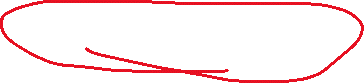
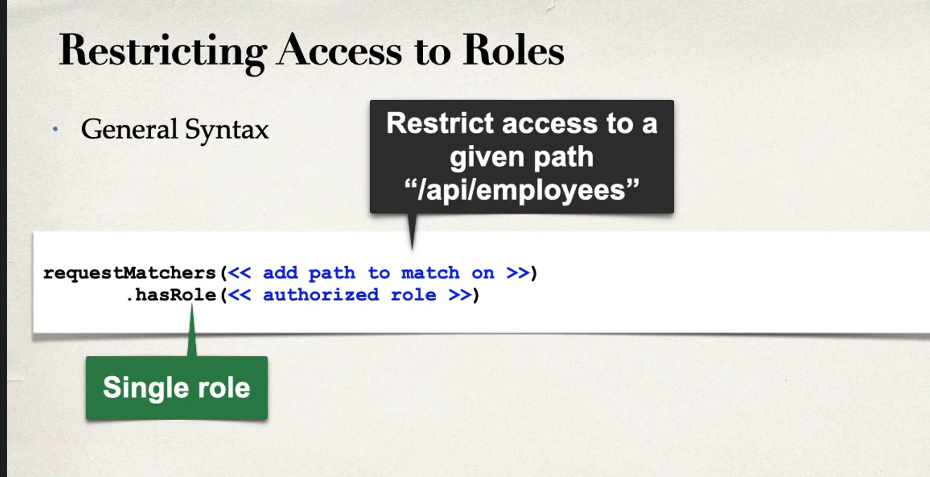
In Spring Security 5.7.0, folosirea lui WebSecurityConfigurerAdapter este depreciata.**In Spring Security 6 in genere a fost stearsa.**

@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {



* Acum nu mai trebuie sa extindem clasa data, insa sa cream mai bine un bean pentru orice facem
* De ex, nu va trebui sa mai suprascriem metoda configure(HttpSecurity http)
* Acum, vom crea deja beanuri

**Roles, Acces**

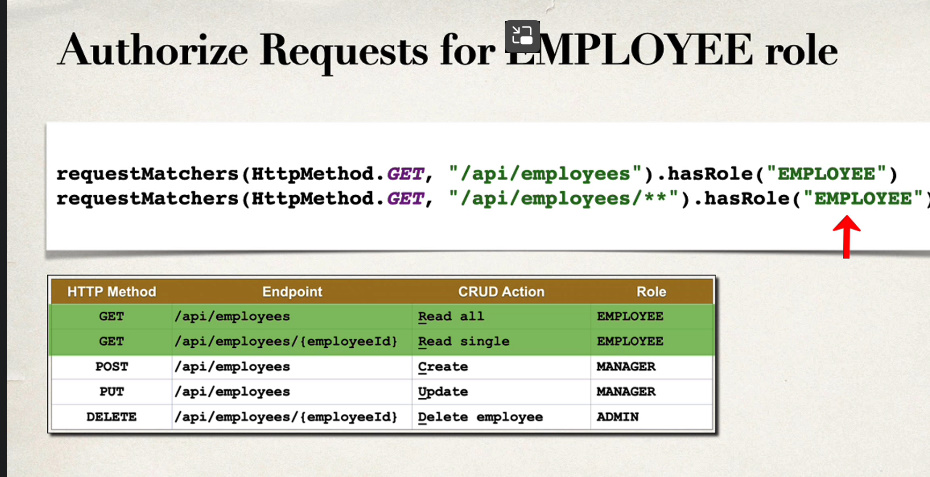


**- Atentie! antMatchers() a fost depreciata si *stearsa* in Spring Security 6!!! In locul ei folosim requestMatchers()**

**- authorizeRequests tot e depreciata, si mai bine de folosi authorizeHttpRequests()**

@Bean  
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http.authorizeHttpRequests(  
 req -> req.requestMatchers(HttpMethod.*GET*,"/teachers").hasAnyRole("Member","Manager","Admin")  
 .requestMatchers(HttpMethod.*GET*,"/teachers/\*\*").hasAnyRole("Member","Manager","Admin")  
 .requestMatchers(HttpMethod.*POST*,"/teachers").hasAnyRole("Manager","Admin")  
 .requestMatchers(HttpMethod.*DELETE*,"/teachers/\*\*").hasRole("Admin")  
 );  
 **http.httpBasic(Customizer.*withDefaults*());**  
 http.csrf(csrf->csrf.disable());  
  
