**Problema cu logarea**

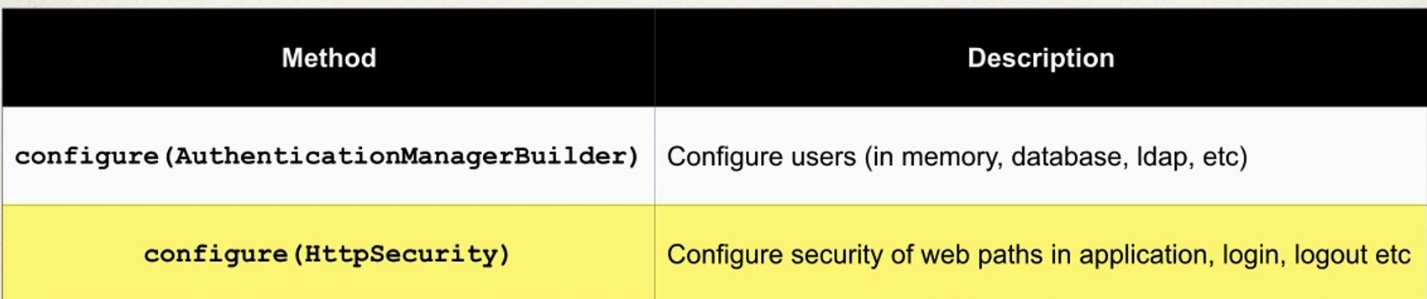
* Daca de ex userul se logheaza, apoi noi facem ceva vreo modificare in cod, cum ar fi in vreo pagina JSP, si dam update la resources, problema e ca userul da refresh la pagina si vede deodata toate schimbarile, fara sa trebuiasca sa se relogheze
* EVIDENT, daca dam restart la TOMCat, ii va cere sa se relogheze, dar nu si daca dam update la resurse!

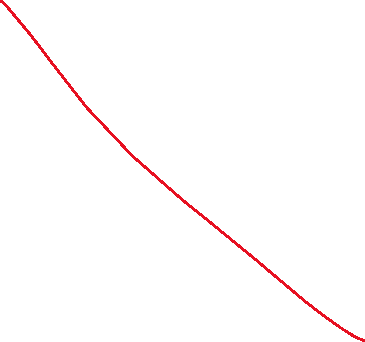
**Session**

* Userul continua sa ramana logat din cauza ca logarea sa e bazata pe **web browser session**.
* Session face ca atat timp cat browserul e deschis, sau procesul de rulare a aplicatiei continua, userul sa fie logat.
* **Aceasta nu e o problema in proiectele reale!**
* Doar in cazuri absolut necesare,in proiecte foarte securizate, se poate cere relogarea la update la resurse
* Sesiunea ramane in intregul browser, deci chiar de deschidem o noua fereastra,in acelasi browser, tot ramanem logati inca.
* Daca deschidem un nou browser, se va crea o noua sesiune si va trebui logare.
* O fila incognito creaza mereu o noua sesiune

**Custom Login Form**

1. Trebuie modificat Spring Security Configuration pentru a adauga custom login form**:(OLD)**





Deci, Spring ofera 2 metode configure():

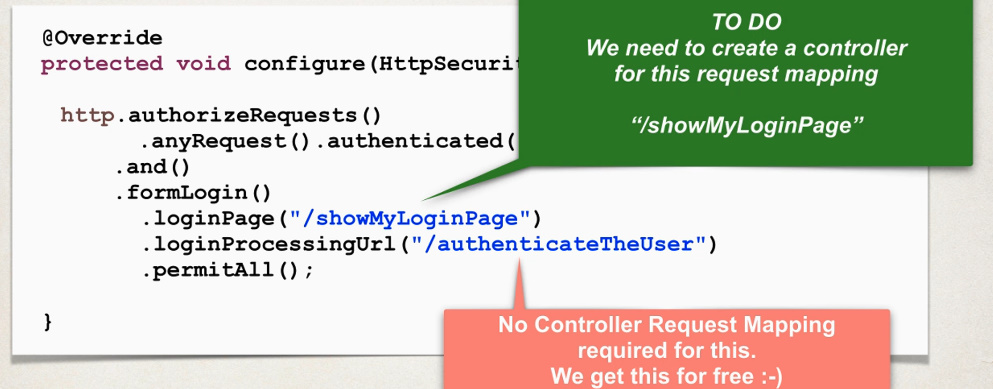
* una cu un parametru de tip AuthenticationManagerBuilder, pentru a crea userii,
* alta cu un parametru de tip HttpSecurity, care ne permite sa modificam login page, login form, si in general procesul de logare.
* O folosim si pe a 2!



