



[한국ICT인재개발원] 스프링 프레임워크

11. 스프링 시큐리티 기초

前) 광고데이터 분석 1년

前) IT강의 경력 2년 6개월

前) 머신러닝을 활용한 데이터 분석 프로젝트반 운영 1년

前) 리그오브 레전드 데이터 분석 등...

現) 국비반 강의 진행중

스프링 시큐리티는 특정 사이트로 들어오는 접근을 가로채서
로그인이 되어있는지 등을 검증한 다음 로직을 실행시켜줍니다.
원래 **JSP**에서 “필터”라는 기능을 이용해서 구현 가능했지만
스프링에서는 빈 컨테이너에서 관련 자원들을 같이 관리하기 때문에
이미 작성된 빈들을 연계한 여러가지 인증 방식을 구현하기 쉬워집니다.

```
security_prj [spring_workspace master]
├── src/main/java
├── src/main/resources
├── src/test/java
├── src/test/resources
├── JRE System Library [JavaSE-1.8]
├── Maven Dependencies
├── src
├── target
└── pom.xml
```



1. Spring Security Web

[org.springframework.security](#) » [spring-security-web](#)

Spring Security

Last Release on Aug 16, 2021

Spring Security Web » 5.0.7.RELEASE

Spring Security



1. Spring Security Config

[org.springframework.security](#) » [spring-security-config](#)

Spring Security

Last Release on Aug 16, 2021

Spring Security Config » 5.0.7.RELEASE

Spring Security



1. Spring Security Core

[org.springframework.security](#) » [spring-security-core](#)

Spring Security

Last Release on Aug 16, 2021

Spring Security Core » 5.0.7.RELEASE

Spring Security



1. Spring Security Taglibs

[org.springframework.security](#) » [spring-security-taglibs](#)

Spring Security

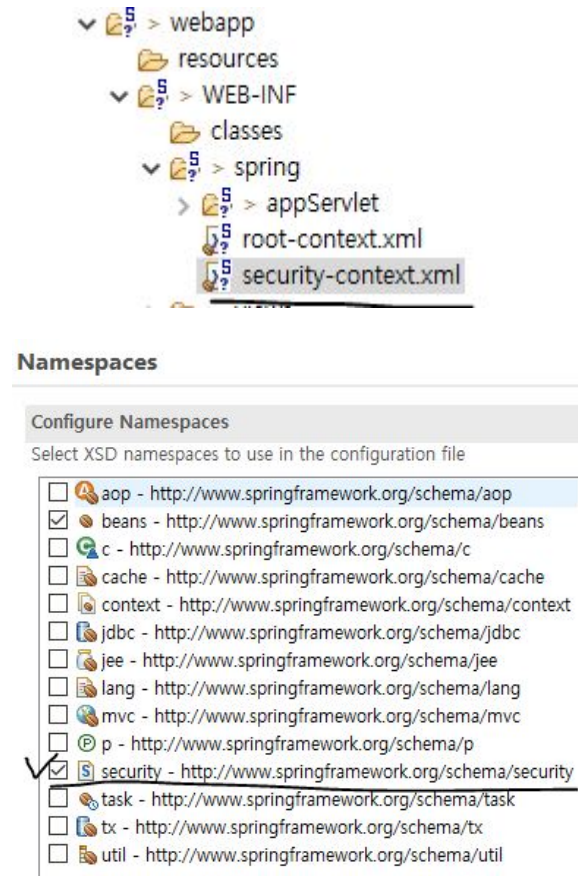
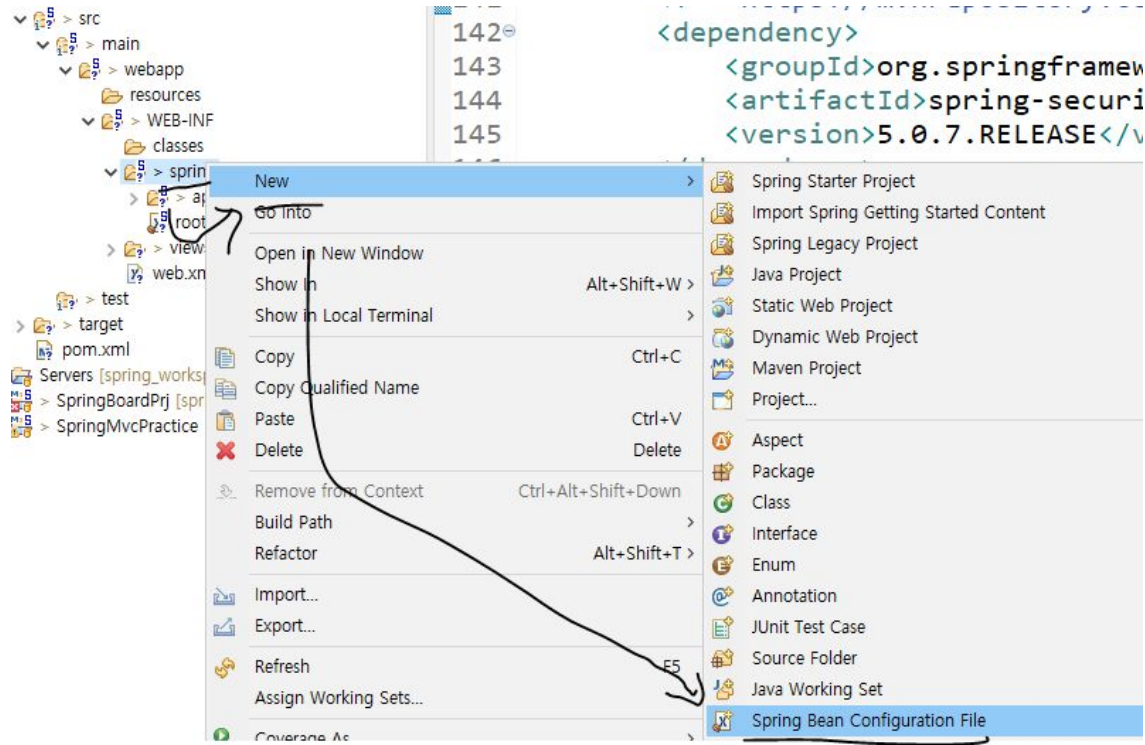
Last Release on Aug 16, 2021

Spring Security Taglibs » 5.0.7.RELEASE

Spring Security

먼저 security_prj 프로젝트를 생성합니다.

스프링 생성 처리는 05번 교안을 따라가고, 거기에 추가로 위의 스프링 시큐리티 관련 라이브러리를 pom.xml에 추가합니다.



이제 프로젝트 내부 WEB_INF폴더 내부의 spring폴더에 Spring Bean Configuration File을 선택해 security-context.xml파일을 만듭니다.

이 파일의 용도는 스프링 시큐리티용 bean 관리 컨테이너입니다.

security-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd" >

</beans>
```

이때, 디폴트로 생성한 xml파일 스키마는 에러가 나기 때문에
security-context.xml 파일의 초기 코드를 위와 같이 고쳐주세요.

web.xml ;

```
34      <!-- 필터 설정 -->
35      <filter>
36          <filter-name>springSecurityFilterChain</filter-name>
37          <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
38      </filter>
39      <filter-mapping>
40          <filter-name>springSecurityFilterChain</filter-name>
41          <url-pattern>/*</url-pattern>
42      </filter-mapping>
43 </web-app>
```

다음으로 web.xml 설정을 합니다.

필터를 이용해 스프링 동작에 관여하도록 코드를 수정해줍니다.


오타가 나지 않게 주의해주세요. 제대로 적어도 위와같이 에러가 납니다.

web.xml :

```
6      <!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
7      <context-param>
8          <param-name>contextConfigLocation</param-name>
9          <param-value>/WEB-INF/spring/root-context.xml
10             /WEB-INF/spring/security-context.xml
11      </param-value>
12 </context-param>
```

에러를 없애려면 먼저 빈 컨테이너에 있는 시큐리티 관련 로직을 web.xml에서 인식하도록 설정해줘야합니다.

위와 같이 기존의 root-context 인증 경로 아래에 security-context.xml을 추가해주세요.

 security-context.xml

```
<security:http>
    <security:form-login/>
</security:http>

<security:authentication-manager>
</security:authentication-manager>
```

다음으로 security-context.xml에

Authentication Manager(인증 매니저)를 추가해줍니다.

```
package org.ict.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import lombok.extern.log4j.Log4j;

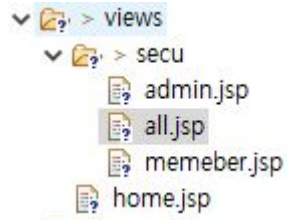
@Log4j
@RequestMapping("/secu/*")
@Controller
public class SecurityController {
```

서버 실행 후 에러가 나지 않는것을 확인했다면, **URI**를 세팅하겠습니다.

SecurityController를 추가합니다.

