MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Departamentul de Calculatoare

Social media app

Ernst Robert

Hruban Andrada-Bianca

Popa Alexandru

Grupa: 30236/1

Indrumator de proiect: Marghescu Luminița

Cuprins

1. Introducere

În era digitală contemporană, rețelele de socializare au remodelat modul în care comunicăm, interacționăm și ne exprimăm. Răspunzând acestei evoluții continue, proiectul "YOLO" își propune să redefinească experiența utilizatorului în lumea digitală, combinând familiaritatea și popularitatea platformei Instagram cu inovații unice și o interfață intuitivă.

YOLO, un nume care evocă spontaneitatea și autenticitatea, este construit pe o arhitectură robustă și modernă. Backend-ul său este dezvoltat în Java, folosind Spring, ceea ce asigură o platformă sigură, eficientă și scalabilă. Această alegere tehnologică reflectă angajamentul nostru pentru performanță și fiabilitate, oferind în același timp flexibilitatea necesară pentru adaptarea rapidă la cerințele în schimbare ale mediului de social media.

Pentru front-end, am ales React, un framework JavaScript de ultimă generație, pentru a crea o experiență de utilizator captivantă și receptivă. Prin aceasta, YOLO oferă o interfață curată, ușor de navigat și vizual plăcută, facilitând interacțiunea utilizatorilor cu conținutul și cu ceilalți membri ai comunității.

YOLO nu este doar o aplicație de social media; este o platformă care îmbrățișează creativitatea, conexiunea și expresia de sine. Cu o gamă largă de funcționalități - de la partajarea de fotografii și videoclipuri, la crearea de povești și interacțiunea în timp real - YOLO se adresează nevoii omniprezente de conectare într-o lume tot mai virtuală.

În inima fiecărei platforme de social media, inclusiv a celei a proiectului "YOLO", se află dorința profundă a tinerilor de a rămâne conectați cu prietenii lor și de a-și împărtăși experiențele de zi cu zi. Într-o lume în care comunicarea digitală este la fel de semnificativă ca interacțiunea față în față, YOLO oferă un spațiu unic pentru ca tinerii să se exprime liber și să mențină legătura cu cercul lor social.

Asemenea platformei Instagram, care a revoluționat modul în care tinerii împărtășesc fragmente din viața lor cotidiană prin imagini și videoclipuri, YOLO își propune să meargă și mai departe. Prin funcționalitățile sale avansate, utilizatorii YOLO pot nu doar să posteze conținut, ci și să interacționeze în moduri mai profunde și mai semnificative. Funcția de Povești, de exemplu, permite utilizatorilor să creeze narațiuni captivante despre viața lor cotidiană, oferind astfel o fereastră autentică în experiențele lor zilnice.

Mai mult, YOLO pune un accent deosebit pe comunicarea interactivă. Cu un sistem de mesagerie îmbunătățit și grupuri de discuții, platforma facilitează dialogul constant și partajarea de experiențe între prieteni. Acest lucru nu doar că răspunde nevoii tinerilor de conectare socială, dar creează și un sentiment de comunitate și apartenență.

În plus, YOLO încurajează explorarea și descoperirea. Cu o secțiune dedicată conținutului de interes și un algoritm inteligent care recomandă conținut personalizat, utilizatorii pot descoperi noi interese și pot interacționa cu comunități care partajează aceleași pasiuni. Acest lucru se aliniază cu dorința tinerilor de a explora și a se dezvolta prin experiențe diverse.

În documentația ce urmează, vom explora în detaliu fiecare aspect al aplicației YOLO, de la arhitectura tehnică și implementarea funcționalităților, până la strategiile de securitate și planurile de dezvoltare viitoare. Vom analiza, de asemenea, impactul și locul YOLO în peisajul actual al social media, subliniind cum această platformă reușește să se distingă într-un domeniu atât de competitiv.

2. Limbaje de programare

2.1. Java

Java, unul dintre limbajele de programare fundamentale pentru dezvoltarea aplicației YOLO, joacă un rol crucial în crearea și menținerea unei infrastructuri back-end robuste și eficiente. Acest limbaj orientat pe obiecte este binecunoscut pentru portabilitatea, scalabilitatea și performanța sa, fiind o alegere excelentă pentru o platformă de social media dinamică și interactivă precum YOLO.

Portabilitate și Scalabilitate: Java este cunoscut pentru sloganul său, "Write Once, Run Anywhere" (WORA), ceea ce înseamnă că codul scris în Java poate fi rulat pe orice dispozitiv care suportă Java Virtual Machine (JVM). Acest lucru este esențial pentru YOLO, care urmărește să ofere o experiență uniformă pe diverse platforme și dispozitive. Scalabilitatea oferită de Java asigură că YOLO poate gestiona eficient creșterea numărului de utilizatori și a traficului de rețea.

Securitate: Java oferă un set robust de caracteristici de securitate, care sunt esențiale pentru protejarea datelor utilizatorilor YOLO. Mecanismele de securitate încorporate în Java ajută la prevenirea atacurilor cum ar fi SQL Injection și Cross-Site Scripting (XSS), care sunt critice pentru o aplicație de social media unde securitatea datelor și a informațiilor personale este o prioritate.

Performanță și Eficiență: Cu ajutorul gestionării eficiente a memoriei și al colectorului de gunoi (Garbage Collector), Java asigură că resursele sunt utilizate în mod optim, ceea ce este vital pentru o aplicație care procesează un volum mare de date și interacțiuni în timp real. De asemenea, performanța Java a fost îmbunătățită semnificativ în ultimele versiuni, ceea ce o face ideală pentru backend-ul unei aplicații precum YOLO.

Folosind Java pentru back-end-ul YOLO, proiectul beneficiază de un limbaj de programare matur, cu performanțe înalte și o comunitate vastă. Aceasta permite ca YOLO să fie o platformă sigură, eficientă și ușor de întreținut, capabilă să evolueze și să se adapteze în funcție de nevoile în continuă schimbare ale utilizatorilor săi.

2.2. 

TypeScript, un superset al JavaScript, aduce o serie de îmbunătățiri semnificative și caracteristici de tipizare statică în dezvoltarea aplicației YOLO. Utilizarea lui TypeScript în combinație cu React pentru dezvoltarea front-end a YOLO oferă avantaje distincte, contribuind la crearea unui cod mai robust, mai ușor de întreținut și mai sigur.

Tipizare Statică și Siguranța la Compilare: Unul dintre principalele avantaje ale TypeScript este sistemul său de tipizare statică. Acest lucru permite dezvoltatorilor să definească tipuri de date pentru variabile, parametri și structuri de date. Prin aceasta, erorile sunt detectate în timpul compilării în loc de a fi descoperite la runtime, ceea ce crește semnificativ calitatea și siguranța codului în aplicația YOLO.

Claritate și Scalabilitate în Cod: Codul scris în TypeScript este adesea mai clar și mai ușor de înțeles, deoarece tipurile de date oferă informații suplimentare despre cum este destinat să fie folosit codul. Aceasta facilitează colaborarea în cadrul unei echipe mari de dezvoltatori și ajută la menținerea și scalarea codului pe măsură ce aplicația YOLO crește și se dezvoltă.

Integrare cu React și Ecosistemul JavaScript: TypeScript se integrează fără probleme cu React și alte biblioteci sau cadre JavaScript. Acest lucru înseamnă că dezvoltatorii pot beneficia de avantajele TypeScript (cum ar fi tipizarea puternică și tooling-ul avansat) fără a renunța la ecosistemul bogat și flexibilitatea pe care JavaScript o oferă.

Integrarea TypeScript în stack-ul tehnologic al YOLO aduce claritate, eficiență și robustețe în procesul de dezvoltare front-end. Avantajele sale în tipizare, siguranță la compilare și tooling ajută la construirea unei baze solide pentru o aplicație de social media modernă, fiabilă și ușor de extins.

3. Gestionarea Datelor și Baza de Date

3.1. SQL

SQL (Structured Query Language) este un limbaj standardizat pentru gestionarea bazelor de date relaționale și nu se încadrează în categoria tradițională a limbajelor de programare precum Java sau TypeScript. În schimb, SQL este folosit pentru interogarea, actualizarea și manipularea datelor dintr-o bază de date.

SQL joacă un rol cheie în gestionarea datelor în aplicația YOLO. Prin intermediul acestui limbaj, echipa de dezvoltare poate interoga și manipula eficient baza de date, asigurând că informațiile sunt stocate, actualizate și recuperate în mod eficient. SQL permite realizarea de interogări complexe, cum ar fi căutarea postărilor, filtrarea conținutului pe baza anumitor criterii și gestionarea relațiilor între diferite entități de date, cum ar fi utilizatorii și postările acestora.

Folosirea SQL în YOLO facilitează scalabilitatea, permițând aplicației să gestioneze o cantitate crescândă de date fără a compromite performanța. De asemenea, SQL susține o menținere ușoară a bazei de date, permițând actualizări și modificări eficiente pe parcursul dezvoltării aplicației.

3.2.NOSQL

NoSQL (Not Only SQL) reprezintă o categorie de sisteme de management al bazelor de date care oferă un mecanism de stocare și recuperare a datelor modelate în moduri diferite de tabelele relaționale tradiționale. Acest tip de bază de date este ideal pentru anumite aspecte ale unei aplicații moderne de social media, cum ar fi YOLO.

NoSQL este cunoscut pentru flexibilitatea sa în ceea ce privește structurile de date. Poate gestiona date structurate, semi-structurate și nestructurate, cum ar fi documente JSON, grafuri sau cheie-valoare. Această flexibilitate este esențială pentru YOLO, care poate necesita stocarea unei varietăți largi de tipuri de date, inclusiv postări textuale, imagini, videoclipuri și metadate complexe.

Bazele de date NoSQL sunt proiectate pentru a oferi o scalabilitate orizontală excelentă, ceea ce le face ideale pentru aplicații care trebuie să gestioneze volume mari de date și trafic intens, caracteristici comune pentru platformele de social media. Scalabilitatea oferită de NoSQL poate contribui la menținerea unei performanțe ridicate a aplicației YOLO, chiar și pe măsură ce baza de utilizatori crește.

4.IDE

IntelliJ IDEA: Este un IDE avansat dezvoltat de JetBrains, optimizat pentru dezvoltarea în Java. Este cunoscut pentru eficiența sa în codarea rapidă, analiza de cod de înaltă calitate, refactoring puternic și integrare excelentă cu diverse sisteme de build și versiune. IntelliJ IDEA sprijină dezvoltatorii YOLO printr-o interfață intuitivă și un set impresionant de funcționalități, cum ar fi suportul pentru Spring Boot, gestionarea eficientă a dependențelor și integrarea cu sistemele de control al versiunilor.

WebStorm: De asemenea dezvoltat de JetBrains, WebStorm este un IDE puternic pentru dezvoltarea front-end, inclusiv suport pentru TypeScript. Oferă funcții avansate precum autocompletarea codului, navigarea ușoară în proiecte, și instrumente de debugging. Pentru echipa YOLO, WebStorm facilitează dezvoltarea eficientă și organizată a interfeței utilizatorului, permițând o integrare ușoară cu biblioteci precum React.

pgAdmin: Este un instrument de administrare open-source pentru PostgreSQL, unul dintre cele mai populare sisteme de gestionare a bazelor de date relaționale. pgAdmin oferă o interfață grafică puternică pentru gestionarea bazelor de date PostgreSQL, facilitând crearea, modificarea și interogarea bazelor de date SQL. Este ideal pentru echipa YOLO în gestionarea, monitorizarea și optimizarea bazelor de date SQL.

MongoDB Compass: Este un instrument GUI oficial pentru MongoDB, un sistem popular de baze de date NoSQL. Compass permite vizualizarea și interogarea datelor, gestionarea indexurilor și performanța interogărilor, și oferă o viziune clară asupra structurii bazei de date. Acesta este un instrument esențial pentru echipa YOLO pentru a interacționa eficient cu datele NoSQL, optimizând și simplificând procesul de dezvoltare.

5.Framework

1. Spring pentru Backend:

Spring :Este un framework extensiv pentru dezvoltarea aplicațiilor Java, care simplifică procesul de configurare și dezvoltare a serviciilor backend. În contextul YOLO, Spring oferă o platformă robustă și eficientă, facilitând crearea rapidă a microserviciilor și a API-urilor REST.

Unul dintre principalele avantaje ale Spring este capacitatea sa de a simplifica configurarea inițială a proiectelor. Aceasta include configurarea automată a componentelor și integrarea simplificată cu baze de date și alte servicii externe, ceea ce este crucial pentru scalabilitatea și adaptabilitatea YOLO.

Spring vine cu suport încorporat pentru securitatea aplicațiilor, inclusiv autentificarea și autorizarea. Acest lucru asigură că datele utilizatorilor și interacțiunile în YOLO sunt gestionate într-un mod sigur. De asemenea, framework-ul sprijină performanța optimă și gestionarea eficientă a resurselor.

2. React pentru Frontend:

Este un framework JavaScript dezvoltat de Facebook, utilizat pentru construirea interfețelor de utilizator, în special pentru aplicații single-page precum YOLO. React se remarcă prin capacitatea sa de a crea interfețe de utilizator interactive și performante.

React introduce un model bazat pe componente, permițând dezvoltatorilor să creeze blocuri de interfață reutilizabile. Acest lucru îmbunătățește consistența și eficiența dezvoltării interfeței utilizatorului YOLO.