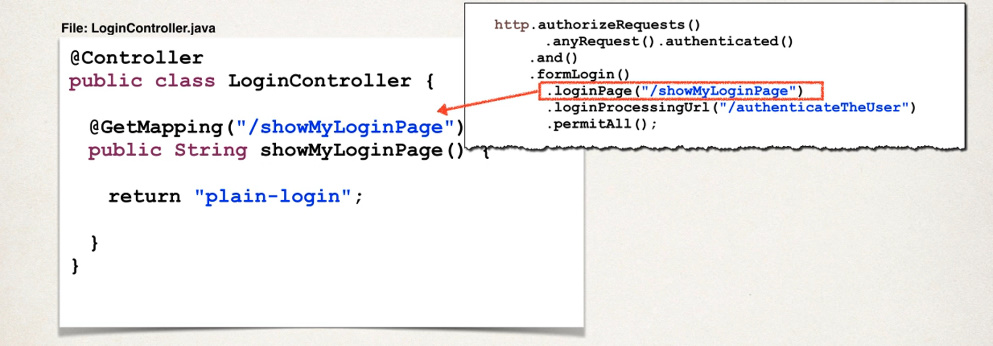


**ATENTIE!!! Rolurile sunt obligatoriu de setat!**

* **~~.authorizeRequests()~~** **authorizeHttpRequests()** – arata ca aratam cum autorizam accesul la HttpRequest, si mai apoi aratam cum anume
* **.anyRequest()** – aratam ca ne referim la orice request facut de user, si apoi vom arata ce facem cu fiecare request
* **authenticated()** – niciun request nu va fi posibil fara logare!Iata de ce, mereu userulului i se cere sa se logheze, chiar cand siteul abea se deschide
* **~~and()~~** ~~– doar returneaza obiectul, ca sa mai setam ceva in el~~
* **.formLogin()** – customizam loggin process
* **.loginPage(“/showMyLoginPage”)** - anume acest request “/showMyLoginPage” e request din Controller care va returna pagina de logare
* **loginProcessingUrl(“/authenticateTheUser”)** – anume la acest request forma va trimite datele pentru procesare(check user, password ...). @Post logic!!!
* **permitAll()** – oricine poate vedea aceasta pagina, fara a fi logat,logic

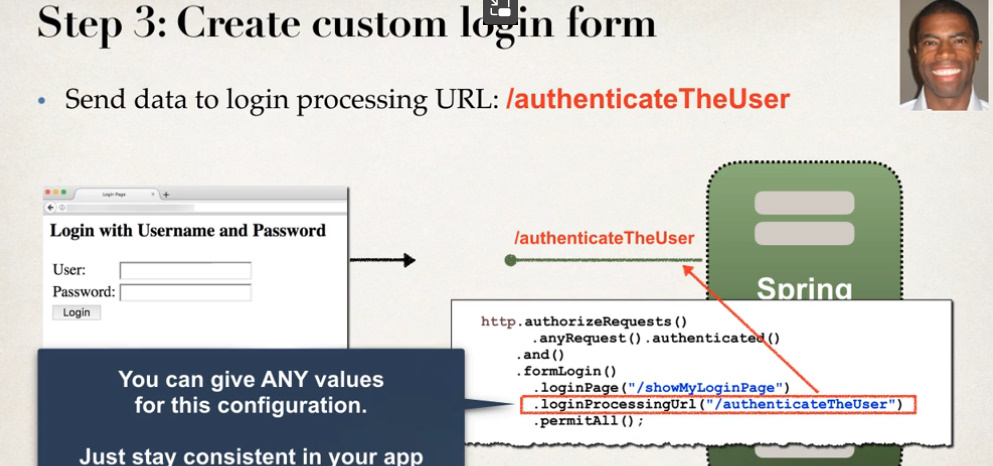
2.

