Documentatie

Aplicatie Album de Fotografii

folosind Spring Boot si Angular

(Tema 35)

Anca Trandafir, Alexandru Sabou

SDI, febr. 2022

# Descrierea aplicatiei

Aplicatia propusa este o aplicatie web de gestiune a unui album de fotografii cu numeroase functionalitati si design user-friendly. In dezvoltarea acesteia am integrat modelul client-server, protocolul HTTP si principiile REST. Am ales sa folosim ca tehnologii Spring Boot framework pentru partea de server, Angular pentru partea de client si H2 pentru baza de date.

# Functionalitati:

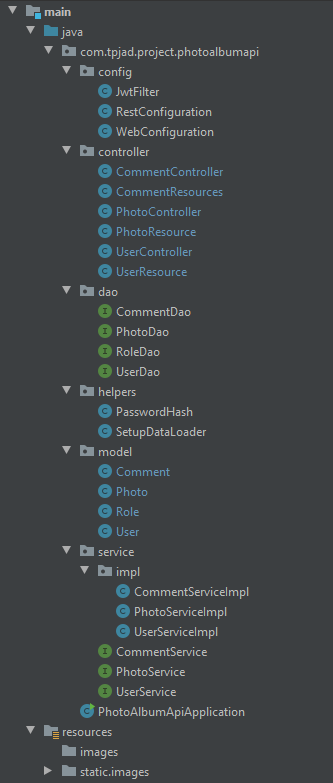
Aplicatia permite:

* Mai multe tipuri de utilizator, cu roluri diferite: USER\_ROLE, ADMIN\_ROLE
* Inregistrare user nou -> creare cont
* Login / Logout pentru fiecare tip de utilizator
* Vizualizarea meniului principal cu mai multe rute de navigare, in functie de rolul fiecarui utilizator
* Vizualizarea unui dashboard cu toate fotografiile incarcate pe platforma de catre toti userii
* Vizualizarea unui clasament al celor mai apreciate fotografii, in functie de numarul de like-uri primit
* Vizualizarea detaliilor unei fotografii (nume, descriere, numar de Like-uri primite, comentarii primite), aprecierea acesteia prin apasarea butonului Like sau retragerea aprecierii prin Dislike, adaugarea unui comentariu la fotografia respectiva, cu mentiunea ca functionalitatile de adaugare Like si Comment sunt disponbile unui user pentru o fotografie ce nu ii apartine (user nu isi poata da siesi, adica unei fotografii proprii, Like si Comment)
* Adaugare fotografie noua, prin mentionarea unor proprietati precum nume, descriere, si upload de pe computerul propriu.
* Stergerea unei fotografii (functionalitate permisa unui user in relatie cu propriile fotografii)
* Functionalitatile prezentate anterior sunt comune utilizatorilor indiferent de tipul de rol ce il detine, user sau admin.
* In plus, un admin cu rol ADMIN\_ROLE are acces la un dashboard intitulat User Management si are si o serie de functionalitati suplimentare, intre care:
  + Vizualizarea tuturor utilizatori inscrisi in aplicatie
  + Vizualizarea setului de fotografii corespondent fiecarui utilizator, a numarului de aprecieri primite pentru fiecare imagine incarcata, cate Like-uri a impartit respectivul user altor fotografii, ce comentarii a scris respectivul user altor utlizatori
  + Stergerea unei fotografii sau a unui comentariu, daca acestea sunt inadecvate, si stergerea unui utilizator, implicit a tuturor fotografiilor, comentariilor si like-urilor ce ii apartin, cu mentiunea ca un Admin nu isi poate sterge user-ul propriu, aplicatia aruncand o exceptie.

# Structura aplicatiei

Pachetele aplicatiei sunt organizate dupa cum urmeaza:

* Controller : contine controllerele ce furnizeaza endpoint-urile cu care opereaza front-end.
* Service : contine implementarile serviciilor oferite de endpoint-uri.
* Repository : contine interfetele specifice JPA Repository pentru operatii tip CRUD
* Model : contine entitatile proiectului, respective User, Role, Photo, Comment
* Config : contine configurarea setarilor de securitate, precum activarea partajarii CORS (Cross-Origin Resource Sharing) si inglobarea continutului in alte domenii.
* Helpers: contine resurse criptografice de encriptare a parolelor pe baza algoritmului SHA-512 si a functiei Salt si resurse de populare a bazei de date cu date initiale, intrucat SGBD-ul folosit este baza de date relationala H2, care este de tip in-memory si nu deci nu asigura persistenta datelor.



*Fig. 1 – Structura back-end*

# Arhitectura aplicatiei

Aplicatia opereaza dupa principiile serviciilor Web REST-ful, folosind o arhirectura de tip client-server, in care clientul (web-browser-ul) comunica cu application server prin apeluri de forma REST endpoint. Fiecarui request venit de la client ii este specifica o metoda anume si reprezinta in sine o functionalitate a aplicatiei.

