1. User type in <http://localhost:8080/> at URL of browser. The request URI “/” will be caught up by Spring Security(SS).
2. SS firstly load all URL and ROLE mapping list in its memory (loadResourcesDefine() method) when user type the above URL in its browser and submit the request.
3. Since SS has all the URI and ROLE mapping list in its memory already, so it will get request URI “/”, and check whether the “/” is in the mapping list (getAttributes()). If not, return “/login?error” page to browser which is pre-defined in the “WebSecurityConfig.config()” method. If yes, then get ROLE of “/” from its mapping list.
4. SS then go to check the ROLE in user request (decide() method before successful login, the user’s ROLE is ROLE\_ANONMOUS). If the user ROLE in the ROLE list that SS get from its mapping list, then pass the verification and allow the URI “/” go to find the matched controller method. If not, then use the pre-defined login URI “/login” to simulate the user request and submit, in the case, SS will allow the request to find its matched controller method “@RequestMapping(“/login”), so that the method will return “login.html” page to browser.
5. User input username and password in the “login” or “login?error” page and submit the request. At this moment, the value of request URI is coming from html page “action” item. In the case, the value at “action” item is “/login”.
6. Again, the process starts form a), SS retrieve all the URI and ROLE mapping list first. Then get the request URI “/login” and check whether the URI in the mapping list. Since the “/login” URI is not in the mapping list, suppose SS will give back “/login?error” page to user, however, SS check the WebSecurityConfig.config() method and know the URI “/login” is SIGNIN PAGE. So SS will call loadUserByUsername() to find the user details by user input name and password.
7. If the above loadUserByUsername() is not pass, then return pre-defined page “/login?error” to browser. Is pass, the SS will get ALL allowed URI and ROLE list for the user, and create a new object USER with username, password and ROLE info. The USER is hold by SS and available until user logout.
8. SS continue to call “decide()” method to verify the user’s ROLE which comes from above USER object with the ROLE list that SS get from its URI and ROLE mapping list. If the user’s ROLE in the list, then allow the request URI find its matched controller method. If now, return browser “login?error” page.
9. If user successfully login, so it will get “home.html” page. In the case, if user click the “logout”, then another action “/logout” is submitted to SS, since the current URI and ROLE mapping list doesn’t have the URI, so SS will give back “login” page to browser, the page has themyleaf “${param.logout}” tag, so it will get string “logout” as pathVariable from the request URI “/logout” and understand it is logout operation, so will based on the condition show some information in the “login” page.
10. User type in “<http://localhost:8080/>login” at URL of browser. The request URI “/login” will be caught up by Spring Security(SS).
11. SS firstly load all URL and ROLE mapping list in its memory (loadResourcesDefine() method) when user type the above URL in its browser and submit the request.
12. Since SS has all the URI and ROLE mapping list in its memory already, so it will get request URI “/”, and check whether the “/login” is in the mapping list (getAttributes()). If not, suppose SS will return “/login?error” page to browser which is pre-defined in the “WebSecurityConfig.config()” method. However, the “WebSecurityConfig.config()” method also defined the “/login” URI is login form, so SS won’t return “/login?error” page, it will call loadUserByUsername() to check the username and password (loadUserByUsername() method will be call to get user details).
13. SS then call getAttributes() to check whether the request URI in the URI list of not. If yes, then call decide() method to see whether the user’s ROLES in the SS URI and ROLE mapping list, if yes, then pass the verification and redirect to the pre-defined success URI “/” to find the matched controller method. If not, then return “login?error” page to browser.

Attention: don’t add login URI “/login” to URI resources list, because the URI will make a dead loop in SS. SS call getAttributes() method to check whether user’s URI in mapping list, so if it find the login URI in its mapping list, then it will transfer user request to Controller method @RequestMapping(“/login”), the controller method will return “login” page to browser, so it makes dead loop.