

# PROIECT DE DIPLOMĂ

Îndrumător proiect/Coordonator științific,  
Ş.l.dr.ing. Nicolae JĂSCANU

Absolvent,  
Adrian Dumitru BORŞAN

Galați  
Anul 2023

PROGRAMUL DE STUDII: Calculatoare și Tehnologia Informației

## **Aplicație web bazată pe microservicii folosind framework-ul Spring Boot**

Îndrumător proiect/Coordonator științific,  
Ş.l.dr.ing. Nicolae JÂSCANU

Absolvent,  
Adrian Dumitru BORŞAN

Galați  
Anul 2023

## **Rezumat**

Kiire (în finlandeză însemnând ceva care ocupă timpul) reprezintă o aplicație web, care utilizează arhitectura de microservicii. Două componente principale ale acestei infrastructuri sunt o aplicație web de socializare destinată utilizatorilor și un modul administrativ, care rulează în paralel și oferă funcționalități diferite în cadrul sistemului. Aplicațiile au fost dezvoltate utilizând un set robust de tehnologii, inclusiv Spring Boot cu Gradle și Java, Angular, TypeScript, Angular Material și NGINX.

Acest proiect a implicat implementarea și gestionarea unui cluster Kubernetes Minikube, responsabil de găzduirea a trei aplicații separate Spring Boot. Clusterul a gestionat, de asemenea, două aplicații Angular care rulează pe NGINX. Această configurație a permis alocarea eficientă a resurselor, scalabilitatea fără intreruperi și operațiunile de servicii izolate și independente, oferind astfel o demonstrație practică a avantajelor oferite de microservicii.

Aplicația web de socializare permite utilizatorilor să efectueze operații CRUD pe postări, să raporteze postări care încalcă termenii și condițiile și să vizualizeze profilurile altor utilizatori. Modulul administrativ este responsabil de monitorizarea conținutului, de decizia dacă o postare raportată încalcă regulile și de vizualizarea în timp real a noilor loguri generate de aplicația client. Ambele aplicații au folosit capacitatea unui set variat de baze de date precum MySQL, PostgreSQL, MongoDB și Neo4j pentru a răspunde nevoilor diverse de stocare a datelor. Sistemul a integrat, de asemenea, RabbitMQ ca broker de mesaje pentru comunicare fără intreruperi și asincronă între microservicii.

Proiectul a inclus, de asemenea, utilizarea Keycloak pentru gestionarea securizată a identității și accesului, personalizată cu pluginurile Keycloakify și Keycloak-Event-Listener-RabbitMQ. Pentru gestionarea eficientă a fișierelor, s-a configurat un serviciu Minio, și s-a folosit Ingress NGINX pentru gestionarea accesului extern la serviciile din cadrul clusterului Kubernetes.

Direcțiile pentru viitor includ explorarea strategiilor de implementare automatizată, optimizarea performanței și adoptarea unei plase de servicii pentru controlul avansat al comunicării între servicii.

În esență, această teză nu numai că demonstrează modul în care tehnologiile diverse pot fi integrate într-o arhitectură de microservicii pentru a asigura scalabilitate, reziliență și ușurință în întreținere, dar și furnizează o aplicație web de socializare.

## CUPRINS

Introducere .....	3
Capitolul 1 STADIUL ACTUAL AL LUCRĂRILOR DIN DOMENIU.....	4
1.1    Migrarea de la monolic la microservicii.....	4
1.2    Stadiul Pieței existente .....	5
1.2.1    Netflix .....	5
1.2.2    Zalando.....	6
1.2.3    Alibaba .....	6
1.2.4    Instagram .....	7
Capitolul 2 Cerințe și Specificații.....	8
Capitolul 3 Proiectarea Sistemului.....	10
3.1    Proiectarea Aplicației.....	10
3.1.1    Diagrama Site-ului .....	10
3.2    Prezentarea și Modelarea Bazei de Date .....	10
3.2.1    Nivelul Conceptual.....	11
3.2.2    Nivelul Fizic al Bazei de Date ( MCD ).....	13
3.3    Prezentarea API endpoints .....	15
Capitolul 4 Implementarea Sistemului și tehnologii .....	18
4.1    Instrumentele Informatice Utilizate în Elaborarea Aplicației .....	18
4.2    Limbaje folosite.....	19
4.3    Framework.....	20
4.4    Librării .....	21
4.5    Tehnologii.....	22
4.6    Prezentarea aplicației .....	22
4.6.1    Prezentare rol client.....	23
4.6.2    Prezentare admin.....	30
4.6.3    Prezentarea administrator de sistem .....	32
Capitolul 5 PLANURI DE VIITOR .....	38
Capitolul 6 Concluzii .....	40
Bibliografie .....	41

## LISTA FIGURILORE

Figură 1.1posibilă arhitectură Netflix.....	5
Figură 1.2arhitectură microservicii Zalando.....	6
Figură 1.3arhitectură microservicii Alibaba.....	7
Figură 2.1Diagrama cluster-ului.....	8
Figură 3.1diagrama site-ului admin.....	10
Figură 3.2diagrama site-ului utilizatorului.....	10
Figură 3.3 fragment orientativ din modelul conceptual pentru schema bazei de date Postgres folosită de Keycloak.....	12
Figură 3.4Modelul conceptual al bazei de date Neo4j .....	12
Figură 3.5exemplu bază de date Neo4j populate și exemplu utilizator cu multe postări	12
Figură 3.6model conceptual MySQL .....	13
Figură 3.7proprietăți logare în mongodb.....	14
Figură 3.8exemplu împărțire a fișierelor postărilor în funcție de utilizator .....	15
Figură 4.1pagina de login client în Angular .....	23
Figură 4.2pagina de logare și pagina de înregistrare în Keycloak .....	23
Figură 4.3Pagina de termeni și condiții .....	24
Figură 4.4pagina home .....	24
Figură 4.5meniu pagina home.....	25
Figură 4.6pagina pentru schimbarea parolei din Keycloak .....	25
Figură 4.7pagina de profil a utilizatorului .....	26
Figură 4.8 dialogul de editare profil și profilul cu editările făcute .....	26
Figură 4.9 formular adăugare postare .....	27
Figură 4.10noua postare creată.....	27
Figură 4.11deschiderea meniului contextual al postării .....	28
Figură 4.12formularul de editare postare .....	28
Figură 4.13 cum apare postarea după ce a fost editată .....	29
Figură 4.14meniul postărilor nedeținute de utilizatorul curent.....	29
Figură 4.15mesaj de confirmare raportare cu succes .....	30
Figură 4.16pagina de conectare admin.....	30
Figură 4.17pagina de logare admin și prima conectare .....	31
Figură 4.18componenta de tratat report postare .....	31
Figură 4.19componenta de tratare report când nu mai există nici un report netratat ....	32
Figură 4.20 informațiile paginate și informațiile prezentate în timp real .....	32
Figură 4.21starea de sănătate a servcilor.....	33
Figură 4.22secțiunea de informații cu privire la memoria consumată și puterea de procesare.....	33
Figură 4.23secțiunea services.....	34
Figură 4.24pagina principală si pagina de logare Keycloak.....	34
Figură 4.25meniul de alegere a tărâmului și meniul de creare a unui nou admin .....	34
Figură 4.29setarea și confirmarea parolei a unui cont temporar .....	35
Figură 4.30informații cu privire la datele log entries.....	35
Figură 4.31interrogare de selecție a maximum 25 de noduri.....	35

Figură 4.32monitorizarea performanței lui postgres folosind pgadmin .....	36
Figură 4.33exemplu interogare pentru a vizualiza toți utilizatorii și numele tărâmului din care face parte contul.....	36
Figură 4.34vizualizarea tuturor tabelelor din MySQL.....	36
Figură 4.35vizualizarea informațiilor a unor rânduri din tabela postărilor.....	37
Figură 4.36consola RabbitMQ .....	37

## **LISTA TABELELOR**

3.3.1.1.1 Tabelul 1.1 PostController.....	15
3.3.1.1.2 Tabelul 1.1 UserController.....	16
3.3.1.1.3 Tabelul 1.1 LogEntryController .....	16
3.3.1.1.4 Tabelul 1.1 ReportController.....	17
3.3.1.1.5 Tabelul 1.1 WebSocketConfig.....	17

## INTRODUCERE

În era digitală, aplicațiile web joacă un rol central în domenii precum comunicarea, partajarea informațiilor, divertismentul și comerțul online. Arhitectura bazată pe microservicii, cu abordarea sa granulară în designul software, a devenit din ce în ce mai populară datorită scalabilității sale, ușurinței de întreținere și avantajelor de implementare independentă. Această teză explorează detaliile creării unei aplicații web folosind abordarea microserviciilor, cu accent specific pe framework-ul Spring Boot, implementată pe un cluster Kubernetes MiniKube pentru o platformă de social media, divizată în interfețe client și administrative. Aplicația orientată către utilizatori oferă o platformă digitală pentru comunicare și partajare de conținut, în timp ce aplicația administrativă monitorizează conformitatea platformei prin reglementarea conținutului. Aplicațiile sunt construite utilizând o combinație de baze de date, inclusiv MySQL, PostgreSQL, MongoDB și Neo4J, asigurând o gestionare optimă a datelor. Toate aceste componente sunt găzduite pe un cluster Kubernetes pentru a oferi un sistem scalabil, cu disponibilitate ridicată și toleranță la evenimente neprevăzute.

Scopul principal al acestei teze este de a proiecta și implementa o aplicație web folosind modelul de microservicii, care să fie scalabilă, eficientă și distribuită. Obiectivele specifice includ:

- crearea aplicațiilor client și administrative folosind Angular, TypeScript și Angular Material.
- construirea serviciilor backend utilizând framework-ul Spring Boot.
- incorporarea microserviciilor RabbitMQ pentru a facilita comunicarea în timp real a informațiilor și sincronizarea datelor.
- utilizarea Keycloak pentru autentificarea și autorizarea securizată a utilizatorilor, completată cu pluginurile Keycloakify și Keycloak-Event-Listener-RabbitMQ.
- implementarea unei structuri multi-database cu MySQL, PostgreSQL, Neo4J, MongoDB și Minio pentru a acoperi diversele nevoi de gestionare a datelor.
- implementarea tuturor serviciilor într-un cluster Kubernetes MiniKube pentru a asigura un sistem robust și rezistent.
- evaluarea performanței și eficienței sistemului.

## CAPITOLUL 1                    STADIUL ACTUAL AL LUCRĂRILOR DIN DOMENIU

### 1.1 Migrarea de la monolic la microservicii

Domeniul dezvoltării de aplicații web bazate pe microservicii utilizând Spring Boot a câștigat un avânt semnificativ în ultimii ani. Pe măsură ce organizațiile se străduiesc din ce în ce mai mult pentru flexibilitate, scalabilitate și reziliență în infrastructura lor software, ele consideră arhitectura microserviciilor ca fiind o soluție eficientă pentru nevoile lor.

Tradițional, aplicațiile web erau dezvoltate folosind o arhitectură monolică, cu toate componentele integrate într-o singură unitate indivizibilă. Deși această abordare simplifică implementarea și operarea, ea nu se potrivește pentru gestionarea aplicațiilor complexe și la scară largă, conducând la probleme cu scalabilitatea, reziliența și timpul de implementare. Introducerea microserviciilor a abordat aceste probleme prin descompunerea unei aplicații într-o colecție de servicii slab cuplate. Această abordare permite ca serviciile individuale să fie dezvoltate, implementate și scalate independent, îmbunătățind astfel fiabilitatea și reactivitatea generală a sistemului.

Un microserviciu este un proces autonom care se ocupă de o funcționalitate specifică a unei aplicații. Acesta încapsulează logica specifică domeniului său și poate fi implementat independent, permitându-i să funcționeze și să evolueze separat de celelalte părți ale sistemului. Această separare a responsabilităților este fundamentală pentru conceptul de microservicii, punând accent pe modularitate și design centrat pe afaceri. Arhitectura bazată pe microservicii se concentrează pe organizarea acestor unități autonome într-un sistem coerent. Fiecare microserviciu este dedicat unei funcționalități specifice, cum ar fi gestionarea utilizatorilor, manipularea postărilor, notificări sau sarcini administrative într-o aplicație web de socializare. Acestea comunică între ele prin intermediul API-urilor și protocolelor bine definite, permitând un design flexibil și distribuit. Microserviciile pot fi dezvoltate, implementate și scalate individual în funcție de nevoi, facilitând astfel un ciclu agil de dezvoltare și întreținere. Acest stil arhitectural promovează, astfel, un grad ridicat de modularitate și poate rezulta într-o aplicație mai ușor de întreținut și mai robustă.

Spring Boot este construit pe baza frameworkului Spring, proiectat pentru a simplifica inițializarea și dezvoltarea aplicațiilor Spring. Acesta oferă o serie de caracteristici, cum ar fi auto-configurarea și încorporarea directă a serverelor de aplicații care facilitează configurarea și rularea aplicațiilor.

În domeniul microserviciilor, Spring Boot și-a stabilit poziția ca unul dintre cele mai populare alegeri, datorită scalabilității sale, eficienței și compatibilității largi cu diverse limbaje de programare, baze de date și servere de aplicații. Organizații notabile precum Netflix, Alibaba și Amazon au încorporat Spring Boot în aplicațiile lor web bazate pe microservicii, evidențiind practicitatea și versatilitatea sa în scenarii din lumea reală.

Stiva de tehnologie modernă pentru dezvoltarea microserviciilor a evoluat semnificativ de-a lungul anilor. Combinarea Spring Boot cu platforme de orchestrare a containerelor precum Kubernetes, sisteme de mesagerie precum RabbitMQ și baze de date precum MySQL, Postgres, Neo4j și MongoDB este acum considerată standard.

Mai mult, cu creșterea necesității de gestionare sigură a identității, unelte precum Keycloak au fost integrate în stivă. Partea de client a aplicațiilor a văzut, de asemenea, evoluții, Angular fiind una dintre alegerile de top pentru construirea interfețelor utilizator.

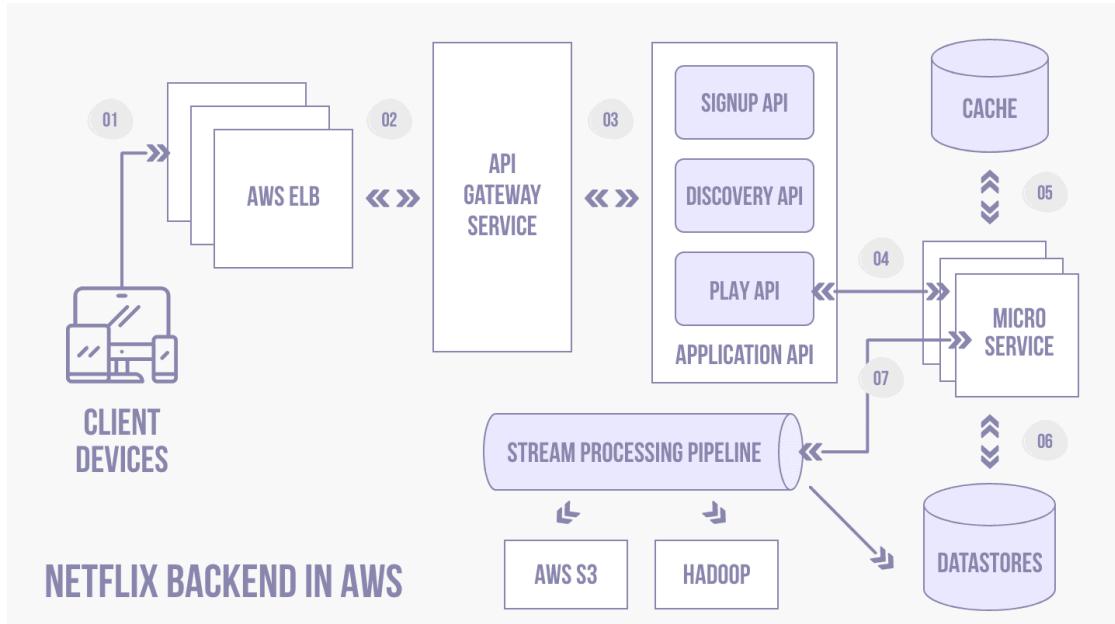
În ciuda numeroaselor sale avantaje, arhitectura microserviciilor nu este fără provocări. Acestea includ complexități legate de coordonarea serviciilor, consistența datelor și securitate, care necesită un design atent. În plus, tranziția de la o arhitectură monolică la una bazată pe microservicii poate fi o sarcină substanțială, necesitând timp și resurse semnificative.

Deși domeniul dezvoltării de aplicații web bazate pe microservicii utilizând cadrul Spring Boot este în creștere și evoluție rapidă, acesta prezintă încă provocări care necesită cercetare și inovație continuă. Industria se orientează către o orchestrare a serviciilor mai eficientă, soluții de securitate îmbunătățite și metode eficiente de gestionare a consistentei datelor. Cu aceste eforturi continue, se așteaptă ca domeniul să continue să înflorească, oferind oportunități și mai mari pentru construirea de aplicații web flexibile, scalabile și reziliente.

## 1.2 Stadiul Pieței existente

### 1.2.1 Netflix

Netflix, un gigant global în industria de streaming, a fost în avangarda adoptării și inovării cu arhitectura microserviciilor, cu Spring Boot fiind o componentă vitală a stivei lor tehnologice. Echipa de ingineri de la Netflix a dezvoltat o suită de software open-source, inclusiv Zuul 2, un gateway de servicii esențial în ecosistemul serviciilor lor. Deși Zuul 2 nu este construit direct cu Spring Boot, el face parte din proiectul Spring Cloud Netflix, care oferă integrări Spring Cloud pentru componente Netflix. Acest lucru permite dispozitivelor și navigatoarelor web să mențină conexiuni persistente cu Netflix, îmbunătățind performanța și permitând o mai bună înțelegere și depănare a experienței utilizatorului.

