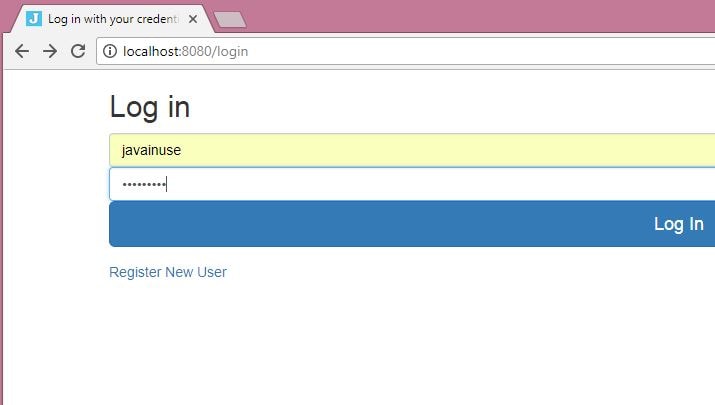
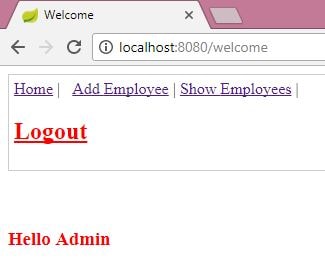
**Spring Boot Security - Enabling CSRF Protection**

In a previous post we had implemented [Spring Boot Security - Password Encoding Using Bcrypt](https://www.javainuse.com/spring/boot_security_jdbc_authentication_bcrypt).  
But till now in all our examples we had disabled CSRF. CSRF stands for Cross-Site Request Forgery. It is an attack that forces an end user to execute unwanted actions on a web application in which they are 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.

Understanding CSRF attack-

Previously we had [Spring Boot Security - Password Encoding Using Bcrypt](https://www.javainuse.com/spring/boot_security_jdbc_authentication_bcrypt). Start this application and login using a valid password.  
  
  
  
Do not close the above window. Now suppose you receive a mail with following content.

Hi JavaInUse

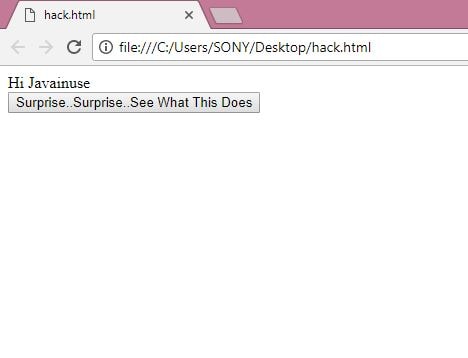
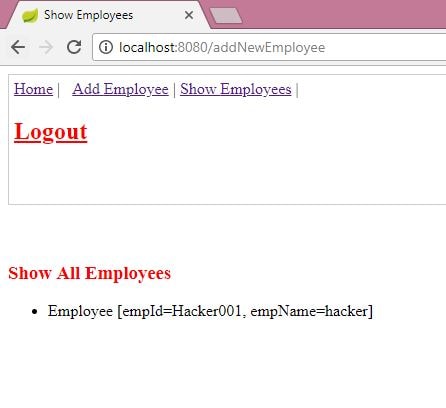
<form method = "post" action="http://localhost:8080/addNewEmployee">

<input id ="empId" type="hidden" name="empId" value="Hacker001"/>

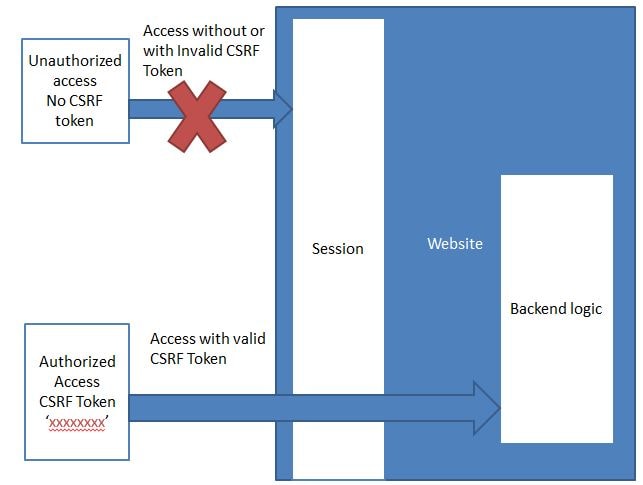
<input id ="empName" type="hidden" name="empName" value="hacker"/>

<input type="SUBMIT" value="Surprise..Surprise..See What This Does" />

</form>

You open this page and click on the surprise button-  
  
  
  
We see that it has added an Employee with name Hacker to our application. This is a CSRF attack. Next we see how to tackle this CSRF attack.

