**InMemory Authentication**

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration extends WebSecurityConfigurerAdapter {

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication()

.withUser("abc")

.password("abc") // cleartext

.authorities("ADMIN")

.and()

.withUser("xyz")

.password("xyz") // cleartext

.authorities("USER");

or

InMemoryUserDetailsManager userDetails= new InMemoryUserDetailsManager();

UserDetails user1=User

.withUsername("aaa")

.password("aaa")

.authorities("ADMIN")

.build();

UserDetails user2=User

.withUsername("bbb")

.password("bbb")

.authorities("USER")

.build();

userDetails.createUser(user1);

userDetails.createUser(user2);

auth.userDetailsService(userDetails);

}

@Bean

public PasswordEncoder getPasswordEncoder()

{

return NoOpPasswordEncoder.getInstance();

}

@Override //Authorisation

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.anyRequest().authenticated()

.and() .formLogin().loginProcessingUrl("/login").successForwardUrl("/listOfCustomers").permitAll()

.and()

.cors().and().csrf().disable();

}

**Password Encoder**

PasswordEncoder encoder= PasswordEncoderFactories.createDelegatingPasswordEncoder()

UserDetails user1=User

.withUsername("aaa")

.password(encoder.encode("aaa"))

.authorities("ADMIN")

.build();

**Comment the NoPasswordEncoder**

**Role Based Authorization**

step 1 : make a fresh spring boot application using spring initializer

step 2 : add dependency for

1)web

2)security

3) jpa

4) mySql connector java

5)Lombok

6)Thymeleaf

step 3 : import the project in eclipse

add the following dependency(if you are using JSP as frontend)

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-taglibs</artifactId>

<version>5.2.2.RELEASE</version>

</dependency>

application.properties

server.port=8084

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://localhost:3306/rbadb

spring.datasource.username=root

spring.datasource.password=

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

#server.servlet.context-path=/LibraryManagement

#spring.mvc.view.prefix: /WEB-INF/views/

#spring.mvc.view.suffix: .jsp

#spring.security.user.name=admin

#spring.security.user.password=admin

step 4:in the "model/entity" package create 2 classes ,named **Role** and **User** with many to many mapping

@Entity

@Data

**public** **class** Role {

@Id

@Column(name = "role\_id")

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Integer id;

@Column(name="name")

**private** String name;

}

------------

@Entity

@Data

**public** **class** User {

@Id

@Column(name = "user\_id")

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

@Column(name="username")

**private** String username;

@Column(name="password")

**private** String password;

@ManyToMany(cascade = CascadeType.***ALL***, fetch = FetchType.***EAGER***)

@JoinTable(

name = "users\_roles",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "role\_id")

)

**private** List<Role> roles = **new** ArrayList<>();

}

------------------

step 5 as setup the database and add entries to the User, Role & User\_Role tables

**INSERT** **INTO** `execustomerdb`.`role` (

`roleid` ,  
`rolename`

)  
**VALUES** (

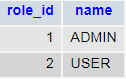
**NULL** , 'ADMIN'

), (

**NULL** , 'USER'

);

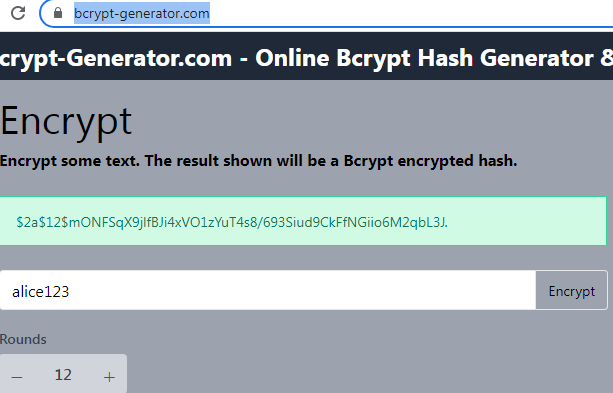
**select \* from Role**



for inserting the password , we need to encrypt it before inserting in the database

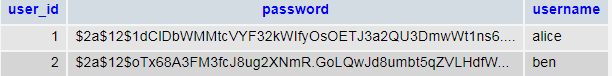
website : https://bcrypt-generator.com/

NOTE : use 12 Rounds of encryptions



insert this encrypted password in the users table

**select \* from User**



**INSERT** **INTO** `execustomerdb`.`user` (

`userid` ,  
`password` ,  
`username`

)  
**VALUES** (

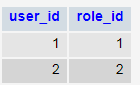
**NULL** , '$2a$12$Rhfq2LBfxdKPwUoeysqwduWmIBbMAoSLoDcX.y7pivnwtLvE2BsVi', 'alice'

