curs de editare. Atributele principale ale unui articol sunt menținute în starea aplicației, permițând actualizări în timp real. Conținutul articolului este gestionat în două componente în cadrul editorului:

- Starea *articleContent* care reprezintă o colecție ordonată de blocuri de conținut, fiecare având un tip ("Image", "h2", "p") și conținutul efectiv aferent. De asemenea,
- Starea *articleImages* care reține, tot în manieră ordonată, imaginile încărcate, care urmează a fi încărcate utilizând serviciul cloudinary în pagina de publicare.

Tipul de conținut ce urmează a fi adăugat este organizat printr-un sistem de tab-uri, care permite selectarea rapidă a tipului de conținut, fiecare având o componentă dedicată care facilitează introducerea și configurarea sa.

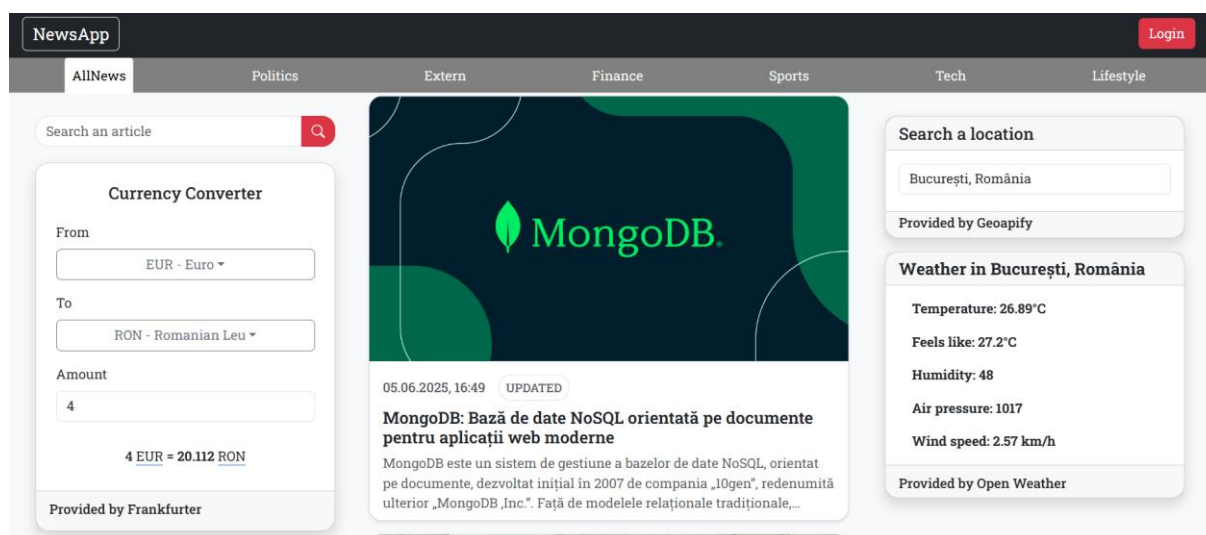
Odată adăugate, componentele articolului vor putea fi manipulate în pagina de previzualizare, unde un redactor le poate schimba ordinea, conținutul sau a le elimina efectiv din componența articolului aferent. Prin aceste mecanisme, aplicația oferă un ecosistem complet și ergonomic pentru crearea și gestionarea conținutului jurnalistic, optimizând fluxul de lucru al redactorilor.

4.3. Prezentarea aplicației

Aplicația web se deschide cu pagina principală, care oferă utilizatorului, care nu necesită momentan un cont, o perspectivă cronologică asupra tuturor articolelor publicate. Acestea sunt afișate sub forma a două tipuri de carduri de accesare, încărcate progresiv în seturi de câte 20,

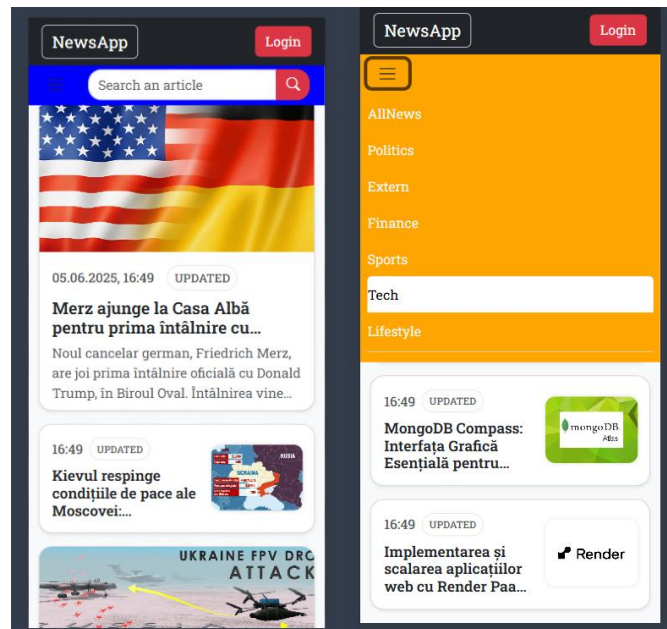
pe măsură ce utilizatorul navighează în josul paginii, până la epuizarea conținutului disponibil, utilizând un feed cu încărcare continuă precum o rețea de socializare.

Pagina principală integrează, de asemenea, un sistem de filtrare tematică bazat pe șase categorii distincte, care permit rafinarea conținutului afișat. Selectarea unei categorii determină și modificarea culorii barei de navigare, oferind utilizatorului un indiciu vizual clar asupra contextului selectat. În plus, este disponibilă o bară de căutare ce utilizează etichetele asociate articolelor pentru a facilita identificarea rapidă a conținutului relevant.



Figură 41 – Homepage: Desktop

În versiunea pentru desktop, sunt integrate două formulare: primul permite interogarea cursului valutar pentru diverse monede, prin intermediul API-ului oferit de serviciul Frankfurter, iar al doilea furnizează informații meteo, utilizând funcționalitatea Geocode Autocomplete oferită de Geoapify pentru identificarea locației și API-ul OpenWeather pentru obținerea condițiilor meteorologice actuale. Aceste două componente interactive sunt eliminate automat în versiunea destinată dispozitivelor mobile, pentru a optimiza experiența de utilizare.



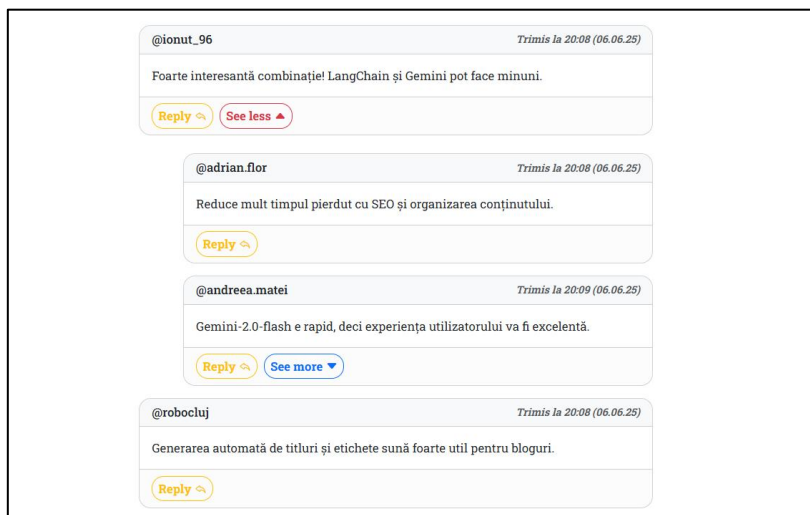
Figură 42 - Homepage: Mobile

Utilizatorul are posibilitatea de a accesa fiecare articol în parte, prezentat într-o structură clasică, specifică formatului editorial: titlu principal, subtitluri, paragrafe de conținut și imagini ilustrative. La finalul articolului, sunt disponibile mai multe elemente interactive care încurajează implicarea utilizatorului: un buton pentru exprimarea aprecierii (like), însoțit de un contor numeric, un buton de salvare a articolului, precum și un buton de distribuire (share), dezactivate însă pentru utilizatorii anonimi (neautentificați). De asemenea, sunt afișate etichetele asociate articolului, care pot fi accesate pentru a vizualiza alte materiale ce împărtășesc aceleași teme sau subiecte.