 return http.build();  
}

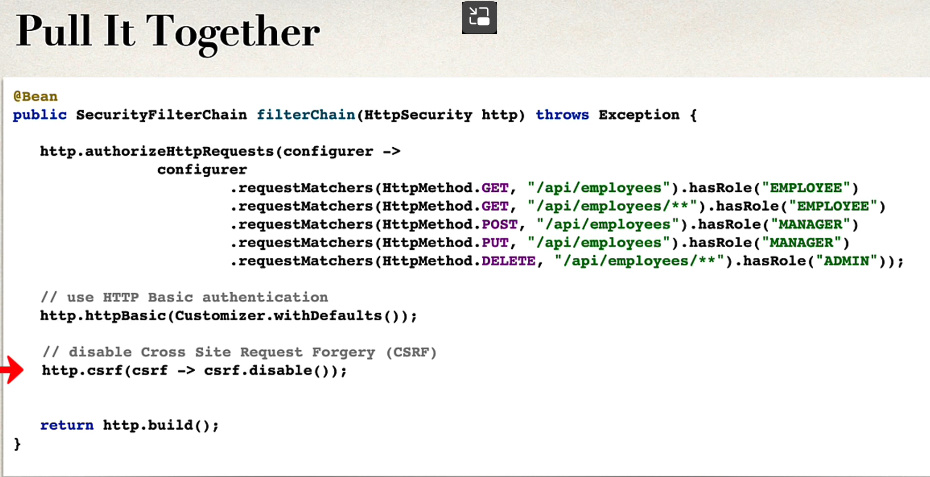
* hasRole() poate lua un singur rol ca argument
* hasAnyRole(rol1,rol2 etc.) – accepta mai multe roluri
* Metoda requestMatchers mai are un prototip in care nu ia pur si simplu requestul, ci si metoda, gen post, get etc.



* Daca vrem sa oferim acces la orice link ce incepe cu /api/employees/, punem \*\*, asa /api/employees/\*\* si asta e pentru cand extragem un employee, gen api/employees/5

**http.httpBasic(Customizer.withDefaults())**

* metoda data indica ca folosim Http Basic Authentification, adica authentificare simpla cu username si parola si este obligatorie



**disable CSRF protection**

* http.csrf(csrf->csrf.disable()) – dezactiveaza CSRF protection
* E mai bine sa o dezactivam cand rest controllerul face si operatii de put,post,delete.

**In memory users**

|  |  |  |
| --- | --- | --- |
|  |  | **Sau ca sa nu punem mereu la password {noop}**  @Bean public InMemoryUserDetailsManager userDetailsManager(){  User.UserBuilder userBuilder = User.*withDefaultPasswordEncoder*();  UserDetails user1 = userBuilder.username("kerbecs").password("test123").roles("ADMIN","EMPLOYEE").build();  UserDetails user2 = userBuilder.username("test").password("test123").roles("EMPLOYEE").build();   return new InMemoryUserDetailsManager(user1,user2); } |
|  |  | * UserBuilder – ne ajuta sa cream usor useri * Deocamdata cream noi direct niste useri. Can vom lucra cu DB, i vom lua de acolo. * User.withDefaultPasswordEncoder() – parola fara criptare, simplu text. Anume aici si specificam metoda de salvare a parolei |

**Pentru a lua useri din baza de date**

|  |
| --- |
|  |
| @Configuration |
|  |
|  | public class DemoSecurityConfig { |
|  |  |
|  | // add a reference to our security data source |
|  |  |
|  | private DataSource securityDataSource; |
|  |  |
|  | @Autowired |
|  | public DemoSecurityConfig(DataSource theSecurityDataSource) { |
|  | securityDataSource = theSecurityDataSource; |
|  | } |
|  |  |
|  | @Bean |
|  | public UserDetailsManager userDetailsManager() { |
|  | return new JdbcUserDetailsManager(securityDataSource); |
|  | } |
|  |  |