), (

**NULL** , '$2a$12$cDSmnx8nONg3xnpctJpJrOtpWHueejtN.D/uMMx58anZeeT/INMGi', 'ben'

);

**select \* from Users\_Roles**



**INSERT** **INTO** `execustomerdb`.`users\_roles` (

`userid` ,  
`roleid`

)  
**VALUES** (

'1', '1'

), (

'2', '2'

);

step 6: DAO Authentication (i.e the user name and password will not be hardcode rather it will be fetched from the database , from the users table )make a new package by the name "security" and create a user defined class by the name WebSecurityConfig extends WebSecurityConfigurerAdapter

@Configuration

public class GLWebSecurityConfig extends WebSecurityConfigurerAdapter{

@Bean

public UserDetailsService userDetailsService() {

return new UserDetailsServiceImpl();

}

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public DaoAuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

authProvider.setUserDetailsService(userDetailsService());

authProvider.setPasswordEncoder(passwordEncoder());

return authProvider;

}

//Authentication

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.authenticationProvider(authenticationProvider());

}

//Authorization

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests()

.antMatchers("/","/books/save","/books/showFormForAdd","/books/403").hasAnyAuthority("USER","ADMIN")

.antMatchers("/books/showFormForUpdate","/books/delete").hasAuthority("ADMIN")

.anyRequest().authenticated()

.and()

.formLogin().loginProcessingUrl("/login").successForwardUrl("/books/list").permitAll()

.and()

.logout().logoutSuccessUrl("/login").permitAll()

.and()

.exceptionHandling().accessDeniedPage("/books/403")

.and()

.cors().and().csrf().disable();

}

}

step 7 : in the service package make a class UserDetailsServiceImpl implements UserDetailsService

@Service

**public** **class** UserDetailsServiceImpl **implements** UserDetailsService {

@Autowired

**private** UserRepository userRepository;

@Override

**public** UserDetails loadUserByUsername(String username)

**throws** UsernameNotFoundException {

User user = userRepository.getUserByUsername(username);

**if** (user == **null**) {

**throw** **new** UsernameNotFoundException("Could not find user");

}

**return** **new** MyUserDetails(user);

}

}

step 8 : in the repository package make a interface by User UserRepository

**public** **interface** UserRepository **extends** JpaRepository<User,Long> {

@Query("SELECT u FROM User u WHERE u.username = ?1")

**public** User getUserByUsername(String username);

}

step 9 : in the security package make a class by MyUserDetails implements UserDetails

**public** **class** MyUserDetails **implements** org.springframework.security.core.userdetails.UserDetails {

**private** User user;

**public** MyUserDetails(User user) {

**this**.user = user;

}

@Override

**public** Collection<? **extends** GrantedAuthority> getAuthorities() {

List<Role> roles = user.getRoles();

List<SimpleGrantedAuthority> authorities = **new** ArrayList<>();

**for** (Role role : roles) {

authorities.add(**new** SimpleGrantedAuthority(role.getName()));

}

**return** authorities;

}

@Override

**public** String getPassword() {

**return** user.getPassword();

}

@Override

**public** String getUsername() {

**return** user.getUsername();

}

@Override

**public** **boolean** isAccountNonExpired() {

**return** **true**;

}

@Override

**public** **boolean** isAccountNonLocked() {

**return** **true**;

}

@Override

**public** **boolean** isCredentialsNonExpired() {

**return** **true**;

}

@Override

**public** **boolean** isEnabled() {

**return** **true**;

}

}

step 10 : create a sample endpoint and try to access it

@Controller

@RequestMapping("/books")

**public** **class** RbaController {

@GetMapping("/welcome")

**public** String greet()

{

**return** "Welcome";

}

}

linking this RBA to our project

Step 11 : create Book class in model package

@Entity

@Getter

@Setter

**public** **class** Book {

// define fields

@Id

@GeneratedValue(strategy=GenerationType.***IDENTITY***)

**private** **int** id;

**private** String name;

**private** String category;

**private** String author;

**public** Book()

{

}

**public** Book(String name, String category, String author) {

**super**();

**this**.name = name;

**this**.category = category;

**this**.author = author;

}

}

step 11 : add the remaining controller endpoints of Library project

@Controller

@RequestMapping("/books")

**public** **class** RbaController {

@Autowired

LibraryService bookService;

