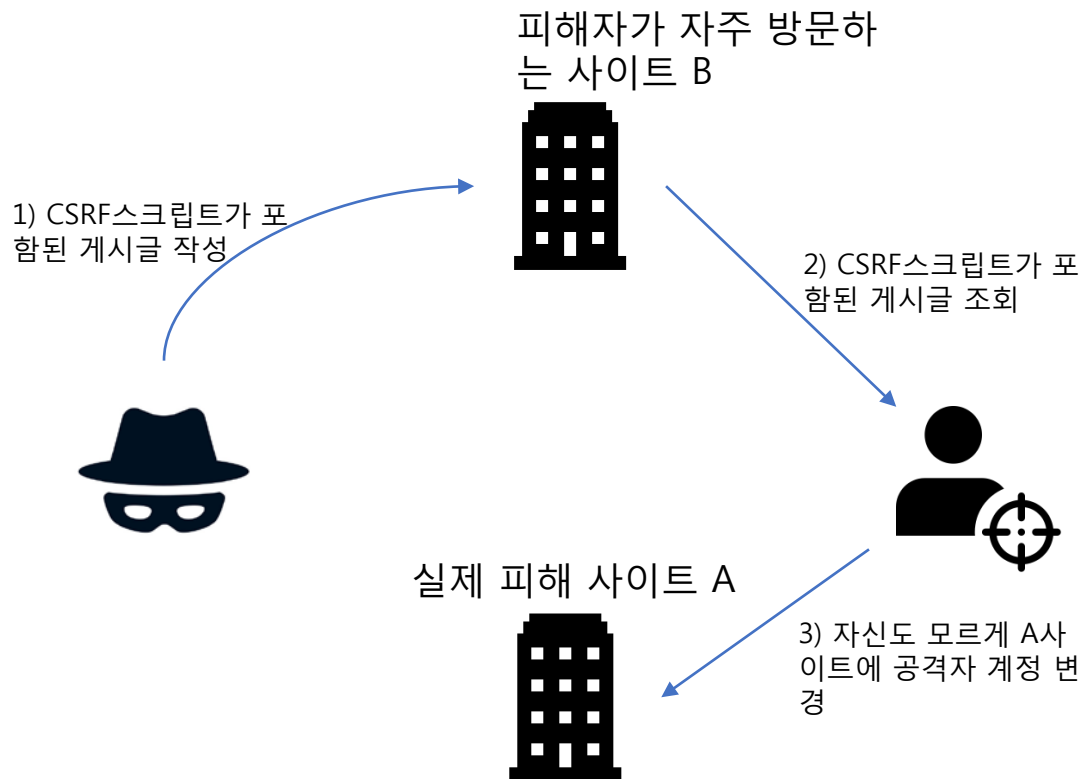


코드로 배우는 스프링 웹 프로젝트

• PART 7



- 사이트간 요청 위조(Cross-site request forgery) 공격
- 웹 기본적으로 출처를 따지지 않는다는 점을 이용



- 스프링 시큐리티는 기본적으로 GET방식을 제외하고 모든 요청에 CSRF토큰 사용
- <form> 등의 데이터 전송시에 CSRF토큰을 같이 전송하도록 처리

```
▼ <form method="post" action="/login">  
  ▶ <div>...</div>  
  ▶ <div>...</div>  
  ▶ <div>...</div>  
  <input type="hidden" name="_csrf" value="5a0ded0c-a151-4f6d-95f5-4c66664308d1">  
</form>
```

```
▼ <form method="post" action="/login">  
  ▶ <div>...</div>  
  ▶ <div>...</div>  
  ▶ <div>...</div>  
  <input type="hidden" name="_csrf" value="3b5b49da-11ba-44a4-a021-8117ebefd5b7">  
</form>
```