@GetMapping("/allPhotos")  
public List<Photo> getAllPhotos ()

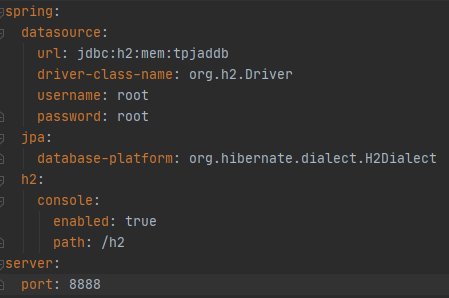
@PostMapping("/register")  
public User registerUser(@RequestBody User user)

@DeleteMapping(value = "/comment/delete/{id}")  
public ResponseEntity deleteCommentById(@PathVariable Long id)

*Fig. 2 - Endpoints*

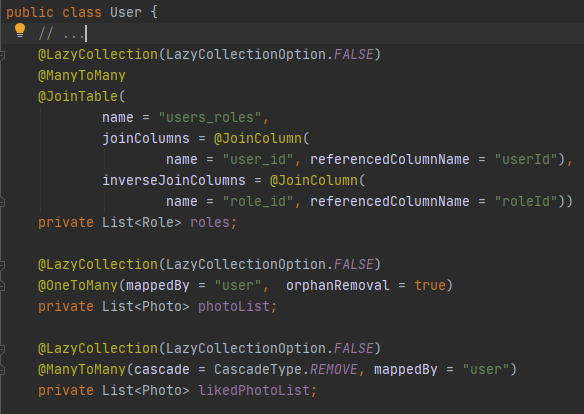
# Persistenta

H2 este o baza de date relationala ce permite crearea tabelelor atat in-memory cat si pe disk. Configurarea conexiunii aplicatiei curente la baza de date se face in fisierul application.yml, in care specificam tipul in-memory -> h2:mem:tpjaddb si faptul ca accesul la consola DB-ului se face la adresa: localhost:8888/h2.

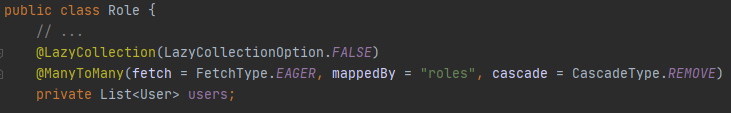


*Fig. 3 – application.yml*

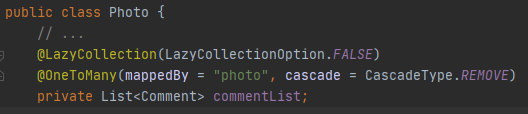
Aplicatia foloseste tabele temporare, in-memory, pentru fiecare entitate existenta, anume User, Role, Photo, Comment, precum si o tabela de legatura User\_Roles, ce evidentiaza relatia Many-To-Many intre entitatile User si Role. Alte legaturi existente intre tabele sunt: One-To-Many intre User si Photo (un user detine o lista de fotografii) si One-To-Many intre Photo si Comment (o fotografie contine o lista de comentarii), si sunt marcate bidirectional cu adnotarile specifice: @ManyToMany, @OneToMany etc.



