



Universitatea
Transilvania
din Brașov
FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ

Programul de studii :
Informatică Aplicată

Lucrare de licență

Platformă de Social Media

Autor: **Ştefan Adriana**

Coordonator
științific: **Lector Dr. Răzvan Bocu**

Brașov, 2021

Contents

1. Introducere.....	4
1.1. Contextul proiectului.....	4
1.2. Motivația.....	5
2. Obiectivele Proiectului.....	6
2.1. Obiectivul principal.....	6
2.2. Obiective secundare.....	6
3. Tehnologii folosite	8
3.1. Java 11	8
3.1.1. Utilitate	8
3.1.2. Spring Boot.....	9
3.1.3. Apache Maven.....	9
3.1.4. Hibernate.....	10
3.1.5. MySQL.....	11
3.1.6. JWT.....	11
3.1.7. Angular.....	11
4. Arhitectura sistemului	12
5. Arhitectura componentei backend	13
5.1. Detalierea componentelor.....	14
5.1.1. Controller.....	15
5.1.2. Service	16
5.1.3. Repository	18
5.1.4. Model.....	20
5.1.5. Configuration.....	22
5.1.6. Security	24
5.1.7. DTO	25
5.1.8. Util.....	26
6. Arhitectura componentei frontend.....	27

6.1.	Pachetul component.....	28
6.1.1.	Componenta Header.....	31
6.1.2.	Componenta Profile-Page	34
6.1.3.	Componenta Login	39
6.1.4.	Componenta Home-Pages.....	42
6.1.5.	Componenta Post-Pages	49
6.1.6.	Componenta Chat-Pages	50
6.2.	Pachetul Model	52
1.1.	Pachetul Service.....	53
1.2.	Pachetul Pipe.....	55
7.	Structura bazei de date.....	55
8.	Concluzii.....	60
8.1.	Obiective propuse	60
8.2.	Idei de dezvoltare ulterioare.....	60
	Bibliografie	62

1. Introducere

1.1. Contextul proiectului

Platformele de social media permit oamenilor să se conecteze cu cei dragi și facilitează comunicarea între aceștia. Totodată ajută la creșterea comunicării și stabilirii conexiunilor cu oamenii din întreaga lume, permit creezeare și schimbul de informații și idei și formează comunități virtuale de oameni.

Social media este interacțiunea comună între oameni în care creează, împărtășesc sau schimbă informații și idei în comunitățile virtuale. A devenit nevoie de bază și calitatea ființelor umane de a fi social. Evoluțiile spectaculoase în comunicații și tipurile de divertisment inovatoare și uimitoare au dat acces la informații și capacitatea de a da voce oamenilor care nu ar fi fost auziți niciodată. Actuala generație este destul de norocoasă să asiste la unele dintre cele mai uimitoare evoluții tehnologice din istorie.

Oamenii doresc întotdeauna să se conecteze cu societatea într-un fel său altul. În zilele anterioare, modurile de comunicare erau limitate. Oamenii au socializat cu ceilalți în propriile lor moduri. Mai devreme, socializarea a fost restrânsă la vizitarea reciprocă a locului, având întâlniri mari, întâlniri în cluburi, parcuri și alte locuri publice. Acum timpul s-a schimbat. Oamenii și-au minimizat viața socială din cauza vieții agitate și a creșterii distanței geografice și a preocupărilor economice.

Odată cu venirea tehnologiei, site-urile și aplicațiile de rețele sociale au anunțat o revoluție în lume. A adus într-adevăr oameni din întreaga lume mai aproape prin crearea, schimbul sau schimbul de informații și idei în comunități și rețele virtuale. Aceste site-uri de rețele sociale se bazează pe tehnologii web și creează platforme extrem de interactive. A căpătat impuls la nivel global datorită caracteristicilor săle mai bune, accesului, frecvenței, imediatității, utilizabilității și permanenței. A fost recunoscută atât de larg și utilizarea a crescut atât de incredibil astăzi încât s-a mutat de la computerele desktop la laptopuri la telefoanele mobile. Platforma de social media este, fără îndoială, ușor de obținut și accesibilă. Astăzi, fiecare persoană este dependentă de rețelele sociale și asta cu o viteză fulgerătoare.

Avantajele unei astfel de platforme enumera:

- Vizează un public larg, făcându-l un instrument util și eficient pentru societate.

- Aceasta ajunge la oameni chiar și în zonele îndepărtate, iar informațiile sunt răspândite rapid.
- Distanța nu mai este o limitare din cauza rețelelor sociale. Poți fi în permanență actualizat cu cele mai recente știri și evenimente din societate și mediu prin intermediul site-urilor de socializare.
- Site-uri și bloguri precum Facebook, Twitter, Instagram, LinkedIn și multe altele au devenit un instrument pentru conectarea oamenilor de pe tot globul. Oamenii pot participa la discuții live sau sesiuni live sau prelegeri care se desfășoară oriunde în lume în timp ce stau acasă.
- Profesorii și profesorii pot predă pe diferite teme din locuri îndepărtate.
- Rețelele sociale permit companiilor să utilizeze aceste site-uri ca rețea pentru a genera conștientizare despre produsul lor, pentru a-și promova marca și, prin urmare, pentru a-și crește vânzările. Economisește costurile de marketing și publicitate.
- Aceste site-uri de rețea de pe rețelele de socializare oferă o platformă cuprinzătoare pentru tinerii care aspiră la artiști pentru a-și prezenta pasiunea și abilitățile.
- Liderii politici folosesc platforma rețelelor sociale pentru răspândirea comunicării sociale în masă. În aceste zile, candidații politici comunică și cu alegătorii prin intermediul rețelelor sociale.

Dar aceste platforme au de asemenea și dezavantaje. Aceste dezavantaje sunt reprezentate de:

- Utilizatorii rețelelor sociale devin victime ale unor escrocherii frauduloase și online care par a fi autentice.
- Deschide posibilitatea hackerilor de a comite fraude și de a lansa atacuri virale
- Productivitatea oamenilor este îngreunată din cauza utilizării și îngăduinței extreme pe aceste site-uri de socializare.
- Și studenții sunt extrem de activi pe site-urile de socializare în aceste zile, ceea ce îi limitează de la activități în aer liber.
- Unele site-uri sunt utilizate pentru a exprima furia sau disputa personală din cauza căreia se creează mult haos și confuzie.

1.2. Motivația

Motivatia care a stat în spatele acestei alegeri de tema pentru proiectul de licenta a fost dorinta de a aprounda notiunile tehnice invatate pe parcursul celor trei ani de studii ce vizeaza partea de dezvoltare web. Tematica aplicației a starnit interes deoarece notiunile

abordate au reprezentat o provocare pe parcusul facultatii, notiuni precum lucru cu o baza de date complexa, folosirea securitatii pentru asigurarea integritatii aplicatiei și lucru cu o arhitectura cat mai bine definita a proiectului.

2. Obiectivele Proiectului

2.1. Obiectivul principal

Scopul principal al proiectului este reprezentat de dezvoltarea unei platforme social media cu ajutorul căreia se va facilita comunicarea și împărașirea de informații între utilizatori. Această platformă va fi folosită de către utilizatori, cărora la momentul înregistrării li se vor atribui un profil cu care se vor putea face identificări în momentul căutării de către ceilalți utilizatori.

2.2. Obiective secundare

Scopurile secundare ale proiectului urmăresc oferirea posibilității de a împărtăși opiniile într-o manieră rapidă și facilă prin intermediul platformei:

- Exprimarea părerilor față de o postare prin apreciere și prin comentariu

Utilizatorul își poate exprima aprecierea față de postarea unui alt utilizator folosind un buton destinat special acestei acțiuni, buton sub forma de inimă, și își poate retrage aprecierea în cazul în care acesta consideră că nu este cazul său că s-a produs o greșeală. De asemenea un utilizator poate lăsa comentarii la o postare.

- Notificarea în momentul în care se produce acțiune ce îl vizează pe utilizatorul logat

Dacă utilizatorul logat va primi o apreciere, i se va comenta la o postare, va începe să fie urmarit de un alt utilizator sau primește un mesaj, atunci platforma îi va trimite automat o notificare pentru a-l înștiința. Ideea din spatele acestei funcționalități este de a putea răspunde cursiv și instant la acțiunile ce îl privesc pe acesta.

- Căutare altor utilizatori după username sau numele complet

Această funcționalitate urmărește să permită utilizatorilor posibilitatea de a căuta alții utilizatori în mod dinamic, în funcție de username-ul acestora sau după numele complet. Un câmp de căutare va trimite request-uri constant cu modificările aduse în câmp pentru a aduce din baza de date cele mai bune variante în urma căutării.

- Adăugarea de informații personale în profil

Utilizatorul logat are posibilitatea de a-și îmbogăți profilul personal prin adăugarea unei poze personale, a unui nume complet personalizat și a unei descrieri prin care acesta dorește să fie remarcat. De asemenea în detaliile utilizatorului vor apărea informații cu privire la numărul de personae urmărite, numărul de personae ce urmăresc utilizatorul și la numărul de postări adăugate de acesta.

- Crearea de postări

Un utilizator își poate exprima gândurile prin crearea unei postări personalizate ce va conține neapărat o poză, dar este posibilă și adăugarea unei descrieri corespunzătoare postării cât și crearea de etichete pentru utilizatorii existenți în platformă, aceste etichete au ca scop divertismentul cât și o împărtășire mai ușoară a posibilelor conexiuni cu alte profile în cazul ca acestea nu există.

- Urmărirea utilizatorilor ce nu se află deja în lista de urmăriți ai utilizatorului logat

Utilizatorul are posibilitatea de a-și reînnoi lista de urmăriți folosindu-se de o listă special filtrată unde el poate vizualiza și adăuga la urmăriți utilizatorii pe care acesta nu îi urmărește deja.

- Vizualizarea detaliilor unei postări

Această funcționalitate dorește crearea unei ferestre speciale unde utilizatorul poate vizualiza în deaproape o postare cu toate detaliile acesteia precum: cine a creat-o, cine este etichetat în ea, descrierea acesteia, câte aprecieri și comentarii are, cât și comentariile existente la postare.

- Vizualizarea detaliilor unui profil

Utilizatorul logat va putea să vizualizeze detaliile profilului său: poză de profil, username-ul, numele complet, descrierea, numărul de postări, numărul de personae urmărite și numărul de următori căt postările create de acesta și postările create de-a lungul timpului.

- Posibilitatea de a modifica sau șterge o postare

Dacă cel care crează o postare va dori să modifice persoanele etichetate și descrierea postării atunci acesta va avea această posibilitate. Poză postării nu va putea fi modificată,

postarea în sine fiind definite de poză creată. De asemenea utilizatorul are posibilitatea de a-și șterge postarea.

- Posibilitatea de a șterge un comentariu

Utilizatorul logat va avea posibilitatea de a-și șterge propriile comentarii lasăte la postări cât și comentariile lăsate de ceilalți utilizatori la propriile postări.

3. Tehnologii folosite

Pentru realizarea lucrării am optat pentru back-end în Spring Boot configurat pentru Java 11, iar pentru front-end am ales să folosesc Angular.

3.1. Java 11

Java este un limbaj de programare obiect-orientat, cu un scop general și destinat lucrului cu clase, special creat pentru a avea cât mai puține dependințe la implementare. Acest limbaj este destinat dezvoltării de aplicații, prin urmare, este rapid, sigur și de încredere, fiind astfel utilizat la scară largă pentru dezvoltarea de diverse aplicații dintr-o gama largă de proiecte cu diferite cerințe.

Java a fost dezvoltat de Sun Microsystems în anul 1995. James Gosling este cunoscut că tatăl Java. Înainte de Java, numele său era Oak. Deoarece Oak era deja o companie înregistrată, așa că James Gosling și echipa să au schimbat numele Oak în Java.

3.1.1. Utilitate

Unul dintre cele mai mari motive pentru care Java este atât de popular este independența platformei. Programele pot rula pe mai multe tipuri diferite de calculatoare; atâtă timp cât computerul are instalat un mediu Java Runtime Environment (JRE), poate rula un program Java. Prin urmare este foarte ușor de creat un mediu de dezvoltare în care se utilizează acest limbaj de programare. Java este utilizat în diferite domenii precum:

- i) Crearea de aplicații Android

Android este o platformă software open source ce permite dezvoltatorilor să scrie cod gestionat folosind Java pentru a gestiona și controla dispozitivul Android. Aplicațiile Android pot fi dezvoltate utilizând limbajul de programare Java și Android SDK. Deci, familiarizarea cu elementele de bază ale limbajului de programare Java este o condiție prealabilă pentru

programarea pe platforma Android. Deși există și alte modalități de a crea aplicații Android, majoritatea aplicațiilor sunt scrise în Java folosind API-ul Android Google.

ii) Createa de aplicații Web

Java este unul dintre cele mai utilizate limbaje de programare pentru dezvoltarea aplicațiilor web dinamice. O aplicație web este un software de calculator care utilizează browserul web și tehnologiile pentru a efectua sărcini pe internet. O aplicație web este implementată pe un server web.

Java oferă câteva tehnologii precum Servlet și JSP care ne permit să dezvoltăm și să implementăm cu ușurință o aplicație web pe un server. De asemenea, oferă câteva cadre precum Spring, Spring Boot care simplifică activitatea și oferă un mod eficient de a dezvoltă o aplicație web. Acestea reduc efortul dezvoltatorului.

Multe departamente guvernamentale, de asistență medicală, asigurări, educație și apărare au aplicațiile web construite în Java. Un exemplu semnificativ în acest sens este Gmail Google.

3.1.2. Spring Boot

Spring Boot este un framework pentru crearea de aplicații independente și este o extensie a framework-ului Spring. Configurarea este mai ușoară, reducând timpul de dezvoltare și crescând astfel productivitatea.

Funcțiile principale sunt configurarea automată a Spring, existența fișierului pom.xml pentru configurația Maven, integrarea cu Tomcat, Jetty, înlocuirea fișierelor XML cu comentarii etc.

Avantajele utilizării Spring Boot enumeră dezvoltarea aplicațiilor ușor de înțeles și de dezvoltat, modularitatea codului și reutilizarea acestuia, productivitatea sporită și reducerea timpului efectiv de dezvoltare.

Motivarea alegerii acestei tehnologii a constat în faptul că această permite configurarea ușoară a proiectului și a dependințelor necesare, conectarea ușoară la baza de date, existența anotarilor ce permit manipularea ușoară a codului.

3.1.3. Apache Maven

Maven este un sistem de build și administrare a proiectelor, scris în Java. Face parte din proiectele găzduite de Apache Software Foundation. Funcționalitățile sale principale sunt descrierea procesului de build al software-ului și descrierea dependentelor acestuia.

Proiectele sunt descrise printr-unul sau mai multe fișier XML, denumite POMuri (Project Object Model), dar au o structură implicită, ceea ce încurajează structurarea similară a proiectelor.

3.1.4. Hibernate

Hibernate este un framework Java care simplifică dezvoltarea aplicației Java pentru a interacționa cu baza de date. Este un instrument open source, ușor, ORM (Object Relational Mapping). Hibernate implementează specificațiile JPA (Java Persistence API) pentru persistența datelor.

Următoarele sunt avantajele cadrului de hibernare:

- Open Source și ușor

Cadrul Hibernate este open source sub licență LGPL și ușor.

- Performanță rapidă

Performanța cadrului de hibernare este rapidă, deoarece memoria cache este utilizată intern în cadrul de hibernare. Există două tipuri de cache în cache-ul de primul nivel și în cel de-al doilea nivel al cadrului de hibernare. Memoria cache de primul nivel este activată în mod implicit.

- Interrogare independentă a bazei de date

HQL (Hibernate Query Language) este versiunea orientată obiect a SQL. Generează interrogări independente din baza de date. Deci nu este nevoie să scrieți interrogări specifice bazei de date. Înainte de Hibernate, dacă baza de date este modificată pentru proiect, trebuie să schimbăm și interrogarea SQL care duce la problema de întreținere.