@GetMapping("/welcome")

**public** String greet()

{

**return** "Welcome";

}

@RequestMapping("/list")

**public** String listBooks(Model theModel) {

// get Books from db

List<Book> theBooks = bookService.findAll();

// add to the spring model

theModel.addAttribute("Books", theBooks);

**return** "list-Books";

}

@RequestMapping("/showFormForAdd")

**public** String showFormForAdd(Model theModel) {

// create model attribute to bind form data

Book theBook = **new** Book();

theModel.addAttribute("Book", theBook);

**return** "Book-form";

}

@RequestMapping("/showFormForUpdate")

**public** String showFormForUpdate(@RequestParam("bookId") **int** theId,

Model theModel) {

// get the Book from the service

Book theBook = bookService.findById(theId);

// set Book as a model attribute to pre-populate the form

theModel.addAttribute("Book", theBook);

// send over to our form

**return** "Book-form";

}

@PostMapping("/save")

**public** String saveBook(@RequestParam("id") **int** id,

@RequestParam("name") String name,@RequestParam("category") String category,@RequestParam("author") String author) {

System.***out***.println(id);

Book theBook;

**if**(id!=0)

{

theBook=bookService.findById(id);

theBook.setName(name);

theBook.setCategory(category);

theBook.setAuthor(author);

}

**else**

theBook=**new** Book(name, category, author);

// save the Book

bookService.save(theBook);

// use a redirect to prevent duplicate submissions

**return** "redirect:/books/list";

}

@RequestMapping("/delete")

**public** String delete(@RequestParam("bookId") **int** theId) {

// delete the Book

bookService.deleteById(theId);

// redirect to /Books/list

**return** "redirect:/books/list";

}

@RequestMapping("/search")

**public** String search(@RequestParam("name") String name,

@RequestParam("author") String author,

Model theModel) {

// check names, if both are empty then just give list of all Books

**if** (name.trim().isEmpty() && author.trim().isEmpty()) {

**return** "redirect:/books/list";

}

**else** {

// else, search by first name and last name

List<Book> theBooks =

bookService.searchBy(name, author);

// add to the spring model

theModel.addAttribute("Books", theBooks);

// send to list-Books

**return** "list-Books";

}

}

@RequestMapping(value = "/403")

**public** ModelAndView accesssDenied(Principal user) {

ModelAndView model = **new** ModelAndView();

**if** (user != **null**) {

model.addObject("msg", "Hi " + user.getName()

+ ", you do not have permission to access this page!");

} **else** {

model.addObject("msg",

"you do not have permission to access this page!");

}

model.setViewName("403");

**return** model;

}

step 12: add the LibraryService

**public** **interface** LibraryService {

**public** List<Book> findAll();

**public** Book findById(**int** theId);

**public** **void** save(Book theBook);

**public** **void** deleteById(**int** theId);

**public** List<Book> searchBy(String name, String author);

}

step 13 : add the LibraryServiceImpl

@Service

**public** **class** LibraryServiceImpl **implements** LibraryService {

@Autowired

BookRepository bookRepository;

@Override

**public** List<Book> findAll() {

// **TODO** Auto-generated method stub

List<Book> books=bookRepository.findAll();

**return** books;

}

@Override

**public** Book findById(**int** theId) {

// **TODO** Auto-generated method stub

**return** bookRepository.findById(theId).get();

}

@Override

**public** **void** save(Book theBook) {

// **TODO** Auto-generated method stub

bookRepository.save(theBook);

}

@Override

**public** **void** deleteById(**int** theId) {

// **TODO** Auto-generated method stub

bookRepository.deleteById(theId);

}

@Override

**public** List<Book> searchBy(String name, String author) {

// **TODO** Auto-generated method stub

List<Book> books=bookRepository.findByNameContainsAndAuthorContainsAllIgnoreCase(name, author);

**return** books;

}

}

step 14: add the BookRepository

@Repository

**public** **interface** BookRepository **extends** JpaRepository<Book, Integer> {

List<Book> findByNameContainsAndAuthorContainsAllIgnoreCase(String name,String author);

}

Step 15 : Override the configure(HttpSecurity http) method of the WebSecurityConfig

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.antMatchers("/","/books/save","/books/showFormForAdd","/books/403").hasAnyAuthority("USER","ADMIN")

.antMatchers("/books/showFormForUpdate","/books/delete").hasAuthority("ADMIN")

.anyRequest().authenticated()

.and()

.formLogin().loginProcessingUrl("/login").successForwardUrl("/books/list").permitAll()