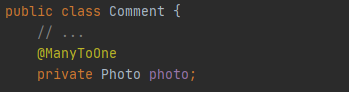
*Fig. 4 - User.class*



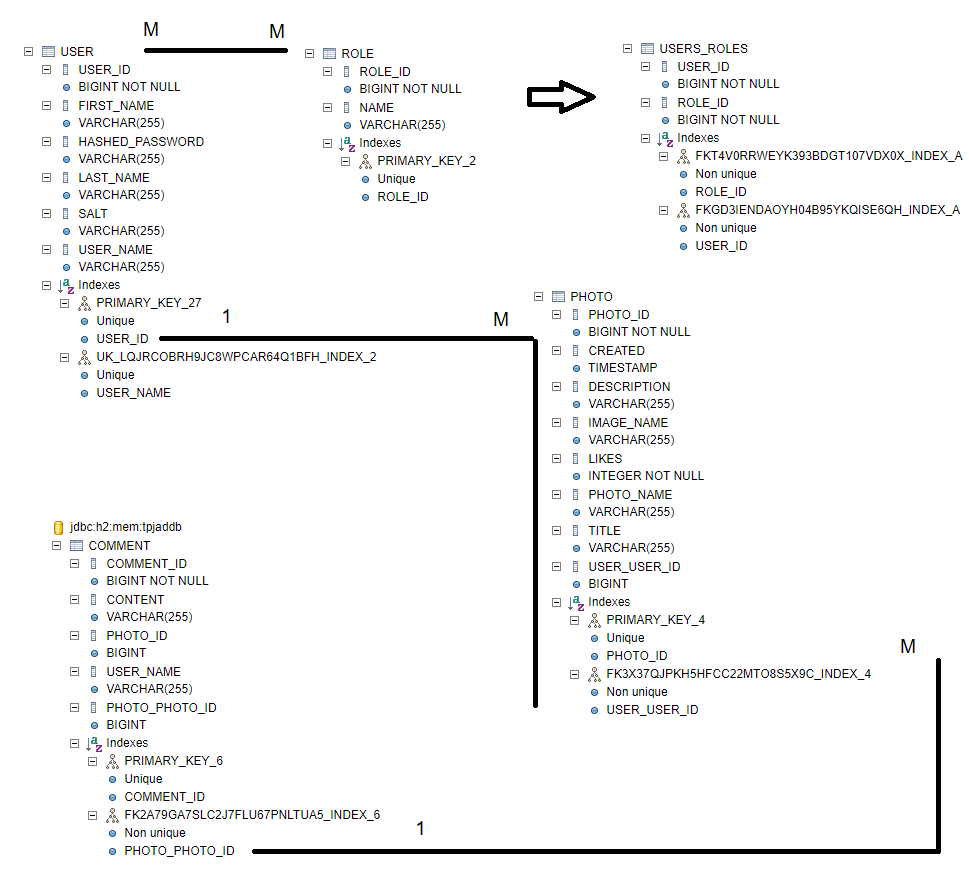
*Fig. 5 -Role.class*



*Fig. 6 - Photo.class*



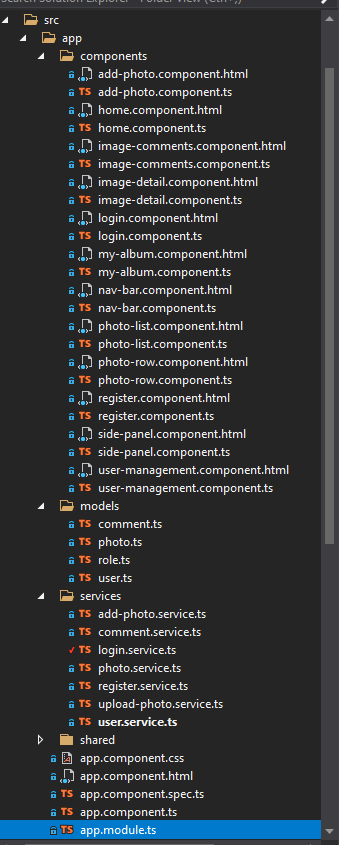
*Fig. 7 - Comment.class*



*Fig. 8 – Diagrama bazei de date*

# Angular

Frontend-ul aplicatiei este dezvoltat cu Angular, care este un framework structural JavaScript open-source pentru aplicatiile web prezentate ca o singura pagina dinamica (SPA - Single Page Application). O aplicatie de o pagina unica este interpretata drept o componenta de baza care ar putea contine cateva alte componente. O componenta este, de fapt, o colectie logica de template-uri HTML, stiluri asociate si modele care ar putea sa opereze intr-un template folosind imbinarile/legaturile din Angular.



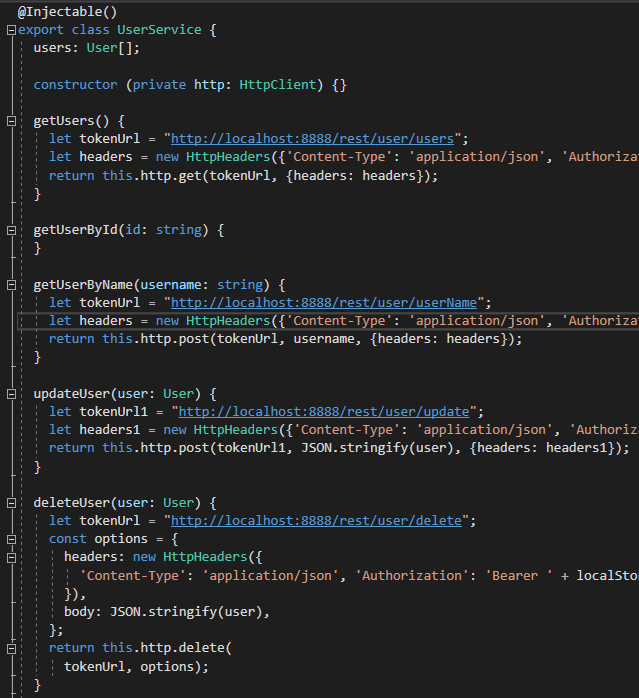
Intre caracteristicile specifice

unei aplicatii Angular amintim:

* Șablonul (Vizualizare)
* Domeniul de aplicare (model)
* Servicii
* Filtre
* Directivele

