* Spring Security defines a framework for spring security
* It is implemented using Servlet filters in the background
* Two methods of securing a web app : Declarative & programmatic

Servlet Filters

* Servlet Filters are used to pre-process/post-process web request
* Servlet filters can route web request depending on web logic

Spring security filters

(Intercepts)

User role & password

Protected data

Web browser

Spring app configuration file

Authentication: check whether user is authenticatedor not ..Check the identity of user.

If it is not authenticated then sent it back to login page

Authorization: check whether user has permissions to access the page.else send access denied message.

Suppose user requested some data you will authenticate user. But for next request you should not authenticate user. You need session management for that. But http is stateless protocol.

No information is maintained from earlier request.

So two ways to do autherization:

1)Session tokens

When the client sends the request first time , server authenticates it . It maintains the session log and generates session ID . and with the response send this session ID to client.

.It saves session id in cookie.

But there is problem with this approach.

Server 1

Load balancer

request

Server 2

Server 3

to balance the load of server request will be sent to the server which has less load.

Suppose if you send request to server 1 then it will create session log but if you send next request and it will redirected to server2 then it wont have previous history.

Solution : load balancer have to maintain all information but it will be too much for load balancer so JWT is solution.

2)JWT JSON web tokens-javascript object notation <https://jwt.io/introduction/>

way for securely transmitting information between parties as a JSON object.

* Compact: Because of its relatively small size, a JWT can be sent through a URL, through a POST parameter, or inside an HTTP header, and it is transmitted quickly.
* Self-contained: A JWT contains all the required information about an entity to avoid querying a database more than once. The recipient of a JWT also does not need to call a server to validate the token.

The token is mainly composed of header, payload, signature. These three parts are separated by dots(.).

Different login forms

1)HTTP Basic authentication

2)spring default login form

3)Custom login page

Steps:

1)In maven –maven-war-plugin jar is used instead of web.xml

Add dependencies

* spring mvc
* servlet
* javax.servlet.jstl.api
* javax.servlet.api

2)Spring configuration

3)Dispatcher servlet: need to mention configuration classes.

SpringConfig.xml or AppInitializer class

4)View resolver: for JSP pages

Web.xml or VIewResolver

**Cross-Site Request Forgery (CSRF)** is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. **CSRF** attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

**Protecting Against CSRF Attacks**

The reason that a CSRF attack is possible is that the HTTP request from the victim’s website and the request from the attacker’s website are exactly the same. This means there is no way to reject requests coming from the evil website and allow requests coming from the bank’s website. To protect against CSRF attacks we need to ensure there is something in the request that the evil site is unable to provide so we can differentiate the two requests.

Spring provides two mechanisms to protect against CSRF attacks:

* The [the section called “Synchronizer Token Pattern”](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/" \l "csrf-protection-stp" \o "Synchronizer Token Pattern)
* Specifying the [the section called “SameSite Attribute”](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/" \l "csrf-protection-ssa" \o "SameSite Attribute) on your session cookie

##### Synchronizer Token Pattern

The predominant and most comprehensive way to protect against CSRF attacks is to use the [Synchronizer Token Pattern](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#General_Recommendation:_Synchronizer_Token_Pattern). This solution is to ensure that each HTTP request requires, in addition to our session cookie, a secure random generated value called a CSRF token must be present in the HTTP request.

When an HTTP request is submitted, the server must look up the expected CSRF token and compare it against the actual CSRF token in the HTTP request. If the values do not match, the HTTP request should be rejected.

##### SameSite Attribute

An emerging way to protect against [CSRF Attacks](https://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#csrf) is to specify the [SameSite Attribute](https://tools.ietf.org/html/draft-west-first-party-cookies" \t "_top) on cookies. A server can specify the SameSite attribute when setting a cookie to indicate that the cookie should not be sent when coming from external sites.

Password Encryption

* Spring security recommends using popular bcrypt algorithm.
* It performs hashing
* It performs one way encryption. It can not be decrypted. So for authentication’

It encrypts the password received from login form and check it with encrypted password stored in database.

* Add random salt(String) to password for protection.
* It includes support for defeat brute force attack.

A **Brute Force Attack** is the simplest method to gain access to a site or server (or anything that is password protected). It tries various combinations of usernames and passwords again and again until it gets in. This repetitive action is like an army attacking a fort.

To implement bcrpt you need to set character length of password=68 in database.

There are main 5 features that spring security provides.

1)Authentication:

Verify identity of user.

By checking username and password with the database.

Spring security provides default login and logout page .You just need to add springboot security parent dependency.

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfig **extends** WebSecurityConfigurerAdapter {

@Override

**public** **void** configure(WebSecurity web) **throws** Exception {

web.ignoring().antMatchers("/resources/\*\*");

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests().antMatchers("/").permitAll().antMatchers("/welcome").hasAnyRole("USER", "ADMIN")

.antMatchers("/getEmployees").hasAnyRole("USER", "ADMIN").antMatchers("/addNewEmployee")

.hasAnyRole("ADMIN").anyRequest().authenticated().and().formLogin().permitAll().and().logout()

.permitAll();

http.csrf().disable();

}

@Autowired

**public** **void** configureGlobal(AuthenticationManagerBuilder authenticationMgr) **throws** Exception {

authenticationMgr.inMemoryAuthentication().withUser("employee").password("{noop}employee")

.authorities("ROLE\_USER").and().withUser("javainuse").password("{noop}javainuse")

.authorities("ROLE\_USER", "ROLE\_ADMIN");

}

2)Autherization

3)Principal: currently logged in user

4)Role:

5)Autherize: