



**Universitatea
Transilvania
din Brașov**

**FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ**

Programul de studii:
Informatică Aplicată

LUCRARE DE LICENȚĂ

FitClub

Absolvent: Necula Bogdan Alexandru

Coordonator: Lect. univ. Bocu Răzvan

Brașov
Iulie 2022

Cuprins

1	Introducere	4
1.1	Descrierea succintă a temei	4
1.2	Motivarea alegerii temei	5
1.3	Structura lucrării	5
2	Tehnologii folosite	7
2.1	Descrierea unei aplicații web	7
2.2	Generalități despre HTML	8
2.3	HTML5	9
2.4	Bootstrap	9
2.5	CSS	10
2.6	Framework-uri în Java	10
2.6.1	Spring Boot	11
2.6.2	Hibernate	11
2.7	Funcționarea unei aplicații web	12
2.8	Arhitectura unei aplicații web	14
2.8.1	Arhitectura pe două niveluri (client/server)	14
2.8.2	Arhitectura pe trei niveluri	15
2.9	Arhitectura Java EE	16
2.10	Arhitectura MVC	17
2.10.1	Stratul Model	17
2.10.2	Stratul View	18
2.10.3	Stratul Controller	18
2.11	React	19
2.12	JSON Web Token	20
2.13	MySQL	21
2.14	Git	21
2.15	Editoare de cod sursă	22
2.15.1	Visual Studio Code	22
2.15.2	IntelliJ IDEA	23
2.16	Găzduirea aplicației pe Microsoft Azure	23

3	Proiectarea aplicației	26
3.1	Implementarea server-ului web	26
3.1.1	Securitatea aplicației	26
3.1.2	Utilizarea JWT	29
3.1.3	Distribuirea postărilor	30
3.1.4	Afișarea postărilor	33
3.1.5	Ștergerea postărilor	34
3.1.6	Reacțiile utilizatorului la o postare	35
3.1.7	Revizuirea credențialelor de securitate	36
3.1.8	Afișarea listei cu utilizatori	38
3.1.9	Tratarea excepțiilor și a erorilor de validare	39
3.2	Implementarea aplicației client-side	40
3.2.1	Starea componentelor în React	40
3.2.2	React Hooks	41
3.2.3	Ciclul de viață al componentelor în React	43
3.2.4	Reîmprospătarea fluxului de postări	45
4	Ghid de utilizare a aplicației	47
4.1	Autentificarea	47
4.2	Înregistrarea	48
4.3	Navigarea pe site	51
4.4	Accesarea nepermisă a aplicației	56
5	Concluzii	58

Capitolul 1

Introducere

1.1 Descrierea succintă a temei

Această lucrare constă în dezvoltarea unei platforme web de social networking. Ideea acestei aplicații este de a împărtăși și de a găsi cele mai bune fotografii și videoclipuri. Fiecare profil de utilizator are un număr de urmăritori, reprezentând numărul de utilizatori care îl urmăresc, și respectiv, un număr de urmăriți, reprezentând câți alți utilizatori urmărește acesta, la rândul său.

Utilitatea aplicației se rezuma la posibilitatea de a partaja, în timp real, postări ce conțin text și chiar și imagini, însă beneficiază și de un sistem de reacții din partea următorilor, așadar un utilizator care urmărește un alt utilizator poate să aprecieze sau să nu aprecieze postările acestuia.

Dacă un utilizator dorește să urmărească un alt utilizator, acest lucru este posibil din pagina de profil a respectivului utilizator. Ca și precondiții, este necesară autentificarea cu un cont de utilizator, prin care oricine poate găsi și vizualiza profilul, împreună cu postările recente de pe site. În caz contrar, vizualizarea profilului unui utilizator este în continuare posibilă, însă accesul la resursele disponibile va fi limitat, precum faptul că postările respectivului utilizator nu vor putea fi vizibile.

Din punct de vedere al implementării pe partea de server, am ales framework-ul Spring[1], deoarece oferă un model cuprinzător de programare și configurare pentru aplicațiile moderne bazate pe Java, fiind, de asemenea, bine documentat, iar dezvoltarea unei aplicații nu necesită cumpărarea unei licențe.

Din punct de vedere al implementării pe partea de client, am ales să folosesc React[2], deoarece dispune de diverse instrumente preinstalate care pot fi utilizate direct pentru dezvoltare. De asemenea, React beneficiază și de o comunitate numeroasă, iar acest lucru încurajează rezolvarea eventualelor blocaje tehnice în timpul dezvoltării aplicației.

1.2 Motivarea alegerii temei

Transformarea digitală are implicații profunde pentru companii, și pentru societate în ansamblu. Această revoluție digitală este atât de importantă, încât unii experți o compară cu nașterea tiparului în urmă cu mai bine de cinci secole. În această nouă eră, canalele digitale se înmulțesc, iar utilizările lor cresc.

Rețelele sociale au devenit instrumente de comunicare esențiale. În prezent, companiile trebuie să profite de oportunitățile digitale pentru a-și dezvolta reputația, pentru a-și adapta cultura și pentru a-și fideliza clienții. Creșterea vizibilității mărcii sau împărtășirea noutăților sunt câteva dintre numeroasele avantaje pe care le poate oferi o platformă de social media unei companii.

De exemplu, Facebook este una dintre cele mai cunoscute și populare platforme de socializare, având un impact puternic în mediul online, prin faptul că a introdus noi strategii de marketing, comunicare și vânzare, unde succesul acestei platforme este datorat, în special, utilizatorilor dornici de web, de noi forme de consum.

Pentru mine, aceste fapte reprezintă un interes deosebit, și totodată, acestea au stat la baza alegerii temei, precum și dorință de a realiza o lucrare cât mai practică, și de actualitate.

Platforma se intitulează FitClub și se adresează clienților unei săli de fitness, oferindu-le posibilitatea de a comunica rapid și eficient.

1.3 Structura lucrării

- **Capitolul 1** - Capitolul curent, cu rol introductiv în ceea ce privește tema aleasă, motivarea alegerii temei, dar și structura lucrării.

- **Capitolul 2** - Acest capitol prezintă aspecte teoretice ale lucrării precum și o prezentare detaliată a tehnologiilor folosite la implementarea proiectului folosind atât exemple cât și explicații.
- **Capitolul 3** - În acest capitol vor fi prezentate detalii cu privire la implementarea propriu-zisă a aplicației, prezentarea modulelor acesteia, cu explicații sugestive ale codului folosit.
- **Capitolul 4** - Acest capitol va cuprinde prezentarea aplicației în sine, cu accent pe interfața de utilizator. Capitolul va cuprinde, de asemenea, și un ghid de utilizare al aplicației, cu diverse exemple și instrucțiuni de interacțiune.
- **Capitolul 5** - Concluziile și funcționalitățile adiționale pentru a spori complexitatea aplicației, în viitor, vor fi cuprinse în cadrul acestui capitol.

Capitolul 2

Tehnologii folosite

2.1 Descrierea unei aplicații web

O aplicație web poate fi manipulată direct online cu ajutorul unui browser web și nu trebuie instalată. Aceasta poate fi plasată și pe un server, iar accesul este simplu și se face prin intermediul unei rețele de calculatoare (internet, rețea locală etc.).

O aplicație web dinamică constă într-un set de pagini statice și dinamice al căror conținut este parțial sau total determinat. Conținutul final al unei pagini este determinat numai atunci când utilizatorul solicită o pagină de la serverul web, iar aceasta variază de la o cerere la alta, în funcție de acțiunile utilizatorului.

Spunem că o pagină web este statică dacă această pagină are un conținut fix și afișează acest conținut către toți utilizatorii.

În opoziție, o pagină web dinamică este mai complexă din punct de vedere tehnic, deoarece utilizează baze de date pentru a încărca informații, iar conținutul este actualizat de fiecare dată când utilizatorul se conectează la aplicație.

Există multe limbaje de programare pentru dezvoltarea de aplicații web. Asadar, pentru a programa această aplicație, am ales unul din cele mai populare framework-uri Java Enterprise Edition (Java EE), și anume, Spring, pe care îl voi prezenta ulterior.

2.2 Generalități despre HTML

HTML¹ este un limbaj de codare utilizat pentru a crea pagini pe care un browser web le poate afișa. Fișierele de tip HTML sunt alcătuite din elemente pereche, unul marcând deschiderea iar perechea sa, închiderea. Site-urile web sunt așadar compuse din diverse pagini HTML legate între ele, stocate undeva pe un server. De asemenea, acesta este motivul pentru care HTML este frecvent numit și "coloana vertebrală a internetului".

Browser-ul folosit de către utilizatori este capabil, astfel, să interpreteze acest limbaj, fără să țină cont de majuscule sau minuscule, afișând rezultatul prin intermediul unei ferestre web.

Organizarea unui document HTML:

- `<html> </html>`

Acest tag reprezintă elementul de nivel superior sau rădăcina oricărui document HTML. Browser-ul înțelege prin intermediul acestui tag că este vorba de un fișier HTML pentru a-l putea afișa. De asemenea, toate celelalte tag-uri trebuie să fie descendente ale acestui tag.

- `<head> </head>`

Între aceste tag-uri sunt conținute informații care pot fi citite automat (metadate) despre document precum titlul, script-urile, iar acestea vor fi prelucrate de browser.

- `<title> </title>`

Aceste tag-uri sunt folosite pentru a da un titlu documentului. Astfel, acest text se va afișa în bara de titlu a browser-ului.

- `<body> </body>`

Într-un document HTML, poate exista o singură astfel de pereche de tag-uri, ce reprezintă conținutul documentului care va fi afișat pe ecran.

¹Hyper Text Mark-up Language

2.3 HTML5

A fost prezentat oficial publicului pe 22 ianuarie 2008, având următoarea actualizare majoră în octombrie 2014. Obiectivele acestui update au fost de a îmbunătăți limbajul cu suport pentru cele mai recente caracteristici multimedia și alte caracteristici noi, de a menține limbajul ușor de înțeles atât de către oameni, cât și de către computere, fără rigiditatea XHTML-ului, dar și de a rămâne compatibil cu software-ul mai vechi.

HTML4 vs HTML5 :

În comparație cu ediția anterioară de HTML, respectiv HTML4, au fost modificate în mod direct structura, echivalentă marcării, caracteristicile care permit redarea (culori, font etc.), echivalente cu directivele destinate stilului paginilor web și partea de conținut.

HTML5 oferă totodată și o mai bună gestionare a erorilor și are o coerență ridicată în cazul documentelor malformate. De asemenea, versiunea nouă oferă suport în ceea ce privește stocarea de cantități consistente, descărcate de pe web pe spațiul local de memorie, permițând astfel utilizarea offline unor aplicații web.

2.4 Bootstrap

Bootstrap este un framework de dezvoltare web gratuit și open-source, dezvoltat de Twitter în 2011 și lansat pe GitHub în același an, care reprezintă o colecție cu numeroase fragmente de cod reutilizabile și versatile scrise în CSS, HTML și JavaScript pentru a dezvolta rapid și ușor pagini web reactive.

O pagină web reactivă este capabilă să se adapteze automat pentru a arăta mai bine în funcție de rezoluția sau dimensiunea ecranului ori aspectul de imagine.

Așadar, folosind Bootstrap, se pot crea dropdown-uri, alerte, pop-up-uri, navigări etc.

2.5 CSS

CSS² permite un control exact asupra aspectului elementelor HTML în browser, și astfel, se pot crea interfețe de utilizator reprezentative, fiecare pagină web putând avea un design unic. În CSS se poate defini dimensiunea, culoarea, poziția textului și a altor etichete HTML, în timp ce fișierele HTML definesc conținutul și organizarea acestuia.

HTML poate defini atât structura, cât și prezentarea, dar WWW³ recomandă utilizarea CSS, pentru a separa structura unui document HTML de prezentarea sa, deoarece în loc să se definească stilul fiecărui bloc de text și al fiecărui tabel în HTML-ul unei pagini web, CSS sprijină dezvoltatorii front-end să creeze un aspect uniform ce se poate aplica pe mai multe pagini ale unui site web, definindu-le o singură dată într-un fișier/document CSS, având extensia specifică ".css".

2.6 Framework-uri în Java

Obiectivul unui framework [3] este, în general, de a simplifica munca dezvoltatorilor, oferindu-le o arhitectură "gata de utilizare" care să le permită să nu înceapă de la zero la fiecare nou proiect. Se prezintă într-o varietate de forme care pot include unele sau toate următoarele:

- Un set de clase grupate de obicei sub formă de biblioteci pentru a furniza servicii mai mult sau mai puțin sofisticate.
- Proiectare eficientă bazată pe modele de proiectare pentru a propune tot sau o parte din scheletul unei aplicații.
- Standarde de dezvoltare
- Diverse instrumente pentru a facilita dezvoltarea aplicației

Aplicația FitClub folosește 2 framework-uri Java pe partea de server, și anume Spring Boot și Hibernate.

²Cascading Style Sheets

³World Wide Web

2.6.1 Spring Boot

Conceptul fundamental al Spring este injectarea dependențelor [4]. Acest framework a apărut ca răspuns la 3 factori de codare proastă, definiți în 1994 de Robert C. Martin, iar injectarea dependențelor este o soluție la această problemă:

- Rigiditate: este dificil să schimbi ceva, deoarece fiecare schimbare afectează prea mult funcționarea sistemului.
- Fragilitate: o modificare face ca funcționalitățile deja implementate care ar trebui să funcționeze corect să funcționeze defectuos.
- Imobilitate: este dificil de reutilizat codul într-o altă aplicație.

Spring Boot este, în principiu, o extensie a cadrului Spring, care elimină configurațiile de tip boilerplate necesare pentru configurarea unei aplicații Spring. Acesta adoptă o viziune avizată a platformei Spring, care deschide calea pentru un ecosistem de dezvoltare mai rapid și mai eficient.

Așadar, pentru a crea o aplicație web folosind Spring Boot, avem nevoie de dependențe, unde folosim Maven pentru a le gestiona, aceasta fiind una dintre cele mai bine cunoscute caracteristici ale sale.