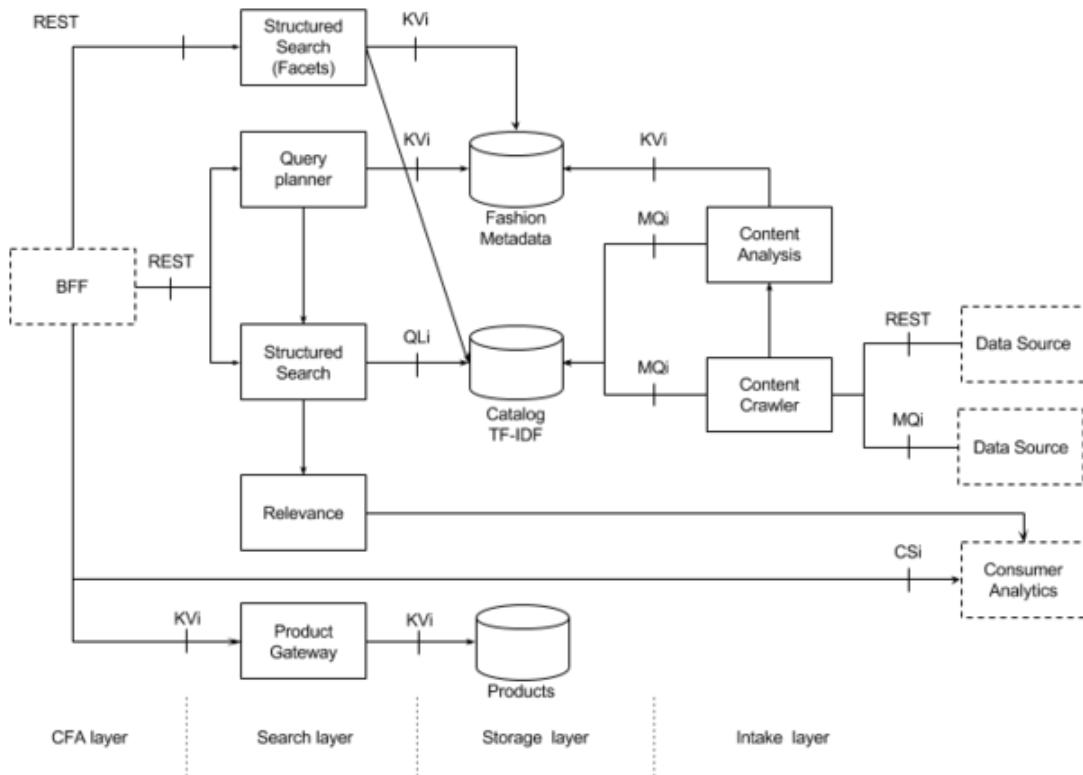


Figură 1.1 posibilă arhitectură Netflix

Netflix a fost de asemenea un contribuitor semnificativ la Spring Cloud, un instrument folosit împreună cu Spring Boot pentru dezvoltarea de microservicii. Spring Cloud oferă un set de instrumente pentru descoperirea serviciilor, rutarea și implementarea de microservicii în mediu cloud, iar contribuțiile Netflix au jucat un rol esențial în evoluția acestui ecosistem. Această colaborare între Netflix și Spring Cloud subliniază importanța Spring Boot în dezvoltarea de microservicii robuste și scalabile.

### 1.2.2 Zalando

Zalando, una dintre cele mai mari platforme de modă online din Europa, a realizat cu succes tranziția de la o arhitectură monolică la una bazată pe microservicii, Spring Boot fiind un element cheie în această transformare. Echipa de ingineri de la Zalando a adoptat Spring Boot datorită principiului său "convenție peste configurare", care simplifică considerabil dezvoltarea și implementarea microserviciilor.

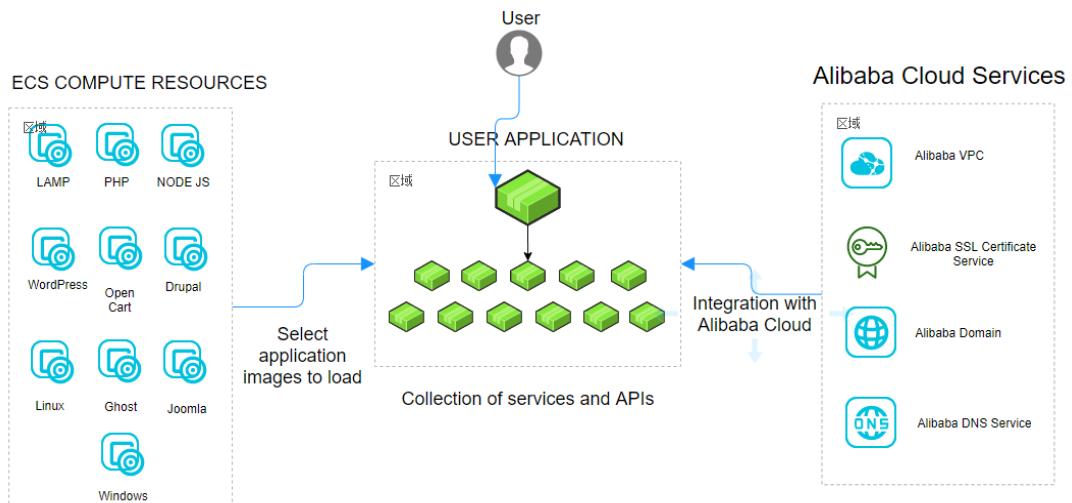


Figură 1.2 arhitectură microservicii Zalando

Utilizând serverul încorporat, configurarea automată și capacitatele de implementare autonomă ale Spring Boot, Zalando a reușit să dezvolte și să implementeze microservicii independente. Acest lucru a permis o scalabilitate îmbunătățită și o izolare eficientă a erorilor. De asemenea, Zalando folosește Spring Boot Actuator, un modul care oferă funcționalități de producție, cum ar fi monitorizarea stării aplicației și colectarea de metrii, esențiale pentru monitorizarea și gestionarea microserviciilor într-un mediu de producție.

### 1.2.3 Alibaba

Alibaba, unul dintre liderii globali în comerțul electronic, a adoptat Spring Boot și Spring Cloud Alibaba pentru dezvoltarea aplicațiilor sale distribuite. Capacitatea Spring Boot de a configura rapid aplicații Spring gata pentru producție a fost crucială pentru Alibaba pentru a gestiona volumul imens de tranzacții zilnice.



Figură 1.3 arhitectură microservicii Alibaba

Prin utilizarea Spring Boot și Spring Cloud Alibaba, Alibaba a demonstrat puterea acestor tehnologii atunci când sunt folosite împreună. Spring Boot oferă cadrul pentru dezvoltarea rapidă a microserviciilor, în timp ce Spring Cloud Alibaba oferă o suită de componente specifice Alibaba pentru descoperirea serviciilor, gestionarea configurațiilor și gestionarea mesajelor. Prin integrarea acestor funcționalități în propriul ecosistem, Alibaba a putut să creeze aplicații bazate pe microservicii într-un mod rapid și eficient.

#### 1.2.4 Instagram

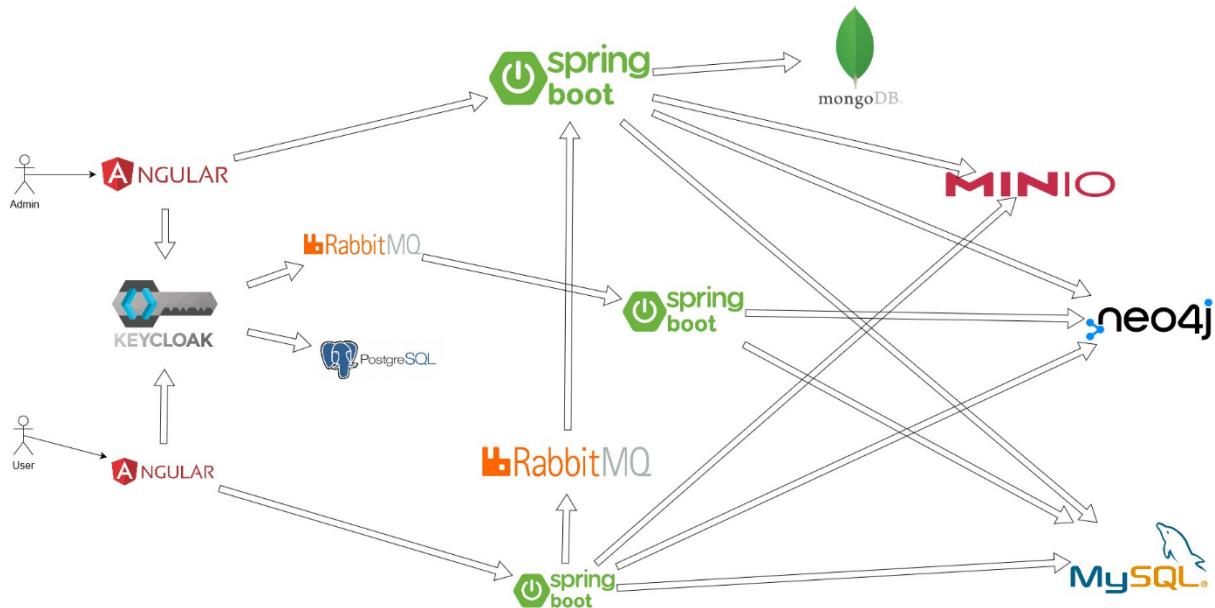
Instagram, o platformă globală de social media renumită, deși nu folosește Spring Boot, este o rețea de socializare care a adoptat în mod notabil arhitectura microserviciilor. Infrastructura de backend este construită pe o multitudine de microservicii care gestionează diferite aspecte ale platformei, de la profiluri de utilizatori la încărcarea de fotografii și fluxul de stiri.

Principiile de bază ale Instagram atunci când aleg un sistem sunt de a-l menține foarte simplu, de a nu reinventa roata și de a merge cu tehnologii solide și dovedite atunci când este posibil. Ei rulează Ubuntu Linux pe Amazon EC2, cu fiecare cerere către serverele Instagram trecând prin mașini de echilibrare a solicitării.

Majoritatea datelor Instagram, inclusiv utilizatori, metadate foto, etichete etc., se află în PostgreSQL. Au constatat că sistemul de disc în rețea al Amazon (EBS) nu suportă suficiente căutări pe disc pe secundă, așa că este extrem de important să aibă tot setul lor de lucru în memorie. De asemenea, folosesc Redis pe scară largă, care alimentează fluxul lor principal, fluxul de activitate, sistemul lor de sesiuni și alte sisteme conexe.

Deși backend-ul Instagram nu este construit în mod specific cu Spring Boot, principiile și practicile pe care le-au aplicat în arhitectura lor de microservicii se aliniază îndeaproape cu cele susținute de cadrul Spring Boot. De exemplu, microserviciile Instagram sunt implementabile și scalabile în mod independent, iar ele comunică între ele prin intermediul unor API-uri bine definite - toate acestea fiind principii cheie în microserviciile bazate pe Spring Boot.

## CAPITOLUL 2 CERINȚE ȘI SPECIFICAȚII



*Figură 2.1 Diagrama cluster-ului*

Aplicația web Kiire (în finlandeză însemnând ceva care ocupă timpul) este arhitecturată ca un mediu de microservicii implementat pe un cluster Kubernetes folosind framework-ul Spring Boot, Gradle și Java pentru dezvoltarea backend-ului. Frontend-ul este construit folosind Angular cu TypeScript, Angular Material și NGINX.

Aplicația prezintă o platformă de socializare în care utilizatorii pot efectua diverse operații CRUD asupra postărilor, pot vizualiza profilurile altor utilizatori, iar administratorii pot efectua sarcini administrative, cum ar fi moderarea postărilor. Aplicația găzduiește trei roluri distincte de utilizatori: client, admin și administrator de sistem, fiecare oferind capacitați unice.

Aplicația necesită un cluster Kubernetes operațional pentru a permite implementarea și scalarea microserviciilor implicate. Clusterul trebuie să găzduiască trei aplicații Spring Boot, două aplicații Angular, microservicii RabbitMQ separate, Keycloak cu keycloakify și un set de baze de date, inclusiv MySQL, PostgreSQL, Neo4J, MongoDB și Minio.

În ceea ce privește rețeaua, expunerea externă trebuie să fie limitată la Keycloak, aplicația Angular pentru client și aplicația Angular pentru admin. Această expunere este facilitată prin intermediul unui Ingress, asigurând un rutaj controlat și securizat al rețelei.

Ca parte a backend-ului aplicației, programarea orientată pe aspecte ar trebui utilizată pentru a înregistra informații despre metodele executate. Aceste informații trebuie transmise printr-o instanță RabbitMQ către backend-ul adminului, unde sunt stocate în MongoDB. Acest lucru creează un jurnal de audit și oferă o perspectivă în timp real asupra activității aplicației pentru administratori.

În interesul securității robuste, autentificarea și autorizarea sunt implementate prin intermediul Keycloak. Pentru a sincroniza conturile Keycloak cu conturile din Neo4J și

MySQL, este implementat un plugin Keycloak RabbitMQ. Această configurare permite unei aplicații separate Spring Boot să captureze evenimente de înregistrare și să sincronizeze datele utilizatorilor în întreaga rețea de servicii.

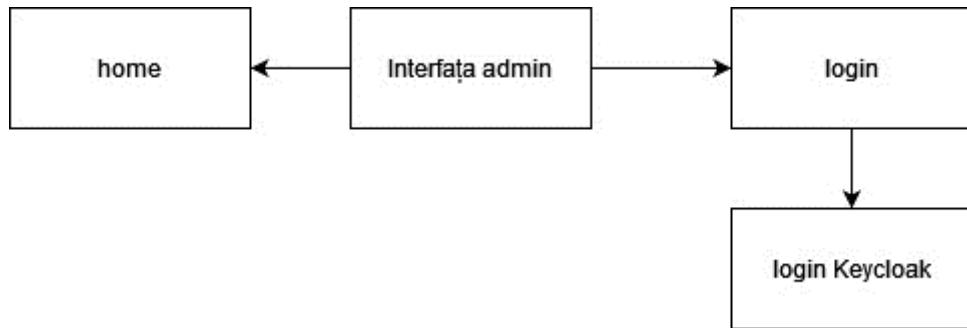
Pentru stocarea datelor, aplicația utilizează un set divers de baze de date pentru a se adapta nevoilor specifice. Conținutul postărilor și metadatele asociate fișierelor postărilor sunt stocate în MySQL, în timp ce relațiile dintre postări sunt gestionate în Neo4J pentru a permite interogări mai complexe, iar fișierele sunt stocate în Minio .

Rolurile utilizatorilor și capacitatele lor în cadrul aplicației sunt definite în mod clar:

Rolul "client" include capacitatea de a crea un cont, de a se autentifica și deconecta, de a-și actualiza profilul, de a efectua operații CRUD asupra postărilor proprii, de a vizualiza postările altor utilizatori, de a raporta postări, de a-și schimba parola și de a vizualiza profilul oricărui utilizator, inclusiv al lor.

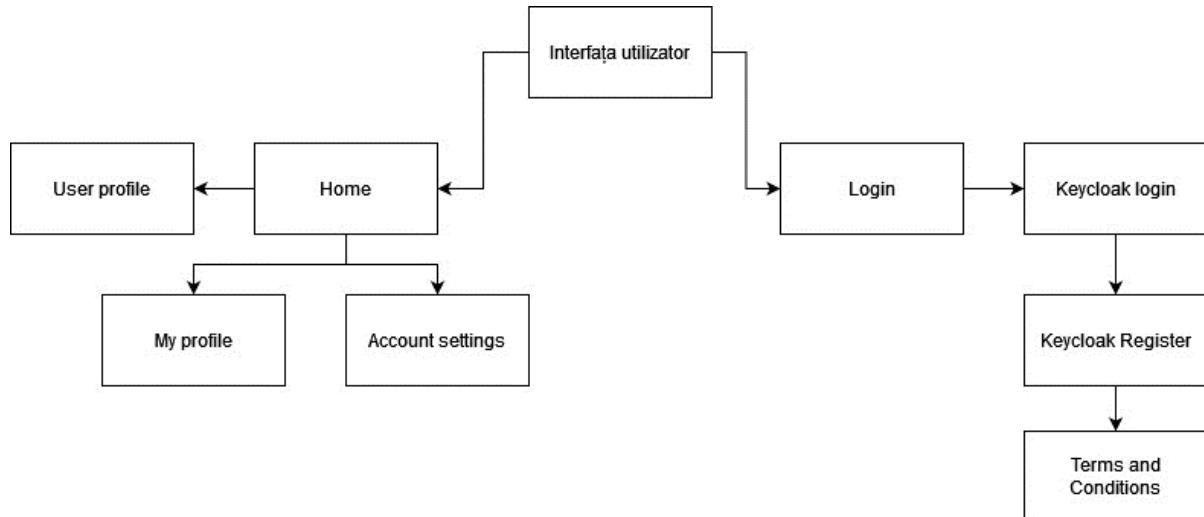
Rolul "admin" permite utilizatorilor să se autentifice, să modereze postările raportate prin confirmarea sau negarea încălcărilor regulilor și să vizualizeze informații în timp real provenite de la backend-ul clientului. Această funcționalitate este realizată prin intermediul unei API reactiv, deși jurnalele pot fi accesate și prin intermediul unei API-uri RESTful.