- Crearea automată a tabelelor

Cadrul Hibernate oferă facilitatea de a crea automat tabelele bazei de date. Deci, nu este nevoie să creați manual tabele în baza de date.

- Simplifică Complex Join

Preluarea datelor din mai multe tabele este ușoară în cadrul hibernării.

- Oferă statisticile interrogărilor și starea bazei de date

Hibernate acceptă Query cache și oferă statistici despre interrogare și starea bazei de date.

Arhitectura Hibernate include multe obiecte, cum ar fi obiectul persistent, fabrica de sesiuni, fabrica de tranzacții, fabrica de conexiuni, sesiunea, tranzacția etc.

Aplicația de hibernare poate fi creată cu adnotare. Există multe adnotări care pot fi folosite pentru a crea aplicații de hibernare, cum ar fi @Entity, @Id, @Table etc.

3.1.5. MySQL

Este un sistem de gestionare a bazelor de date relaționale, open-source, ideal atât pentru aplicații mici, cât și pentru aplicații mari, foarte rapid, fiabil, scalabil și ușor de utilizat, multiplatform și în conform cu standardul ANSI SQL.

3.1.6. JWT

Este un standard JSON pentru securitatea aplicațiilor, în special pentru autentificarea utilizatorilor în aplicații. Standardul definește un model compact și autonom pentru transmiterea sigură a informațiilor prin obiecte JSON. De fiecare dată când este apelat punctul final, acesta verifică dacă există un simbol valid și, dacă este invalid, utilizatorul nu va putea obține autorizație în aplicație. JWT poate fi semnat cu o cheie pentru verificarea și utilizarea jetoanelor.

Datorită dimensiunii săle relativ mici, un JWT poate fi trimis printr-un URL, printr-un parametru POST sau în interiorul unui antet HTTP și este transmis rapid. Un JWT conține toate informațiile necesare despre o entitate pentru a evita interogarea unei baze de date de mai multe ori. De asemenea, destinatarul unui JWT nu trebuie să apeleze un server pentru a valida simbolul.

3.1.7. Angular

Angular este o platformă și un cadru pentru crearea de aplicații client cu o singură pagină folosind HTML și TypeScript. Angular este scris în TypeScript. Implementează funcționalități de bază și optionale ca un set de biblioteci TypeScript pe care le importați în aplicațiile dvs.

4. Arhitectura sistemului

Arhitectura client / server este un model de calcul în care mai multe componente funcționează în roluri strict definite pentru a comunica. Serverul găzduiește, livrează și gestionează majoritatea resurselor și serviciilor care trebuie consumate de client. Acest tip de arhitectură de resurse partajate are unul sau mai multe computere client conectate la un server central printr-o rețea sau conexiune la internet.

Arhitectura client / server funcționează atunci când computerul client trimite o cerere de resursă sau proces către server prin conexiunea la rețea, care este apoi procesată și livrată clientului. Un computer server poate gestiona mai mulți clienti simultan, în timp ce un client poate fi conectat la mai multe servere simultan, fiecare furnizând un set diferit de servicii.

Cum funcționează modelul client-server?

Client: Când vorbim cuvântul Client, înseamnă să vorbim despre o persoană sau o organizație care utilizează un anumit serviciu. În mod similar, în lumea digitală, un client este un computer (gazdă), adică capabil să primească informații sau să utilizeze un anumit serviciu de la furnizorii de servicii (servere).

Servere: în mod similar, atunci când vorbim cuvântul Servere, înseamnă o persoană sau un mediu care servește ceva. În mod similar, în această lume digitală, un server este un computer la distanță care oferă informații (date) sau acces la anumite servicii.

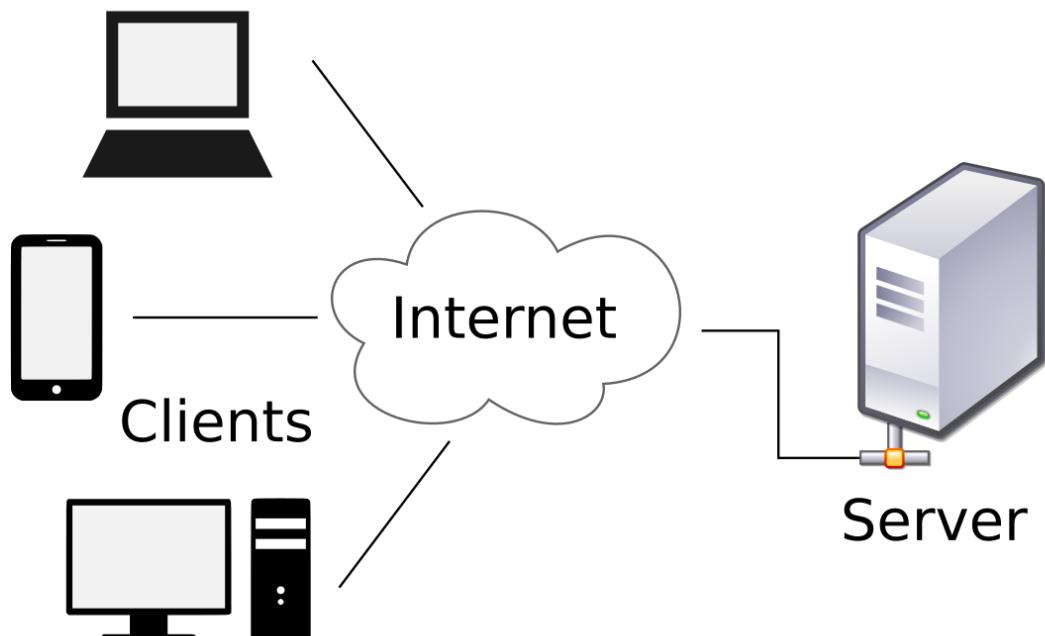


Figura 1 Model al arhitecturii de tip Client-Server

5. Arhitectura componentei backend

În vederea dezvoltării componentei backend a aplicației am optat pentru alegerea tehnologiei Spring Boot, codul fiind scris în limbajul de programare Java 11. Dezvoltarea acestei componente a urmat săblonul pus la dispoziție de arhitectură pe trei nivele cunoscută și sub denumirea de Three Layers. Această arhitectură presupune separarea în trei nivele generale a aplicației, după cum este ilustrat și în figura 2:

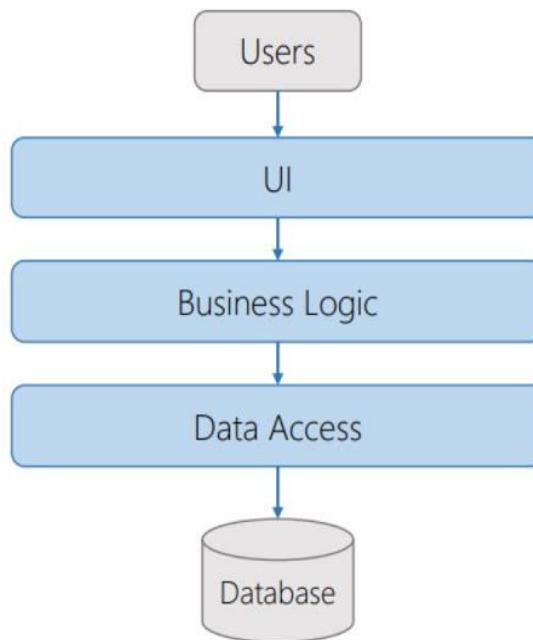


Figura 2 Model al arhitecturii de tip Three Layers

- Nivelul UI sau de prezentare se ocupă de gestionarea request-urilor făcute de utilizator pe interfață și trimitera unui răspuns browser-ului. Gestionarea acestor requesturi se face de regulă în clasele de tip Controller, unde sunt definite funcții speciale ce reprezintă endpoint-uri cărora li se vor trimite informații de pe frontend. Acest nivel este specializat pe găsirea celei mai bune funcții unde se va trece la următorul nivel.
- Nivelul de Business Logic se ocupă de partea de logică aplicată request-ului creat de utilizator la nivelul UI și aplicarea de diverse operații asupra acestora. Aceste modificări pot fi reprezentate de algoritmi de sortare, de filtrare, de crearea de obiecte DTO (Data Transfer Object) și se ocupă de crearea legăturii

cu următorul nivel. Pentru acest nivel, clasele de tip Service sunt cele mai reprezentative.

- Nivelul de Data Access, cunoscut și ca Persistence Layer, execută operații CRUD asupra obiectelor anterior gestionate în nivelul de Business Logic. Clasele de tip Repository sunt responsabile pentru de aducere a datelor, modificarea, ștergerea și inserarea acestora. Aceste clase ușurează cu mult lucrurile cu bazele de date, ele conținând funcții ce reprezintă query-uri de tip SQL ce se pot personaliza. În acest proiect am optat pentru interfetele de tip CrudRepository, pentru executarea acțiunilor asupra bazei de date, cât și pentru PagingAndSortingRepository pentru aducerea unui număr limitat de inserții din baza de date.

Motivele principale pentru alegerea acestui model de arhitectură sunt scalabilitatea îmbunătățită a aplicației, astfel se pot dezvoltă independent unele de altele cele trei nivele și posibilitatea de a dezvoltă rapid aplicația.

5.1. Detalierea componentelor

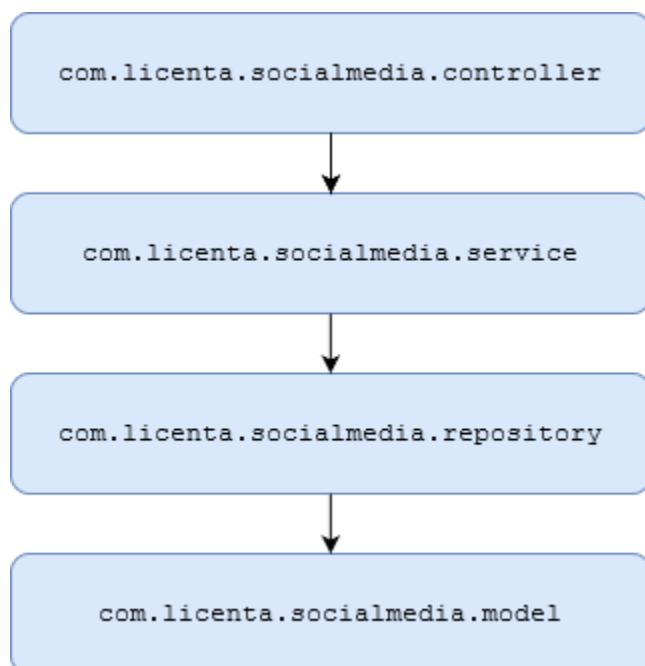


Figura 3 Arhitectura proiectului

Dezvoltarea independentă a acestor componente este posibilă datorită responsabilităților diferite a fiecărei dintre acesteia, responsabilitatile fiind foarte specifice fiecărui tip de nivel, prin urmare:

5.1.1. Controller

Clasele de tip Controller se ocupă cu gestionarea fiecărui request primit prin intermediul url-urilor. Aceste clase sunt anotate cu `@RestController`, anotare ce permite crearea de end-point-uri care vor transmite un răspuns automat în momentul în care o funcție va da return, nemaifiind nevoie de anotarea `@ResponseBody`. Aceste răspunsuri sunt în mod automat serializate sub formă de JSON în `HttpResponse`.

Un exemplu pentru un end-point creat într-o clasă de tip Controller se află în următoarea figura 4:

```
@ResponseStatus(HttpStatus.OK)
@RequestMapping(path = "/updateChat")
public Chat updateChat(@RequestBody Chat chat) throws Exception {
    return chatService.add(chat);
}
```

Figura 4

- `@ResponseStatus` este responsabilă de trimitera unui `HttpResponse` în momentul în care se va executa cu succes modificarea în baza de date a unui obiect existent de tip `Chat`.
- `@RequestMapping` are scopul de a crea un endpoint disponibil pentru noi request-uri.
- În parametrii funcției se va cere explicit un `@RequestBody` de tip `Chat`, astfel funcția așteaptă ca numai obiecte de tip `Chat` vor fi trimise pentru a fi gestionate.
- În interiorul funcției se face direct return cu răspunsul oferit de Service-ul instantiat cu denumirea de `chatService`, în acest caz se va face update la un obiect deja existent din baza de date.

Pentru utilizarea serviciilor direct injectate în interiorul unui Controller acesta va fi decorat cu anotarea `@AllArgsConstructor` ce permite utilizarea acestuia fără a crea un

constructor în care să se initializeze toate serviciile necesăre. Adnotarea @CrossOrigin marchează metodă sau tipul adnotat că permitând requesturi de tip Cross Origin.

În pachetul com.licență.socialmedia.controller am inclus clasele: ProfileController, ChatController, LikeController, UserController, AuthenticationController, PostController, CommentController, NotificationController și StoryController, clase ilustrate și în figura 5:

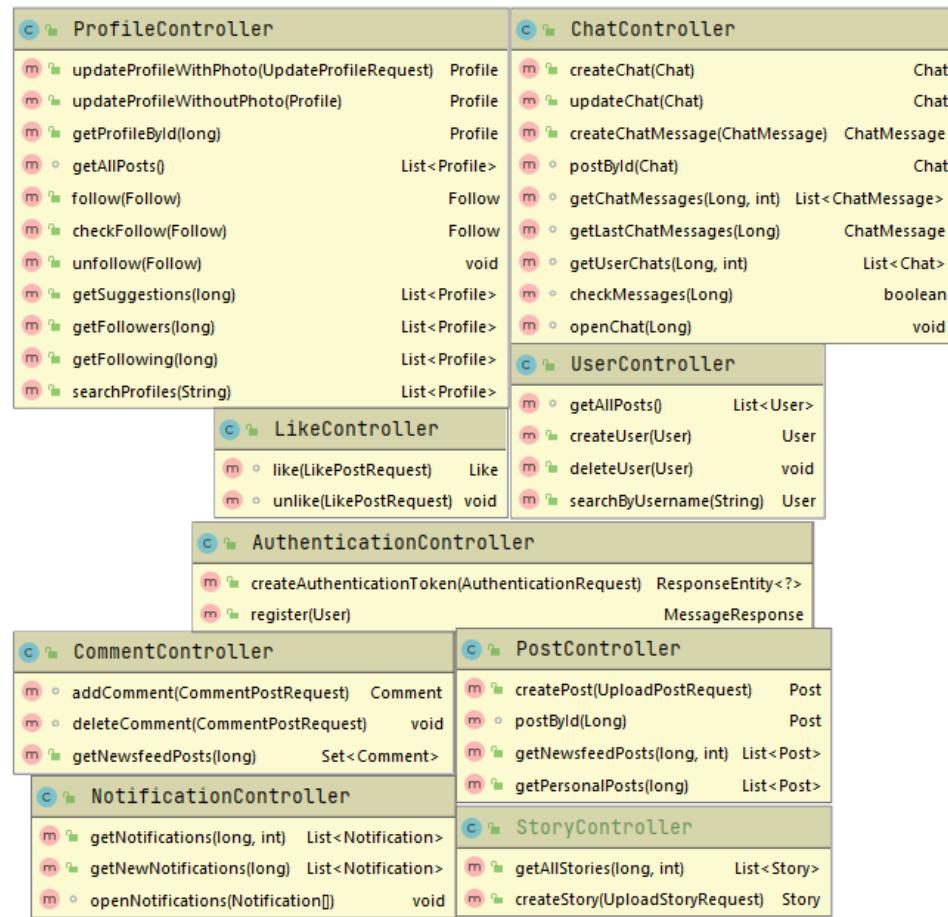


Figura 5

5.1.2. Service

Clasele de tip Service reprezintă nivelul Business Logic al aplicației , unde sunt aplicate diverse operații asupra obiectelor primite în Controllere și transmise la acest nivel. Aceste definește funcționalitățile pe furnizate în aplicație, modul în care acestea sunt accesăte și oferă modularitatea codului.

