

9장. 스프링 Web Security



Security - login



◆ 스프링 시큐리티

일반적인 웹(JSP)에서는 세션이나 쿠키등을 이용하여 로그인을 하는데, 스프링은 인터셉터(Interceptor)등을 이용한다.

기본 동작 방식은 서블릿의 여러 종류의 필터와 인터셉터를 이용해서 처리된다.

필터는 서블릿의 필터를 의미하고, 인터셉터(Interceptor)는 스프링에서 필터와 유사한 역할을 한다.

인터셉터는 스프링의 빈으로 관리되면서 스프링의 컨텍스트 내에 속한다.



Spring Web Security

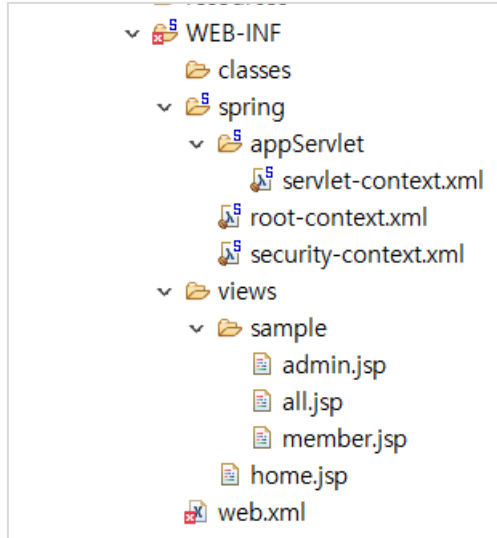
◆ 스프링 AOP pom.xml 에 security 관련 라이브러리 추가

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-core</artifactId>
  <version>5.2.2.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>5.2.2.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>5.2.2.RELEASE</version>
</dependency>
```



Spring Web Security

◆ security-context.xml 생성



Configuration Bean File로 생성

```
<?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
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans"
  >

</beans>
```



Spring Web Security

◆ web.xml에 등록

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/root-context.xml</param-value>
  <param-value>/WEB-INF/spring/security-context.xml</param-value>
</context-param>
```

```
<!-- Spring Security -->
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</
</filter>

<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```



Spring Web Security

◆ security-context.xml 에 등록 및 서버 실행

```
<!-- 기본 서비스 -->
<security:http>

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

<!-- 인증 및 권한 설정 -->
<security:authentication-manager>

</security:authentication-manager>
```

```
INFO: Initializing Spring DispatcherServlet 'appServlet'
INFO : org.springframework.web.servlet.DispatcherServlet - Initializing
INFO : org.springframework.web.servlet.DispatcherServlet - Completed ini
8월 27, 2022 8:08:01 오전 org.apache.coyote.AbstractProtocol start
INFO: 프로토콜 핸들러 ["http-nio-8080"]을(를) 시작합니다.
8월 27, 2022 8:08:01 오전 org.apache.catalina.startup.Catalina start
INFO: Server startup in 4023 ms
INFO : com.spring.controller.HomeController - Welcome home! The client 1
```



Spring Web Security

◆ 시큐리티 권한에 따른 URI 설계

- /sample/all -> 로그인을 하지 않은 사용자도 접근 가능
- /sample/member -> 로그인한 사용자만 접근 가능
- /sample/admin -> 로그인한 사용자들 중에서 관리자 권한을 가진 사용자만 접근 가능



Spring Web Security

◆ 시큐리티 권한에 따른 URI 설계

```
@Log4j
@RequestMapping("/sample/*")
@Controller
public class SampleController {

    @GetMapping("/all") //localhost:8080/sample/all
    public void doAll() {
        Log.info("do all can access everybody");
    }

    @GetMapping("/member")
    public void doMember() {
        Log.info("logged member");
    }

    @GetMapping("/admin")
    public void doAdmin() {
        Log.info("admin only");
    }
}
```

```
src/main/java
├── com.spring.board
├── com.spring.board.impl
├── com.spring.common
├── com.spring.controller
│   ├── HomeController.java
│   └── SampleController.java
└── com.spring.user
```



Spring Web Security

◆ 시큐리티 권한에 따른 URI 설계

```
<title>인증</title>
</head>
<body>
  <h1>/sample/all page</h1>
</body>
</html>
```

```
▼ views
  ▼ sample
    admin.jsp
    all.jsp
    member.jsp
    home.jsp
```

```
<title>인증</title>
</head>
<body>
  <h1>/sample/member page</h1>
</body>
</html>
```

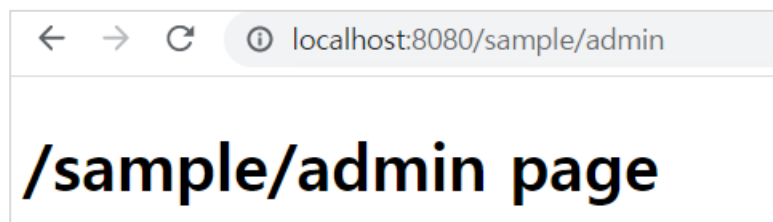
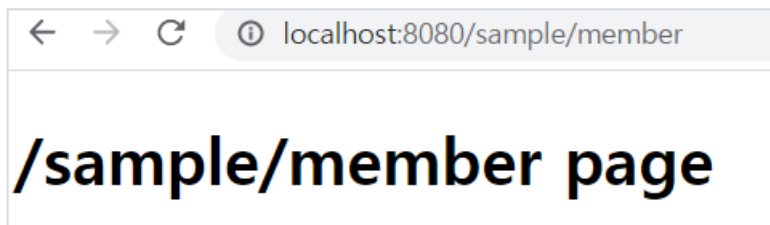
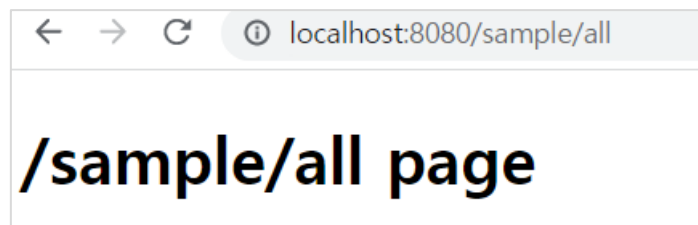
```
<body>
  <h1>/sample/admin page</h1>

  <a href="/customLogout">Logout</a>
</body>
```



Spring Web Security

◆ 시큐리티 권한에 따른 URI 설계



Spring Web Security

◆ 로그인과 로그아웃

권한 설정 - security-context.xml

```
<!-- URI 경로 접근 -->
<security:http>
    <security:intercept-url pattern="/sample/all" access="permitAll" />
    <security:intercept-url pattern="/sample/member" access="hasRole('ROLE_MEMBER')" />