Figură 43 - Pagină articol

Pagina articolului include și un formular dedicat adăugării de comentarii, oferind utilizatorului posibilitatea de a interacționa cu conținutul în mod activ. Totodată, este disponibilă o secțiune care afișează comentariile existente, în ordinea publicării. Participarea la firul discuției este, de asemenea, disponibilă doar utilizatorilor autentificați, iar administratorul poate elimina comentariile considerate a nu respecta politicile redacției. În plus, un utilizator primește feedback vizual prin intermediul unui modal când încearcă să interacționeze cu elemente la care nu are acces.



Figură 44 - Secțiune comentarii

Pentru a beneficia de funcționalitățile complete ale aplicației, utilizatorul are la dispoziție o pagină dedicată autentificării (Login), iar în cazul în care nu deține un cont, acesta poate comuta rapid către pagina de înregistrare (Sign Up), unde are posibilitatea de a-și crea un cont personal prin completarea unui formular specializat. Formularul de înregistrare solicită utilizatorului completarea unui set de date personale esențiale pentru crearea contului. Acestea includ: numele, prenumele, adresa de domiciliu (selectată cu ajutorul funcționalității Geocode Autocomplete pentru o identificare precisă a locației), data nașterii, genul, adresa de e-mail și o parolă. Sistemul validează în timp real completarea corectă a câmpurilor, oferind feedback vizual pentru eventualele omisiuni sau pentru parolele care nu respectă standardele de securitate impuse.

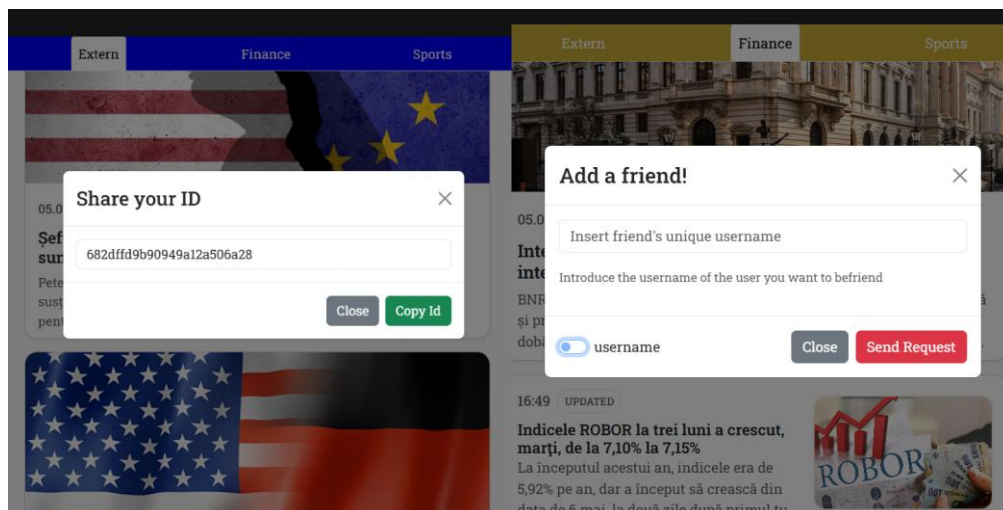
Figură 45 - Login/SignUp

Odată autentificat, utilizatorul beneficiază de acces complet la funcționalitățile disponibile pentru un cont standard. Acestea includ posibilitatea de a distribui articole către o listă de prieteni, precum și de a adăuga comentarii în secțiunile dedicate fiecărui articol. Astfel, autentificarea nu doar extinde interacțiunea utilizatorului cu platforma, ci contribuie și la consolidarea unei comunități active și implicate în jurul conținutului publicat.

Figură 46 - Share & Comm

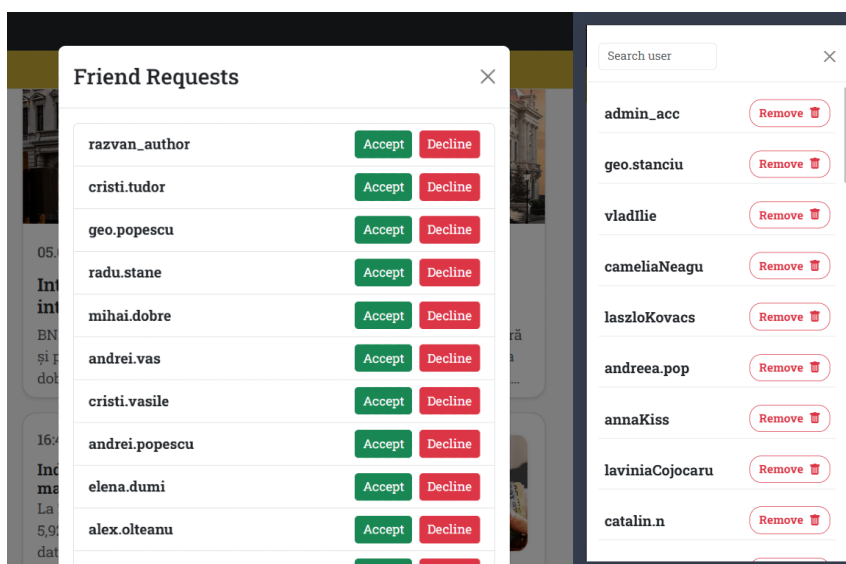
În ceea ce privește interacțiunea cu alți utilizatori, aplicația oferă un sistem de gestionare a prietenilor, menit să faciliteze conexiunile între membri. Utilizatorul își poate partaja propriul identificator unic (ID) prin copierea acestuia în clipboard, ceea ce permite transmiterea facilă

către alte persoane. De asemenea, are posibilitatea de a trimite cereri de prietenie fie prin introducerea ID-ului destinatarului, fie utilizând numele de utilizator asociat contului respectiv.



Figură 47 - Share ID & Add a friend

Utilizatorul autentificat are, de asemenea, posibilitatea de a gestiona în mod activ relațiile de prietenie deja stabilite. Prin intermediul barei laterale de navigare, acesta poate accesa o interfață dedicată, unde are opțiunea de a accepta cereri de prietenie în așteptare sau de a elimina prieteni existenți din lista personală. În plus, aplicația oferă un sistem de notificări vizuale în timp real, care semnalează utilizatorului evenimente relevante precum primirea unei noi cereri de prietenie sau distribuirea unui articol de către un prieten din listă. Aceste notificări contribuie la creșterea gradului de interactivitate și la menținerea unei comunicări eficiente între utilizatori.