Nu există mari dificultăți în gestionarea dependențelor pentru un singur proiect, dar atunci când avem de-a face cu proiecte cu mai multe module și aplicații formate din zeci sau sute de module, Maven poate ajuta la menținerea unui nivel ridicat de control și stabilitate.

2.6.2 Hibernate

Datele sunt cel mai adesea stocate în baze de date relaționale, astfel încât se recomandă utilizarea unui framework de mapare Obiect/Relațional pentru a asigura viteza, scalabilitatea și mentenabilitatea dezvoltărilor. Hibernate[5] răspunde acestei nevoi și are un mare succes de mulți ani. Acest succes poate fi explicat prin arhitectura sa, care este perfect adaptabilă la orice tip de dezvoltare și prin suportul pentru majoritatea bazelor de date.

Hibernate este un framework de persistență utilizat pentru a gestiona persistența obiectelor Java într-o bază de date. Termenul de mapare

obiect/relațională (ORM) descrie tehnica de conectare a reprezentării obiectuale a datelor la reprezentarea relațională a acestora, pe baza unei scheme SQL.

În termeni concreți, Hibernate permite ca un obiect definit în Java să fie legat/mapat la o tabelă dintr-o bază de date, prin intermediul unui fișier de corespondență declarativ. Sistemul se poate ocupa de crearea tabelelor în conformitate cu fișierele de configurare și, de asemenea, de actualizarea tabelelor, dacă este necesar, atunci când unul dintre fișierele de configurare se modifică.

Hibernate are mai multe moduri de a efectua interogări. Este posibil să se exprime interogări în SQL sau în HQL⁴.

2.7 Funcționarea unei aplicații web

Funcționarea unei aplicații web are la bază anumite mașini care comunică în rețea folosind un limbaj comun. Dintre aceste mașini, se face o distincție între cele care oferă resurse, serverele, și cei care le folosesc, și anume utilizatorii finali sau clienții. Resursele pot fi, de exemplu, documente HyperText, imagini, fișiere XML sau chiar programe (PHP, Java) responsabile pentru generarea lor la cerere.

Un server este un proces care execută o operație la cererea unui client și transmite răspunsul către client, care la rândul său este un procedeu care solicită executarea unei operații de la un web server process prin trimiterea unui mesaj care conține o descriere a operației care urmează să fie efectuată și așteaptă răspunsul la această operație printr-un mesaj de răspuns. Atunci când un client accesează o resursă (consultarea unui document, modificarea datelor stocate pe server etc.), acesta trebuie să utilizeze protocolul de comunicare HTTP pentru a ajunge la aceasta: acesta definește o semantică foarte simplă (GET, POST și alte comenzi) care permite formularea de cereri care sunt interpretate pe server de un program specific, și anume, serverul web.

Un server web este un simplu server de fișiere. Clienții i se adresează prin intermediul protocolului HTTP pentru a obține o resursă. Atunci când serverul web primește o astfel de cerere HTTP, acesta extrage pur și simplu numele resursei solicitate din cerere, apoi extrage resursa de pe disc și o

⁴Hibernate Query Language: limbaj specific pentru limbajul propriu Hibernate

împachetează într-un răspuns HTTP, pentru a o transmite clientului, fără a o prelucra înainte de a o transmite acestuia. Prin urmare, acesta poate transfera către un client o pagină HTML, o imagine, un sunet sau chiar un fișier executabil. Tipul de conținut al resursei solicitate este total irrelevant pentru aceasta. Atunci când un server web primește o cerere pentru o pagină web statică, acesta transmite această pagină direct către browserul care o solicită.

Cu toate acestea, atunci când serverul web primește o cerere pentru o pagină web dinamică, reacționează diferit: transmite pagina către un software special care este responsabil pentru finalizarea paginii. Acest software special se numește server de aplicații.

Un server de aplicații este radical diferit de serverul web, deoarece resursele care îi sunt oferite nu sunt doar fișiere statice, ci coduri pe care le va executa în numele clienților care le solicită. Atunci când un server de aplicații primește o cerere HTTP, acesta analizează și această cerere pentru a determina ce resursă este solicitată. De obicei, cererea este pentru un cod executabil găzduit de server. Spre deosebire de un server web aflat în aceeași situație, acesta nu transferă codul către client, ci îl execută, iar rezultatul acestui cod va fi returnat clientului.

Server-ul de aplicații citește codul din pagină, completează pagina în conformitate cu instrucțiunile cuprinse în cod și ulterior, elimină codul. Server-ul de aplicații trimite înapoi la server-ul web rezultatul acestui proces care constă într-o pagină statică. La rândul său, trimite această pagină la browser-ul solicitant. Browser-ul primește, în consecință, doar cod HTML pur atunci când îi este trimisă pagina, pe care îl prelucrează corespunzător.

Figura 2.1 prezintă o imagine de ansamblu a acestui proces:

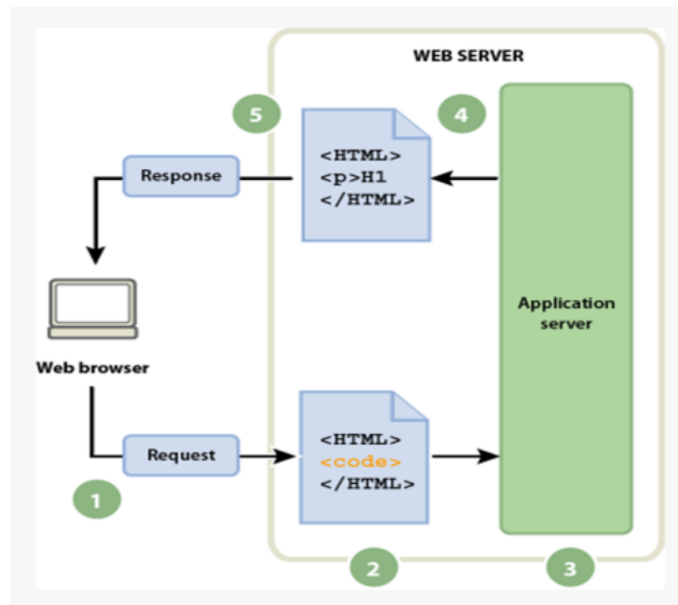


Figura 2.1: Funcționarea unei aplicații web

Observație: O resursă este identificată prin url (șir de caractere). Atunci când clientul dorește să ajungă la o resursă la distanță, acesta trimite o cerere HTTP în care menționează adresa URL a resursei.

2.8 Arhitectura unei aplicații web

2.8.1 Arhitectura pe două niveluri (client/server)

Arhitectura pe două niveluri numită și arhitectura ”client-server” nu este deloc complicată. Pe de o parte se află clientul, iar pe de altă parte server-ul, așa cum este reprezentat și în Figura 2.2.

Acest tip de arhitectură poate fi realizat pe orice tip de arhitectură hardware interconectată.

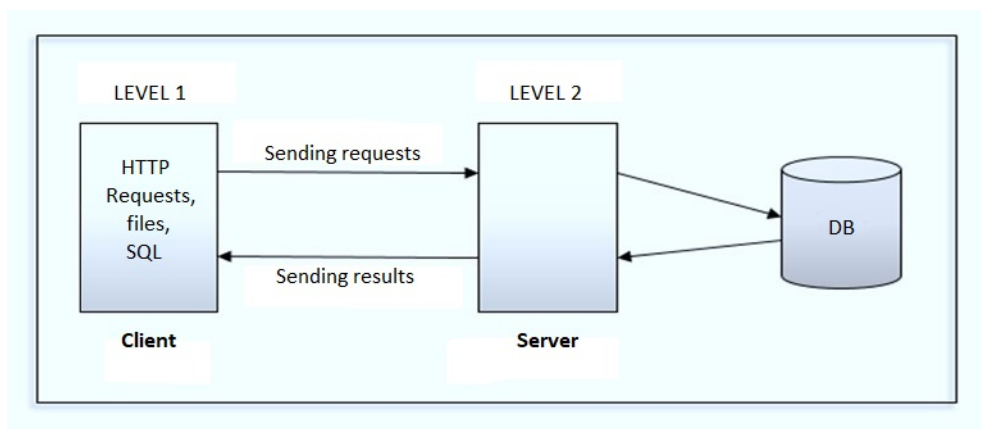


Figura 2.2: Arhitectura unei aplicații pe 2 niveluri

Punctele importante ale unei aplicații pe două niveluri:

- Clientul solicită un serviciu de la server, cum ar fi să i se afișeze o anumită pagină.
- Server-ul primește această cerere HTTP, efectuează procesarea și returnează resursa solicitată de client.
- Clientul primește resursa solicitată.

2.8.2 Arhitectura pe trei niveluri

În arhitectura pe trei niveluri (Figura 2.3), apare un nou nivel. Într-adevăr, mai avem încă primul nivel, care este clientul, care este foarte ușor, deoarece nu are niciun rol de procesare, utilizează aplicația, dar are doar o mică parte a aplicației pe dispozitivul său de lucru.

Toată procesarea se face apoi pe partea server-ului. La al doilea nivel se află server-ul de aplicații și, în sfârșit, la ultimul nivel, server-ul de baze de date, de unde se vor procesa anumite interogări SQL. Funcția fundamentală a unui server de baze de date este reprezentată de: stocarea, regăsirea și reactualizarea datelor. Pentru asigurarea acesteia, server-ul trebuie să ascundă față de utilizatorul care accesează site-ul, informațiile referitoare la implementarea fizică internă, cum ar fi detalii despre organizarea fișierelor, dar și structurile de stocare.

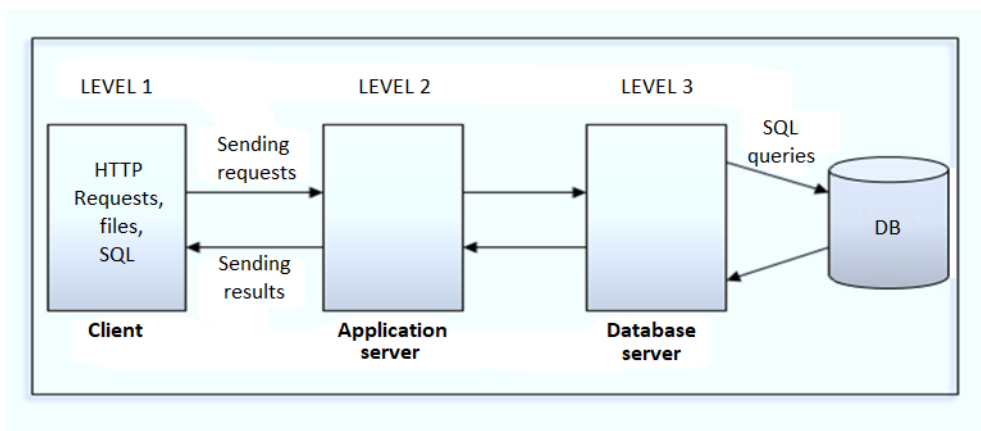


Figura 2.3: Arhitectura unei aplicații pe 3 niveluri

Punctele importante ale unei aplicații pe trei niveluri:

- Clientul este utilizat doar pentru a solicita și afișa răspunsurile serverului.
- Server-ul de aplicații se ocupă de calcule și chiar solicită servere suplimentare.
- Server-ul DB se ocupă de stocarea datelor.

2.9 Arhitectura Java EE

Arhitectura Java EE se bazează pe o arhitectură stratificată, care se bazează pe o împărțire în straturi, astfel încât vorbim de aplicații pe mai multe niveluri. Astfel, sunt reprezentate trei straturi principale, în care sunt distribuite diferite componente pe care le vom vedea în restul acestui capitol. Scopul acestei diviziuni este de a propune o mai bună distribuție a rolurilor (fiecare strat are un rol clar definit) și o mai bună mentenabilitate, având în vedere că fiecare strat este independent de celelalte.

Are loc, așadar, următoarea împărțire în straturi:

- Stratul destinat afișării paginilor web
- Stratul destinat gestionării logicii de afaceri
- Stratul destinat salvării informațiilor din baza de date

Stratul destinat afișării paginilor web gestionează afișarea datelor și interacțiunile aplicației cu utilizatorul. Separarea acestui strat face posibilă oferirea mai multor prezentări pentru aceeași aplicație: același strat de procesare poate fi utilizat pentru o aplicație grea și pentru o aplicație ușoară.

Stratul destinat gestionării logicii de afaceri se referă atât la sarcinile care trebuie efectuate de aplicație asupra datelor, cât și la prelucrările necesare în urma unei acțiuni din partea utilizatorului: (verificarea stării utilizatorului: autentificat/neautentificat, diverse calcule etc.).

Stratul destinat salvării informațiilor din baza de date (sistemul informatic al aplicației) grupează mecanismele de stocare și de acces la date, astfel încât acestea să poată fi utilizate de aplicație la nivelul de procesare.

2.10 Arhitectura MVC

Majoritatea platformelor de dezvoltare a aplicațiilor web, inclusiv Java EE, nu impun o anumită ordine a codului, adică putem dezvolta în orice mod doriți. Problema este că, dacă dezvoltăm în orice mod, codul rezultat va fi nesatisfăcător din punct de vedere calitativ și va deveni dificil să găsim o bucată de cod sau o funcție pe care dorim să o modificăm.

Pentru a evita această problemă, este recomandat să utilizăm bune practici de dezvoltare numite Design Patterns. Un model de proiectare permite descrierea liniilor principale ale unei soluții.

Modelul MVC descompune aplicația în straturi distincte și, prin urmare, are un impact puternic asupra organizării codului. MVC este acronimul de la Model - View - Controller. Practic, atunci când dezvoltăm o aplicație folosind arhitectura MVC, vom segmenta codul în trei părți sau straturi, fiecare strat având o funcție specifică.

2.10.1 Stratul Model

Aceasta este partea de cod care execută logica de afaceri a aplicației. Acest lucru înseamnă că este responsabil pentru recuperarea datelor, conversia lor în conformitate cu conceptele logicii aplicației, cum ar fi procesarea, validarea, asocierea și orice alte sarcini legate de manipularea datelor.