```
public class SecurityController {  
  
    @GetMapping("/all")  
    public void doAll() {  
        Log.info("모든 사람이 접속 가능한 all 로직");  
    }  
  
    @GetMapping("/member")  
    public void doMember() {  
        Log.info("회원들이 접속 가능한 member 로직");  
    }  
  
    @GetMapping("/admin")  
    public void doAdmin() {  
        Log.info("운영자만 접속 가능한 admin 로직");  
    }  
}
```

다음으로 권한에 따라 접속할 수 있는 형태를 차등으로 두기 위해서
비회원, 회원, 운영자 3단계로 나눠서 접속 주소를 컨트롤러에 세팅합니다.



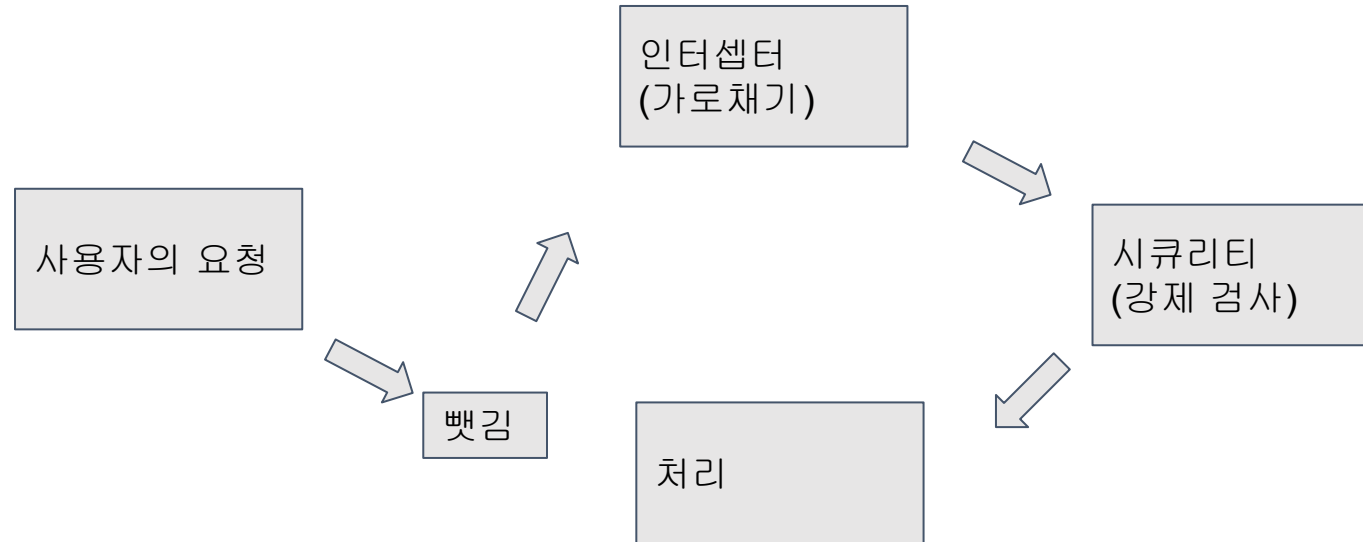
```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body>
10 <h1>all 주소</h1>
11 </body>
12 </html>
```

```
<body>
<h1>admin 주소</h1>
</body>
```

```
<body>
<h1>member 주소</h1>
</body>
```

views아래에 설정한 주소에 맞는 .jsp파일 3개를 만들어주신 다음

접속이 잘 되는지 확인부터 해보겠습니다.



연결이 완료되었다면 이제 필터와 스프링 시큐리티의 역할에 대해 보겠습니다.

기본적으로는 접속요청을 스프링 시큐리티가 가로채서 확인을 한 다음 진행시킬지 말지를 결정시킨다는 것입니다.

```
<security:http>

  <!-- /secu/all 접속시 모든 유저에게 접근 허용 -->
  <security:intercept-url pattern="/secu/all" access="permitAll" />

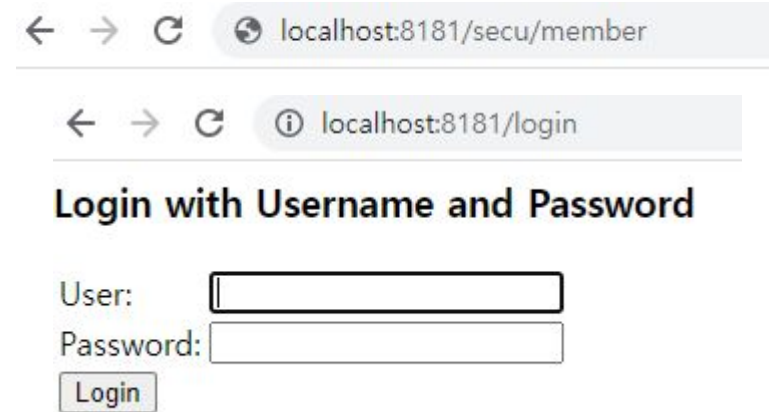
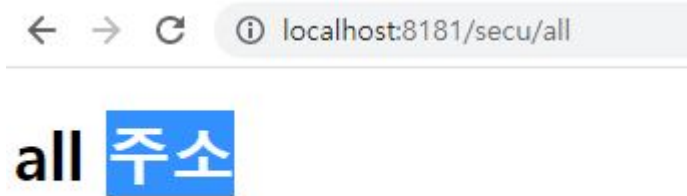
  <!-- /secu/member는 Member에게만 접근 허용 -->
  <security:intercept-url pattern="/secu/member" access="hasRole('ROLE_MEMBER')" />

  <security:form-login/>

</security:http>
```

이를 위해 **security-context** 내부에 어떤 url 접속을 가로챌지, 그리고 접근에 따른 처리를 어떻게 할지 세팅해야합니다.

security:http 내부에 **security:intercept-url**로 접근 url을 먼저 지정할 수 있고 **pattern**에는 접근패턴을, **access**에는 권한이 부여된 사용자를 처리할때는 위와 같이 **hasRole('ROLE_직접부여한권한명')** 을 적습니다.



이제 /secu/member는 접속이 안 되는것을 볼 수 있습니다.

MEMBER 권한을 부여받은 사용자가 로그인한 경우만 접근가능하기
때문입니다.

위의 로그인창은 우리가 만든것이 아닌 스프링 시큐리티가 임시로 생성하는
로그인창입니다. 나중에 커스터마이징하겠습니다.

```
<security:authentication-manager>

    <security:authentication-provider>
        <security:user-service>
            <security:user name="member" password="member" authorities="ROLE_MEMBER" />
        </security:user-service>
    </security:authentication-provider>

</security:authentication-manager>
```

이제 로그인 가능한 아이디 하나를 security-context.xml에 부여해보겠습니다.
이러면 동작여부를 DB를 거치지 않고도 임시로 확인 가능합니다.

아이디 비밀번호는 name, password에 각각 member로
권한은 authorities에 "ROLE_MEMBER"를 줍니다.



member계정으로 로그인을 하면 위와 같이

500에러가 뜨면서 PasswordEncoder가 null이라는 오류가 뜹니다.

우선 저렇게 나오면 제대로 설정하신겁니다.

```
<security:authentication-manager>

    <security:authentication-provider>
        <security:user-service>
            <security:user name="member" password="{noop}member" authorities="ROLE_MEMBER" />
        </security:user-service>
    </security:authentication-provider>

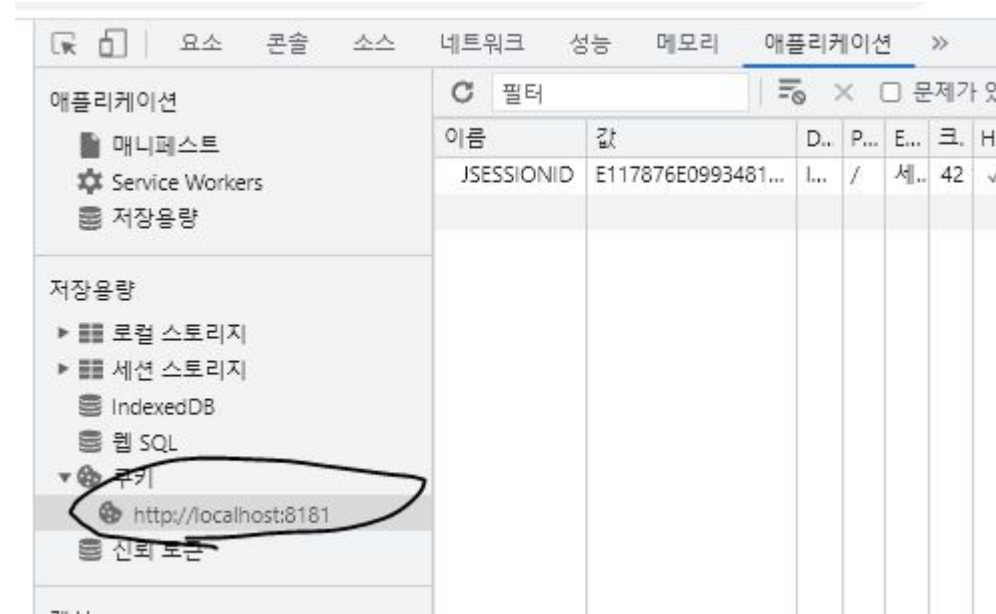
</security:authentication-manager>
```

스프링 시큐리티에서는 스프링 5버전부터는 반드시 비밀번호를 복호화해서 처리하도록 강제됩니다.

우리는 복호화 설정을 하지 않았기 때문에, 우선 복호화 이전에 로그인만 테스트하기위해 비밀번호 앞에 "{noop}" 를 추가해 복호화 기능을 꺼둡니다.

← → ↻ ⓘ localhost:8181/secu/member

member 주소



다시 로그인하면 이제 /secu/member가 잘 접속됩니다.

로그인을 풀고싶다면 f12->application -> stroage-> cookies 에서 우클릭후

쿠키를 전부 삭제해주면 로그인이 풀립니다.

```
<!-- /secu/admin은 admin에게만 접근 허용 -->
<security:intercept-url pattern="/secu/admin" access="hasRole('ROLE_ADMIN')" />

</security:http>

<security:authentication-manager>
    <security:authentication-provider>
        <security:user-service>
            <security:user name="member" password="{noop}member" authorities="ROLE_MEMBER" />
            <security:user name="admin" password="{noop}admin" authorities="ROLE_MEMBER,ROLE_ADMIN" />
        </security:user-service>
    </security:authentication-provider>
</security:authentication-manager>
```

추가

다음으로, 이제 ADMIN 권한을 처리하겠습니다.

ADMIN은 MEMBER, ADMIN 에 모두 접속 가능하게 두 권한을 나열합니다.

설정 후 /secu/admin도 접속이 의도대로 처리되는지 확인해주세요.



admin 주소에 member로 로그인시 다음과 같이 권한 부족으로 접속이 금지됩니다.

이제 톰캣 에러페이지가 아닌 다른 화면을 처리해보겠습니다.

