Site cu pagini diferite pentru Admin si User

Dumitru Mihnea

Hanganu Radu

442G

Cuprins :

Listă figuri :

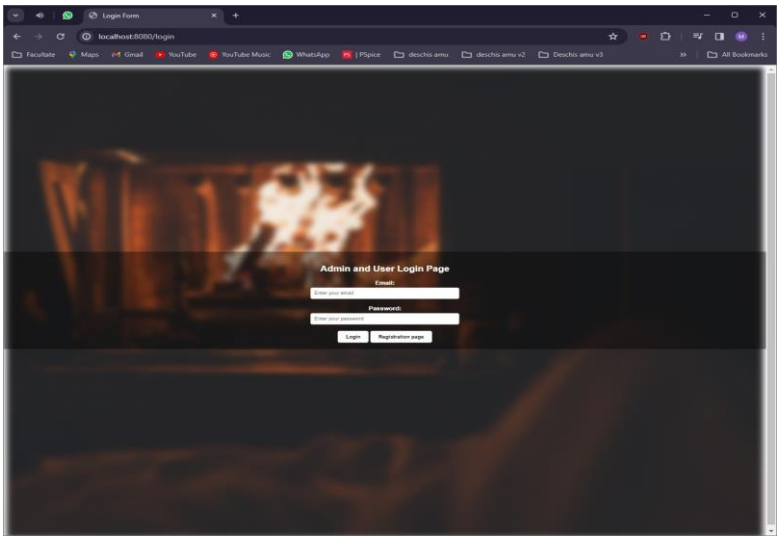

Figura 1

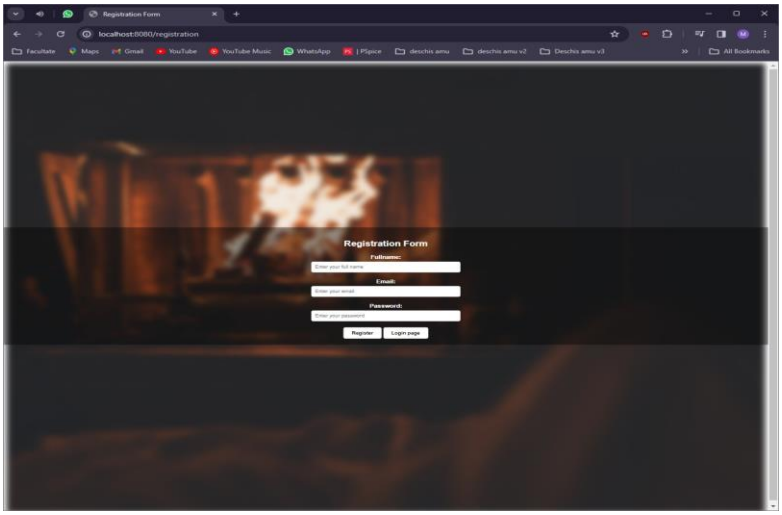

Figura 2

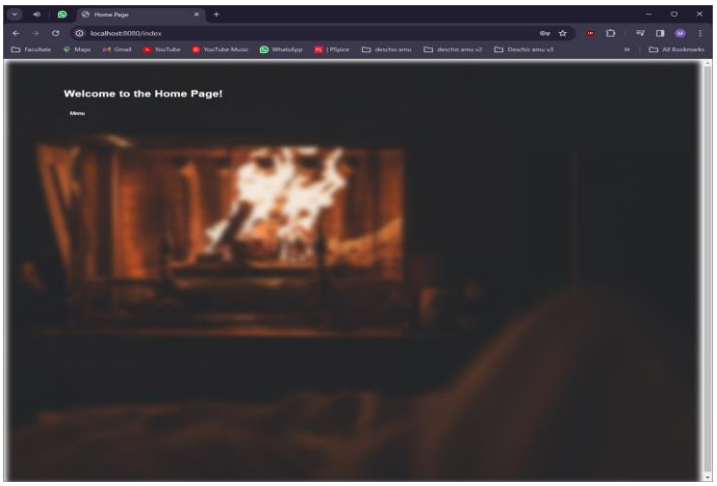

Figura 3

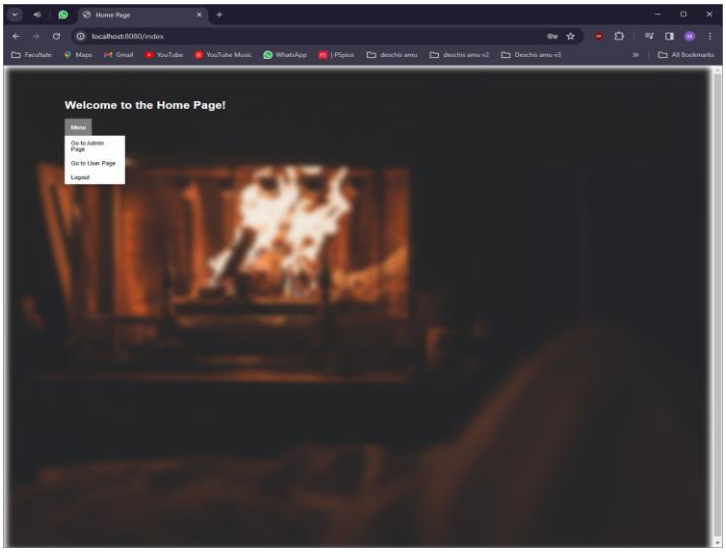

Figura 4

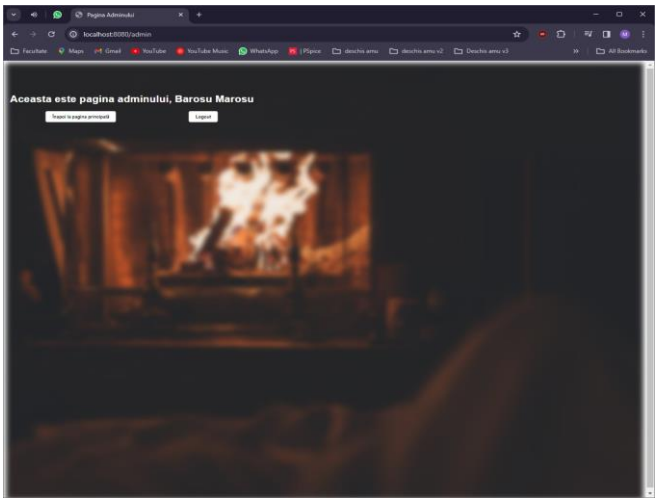

Figura 5

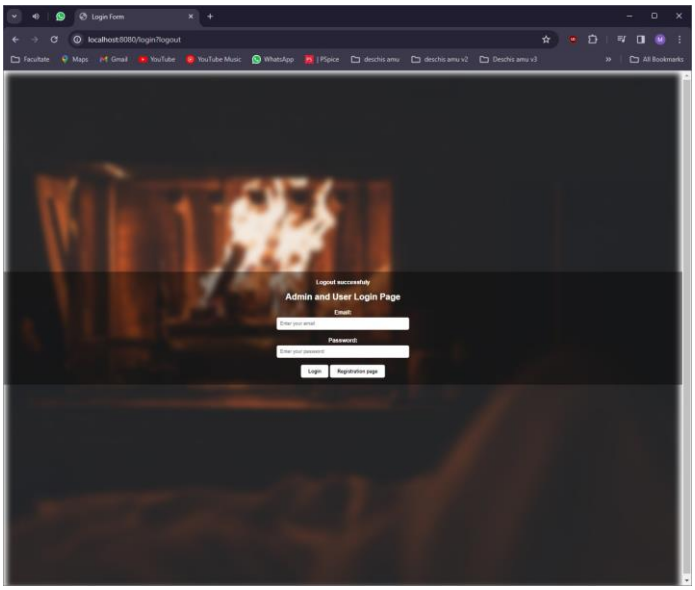

Figura 6

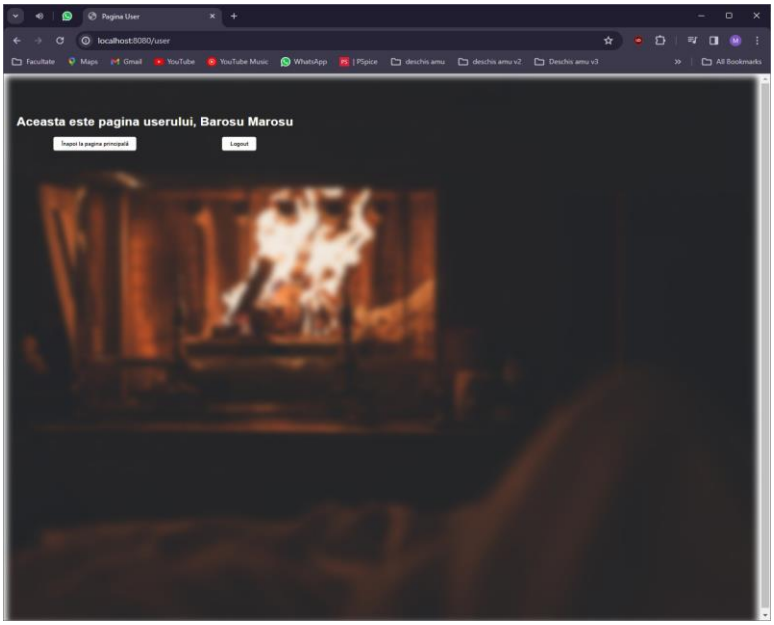

Figura 7

I) Introducere

O pagină web reprezintă un document aflat pe World Wide Web având "propria sa adresa". Paginile web sunt furnizate utilizatorului prin intermediul unui server web și afișarea lor se face într-un browser web, precum Google Chrome, Microsoft Edge sau Mozilla Firefox, având rol de "unitate de recuperare" a informației stocate în acesta. Un website este format din multe pagini web, legate împreună printr-un nume de domeniu.

O pagină web reprezintă un document structurat. Elementul principal al paginii web este reprezentat de fișierul text scris în limbajul HTML, având rolul de a specifica conținutul paginii web (incluzând și link-urile denumite hyperlinks) către alte resurse web și către diferite secțiuni alea aceleiași pagini web. Conținutul multimedia de pe web, cum e și cazul imaginilor și al filmuletelor, pot fi incorpotate direct într-o pagina web pentru a forma un document compus.

Fiecare pagina web este identificata printr-un link distinct, denumit Uniform Resource Locator (URL). Când utilizatorul înserează URL-ul unei pagini web într-un browser web, browser-ul descarcă un fișier HTML de pe serverul web și transforma toate elementele sale într-o reprezentare vizuala interactiva pe dispozitivul utilizatorului.

Daca utilizatorul accesează hyperlink-ul, browser-ul repeta acest proces pentru a încarca pagina indicata de hyperlink, care ar putea fi parte din website-ul curent sau unul diferit. Browser-ul are funcții ale interfeței cu utilizatorul care indica pagina afișată.

O aplicație web scrisa în limbajul Java generează pagini web interactive cu conținut dinamic. O astfel de aplicație este alcătuită din diferite componente web, cum ar fi JavaServer Pages (JSP), servlets și JavaBeans pentru a modifica și stoca date, interacționă cu diverse baze de date și servicii web, precum și furniza conținutul cerut de utilizator.

II) Instrumente si tehnologii folosite :

Pentru a dezvolta o pagină web, există numeroase tehnologii și instrumente disponibile în funcție de nevoile și preferințele developer-ului. Iată opțiunile noastre:

- Limbaje de programare: HTML, CSS și Java - Fundamentale pentru dezvoltarea paginilor web. HTML oferă structura, CSS se ocupă de stilizare, iar Java adaugă funcționalități.

- Framework-uri și biblioteci : Spring Boot - Pentru dezvoltarea rapidă a aplicațiilor Java pentru server
- Baze de date : MySQL – pentru structuri de date relaționale
- Mediu de dezvoltare : Intellij IDEA

III) Explicatii asupra codului :

Clasa “SecurityConfig”:

Aceasta clasa a fost făcută folosind “@Configuration” și “@EnableWebSecurity”, indicând că este o clasa de configurare folosita în Spring Security într-o aplicație web.

Comentariul “@Autowired” : Acest comentariu este folosit pentru a insera instanțe ale “CustomSuccesHandler” și “CustomUserDetailsService” în clasa de configurare.