Figură 48 - Listă prieteni & listă cereri

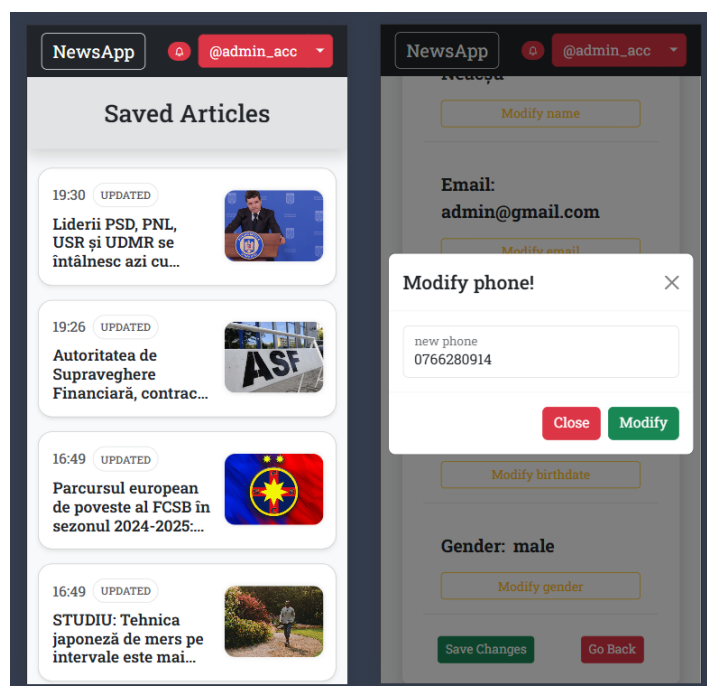
Articolele distribuite de către prieteni sunt afișate într-o secțiune dedicată, sub forma unor carduri interactive. Fiecare card conține informații esențiale privind conținutul transmis: numele utilizatorului care a realizat distribuirea, titlul articolului, precum și data și ora la care distribuirea a avut loc. De asemenea, sistemul indică explicit dacă articolul respectiv a fost deja accesat de destinatar, facilitând o gestionare mai eficientă a conținutului nou.

În situația în care un articol distribuit este ulterior eliminat din platformă, acest fapt este reflectat automat în cardul de distribuire, prin actualizarea conținutului vizual și textual aferent. Astfel, aplicația menține coerența informațională și evită afișarea de conținut inaccesibil, protejând experiența utilizatorului final.



Figură 49 - Carduri cu distribuiți & bara laterală

Pe lângă funcționalitățile deja menționate, utilizatorul are acces la pagina de salvări și pagina de profil personal. În pagina de salvări, utilizatorul poate vizualiza articolele marcate, având posibilitatea de a le accesa rapid într-un spațiu dedicat. Pagina de profil permite gestionarea datelor personale introduse în momentul înregistrării. Utilizatorul are opțiunea de a modifica aceste informații, într-un formular specializat, cu validare în timp real. Odată salvate, noile date înlocuiesc informațiile anterioare, fiind reflectate în toate componentele aplicației unde sunt utilizate. Acest mecanism asigură atât flexibilitate în actualizarea informațiilor, cât și coerență la nivelul profilului digital al fiecărui utilizator.



Figură 50 - Articole salvate & pagină de profil

Redactorul beneficiază de o interfață grafică intuitivă, structurată pe baza unui sistem de tab-uri, care permite selectarea și adăugarea rapidă a diverselor tipuri de conținut editorial: paragrafe, subtitluri și imagini. Această abordare modulară facilitează organizarea logică și vizuală a articolului în procesul de redactare, oferind un grad ridicat de flexibilitate în structurarea informației.

Aplicația include, de asemenea, o componentă dedicată previzualizării, în care redactorul poate vizualiza în timp real conținutul adăugat. În această secțiune, utilizatorul are posibilitatea de a modifica ordinea elementelor, de a înlocui conținutul deja introdus sau de a elimina complet anumite secțiuni, în funcție de nevoile editoriale. Pentru o mai bine vizibilitate poate alege să închidă formulare de adăugări de elemente cât timp verifică ce a redactat deja.

În ceea ce privește titlul articolului, acesta poate fi introdus manual de către redactor sau generat automat cu ajutorul unui modulului de inteligență artificială Gemini, pe baza analizei conținutului textual deja redactat. Această funcționalitate avansată sprijină procesul creativ și contribuie la optimizarea formulării titlurilor relevante și atractive.

NewsApp

@admin_acc

Article Title: Generate

Paragraph Image Header Hide

Image

Alege fișierul Screenshot 2025-06-12 153858.png

Add Image

Article Preview:

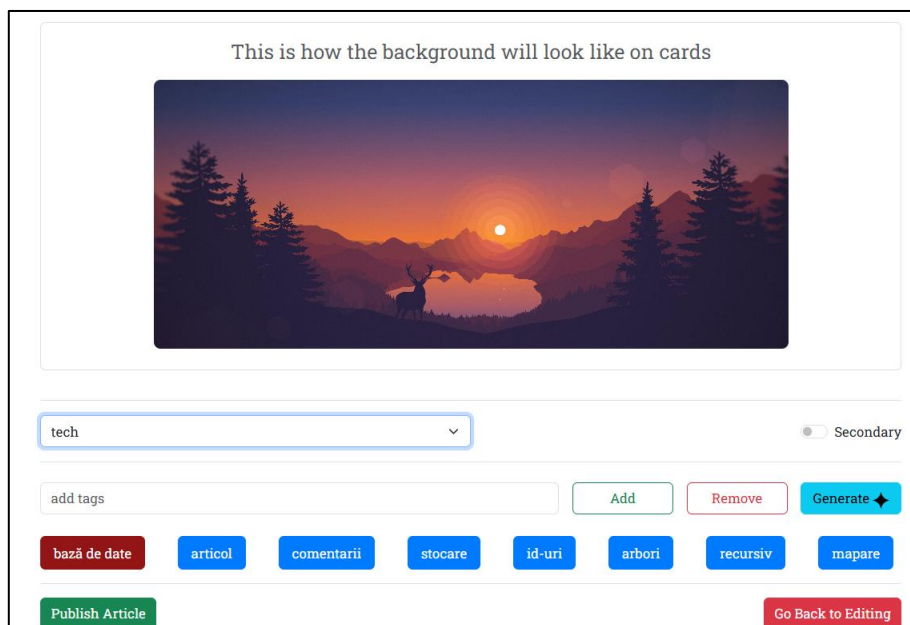
La nivelul bazei de date, fiecare articol include un câmp comments, care reprezintă o colecție de obiecte, care definesc lista de comentarii pentru fiecare pagină unde un articol este postat. Din raționamente de eficientizare a stocării, fiecare comentariu reține id-ul comentariului căruia îi răspunde, urmând ulterior a se construi o listă de arbori pe baza acestor id-uri, necesară construirii secțiunii de comentarii, care este realizată în manieră recursivă. În plus, pentru eficientizarea interogării bazei de date lista s-a optat pentru definirea listei de comentarii drept un câmp al fiecărui articol și nu o colecție separată, care ar necesita căutări suplimentare.

Figură 51 - Pagina de editare

În etapa finală a procesului editorial, redactorul este redirecționat către pagina de publicare, unde dispune de opțiuni suplimentare pentru personalizarea modului în care articolul va fi prezentat pe platformă. Aici, utilizatorul poate selecta un fundal vizual (background) pentru cardul de afișare al articolului, având de asemenea posibilitatea de a alege tipul de card utilizat: principal (*main*) sau secundar, în funcție de relevanța și importanța conținutului.

Adăugarea de etichete de căutare pentru articol se poate realiza fie manual, prin introducerea directă a cuvintelor-cheie, fie automat, cu sprijinul aceluiasi modul de inteligență artificială, Gemini, care generează etichete relevante pe baza conținutului redactat anterior. Aceste etichete contribuie la o mai bună indexare și filtrare a materialelor în cadrul platformei.