Este, de asemenea, responsabil pentru interacțiunea cu baza de date, știe cum să se conecteze la o bază de date și să execute interogări, folosind cele patru operațiuni de bază ale stocării persistente (CREATE, READ, UPDATE, DELETE) pe o bază de date.

2.10.2 Stratul View

Aceasta este partea de cod care se va ocupa de prezentarea datelor către utilizator, care returnează o vizualizare a datelor din model, cu alte cuvinte, este responsabilă pentru producerea interfețelor de prezentare ale aplicației din informațiile pe care le deține (de exemplu, pagina HTML).

Acest nivel nu se limitează însă doar la reprezentarea datelor în format HTML sau text, ci poate fi utilizat și pentru a oferi o mare varietate de formate, în funcție de ceea ce avem nevoie să utilizăm pentru a implementa o anumită funcționalitate.

2.10.3 Stratul Controller

Acesta este stratul responsabil cu rutarea informațiilor, va decide cine va prelua informațiile și le va procesa. De asemenea, gestionează cererile utilizatorilor și returnează un răspuns cu ajutorul straturilor Model și View, descrise anterior.

Figura 2.4 prezintă o imagine de ansamblu a arhitecturii MVC:

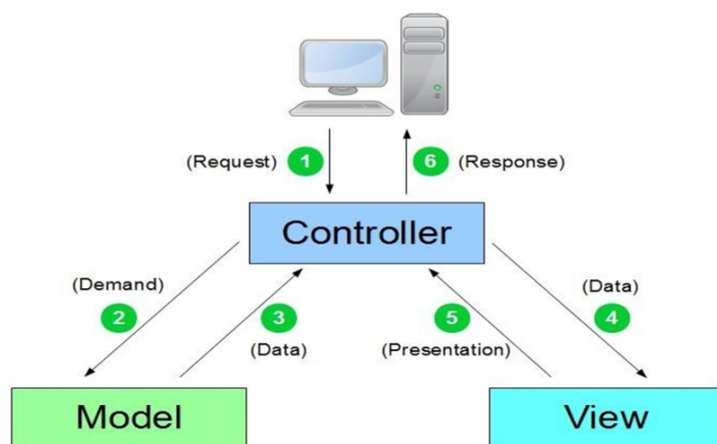


Figura 2.4: Arhitectura Model - View - Controller

Principiile arhitecturii MVC:

- Utilizatorul trimite cererea HTTP, o transmite server-ului de aplicații care o transmite direct către partea de cod numită Controller.
- Controller-ul se ocupă de direcționarea informațiilor, hotărând cine va prelua informațiile și apoi le va procesa, apelând, de fapt, modelul care conține informațiile structurate.
- Modelul va efectua calcule sau prelucrări pe baza acestor informații și le va trimite înapoi la Controller.
- Controller-ul va cere View-ului să genereze un View (pagină web).
- View-ul generează o pagină web așa cum a fost solicitată de controller și o trimite către Controller.
- Controller-ul primește pagina web pe care o va trimite utilizatorului, ca urmare a solicitării.

2.11 React

React este o bibliotecă JavaScript open-source dezvoltată de Facebook începând din 2013, ce s-a axat pe crearea de interfețe pentru utilizator declarative folosind un concept bazat pe componente, în care fiecare componentă are propria stare.

Dacă o componentă a paginii web tocmai a fost actualizată (de exemplu, dacă apăsăm pe un buton care declanșează apariția unui text pe pagină), acea parte este singura modificată de React fără a actualiza întreaga pagină. React utilizează "DOM virtual", care este o reprezentare a interfeței cu utilizatorul, stocată în memorie și sincronizată în mod constant cu "DOM-ul" real.

Așadar, React nu este un framework, ci mai exact o bibliotecă, deoarece se ocupă doar de redarea interfețelor de utilizare și rezervă multe lucruri la discreția proiectelor individuale, putând fi considerată "View-ul" în modelul MVC. De asemenea, bibliotecile externe pot ajuta dezvoltatorul să utilizeze atât funcționalitățile HTML, cât și cele CSS și să le construiască în JSX.

Observație: JSX este o sintaxă de extensie JavaScript utilizată în React cu scopul de a scrie cu ușurință HTML și JavaScript împreună.

Totuși, React are anumite limitări:

- Adesea, devine necesar ca anumite pachete să fie descărcate pe stația de lucru a dezvoltatorului pentru a putea accesa diverse funcționalități implementate deja în React
- Deși React suportă o mulțime de biblioteci externe, există foarte puține biblioteci native.
- Navigarea datelor în React este complicată și complexă, deoarece manipularea paralelă a datelor nu este suportată.

2.12 JSON Web Token

JWT⁵, este un mecanism utilizat pentru a partaja informații între două părți în siguranță - un client și un server. În cele mai multe cazuri, JWT este la bază un JSON codificat care conține un set de declarații și o semnătură. De obicei, este utilizat în contextul diverselor mecanisme de autentificare, pentru a partaja informații legate de utilizator. Este, de asemenea, o modalitate populară de autentificare/autorizare a utilizatorilor într-o arhitectură de microservicii.

Autentificarea JWT este un mecanism de autentificare fără stare, bazat pe token-uri, unde aceste token-uri, odată generate, nu mai pot fi modificate. Este utilizat în mod popular ca o sesiune fără stare bazată pe client, ceea ce înseamnă că serverul nu trebuie să se bazeze în totalitate pe un depozit de date sau pe o bază de date pentru a salva informațiile despre sesiune.

JWT-urile pot fi criptate, dar de obicei sunt codificate și semnate. Scopul JWT-urilor semnate nu este de a ascunde datele, ci de a asigura autenticitatea acestora.

Un JSON Web Token este împărțit în trei șiruri consecutive de caractere, separate print-un punct de suspensie:

- Primul șir reprezintă un Header, codificat în base64, în care se scrie tipul de token, algoritmul de semnătură utilizat etc.
- Al doilea șir de caractere va conține informațiile ce se doresc a fi reținute în componența token-ului pentru a fi utilizate ulterior, cum ar fi drepturile persoanei care deține acest token, etc. Aceste informații sunt

⁵JSON Web Token

structurate sub forma "subiect-informație": "informație-concretă". În acest context, apare noțiunea de "claims", ce reprezintă subiecte speciale care vor conține informații speciale, cum ar fi data de valabilitate a token-ului, numele serverului care a emis JWT-ul etc.

- Al treilea șir reprezintă semnătura. Scopul este de a concatena prima și a doua și apoi de a le hash-ui cu o cheie secretă. Astfel, este posibilă verificarea integrității token-ului, deoarece, dacă primul sau al doilea șir este modificat, al treilea șir nu va mai corespunde și se va ști că token-ul a fost modificat și, prin urmare, nu este valid.

Așadar, utilizarea unui astfel de mecanism de securitate devine relevantă în contextul protejării integrității informațiilor conținute în baza de date, deoarece fiecare utilizator primește un token unic în momentul autentificării pe site, ceea ce face ca utilizarea API-urilor disponibile de către orice altă sursă necunoscută să nu fie permisă.

2.13 MySQL

MySQL este un sistem de gestionare a bazelor de date care permite stocarea datelor într-un mod structurat (bază de date, tabele, câmpuri, înregistrări etc.). Nucleul acestui sistem permite accesul la informațiile stocate prin intermediul unui limbaj specific numit SQL.

Popularitatea MySQL este datorată, în special, de natura sa open-source, stabilitate, dar și de ușurința de utilizare și a performanței ridicate.

2.14 Git

Git este un software descentralizat de control al versiunilor creat de Linus Torvalds, autorul nucleului Linux. Git poate fi descris ca un instrument de urmărire a conținutului utilizat în principal pentru a stoca cod datorită celorlalte caracteristici pe care le oferă. Acest cod stocat continuă să se schimbe pe măsură ce se adaugă funcționalități sau se modifică cele deja existente.

Git are un depozit la distanță unde sunt stocate toate aceste informații și un depozit local care este stocat pe computer-ul fiecărui dezvoltator. Acest

lucru înseamnă că varianta completă a codului unei aplicații nu există doar pe depozitul central, ci este prezentă pe toate computer-ele dezvoltatorilor care lucrează la aplicația respectivă.

Așadar, un astfel de sistem de control al versiunilor ajută prin păstrarea unui istoric al modificărilor ce au avut loc de-a lungul dezvoltării aplicației.

2.15 Editoare de cod sursă

2.15.1 Visual Studio Code

În centrul său, Visual Studio Code oferă un editor de cod sursă foarte rapid, perfect pentru utilizarea zilnică. Cu suport pentru sute de limbaje, VS Code ajută prin caracteristicile sale, dar și prin mulțimea de extensii disponibile în Marketplace-ul integrat să menținem productivitatea la un nivel ridicat prin ilustrarea sintaxei, potrivirea corectă a parantezelor, indentare automată, selectarea fragmentelor de cod, a casetelor și multe altele. Scurtăturile de la tastatură sunt intuitive, datorită comunității numeroase de utilizatori care au contribuit cu astfel de sugestii, iar personalizarea ușoară a tuturor opțiunilor disponibile incluzând și schimbarea acestor scurtături de la tastatură, în funcție de propriile preferințe, permit navigarea extrem de ușoară și facilă prin codul aplicației.

Menționăm câteva dintre caracteristicile unice ale Visual Studio Code:

- Include un suport încorporat îmbogățit pentru dezvoltarea Node.js cu JavaScript, având instrumente excelente pentru tehnologii web, cum ar fi JSX/React, HTML, CSS, Bootstrap.
- Include suport încorporat pentru finalizarea codului, numit IntelliSense, folosit așadar pentru înțelegere și navigare semantică bogată a codului și refactorizare a codului.
- De obicei, suportă toate limbajele de programare, dar, dacă se dorește utilizarea unui limbaj de programare care nu este acceptat, atunci se poate descărca și utiliza o extensie.
- Include suport pentru Git, astfel încât se poate sincroniza în timp real cu sistemul de control al versiunilor fără a părăsi editorul, inclusiv vizualizarea modificărilor în așteptare.

2.15.2 IntelliJ IDEA

IntelliJ este unul dintre cele mai puternice și populare medii de dezvoltare integrată (IDE) pentru Java. Este dezvoltat și întreținut de JetBrains și este disponibil în edițiile Community și Ultimate. Acest IDE bogat în funcții permite dezvoltarea rapidă și ajută la îmbunătățirea calității codului. Algoritmul său predictiv poate presupune cu exactitate ceea ce un dezvoltator încearcă să tasteze și astfel, devine reprezentativă utilitatea acestei funcționalități prin diversele sugestii ce sunt afișate, chiar dacă nu cunoaște numele exact al unei anumite clase, membru sau orice altă resursă.

Menționăm, așadar, câteva dintre caracteristicile productive de top ale IntelliJ IDEA:

- Suportă completarea inteligentă a codului bazată pe context. Oferă o listă cu cele mai relevante simboluri aplicabile în contextul curent.
- Suportă completarea codului în lanț care este o funcție avansată ce listează simbolurile aplicabile accesibile prin metode în contextul curent.
- Permite utilizarea de metode sau constante statice, și adaugă automat declarațiile de import necesare pentru a evita erorile de compilare.
- Găsește din mers fragmentele de cod duplicat și oferă utilizatorului o notificare/sugestie în acest sens.
- Capabil să detecteze în timp real o posibilă greșeală, fie că vorbim despre o greșeală de scriere, sau accesarea unei variabile/metode inexistente sau orice altă situație ce poate genera eroare de compilare, caz în care o mică notificare cu becul apare pe aceeași linie. Inspectând această notificare, se afișează o listă de sugestii prin care dezvoltatorul poate remedia rapid greșeala.

2.16 Găzduirea aplicației pe Microsoft Azure

Microsoft Azure pune la dispoziție un abonament destinat studenților care oferă 100 de dolari în credite Azure ce pot fi folosite pentru a accesa diverse servicii⁶ Azure, fără a fi nevoie de un card de credit la înscriere. Printre

⁶Azure App Services

aceste servicii, se numără și posibilitatea de a găzdui o aplicație web.

Ca resurse principale, Microsoft Azure se bazează pe serviciile descrise anterior, în scopul găzduirii aplicațiilor web și a API-urilor web pentru orice platformă sau dispozitiv, și nu numai[6].

Pentru găzduirea platformei FitClub au fost necesare doua astfel de servicii Azure: unul pentru API-urile web, și unul pentru aplicația client-side. De asemenea, persistența datelor este asigurată și în acest caz, prin folosirea unui serviciu Azure care permite rularea unei instanțe de baze de date relaționale, numit Azure Database for MySQL: Single Server. Toate aceste servicii sunt definite în cadrul unui grup de resurse.

Aplicațiile sunt găzduite în planurile App Service, care sunt create într-un App Service Environment. Un App Service Plan este, în esență, un profil de planificare și configurare pentru găzduirea unei aplicații.

Fiecare aplicație, fie că este o aplicație web sau o aplicație ce implementează API-uri web, funcționează pe baza unui App Service Plan, fiind, de asemenea, limitată la puterea de calcul a acelui hardware dedicat, numit gazdă, care diferă în funcție de abonamentul și configurările specifice fiecărui utilizator[7].

Observație: Serviciul Azure responsabil pentru rularea unei instanțe de baze de date relaționale ce a fost descris anterior, nu are nevoie să opereze în cadrul unui App Service Plan.

Pentru găzduirea aplicației ce implementează API-urile web, este necesară generarea unor artefacte, folosind în terminal comanda `mvn clean install`. Concret, se generează un fișier cu extensia `.jar`, după rularea cu succes a tuturor testelor și curățarea tuturor rezultatelor compilărilor anterioare. După aceea, sunt necesare câteva configurări adiționale în cadrul proiectului care vor fi folosite pentru a încărca aplicația cu succes pe Microsoft Azure.

