**1. Introducere**

**1.1 Descrierea aplicatiei**

Platformele de social media permit oamenilor sa se conecteze cu cei dragi si faciliteaza comunicarea intre acestia. Totodata ajuta la cresterea comunicarii si stabilirii conexiunilor cu oamenii din intreaga lume, permit creeare si schimbul de informatii si idei si formeaza comunitatii virtuale de oameni.

In vederea realizarii lucrarii de licenta am optat pentru creare unei aplicatii similara celor de social media unde utilizatorii

**2. Obiectivele Proiectului**

2.1. Obiectivul principal

Proiectul are ca scop dezvoltarea unei platforme social media cu ajutorul creia se va facilita comunicarea si imparasirea de informatii intre utilizator. Aceasta platform va fi folosita de catre utilizatori, carora la momentul inregistrarii li se vor atribui profile cu care se vor putea face identificati in momentul cautarii de catre ceilalti utilizatori.

2.2. Obiective secundare

Ca și obiective secundare, sistemul dorește să ofere posibilitatea de a impartasi opiniile intr-o maniera rapida si facila:

* Exprimarea parerilor fata de o postare prin apreciere si prin comentariu

Utilizatorul isi poate exprima aprecierea fata de postarea unui alt utilizator folosind un buton destinat special acestei actiuni, buton sub forma de inima, si isi poate retrage aprecierea in cazul in care acesta considera ca nu este cazul sau s-a produs o greseala. De asemenea un utilizator poate lasa comentarii la o postare.

• Notificarea in momentul in care se produce actiune ce il vizeaza pe

utilizatorul logat

Daca utilizatorul logat va primi o apreciere, i se va comenta la o postare, va incepe sa fie urmarit de un alt utilizator sau primeste un mesaj, atunci platforma ii va trimite automat o notificare pentru a-l instiinta, nefiind necesar un refresh la pagina daca. Ideea din spatele acestei functionalitati este de a putea raspunde cursiv si instant la actiunile ce il privesc pe acesta.

• Cautare altor utilizatori dupa username sau numele complet

Aceasta functionalitate urmareste sa permita utilizatorilor posibilitatea de a cauta alti utilizatori in mod dinamic, in functie de username-ul acestora sau dupa numele complet. Un camp de cautare va trimite request-uri constant cu modificarile aduse in camp pentru a adduce din baza de date cele mai bune variante in urma cautarii.

• Adaugarea de informatii personale in profil

Utilizatorul logat are posibilitatea de a-si imbogati profilul personal prin adaugare unei poze personale, a unui nume complet personalizat si a unei descrieri prin care acesta doreste sa fie remarcat.

De asemenea in detaliile utilizatorului vor aparea informatii cu privire la numarul de personae urmarite, numarul de personae ce urmaresc utilizatorul si la numarul de postari adaugate de acesta.

• Crearea de postari

Un utilizator isi poate exprima gandurile prin crearea unei postari personalizate ce va contine neaparat o poza, dar este posibila si adaugarea unei descrieri corespunzatoare postarii cat si crearea de etichete pentru userii existenti in platforma, aceste etichete au ca scop divertismentul cat si o impartasire mai usoara a posibilelor conexiuni cu alte profile in cazul ca acestea nu exista

• Urmarirea utilizatorilor ce nu se afla deja in lista de urmariti ai

utilizatorului logat

Utilizatorul are posibilitatea de a-si reiinoi lista de urmariti folosindu-se de o lista speciat filtrate unde el poate vizualiza si adauga la urmariri utilizatorii pe care acesta nu ii urmareste deja.

• Vizualizarea detaliilor unei postari

Aceasta functionalitate doreste crearea unei ferestre speciale unde utilizatorul poate vizualiza in de-aproape o postare cu toate detaliile acesteia precum: cine a create-o, cine este etichetat in ea, descrierea acesteia, cate aprecieri si comentarii are, cat si comentariile existente la postare.

• Vizualizarea detaliilor unui profil

Utilizatorul logat va putea sa vizualizeze detaliile profilului sau: poza de profil, username-ul, numele complet, descrierea, numarul de postari, numarul de personae urmarite si numarul de urmaritori cat postarile create de acesta si postarile create de-a lungul timpului.

• Posibilitatea de a modifica sau sterge o postare

Daca cel care creaza o postare va dori sa modifice persoanele etichetate si descrierea postarii atunci acesta va avea aceasta posibilitate. Poza postarii nu va putea fi modificata, postarea in sine fiind definite de poza creata. De asemena utilizatorul are posibilitatea de a-si sterge postarea.

• Posibilitatea de a sterge un comentariu

Utilizatorul logat va avea posibilitatea de a-si sterge propriile comentarii lasate la postari cat si comentariile lasate de ceilalti utilizatori la propriile postari.

3. Tehnologii folosite

Pentru realizarea lucrarii am optat pentru back-end in Spring Boot configurat pentru Java 11, iar pentru front-end am ales sa folosesc Angular.

**3.1 Java 11**

Java este un limbaj de programare obiect-orientat, cu un scop general si destinat lucrului cu clase, special creat pentru a avea cat mai putine dependinte la implementare. Acest limbaj este destinat dezvoltarii de aplicatii, prin urmare, este rapid, sigur si de incredere, fiind astfel utilizat la scara larga pentru dezvoltarea de diverse aplicatii dintr-o gama larga de proiecte cu diferite cerinte.