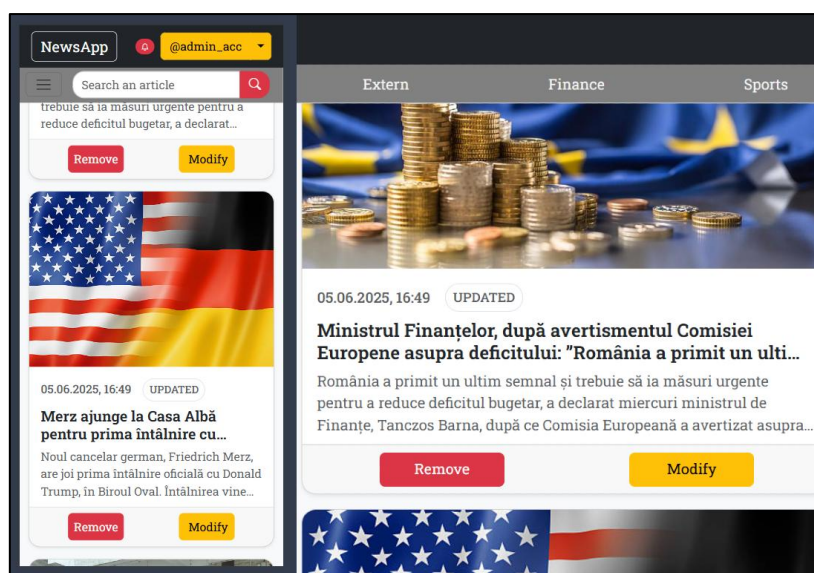
În cazul în care, în urma previzualizării, redactorul identifică necesitatea unor ajustări suplimentare, aplicația oferă posibilitatea de revenire la pagina de editare, fără pierderea progresului realizat. Astfel, se asigură un flux de lucru flexibil și eficient, orientat spre calitatea finală a conținutului publicat.



Figură 52 - Pagina de publicare

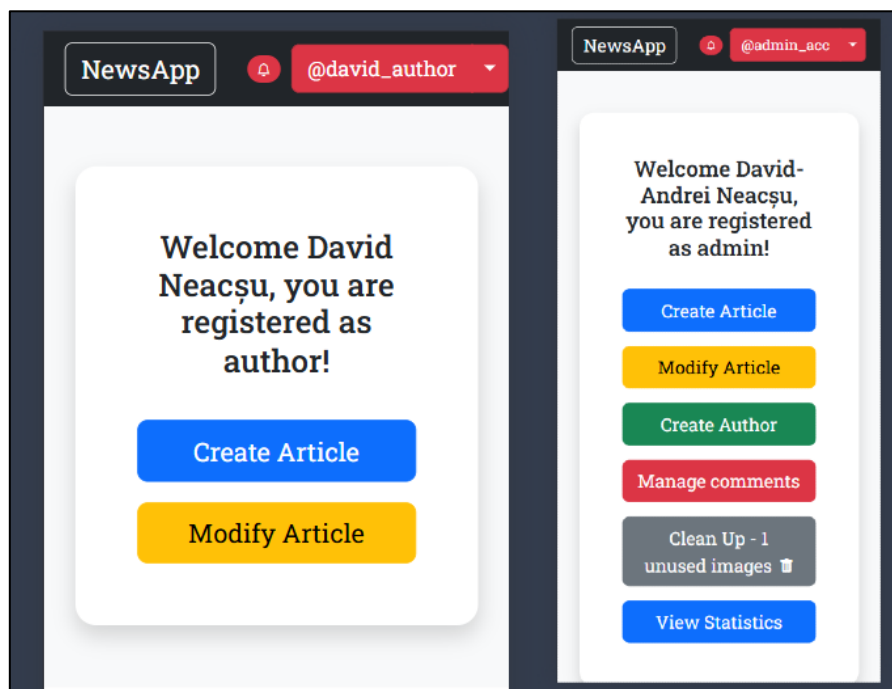
De asemenea, redactorul are posibilitatea de a **modifica** sau **elimina** articolele pe care le-a creat anterior, prin intermediul opțiunii de gestionare disponibile în cadrul dashboard-ului personal. Accesând secțiunea dedicată articolelor proprii, redactorul poate edita conținutul deja publicat — atât textual, cât și vizual — sau poate opta pentru eliminarea completă a materialului din platformă.

Această funcționalitate oferă un grad ridicat de flexibilitate editorială, permițând actualizarea și menținerea conținutului la un standard de calitate adecvat, în concordanță cu eventuale schimbări de context, corecturi sau decizii redacționale ulterioare publicării.



Figură 53 - Pagină de modificare sau eliminare articole

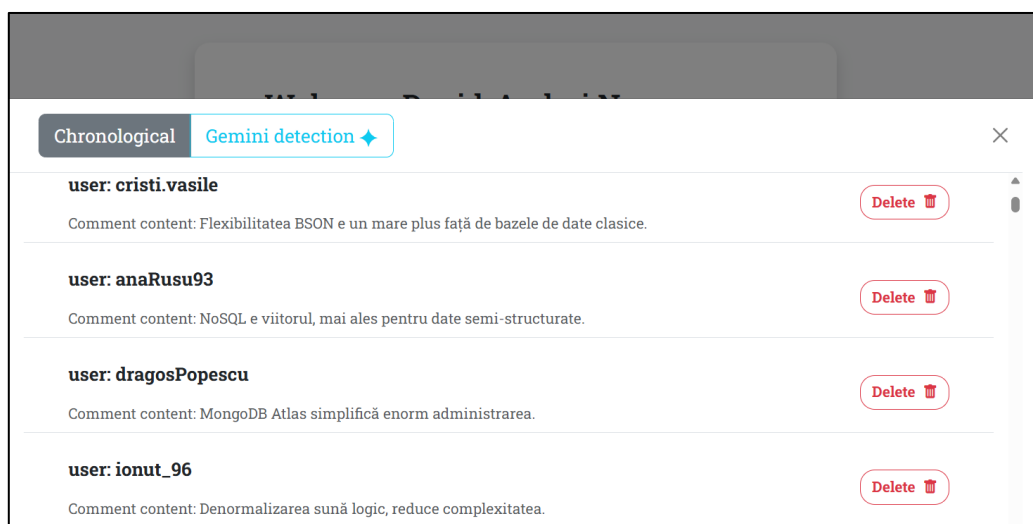
În momentul autentificării cu un cont de **redactor** sau **administrator**, utilizatorii sunt redirecționați automat către un dashboard dedicat, care se afișează în mod condițional, în funcție de rolul asociat contului. Această redare dinamică a interfeței asigură accesul diferențiat la funcționalitățile specifice fiecărui tip de utilizator, respectând principiile de securitate și organizare ierarhică a aplicației.



Figură 54 - Dashboard redactor și administrator

Administratorul beneficiază de un set extins de funcționalități, care depășesc atribuțiile unui redactor obișnuit, conferindu-i un control avansat asupra platformei, însă păstrează și atribuțiile unui redactor, având posibilitatea de a crea, edita și publica articole. Totodată acesta are posibilitatea de a crea conturi noi de redactori, utilizând o versiune modificată a paginii de „Sign Up”.

În plus, administratorul este responsabil de moderarea comentariilor postate de utilizatori, având autoritatea de a interveni asupra conținutului care încalcă regulile și politicile platformei. Acesta poate parcurge manual comentariile pentru a identifica posibile nereguli sau, alternativ, poate utiliza modulul de inteligență artificială Gemini pentru a filtra automat ultimele 100 de comentarii, identificând conținutul considerat nepotrivit.