*Fig. 10 – Structura front-end*

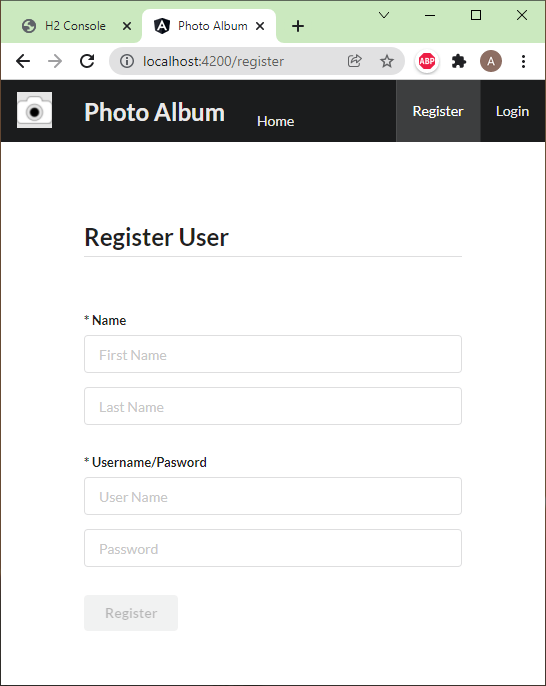
Transmiterea request-urilor de la client se face prin intermediul service-ului, ce contine logica aplicatiei si rutele catre server unde resursele necesare sunt disponibile.



*Fig. 11 – user.service.ts*

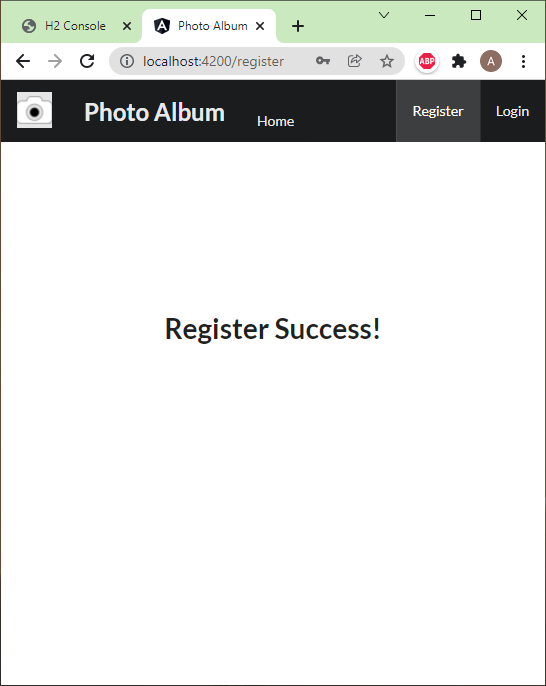
In continuare prezentam capturi din aplicatie si flow-ul acesteia.

* Inregistrare utilizator nou (toate campurile sunt obligatorii); Validare camp Username unic.



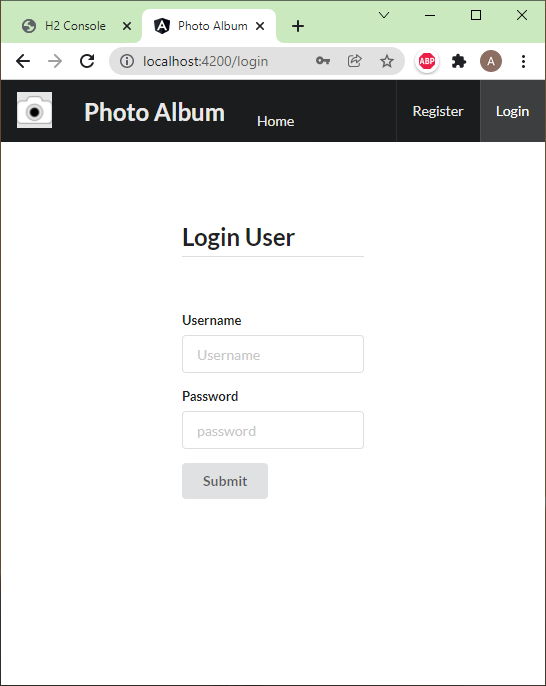
*Fig. 12 – Register*

* Inregistrarea a reusit => user nou a fost creat.



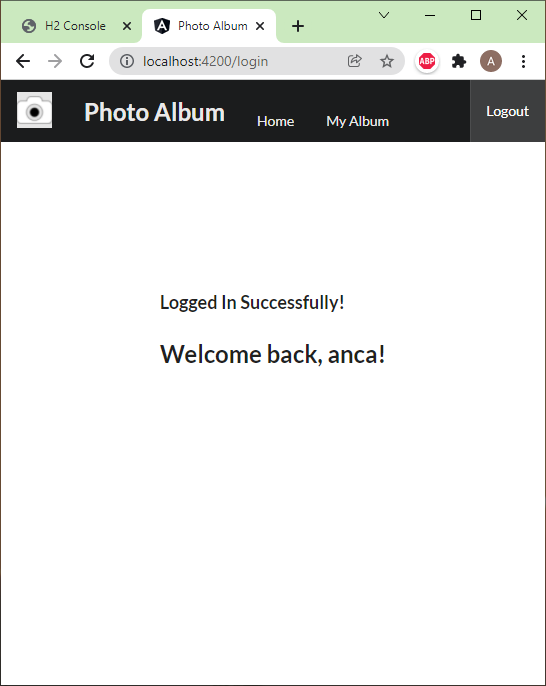
*Fig. 13 – Successful Register*

* Login in aplicatie. Fara aceasta, utilizatorul are statut de vizitator si acces doar la sectiunea publica de Home, unde poate doar viziona fotografiile disponibile si ranking-ul acestora, dar nu si detalii precum numar de aprecieri si lista de comentarii.