Java a fost dezvoltat de Sun Microsystems în anul 1995. James Gosling este cunoscut ca tatăl Java. Înainte de Java, numele său era Oak. Deoarece Oak era deja o companie înregistrată, așa că James Gosling și echipa sa 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; atâta timp cât computerul are instalat un mediu Java Runtime Environment (JRE), poate rula un program Java. Prin urmare este foarte usor de creat un mediu de dezvoltare in care s epoate utiliza acest limbaj de programare.

Java este utilizat in diferite domenii precum:

1. Crearea de aplicatii 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.

2. Createa de aplicatii 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 sarcini 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 dezvolta 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.

**3.2 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 utilizarii Spring Boot enumera dezvoltarea aplicatiilor usor de inteles si de dezvoltat, modularitatea codului si reutilizara acestuia, productivitatea sporita si reducerea timpului efectiv de dezvoltare.

Motivarea alegerii acestei tehnologii a constat in faptul ca aceasta permite configurarea usoara a proiectului si a dependintelor necesare, conectarea usoara la baza de date, existenta anotarilor ce permit manipularea usoara a codului.

* 1. **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 dependențelor 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.

* 1. **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:

1) Open Source și ușor

Cadrul Hibernate este open source sub licență LGPL și ușor.

2) 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.

3) Interogare independentă a bazei de date

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

4) 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.

5) Simplifică Complex Join

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

6) Oferă statisticile interogărilor și starea bazei de date

Hibernate acceptă Query cache și oferă statistici despre interogare 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.

* 1. **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 si in conform cu standardul ANSI SQL

* 1. **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 sale 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.

* 1. **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 opționale ca un set de biblioteci TypeScript pe care le importați în aplicațiile dvs.

1. **Arhitectura componentei backend**

Diagram

Description automatically generatedIn vederea devzoltarii componentei backend a aplicatiei am optat pentru alegerea tehnologiei Spring Boot, codul fiind scris in limbajul de programare Java 11. Dezvoltarea acestei componente a urmat sablonul pus la dispozitie de arhitectura pe trei nivele cunoscuta si sub denumirea de Three Layers. Aceasta arhitectura presupune separarea in trei nivele generale a aplicatiei, dupa cum este ilustrat si in figura:

* Nivelul UI sau de prezentare se ocupa de gestionarea request-urilor facute de utilizator pe interfata si trimiterea unui raspuns browser-ului. Gestionarea acestor requesturi se face de regula in clasele de tip Controller, unde sunt definite funcii speciale ce reprezinta endpoint-uri carora li se vor trimite informatii de pe frontend. Acest nivel este specializat pe gasirea celei mai bune functii unde se va trece la urmatorul nivel.
* Nivelul de Business Logic se ocupa de partea de logica aplicata request-ului creat de utilizator la nivelul UI si aplicarea de diverse operatii asupra acestora. Aceste modificari pot fi reprezentate de algoritmi de sortare, de filtrare, de crearea de obiecte DTO (Data Transfer Object) si se ocupa de crearea legaturii cu urmatorul nivel. Pentru acest nivel, clasele de tip Service sunt cele mai reprezentative.
* Nivelul de Data Access, cunoscut si ca Persistence Layer, executa operatii CRUD asupra obiectelor anterior gestionate in nivelul de Business Logic. Clasele de tip Repository sunt responsabile pentru de aducere a datelor, modificarea, stergerea si inserarea acestora.

Aceste clase usureaza cu mult lucrurl cu bazele de date, ele continand functii ce reprezinta query-uri de tip SQL ce se pot personaliza. In acest proiect am optat pentru interfetele de tip CrudRepository, pentru executarea actiunilor asupra bazei de date, cat si pentru PagingAndSortingRepository pentru aducerea unui numar limitat de insertii din baza de date.

Motivele principale pentru alegerea acestui model de arhitectură sunt scalabilitatea imbunatatita a aplicatiei, astfel se pot dezvolta independent unele de altele cele trei nivele si posibilitatea de a dezvolta rapid aplicatia.

**4.1 Detalierea componentelor**

**Graphical user interface, text, application, chat or text message

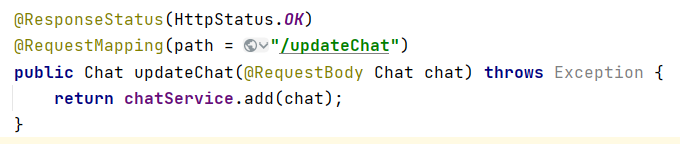
Description automatically generated**

Dezvoltarea independenta a acestor componenta este posibila datorita responsabilitatilor diferite a fiecareia dintre acesteia, responsabilitatile fiind foarte specifice fiecarui tip de nivel, prin urmare:

**4.1.1 Controller**

Clasele de tip Controller se ocupa cu gestionarea fiecarui request primit prin intermediul url-urilor. Aceste clase sunt adnotate cu@RestController, adnotare ce permite crearea de end-point-uri care vor transmite un raspuns automat in momentul in care o functie va da return, nemaifiind nevoie de adnotarea @ResponseBody. Aceste raspunsuri sunt in mod automat serializate sub forma de JSON in HttpResponse.