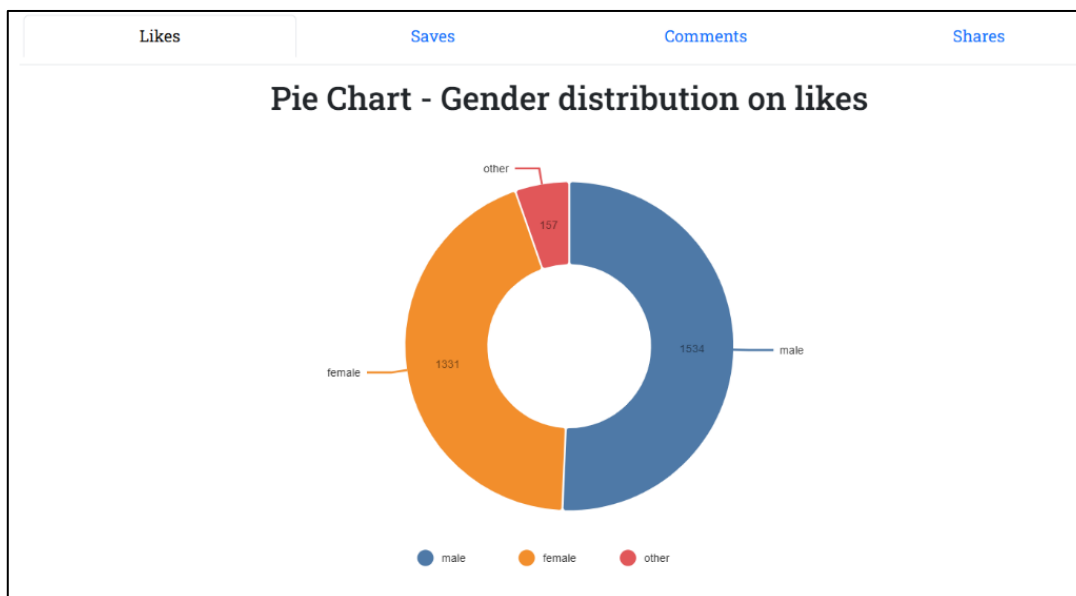
Figură 55 - Listă comentarii moderator

Pentru o intervenție eficientă și contextuală, administratorul are posibilitatea de a elimina comentariile direct din secțiunea de comentarii asociată fiecărui articol, asigurând astfel o moderare rapidă, integrată și vizibilă în timp real. Această funcționalitate contribuie la menținerea unui mediu digital sigur, civilizat și adecvat pentru toți utilizatorii platformei.

O funcționalitate suplimentară esențială constă în accesul la un sistem autonom de gestionare a resurselor media, prin care administratorul poate elibera spațiul de stocare din Cloudinary prin identificarea și ștergerea imaginilor neutilizate, contribuind astfel la optimizarea resurselor și la menținerea unei infrastructuri eficiente.

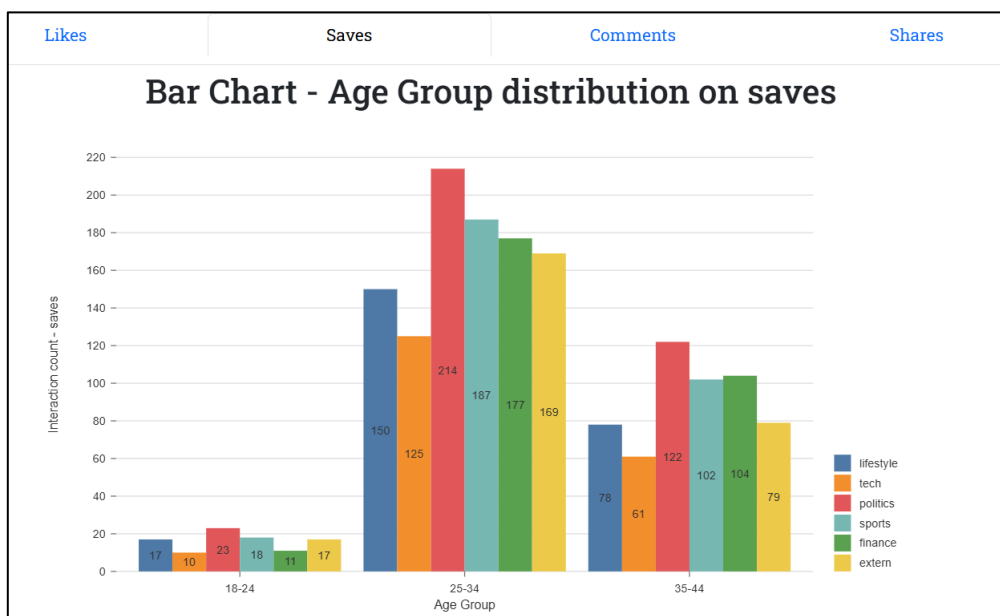
Pe componenta analitică, administratorul are acces la o secțiune dedicată vizualizării statistice a activității platformei, unde datele sunt prezentate sub formă de grafice interactive și relevante, menite să faciliteze interpretarea tendințelor de utilizare și comportament digital. În plus, administratorul poate modifica tipul de interacțiune analizat printr-un sistem de tab-uri funcționale, care filtrează datele în timp real.

În această secțiune, administratorul poate consulta distribuția utilizatorilor pe gen sau vârstă, raportată la fiecare tip de interacțiune (comentarii, aprecieri, distribuiri, salvări), prin intermediul graficele de tip „pie chart”, care oferă o reprezentare procentuală clară și intuitivă.

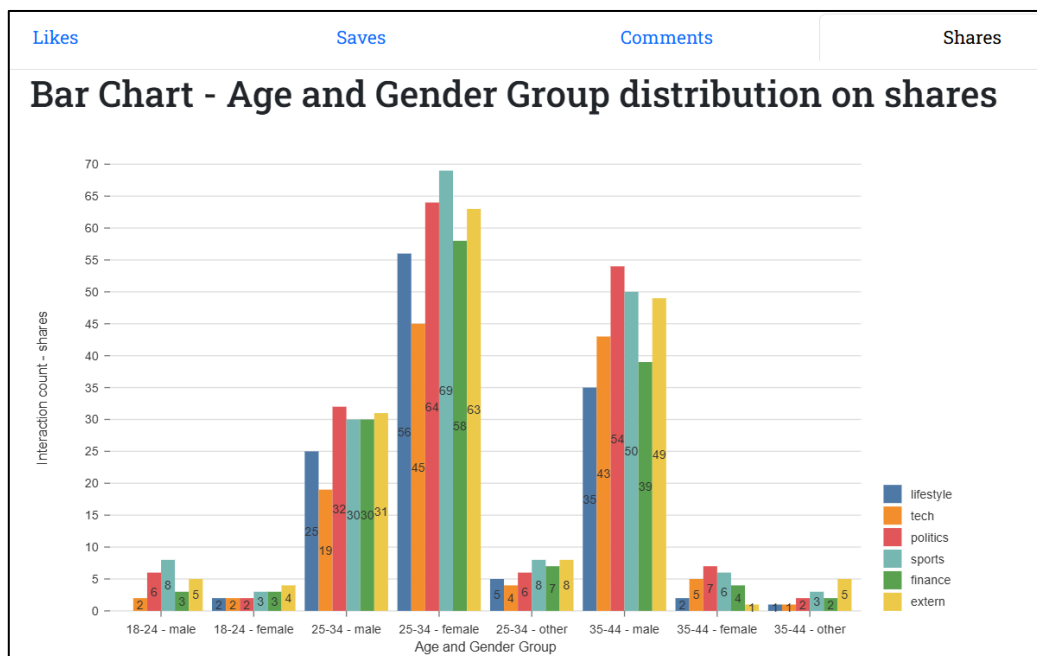


Figură 56 - Pie Chart Gender

De asemenea, este posibilă vizualizarea distribuției demografice, pe criterii de gen și vârstă, pentru fiecare tip de articol în parte, utilizând grafice de tip bar chart. Aceste grafice pot fi afișate atât pe fiecare criteriu, pentru o analiză specifică a fiecărei, cât și combinate, facilitând astfel compararea detaliată între diferitele categorii de articole și segmente demografice.