Acstea clase implementează funcții definite în interfețe ce sunt special create pentru această operație și sunt decorate cu anotarea @Service pentru a putea fi recunoscute de aplicație că servicii. Următorul exemplu reprezintă o funcție implementată din interfață IPostService în serviciul PostService:

```
@Override  
public Post add(Post post, List<Profile> followers) {  
    boolean existing=post.getId()==0;  
    post=postRepository.save(post);  
    post.setPhoto(PhotoUtils.decompressBytes(post.getPhoto()));  
    if(existing)  
        for(var profile:followers){  
            template.convertAndSend(destination: POST_CREATED+profile.getUser().getId(), post);  
        }  
    return post;  
}
```

Figura 6

În cadrul funcției din figura 6 putem observa:

- Parametrii asupra cărora se vor aplica regulile de business, parametrii ce reprezintă un obiect de tip postare și o listă de obiecte de tip Profile.
- În cadrul funcției se verifică dacă există o postare cu același ID în baza de date, dacă identificatorul unic al acestei postări este reprezentat de numărul 0, atunci se va salva obiectul exact cum a fost trimis în urma request-ului. După executarea blocului de cod conditional se va seta atributul photo cu o versiune de biti care să se aplică o decompresie. Dacă o astfel de postare nu există, atunci tuturor utilizatorilor din lista de profile să se va trimite o notificare cu noua postare creată. În final se va returna obiectul de tip post, asupra cărora s-au aplicat toate operațiile prezentate anterior.
- În cadrul serviciului PostService s-a injectat un câmp de tip IpostRepository, folosindu-se anotarea @AllArgsConstructor pentru a simplifica implementarea.

În plus față de logica aplicației, aceste clase se ocupă și de entități mapate din baza de date prin intermediul obiectelor de transfer de date (DTO). De exemplu, dacă o metodă GET specifică obține o listă de entități din baza de date, această clasă va transforma

răspunsul și va trimite lista obiectelor de transmisie la stratul de prezentare. Lista va fi recuperată de componenta front-end și va apărea în interfața cu utilizatorul.

În pachetul com.licenta.socialmedia.service am inclus clasele: NotificationService, FollowService, PostService, ChatMessageService, ChatService, UserService, ProfileService, CommentService, StoryService, LikeService, SecurityService, MyUserDetailsService.

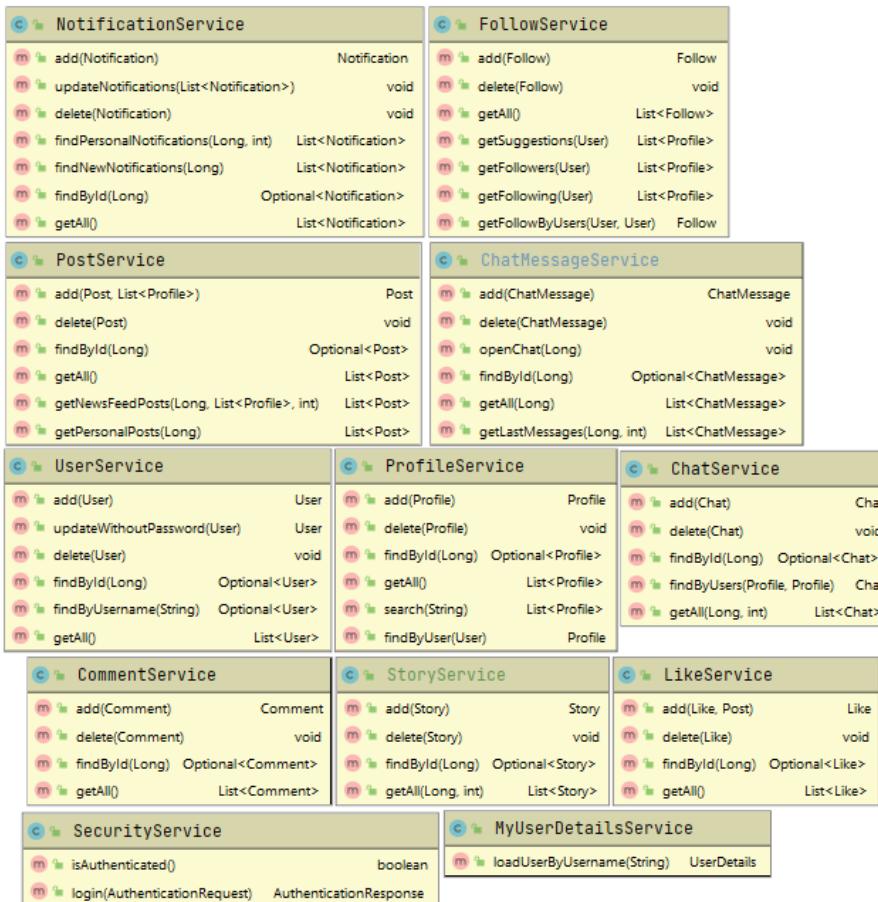


Figura 7

5.1.3. Repository

Acest tip de clase reprezintă nivelul de Data Access, cunoscut și că Persistance Layer. Aceste clase sunt responsabile pentru comunicare în mod direct cu baza de date și crearea de query-uri automate său personalizate. Toate operațiile aplicabile bazei de date se efectuează folosind funcțiile puse la dispoziție de claselele Repository. Pentru a crea un astfel de tip de

clăsă este necesă implementare funcțiilor din clasele specializate pentru gestionarea informației păstrate în baza de date.

În proiectul de față am ales să moștenesc clasele CrudRepository și PagingAndSortingRepository. Următorul exemplu ilustrează o interfață creată:

```
@Repository
public interface IChatRepository extends PagingAndSortingRepository<Chat, Long> {
    @Query(value = "select * from chat c"
        + " where c.user1_id =:id "
        + " or c.user2_id=:id ", nativeQuery = true)
    Page<Chat> findAllByUser1_IdOrUser2_Id(Long id, Pageable pageRequest);
    Chat findByUser1AndUser2(Profile user1, Profile user2);
}
```

Figura 8

- @Repository are ca scop înștiințarea aplicației cu privirea la repository-ul declarat
- Fragmentul „extends PagingAndSortingRepository<Chat,Long> specifică faptul că acest repository va crea funcții ce vor returna liste de obiecte de un anumit număr și pe o pagină specifică.
- Annotarea @Query permite crearea unui query personalizat de sintaxă SQL, astfel se vor selecta toate obiectele de tip chat din baza de date unde oricare dintre utilizatorii ce sunt înregistrati într-o conversație au id-ul specificat în parametrii funcției.
- Tipul returnat de funcție este reprezentat de pagina de obiecte Chat, pagină cu atribute specificate de parametrul Pageable pageRequest.
- Denumirea funcției corespunde acțiunii query-ului personalizat, astfel numai din nume se poate deduce rezultatul returnat.
- Parametrii reprezintă id-ul după care se face căutarea, și un obiect de tip Pagable care specifică numărul paginii ce va fi rezultat și numărul de obiecte de afișat. Astfel dacă pageRequest = PageRequest.of(0 , 5) atunci se vor afișa primele 5 elemente, iar dacă pageRequest = PageRequest.of(1 , 5) atunci se vor afișa urmatoarele 5 elemente după ce se va sări peste primele 5

În pachetul com.licenta.socialmedia.repository am inclus clasele: IFollowRepository, IChatRepository, IPostRepository, IStoryRepository, IChatMessageRepository, RoleRepository, ICommentRepository, IProfileRepository, IUserRepository, INotificationRepository, ILikeRepository.

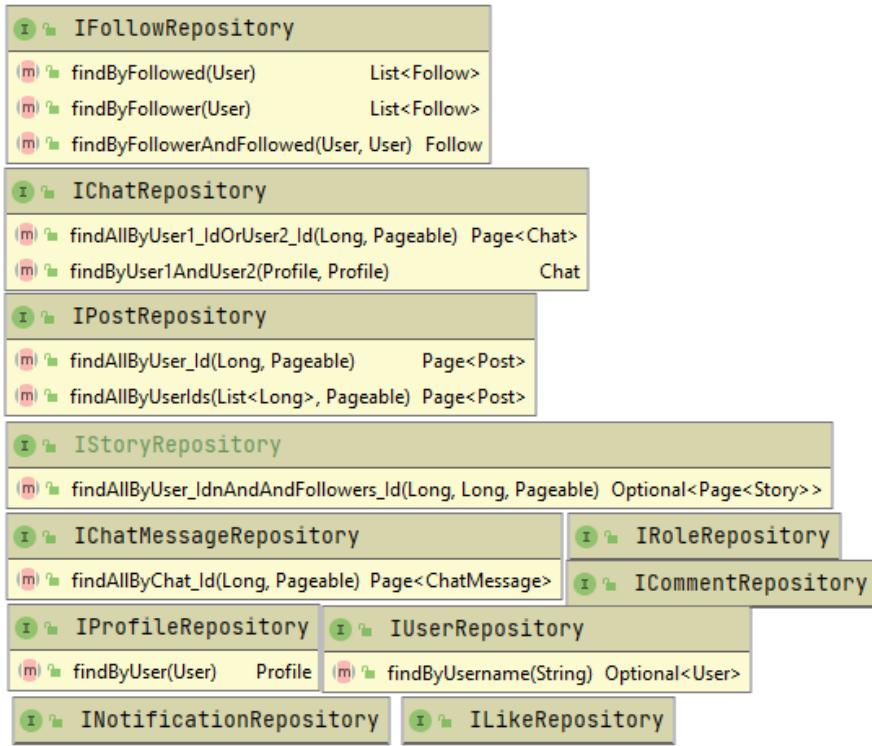


Figura 9

5.1.4. Model

Aceste clase sunt create pentru generarea în baza de date a entităților. O entitate reprezintă un tabel din baza de date, iar fiecare obiect reprezintă o linie din tabela. Atributele din aceste clase reprezintă la rândul lor coloanele tabelelor.

Pentru crearea tăbelei este necesă decorarea clasei cu anotarea `@Entity`. Această permite maparea cu entitatea din baza de date. Această adnotare va cere în mod explicit că entitatea să conțină un identificator unic declarat cu adnotarea `@Id`, căreia i se va asigura auto-incrementearea acestei chei primare folosind `@GeneratedValue(strategy = GenerationType.AUTO)`. Relațiile de agregare cu alte entități se vor declara folosind adnotări ce vor define relațiile stabilite. Exemple de relații sunt `@ManyToOne`, `@OneToOne` și `@OneToMany`.

Un exemplu de astfel de clasă este ilustrat în figura 10:

```

@Entity
@NoArgsConstructor
@AllArgsConstructor
@Setter
@Getter
@ToString
public class Like {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToOne
    @JoinColumn
    private User user;

}

```

Figura 10

- `@Entity` este folosit pentru crearea tablei
- `@NoArgsConstructor`, `@AllArgsConstructor` permit instantierea unui obiect de tip `Like` folosind oricare dintre cele două tipuri de constructori.
- `@Setter` și `@Getter` ajută la scurtarea timpului de implementare, astfel se autogenerează funcții de `Get` și `Set` pentru fiecare atribut.
- Cheia unică este decorată cu anotările `@Id` și `@GeneratedValue`
- Se stabilește o relație de tip `@ManyToOne` cu entitatea `User`.

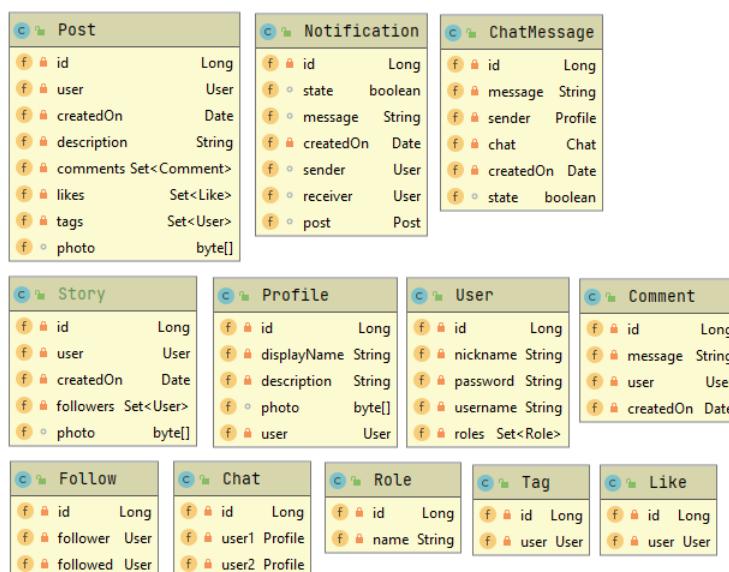


Figura 11

În pachetul com.licenta.socialmedia.model am inclus clasele: Chat, ChatMessage, Comment, Follow, Like, Notification, Post, Profile, Role, Story, Tag, User.

5.1.5. Configuration

Pachetul configuration conține clase ce implementează funcții responsabile cu configurațiile necesare aplicației. Astfel în acest pachet este creată clasă WebSecurityConfigurer ce este responsabilă cu restricționarea accesului la endpoint-urile create în controlere a utilizatorilor neautorizați, următoare funcție fiind cea care se ocupă de această acțiune:

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
    http.cors().and().csrf().disable()  
        .exceptionHandling().authenticationEntryPoint(unauthorizedHandler).and()  
        .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()  
        .authorizeRequests().antMatchers( ...antPatterns: "/css/**", "/", "/js/**", "/register", "/socket/**", "/auth").permitAll()  
        .antMatchers( ...antPatterns: "/admin/**", "/profile/**").permitAll()  
        .antMatchers( ...antPatterns: "/user/**", "/test/**").permitAll()  
        .anyRequest().authenticated();  
  
    http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);  
}
```

Figura 12

- Variabila http va gestiona sesiunile de logare astfel pentru oricare din url-urile ce contin „/register”, „/”, „/socket/**” și „/auth” se va permite accesul și fără verificarea header-ului request-ului. Orice alt request la backend va necesita autentificarea utilizatorului.
- Funcția de
http.addFilterBefore(authenticationJwtTokenFilter(),UsernamePasswordAuthenticationFilter.class) va verifica credențialele userului autentificat prin decodarea obiectului JWT și căutarea rezultatului în baza de date.

Clasă WebSocketConfigurer este responsabilă pentru înregistrarea url-urilor specific Web Socket, astfel ea implementează funcții care se ocupă cu configurația aplicației. Următoarele funcții sunt esențiale în setarea unei astfel de conexiuni:

- i) Funcția registerStompEndpoints

```

@Override
public void registerStompEndpoints(StompEndpointRegistry registry) {
    registry.addEndpoint( ...strings: "/socket" ).setAllowedOrigins("http://localhost:4200").withSockJS();
}

```

Figura 13

- Aceasta funcție este responsabilă de înregistrarea prefixului la care se vor face request-uri de către Web Socket.
- Permite requesturi de la url-ul "http://localhost:4200"

ii) Funcția configureMessageBroker

