**UNIVERSITRTI POLITEKNIK TIRANES**

**FAKULTETI TEKNOLOGJISE INFORMACIONIT**

****

**RELACIONI DETYRES KURSIT**

**LENDA: CLOUD APP**

**E PUNOI: ENEA RAPAJ 2023-2024 E PRANOI: Edurat Torba**

**Dokumenti I Relacionit**

**1.1 Krijimi i web image**

**Qëllimi:**

Ky Dockerfile është krijuar për të krijuar një imazh Docker që shërben një faqe interneti të personalizuar duke përdorur NGINX si server në internet.

**Komponentët:**

1. **Imazhi bazë**: Përdor imazhin bazë zyrtar të NGINX ( nginx:latest), i cili ofron një mjedis të qëndrueshëm dhe të besueshëm për të shërbyer përmbajtje në ueb.
2. **Kopjimi i skedarit**:
   * Kopjon index.htmlnga drejtoria lokale ( ./index.html) në direktorinë e paracaktuar publike html të NGINX ( /usr/share/nginx/html). Kjo siguron që përmbajtja e personalizuar e faqes në internet të shërbehet nga NGINX.
3. **Ekspozimi i portit**:
   * Ekspozon portën 80 (´EXPOSE 80´) për të lejuar akses të jashtëm në serverin e internetit NGINX. Porti 80 është porti standard për trafikun HTTP.
4. **Komanda**:
   * Përdor ubuntu panel udhëzimin për të ekzekutuar NGINX në plan të parë (nginx -g 'daemon off;').
     + nginx: Nis serverin NGINX.
     + -g 'daemon off;': E mban NGINX të funksionojë në plan të parë në mënyrë që Docker të mund ta menaxhojë procesin në mënyrë efektive.

**Marrëdhëniet:**

* **NGINX**: Vepron si komponenti kryesor i imazhit Docker, përgjegjës për shërbimin e kërkesave HTTP.
* **index.html**: Përfaqëson përmbajtjen e personalizuar të uebsajtit që do të shfaqet kur hyni në kontejnerin Docker përmes portit 80.
* **Porta 80**: Ekspozohet për të lejuar sistemet ose përdoruesit e jashtëm të hyjnë në serverin NGINX dhe të shikojnë përmbajtjen e faqes në internet.

**Përdorimi:**

Për të ndërtuar imazhin Docker duke përdorur këtë Dockerfile:

docker build -t my-custom-nginx.

* Zëvendësoni my-custom-nginxme emrin e dëshiruar të figurës.

Për të drejtuar një kontejner bazuar në imazhin e ndërtuar:

docker run -d -p 80:80 my-custom-nginx

 -d: Drejton kontejnerin në modalitetin e shkëputur.

 -p 80:80: Harton portin 80 të kontejnerit në portin 80 në host, duke mundësuar aksesin në serverin NGINX.

 my-custom-nginx: Specifikon emrin e imazhit Docker për t'u ekzekutuar si një kontejner.

#### Mirëmbajtja:

Sigurohuni që index.htmlose çdo skedar tjetër i kërkuar të përditësohet dhe kopjojini ato në imazhin e Docker ( COPY ./index.html /usr/share/nginx/html) nëse bëhen ndryshime në përmbajtjen e faqes së internetit.

#### Siguria:

 **Kontrolli i aksesit**: Merrni parasysh zbatimin e kontrolleve të duhura të aksesit dhe mekanizmave të vërtetimit nëse faqja e internetit përmban informacione të ndjeshme.

 **Përditësimet NGINX**: Përditësoni rregullisht imazhin bazë të NGINX (´nginx:latest´) për të përfituar nga arnimet dhe përmirësimet e sigurisë.

**Monitorimi:**

* Zbatoni mjete monitorimi për të gjurmuar regjistrat e NGINX dhe metrikat e performancës për të siguruar që faqja e internetit të funksionojë pa probleme dhe me efikasitet.

**Konkluzioni:**

Ky Dockerfile paketon në mënyrë efikase një faqe interneti të personalizuar me NGINX, duke e bërë atë të aksesueshëm përmes portit 80. Duke ndjekur praktikat më të mira në sigurinë, mirëmbajtjen, shkallëzueshmërinë dhe monitorimin, shembulli NGINX i kontejneruar mund të sigurojë një platformë të fuqishme për pritjen e përmbajtjes së uebit.Top of Form

Bottom of Form

**1.2 Krijimi i Postgres Image**

**Objektivi:**  
Ky Dockerfile është krijuar për të krijuar një imazh të personalizuar PostgreSQL Docker që konfiguron një konfigurim specifik të bazës së të dhënave për një aplikacion cloud.