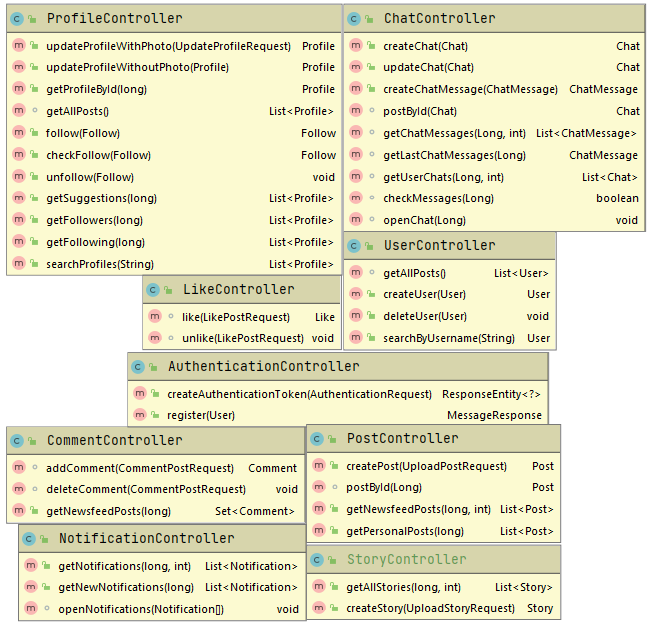
Un exemplu pentru un end-point creat intr-o clasa de tip Controller se afla in urmatoarea figura:

****

* @ResponseStatus este responsabila de trimiterea unui HttpResponse in momentul in care se va executa cu succes modificarea in baza de date a unui obiect existent de tip Chat.
* @RequestMapping are scopul de a crea un endpoint disponibil pentru noi requesturi.
* In parametri functiei se va cere explicit un @RequestBody de tip Chat, astfel functia asteapta ca numai obiecte de tip Chat vor fi trimise pentru a fi gestionate.
* In interiorul functiei se face direct return cu raspunsul oferit de Service-ul instantiat cu denumirea de chatService, in acest caz se va face update la un obiect deja existent din baza de date.

Pentru utilizarea serviciilor direct injectate in interiorul unui Controller acesta va fi decorat cu adnotarea @AllArgsConstructor ce permite utilizarea acestuia fara a crea un constructor in care sa se initializeze toate serviciile necesare. Adnotarea @CorssOrigin marchează metoda sau tipul adnotat ca permițând requesturi de tip Cross Origin.

In pachetul com.licenta.socialmedia.controller am inclus clasele: ProfileController, ChatController, LikeController, UserController, AuthenticationController, PostController, CommentController, NotificationController si StoryController, clase ilustrate si in figura:



**4.1.2 Service**

Clasele de tip Service reprezinta nivelul Business Bogic al aplicatiei, unde sunt aplicate diverse operatii asupra obiectelor primite in Controllere si transmise la acest nivel. Acesta definește funcționalitățile pe furnizate in aplicatie, modul în care acestea sunt accesate și oferă modularitatea codului.

Aceste clase implementeza functii defintite in interfete ce sunt special create pentru aceasta operatie si sunt decorate cu adnotarea @Service pentru a putea fi recunoscute de aplicatie ca servicii. Urmatorul exemplu reprezinta o functie implementata din interfata IPostService in serviciul PostService:

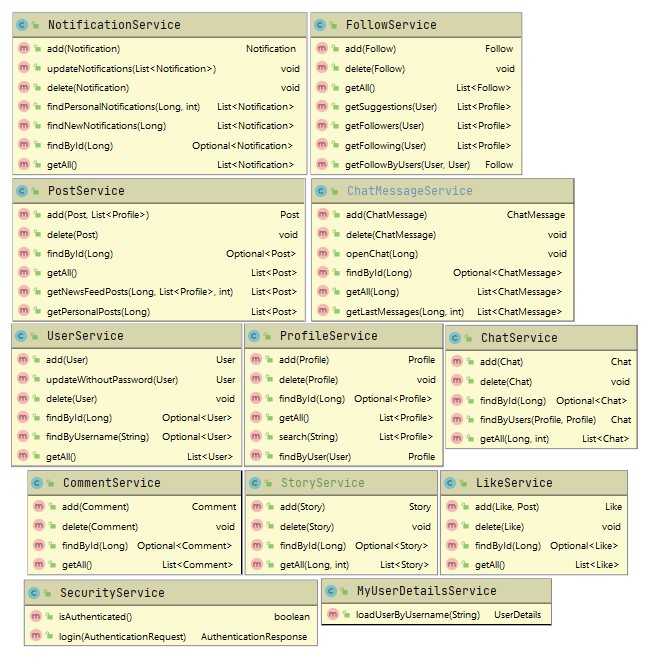


In cadrul functiei putem observa:

* Parametrii asupra carora se vor aplica regulile de business, parametrii ce reprezinta un obiect de tip postare si o lista de obiecte de tip Profile.
* In cadrul functiei se verifica daca exista o postare cu acelasi ID in baza de date, daca identificatorul unic al acestei postari este reprezentat de numarul 0, atunci se va salva obiectul exact cum a fost trimis in urma request-ului. Dupa executarea blocului de cod conditional se va seta atributul photo cu o versiune de biti carora li se aplicat o decompresie. Daca o astfel de postare nu exista, atunci tuturor utilizatorilor din lista de profile li se va trimite o notificare cu noua postare creata. In final se va returna obiectul de tip post, asupra carora s-au aplicat toate operatiile prezentate anterior.
* In cadrul serviciului PostService s-a injectat un camp de tip IpostRepository, folosindu-se adnotarea @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.