```
<security:http auto-config="true" use-expressions="true">

    <!-- /secu/all 접속시 모든 유저에게 접근 허용 -->
    <security:intercept-url pattern="/secu/all" access="permitAll" />

    <!-- /secu/member는 member에게만 접근 허용 -->
    <security:intercept-url pattern="/secu/member" access="hasRole('ROLE_MEMBER')" />

    <!-- /secu/admin은 admin에게만 접근 허용 -->
    <security:intercept-url pattern="/secu/admin" access="hasRole('ROLE_ADMIN')" />

    <security:form-login/>

    <security:access-denied-handler error-page="/accessError" />

</security:http>
```

이제 접근 제한 페이지를 바꿔보겠습니다.

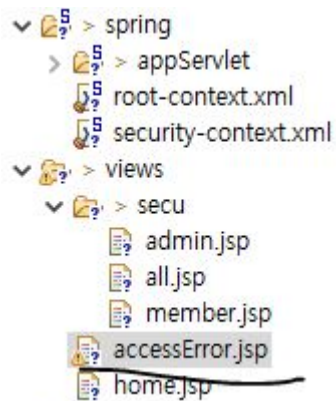
security-context.xml 상단의 **security:http** 태그 내부를 위와 같이 고칩니다.

접근 주소는 **/accessError** 이외에 다른 주소를 넣으면 작동하지 않습니다.

```
1 package org.ict.controller;
2
3 import org.springframework.security.core.Authentication;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.GetMapping;
7
8 import lombok.extern.log4j.Log4j;
9
10 @Log4j
11 @Controller
12 public class CommonController {
13
14     @GetMapping("/accessError")
15     public void accessDenied(Authentication auth, Model model) {
16
17         log.info("접근 거부 : " + auth);
18
19         model.addAttribute("errorMessage", "접근 거부");
20     }
21 }
```

그리고 컨트롤러로 CommonController를 생성해서

접근주소 /accessError를 처리하도록 세팅합니다.



```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
4 <%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec" %>
5 <%@ page import="java.util.*" %>
6 <!DOCTYPE html>
7 <html>
8 <head>
9   <meta charset="UTF-8">
10  <title>Insert title here</title>
11 </head>
12 <body>
13   <h1>접근 실패!</h1>
14   <h2><c:out value="${SPRING_SECURITY_403_EXCEPTION.getMessage()}" /></h2>
15
16   <h2><c:out value="${errorMessage}" /></h2>
17 </body>
18 </html>
```

views 폴더에 accessError.jsp를 생성해주시고

에러메세지를 출력하도록 코드를 작성해보겠습니다.

← → ↻ ⓘ localhost:8181/secu/admin

접근 실패!

Access is denied

접근 거부

특정 페이지에 권한 없는 계정 접속시 에러가 잘 뜨는지 확인해주세요.

```
@Log4j
public class CustomAccessDeniedHandler implements AccessDeniedHandler {

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
        AccessDeniedException accessDeniedException) throws IOException, ServletException {

    }

}
```

위 방식은 스프링 시큐리티에서 제공하는 기본 에러 페이지 처리 방식이고 커스터마이징을 할 경우는 **AccessDeniedHandler** 인터페이스를 직접 오버라이딩해 구현해야 합니다.

대신 오버라이딩을 하는 만큼 세팅된 동작에 추가로 작업을 작성할 수 있습니다. 먼저 **org.ict.security** 패키지를 만들고 **AccessDeniedHandler**를 구현할 **CustomAccessDeniedHandler**를 만듭니다.


```
@Log4j
public class CustomAccessDeniedHandler implements AccessDeniedHandler {

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
        AccessDeniedException accessDeniedException) throws IOException, ServletException {

        Log.error("커스텀 접근 거부 핸들러 실행");
        Log.error("/accessError 페이지로 리다이렉트");

        response.sendRedirect("/accessError");
    }
}
```

자동 구현되는 `handle` 메서드에는 에러가 발생했을때 처리할 코드를 작성하는데 여기서는 리다이렉트 코드를 작성하겠습니다.

```
<bean id="customAccessDenied" class="org.ict.security.CustomAccessDeniedHandler" />
```

<security:http> => 내부속성 삭제

```
<!-- /secu/all 접속시 모든 유저에게 접근 허용 -->
```

```
<security:intercept-url pattern="/secu/all" access="permitAll" />
```

```
<!-- /secu/member는 member에게만 접근 허용 -->
```

```
<security:intercept-url pattern="/secu/member" access="hasRole('ROLE_MEMBER')" />
```

```
<!-- /secu/admin은 admin에게만 접근 허용 -->
```

```
<security:intercept-url pattern="/secu/admin" access="hasRole('ROLE_ADMIN')" />
```

```
<security:form-login login-page="/customLogin" />
```

```
<!-- <security:access-denied-handler error-page="/accessError" /> -->
```

```
<security:access-denied-handler ref="customAccessDenied"/>
```

```
</security:http>
```

customAccessDenied

추가

이제 디폴트 **AccessDeniedHandler**를 사용하는 대신
커스텀 핸들러를 쓰도록 **security-context.xml**을 위와 같이 수정합니다.
security:http 내부 속성도 모두 삭제해 디폴트 설정을 못쓰게 만들어주세요.
beans graph에도 추가되어있는지 확인해주세요.
스프링 시큐리티는 계층구조가 매우 복잡하니 주의하세요.

```
er.HomeController - Welcome home! The client locale is ko_KR.|
CustomAccessDeniedHandler - 커스텀 접근 거부 핸들러 실행
CustomAccessDeniedHandler - /accessError 페이지로 리다이렉트
er.CommonController - 접근 거부 : org.springframework.security.authenticat
```

콘솔창에 이전과 달리 먼저 **CustomAccessDeniedHandler**를 거치는 내용의 로그가 뜬다면 제대로 연동한것입니다.

```
<security:form-login login-page="/customLogin" />  
  

```

```
@GetMapping("/customLogin")
public void loginInput(String error, String logout, Model model) {
    Log.info("error 여부 : " + error);
    Log.info("logout 여부 : " + logout);

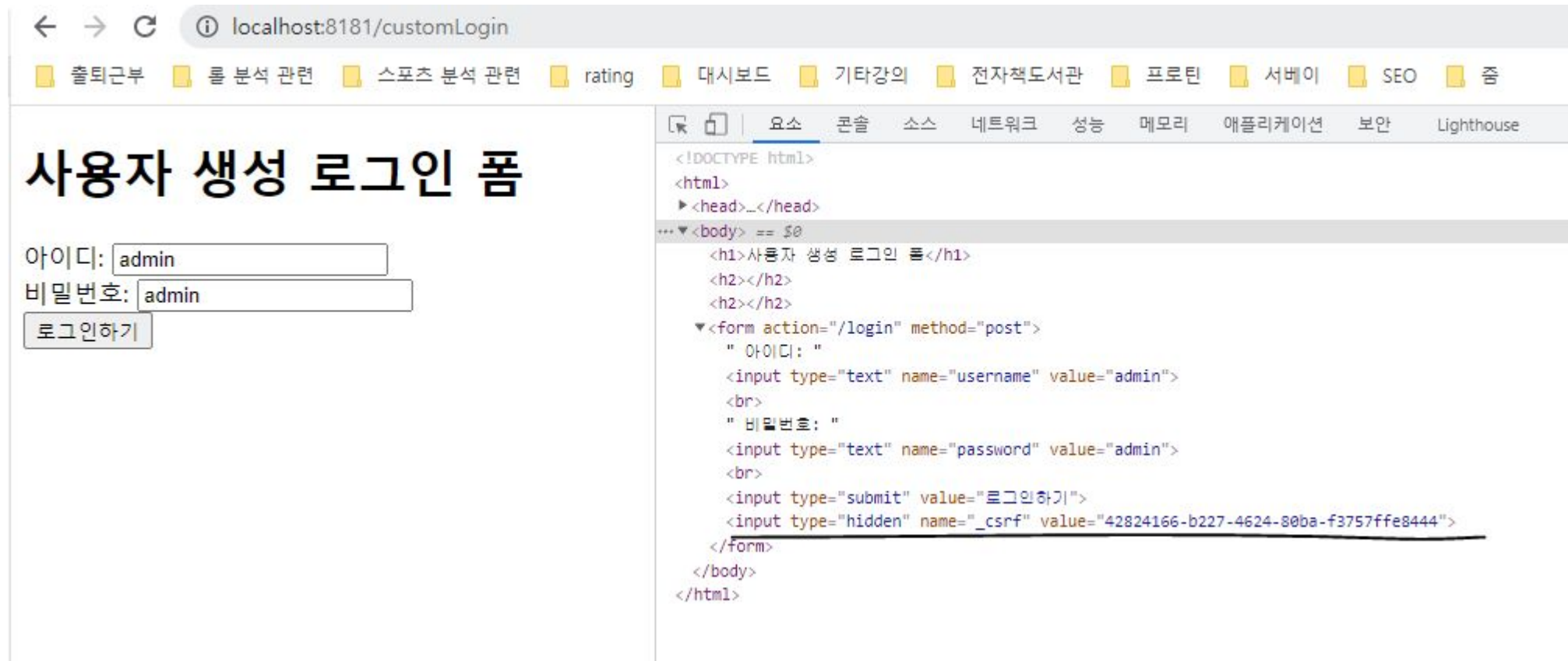
    if(error != null) {
        model.addAttribute("error", "로그인 관련 에러입니다. 계정확인을 다시해주세요");
    }
    if(logout != null) {
        model.addAttribute("logout", "로그아웃 했습니다.");
    }
}
```

그리고 CommonController에 새롭게 /customLogin 을 주소로 하는 메서드를 만들고

에러로 접근했는지 로그아웃 상태라 접근했는지 구분하도록 사전세팅합니다.


