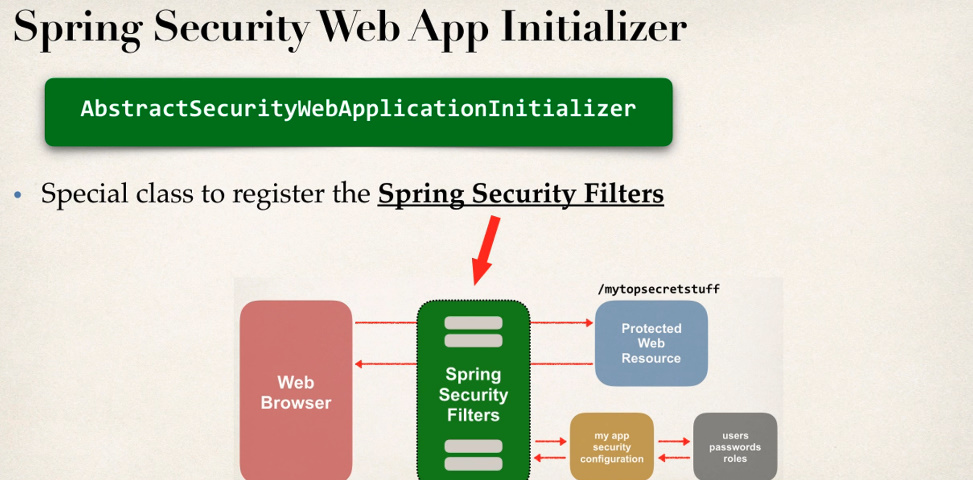
**OLD VERSION**



* Spring ofera un **webapp security initializer** pentru a initializa security framework
* Deci security cod al nostru va folosi acest initializer pentru a initializa servlet container pentru Spring Security
* Totodata, cel mai important lucru pe care acest initialzier il face este de a inregistra Spring Security Filters si le va face active
* E foarte important ca sa avem initializata clasa de sus, asa cum ea inregistreaza Spring Security Filters
* Deci, anume ea e cea care va inregistra servlet container pentru Security,va activa filtrele de Securitate si va folosi @Configuration pentru security, exact asa cum face si servlet container creat prin extinderea la

AbstractAnnotationConfigDispatcherServletInitializer

* **Servlet container** - is the part of web server which can be run in a separate process



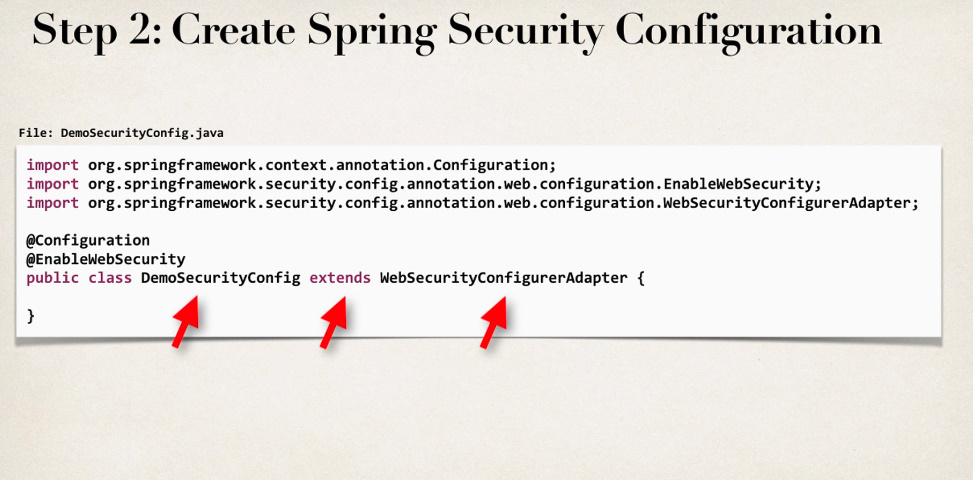
E foarte important ca sa avem acest fisier, de altfel Spring Security Filters nu vor fi registrate si nici activate