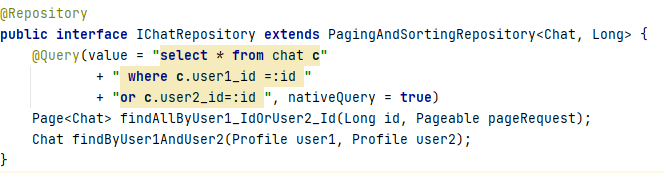
In pachetul com.licenta.socialmedia.service am inclus clasele: NotificationService, FollowService, PostService, ChatMessageService, ChatService, UserService, ProfileService, CommentService, StoryService, LikeService, SecurityService, MyUserDetailsService.



**4.1.3 Repository**

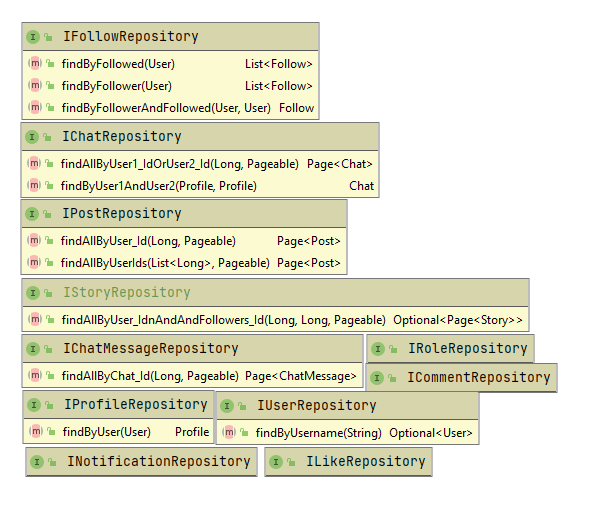
Acest tip de clase reprezinta nivelul de Data Access, cunoscut si ca Persistance Layer. Aceste clase sunt responsabile pentru comunicare in mod direct cu baza de date si crearea de query-uri automate sau personalizate. Toate operatiile aplicabile bazei de date se efectueaza folosind functiile puse la dispozitie de claselele Repository. Pentru a crea un astfel de tip de clasa este necesara implementare functiilor din clasele specializate pentru gestionarea informatiei pastrate in baza de date.

In proiectul de fata am ales sa mostenesc clasele CrudRepository si PagingAndSortingRepository. Urmatorul exemplu ilustreaza o interfata creata:



* @Repository are ca scop instiintarea aplicatiei cu privirea la repository-ul declarat
* Fragmentul „extends PagingAndSortingRepository<Chat,Long> specifica faptul ca acest repository va creea functii ce vor returna liste de obiecte de un anumit numar si pe o pagina specifica.
* Adnotarea @Query permite crearea unui query personalizat de sintaxa SQL, astfel se vor selecta toate obiectele de tip chat din baza de date unde oricare dintre utilizatorii ce sunt inregistrati intr-o conversatie au id-ul specificat in parametrii functiei.
* Tipul returnat de functie este reprezentat de pagina de obiecte Chat, pagina cu atribute specificate de parametrul Pageble pageRequest.
* Denumirea functiei corespune actiunii query-ului personalizat, astfel numai din nume se poate deduce rezultatul returnat.
* Parametrii reprezinta id-ul dupa care se face cautare, si un obiect de tip Pagable care specifica numarul paginii ce va fi rezultata si numarul de obiecte de afisat. Astfel daca pageRequest = PageRequest.*of*( 0 , 5) atunci se vor afisa primele 5 elemente, iar daca pageRequest = PageRequest.*of*( 1 , 5) atunci se vor afisa urmatoarele 5 elemente dupa ce se va sari peste primele 5

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



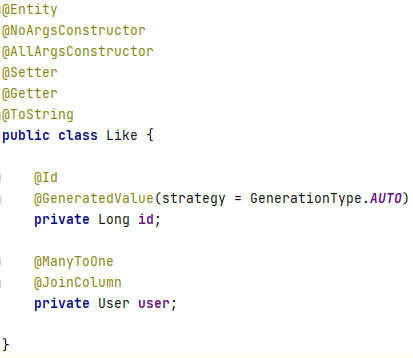
**4.1.4 Model**

Aceste clase sunt create pentru generarea in baza de date a entitatilor. O entitate reprezinta un tabel din baza de date, iar fiecare obiect reprezinta o linie din tabela. Atributele din aceste clase reprezinta la randul lor coloanele tabelelor.