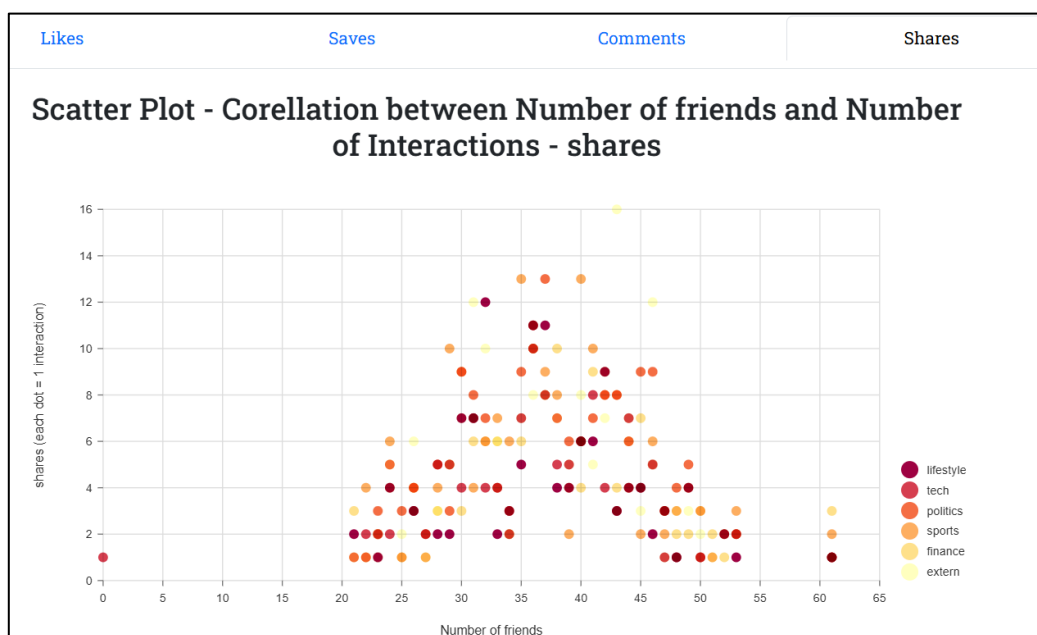


Figură 57 - Bar Chart Age



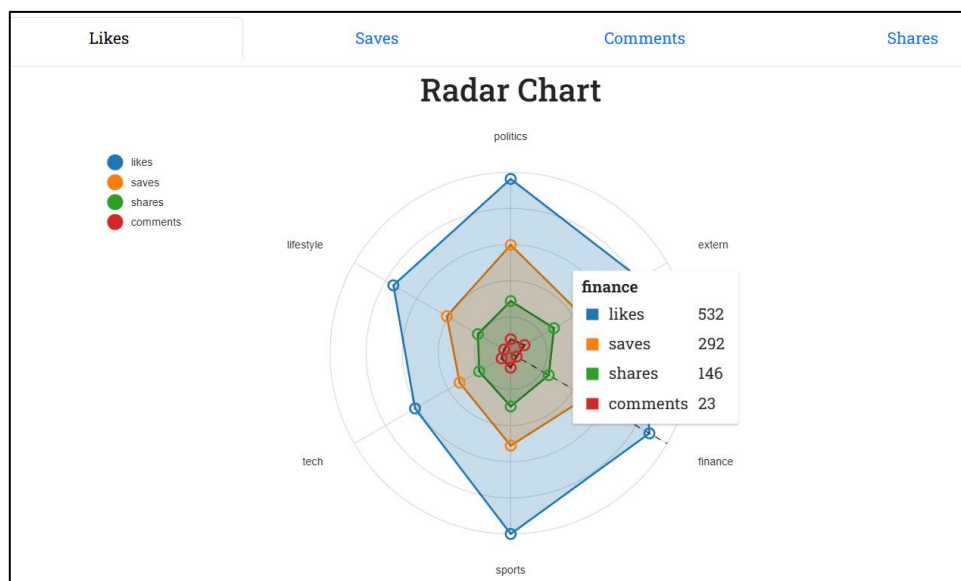
Figură 58 - Bar Chart Age-Gender

Pentru o analiză mai avansată a comportamentului utilizatorilor, aplicația oferă un grafic de tip „Scatter plot” în care axa X reprezintă numărul de prieteni ai unui utilizator, iar axa Y indică numărul de interacțiuni realizate. Fiecare punct din grafic este colorat în funcție de categoria articolelor cu care utilizatorul a interacționat, oferind o viziune complexă asupra relației dintre activitatea socială și implicarea în platformă.



Figură 59 - Scatter Plot

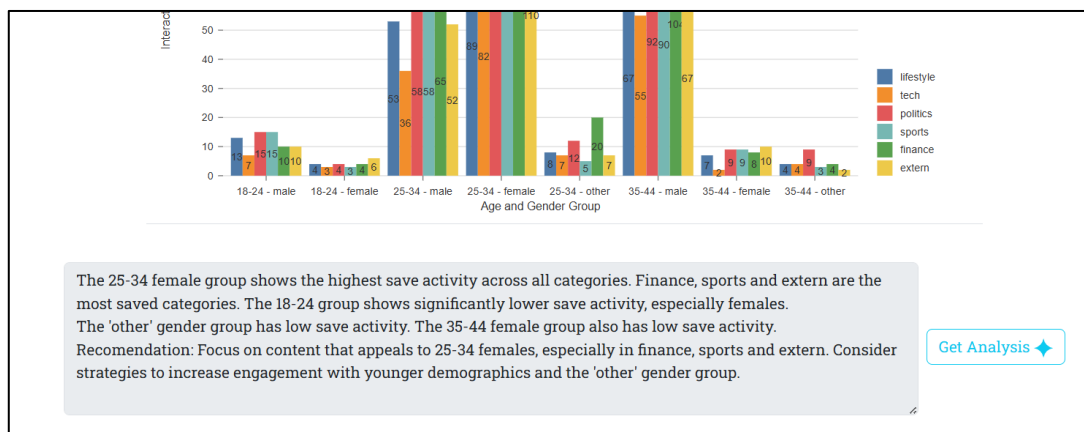
Aplicația include și un grafic de tip „Radar” care ilustrează totalul interacțiunilor aferente fiecărei categorii de articol, oferind astfel o imagine de ansamblu clară și detaliată asupra domeniilor de conținut pe baza activității utilizatorilor.



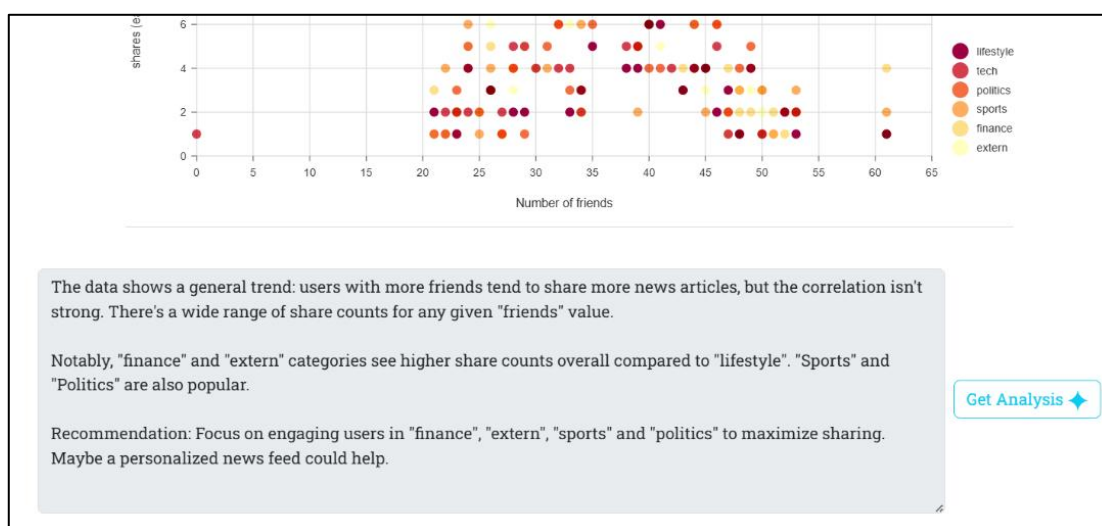
Figură 60 - Radar

Această suită de instrumente vizuale reprezintă un suport valoros pentru procesul decizional la nivel administrativ, facilitând o înțelegere aprofundată și bine documentată a dinamicii comportamentale a utilizatorilor platformei. Suplimentar, toate reprezentările grafice beneficiază de posibilitatea unei interpretări asistate prin intermediul modului de inteligență artificială **Gemini**.