```

@Override
public void configureMessageBroker(MessageBrokerRegistry registry) {

    registry.enableSimpleBroker( ...destinationPrefixes: "/topic");
    registry.setApplicationDestinationPrefixes("/topic");
}

```

Figura 14

- Înregistrează prefixul end-point-urilor ce vor transmite informații

În pachetul com.licenta.socialmedia.configuration am inclus clasele: WebSecurityConfigurer și WebSocketConfigurer.

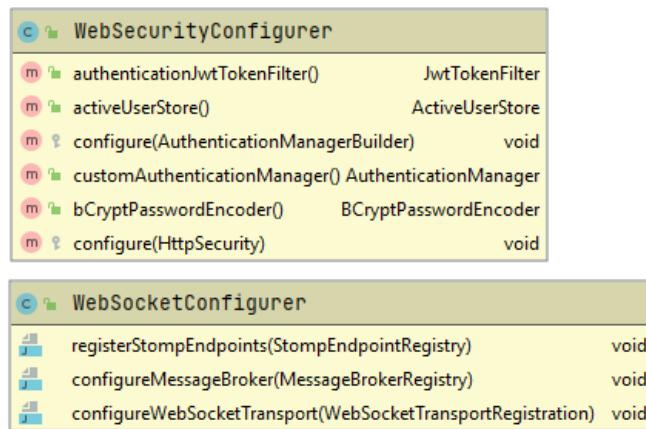


Figura 15

5.1.6. Security

Acest pachet conține clase care gestionează partea de securitate a aplicației. Rolul acestor clase este de a implementa partea de securitate a punctului final prin tokenul JWT.

- JwtUtil este însărcinat cu analizarea token-ului în obiectul User și generarea token-ului din obiectul User.
- JwtEntryPoint este concepută pentru a trimite eroare atunci când simbolul este nevalid
- JwtTokenFilter este punctul de intrare în procesul de autentificare JWT; filtrul extrage token-ul JWT din headerele requestului și delegă autentificarea către AuthenticationManager-ul injectat. Dacă jetonul nu este găsit, se aruncă o excepție care oprește requestul de la procesare. De asemenea, avem nevoie de o suprascrisere pentru autentificarea cu succes, deoarece fluxul Spring implicit ar opri lanțul de filtrare și ar continua cu o redirecționare.
- UserDetailsImpl se ocupă de procesarea token-ului transmis și stabilirea dacă există sau nu în baza de date un utilizator cu credentialele obținute în urmă decodarea token-ului.

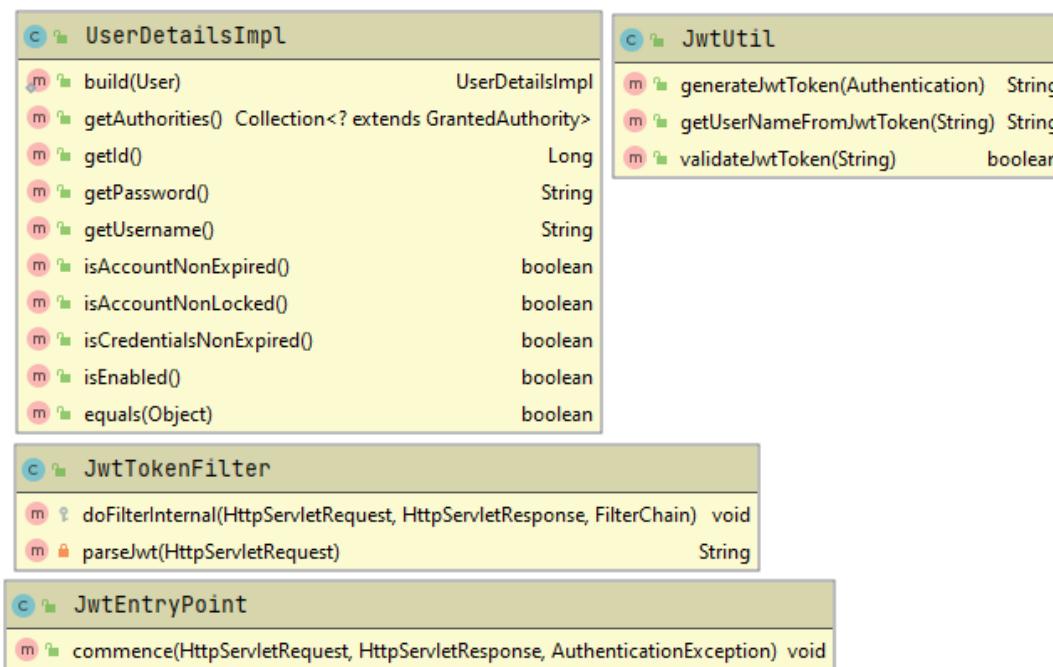


Figura 16

5.1.7. DTO

Acest pachet contine obiecte folosite pentru a transfera informatii, obiecte cunoscute și ca DTO (Data Transfer Object). Acest pachet este structurat pe doua categorii de DTO: Response și Request.

Obiectele DTO de tip Response sunt folosite în Controllere pentru a transmite informatii cat mai complete.

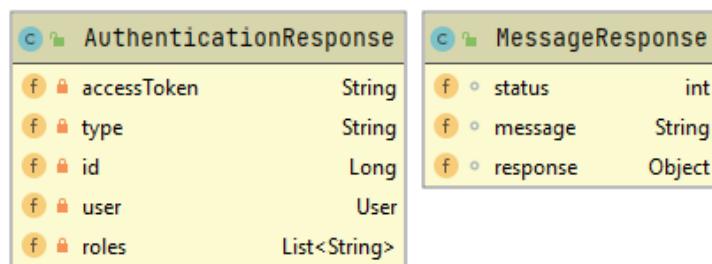


Figura 17

Obiectele DTO de tip Request sunt create în scopul de a conține informațiile a mai multe obiecte, sau informatii ce nu pot fi convertite direct cu @ResponseBody într-un anumit tip de entitate. Acest pachet contine clasele ilustrate în figura 18:

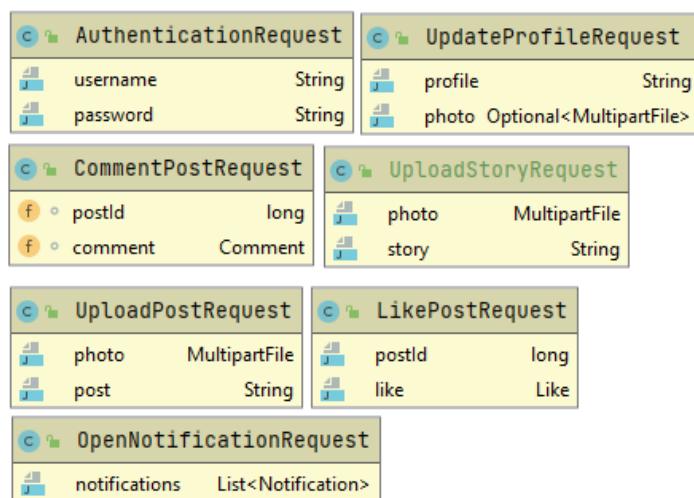


Figura 18

5.1.8. Util

Acest pachet conține clase folosite pentru diverse scopuri precum conversia imaginilor sau crearea de strîng-uri pentru endpoint-uri.

Clasă PhotoUtils este special creată pentru a facilita lucrul cu imagini în baza de date. La momentul inserării unei imagini în baza de date, această este convertită într-un vector de biți. Pentru a optimiza procesul de adăugare și de aducere a imaginii, conversia biților este o soluție optimă. Astfel în momentul inserării se fă apela funcția de compressBytes:

```
public static byte[] compressBytes(byte[] data) {
    Deflater deflater = new Deflater();
    deflater.setInput(data);
    deflater.finish();
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream(data.length);
    byte[] buffer = new byte[1024];
    while (!deflater.finished()) {
        int count = deflater.deflate(buffer);
        outputStream.write(buffer, off: 0, count);
    }
    try {
        outputStream.close();
    } catch (IOException e) {
    }
    return outputStream.toByteArray();
}
```

Figura 19

- Această funcție este responsabilă de micșorarea lungimii vectorului fără a pierde din calitatea imaginii

La momentul aducerii imaginii din baza de date, aceasta vine în forma să modificată, astfel este nevoie o decompresie a imaginii. De aceasta acțiune este responsabilă funcția decompressBytes.

```

public static byte[] decompressBytes(byte[] data) {
    Inflater inflater = new Inflater();
    inflater.setInput(data);
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream(data.length);
    byte[] buffer = new byte[1024];
    try {
        while (!inflater.finished()) {
            int count = inflater.inflate(buffer);
            outputStream.write(buffer, off: 0, count);
        }
        outputStream.close();
    } catch (IOException ioe) {
    } catch (DataFormatException e) {
        e.printStackTrace();
    }
    return outputStream.toByteArray();
}

```

Figura 20

6. Arhitectura componentei frontend

Componenta frontend a proiectului a fost dezvoltată cu ajutorul tehnologiei Angular. Paginile sunt create cu ajutorul HTML și CSS, iar funcționalitatea logică a fost scrisă în TypeScript.

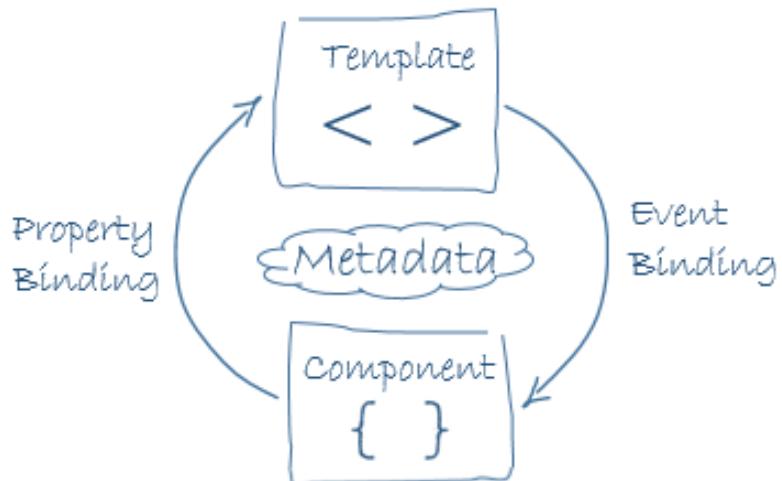


Figura 20

Angular urmărește arhitectura bazată pe componente. În acest tip de arhitectură se urmărește modularizarea codului și divizia acestuia pe cat mai multe pachete, cunoscute și

sub denumirea de module, pentru a avea usor acces la fiecare componentă și pentru a dezvolta independent unele de altele fiecare componentă. Prin urmare, o aplicație mare este divizată (decuplată) în componente funcționale și logice, cu un cod scris într-o maniera curată și care să respecte principiile MVC. Prin urmarirea principiului MVC se dorește crearea de componente cu responsabilități cat mai specifice și limitate, de scriere de cod specializat pe o serie limitată de acțiuni. Un modul în aplicația Angular este un grup de componente, directive, pipe-uri și servicii, care sunt legate de aplicație și construiesc în mod colectiv o funcționalitate comună.

Angular oferă o mulțime de module de construcție, cum ar fi modulul http (pentru a efectua apeluri http din aplicație). Este obligatoriu ca fiecare aplicație Angular să aibă un modul root. Fiind o aplicație web cu o singura pagină, Componenta AppComponent reprezintă de fapt containerul în care se vor folosi pe rand, în funcție de acțiunile și alegerile utilizatorului, partile funcționale și vizuale ale componentelor definite. Aceasta componentă este de fapt componenta principală a aplicației, care se află în fișierul app.component.ts.

Fiecare componentă, serviciu, directivă, pipe și model utilizate în aplicație trebuie declarate în fișierul app.modules.ts, altfel va fi aruncată o eroare în consola browserului. Fișierul app.routing.modules.ts conține AppRoutingModule care configurează routerul de rutare. Ruta îi spune routerului ce vizualizare trebuie afișată pentru fiecare legătură utilizată în aplicație, specifică adresă URL din cale, iar în componentă specifică componentele pe care routerul trebuie să le creeze atunci când navighează la calea specificată în cale. Imaginele folosite în aplicație sunt în „src / assets / images”.

Pentru a avea o mai bună evidență a locației componentelor, proiectul este structurat pe pachetele: component, service, pipe, model.

6.1. Pachetul component

Acest pachet conține toate interfețele grafice ale aplicației, o componentă reprezentând de fapt o colecție logică de template-uri HTML, de stiluri asociate și modele care ar putea să opereze într-un template folosind îmbinările/legăturile din Angular.

O componentă în Angular este un element de construcție al UI și constă dintr-un fișier de clasă, un template și un fișier CSS pentru stilizarea componentei. Clasă este scrisă în

limbaj TypeScript și este decorată cu adnotarea @Component, că în figura 21, ce este utilizată pentru a specifică metadata clasei.

```
@Component({
  selector: 'app-suggestions',
  templateUrl: './suggestions.component.html',
  styleUrls: ['./suggestions.component.css']
})
```

Figura 21

Decoratorul @Component () specifică următoarele informații specifice Angular:

- Un selector care definește modul în care este utilizată componentă într-un alt template. Acest selector are o denumire unică, astfel în momentul în care în interiorul unui fișier HTML se va crea un element, spre exemplu, de tipul „<app-suggestions/>”, aplicația va știi că este necesăra înlocuirea spațiului alocat cu componentă specificată.
- Un fișier HTML care instruiește Angular cum să redea componentă. Cu acest template se construiește partea vizuală a componentei ce poate conține informații în tabele, diferite tipuri de elemente de introducere a datelor precum casuțe de scriere, date-picker-uri, butoane etc. Tot în această componentă se pune în valoare funcția de date binding pe care Angular-ul o pune la dispoziție, astfel eliminându-se mult cod. Sincronizarea elementelor cu variabilele și funcțiile din clasă componentei se face dinamic.
- Un set optional de fișiere CSS care definesc aspectul elementelor HTML ale template-ului.

Componentele pot implementa funcții ce ajută la manipularea informației și modificarea template-ului precum funcția “ngOnInit()”. Această este responsabilă de aplicarea de diverse acțiuni asupra atributelor utilizate în Data Binding înainte că fișierul HTML să construiască elementele vizuale.

În figura 22:

```
suggestions: Suggestion[] = []
async ngOnInit() {
  await this.profileService.getSuggestions().then(data => {
    data.forEach(profile => {
      this.suggestions.push({ profile, disabled: false })
    });
  }).catch(err => { console.log(err) })
  this.loaded = true
}
```

Figura 22

- În momentul inițializării componentei, lista “suggestions” nu conține niciun element. Această fiind setată cu o lista goală.
- În interiorul funcției “ngOnInit()” se apelează funcția asincronă getSuggestions() din serviciul ProfileService, ce va adduce din baza de date informațiile dorite.
- Cu ajutorul funcției “forEach()” se parcurge fiecare element din rezultatul obținut și se va adăuga în lista “suggestions”
- Funcția “ngOnInit()” este declarată că fiind asincronă deoarece este necesă folosirea cuvântului cheie “await” pentru a aștepta executarea funcției asincrone “getProfiles()”, o abordare diferită, în care nu se va folosi perechea “async-await” poate genera probleme grave aplicației dacă viteza de aducere a datelor nu este una mare.

Funcția “ngAfterViewInit()” se ocupă cu modificarea atributelor după momentul creării componentelor UI.