**Komponentët:**

1. **Imazhi bazë:**
   * **Burimi:** Official PostgreSQL Docker image (postgres:latest)
   * **Qëllimi:** Ofron një mjedis të besueshëm dhe të mirëmbajtur të serverit PostgreSQL.
2. **Variablat e mjedisit:**
   * **POSTGRES\_USER:** postgres
     + **Qëllimi:** Specifikon emrin e parazgjedhur të përdoruesit për PostgreSQL.
   * **POSTGRES\_PASSWORD:** root
     + **Qëllimi:** Vendos fjalëkalimin për përdoruesin PostgreSQL.
   * **POSTGRES\_DB:** cloudapp
     + **Qëllimi:** Specifikon emrin e bazës së të dhënave të paracaktuar që do të krijohet brenda PostgreSQL.
3. **Portet e ekspozuara:**
   * **5432 (porta e parazgjedhur PostgreSQL):** Hartuar në portin pritës 2022
     + **Qëllimi:** Ekspozon portën e paracaktuar PostgreSQL për të lejuar lidhjet nga aplikacionet që funksionojnë jashtë kontejnerit Docker.
4. **Skript opsional i inicializimit:**
   * **Përshkrimi:** Një skript i personalizuar i inicializimit SQL ( init.sql) mund të përfshihet në Dockerfile për të ekzekutuar konfigurimin e skemës së bazës së të dhënave ose detyrat e inicializimit kur kontejneri niset për herë të parë.
   * **Udhëzim për çkomentim:** Komanda e komentuar COPY( # COPY ./init.sql /docker-entrypoint-initdb.d/) duhet të mos komentohet për të aktivizuar këtë veçori. Kjo komandë udhëzon Docker-in të kopjojë init.sqlskriptin në /docker-entrypoint-initdb.d/drejtorinë brenda kontejnerit, i cili ekzekutohet automatikisht gjatë inicializimit të PostgreSQL.
   * **Qëllimi:** Lehtëson konfigurimin automatik të skemës së kërkuar të bazës së të dhënave ose futjen e të dhënave fillestare, duke përmirësuar vendosjen e postgreSQL të kontejneruar për aplikacionet cloud.

**Udhëzime përdorimi:**

* **Build Command:** Përdoreni docker build -t custom-postgres . për të ndërtuar imazhin Docker nga ky Dockerfile.
* **Komanda e ekzekutimit:** Ekzekutoni docker run -p 2022:5432 --name postgres-db -d custom-postgrespër të ekzekutuar bazën e të dhënave PostgreSQL të kontejneruar, duke ekspozuar portin 2022 në sistemin pritës për akses të jashtëm.

**Shënime:**

* Sigurohuni që init.sqlskripti, nëse përdoret, është i formatuar saktë dhe përmban komandat e nevojshme SQL të pajtueshme me PostgreSQL.
* Rregulloni versionin PostgreSQL ( latestnë këtë rast) bazuar në kërkesat specifike dhe pajtueshmërinë me varësitë e aplikacionit.

**Konkluzioni:**

Ky Dockerfile ofron një qasje të efektshme për të vendosur një shembull të bazës së të dhënave PostgreSQL të përshtatur për nevojat e aplikacionit cloud, me fleksibilitet për të inicializuar konfigurimet e personalizuara të bazës së të dhënave nëse kërkohet, duke përmirësuar menaxhimin e bazës së të dhënave të kontejneruar dhe proceset e vendosjes.

**1.3 Doker Compose**

Ky dokument përshkruan marrëdhëniet dhe varësitë midis shërbimeve të përcaktuara në docker-compose.yml.

**Shërbimet:**

1. **serveri i internetit:**
   * **Përshkrimi:** Ky shërbim drejton një server në internet.
   * **Ndërtoni rrugën:** ./webserver
   * **Portet e ekspozuara:**
     + 80:80(Mikpritës

)

* + **Varësitë:**
    - Varet nga databaseshërbimi.

1. **baza e të dhënave:**
   * **Përshkrimi:** Ky shërbim drejton një server të bazës së të dhënave.
   * **Ndërtoni rrugën:** ./database
   * **Portet e ekspozuara:**
     + 5432:5432(Mikpritës

)

* + **Varësitë:**
    - Nuk ka varësi të qartë nga shërbimet e tjera.

**Marrëdhëniet:**

* **serveri në internet** varet nga **baza e të dhënave** :
  + Shërbimi webservermbështetet në databasedisponueshmërinë dhe fillimin e shërbimit përpara se të mund të inicializohet ose të funksionojë siç duhet. Kjo varësi menaxhohet nga mekanizmi i porosisë së fillimit të shërbimit të Docker Compose.

**Komunikimi:**

* **serveri në internet** në **bazën e të dhënave** :
  + Shërbimi webserverkomunikon me databaseshërbimin përmes rrjetit (me gjasë duke përdorur rrjetin e brendshëm Docker) në port 5432. Ky është porti i parazgjedhur i PostgreSQL, duke supozuar se databasepo ekzekuton një bazë të dhënash PostgreSQL.