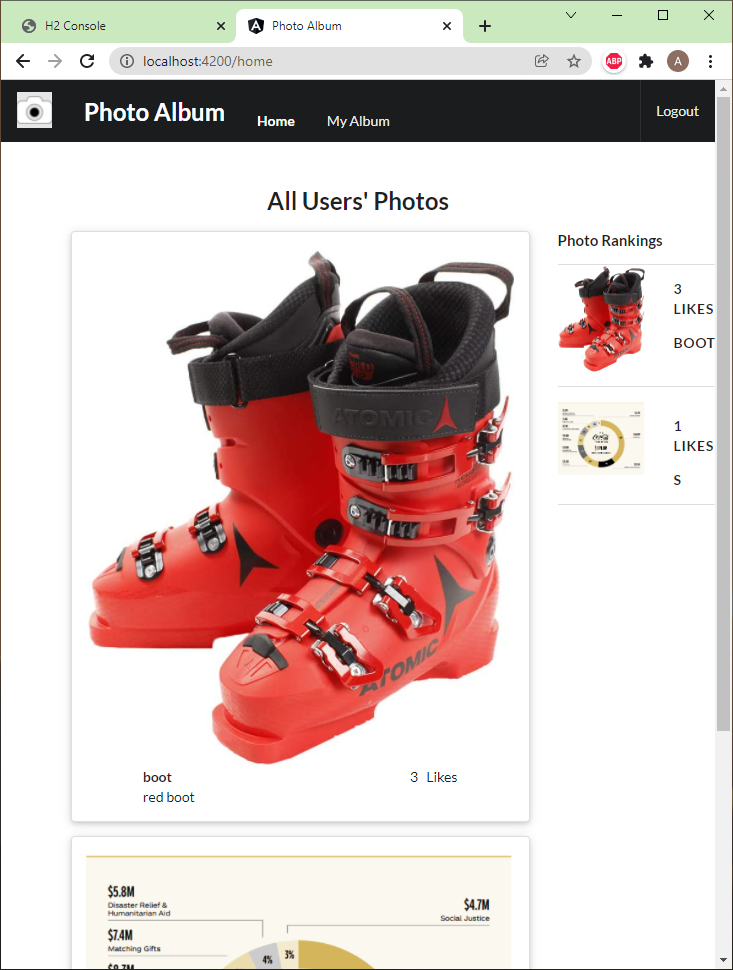
*Fig. 14 – Login*

* Login reusit



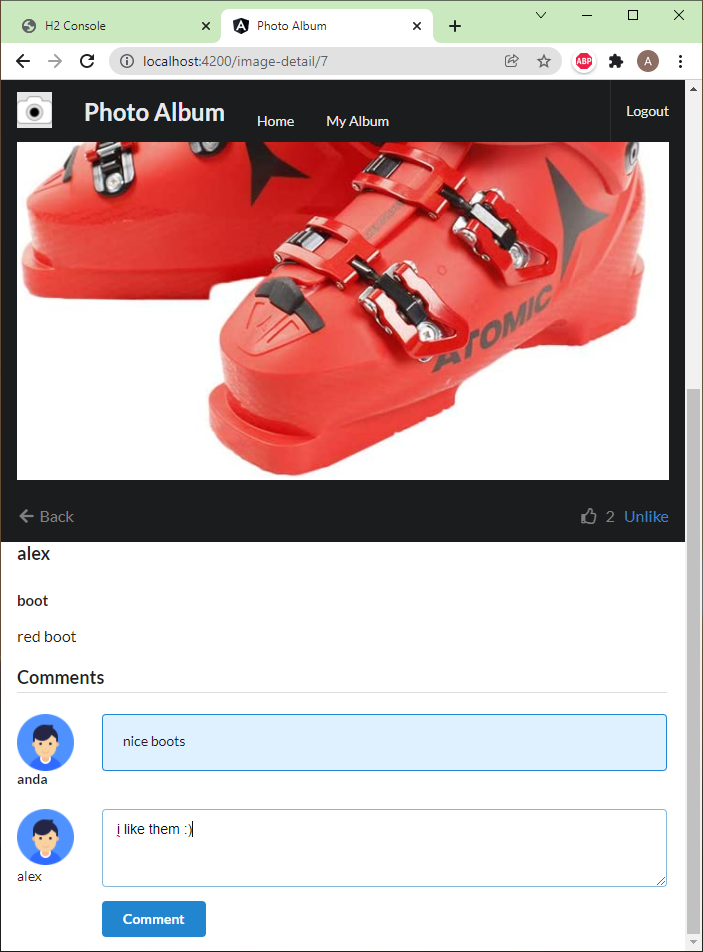
*Fig. 15 – Successful Login*

* Home – ROLE\_USER mode : vizualizare fotografii si clasamentul acestora



*Fig. 16 – Home*

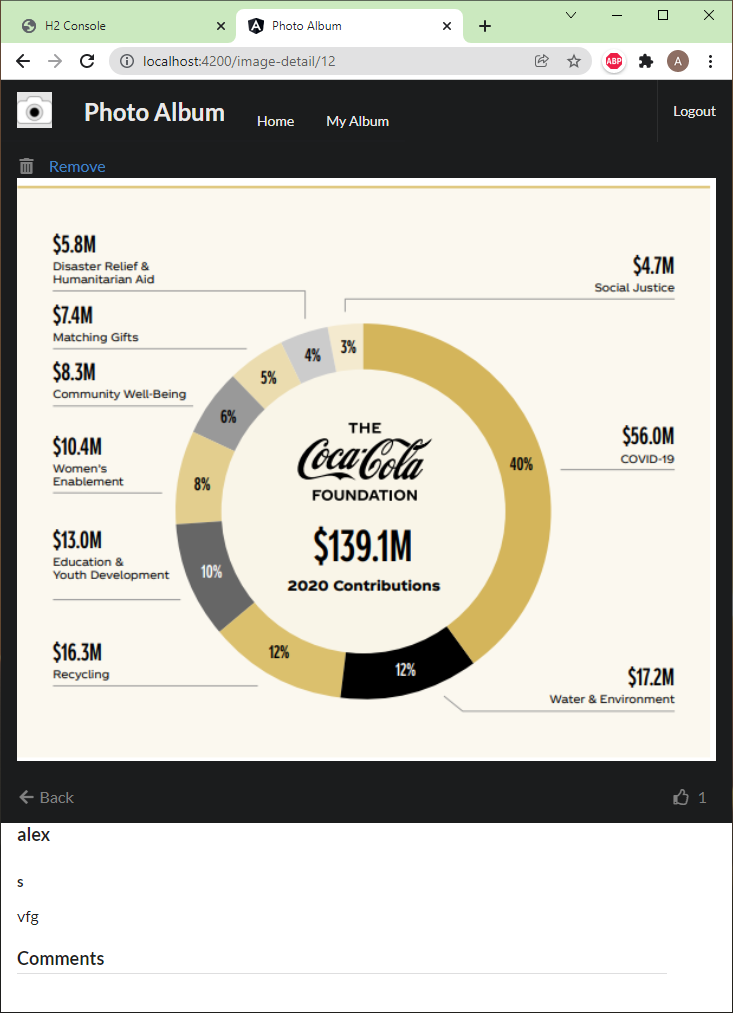
* Click fotografie => Image Detail : Vizualizare detalii fotografie (nume, descriere), numar de aprecieri, comentarii, adaugare like / dislike, adaugare comentariu.