```
@ViewChild(MatMenu, { static: false }) matMenu!: MatMenu;
@ViewChild(MatMenu, { static: false }) matMenu2!: MatMenu;
ngAfterViewInit(): void {
  this.matMenu.closed.subscribe(() => {
    this.onMenuClose()
  })
  this.matMenu2.closed.subscribe(() => {
    this.onMenuClose()
  })
}
```

Figura 23

- În interiorul funcției se va stabili comportamentul elementului de UI "MatMenu", care la momentul închiderii va apela funcția "onMenuClose()" ce va produce modificări asupra componentei.

De asemenea fiecare componentă conține câte un constructor ce este responsabil de instantierea atributelor din cadrul clasei. Aceste componente sunt reutilizabile, prin urmare pot fi utilizate în orice altă parte a aplicației. De asemenea ele sunt independente, prin urmare pot fi testate independent, această arhitectură face codul foarte testabil.

În pachetul component am definit componente pentru fiecare pagina de care am avut nevoie în platforma Social Media.

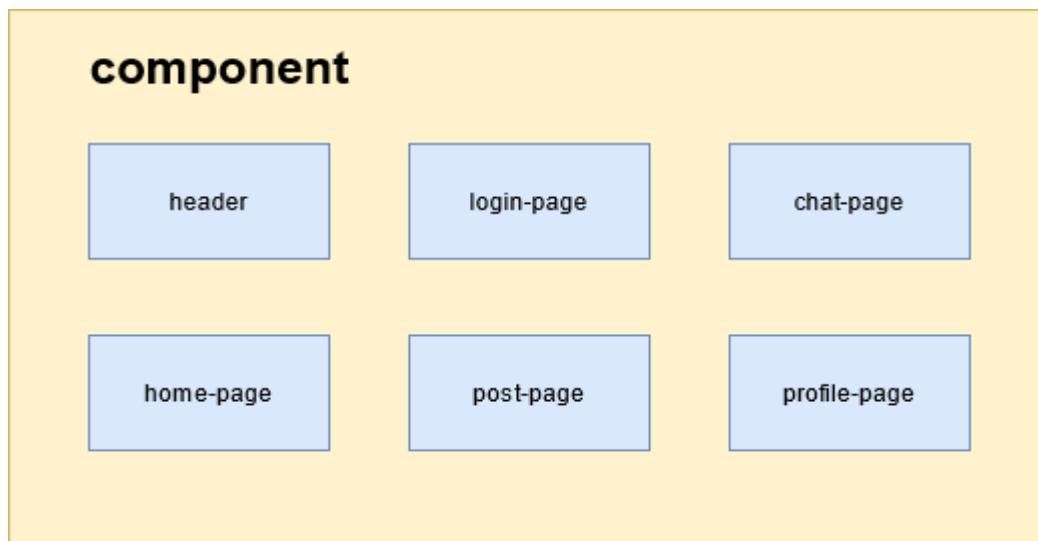


Figura 24

6.1.1. Componența Header

Această componentă conține butoane și elemente necesare peste tot în aplicație.

Butonul de „home” redirecționează aplicația către componentă unde se pot vizualiza postările create de ceilalți useri, sugestiile de urmărire și story-urile. Redirecționarea la componentă Home se face adăugând în template-ul HTML „routerLink="/home"" în declararea elementului img ce continuă și locația imaginii.

Un câmp special de căutare permite utilizatorilor să caute alți utilizatori ai aplicației prin introducerea numelui complet al acestora sau a username-ului cu care sunt înregistrat. Pentru a oferi sugestii pe măsură ce noi caractere sunt introduse, am folosit elemental

Autocomplete din Angular Material, care pe măsură ce se modifică sirul de caractere introdus, va sugera o serie de recomandări ce corespund căutării. Pentru implementarea acestei funcționalități a fost necesă adăugare unui element HTML de tip input, care în momentul modificării va apela funcția "onSearchChange()". De asemenea, crearea elementului de Autocomplete necesită o structură aparte, după cum ilustrează și figura 25:

```
<mat-autocomplete #auto="matAutocomplete">
  <mat-option *ngFor="let option of options" [value]="option">
    <button mat-menu-item [routerLink]="/profile" [queryParams]="{ username:option.user.username }"
      <div class="row" style="width: 100%; margin-left: 0;">
        <div class="col-sm-4">
          <img [src]= "option.photo ? (option.photo| image) : 'assets/resources/user.png'" style="width: 100%; height: 100%; object-fit: cover; border-radius: 50%;">
          <div class="col-sm-8">
            <label style="font-size: smaller;">{{option.user.nickname}}</label>
          </div>
        </div>
      </div>
    </button>
  </mat-option>
</mat-autocomplete>
```

Figura 25

- Secvența “`*ngFor="let option of options"`” este responsabilă de parcurgerea tuturor elementelor din lista de obiecte “options”, lista filtrată în momentul tastării în câmpul de căutare.
 - `[value]="option"` atribuie valorii selectate obiectul `option`, astfel se vor reține în întregime toate attributele aflate în `option`.
 - Structura button are că scop afisarea datelor despre utilizatorul căutat, iar în momentul în care este selectat, apăsând click, una dintre sugestii, atunci se va naviga către componentă “`ProfileComponent`” unde se va descifra obiectul trimis prin `“[queryParams] = “{username:option.user.username}””`

O componentă pentru notificări este de asemenea adăugată în header. Această conține un badge ce, în momentul în care se crează noi notificări destinate utilizatorului logat, se modifica. Această componentă conține și o listă cu notificări filtrate pe utilizatorul logat și pe data la care acestea au fost create.

După cum este ilustrat în figura 26, lista de notificări conține un număr limitat de notificări, iar în momentul în care se apasă pe butonul de "Load more notifications.", se vor aduce următoarele notificări, filtrate după date creării. Pentru identificarea noilor notificări, componenta HeaderComponent se abonează în constructor la metoda „findNew()” din serviciul NotificationService ce menține constantă comunicarea cu backendul pentru end-point-ul „/topic/notifications/new/”, căreia i se concatenează și id-ul utilizatorului logat. Acest endpoint este înregistrat pentru a deservi la conexiunea realizată de serviciul WebSocketService.

Similar cu componentă de notificări este creată și componentă de chat, această fiind notificată de fiecare dată când se crează un nou mesaj destinate utilizatorului logat. Ultimele mesaje sunt filtrate astfel încât utilizatorul să vadă în ordine descrescătoare dată la care au fost create toate mesajele. Mesajele necitite sunt evidențiate. Iar utilizatorul are posibilitatea de a accesă pagină destinate chat-ului unde și poate selecta mesajele pe care vrea să le deschidă.

După cum este ilustrat în figura 26, lista de notificări conține un număr limitat de notificări, iar în momentul în care se apasă pe butonul de "Load more notifications.", se vor aduce următoarele notificări, filtrate după date creării. Pentru identificarea noilor notificări, componentă HeaderComponent se abonează în constructor la metoda „findNew()” din serviciul NotificationService ce menține constantă comunicarea cu backendul pentru end-point-ul „/topic/notifications/new/”, căreia i se concatenează și id-ul utilizatorului logat. Acest endpoint este înregistrat pentru a deservi la conexiunea realizată de serviciul WebSocketService.

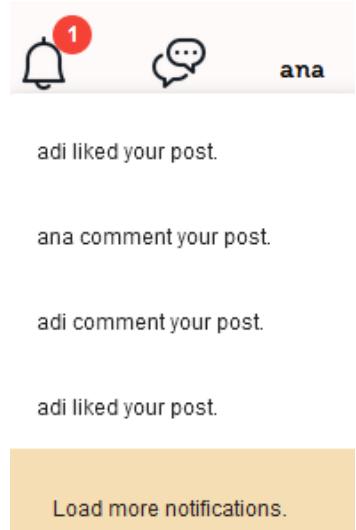


Figura 26

Tot în componentă Header, utilizatorul are posibilitatea de a-și accesă profilul personal folosind elementele de UI illustrate în figura 27:



Figura 27

Aceste elemente folosesc secvență routerLink="/profile" [queryParams]="{{username:profile.user.username}}" pentru a naviga pe pagină utilizatorului logat, unde se pot vizualiza mai multe detalii.

Cea din ultima acțiune ce se poate executa folosind elementele prezentate în componentă Header este delogarea din sesiune actuală. Butonul de delogare, ilustrat în figura 28, apelează funcția ce este responsabilă cu ștergerea itemului "currentProfile" din localStorage și resetarea token-ului de autentificare.

```
logout() {  
  localStorage.removeItem('currentProfile');  
  this.authenticationService.logout()  
}
```



Figura 28

6.1.2. Componenta Profile-Page

Componenta Profile-Page conține subcomponente specializate pentru afișarea unor detalii despre profilul accesat. Aceste subcomponente sunt reprezentate de componentele ProfileInformationComponent, ProfilePostsComponent, EditProfileComponent și FollowDetailsComponent. Aceste componente sunt combinate în ProfileComponent, acesta reprezentând componentă principala.

În figura 29 este prezentată interfață vizuală a componenței profile, după cum se poate observă, componentă Profile conține informații referitoare la profilul accesat precum: usernamel user-ului, fotografie de profil al acestuia, numărul de postări create, numărul de persoane urmărite și de următori, numele complet și o descriere personalizată. Utilizatorul își poate modifica date profilului dacă profilul accesat este cel personal, în caz contrar atunci poate trimite un request de urmarie apasând butonul „Follow” și poate vizualiza toate postările create de utilizatorul cu profilul accesat. În cele ce urmează voi discuta despre fiecare sub-componentă din componentă Profile:

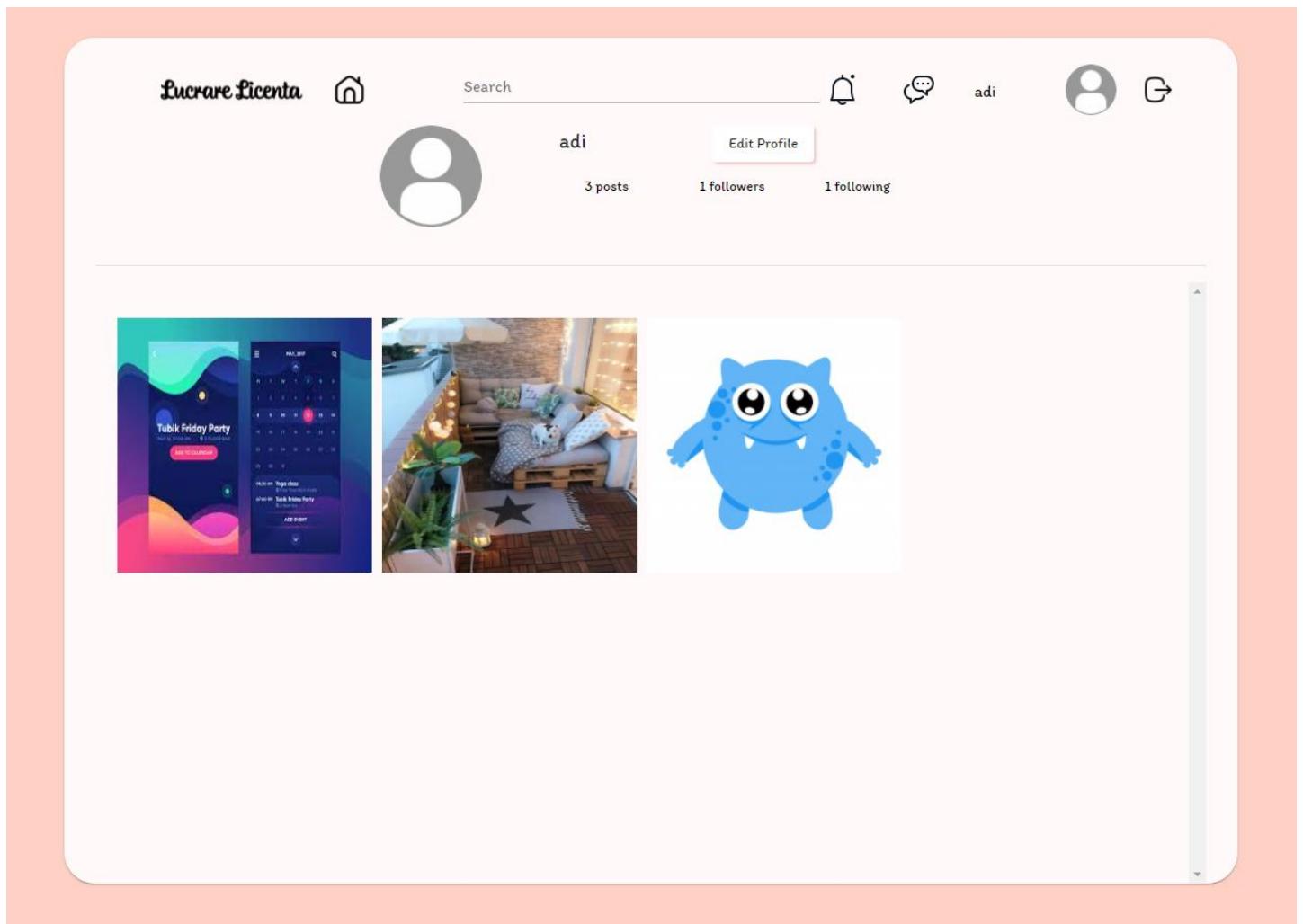


Figura 29

6.1.2.1. ProfileInformationComponent

Această componentă primește username-ul trimis prin queryParams la navigare și initializează componenta UI astfel încât atributele să corespundă profilului accesat. În funcția

```
editProfile() {
    this.dialog.open(EditProfileComponent, {
        width: '600px',
        height: '700px'
    })
}
```

Figura 30

„ngOnInit()” se verifica dacă utilizatorul logat este același cu cel trimis prin queryParams.

Dacă verificarea are un rezultat pozitiv atunci atributul “profile” va fi inițializat utilizându-se funcția “getPersonalProfile ()” din ProfileService, iar variabilă ce este responsabilă cu afișarea butonului de “Edit Profile”, variabilă “isPersonalProfile, se va seta cu valoarea true. Dacă această valoare este adevărată atunci utilizatorul își va putea modifica detaliile profilului apelând funcția din figura 30 ce va deschide o fereastră cu componentă EditProfileComponent.

În cazul în care variabilă isPersonalProfile este setată că fiind false butonul de “Edit Profile” va fi înlocuit de butonul “Follow”, a cărui funcționalitate am enunțat-o anterior, iar atributul “profile” se va inițializa cu profilul rezultat în urmă apelării funcției “getProfile(User)” ce va returna profilul accesat. Se va mai face o verificare pentru a stabili dacă profilul accesat este deja urmărit de utilizatorul logat, iar în cazul în care se stabilește că această verificare este adevărată atunci butonul de “Follow” va fi înlocuit cu un buton de “Unfollow”.

Tot în această componentă putem deschide liste de profile urmărite sau profilele urmăritorilor cu ajutorul funcției din figura 31:

```

followDetails(value:boolean) {
  let details
  if(value)
    details={profiles:this.followers,followers:true}
  else
    details={profiles:this.following,followers:false}
  this.dialog.open(FollowDetailsComponent,{
    width: '600px',
    height: '700px',
    data: {dataKey: details}
  })
}

```

Figura 31

- Parametrul „value” primește din template-ul HTML tipul listei de afisăt, dacă utilizatorul solicită să vizualize lista de persoane ce îi urmăresc profilul atunci parametrul va fi setată cu valoarea true, în caz contrar acesta va primi valoarea false.
- În cadrul funcției se verifică valoarea parametrului „value” iar în funcție de rezultat se initializează variabilă locală „details” cu lista de obiecte de tip Profile solicitată și variabilă „followers” cu valoarea true în cazul în care se dorește vizualizarea listei de următori sau false în caz contrar.
- Funcția „this.dialog.open()” se ocupă cu deschiderea unei ferestre ce conține componentă FollowDetailsComponent, căreia îi este injectată informația creată în variabilă „details”.

6.1.2.2. ProfilePostsComponent

În această componentă se va stabili profilul accesat, în aceeași manieră cum s-a stabilit și pentru ProfileInformationComponent. În cadrul funcției „ngOnInit()” se vor încărca datele necesare din PostService pentru profilul accesat și se va inițializa lista de obiecte de tip NewsFeedPost, posts, cu ajutorul funcției „getPersonalPosts()” ce va returna exclusiv postările create de utilizatorul deținător al profilului.

După inițializarea listei posts, această va fi parcursă element cu element în cadrul ngFor din fișierul HTML unde fiecare postare va fi încadrată folosind „mat-grid-list”, componentă ce permite afișarea tabulară a imaginilor. Pe fiecare dintre elemente le vom

putea vizualiza într-o fereastră separată, apasând pe imaginea reprezentativă postării. În urmă acestei acțiuni se va apela funcția din figura 32 ce este responsabilă cu deschiderea unei ferește ce conține detalii despre postare.