.and()

.logout().logoutSuccessUrl("/login").permitAll()

.and()

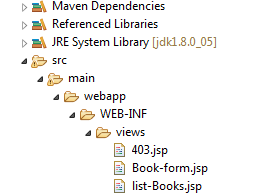
.exceptionHandling().accessDeniedPage("/books/403")

.and()

.cors().and().csrf().disable();

}

Step 16 : add the front end files in the following hierarchy



403.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html>

<body>

<h1>HTTP Status 403 - Access is denied</h1>

<h2>${msg}</h2>

</body>

</html>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Book-form.jsp

<%@ page contentType=*"text/html;charset=UTF-8"* language=*"java"* %>

<%@taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"*%>

<!DOCTYPE html>

<html>

<head>

<!-- Required meta tags -->

<meta charset=*"utf-8"*>

<meta name=*"viewport"*

content=*"width=device-width, initial-scale=1, shrink-to-fit=no"*>

<!-- Bootstrap CSS -->

<link rel=*"stylesheet"*

href=*"https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css"*

integrity=*"sha384-GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMp0DgiMDm4iYMj70gZWKYbI706tWS"*

crossorigin=*"anonymous"*>

<title>Save Book</title>

</head>

<body>

<div class=*"container"*>

<h3>Book Directory</h3>

<hr>

<p class=*"h4 mb-4"*>Enter Book</p>

<form action=*"/LibraryManagement/books/save"* method=*"POST"*>

<!-- Add hidden form field to handle update -->

<input type=*"hidden"* name=*"id"* value=*"*${Book.id}*"* />

<div class=*"form-inline"*>

<input type=*"text"* name=*"name"* value=*"*${Book.name}*"*

class=*"form-control mb-4 col-4"* placeholder=*"Name"*>

</div>

<div class=*"form-inline"*>

<input type=*"text"* name=*"category"* value=*"*${Book.category}*"*

class=*"form-control mb-4 col-4"* placeholder=*"Category"*>

</div>

<div class=*"form-inline"*>

<input type=*"text"* name=*"author"* value=*"*${Book.author}*"*

class=*"form-control mb-4 col-4"* placeholder=*"Author"*>

</div>

<button type=*"submit"* class=*"btn btn-info col-2"*>Save</button>

</form>

<hr>

<a href=*"/LibraryManagement/books/list"*>Back to Books List</a>

</div>

</body>

</html>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

list-Books.jsp

<%@ page contentType=*"text/html;charset=UTF-8"* language=*"java"*%>

<%@taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"*%>

<%@ taglib prefix=*"sec"* uri=*"http://www.springframework.org/security/tags"* %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<!-- Required meta tags -->

<meta charset=*"utf-8"*>

<meta name=*"viewport"*

content=*"width=device-width, initial-scale=1, shrink-to-fit=no"*>

<!-- Bootstrap CSS -->

<link rel=*"stylesheet"*

href=*"https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css"*

integrity=*"sha384-GJzZqFGwb1QTTN6wy59ffF1BuGJpLSa9DkKMp0DgiMDm4iYMj70gZWKYbI706tWS"*

crossorigin=*"anonymous"*>

<title>Books Directory</title>

</head>

<body>

<div class=*"container"*>

<h3>Books Directory</h3>

<hr>

<!-- Add a search form -->

<form action=*"/LibraryManagement/books/search"* class=*"form-inline"*>

<!-- Add a button -->

<a href=*"/LibraryManagement/books/showFormForAdd"*

class=*"btn btn-primary btn-sm mb-3"*> Add Book </a> <input

type=*"search"* name=*"name"* placeholder=*"Name"*

class=*"form-control-sm ml-5 mr-2 mb-3"* /> <input type=*"search"*

name=*"author"* placeholder=*"Author"*

class=*"form-control-sm mr-2 mb-3"* />

<button type=*"submit"* class=*"btn btn-success btn-sm mb-3"*>Search</button>

<a href=*"/LibraryManagement/logout"*

class=*"btn btn-primary btn-sm mb-3 mx-auto"*> Logout </a>

</form>

<table class=*"table table-bordered table-striped"*>

<thead class=*"thead-dark"*>

<tr>

<th>Name</th>

<th>Category</th>

<th>Author</th>

<th>Action</th>

</tr>

</thead>

<tbody>

<c:forEach items=*"*${Books}*"* var=*"tempBook"*>

<tr>

<td><c:out value=*"*${tempBook.name}*"* /></td>

<td><c:out value=*"*${tempBook.category}*"* /></td>

<td><c:out value=*"*${tempBook.author}*"* /></td>

<td>

<!-- Add "update" button/link --> <a

href=*"/LibraryManagement/books/showFormForUpdate?bookId=*${tempBook.id}*"*

class=*"btn btn-info btn-sm"*> Update </a> <!-- Add "delete" button/link -->

<a href=*"/LibraryManagement/books/delete?bookId=*${tempBook.id}*"*

class=*"btn btn-danger btn-sm"*

onclick="if (!(confirm('Are you sure you want to delete this employee?'))) return false">