Foloseste Virtual DOM pentru a optimiza actualizările interfeței, ceea ce asigură o experiență de utilizator fluidă și responsivă, esențială pentru o platformă de social media modernă.

6. Arhitectura și Organizarea Codului

În inima dezvoltării aplicației YOLO se află o abordare sofisticată și meticuloasă a organizării codului, reflectând cele mai înalte standarde de inginerie software. Am adoptat o structură bimodală de repository-uri Git, segregând în mod strategic codul sursă al backend-ului și frontend-ului. Această decizie nu numai că optimizează fluxurile de lucru ale dezvoltatorilor, dar și asigură o claritate arhitecturală și facilitează o integrare armonioasă între cele două componente esențiale ale aplicației.

GIT REPO:

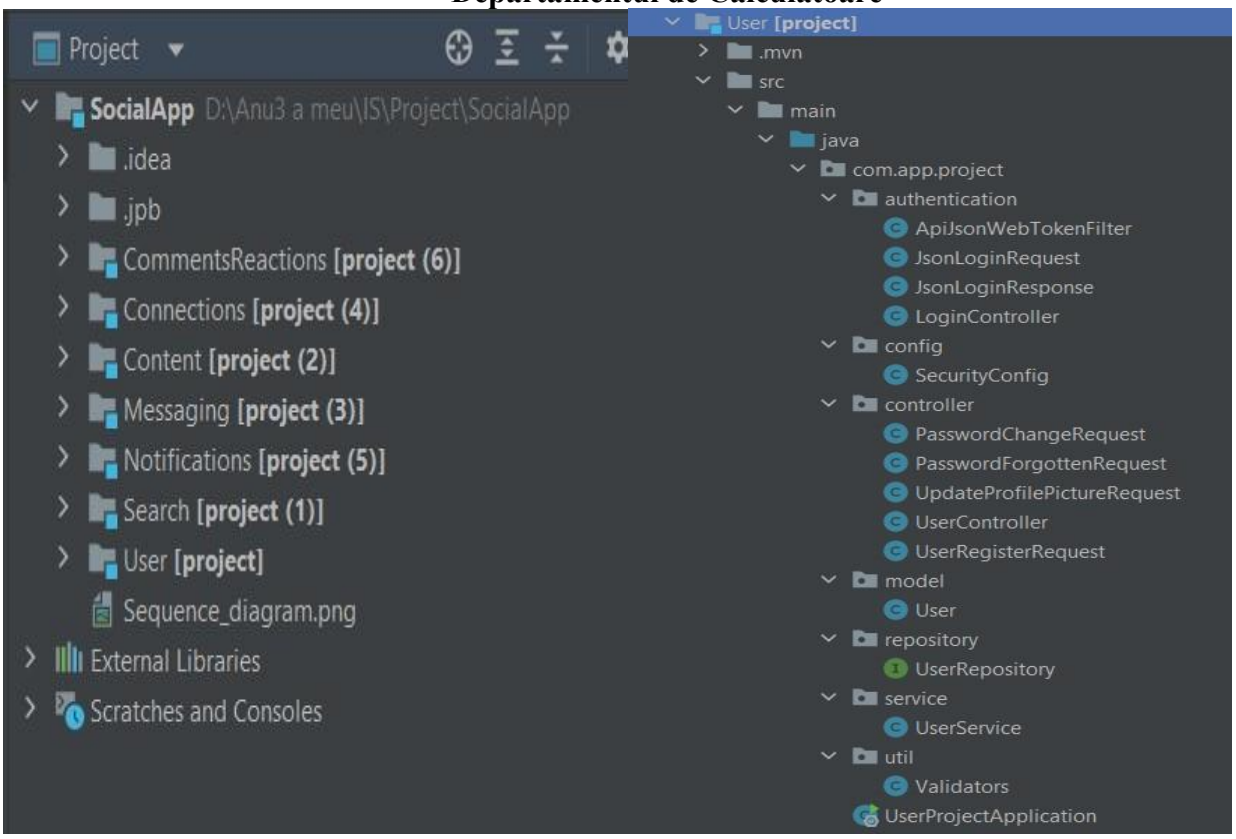
Backend https://github.com/Popa24/SocialApp

Frontend https://github.com/6ernst9/social-media-front

1. Backend - Un Ansamblu de Microservicii Spring:

- Backend-ul YOLO, o simfonie de funcționalități complexe, este organizat în mai multe proiecte Spring, fiecare dedicat unei funcționalități principale. Această abordare modulară, centrată pe microservicii, permite o scalabilitate elegantă și o gestionare eficientă a codului.

  Fiecare microserviciu este o entitate auto-suficientă, proiectată să îndeplinească o funcție specifică, de la gestionarea utilizatorilor până la procesarea datelor. Această segregare meticuloasă asigură nu doar un grad înalt de coeziune și acuratețe funcțională, dar și o adaptabilitate remarcabilă în fața cerințelor evolutive ale aplicației.



2. Frontend - O Viziune Separată, dar Sincronizată:

- Frontend-ul YOLO, dezvoltat cu React, rezidă într-un repository Git separat. Această separare strategică a responsabilităților între frontend și backend permite echipei noastre să se concentreze pe optimizarea experienței utilizatorului, asigurându-se că interfața este nu doar estetic plăcută, dar și funcțional coerentă cu logica de afaceri a backend-ului.

Această organizare duală și sofisticată a codului sursă în YOLO ilustrează angajamentul nostru pentru excelență tehnică și eficiență operațională. Prin această structură, garantăm nu doar o arhitectură de cod curată și ușor de întreținut, dar și un cadru solid pentru evoluția continuă și inovația în cadrul aplicației noastre de social media.

7. Functionalități

7.1. Login and Register

Autentificarea (Login) în aplicația YOLO este proiectată să fie atât sigură, cât și accesibilă. Folosind cele mai recente practici în materie de securitate, cum ar fi criptarea datelor și autentificarea bazată pe token, ne asigurăm că informațiile de autentificare ale utilizatorilor sunt protejate în mod corespunzător. Acest proces este vital pentru păstrarea confidențialității și securității conturilor utilizatorilor. Ne-am concentrat pe crearea unei experiențe de autentificare fără probleme, permițând utilizatorilor să acceseze conturile lor rapid și eficient. Interfața simplificată și instrucțiunile clare asigură că procesul de login este intuitiv, chiar și pentru utilizatorii noi.

Înregistrarea (Register) pentru un nou cont în YOLO este un proces simplu și direct. Utilizatorii sunt ghidați prin pași ușor de urmărit pentru a crea un cont nou, cu o minimă cerință de informații necesare pentru a începe. Aceasta include detalii de bază precum numele utilizatorului, adresa de email și o parolă sigură. Pentru a asigura autenticitatea și securitatea conturilor, YOLO implementează un sistem de validare și verificare a datelor în timpul procesului de înregistrare. Acest lucru include verificarea adresei de email pentru a confirma identitatea utilizatorului și pentru a preveni înregistrările false sau duplicate. Imediat după înregistrare, utilizatorii sunt încurajați să își personalizeze profilurile, adăugând informații suplimentare și preferințe. Aceasta îmbunătățește experiența utilizatorului și ajută la crearea unei comunități mai conectate și mai personalizate în YOLO. Implementarea sistemului de autentificare și înregistrare în YOLO este realizată cu o atenție deosebită la securitate, eficiență și experiența utilizatorului. Următoarele puncte descriu în detaliu aceste aspecte:

1. **Autentificare Securizată (Login):**

Filtru JWT (JSON Web Token): Utilizăm **ApiJsonWebTokenFilter** pentru a valida accesul la API-uri. Acest filtru verifică existența și validitatea tokenului JWT în antetul HTTP al fiecărei cereri, asigurând că accesul la resursele protejate este permis doar utilizatorilor autentificați.

```java
@Override
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
    HttpServletRequest httpRequest = (HttpServletRequest) request;
    HttpServletResponse httpResponse = (HttpServletResponse) response;

    String requestURI = httpRequest.getRequestURI();

    if (excludedPaths.stream().anyMatch(requestURI::startsWith)) {
        chain.doFilter(request, response);
        return;
    }

    String authHeader = httpRequest.getHeader( s: "Authorization");

    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
        httpResponse.sendError(HttpServletResponse.SC_UNAUTHORIZED, s: "Missing or invalid Authorization header");
        return;
    }

    String token = authHeader.substring( beginIndex: 7);

    try {
        LoginController.decodeJWT(token);
        chain.doFilter(request, response);
    } catch (Exception e) {
        httpResponse.sendError(HttpServletResponse.SC_UNAUTHORIZED, s: "Invalid token");
    }
}
```

Gestionarea Tokenurilor: Procesul de login din **LoginController** generează un token JWT, care este atribuit utilizatorului după autentificarea cu succes. Acest token asigură o sesiune sigură și este folosit pentru toate cererile ulterioare către server.

```java
no usages  Popa +1
@PostMapping("/api/login")
public Mono<ResponseEntity<JsonLoginResponse>> login(@RequestBody @NonNull final JsonLoginRequest request) {
    User user = new User();
    BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    if (request.getEmail() != null) {
        user = userService.findByEmail(request.getEmail());
    } else if (request.getUsername() != null) {
        user = userService.findByUsername(request.getUsername());
    } else if (request.getPhoneNumber() != null) {
        user = userService.findByPhoneNumber(request.getPhoneNumber());
    }

    if (user != null && passwordEncoder.matches(request.getPassword(), user.getPassword())) {
        String token = String.valueOf(createJWT(user.getUserId().toString(), user.getEmail(), ttlMillis: 999999999));
        return Mono.just(ResponseEntity.ok(new JsonLoginResponse(token, user)));
    } else {
        return Mono.just(ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(null));
    }
}

3 usages  Popa
public static String createJWT(String id, String subject, long ttlMillis) {
    SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;

    long nowMillis = System.currentTimeMillis();
    Date now = new Date(nowMillis);
```

2. **Înregistrare Flexibilă și Sigură (Register):**

Prin UserController, oferim utilizatorilor posibilitatea de a se înregistra folosind adresa de email, numărul de telefon sau numele de utilizator. Codul nostru asigură că toate datele sunt validate corespunzător înainte de a crea un nou cont.

Securitatea Parolei: Folosim **BCryptPasswordEncoder** pentru a cripta parolele utilizatorilor, asigurând astfel că datele sensibile sunt stocate în siguranță și protejate împotriva accesului neautorizat.

```
no usages  alex popa +3
@PostMapping("/register")
public Mono<ResponseEntity<JsonLoginResponse>> saveUser(@RequestBody @NonNull final UserRegisterRequest request) {
    var user= userService.saveUserRegister(request);
    String token = String.valueOf(createJWT(user.getUserId().toString(), user.getEmail(), ttlMillis: 999999999));
    return Mono.just(ResponseEntity.ok(new JsonLoginResponse(token,user)));
}
```

3. **Gestionarea Avansată a Contului Utilizatorului:**

Update și Ștergere: Permite utilizatorilor să actualizeze sau să șteargă contul lor, oferind control complet asupra informațiilor personale.

4. **Configurația de Securitate:**

SecurityConfig: Configurăm și personalizăm securitatea aplicației, stabilind ce rute sunt excluse din autentificare, pentru a permite accesul neîngrădit la serviciile de înregistrare și resetare a parolei.

```
@Bean
public FilterRegistrationBean<ApiJsonWebTokenFilter> apiJsonWebTokenFilterRegistrationBean() {
    FilterRegistrationBean<ApiJsonWebTokenFilter> registrationBean = new FilterRegistrationBean<>();
    ApiJsonWebTokenFilter apiJsonWebTokenFilter = new ApiJsonWebTokenFilter();
    registrationBean.setFilter(apiJsonWebTokenFilter);
    registrationBean.addUrlPatterns("/api/*");
    registrationBean.addInitParameter( name: "excludedPaths",  value: "/api/login,/api/user/register,/api/user/{userId}/change-passwo

    return registrationBean;
}
```

7.2. Forgot and Change Password

Funcționalitățile de recuperare și schimbare a parolei sunt componente esențiale în YOLO, asigurând că utilizatorii pot gestiona în mod eficient accesul la conturile lor. Aceste mecanisme nu numai că îmbunătățesc securitatea, dar oferă și o experiență de utilizare mai flexibilă și mai accesibilă.

Recuperarea Parolei (Forgot Password): Această funcție permite utilizatorilor să-și reseteze parola în cazul în care o uită. Procesul este conceput pentru a fi simplu și direct, protejând totodată securitatea contului. Un email de resetare a parolei este trimis la adresa asociată contului, prin care utilizatorul este ghidat în pașii necesari pentru a-și seta o nouă parolă.

Schimbarea Parolei (Change Password): Utilizatorii au opțiunea de a-și schimba parola ori de câte ori consideră necesar. Aceasta este o practică de securitate recomandată, în special dacă există suspiciuni privind securitatea contului. Schimbarea periodică a parolei ajută la protejarea contului împotriva accesului neautorizat.