Modulul primește informații detaliate referitoare la tipologia graficului, structura și formatul datelor, precum și contextul analitic — incluzând natura interacțiunilor, categoria de conținut analizată și variabilele demografice. Pe baza acestor elemente, sistemul generează interpretări automatizate și formulează sugestii relevante pentru redacție, contribuind astfel la optimizarea procesului editorial, oferirea de noi strategii de business și sugestii legate de adaptarea conținutului în funcție de comportamentele și preferințele publicului.



Figură 61 - Analiză LLM 2



Figură 62 - Analiză LLM 1

Această abordare integrată dintre vizualizarea interactivă a datelor și interpretarea asistată de inteligență artificială conferă aplicației un caracter modern, orientat spre eficiență și adaptabilitate. Prin facilitarea înțelegerii profunde a comportamentului utilizatorilor și sprijinirea procesului decizional cu instrumente analitice avansate, platforma devine un instrument strategic esențial în gestionarea și optimizarea conținutului digital într-un mediu în continuă schimbare.

Concluzie

Lucrarea de față a avut ca obiectiv dezvoltarea unei aplicații web de tip CMS, dedicată redacțiilor de știri, care integrează funcționalități specifice rețelelor de socializare, într-un demers de modernizare a procesului editorial și de consolidare a interacțiunii cu publicul. Într-un peisaj media aflat într-o continuă transformare, marcat de fragmentarea atenției, dezinformare și competiție pentru vizibilitate, soluția propusă oferă un model inovator, care îmbină rigoarea jurnalismului tradițional cu dinamica și interactivitatea mediului digital actual.

Aplicația a fost concepută ca un sistem complet, modular și scalabil, bazat pe tehnologii moderne de tip full-stack JavaScript și servicii cloud, capabil să susțină activitatea unei redacții de știri, de la crearea articolelor până la publicarea și analiza impactului acestora. În plus, integrarea unui modul de inteligență artificială (Gemini) a permis automatizarea interpretării datelor de interacțiune, oferind redacției sugestii relevante pentru optimizarea conținutului, adaptarea la preferințele utilizatorilor și fundamentarea deciziilor strategice.

Prin această abordare, aplicația răspunde nevoii de reinventare a modului în care conținutul jurnalistic este produs și consumat, contribuind la restabilirea unei relații de încredere între redacție și cititori. Mai mult decât un instrument tehnologic, soluția propusă poate deveni o platformă de comunicare etică, interactivă și participativă, aliniată la noile tendințe ale societății informaționale.

Pe termen lung, direcțiile de dezvoltare pot include extinderea funcționalităților bazate pe AI, utilizarea aprofundată a elementelor de localizare, extinderea analizelor statistice și integrarea de modele de monetizare sustenabilă și adaptarea platformei pentru aplicații mobile native, nu doar adaptarea ecranului în browser, consolidând astfel potențialul aplicației de a deveni un punct de referință în transformarea digitală a industriei media.

Anexă

Lista figurilor

Figură 1 - Diagrama cazurilor de utilizare	14
Figură 2 - Diagramă de activitate: Secțiunea de comentarii	16
Figură 3 - Diagramă de activitate: Flux operațional cereri de prietenie.....	17
Figură 4 - Diagramă de activitate: Redare contextuală Login.....	18
Figură 5 - Diagramă de secvență: Sistem autonom de șterge a imaginilor nefolosite	20
Figură 6 - Diagramă de comunicare: Comunicare Geocode și OpenWeather.....	21
Figură 7 - Diagramă activitate: Prelucrare date pentru analiză prin LLM	22
Figură 8 - Diagramă BPMN	23
Figură 9 - Diagramă de deployment.....	24
Figură 10 - Diagrama relațiilor dintre entități	28
Figură 11 - Structură folder backend.....	43
Figură 12 - Structură folder frontend	44
Figură 13 - Interfață Render pentru variabile de mediu	45
Figură 14 - Conexiune bază de date mongodb	46
Figură 15 - Scheme mongoose	47
Figură 16 - Configurare langchain/google-genai	48
Figură 17 - Configurare Cloudinary	48
Figură 18 - Server: configurare middleware	49
Figură 19 - Server: Routere express dedicate.....	50
Figură 20 - Server: servirea interfeței React SPA	50
Figură 21 - Render: comenzi pentru deployment	51
Figură 22 - Funcția loginUser	52
Figură 23 - Server: cookies	53
Figură 24 - funcție middleware pentru autentificare	54
Figură 25 - Autentificare RBAC	55
Figură 26 - configurare cloudinary storage și middleware multer	56
Figură 27 - Funcție pentru obținerea de url-urilor.....	57
Figură 28 - Funcție pentru obținerea id-urilor publice nefolosite	58
Figură 29 - Funcție generare tag-uri cu LLM.....	59
Figură 30 - Funcție filtrare comentarii cu conținut nepotrivit cu LLM.....	60
Figură 31 - Algoritm generare comentarii pe baza LLM	61

Figură 32 - Funcție cu prompt pentru analiză grafice cu LLM	62
Figură 33 - Pipeline agregare date.....	64
Figură 34 - useEffect formatare date pentru grafice.....	65
Figură 35 - useEffect combinare date pentru grafic de tip Radar	66
Figură 36 - Validări cereri de prietenie.....	67
Figură 37 - creare mapă comentarii.....	69
Figură 38 - Creare listă de arbori cu comentariile.....	69
Figură 39 - Funcție recursivă: secțiune comentarii	70
Figură 40 - Implementare api geocode autocomplete	71
Figură 41 – Homepage: Desktop.....	73
Figură 42 - Homepage: Mobile	74
Figură 43 - Pagină articol	74
Figură 44 - Secțiune comentarii	75
Figură 45 - Login/SignUp	76
Figură 46 - Share & Comm	76
Figură 47 - Share ID & Add a friend.....	77
Figură 48 - Listă prieteni & listă cereri	77
Figură 49 - Carduri cu distribuiri & bara laterală	78
Figură 50 - Articole salvate & pagină de profil.....	79
Figură 51 - Pagina de editare.....	80
Figură 52 - Pagina de publicare	81
Figură 53 - Pagină de modificare sau eliminare articole.....	81
Figură 54 - Dashboard redactor și administrator.....	82
Figură 55 - Listă comentarii moderator	83
Figură 57 - Pie Chart Gender	84
Figură 58 - Bar Chart Age	84
Figură 59 - Bar Chart Age-Gender	85
Figură 60 - Scatter Plot	85
Figură 61 - Radar.....	86
Figură 62 - Analiză LLM 2.....	87

Bibliografie

- Apidog, Inc. (2024). *What is Bearer Token and How it Works?* Preluat de pe apidog.com:
<https://apidog.com/articles/what-is-bearer-token/>
- Cloudinary. (2025). *cloudinary.com*. Preluat de pe Image & Video APIs overview:
https://cloudinary.com/documentation/programmable_media_overview
- Cloudinary. (2025). *cloudinary.com*. Preluat de pe node_integration:
https://cloudinary.com/documentation/node_integration
- Eldridge, A. (2025, May 8). *Instagram*. Preluat de pe Encyclopedia Britannica:
<https://www.britannica.com/money/Instagram>
- Eldridge, A. (2025, May 4). *Reddit*. Preluat de pe Encyclopedia Britannica:
<https://www.britannica.com/money/Reddit>
- Elmasri, R. (2008). *Fundamentals of database systems*. India: Pearson Education.
- Fakhroutdinov, K. (2025). *Activity Diagrams*. Preluat de pe The Unified Modeling Language:
<https://www.uml-diagrams.org/>
- Fakhroutdinov, K. (2025). *UML Communication Diagrams Overview*. Preluat de pe The Unified Modeling Language: <https://www.uml-diagrams.org/communication-diagrams.html>
- Fakhroutdinov, K. (2025). *UML Component Diagrams*. Preluat de pe The Unified Modeling Language: <https://www.uml-diagrams.org/component-diagrams.html>
- Fakhroutdinov, K. (2025). *UML Sequence Diagrams*. Preluat de pe The Unified Modeling Language: <https://www.uml-diagrams.org/sequence-diagrams.html>
- Fakhroutdinov, K. (2025). *UML Use Case Diagrams*. Preluat de pe The Unified Modeling Language: <https://www.uml-diagrams.org/use-case-diagrams.html>
- Flanagan, D. (2006). JavaScript - The Definitive Guide, 5th ed. În D. Flanagan, *JavaScript - The Definitive Guide, 5th ed.* (p. 497). Sebastopol, CA: O'Reilly. Preluat de pe https://books.google.ro/books?id=2weL0iAfrEMC&q=%22single-page%22&redir_esc=y#v=snippet&q=%22single-page%22&f=false
- GeeksForGeeks. (2024, Iulie 8). *MVC Framework Introduction*. Preluat de pe geeksforgeeks:
<https://www.geeksforgeeks.org/mvc-framework-introduction/>