Pentru crearea tabelei este necesara decorarea clasei cu anotatea @Entity. Aceasta permite maparea cu entitatea din baza de date. Aceasta adnotare va cere in mod explicit ca entitatea sa contina un identificator unic declarat cu adnotarea @Id, careia i se va asigura auto-incrementearea acestei chei primare folosind @GeneratedValue(strategy = GenerationType.***AUTO***).

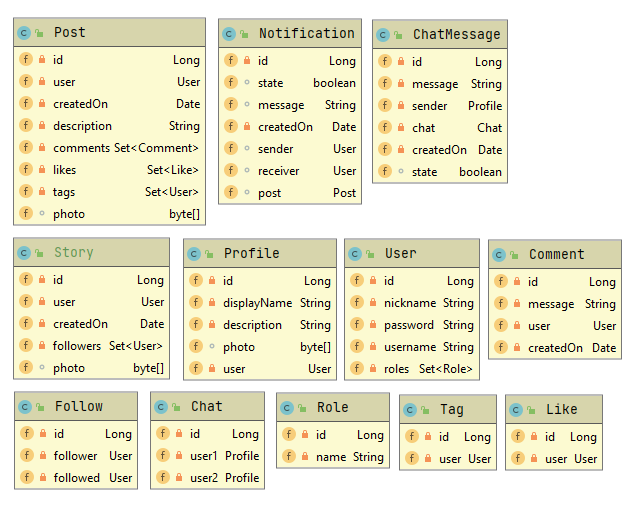
Relatiile de agregare cu alte entitati se vor declara folosind adnotari ce vor define relatiile stabilite. Exemple de relatii sunt @ManyToOne, @OneToOne si @OneToMany.

Un exemplu de astfel de clasa este ilustrat in figura:



* @Entity este folosit pentru crearea tabelei
* @NoArgsConstructor, @AllArgsConstructor permit instantierea unui obiect de tip Like folosind oricare dintre cele doua tipuri de constructori.
* @Setter si @Getter ajuta la scurtarea timpului de implementare, astfel se autogenereaza functii de Get si Set pentru fiecare atribut.
* Cheia unica este decorata cu adnotarile @Id si @GeneratedValue
* Se stabileste o relatie de tip @ManyToOne cu entitatea User.

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



**4.1.5 Configuration**  
 Pachetul configuration contine clase ce implementeaza functii responsabile cu configurarile necesare aplicatiei.

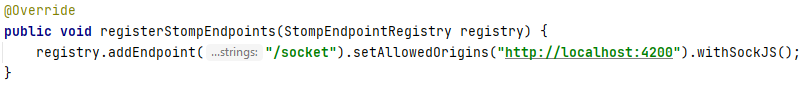
Astfel in acest pachet este creata clasa WebSecurityConfigurer ce este responsabila cu restrictionarea accesului la endpoint-urile create in controllere a utilizatorilor neautorizati, urmatoare functie fiind cea care se ocupa de aceasta actiune:



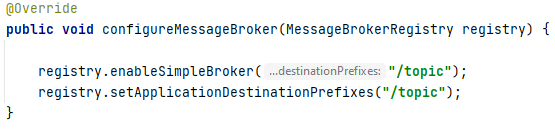
* Variabila http va gestiona sesiunile de logare astfel pentru oricare din url-rile ce contin „/register”, „/”, „/socket/\*\*” si „/auth” se va permite accesul si fara verificarea header-ului requestului. Orice alt request la backend va necesita autentificarea utilizatorului.
* Functia de http.addFilterBefore(authenticationJwtTokenFilter(),UsernamePasswordAuthenticationFilter.**class**) va verifica creditentialele userului autentificat prin decodarea obiectului JWT si cautarea rezultatului in baza de date.

Clasa WebSocketConfigurer este responsabila pentru inregistrarea url-urilor specific Web Socket, astfel ea implementeaza functii care se ocupa cu configurarea aplicatiei. Urmatoarele functii sunt esentiale in setarea unei astfel de conexiuni:

* + 1. Functia registerStompEndpoints

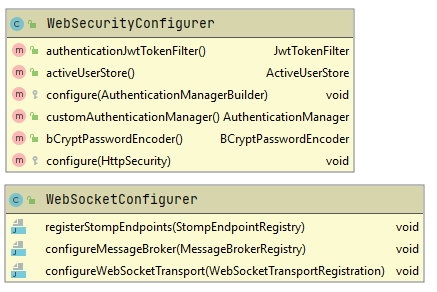


* Aceasta functie este responsabila se inregistrarea prefixului la care se vor face requesturi de catre Web Socket
* Permite requesturi de la url-ul **"http://localhost:4200"**
  + 1. Functia configureMessageBroker



* Inregistreaza prefixul end-point-urilor ce vor transmite informatii

In pachetul com.licenta.socialmedia.configuration am inclus clasele: WebSecurityConfigurer si WebSocketConfigurer.



**4.1.5 Security**

Acest pachet conține clase care gestionează partea de securitate a aplicației.