* Deci, e logic ca avem nevoie de a crea un request mapping in Controller pentru a arata pagina de logare
* Insa, nu e nevoie de a crea un request mapping in Controller si pentru a verifica daca userul exista. Spring va face asta. El pur si simplu va face ca orice request “/authenticateTheUser” sa ajunga la metoda configure(AuthentificationManagerBuilder), unde sunt userii nostri, si sa verifice daca datele introduse pentru user exista. Deci, avem nevoie de ambele metode!!!



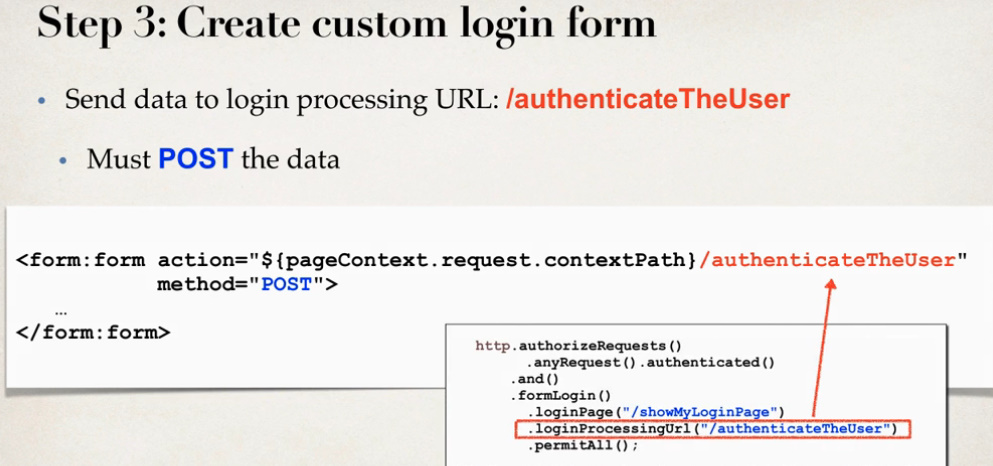
1. Cream pagina si forma

Forma va trimite datele la /authenticateTheUser, asa cum am setat

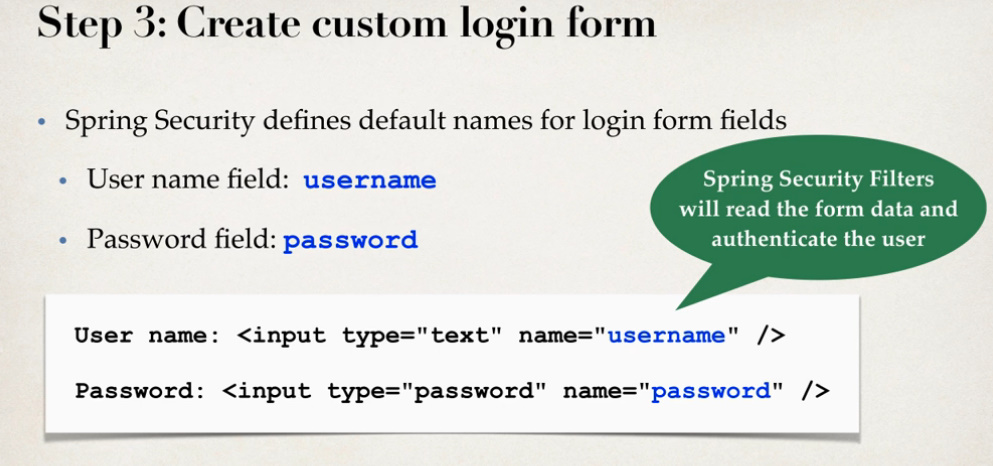


Totusi, nu noi vom crea codul pentru a ne ocupa de acest request, adica nu noi vom verifica loginul si parola, ci Spring Security cu filtrele sale o va face, dar pe baza la metoda configure() unde adaugam userii, evident, doar ca automat va trece la ea.



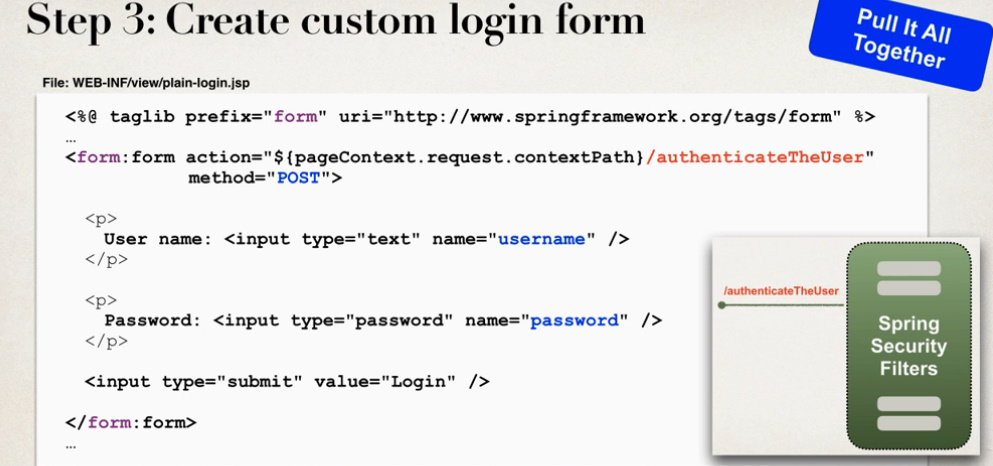






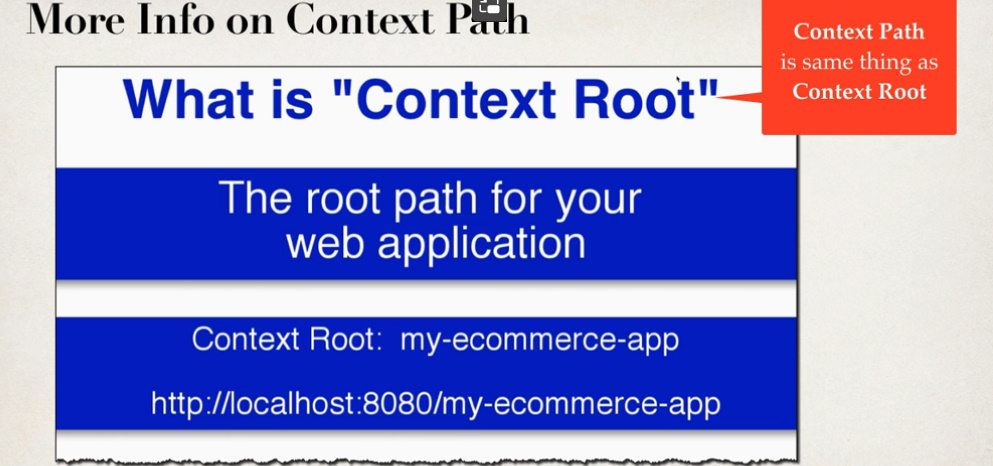
Atribut nu avem, deci ne folosim de param. Spring Filters sunt facute ca sa preia automat datele de la obiectele din param de tip String cu numele “username” si “password”

Chiar de nu trimite atribut, oricum folosim Post, caci este mult mai securizat.



**De ce folosim ${pageContext.request.contextPath}?**

* Context root si Context Path sunt cam aceleasi lucruri.