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

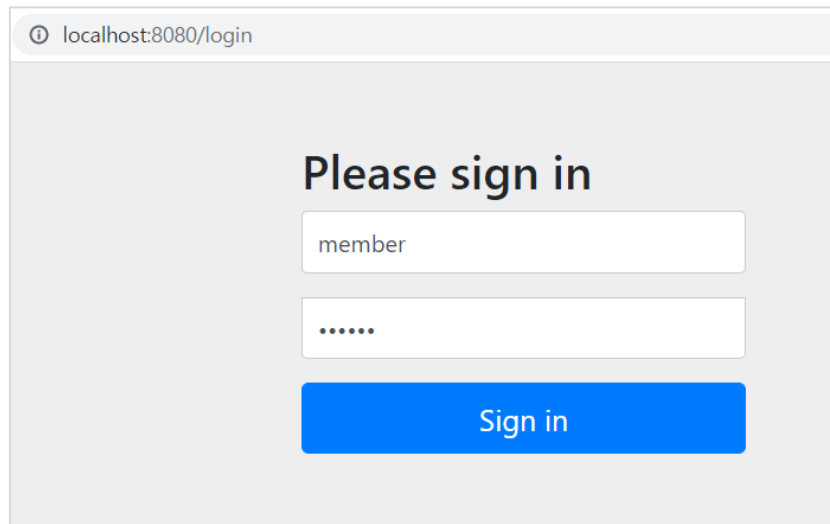
<!-- 인증 - 권한 -->
<security:authentication-manager>
    <security:authentication-provider>
        <security:user-service>
            <security:user name="member" password="member" authorities="ROLE_MEMBER"/>
        </security:user-service>
    </security:authentication-provider>
</security:authentication-manager>
```



Spring Web Security

◆ 로그인과 로그아웃

- /sample/member로 요청(username- member, password-member)



localhost:8080/login

Please sign in

member

.....

Sign in

PasswordEncoder 클래스가 없어서 에러발생

```
SEVERE: 경로가 []인 컨텍스트의 서블릿 [appServlet]을(를) 위한 Servlet.service() 호출이 예외를 발생.  
java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "nu  
    at org.springframework.security.crypto.password.DelegatingPasswordEncoder$Unn  
    at org.springframework.security.crypto.password.DelegatingPasswordEncoder.mat
```



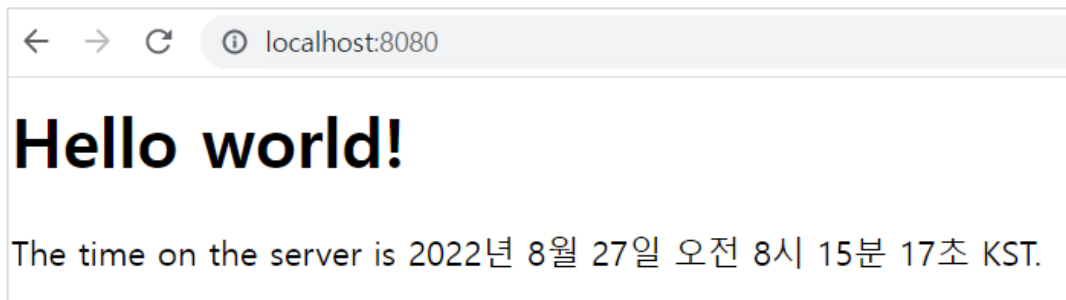
Spring Web Security

◆ 로그인과 로그아웃

- 비밀번호앞에 {noop}를 붙이면 에러 해결됨

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

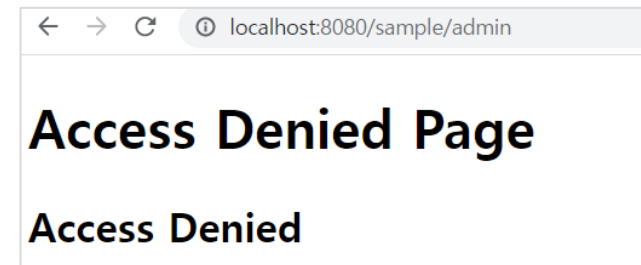
- 로그인후 '/' 경로로 이동함



Spring Web Security

◆ 로그인과 로그아웃

- admin 사용자는 /sample/member, /sample/admin 모두 접근 가능
- member 사용자는 /sample/member에만 접근 가능
- ✓ member 사용자가 admin에 접근하면 에러 발생



```
<security:form-login/>
<security:logout/>
<!-- 에러 처리 -->
<security:access-denied-handler error-page="/accessError" />
</security:http>
```



Spring Web Security

◆ 접근 에러 처리 1

```
@Log4j
@Controller
public class CommonController {

    @GetMapping("/accessError")
    public void accessDenied(Authentication auth, Model model) {
        log.info("access Denied : " + auth);
        model.addAttribute("msg", "Access Denied");
    }
}
```

```
<body>
    <h1>Access Denied Page</h1>

    <h2><c:out value="${msg}" /> </h2>
</body>
```

accessError.jsp



Spring Web Security

◆ 접근 에러 처리 2

- AccessDeniedHandler 인터페이스를 구현하는 경우

접근제한이 되었을 때 쿠키나 세션에 특정한 작업을 하거나 HttpServletResponse에 특정한 헤더 정보를 추가하는 등의 기능을 구현할 수 있음

```
@Log4j
public class CustomAccessDeniedHandler implements AccessDeniedHandler{

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
        AccessDeniedException accessDeniedException) throws IOException, ServletException {
        Log.error("AccessDeniedHandler");

        Log.error("Redirect.....");

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

- ▼ com.spring.controller
 - > CommonController.java
 - > HomeController.java
 - > SampleController.java
- ▼ com.spring.security
 - > CustomAccessDeniedHandler.java
 - > CustomLoginSuccessHandler.java



Spring Web Security

◆ 접근 에러 처리 2

- CustomAccessDeniedHandler를 빈으로 등록

```
<!-- CustomAccessDeniedHandler를 빈으로 등록 -->
<bean id="customAccessDenied" class="com.spring.security.CustomAccessDeniedHandler"/>

<!-- 기본 서비스 -->
<security:http>
    <security:intercept-url pattern="/sample/all" access="permitAll" />
    <security:intercept-url pattern="/sample/member" access="hasRole('ROLE_MEMBER')" />
    <security:intercept-url pattern="/sample/admin" access="hasRole('ROLE_ADMIN')" />

    <security:form-login/>

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

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

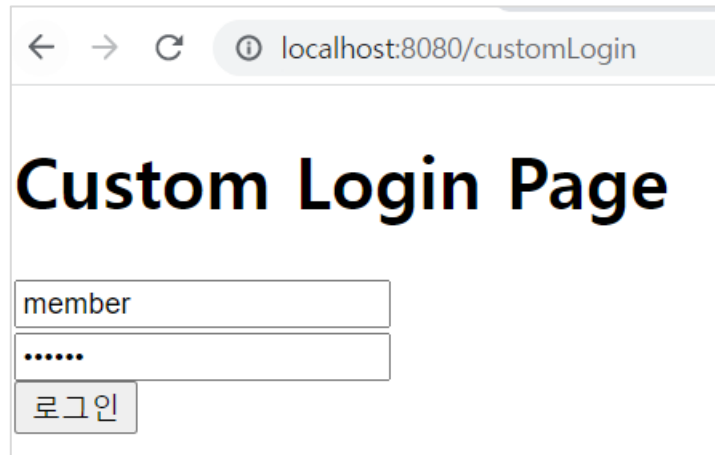


A yellow arrow points from the `ref="customAccessDenied"` attribute in the `<security:access-denied-handler>` tag to the `id="customAccessDenied"` attribute in the `<bean>` tag above. A red dashed box highlights the `ref="customAccessDenied"` attribute.

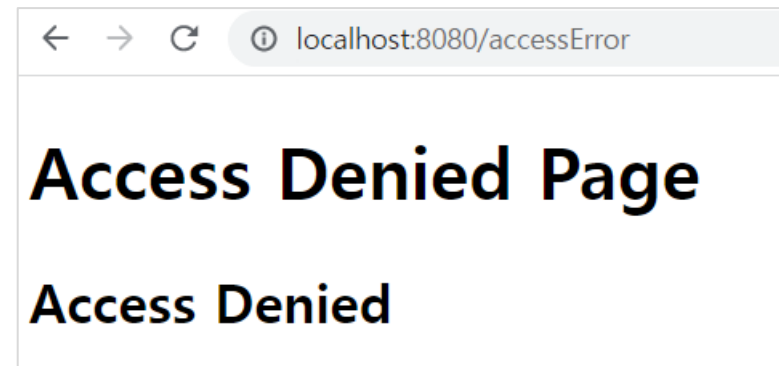


Spring Web Security

◆ 접근 에러 처리 2



A screenshot of a web browser window showing a custom login page. The address bar displays 'localhost:8080/customLogin'. The page title is 'Custom Login Page'. It features a form with two input fields: the first is labeled 'member' and contains the text 'member'; the second is a password field with masked characters '.....'. Below the password field is a button labeled '로그인' (Login).



A screenshot of a web browser window showing an access denied page. The address bar displays 'localhost:8080/accessError'. The page content consists of the text 'Access Denied Page' followed by 'Access Denied' on a new line.



Spring Web Security

◆ 커스텀 로그인 페이지

- 스프링에서 제공하는 로그인 폼을 사용하지 않고 직접 로그인 폼을 제작해서 사용하기