**Shënime:**

* Sigurohuni që databaseshërbimi të funksionojë dhe të funksionojë përpara se të filloni webservershërbimin për të shmangur problemet e lidhjes.
* Portat 80dhe 5432janë të ekspozuara në makinën pritës ( localhost) dhe të hartuara në portat përkatëse të kontejnerit.

Ky dokument i marrëdhënies ofron një përmbledhje të mënyrës se si ndërveprojnë shërbimet webserverdhe databaseshërbimet brenda konfigurimit tuaj të Docker Compose. Rregulloni ose zgjeroni atë në bazë të konfigurimeve shtesë ose kërkesave specifike të aplikacionit tuaj.

**2 Krijimi Crudit me Spring**

Kemi krijuar nje crud me spring boot qe eshte nje library e javes application dhe postman. E cila sherben per funksione si regjistrim user apo admin, autentifikimi userit dhe modifikimi te dhenave te userit. Ku baza e aplikacionit eshte krijimi i entity dhe sigurise passwordit. Ku security eshte nje folder security qe perbehet nga tre file class: UnathurizationEntrypoint.java, DisableSecurityConfig.java, SecurityConfig.java. Ku keto do shpjegohen ne vijim.

**UnathurizationEntrypoint.java**

Kjo pjesë e kodit Java është një zbatim i personalizuar i Spring Security's AuthenticationEntryPoint. Ndërfaqja AuthenticationEntryPointpërdoret për të filluar një skemë vërtetimi (zakonisht duke dërguar një përgjigje të paautorizuar 401) kur vërtetimi dështon.

Kodi perbehet nga keto pjese:

* + 1. Paketa dhe importet:

package com.basicauth.app.security;

import java.io.IOException;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

 Kodi është i organizuar brenda com.basicauth.app.securitypaketës.

 Importet e nevojshme janë të përfshira nga Spring Security dhe Jakarta Servlet API.

* + 1. Shënimi i komponentit:

@Component

public class UnauthorizedEntrypoint implements AuthenticationEntryPoint {

* Klasa UnauthorizedEntrypointështë shënuar me @Component, duke treguar se është një komponent i menaxhuar nga Spring dhe duhet të zbulohet dhe regjistrohet automatikisht gjatë skanimit të komponentëve.
  + 1. Përkufizimi i klasës:
  + ´UnauthorizedEntrypoint`implementon ´AuthenticationEntryPoint´ ndërfaqen e ofruar nga Spring Security.

4.Anulimi i metodës:

@Override

public void commence(HttpServletRequest request, HttpServletResponse response,

AuthenticationException authException) throws IOException, ServletException {

response.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);

}

 Metoda commenceështë anashkaluar nga AuthenticationEntryPointndërfaqja. Kjo metodë quhet kur një përdorues përpiqet të hyjë në një pikë fundore të sigurt pa vërtetimin e duhur.

**Parametrat:**

* HttpServletRequest request: Përfaqëson kërkesën HTTP të bërë nga klienti.
* HttpServletResponse response: Përfaqëson përgjigjen HTTP që do t'i dërgohet klientit.
* AuthenticationException authException: Përfaqëson përjashtimin që shkaktoi dështimin e vërtetimit.

**Funksionaliteti:**

* Brenda commencemetodës, statusi i përgjigjes është vendosur në HttpServletResponse.SC\_UNAUTHORIZED, që korrespondon me kodin e statusit HTTP 401 (I paautorizuar). Kjo informon klientin se vërtetimi kërkohet dhe ka dështuar.
* Në mënyrë tipike, pas vendosjes së kodit të statusit, mund të shtohet logjikë shtesë për të personalizuar më tej përgjigjen, si p.sh. vendosja e titujve ose ofrimi i një mesazhi gabimi personal.

### **Përmbledhje:**

* **Qëllimi:** Klasa UnauthorizedEntrypointështë krijuar për të trajtuar situata ku një përdorues i paautentikuar përpiqet të aksesojë një burim të sigurt.
* **Sjellja:** Kur aktivizohet (pas dështimit të vërtetimit), ai vendos statusin e përgjigjes HTTP në 401 (i paautorizuar).
* **Përdorimi:** Kjo klasë mund të konfigurohet në një konfigurim Spring Security për të siguruar që përpjekjet për akses të paautorizuar të marrin kodin e duhur të statusit HTTP dhe trajtimin e përgjigjeve.

Ky konfigurim siguron që aplikacioni juaj Spring Security të përgjigjet saktë kur përdoruesit përpiqen të aksesojnë burimet e mbrojtura pa vërtetimin e duhur.

Top of Form

Bottom of Form

**SecurityConfig.java**

Ky fragment kodi është një konfigurim Spring Security në një aplikacion Java duke përdorur Spring Boot. Le të zbërthejmë përbërësit kryesorë dhe funksionalitetin e tyre:

1. \*\*Shënime dhe importe\*\*:

- `@Configuration`: Tregon që kjo klasë ofron konfigurimin e bean-it të Spring.