Detalii Implementare:

1. **Recuperarea Parolei:**

În **UserController**, metoda **forgotPassword** inițiază procesul de resetare a parolei. Aceasta generează un link de resetare și îl trimite prin email utilizatorului.

```
no usages  AndradaHr +1
@GetMapping("/forgot-password/{userId}")
public void forgotPassword(
        @PathVariable Long userId, @RequestBody PasswordForgottenRequest request){
    userService.forgotPassword(userId, request.getEmail());
}
no usages  AndradaHr
@PostMapping("/forgot-password/{userId}")
public void forgotPasswordChange(
        @PathVariable Long userId, @RequestBody PasswordChangeRequest request){
    userService.forgotPasswordChange(userId, request.getNewPassword());
}
```

Metoda **sendEmailForForgottenPassword** din **UserService** gestionează crearea și trimiterea emailului de resetare. Link-ul de resetare conduce utilizatorul la o pagină unde poate introduce noua parolă.

```
2 usages  AndradaHr
private void sendEmailForForgottenPassword(User user, boolean forgotOrChange) {
    if (forgotOrChange == true) {
        String forgettenPasswordLink = "http://localhost:8080/api/user/forgot-password/" + user.getUserId();
```

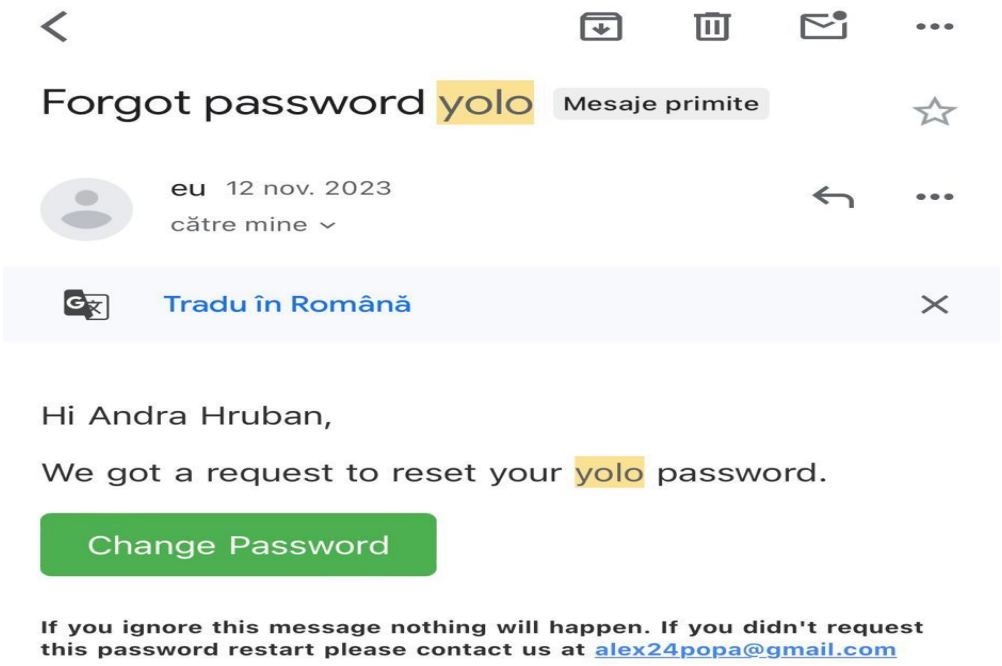
2. **Schimbarea Parolei:**

Utilizatorii pot solicita schimbarea parolei prin metoda **changePassword** din **UserController**. Aceasta necesită verificarea parolei actuale pentru a asigura că solicitarea vine de la proprietarul legitim al contului.

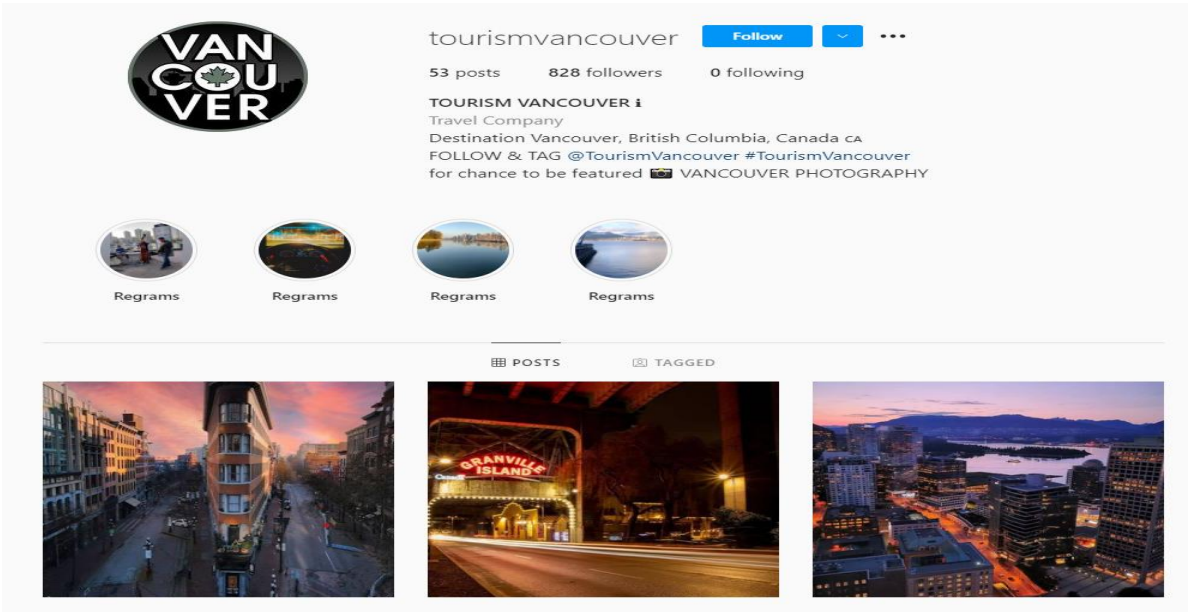
```
no usages  AndradaHr +1
@GetMapping("/{userId}/change-password")
public void changePassword(
        @PathVariable Long userId){

    userService.changePassword(userId);
}
no usages  AndradaHr +1
@PostMapping("/{userId}/change-password")
public void changePassword(
        @PathVariable Long userId, @RequestBody PasswordChangeRequest request){

    userService.changePasswordChange(userId, request.getOldPassword(), request.getNewPassword());
}
```

În **UserService**, metoda **changePasswordChange** se ocupă de validarea parolei actuale și aplicarea noii parole. Acest proces implică criptarea noii parole înainte de a o actualiza în baza de date, asigurând astfel păstrarea în siguranță a datelor sensibile.

```
1 usage  Popa +1
public void changePasswordChange(Long userId, String oldPassword, String newPassword) {
    BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
    User user = userRepository.findById(userId)
            .orElseThrow(() -> new RuntimeException("User not found"));

    if (passwordEncoder.matches(user.getPassword(), oldPassword)) {
        user.setPassword(passwordEncoder.encode(newPassword));
        userRepository.save(user);
    }
}
```


Forgot password yolo
Mesaje primite
eu 12 nov. 2023
către mine
Tradu în Română
Hi Andra Hruban,
We got a request to reset your yolo password.
Change Password
If you ignore this message nothing will happen. If you didn't request this password restart please contact us at alex24popa@gmail.com

7.3. Profile


tourismvancouver
Follow
53 posts
828 followers
0 following
TOURISM VANCOUVER
Travel Company
Destination Vancouver, British Columbia, Canada ca
FOLLOW & TAG @TourismVancouver #TourismVancouver
for chance to be featured VANCOUVER PHOTOGRAPHY
Regrams
Regrams
Regrams
Regrams
POSTS
TAGGED
GRANVILLE ISLAND

Funcționalitatea de profil în YOLO este una dintre cele mai esențiale caracteristici ale platformei, oferind utilizatorilor un spațiu personalizat pentru a-și exprima identitatea și pentru a interacționa cu alți utilizatori.

Utilizatorii pot personaliza profilurile lor prin adăugarea sau modificarea diferitelor elemente, cum ar fi o fotografie de profil, o biografie, date personale și setări de confidențialitate. Aceasta le permite să-și creeze o identitate unică în cadrul comunității YOLO.

Profilurile oferă o vedere asupra activității utilizatorului, inclusiv postările, interacțiunile și conexiunile cu alți membri ai rețelei. Utilizatorii pot vizualiza și interacționa cu profilurile altor membri, consolidând astfel relațiile sociale în cadrul platformei.

Detalii Implementare:

1. **Structura Modelului de Utilizator:**

Clasa **User** în pachetul **model** definește structura de bază a profilului utilizatorului, cu câmpuri pentru nume, email, parolă, date personale și preferințe.

Acest model este folosit pentru a stoca și a recupera informații despre utilizatori din baza de date.

2. **Funcționalități de Gestionare a Profilului:**

În **UserController**, metodele precum **updateUser** și **updateProfilePicture** permit utilizatorilor să-și actualizeze informațiile de profil și fotografia de profil.

Acest controller gestionează cererile HTTP pentru actualizarea profilului, asigurându-se că modificările sunt reflectate corespunzător în baza de date.

```
no usages  AndradaHr +1
@PutMapping("/update")
public ResponseEntity<User> updateUser(@RequestBody User user) {
    return ResponseEntity.ok(userService.updateUser(user));
}

no usages  AndradaHr +1
@DeleteMapping("/delete/{userId}")
public ResponseEntity<?> deleteUser(@PathVariable Long userId) {
    userService.deleteUser(userId);
    return ResponseEntity.ok().build();
}

no usages  popaa
@PatchMapping("/updateProfilePicture")
public ResponseEntity<?>updateProfilePicture(@RequestBody UpdateProfilePictureRequest request){
    var user= userService.findById(request.id).block();
    user.setProfilePicture(request.photoLink);
    userService.saveUser(user);
    return ResponseEntity.ok().build();
}
```

3. **Securitate și Confidențialitate:**

Utilizatorii au opțiunea de a-și seta profilul ca privat, permițând astfel controlul asupra cine poate vizualiza informațiile lor personale și activitățile pe platformă.

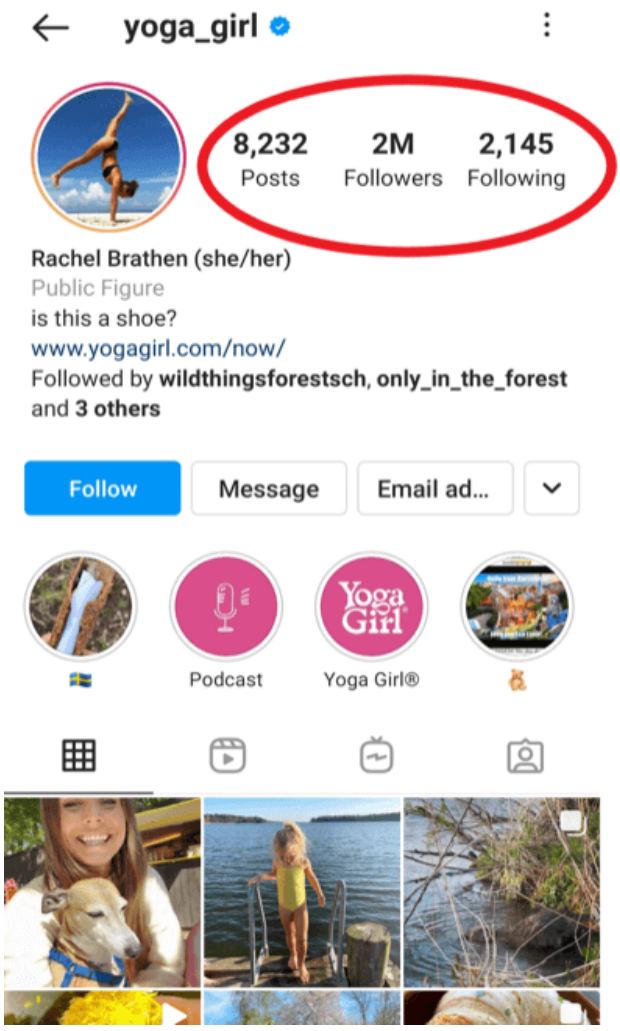
Sistemul de autentificare și autorizare asigură că numai utilizatorii autorizați pot accesa și modifica profilurile.

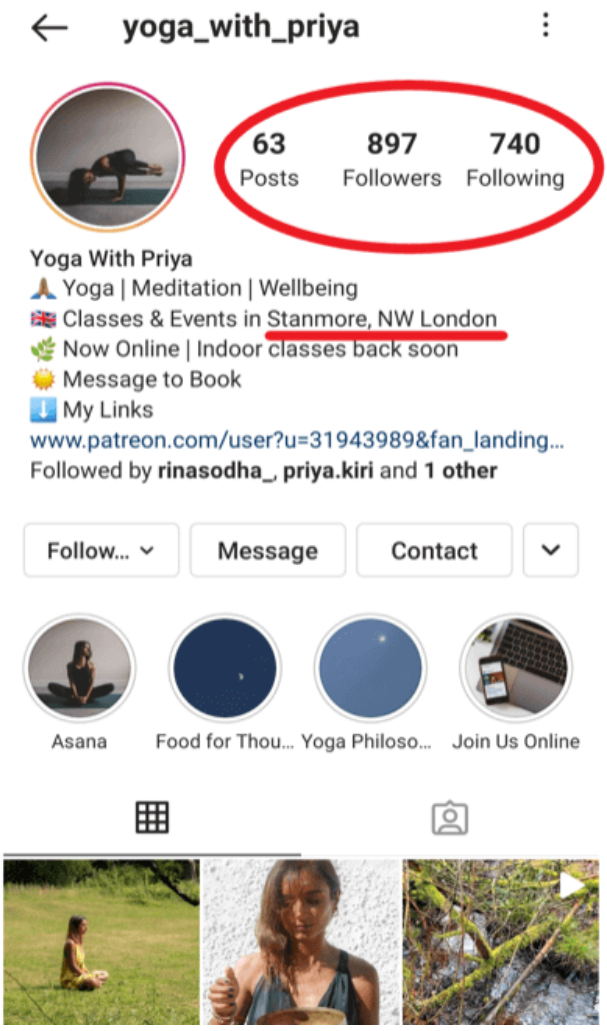
7.4. Followers and Following

Funcționalitățile de urmărire (following) și de a avea urmăritori (followers) sunt esențiale pentru orice platformă de social media, inclusiv YOLO. Acestea permit utilizatorilor să se conecteze și să interacționeze unii cu alții, creând o rețea dinamică și interactivă.