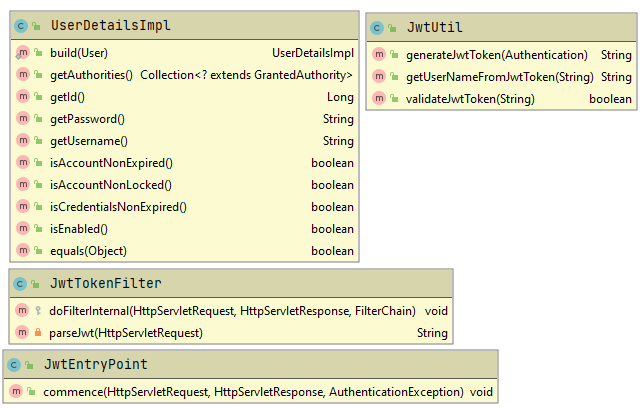
Rolul acestor clase este de a implementa partea de securitate a punctului final prin tokenul JWT.

• JwtUtil este însărcinat cu analizarea token-ului în obiectul User și generarea token-ului din obiectul User.

• JwtEntryPoint este concepută pentru a trimite o 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 arunca o excepție care oprește requestul de la procesare. De asemenea, avem nevoie de o suprascriere pentru autentificarea cu succes, deoarece fluxul Spring implicit ar opri lanțul de filtrare și ar continua cu o redirecționare.

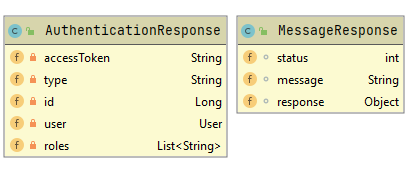
• UserDetailsImpl se ocupa de procesarea token-ului transmis si stabilirea daca exista sau nu in baza de date un utilizator cu creditentialele obtinute in urma decodarii token-ului.



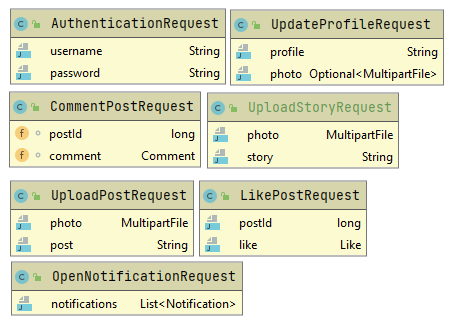
**4.1.6 DTO**

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

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



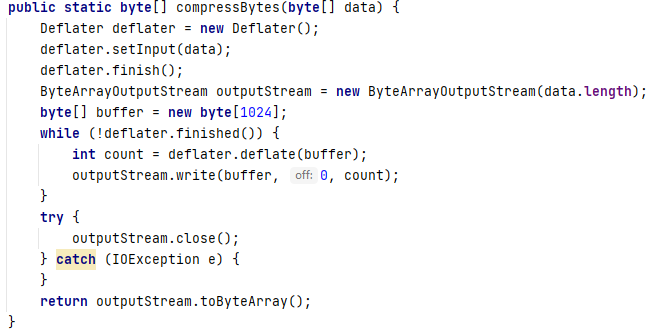
Obiectele DTO de tip Request sunt create in scopul de a contine informatiile a mai multe obiecte, sau informatii ce nu pot fi convertite direct cu @ResponseBody intr-un anumit tip de entitate. Acest pachet contine clasele ilustrate in figura:



**4.1.6 Util**

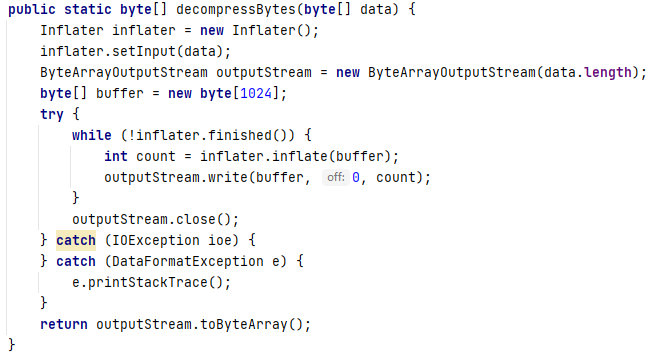
Acest pachet contine clase folosite pentru diverse scopuri precum conversia imaginilor sau crearea de string-uri pentru endpoint-uri.

Clasa PhotoUtils este special creata pentru a facilita lucrul cu imagini in baza de date. La momentul inserarii unei imagini in baza de date, aceasta este convertitata intr-un vector de biti. Pentru a optimiza procesul de adaugare si de aducere a imaginii, conversia bitilor este o solutie optima. Astfel in momentul inserarii se fa apela functia de compressBytes:



* Aceasta functie este responsabila de micsorarea lungimii vectorului fara a pierde din calitatea imaginii

La momentul aducerii imaginii din baza de date, aceasta vine in forma sa modificata, astfel este nevoie o decompresie a imaginii. De aceasta actiune este responsabila functia decompressBytes.



1. **Arhitectura componentei frontend**

Componenta frontend a proiectului a fost dezvoltata cu ajutorul tehnologiei Angular. Paginile sunt create cu ajutorul HTML si CSS, iar functionalitatea logica a fost scrisa in TypeScript.