02 로그인 성공과 AuthenticationSuccessHandler

- 로그인 성공후 특정 URI로 이동하거나 쿠키 처리 등의 추가적인 작업

```
@Log4j
public class CustomLoginSuccessHandler implements AuthenticationSuccessHandler {

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

        Log.warn("Login Success");

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

        auth.getAuthorities().forEach(authority -> {
            roleNames.add(authority.getAuthority());
        });

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

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

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

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

```

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

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

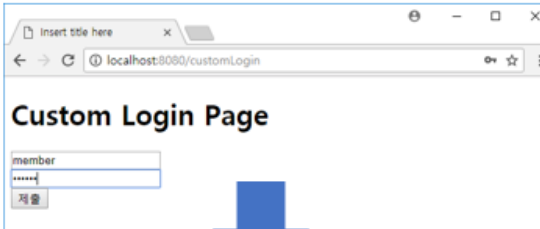
    <security:form-login login-page="/customLogin" authentication-success-handler-
ref="customLoginSuccess" />

    <!-- <security:csrf disabled="true"/> -->

</security:http>

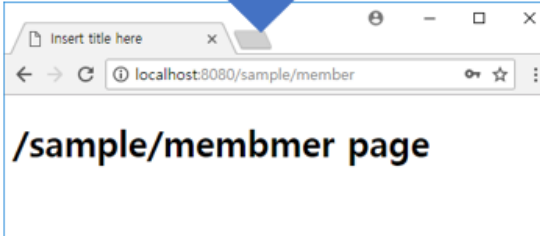
```

http://localhost:8080/customLogin



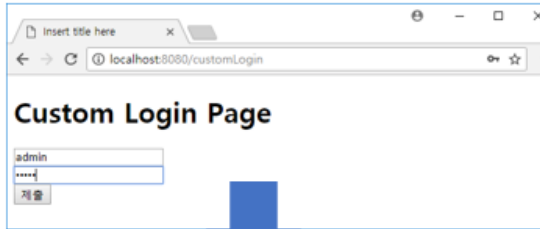
Custom Login Page

member



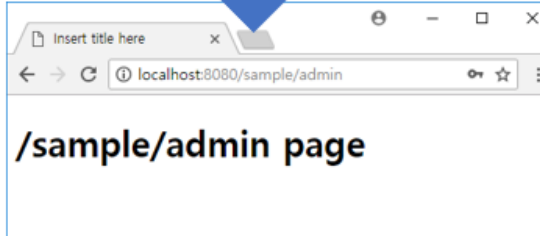
/sample/membmer page

http://localhost:8080/customLogin



Custom Login Page

admin



/sample/admin page

02 로그아웃의 처리와 LogoutSuccessHandler

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

```
<h1> Logout Page</h1>  
  
<form action="/customLogout" method='post'>  
<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>  
<button>로그아웃</button>  
</form>
```

02 JDBC를 이용하는 간편 인증/권한 처리

- 현실적으로 데이터베이스에 회원 정보를 이용해서 로그인 처리
- 패스워드는 PasswordEncoder를 지정해서 처리
- <https://github.com/spring-projects/spring-security/blob/master/core/src/main/java/org/springframework/security/provisioning/JdbcUserDetailsManager.java>