(Following): Utilizatorii pot "urmari" alți utilizatori pentru a vedea actualizările și postările lor în fluxul de știri. Aceasta facilitează descoperirea de conținut nou și interesant și permite utilizatorilor să rămână conectați cu persoanele și subiectele care îi interesează.

(Followers): Urmăritorii sunt utilizatorii care aleg să urmărească activitatea unui anumit utilizator. A avea urmăritori permite utilizatorilor să-și extindă influența și să ajungă la un public mai larg.





Detalii Implementare:

1. **Modelul de Conexiune:**

Clasa **Connection** reprezintă conexiunile unui utilizator, conținând informații despre utilizatorii pe care îi urmărește (**following**).

Această abordare permite o gestionare eficientă a conexiunilor și urmăririi activității între utilizatori.

2. **Gestionarea Urmăririi și Urmăritorilor:**

Controller-ul **ConnectionController** gestionează adăugarea și ștergerea conexiunilor. Metodele **addFollowing** și **removeFollowing** permit utilizatorilor să înceapă sau să înceteze urmărirea altor conturi.

```
no usages  popaa +1
@PutMapping("/{userId}/follow")
public ResponseEntity<?> addFollowing(@PathVariable Long userId, @RequestBody Long followingId) {
    userService.addFollowing(userId, followingId);
    return ResponseEntity.ok().build();
}

no usages  popaa +1
@GetMapping("/{userId}/following")
public ResponseEntity<List<Long>> getFollowing(@PathVariable Long userId) {
    List<Long> followingIds = userService.getFollowingByUserId(userId);
    return ResponseEntity.ok(followingIds);
}
```

Metoda **getFollowing** oferă o listă a tuturor utilizatorilor pe care un utilizator îi urmărește, facilitând accesul la rețeaua sa socială.

```
no usages  popaa +1
@DeleteMapping("/{userId}/unfollow")
public ResponseEntity<?> removeFollowing(@PathVariable Long userId, @RequestBody Long followingId) {
    userService.removeFollowing(userId, followingId);
    return ResponseEntity.ok().build();
}

no usages  popaa +1
@GetMapping("/getSuggestedFriends/{userId}")
public Flux<SuggestedFriendsResponse> getSuggestedFriends(@PathVariable Long userId) {
    return userService.getSuggestedFriends(userId);
}

no usages  AndradaHr
@GetMapping("/getFirstTimePostsAndRecommendations")
public Flux<User> getNoFollowingSuggestedFriends() { return userService.getRandomUsers(); }
```

3. **Sugestii de Prieteni și Recomandări:**

Metoda **getSuggestedFriends** din **ConnectionController** generează sugestii de prieteni pe baza algoritmilor de rețea, ajutând utilizatorii să descopere noi conexiuni posibile.

4. **Securitate și Confidențialitate:**

Filtrul **ApiJsonWebTokenFilter** asigură că numai utilizatorii autentificați pot accesa și modifica datele legate de urmărirea altor utilizatori, protejând astfel confidențialitatea și integritatea informațiilor de pe platformă.

În implementarea aplicației YOLO, **WebClient** joacă un rol crucial în facilitarea comunicării asincrone și non-blocante între diferite componente ale aplicației, cum ar fi microserviciile backend sau serviciile externe. Aici este cum WebClient este integrat și utilizat în YOLO:

```
@Bean
public WebClient webClient() {
    var httpClient = createHttpClient();

    return WebClient
            .builder()
            .defaultHeader(HttpHeaders.ACCEPT, MediaType.APPLICATION_XML_VALUE)
            .clientConnector(new ReactorClientHttpConnector(httpClient))
            .build();
}
```

1. Configurarea WebClient:

În **WebClientConfig**, se configurează instanța de **WebClient**. Aceasta implică setarea unor parametri precum timpul de răspuns maxim și acceptarea anumitor tipuri de media (de exemplu, **MediaType.APPLICATION_XML_VALUE**).

Configurarea include și setarea unui client HTTP personalizat cu un context SSL, ceea ce este esențial pentru securizarea comunicațiilor.

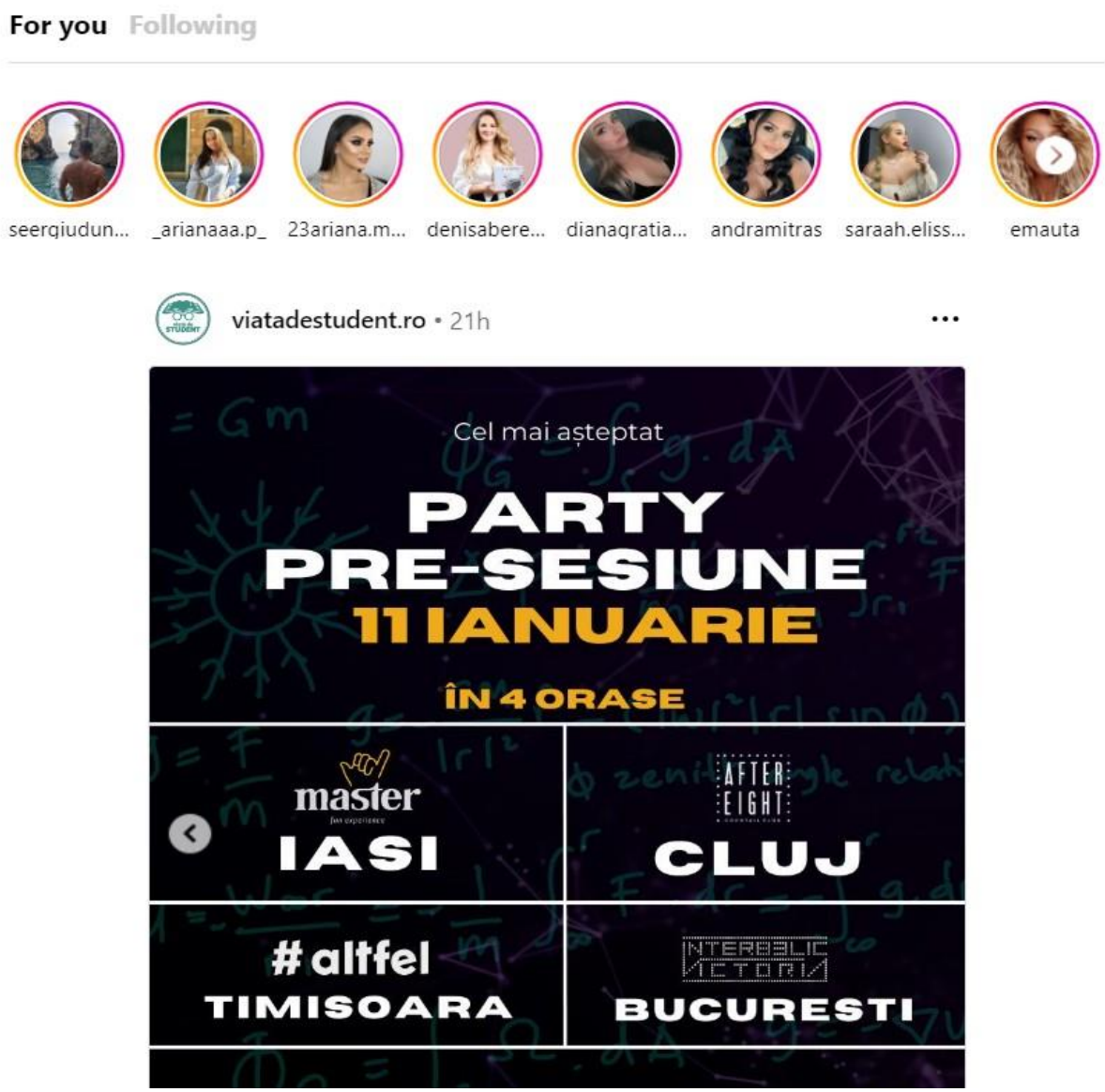
2. Utilizarea WebClient în Interacțiuni Asincrone:

WebClient este folosit pentru a efectua cereri HTTP asincrone către backend sau alte servicii. De exemplu, în **ConnectionController**, s-a utilizat WebClient pentru a prelua informații despre utilizatori si a trimis datele actualizate ale urmăririi utilizatorilor.

3. Gestionarea Cererilor și Răspunsurilor:

Prin intermediul WebClient, cererile și răspunsurile sunt gestionate în mod asincron, ceea ce înseamnă că aplicația poate continua să execute alte operațiuni în timp ce așteaptă răspunsul. Acest lucru este esențial pentru îmbunătățirea performanței și scalabilității aplicației.

7.5. Content : Post, feed, Story



1. Postări:

Utilizatorii pot crea postări prin încărcarea de imagini și adăugarea de descrieri. Aceasta permite exprimarea personală și partajarea experiențelor, gândurilor sau activităților. Postările permit altor utilizatori să interacționeze prin aprecieri, comentarii și partajări, creând o experiență socială activă. Utilizatorii pot controla cine vede postările lor prin setări de confidențialitate, permițând o experiență personalizată în funcție de preferințele fiecărui utilizator.

2. Feed-ul Utilizatorului:

Feed-ul prezintă o colecție de postări de la utilizatorii urmăriți și sugestii personalizate, asigurând că utilizatorii sunt mereu la curent cu ultimele noutăți din rețeaua lor. Feed-ul sugerează noi postări și utilizatori pe baza intereselor și interacțiunilor anterioare, încurajând descoperirea și explorarea de noi conținuturi. Conținutul feed-ului este actualizat constant, oferind utilizatorilor un flux continuu de conținut relevant și interesant.

3. Povești:

Poveștile permit utilizatorilor să împărtășească momente efemere, care sunt vizibile pentru o perioadă limitată de timp (24 de ore), adăugând un sentiment de imediat și spontaneitate experienței de partajare. Poveștile sunt prezentate într-un format vizual atractiv, ideal pentru partajarea experiențelor rapide și captivante. Poveștile pot include elemente interactive, cum ar fi sondaje sau întrebări, încurajând angajamentul și interacțiunea cu urmăritorii.

1. Postări (Content):

ContentController

Responsabil pentru gestionarea postărilor.

- **getFeedPosts** și **getSuggestedPosts** sunt metode care returnează postările pentru feed-ul unui utilizator și sugestii de postări, respectiv.

ContentService

- Interacționează cu **ContentRepository** pentru a prelua postări din baza de date.
- **getPosts** returnează postările unui utilizator și ale prietenilor săi.
- **getSuggestedPosts** oferă postări sugerată pe baza algoritmului intern.

ContentRepository

- Este un JPA Repository care facilitează interacțiunea cu baza de date pentru entitatea **Content**.
- **findByUserId** este o metodă personalizată pentru a găsi postările unui utilizator specific.

Modelul Content

- Reprezintă structura de bază a unei postări, cu câmpuri cum ar fi **userId**, **photo**, **datePosted**, și **description**.

```
@Autowired
private ContentService contentService;

no usages  popaa
@GetMapping("/getFeedPosts/{userId}")
public Flux<ContentResponse> getFeedPosts(@PathVariable Long userId) { return contentService.getPosts(userId); }

no usages  popaa
@GetMapping("/getSuggestedPosts/{userId}")
public Flux<ContentResponse> getSuggestedPosts(@PathVariable Long userId) {
    return contentService.getSuggestedPosts(userId);
}
```

2. Feed-ul Utilizatorului:

- Logica de feed este încorporată în **ContentController** și **ContentService**.
- Feed-ul este populat cu postări de la utilizatorii urmăriți, plus postări sugerate.
- Interacțiunea cu alte servicii, cum ar fi obținerea listei de prieteni, se face prin **WebClient**, demonstrând o abordare de arhitectură bazată pe microservicii.

3. Povești (Stories):

StoryController

- Gestionează preluarea poveștilor pentru un utilizator.
- **getFeedStories** și **getRandomFeedStories** sunt metode pentru obținerea poveștilor din feed și, respectiv, povești aleatorii.

StoryService

- Similar cu **ContentService**, dar pentru povești.
- Interacționează cu **StoryRepository** pentru a prelua povești.
- Filtrarea poveștilor mai vechi de 24 de ore pentru a păstra natura efemeră a poveștilor.

StoryRepository

- Este un JPA Repository pentru entitatea **Story**.
- **findByUserId** pentru a găsi poveștile postate de un utilizator specific.