```
<security:form-login/>  
<!-- 로그인 페이지 직접 제작 -->  
<security:form-login login-page="/customLogin" />  
<security:logout logout-url="/customLogout" invalidate-session="true" />
```



Spring Web Security

◆ 로그인 페이지 만들기 – CommonController.java

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

    if(error != null) {
        model.addAttribute("error", "Login Error Check Your Account");
    }

    if(logout != null) {
        model.addAttribute("logout", "LogOut!!");
    }
}
```



Spring Web Security

◆ 로그인 페이지 만들기 – customLogin.jsp

```
<h1>Custom Login Page</h1>

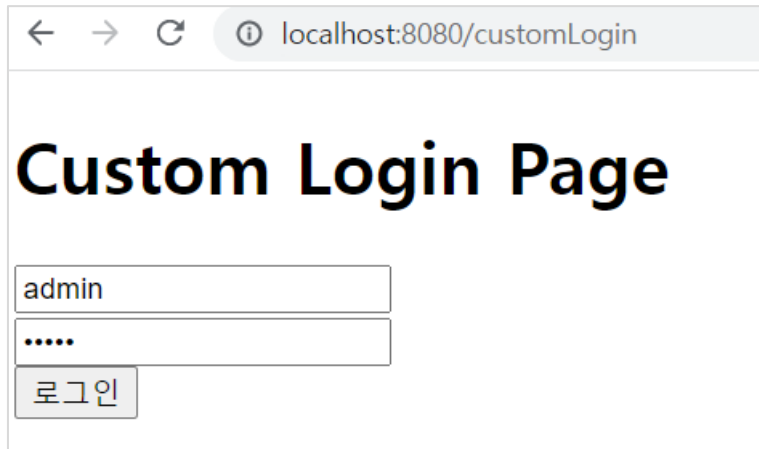
<h2><c:out value="${error}" /></h2>
<h2><c:out value="${logout}" /> </h2>

<form method="post" action="/login">
    <div>
        <input type="text" name="username" value="admin">
    </div>
    <div>
        <input type="password" name="password" value="admin">
    </div>
    <div>
        <input type="submit" value="로그인">
    </div>
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
</form>
```



Spring Web Security

◆ 로그인 - /sample/admin으로 요청



localhost:8080/customLogin

Custom Login Page

admin

.....

로그인

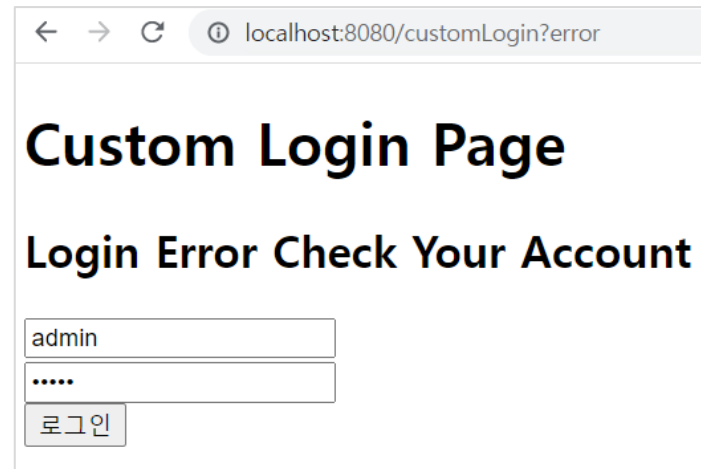


localhost:8080/sample/admin

/sample/admin page

[Logout](#)

username or password가
일치하지 않는 경우



localhost:8080/customLogin?error

Custom Login Page

Login Error Check Your Account

admin

.....

로그인



◆ CSRF(Cross-site request forgery) 공격과 토큰

스프링 시큐리티에서 POST 방식을 이용하는 경우 필수적으로 CSRF 토큰을 이용한다. CSRF 공격은 "사이트간 요청 위조"라고 번역될 수 있다.

서버에서 받아들이는 정보가 특별히 사전 조건을 검증하지 않는다는 단점을 이용하는 해킹 공격 방식이다.

실제로 2008년에 국내 인터넷 A 쇼핑몰이 이 기법으로 관리자 계정을 탈취당해 개인 정보들이 유출되었다.

CSRF를 통해 단순히 게시물의 조회수를 늘리는 등의 조작부터 피해자의 계저을 이용하는 다양한 공격이 가능하다.

● CSRF 토큰

사용자가 임의로 변하는 특정한 토큰값을 서버에서 체크하는 방식이다.

서버에는 브라우저에 데이터를 전송할 때 CSRF 토큰을 함께 전송하다.

사용자가 POST 방식을 특정한 작업을 할 때는 브라우저에서 전송된 CSRF 토큰의 값과 서버가 보관하고 있는 토큰의 값을 비교하여 만일, 토큰의 값이 다르다면 작업을 처리하지 않는 방식이다.



Spring Web Security

◆ 로그인 성공과 AuthenticationSuccessHandler

```
@Log4j
public class CustomLoginSuccessHandler implements AuthenticationSuccessHandler{

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse
        Authentication auth) throws IOException, ServletException {

        Log.warn("Login Success");

        List<String> roleNames = new ArrayList<>();

        auth.getAuthorities().forEach(authority -> { //람다식 반복문
            roleNames.add(authority.getAuthority()); //ROLE_ADMIN, ROLE_MEMBER 저장
        });
    }
}
```



Spring Web Security

◆ 로그인 성공과 AuthenticationSuccessHandler

```
Log.warn("ROLE NAMES: " + roleNames);

if(roleNames.contains("ROLE_ADMIN")) {
    response.sendRedirect("/sample/admin"); //admin.jsp로 이동
    return;
}

if(roleNames.contains("ROLE_MEMBER")) {
    response.sendRedirect("/sample/member"); //member.jsp로 이동
    return;
}

response.sendRedirect("/"); //index.jsp로 이동
}
```



Spring Web Security

◆ 로그인 성공과 AuthenticationSuccessHandler

```
<!-- CustomAccessDeniedHandler를 빈으로 등록 -->
<bean id="customAccessDenied" class="com.spring.security.CustomAccessDeniedHandler" />
<bean id="customLoginSuccess" class="com.spring.security.CustomLoginSuccessHandler" />

<!-- 기본 서비스 -->
<security:http>
    <security:intercept-url pattern="/sample/all" access="permitAll" />
    <security:intercept-url pattern="/sample/member" access="hasRole('ROLE_MEMBER')" />
    <security:intercept-url pattern="/sample/admin" access="hasRole('ROLE_ADMIN')" />

    <!-- <security:form-login/> -->
    <!-- 로그인 페이지 직접 제작 -->
    <security:form-login login-page="/customLogin"
                        authentication-success-handler-ref="customLoginSuccess" />
    <security:logout logout-url="/customLogout" invalidate-session="true" />
```



Spring Web Security

◆ /customLogin으로 직접 요청

← → ↻ ⓘ localhost:8080/customLogin

Custom Login Page

admin

.....