```
// GroupManager SQL
public static final String DEF_FIND_GROUPS_SQL = "select group_name from groups";
public static final String DEF_FIND_USERS_IN_GROUP_SQL = "select username from group_members gm, groups g "
    + "where gm.group_id = g.id" + " and g.group_name = ?";
public static final String DEF_INSERT_GROUP_SQL = "insert into groups (group_name) values (?)";
public static final String DEF_FIND_GROUP_ID_SQL = "select id from groups where group_name = ?";
public static final String DEF_INSERT_GROUP_AUTHORITY_SQL = "insert into group_authorities (group_id, authority) values (?,?)";
public static final String DEF_DELETE_GROUP_SQL = "delete from groups where id = ?";
public static final String DEF_DELETE_GROUP_AUTHORITIES_SQL = "delete from group_authorities where group_id = ?";
public static final String DEF_DELETE_GROUP_MEMBERS_SQL = "delete from group_members where group_id = ?";
public static final String DEF_RENAME_GROUP_SQL = "update groups set group_name = ? where group_name = ?";
public static final String DEF_INSERT_GROUP_MEMBER_SQL = "insert into group_members (group_id, username) values (?,?)";
public static final String DEF_DELETE_GROUP_MEMBER_SQL = "delete from group_members where group_id = ? and username = ?";
public static final String DEF_GROUP_AUTHORITIES_QUERY_SQL = "select g.id, g.group_name, ga.authority "
    + "from groups g, group_authorities ga "
    + "where g.group_name = ? "
    + "and g.id = ga.group_id ";
public static final String DEF_DELETE_GROUP_AUTHORITY_SQL = "delete from group_authorities where group_id = ? and authority = ?";
```

02 회원 테이블의 설계

```
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));  
  
create unique index ix_auth_username on authorities (username,authority);  
  
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;
```


02 Security-context 설정

```
<security:authentication-manager>

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

</security:authentication-manager>
```

02 PasswordEncoder의 설정

- 4버전까지는 위와 같이 별도의 PasswordEncoder를 이용하고 싶지 않을 때 NoOpPasswordEncoder를 이용해서 처리할 수 있었지만, 5버전부터는 Deprecated되어서 더 이상 사용할 수 없음으로 주의
- 암호화를 피하고 싶다면 직접 PasswordEncoder를 구현

```
<security:authentication-manager>

    <security:authentication-provider>

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

        <security:password-encoder
            ref="customPasswordEncoder" />

    </security:authentication-provider>

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

02 기존의 테이블을 이용하는 경우

```
create table tbl_member(  
    userid varchar2(50) not null primary key,  
    userpw varchar2(100) not null,  
    username varchar2(100) not null,  
    regdate date default sysdate,  
    updatedate date default sysdate,  
    enabled char(1) default '1');  
  
create table tbl_member_auth (  
    userid varchar2(50) not null,  
    auth varchar2(50) not null,  
    constraint fk_member_auth foreign key(userid) references tbl_member(userid)  
);
```

- bcrypt는 태생 자체가 패스워드를 저장하는 용도로 설계된 해시 함수로 특정 문자열을 암호화하고, 체크하는 쪽에서는 암호화된 패스워드가 가능한 패스워드인지만 확인

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

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

- 테스트 코드를 이용해서 패스워드를 인코딩한 후 insert

```
@Setter(onMethod_ = @Autowired)
private PasswordEncoder pwencoder;

@Setter(onMethod_ = @Autowired)
private DataSource ds;

...

pstmt.setString(2, pwencoder.encode("pw" + i));
```

02 쿼리를 이용하는 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" /> -->
```

```
<security:password-encoder
```

```
    ref="bcryptPasswordEncoder" />
```

Insert title here x
localhost:8080/customLogin

Custom Login Page

admin90

....

저출

```
INFO : jdbc.sqltiming - select userid , userpw , enabled from tbl_member where userid = 'admin90'
{executed in 1 msec}
INFO : jdbc.resultset - 6. ResultSet.new ResultSet returned
INFO : jdbc.audit - 6. PreparedStatement.executeQuery() returned net.sf.log4jdbc.sql.jdbcapi.ResultSetSpy@255b5041
INFO : jdbc.resultset - 6. ResultSet.next() returned true
INFO : jdbc.resultset - 6. ResultSet.getString(1) returned admin90
INFO : jdbc.resultset - 6. ResultSet.getString(2) returned $2a$10$M2vu0yzEysYp9N1VDcyNuM5h6i.1dfXrTZ.J6h9K1AHNDtiDwG1S
INFO : jdbc.resultset - 6. ResultSet.getBoolean(3) returned true
INFO : jdbc.resultsettable -
```

userid	userpw	enabled
admin90	\$2a\$10\$M2vu0yzEysYp9N1VDcyNuM5h6i.1dfXrTZ.J6h9K1AHNDtiDwG1S	true