Modelul Story

- Reprezintă structura unei povești, cu câmpuri precum **storyId**, **userId**, **photo**, **datePosted**.

```
@Autowired
private StoryService storyService;

no usages  popaa
@GetMapping("/getFeedStories/{userId}")
public Flux<StoryResponse> getFeedStories(@PathVariable Long userId) { return storyService.getStories(userId); }

no usages  popaa
@GetMapping("/getRandomFeedStories")
public Flux<StoryResponse> getRandomFeedStories() { return storyService.getRandomStories(); }
```

Securitate și Autorizare:

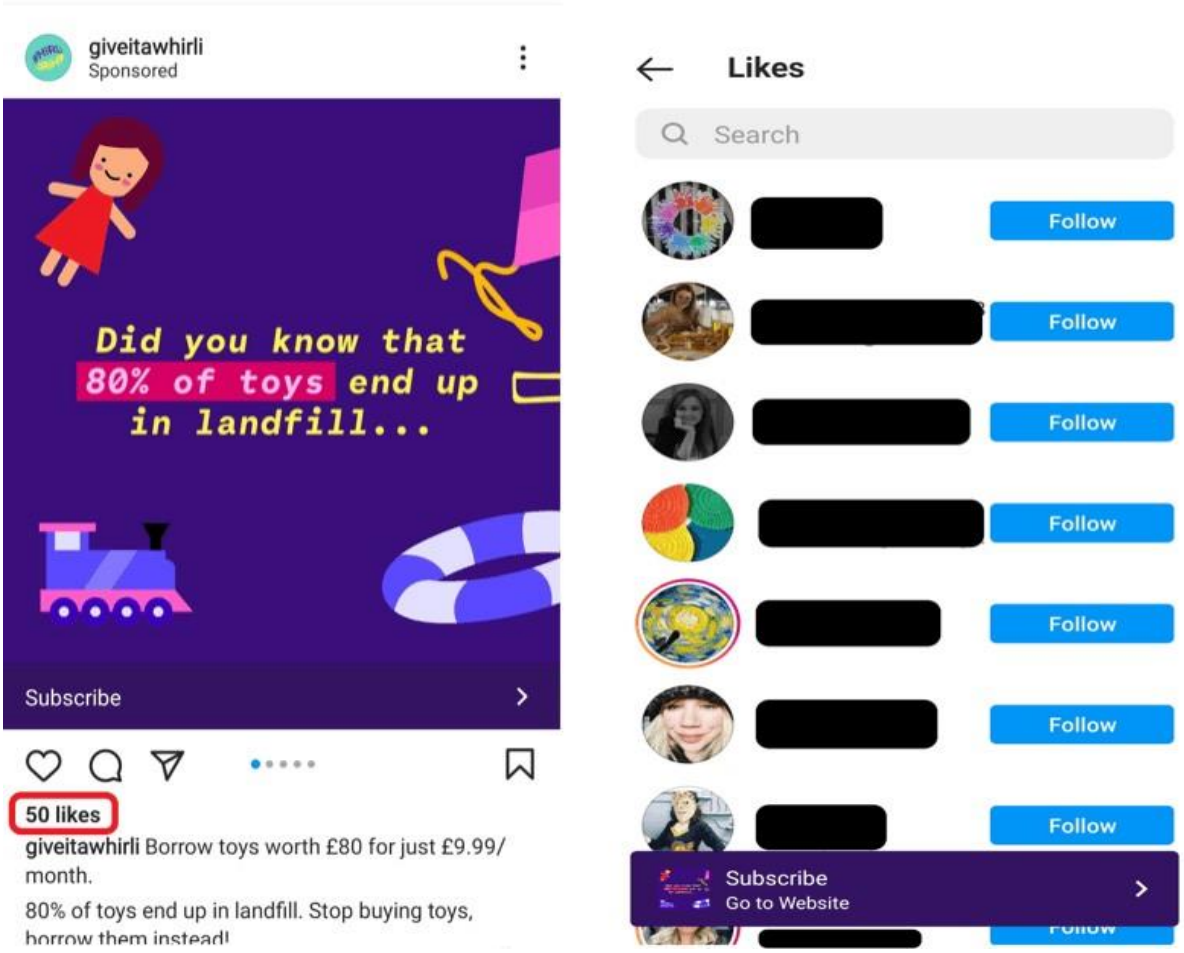
ApiJsonWebTokenFilter

- Filtru de securitate pentru a valida tokenurile JWT.
- Asigură că numai cererile autorizate au acces la endpoint-urile protejate.

Concluzie Implementare:

- Arhitectura aplicației YOLO este construită în jurul principiilor de securitate, modularitate și scalabilitate.
- Utilizarea tehnologiilor Spring Boot, JPA, și Reactor Netty (**WebClient**) demonstrează o abordare modernă și eficientă.
- Filtrarea securizată a JWT și structura microserviciilor oferă un nivel solid de securitate și împarte logic aplicația în componente gestionabile.

7.6. Likes



În inima oricărei platforme de social media, inclusiv în aplicația YOLO, se află capacitatea utilizatorilor de a interacționa prin "likes" (aprecieri) și comentarii. Aceste două elemente joacă un rol vital în crearea unei experiențe sociale bogate și interactive.

1. Likes (Aprecierea Postărilor):

Un "like" este o modalitate rapidă și eficientă prin care utilizatorii pot arăta aprecierea pentru o postare. Este un gest simplu, dar puternic, care transmite un mesaj pozitiv de aprobare sau de bucurie. În multe cazuri, un "like" poate înlocui nevoia de a scrie un comentariu, permițând utilizatorilor să-și exprime sentimentele fără a folosi cuvinte. Numărul de "likes" poate influența algoritmul de afișare a postărilor în feed-ul utilizatorilor, promovând conținutul popular și creând o experiență personalizată.

2. Comments (Comentariile):

Comentariile permit utilizatorilor să intre în discuții, să pună întrebări și să ofere feedback detaliat. Acestea creează oportunități pentru conversații mai profunde și interacțiuni semnificative. Prin comentarii, utilizatorii pot construi relații și comunități în jurul intereselor comune. Comentariile pot duce la noi prietenii și conexiuni în cadrul platformei. Comentariile aduc diverse perspective și viziuni, îmbogățind conținutul inițial și oferind o experiență mai dinamică și variată utilizatorilor.

Integrarea în Aplicația YOLO:

- Ușor de Accesat și de Utilizat: Atât butoanele de "like", cât și secțiunile de comentarii sunt proiectate pentru a fi ușor accesibile în interfața aplicației, facilitând interacțiunea rapidă și eficientă.
- Notificări și Feedback: Utilizatorii primesc notificări când postările lor sunt apreciate sau comentate, stimulând astfel un ciclu continuu de interacțiune și implicare în comunitatea YOLO.

Implementarea funcționalităților de "likes" și comentarii în YOLO implică o abordare bazată pe servicii și repository-uri pentru a gestiona interacțiunile utilizatorilor cu postările. Utilizarea MongoRepository pentru stocarea și accesarea datelor reacțiilor și metodele din ReactionService și ReactionController asigură un proces eficient și scalabil de gestionare a acestor interacțiuni fundamentale în cadrul aplicației sociale.

1. Likes (Aprecierea Postărilor)

ReactionService - Gestionarea Aprecierea Postărilor:

Detalii implementare:

1. **Metoda addLike (Adăugarea unui Like) si Metoda removeLike (Ștergerea unui Like) - ReactionService**

addLike: Scopul acestei metode este de a adăuga un "like" pentru un post specific.

Când un utilizator apasă butonul de like, **addLike** este invocată cu **postId** și **userLikes** (ID-ul utilizatorului care a dat like).

Metoda caută prima dată o reacție existentă pentru postul dat. Dacă găsește una și utilizatorul curent nu a dat încă un like, adaugă ID-ul utilizatorului în lista de likes și salvează modificările în baza de date.

removeLike: Această metodă este responsabilă pentru eliminarea unui "like" de la un post specific. Ea este apelată atunci când un utilizator care a dat deja like unui post decide să își retragă aprecierea.

```
1 usage   AndradaHr
public void addLike(Long postId, Long userLikes) {
    Reaction reaction = reactionRepository.findByPostId(postId).orElse( other: null);
    if (reaction != null && !reaction.getUserLikes().contains(userLikes)) {
        reaction.getUserLikes().add(userLikes);
        reactionRepository.delete(reaction);
        reactionRepository.save(reaction);
    }
}

1 usage   AndradaHr
public void removeLike(Long postId, Long userLike) {
    Reaction reaction = reactionRepository.findByPostId(postId).orElse( other: null);
    if (reaction != null && reaction.getUserLikes().contains(userLike)) {
        reaction.getUserLikes().remove(userLike);
        reactionRepository.delete(reaction);
        reactionRepository.save(reaction);
    }
}
```

2. **Metoda getLikesByPost (Obținerea Numărului de Likes) - ReactionService:**

Această metodă returnează toate likes-urile pentru un post specific.

Când se dorește afișarea numărului de likes pentru un post, **getLikesByPost** este apelată.

Metoda interoghează baza de date pentru a găsi reacția asociată cu postul și returnează lista de ID-uri ale utilizatorilor care au dat like.

```
1 usage   AndradaHr
public void addReaction(Long postId, List<Long> userLikes) {
    Reaction user = new Reaction();
    user.setPostId(postId);
    user.setUserLikes(userLikes);
    reactionRepository.save(user);
}

1 usage   AndradaHr
public List<Long> getLikesByPost(Long postId) {
    return reactionRepository.findByPostId(postId) Optional<Reaction>
            .map(Reaction::getUserLikes) Optional<List<Long>>
            .orElse(Collections.emptyList());
}
```

2. ReactionRepository - Interfață pentru Gestionarea Reacțiilor

ReactionRepository extinde **MongoRepository** și include o metodă pentru a găsi reacții asociate cu un anumit post.

3. Modelul Reaction - Structura Datelor pentru Reacții

Clasa **Reaction** definește structura de date pentru stocarea reacțiilor utilizatorilor la postări.

4. ReactionController - Controler pentru Gestionarea Reacțiilor

ReactionController gestionează cererile HTTP pentru adăugarea de noi reacții la postări.

7.7. Direct Messiging

Direct Messaging (mesageria directă) reprezintă o componentă esențială a experienței de social media, oferind utilizatorilor o cale privată și personală de comunicare. În cadrul aplicației YOLO, această funcționalitate joacă un rol crucial în aprofundarea relațiilor dintre utilizatori și îmbogățirea experienței de socializare.

1. MessageService - Serviciul Principal pentru Gestionarea Mesajelor

- **getMessagesBetweenUsers**: Această metodă returnează toate mesajele dintre doi utilizatori, identificați prin ID-urile lor (senderId și receiverId). Este esențială pentru afișarea istoricului conversațiilor.
- **updateMessageReadStatus**: Actualizează starea de citire a unui mesaj specific. Este folosită pentru a marca mesajele ca fiind citite de către receptor.
- **updateMessageContent**: Permite modificarea conținutului unui mesaj, utilă în cazul în care utilizatorii doresc să editeze mesajele trimise.
- **deleteMessage**: Oferă posibilitatea de a șterge un mesaj, adăugând un nivel suplimentar de control utilizatorilor asupra conversațiilor lor.
- **saveMessage**: Salvează un nou mesaj în baza de date. Este esențială pentru crearea și trimiterea de noi mesaje.

```
1 usage  popaa +1
public Message updateMessageReadStatus(Long messageId, Boolean isRead) {
    Message message = messageRepository.findById(messageId)
            .orElseThrow(() -> new EntityNotFoundException("Message not found"));
    message.setIsRead(isRead);
    return messageRepository.save(message);
}

1 usage  popaa +2
public Message updateMessageContent(Long messageId, String newContent) {
    Message message = messageRepository.findById(messageId)
            .orElseThrow(() -> new EntityNotFoundException("Message not found"));
    message.setContent(newContent);
    message.setIsEdited(true);
    return messageRepository.save(message);
}

1 usage  popaa
public void deleteMessage(Long messageId) { messageRepository.deleteById(messageId); }

1 usage  popaa +1
public Message saveMessage(Message message) { return messageRepository.save(message); }
```

2. MessageRepository - Interfață JPA pentru Accesul la Date

- Implementează metode personalizate precum **findBySenderAndReceiver**, care ajută la extragerea mesajelor dintre doi utilizatori, facilitând astfel construirea unui flux de conversație.

3. Modelul Message - Structura de Date pentru Mesaje

- Include atribute cum ar fi ID-ul mesajului, sender, receiver, conținutul mesajului, timestamp-ul, starea de citire și dacă mesajul a fost editat.

4. WebSocketController - Controler pentru Gestionarea Mesajelor în Timp Real

- Implementează endpoint-uri WebSocket pentru diverse acțiuni, cum ar fi trimiterea, actualizarea și ștergerea mesajelor.
- Folosește **@MessageMapping** pentru a defini puncte de acces pentru mesaje și **@SendTo** pentru a redirecționa mesajele către un topic specific.

```
no usages  popaa
@Controller
@CrossOrigin
public class WebSocketController {

    5 usages
    private final MessageService messageService;

    no usages  popaa
    @Autowired
    public WebSocketController(MessageService messageService) { this.messageService = messageService; }
```

5. Configurația WebSocket - Setarea Broker-ului de Mesaje și Endpoint-urilor

- Configurația WebSocket definește broker-ul de mesaje și endpoint-urile pentru STOMP (Simple Text Oriented Messaging Protocol).
- **registerStompEndpoints** definește endpoint-ul **/chat** pentru conexiunile WebSocket și activează SockJS pentru compatibilitatea cu browserele care nu suportă WebSocket nativ.