로그인



← → ↻ ⓘ localhost:8080/sample/admin

/sample/admin page

[Logout](#)

← → ↻ ⓘ localhost:8080/customLogin

Custom Login Page

member

.....

로그인



← → ↻ ⓘ localhost:8080/sample/member

/sample/member page

```
WARN : com.spring.security.CustomLoginSuccessHandler - Login Success
WARN : com.spring.security.CustomLoginSuccessHandler - ROLE NAMES: [ROLE_ADMIN, ROLE_MEMBER]
```



Spring Web Security

◆ 로그 아웃

```
<!-- <security:form-login/> -->
<!-- 로그인 페이지 직접 제작 -->
<security:form-login login-page="/customLogin"
                    authentication-success-handler-ref="customLoginSuccess" />

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

```
//로그아웃
@GetMapping("/customLogout")
public void logout() {
    Log.info("custom logout");
}
```

CustomLoginSuccessHandler.java



Spring Web Security

◆ 로그 아웃

```
<h1>Logout Page</h1>
```

```
<h2><c:out value="${error}" /></h2>
```

```
<h2><c:out value="${logout}" /> </h2>
```

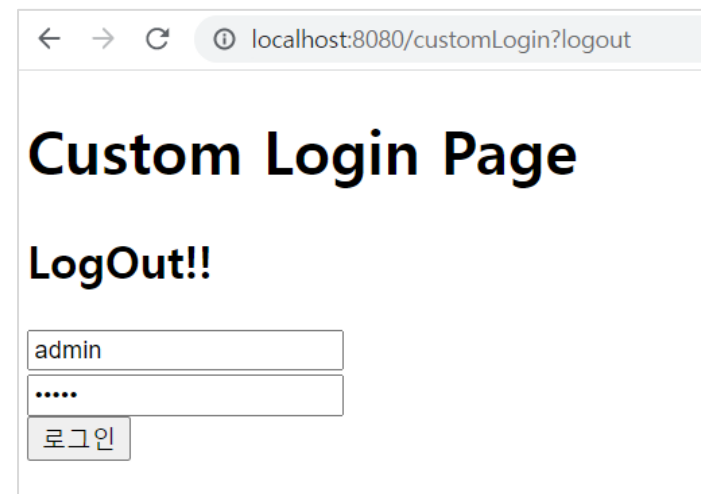
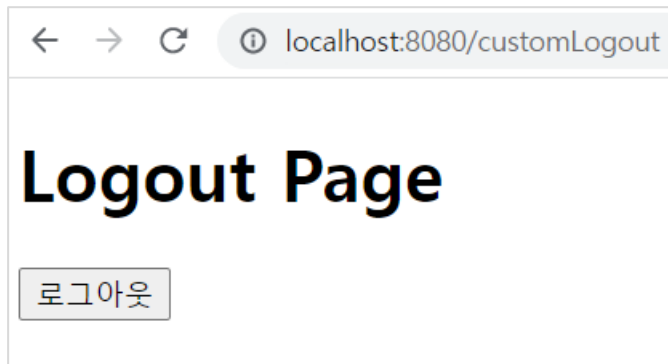
```
<form action="/customLogout" method="post" >
```

```
  <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
```

```
  <button type="submit">로그아웃</button>
```

```
</form>
```

logout.jsp



Spring Web Security

◆ JDBC를 이용하는 간편 인증/권한 처리

```
-- Spring security
CREATE TABLE users(
    username VARCHAR2(50) NOT NULL PRIMARY KEY,
    password VARCHAR2(50) 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)
);

-- index 생성
CREATE UNIQUE INDEX ix_auth_username ON authorities (username, authority);
```



Spring Web Security

◆ JDBC를 이용하는 간편 인증/권한 처리

```
-- users 추가
INSERT INTO users(username, password) VALUES ('user00', 'pw00');
INSERT INTO users(username, password) VALUES ('member00', 'pw00');
INSERT INTO users(username, password) VALUES ('admin00', 'pw00');

-- 권한 추가
INSERT INTO authorities(username, authority) VALUES ('user00', 'ROLE_USER');
INSERT INTO authorities(username, authority) VALUES ('member00', 'ROLE_MEMBER');
INSERT INTO authorities(username, authority) VALUES ('admin00', 'ROLE_MEMBER');
INSERT INTO authorities(username, authority) VALUES ('admin00', 'ROLE_ADMIN');
```



Spring Web Security

◆ JDBC를 이용하는 간편 인증/권한 처리

```
<!-- DataSource 설정 -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
    <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
    <property name="jdbcUrl" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="username" value="c##spring" />
    <property name="password" value="spring" />
</bean>

<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource"
    destroy-method="close">
    <constructor-arg ref="hikariConfig" />
```



Spring Web Security

◆ JDBC를 이용하는 간편 인증/권한 처리

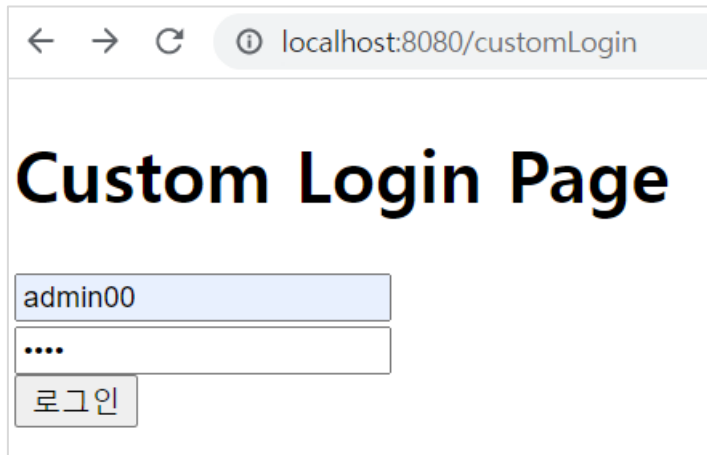
```
<!-- 인증 및 권한 설정 -->
<security:authentication-manager>
  <security:authentication-provider>
    <security:jdbc-user-service data-source-ref="dataSource" />
    <security:password-encoder ref="customPasswordEncoder" />

    <!-- <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>
```

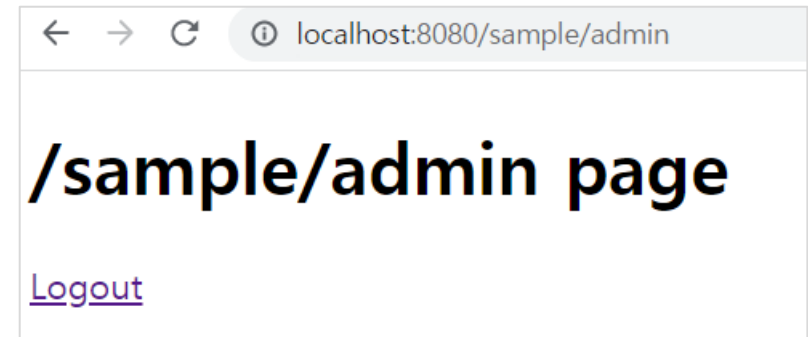


Spring Web Security

◆ JDBC를 이용하는 간편 인증/권한 처리



A screenshot of a web browser window showing a custom login page. The address bar displays 'localhost:8080/customLogin'. The page title is 'Custom Login Page'. Below the title, there is a text input field containing 'admin00', a password input field with four dots, and a button labeled '로그인' (Login).



A screenshot of a web browser window showing an admin page. The address bar displays 'localhost:8080/sample/admin'. The page content includes the text '/sample/admin page' and a purple link labeled 'Logout'.



Spring Web Security

◆ JDBC를 이용하는 간편 인증/권한 처리