```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
4 <!DOCTYPE html>
5 <html>
6 <head>
7 <meta charset="UTF-8">
8 <title>Insert title here</title>
9 </head>
10 <body>
11     <h1>사용자 생성 로그인 폼</h1>
12     <h2><c:out value="${error}" /></h2>
13     <h2><c:out value="${logout}" /></h2>
14
15     <form action="/login" method='post'>
16         아이디: <input type="text" name="username" value="admin"><br/>
17         비밀번호: <input type="text" name="password" value="admin"><br/>
18         <input type="submit" value="로그인하기">
19         <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
20     </form>
21 </body>
22 </html>
```

views에는 customLogin.jsp를 준비합니다. 이 페이지는 어떻게 이 페이지에 오게 되었는지와, 로그인시 사용할 폼을 제공합니다.
하단의 csrf 토큰에 대해서는 다음페이지에 설명합니다.

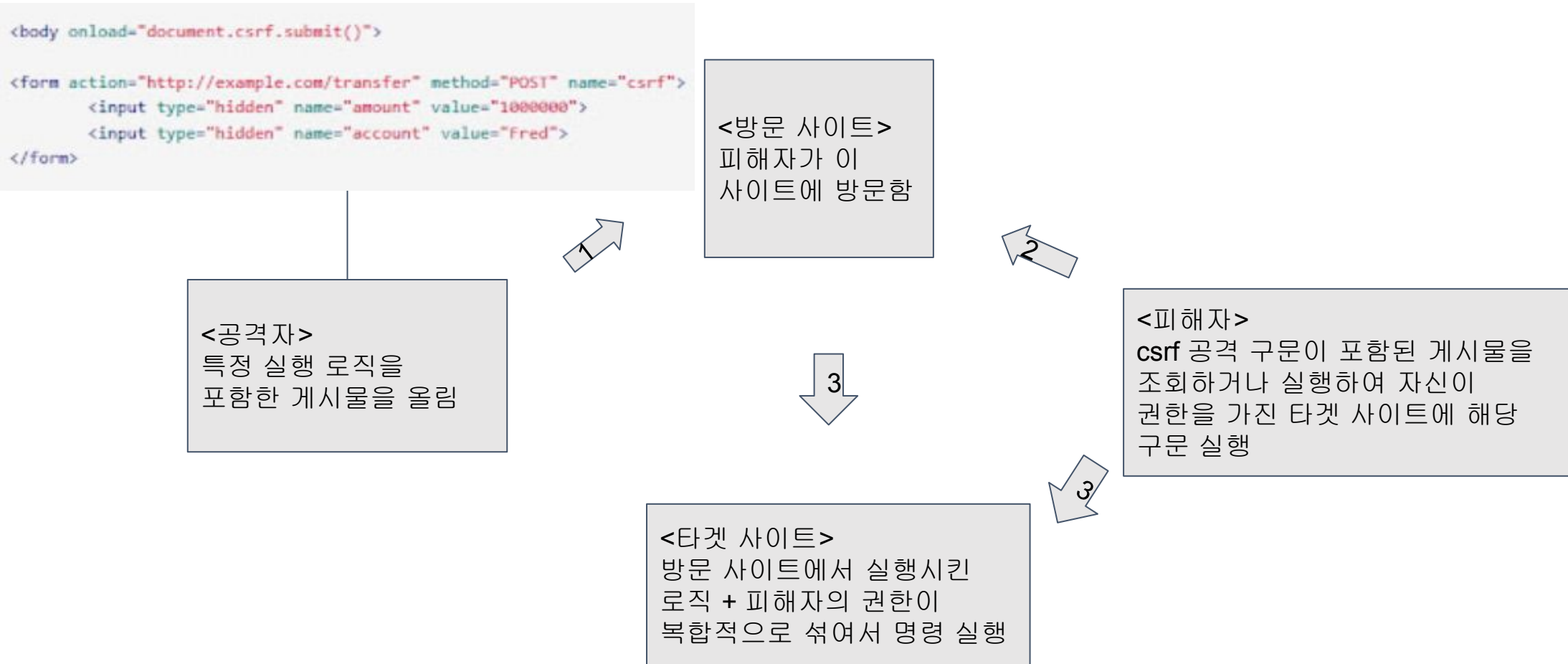


The screenshot shows a web browser at `localhost:8181/customLogin`. The page title is "사용자 생성 로그인 폼". The form contains two input fields: "아이디:" with the value "admin" and "비밀번호:" with the value "admin". Below the fields is a button labeled "로그인하기".

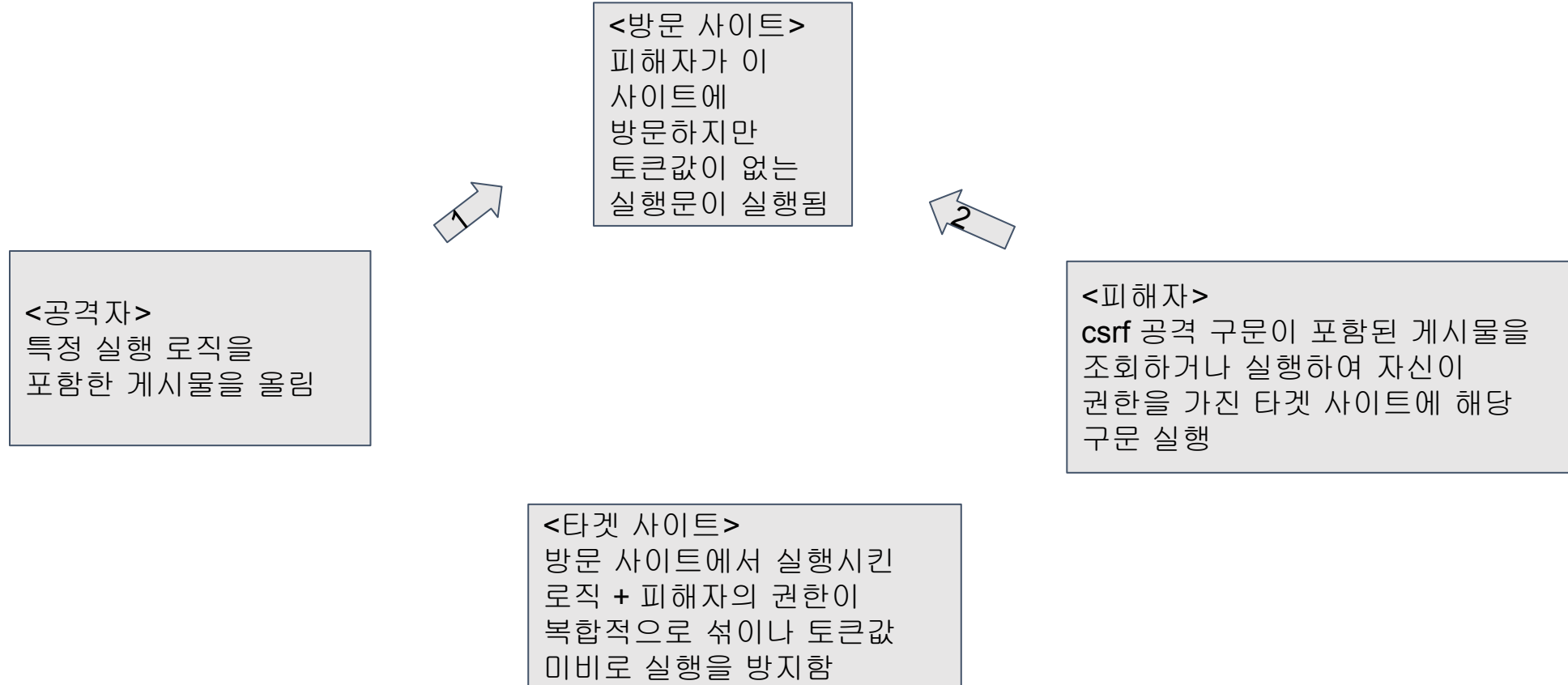
The right side of the image shows the browser's developer tools with the HTML source code expanded. The code is as follows:

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body> == $0
    <h1>사용자 생성 로그인 폼</h1>
    <h2></h2>
    <h2></h2>
    <form action="/login" method="post">
      " 아이디: "
      <input type="text" name="username" value="admin">
      <br>
      " 비밀번호: "
      <input type="text" name="password" value="admin">
      <br>
      <input type="submit" value="로그인하기">
      <input type="hidden" name="_csrf" value="42824166-b227-4624-80ba-f3757ffe8444">
    </form>
  </body>
</html>
```

적용 후 잘 보시면 아까 csrf토큰 관련 세팅을 해준 쪽의 value가 임의로
배정된것을 볼 수 있고, csrf토큰값은 쿠키가 삭제될때마다 바뀝니다.

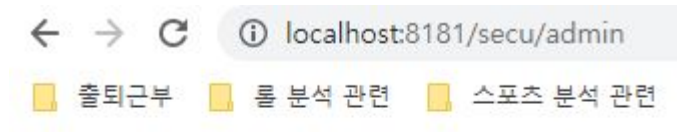


csrf는 cross-site request forgery 공격이라고 불리며, csrf검증로직이 없는 사이트는, form이나 img태그 내부의 링크설정을 곧이곧대로 받아 처리하기 때문에 악의적으로 공격할 수 있게 됩니다.



csrf토큰은 매번 value값이 바뀌고, hidden으로 포함되어 들어가기 때문에 공격자 입장에서는 고정된 쿼리문만 전송해서는 더이상 명령이 작동하지 않고

매번 바뀌는 csrf토큰값을 그때그때 찍어서 맞춰야 해서 사실상 완전방어가 됩니다.



admin 주소

```
.controller.CommonController - logout 여부 : null  
.controller.HomeController - Welcome home! The client locale is ko_KR.  
.controller.SecurityController - 회원들이 접속 가능한 member 로직  
.controller.SecurityController - 운영자만 접속 가능한 admin 로직
```

이제 로그인을 해보면 제대로 로그인시 제대로 된 페이지가 나오고

아이디 비밀번호를 잘못 입력하면 에러페이지가 나옵니다.