Angular urmărește arhitectura bazată pe componente. In acest tip de arhitectura se urmareste modularizarea codului si divizarea acestuia pe cat mai multe pachete, cunoscute si sub denumirea de module, pentru a avea usor acces la fiecare componenta si pentru a dezvolta independent unele de altele fiecare componenta. Prin urmare, o aplicație mare este divizată (decuplată) în componente funcționale și logice, cu un cod scris intr-o maniera curata si care sa respecte principiile MVC. Prin urmarirea principiului MVC se doreste crearea de componente cu responsabilitati cat mai specifice si limitate, de scriere de cod specializat pe o serie limitata de actiuni. 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 aplicatie web cu o singura pagina, Componenta AppComponent reprezinta de fapt containerul in care se vor folosi pe rand, in functie de actiunile si alegerile utiliatorului, partile functionale si vizuale ale componentelor definite. Aceasta componenta este de fapt componenta principala a aplicatiei, care se află în fișierul app.component.ts.

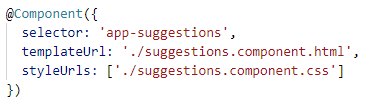
Fiecare componentă, serviciu, directivă, pipe și model utilizate în aplicație trebuie declarate în fișierul app.modules.ts, altfel va fi aruncată o eroare în consola browserului. Fișierul app.routing.modules.ts conține AppRoutingModule care configurează routerul de rutare. Ruta îi spune routerului ce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

Pentru a avea o mai buna evidenta a locatiei componentelor, proiectul este structurat pe pachetele: component, service, pipe, model.

**5.1 Pachetul component**

Acest pachet contine toate interfetele grafice ale aplicatiei, o componenta reprezentand de fac o colectie logica de template-uri HTML, de stiluri asociate si modele care ar putea sa opereze intr-un template folosind imbinarile/legaturile din Angular.

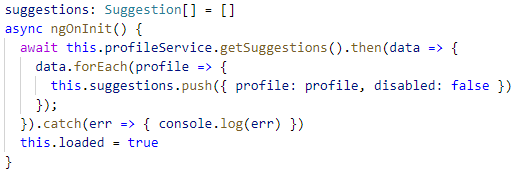
O componentă în Angular este un element de constructie al UI si constă dintr-un fișier de clasă, un template și un fișier CSS pentru stilizarea componentei. Clasa este scrisa in limbaj TypeScript si este decorata cu adnotarea @Component, ca in figura, ce este utilizata pentru a specifica metadata clasei.



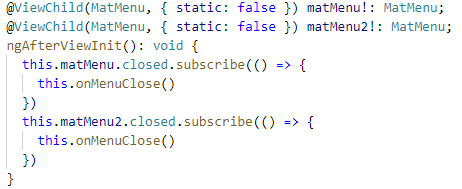
Decoratorul @Component () specifică următoarele informații specifice Angular:

* Un selector care definește modul în care este utilizată componenta într-un alt template. Acest selector are o denumire unica, astfel in momentul in care in interiorul unui fisier HTML se va crea un element, spre exemplu, de tipul „<app-suggestions/>”, aplicatia va stii ca este necesara inlocuirea spatiului alocat cu componenta specificata.
* Un fisier HTML care instruiește Angular cum să redea componenta. Cu acest template se construieste partea vizuala a componentei ce poate contine informatii in tabele, diferite tipuri de elemente de introducere a datelor precum casute de scriere, date-picker-uri, butoane etc. Tot in aceasta componenta se pune in valoare functia de data binding pe care Angular-ul o pune la dispozitie, astfel eliminandu-se mult cod. Sincronizarea elementelor cu variabilele si functiile din clasa componentei se face dinamic.
* Un set opțional de fisiere CSS care definesc aspectul elementelor HTML ale tempalte-ului.

Componentele pot implementa functii ce ajuta la manipularea informatiei si modificarea template-ului precum functia “ngOnInit()”. Aceasta este responsabila de aplicarea de diverse actiuni asupra atributelor utilizate in Data Binding inainte ca fisierul HTML sa construiasca elementele vizuale.

In figura:

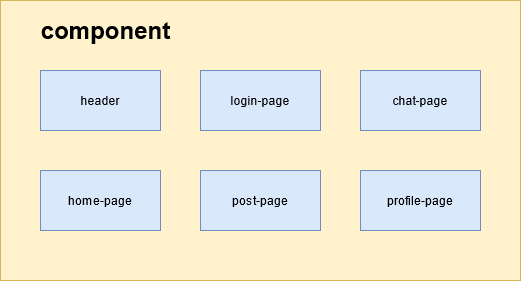
* In momentul initializarii componentei, lista “suggestions” nu contine niciun element. Aceasta fiind setata cu o lista goala.
* In interiorul functiei “ngOnInit()” se apeleaza functia asincrona getSuggestions() din serviciul ProfileService, ce va adduce din baza de date informatiile dorite.
* Cu ajutorul functiei “forEach()” se parcurge fiecare element din rezultatul obtinut si se va adauga in lista “suggestions”
* Functia “ngOnInit()” este declarata ca fiind asincrona deoarece este necesara folosirea cuvantului cheie “await” pentru a astepta executarea functiei asincrone “getProfiles()”, o abordare diferita, in care nu se va folosi perechea “async-await” poate genera probleme grave aplicatiei daca viteza de aducere a datelor nu este una mare.

Functia “ngAfterViewInit()” se ocupa cu modificarea atributelor dupa momentul crearii componentelor UI.