*Fig. 17 –Image Detail*

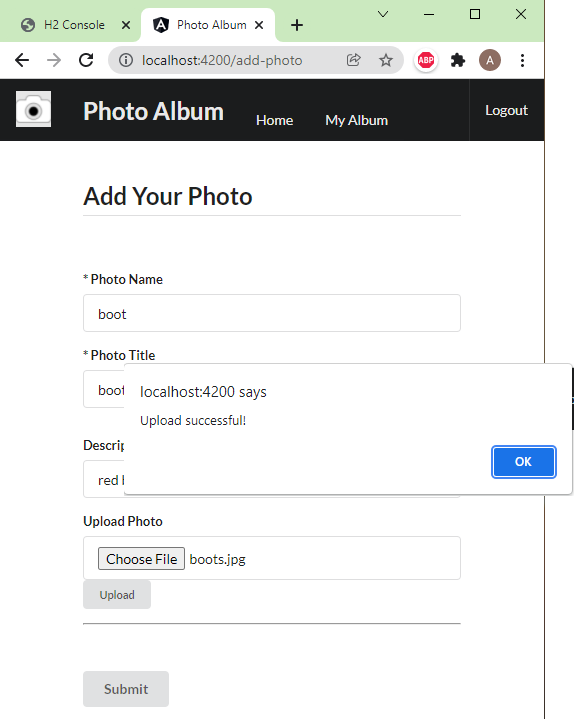
Nota:

Daca fotografia ale carei detalii le vizioneaza apartine user-ului logat, acesta nu isi poate sa siesi like sau comment, poate doar viziona comentariile primite si numarul de like-uri adunate. De asemenea, utilizatorul poate sterge respectiva fotografie, in cascada, cu toate comentariile, respectiv like-urile atasate.