**Atentie! WebSecurityConfigurerAdapter a fost sters din Spring Boot 3**

1. Trebuie sa cream o clasa care pur si simplu extinde clasa lunga de sus, si atat, fara override la veo metoda. Anume extinderea ei va fi inregistrata de Spring, care va intelege ca trebuie sa intitalizeze Servlet Container,sa activeze filtrele de security si Controllerul



1. Spring Security tot are nevoie de o configuratie, asa ca Spring cum are nevoie de applicationContext



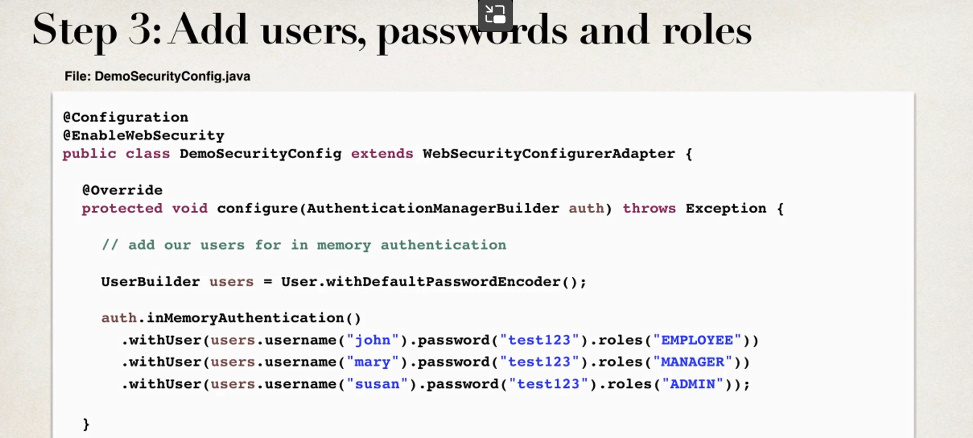
deci folosim anotatiile

@Configuration

@EnableWebSecurity

si extindem clasa WebSecurityConfigureAdapter

1. Anume in acest config vom adauga userii, parolele, rolurile etc. Deocamdata o facem fara Baza de date, ca sa fie mai simplu



* Deci,pentru a adauga userii, trebuie sa suprascriem metoda configure()
* Ea are un parametru AuthenticationManagerBuilder care e initializat de Spring Framework, si il folosim pentru a adauga tipul de autentificare pe care avem de gand sa il folosim si datele despre useri.
* UserBuilder – ne ajuta sa cream usor useri
* Deocamdata cream noi direct niste useri. Can vom lucra cu DB, i vom lua de acolo. Asta si spune auth.inMemoryAuthentication(), ca luam datele din memoria la aplicatie.
* User.withDefaultPasswordEncoder() – parola fara criptare, simplu text. Anume aici si specificam metoda de salvare a parolei