```
popupPost(post:NewsFeedPost) {  
  this.dialog.open(PostPopupComponent, {  
    width: '900px',  
    height: '780px',  
    data: {  
      dataKey: post  
    }  
  })  
}
```

Figura 32

6.1.2.3. EditProfileComponent

Componenta responsabilă cu modificarea detaliilor existente despre profilul utilizatorului, detalii precum fotografia de profil, username-ul, numele complet și descrierea.



Figura 33

Interfața grafică a acestei componente este ilustrată în figura 33. Setarea imaginii se face cu ajutorul unui câmp de tip input ce este restrictionat să primească doar fișiere de tip imagine.

În momentul în care este selectată o imagine din browser-ul deschis se vă apela funcția „preview()” ce vă modifica calea imaginii actuale cu noua cale a imaginii convertite în base64. Variabila „selectedFile” este reinitializată cu noul fișier selectat.

În momentul apasării butonului „Save” se va apela funcția „saveProfile()” ce este responsabilă de trimitera parametrilor de tip Profile și File către metoda „updateProfile” din ProfileService. Butonul “Discard” închide fereastra fară a executa operația de modificare a profilului.

6.1.2.4. FollowDetailsComponent

Prin această componentă, utilizatorul poate vedea atât lista de următori cât și lista de profile urmărite, tipul listei se stabilește în ProfileInformationComponent. În figura 34 este ilustrată o lista de persoane urmărite, cărora utilizatorul logat le poate aplica acțiunea de „unfollow”. În figura 35, profilelor din lista utilizatorul le poate da „remove”.

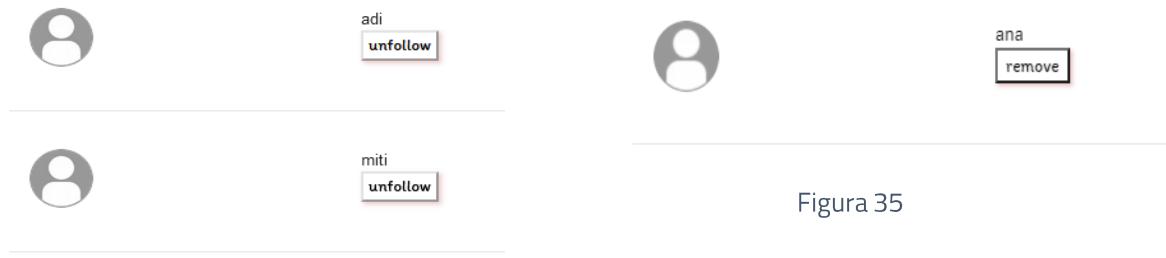


Figura 35

Figura 34

6.1.3. Componenta Login

Această componentă este responsabilă de înregistrarea și autentificarea utilizatorului. Această componentă este cea care este apelată în momentul deschiderii aplicației. În interiorul constructorului se verifică dacă este deja creată o sesiune de logare pentru utilizator, verificându-se dacă funcția „getCurrentUser()” din AuthenticationService returnează sau nu o valoare. În cazul în care rezultatul funcției este diferit de „undefined”

atunci se consideră că există o sesiune de logare încă valabilă și se vă naviga spre componenta HomeComponent. În caz contrar utilizatorul are puse la dispoziție următoarele scenarii:

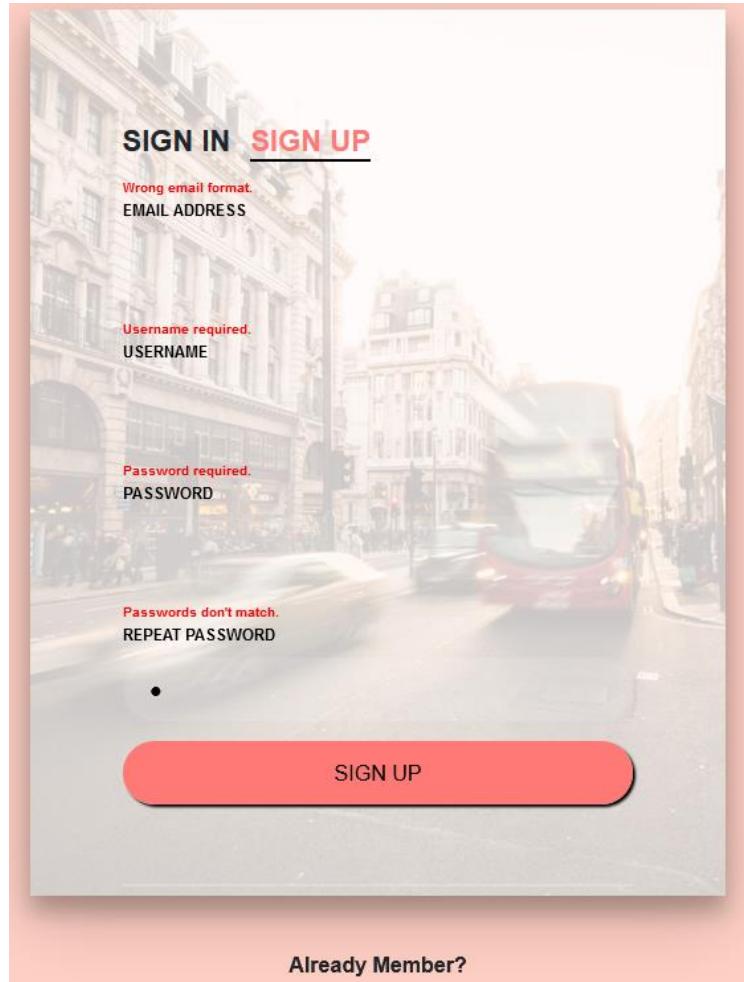


Figura 36

Utilizatorul își poate crea cont în secțiunea „SIGN UP”:

- Utilizatorul trebuie să-și introducă un email, un username (care vă fi echivalentul nickname-ului în aplicație), trebuie să introducă o parola și să o repete.
- Parolele trebuie să corespunda.
- Toate câmpurile menționate sunt obligatorii, în cazul în care un câmp este omis, nu se vă putea trimite request-ul responsabil de înregistrarea utilizatorului după cum se poate observa și în figura 36.

- S-a folosit un FormGroup pentru crearea câmpurilor necesare înregistrării și pentru aplicarea validatorilor.
- În momentul completării corecte a câmpurilor, Authentication Service va apela funcția „register()” ce va primi ca parametru datele din registerForm. Prin abonarea la această funcție se așteaptă un răspuns de la server, în cazul în care răpsunsul are statusul 201, atunci se va face logarea automată cu noul cont creat.

Utilizatorul se poate autentifica în secțiunea „SIGN IN”:

- Pentru crearea câmpurilor de introducere a e-mailului și a parolei am folosit un FormGroup denumit loginForm. Acesta este responsabil de validarea câmpurilor și nu permite apelarea funcției de „login()” a AuthenticationService dacă datele introduse nu sunt valide.
- Dacă loginForm este valid, atunci se permite apelarea funcției „login()”, iar dacă răspunsul server-ului este folosit pentru înregistrarea în localStorage a detaliilor referitoare la utilizatorul logat și se va naviga către componenta HomeComponent.

```
async login() {
  if (!this.loginForm.valid)
    this.attempedLogin = true
  else
    this.authenticationService.login(this.loginForm.getRawValue())
      .subscribe(async data => {
        this.profileService.getProfile(data.user)
          .then(data => {
            this.profileService.setCurrentProfile(data)
            this.router.navigateByUrl("/home");
          });
      }, error => {
    });
};
```

Figura 37

6.1.4. Componenta Home-Pages

Această componentă este divizată în alte șase sub componente ce sunt apelate din componenta principală HomeComponent. În cele ce urmează voi aborda individual fiecare componentă:

6.1.4.1. HomeComponent

Prin componenta HomeComponent se pun la dispoziție trei componente după cum ilustrează și figura 38, această componentă deservind drept container pentru a permite utilizatorului să poată vizualiza o varietate de elemente diferite precum sugestii de urmărire, postările create de persoanele urmărite, posibilitatea de a crea postări și poveștile adăugate de utilizatorii urmăriți. Puse împreună, aceste componente alcătuiesc imaginea din figura 38:

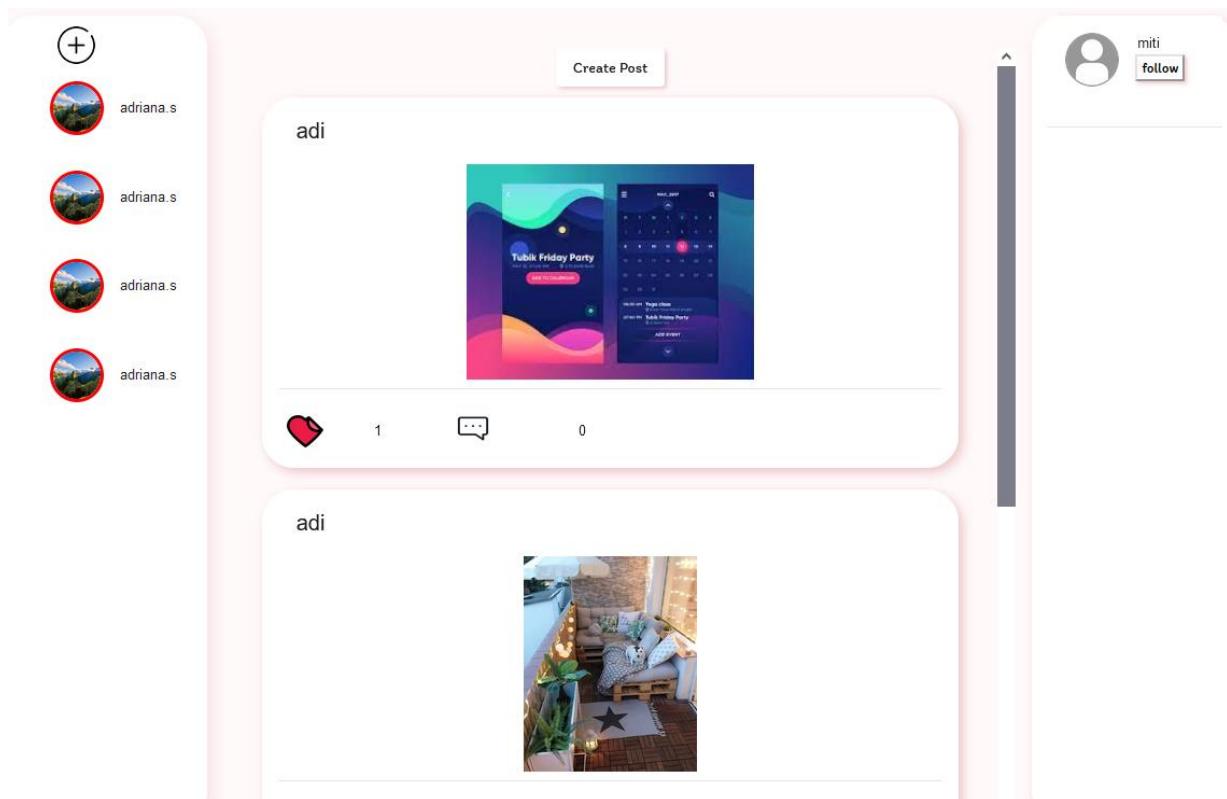


Figura 38

6.1.4.2. NewsFeedComponent

Această componentă este responsabilă de aducerea celor mai recente postări adăugate de utilizatorii pe care utilizatorul logat îi urmărește. Aici utilizatorul are posibilitatea de a crea noi postări, de a aprecia postările de pe pagina și de a încărca mai multe postări pe

măsură ce acesta navighează printre postările existente. În cele ce urmează, voi enumera cele mai importante funcționalități din această componentă.

- Butonul „Create Post” este responsabil de deschiderea unei ferestre ce conține componenta CreatePostComponent, unde utilizatorul are posibilitatea de a adauga o nouă postare. De deschiderea acestei ferestre este responsabilă funcția „createPost()”, ilustrată în figura 39:

```
createPost() {
  this.dialog.open(CreatePostComponent, {
    width: '600px',
    height: '700px'
  })
}
```

Figura 39

- Afisarea postărilor create de utilizatorii urmăriți se face prin apelarea funcției „findAll()” din cadrul serviciului PostService, unde se va face request către endpoint-ul responsabil de aducerea postărilor solicitate. Prin abonarea la aceasta funcție se așteaptă răspunsul de la backend, răspuns care va fi convertit și adăugat în lista „posts”. Funcția responsabilă de convertire, verifică dacă postarea este apreciată de utilizator, stabilește numărul de aprecieri și de comentarii și crează un obiect de tip NewsFeedPost.
- Utilizatorul poate aprecia o postare apăsând butonul în formă de inima care va apela funcția „onLike” din figura 40, în cadrul funcției se va incrementa numărul de aprecieri după ce se va face abonarea la funcția „likePost”,

```
async onLike(event: NewsFeedPost) {
  this.postService.likePost(event.post.id)
    .subscribe(()=>{
      event.liked=true
      event.numberOfLikes+=1
    })
}

async onUnlike(event: NewsFeedPost) {
  this.postService.unlikePost(event.post.id)
    .subscribe(()=>{
      event.liked=false
      event.numberOfLikes-=1
    })
}
```

Figura 40

responsabilă de trimiterea request-ului, ce conține informații despre apreciere, către backend.

- În cazul în care utilizatorul vrea să-și retragă aprecierea atunci acesta poate apăsa pe butonul în formă de inimă, ce este colorat cu roșu, și se va apela funcția „onUnlike”, a cărei funcționalitate este similară cu cea de „onLike”, diferențele constând în faptul că se va decrementa numărul de aprecieri, iar request-ul responsabil de ștergerea aprecierii va apela un endpoint specializat pe aceasta acțiune.
- În momentul în care utilizatorul ajunge la finalul listei de postări, prin bottom-scroll, atunci va putea observa un buton „Load more”, care în momentul apăsării va face un request către backend ce va returna urmatoarele postări, în ordinea lor cronologică. Prin apăsarea butonul se va apela funcția „loadMorePosts” care va incrementa și variabila „loadMorePostsRequestNumber”, variabila utilizată pentru paginarea postărilor în backend.
- Dacă un utilizator urmarit adaugă o nouă postare, atunci în această pagina va apărea un buton ce permite încarcarea celor mai recente postări. Această funcționalitate este posibilă prin abonarea la suncția „onPost()” din PostService, unde serviciul de Web Socket înregistrează endpoint-ul '/topic/posts/created/', căruia îl se concatenează și id-ul utilizatorului logat. Cu acest endpoint se comunica din backend și se notifică în momentul în care un utilizator urmarit crează o nouă postare.
- Prin apăsarea pe imaginea unei postări se va deschide o fereastră cu componenta PostPopupComponent, unde sunt prezentate detalii despre postare.

6.1.4.3. SuggestionsComponent

Aceasta componentă este responsabilă de afișarea utilizatorilor pe care utilizatorul logat nu îi urmărește. În funcția „ngOnInit()” este apelată funcția „getSuggestions()” din ProfileService, iar prin abonarea la această funcție se va folosi rezultatul pentru a initializa



Figura 41

lista suggestions. În momentul apasării butonului „follow”, ilustrat în figura 41 se va apela funcția „follow()”.

Funcția „follow()” are urmatoarea structură:

- Prin abonarea la funcția „follow()” din ProfileService, ce primește ca parametru utilizatorul ce urmează să fie urmarit, se va dezactiva butonul de „follow” asociat utilizatorului selectat. După trei secunde se va refiltra filtra de sugestii