```
INFO : jdbc.sqltiming - select userid, auth from tbl_member_auth where userid = 'admin90'
{executed in 0 msec}
INFO : jdbc.resultset - 6. ResultSet.new ResultSet returned
INFO : jdbc.audit - 6. PreparedStatement.executeQuery() returned net.sf.log4jdbc.sql.jdbcapi.ResultSetSpy@39706d6d
INFO : jdbc.resultset - 6. ResultSet.next() returned true
INFO : jdbc.resultset - 6. ResultSet.getString(2) returned ROLE_ADMIN
INFO : jdbc.resultsettable -
```

userid	auth
[unread]	ROLE_ADMIN

- 사용자가 원하는 방식으로 인가/인증 처리를 하기 위해서는 직접 UserDetailsService 인터페이스를 구현해서 처리

Method Summary	
All Methods	Instance Methods
Abstract Methods	
Modifier and Type	Method and Description
UserDetails	<code>loadUserByUsername(java.lang.String username)</code> Locates the user based on the username.

`loadUserByUsername()`이라는 메서드의 반환 타입인 `UserDetails` 역시 인터페이스로, 사용자의 정보와 권한 정보 등을 담는 타입

`UserDetails` 타입은 `getAuthorities()`, `getPassword()`, `getUserName()` 등의 여러 추상 메서드를 가지고 있어서, 개발 전에 이를 직접 구현할 것인지 `UserDetails` 인터페이스를 구현해둔 스프링 시큐리티의 여러 하위 클래스를 이용할 것인지 판단해야 함

02 회원 도메인, 회원 Mapper 설계

```
▼ src/main/java
  > org.zerock.controller
  ▼ org.zerock.domain
    > AuthVO.java
    > MemberVO.java
```

```
package org.zerock.domain;

import java.util.Date;
import java.util.List;

import lombok.Data;

@Data
public class MemberVO {

    private String userid;
    private String userpw;
    private String userName;
    private boolean enabled;

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

}
```

```
package org.zerock.domain;

import lombok.Data;

@Data
public class AuthVO {

    private String userid;
    private String auth;

}
```


02 MemberMapper

- MyBatis의 <resultMap>을 활용해 Join처리시에 발생하는 1:N 문제를 해결

```
<mapper namespace="org.zerock.mapper.MemberMapper">

  <resultMap type="org.zerock.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>

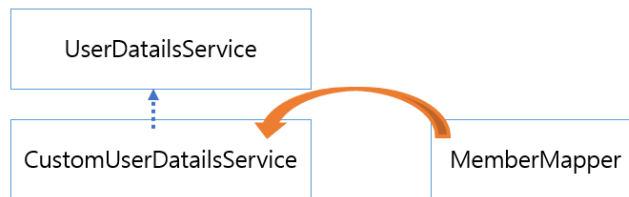
  <resultMap type="org.zerock.domain.AuthVO" id="authMap">
    <result property="userid" column="userid"/>
    <result property="auth" column="auth"/>
  </resultMap>

  <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>

</mapper>
```

02 CustomUserDetailsService 구성

- CustomUserDetailsService는 스프링 시큐리티의 UserDetailsService를 구현하고, MemberMapper 타입의 인스턴스를 주입 받아서 실제 기능을 구현



```
@Log4j
public class CustomUserDetailsService implements UserDetailsService {

    @Setter(onMethod_ = { @Autowired })
    private MemberMapper memberMapper;

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

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

        return null;
    }
}
```

02 security-context.xml의 수정

- 변경된 방식으로 로그인 처리를 하는지 우선적으로 확인