Pentru găzduirea aplicației client-side, este necesară crearea unui director ce conține versiunea de producție a aplicației, prin rularea în terminal a comenzii `npm run build`. Acest director va conține un set minimal de fișiere ce vor putea fi copiate în cadrul oricărui server web și care vor acționa ca aplicația client-side propriu-zisă.

De asemenea, sunt necesare configurări adiționale în cadrul aplicației client-side pentru a putea accesa endpoint-urile corespunzătoare aplicației ce implementează API-urile web.

Capitolul 3

Proiectarea aplicației

Aplicația FitClub este bazată pe două mari module:

- **RESTful API Web Service pe partea de server** - Acesta este creierul aplicației în sine unde se fac operațiile CRUD și accesăm API-uri înainte de a returna date structurate aplicației client-side.
- **Aplicația client-side** - Se realizează apeluri către Web Service-ul descris anterior, și se conectează elementele de UI pentru a accepta datele pe care ar trebui să le vadă fiecare tip de utilizator.

3.1 Implementarea server-ului web

3.1.1 Securitatea aplicației

Pentru a spori securitatea, am configurat manual comportamentul prin care Spring Security[8] interacționează cu aplicația. Implementarea corespunzătoare se află în clasa `SecurityConfiguration`, unde am suprascris metoda `configure` din care putem accesa obiectul corespunzător `HttpSecurity` care este obiectul de configurare utilizat de Spring Security, și în acest mod, putem să-i modificăm comportamentul. În mod implicit, Spring Security permite CSRF¹, care este la bază un token generat de back-end care caută acest token pentru a fi adăugat la cererile ulterioare de tip POST sau PUT generate de client. Am decis să nu folosesc acest token, deoarece aplicația folosește JWT.

¹Cross-site request forgery

Pe partea de autorizare, a fost necesar, de asemenea, să securizez endpoint-urile necesare pentru a evita cazul în care Spring să respingă cererile ulterioare, returnând codul 403 FORBIDDEN.

În acest caz, am folosit `http.authorizeRequests().antMatchers()` pentru a aplica autorizația la una sau mai multe căi, reprezentate prin tipul de cerere (`HTTPMethod`) și URL-ul corespunzător, pe care le specific în `antMatchers()`.

```
http.  
    authorizeRequests().and()  
    .exceptionHandling()  
    .authenticationEntryPoint(unauthorizedHandler)  
    .and()  
    .sessionManagement()  
    .sessionCreationPolicy(SessionCreationPolicy.STATELESS)  
    .and()  
    .authorizeRequests()  
    .antMatchers("/images/**", "/api/1.0/users/find/{search}",  
        "/api/1.0/auth/**").permitAll()  
    .antMatchers(HttpMethod.PUT,  
        "/api/1.0/users/{id:[0-9]+}").authenticated()  
    .antMatchers(HttpMethod.GET,  
        "/api/1.0/users/{username}/posts").authenticated()  
    .antMatchers(HttpMethod.POST,  
        "/api/1.0/posts/upload").authenticated()  
    .antMatchers(HttpMethod.POST,  
        "/api/1.0/posts/**").authenticated()  
    .antMatchers(HttpMethod.DELETE,  
        "/api/1.0/posts/{id:[0-9]+}").authenticated()  
    .antMatchers(HttpMethod.PUT,  
        "/api/1.0/posts/{id:[0-9]+}/like").authenticated()  
    .antMatchers(HttpMethod.PUT,  
        "/api/1.0/posts/{id:[0-9]+}/dislike").authenticated()  
    .antMatchers("/api/1.0/users/{id:[0-9]+}/follow",  
        "/api/1.0/users/{id:[0-9]+}/unfollow").authenticated()  
    .antMatchers(HttpMethod.GET,  
        "/api/1.0/posts/{id:[0-9]+}").authenticated()  
    .antMatchers(HttpMethod.GET,  
        "/api/1.0/posts/**").authenticated()  
    .and().authorizeRequests().anyRequest().permitAll();
```

În codul de mai sus, se remarcă faptul că ordinea din această configurare

este foarte importantă, deoarece dorim să asigurăm totuși un acces limitat unui vizitator atunci când accesează site-ul. Astfel, i se va permite să se autentifice, să caute prin utilizatorii deja existenți și să le vadă poza de profil. De asemenea, pot exista și endpoint-uri care sunt permise să treacă de partea de validare a Spring Security, însă în cadrul acestei aplicații, funcționalitățile de bază necesită verificări de autorizare din partea fiecărui utilizator.

În contextul folosirii JSON Web Token, am personalizat și modul în care aplicația va gestiona excepțiile ce vor rezulta din punctul de intrare pentru autentificare, definit în cadrul clasei `JwtAuthenticationEntryPoint`. Pentru a oferi un context, acest punct de intrare pentru autentificare gestionează excepțiile care trebuie aruncate atunci când un utilizator încearcă să acceseze o resursă care necesită autentificare, cum ar fi posibilitatea de a vedea postările unui alt utilizator.

În ceea ce privește gestionarea sesiunii, am optat pentru configurarea `SessionCreationPolicy.STATELESS`, deoarece acesta este un REST API și nu am dorit ca acesta să salveze sesiuni sau cookie-uri, acesta fiind și principalul motiv pentru care am optat să folosesc JWT, astfel încât serverul să nu fie nevoit să rețină sesiunea și de fiecare dată când o cerere vine cu un token valid, serverul o va prelucra corespunzător și va trimite un răspuns. Token-urile vor avea date de expirare, pot conține informații succinte despre utilizator, însă, la bază, aplicația folosește un JWT din dorința de a ne asigura că nicio stare nu se va salva pe server și pentru a valida cererile ce vin de pe partea de client. În schimb, sesiunea se va salva prin intermediul React Redux, indiferent dacă utilizatorul este sau nu autentificat.

De asemenea, în cadrul fișierului de configurări, am ales să evit injectarea directă a valorilor, și am creat o clasă denumită `AppConfiguration` pentru aceste proprietăți personalizate. Presupunând că este necesară o proprietate pe care dorim să o numim `uploadsPath` în cadrul aplicației, atunci, pentru ca Spring să o poată recunoaște în fișierul de configurări denumit `application.yml`, o vom scrie prin unirea cuvintelor cu liniuță, adică `uploads-path`. Pentru această clasă, trebuie, de asemenea, să folosim anotarea `@ConfigurationProperties`, care este necesară pentru a lega proprietățile externe de acest obiect și putem, de asemenea, să definim un prefix pentru aceste proprietăți personalizate, cum ar fi "fitclub". Concret, în fișierul de configurări `application.yml`, valoarea setată prin `fitclub.uploads-path` va seta așadar valoarea proprietății personalizate din clasă.

Așadar, utilitatea acestei clase se rezumă la accesarea rapidă a căii unde se

vor salva atașamentele ce pot veni de pe partea de client, fie din schimbarea pozei de profil a utilizatorului sau din atașarea unei imagini la o postare.

3.1.2 Utilizarea JWT

Implementarea JWT este relevantă în cadrul configurării pentru autentificarea fără stare, folosind un token JWT. Pentru a personaliza Spring Security în acest sens, am adăugat configurări suplimentare în clasa de configurare prezentată anterior astfel încât să configurăm managerul de autentificare cu furnizorul corect.

Pentru ca utilizatorul să se poată autentifica cu succes pe site, avem nevoie să încărcăm toate datele acestuia. În documentația Spring, `UserDetailsService` este descrisă ca o interfață de bază care încarcă datele specifice utilizatorului. În cele mai multe cazuri de utilizare, furnizorii de autentificare extrag informațiile privind identitatea utilizatorului pe baza credențialelor dintr-o bază de date și apoi efectuează validarea. Deoarece acest caz de utilizare este atât de comun, dezvoltatorii Spring au decis să îl extragă ca o interfață separată, care expune funcția unică `loadUserByUsername` ce acceptă numele de utilizator ca parametru și returnează obiectul de identitate al utilizatorului.

În implementarea acestei aplicații, am optat să construiesc o clasă nouă, numită `CustomUserDetailsService` care să extindă interfața `UserDetailsService`, astfel încât să pot obține acces la anumite configurări suplimentare.

```
@Override
public UserDetails loadUserByUsername(String username)
throws UsernameNotFoundException {
    User user = userRepository.findByUsername(username);
    if (user == null) {
        throw new UsernameNotFoundException("User not found with
            username: " + username);
    }
    return UserPrincipal.create(user);
}
```

După autentificare, ar putea fi necesar să verificăm credențialele utiliza-

torului, în cazul în care un hacker poate fura un token validat printr-un atac și îl poate injecta într-o cerere abuzivă. În acest context, Spring Security furnizează clasa `UsernamePasswordAuthenticationFilter` pentru a ajuta la efectuarea validării credențialelor, prin faptul că obținem acces la identitatea utilizatorului pentru a efectua autorizarea, iar extragerea acesteia are loc în cadrul filtrului JWT pe baza token-ului JWT furnizat. Așadar, am creat clasa `UserPrincipal` care implementează interfața `UserDetails` pentru a facilita `UsernamePasswordAuthenticationFilter`.

După autentificarea și autorizarea credențialelor utilizatorului, aplicația trebuie să gestioneze excepțiile în timpul acestor procese. Pentru a face acest lucru posibil, am creat o clasă numită `JwtAuthenticationEntryPoint` care să extindă `AuthenticationEntryPoint` și care suprascrie metoda `commence` pentru a personaliza mesajele de returnare.

```
@Component
public class JwtAuthenticationEntryPoint implements
    AuthenticationEntryPoint {

    private static final Logger logger =
        LoggerFactory.getLogger(JwtAuthenticationEntryPoint.class);
    @Override
    public void commence(HttpServletRequest httpServletRequest,
        HttpServletResponse httpServletResponse,
        AuthenticationException e) throws IOException,
        ServletException {
        logger.error("Responding with unauthorized error. Message
            - {}", e.getMessage());
        httpServletResponse.sendError(HttpServletResponse.SC_UNAUTHORIZED,
            e.getMessage());
    }
}
```

În general, în cadrul acestei clase vor fi capturate excepții de tip token invalid (neautorizat) și credențiale invalide.

3.1.3 Distribuirea postărilor

Un utilizator poate să distribuie postări, iar acestea apar pe pagina principală doar acelor utilizatori pe care îl urmăresc.

Posibilitatea de a vedea postările anumitor utilizatori este definită astfel:

```
public Page<Post> getPostsForUser(Pageable pageable, Long id) {
    User forUser = userService.getById(id);
    Set<User> users = forUser.getFollows();
    users.add(forUser);
    return postRepository.findByUserInOrderByIdDesc(users,
        pageable);
}
```

Se identifică utilizatorul autentificat pe site, și apoi, folosind caracteristicile colecției de obiecte `Set` unde fiecare obiect este unic, adăugăm toți utilizatorii pe care acesta îi urmărește. Folosind acest set, vom căuta mai departe postările ce aparțin acestor utilizatori, folosindu-ne de o instanță a clasei care reprezintă o abstractizare a persistenței datelor și este responsabilă pentru efectuarea de operații CRUD, sortarea și paginarea datelor, care, în fragmentul de cod prezentat anterior, este denumită `postRepository`.

Pentru a identifica pe cine urmărește un utilizator, am conceput o relație "many-to-many" între entitatea denumită `User` și ea însăși, putând stabili așadar o legătură a unei astfel de instanțe cu mai multe instanțe de același tip.

O postare suportă posibilitatea de a adăuga o imagine ca atașament, iar acest lucru necesită adăugarea de noi configurări, motiv pentru care am creat clasa `WebConfiguration` ce va fi verificată de Spring pentru configurațiile web, cum ar fi setarea de static resources sau setarea interceptorilor care vor intercepta cererile HTTP primite de pe partea de client.

```
@Override
public void addResourceHandlers(ResourceHandlerRegistry
    registry) {
    registry.addResourceHandler("/images/**")
        .addResourceLocations("file:" +
            appConfiguration.getUploadPath() + "/")
        .setCacheControl(CacheControl.maxAge(365, TimeUnit.DAYS));
}
```

De asemenea, a fost adăugată și configurarea pentru a salva resursele deja

încărcate pe pagină în cache, pentru a îmbunătăți rapiditatea și performanța site-ului la următoarea accesare, iar prin

`registry.addResourceHandler("/images/**")`, înțelegem că orice începe cu "images" va fi servit de această configurație.

Am tratat și cazul în care utilizatorul adaugă un atașament, însă nu dorește să trimită postarea. În acest caz, fișierele încărcate de mai mult de o oră vor fi șterse automat. Acest lucru este posibil prin faptul că se stochează data și ora încărcării fișierului și, dacă fișierul atașat are cel puțin o oră și nu are o postare atribuită, atunci va fi eliminat.

```
@Scheduled(fixedRate = 60 * 60 * 1000)
public void cleanupStorage() {
    Date oneHourAgo = new Date(System.currentTimeMillis() - (60 *
        60 * 1000));
    List<FileAttachment> oldFiles = fileAttachmentRepository.
        findByDateBeforeAndPostIsNull(oneHourAgo);
    for (FileAttachment file : oldFiles) {
        deleteAttachmentImage(file.getName());
        fileAttachmentRepository.deleteById(file.getId());
    }
}
```

Acesta este serviciul de clean-up periodic din cadrul clasei `FileService`, configurat să ruleze la o oră. Pentru ca acest serviciu să ruleze periodic, este necesară adăugarea adnotării `@EnableScheduling` pentru clasa precizată anterior, și apoi, adăugarea adnotării `@Scheduled` metodei care va rula periodic, denumită `cleanupStorage`. Pentru această adnotare, am fixat parametrul "fixedRate" corespunzător pentru a rula la fiecare 60 de minute, cu valoarea exprimată în milisecunde.

Observație: Serviciul de clean-up nu va afecta însă imaginile de profil ale utilizatorilor, care, în esență, nu au nicio postare atribuită. Aceste imagini nu vor fi tratate ca un `FileAttachment`, așa cum sunt tratate imaginile descrise anterior, ci se vor reține în atributul `image` din cadrul entității `User`, folosind valorile acestora exprimate într-un Base64 String.

3.1.4 Afișarea postărilor

Aplicația este configurată să afișeze un număr inițial de 10 postări pe pagină. Totodată, aplicația permite încărcarea pe pagină atât a postărilor vechi ale utilizatorilor, cât și a celor noi. Inițial, am avut un număr mare de interogări pentru a căuta diverse tipuri de postări, fie postări noi, vechi sau specifice unui utilizator, însă aceste interogări nu se diferențiau decât prin câțiva parametrii între ele. Pentru a face căutarea postărilor mai eficientă, am folosit următoarea metodă: în loc să creez o nouă metodă pentru fiecare diferență de parametrii, am generat interogări dinamice folosind JPA[9], adică am folosit **Specifications**, iar noile interogări s-au bazat pe acești parametrii.

```
private Specification<Post> getPostsAfter(long id) {
return (root, query, cb) -> {
    query.orderBy(cb.desc(root.get("id")));
    return cb.greaterThan(root.get("id"), id);
};
}

private Specification<Post> getPostsBefore(long id) {
return (root, query, cb) -> {
    query.orderBy(cb.desc(root.get("id")));
    return cb.lessThan(root.get("id"), id);
};
}
```

În aceste două exemple, unde am folosit **Specifications**, am tratat două situații care apar frecvent atunci când un utilizator navighează pe site.

Primul exemplu se rezumă la identificarea postărilor noi. Se consideră o postare nouă, dacă aceasta are "id-ul", reținut în baza de date, mai mare decât cea mai recentă postare ce se afișează pe pagină, al cărei id îl primim ca parametru în cadrul metodei numite **getPostsAfter**. A se avea în vedere faptul că ordinea afișării acestor postări este descrescătoare, în funcție de id.

Al doilea exemplu se rezumă la identificarea postărilor vechi. Această căutare funcționează similar cu cea din primul exemplu, doar că în acest caz, se vor identifica doar acele postări care au "id-ul" mai mic decât ultima postare afișată pe pagină, deoarece, așa cum am spus anterior, ordinea afișării acestor postări este descrescătoare, în funcție de id.

De asemenea, aceeași funcționalitate este prezentă și pe pagina de utilizator, afișând pentru fiecare utilizator doar postările care îi aparțin, cu posibilitatea de a încărca postări mai vechi, și de vedea în timp real dacă apare o postare nouă din partea acestuia.

3.1.5 Ștergerea postărilor

Un utilizator autentificat are posibilitatea de a-și șterge una sau mai multe postări. În acest sens, a fost tratată situația de a nu i se permite utilizatorului autentificat pe site să șteargă postările unui alt utilizator. Rezolvarea acestei situații se rezumă la extragerea postărilor din baza de date pentru care se vor identifica utilizatorii care le-au trimis și care vor fi comparați ulterior cu utilizatorul autentificat.

```
@DeleteMapping("/posts/{id:[0-9]+}")
@PreAuthorize
    ("@postSecurityService.isAllowedToDelete(#userPrincipal,#id)")
GenericResponse deletePost(@CurrentUser UserPrincipal
    userPrincipal, @PathVariable long id) {
    postService.deletePost(id);
    return new GenericResponse("Post removed!");
}
```

Această logică nu poate fi acoperită printr-o singură expresie Spring, motiv pentru care am implementat această funcționalitate într-o metodă dintr-o clasă separată numită `PostSecurityService`, apelând așadar această metodă, folosind o expresie Spring.

```
public boolean isAllowedToDelete(UserPrincipal userPrincipal,
    long postId) {
    Optional<Post> optionalPost = postRepository.findById(postId);
    if (optionalPost.isPresent()) {
        Post inDB = optionalPost.get();
        return inDB.getUser().getId() == userPrincipal.getId();
    }
    return false;
}
```

Parametrii acestei metode sunt `userPrincipal`, prin care se identifică utilizatorul autentificat, și, de asemenea, `postId`, ce reprezintă id-ul postării ce se dorește a fi ștearsă. Se face așadar, validarea dintre "id-ul" utilizatorului căruia îi aparține postarea, și cel al utilizatorului autentificat, și se trimite răspunsul corespunzător.

3.1.6 Reacțiile utilizatorului la o postare

Aplicația permite 2 tipuri de reacții la o postare și anume, utilizatorul poate să aprecieze sau să nu aprecieze postarea. Astfel, am declarat un enum cu aceste doua valori (LIKE, DISLIKE), iar pentru a gestiona o astfel de cerere trimisă de un utilizator, am folosit adnotarea `@PutMapping`. Această adnotare acționează ca o prescurtare, adnotarea echivalentă acesteia fiind: `@RequestMapping(method = RequestMethod.PUT)`.

Această metodă este utilizată pentru a modifica/actualiza o resursă din baza de date, utilizatorul trimițând date care actualizează întreaga resursă. În acest sens, PUT este similară cu POST în sensul că poate crea resurse, însă, PUT poate să suprascrie întreaga entitate dacă aceasta există deja sau să creeze o nouă resursă, dacă nu există.

```
private void react(Reaction reaction, long id, User user) {
    Post inDB = postService.getPost(id);
    PostReaction reactionInDB =
        postReactionRepository.findByPostAndUser(inDB, user);
    if (reactionInDB == null) {
        PostReaction postReaction = new PostReaction();
        postReaction.setReaction(reaction);
        postReaction.setPost(inDB);
        postReaction.setUser(user);
        postReactionRepository.save(postReaction);
    } else if (reactionInDB.getReaction() == reaction) {
        postReactionRepository.delete(reactionInDB);
    } else {
        reactionInDB.setReaction(reaction);
        postReactionRepository.save(reactionInDB);
    }
}
```

Pentru a avea o mai bună evidență a tuturor reacțiilor, am creat o

noua entitate, numită `PostReaction`. Astfel, se identifică postarea corespunzătoare, și se creează o nouă înregistrare de tipul `PostReaction`, setând postarea identificată anterior, reacția propriu-zisă, dar și utilizatorul care a reacționat la acea postare. De asemenea, au fost tratate anumite situații de excepție, ce pot apărea în cadrul acestui sistem de reacții:

- Utilizatorul dorește să retragă o reacție, caz în care se va șterge această înregistrare din baza de date.
- Utilizatorul a reacționat deja la o postare și dorește să își modifice reacția, situație în care reacția sa veche va fi înlocuită de cea nouă.

3.1.7 Revizuirea credențialelor de securitate

Utilizatorul are posibilitatea, în orice moment, după ce a finalizat înregistrarea cu succes, să își poată revizui credențialele de securitate, adică numele de utilizator și parola pe care le folosește în momentul autentificării.

Aplicația dispune, așadar, de două acțiuni pe care utilizatorul le poate face, una fiind destinată actualizării adresei de email, și cealaltă fiind folosită pentru schimbarea parolei.

În acest caz, aplicația folosește un sistem prin care asigură securitatea contului utilizatorului, fără de care, în cazul unui atacator, credențialele utilizatorului ar putea fi ușor modificate, iar contul acestuia ar fi compromis.

Astfel, fiecare dintre cele două acțiuni va genera un token valid pentru 24 de ore, ce va transmis printr-un mail către adresa de email furnizată de către utilizator în momentul înregistrării. Dacă utilizatorul a fost într-adevăr cel care a dorit să facă această revizuire, atunci link-ul ce conține token-ul descris anterior, îl va conduce către o pagină de unde va putea să introducă datele actualizate.

Simpla accesare a link-ului, fără a propaga schimbări în baza de date, nu va anula token-ul pentru utilizări ulterioare. Totuși, schimbarea cu succes, fie a adresei de email, fie a parolei, va anula acel token, și nu va mai putea fi folosit ulterior.

Observație: Utilizatorul poate opta, de asemenea, să trimită ambele acțiuni în același timp, și așadar, va primi două mail-uri, însă dacă actualizarea adresei de email se va face prima, această acțiune va determina anularea

token-ului generat în scopul schimbării parolei, deoarece aparține încă de vechea adresă, ceea ce îl face lipsit de actualitate.

```
public boolean changeEmail(String email, EmailRequest
    updatedEmail) {
    try {
        User userInDB = userRepository.findByEmail(email);
        VerificationToken newToken = new
            VerificationToken(userInDB);
        newToken.setEmailToken(
            jwtTokenProvider.generateVerificationToken(
                userInDB.getUsername()));
        userInDB.setVerificationToken(newToken);
        userInDB.setEmail(updatedEmail.getNewEmail());
        userInDB.setEmailVerificationStatus(false);
        userRepository.save(userInDB);
        emailSender.verifyEmail(userInDB);
        return true;
    } catch (RuntimeException e) {
        e.printStackTrace();
    }
    return false;
}
```

Atunci când utilizatorul solicită o schimbare a credențialelor, este necesar să identificăm înregistrarea corespunzătoare acestuia în baza de date. După aceea, se generează un token pe baza acestei înregistrări, folosind metoda `generateVerificationToken` din cadrul clasei `JwtTokenProvider`.

Clasa `JwtTokenProvider` este responsabilă pentru gestionarea token-ilor, care acoperă factori precum generarea, verificarea validității și tratarea excepțiilor ce pot apărea în cadrul folosirii JWT.

De asemenea, actualizarea adresei de email va provoca și schimbarea valorii atributului `emailVerificationStatus`, care este un `boolean` ce va fi setat ca fiind `false`, și care este utilizat pentru a verifica dacă adresa de email a utilizatorului este sau nu confirmată. Așa cum am prezentat anterior, un utilizator va avea acces limitat la resursele disponibile dacă nu are adresa de email confirmată.

Așadar, imediat după actualizarea adresei de email, utilizatorul va primi

un nou mail pe adresa actualizată în care va trebui să acceseze link-ul aferent pentru a-și confirma noua adresă și a avea din nou acces la toate resursele disponibile.

3.1.8 Afișarea listei cu utilizatori

Posibilitatea de a vedea lista cu utilizatorii deja înregistrați, care au, de asemenea, adresa de email confirmată este disponibilă oricărui tip de utilizator, fie autentificat sau anonim, cu mențiunea că unui utilizator anonim nu îi este permis să poată căuta un utilizator specific, în funcție de numele complet sau numele de utilizator.

```
@GetMapping("/users/find/{searchText}")
Page<UserVM> getUsers(@PathVariable String searchText, Pageable
    page) {
    return userService.findAll(searchText, page).map(UserVM::new);
}
```

În cadrul acestei metode, se folosește adnotarea `@PathVariable` pentru a prelua parametrul ce va veni din solicitarea utilizatorului după ce acesta a introdus în câmpul de căutare ceea ce dorește sau mai exact, pe cine dorește să caute.

În cadrul aplicației client-side, acest câmp de căutare este un câmp de text editabil ce afișează un buton de căutare, un buton de ștergere și, dacă nu a fost introdus nimic, se afișează, de asemenea, și un indiciu de text.

Se apelează apoi o altă metodă responsabilă cu căutarea efectivă și returnarea rezultatelor, folosind o interogare. Fiind vorba de utilizatori, vom declara o nouă interogare în cadrul clasei care este responsabilă pentru efectuarea de operații CRUD pentru entitatea `User`, și anume: `UserRepository`.

Pentru această interogare, am folosit adnotarea Spring Data JPA `@Query`[10] care declară interogări direct în cadrul `UserRepository`.

```
@Query("FROM User u WHERE u.emailVerificationStatus = true AND
    (u.displayName LIKE %:searchText% OR u.username LIKE
    %:searchText%) ORDER BY u.username, u.displayName ASC")
```

```
Page<User> findAllUsers(@Param("searchText") String searchText,  
    Pageable page);
```

În mod implicit, Spring Data JPA va utiliza legarea parametrilor pe bază de poziție. Acest lucru face ca metodele de interogare să fie puțin predispuse la erori de refactorizare în ceea ce privește poziția parametrilor. Pentru a rezolva această problemă, am folosit adnotarea `@Param` pentru a da parametrului de metodă `searchText` un nume concret și pentru a lega acest nume în interogarea propriu-zisă.

În situația în care utilizatorul nu a introdus nimic în câmpul de căutare, atunci parametrul `searchText`, fiind un `String`, va avea valoarea implicită `"`, ceea ce nu va afecta rezultatul interogării, și se vor afișa, așadar, toți utilizatorii înregistrați care au, de asemenea, adresa de email confirmată.

3.1.9 Tratarea excepțiilor și a erorilor de validare

În cadrul aplicației, pot apărea diverse excepții și erori pentru diverse acțiuni pe care utilizatorul dorește să le facă. De exemplu, în momentul înregistrării, acesta poate introduce un nume de utilizator invalid. Un nume de utilizator se consideră invalid dacă nu respectă regulile impuse bazate pe mărimea minimă și maximă acceptată sau dacă este deja folosit de către un alt utilizator. Totodată, în cazul în care dorește să își modifice imaginea de profil, și vom presupune că încarcă un fișier de tip text, atunci nu i se va permite să facă o astfel de schimbare, și i se va afișa un mesaj pe pagină în legătură cu acest lucru.

Pentru a gestiona toate aceste situații de excepție, am creat o nouă clasă numită `ExceptionHandlerAdvice` și am folosit o adnotare în cadrul acesteia prin care Spring ne permite să colectăm și să gestionăm aceste excepții, numită `@RestControllerAdvice`.

```
@ExceptionHandler({MethodArgumentNotValidException.class})  
@ResponseStatus(HttpStatus.BAD_REQUEST)  
ApiError  
    handleValidationException(MethodArgumentNotValidException  
        exception, HttpServletRequest request) {  
    ApiError apiError = new ApiError(400, "Validation error",  
        request.getServletPath());
```

```
BindingResult result = exception.getBindingResult();

Map<String, String> validationErrors = new HashMap<>();
for (FieldError fieldError : result.getFieldErrors()) {
    validationErrors.put(fieldError.getField(),
        fieldError.getDefaultMessage());
}
apiError.setValidationErrors(validationErrors);
return apiError;
}
```

Așadar, în cadrul metodei `handleValidationException`, se vor colecta aceste excepții, adăugând atât câmpul care a generat excepția, dar și mesajul aferent, acestea fiind utile în aplicația client-side, deoarece, folosind și o interfață prietenoasă cu utilizatorul, acestuia i se vor afișa mesaje clare și sugestive pentru a-l sprijini să finalizeze ceea ce a dorit să facă pe site.