Metodele “@Bean” :

Metoda “passwordEncoder()” este adnotata cu “@Bean” și creează o instanță a “BCryptPasswordEncoder” Aceasta metoda oferă un “PasswordEncoder” bean care poate fi folosit pentru criptarea parolei.

Metoda “securityFilterChain(HttpSecurity http)” este adnotata cu “@Bean” și configurează setările de securitate folosind obiectul “HttpSecurity”; setează reguli pentru autentificare și autorizare.

Configurarea “securityFilterChain” : Metoda “securityFilterChain” este folosita pentru a configura setările de securitate folosind obiectul “HttpSecurity”

“.csrf(c -> c.disable())” : Protecția CSRF este dezactivata. Protecția CSRF nu este necesara de obicei atunci când lucram cu aplicații stateless sau APIs.

“.authorizeRequests(request -> …)” : configurează autorizarea bazata pe URL. Utilizatorii cu autoritatea “ADMIN” au acces la “/admin” , iar utilizatorii cu autoritatea “UȘER” au acces la “/ușer”. Căile “/registration" și “/css/**” sunt permise tuturor utilizatorilor, iar orice alte cereri necesita autentificarea.

“.formLogin(form -> …)” : configurează autentificarea bazata pe formulări. Autentificarea făcută cu succes este prelucrata mai apoi de “CustomSuccesHandler”

“.logout(form -> …)” : configurează setările de delogare, printre care și invalidarea sesiunii HTTP, curățarea autentificării, specificarea URL-ului de delogare (“/logout”) și redirectionarea către pagina de login

Metoda “configure(AuthentificationManagerBuilder auth)”

Metoda “configure” este adnotata folosind “@Autowired” și configurează “AuthentificationManagerBuilder” să folosească “CustomUserDetailsService” pentru detaliile legate de utilizator și “BCryptPasswordEncoder” pentru criptarea parolei.

Clasa “UserControler”:

Acest cod reprezintă un controler Spring MVC pentru operațiuni legate de utilizator într-o aplicație web. Să analizăm fiecare metodă:

Clasa UserController:

Această clasă este adnotată cu @Controller, indicând că este un controler Spring MVC. Gestionează cereri legate de utilizator, precum înregistrare, autentificare și acces la paginile de utilizator și de administrator. Anotările @Autowired:

Anotarea @Autowired este folosită pentru a injecta instanțe de UserDetailsService și UserService în controler. Metoda getRegistrationPage:

Această metodă gestionează cereri de tip GET către "/registration" și returnează vizualizarea "register". Primește un obiect UserDto ca atribut de model, probabil utilizat pentru a lega datele formularului pentru înregistrarea utilizatorului. Metoda saveUser:

Această metodă gestionează cereri de tip POST către "/registration" și este responsabilă pentru salvarea datelor de înregistrare ale utilizatorului. Primește un obiect UserDto și un Model ca parametri. Apelează metoda save a UserService pentru a persista datele utilizatorului. Adaugă un mesaj "Înregistrare cu succes" în model și returnează vizualizarea "register". Metoda login:

Această metodă gestionează cereri de tip GET către "/login" și returnează vizualizarea "login". Probabil este folosită pentru afișarea formularului de autentificare. Metoda user:

Această metodă gestionează cereri de tip GET către "/user" și este responsabilă pentru afișarea paginii de utilizator. Primește un Model și un Principal ca parametri. Folosește UserDetailsService pentru a încărca detaliile utilizatorului în funcție de numele de utilizator din Principal. Adaugă detaliile utilizatorului în model și returnează vizualizarea "user". Metoda admin:

Această metodă gestionează cereri de tip GET către "/admin" și este responsabilă pentru afișarea paginii de administrator. Similar cu metoda user, primește un Model și un Principal. Folosește UserDetailsService pentru a încărca detaliile utilizatorului în funcție de numele de utilizator din Principal. Adaugă detaliile utilizatorului în model și returnează vizualizarea "admin".