```
@Log4j
public class CustomNoOpPasswordEncoder implements PasswordEncoder{

    @Override
    public String encode(CharSequence rawPassword) {
        Log.warn("before encode :" + rawPassword);
        return rawPassword.toString() ;
    }

    @Override
    public boolean matches(CharSequence rawPassword, String encodedPassword) {
        Log.warn("matches: " + rawPassword + ":" + encodedPassword);
        return rawPassword.toString().equals(encodedPassword);
    }
}
```



Spring Web Security

◆ 인증/권한을 위한 테이블 설계

```
-- Security 회원 테이블과 권한 테이블
CREATE TABLE tbl_member(
    userid VARCHAR2(50) PRIMARY KEY,
    userpw VARCHAR2(100) NOT NULL,
    username VARCHAR2(100) NOT NULL,
    regdate DATE DEFAULT SYSDATE,
    update date DATE DEFAULT SYSDATE,
    enabled CHAR(1) DEFAULT '1'
);

CREATE TABLE tbl_member_auth(
    auth VARCHAR2(50) NOT NULL,
    userid VARCHAR2(50) NOT NULL,
    CONSTRAINT fk_member_auth FOREIGN KEY (userid)
    REFERENCES tbl_member(userid)
);
```



Spring Web Security

◆ BcryptPasswordEncoder 빈 설정

```
<!-- 암호화 하지 않은 패스워드 -->
<!-- <bean id="customPasswordEncoder" class="com.spring.security.CustomNoOpPasswordEncoder" />
<!-- 암호화 한 패스워드 -->
<bean id="bcryptPasswordEncoder"
      class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder" />
```

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



Spring Web Security

◆ 사용자 입력 및 권한 주기

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

    @Autowired
    private DataSource ds;

    @Autowired
    private PasswordEncoder pwencoder;

    //회원 가입 - 100명
    @Test
    public void testInsertMember() {
        String sql = "INSERT INTO tbl_member(userid, userpw, username) VALUES(?,?,?)";

        for(int i=0; i<100; i++) {
            Connection conn = null;
            PreparedStatement pstmt = null;
```



Spring Web Security

```
try {
    conn = ds.getConnection();
    pstmt = conn.prepareStatement(sql);
    pstmt.setString(2, pwencoder.encode("pw" + i)); //비밀번호 암호화
    if(i < 80) { //0~79
        pstmt.setString(1, "user" + i);
        pstmt.setString(3, "일반사용자" + i);
    }else if(i < 90) { //80~89
        pstmt.setString(1, "member" + i);
        pstmt.setString(3, "회원" + i);
    }else { //90~100
        pstmt.setString(1, "admin" + i);
        pstmt.setString(3, "관리자" + i);
    }

    pstmt.executeUpdate(); //실행

} catch (Exception e) {
    e.printStackTrace();
} finally {
    if(pstmt != null) {
        try {
            pstmt.close();
        } catch (SQLException e) {
```



Spring Web Security

USERID	USERPW	USERNAME	REGDATE	UPDATEDATE	ENABLED
1 user0	\$2a\$10\$aWCBpmidANp99rz3Wkk9EuIouYGt0nGse5hoiGjCo6XCAP6x.LzB6	일반사용자0	22/08/27	22/08/27	1
2 user1	\$2a\$10\$GVYIJfc0r1bvAPPsKJbnQ./pafFX9V6b4f0BlCMKrZu9Zz72WqYZi	일반사용자1	22/08/27	22/08/27	1
3 user2	\$2a\$10\$26DeW8Pp6GelZKmMyCG1kOKgZC0oIYbzxhYdttdCzHVVbaDdXwJ3BG	일반사용자2	22/08/27	22/08/27	1
4 user3	\$2a\$10\$46z/EohunFxjwJLYaybPC.DAKlt7yBsRJ5iBkcRSWs9apfL/dNBdO	일반사용자3	22/08/27	22/08/27	1
5 user4	\$2a\$10\$w7/8YpeF2FluOKxYHWInKOBVk0Qos7cgc3OlFZvjB./ynvON.caOK	일반사용자4	22/08/27	22/08/27	1
6 user5	\$2a\$10\$40QfPWj9Ms2R9sgP3fchR.NK5Uskd5s.iF8x72TkqgOjDObf4RE1S	일반사용자5	22/08/27	22/08/27	1
7 user6	\$2a\$10\$PbWIAcnnC5SMzkfSCZpILu9yBjUPHQpDb7pxkJxK/MMM.gp/GvWAm	일반사용자6	22/08/27	22/08/27	1
8 user7	\$2a\$10\$dWJPulzT/cIHpPXkIQnLPOLQLE15hgWiCHxF/XZ1yiNATDB2JwBpu	일반사용자7	22/08/27	22/08/27	1
9 user8	\$2a\$10\$1tEX8xmlQrA1Rt/HI/TXfejxLY0jmU19mlhINaZyFP0qXNUIibyVK	일반사용자8	22/08/27	22/08/27	1
85 member84	\$2a\$10\$v.RI1D33lQ61o3igRCfNrug1LYTEYwTh/qCfaJWX11y9GWWT/XqRK	회원84	22/08/27	22/08/27	1
86 member85	\$2a\$10\$he9izvryBHDwoutmv8EFO6NAsmbE42yvQj02ka2emO.Uq0cSVSNU	회원85	22/08/27	22/08/27	1
87 member86	\$2a\$10\$UmuRA8oChiJVv/Hq9v38feogqw7gGfSvzbysWlTjXwMwyaXzW2L4y	회원86	22/08/27	22/08/27	1
88 member87	\$2a\$10\$CfTa2VOEghHg0AV4uMx5JermwBlPtjaY4dRe.al7.JZ1Jk04HelDa	회원87	22/08/27	22/08/27	1
89 member88	\$2a\$10\$N7oSoRU6gVY8AYzj41y84uoDtNKMh2.2E.Kbtg9zdPg/o/9YwRqUy	회원88	22/08/27	22/08/27	1
90 member89	\$2a\$10\$uBunowakGrLvXsgcfLpaEehquvdVgilSWsIhMRPpZnm9gWW0csNpS	회원89	22/08/27	22/08/27	1
91 admin90	\$2a\$10\$SeN/MdY1ISSJHJrENHN.KuwoyqjQHER7fotn3VH9aeW9fAJ438kaK	관리자90	22/08/27	22/08/27	1
92 admin91	\$2a\$10\$bd8X..Rk4.GNG09wTqPOu2NCPa4/lrnd6D56a6qe4YKDca8S8myq	관리자91	22/08/27	22/08/27	1
93 admin92	\$2a\$10\$N43m5raiMclIEzCnu9uVLeAb79CQ0NDczOTrpY/wSbRAAJLs94qtK	관리자92	22/08/27	22/08/27	1
94 admin93	\$2a\$10\$m/plHHAyup8QlWZQe2VTR.XQ95E/DM/L56oti8KvKLW62pEvHhqiG	관리자93	22/08/27	22/08/27	1
95 admin94	\$2a\$10\$BujmpDbEk8QPzNzzUt2OIOQjLnumRK8JhXMJk7IfhcfXrbTEV3xU6	관리자94	22/08/27	22/08/27	1
96 admin95	\$2a\$10\$rG5CaOqrY6AHBVB/ZbX6Zui/mWumCXEAHnrqSzd52rCLPJpkNeq.G	관리자95	22/08/27	22/08/27	1