3.2 Implementarea aplicației client-side

3.2.1 Starea componentelor în React

În cadrul aplicației client-side, am folosit preponderent, componente care rețin o stare, precum "PostFeed.js" sau "PostSubmit.js", dar și componente funcționale, care sunt în esență, funcții JavaScript, precum "Spinner.js". Aceste componente funcționale pot fi folosite în cadrul componentelor care rețin o stare, acestea având și un nivel de complexitate mai ridicat.

Pentru a gestiona starea componentelor în React, am folosit o bibliotecă JavaScript open-source, numită React Redux[11]. React utilizează Redux în vederea construirii interfeței cu utilizatorul. Acesta permite componentelor React să citească date dintr-un Redux Store și să transmită acțiuni către Store pentru a actualiza datele. Redux joacă un rol important în dezvoltarea aplicației, oferind o modalitate eficientă de gestionare a stării printr-un model de flux de date unidirecțional.

```
let localStorageData = localStorage.getItem('fitClub-auth');
let persistedState = {
    id: 0, username: '', displayName: '', image: '', password: '',
```



```

    isLoggedIn: false, isTokenValid: false, jwt: ''
  };
  if (localStorageData) {
    try { persistedState = JSON.parse(localStorageData);
      apiCalls.setAuthorizationHeader(persistedState);
    } catch (error) { }
  }

  const middleWare = addLogger ? applyMiddleware(thunk, logger) :
    applyMiddleware(thunk)
  const store = createStore(authReducer, persistedState,
    middleWare);
  store.subscribe(() => {
    localStorage.setItem('fitClub-auth',
      JSON.stringify(store.getState()));
    apiCalls.setAuthorizationHeader(store.getState());
  });

```

În fragmentul de cod de mai sus, am prezentat componenta funcțională `configureStore.js` care creează instanța specifică Redux Store în cadrul aplicației și se ocupă de funcționalitatea de actualizare a stocării locale pentru orice modificare a stării sale interne.

3.2.2 React Hooks

În cadrul componentelor funcționale, putem folosi React Hooks[12] și putem obține același comportament în privința actualizării stării în cadrul componentelor funcționale ca și în cazul componentelor de clasă. Așadar, se va returna tot ce ține de JSX, variabile și alte operații logice în interiorul acestei funcții, întrucât variabilele și operațiile logice se vor declara folosind `const`.

De asemenea, React Hooks introduce utilizarea unor valori de stare pentru fiecare dintre variabile, la începutul definirii componentei funcționale, folosind `useState`.

Concret, `useState` returnează o listă și în această listă există 2 elemente: primul este valoarea reținută în stare, și al doilea este funcția de actualizare a acestei valori. Putem obține aceste variabile prin "object destructuring", aceasta fiind o caracteristică JavaScript utilizată pentru a extrage proprietăți din obiecte și a le lega de variabile.

Exemplu: `const [username, setUsername] = useState('');`

Dacă nu furnizăm nicio valoare pentru `useState`, atunci aceasta va fi nedefinită (`undefined`) în mod implicit.

```
const [username, setUsername] = useState("");
const [password, setPassword] = useState("");
const [apiError, setApiError] = useState();
const [pendingApiCall, setPendingApiCall] = useState(false);

const onClickLogin = () => {
  const body = {
    username,
    password,
  };
  setPendingApiCall(true);
  props.actions
    .postLogin(body)
    .then((response) => {
      setPendingApiCall(false);
      props.history.push("/");
    })
    .catch((error) => {
      if (error.response) {
        setPendingApiCall(false);
        setApiError(error.response.data.message);
      }
    });
};
```

React Hooks permite accesarea variabilelor de stare să fie accesate fără a folosi `this.state`, deoarece acestea sunt variabilele funcției ce reprezintă totodată, întreaga componentă funcțională. Având în vedere acest lucru, transmiterea datelor sau a proprietăților de la o altă componentă se va face prin intermediul unui parametru de funcție.

Exemplu: `export const LoginPage = (props) => [...]`.

De asemenea, prin intermediul React Hooks, putem defini o logică de stare personalizată pentru a îmbunătăți performanța aplicației, folosind un "Hook" numit `useReducer`.

```
const reducer = (state, action) => {
  if (action.type === "update-success") {
    return {
      ...state,
      inEditMode: false,
      image: undefined,
      originalDisplayName: undefined,
      pendingUpdateCall: false,
      user: {
        ...state.user,
        image: action.payload,
      },
    };
  }
  return state;
};
```

De exemplu, pentru pagina de utilizator, schimbarea numelui complet și a imaginii de profil necesită 15 randări ale paginii folosind o implementare fără React Hooks, iar cu `useReducer`, au fost necesare doar 9 randări pentru ca întreaga operațiune să fie finalizată.

3.2.3 Ciclul de viață al componentelor în React

În React[13], ciclul de viață al unei componente este împărțit în patru faze:

- Faza inițială
- Faza de montare (Mounting)
- Faza de actualizare (Updating)
- Faza de demontare (Unmounting)

Deoarece am introdus anterior React Hooks, aplicația folosește astfel, alternative mai moderne[14] pentru a satisface cele patru faze ale ciclului de viață al unei componente care pot fi considerate, așadar, ca o modificare a componenteii.

În acest context, aplicația folosește un ”Hook”, numit `useEffect`, cu scopul de a înlocui metodele corespunzătoare ciclului de viață (`componentDidMount`, `componentWillUnmount`, `componentDidUpdate`).

`useEffect` primește 2 parametri: primul este funcția de apelare, iar pentru cel de-al doilea parametru se transmite matricea de variabile (numită listă de dependențe). Aceste variabile vor fi verificate de React înainte de a rula funcția `callback`. Dacă interacțiunea utilizatorului cu aplicația client-side provoacă modificări ale acestor variabile, atunci operația logică definită în cadrul Hook-ului `useEffect` va fi executată din nou.

Revenind la fragmentul de cod anterior, unde am prezentat o parte din funcționalitatea paginii de autentificare, adăugam următoarea secvență de cod:

```
useEffect(() => {  
  setApiError();  
}, [username, password]);
```

Atunci când un utilizator introduce credențiale invalide, i se va afișa un mesaj prin care să i se semnaleze acest lucru. Reprezentarea acestui mesaj în cadrul JSX se face prin intermediul valorii de stare `apiError`, iar funcționalitatea dorită este ca acest mesaj să dispară atunci când utilizatorul introduce noi credențiale, fiind nevoie, așadar, să se apeleze funcția de actualizare a valorii de stare.

Deci, pentru a rula `setApiError` și a seta valoarea de stare `apiError` ca `undefined`, aplicația trebuie să detecteze modificările câmpurilor aferente numelui de utilizator și al parolei. Astfel, ori de câte ori are loc o modificare pentru aceste câmpuri, se va declanșa apelarea acestei funcții de actualizare.

Așadar, `useEffect` urmărește modificările pentru aceste valori de stare, și, pe baza acestor modificări, aplicația identifică interacțiunile utilizatorului și câmpurile pe care le modifică, astfel încât se va elimina mesajul de eroare de pe pagină.

3.2.4 Reîmprospătarea fluxului de postări

Aplicația actualizează periodic fluxul de postări noi pentru un utilizator, atât timp cât sesiunea acestuia nu a expirat. Inițial, după ce pagina s-a încărcat cu succes și a afișat postările celor pe care îi urmărește utilizatorul autentificat, se identifică ID-ul celei mai recente postări. În cazul în care nu există nicio postare de afișat, acest ID va avea valoarea 0 în mod implicit.

În urma apelării endpoint-ului prin care se obține numărul de postări noi, se verifică dacă această valoare este mai mare decât 0, iar în caz afirmativ, se va afișa o notificare pe pagină cu care utilizatorul poate să interacționeze.

```
useEffect(() => {
  const checkCount = () => {
    const posts = page.content;
    let topPostId = 0;
    if (posts.length > 0) {
      topPostId = posts[0].id;
    }
    if (hasFullAccess) {
      apiCalls
        .loadNewPostsCount(topPostId, props.user,
          props.loggedInUser.jwt)
        .then((response) => {
          setNewPostsCount(response.data.count);
        })
        .catch((error) => {
          if (props.user) {
            props.fromChildToParentCallback(true);
          }
          setHasFullAccess(false);
        })
    }
  };
  if (hasFullAccess) {
    intervalRef.current = setInterval(checkCount, 1500);
    return function cleanup() {
      if (isLoadingNewPosts) {
        clearInterval(intervalRef.current);
        intervalRef.current = setInterval(checkCount, 50);
      }
      clearInterval(intervalRef.current);
    };
  }
});
```

```
    };  
  }  
  
  }, [props.user, props, page.content, isLoadingNewPosts,  
    props.loggedInUser, hasFullAccess]);
```

În fragmentul de cod de mai sus, se setează un interval de 1500 milise-cunde (1 secundă și jumătate), pentru a se apela în mod repetat metoda definită în `useEffect` prin care se obține numărul de postări noi pentru un utilizator.

Setarea acestui interval se face folosind o funcție JavaScript, numită `setInterval` care primește ca parametrii funcția care se dorește a fi ape-lată periodic și intervalul propriu-zis, exprimat în milisecunde.

De asemenea, în cazul în care sesiunea utilizatorului expiră, iar apelarea endpoint-ului de verificare a postărilor noi va produce o eroare, atunci nu se va mai putea apela funcția `setInterval` pentru a seta un nou interval, și astfel, utilizatorul va fi nevoit să se autentifice din nou pe site pentru a avea acces la aceste resurse.

Capitolul 4

Ghid de utilizare a aplicației

Utilizarea acestei aplicații constă în urmărirea a câtorva indicații simple. De asemenea, se impun anumite constrângeri cu privire la securitatea aplicației și implicit, protejarea resurselor și a utilizatorilor.

4.1 Autentificarea

Pentru accesul complet și nerestricționat al aplicației, este necesară autentificarea cu un cont valid (Figura 4.1).

Fiecare autentificare cu succes, generează un token valid pentru 24 ore, timp în care utilizatorul poate naviga și folosi resursele disponibile.

După aceste 24 ore, aplicația va elimina accesul utilizatorului de a vedea resursele disponibile de pe site și îi va cere să se autentifice din nou pentru a i se acorda accesul.

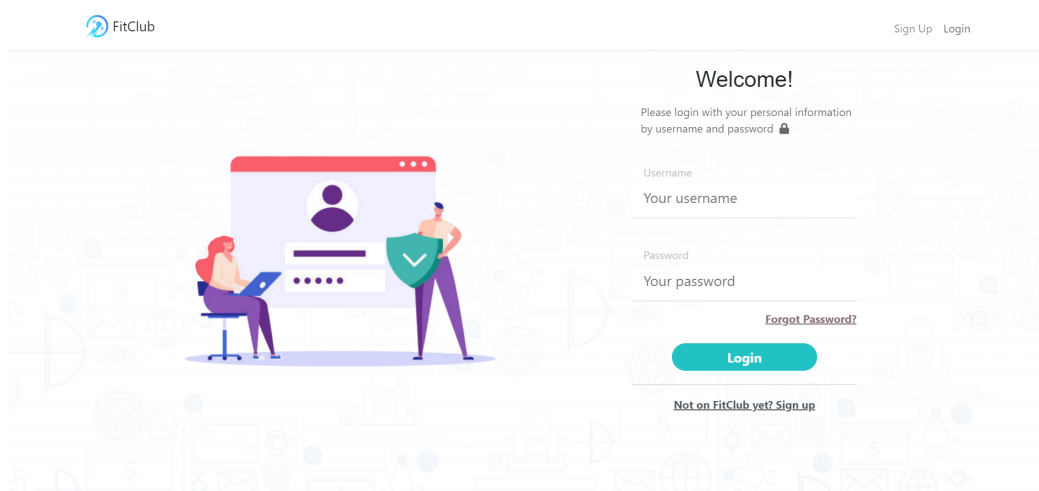



Figura 4.1: Pagina de autentificare

4.2 Înregistrarea

Procedeul de înregistrare (Figura 4.2) a unui utilizator trebuie să respecte câteva reguli cu privire la informațiile furnizate, și anume:

- Numele de utilizator trebuie să fie unic.
- Atât numele de utilizator, cât și numele complet trebuie să aibă cel puțin 4 caractere, și maxim 255 caractere.
- Parola trebuie să aibă cel puțin 8 caractere, și să conțină cel puțin o literă mare, o literă mică, și un caracter special.
- Adresa de email trebuie să fie unică pentru fiecare utilizator în parte și să aibă cel puțin 6 caractere și maxim 255 caractere.

Sign Up Login

Create Account

Display Name
Your display name

Username
Your username

Email


Password
Your password

Password Repeat
Repeat your password

Sign Up

Figura 4.2: Pagina de înregistrare

Pentru a finaliza înregistrarea și a avea acces complet la resursele aplicației, utilizatorul trebuie să confirme adresa de email, accesând link-ul generat automat din cadrul mail-ului pe care îl va primi pe adresa furnizată în momentul înregistrării. (Figura 4.3)

Charles Richard

Confirm your email

Check Your Inbox!

The confirmation of the email is necessary to have access to all the functionalities of the application. A confirmation email has been sent to c.richy@mail.com at account creation. Please access the link inside it to confirm the email address. To submit a new one, click the button below.

Resend Confirmation Email

For assistance, contact FitClub support at:
fitclub.by.alexec@gmail.com

Figura 4.3: Verificarea adresei de email pentru a finaliza înregistrarea

De asemenea, autentificarea se va face automat după înregistrare, însă dacă utilizatorul nu reușește să își confirme adresa de email, atunci, din ca-

uza securității aplicației, nu i se va permite să acceseze resursele disponibile.