다만, 권한검증 등의 로직이 확실하게 처리되지는 않습니다.

```
@Log4j|
public class CustomLoginSuccessHandler implements AuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
        Authentication authentication) throws IOException, ServletException {
        // TODO Auto-generated method stub
    }
}
```

이제 로그인 성공을 처리해보도록 하겠습니다.

AuthenticationSuccessHandler를 이용해서 처리합니다.

org.ict.security에 먼저 **CustomLoginSuccessHandler**를 생성하고

AuthenticationSuccessHandler 인터페이스를 구현하겠습니다.

```
public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
    Authentication authentication) throws IOException, ServletException {

    // 로그인 성공시 어떤 권한인지 체크하기 위해 부여받은 권한(들) 불러오기
    // ROLE_ADMIN의 경우는 ROLE_MEMBER가 함께 부여되기때문에 경우에 따라 권한이 여럿일 수 있음
    Log.warn("로그인 성공");
    List<String> roleList = new ArrayList<>();

    for (GrantedAuthority role : authentication.getAuthorities()) {
        roleList.add(role.getAuthority());
    }

    // roleList에 포함된 권한을 통해 로그인 계정의 권한에 따라 처리
    Log.warn("부여받은 권한들 : " + roleList);
    if(roleList.contains("ROLE_ADMIN")) {
        response.sendRedirect("/secu/admin");
        return;
    }
    if(roleList.contains("ROLE_MEMBER")) {
        response.sendRedirect("/secu/member");
        return;
    }

    response.sendRedirect("/");
}
```

오버라이딩하는 `onAuthenticationSuccess` 메서드는 로그인 성공시 실행될 로직들을 작성해줍니다.
`return`구문도 각 `if`마다 붙여주세요. 안 붙여주면 리다이렉트가 중복호출됩니다.


```
<bean id="customAccessDenied" class="org.ict.security.CustomAccessDeniedHandler" />
<bean id="customLoginSuccess" class="org.ict.security.CustomLoginSuccessHandler" />

<security:http>

    <!-- /secu/all 접속시 모든 유저에게 접근 허용 -->
    <security:intercept-url pattern="/secu/all" access="permitAll" />

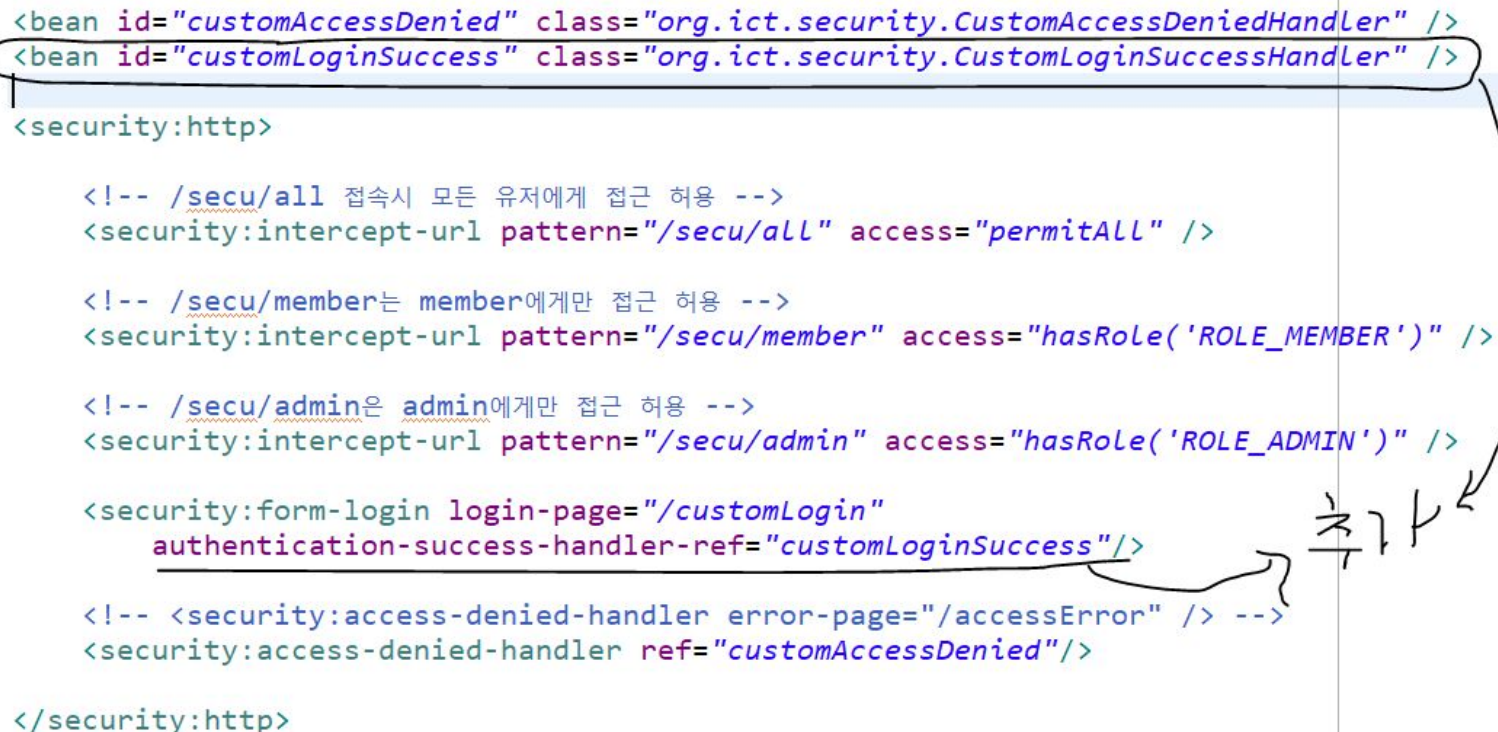
    <!-- /secu/member는 member에게만 접근 허용 -->
    <security:intercept-url pattern="/secu/member" access="hasRole('ROLE_MEMBER')" />

    <!-- /secu/admin은 admin에게만 접근 허용 -->
    <security:intercept-url pattern="/secu/admin" access="hasRole('ROLE_ADMIN')" />

    <security:form-login login-page="/customLogin"
        authentication-success-handler-ref="customLoginSuccess"/>

    <!-- <security:access-denied-handler error-page="/accessError" /> -->
    <security:access-denied-handler ref="customAccessDenied"/>

</security:http>
```



이제 기존 로그인 처리로직을 커스텀 처리로직으로 바꿔 적용하기 위해 security-context.xml 내용을 바꿔줍니다.

앞으로도 다른 로직을 커스텀 작성시

1. 커스텀 로직 작성 => 2. security-context.xml 내부에 bean 생성
- => 3. 태그 적용 순으로 진행하시면 됩니다.

← → ↻ ⓘ localhost:8181/secu/admin

출퇴근부 룰 분석 관련 스포츠 분석 관련

admin 주소

controller.CommonController - logout 처리 : null
security.CustomLoginSuccessHandler - 로그인 성공
security.CustomLoginSuccessHandler - 부여받은 권한들 : [ROLE_ADMIN, ROLE_MEMBER]
controller.SecurityController - 운영자만 접속 가능한 admin 로직

← → ↻ ⓘ localhost:8181/secu/member

출퇴근부 룰 분석 관련 스포츠 분석 관련

member 주소

ity.CustomLoginSuccessHandler - 로그인 성공
ity.CustomLoginSuccessHandler - 부여받은 권한들 : [ROLE_MEMBER]
oller.SecurityController - 회원들이 접속 가능한 member 로직

이제 접근 계정에 따라 로그인이 다르게 처리 잘 되는것을 볼 수 있습니다.

```
<security:form-login login-page="/customLogin"
    authentication-success-handler-ref="customLoginSuccess"/>

<!-- <security:access-denied-handler error-page="/accessError" /> -->
<security:access-denied-handler ref="customAccessDenied"/>

<security:logout logout-url="/customLogout" invalidate-session="true" />
```

추가

```
</security:http>
```

이번에는 로그아웃을 처리해보겠습니다.

security-context.xml에는 **security:logout** 태그를 사용하며,
로그아웃시 사용할 주소를 **logout-url** 속성에 저장하고
invalidate-session 속성에 **true**를 줘서 로그아웃시 세션을 파기하게 합니다.

security:http 내부에 작성해야 합니다.

CommonController.java

```
@GetMapping("/customLogout")
public void logoutGet() {

    Log.info("로그아웃 폼으로 이동");
}
```

customLogout.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6   <meta charset="UTF-8">
7   <title>Insert title here</title>
8 </head>
9 <body>
10   <h1>로그아웃 처리 페이지</h1>
11   <form action="/customLogout" method="post">
12     <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
13     <input type="submit" value="로그아웃" />
14   </form>
15 </body>
16 </html>
```

CommonController에는
이제 customLogout을 처리하는 폼으로 이동하는 메서드를 추가합니다.

추가로 customLogout.jsp도 생성해줍니다.

} CommonController.java |

```
@PostMapping("/customLogout")  
public void logoutPost() {  
    Log.info("포스트방식으로 로그아웃요청 처리");  
}
```

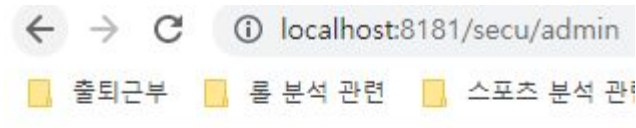
폼에서 날린 post요청을 처리할 수 있도록 CommonController에 마저 로그아웃 로그를 남깁니다.

admin.jsp

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body>
10 <h1>admin 주소</h1>
11 <a href="/customLogout">로그아웃페이지 이동</a>
12 </body>
13 </html>
```

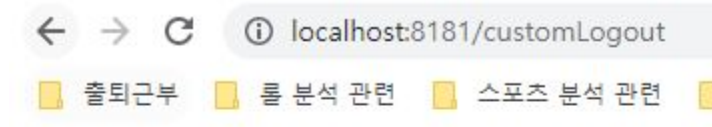
이제 admin.jsp에 customLogout페이지로 이동할 수 있는

링크를 하나 작성합니다. 클릭시 진짜 로그아웃할지 확인하는 폼으로 이동합니다.