Spring Web Security

```
//회원 권한 입력
@Test
public void testInsertAuth() {
    String sql = "INSERT INTO tbl_member_auth(userid, auth) VALUES(?,?)";

    for(int i=0; i<100; i++) {
        Connection conn = null;
        PreparedStatement pstmt = null;

        try {
            conn = ds.getConnection();
            pstmt = conn.prepareStatement(sql);

            if(i < 80) {
                pstmt.setString(1, "user"+i);
                pstmt.setString(2, "ROLE_USER");
            }else if(i < 90) {
                pstmt.setString(1, "member"+i);
                pstmt.setString(2, "ROLE_MEMBER");
            }else {
                pstmt.setString(1, "admin"+i);
                pstmt.setString(2, "ROLE_ADMIN");
            }

            pstmt.executeUpdate();
        }
    }
}
```



Spring Web Security

76	ROLE_USER	user75
77	ROLE_USER	user76
78	ROLE_USER	user77
79	ROLE_USER	user78
80	ROLE_USER	user79
81	ROLE_MEMBER	member80
82	ROLE_MEMBER	member81
83	ROLE_MEMBER	member82
84	ROLE_MEMBER	member83
85	ROLE_MEMBER	member84
86	ROLE_MEMBER	member85
87	ROLE_MEMBER	member86
88	ROLE_MEMBER	member87
89	ROLE_MEMBER	member88
90	ROLE_MEMBER	member89
91	ROLE_ADMIN	admin90
92	ROLE_ADMIN	admin91
93	ROLE_ADMIN	admin92
94	ROLE_ADMIN	admin93
95	ROLE_ADMIN	admin94



Spring Web Security

◆ 쿼리를 이용하는 인증

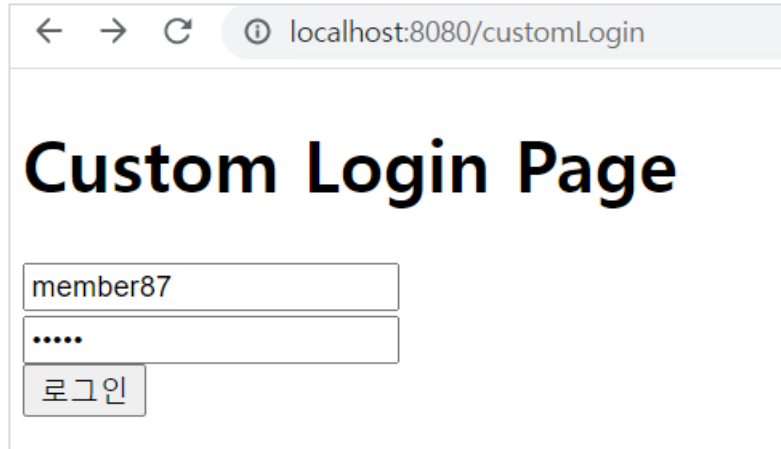
```
<security:authentication-provider>
  <!-- jdbc 연동 -->
  <!-- <security:jdbc-user-service data-source-ref="dataSource" /> -->

  <!-- 비밀번호 암호화 쿼리 사용 -->
  <security:jdbc-user-service
    data-source-ref="dataSource"
    users-by-username-query="select userid, userpw, enabled from
                           tbl_member where userid=?"
    authorities-by-username-query="select userid, auth from
                                   tbl_member_auth where userid=?" />
```

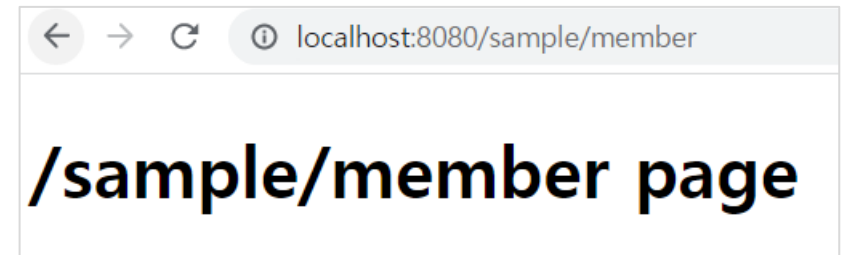


Spring Web Security

◆ Member87/pw87로 인증



A screenshot of a web browser window showing a custom login page. The address bar displays 'localhost:8080/customLogin'. The page title is 'Custom Login Page'. It features a form with two input fields: the first contains 'member87' and the second contains masked characters '.....'. Below the fields is a button labeled '로그인' (Login).



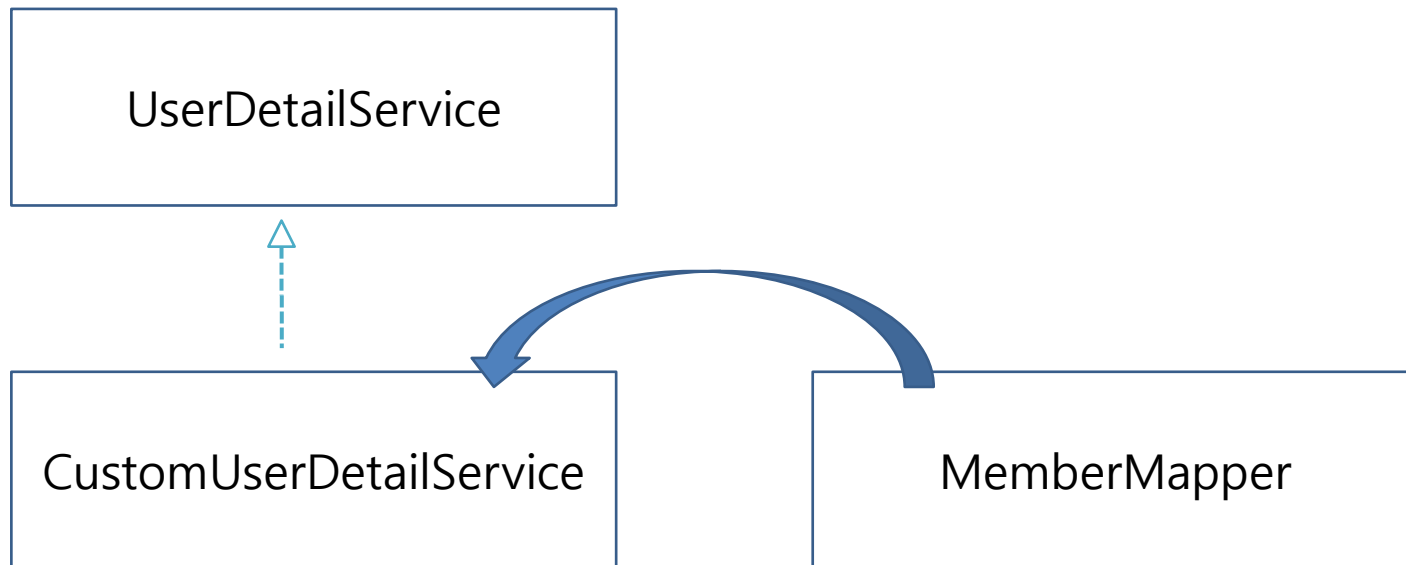
A screenshot of a web browser window showing the member page. The address bar displays 'localhost:8080/sample/member'. The page content is '/sample/member page'.



Spring Web Security

◆ 커스텀 UserDetailsService 활용

CustomUserDetailsService는 스프링 시큐리티의 UserDetails를 구현하고, MemberMapper 타입의 인스턴스를 주입받아서 실제 기능을 구현한다.