Rolul "administrator de sistem" oferă acces cuprinsător la aspectele operaționale ale microserviciilor care rulează în cadrul clusterului. Aceasta include acces la consola de administrare Keycloak, PgAdmin, Adminer, phpMyAdmin, Mongo Express, consola RabbitMQ și consola Minio . Acest rol facilitează în primul rând sarcinile de gestionare și întreținere la nivel de sistem.

**CAPITOLUL 3****PROIECTAREA SISTEMULUI****3.1 Proiectarea Aplicației*****3.1.1 Diagrama Site-ului***

Figură 3.1diagrama site-ului admin

Aplicația de admin pornește prin a prezenta pagina de login din care adminul va fi redirectionat spre pagina de conectare din Keycloak, iar în urma unei conectări cu succes, va fi redirectionat spre pagina home.



Figură 3.2diagrama site-ului utilizatorului

Aplicația client pornește prin a prezenta pagina de login din care clientul va fi redirectionat spre pagina de conectare din Keycloak de unde se poate conecta sau se poate duce la pagina de înregistrare. În urma înregistrării, utilizatorul se va afla în pagina de acceptare a termenilor și condițiilor, iar în urma unei conectări cu succes, va fi redirectionat spre pagina home.

**3.2 Prezentarea și Modelarea Bazei de Date**

Stratul de date al aplicației web este conceput în detaliu, asigurând utilizarea adecvată a sistemului de gestiune al bazelor de date (DBMS) în funcție de fiecare caz

specific. Aceasta are ca scop optimizarea performanței și fiabilității aplicației și gestionarea eficientă a relațiilor complexe între date. Această strategie utilizează o abordare de persistență poliglotă, bazată pe înțelegerea că un singur tip de DBMS este adesea insuficient pentru aplicații complexe și că diferite tipuri de date sunt gestionate mai bine de diferite tipuri de baze de date.

Datele sunt distribuite în diferite baze de date precum PostgreSQL, MySQL, MongoDB, Neo4j și Minio. Această distribuție este următoarea:

**PostgreSQL:** Această bază de date relațională este utilizată pentru a stoca datele din Keycloak, serviciul de gestionare a identității și accesului (IAM) utilizat în aplicație. Robustea, tranzacțiile atomice și limbajul de interogare puternic al PostgreSQL îl fac ideal pentru gestionarea datelor IAM complexe și sensibile.

**MongoDB:** Este utilizată o bază de date orientată pe documente NoSQL, MongoDB, pentru gestionarea datelor de jurnal. Acest tip de date are în mod tipic o varietate de structuri și încarcături ridicate de scriere, ceea ce face ca MongoDB, cu schema sa flexibilă și scrierile rapide, să fie cea mai potrivită soluție.

**MySQL:** Această bază de date relațională este utilizată pentru a stoca datele de bază ale aplicației, inclusiv informații adiționale despre utilizatori, informații despre postări, metadate despre fișiere și date de raportare. MySQL a fost ales pentru fiabilitatea sa ridicată, robustețe și suportul puternic pentru date structurate.

**Minio :** Este utilizat un serviciu de stocare de obiecte, Minio , pentru a stoca fișierele asociate postărilor și imaginilor de profil. Acest sistem de stocare cu performanță ridicată este compatibil cu S3 și oferă o soluție excelentă pentru stocarea datelor mari și nestructurate, cum ar fi imaginile.

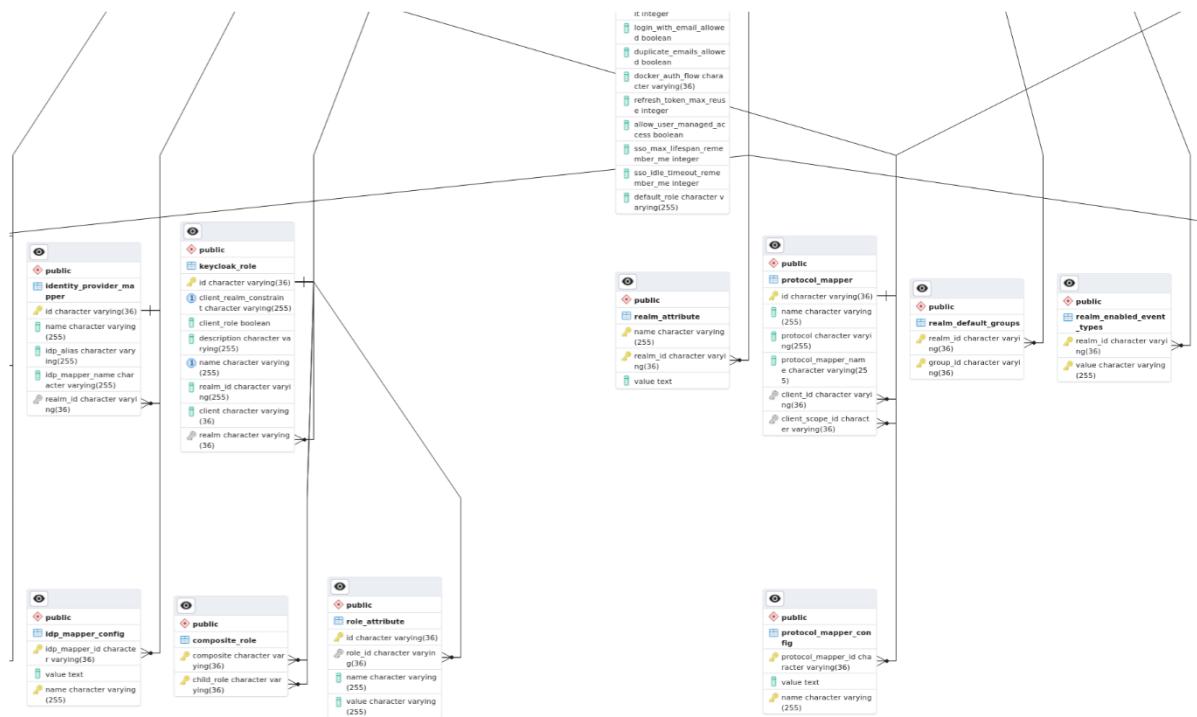
**Neo4j:** Această bază de date graf este utilizată pentru a stoca relațiile dintre utilizatori și postări. Interrogările complexe care necesită înțelegerea relațiilor profunde sunt efectuate în Neo4j pentru a recupera ID-urile utilizatorilor și postărilor relevante. Cu toate acestea, datele reale sunt preluate din MySQL, valorificând punctele forte ale ambelor baze de date.

Sincronizarea între PostgreSQL, Neo4j și MySQL asigură consistența și integritatea datelor în cadrul bazelor de date. Aceasta implică o orchestrare atentă a operațiilor de scriere și actualizare pentru a menține sincronizate diferitele baze de date, oferind aplicației o viziune consistentă asupra datelor.

Prin utilizarea acestor baze de date diferite, se abordează complexitatea diferitelor cerințe de date, asigurând în același timp performanță, fiabilitate și scalabilitate ridicate. Această metodologie de persistență poliglotă permite aplicației să valorifice punctele forte ale fiecărui tip de bază de date, oferind o soluție de gestionare a datelor optimizată, robustă și cuprinzătoare.

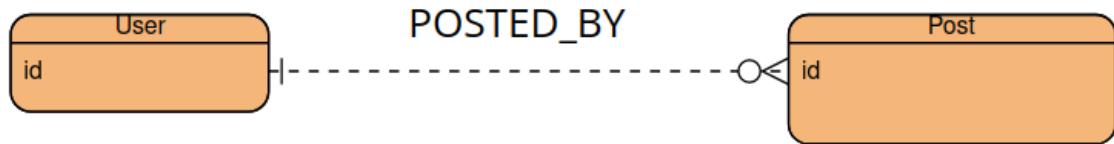
### 3.2.1 Nivelul Conceptual

Deși Keycloak își gestionează în mod automat baza de date fără a fi nevoie de intervenție manuală, aceasta poate fi accesată pentru a vizualiza anumite date, deși majoritatea datelor pot fi deja vizualizate în consola de admin. Detalierea acestei baze de date se află în afara scopului acestei lucrări, dar este de menționat faptul că se rețin date cu privire la elemente compatibile cu oauth2 precum tărmuri, roluri, cliente și multe altele, rezultând în peste 30 de tabele cu dimensiune de la câteva coloane la peste 20 de coloane. De asemenea se rețin date precum sesiuni active, evenimente emise atât de administrator, cât și utilizatori.

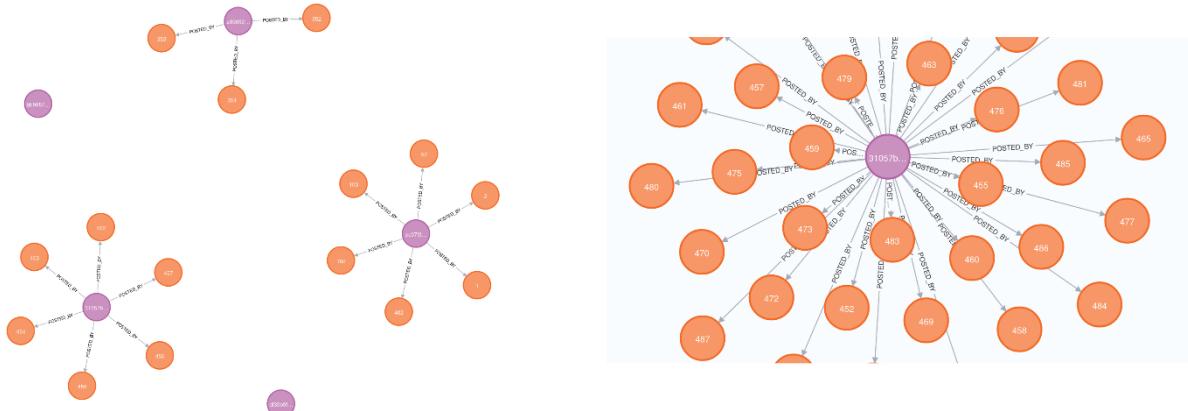


Figură 3.3 fragment orientativ din modelul conceptual pentru schema bazei de date Postgres folosită de Keycloak

Neo4j este folosit strict pentru a declara relațiile între entități. În acest moment singurul tip de relație care există este cea dintre postare și utilizator.

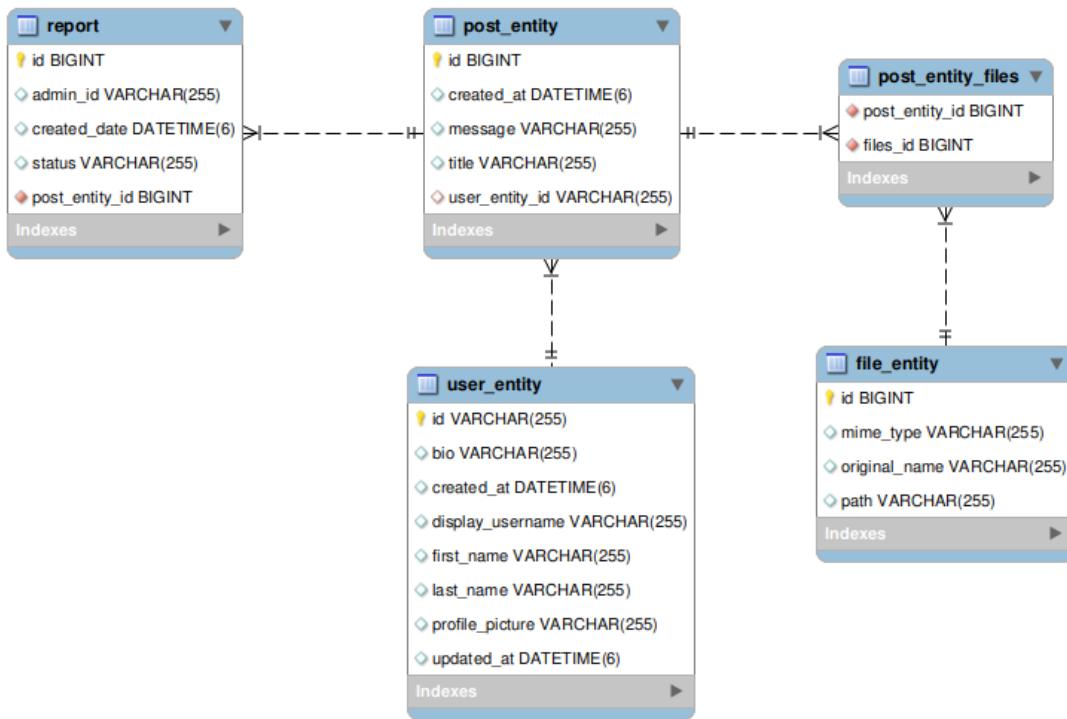


Figură 3.4 Modelul conceptual al bazei de date Neo4j



Figură 3.5 exemplu bază de date Neo4j populate și exemplu utilizator cu multe postări

În MySQL s-a preferat menținerea relațiilor dintre o postare și fișierele sale deoarece singurele interogări posibile asupra fișierelor se fac doar în contextul postării în care există. De asemenea s-a preferat menținerea relațiilor între postare și utilizatori deoarece în cazul interogărilor simple precum a vedea un utilizator și propriile postări se poate folosi calitatea relațională a bazei de date, iar doar în cazurile complexe să se folosească neo4j.



Figură 3.6 model conceptual MySQL

### 3.2.2 Nivelul Fizic al Bazei de Date ( MCD )

Pentru a asigura scalabilitatea sistemului și a gestiona eficient cazul în care un utilizator are un număr mare de postări, s-a ales să se definească relația în sensul "postarea este creată de utilizator" în loc de "utilizatorul a creat postarea". Acest lucru permite evitarea stocării unei liste mari de postări în interiorul fiecărui nod de utilizator, care ar putea deveni prea mare și ar afecta performanța sistemului. În schimb, fiecare postare își cunoaște autorul, ceea ce permite o gestionare mai eficientă a datelor.

```

@Node
@Data
public class PostNode {
    @Id
    private final Long id;

    @Relationship(type = "POSTED_BY", direction = Relationship.Direction.INCOMING)
    private UserNode author;

    @Relationship(type = "LIKED_BY", direction = Relationship.Direction.INCOMING)
    private List<UserNode> likedByUsers=new ArrayList<>();
}
    
```

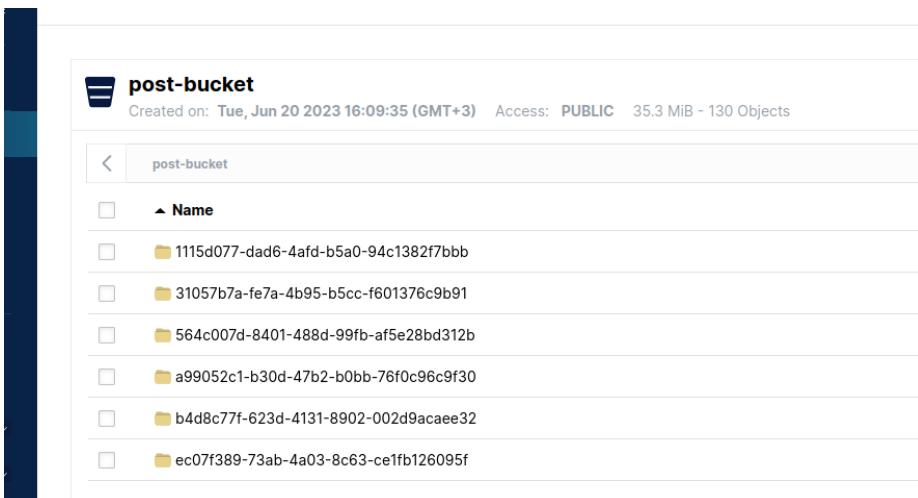
Mongodb nu are o schemă precum bazele de date relaționale, dar datele ce reprezintă logarea informațiilor sunt salvate precum documente json ce conțin chei comune precum id, data când a fost emis de serverul de client, data când a fost primit de serverul de admin, fiind o întârziere dată de comunicarea datelor prin RabbitMQ, și mesajul logării.

The screenshot shows the MongoDB UI interface for the database 'dbadmin'. The collection is 'logEntry'. The 'Schema' tab is selected. At the top, there is a search bar with the placeholder 'Type a query: { field: 'value' }'. Below the search bar, there is a button labeled 'Filter' with a dropdown arrow and a note 'Looking for data modeling tools?'. The schema is defined by five fields:

- \_id**: objectid
- \_class**: string
- createdAt**: string
- message**: string
- receivedAt**: string

Figură 3.7 proprietăți logare în mongodb

Deși Minio nu este o bază de date, ci un sistem de stocat obiecte, este de menționat faptul ca fișierele postărilor sunt salvate într-un bucket, având un director separat pentru fiecare utilizator, numele directorului fiind dat de identificatorului unic al utilizatorului. Meta data fișierelor este salvată în MySQL, iar în Minio fișierele sunt salvate cu un identificator unic urmat de numele original al fișierului astfel încât să nu apară conflicte în cazul în care fișiere, chiar și din postări diferite, au același nume.



