Spring -security:

Authentication : identification of user

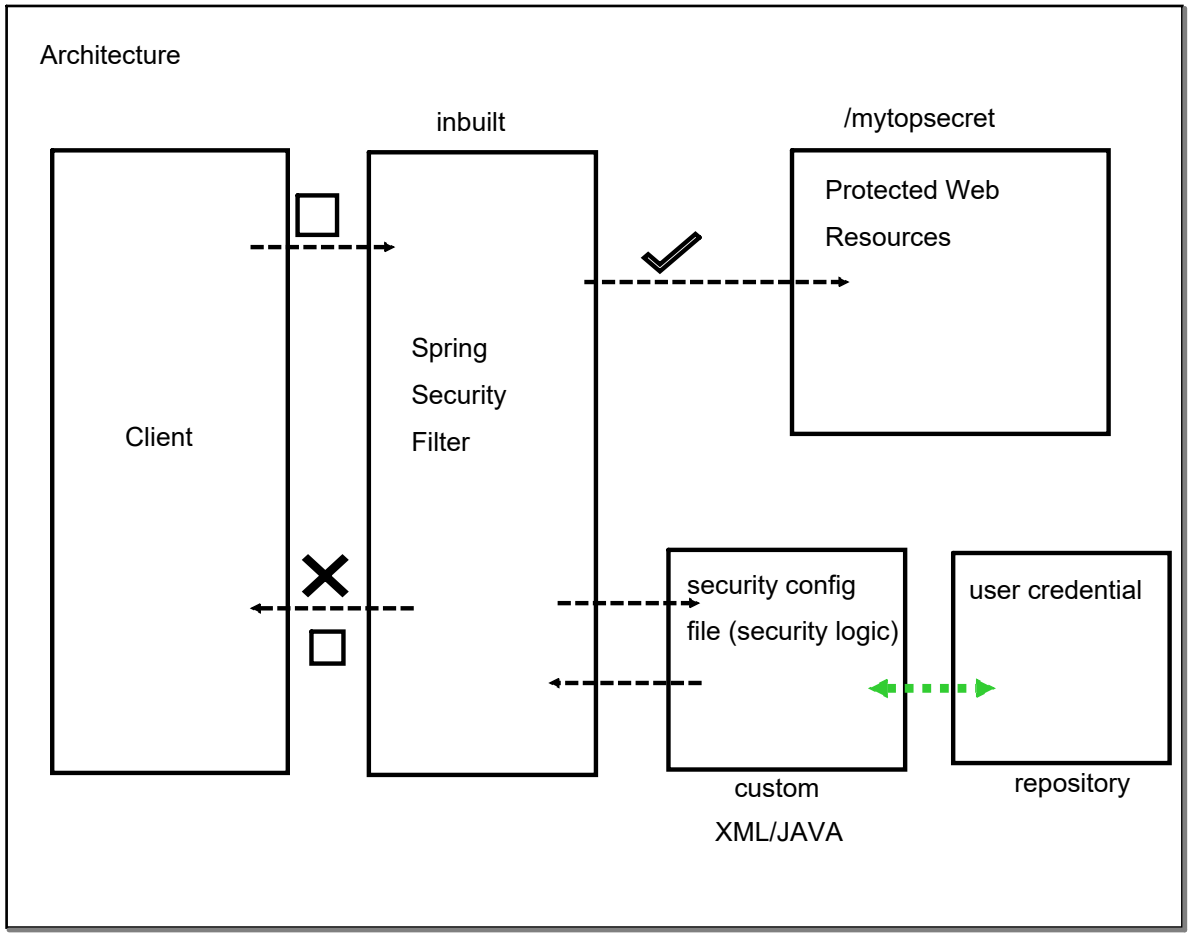
Authorization : access of a particular (role of the user)