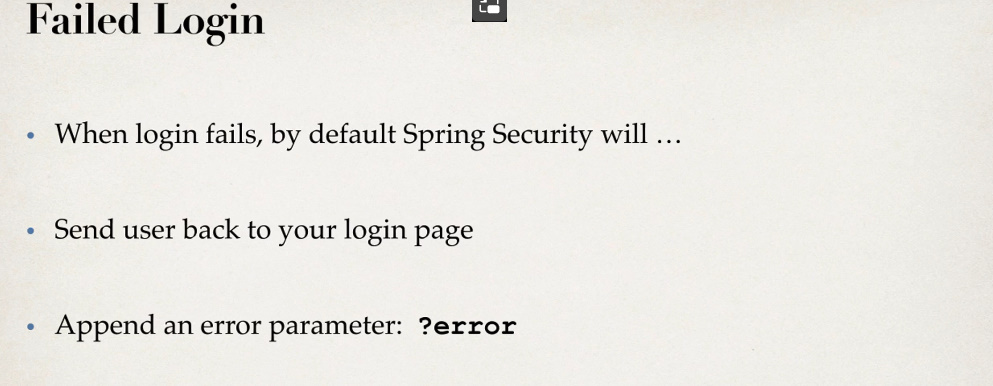


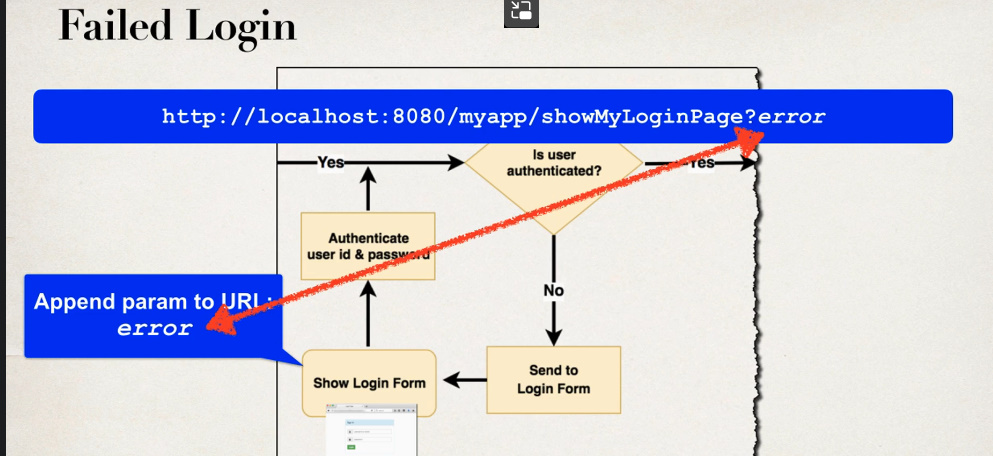
* Acum, daca setam context root, de ex cu MyApp, pai orice adresa in app a noastra va incepe cu MyApp:
* localhost:8080/**MyApp**/
* Si Spring Security trebuie sa primeasca path cu tot cu context root

Anume acest ${pageContext.request.contextPath} pune context root setat de noi

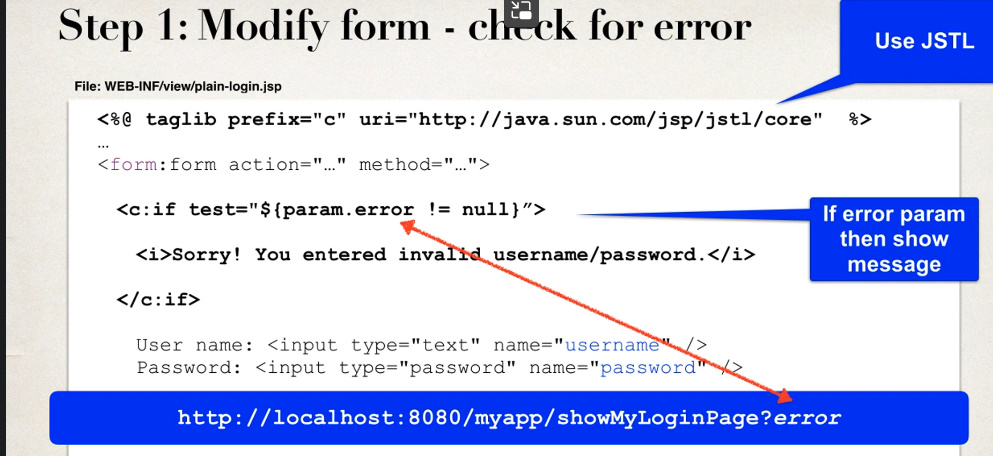
* Am putea sa il scriem de mana, dar daca modificam context root, va trebui sa il modificam peste tot.