```
<security:authentication-manager>

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

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

    </security:authentication-provider>

</security:authentication-manager>
```

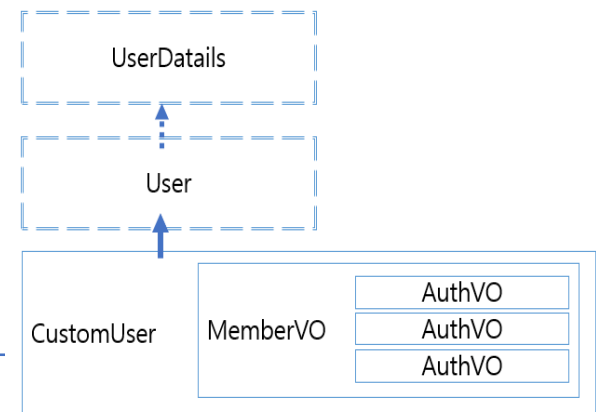
Custom Login Page


```
INFO : org.zerock.controller.CommonController - logout: null
WARN : org.zerock.security.CustomUserDetailsService - Load User By UserName : admin90
ERROR: org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter -
org.springframework.security.authentication.InternalAuthenticationServiceException: UserDetail
at org.springframework.security.authentication.dao.DaoAuthenticationProvider.retrieve
```

02 MemberVO를 UserDetails 타입으로 변환하기

- MemberVO에 UserDetails인터페이스를 추가하거나 확장하는 방식을 사용

```
public class CustomUser extends User {  
  
    private static final long serialVersionUID = 1L;  
  
    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;  
    }  
}
```



02
`@Log4j`

```
public class CustomUserDetailsService implements UserDetailsService {

    @Setter(onMethod_ = { @Autowired })
    private MemberMapper memberMapper;

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

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

        // userName means userid
        MemberVO vo = memberMapper.read(userName);

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

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

02 스프링 시큐리티를 JSP에서 활용하기

- JSP에서는 시큐리티 태그들을 이용해서 처리
- <sec:authentication> 태그와 principal이라는 이름의 속성을 사용

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

```
<p>principal : <sec:authentication property="principal"/></p>
```

```
<p>MemberVO : <sec:authentication property="principal.member"/></p>
```

```
<p>사용자이름 : <sec:authentication property="principal.member.userName"/></p>
```

```
<p>사용자아이디 : <sec:authentication property="principal.username"/></p>
```

```
<p>사용자 권한 리스트 : <sec:authentication property="principal.member.authList"/></p>
```

02 표현식을 이용하는 동적 화면 구성

표현식	설명
<code>hasRole([role])</code> <code>hasAuthority([authority])</code>	해당 권한이 있으면 true
<code>hasAnyRole([role,role2])</code> <code>hasAnyAuthority([authority])</code>	여러 권한들 중에서 하나라도 해당하는 권한이 있으면 true
<code>principal</code>	현재 사용자 정보를 의미
<code>permitAll</code>	모든 사용자에게 허용
<code>denyAll</code>	모든 사용자에게 거부
<code>isAnonymous()</code>	익명의 사용자의 경우(로그인을 하지 않은 경우도 해당)
<code>isAuthenticated()</code>	인증된 사용자면 true
<code>isFullyAuthenticated()</code>	Remember-me로 인증된 것이 아닌 인증된 사용자인 경우 true

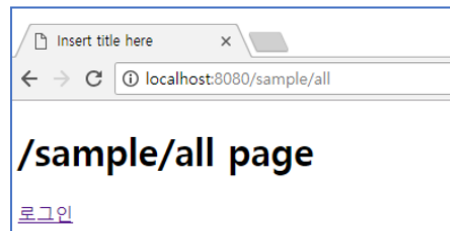
```

<sec:authorize access="isAnonymous()">
    <a href="/customLogin">로그인</a>
</sec:authorize>

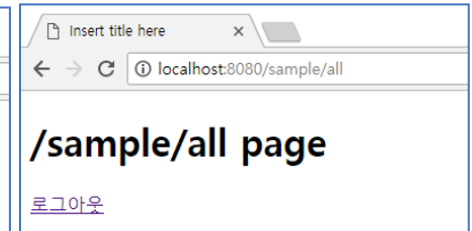
<sec:authorize access="isAuthenticated()">
    <a href="/customLogout">로그아웃</a>
</sec:authorize>