```
async follow(user: User) {
  this.profileService.follow(user).subscribe(async () => {
    this.suggestions.forEach(value => {
      if (value.profile.user.id === user.id)
        value.disabled = true;
    });
    await this.delay(3000)
    this.suggestions = this.suggestions.filter(suggestion => suggestion.profile.user != user)
  });
}
delay(ms: number) {
  return new Promise(resolve => setTimeout(resolve, ms));
}
```

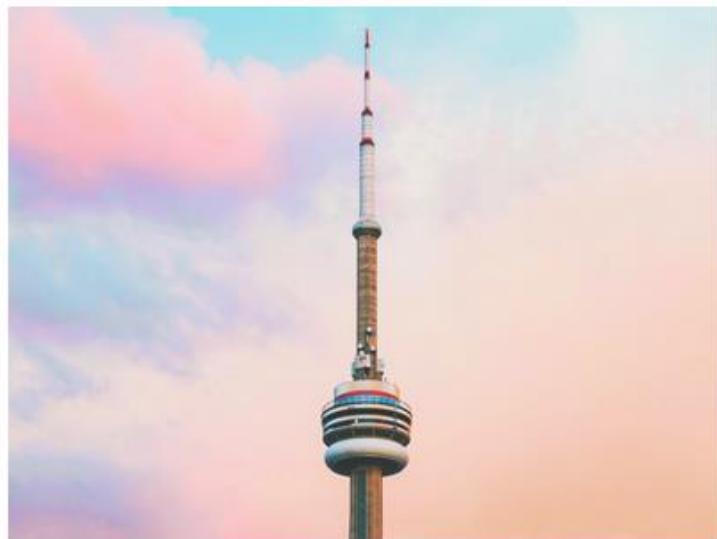
Figura 42

astfel încât să nu contină utilizatorul selectat.

6.1.4.4. CreatePostComponent

Aceasta componentă este utilizată din componenta NewsFeedPostComponent și este folosită pentru crearea unei noi postări, continând un câmp input de tip file, un câmp pentru descriere și un autocomplete cu ajutorul căruia se pot eticheta utilizatori în postare.. Interfața vizuală este ilustrată în figura 43, și pe baza acesteia voi explica funcționalitatile existente în componentă:

- Adăugarea de imagini se face cu ajutorul unui câmp input ce poate primi doar fișiere de tip imagine. Înainte de selectarea unei imagini, sursă componentei img va conține o imagine săblon, astfel încât utilizatorul să-și dea seama cu usurința de faptul că trebuie să apese pe imagine pentru a selecta un fișier. În momentul în care se apasă pe imagine se va apela funcția „preview()”, care



DESCRIPTION

Tower up. #stay #positive

TAG PEOPLE

adi x m

miti

DISCARD

UPLOAD

Figura 43

este

responsabilă de modificarea sursei elementului „img” și modifica valoarea atributului „selectedFile” cu noul fișier selectat.

- Câmpul pentru descriere este reprezentat de un element „textarea” ce reprezinta un câmp de tip input cu o inalțime marita.

- Utilizatorul are posibilitatea de a eticheta persoane folosind câmpul indicat de labelul „TAG PEOPLE”. Acest câmp este o combinatie intre un element de tip Autocomplete și un element de tip MatChipList. Autocomplete-ul este responsabil de filtrarea și aducerea sugestiilor, reprezentate de alți utilizatori, pe măsură ce în câmpul input se tasteaza. Filtrarea datelor se face cu ajutorul funcției din figura 44, astfel; se va reinitializa de fiecare dată lista „options” cu

```

async onSearchChange(){
  if(this.searchControl.value.length>0)
    await this.profileService.search(this.searchControl.value)
    .then(results=>{
      this.options=results as unknown as Profile[]
      this.options.forEach(profile=>{
        })
    })
  else this.options=[]
}

```

Figura 44

răspunsul request-ului creat prin funcția „search” din serviciul ProfileService. În momentul în care se selectează utilizatorii ce se doresc a fi etichetați se va adauga în lista „chipBoxUsers” valoarea selecției. Parcurgând aceasta listă în

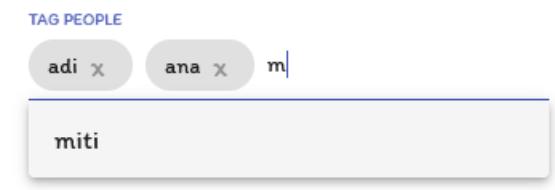


Figura 45

elementul „mat-chip” se pot observa persoane etichetate, după cum ilustreaza figura 45:

- Utilizatorul are posibilitatea de a șterge persoanele etichetate folosind semnul „X” de pe etichetă, vizibil și în figura 45, care va apela funcția de „remove”, responsabilă de ștergerea etichetei din lista „chipBoxUsers”.
- Utilizatorul poate renunța la crearea postării, apasând butonul "DISCARD" ce va inchide fereastra.

- Salvarea postării se face cu ajutorul butonului „UPLOAD” ce va apela funcția din figura 46 ce va trimite catre funcția „createPost”, postarea nou creată și fișierul selectat.

```
onUpload() {
  this.post.tags=this.selectedUsers
  this.postService.createPost(this.post, this.selectedFile)
    .subscribe((response) => {
      alert(response)
      this.closeDialog()
    }
  );
}
```

Figura 46

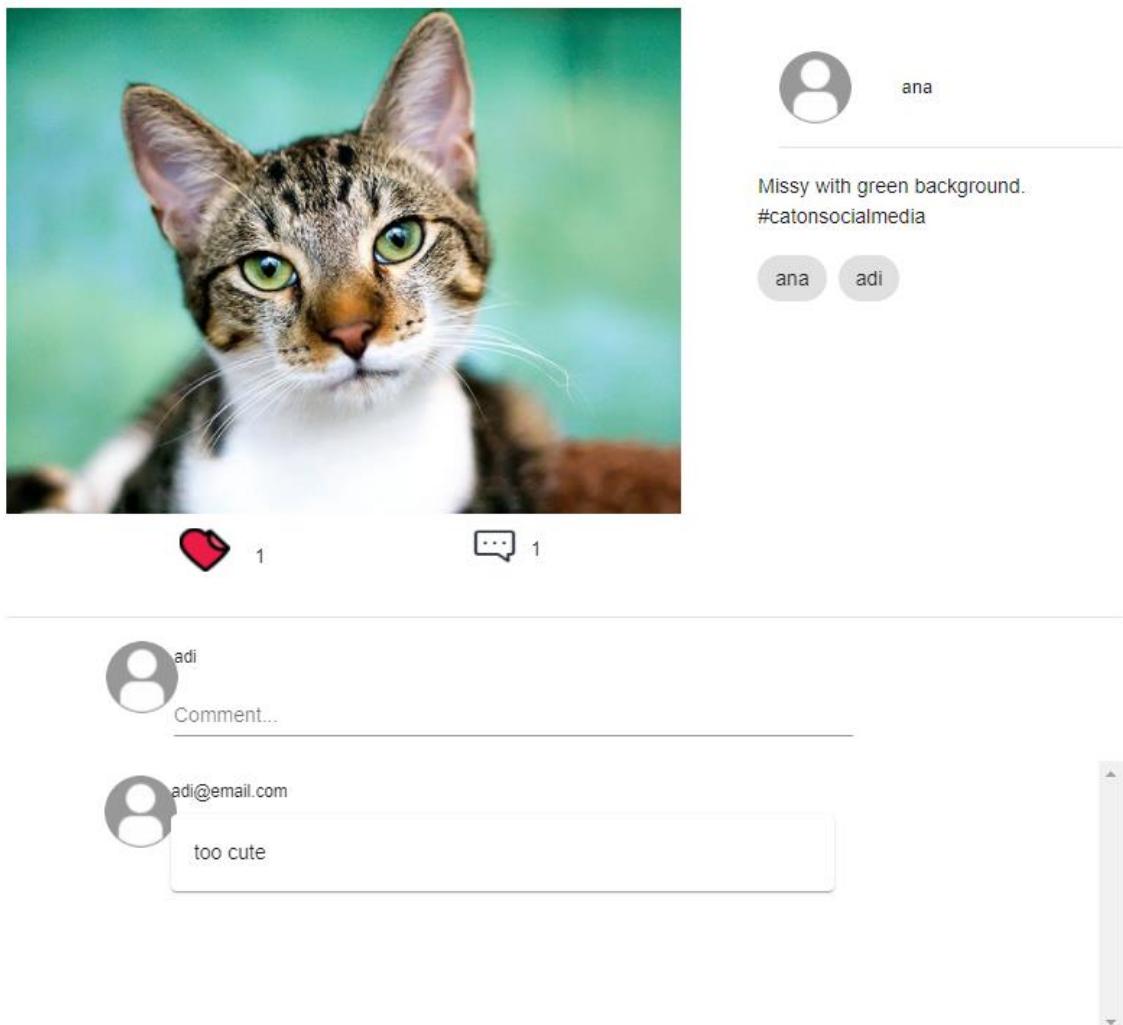


Figura 47

6.1.5. Componența Post-Pages

Cu această componentă utilizatorul poate observa toate detaliile legate de o postare, după cum ilustrează și figura 47. În aceasta componentă sunt prezentate urmatoarele elemente:

- Poza și nickname-ul utilizatorului care a creat postarea, apăsând pe oricare dintre cele două se va naviga spre profilul acestui utilizator.
- Descrierea postării și persoanele etichetate
- Poza încarcată postării
- Numărul de aprecieri și comentarii, iar iconița în forma de inimă este colorată după ce se verifică dacă această postare este apreciată sau nu de utilizatorul logat.
- Un câmp unde utilizatorul poate adăuga comentarii postării. Comentariile sunt trimise către server, care va notifica creatorul postării în legătura cu aceasta acțiune, și se va incrementa numărul de notificări nedeschise. De asemenea, comentariile apar în timp real, astfel utilizatorii pot să vada comentariile proaspăt adăugate și să continue șirul de comentarii.
- Lista de comentarii ce au fost adăugate

6.1.6. Componenta Chat-Pages

Această componentă este alcătuită din componente ActiveChatsComponent și CurrentChatComponent și împreună formează vizual pagina ilustrată în figura 48

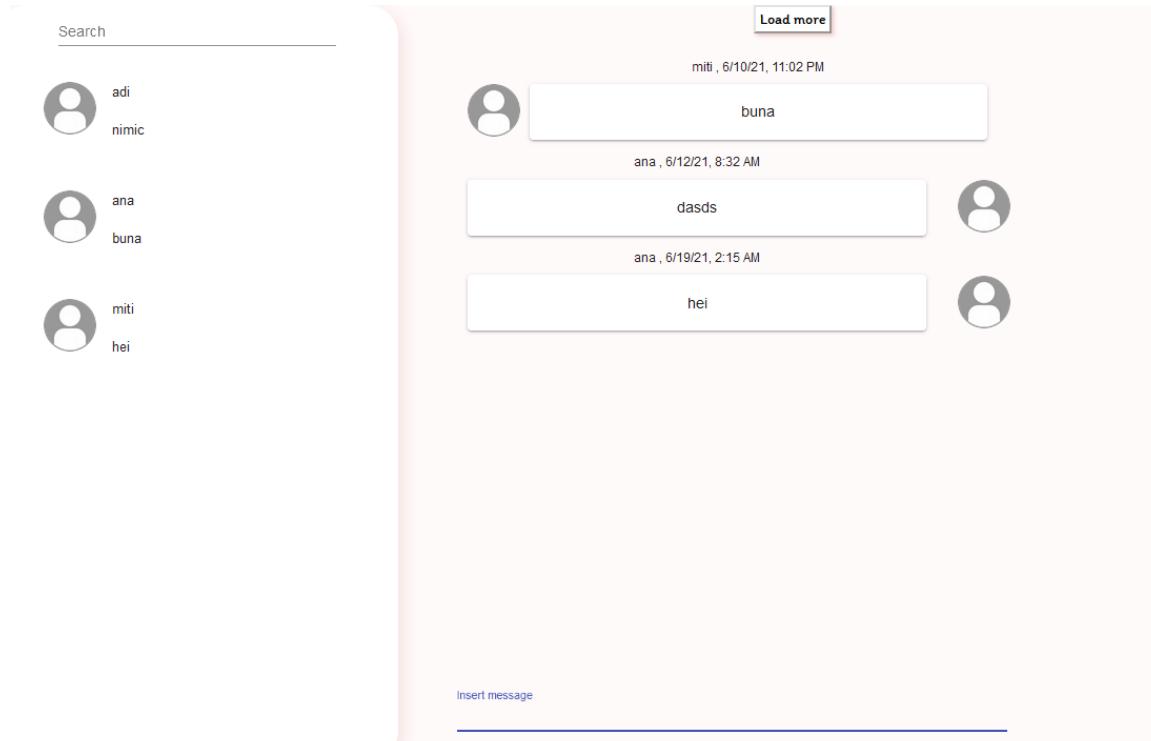


Figura 48

6.1.6.1. ActiveChatsComponent

Această componentă reprezintă partea din dreapta din figură, și conține un câmp de căutarea ce implementează o componentă Autocomplete și o listă de obiecte de tip DisplayChats.

- Autocomplete-ul este responsabil de căutarea tuturor utilizatorilor a căror nickname sau numele complet corespund valorii adăugate. În momentul

```
createChat(profile: Profile, message: ChatMessage | undefined) {
  this.inputValue = ""
  this.chatService.createChat({ id: 0, user1: this.profileService.getPersonalProfile(), user2: profile })
  .subscribe((data) => {
    this.openChat(data, message)
    this.chatService.checkChat(data).then(result => {
      if (!result && this.chats.filter(chat => chat.chat.id == data.id).length == 0)
        this.chats.push({ chat: data, message: { id: 0, message: "No messages.", sender: this.profileService.getPersonalProfile(),
          this.chatService.convertChat(data)
          this.chats = this.chats.sort(ActiveChatsComponent ascendingByLastMessage)
        })
    })
  })
}
```

Figura 49

selecției unei sugestii din lista generată de componentă, se apelează funcția „createChat()”, ilustrată în figura 49, în aceasta funcție se verifică dacă există deja creată o conversație între cei doi utilizatori și crează o conversație după un şablon dacă nu există. Se sortează conversațiile astfel încât conversația selectată să fie prima în lista.

- În momentul selectării unei conversații, fie ea din lista deja existentă sau în urma selectării unei sugestii din Autocomplete se apelează funcția „openChat” din figura 50, ce este responsabilă de trimiterea către componentă parinte, „CurrentChatComponent” a informațiilor despre conversația selectată. Se va crea o copie a conversației selectate folosind succesiunea de funcții „JSON.parse(JSON.stringify(chat))”, astfel se va evita modificarea nedorită a conversației selectate.
- Folosind serviciul WebSocketService, s-a realizat abonarea la endpoint-ul responsabil de notificarea utilizatorului în momentul în care se trimit către acesta un nou mesaj și se vor resorta conversațiile în ordine descrescătoare, după câmpul „createdOn” al ultimului mesaj trimis.
- Toate conversațiile a carui ultim mesaj este nedeschis de utilizatorul căruia îi este destinat sunt evidențiate fată de restul. În momentul deschiderii acestora se va trimite către server ultimul mesaj din conversație, modificat cu atributul