Importanța Implementării

Implementarea Direct Messaging în YOLO permite utilizatorilor să interacționeze în mod privat și instant, crescând gradul de angajament și satisfacție în utilizarea aplicației. Utilizarea WebSocket și STOMP asigură o comunicare în timp real, vitală pentru o experiență de chat fluidă și interactivă. Prin MessageService, aplicația gestionează eficient operațiunile pe mesaje, oferind utilizatorilor posibilitatea de a gestiona conversațiile lor într-un mod dinamic și personalizat.

8. Tehnici Complexe de Programare

Proiectul YOLO integrează tehnici avansate de programare și tehnologii moderne pentru a asigura o experiență de utilizare fluidă și o arhitectură robustă. Printre aceste tehnici, se numără utilizarea Hibernate, Lombok, WebClient și WebSocket, care contribuie la eficiența și scalabilitatea aplicației.

1. Hibernate

- Rol: Hibernate, ca parte a JPA (Java Persistence API), este utilizat pentru a facilita mapearea obiectelor Java la baza de date, permițând gestionarea eficientă a datelor.
- Implementare în YOLO: Hibernate simplifică operațiunile CRUD (Create, Read, Update, Delete) și reduce complexitatea interogărilor de baze de date, cum ar fi în MessageRepository pentru extragerea mesajelor între utilizatori.

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:postgresql://localhost:5432/IsProject
spring.datasource.username=postgres
spring.datasource.password=1410
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.data.jpa.repositories.bootstrap-mode=default
spring.mail.properties.mail.smtp.starttls.enable=true
mailgun.apikey=513eec4c933709bb149ce43a6e2e9d6c-8c9e82ec-7fd20ff7
mailgun.domain=sandboxc88fb7eb763145139b7e312a51af007a.mailgun.org
server.error.include-stacktrace=never
```

2. Lombok

- Rol: Lombok este un instrument care automatizează scrierea codului boilerplate în Java, cum ar fi metodele getter, setter, constructori și altele.
- Implementare în YOLO: Folosind Lombok, modelul de clasă, cum ar fi Message și User, devine mai curat și mai ușor de întreținut, prin eliminarea necesității de a scrie manual cod redundant.

```
import jakarta.persistence.*;
import lombok.*;

import java.time.LocalDate;

Ernst Robert +3
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "\"user\"")
@Entity
```

3. WebClient

- Rol: WebClient este un client HTTP non-blocant și reactiv, care oferă un mod eficient de a efectua cereri HTTP/HTTPS asincrone.
- Implementare în YOLO: În serviciul ContentService, WebClient este utilizat pentru a face interogări asincrone la alte microservicii sau la endpoint-uri externe, asigurând un transfer de date eficient și gestionarea elegantă a răspunsurilor.

4. WebSocket și STOMP

- Rol: WebSocket oferă o conexiune bidirecțională și în timp real între client și server. STOMP (Simple Text Oriented Messaging Protocol) este folosit deasupra WebSocket pentru a facilita comunicarea mesajelor.
- Implementare în YOLO: În WebSocketController, WebSocket împreună cu STOMP sunt utilizate pentru a implementa funcționalitățile de mesagerie în timp real, precum trimiterea de mesaje noi și actualizarea stării acestora.

Alte Tehnici Relevante

- **Spring Boot**: Este folosit pentru a simplifica procesul de configurare și desfășurare a aplicației, oferind o platformă robustă și flexibilă pentru dezvoltarea serviciilor back-end.
- **Reactor și Programare Reactivă**: YOLO utilizează programarea reactivă pentru a gestiona cereri asincrone și fluxuri de date, îmbunătățind performanța și scalabilitatea aplicației.
- **Securitatea API-urilor cu JWT:** JSON Web Tokens (JWT) sunt folosite pentru a asigura autentificarea și autorizarea utilizatorilor în sistem, protejând endpoint-urile și datele sensibile.

- **Gestionarea Stării cu Redux:** este o bibliotecă populară pentru gestionarea stării în aplicațiile JavaScript, inclusiv în cele realizate cu React. Utilizarea Redux în YOLO permite o gestionare centralizată și predictibilă a stării aplicației. Redux oferă o structură clară și uniformă pentru actualizarea stării, facilitând urmărirea schimbărilor și debugging-ul.
- **Programare Reactivă pe Frontend**: RxJS, abrevierea pentru Reactive Extensions for JavaScript, este o bibliotecă pentru programarea reactivă. Ea permite manipularea eficientă a evenimentelor asincrone și a fluxurilor de date. În YOLO, RxJS poate fi folosit pentru a gestiona cereri HTTP, evenimente UI și alte surse de date asincrone, oferind o abordare mai declarativă și mai flexibilă comparativ cu promisiunile și callback-urile tradiționale.

9.Design patterns

1. Singleton

Singleton este un pattern de design creational care asigură că o clasă are doar o singură instanță și oferă un punct de acces global la această instanță. Acest pattern rezolvă două probleme simultan, încălcând principiul responsabilității unice: asigură că o clasă are o singură instanță și oferă un punct de acces global la acea instanță.

Problema pe care Singleton o rezolvă este controlul accesului la o resursă partajată, cum ar fi o bază de date sau un fișier. Funcționarea acestui pattern este simplă: dacă creezi un obiect și mai târziu decizi să creezi un altul, nu vei primi un obiect nou, ci același obiect creat anterior. Acest comportament este imposibil de implementat cu un constructor obișnuit, deoarece apelul unui constructor trebuie să returneze întotdeauna un obiect nou.

O altă latură a Singleton-ului este furnizarea accesului global la obiect. Astfel, clienții pot nici măcar să nu realizeze că lucrează tot timpul cu același obiect. Acest lucru este asemănător cu variabilele globale, dar Singleton-ul protejează instanța de a fi suprascrisă de alt cod.

Toate implementările Singleton au doi pași comuni: constructorul default este privat, pentru a preveni alte obiecte să folosească operatorul new cu clasa Singleton, și se creează o metodă statică de creație care acționează ca un constructor. Această metodă apelează constructorul privat pentru a crea un obiect și îl salvează într-un câmp static. Toate apelurile ulterioare către această metodă returnează obiectul salvat în cache.

În Spring, majoritatea bean-urilor (de exemplu, servicii, repository-uri) sunt create ca Singleton-uri implicit. Așadar, în implementarea mea, Spring creează o singură instanță a fiecărui bean și o reutilizează ori de câte ori este nevoie în cadrul aplicației. Aceasta este o

caracteristică fundamentală a framework-ului Spring și este un mod de a aplica pattern-ul Singleton fără a fi nevoie de o implementare explicită în codul sursă.