Observație: Link-ul transmis prin email este valid doar pentru 24 de ore de la momentul generării.

În cazul în care utilizatorul nu a primit email-ul de confirmare sau acesta a accesat link-ul după ce a expirat, acesta poate solicita încă un email de confirmare care i se va trimite pe adresa furnizată în momentul înregistrării, cu aceeași valabilitate de 24 de ore (Figura 4.4).

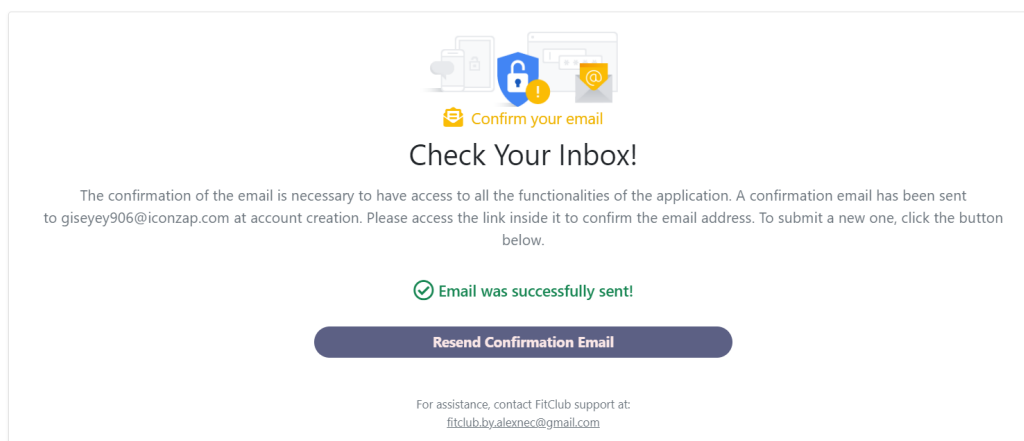


Figura 4.4: Retrimiterrea email-ului de confirmare

Dacă procedeul de înregistrare se sfârșește cu succes, atunci utilizatorului i se va afișa pagina de confirmare (Figura 4.5), și va fi redirecționat în mod automat către pagina principală într-un interval de 5 secunde.

De asemenea, odată ce această verificare a fost finalizată cu succes, nu va mai apărea în cadrul autentificărilor ulterioare.

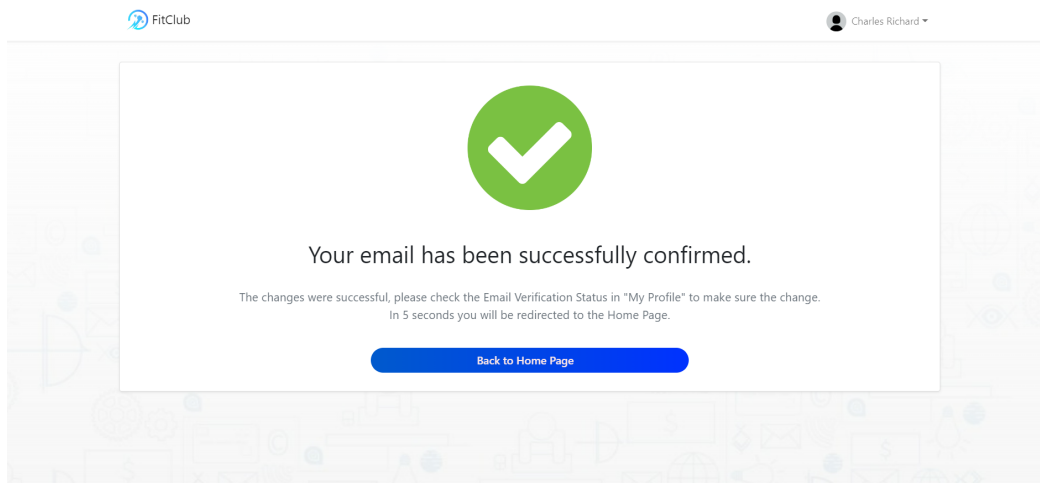


Figura 4.5: Verificarea cu succes a adresei de email

4.3 Navigarea pe site

Pagina principală în care apare fluxul de postări este determinat de postările celor pe care îi urmărește utilizatorul autentificat (Figura 4.6).

Aplicația nu va include postări ale altor utilizatori în acest flux, nici în momentul încărcării postărilor vechi, și nici în momentul încărcării postărilor noi pe pagină.

Utilizatorul are posibilitatea în cadrul acestei pagini de a adăuga o postare făcând click pe chenarul corespunzător unde este afișat indiciul de text: "Share something with your followers".

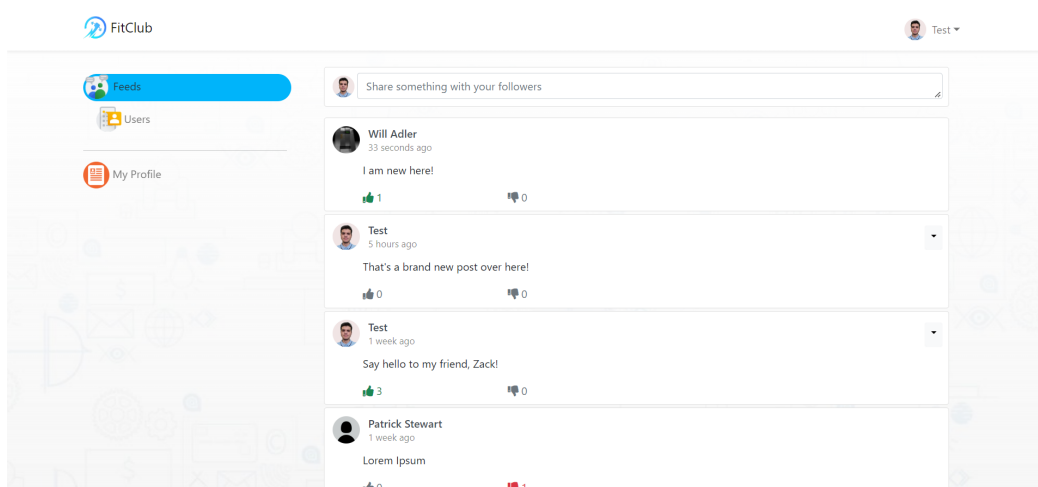


Figura 4.6: Fluxul de postări de pe pagina principală

Atunci când chenarul este focalizat (Figura 4.7), utilizatorului i se vor afișa mai multe opțiuni, și anume să trimită postarea sau să o anuleze.

De asemenea, acesta poate să încarce o imagine sau o animație (fișier cu extensia .gif), făcând click pe butonul "Choose file".

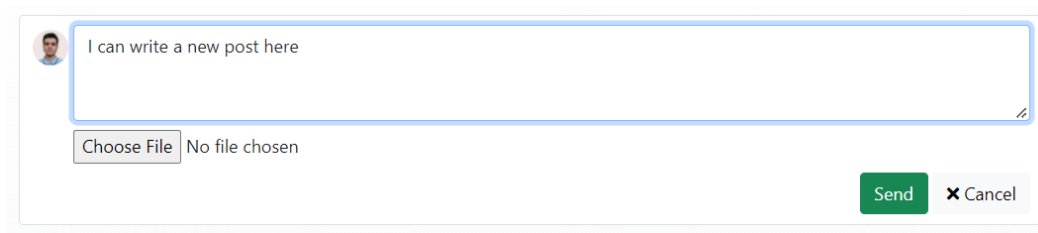


Figura 4.7: Chenarul responsabil pentru scrierea propriu-zisă a postării

Această imagine se va încărca în baza de date printr-o conversie în Base64, iar în momentul afișării pe pagină, aceasta va fi decodată corespunzător și afișată într-un format cât mai prietenos pentru a respecta raportul de aspect al container-ului postării (Figura 4.8).

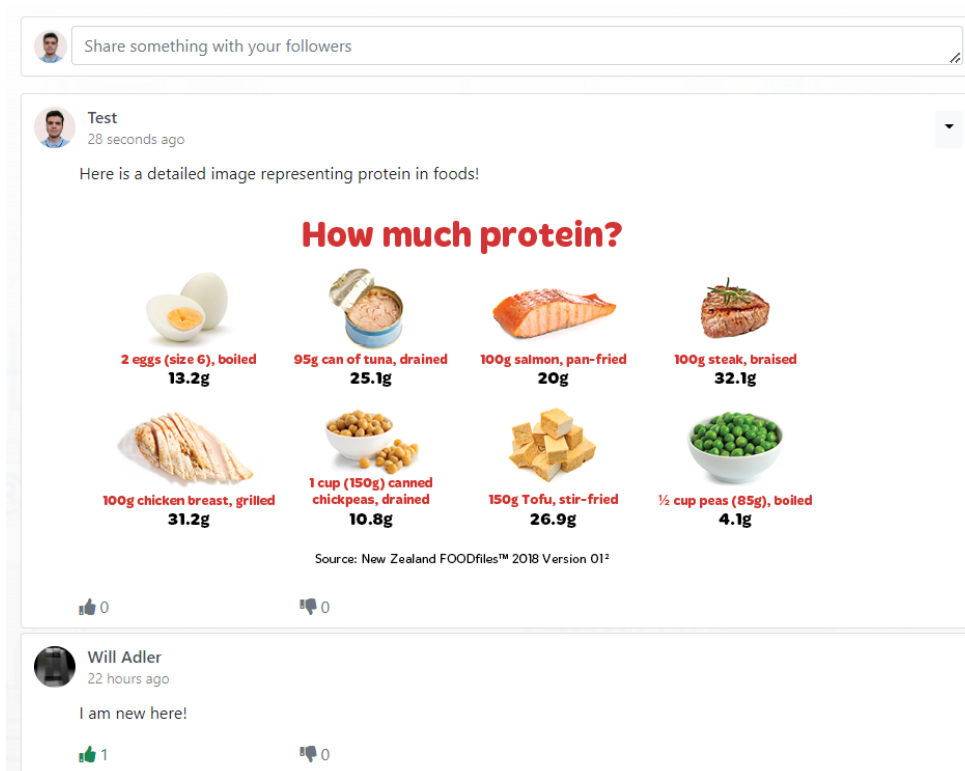


Figura 4.8: Fluxul de postări actualizat cu o postare având o imagine atașată

De asemenea, în cadrul acestei pagini, utilizatorul poate reacționa la postările celor pe care îi urmărește. De exemplu, în Figura 4.8, se poate observa postarea utilizatorului cu numele "Will Adler", în care am reacționat printr-o apreciere.

Pentru a demonstra, totodată, și funcționarea corectă a sistemului de reacții, vom schimba reacția acestei postări (Figura 4.9).

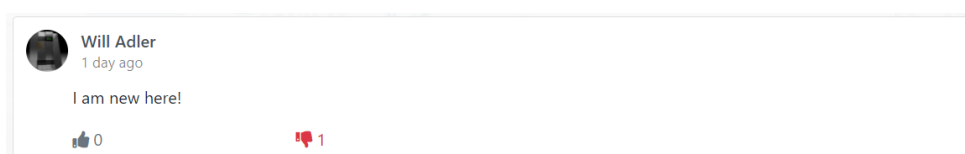


Figura 4.9: Modificarea reacției în cadrul unei postări

Un utilizator autentificat poate să modifice anumite detalii direct de pe pagina sa de profil (Figura 4.10).

În cadrul acestei pagini, i se va afișa și o confirmare a faptului că adresa sa de email a fost confirmată cu succes.

Dacă acesta va opta în viitor să își modifice adresa de email, va fi nevoit să confirme și această adresă pentru a putea accesa în continuare site-ul, fără restricții.

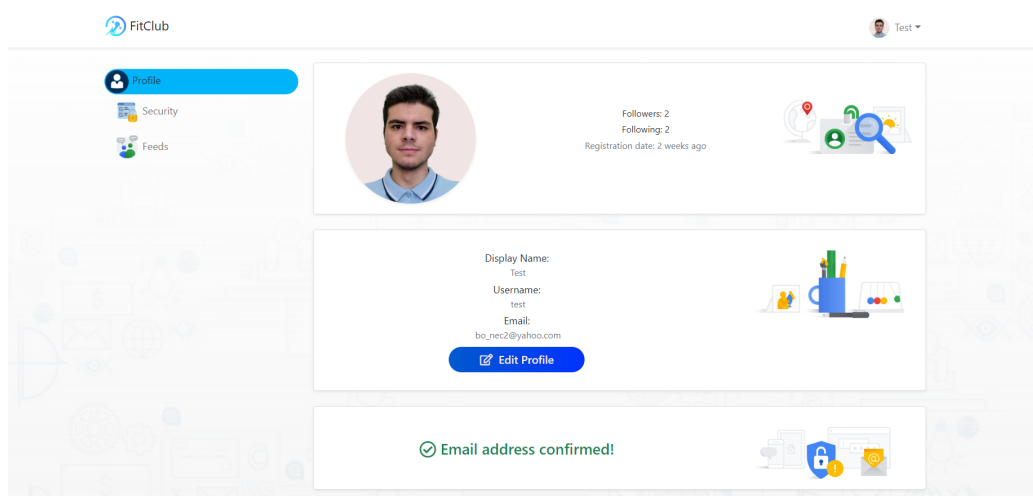


Figura 4.10: Pagina de profil a utilizatorului autentificat

Utilizatorul poate să acceseze și o pagină responsabilă cu revizuirea credențialelor contului său (Figura 4.11). În cadrul acestei pagini, se va putea solicita schimbarea adresei de email, și respectiv a parolei.

În ambele cazuri, înainte de a propaga orice schimbare legată de contul său în baza de date, utilizatorul va trebui să confirme că el acționează în scopul acestei schimbări prin accesarea link-ului inclus în mail-ul pe care îl va primi pe adresa de email confirmată în momentul înregistrării.