#Spring-Security : framework for security

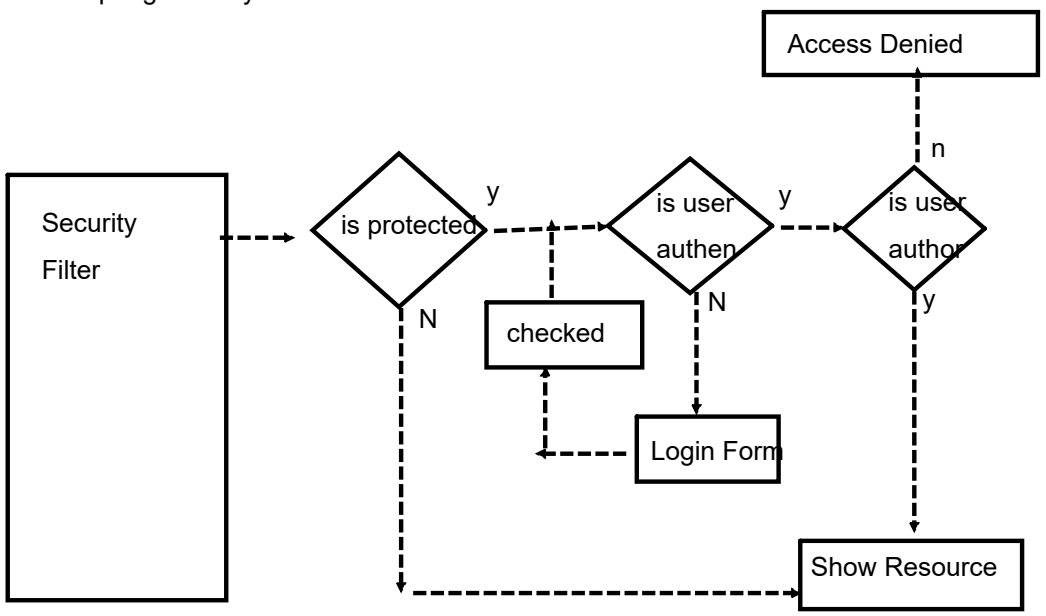
#Implementation : Servlet filters (background): proxy

==>pre-process/post-process the web request

==>Filter can route web requests based on security logic


Architecture
inbuilt
/mytopsecret
Protected Web
Resources
Spring
Security
Filter
Client
security config
file (security logic)
user credential
custom
XML/JAVA
repository

Action of spring security



Security Config: XML (legacy)

Pure Java (modern)

Two implementation

1. Declarative Security : Define security constraints in config file(separation of concerns)

2. Programmatic Security : additional api, greater customized, security concern often integrates with business logic

Different Login method

1. HTTP basic authentication : security framework will instruct browser to reflect a built in login dialog

2. Default login form : security framework has its login form

3. Custom login form :

User Credential Repo:

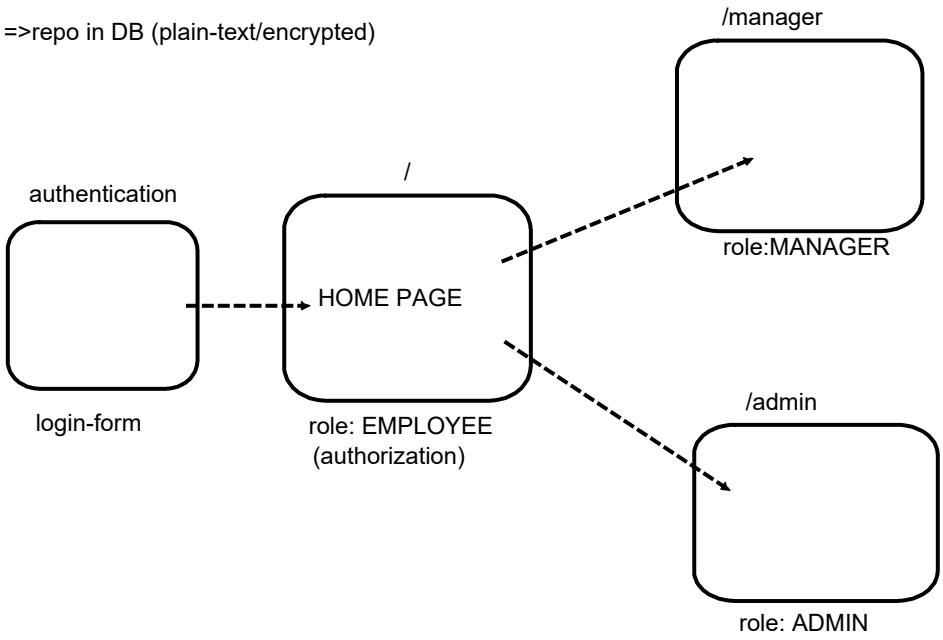
1. In-memory : hard-coded credentials maintained in code

=>JDBC

=>LDAP

=>Custom/Pluggable...