- `@ConditionalOnProperty`: Siguron që ky konfigurim të aktivizohet vetëm kur një veçori specifike (`security.enabled`) vendoset në `“true”` në vetitë e aplikacionit ose skedarët e konfigurimit.

- Importon klasa të ndryshme Spring Security dhe komponentë të nevojshëm për konfigurimin e veçorive të sigurisë.

2. \*\*Varësitë e lidhura automatikisht\*\*:

- `@Autowired private UserDetail userDetail;`: Injekton një shembull të `UserDetail`, i cili supozohet se ofron informacione të lidhura me përdoruesit dhe detaje të vërtetimit.

- `@Autowired private UnauthorizedEntrypoint unauthorizedEntrypoint;`: Injekton një mbajtës të paautorizuar të pikës hyrëse që do të përdoret për trajtimin e dështimeve të vërtetimit.

3. \*\*PasswordEncoder Bean\*\*:

- `@Bean publike statike PasswordEncoder passwordEncoder() {...}`: Përcakton një bean për `BCryptPasswordEncoder`, i cili përdoret për kodimin e sigurt të fjalëkalimeve.

4. \*\*SecurityFilterChain Bean\*\*:

- `@Bean public SecurityFilterChain filterSecurity(HttpSecurity http) hedh Përjashtim {...}`: Konfiguron filtrat e sigurisë dhe rregullat duke përdorur `HttpSecurity`. Kjo metodë konfiguron konfigurimin e sigurisë për pika dhe role të ndryshme HTTP.

Brenda kësaj metode:

* http.csrf(...);: Çaktivizon mbrojtjen CSRF për thjeshtësi. Në një aplikacion prodhimi, mbrojtja CSRF duhet të konfigurohet me kujdes bazuar në nevojat e aplikacionit.
* http.securityMatcher(...);: Konfiguron rregullat e sigurisë bazuar në modelet e URL-së.
* .authorizeHttpRequests(...);: Specifikon rregullat e autorizimit për pika të ndryshme fundore.
* .requestMatchers("/admin-page").hasRole("ADMIN"): Kërkon që përdoruesit me rolin "ADMIN" të kenë qasje në /admin-page.
* .requestMatchers("/index").hasAnyRole("USER", "ADMIN"): Kërkon nga përdoruesit me role "USER" ose "ADMIN" për të hyrë në /indeks.
* .requestMatchers("/contact").permitAll(): Lejon akses publik në /contact.
* .requestMatchers("/register\*").permitAll(): Lejon akses publik në URL-të që fillojnë me /register.
* .requestMatchers("/login").permitAll(): Lejon akses publik në /login.
* .requestMatchers("/profile").permitAll(): Lejon akses publik në /profile.
* .requestMatchers("/{id}").permitAll(): Lejon akses publik në çdo segment URL (ndryshore të rrugës).
* .anyRequest().authenticated(): Kërkon vërtetim për çdo kërkesë tjetër që nuk është specifikuar më sipër.
* .httpBasic(...);: Konfiguron vërtetimin bazë HTTP dhe specifikon UnauthorizedEntrypoint si mbajtësin e pikës hyrëse të vërtetimit.
* .userDetailsService(userDetail);: Vendos shërbimin e personalizuar UserDetail për të ngarkuar detajet e përdoruesit për vërtetim.
* return http.build();: Ndërton dhe kthen objektin e konfiguruar HttpSecurity.

1. WebSecurityCustomizer Bean:

* @Bean publike WebSecurityCustomizer webSecurityCustomizer() {...}: Konfiguron cilësimet e sigurisë në ueb.
* .ignoring().requestMatchers(new AntPathRequestMatcher("/h2-console/\*\*"));: Injoron konfigurimet e sigurisë për kërkesat që përputhen me /h2-console/\*\*, të përdorura zakonisht për aksesin e konsolës së bazës së të dhënave H2.

Në përmbledhje, ky kod konfiguron Spring Security me kontrollin e aksesit të bazuar në role, kodimin e fjalëkalimit, mbrojtjen CSRF të çaktivizuar dhe trajtimin me porosi për URL dhe përjashtime të caktuara. Është një konfigurim tipik për sigurimin e një aplikacioni ueb duke përdorur Spring Security në një mjedis Spring Boot.

**DisableSecurityConfig.java**

Ky kod është një klasë konfigurimi Spring Security e quajtur 'DisableSecurityConfig'. Le të zbërthejmë se çfarë bën secila pjesë:

1. \*\*Shënimet dhe ConditionalOnProperty\*\*:

- `@Configuration`: Tregon që kjo klasë ofron konfigurime Spring Bean.

- `@ConditionalOnProperty(value = "security.enabled", haveValue = "false")`: Ky shënim siguron që ky konfigurim të zbatohet vetëm kur vetia `security.enabled` është caktuar në "false". Nëse kjo veti është "e vërtetë" ose nuk specifikohet, kjo klasë konfigurimi nuk do të përdoret.

2. \*\*Konfigurimi i Zinxhirit të Filterit të Sigurisë\*\*:

- `@Bean public SecurityFilterChain filterSecurity(HttpSecurity http) hedh Përjashtim`: Kjo metodë konfiguron objektin `HttpSecurity` të Spring Security.

- `http.csrf(htpSecurity -> htpSecurity.disable())`: Çaktivizon mbrojtjen CSRF (Cross-Site Request Forgery). Mbrojtja CSRF zakonisht aktivizohet si parazgjedhje në Spring Security, por ky konfigurim e çaktivizon atë.

- `http.authorizeHttpRequests((authorize) -> autorize.anyRequest().permitAll())`: Konfiguron autorizimin në mënyrë që çdo kërkesë HTTP (`anyRequest()`) të lejohet (`permitAll()`). Kjo do të thotë që të gjitha kërkesat do të lejohen pa ndonjë verifikim të vërtetimit ose autorizimit.

Pas konfigurimit të `HttpSecurity`, metoda kthen `http.build()`, e cila ndërton dhe kthen `SecurityFilterChain` të konfiguruar.

3. \*\*PasswordEncoder Bean\*\*:

- `@Bean publike statike PasswordEncoder passwordEncoder()`: Përcakton një Spring Bean të quajtur `passwordEncoder()` që ofron një `PasswordEncoder`.

- `return new BCryptPasswordEncoder();`: Krijon dhe kthen një shembull të `BCryptPasswordEncoder`. Ky është një kodues fjalëkalimi i ofruar nga Spring Security për hashimin e fjalëkalimeve duke përdorur funksionin e hashimit bcrypt.

### Përmbledhje:

- Kjo klasë konfigurimi (`DisableSecurityConfig`) është krijuar për të çaktivizuar të gjitha masat e sigurisë (kërkesat e mbrojtjes dhe autorizimit të `csrf`) në një aplikacion të aktivizuar nga Spring Security, por vetëm nëse vetia `security.enabled` është vendosur në mënyrë eksplicite në 'false' .

- Për më tepër, ai ofron një fasule `PasswordEncoder` (`BCryptPasswordEncoder`) të cilin aplikacionet mund ta përdorin për hashimin e sigurt të fjalëkalimeve.

\*\*Konsiderata të rëndësishme \*\*:

- Çaktivizimi i sigurisë duhet të bëhet me kujdes dhe vetëm në mjedise të caktuara (si zhvillimi ose testimi) ku siguria nuk është shqetësim.

- Sigurohuni që masat e duhura të sigurisë janë riaktivizuar përpara se të vendoseni në prodhim.

- Konfigurimi i "PasswordEncoder" siguron që fjalëkalimet të fshihen në mënyrë të sigurt përpara ruajtjes, duke promovuar trajtimin e sigurt të kredencialeve të përdoruesit.

**2.1 Krijimi regjistrimit userit**

Ky dokument përshkruan marrëdhënien midis komponentëve në një aplikacion themelor të vërtetimit të zbatuar në Java duke përdorur Spring Boot. Aplikacioni përfshin entitete për profilet e përdoruesve, shërbime për regjistrimin e përdoruesve, një numër për rolet e përdoruesve dhe një kontrollues për trajtimin e kërkesave për regjistrimin e përdoruesve.

1. Entity: UserProfile

- Qëllimi: Përfaqëson një profil përdoruesi të ruajtur në bazën e të dhënave.

- Atributet:

- `id`: ID e gjeneruar automatikisht për profilin e përdoruesit.

- `emri`: emri i përdoruesit.

- `atesi`: Atesi i përdoruesit (i paqartë nga konteksti).

- `mbiemri`: mbiemri i përdoruesit.

- `email`: Adresa e emailit të përdoruesit (përdoret si një identifikues unik).

- `numri i telefonit`: numri i telefonit të përdoruesit.

- `fjalëkalimi`: Fjalëkalimi i përdoruesit (i koduar për siguri).

- `role`: Roli i përdoruesit, ose `ROLE\_USER` ose `ROLE\_ADMIN`.

2. Enum: Role

- Qëllimi: Përcakton rolet e përdoruesve.

- Vlerat:

- `ROLE\_USER`: Roli standard i përdoruesit.

- `ROLE\_ADMIN`: Roli i administratorit.

3. Service: RegistrationService

- Qëllimi: Trajton operacionet e regjistrimit të përdoruesve.

- Dependencies:

- `registerRepo`: Depo për aksesimin e të dhënave të përdoruesit.

- `pwdEncoder`: kodues i fjalëkalimit për sigurimin e fjalëkalimeve të përdoruesve.

- Methods:

- `registerUser(UserProfile user)`: Regjistron një përdorues të ri me `ROLE\_USER`.

- `registerAdmin(UserProfile admin)`: Regjistron një përdorues të ri me `ROLE\_ADMIN`.

4. Repository: RegisterNewUserRepository

- Qëllimi: Ofron operacione CRUD për entitetet "Profili i Përdoruesit".