Lets Begin-

We will be using the CSRF security token to grant access only to authorized users.  
  
We will be modifying the code we developed in the previous [Spring Boot Security - Password Encoding Using Bcrypt](https://www.javainuse.com/spring/boot_security_jdbc_authentication_bcrypt)  
Maven Project will be as follows-

In the pom.xml add the spring-security-taglibs dependency.

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

**<dependency>**

**<groupId>org.springframework.security</groupId>**

**<artifactId>spring-security-taglibs</artifactId>**

**</dependency>**

</dependencies>

Next we modify the security configuration to enable CSRF by commenting the csrf disabled command .

package com.javainuse.config;

import javax.sql.DataSource;

@Configuration

@EnableWebSecurity

public class EmployeeSecurityConfiguration extends WebSecurityConfigurerAdapter {

@Autowired

DataSource dataSource;

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

// Enable jdbc authentication

@Autowired

public void configAuthentication(AuthenticationManagerBuilder auth) throws Exception {

auth.jdbcAuthentication().dataSource(dataSource).passwordEncoder(passwordEncoder());

}

@Bean

public JdbcUserDetailsManager jdbcUserDetailsManager() throws Exception {

JdbcUserDetailsManager jdbcUserDetailsManager = new JdbcUserDetailsManager();

jdbcUserDetailsManager.setDataSource(dataSource);

return jdbcUserDetailsManager;

}

@Override

public void configure(WebSecurity web) throws Exception {

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

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().antMatchers("/register").permitAll().antMatchers("/welcome")

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

.antMatchers("/addNewEmployee").hasAnyRole("ADMIN").anyRequest().authenticated().and().formLogin()

.loginPage("/login").permitAll().and().logout().permitAll();

**//http.csrf().disable();**

}

// @Autowired

// public void configureGlobal(AuthenticationManagerBuilder authenticationMgr)

// throws Exception {

// authenticationMgr.inMemoryAuthentication().withUser("admin").password("admin").authorities("ROLE\_USER").and()

// .withUser("javainuse").password("javainuse").authorities("ROLE\_USER",

// "ROLE\_ADMIN");

// }

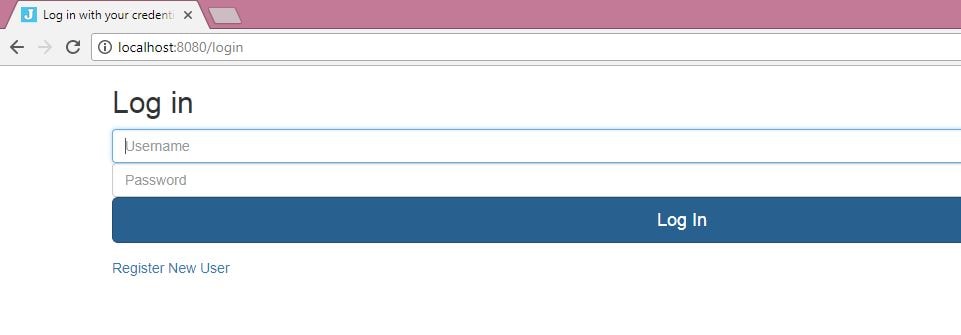
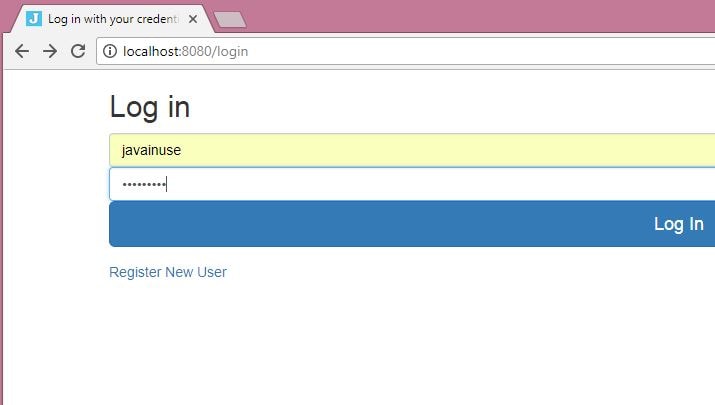
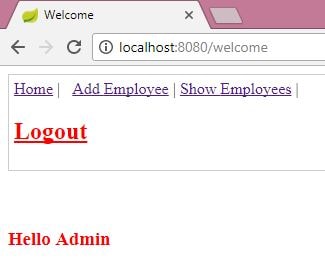
}

Next in all the jsp pages add the spring security taglib and the csrf token tag

<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

<sec:csrfInput />

Start the application -

* Go to localhost:8080/welcome, we will be redirected to the custom login page.  
  
* Login using the credentials  
    
    
  
* Again click on the surprise button of the CSRF attack page  
  