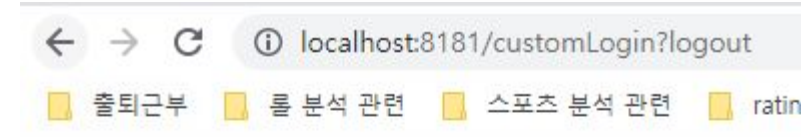
admin 주소

[로그아웃페이지 이동](#)



로그아웃 처리 페이지

로그아웃



사용자 생성 로그인 폼

로그아웃 했습니다.

아이디:

비밀번호:

이제 로그아웃이 실제로 되는지 체크해보세요.

```
CREATE TABLE users(  
    username varchar2(50) not null primary key,  
    password varchar2(100) not null,  
    enabled char(1) default '1');  
  
CREATE TABLE authorities(  
    username varchar2(50) not null,  
    authority varchar2(50) not null,  
    constraint fk_authorities_users foreign key(username) references users(username));  
  
create unique index ix_auth_username on authorities (username, authority);
```

다음으로 이제 **DB**와 연동해서 로그인 처리를 하는법을 알아보겠습니다.

먼저 스프링 시큐리티 디폴트 **DB**테이블 구조를 본따서 테이블을 만들어보겠습니다.

username, password, authority등 컬럼명 및 형식이 고정되어있습니다.

```
insert into users (username, password) values ('user00', 'pw00');
insert into users (username, password) values ('member00', 'pw00');
insert into users (username, password) values ('admin00', 'pw00');

insert into authorities (username, authority) values ('user00', 'ROLE_USER');
insert into authorities (username, authority) values ('member00', 'ROLE_MANAGER');
insert into authorities (username, authority) values ('admin00', 'ROLE_MANAGER');
insert into authorities (username, authority) values ('admin00', 'ROLE_ADMIN');

commit;
```

	USERNAME	PASSWORD	ENABLED
1	user00	pw00	1
2	member00	pw00	1
3	admin00	pw00	1

	USERNAME	AUTHORITY
1	admin00	ROLE_ADMIN
2	admin00	ROLE_MANAGER
3	member00	ROLE_MANAGER
4	user00	ROLE_USER

디폴트 옵션에서는 권한을 **ROLE_USER**, **ROLE_MANAGER**, **ROLE_ADMIN**으로 분류합니다.

그리고 **ROLE_ADMIN** 권한은 **ROLE_MANAGER**를 포함하기 때문에 이에 맞춰서 **insert**를 양 테이블에 해줍니다.

```
<security:authentication-manager>
```

```
<security:authentication-provider>
```

```
<!-- <security:user-service>
```

```
<security:user name="member" password="{noop}member" authorities="ROLE_MEMBER" />
```

```
<security:user name="admin" password="{noop}admin" authorities="ROLE_MEMBER,ROLE_ADMIN" />
```

```
</security:user-service> -->
```

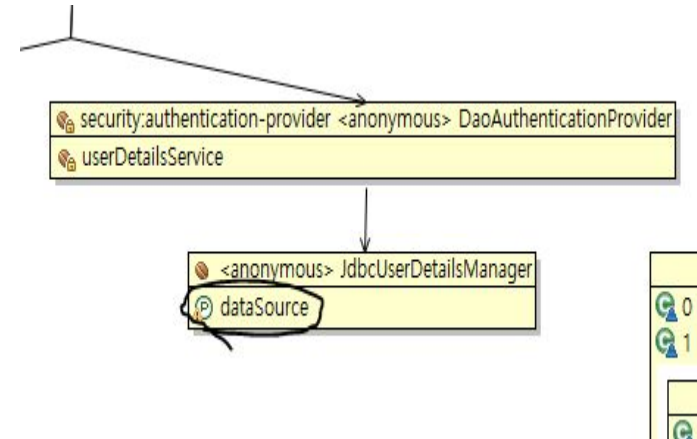
```
<security:jdbc-user-service data-source-ref="dataSource"/>
```

```
</security:authentication-provider>
```

```
</security:authentication-manager>
```

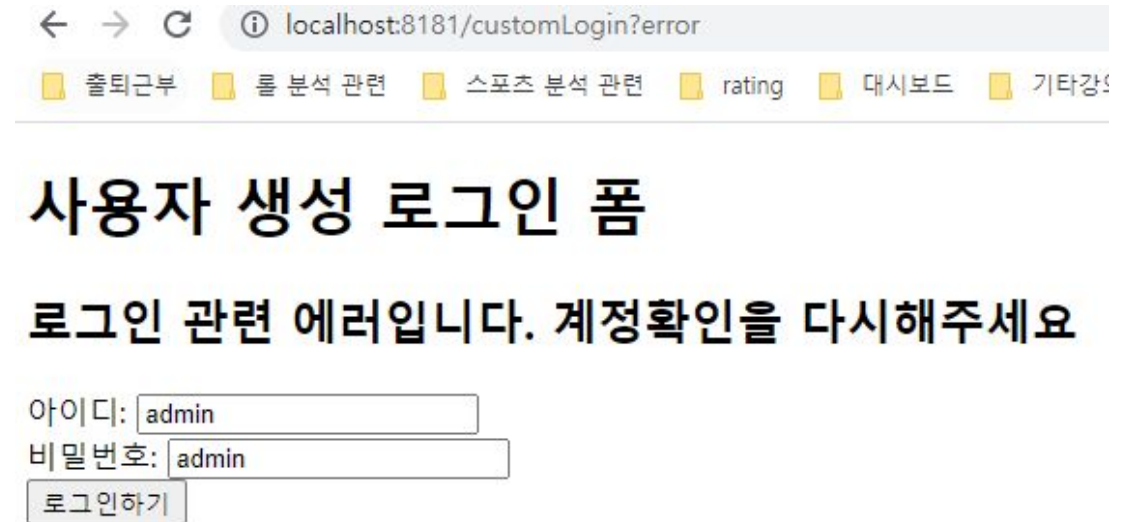
삭제

추가



security-context.xml 내부의 인증 매니저는 이제 **DB**를 쓰도록 변경되어야 합니다.

root-context.xml에 **dataSource**라는 이름으로 히카리 설정시 했던 **DB연동**이 되어있으므로 연결해줍니다.



이제 서버를 돌려서 로그인창에서 user00, member00, admin00 계정으로

패스워드가 복호화되지 않은 평문이라 뜨는 500에러가 뜨지만 DB내용을 근거로 판단하는것을 볼 수 있습니다.


```
10 <bean id="customAccessDenied" class="org.ict.security.CustomAccessDeniedHandler" />
11 <bean id="customLoginSuccess" class="org.ict.security.CustomLoginSuccessHandler" />
12 <bean id="bcryptPasswordEncoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
13
14 <security:http>
15
```

추가

```
    <security:authentication-manager>
        <security:authentication-provider>
            <security:jdbc-user-service data-source-ref="dataSource"/>
            <security:password-encoder ref="bcryptPasswordEncoder"/>
        </security:authentication-provider>
    </security:authentication-manager>
```

추가

이제 비밀번호 복호화를 위해 스프링 시큐리티에서 제공하는 BcryptPasswordEncoder를 사용해보겠습니다.