```
2 usages  AndradaHr
@Repository
public interface ReactionRepository extends MongoRepository<Reaction, Long> {
    3 usages  AndradaHr
    Optional<Reaction>findByPostId(Long postId);
}
```

ReactionRepository, definit mai sus, este un exemplu clar al aplicării patternului Singleton în contextul Spring Framework. Acesta extinde **MongoRepository**, oferind funcționalități specifice pentru interacțiunea cu MongoDB, și este anotat cu **@Repository**. Această anotare semnalizează Spring să trateze **ReactionRepository** ca un bean și, în conformitate cu comportamentul implicit al Spring, să-l instanțieze ca un Singleton.

Prin utilizarea acestui pattern, Spring asigură că o singură instanță a **ReactionRepository** este creată și gestionată, oferind acces consistent și centralizat la operațiunile de baza de date pentru entitatea **Reaction**. Acest lucru optimizează gestionarea resurselor și reduce complexitatea codului, deoarece nu este necesară crearea de instanțe multiple ale repository-ului în diferite părți ale aplicației.

Probleme Rezolvate

În proiectul nostru, alegerea de a utiliza patternul Singleton pentru **ReactionRepository**, gestionat implicit prin Spring Framework, a fost o decizie strategică menită să abordeze și să rezolve mai multe provocări esențiale în gestionarea accesului la date.

Prima problemă pe care o rezolvă acest design este nevoia de a avea un acces centralizat și consistent la baza de date. Prin utilizarea unei singure instanțe a **ReactionRepository** în întreaga aplicație, putem asigura că toate interacțiunile cu baza de date sunt executate într-un mod uniform, eliminând riscul de date incoerente sau conflicte în accesul la baza de date. Această abordare centralizată ne ajută, de asemenea, să menținem o perspectivă clară asupra operațiunilor efectuate cu baza de date, o componentă esențială pentru mentenanța și monitorizarea aplicației.

O altă problemă importantă pe care o adresează acest design este eficiența în utilizarea resurselor. Crearea repetată a instanțelor de repository pentru interacțiunile cu baza de date ar putea fi ineficientă și ar putea conduce la o utilizare excesivă a resurselor de sistem. Prin

Singleton, minimizăm utilizarea resurselor, permițând aplicației să ruleze mai eficient și să gestioneze mai bine conexiunile la baza de date.

\+ **Repository**

Design Pattern-ul Repository este recunoscut ca fiind unul dintre cele mai populare modele de proiectare în domeniul dezvoltării software. Personal, îl apreciez pentru modul său de a respecta principiile SOLID și pentru simplitatea și claritatea pe care o oferă atunci când este implementat corect. Repository Pattern este un model de proiectare în programare care separă logica de acces la date de logica de afaceri a unei aplicații. În esență, acesta servește ca un intermediar între sursa de date (cum ar fi o bază de date) și stratul de logică al afacerii, oferind o interfață abstractă pentru gestionarea colecțiilor de obiecte.

Văd acest pattern având două scopuri principale: primul este de a oferi o abstracție a stratului de date, iar al doilea este de a centraliza gestionarea obiectelor de domeniu.

Detalii Implementare:

- **Abstractizare**: **ReactionRepository** extinde **MongoRepository**, care face parte din Spring Data MongoDB. Prin aceasta, se obține o interfață gata de utilizare pentru operarea cu obiecte **Reaction** în baza de date MongoDB.
- **Metode Predefinite**: **MongoRepository** oferă o serie de metode predefinite, cum ar fi salvarea, găsirea, ștergerea și actualizarea obiectelor.
- **Metode Personalizate**: În **ReactionRepository**, se adaugă o metodă personalizată, **findByPostId**, care este specifică nevoilor aplicației YOLO. Aceasta returnează un **Optional<Reaction>** pentru un anumit **postId**, permițând verificarea și gestionarea eficientă a reacțiilor utilizatorilor la postări.

Problema Rezolvată de Repository Pattern în YOLO

Separarea și Simplificarea Logicii de Acces la Date

- **Decuplarea Logicii de Business de Logica de Persistență**: Prin utilizarea **ReactionRepository**, logica de interacțiune cu baza de date este izolată de restul aplicației. Acest lucru face codul mai curat și ușor de înțeles.
- **Redundanță Redusă**: Evită repetarea codului pentru interogări de bază, deoarece majoritatea operațiunilor CRUD sunt furnizate de **MongoRepository**.
- **Flexibilitate în Testare**: Facilitează testarea unitară a logicii de business fără a fi nevoie de interacțiunea cu baza de date reală. Repository poate fi înlocuit cu un mock în timpul testării.

2.Observer

Designul Observer este un pattern de design comportamental care stabilește o relație între obiecte de tipul "one-to-many", unde un obiect (denumit "Observable") notifică automat multiple obiecte secundare (denumite "Observers") despre orice schimbări de stare sau evenimente la care acestea sunt interesate. Acest mecanism permite obiectelor observatoare să fie informate și să răspundă la schimbările care au loc în obiectul observabil, fără a fi necesară o cuplare strânsă între ele. Pattern-ul Observer este frecvent utilizat în implementarea interfețelor utilizator, unde schimbările în date sau starea sistemului necesită actualizări rapide și sincronizate ale UI-ului, precum și în sistemele de programare orientate pe evenimente, unde anumite acțiuni pot declanșa o serie de răspunsuri în diferite părți ale aplicației.

În cadrul proiectului nostru, am integrat eficient pattern-ul Observer pentru a optimiza gestionarea mesajelor în interfața utilizatorului a aplicației noastre bazate pe WebSockets. Acest design facilitează o interacțiune dinamică și reactivă între diferitele componente ale aplicației, cum ar fi actualizarea dinamică a listei de mesaje atunci când se adaugă sau se primește un mesaj nou.

Am implementat conceptul de Observable și Observer în următorul mod:

- **Observable**: Componenta care gestionează conexiunea WebSocket în proiectul nostru acționează ca un Observable. După stabilirea conexiunii cu WebSocket-ul, acesta rămâne în așteptare pentru a primi mesaje noi de la server. La sosirea unui mesaj nou, Observable-ul declanșează un dispatcher, care notifică toți Observerii despre evenimentul de mesaj nou.
- **Observer**: Componentele din interfața utilizatorului care afișează mesajele funcționează ca Observeri. Aceste componente sunt configurate să asculte Observable-ul și, atunci când un mesaj nou este recepționat, ele reacționează la acest eveniment. Reacția lor include actualizarea interfeței utilizatorului cu noul mesaj, adăugarea mesajului în sistemul de gestionare a stării și asigurarea sincronizării între interfața utilizatorului și starea aplicației.

Fluxul de lucru al acestui mecanism este după cum urmează:

1. **Conexiunea WebSocket**: Se stabilește conexiunea și componenta Observable începe să asculte mesajele de la server.
2. **Primirea Mesajului Nou**: Când un mesaj nou ajunge, Observable-ul activează dispatcher-ul pentru a informa toți Observerii.
3. **Reacția Observerilor**: Fiecare Observer răspunde, actualizând UI-ul cu noul mesaj, adăugând mesajul în sistemul de gestionare a stării și sincronizând UI-ul cu această stare.
4. **Actualizarea UI**: Interfața utilizatorului reflectă noul mesaj primit, fie prin adăugarea acestuia într-o listă de mesaje existentă, fie prin re-renderizarea unei componente pentru a arăta starea actualizată a conversației.

Beneficiile aduse de acest pattern includ:

- **Decuplarea**: Componentele UI sunt independente de sursa de date (WebSocket) și răspund doar la evenimente, ceea ce face codul mai modular și ușor de gestionat.
- **Reactivitate**: Interfața utilizatorului este capabilă să răspundă rapid la schimbările de date, asigurând o experiență fluidă pentru utilizator în aplicațiile de mesagerie.
- **Flexibilitate**: Putem adăuga sau elimina Observeri fără a perturba funcționarea Observable-ului, ceea ce face sistemul nostru extrem de adaptabil și extensibil.
- **Consistență în State Management**: Menținem coerența între starea aplicației și UI, asigurând că starea reflectă mereu situația actuală a datelor.

Probleme Rezolvate

În proiectul nostru, folosirea patternului Observer a rezolvat eficient mai multe probleme complexe, îmbunătățind interacțiunea dintre backend și interfața utilizatorului. Prima problemă abordată este nevoia de actualizări în timp real. Datorită Observerului, atunci când se primesc mesaje noi prin WebSocket, interfața utilizatorului este actualizată imediat, asigurând că utilizatorii primesc informațiile în cel mai scurt timp.

De asemenea, am reușit să sincronizăm și să menținem coerența datelor între backend și frontend. Orice schimbare care apare în datele backend-ului este instantaneu reflectată în interfața utilizatorului, eliminând astfel discrepanțele și îmbunătățind experiența utilizatorilor.

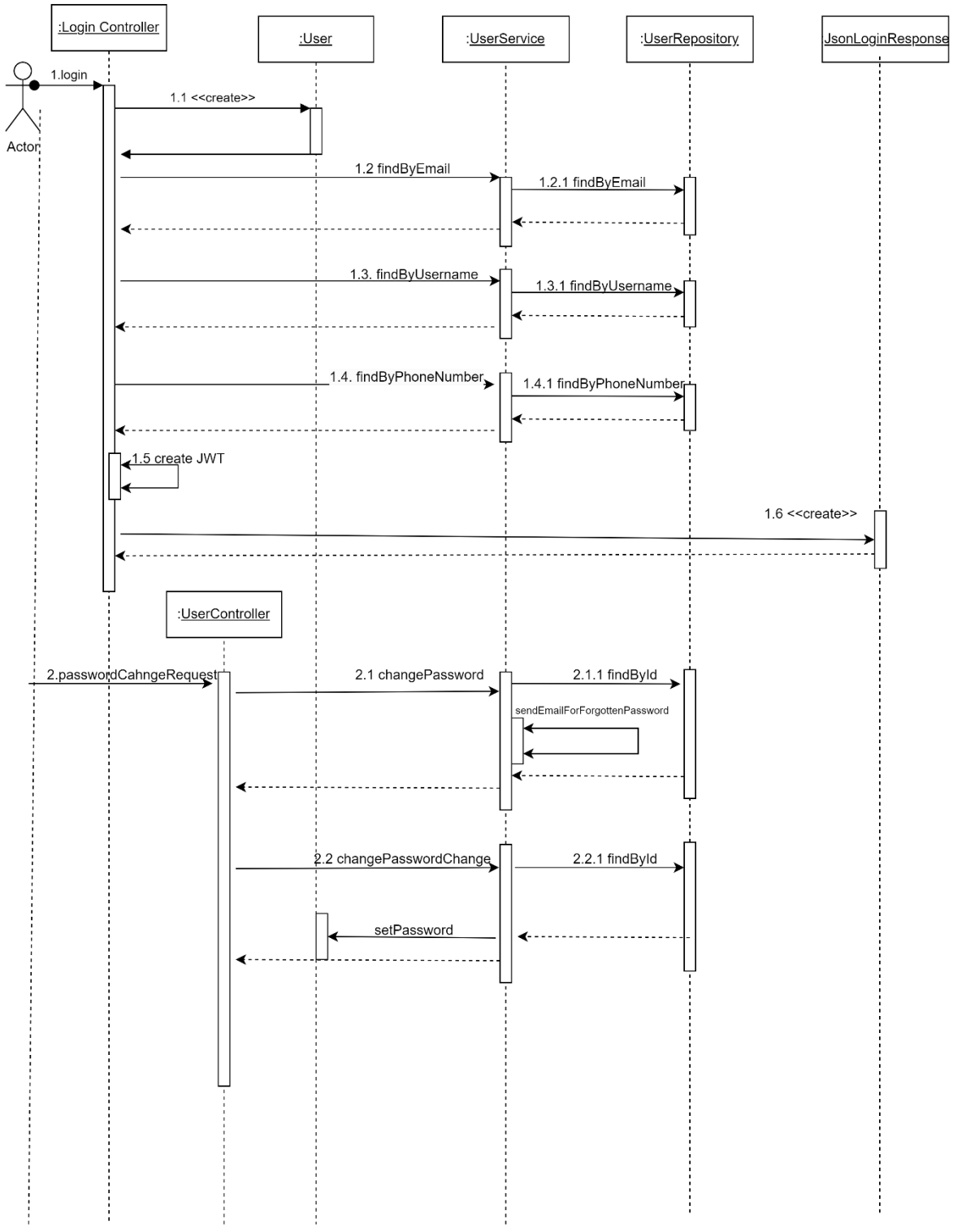
O altă problemă importantă rezolvată este decuplarea componentelor. Prin implementarea Observerului, sursa de date (WebSocket) este separată de consumatorii de date (componentele UI). Această decuplare a redus interdependența între diferite părți ale aplicației, făcând fiecare componentă mai independentă și ușor de gestionat.

Patternul Observer ne-a oferit, de asemenea, flexibilitatea de a adăuga sau elimina observatori după cum este necesar. Acest lucru se dovedește util în gestionarea dinamică a evenimentelor și actualizărilor UI, permițând adaptarea rapidă la schimbările cerințelor sau comportamentului utilizatorilor.

Din punct de vedere al performanței, abordarea bazată pe evenimente este mai eficientă decât alte tehnici, cum ar fi polling-ul. Reducerea sarcinii pe server și îmbunătățirea performanței aplicației au fost rezultate directe ale acestei implementări.

10.Diagrame

Diagrama de secventa


:Login Controller
:User
:UserService
:UserRepository
JsonLoginResponse
Actor
1.login
1.1 <<create>>
1.2 findByEmail
1.2.1 findByEmail
1.3. findByUsername
1.3.1 findByUsername
1.4. findByPhoneNumber
1.4.1 findByPhoneNumber
1.5 create JWT
1.6 <<create>>
:UserController
2.passwordCahngeRequest
2.1 changePassword
2.1.1 findById
sendEmailForForgottenPassword
2.2 changePasswordChange
2.2.1 findById
setPassword

Această diagramă de secvență UML ilustrează procesul de autentificare și schimbare a parolei în cadrul unui sistem software. Diagrama este organizată în două părți principale, fiecare corespunzând unui caz de utilizare specific.