* In interiorul functiei se va stabili comportamentul elementului de UI “MatMenu”, care la momentul inchiderii va apela functia “onMenuClose()” ce va produce modificari asupra componentei.

De asemenea fiecare componenta contine cate 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.

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

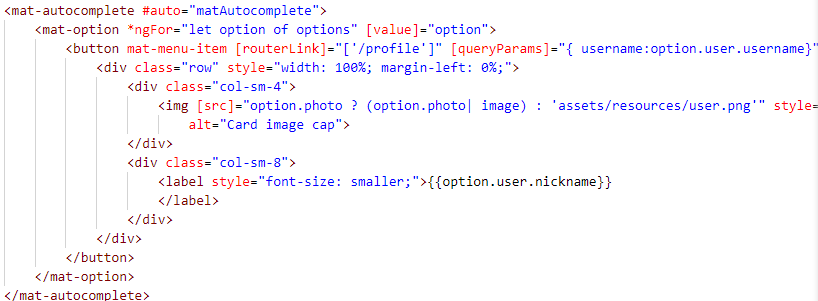


* + 1. **Componenta Header**

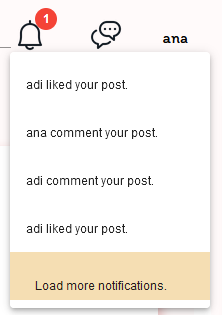
Aceasta componenta contine butoane si elemente necesare peste tot in aplicatie.

Butonul de „home” redirectioneaza aplicatia catre componenta unde se pot vizualiza postarile create de ceilalti useri, sugestiile de urmarire si story-urile. Redirectionarea la componenta Home se face adaugand in template-ul HTML „routerLink="/home”” in declararea elementului img ce continue si locatia imaginii.

Un camp special de cautare permite utilizatorilor sa caute alti ultilizatori ai aplicatiei prin introducerea numelui complet al acestora sau a username-ului cu care sunt inregistrat. Pentru a oferi sugestii pe masura ce noi caractere sunt introduse, am folosit elemental Autocomplete din Angular Material, care pe masura ce se modifica sirul de caractere introdus, va sugera o serie de recomandari ce corespund cautarii. Pentru implementarea acestei functionalitati a fost necesara adaugare unui element HTML de tip input, care in momentul modificarii va apela functia "onSearchChange()". De asemenea, crearea elementului de Autocomplete necesita o structura aparte, dupa cum ilustreaza si figura:



* Secventa “\*ngFor="let option of options"“ este responsabila de parcurgerea tuturor elementelor din lista de obiecte “options”, lista filtrata in momentul tastarii in campul de cautare.
* [value]="option" atribuie valorii selectate obiectul option, astfel se vor retine in intregime toate atributele aflate in option.
* Structura button are ca scop afisarea datelor despre utilizatorul cautat, iar in momentul in care este selectat, apansand click, una dintre sugestii, atunci se va naviga catre componenta “ProfileComponent” unde se va descifra obiectul trimis prin “[queryParams]="{ username:option.user.username}"”

O componenta pentru notificari este de asemenea adaugata in header. Aceasta continue un badge ce, in momentul in care se creaza noi notificari destinate utilizatorului logat, se modifca. Aceasta componenta contine si o lista cu notificari filtrate pe utilizatorul logat si pe data la care acestea au fost create.

Dupa cum este ilustrat in figura, lista de notificari contine un numar limitat de notificari, iar in momentul in care se apasa pe butonul de “Load more notifications.”, se vor aduce urmatoarele notificari, filtrate dupa date crearii.Pentru identificarea noilor notificari, componenta HeaderComponent se aboneaza in constructor la metoda „findNew()” din serviciul NotificationService ce mentine constanta comunicarea cu backendul pentru end-point-ul „/topic/notifications/new/”, careia i se concatenteaza si id-ul utilizatorului logat. Acest endpoint este inregistrat pentru a deservi la conexiunea realizata de serviciul WebSocketService.

Similar cu componenta de notificari este creata si componenta de chat, aceasta fiind notificata de fiecare data cand se creaza un nou mesaj destinate utilizatorului logat. Ultimele mesaje sunt filtrate astfel incat utilizatorul sa vada in ordine descrescatoare datii la care au fost create toate mesajele. Mesajele necitite sunt evidentiate. Iar utilizatorul are posibilitatea de a accesa pagina destinate chat-ului unde si poate selecta mesajele pe care vrea sa le deschida.

 Tot in componenta Header, utilizatorul are posibiliatatea de a-si accesa profilul personal folosing elementele de UI illustrate in figura:

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

Cea din ultima actiune ce se poate executa folosind elementele prezentate in componenta Header este delogarea din sesiune actuala. Butonul de delogare, ilustrat in figura, apeleaza functia

  logout() {

    localStorage.removeItem('currentProfile');

    this.authenticationService.logout()

  }

Aceasta functie este responsabila cu stergerea itemului “currentProfile” din localStorage si resetarea token-ului de autentificare.

* + 1. **Componenta Profile-Page**

Componenta Profile-Page contine subcomponente specializate pentru afisarea anumitor detalii despre profilul accesat. Aceste subcomponente sunt reprezentate de componenta ProfileInformationComponent, ProfilePostsComponent