```

로그인하지 않은 사용자



로그인된 사용자



02 자동 로그인(remember-me)

- 스프링 시큐리티의 경우 'remember-me' 기능을 메모리상에서 처리하거나, 데이터베이스를 이용하는 형태로 약간의 설정만으로 구현이 가능
- security-context.xml에는 <security:remember-me> 태그를 이용해서 기능을 구현
- <security:remember-me>에는 아래와 같이 여러 속성
 - key: 쿠키에 사용되는 값을 암호화하기 위한 키(key)값
 - data-source-ref: DataSource를 지정하고 테이블을 이용해서 기존 로그인 정보를 기록(옵션)
 - remember-me-cookie: 브라우저에 보관되는 쿠키의 이름을 지정합니다. 기본값은 'remember-me'입니다.
 - remember-me-parameter: 웹 화면에서 로그인할 때 'remember-me'는 대부분 체크박스를 이용해서 처리합니다. 이 때 체크박스 태그의name속성을 의미합니다.
 - token-validity-seconds: 쿠키의 유효시간을 지정합니다.

02 데이터베이스를 이용하는 자동로그인

- 별도의 코드 생성없이 테이블 생성만으로도 처리 가능

```
create table persistent_logins (  
  username varchar(64) not null,  
  series varchar(64) primary key,  
  token varchar(64) not null,  
  last_used timestamp not null);
```

```
<security:remember-me
```

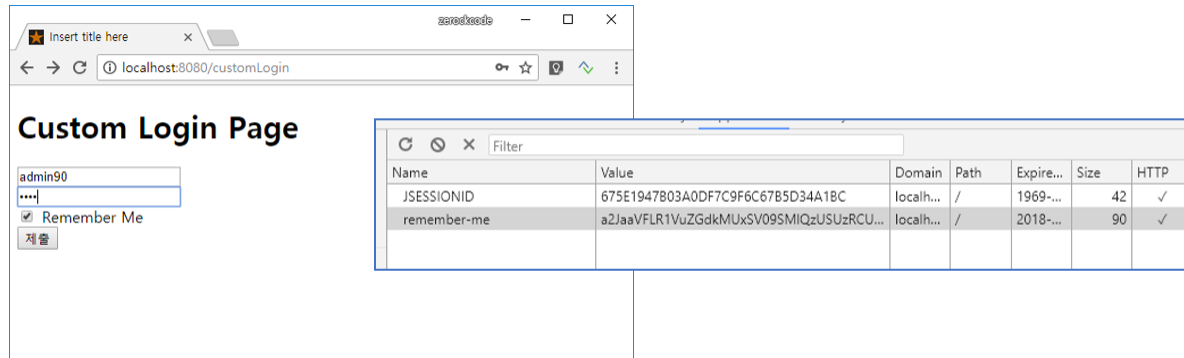
```
  data-source-ref="dataSource" token-validity-seconds="604800" />
```

02 로그인 화면에서 자동 로그인

```
<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='checkbox' name='remember-me'> Remember Me
  </div>

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



- 자동 로그인 기능을 이용하는 경우에 사용자가 로그아웃을 하면 기존과 달리 자동 로그인에 사용하는 쿠키도 삭제해 주도록 쿠키를 삭제하는 항목을 security-context.xml에 지정

```
<security:logout logout-url="/customLogout"
    invalidate-session="true" delete-cookies="remember-me, JSESSION_ID" />
```

Q & A

열공합시다!!