Spring Web Security

◆ 회원 도메인, 회원 Mapper 설계

```
▼ com.cloud.domain
  > AuthVO.java
  > BoardVO.java
  > MemberVO.java
  > UserVO.java
```

```
@Data
public class AuthVO {

    private String userid;
    private String auth;
}
```

```
@Data
public class MemberVO {
    private String userid;
    private String userpw;
    private String userName;
    private String enabled;

    private Date regDate;
    private Date updateDate;
    private List<AuthVO> authList;
}
```



Spring Web Security

◆ 회원 도메인, 회원 Mapper 설계

```
package com.cloud.mapper;  
  
import com.cloud.domain.MemberVO;  
  
public interface MemberMapper {  
    public MemberVO read(String userid);  
}
```

MemberMapper.java



Spring Web Security

```
<mapper namespace="com.cloud.mapper.MemberMapper">
  <resultMap type="com.cloud.domain.MemberVO" id="memberMap">
    <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="com.cloud.domain.AuthVO" id="authMap">
      <result property="userid" column="userid"/>
      <result property="auth" column="auth"/>
    </resultMap>

    <select id="read" resultMap="memberMap">
      SELECT
        mem.userid, userpw, username, enabled, regdate, updatedate, auth
      FROM
        tbl_member mem LEFT OUTER JOIN tbl_member_auth auth on mem.userid = auth.userid
      WHERE mem.userid = #{userid}
    </select>
```

MemberMapper.xml



Spring Web Security

◆ MemberMapper 테스트

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

    @Autowired
    private MemberMapper mapper;

    @Test
    public void testRead() {
        MemberVO vo = mapper.read("admin93");
        Log.info(vo);

        vo.getAuthList().forEach(authVO -> Log.info(authVO));
    }
}
```

```
INFO : com.cloud.security.MemberMapperTests - MemberVO(userid=admin93, userpw=$2a$10$ir6Pg9DNfa4e7gj/E
INFO : com.cloud.security.MemberMapperTests - AuthVO(userid=admin93, auth=ROLE_ADMIN)
```



Spring Web Security

◆ CustomUserDetailsService 구성

```

v [icon] > com.cloud.security.domain
  > [icon] CustomUser.java
  > [icon] com.cloud.service
v [icon] > com.spring.security
  > [icon] CustomAccessDeniedHandler.java
  > [icon] CustomLoginSuccessHandler.java
  > [icon] CustomNoOpPasswordEncoder.java
  > [icon] CustomUserDetailsService.java
```



Spring Web Security

◆ CustomUserDetailsService 구성

```
@Log4j
public class CustomUserDetailsService implements UserDetailsService{

    @Autowired
    private MemberMapper mapper;

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

        Log.warn("Load User By UserName : " + username);

        MemberVO vo = mapper.read(username);

        Log.warn("queried by member mapper: " + vo);

        return vo == null ? null : new CustomUser(vo); //vo가 있으면 CustomerUser
    }
}
```



Spring Web Security

◆ CustomUser 클래스

```
@Getter
public class CustomUser extends User{
    private static final long serialVersionUID = 11L;

    private MemberVO member;

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

    public CustomUser(MemberVO vo) {
        super(vo.getUserid(), vo.getUserpw(),
            vo.getAuthList()
                .stream()
                .map(auth -> new SimpleGrantedAuthority(auth.getAuth()))
                .collect(Collectors.toList()));

        this.member = vo;
    }
}
```



Spring Web Security

◆ CustomUserDetailsService 빈 등록 및 권한 변경

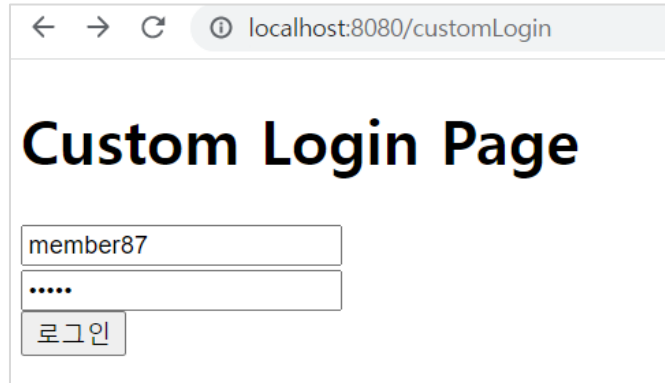
```
<bean id="customUserDetailsService"  
      class="com.spring.security.CustomUserDetailsService" />
```

```
<!-- 인증 - 권한 -->  
<security:authentication-manager>  
  <security:authentication-provider  
    user-service-ref="customUserDetailsService">  
    <!-- <security:jdbc-user-service data-source-ref="dataSource"/> -->  
  
    <!-- sql query를 이용한 인증 -->  
    <!-- <security:jdbc-user-service  
      data-source-ref="dataSource"  
      users-by-username-query="select userid, userpw, enabled from  
        tbl_member where userid=?"  
      authorities-by-username-query="select userid, auth from  
        tbl_member_auth where userid=?" /> -->  
  
    <!-- <security:password-encoder ref="customPasswordEncoder" /> -->  
    <!-- 패스워드 암호화 -->  
    <security:password-encoder ref="bcryptPasswordEncoder" />
```



Spring Web Security

◆ customLogin 페이지 인증 - 최종 완성



← → ↻ ⓘ localhost:8080/customLogin

Custom Login Page

member87

.....

로그인