=>how to secure spring mvc web apps
=>work around login (default/custom)
=>define user and roles (simple authentication)
=>Protect the contents and the URL based on roles
=>show/hide UI based on role
=>repo in DB (plain-text/encrypted)



Maven dependencies:

#spring-security-web

#Spring-security-config

Spring Framework

Spring Security

These are two different project...

#different release cycle

Might be incompatibility issues

Research : to find compatible version

Development Process...

1. Create a Spring security initializer (initialize the security filter)
2. Create Spring Security config file (pure JAVA)
3. Repo (in-memory: ...)

#inbuilt class provided to register Filter

=>create a class and extends the inbuilt class (that's it)

##inherit an inbuilt web security configurer

#security filter by default intercept all request and protects all resource such that acces over them has be authenticated

Custom Login Form:

1. Modify the security config to reference the custom login form
2. Develop a controller to show the custom login form
3. Create custom login form...

HTML forms are not compatible with spring security

#need to use spring forms

Spring maps a relative path (/mylogin relative to current/requested path)

<context path>/mylogin

Additionally helpful to access CSS/JS/IMAGES

Create an appropriate logout mechanism:

1. Add logout support in security config file...
2. Add logout button in JSP
3. update the login form to show the logout message

Logout processing:

#Invalidate user's session and remove session cookies

#send user back to login page

#Append a logout parameter : ?logout (to login page)

Using Spring MVC Form instead of plain HTML form:

#automatic support for security defenses (protection against CSRF)

Cross-Site Request Forgery

A security attack where an evil website tricks you into executing an action on web application that you are logged in...

1. Embed an addn. authentication token into all HTML forms
2. On subsequent submission : web app will verify token before processing

#using Spring MVC form make sure that authentication is associated with HTML form:

#submission of form shall be using POST method

Manually attach CSRF token support in plain HTML form

#add hidden field

Authorization (Roles and rules associated with them)

Spring security provides JSP custom tags for accessing logged in user details

=>get support of custom JSP Tag Library

#All details of logged in user is available in an inbuilt object : principal

Development Process:

#Create supporting controller and view pages (manager & admin)

#Update user roles (repo)

#Restrict the access based on roles

Access Restriction:

antMatchers(<path to restrict>).hasRole(<authorized role>)

antMatchers(<path to restrict>).hasAnyRole(<list of authorized roles>)

#Need to have custom Access denied page...

#Create Controller code and view page

#configure the spring security config file

#control the UI based on Roles

#Security JSP tags to check for the roles of user

Database Access for user credentials

#Spring Security can read user credentials from DB

#By default: predefined table schemas

#will create appropriate JDBC code in background

Default implementation

#need to have appropriate dB

#add support for DB in POM file (mysql-connector) (DB Connection POOL : c3p0)

#configure DataSource

#configure spring config file to use jdbc authentication

Default table schema

1. users

username varchar (PK)

password varchar

enabled TINYINT(1)

2. authorities

(FK)username varchar | unique key

authority varchar | unique key

Spring security 5 , password are stored using a specific format

Passwords can be stored in different encoded forms

syntax : store any password

{id}encodedPassword

id : type of encryption(encryption id)

eg:

plain text password : noop ({noop}abc)

BCrypt password hashing : bcrypt (recommendation)

One way encrypted hashing

Change the password col size : min 68:

{bcrypt} : 8 char

encoded password : 68 char

various bcrypt hash :

Spring Security supports : $2a

Customization:

Table schemas

eg:

user credentials : members

user_id

pw

active

Roles : roles

user_id

role

#Need to tell Spring how to query from custom tables

=>Provide query to find user by username

=>Provide query to find roles by username

Updating Spring Security Config