security-context.xml을 수정합니다.


```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({
    "file:src/main/webapp/WEB-INF/spring/root-context.xml",
    "file:src/main/webapp/WEB-INF/spring/security-context.xml"
})
@Log4j
public class MemberTests {
    @Autowired// 복호화 담당
    private PasswordEncoder pwen;
    @Autowired// DB접근 담당
    private DataSource ds;

    @Test
    public void testCryptDefaultDB() {

        String[] idList = {"user00", "member00", "admin00"};
```

```
@Test
public void testCryptDefaultDB() {

    String[] idList = {"user00", "member00", "admin00"};

    String sql = "UPDATE USERS set password=? WHERE username=?";
    try {
        Connection con = ds.getConnection();

        for(String id : idList) {
            PreparedStatement pstmt = con.prepareStatement(sql);
            pstmt.setString(1, pwen.encode("pw00")); // pwen.encode("문자열") 시 자동복호화
            pstmt.setString(2, id);
            pstmt.executeUpdate();
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

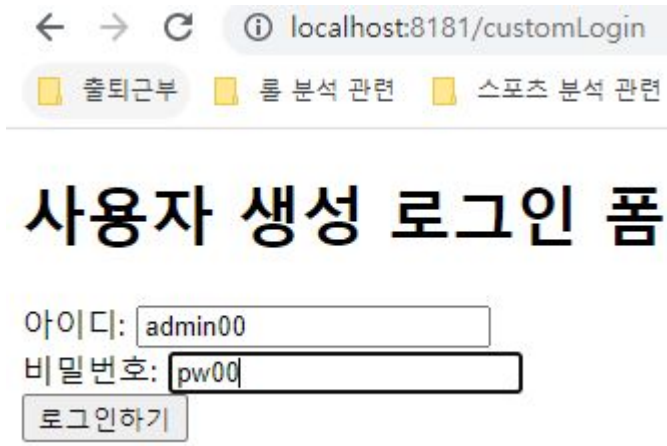
이제 src/test/java 내부에 org.ict.security 패키지를 생성한 후
평문 비밀번호를 전부 복호화하기 위해 테스트코드를 실행하겠습니다.
MemberTests클래스 추가 후 위와 같이 작성해주세요.
왼쪽 그림과 오른쪽 그림은 하나의 파일을 나눠서 집어넣은것입니다.

	USERN...	PASSWORD	ENABLED
1	user00	\$2a\$10\$9vvFuumGTwBhdPdHu.rGdOneGY72.ZFCdrNfLRdbOtVf0RCI4aHzi	1
2	member00	\$2a\$10\$74MS2JQFUPZ.PFmuuBGDwuDBiuhbFYI2MiLeA7rz0.PnD2wqnr5/i	1
3	admin00	\$2a\$10\$IL.4GTxbIfctxFDFeohIi.uQPRIIMZjr.EXqJOulYveDg2RbnFqpC	1

이제 실행 후 **SELECT * FROM users;** 를 해보면
위와 같이 복호화된 비밀번호가 교체되어 들어가 있습니다.

password 컬럼이 **varchar2(100)**인 이유는 복호화시 비밀번호가 60자
이상으로 늘어나기 때문입니다.

위와 같이 저장되기때문에 이제 **DB관리자**도 비번이 정확하게 뭔지
알 방법이 없습니다.



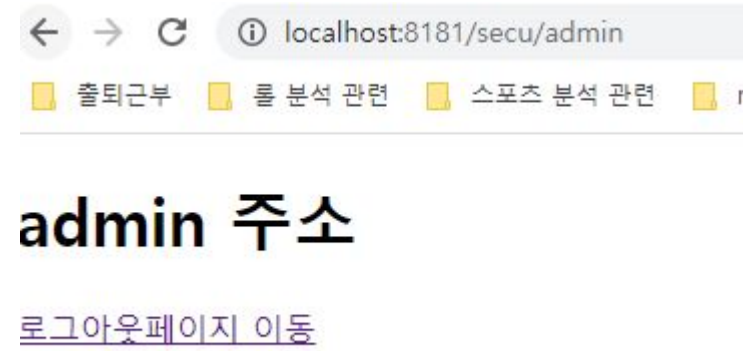
← → ↻ ⓘ localhost:8181/customLogin

출퇴근부 룰 분석 관련 스포츠 분석 관련

사용자 생성 로그인 폼

아이디:

비밀번호:



← → ↻ ⓘ localhost:8181/secu/admin

출퇴근부 룰 분석 관련 스포츠 분석 관련 r

admin 주소

[로그아웃페이지 이동](#)

로그인 시 문제 없이 진행되는지 확인해주세요.

```
CREATE TABLE member_tbl(  
    userid varchar2(50) not null primary key,  
    userpw varchar2(100) not null,  
    username varchar2(100) not null,  
    regdate date default sysdate,  
    update date default sysdate,  
    enabled char(1) default '1');  
  
CREATE TABLE member_auth(  
    userid varchar2(50) not null,  
    auth varchar2(50) not null,  
    constraint fk_member_auth foreign key(userid) references member_tbl(userid));
```

이제 사용자가 설정한 커스텀 DB를 활용한 로그인 방식을 처리해보겠습니다.

먼저 커스텀 DB구조부터 생성해보겠습니다.

위의 쿼리문을 실행해 테이블을 만들어주세요.

```
@Test
public void testCryptCustomDB() {
    try {
        Connection con = ds.getConnection();
        String sql = "INSERT INTO member_tbl(userid, userpw, username) values(?,?,?)";

        for(int i=0; i<30; i++) {
            PreparedStatement pstmt = con.prepareStatement(sql);

            pstmt.setString(2, pwen.encode("pw" + i)); // userpw에 복호화 비번 입력
            if(i < 10) {
                pstmt.setString(1, "user" + i);
                pstmt.setString(3, "준회원" + i);
            } else if(i < 20) {
                pstmt.setString(1, "user" + i);
                pstmt.setString(3, "정회원" + i);
            } else if(i < 30) {
                pstmt.setString(1, "user" + i);
                pstmt.setString(3, "운영자" + i);
            }
            pstmt.execute();
        }

    } catch(Exception e) {
        e.printStackTrace();
    }
}
```

테이블에 테스트코드를 이용해 데이터를 넣어보겠습니다.

MemberTest.java 내부에 **testCryptCustomDB** 메서드를 만들어 실행합니다.

	USERID	USERPW	USERNAME	REGDATE	UPDATEDATE	ENABLI
7	user6	\$2a\$10\$Un.hgMul0/iSrZE8si5.sOjr0RRpvuxdAXtf0DZWx7i3OLgVKcWuO	준회원6	21/10/03	21/10/03	1
8	user7	\$2a\$10\$.Bjt4e0hTJ4QiPzjEVB0004YH6N9XAJHUfdQ9iCpuFoHZ918Bs8nq	준회원7	21/10/03	21/10/03	1
9	user8	\$2a\$10\$LHXUCzFnSpnsyVrSF2ePne/VYSoapFLYGiB0e4wDqTjywSVf4MrQu	준회원8	21/10/03	21/10/03	1
10	user9	\$2a\$10\$80P50zX.14mC0n/ISfS0d.0Fh0NEabA5MUT0RCzKM8oJEXQwaaID.	준회원9	21/10/03	21/10/03	1
11	user10	\$2a\$10\$pHCq4X74u3XpqdhvqvvtWrOoyqnD1ZFbmYdI8zp5noAd2JZAM1KL02	정회원10	21/10/03	21/10/03	1
12	user11	\$2a\$10\$XDQJnN9kwtEmXJVvbaYfj.teO2WYZ/kixIf1Hc40Zo4K2rIz1CFmC	정회원11	21/10/03	21/10/03	1
13	user12	\$2a\$10\$xLelpu./pjzIztAj81lcWuWhQPdKNjSeiyLaNP1YqvSoxMt2pHkEi	정회원12	21/10/03	21/10/03	1
14	user13	\$2a\$10\$DUSpm6DN.vDQIU5R504v6.HPif5Fg6C5DL1.ElqQZTBiYgqwyEgL.	정회원13	21/10/03	21/10/03	1
15	user14	\$2a\$10\$uw9o6SY4/aNjojIJ4VKzBuVPlZdRWVt61DrtgzW56YaXkpyUwVuJ.	정회원14	21/10/03	21/10/03	1
16	user15	\$2a\$10\$JFtmLqg2wU9PEHhr8//nb.dqQp3aEMBC1AGR5vzeX10db.GgLrUWy	정회원15	21/10/03	21/10/03	1
17	user16	\$2a\$10\$/flKwDBg2Zs0lt6ILcclGeYJV2jn6s26B2C5ec03lsLfyFzI.nOIi	정회원16	21/10/03	21/10/03	1
18	user17	\$2a\$10\$KiMpr.x6IRa.d3EDxun19ec0UY7YHSEK6mLwWGC1fJVAcZ83NiMgC	정회원17	21/10/03	21/10/03	1
19	user18	\$2a\$10\$0ASUxLThGrUHCY.d57YBUeP0RDnUu/0iIUZA24Tt35EOxbfolHYnq	정회원18	21/10/03	21/10/03	1
20	user19	\$2a\$10\$teR8caitdm/5saZvFBgx/.W8304041WbbkWj1tbEOxdB.UZTvltu	정회원19	21/10/03	21/10/03	1
21	user20	\$2a\$10\$XgjW8j8LrRvFMtU6BtMPee103dC1lwHfpfr6xBZgimvY/LRbaqAJC	운영자20	21/10/03	21/10/03	1
22	user21	\$2a\$10\$D6KCvGvKz0k01/vk3lhz9edMEx5WI.wZHfN2XA5sJZ/FEZH8ww9we	운영자21	21/10/03	21/10/03	1
23	user22	\$2a\$10\$t3A3oPZKdLJ98FDa5HAmSOMgSYndD0Wy6QHvg2pSVQfCy4qhNCB7C	운영자22	21/10/03	21/10/03	1
24	user23	\$2a\$10\$GxDiljuYCzhCVuPjBu3NSezk/OXMXkg.qY6ZEZ5WvkeDxAuKPvLt2	운영자23	21/10/03	21/10/03	1

처리 후 sql에서 `select * from member_tbl;` 실행해
위와 같이 준회원, 정회원, 운영자가 10개씩 합쳐서 30개 들어가면
성공입니다.


```
@Test
public void testInsertAuth() {
    try {
        Connection con = ds.getConnection();
        String sql = "INSERT INTO member_auth(userid, auth) values(?,?)";

        for(int i=0; i<30; i++) {
            PreparedStatement pstmt = con.prepareStatement(sql);

            if(i < 10) {
                pstmt.setString(1, "user" + i);
                pstmt.setString(2, "ROLE_USER");
            } else if(i < 20) {
                pstmt.setString(1, "user" + i);
                pstmt.setString(2, "ROLE_MEMBER");
            } else if(i < 30) {
                pstmt.setString(1, "user" + i);
                pstmt.setString(2, "ROLE_ADMIN");
            }
            pstmt.execute();
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

이제, 이름에 맞는 권한을 부여하겠습니다.

`testInsertAuth` 메서드를 테스트코드로 작성, 실행해 맞게 들어갔는지 확인해주세요.

커스텀 DB이므로 `ROLE_MANAGER`가 아닌 `ROLE_MEMBER`입니다.