Figură 3.8 exemplu împărțire a fișierelor postărilor în funcție de utilizator

### 3.3 Prezentarea API endpoints

Pentru PostController din aplicația Spring Boot a clientului:

3.3.1.1 Tabelul 1.1 PostController

Endpoint	Metodă HTTP	Exemplu corp cerere	Exemplu corp răspuns
/api/v1/posts	GET		[{"id": 1, "title": "Post 1", "message": "This is post 1", "files": [], "userEntity": {}, "createdAt": "2023-07-06T00:00:00"}, {"id": 2, "title": "Post 2", "message": "This is post 2", "files": [], "userEntity": {}, "createdAt": "2023-07-06T00:00:00"}]
/api/v1/posts/{id}/report	POST		
/api/v1/posts	POST	{"title": "New Post", "message": "This is a new post", "files": []}	{"id": 3, "title": "New Post", "message": "This is a new post", "files": [], "userEntity": {}, "createdAt": "2023-07-06T00:00:00"}
/api/v1/posts/{id}	GET		{"id": 1, "title": "Post 1", "message": "This is post 1", "files": [], "userEntity": {}, "createdAt": "2023-07-06T00:00:00"}
/api/v1/posts/{id}	DELETE		
/api/v1/posts/{id}	PUT	{"title": "Updated Post", "message": "This is an updated post", "fileIdsToKeep": [], "newFiles": []}	{"id": 1, "title": "Updated Post", "message": "This is an updated post", "files": [], "userEntity": {}, "createdAt": "2023-07-06T00:00:00"}

Pentru UserController din aplicația Spring Boot a clientului:

3.3.1.1.2 Tabelul 2.1 UserController

Endpoint	Metodă HTTP	Exemplu corp cerere	Exemplu corp răspuns
/api/v1/users/{id}	GET		{"id": "123", "displayUsername": "user1", "firstName": "John", "lastName": "Doe", "bio": "Hello, I'm John", "profilePicture": "profile.jpg", "createdAt": "2023-07-06T00:00:00", "updatedAt": "2023-07-06T00:00:00"}
/api/v1/users/{id}	PUT	{"displayUsername": "user1", "firstName": "John", "lastName": "Doe", "bio": "Hello, I'm John", "profilePicture": "profile.jpg"}	{"status": 200, "message": "OK"}
/api/v1/users/{id}/posts	GET		{"content": [{"id": 1, "title": "Post title", "message": "Post message", "files": [{"id": 1, "path": "file.jpg", "mimeType": "image/jpeg", "originalName": "file.jpg"}], "userEntity": {"id": "123", "displayUsername": "user1", "firstName": "John", "lastName": "Doe", "bio": "Hello, I'm John", "profilePicture": "profile.jpg", "createdAt": "2023-07-06T00:00:00", "updatedAt": "2023-07-06T00:00:00"}, "createdAt": "2023-07-06T00:00:00"}, {"pageable": {"sort": {"sorted": true, "unsorted": false, "empty": false}, "offset": 0, "pageSize": 10, "pageNumber": 0, "paged": true, "unpaged": false}, "totalPages": 1, "totalElements": 1, "last": true, "size": 10, "number": 0, "sort": {"sorted": true, "unsorted": false, "empty": false}, "numberOfElements": 1, "first": true, "empty": false}}

Pentru LogEntryController din aplicația Spring Boot a adminului:

3.3.1.1.3 Tabelul 3.1 LogEntryController

Endpoint	Metodă HTTP	Exemplu corp cerere	Exemplu corp răspuns
----------	-------------	---------------------	----------------------

/api/v1/rest/logs/{id}	GET		{ "id": "123", "createdAt": "2023-07-06T12:34:56Z", "receivedAt": "2023-07-06T12:34:56Z", "message": "Log message" }
/api/v1/rest/logs	GET		{ "content": [ { "id": "123", "createdAt": "2023-07-06T12:34:56Z", "receivedAt": "2023-07-06T12:34:56Z", "message": "Log message" } ], "pageNumber": 0, "pageSize": 100, "totalElements": 200, "totalPages": 2 }

Pentru ReportController din aplicația Spring Boot a adminului:

3.3.1.1.4 Tabelul 4.1 ReportController

Endpoint	Metodă HTTP	Exemplu corp cerere	Exemplu corp răspuns
/api/v1/reports/{report_id}/review	POST	{ "violation": true }	
/api/v1/reports			{ "id": 1, "status": "PENDING", "post": { "postId": 123, "content": "Post content", "createdAt": "2023-07-06T12:34:56Z" }, "createdDate": "2023-07-06T12:34:56Z" }

Pentru WebSocketConfig din aplicația Spring Boot a adminului:

3.3.1.1.5 Tabelul 5.1 WebSocketConfig

Endpoint	Metodă Websocket	Exemplu corp cerere	Exemplu corp răspuns
/api/v1/reactive/logs	CONNECT		
/api/v1/reactive/logs	MESSAGE		{ "id": "123", "createdAt": "2023-07-06T12:34:56Z", "receivedAt": "2023-07-06T12:34:56Z", "message": "Log message" }

## CAPITOLUL 4      IMPLEMENTAREA SISTEMULUI ȘI TEHNOLOGII

### 4.1 Instrumentele Informaticice Utilizate în Elaborarea Aplicației

Acest proiect a folosit un număr de unelte și tehnologii sofisticate, fiecare fiind aleasă în mod deliberat pentru avantajele sale specifice în contextul unei arhitecturi de microservicii, contribuind la implementarea cu succes a aplicației.

Controlul versiunilor a fost gestionat prin intermediul Git, un sistem renomit pentru urmărirea modificărilor în codul sursă în timpul dezvoltării software. Acest instrument oferă funcții precum ramuri pentru a izola și combina modificările. GitHub, un serviciu de găzduire a repository-urilor Git, a fost utilizat ca platformă pentru a stoca și partaja codul proiectului.

Linux Mint, o distribuție Linux bazată pe Ubuntu, a fost sistemul de operare ales datorită capacitatea sale de a gestiona tunelurile Minikube. Această caracteristică facilitează accesul la serviciile expuse prin adresa lor, în loc de a redirecționa toate serviciile către 127.0.0.1, aşa cum se întâmplă în Windows, evitând posibilele conflicte de porturi.

IntelliJ și Visual Studio Code au fost folosite ca medii de dezvoltare integrate (IDE-uri). IntelliJ, renomit pentru experiența sa imediată după instalare, a oferit un suport robust pentru Java și Spring Boot, în timp ce Visual Studio Code, cu arhitectura sa ușoară și biblioteca mare de extensii, a oferit suport cuprinzător pentru dezvoltarea în TypeScript și Angular.

Docker a fost utilizat pentru containerizare, permitând aplicației să ruleze în mod fiabil în diferite medii de calcul. Platforma creează containere ușoare pentru aplicații software care pot fi ușor gestionate și scalate. Acest lucru s-a îmbinat perfect cu Minikube, un instrument care facilitează rularea locală a Kubernetes. Kubernetes în sine, gestionat prin interfața de linie de comandă kubectl, a servit ca un instrument de orchestrare, automatizând implementarea, scalarea și gestionarea aplicațiilor containerizate pe întreg clusterul.

Angular CLI, o interfață de linie de comandă, a fost instrumental în automatizarea sarcinilor de dezvoltare în cadrul framework-ului Angular, accelerând procesul de configurare și dezvoltare. Node.js, un mediu de execuție JavaScript, și npm, managerul său de pachete, au oferit mediul pentru a gestiona dependențele pentru aplicațiile Angular.

Postman, un client API popular, a facilitat testarea API-urilor, permitând trimiterea de cereri către aplicație și verificarea răspunsurilor.

În orchestrarea aplicației, Skaffold a fost o altă unealtă cheie alături de Minikube, kubectl, Helm și Kustomize. Skaffold este o unealtă în linia de comandă care facilitează dezvoltarea continuă pentru aplicațiile Kubernetes. Funcționalitatea sa include construirea și implementarea automată a aplicațiilor într-un mediu Kubernetes ori de câte ori se face o modificare în codul sursă.

Kustomize a fost utilizat împreună cu Skaffold pentru gestionarea personalizărilor resurselor Kubernetes. Kustomize, o altă soluție nativă Kubernetes pentru configurare, a permis modificarea fișierelor YAML pentru resursele Kubernetes, menținând configurația aproape de implementare, ceea ce a îmbunătățit lizibilitatea și întreținerea. Helm și Kustomize, când sunt combinate, permit gestionarea eficientă și controlată a configurației și implementării în ecosistemul Kubernetes.

În cele din urmă, MySQL Workbench a fost folosit pentru proiectarea și gestionarea bazei de date MySQL. Acest instrument vizual a facilitat crearea și gestionarea bazei de date relationale a aplicației.

## 4.2 Limbaje folosite

În dezvoltarea acestei aplicații web, s-a folosit o gamă diversă de limbaje de programare și limbaje de serializare a datelor, fiecare ales pentru punctele sale forte și capabilități specifice.

Limbajul principal utilizat pentru microserviciile de back-end este Java, ales pentru caracteristicile orientate pe obiect, librăriile open-source extinse și suportul larg al comunității. În mod specific, microserviciile au fost construite folosind framework-ul Spring Boot, care folosește platforma Spring împreună cu conveniențe suplimentare pentru a accelera procesul de creare a aplicațiilor autonome, de grad de producție. Siguranța tipurilor din Java și gestionarea predictibilă a erorilor reduc semnificativ riscul erorilor la runtime, sporind fiabilitatea microserviciilor.

Complementând Java, Groovy a fost utilizat în sistemul de build Gradle, profitând de expresivitatea și flexibilitatea sa pentru a scripta construirea microserviciilor bazate pe Java. Groovy este complet compatibil cu Java, dar include funcționalități suplimentare precum tipare dinamice și închideri care simplifică și accelerează sarcinile de scripting. Integrarea fără probleme a Groovy cu Java și capabilitățile sale puternice de scripting l-au făcut o alegere potrivită pentru sistemul de build al proiectului.

HTML5, un limbaj standard de marcare pentru construirea paginilor web, a fost integral în crearea interfețelor interactive ale aplicațiilor, datorită simplității sale sintactice și compatibilității extinse care permit crearea unui conținut structurat și semnificativ. În același timp, Cascading Style Sheets (CSS) au fost utilizate pentru stilizarea vizuală și structura aplicațiilor. Acest lucru a permis definirea culorilor, fonturilor, dimensiunilor și poziționării personalizate a elementelor, asigurând astfel un aspect și o experiență coerentă în diferite părți ale aplicațiilor. În plus, Sassy CSS (SCSS), un preprocesor CSS, a fost folosit pentru gestionarea fișierelor de stil. Chiar dacă acest proiect nu a beneficiat de funcționalitățile extinse ale SCSS, cum ar fi variabilele sau imbricarea claselor de stil, compatibilitatea SCSS cu toate versiunile CSS a fost benefică. Astfel, CSS-ul obișnuit a putut fi scris în interiorul fișierelor SCSS, oferind potențial pentru scalabilitatea viitoare în cazul în care funcționalitățile avansate ale SCSS vor fi incluse în iterările ulterioare ale aplicației. Prin urmare, chiar și cu utilizarea minimă a funcționalităților specifice SCSS, includerea acestuia a adăugat potențial pentru îmbunătățirea codului și o mai bună menținere a acestuia.

Pentru dezvoltarea front-end-ului aplicației, au fost utilizate în primul rând două limbaje de programare: JavaScript și TypeScript. JavaScript, temelia dezvoltării web, a fost folosit pentru natura sa dinamică, funcțiile de clasă întâi și prezența nelipsită în rândul browserelor web. Modelul său I/O non-blocking și bazat pe evenimente este potrivit pentru dezvoltarea de aplicații web interactive, precum aplicația de social media descrisă în acest proiect.

TypeScript, un superset static tipizat al JavaScript, a fost folosit în conjuncție cu framework-ul Angular pentru dezvoltarea front-end-ului aplicației. TypeScript îmbunătățește JavaScript cu tipuri statice, clase și interfețe, care ajută la prinderea erorilor în timpul compilării în loc de runtime, o caracteristică neprețuită pentru aplicațiile mari, complexe. În plus, compatibilitatea TypeScript cu Angular l-a făcut alegerea naturală pentru acest proiect.

În final, YAML, un limbaj de serializare a datelor lizibil pentru oameni, a fost folosit pentru gestionarea configurației în Kubernetes. Deși nu este un limbaj de programare propriu-zis, sintaxa clară și concisă a YAML, împreună cu capacitatea sa de a reprezenta structuri de date complexe, îl face o alegere optimă pentru fișierele de configurație. Kubernetes, care este în centrul orchestrării microserviciilor, se bazează în mare măsură

pe YAML pentru definirea și gestionarea resurselor sale, făcând din YAML o parte integrală a stivei noastre de aplicații.

#### 4.3 Framework

În dezvoltarea aplicației web, s-au folosit mai multe framework-uri pentru avantajele și capacitatele lor unice, toate facilitând implementarea diverselor aspecte ale proiectului.

Începând cu Spring Boot, un framework bazat pe Java, folosit în mod extensiv pentru crearea de aplicații Spring independente și ușor de implementat. Java, un limbaj de programare bazat pe clase și orientat spre obiecte, conceput pentru a avea dependențe de implementare minime, servește ca limbaj de bază pentru acest framework. Prezența sa răspândită în aplicațiile de nivel enterprise, împreună cu performanța și caracteristicile de securitate, l-a făcut o alegere optimă pentru backend. Spring Boot completează Java prin eliminarea unei mare părți din codul boilerplate asociat cu configurarea unei aplicații Spring, simplificând gestionarea dependențelor și furnizând un server încorporat, ceea ce elimină nevoia de implementare pe un server extern.

În aplicația Spring Boot pentru client s-a folosit Spring AOP ( aspect oriented programming ), o componentă crucială în ecosistemul Spring Boot. AOP facilitează separarea transversală a intereselor, cum ar fi jurnalizarea, permitând un focus mai clar asupra logicii centrale. În cadrul aplicației noastre, Spring AOP a fost utilizat pentru a defini "advices" care se declanșează la fiecare apel al unor metode din clasele de servicii. Acestea sunt identificate printr-un sistem de "pointcut"-uri, care permit gestionarea în mod centralizat și eficient informațiile logistice, ajutând adminii să monitorizeze comportamentul aplicației.

Un alt framework esențial folosit în dezvoltarea părților de client și administrare a aplicației este Angular, o platformă bazată pe TypeScript pentru construirea aplicațiilor web. TypeScript, un superset static tipizat de JavaScript, oferă tipizare statică optională, programare orientată pe obiecte bazată pe clase și capacitate îmbunătățite de instrumentare. Angular folosește capacitatele TypeScript și le îmbunătățește cu ecosistemul său dezvoltat, arhitectura bazată pe componente și funcționalități încorporate, cum ar fi serviciile HTTP și gestionarea formularelor. Caracterul său cuprinzător permite o structură organizată și crearea de aplicații single-page complexe, reactive, cu mai puțin cod.

În domeniul gestionării persistenței stocării, s-a folosit framework-ul Hibernate, care lucrează în sinergie cu Java Persistence API (JPA). Hibernate, un framework pentru maparea unui model de domeniu orientat pe obiecte la o bază de date relațională tradițională, utilizează JPA, specificația standard Java pentru accesarea, persistența și gestionarea datelor între obiectele Java și o bază de date relațională. Ușurința de utilizare a Hibernate, precum și integrarea sa fără probleme cu Spring Boot, îl fac o alegere excelentă pentru gestionarea stratului de baze de date al aplicației.

În final, React, folosit în cadrul configurației Keycloakify, a fost crucial în crearea unui UI interactiv. React este o bibliotecă JavaScript folosită în principal pentru construirea interfețelor de utilizator, în special aplicații de tip single-page unde este nevoie de o interfață de utilizator rapidă și interactivă. Alegerea React a fost motivată de eficiență și flexibilitatea sa, precum și de capacitatea sa de a crea interfețe de utilizator complexe și interactive într-un modメンtenabil și scalabil. În contextul acestei aplicații, React a fost utilizat împreună cu Keycloakify pentru a crea o interfață de autentificare personalizată. Keycloakify utilizează puterea React pentru a permite personalizarea ușoară a temelor Keycloak.

#### 4.4 Librării

Construcția aplicației noastre web s-a bazat în mare parte pe o varietate de biblioteci. Fiecare a fost selectată cu grijă pentru a oferi funcționalitate optimă și pentru a îmbunătăți eficiența în diverse aspecte ale dezvoltării, principiul de bază fiind sinergia cu structura bazată pe microservicii.

Spring Boot oferă un set de dependențe 'starter' concepute pentru a simplifica configurațiile de construcție Gradle. Au fost folosite spring-boot-starter-amqp pentru a facilita conexiunea și schimbul de mesaje cu microserviciile RabbitMQ, care susțin nevoile de mesagerie asincronă ale arhitecturii. În plus, spring-boot-starter-data-jpa a fost utilizat pentru a interfața cu bazele de date MySQL. Acest starter oferă un strat de acces la date, reducând codul boilerplate și făcând operațiunile de date mai eficiente și gestionabile.

Spring-boot-starter-oauth2-resource-server a fost crucial în gestionarea protocolului OAuth2 pentru serverul de resurse securizate, permitând un control de acces sigur. Pentru funcționalitățile aplicațiilor bazate pe web, am folosit spring-boot-starter-web, care include biblioteci pentru a construi aplicații web, inclusiv RESTful, folosind Spring MVC.

MySQL-connector-j a fost folosit pentru a stabili o conexiune JDBC cu baza de date MySQL, asigurând persistența datelor. Ca parte a bazelor de date NoSQL, spring-boot-starter-data-neo4j și spring-boot-starter-data-mongodb au fost esențiale pentru interacțiunea cu Neo4j și MongoDB, simplificând operațiunile pe aceste baze de date.

Biblioteca Lombok a fost o parte semnificativă a proiectului, promovând un cod mai curat prin reducerea boilerplate-ului în clasele Java prin adnotări pentru modele comune, cum ar fi getteri și setteri. Minio, un server de stocare a obiectelor, a fost încorporat pentru stocarea fișierelor și altor obiecte, permitând soluții de stocare eficiente și scalabile.

Comunicarea bidirectională în timp real între client și server a fost posibilă prin intermediul spring-boot-starter-websocket, asistând la actualizări instant. În cele din urmă, reactor-core a fost încorporat pentru a oferi o paradigma de programare reactivă, non-blocantă pentru aplicațiile Spring Boot, susținând crearea de servicii eficiente și scalabile.