```
security.CustomUserDetailsService - Load User By UserName : member81
security.CustomUserDetailsService - queried by member mapper: MemberVO(userid=member81, userpw=$2a$10$
security.CustomLoginSuccessHandler - Login Success
security.CustomLoginSuccessHandler - ROLE NAMES: [ROLE_MEMBER]
security.CustomUserDetailsService - Load User By UserName : admin93
security.CustomUserDetailsService - queried by member mapper: MemberVO(userid=admin93, userpw=$2a$10$
security.CustomLoginSuccessHandler - Login Success
security.CustomLoginSuccessHandler - ROLE NAMES: [ROLE_ADMIN]
```



Spring Web Security

◆ JSP에서 로그인한 사용자 정보 보여주기 – admin.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>인증</title>
</head>
<body>
    <h1>/sample/admin page</h1>

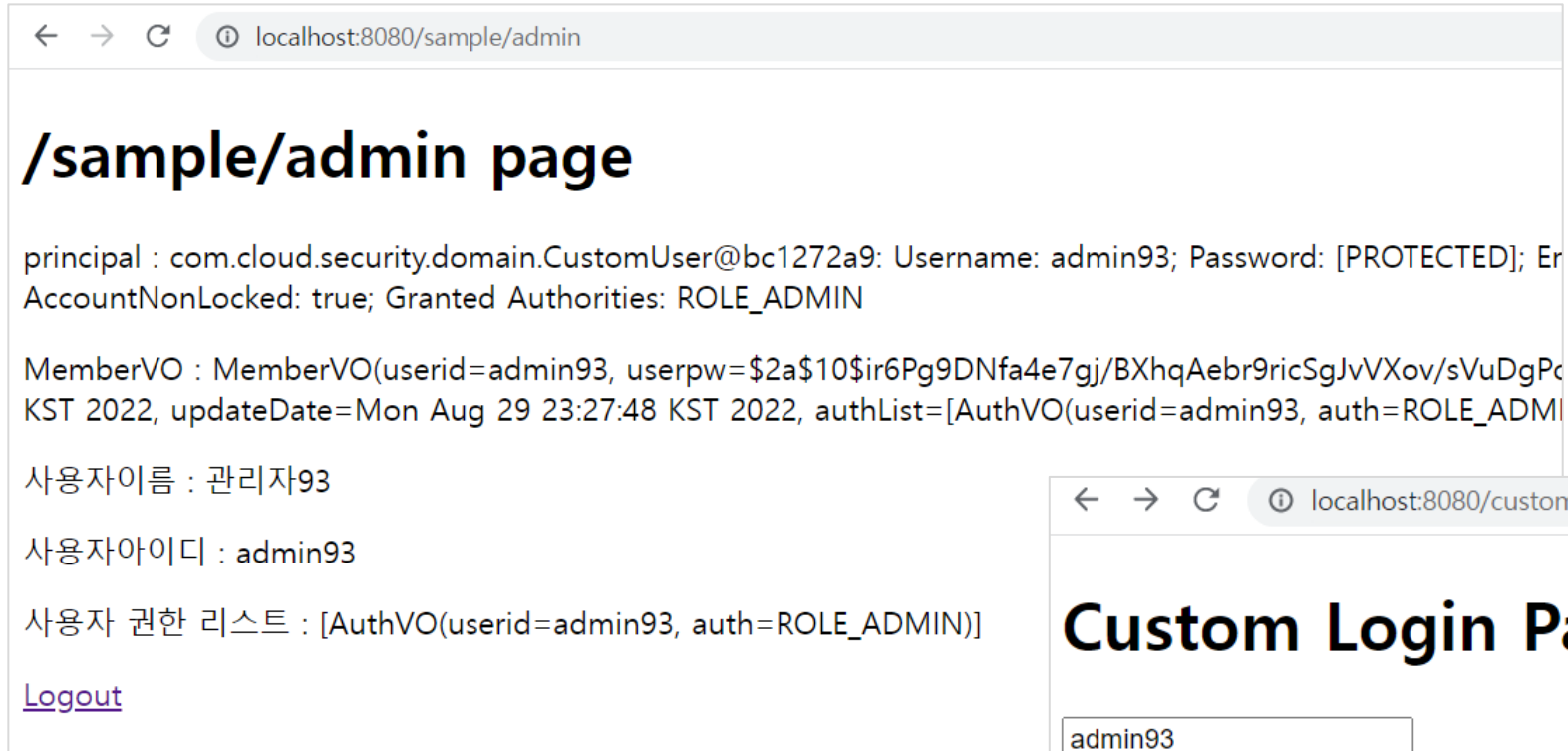
    <p>principal : <security:authentication property="principal"/></p>
    <p>MemberVO : <security:authentication property="principal.member"/></p>
    <p>사용자이름 : <security:authentication property="principal.member.userName"/>
    <p>사용자아이디 : <security:authentication property="principal.username"/></p>
    <p>사용자 권한 리스트 : <security:authentication property="principal.member.authList"/></p>

    <a href="/customLogout">Logout</a>
</body>
</html>
```



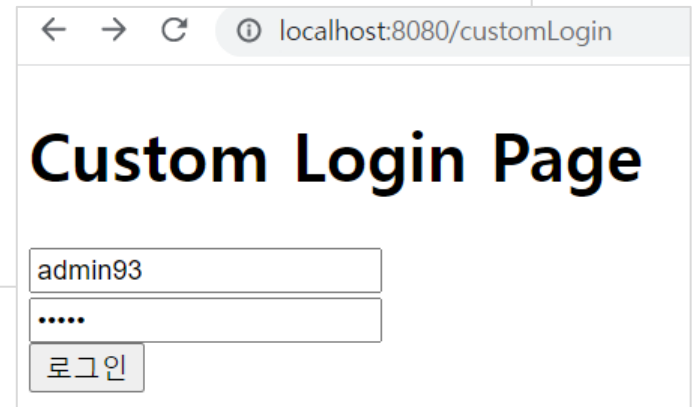
Spring Web Security

◆ JSP에서 로그인한 사용자 정보 보여주기



A screenshot of a web browser window showing the URL `localhost:8080/sample/admin`. The page title is `/sample/admin page`. The content displays user information for a logged-in user:

- principal : com.cloud.security.domain.CustomUser@bc1272a9: Username: admin93; Password: [PROTECTED]; Er
- AccountNonLocked: true; Granted Authorities: ROLE_ADMIN
- MemberVO : MemberVO(userid=admin93, userpw=\$2a\$10\$ir6Pg9DNfa4e7gj/BXhqAebr9ricSgJvVXov/sVuDgPc
- KST 2022, updateDate=Mon Aug 29 23:27:48 KST 2022, authList=[AuthVO(userid=admin93, auth=ROLE_ADMIN)
- 사용자이름 : 관리자93
- 사용자아이디 : admin93
- 사용자 권한 리스트 : [AuthVO(userid=admin93, auth=ROLE_ADMIN)]
- [Logout](#)



A screenshot of a web browser window showing the URL `localhost:8080/customLogin`. The page title is `Custom Login Page`. The form contains:

- A text input field containing `admin93`.
- A password input field with masked characters `.....`.
- A button labeled `로그인` (Login).



Spring Web Security

◆ 표현식을 이용하는 동적 화면 구성

표현식	설명
hasRole([role]) hasAuthority([authority])	해당 권한이 있으면 true
hasAnyRole([role1, role2]) hasAnyAuthority([authority])	여러 권한들 중에서 하나라도 해당하는 권한이 있으면 true
principal	현재 사용자의 정보를 의미
permitAll	모든 사용자에게 허용
denyAll	모든 사용자에게 거부
isAnonymous()	익명의 사용자의 경우(로그인을 하지 않은 경우도 해당)
isAuthenticated()	인증된 사용자만 true



Spring Web Security

◆ all.jsp

```
<h1>/sample/all page</h1>
<!-- 로그인하지 않은 사용자 -->
<security:authorize access="isAnonymous()">
    <a href="/customLogin">로그인</a>
</security:authorize>

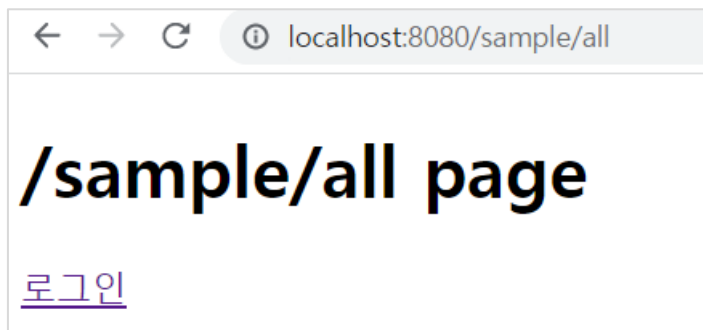
<!-- 로그인한 사용자 -->
<security:authorize access="isAuthenticated()">
    <a href="/customLogout">로그아웃</a>
</security:authorize>
```



Spring Web Security

◆ 권한 접근

로그인 하지 않은 사용자



로그인한 사용자



◆ 어노테이션을 이용하는 스프링 시큐리티 설정

@ Secured: 스프링 시큐리티 초기부터 사용되었고, ()안에 'ROLE_ADMIN'과 같은 문자열 혹은 문자열 배열을 이용한다.

@PreAuthorize, @PostAuthorize: 스프링 3버전 부터 지원되어 ()안에 표현식을 사용할 수 있으므로 최근에는 더 많이 사용된다.



Spring Web Security

◆ 어노테이션을 이용하는 스프링 시큐리티 설정

servlet-context.xml 설정

```
<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /W
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
</beans:bean>

<context:component-scan base-package="com.cCloud" />

<security:global-method-security
    pre-post-annotations="enabled" secured-annotations="enabled" />
```