Clasa “UserDto”:

Codul furnizat reprezintă un Obiect de Transfer de Date (DTO) numit UserDto. Obiectele DTO sunt folosite în mod obișnuit pentru transferul datelor între diferite straturi ale unei aplicații, cum ar fi între straturile de controler și de serviciu. Hai să analizăm aspectele cheie ale acestei clase UserDto:

Variabile de Instanță:

email: Reprezintă adresa de email a utilizatorului.

password: Reprezintă parola utilizatorului.

role: Reprezintă rolul utilizatorului (de exemplu, "USER" sau "ADMIN").

fullname: Reprezintă numele complet al utilizatorului.

Constructor:

Există un constructor cu parametri care permite crearea unui obiect UserDto cu valori pentru toate variabilele de instanță. Acest constructor este probabil utilizat atunci când se creează instanțe de UserDto în timpul înregistrării utilizatorului sau pentru transferul de date.

Metode Getter și Setter:

Metodele Getter (getEmail(), getPassword(), getRole(), getFullname()) oferă acces la valorile variabilelor de instanță.

Metodele Setter (setEmail(), setPassword(), setRole(), setFullname()) permit setarea valorilor pentru variabilele de instanță.

În rezumat, clasa UserDto servește ca un container pentru datele legate de utilizator și este proiectată pentru a fi utilizată pentru transferul informațiilor despre utilizator între diferite părți ale aplicației. Acest lucru poate fi deosebit de util atunci când datele trebuie să fie transferate între controler, straturile de serviciu și potențial stratul de vizualizare (în cazul formularelor de înregistrare a utilizatorului). Încapsulează adresa de email, parola, rolul și numele complet al utilizatorului într-un singur obiect pentru manipularea convenabilă a datelor.

Clasa “User”:

Codul furnizat definește o clasă Java numită User care reprezintă o entitate de utilizator într-o aplicație Spring. Iată o explicație a elementelor cheie:

Anotarea Entity:

Clasa este adnotată cu @Entity, indicând că instanțele acestei clase vor fi mapeate la o tabelă din baza de date. Acest lucru este utilizat de obicei în conjuncție cu un cadru ORM (Object-Relational Mapping), cum ar fi Hibernate. Anotarea Table:

Anotarea @Table specifică numele tabelei din baza de date pentru această entitate (users). De asemenea, include o restricție unică pe coloana email pentru a asigura că adresele de email sunt unice în cadrul tabelei. Anotarea Id:

Anotarea @Id desemnează câmpul id ca fiind cheia primară pentru entitate. Anotarea GeneratedValue:

Anotarea @GeneratedValue specifică faptul că valorile câmpului id vor fi generate automat (incrementate automat) de către baza de date. Strategia este setată la GenerationType.AUTO. Variabile de Instanță:

id: Reprezintă identificatorul unic pentru un utilizator. Generat automat de către baza de date. email: Reprezintă adresa de email a utilizatorului. password: Reprezintă parola utilizatorului. role: Reprezintă rolul utilizatorului (de exemplu, "USER" sau "ADMIN"). fullname: Reprezintă numele complet al utilizatorului. Constructori: Există un constructor implicit fără argumente, precum și un constructor cu parametri care permite crearea unui obiect User cu valori specifice. Metode Getter și Setter:

Metodele Getter (getId(), getEmail(), getPassword(), getRole(), getFullname()) oferă acces la valorile variabilelor de instanță. Metodele Setter (setId(), setEmail(), setPassword(), setRole(), setFullname()) permit setarea valorilor pentru variabilele de instanță.

Interfata de repository “UserRepository”:

Codul furnizat definește o interfață de repository Spring Data JPA numită UserRepository. Să analizăm elementele cheie:

Anotare Repository:

Interfața este adnotată cu @Repository, indicând că este un bean Spring și este eligibilă pentru scanarea componentelor.

Interfața JpaRepository:

Interfața extinde JpaRepository<User, Long>. JpaRepository este o interfață Spring Data JPA care oferă operații CRUD (Create, Read, Update, Delete) pentru tipul de entitate (User) specificat ca un parametru generic. Al doilea parametru generic (Long) este tipul cheii primare a entității.

Metoda findByEmail:

Interfața declară o metodă numită findByEmail care primește un parametru de tip String (email).

Această metodă este o metodă de interogare implementată automat de Spring Data JPA. Ea recuperează o entitate User în funcție de proprietatea email, iar Spring Data JPA generează interogarea corespunzătoare pe baza numelui metodei.