Controller: RegisterUserController

Qëllimi: Trajton kërkesat HTTP në lidhje me regjistrimin e përdoruesit.

Pikat përfundimtare:

POST /regjistrohu: Pika përfundimtare për regjistrimin e përdoruesve të rinj.

Funksionaliteti:

Vlerëson të dhënat e përdoruesit duke përdorur UserValidator.

Përdor RegistrationService për të regjistruar përdoruesit.

Rikthen përgjigjet e duhura bazuar në suksesin e regjistrimit ose dështimin e vërtetimit.

Marrëdhëniet

UserProfile <-> Role: Çdo entitet UserProfile ka një atribut roli që përcakton nëse përdoruesi është një përdorues standard apo një administrator. Roli përcaktohet duke përdorur numrin e roleve (ROLE\_USER ose ROLE\_ADMIN).

RegistrationService <-> RegisterNewUserRepository: Shtresa e shërbimit (RegistrationService) ndërvepron me depon (RegisterNewUserRepository) për të ruajtur të dhënat e përdoruesit. Ai gjithashtu përdor PasswordEncoder për ruajtjen e sigurt të fjalëkalimeve.

RegisterUserController <-> RegistrationService: Kontrolluesi (RegisterUserController) delegon kërkesat për regjistrimin e përdoruesve te RegistrationService, i cili kryen logjikën e biznesit si vendosja e roleve dhe kodimi i fjalëkalimeve përpara se të ruajë përdoruesit.

Konkluzioni

Kjo arkitekturë ndan shqetësimet në mënyrë efektive, me përgjegjësi të qarta për entitetet, numrat, shërbimet, depot dhe kontrollorët. Përdorimi i numrave për role siguron qartësi dhe mirëmbajtje në përcaktimin e lejeve të përdoruesit. Shtresa e shërbimit përmbledh logjikën e biznesit, ndërsa kontrolluesi menaxhon pikat fundore të HTTP dhe vërtetimin e hyrjes, duke promovuar një sistem vërtetimi të fortë dhe të shkallëzuar.

**2.2 Authentikimi userit**

Ky dokument përshkruan marrëdhënien midis komponentëve në një aplikacion themelor të vërtetimit të zbatuar në Java duke përdorur Spring Boot. Aplikacioni përfshin entitete për profilet e përdoruesve, shërbime për regjistrimin e përdoruesve, një numër për rolet e përdoruesve dhe një kontrollues për trajtimin e kërkesave për regjistrimin e përdoruesve.

Componentët:

1. Entiteti: User Profile

- Qëllimi: Përfaqëson një profil përdoruesi të ruajtur në bazën e të dhënave.

- Atributet:

- `id`: ID e gjeneruar automatikisht për profilin e përdoruesit.

- `emri`: emri i përdoruesit.

- `atesi`: Atesi i përdoruesit (i paqartë nga konteksti).

- `mbiemri`: mbiemri i përdoruesit.

- `email`: Adresa e emailit të përdoruesit (përdoret si një identifikues unik).

- `numri i telefonit`: numri i telefonit të përdoruesit.

- `fjalëkalimi`: Fjalëkalimi i përdoruesit (i koduar për siguri).

- `role`: Roli i përdoruesit, ose `ROLE\_USER` ose `ROLE\_ADMIN`.

2. Enum: Role

- Qëllimi: Përcakton rolet e përdoruesve.

- Vlerat:

- `ROLE\_USER`: Roli standard i përdoruesit.

- `ROLE\_ADMIN`: Roli i administratorit.

3. **Service: RegistrationService**

- Qëllimi: Trajton operacionet e regjistrimit të përdoruesve.

- dependecy:

- `registerRepo`: Depo për aksesimin e të dhënave të përdoruesit.

- `pwdEncoder`: kodues i fjalëkalimit për sigurimin e fjalëkalimeve të përdoruesve.

- \*\* Methods: \*\*

- `registerUser(UserProfile user)`: Regjistron një përdorues të ri me `ROLE\_USER`.

- `registerAdmin(UserProfile admin)`: Regjistron një përdorues të ri me `ROLE\_ADMIN`.

4. Repository: RegisterNewUserRepository

- Qëllimi: Ofron operacione CRUD për entitetet "Profili i Përdoruesit".

5. Controler: RegisterUserController

- Qëllimi: Trajton kërkesat HTTP në lidhje me regjistrimin e përdoruesit.

- Pika përfundimtare:

- `POST /regjistrohu`: Pika përfundimtare për regjistrimin e përdoruesve të rinj.

- Funksionaliteti:

- Vlerëson hyrjen e përdoruesit duke përdorur `UserValidator`.

- Përdor 'RegistrationService' për të regjistruar përdoruesit.

- Rikthen përgjigjet e duhura bazuar në suksesin e regjistrimit ose dështimin e vërtetimit.