Pe partea de frontend, aplicațiile Angular au fost susținute de un număr de biblioteci. S-a utilizat ngx-cookieconsent pentru gestionarea stării de consimțământ a cookie-urilor pe site, asigurând conformitatea cu GDPR. Biblioteca keycloak-angular a fost esențială în securizarea aplicației și gestionarea autentificării și autorizării utilizatorilor folosind Keycloak.

Interfața aplicației a beneficiat foarte mult de Angular Material, o bibliotecă care oferă componente UI preconstruite conform specificației Google's Material Design, ajutând la crearea unui layout consecvent, responsiv și cu un aspect modern. Angular Router a fost vital pentru navigarea între diferitele componente, menținând natura aplicației de single page a Angular.

S-a folosit ngx-owl-carousel-o pentru un afișaj de carusel interactiv în aplicație a fișierelor postărilor, promovând o experiență de utilizator dinamică și atractivă. În sfârșit, biblioteca rxjs, care oferă extensii reactive pentru JavaScript, a fost esențială în gestionarea datelor asincrone, cum ar fi actualizările interfeței de utilizator, și manipularea mai multor valori de-a lungul timpului. Acest lucru a fost deosebit de util pentru caracteristicile în timp real, făcând aplicația mai receptivă și prietenoasă cu utilizatorul.

#### 4.5 Tehnologii

Dezvoltarea aplicației web a fost realizată în primul rând utilizând o gamă diversă de tehnologii, fiecare cu capacitatele lor unice care servesc cel mai bine cerințele componentelor specifice ale sistemului.

Nginx este un server HTTP și reverse server proxy de înaltă performanță și open-source. În contextul acestui proiect, a fost utilizat pentru a servi aplicațiile Angular. Vechimea framework-ului Angular combinată cu viteza și flexibilitatea Nginx a îmbunătățit performanța generală a aplicației și experiența utilizatorului.

Keycloak, o soluție open-source de Identitate și Management al Accesului larg utilizată, a fost desfășurată pentru a gestiona procesele de autentificare și autorizare ale aplicației. Aceasta a fost extins cu două plugin-uri: Keycloakify, un instrument care permite crearea de pagini de autentificare personalizate folosind React, și Keycloak-event-listener-RabbitMQ, care surprinde și transmite evenimentele Keycloak la RabbitMQ. Aceste plugin-uri au fost alese pentru a augmenta flexibilitatea Keycloak și pentru a-l integra mai bine în arhitectura aplicației.

Pentru administrarea bazei de date MySQL, au fost utilizate două instrumente bazate pe PHP, Adminer și phpMyAdmin. Aceste instrumente oferă o interfață web intuitivă pentru sarcinile de gestionare a bazei de date, simplificând astfel și eficientizând sarcinile de administrare a bazei de date.

Clusterul Kubernetes a fost configurat pe Minikube și a fost utilizat un controller de Ingress Nginx. Controllerul de Ingress este responsabil pentru îndeplinirea regulilor de Ingress care permit accesul extern la serviciile din cluster. Acest lucru a permis aplicației să fie accesibile din afara mediului Minikube.

RabbitMQ, un software broker de mesaje, a fost selectat pentru două roluri majore. În primul rând, a fost folosit pentru a asculta evenimentele expediate de Keycloak, asigurând astfel propagarea pe întregul sistem a evenimentelor de înregistrare a utilizatorilor. În al doilea rând, RabbitMQ a servit ca mijloc de comunicare între serviciile de backend ale clientului și ale adminului, decuplând astfel aceste servicii și sporind reziliența generală a sistemului.

Minio, un sistem de stocare a obiectelor de înaltă performanță și compatibil cu Kubernetes, a fost utilizat în scopuri de stocare a fișierelor, în special pentru stocarea postărilor și imaginilor de profil ale utilizatorilor. A fost ales datorită compatibilității sale cu serviciul de stocare în cloud Amazon S3, fiind posibilă migrarea de la stocare locală, precum și a capacitatei sale de a gestiona implementările Kubernetes la scară mare.

Mongo Express a fost utilizat pentru monitorizarea instanțelor MongoDB. Această interfață de administrare MongoDB bazată pe web a permis o interacțiune ușoară cu sistemul de baze de date și o monitorizare îmbunătățită, simplificând astfel sarcinile de administrare a bazei de date.

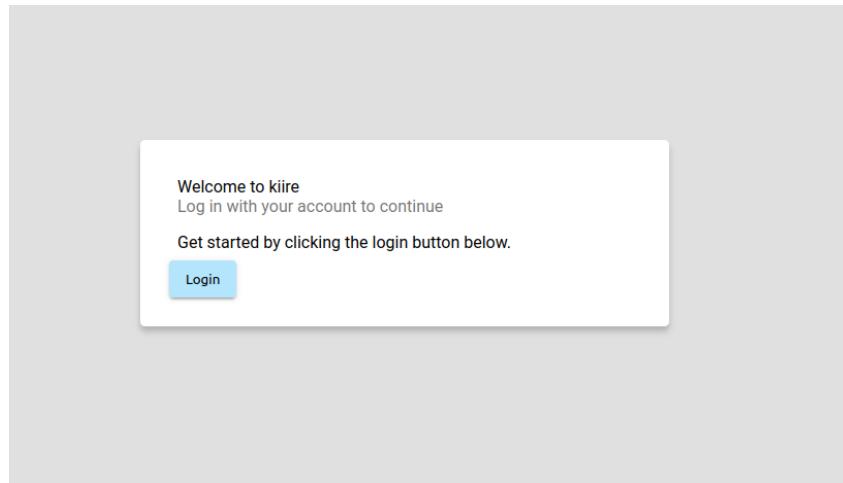
PGAdmin, un instrument popular de administrare și management open-source pentru baza de date PostgreSQL, a fost utilizat pentru gestionarea bazei de date folosită de Keycloak.

În final, proiectul a folosit Gradle ca instrument de automatizare a construirii. Gradle este un instrument de construire flexibil și puternic care este scriptabil în Groovy, și a fost ales pentru performanța lui crescută comparativ cu maven.

#### 4.6 Prezentarea aplicației

#### 4.6.1 Prezentare rol client

Când clientul deschide pagina web, va fi întâmpinat de pagina de login din Angular. În urma acțiunării butonului de login, utilizatorul va fi redirectat spre pagina de logare din Keycloak unde poate alege să se conecteze sau să creie un nou cont.

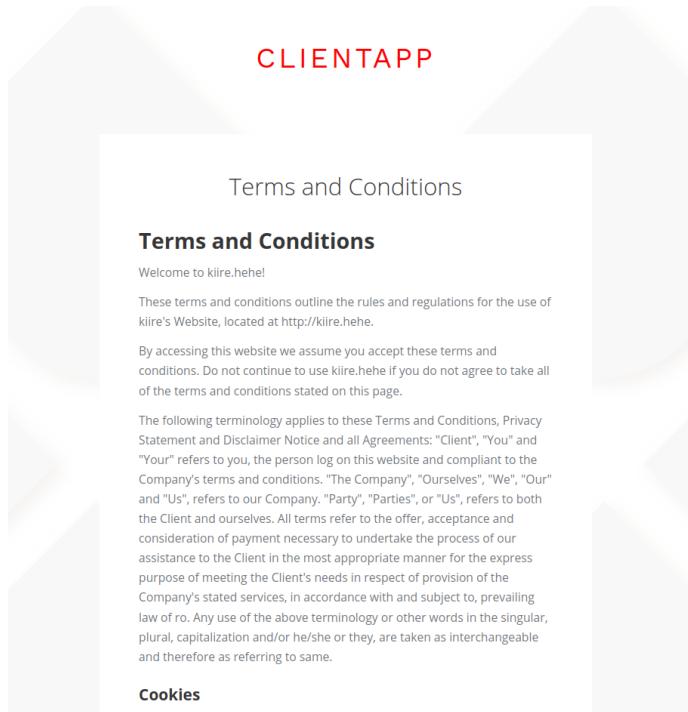


Figură 4.1 pagina de login client în Angular

A diagram showing two overlapping windows. The left window, titled "CLIENTAPP", contains a "Sign In" form with fields for "Username" and "Password", and a blue "Sign In" button. Below the button is the text "New user? [Register](#)". The right window, also titled "CLIENTAPP", contains a "Register" form with fields for "Username" (containing "adrianedel0"), "Password", and "Confirm password", and a blue "Register" button. Below the "Register" button is the text "[« Back to Login](#)". Both windows have a faint watermark-like background of a house shape.

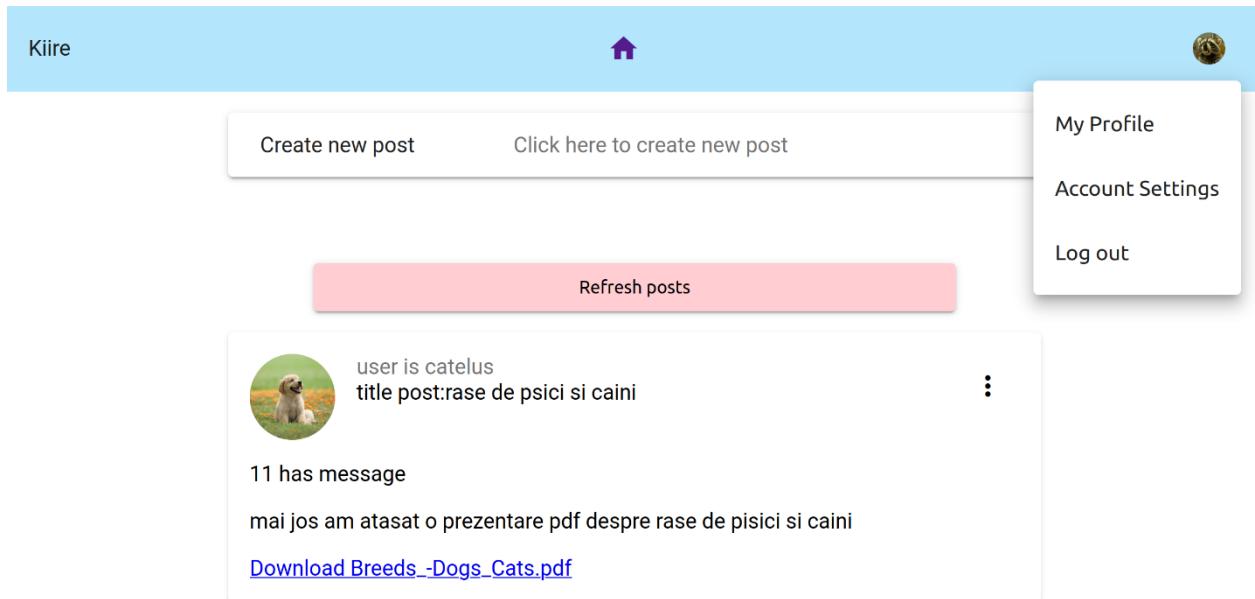
Figură 4.2 pagina de logare și pagina de înregistrare în Keycloak

În urma creării unui nou cont este necesar acceptarea termenilor și condițiilor. Deși contul a fost creat și există, la orice reconectare, utilizatorul va fi rugat să accepte termenii și condițiile altfel nu va putea accesa aplicația.



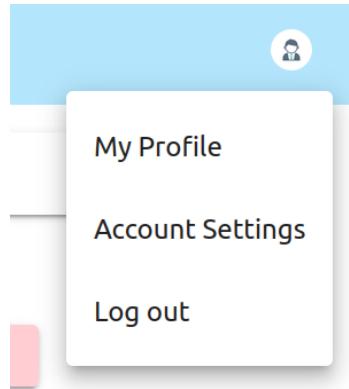
Figură 4.3 Pagina de termeni și condiții

Utilizatorul conectat cu succes va intra în pagina home unde va putea vedea postările de la toți utilizatorii, sortate în funcție de cele mai recente postări. De asemenea deasupra postărilor se află secțiunea de creare a unei noi postări, vizibilă doar în pagina home și pagina propriului profil.



Figură 4.4pagina home

Prin acționarea imaginii de profil din colțul dreapta sus, se va deschide un meniu din care utilizatorul se poate duce la pagina propriului profil, să se duca la setările de securitate a contului sau să se deconecteze.



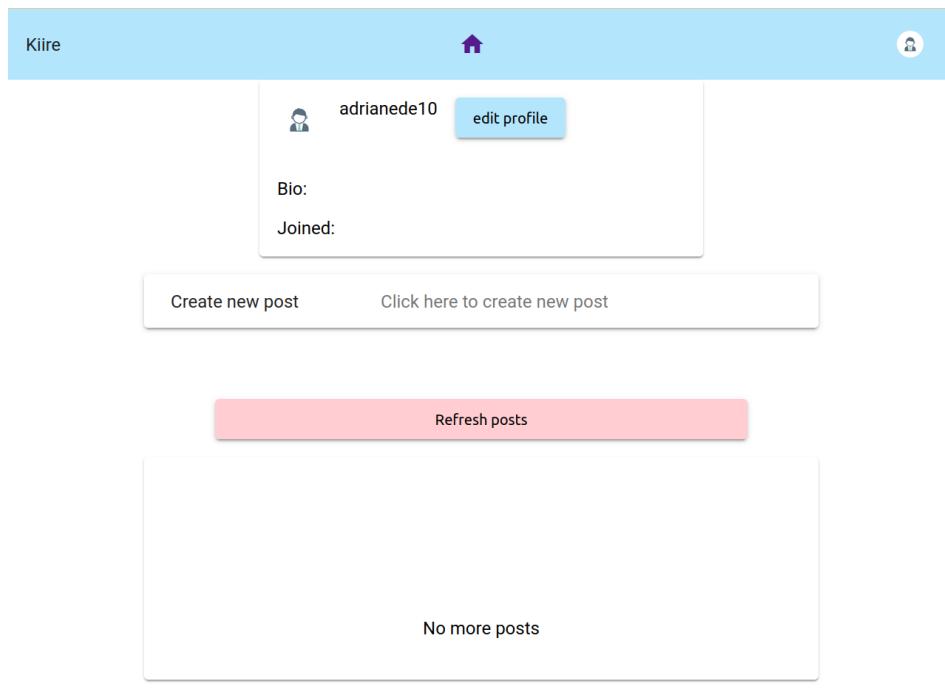
Figură 4.5 meniu pagina home

După apăsarea account settings, utilizatorul va fi redirecționat la pagina de schimbă parola contului din Keycloak.

A screenshot of the Keycloak "Change Password" page. The page has a black header with the Keycloak logo and navigation links "Back to clientfront" and "Sign out". The main content area has a light gray background. On the left, there is a sidebar with a blue header labeled "Password" and a right-pointing arrow. The main content area is titled "Change Password" and contains three input fields: "Password", "New Password", and "Confirmation", each with a corresponding text input box. Below these fields is a small note "All fields required". At the bottom right is a blue "Save" button.

Figură 4.6 pagina pentru schimbarea parolei din Keycloak

Dacă utilizatorul alege să meargă spre pagina lui de profil, acolo va putea vedea detaliile despre propriul cont, propriile postări și își poate edita datele personale.



Figură 4.7pagina de profil a utilizatorului

Edit Profile

Username\* adrianede10

First Name\* adrian

Last Name\* borsan

Bio sunt un vizitor

[Choose Profile Picture](#)

[Submit](#)

Kiire

adrianede10 [edit profile](#)

adrian borsan

Bio: sunt un vizitor

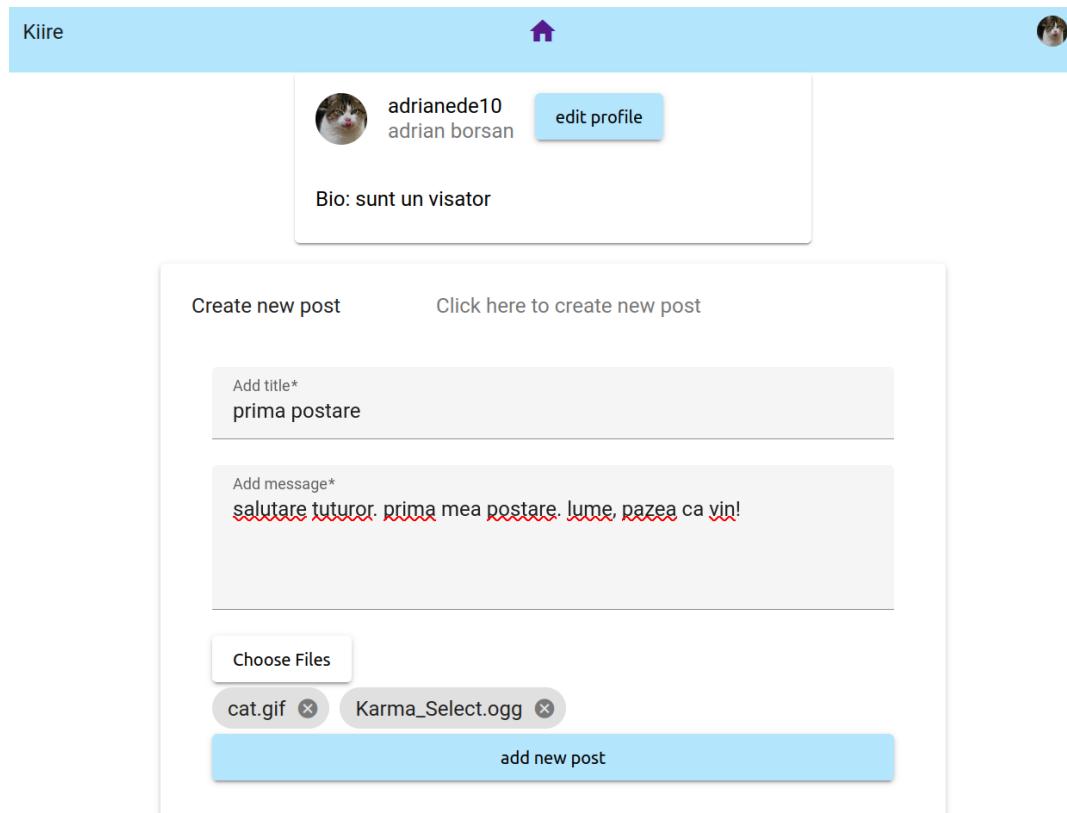
Joined:

Create new post [Click here to create new post](#)

Refresh posts

Figură 4.8 dialogul de editare profil și profilul cu editările făcute

Prin apăsarea secțiunii de creare a unei noi postări, se va deschide un formular, unde se poate completa cu titlul postării, mesajul postării și se pot adăuga un număr variabil de fișiere.