**ShowLoginError**

* Totusi, chiar de introducem date gresite, nu apare vreun mesaj de eroare
* 



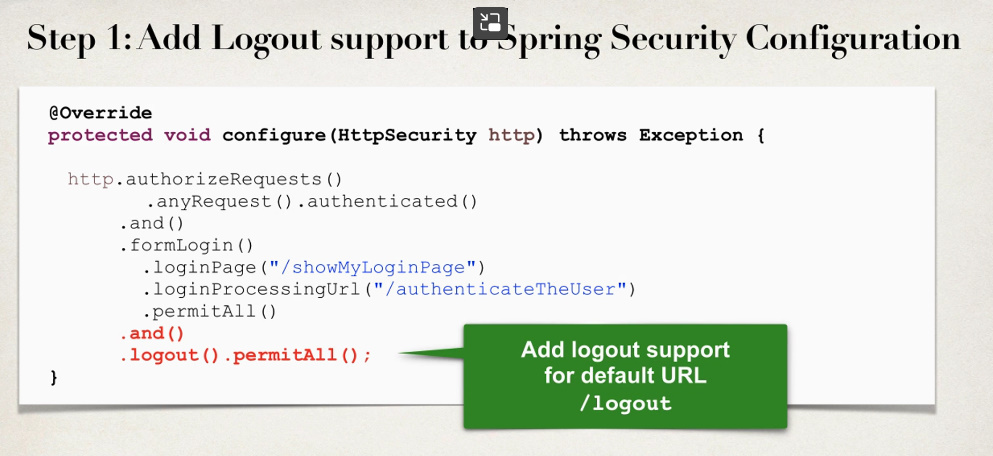
Deci, cand Spring gaseste vreo eroare la logare, el trimite inapoi pagina de logare, dar adauga in param un String cu numele “error” ce contine mesajul erorii.

* 

Deci, vom folosi tagul c: din jstl, pentru a verifica daca param.error exista, adica daca a fost trimis un asa obiect in param de catre Spring

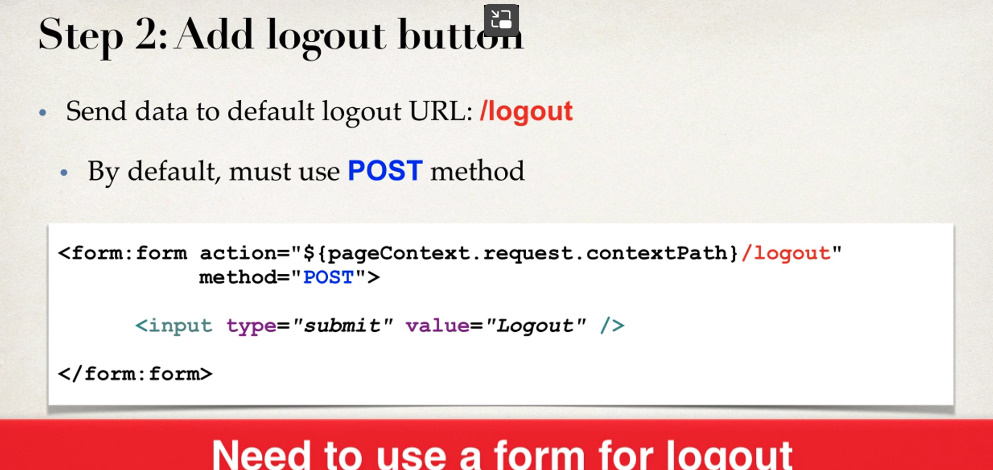
test=”conditie de evaluat”