**Marrëdhëniet**

- Profili i Përdoruesit <-> Roli: Çdo ent "Profili i Përdoruesit" ka një atribut "roli" që përcakton nëse përdoruesi është përdorues standard apo administrator. Roli përcaktohet duke përdorur numrin "Role" ("ROLE\_USER" ose "ROLE\_ADMIN").

- RegistrationService <-> RegisterNewUserRepository: Shtresa e shërbimit (`RegistrationService`) ndërvepron me depon (`RegisterNewUserRepository`) për të ruajtur të dhënat e përdoruesit. Ai përdor gjithashtu "PasswordEncoder" për ruajtjen e sigurt të fjalëkalimeve.

- RegisterUserController <-> RegistrationService: Kontrolluesi (RegisterUserController) delegon kërkesat për regjistrimin e përdoruesve te RegistrationService, i cili kryen logjikën e biznesit si vendosja e roleve dhe kodimi i fjalëkalimeve përpara se të ruajë përdoruesit.

**Konkluzioni**

Kjo arkitekturë ndan shqetësimet në mënyrë efektive, me përgjegjësi të qarta për entitetet, numrat, shërbimet, depot dhe kontrollorët. Përdorimi i numrave për role siguron qartësi dhe mirëmbajtje në përcaktimin e lejeve të përdoruesit. Shtresa e shërbimit përmbledh logjikën e biznesit, ndërsa kontrolluesi menaxhon pikat fundore të HTTP dhe vërtetimin e hyrjes, duke promovuar një sistem vërtetimi të fortë dhe të shkallëzuar.

**2.3 Modifikimi Userit**

Componentët:

1. UserProfileService

- Qëllimi: Menaxhon operacionet në lidhje me profilet e përdoruesve.

- Vendndodhja: `com.basicauth.app.service.UserProfileService`

- Dependeces:

- `RegisterNewUserRepository`: Trajton operacionet e aksesit të të dhënave për profilet e përdoruesve.

2. UserProfileController

- Qëllimi: Ekspozon pikat fundore për ndërveprim me profilet e përdoruesve nëpërmjet kërkesave HTTP.

- Vendndodhja: `com.basicauth.app.entity.UserProfileController`

- Dependeces:

- `UserProfileService`: Përdor metodat e shërbimit për të kryer operacione në profilet e përdoruesve.

3. UserProfile

- Qëllimi: Përfaqëson strukturën e të dhënave të një profili përdoruesi.

- Vendndodhja: `com.basicauth.app.entity.UserProfile`

- Atributet:

- `id`: Identifikues unik për profilin e përdoruesit.

- `emri`: Emri i përdoruesit.

- `email`: Adresa e emailit të përdoruesit.

- Atribute të tjera sipas nevojës.

4. RegisterNewUserRepository:

- Qëllimi: Trajton operacionet e qëndrueshmërisë për profilet e përdoruesve.

- Vendndodhja: `com.basicauth.app.repository.RegisterNewUserRepository`

- Metodat e Zbatuara:

- `findById(Long id)`: Merr një profil përdoruesi sipas identifikuesit të tij unik.

- `ruaj(profili i përdoruesit)`: Ruan ose përditëson një profil përdoruesi në bazën e të dhënave.

**Ndërveprimet**

- UserProfileService <-> RegisterNewUserRepository:

- `UserProfileService` përdor `RegisterNewUserRepository` për të marrë dhe përditësuar profilet e përdoruesve të ruajtura në bazën e të dhënave.

- UserProfileController <-> UserProfileService:

- "UserProfileController" komunikon me "UserProfileService" për të trajtuar kërkesat HTTP ("GET", "PUT") që lidhen me profilet e përdoruesve.

Rrjedha e operacioneve:

1. Get /profil/{id}

- Trajtimi i kërkesës: Marrë nga `UserProfileController`.

- Veprim: Kërkon metodën "getUserProfile(id)" nga "UserProfileService".

- Përgjigja: Kthen të dhënat e 'Profilit të Përdoruesit' të mbështjella në një 'Entitet Response'.

2. PUT /profile/{id}

- Trajtimi i kërkesës: Marrë nga `UserProfileController`.

- Veprim: Nxjerrë 'Profilin e Përdoruesit' nga trupi i kërkesës.

- Thirret: metoda `updateUserProfile(id, userProfile)` nga `UserProfileService`.

- Përgjigja: Kthen të dhënat e përditësuara të "Profilit të Përdoruesit" të mbështjella në një "Entitet të Përgjigjes".

**Shënime**:

* UserProfileService përfshin logjikën e biznesit në lidhje me profilet e përdoruesve, duke përfshirë operacionet e rikthimit dhe përditësimit.
* UserProfileController vepron si një shtresë ndërfaqeje për kërkesat HTTP, duke i përkthyer ato në thirrjet e metodës së shërbimit.
* Operacionet e aksesit të të dhënave menaxhohen përmes RegisterNewUserRepository, duke siguruar ndarjen e shqetësimeve midis logjikës së biznesit dhe qëndrueshmërisë.