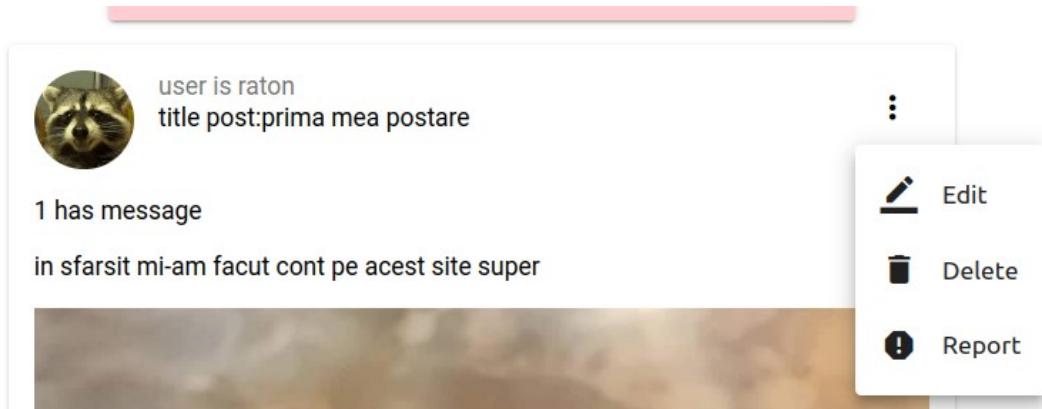
Figură 4.9 formular adăugare postare

După acționarea butonului de adăugare, lista de postări va fi reîmprospătată și postarea creată va apărea



Figură 4.10 noua postare creată

Pentru a edita o postare creată de utilizatorul curent, se poate actiona meniu contextual al postării care se dorește a se edită și se selectează opțiunea edit. În dialogul de editare a unei postări se pot modifica titlul, mesajul, să fie înălțurate unele fișierele existente și adăugate altele noi.

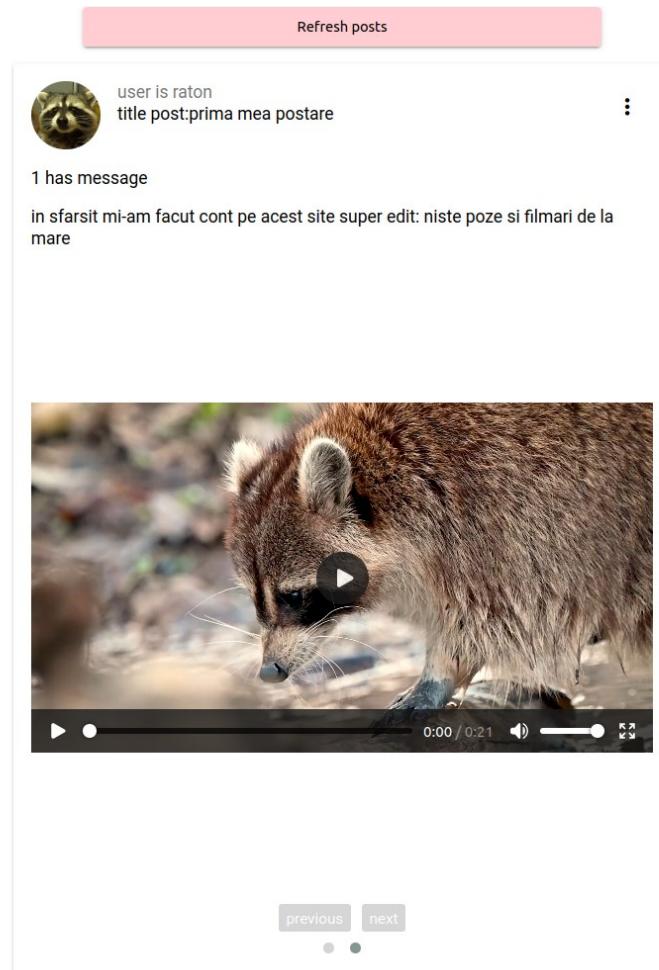


Figură 4.11 deschiderea meniului contextual al postării

A screenshot of a modal window titled "Update post with id 1". It contains fields for "Edit title" (with "prima mea postare") and "Edit message" (with "in sfarsit mi-am facut cont pe acest site super" and "edit: liste poze și filme de la mare"). Below these are sections for "existing files" (showing "happy.jpg") and "add new files" (with a "Choose Files" button and "raccoon video.mp4"). At the bottom are "Cancel Post edit" and "Edit post" buttons.

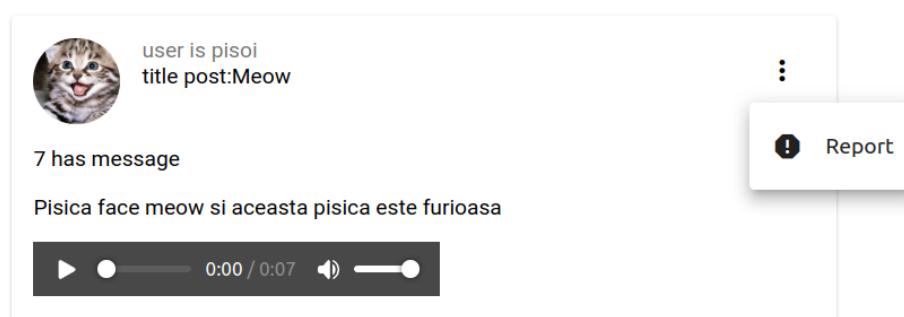
Figură 4.12 formularul de editare postare

În cazul în care editările au fost salvate, lista postărilor va fi reîmprospătată și postarea va apărea cu editările făcute.



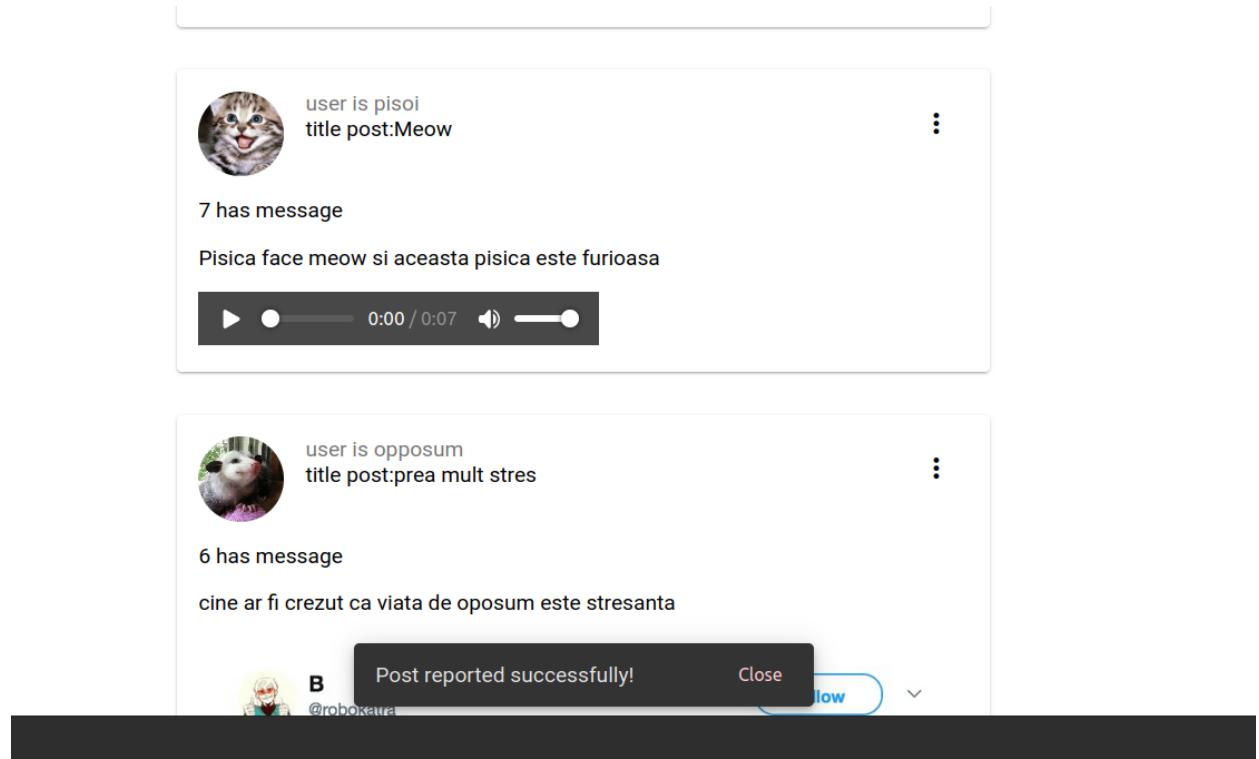
Figură 4.13 cum apare postarea după ce a fost editată

Pe pagina home, în urma acțiunării meniului contextual pentru postările care nu sunt deținute de utilizatorul curent, va apărea doar butonul de report.



Figură 4.14 meniul postărilor nedeținute de utilizatorul curent

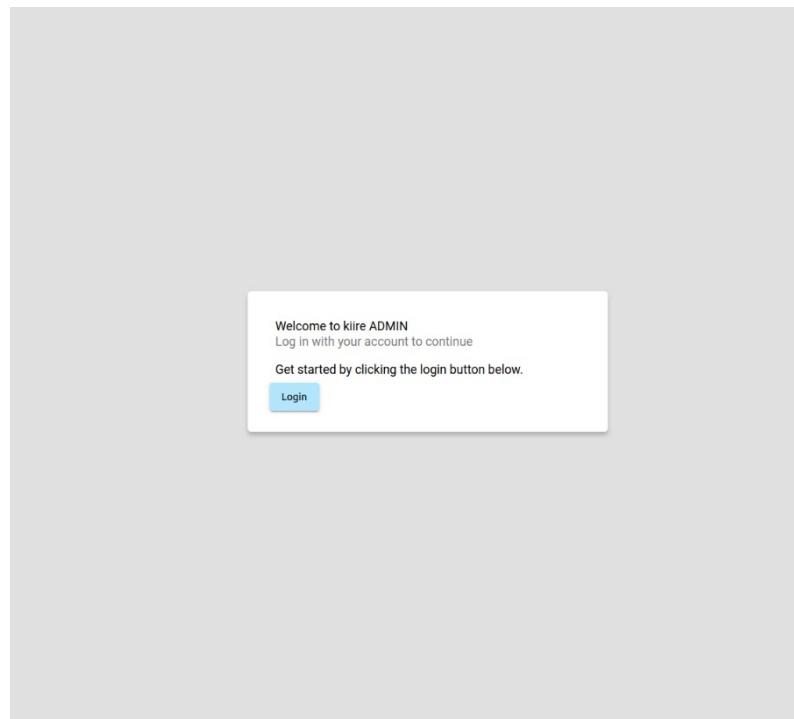
Dacă o postare a fost reportată cu succes, va apărea un mesaj de confirmare în josul paginii.



Figură 4.15 mesaj de confirmare raportare cu succes

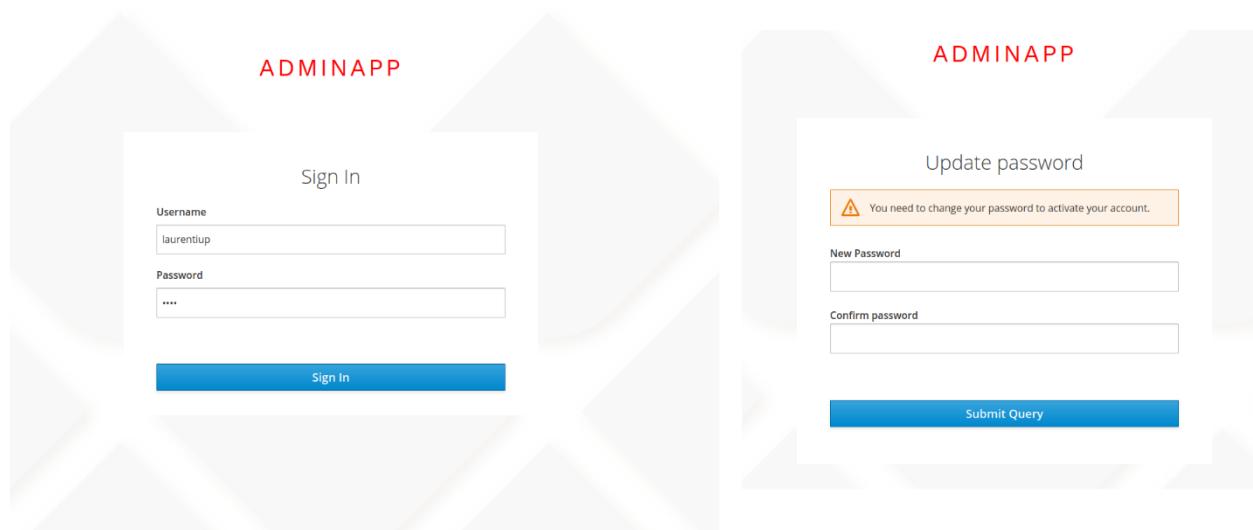
#### 4.6.2 Prezentare admin

Un admin este întâmpinat de o pagină de conectare similară cu cea a unui utilizator:



Figură 4.16 pagina de conectare admin

În comparație de aplicația de utilizator, un admin nu își poate crea singur un cont. Datele de conectare sunt oferite de către un administrator de sistem și la prima conectare este necesară alegerea unei parole noi.



Figură 4.17pagina de logare admin și prima conectare

În urma conectării cu succes, adminul va fi întâmpinat de pagina principală unde poate trata un report, alegând dacă încalcă sau nu anumite reguli. În cazul în care postarea încalcă regulile, aceasta va fi în mod automat ștersă.

rase de psici si caini

11 has message

mai jos am atasat o prezentare pdf despre rase de pisici si caini  
application/pdf

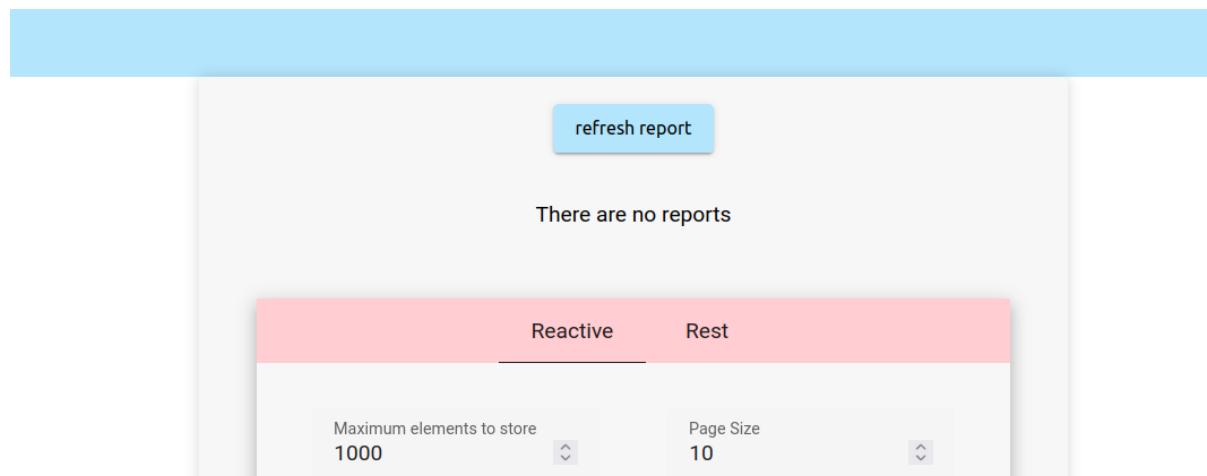
Breeds\_-Dogs\_Cats.pdf

[Download Breeds\\_-Dogs\\_Cats.pdf](#)

ao

Figură 4.18componenta de tratat report postare

Dacă toate reporturile au fost tratate, va apărea un mesaj ce indică acest lucru. Dacă într-o perioadă de timp apare un nou report, la actionarea butonului de refresh report va apărea ultimul report.



Figură 4.19 componenta de tratare report când nu mai există nici un report nefiltrat

De asemenea, admin are acces si la o componentă tabulară unde poate vizualiza informații cu privire la activitatea din aplicația clientului. Aceste informații pot fi vizualizate odată prin paginarea datelor deja existente, sau a vedea în mod reactiv noile intrări primite în timp real de la aplicația client.