*Fig. 18 – Image Detail, Same user*

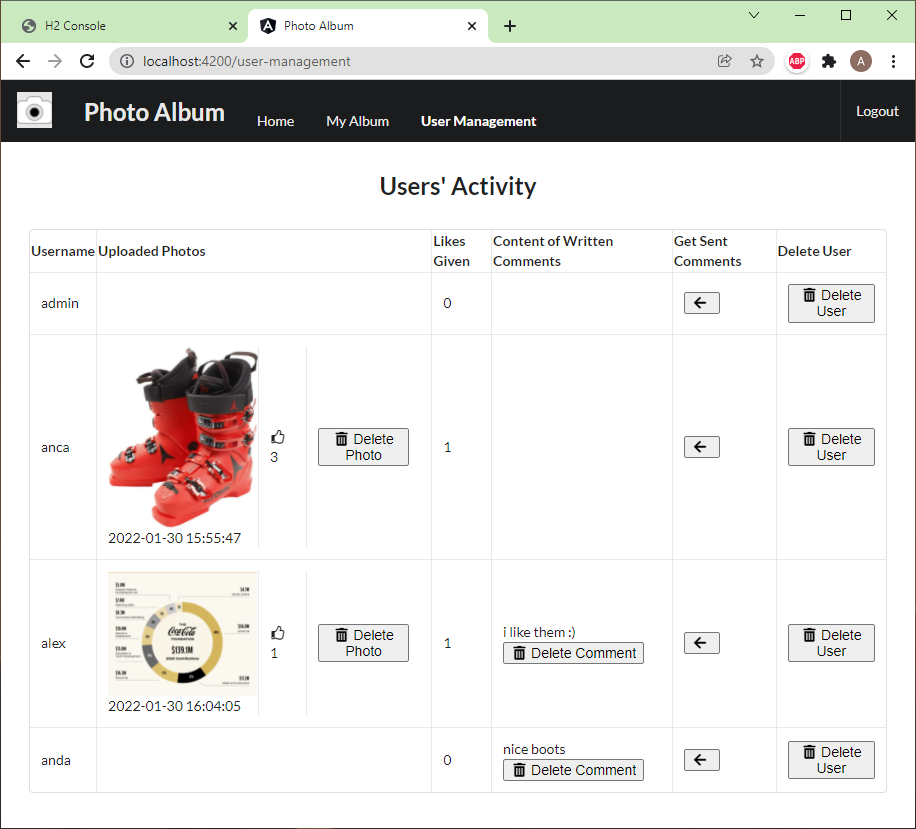
* My Album – Vizualizare fotografii proprii si adaugare fotografie noua. Daca upload reuseste, utilizator este instiintat printr-un mesaj pop-up.



*Fig. 19 – Add Image*

* ROLE\_ADMIN -> Username: admin; Password: admin

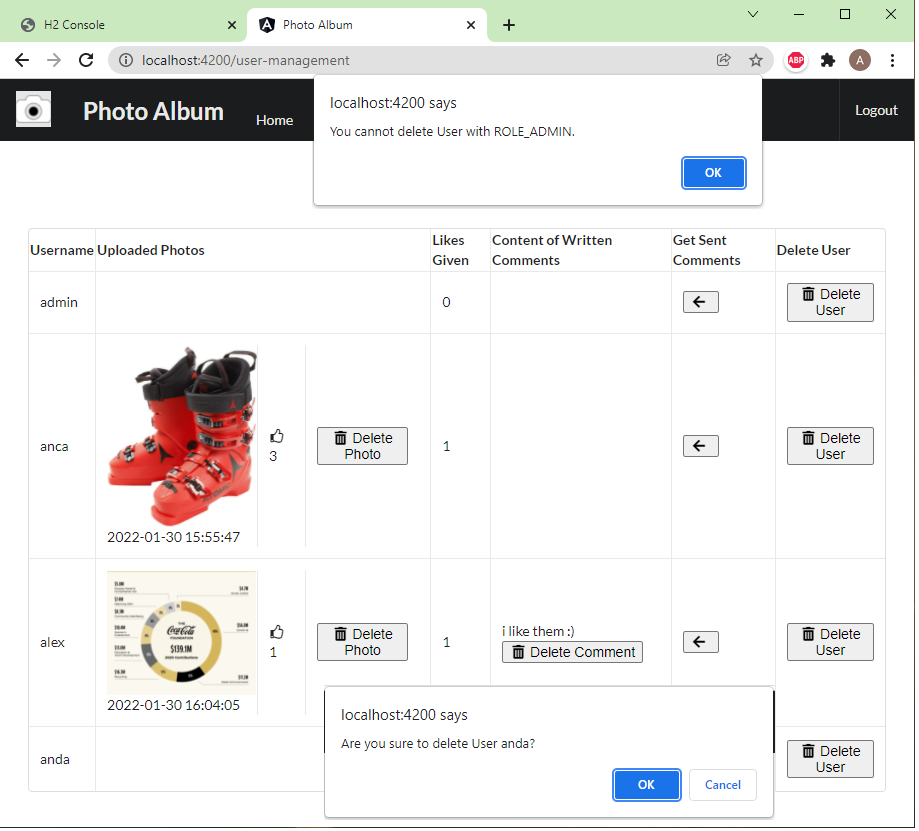
Utilizatorul Admin poate viziona toti user-ii inscrisi in aplicatie, lista fiecaruia de fotografii, totalul like-urile primite la fiecare fotografie ce ii apartine, totalul like-urilor date altora si comentariile date altora. De asemenea poate efectua operatii de stergere de fotografii, comentarii sau chiar utilizatori, daca acestia se abat de la niste norme etice, de exemplu.