Delete </a>

</td>

</tr>

</c:forEach>

</tbody>

</table>

</div>

</body>

</html>

for Spring version 3.0 and above

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration {

@Bean

**public** UserDetailsService userDetailsService() {

**return** **new** UserDetailsServiceImpl();

}

@Bean

**public** BCryptPasswordEncoder passwordEncoder() {

**return** **new** BCryptPasswordEncoder();

}

@Bean

**public** DaoAuthenticationProvider authenticationProvider() {

DaoAuthenticationProvider authProvider = **new** DaoAuthenticationProvider();

authProvider.setUserDetailsService(userDetailsService());

authProvider.setPasswordEncoder(passwordEncoder());

**return** authProvider;

}

@Bean

**public** SecurityFilterChain filterChain(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.antMatchers("/","/books/save","/books/showFormForAdd","/books/403").hasAnyAuthority("USER","ADMIN")

.antMatchers("/books/showFormForUpdate","/books/delete").hasAuthority("ADMIN")

.anyRequest().authenticated()

.and()

.formLogin().loginProcessingUrl("/login").successForwardUrl("/books/list").permitAll()

.and()

.logout().logoutSuccessUrl("/login").permitAll()

.and()

.exceptionHandling().accessDeniedPage("/books/403")

.and()

.cors().and().csrf().disable();

http.authenticationProvider(authenticationProvider());

**return** http.build();

}

}

---

package com.auth.MyAuthorisation.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;

import com.auth.MyAuthorisation.service.UserDetailsServiceImpl;

@Configuration

@EnableWebSecurity

public class WebSecurityConfig

{

// @Override

// protected void configure(AuthenticationManagerBuilder auth) throws Exception {

// auth.authenticationProvider(myAuth());

// }

@Bean

public DaoAuthenticationProvider myAuth() {

DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

authProvider.setUserDetailsService(myUser());

authProvider.setPasswordEncoder(myPas());

return authProvider;

}

@Bean

public BCryptPasswordEncoder myPas() {

return new BCryptPasswordEncoder();

}

@Bean

public UserDetailsService myUser() {

return new UserDetailsServiceImpl();

}

// @Override

// protected void configure(HttpSecurity http) throws Exception {

// http.authorizeRequests()

// .antMatchers("/","/books/save","/books/showFormForAdd","/books/403").hasAnyAuthority("USER","ADMIN")

// .antMatchers("/books/showFormForUpdate","/books/delete").hasAuthority("ADMIN")

// .anyRequest().authenticated()

// .and()

// .formLogin().loginProcessingUrl("/login").successForwardUrl("/books/list").permitAll()

// .and()

// .logout().logoutSuccessUrl("/login").permitAll()

// .and()

// .exceptionHandling().accessDeniedPage("/books/403")

// .and()

// .cors().and().csrf().disable();

//}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.authorizeRequests()

.antMatchers("/","/books/save","/books/showFormForAdd","/books/403").hasAnyAuthority("USER","ADMIN")

.antMatchers("/books/showFormForUpdate","/books/delete").hasAuthority("ADMIN")

.anyRequest().authenticated()

.and()

.formLogin().loginProcessingUrl("/login").successForwardUrl("/books/list").permitAll()

.and()

.logout().logoutSuccessUrl("/login").permitAll()

.and()

.exceptionHandling().accessDeniedPage("/books/403")

.and()

.cors().and().csrf().disable();

http.authenticationProvider(myAuth());

return http.build();

}

}

step 1:in pom.xml

sb version 3.1.11

java 17

step 2: in application.properties

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

step 3: in entity classes

replace javax by jakarta

step 4: in WebConfig class

replace antMatchers by requestMatchers

.antMatchers("/","/employees","/employees/new","/403").hasAnyAuthority("USER","ADMIN")

replaced by

.requestMatchers("/","/employees","/employees/new","/403").hasAnyAuthority("USER","ADMIN")

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception

{

return http.build();

}