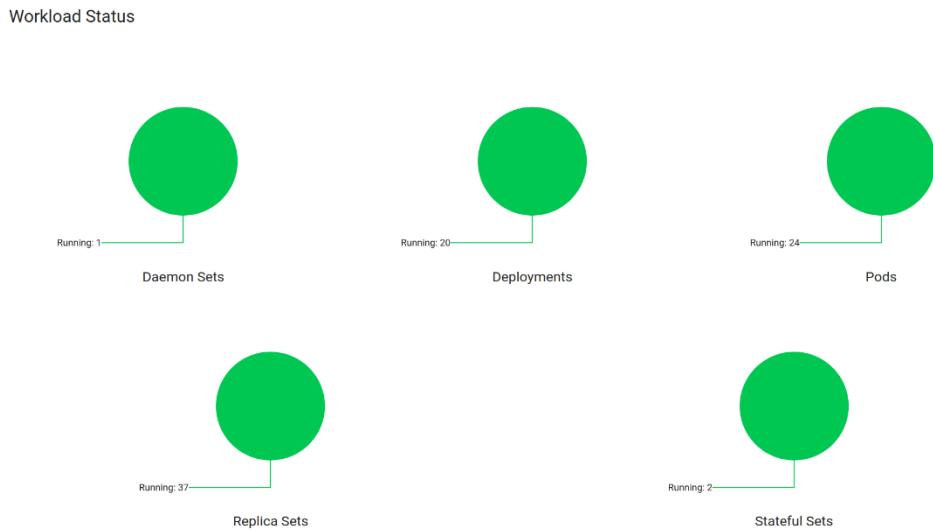
Reactive				Rest			
Id		createdAt	receivedAt ↓	message			
64a193b530a40	2023-07-02 1	2023-07-02 1 8:11 48:998	8:11 48:998	execution(public org.springframework.data.domain.Page com.adriandborsan.clientba ck.post.controllers.PostController.findAll(org.springframework.data.domain.Pageabl e))			
64a193a830a40	2023-07-02 1	2023-07-02 1 8:11 36:429	8:11 36:431	execution(public void com.adriandborsan.clientback.post.controllers.PostController.c reate(java.lang.Long))			
64a193a630a40	2023-07-02 1	2023-07-02 1 8:11 34:101	8:11 34:103	execution(public org.springframework.data.domain.Page com.adriandborsan.clientba ck.post.controllers.PostController.findAll(org.springframework.data.domain.Pageabl e))			
64a193a530a40	2023-07-02 1	2023-07-02 1 8:11 33:638	8:11 33:645	execution(public org.springframework.data.domain.Page com.adriandborsan.clientba ck.post.controllers.PostController.findAll(org.springframework.data.domain.Pageabl e))			
64a193a430a40	2023-07-02 1	2023-07-02 1 8:11 32:448	8:11 32:449	execution(public org.springframework.data.domain.Page com.adriandborsan.clientba ck.post.controllers.PostController.findAll(org.springframework.data.domain.Pageabl e))			
64a192d130a40	2023-07-02 1	2023-07-02 1 8:08 01:133	8:08 01:137	execution(public org.springframework.data.domain.Page com.adriandborsan.clientba ck.post.controllers.PostController.findAll(org.springframework.data.domain.Pageabl e))			
64a192d030a40	2023-07-02 1	2023-07-02 1 8:08 00:060	8:08 00:064	execution(public org.springframework.data.domain.Page com.adriandborsan.clientba ck.post.controllers.PostController.findAll(org.springframework.data.domain.Pageabl e))			
64a192ce30a40	2023-07-02 1	2023-07-02 1 8:08 58:033	8:07 58:042	execution(public org.springframework.data.domain.Page com.adriandborsan.clientba ck.post.controllers.PostController.findAll(org.springframework.data.domain.Pageabl e))			
64a192d030a40	2023-07-02 1	2023-07-02 1 8:07 57:004	8:07 57:005	execution(public org.springframework.data.domain.Page com.adriandborsan.clientba ck.post.controllers.PostController.findAll(org.springframework.data.domain.Pageabl e))			
64a192cb30a40	2023-07-02 1	2023-07-02 1 8:07 55:920	8:07 55:922	execution(public org.springframework.data.domain.Page com.adriandborsan.clientba ck.post.controllers.PostController.findAll(org.springframework.data.domain.Pageabl e))			

*Figură 4.20 informațiile paginate și informațiile prezentate în timp real*

#### **4.6.3 Prezentarea administrator de sistem**

Administratorul de sistem nu are o interfață creată încăcăt rolul lui este de a gestiona clusterul Kubernetes și de a asigura funcționarea corespunzătoare a sistemului. Acesta se foloseste de diverse instrumente.

Folosind minikube dashboard, se pot vizualiza toate microserviciile prezente în cluster, cât și informații referitoare la dimensiunea lor și puterea de procesare necesară să ruleze și statistici referitoare la sănătatea acestora. Serviciile care au probleme vor fi marcate cu roșu, iar cele sănătoase cu verde.



Figură 4.21 starea de sănătate a serviciilor

Pods									
Name	Images	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created	
mongo-express-deployment-68d5b6d798-kjcs	mongo-express	app: mongo-express pod-template-hash: 68d5b6d798	minikube	Running	31	<div style="width: 1.00m;"></div>	40.48Mi	12 days ago	⋮
mongo-deployment-554dc5459-j6h69	mongo:6	app: mongo pod-template-hash: 554dc5459	minikube	Running	13	<div style="width: 8.00m;"></div>	14.37Mi	12 days ago	⋮
phpmyadmin-deployment-7fc4f659cb-pkwrf	phpmyadmin	app: phpmyadmin pod-template-hash: 7fc4f659cb	minikube	Running	13	<div style="width: 1.00m;"></div>	21.42Mi	12 days ago	⋮
mysql-deployment-b7b467544-xbdk9	mysql 8.0.31	app: mysql pod-template-hash: b7b467544	minikube	Running	14	<div style="width: 8.00m;"></div>	66.34Mi	12 days ago	⋮
postgres-deployment-8577d54f5-qcpfa	postgres:15	app: postgres pod-template-hash: 8577d54f5	minikube	Running	13	<div style="width: 1.00m;"></div>	39.14Mi	12 days ago	⋮
adminer-deployment-765dd48899-5k8pd	adminer	app: adminer pod-template-hash: 765dd48899	minikube	Running	13	<div style="width: 1.00m;"></div>	16.00Mi	12 days ago	⋮
rabbitmq-deployment-9fdb9fb4-hvch6	rabbitmq:3.11-management-alpine	app: rabbitmq pod-template-hash: 9fdb9fb4	minikube	Running	13	<div style="width: 16.00m;"></div>	99.98Mi	12 days ago	⋮
keycloak-rabbitmq-deployment-98bbb7bb-78hrl	rabbitmq:3.11-management-alpine	app: keycloak-rabbitmq pod-template-hash: 98bbb7bb	minikube	Running	14	<div style="width: 18.00m;"></div>	83.13Mi	12 days ago	⋮
<a href="#">Show all</a>									
neo4j-0	neo4j:5.9.0	app: neo4j-deployment controller-revision-hash: neo4j-6d6465f 444 helm.neo4j.com/clustering: false	minikube	Running	11	<div style="width: 8.00m;"></div>	337.83Mi	11 days ago	⋮
cypher-shell	neo4j:5.9.0	run: cypher-shell	minikube	Running	32	<div style="width: 3.00m;"></div>	120.51Mi	11 days ago	⋮

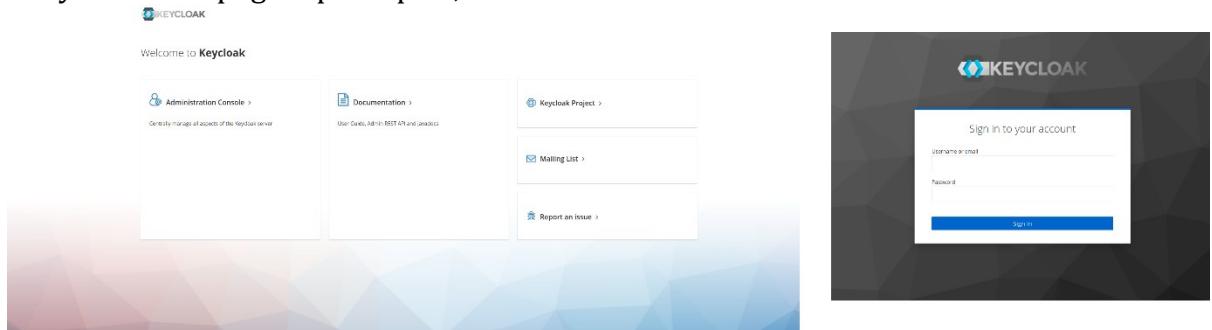
Figură 4.22 secțiunea de informații cu privire la memoria consumată și puterea de procesare

În secțiunea serviciilor, administratorul de sistem va putea accesa fiecare serviciu în parte.

Services							
Name ↑	Labels	Type	Cluster IP	Internal Endpoints	External Endpoints	Created	⋮
adminback-service	skaffold.dev/run-id: d9d8307f-1b9d-4478-a418-3dc6aa39dtee	LoadBalancer	10.105.134.48	adminback-service:8080 TCP adminback-service:32050 TCP	10.105.134.48:8080 ⚡	13 days ago	⋮
adminer-service	-	LoadBalancer	10.108.254.92	adminer-service:8080 TCP adminer-service:32128 TCP	10.108.254.92:8080 ⚡	13 days ago	⋮
adminfront-service	skaffold.dev/run-id: 14941cef-a365-46ef-973a-d537f27930b6	LoadBalancer	10.98.104.8	adminfront-service:80 TCP adminfront-service:32610 TCP	10.98.104.8:80 ⚡	13 days ago	⋮
authlistener-service	skaffold.dev/run-id: 2297f819-1e9e-45d2-9e48-5364538236f1	LoadBalancer	10.99.82.124	authlistener-service:8080 TCP authlistener-service:31759 TCP	10.99.82.124:8080 ⚡	13 days ago	⋮
clientback-service	skaffold.dev/run-id: a4090714-fd1e-4102-8ee1-75daea0f11256	LoadBalancer	10.105.93.62	clientback-service:80 TCP clientback-service:31994 TCP	10.105.93.62:8080 ⚡	13 days ago	⋮
clientfront-service	skaffold.dev/run-id: 89df73c-2662-4e63-bd0a-4ed4729e520	LoadBalancer	10.100.238.171	clientfront-service:80 TCP clientfront-service:30601 TCP	10.100.238.171:80 ⚡	9 days ago	⋮
grafana	app.kubernetes.io/instance: grafana app.kubernetes.io/managed-by: Helm app.kubernetes.io/name: grafana	LoadBalancer	10.105.121.170	grafana:80 TCP grafana:32605 TCP	10.105.121.170:80 ⚡	7 days ago	⋮
jaeger-agent	app: jaeger app.kubernetes.io/component: agent app.kubernetes.io/name: jaeger	ClusterIP	None	jaeger-agent:5775 UDP jaeger-agent:0 UDP jaeger-agent:4321 UDP jaeger-agent:0 UDP jaeger-agent:6322 UDP jaeger-agent:0 UDP jaeger-agent:5778 TCP jaeger-agent:0 TCP	-	8 days ago	⋮

Figură 4.23 secțiunea services

Administratorul de sistem se ocupă și de gestionarea conturilor de admin, folosind consola keycloak. Din pagină principală, se va conecta cu un master account.

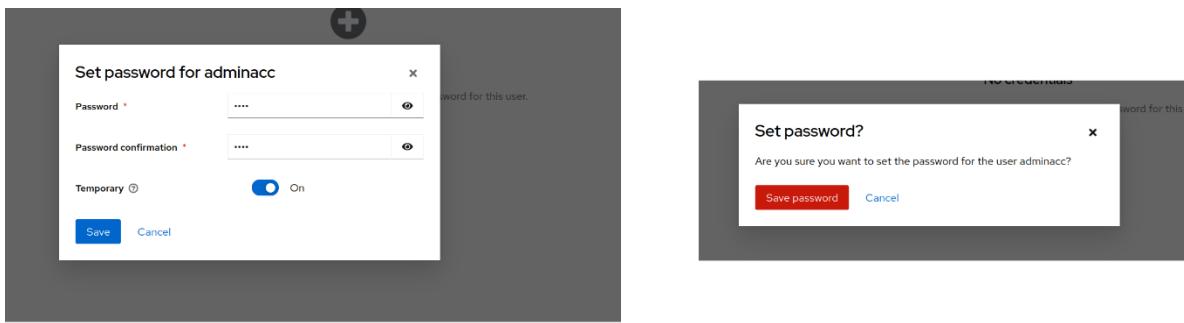


Figură 4.24 pagina principală și pagina de logare Keycloak

Pentru a crea un cont nou pentru admin, se va alege tărâmul adminapp, după care se va duce la secțiunea users și se selectează create new user. Dupa alegerea unui username, se va duce la secțiunea credentials și se alege set password unde se setează o parolă temporară bifând această opțiune astfel încât la prima conectare, adminul își va alege o nouă parolă.

The image shows the 'Create user' form in the Keycloak administration console. On the left, there's a sidebar with a dropdown menu showing 'master' selected, and a list of realms including 'adminapp', 'clientapp', and 'master'. The main area shows 'master realm' with tabs for 'Server info' and 'Provider info'. Under 'Server info', it shows version 21.1.1, product 'Default', and memory usage (Total memory: 512 MB, Free memory: 438 MB, Used memory: 73 MB). Under 'Profile', it lists 'Enabled features' (ACCOUNT3, ADMIN\_FINE\_GRAINED\_ACCESS, DOCKER, DYNAMIC\_SCOPES, RECOVERY\_CODES, SCRIPTS) and 'Disabled features' (none listed). On the right, the 'Create user' form is filled out with 'Username' set to 'adminacc', 'Email' field empty, 'Email verified' set to 'No', 'First name' field empty, 'Last name' field empty, and 'Groups' set to 'master'. There are 'Join Groups' and 'Create' buttons at the bottom.

Figură 4.25 meniul de alegere a tărâmului și meniul de creare a unui nou admin



Figură 4.26 setarea și confirmarea parolei a unui cont temporar

Un administrator de sistem are acces și la o instanță mongo-express unde poate vizualiza informații despre datele obținute din monitorizarea aplicației client.

Collections (incl. system.namespaces)	1
Data Size	265 KB
Storage Size	94.2 KB
Avg Obj Size #	353 Bytes
Objects #	751
Indexes #	1
Index Size	53.2 KB

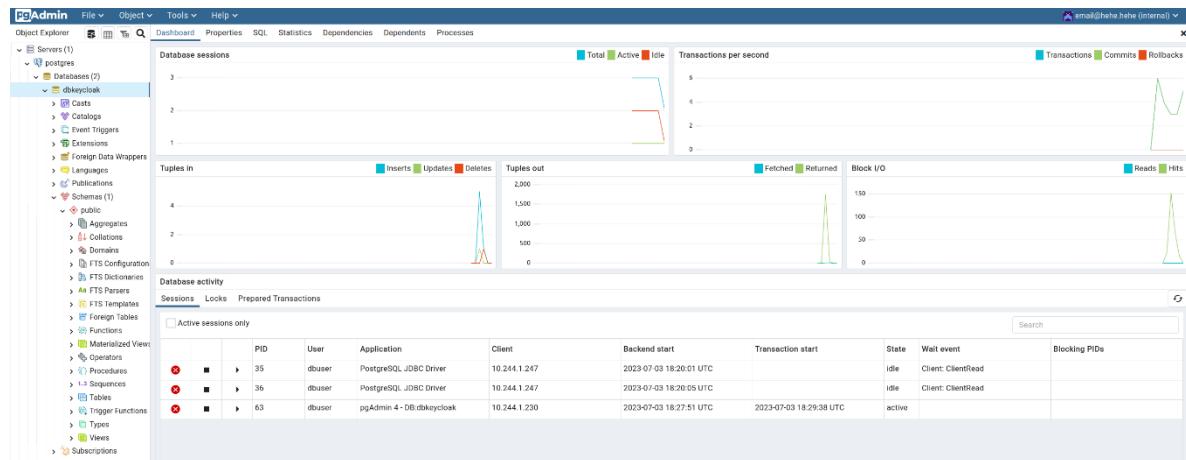
Figură 4.27 informații cu privire la datele log entries

Neo4j oferă propria consolă în care se pot vizualiza nodurile și relațiile între acestea cu posibilitatea de a executa și interogări.

Figură 4.28 interogare de selecție a maximum 25 de noduri

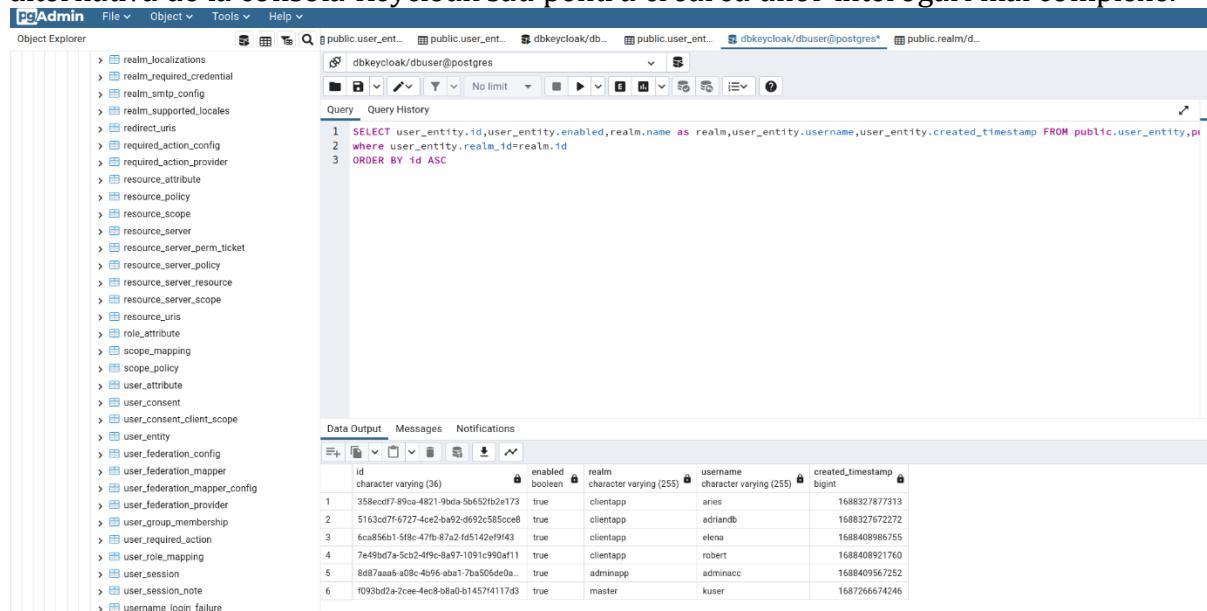
PgAdmin permite administratorului să observe informații referitoare la baza de date postgres folosită de keycloak.