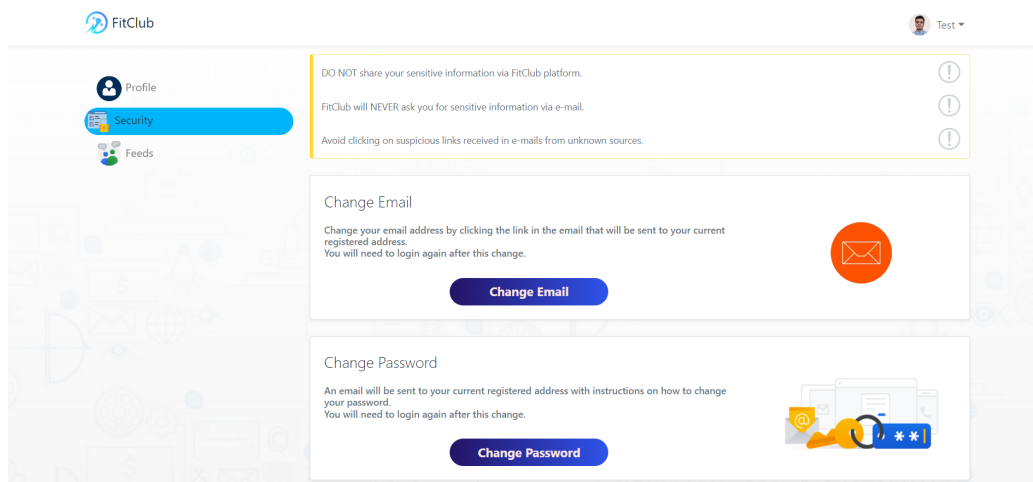


Figura 4.11: Pagina de unde se pot revizui credențialele de securitate

Navigând către pagina de profil a oricărui utilizator, se pot vizualiza strict postările acestuia, și, de asemenea, sistemul de reacții funcționează și de pe această pagină (Figura 4.12).

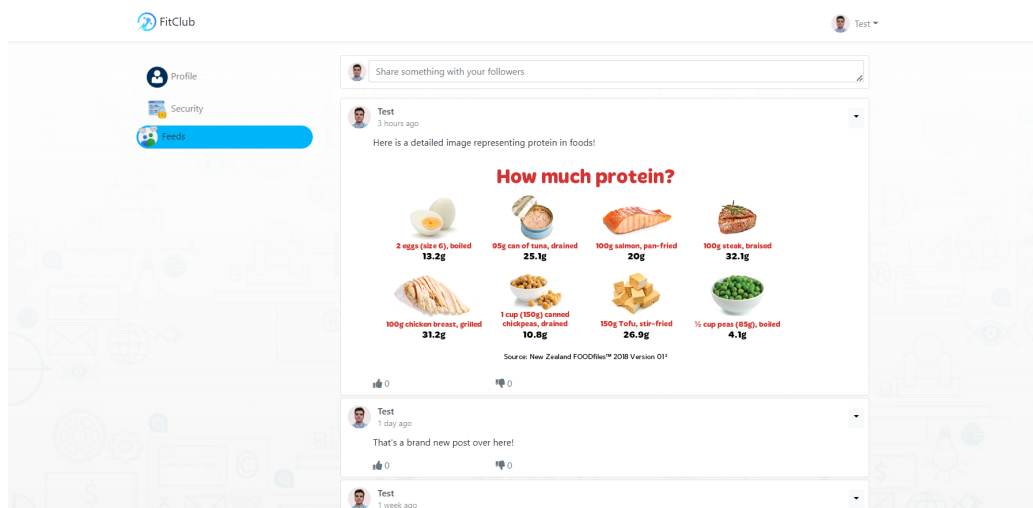


Figura 4.12: Postările de pe pagina de profil a unui utilizator

4.4 Accesarea nepermisă a aplicației

Dacă un utilizator nu este autentificat, atunci acesta va putea să navigheze restricționat pe site, adică nu va putea să vizualizeze fluxul de postări de pe pagina principală, și nici postările specifice unui anumit utilizator (Figura 4.13).

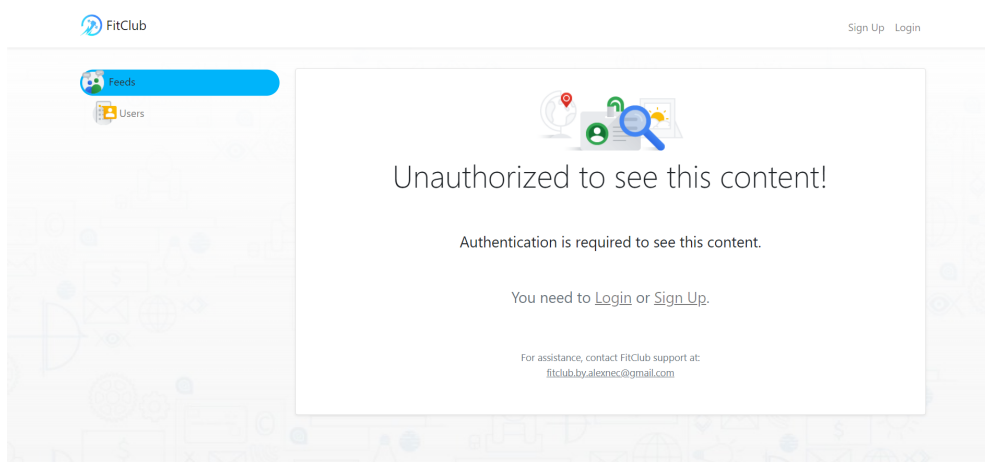


Figura 4.13: Acces restricționat în cazul unui utilizator anonim

Un utilizator anonim va putea să acceseze parțial paginile de profil ale utilizatorilor înregistrați pe site (Figura 4.14). Acest acces parțial se rezumă la posibilitatea de a vedea doar numele de utilizator, și respectiv, numele complet al unui utilizator.

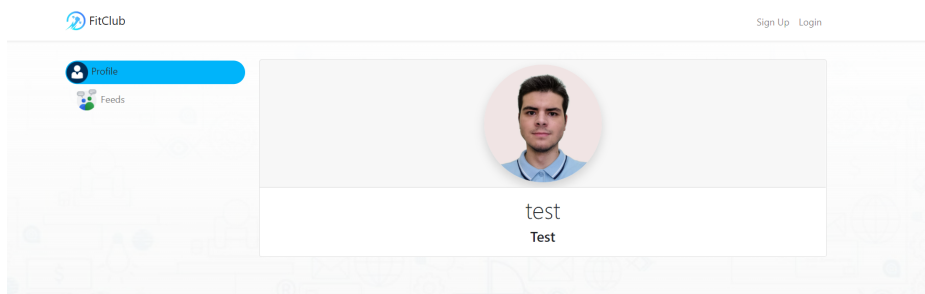


Figura 4.14: Acces parțial la pagina de profil a unui utilizator

De asemenea, același comportament este întâlnit și în cazul în care utilizatorul este autentificat, însă sesiunea acestuia expiră și este nevoit să se autentifice din nou pentru a obține un token valabil pentru încă 24 de ore (Figura 4.15).

Chiar dacă utilizatorul este autentificat, și poate să intre în continuare pe pagina sa de profil, aplicația restricționează orice încercare de a face vreo schimbare referitoare la contul său, aplicația afișându-i un mesaj clar menit să-l îndrume pentru a putea dobândi din nou accesul.

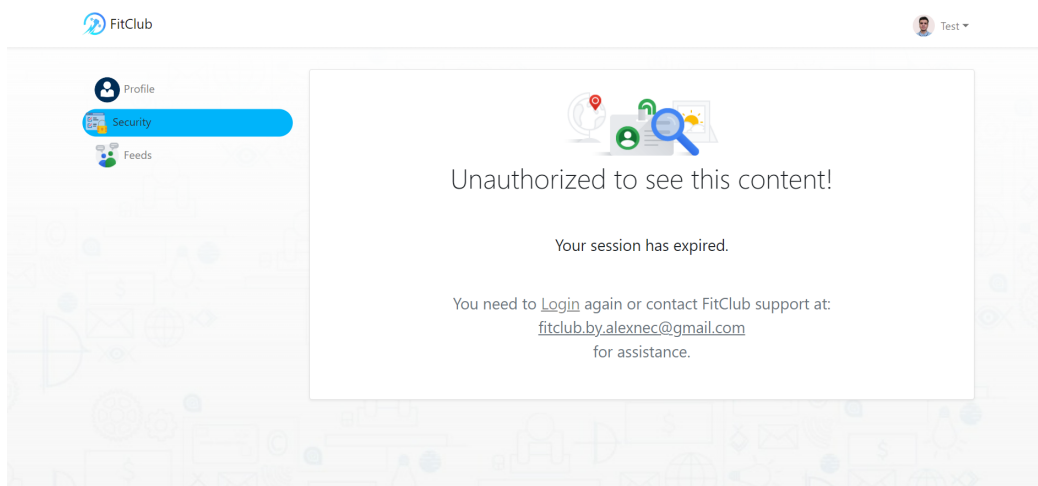


Figura 4.15: Acces restricționat în cazul sesiunii expirate

Capitolul 5

Concluzii

Această lucrare a avut ca scop prezentarea unui prototip de platformă online capabil să conecteze rapid membrii unei săli de fitness în vederea colaborării acestora și pentru a-și împărtăși diverse informații, filmulețe și imagini. Am propus o arhitectură viabilă care să poată gestiona conexiunile utilizatorilor și să trateze corespunzător diverse scenarii care să faciliteze comunicarea în timp real a acestora.

De asemenea, am abordat și aspectele de securitate pentru a proteja utilizatorii și conținutul pe care îl împărtășesc.

Aplicația realizată poate fi îmbunătățită în viitor, atât pentru a-i spori securitatea, cât și pentru a-i simplifica utilizarea. De asemenea, pot fi implementate multe caracteristici noi:

- Înregistrarea să se poată efectua utilizând și alte platforme precum Facebook, Google, etc. (OAuth)
- Utilizatorii să poată adăuga comentarii în cadrul unei postări, acestea putând fi considerate tot postări, dar care dețin o "postare-părinte".
- Posibilitatea de a cita o postare în cadrul unei noi postări.
- Adăugarea unui tab pe pagina de utilizator în cadrul căruia se vor afișa postările preferate ale utilizatorului respectiv.

De asemenea, singura comunicare efectivă între utilizatori este cea reprezentată prin pagina principală de postări. În acest context, se poate adăuga și posibilitatea transmiterii mesajelor private între doi utilizatori pentru a oferi mai multă complexitate și a face comunicarea dintre utilizatori mai ușoară

și rapidă.

În urma dezvoltării acestei aplicații, am reușit să dobândesc noi informații și cunoștințe, dar și să aprofundez cunoștințele pe care le aveam deja. De asemenea, din dorința de a crea un sistem cât mai stabil, am construit și menținut un cod eficient, reutilizabil și ușor de înțeles, astfel încât să sporesc productivitatea și să reduc timpul pierdut în cazul depanării cazurilor de eroare neașteptate.

Bibliografie

- [1] *Spring Framework*, URL: <https://spring.io/projects/spring-framework>.
- [2] *ReactJS*, URL: <https://reactjs.org/docs/getting-started.html>.
- [3] *Development frameworks*, 2005, URL: <https://ieeexplore.ieee.org/abstract/document/1381270>.
- [4] *Spring Microservices in Action*, 2021, URL: https://books.google.ro/books?id=_qUuEAAAQBAJ&printsec=frontcover#v=onepage&q&f=false.
- [5] *Hibernate*, URL: <https://www.springboottutorial.com/hibernate-jpa-tutorial-with-spring-boot-starter-jpa#:~:text=Hibernate%20understands%20the%20mappings%20that,a%20lock%20in%20to%20Hibernate>.
- [6] *Azure for developers overview*, URL: <https://docs.microsoft.com/en-us/azure/developer/intro/azure-developer-overview>.
- [7] *Microsoft Azure - Planning, Deploying, and Managing Your Data Center in the Cloud*, 2015, URL: <https://link.springer.com/book/10.1007/978-1-4842-1043-7?noAccess=true>.
- [8] *Spring Boot*, 2015, URL: https://books.google.ro/books?hl=en&lr=&id=IzkzEAAAQBAJ&oi=fnd&pg=PT11&dq=dependency+spring+book&ots=kILEX6rmDZ&sig=1XEkf8UbeypFhalETbjvOp4cQUk&redir_esc=y#v=onepage&q=dependency%20spring%20%20book&f=false.
- [9] *Advanced Spring JPA*, URL: <https://spring.io/blog/2011/04/26/advanced-spring-data-jpa-specifications-and-querydsl/>.
- [10] *Optimize Java persistence performance in spring boot applications*, 2020, URL: https://books.google.ro/books?hl=en&lr=&id=dIvgDwAAQBAJ&oi=fnd&pg=PR5&dq=java+spring+book+list&ots=ouKs0V44w6&sig=t79XpKL_htDHvBtD4L07ZfleaDQ&redir_esc=y#v=onepage&q=java%20spring%20book%20list&f=false.

- [11] *React Quickly: Painless web apps with React, JSX, Redux*, 2017, URL: https://books.google.ro/books?hl=en&lr=&id=_TgzEAAAQBAJ&oi=fnd&pg=PT23&dq=react+book&ots=aE40-uFt7R&sig=GjICrC5VouK-81Elq2pZjk46kUE&redir_esc=y#v=onepage&q=react%20book&f=false.
- [12] *React Hooks*, URL: <https://reactjs.org/docs/hooks-reference.html>.
- [13] *React.js for the Visual Learner*, 2018, URL: <https://leanpub.com/reactjsforthevisuallearner/read#leanpub-auto-chapter-10--reacting-to-what-weve-learned>.
- [14] *React and React Native: A complete hands-on guide to modern web development with React.js*, 2020, URL: https://books.google.ro/books?hl=en&lr=&id=XCLhDwAAQBAJ&oi=fnd&pg=PP1&dq=react+js+useReducer&ots=Bu8tWzwwiP&sig=-rBhqWDg7wq_F8Z5bKuApkiCen8&redir_esc=y#v=onepage&q=react%20js%20useReducer&f=false.