Ky dokument i marrëdhënieve ofron një pasqyrë të qartë të mënyrës se si komponentët ndërveprojnë brenda aplikacionit të dhënë Spring Boot për të menaxhuar profilet e përdoruesve në mënyrë efektive.

**2.4 Krijimi regexit**

Ky dokument përshkruan marrëdhëniet dhe funksionalitetet e klasës `UserValidator` brenda paketës `com.basicauth.app.entity`.

Klasa `UserValidator`:

- Qëllimi: Ofron metoda statike për të vërtetuar fusha të ndryshme të hyrjes së përdoruesit si emri, numri i telefonit, data e lindjes, emaili dhe fjalëkalimi duke përdorur shprehje të rregullta (regex).

- Vendndodhja: `com.basicauth.app.entity.UserValidator`

Modele Regex të përdorura

1. NAME\_REGEX

- Qëllimi: Vërteton emrin dhe mbiemrin.

- Modeli: `^[a-zA-Z]{1,20}$`

-Përdorimi: Përdoret për vërtetimin e emrave për t'u siguruar që ato përbëhen vetëm nga shkronja dhe janë nga 1 deri në 20 karaktere të gjata.

2. PHONE\_REGEX

- Qëllimi: Vërteton numrat e telefonit.

- Modeli: `^\+355\d{9}$`

- Përdorimi: Vlerëson në mënyrë specifike numrat e telefonit shqiptarë që fillojnë me `+355`.

3. DATE\_REGEX

- \*\*Qëllimi\*\*: Vërteton datën e lindjes.

- \*\*Modeli \*\*: `\d{4}-\d{2}-\d{2}`

- \*\*Përdorimi\*\*: Siguron që data të jetë në formatin `vvvv-MM-dd`.

4. EMAIL\_REGEX

- Qëllimi: Vlereson adresat e emailit.

- Modeli: `^(?!.\*?@[\\w.]+(?:\\.[a-zA-Z]{2,}){2,})(?:[ a-zA-Z0-9\_+&\*-]+(?:\\.[a-zA-Z0-9\_+&\*-]+)\*@[a-zA-Z0-9-]+(?: \\.[a-zA-Z0-9-]+)\*(?:\\.[a-zA-Z]{2,7}))$`

- \*\*Përdorimi\*\*: Kontrollon për një format të vlefshëm adrese emaili, ndërkohë që sigurohet që nuk përfundon me domene të shumta të nivelit të lartë.

5. PASSWORD\_REGEX

- Qëllimi: Vlerëson fjalëkalimet.

-Modeli:`^(?=.\*[0-9])(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[@#$%^& +=])(?=\S+$).{8,}$`

- Përdorimi: Kërkon që fjalëkalimet të kenë të paktën 8 karaktere, duke përfshirë të paktën një shifër, një shkronjë të vogël, një shkronjë të madhe dhe një karakter të veçantë.

Metodat në `UserValidator`

- validateFirstName(string firstname)

- Qëllimi: Vlerëson formatin e emrit të parë duke përdorur `NAME\_REGEX`.

- validateAtesia(emri i vargut)

- Qëllimi: Kjo metodë duket të jetë një dublikatë e `validateFirstName(String firstName)` për shkak të zbatimit identik dhe regex.

- validateMbiemrin(mbiemri i vargut)

- Qëllimi: Vlerëson formatin e mbiemrit duke përdorur `NAME\_REGEX`.

- vleftëso numrin e telefonit (Numri i telefonit të vargut)

-Qëllimi: Vlerëson formatin e numrit të telefonit duke përdorur `PHONE\_REGEX`.

- validateDateOfBirth(string dob)

- Qëllimi: Vlerëson formatin e datës së lindjes duke përdorur "DATE\_REGEX".

- validateEmail(e-mail varg)

- Qëllimi: Vlerëson formatin e adresës së emailit duke përdorur `EMAIL\_REGEX`.

- validatePassword(fjalëkalim i vargut)

- Qëllimi: Vlerëson formatin e fjalëkalimit duke përdorur `PASSWORD\_REGEX`.

**Shënime të përdorimit**

* Klasa UserValidator ofron metoda statike të ripërdorshme për vërtetimin e fushave të zakonshme të hyrjes së përdoruesit bazuar në modelet e paracaktuara regex.
* Çdo metodë vërtetimi korrespondon me një model specifik regex të përshtatur për të siguruar përputhjen e formatit të hyrjes.
* Metodat statike lejojnë që këto verifikime të aksesohen pa instancuar klasën UserValidator, duke promovuar lehtësinë e përdorimit dhe efikasitetin në operacionet e vërtetimit.
* Ky dokument përshkruan se si funksionon klasa UserValidator brenda aplikacionit për të zbatuar standardet e formatit të të dhënave për informacionin e lidhur me përdoruesit duke përdorur shprehje të rregullta.