Clasa “CustomSuccesHandler”:

Codul furnizat definește o clasă de manipulare personalizată a autentificării numită CustomSuccessHandler într-o aplicație Spring. Iată o explicație a elementelor cheie:

Anotare Service:

Clasa este adnotată cu @Service, indicând că este un bean de serviciu Spring și este eligibilă pentru scanarea componentelor.

Implementare a AuthenticationSuccessHandler:

Clasa implementează interfața AuthenticationSuccessHandler furnizată de Spring Security. Această interfață permite manipularea personalizată a evenimentelor de autentificare reușită.

Metoda onAuthenticationSuccess:

Metoda onAuthenticationSuccess este suprascrisă din interfața AuthenticationSuccessHandler.

Această metodă este apelată când autentificarea este reușită.

Are trei parametri:

HttpServletRequest request: Reprezintă cererea HTTP.

HttpServletResponse response: Reprezintă răspunsul HTTP.

Authentication authentication: Reprezintă obiectul de autentificare care conține informații despre utilizatorul autentificat.

Redirecționări bazate pe roluri:

Metoda preia autoritățile (rolurile) utilizatorului din obiectul Authentication.

Utilizează un stream pentru a mapa autoritățile la numele lor de rol corespunzătoare și apoi preia primul rol folosind findFirst().

În funcție de rolul utilizatorului, acesta este redirecționat către pagini diferite:

Dacă utilizatorul are rolul "ADMIN", este redirecționat către "/admin".

Dacă utilizatorul are rolul "USER", este redirecționat către "/user".

Dacă utilizatorul are orice alt rol sau nu are niciun rol, este redirecționat către "/error".

Clasa “CustomUserDetails”:

Codul furnizat definește o clasă numită CustomUserDetails care implementează interfața Spring Security UserDetails. Această clasă este folosită pentru a încapsula entitatea User într-o formă care poate fi utilizată de Spring Security pentru autentificare și autorizare. Hai să analizăm elementele cheie:

Constructor:

Clasa are un constructor care primește un obiect User ca parametru și inițializează câmpul user.

Metoda getAuthorities:

Metoda getAuthorities este implementată din interfața UserDetails.

Aceasta returnează o colecție de obiecte GrantedAuthority. În acest caz, este returnată o singură autoritate (rol), iar rolul este obținut din obiectul User asociat.

Metoda getFullname:

Există o metodă personalizată getFullname care returnează numele complet al utilizatorului. Această metodă nu face parte din interfața UserDetails, ci este o adăugire personalizată.

Metoda getPassword:

Metoda getPassword este implementată din interfața UserDetails, returnând parola asociată utilizatorului.

Metoda getUsername:

Metoda getUsername este implementată din interfața UserDetails, returnând adresa de email a utilizatorului.

Metodele isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled:

Aceste metode sunt implementate pentru a indica faptul că contul utilizatorului este întotdeauna neexpirat, neblocat, că credențialele nu au expirat și că utilizatorul este întotdeauna activat. Acestea sunt adesea setate implicit ca fiind adevărate, dar pot fi personalizate în funcție de cerințele specifice.

Clasa “CustoUserDetailsService”:

Codul furnizat definește o clasă numită CustomUserDetailsService care implementează interfața Spring Security UserDetailsService. Această clasă este responsabilă pentru încărcarea detaliilor utilizatorului în timpul procesului de autentificare. Hai să analizăm elementele cheie:

Anotare Service:

Clasa este adnotată cu @Service, indicând că este un bean de serviciu Spring și este eligibilă pentru scanarea componentelor.

Autowired UserRepository:

Clasa are o anotare @Autowired pe câmpul UserRepository, indicând că o instanță a UserRepository va fi injectată de Spring.

Metoda loadUserByUsername:

Clasa implementează metoda loadUserByUsername din interfața UserDetailsService.

Această metodă este apelată de Spring Security în timpul procesului de autentificare pentru a încărca detaliile utilizatorului în funcție de numele de utilizator furnizat (în acest caz, adresa de email).

Obținerea utilizatorului din UserRepository:

În interiorul metodei, este apelată metoda findByEmail a UserRepository injectat pentru a obține o entitate User în funcție de numele de utilizator furnizat.