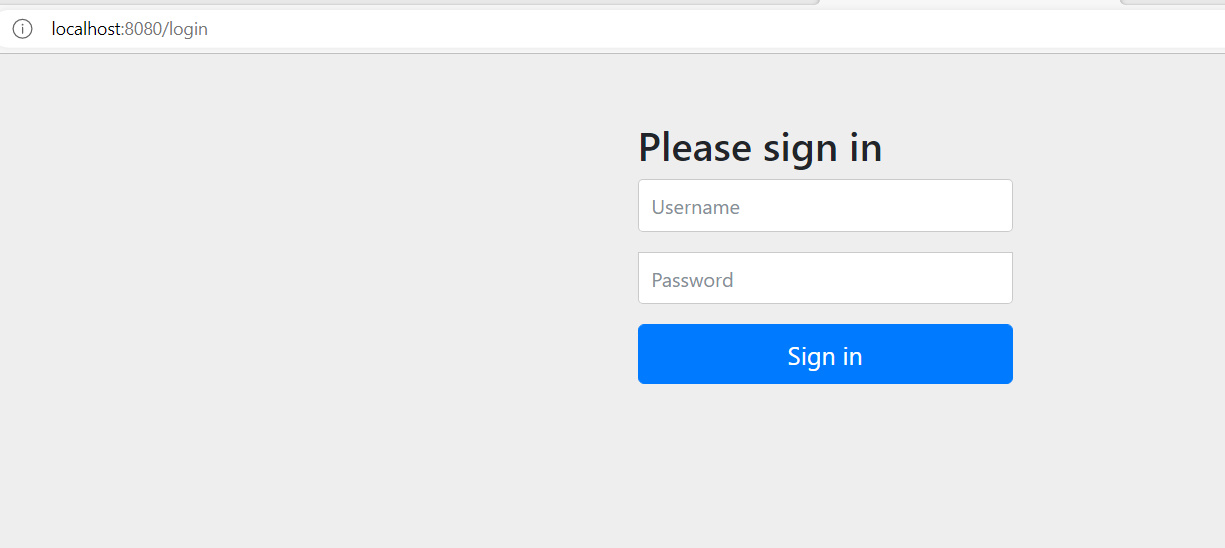
[Spring & Hibernate for Beginners (includes Spring Boot) | Udemy](https://www.udemy.com/course/spring-hibernate-tutorial/learn/lecture/9500224#overview)

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

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

localhost:8080

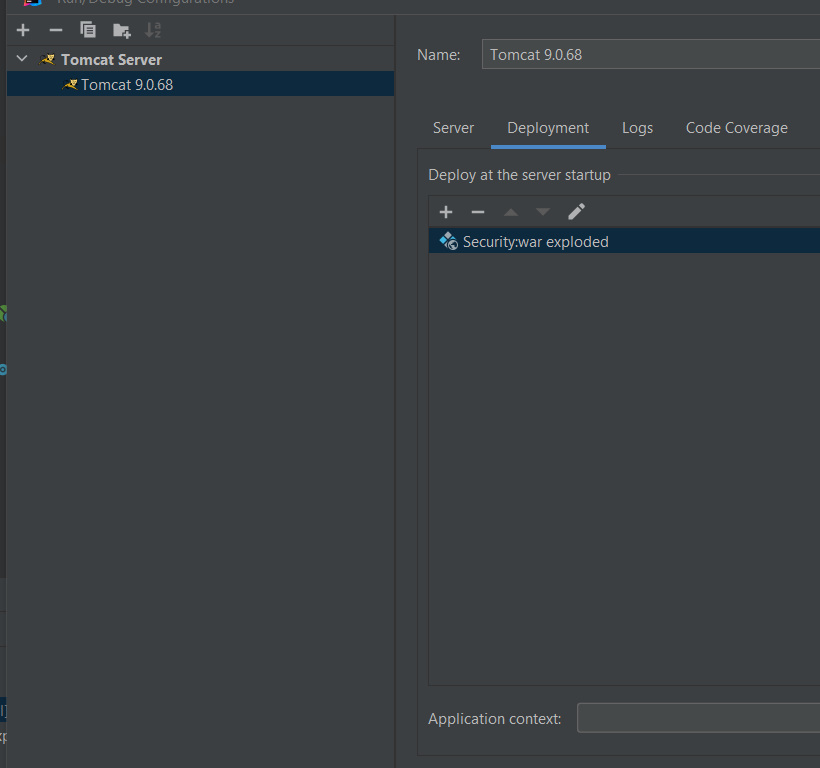
Ci localhost:8080/login, care ofera o forma de log default:

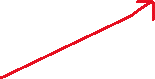


Acum, login si parola introduse de noi vor fi cautate in datele pentru useri introduse de noi, si daca nu le gaseste, va da o eroare de autentificare.

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.