GeeksforGeeks, Sanchhaya Education Private Limited. (2025, Martie 18). *Mongoose Schemas Creating a Model*. Preluat de pe [geeksforgeeks.org](https://www.geeksforgeeks.org/mongoose-schemas-creating-a-model/):
<https://www.geeksforgeeks.org/mongoose-schemas-creating-a-model/>

Geoapify. (2025). *Geocoding API*. Preluat de pe [geoapify](https://www.geoapify.com/geocoding-api/):
<https://www.geoapify.com/geocoding-api/>

Geoapify. (2025). *Welcome to Geoapify Location Platform*. Preluat de pe [Geoapify](https://www.geoapify.com/geocoding-api/):
<https://www.geoapify.com/geocoding-api/>

Google. (2025). *Gemini Developer API*. Preluat de pe [ai.google.dev](https://ai.google.dev/gemini-api/docs):
<https://ai.google.dev/gemini-api/docs>

IBM. (2025, April 24). *What is REST API?* Preluat de pe [ibm.com](https://www.ibm.com/think/topics/rest-apis):
<https://www.ibm.com/think/topics/rest-apis>

IBM Inc. (2024, August 5). *What is an API endpoint?* Preluat de pe [IBM - United States](https://www.ibm.com/think/topics/api-endpoint):
<https://www.ibm.com/think/topics/api-endpoint>

Johnson, B. (2019). *Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers*. Preluat de pe [books.google.ro](https://books.google.ro/books?hl=ro&lr=&id=t1enDwAAQBAJ&oi=fnd&pg=PR15&dq=visual+studio+code&ots=Qeh-f4zH7P&sig=ugm2ZxYdEks6W3AdD7XyVI7IKC4&redir_esc=y#v=onepage&q=visual%20studio%20code&f=false):
https://books.google.ro/books?hl=ro&lr=&id=t1enDwAAQBAJ&oi=fnd&pg=PR15&dq=visual+studio+code&ots=Qeh-f4zH7P&sig=ugm2ZxYdEks6W3AdD7XyVI7IKC4&redir_esc=y#v=onepage&q=visual%20studio%20code&f=false

LangChain, Inc. (2025). *Introduction*. Preluat de pe [js.langchain.com](https://js.langchain.com/docs/introduction/):
<https://js.langchain.com/docs/introduction/>

Lindemulder, G., & Kosinski, M. (2024, August 20). *What is role-based access control (RBAC)?* Preluat de pe [IBM](https://www.ibm.com/think/topics/rbac): <https://www.ibm.com/think/topics/rbac>

Matthias, M. (2024, August 7). *Git*. Preluat de pe [Encyclopedia Britannica](https://www.britannica.com/technology/Git):
<https://www.britannica.com/technology/Git>

Meta Platforms, Inc. (2025). *Hooks at a Glance*. Preluat de pe [React](https://legacy.reactjs.org/docs/hooks-overview.html):
<https://legacy.reactjs.org/docs/hooks-overview.html>

Meta Platforms, Inc. (2025). *React. The library for web and native user interfaces*. Preluat de pe react.dev: <https://react.dev/>

Microsoft . (2025). *Visual Studio Code documentation*. Preluat de pe [code.visualstudio.com](https://code.visualstudio.com/docs):
<https://code.visualstudio.com/docs>

MongoDB, Inc. (2024). *Schemas*. Preluat de pe [mongodb.com](https://www.mongodb.com/docs/atlas/app-services/schemas/):
<https://www.mongodb.com/docs/atlas/app-services/schemas/>

MongoDB, Inc. (2024). *www.mongodb.com*. Preluat de pe Get Started with Atlas:
<https://www.mongodb.com/docs/atlas/getting-started/>

MongoDB, Inc. (2025). *What is a Document Database?* Preluat de pe [mongodb.com](https://www.mongodb.com/resources/basics/databases/document-databases):
<https://www.mongodb.com/resources/basics/databases/document-databases>

MongoDB, Inc. (2025). *What is MongoDB Compass?* Preluat de pe [www.mongodb.com](https://www.mongodb.com/docs/compass/current/):
<https://www.mongodb.com/docs/compass/current/>

MongoDB, Inc. (2025). *What Is MongoDB?* Preluat de pe [www.mongodb.com](https://www.mongodb.com/company/what-is-mongodb):
<https://www.mongodb.com/company/what-is-mongodb>

Mozilla Developer Network. (2025). *HTTP*. Preluat de pe [developer.mozilla.org](https://developer.mozilla.org/en-US/docs/Web/HTTP):
<https://developer.mozilla.org/en-US/docs/Web/HTTP>

Mozilla Developer Network. (2025). *JavaScript*. Preluat de pe Mozilla Developer Network:
<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

Mozilla Developer Network. (2025). *Salt*. Preluat de pe Resources for Developers:
<https://developer.mozilla.org/en-US/docs/Glossary/Salt>

Munro, A. (2025, April 6). *JavaScript*. Preluat de pe Encyclopedia Britannica:
<https://www.britannica.com/technology/JavaScript>

Object Management Group. (2017). *Unified Modeling Language v2.5.1*. Preluat de pe [omg.org](https://www.omg.org/spec/UML/2.5.1/PDF):
<https://www.omg.org/spec/UML/2.5.1/PDF>

OpenJS Foundation. (2025). *About Node.js®*. Preluat de pe [nodejs.org](https://nodejs.org/en/about):
<https://nodejs.org/en/about>

OpenJS Foundation. (2025). *Express. Fast, unopinionated, minimalist web framework for Node.js*. Preluat de pe expressjs.com:
<https://expressjs.com/>

OpenJS Foundation. (2025). *Introduction to Node.js*. Preluat de pe [nodejs.org](https://nodejs.org/en/learn/getting-started/introduction-to-nodejs#introduction-to-nodejs):
<https://nodejs.org/en/learn/getting-started/introduction-to-nodejs#introduction-to-nodejs>

Render. (2025). *Render Documentation*. Preluat de pe render.com: <https://render.com/docs>

Visual Paradigm. (2024). *Drawing BPMN Business Process Diagram*. Preluat de pe visual-paradigm.com: https://www.visual-paradigm.com/support/documents/vpuserguide/2821/286/7114_drawingbusin.html

Visual Paradigm. (2025). *What is Unified Modeling Language (UML)?* Preluat de pe <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/>.

W3C. (2025). *HTML: Living standard*. Preluat de pe World Wide Web Consortium: <https://html.spec.whatwg.org>

Wikipedia. (2023, Martie 18). *Application Programming Interface*. Preluat de pe ro.wikipedia.org: https://ro.wikipedia.org/wiki/Application_Programming_Interface

Yasar, K. (2024, Mai 17). *What is hashing?* Preluat de pe Informa TechTarget: <https://www.techtarget.com/searchdatamanagement/definition/hashing>

Yasar, K. (2025, January). *What is Reddit? How it works, history and pros and cons*. Preluat de pe www.techtarget.com: <https://www.techtarget.com/searchcio/definition/Reddit>