```
openChat(chat: Chat, message: ChatMessage | undefined) {
  this.open.emit(JSON.parse(JSON.stringify(chat)))
  if (message)
    if (message.sender.user.id != this.authenticationService.getCurrentUser().id)
      this.chatService.openChat(chat).then(() => {
        message.state = true
      }).catch(
      )
}
```

Figura 50

state setat pe „true”, astfel în baza de date se va actualiza mesajul și se va considera în continuare ca fiind citit.

- În momentul afișării componentei, aceasta va solicita server-ului afișarea unui număr finit de conversații. În cazul în care utilizatorul dorește să parcurgă din

ordine descendantă conversațiile, acesta poate opta să încarce mai multe conversații, folosind butonul „Load more”.

6.2. Pachetul Model

În acest pachet sunt create interfețe folosite pentru transferul de date cu backend-ul. Aceste interfețe sunt identice entităților create în backend deoarece în momentul transmiterii de obiecte către endpoint, corpul request-ului trebuie să conțină obiecte ce pot fi deserializate instant în tipul cerut în parametrii. Astfel, dacă în backend am o entitate definită de un id de tip Long și o denumire de tip String, interfață declarată în pachetul Model corespunzătoare acestei entități trebuie să conțină de asemenea aceleasi atribute de același tip. De asemenea acest pachet conține și interfețe ce au scop de a ajuta în prezentarea într-o manieră mai ușoară a anumitor modele, cu cât mai multe detalii. Situație întâlnită la interfață NewsFeedPost ce conține un atribut de tip Post, dar și numărul de aprecieri, numărul de comentarii și verificarea dacă postarea este apreciată, sau nu, de utilizatorul logat.

Acest pachet conține fișierele chat.ts, comment.ts, displaychat.ts, like.ts, message.ts, newsfeedpost.ts, notification.ts, post.ts, profile.ts, role.ts, story.ts, suggestion.ts, user.ts. Interfețele sunt ilustrate și în figura 51:

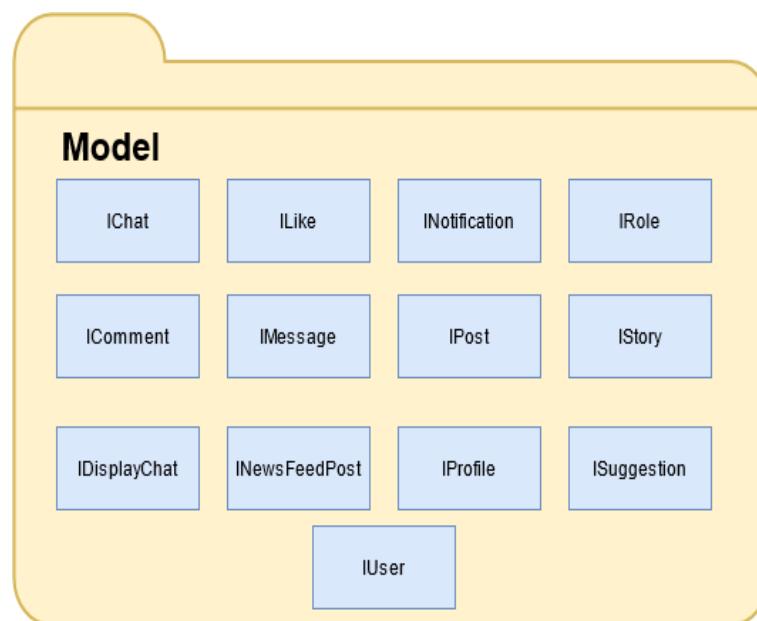


Figura 51

6.3. Pachetul Service

Acest pachet conține toate componentele concepute pentru a apela endpoint-urile din backend. Rolul majorității serviciilor este de a apela endpoint-urile de la componente back-end pentru a citi, adăuga, edita și șterge date.

Printre servicii, se remarcă serviciile din pachetul auth-service ce se ocupă de partea de autentificare și securitate a interfeței. Serviciul de autentificare "AuthenticationService" verifică adresă de e-mail și parolă utilizatorului pe pagină de autentificare. În cadrul funcției "login" din acest serviciu se face verificarea credentialelor, dacă acestea sunt corecte, va crea o nouă sesiune în care va fi sălvat simbolul JWT pentru a verifica apelul către backend, e-mail, utilizator de notificare ID și rol și se va atașa în localStorage o nouă valoare pentru token-ul de acces.

Pentru că backendul să verifice faptul că utilizatorul este autentificat cu succes în aplicație, se vor trimite headerele necesare autorizării utilizatorului, folosindu-se serviciul de HttpInterceptor, ce implementează interfață cu același nume. În cadrul funcției "intercept", prezență și în figura 52, se va atașa request-ului interceptat un nou header de autorizare ce va conține token-ul de logare. Acest header este verificat în backend, iar pe baza rezultatului, aplicația va permite finalizarea request-ului la endpoint-ul trimis.

```
intercept(req: HttpRequest<any>, next: HttpHandler) {
  let token = this.token.getToken()
  req = req.clone({
    headers: req.headers.set('Authorization', 'Bearer ' + token)
  })
  return next.handle(req);
}
```

Figura 52

Serviciul AuthGuard este responsabil de redirecționarea la componentă de "LoginComponent" în momentul în care utilizatorul nu este logat. Această clasă implementează funcția canActivate din interfață "CanActivate", ilustrată în figura 53: , ce verifică token-ul sălvat în TokenService, iar dacă acesta nu este gol atunci se permite

accesarea url-ului. Dacă rezultatul este opus, atunci se va redirectiona utilizatorul pe pagină de logare.

Această clasă este atașată rutelor dorite din AppRoutingModule pentru a restricționa accesul persoanelor neautorizate. Sintaxa este următoarea:
“{ path: 'home', component: HomeComponent, canActivate: [AuthGuard] }”.

```
canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
  if (this.tokenService.getToken() != "") {
    return true;
  } else {
    this.router.navigate(['/login'], { queryParams: { returnUrl: state.url } });
    return false;
}
```

Figura 53

Un alt serviciu important din acest pachet este WebSocketService. Cu acest serviciu se stabilește conexiunea la url-ul de web socketing ce se conectează la API-ul create folosind atât SockJs cât și STOMP.js. Websocket este, de asemenea, un protocol de comunicare care permite implementarea de canale de comunicare bidirectionale între un server și client. Odată, conexiunea WebSocket este instantiată între server și client, ambele pot face schimb de informații la nesfârșit până când conexiunea este închisă de oricine din cauza acestui WebSocket este preferat față de HTTP unde clientul și serverul schimbă informații la frecvență ridicată și cu latență scăzută deoarece conexiunea HTTP este închisă odată ce o cerere este servită de server și pentru a deschide din nou o conexiune HTTP, există o constrângere de timp. Websocket este bidirectional, deoarece clienții se pot abona la un eveniment, iar serverele pot publică evenimente.

Pentru conectarea la API este folosită funcția „connect()”, ilustrată în figura 54, care va schimba Subiectul comportamentului de la ATTEMPTING la CONNECTED.

```
private connect(): Observable<Client> {
  return new Observable<Client>(observer => {
    this.state.pipe(filter(state => state === SocketClientState.CONNECTED))
      .subscribe(() => {
        observer.next(this.stompClient);
      });
}
```

Figura 54

Folosind funcția "subscribeToNotifications()" se permite abonarea la mai multe endpoint-uri ale WebSocket-ului, folosind aceeași conexiune la API. Funcția de "disconnect()" este responsabilă de încheierea conexiunii clientului Stomp cu backend-ul.

Tot în acest pachet se găsește și serviciul RestService, a cărui scop este a facilita trimiterea de request-uri la endpoint-uri din aplicația client. Astfel am creat funcții standard

```
public post(url: string, body: any): Observable<any> {
  return this.http.post("http://localhost:8080/" + url, body).pipe(
    catchError((x => this.handleError(x)))
  );
}
```

Figura 55

precum cea din figura 55, ce este apelată doar cu un sir de caractere, ce reprezintă url-ul apelat, și corpul request-ului.

6.4. Pachetul Pipe

Acest pachet conține clase ce implementează interfața PipeTransform. Aceste clase au o structură simplă, ele implementând doar funcția transform, și sunt responsabile cu transformarea și/sau formatarea datelor din template-urile HTML, acestea primind o valoare input și returnând o valoare output ce reprezintă valoarea primită transformată. Acest tip de clase sunt utile, deoarece se pot utiliza în întreaga aplicație, în timp ele sunt declarate o singură dată. Pentru a forma sursă imaginilor primite, ce reprezintă imaginea codificată în base64, dar care nu conține prefixul necesar decodificării, am folosit un pipe personalizat ce atașează sursei imaginii prefixul „data:image/png;base64,” și este folosit direct în fișierul HTML astfel: “profile.photo|image”.

7. Structura bazei de date

Pentru a stoca un număr foarte mare de informație și pentru a asigura persistarea acesteia am optat pentru folosirea unei baze de date relaționale, și anume MySql. Un avantaj vital al utilizării unui astfel de sistem de gestionare a bazei de date este faptul că relațiile definite între entități sunt ușor de accesat cu ajutorul acțiunilor de combinare prin chei strâine a tabelelor. Această particularitate specifică bazelor de date relaționale are că rezultat

rapiditate în căutare și accesarea ușoară a informației dorite, adăugarea, modificarea și ștergea datelor nu necesită mult cod de scris.

O baza de date normalizată reprezintă o baza de date căreia însă au aplicat procese de normalizare, ce au rezultat în structurarea cât mai logică a tabelelor minimizarea anomalilor la adăugare, ștergere și modificare și minimizarea datelor redundante. Baza de date creată este în faza a treia a normalizării deoarece îndeplinește primele două forme ale normalizării, această conținând numai valori atomice ce nu mai pot fi decompuse și sunt dependente de cheia primară, iar dependințele funcționale tranzitive au fost înlocuite. Pentru proiectul de față am ales crearea următoarelor tabele principale:

User – cuprinde informațiile necesare la logare ale utilizatorului:

- id - tip de date:integer; reprezintă cheia primară
- nickname - tip de date:varchar(255); reprezintă porecla care va fi afisată în aplicație pentru utilizator
- username - tip de date:varchar(255); reprezintă username-ul
- password - tip de date:varchar(255); reprezintă parola încriptată
- role_id - tip de date:bigint; reprezintă cheia străină spre tabela Role

Role – cuprinde informațiile necesare la logare ale utilizatorului:

- id - tip de date:integer; reprezintă cheia primară
- name - tip de date:varchar(255); reprezintă denumirea rolului

Profile – cuprinde informațiile despre utilizator:

- id - tip de date:integer; reprezintă cheia primară
- display_name: tip de date:varchar(255); reprezintă numele complet
- description: tip de date:varchar(255); reprezintă descrierea profilului
- photo: tip de date:longblob; reprezintă fotografia de profil
- user_id: tip de date:bigint; reprezintă cheia străină spre tabela User

Chat- cuprinde informațiile despre o conversație creată între doi utilizatori:

- id - tip de date:integer; reprezintă cheia primară

- user1_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul care a inițiat pentru prima oară o conversație între cei doi utilizatori.
- user2_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul cu care user1 a ales să discute.

Chat_message- cuprinde informațiile despre un mesaj trimis în cadrul unui chat:

- id - tip de date:integer; reprezintă cheia primară
- sender_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul care crează un nou mesaj.
- created_on - tip de date:datetime; reprezintă momentul în care a fost creat mesajul.
- message - tip de date:vachar(255); reprezintă conținutul mesajului.
- state - tip de date:bit(1); reprezintă statusul mesajului trimis, această este setată inițial cu valoarea 0, adică false, iar în momentul deschiderii mesajului această valoare va deveni 1.
- chat_id: tip de date:bigint; cheia străină spre tabela Chat și reprezintă conversația în cadrul căreia utilizatorul a trimis mesajul.

Comment- cuprinde informațiile despre un comentariu adăugat unei postări:

- id - tip de date:integer; reprezintă cheia primară
- user_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul care crează un nou comentariu.
- created_on - tip de date:datetime; reprezintă momentul în care a fost creat mesajul.
- message - tip de date:vachar(255); reprezintă conținutul mesajului.

Like- cuprinde informațiile despre o nouă apreciere adăugată unei postări:

- id - tip de date:integer; reprezintă cheia primară
- user_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul care crează o nouă apreciere.

Tag- cuprinde informațiile despre o nouă eticheta adăugată unei postări:

- id - tip de date:integer; reprezintă cheia primară

- user_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul care este etichetat.

Follow - cuprinde informațiile despre o urmărire:

- id - tip de date:integer; reprezintă cheia primară
- followed_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul începe să fie urmărit.
- follower_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul ce a ales să-l urmărească pe utilizatorul reprezentat de „followed_id”.

Notification- cuprinde informațiile despre o notificare nou creată și destinată unui utilizator:

- id - tip de date:integer; reprezintă cheia primară
- created_on - tip de date:datetime; reprezintă momentul în care a fost creată notificarea.
- sender_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul a cărui acțiune a inițial creare unei noi insertii în această tabelă.
- message - tip de date:vachar(255); reprezintă conținutul mesajului notificării.
- state - tip de date:bit(1); reprezintă statusul notificării trimise, această este setată inițial cu valoarea 0, adică false, iar în momentul deschiderii notificării această valoare va deveni 1.
- receiver_id: tip de date:bigint; cheia străină spre tabela User și reprezintă utilizatorul căruia îi este destinată notificarea.

Post – cuprinde informațiile despre o postare:

- id - tip de date:integer; reprezintă cheia primară
- created_on - tip de date:datetime; reprezintă momentul în care a fost creată postarea.
- description: tip de date:varchar(255); reprezintă descrierea postării
- photo: tip de date:longblob; reprezintă fotografia adăugată
- user_id: tip de date:bigint; reprezintă cheia străină spre tabela User

Story – cuprinde informațiile despre o nouă poveste adăugată:

- id - tip de date:integer; reprezintă cheia primară
- created_on - tip de date:datetime; reprezintă momentul în care a fost creată povestea.
- photo: tip de date:longblob; reprezintă fotografie adăugată.
- user_id: tip de date:bigint; reprezintă cheia străină spre tabela User.

8. Concluzii

Acest capitol cuprinde concluziile finale la care am ajuns în urma implementării platformei de social media propuse.

8.1. Obiective propuse

Am dorit implementarea unei platforme sociale ce să conțină cel puțin funcționalitățile de bază astfel încât un utilizator să poată reuși să creze conexiuni cu persoanele căutate, iar acest lucru a fost făcut posibil. De asemenea s-au implementat și funcționalități mai ample ce îi permit utilizatorului să:

- Își poate crea cont
- Poate edita profilul personal
- Poate adauga, edita și șterge postări
- Poate adauga și șterge aprecieri
- Poate adauga și șterge comentarii
- Poate vizualiza postările persoanelor urmărite în ordine descendentă
- Trimită mesaje
- Fie notificat cand i se apreciază și/sau comentează o postare
- Fie notificat cand incepe să fie urmărit
- Fie adauge story-uri
- Fie fie notificat cand i se trimit mesaje
- Fie vizualize datele personale ale unui profil

8.2. Idei de dezvoltare ulterioare

Platforma poate fi imbunătățită având în vedere că în momentul de față conține funcționalități de bază, astfel pe viitor aceasta ar putea fi dezvoltată astfel încăt să conțină urmatoarele funcționalități:

- Adaugare de multiple poze pentru o postare
- Adaugarea de poze pentru story-uri
- Verificarea e-mailului
- Permiterea raportării postărilor pe baza criteriilor

- Creare funcționalități părții pentru admin
- Evaluarea postărilor
- Sugerarea conexiunilor pe baza de criterii mai complexe
- Încarcarea de videoclipuri

În concluzie platforma reușește să ofere utilizatorului, pe lângă funcționalitățile de bază, posibilitatea de a comunica prin intermediul mesajelor cu cei dragi, de a comenta la postări și de a fi notificat cu privirea la diverse acțiuni. De asemenea vizează și dezvoltarea ulterioară a funcționalităților pentru îmbunătățirea experienței în cadrul acesteia.

Bibliografie

- Alex Antonov, Spring Boot Cookbook:
https://books.google.ro/books?hl=ro&lr=&id=1dpOCwAAQBAJ&oi=fnd&pg=PP1&dq=spring+boot&ots=RpBS0Ckstk&sig=zomt8MlMW3_CbrUX_g6WAb4SKcw&redir_esc=y#v=onepage&q&f=false
- Craig Walls, Spring Boot IN ACTION :
<https://doc.lagout.org/programmation/Spring%20Boot%20in%20Action.pdf>
- Anton Moiseev, Angular Development with TypeScript :
https://books.google.ro/books?hl=ro&lr=&id=1TgzEAAAQBAJ&oi=fnd&pg=PT20&dq=angular+framework+typescript&ots=yfFWTbDJA-&sig=cW9B0030TTVAm_DLxL0bMEo-cJA&redir_esc=y#v=onepage&q=angular%20framework%20typescript&f=false
- R. Shaik, Spring Boot Security + JSON Web Token Tutorial:
<https://dzone.com/articles/spring-boot-security-json-web-tokenjwt-hello-world>
- Steve Suehring, MySql Bible:
<http://justplain.com/eBooks/Databases/MySQL/MySQL%20Bible.pdf>