*Fig. 20 – User Management*

Nota:

Aplicatia nu permite ca un Admin sa stearga un utilizator cu rol ROLE\_ADMIN (adica pe sine insusi) din baza de date, dar permite stergerea utilizatorilor cu rol ROLE\_USER.



*Fig. 21 – User Management Error*

1. Testarea

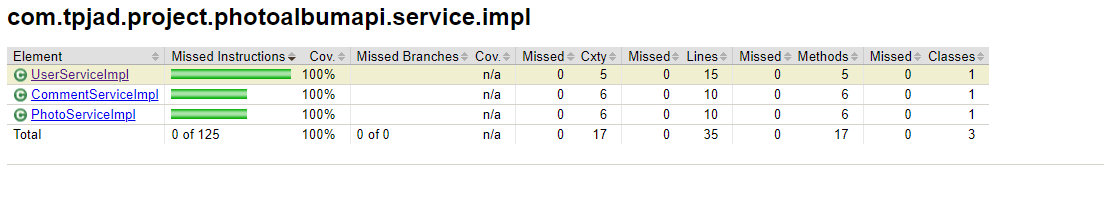
Pentru a testa functionalitatile principale oferite de serviciile implementate am folosit framework-ul Mockito. Acesta in principiu se bazeaza pe obiecte false care se presupun a proven din baza de date. Mai pe scurt, se ‘falsifica’ baza de date.

Exemple de testare cu Mockito arata ca in figura de mai jos:

Graphical user interface, text, application

Description automatically generated

Pentru prezentarea procentajului de acoperire al testelor unitare cu Mockito am ales sa integrez un plugin numit Jacoco. Acesta prezinta rata de acoperire a tuturor cazurilor de testare.



Text

Description automatically generated

1. Build-ul si deploy-ul

Pentru build-ul si deploy-ul aplicatiei am folosit docker/docker-compose. Acesta este o masina virtuala ‘portabila’ in sensul ca daca pe host e instalat docker, se poate descarca orice imagine pentru pregatirea mediului de lucru. Astfel, pentru build-ul aplicatiei m-am folosit de o imagine a lui gradle compatibil cu java 11, iar pentru buildul aplicatiei angular am luat o imagine a lui node. In containerul reprezentat de node, am instalat angular cli pentru a face buildul acesteia.

Astfel, docker-compose file-ul pentru buildul aplicatiei arata ca in figura de mai jos:

Graphical user interface, text, application

Description automatically generated

Fisierele aditionale folosite pentru build arata ca in figura de mai jos:

Graphical user interface, text, application

Description automatically generated

Build-ul pentru aplicatia angular, dupa cum se poate vedea depinde de un parametru $ACTION pasat in sectiunea de environment al docker-compose file-ului. Si astfel, putem sa facem mai multe actiuni cum ar fi crearea unui nou proiect, build-ul propriu-zis, adaugarea si salvarea unei dependinte, instalarea tuturor dependintelor, crearea unei noi componente.

Pentru deploy-ul aplicatiei, de asemenea am luat imagini ale lui openjdk, a server-ului web nginx pentru servirea fisierelor statice, si o imagine a lui node pentru servirea fisierelor builduite de catre ngcli.

Fisierul docker-compose arata ca in figura de mai jos:

Graphical user interface, text, application, email

Description automatically generated

Fisierele sh aditionale folosite arata ca in figurile de mai jos:

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Acesta din urma este defapt fisierul de configurare pentru nginx.

# Contributia fiecarui membru al echipei la dezvoltarea aplicatie

* Alexandru Sabou: backend, testare, deploy
* Anca Trandafir: frontend, documentatie

Contents

[1. Descrierea aplicatiei 2](#_Toc94615844)

[2. Functionalitati: 2](#_Toc94615845)

[3. Structura aplicatiei 3](#_Toc94615846)

[4. Arhitectura aplicatiei 5](#_Toc94615847)

[5. Persistenta 5](#_Toc94615848)

[6. Angular 8](#_Toc94615849)

[7. Contributia fiecarui membru al echipei la dezvoltarea aplicatie 20](#_Toc94615850)