Dacă utilizatorul nu este găsit, este aruncată o UsernameNotFoundException.

Crearea CustomUserDetails:

Dacă utilizatorul este găsit, este creată o nouă instanță a CustomUserDetails, pasând obiectul User recuperat constructorului.

UserDetails returnat:

Metoda returnează obiectul CustomUserDetails creat, care implementează interfața UserDetails.

Interfata de serviciu "UserService":

Codul furnizat definește o interfață de serviciu numită UserService. Hai să analizăm elementele cheie:

Definiția Interfeței:

Interfața declară o metodă numită save, care primește un parametru de tip UserDto și returnează un obiect User.

Semnătura Metodei save:

Metoda save este destinată să salveze informațiile utilizatorului pe baza obiectului UserDto furnizat.

Parametrul UserDto:

Parametrul UserDto sugerează că această metodă este concepută pentru a gestiona înregistrarea utilizatorului sau crearea utilizatorului, deoarece primește un obiect de transfer de date (UserDto) care conține informații legate de utilizator.

Tipul Returnat:

Tipul returnat al metodei este User, indicând că returnează o entitate User. Acest lucru sugerează că metoda este probabil responsabilă pentru persistarea informațiilor utilizatorului într-o anumită stocare (de exemplu, o bază de date) și returnarea entității de utilizator salvată.

Clasa “UserServiceImpl”:

Codul furnizat definește o clasă de implementare a serviciului numită UserServiceImpl care implementează interfața UserService. Hai să analizăm elementele cheie:

Anotarea Service:

Clasa este adnotată cu @Service, indicând că este un bean de serviciu Spring și este eligibilă pentru scanarea componentelor.

Câmpuri Autowired:

Clasa are două câmpuri @Autowired:

PasswordEncoder passwordEncoder: Folosit pentru a codifica în mod securizat parolele.

UserRepository userRepository: Folosit pentru a interacționa cu baza de date pentru a salva informațiile utilizatorului.

Implementarea Metodei save:

Clasa implementează metoda save din interfața UserService.

În interiorul metodei, se creează un nou obiect User folosind datele din UserDto-ul furnizat. Parola este codificată folosind PasswordEncoder-ul injectat.Se apelează metoda userRepository.save(user) pentru a persista informațiile utilizatorului în baza de date.

Tipul Returnat:

Metoda save returnează un obiect User, care probabil reprezintă entitatea utilizatorului după ce a fost salvată în baza de date.

În rezumat, clasa UserServiceImpl furnizează o implementare pentru interfața UserService. Este responsabilă pentru gestionarea logicii de afaceri legate de crearea sau înregistrarea utilizatorului. Parola este codificată în mod securizat folosind PasswordEncoder-ul injectat, iar informațiile utilizatorului sunt apoi salvate în baza de date folosind UserRepository-ul injectat. Această clasă este un component esențial al stratului de serviciu al aplicației, gestionând operațiile legate de utilizator.

IV) Concluzii:

Feedback-ul primit de la utilizatori indică o experiență pozitivă. Interfața este intuitivă și ușor de navigat, iar fluxul de înregistrare este clar și simplu. Cu toate acestea, am identificat posibilități de îmbunătățire a aspectului vizual pentru a îmbunătăți atracția vizuală a paginii.

Codul nostru este structurat într-un mod care permite extinderea ușoară a funcționalităților viitoare. Arhitectura modulară permite adăugarea simplă a altor caracteristici sau integrarea cu alte sisteme.

Am primit feedback pozitiv de la utilizatori, dar și sugestii pentru îmbunătățiri. Urmează să implementăm o funcționalitate de confirmare a adresei de e-mail și să optimizăm aspectele de design menționate.

V) Bibliografie:

[1] https://en.wikipedia.org/wiki/Web_page

[2] https://www.javaguides.net/2018/10/user-registration-module-using-springboot-springmvc-springsecurity-hibernate5-thymeleaf-mysql.html

[3] https://www.digitalocean.com/community/tutorials/java-web-application-tutorial-for-beginners

[4] https://planetscale.com/

[5] https://www.jetbrains.com/help/idea/your-first-spring-application.html#add-home-page

[6] https://www.youtube.com/watch?v=7bIx4B5XhIA