**LogOut**

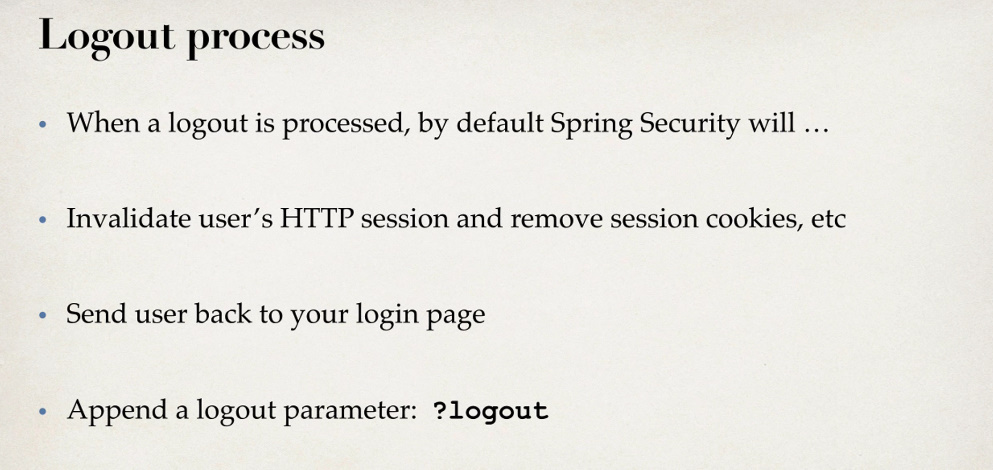


* deci, la folosirea requestului /logout, Spring va face logout automat.

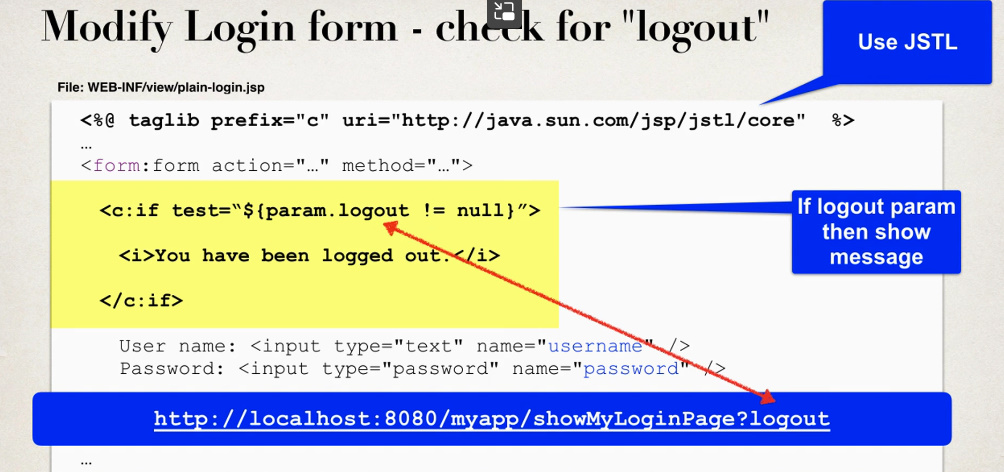


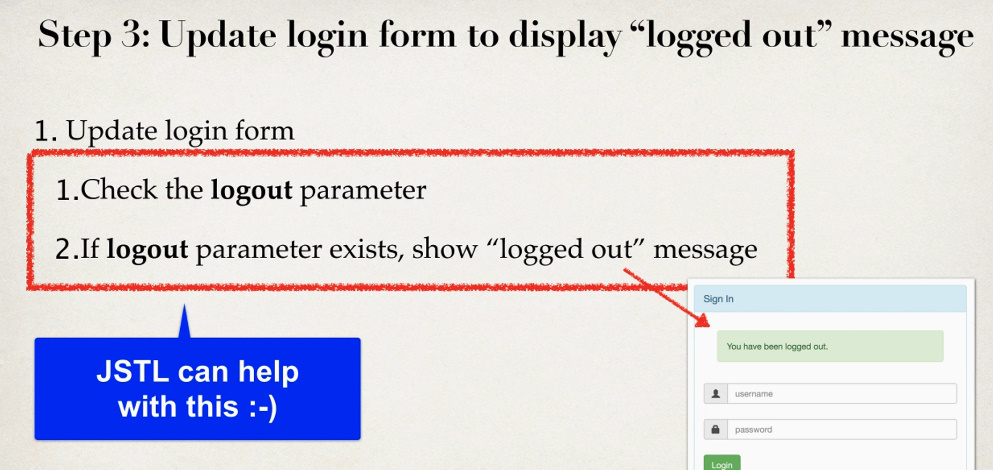


Trebuie neaparat o forma pentru logout! Butoanele nu vor merge, caci ele mereu trimit GET, dar pentru Spring Security, trebuie POST



Deci, spring va returna cu pagina setata pentru logout(index deobicei) si un parametru logout cu un mesaj de delogare.





Si la pagina jsp facem la fel ca si pentru error:

<**c:if** test="${param.logout != null}">  
 <i>Successfully logout</i>  
 <br>  
</**c:if**>