	USERID	AUTH
7	user6	ROLE_USER
8	user7	ROLE_USER
9	user8	ROLE_USER
10	user9	ROLE_USER
11	user10	ROLE_MEMBER
12	user11	ROLE_MEMBER
13	user12	ROLE_MEMBER
14	user13	ROLE_MEMBER
15	user14	ROLE_MEMBER
16	user15	ROLE_MEMBER
17	user16	ROLE_MEMBER
18	user17	ROLE_MEMBER
19	user18	ROLE_MEMBER
20	user19	ROLE_MEMBER
21	user20	ROLE_ADMIN
22	user21	ROLE_ADMIN
23	user22	ROLE_ADMIN
24	user23	ROLE_ADMIN

```

1 package org.ict.domain;
2
3 import java.sql.Date;
4 import java.util.List;
5
6 public class MemberVO {
7
8     private String userid;
9     private String userpw;
10    private String userName;
11    private boolean enabled;
12
13    private Date regDate;
14    private Date updateDate;
15    private List<AuthVO> authList;
16 }
17

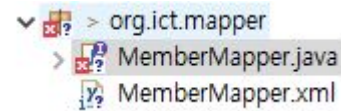
```

```

1 package org.ict.domain;
2
3 import lombok.Data;
4
5 @Data
6 public class AuthVO {
7
8     private String userid;
9     private String auth;
10 }
11

```

데이터 입력이 확인되었다면 이제 각 유저테이블에 맞는 VO를 만듭니다.
org.ict.domain을 src/java/main에 만들고, 거기에 MemberVO, AuthVO를 만들어줍니다.



```
1 package org.ict.mapper;
2
3 public interface MemberMapper {
4
5     public MemberVO read(String userid);
6 }
7
```

이제 처리된 Member관련 테이블을 Mybatis로 처리해보겠습니다.

org.ict.mapper 패키지 내에 MemberMapper 인터페이스와 xml을 같이 만들겠습니다.

```

> org.ict.mapper
> MemberMapper.java
> MemberMapper.xml
<mapper namespace="org.ict.mapper.MemberMapper">
  <resultMap type="org.ict.domain.MemberVO" id="memberMap">
    <!-- 조인 등으로 기존 VO와 구조가 다른 데이터가 리턴되는 경우
    resultMap으로 묶습니다. primary key를 id 태그에 넣고
    result의 property에는 VO의 변수명, column은 DB상의 컬럼명을 넣습니다.
    그러면 VO의 변수명과 DB상의 컬럼명이 달라도 매칭이 됩니다.-->
    <id property="userid" column="userid" />
    <result property="userid" column="userid" />
    <result property="userpw" column="userpw" />
    <result property="username" column="username" />
    <result property="regDate" column="regdate" />
    <result property="updateDate" column="updatedate" />
    <collection property="authList" resultMap="authMap">
    </collection>
  </resultMap>

  <resultMap type="org.ict.domain.AuthVO" id="authMap">
    <result property="userid" column="userid" />
    <result property="auth" column="auth" />
  </resultMap>

  <select id="read" resultMap="memberMap">
    SELECT
      m.userid, userpw, username, enabled, regdate, updatedate, a.auth
    FROM
      member_tbl m LEFT OUTER JOIN member_auth a on m.userid = a.userid
    WHERE m.userid = #{userid}
  </select>
</mapper>

```

ResultMap은 기존 컬럼명과 VO의 변수명이 다르거나 혹은 구조가 달라질 때 (지금같은경우 List<AuthVO>는 두 개 이상이 들어올 수 있습니다.) 처리할 구조에 대해 미리 설정해두는 태그입니다.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration({
    "file:src/main/webapp/WEB-INF/spring/root-context.xml"
})
@Log4j
public class MemberMapperTest {

    @Autowired
    private MemberMapper mapper;

    @Test
    public void testRead() {
        MemberVO vo = mapper.read("user25");

        Log.info(vo);

        vo.getAuthList();
    }
}
```

이제 **MemberMapper**를 계정명을 집어넣으면 계정 관련 정보가 정확하게 나오는지 테스트코드로 확인해보겠습니다.

위와같이 작성한 후 실행해주세요.



userid	userpw	username	enabled
user25	\$2a\$10\$0h0JbTQhiZ5KqBqEaJs5T.iI2TNe0fcSLRcphUmAmGipdGIqEOccS	운영자25	[unread]

```
regDate=2021-10-03, updateDate=2021-10-03, authList=[AuthVO(userid=user25, auth=ROLE_ADMIN)])
```

콘솔창에 위와 같이 뜨면 성공이며, **authList**라는 어레이리스트형에 제대로 들어갔는지 여부가 중요합니다.

authList에 자료가 안 들어갔다면 **MemberMapper.xml**에서 **resultMap**과 **collection**태그를 다시 손봅니다.


```
1 package org.ict.security;
2
3 import org.ict.mapper.MemberMapper;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.security.core.userdetails.UserDetails;
6 import org.springframework.security.core.userdetails.UserDetailsService;
7 import org.springframework.security.core.userdetails.UsernameNotFoundException;
8
9 import lombok.extern.log4j.Log4j;
10
11 @Log4j
12 public class CustomUserDetailsService implements UserDetailsService {
13
14     @Autowired
15     private MemberMapper mapper;
16
17     @Override
18     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
19
20         log.warn("유저 이름 확인: " + username);
21         return null;
22     }
23
24 }
```

이제 받아온 MemberMapper의 데이터를 스프링 시큐리티에서 처리할 수 있도록 커스터마이징해보겠습니다.

org.ict.security에 CustomUserDetailService를 만들고
UserDetailService를 implements해줍니다.

```
<bean id="bcryptPasswordEncoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
<bean id="customUserDetailsService" class="org.ict.security.CustomUserDetailsService" />
```

```
<security:http>
```

추가

```
<security:authentication-manager>
```

```
<security:authentication-provider
  user-service-ref="customUserDetailsService">
```

추가

```
<!-- <security:jdbc-user-service data-source-ref="dataSource"/> -->
```

```
<security:password-encoder ref="bcryptPasswordEncoder"/>
```

삭제

```
</security:authentication-provider>
```

```
</security:authentication-manager>
```

security-context.xml에는 CustomUserDetailsService를 빈 컨테이너에 넣어주고, 그걸 이용해 로그인하도록 바꿔줍니다.

```
INFO : org.ict.controller.CommonController - error 여부 :  
INFO : org.ict.controller.CommonController - logout 여부 : null  
WARN : org.ict.security.CustomUserDetailsService - 유저 이름 확인: user25  
ERROR: org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter - An  
org.springframework.security.authentication.InternalAuthenticationServiceException: UserDetailsSe  
at org.springframework.security.authentication.dao.DaoAuthenticationProvider.retrieveUser
```

```
@Log4j  
public class CustomUserDetailsService implement  
  
    @Autowired  
    private MemberMapper mapper;  
  
    @Override  
    public UserDetails loadUserByUsername(Strin  
        Log.warn("유저 이름 확인: " + username);  
        return null;  
    }
```

위와 같이 `InternalAuthenticationServiceException`이 나온다면 제대로 작동한 것입니다.

저렇게 나오는 이유는 `UserDetails`를 리턴하기로 해놓고 `null`을 리턴하고 있어서 입니다.

```
package org.ict.domain;

import java.util.Collection;
import java.util.stream.Collectors;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;

import lombok.Getter;
```

```
@Getter
public class CustomUser extends User {

    private static final long serialVersionUID = 1L;

    private MemberVO member;

    public CustomUser(String username, String password,
        Collection<? extends GrantedAuthority> auth) {
        super(username, password, auth);
    }

    public CustomUser(MemberVO vo) {
        super(vo.getUserid(), vo.getUserpw(),
            vo.getAuthList().stream().map(author ->
                new SimpleGrantedAuthority(author.getAuth()))
                .collect(Collectors.toList()));
        this.member = vo;
    }
}
```

그럼 이제 로그인 로직에서 MemberVO를 UserDetails로 바꾸도록 처리하겠습니다.

위와 같이 User를 상속한 CustomUser를 먼저 만들겠습니다.


```
@Log4j
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    private MemberMapper mapper;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        Log.warn("유저 이름 확인: " + username);

        MemberVO vo = mapper.read(username);

        Log.info("확인된 유저이름으로 얻어온 정보 : " + vo);

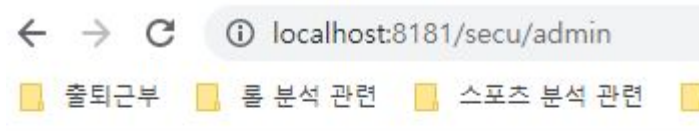
        return vo == null? null : new CustomUser(vo);
    }
}
```

```
* in-memory and reusing them. If so, make sure you return a copy of
* {@code UserDetailsService} each time it is invoked.
*
* @author Ben Alex
* @author Luke Taylor
*/
public class User implements UserDetails, CredentialsContainer {

    private static final long serialVersionUID = SpringSecurityCore
    private static final Log logger = LoggerFactory.getLog(User.class)

    // ~ Instance fields
```

이제 MemberVO가 입력되면 CustomUser의 생성자를 이용해 vo를 변환할 수 있고, CustomUser클래스는 User를 상속받았는데, User는 다시 스프링 시큐리티에서 사용하는 UserDetails를 상속받은 상태입니다. 따라서 MemberVO를 스프링 시큐리티에서 쓸 수 있게 수정한 것입니다.



admin 주소

[로그아웃페이지 이동](#)

```
INFO : org.ict.controller.HomeController - Welcome home! The client locale is ko_KR.  
WARN : org.ict.security.CustomUserDetailsService - 유저 이름 확인: user25  
WARN : org.ict.security.CustomLoginSuccessHandler - 로그인 성공  
WARN : org.ict.security.CustomLoginSuccessHandler - 부여받은 권한들 : [ROLE_ADMIN]  
INFO : org.ict.controller.SecurityController - 운영자만 접속 가능한 admin 로직
```

로그인이 되는지 확인이 되면 커스텀 DB작업이 끝납니다.