## Capitolul 4 Implementarea Sistemului și tehnologii



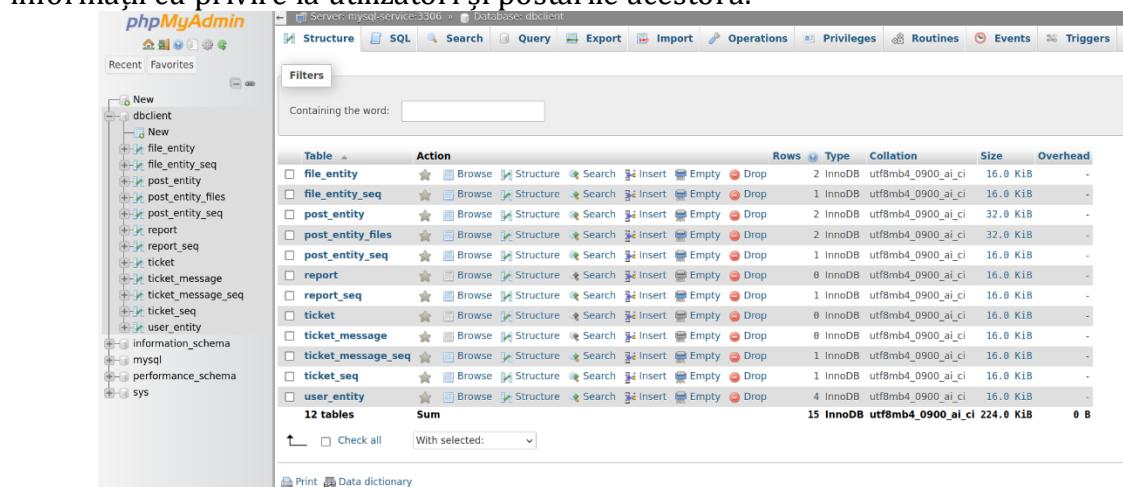
Figură 4.29 monitorizarea performanței lui postgres folosind pgadmin

De asemenea, PgAdmin permite interogarea datelor din Keycloak precum o alternativă de la consola Keycloak sau pentru crearea unor interogări mai complexe.



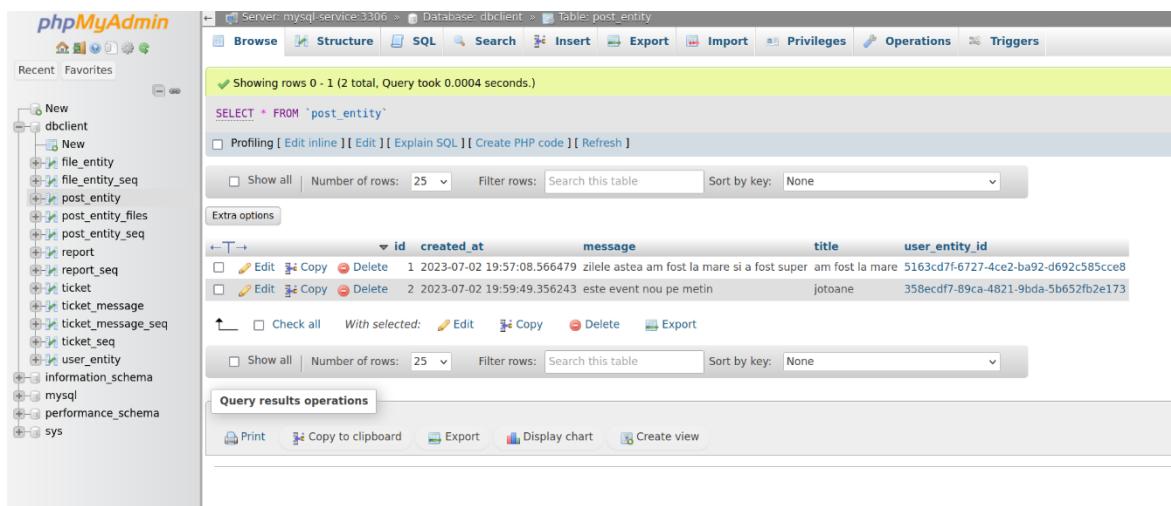
Figură 4.30 exemplu interogare pentru a vizualiza toți utilizatorii și numele sărâmului din care face parte contul

Cu ajutorul unelei phpmyadmin, un administrator de sistem poate vizualiza informații cu privire la utilizatori și postările acestora.



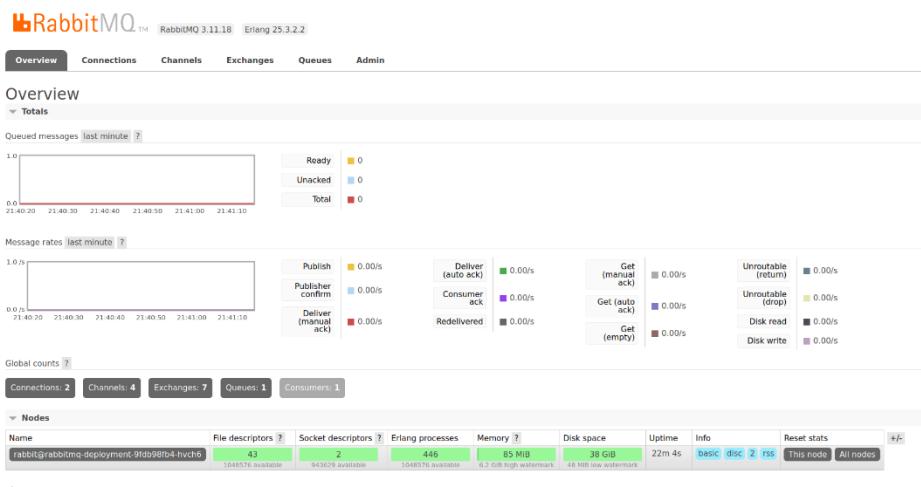
Figură 4.31 vizualizarea tuturor tabelelor din MySQL

## Capitolul 4 Implementarea Sistemului și tehnologii



Figură 4.32 vizualizarea informațiilor a unor rânduri din tabela postărilor

Pentru ambele instanțe de RabbitMQ se poate accesa consola pentru a monitoriza traficul prin ele și pentru a vedea detalii adiționale precum ce cozi sunt deschise și funcționale în acel moment și dacă există o problemă la primirea sau trimiterea datelor.



Figură 4.33 consola RabbitMQ

## CAPITOLUL 5      PLANURI DE VIITOR

Planurile viitoare pentru acest proiect presupun o serie de îmbunătățiri semnificative menite să îmbunătățească experiența utilizatorului, performanța operațională și scalabilitatea sistemului, adăugând în același timp noi funcționalități. Aceste obiective ar fi realizate prin combinarea restructurării anumitor părți ale sistemului actual și integrarea noilor tehnologii.

Pentru a asigura separația intereselor în sistemul existent, există planuri de refactoring atât pentru aplicațiile Angular cât și pentru cele Spring Boot. Aplicația Angular ar fi împărțită în submodule pentru a permite o mai bună organizare a codului și a facilita dezvoltarea colaborativă. Serviciile Spring Boot, pe de altă parte, ar fi împărțite în microservicii mai mici și mai concentrate, fiecare cu o singură responsabilitate.

Internaționalizarea și suportul multi-lingvistic ar fi integrate în ambele aplicații Angular și Spring Boot, promovând accesibilitatea și utilizabilitatea pentru un public global.

Pentru a asigura o experiență de utilizator captivantă și personalizată, există intenția de a implementa un sistem de clasificare a postărilor folosind TensorFlow Recommenders, Progressive Neural Architecture Search și AutoML. Acest sistem ar fi alimentat de un model neural instruit pe date legate de relevanța postării, angajamentul prezis, popularitate și nouitate. Pentru a asigura performanța unei astfel de operațiuni intensive, ar fi utilizată o bază de date cache, în special Redis astfel încât folosirea rețelei ar fi făcută doar periodic, iar rezultatele ținute în memorie.

Pentru a gestiona starea în aplicația Angular, ar fi introdus managerul de stare NgRx, care oferă un container de stare previzibil ce încorporează principiile Redux și RxJS.

Cu scopul de a îmbogăți funcționalitatea platformei de socializare, este planificată capacitatea de a adăuga jocuri, care exploatează capacitatele Unity de a construi jocuri HTML. Acest lucru ar implica crearea unui sistem în care fiecare joc ar putea fi găzduit ca un microserviciu, fără a modifica aplicația Angular de bază.

O altă îmbunătățire ar fi implementarea unui sistem de mesagerie prin chat folosind Apache Cassandra pentru stocare și WebSockets pentru comunicare în timp real. În plus, ar fi adăugate capabilități de chat vocal și video, utilizând aplicația Spring Boot ca server de semnalizare și WebRTC pentru comunicare peer-to-peer.

S-ar implementa un sistem de raportare mai elaborat, trecând de la un model de raportare simplu la un sistem bazat pe tickets mai complet, cu capacitatea de a gestiona diverse întrebări ale utilizatorilor în afară de rapoarte. Sistemul a fost deja implementat în server, necesitând implementarea lui și în Angular.

Securitatea este primordială și va fi întărită prin actualizarea tuturor serviciilor pentru a utiliza HTTPS. În plus, ar fi implementată etichetarea automată a postărilor folosind modelul Inception de la Google, îmbunătățind organizarea conținutului și căutarea.

Pentru a asigura robustețea aplicației, ar fi introduse testarea unitară și cea de integrare în aplicațiile Spring Boot și Angular. Toleranța la erori în Kubernetes ar fi adăugată pentru a îmbunătăți reziliența sistemului.

Introducerea unui sistem de notificare ar menține utilizatorii la curent cu activitățile relevante pe platformă. În cele din urmă, pentru a extinde acoperirea platformei către utilizatorii de dispozitive mobile, aplicația Angular ar fi portată pe platforme mobile folosind tehnologii precum Tauri, largind efectiv baza de utilizatori.

Pentru a îmbunătăți funcționalitatea platformei, ar fi implementat un sistem de prietenie. Acest lucru le-ar permite utilizatorilor să se conecteze unul cu celălalt și să își construiască rețelele sociale pe platformă, făcând experiența mai captivantă și interactivă.

În plus, pentru a oferi mai multă comoditate și ușurință utilizatorilor, ar fi adăugată funcționalitatea de Single Sign-On (SSO), permitând utilizatorilor să se autentifice folosind conturile lor Google sau alte acreditări de la terți. Acest lucru ar simplifica procesul de autentificare și ar facilita aderarea noilor utilizatori la platformă.

Pentru a îmbunătăți capacitatele de descoperire a conținutului platformei, ar fi implementată o funcționalitate de căutare folosind ElasticSearch. Acesta ar oferi o căutare text robustă, scalabilă și posibilitatea de a gestiona o varietate largă de tipuri de date.

În plus, există planuri de implementare a Google AdSense, permitând astfel platformei să servească reclame întâmpinate utilizatorilor săi și să genereze potențial venituri.

Experiența utilizatorului ar fi îmbunătățită și prin analizarea linkurilor speciale din postări, cum ar fi videoclipurile de pe YouTube, pentru a le permite să fie redate direct în aplicație.

Aceste planuri ambițioase urmăresc să ridice aplicația web la statutul de platformă de socializare robustă, plină de funcții și versatilă, oferind o experiență de utilizator captivantă și dinamică, asigurând în același timp scalabilitatea și performanța sistemului.

## CAPITOLUL 6 CONCLUZII

Obiectivul principal al acestei teze, dezvoltarea unei aplicații web bazate pe microservicii folosind cadrul Spring Boot, a fost realizat. Prin crearea unui cluster Kubernetes Minikube care cuprinde multiple microservicii, inclusiv trei aplicații Spring Boot, două aplicații Angular, servicii RabbitMQ, Keycloak și diverse baze de date, a fost stabilită o aplicație de social media funcțională. Această aplicație permite utilizatorilor să interacționeze cu funcționalitățile de social media, cum ar fi crearea, citirea, actualizarea și ștergerea postărilor. În plus, a fost dezvoltată o interfață de admin pentru monitorizarea și gestionarea conținutului raportat, stabilind un mediu sigur și prietenos pentru utilizatori.

Originalitatea lucrării constă în utilizarea arhitecturii bazate pe microservicii pentru o aplicație web, abordare care nu este comună în aplicațiile de social media. Acest proiect prezintă o soluție inovatoare prin combinarea Spring Boot, Angular, RabbitMQ, Keycloak și diverse baze de date pentru a crea o aplicație cu o scalabilitate și reziliență robustă. În plus, integrarea diferitelor baze de date, inclusiv MySQL, Postgres, Neo4J, MongoDB și Minio, nu numai că a îmbogățit noutatea proiectului, dar a oferit și o bază solidă pentru gestionarea și stocarea datelor.

Aplicația dezvoltată și arhitectura sa pot fi potențial aplicate într-o gamă largă de platforme de rețele sociale și alte industrii care necesită medii colaborative în timp real. În plus, versatilitatea microserviciilor poate facilita dezvoltarea de aplicații în diverse domenii, cum ar fi comerțul electronic, sănătatea, finanțele și altele.

Cu toate acestea, drumul către această realizare nu a fost fără provocări. Complexitatea gestionării multiplelor microservicii și integrarea diverselor tehnologii a reprezentat un obstacol semnificativ. În plus, asigurarea unei comunicări eficiente între microservicii și furnizarea de măsuri de securitate eficiente utilizând Keycloak a necesitat un design atent. Cu toate acestea, aceste provocări au servit drept experiențe de învățare importante, oferind perspective valoroase asupra arhitecturii microserviciilor și a implementării sale practice.

Având în vedere experiența dobândită pe parcursul acestui proiect, printre planurile de viitor se află și o serie de îmbunătățiri și extinderi posibile ale sistemului. În primul rând, securitatea ar putea fi îmbunătățită prin implementarea protocolelor HTTPS și SSL. Acestea ar asigura o comunicare criptată între client și server, sporind astfel securitatea datelor utilizatorilor. Un alt aspect care ar putea aduce un plus de valoare aplicației ar fi introducerea unui sistem de prietenie între utilizatori. Aceasta ar putea îmbunătăți experiența utilizatorilor și ar putea stimula interacțiunea și angajamentul în cadrul platformei.

Succesul acestui proiect subliniază potențialul imens al microserviciilor în dezvoltarea de aplicații web scalabile și robuste și evidențiază semnificația adoptării de abordări noi și a tehnologiilor diverse în răspuns la nevoile în continuă evoluție ale utilizatorilor digitali. Acest proiect stabilește un precedent robust pentru viitoarele eforturi în domeniul microserviciilor, servind drept fundament pentru aplicații mai avansate și centrate pe utilizator.

## BIBLIOGRAFIE

- [1] <https://angular.io/> 03-07-2023
- [2] <https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html> 06-07-2023
- [3] <https://en.wiktionary.org/wiki/kiire> 03-07-2023
- [4] <https://engineering.zalando.com/posts/2017/02/using-microservices-to-power-fashion-search-and-discovery.html> 06-07-2023
- [5] <https://github.com/> 05-07-2023
- [6] <https://github.com/Netflix/zuul> 06-07-2023
- [7] <https://git-scm.com/> 05-07-2023
- [8] <https://instagram-engineering.com/what-powers-instagram-hundreds-of-instances-dozens-of-technologies-adf2e22da2ad> 05-07-2023
- [9] <https://kubernetes.io/> 03-07-2023
- [10] <https://linuxmint.com/> 05-07-2023
- [11] <https://material.angular.io/> 03-07-2023
- [12] <https://min.io/> 03-07-2023
- [13] <https://minikube.sigs.k8s.io/docs/> 03-07-2023
- [14] <https://netflixtechblog.com/zuul-2-the-netflix-journey-to-asynchronous-non-blocking-systems-45947377fb5c> 06-07-2023
- [15] <https://oauth.net/2/> 03-07-2023
- [16] <https://spring.io/> 03-07-2023
- [17] <https://spring.io/projects/spring-cloud-alibaba> 06-07-2023
- [18] <https://spring.io/projects/spring-cloud-netflix> 06-07-2023
- [19] [https://www.alibabacloud.com/blog/cloud-native-and-application-management-with-ease-part-6-microservices\\_597384](https://www.alibabacloud.com/blog/cloud-native-and-application-management-with-ease-part-6-microservices_597384) 06-07-2023
- [20] <https://www.infoq.com/news/2016/02/Monolith-Microservices-Zalando/> 06-07-2023
- [21] <https://www.java.com/en/> 03-07-2023
- [22] <https://www.techheadcorp.com/blog/design-of-microservices-architecture-at-netflix/> 06-07-2023
- [23] <https://www.tensorflow.org/recommenders> 04-07-2023
- [24] <https://www.typescriptlang.org/> 03-07-2023