Diagrama de secvență UML pe care ați încărcat-o ilustrează procesele de autentificare și schimbare a parolei într-un sistem software. Voi explica acum această diagramă ca și cum aș documenta-o în cadrul unui proiect pe care l-am realizat.

1. **Procesul de Autentificare**:
   - Începe când un actor (de obicei un utilizator) inițiază o cerere de autentificare (**login**) la **Login Controller**.
   - **Login Controller** creează un nou obiect **User** pentru a gestiona cererea.
   - **Login Controller** solicită **UserService** să găsească utilizatorul folosind metoda **findByEmail**.
   - Dacă **findByEmail** nu returnează un utilizator, se continuă cu **findByUsername**.
   - Dacă și **findByUsername** eșuează, se încearcă **findByPhoneNumber**.
   - Odată ce utilizatorul este găsit, **UserService** solicită **UserRepository** să creeze un JWT (JSON Web Token) pentru autentificarea și autorizarea utilizatorului.
   - JWT-ul este creat și trimis înapoi la **Login Controller**.
   - La sfârșitul procesului, **JSON Login Response** este ultima entitate din procesul de autentificare care returnează răspunsul final al cererii de autentificare.

2. **Procesul de Schimbare a Parolei**:
   - Utilizatorul inițiază o cerere de schimbare a parolei (**passwordChangeRequest**) în **UserController**.
   - **UserController** solicită **UserService** să găsească utilizatorul bazat pe ID (**findById**).
   - După identificarea utilizatorului, **UserService** poate iniția procedura de resetare a parolei, cum ar fi trimiterea unui email pentru resetarea parolei uitată (**sendEmailForForgottenPassword**).
   - Când utilizatorul confirmă schimbarea parolei (prin **changePasswordChange**), **UserService** actualizează parola în **UserRepository** folosind metoda **setPassword**.

Este important să notăm că fiecare interacțiune între controlere, servicii și repositoare este realizată în mod asincron, așa cum indică liniile punctate.

JWT-urile sunt folosite pentru a menține starea autentificată a utilizatorului între diferitele cereri HTTP, care sunt în mod natural fără stare.

În fiecare caz, interacțiunea dintre controlere, servicii și repositoare indică separarea clară a responsabilităților în arhitectura sistemului. Controlerele gestionează solicitările de intrare și răspunsurile de ieșire. Serviciile conțin logica de afaceri. Repositoarele gestionează accesul la date, fie pentru a recupera informații despre utilizator sau pentru a actualiza detaliile acestuia în baza de date.

Diagrama de stare

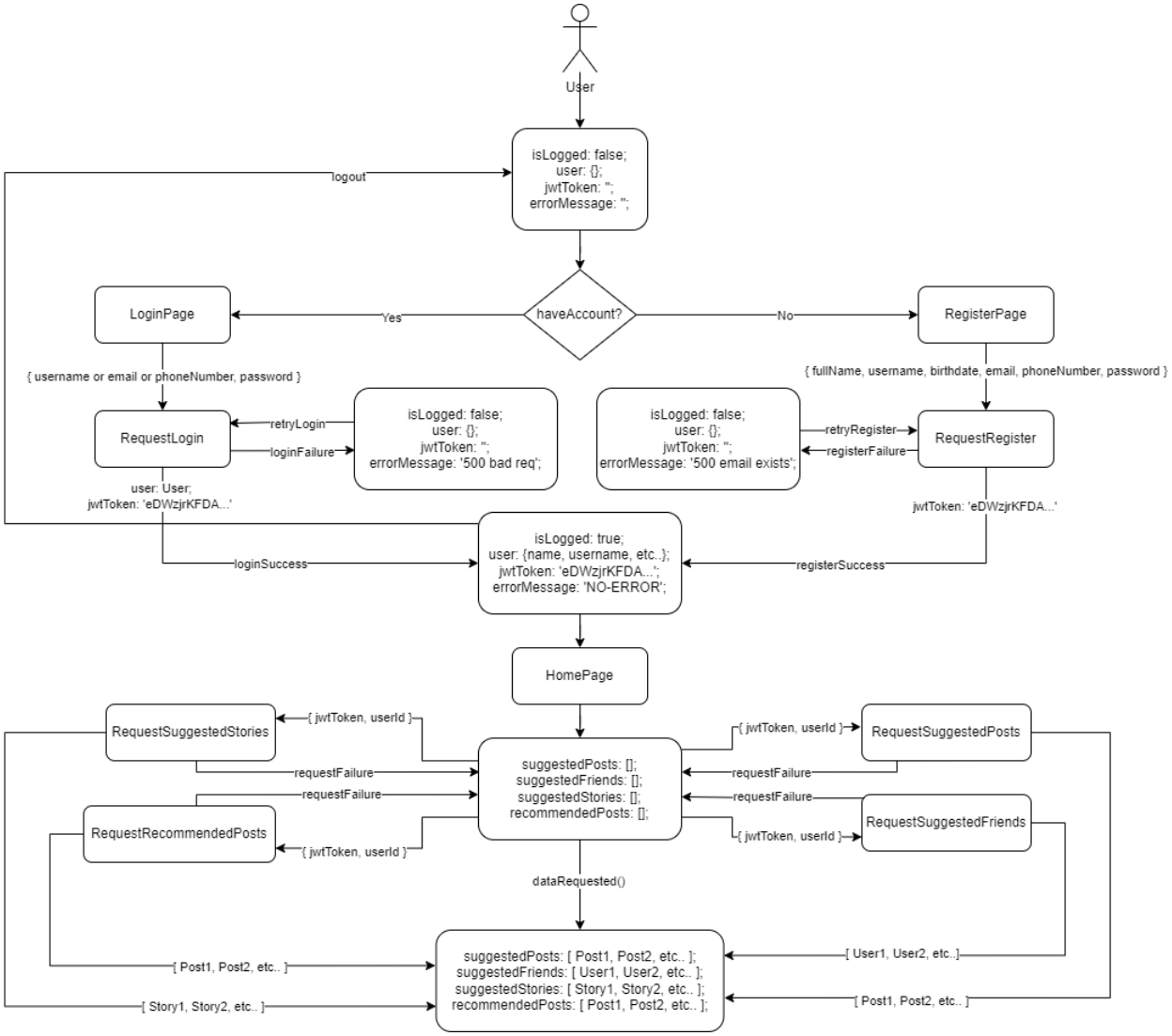


Diagrama de stări prezentată ilustrează fluxul de autentificare al utilizatorului și interacțiunea cu pagina de start a unei aplicații de rețea socială.

Autentificare și Înregistrare

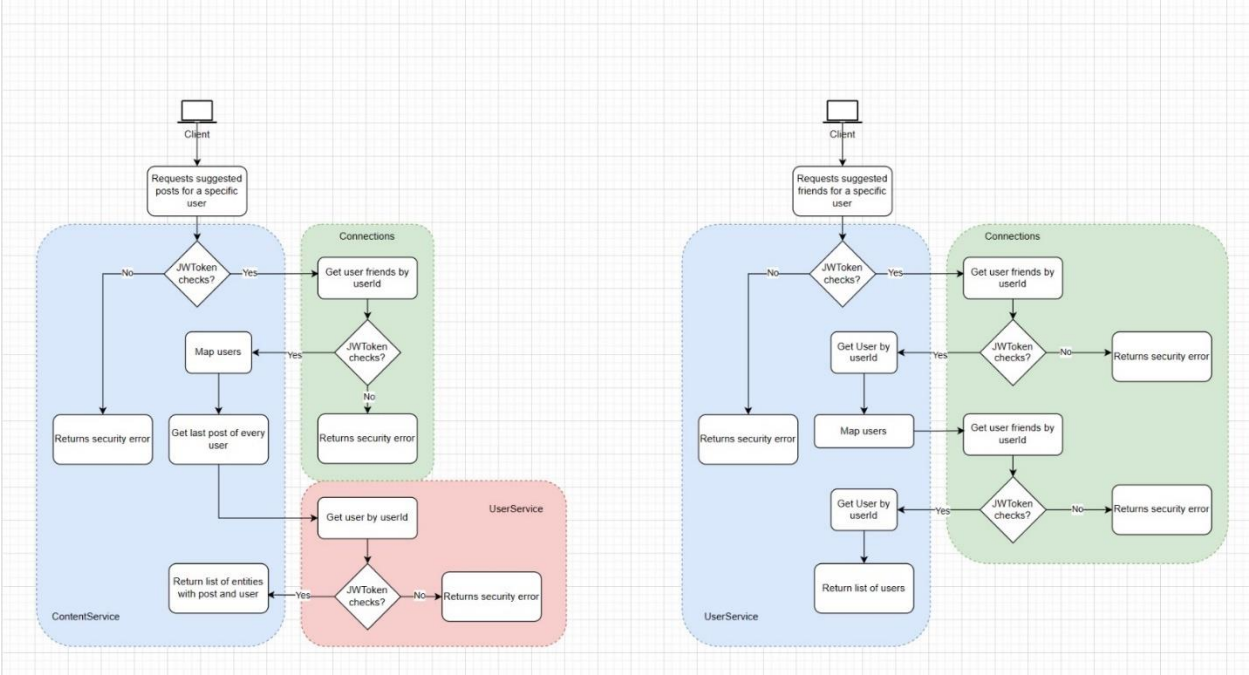
- **Pagina de Autentificare (LoginPage)**: Utilizatorul introduce credențialele (numele de utilizator, email sau numărul de telefon și parola) și solicită autentificarea (RequestLogin).
- Dacă autentificarea eșuează (loginFailure), se afișează un mesaj de eroare și utilizatorul are opțiunea de a încerca din nou (retryLogin).
- Dacă autentificarea reușește (loginSuccess), utilizatorul primește un token JWT și este redirecționat către pagina principală (HomePage).
- **Pagina de Înregistrare (RegisterPage)**: Dacă utilizatorul nu are un cont, este ghidat către procesul de înregistrare.
- Utilizatorul completează formularul de înregistrare și solicită crearea contului (RequestRegister).
- Dacă înregistrarea eșuează (registerFailure), de exemplu, din cauza unei adrese de email care există deja, utilizatorul este informat și poate încerca din nou (retryRegister).
- Dacă înregistrarea reușește (registerSuccess), utilizatorul primește un token JWT și este redirecționat către pagina principală (HomePage).

Interacțiunea cu Pagina Principală

- Pe **HomePage**, utilizatorul poate solicita diverse conținuturi din rețeaua socială:
    - **Povești sugerate (RequestSuggestedStories)**: Aplicația încearcă să preia poveștile sugerate pentru utilizator. Dacă solicitarea eșuează (requestFailure), se poate încerca din nou.
    - **Postări recomandate (RequestRecommendedPosts)**: Solicită postările recomandate bazate pe interesele utilizatorului.
    - **Postări sugerate (RequestSuggestedPosts)**: Solicită postări care ar putea interesa utilizatorul.
    - **Prieteni sugerați (RequestSuggestedFriends)**: Solicită o listă de prieteni sugerați pe care utilizatorul i-ar putea adăuga.
- Toate aceste solicitări folosesc tokenul JWT și ID-ul utilizatorului pentru a valida sesiunea și pentru a personaliza conținutul.

- Rezultatele acestor solicitări sunt diversele seturi de date (suggestedPosts, suggestedFriends, etc.) care sunt afișate pe pagina principală.

Diagrama de actiune



Această diagramă descrie fluxurile de procesare pentru două servicii în cadrul unei aplicații web sau mobile care gestionează conținutul utilizatorilor și rețeaua lor socială. Fiecare flux începe cu o cerere de la client și este structurat pentru a verifica autenticitatea cererii folosind un token JWT. În documentația de proiect, aș descrie fiecare secțiune a diagramăi astfel:

Descrierea Fluxului Serviciului de Conținut (ContentService)

- Clientul inițiază fluxul prin solicitarea postărilor sugerate pentru un anumit utilizator.
- Sistemul verifică prezența unui JWT valid. Aceasta este prima linie de apărare împotriva accesului neautorizat.
- În cazul în care tokenul este invalid, fluxul se termină, iar sistemul returnează o eroare de securitate, prevenind orice altă procesare.
- Dacă tokenul este verificat cu succes, serviciul trece la maparea utilizatorilor și extrage ultima postare a fiecărui utilizator.

- La finalul fluxului, ContentService compune o listă de entități care include atât postările cât și detaliile utilizatorilor asociați și o returnează clientului.

Descrierea Fluxului Serviciului de Utilizatori (UserService)

- Clientul face o solicitare similară pentru a obține prieteni sugerați pentru un utilizator.
- Serviciul verifică, de asemenea, JWT-ul pentru a asigura că cererea este autentică.
- O eroare de securitate este returnată dacă tokenul este invalid, oprirea procesării și informarea clientului.
- Cu un JWT valid, sistemul obține detalii despre utilizator și lista sa de prieteni, bazându-se pe ID-ul utilizatorului.
- În final, UserService returnează o listă de utilizatori, care reprezintă prietenii sugerați, clientului.

Notă Comună pentru Ambele Servicii

- Un sub-flux numit Connections operează paralel cu ambele servicii principale, executând o verificare JWT separată pentru a obține prietenii utilizatorului.
- Acest sub-flux reprezintă o operațiune comună pentru ambele servicii și oferă o verificare suplimentară de securitate înainte de a furniza datele.

Diagrama de use case

Diagrama este diagrama de cazuri de utilizare (use case diagram) a aplicației de social media. Acest tip de diagramă este utilizat pentru a descrie funcționalitățile și procesele pe care o aplicație le oferă diferitelor tipuri de utilizatori.

Explorarea și Căutarea

- Explorarea Feed-ului: Utilizatorii pot naviga printr-un flux de postări actualizate, fie că sunt de la utilizatorii urmăriți, fie din recomandări generale.
- Căutarea Utilizatorilor, Tag-urilor și Locațiilor: Acest caz de utilizare permite utilizatorilor să caute și să găsească alte profile, postări etichetate sau locații.

Autentificare și Managementul Contului

- Autentificare: Procesul prin care utilizatorii își verifică identitatea pentru a accesa aplicația.

- Creare cont: Utilizatorii noi își pot înregistra un cont în aplicație.
- Editarea profilului include cazurile de utilizare detaliate pentru Modificarea pozei de profil, Modificarea descrierii, și Modificarea numelui/username-ului.
- Setările de Confidențialitate: Aici utilizatorii pot decide dacă profilul lor este public sau privat și cine poate vedea poveștile lor sau trimite mesaje.
- Securitatea Contului: Include cazuri de utilizare pentru schimbarea parolei și email-ului.

Crearea și Gestionarea Conținutului

- Salvarea postărilor: Permite utilizatorilor să salveze postări pentru a le vizualiza mai târziu.
- Postarea de Fotografii și Videoclipuri: Utilizatorii pot încărca și partaja conținut vizual.
- Adăugarea de Story-uri: Utilizatorii pot posta conținut care este vizibil temporar.
- Adăugarea de Text și Hashtag-uri: Utilizatorii pot adăuga context și etichete la postările lor pentru a îmbunătăți vizibilitatea.

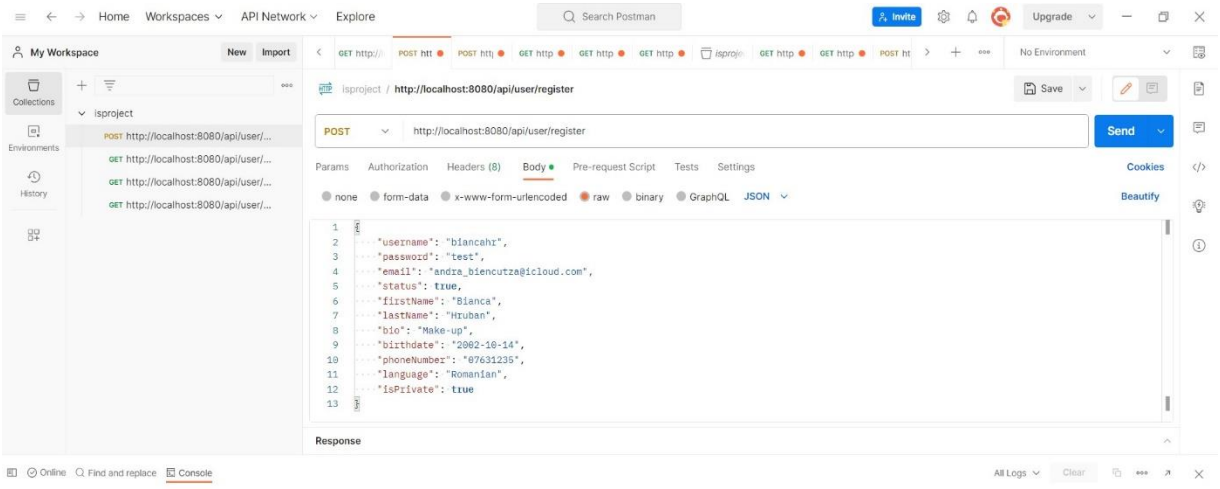
Interacțiunea cu Alți Utilizatori

- Urmărirea Altor: Permite utilizatorilor să urmărească activitatea altor conturi.
- Mesaje Directe: Utilizatorii pot trimite mesaje private unul altuia.
- Partajarea Conținutului: Utilizatorii pot partaja postări cu alții.
- Like-uri și Comentarii: Utilizatorii pot interacționa cu postările prin aprecieri și comentarii.


Explorarea Feed-ului
Căutarea Utilizatorilor, Tag-urilor și Locațiilor
Explorarea și Căutarea
Modificare poza profil
Modificare descriere
Autentificare
Editarea profilului
Modificare nume/username
Creare cont
Setări de Confidențialitate
Cont public/privat
User
Gestionarea profilului
Who can see story/send message
Crearea și Gestionarea Conținutului
Securitatea Contului
Schimbare email
Salvarea postarilor
Postarea de Fotografii și Videoclipuri
Schimbare parola
Interacțiunea cu Alți Utilizatori
Adăugarea de Story-uri
Adăugarea de Text și Hashtag-uri
Urmărirea Altora
Mesaje Directe
Partajarea Conținutului
Like-uri și Comentarii

10. Strategii de Testare

Postman: Este un instrument esențial pentru testarea și validarea interfețelor API în aplicația YOLO. Postman oferă o interfață ușor de utilizat pentru a trimite cereri către serverul de backend și a vizualiza răspunsurile. Acesta permite dezvoltatorilor să testeze rapid și eficient diferite scenarii de cereri API, inclusiv GET, POST, PUT și DELETE, facilitând astfel procesul de debugging și verificare a integrității API-urilor. Astfel, în prima fază ne-am folosit de serviciile Postman pentru a ne testa codul și a ne asigura că funcționează pe deplin așa cum ne-am așteptat.



Odată ce API-urile sunt validate, următorul pas crucial este integrarea și testarea conexiunii cu frontend-ul. Acest proces implică testarea interacțiunii dintre codul frontend-ului React și serviciile backend. Ne-am concentrat pe asigurarea unei comunicări fără cusur și eficiente între cele două părți ale aplicației, validând fluxurile de date și gestionarea erorilor.