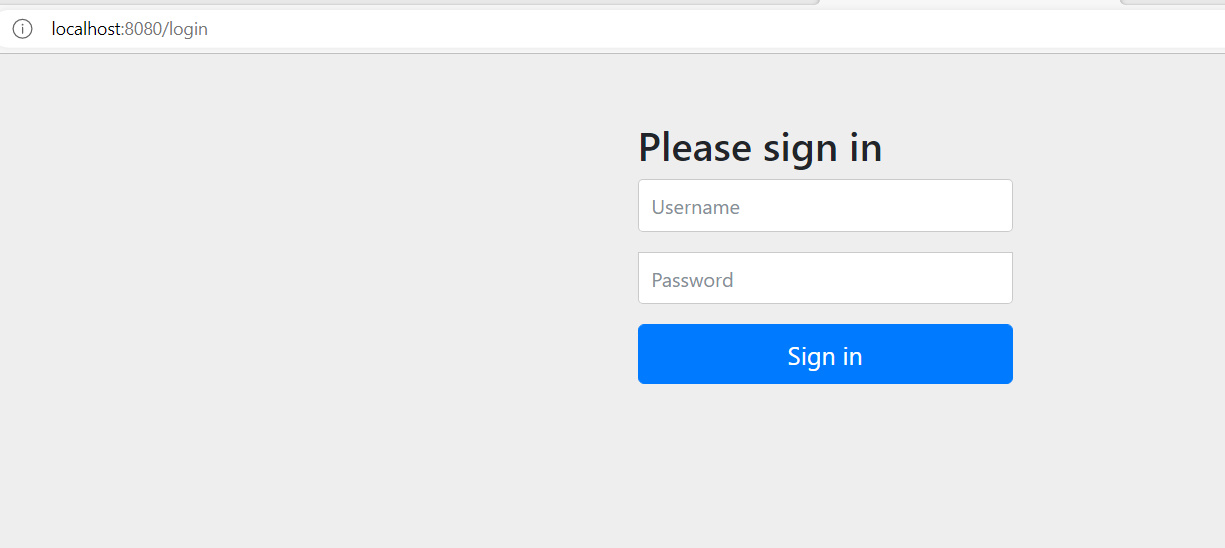
[Spring & Hibernate for Beginners (includes Spring Boot) | Udemy](https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/9500224#overview)

!!!!!!!!!!!!!!!!

Acum, cand vom da start la aplicatie, nu se va mai deschide automat

localhost:8080

Ci localhost:8080/login, care ofera o forma de log default:



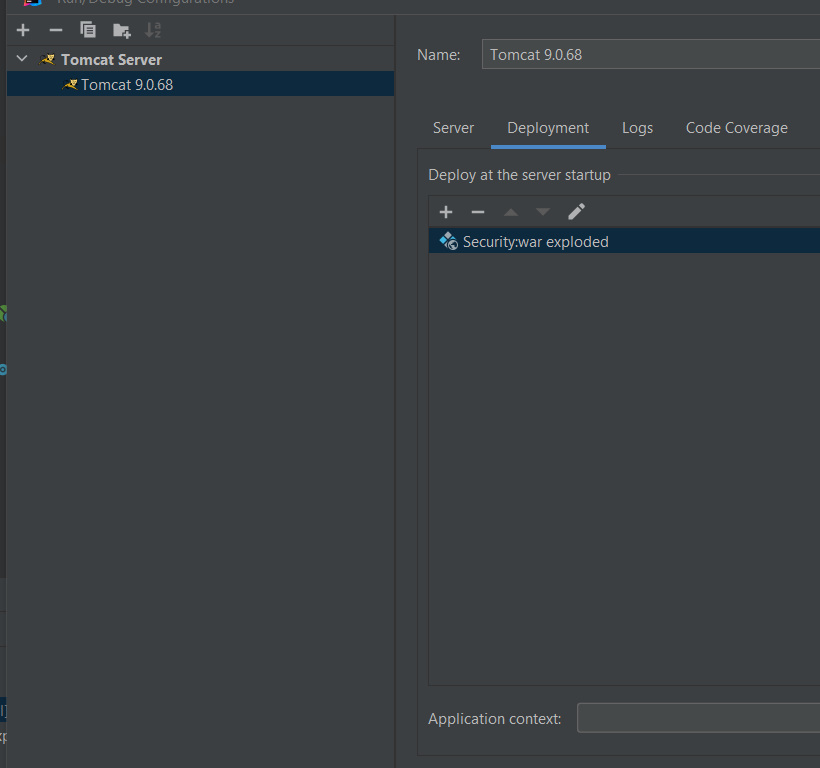
**Atentie**

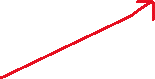
* Metoda **authorizeHttpRequests**() fara parametru va fi stearsa si e depreciata!Cea cu parametru lambda e cea recomandata
* La fel si metoda **and**()!!! Deci, folosim de acum doar lambda cu **authorizeHttpRequests()**

Acum, login si parola introduse de noi vor fi cautate in datele pentru useri introduse de noi, si daca nu le gaseste, va da o eroare de autentificare.

Daca le gaseste, ne duce la pagina localhost:8080/

**Context Root**





context root

Este root path pentru webapp noastra, sau un nume unic per project

Daca de ex punem TestApp, o vom putea accesa la:

localhost:8080/TestApp

Este exact ca un nume pentru project, doar ca e numit context root, ca sa putem rula mai mult projecte pe acelasi tomct server

Deci, daca vrem sa rulam mai multe app pe TomCat, e suficient sa le punem nume diferite la context root si gata.