

1. Spring Security

참고사이트:

<https://blog.naver.com/tmondev/220310743818>

<https://sjh836.tistory.com/165>

<https://to-dy.tistory.com/86>

<https://offbyone.tistory.com/88> - 실습참조

가. 인증, 인가

1) 인증(Authentication)

애플리케이션에 액세스 할 수 있는 사용자를 제한

2) 인가(authorization)

특정한 사용자에게 대해서 조회 가능한 정보와 실행 가능한 동작을 제한하는 기능. 인증 기능을 통해 사용자는 애플리케이션에 액세스하는 것은 인정되지만, 모든 사용자에게 동일한 권한이 주어진다고 볼 수는 없다.

3) 룰

어떤 사용자에게 어떤 권한을 부여할 것인지에 대한 정의.

리소스에 대한 참조 및 조작에 대한 권한

나. 주요특징

1) 인증.인가 공통 기반을 제공

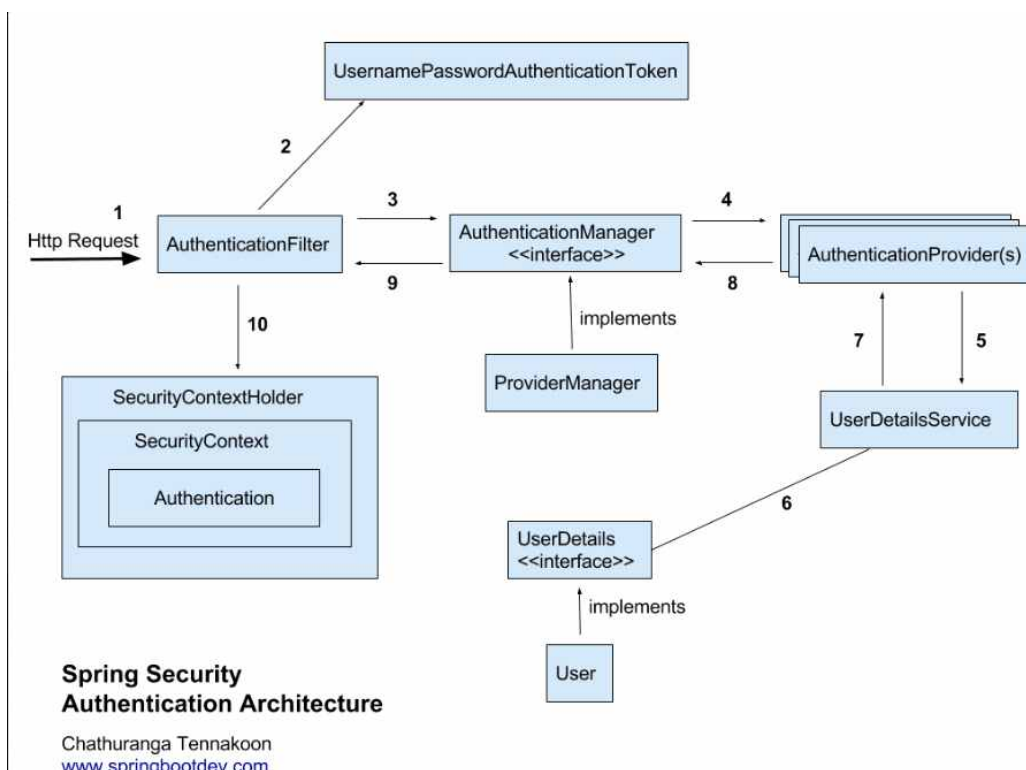
2) 웹 애플리케이션에서 인증, 인가를 구현하기 위한 각종 필터 클래스를 제공

3) Bean 정의 파일이나 프로퍼티파일, 데이터베이스, LDAP 등 여러 리소스로부터 인증, 인가 정보취득 가능

4) HTTP BASIC 인증이나 화면에서의 폼 인증 등, 웹 애플리케이션에서 일반적으로 채용되는 인증 지원

5) 인가 정보에 기반한 화면 표시 제어를 위한 JSP 태그 라이브러리를 제공

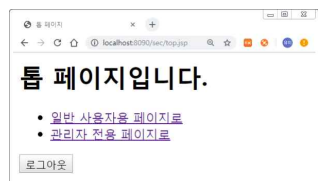
다. Architecture



라. 실습


1) 화면구성

- top.jsp

A screenshot of a web browser showing the 'top.jsp' page. The page title is '톱 페이지입니다.' (Top page). It contains a list with two items: '일반 사용자용 페이지로' (Go to general user page) and '관리자 전용 페이지로' (Go to admin page). There is a '로그아웃' (Logout) button.


```
<h1> 톱 페이지입니다.</h1>
<ul>
<li><a href= "user/user.jsp"> 일반 사용자용 페이지로</a></li>
<li><a href= "admin/admin.jsp"> 관리자 전용 페이지로</a></li>
</ul>
<form action= "logout" method= "post">
  <button>로그아웃</button>
</form>
```

- admin/admin.jsp

A screenshot of a web browser showing the 'admin/admin.jsp' page. The page title is '관리자 전용 페이지' (Admin page). It contains a message '관리자 전용 페이지입니다.' (This is an admin page) and a list with one item: '톱 페이지로' (Go to top page).

```
<h1> 관리자 전용 페이지</h1>
관리자 전용 페이지입니다.<br>
<ul>
  <li><a href= "../top.jsp"> 톱 페이지로</a></li>
</ul>
```

- user/user.jsp

A screenshot of a web browser showing the 'user/user.jsp' page. The page title is '일반 사용자용 페이지' (General user page). It contains a message '일반 사용자용 페이지입니다.' (This is a general user page) and a list with one item: '톱 페이지로' (Go to top page).

```
<h1> 일반 사용자용 페이지</h1>
일반 사용자용 페이지입니다.<br>
<ul>
  <li><a href= "../top.jsp"> 톱 페이지로</a></li>
</ul>
```

2) 라이브러리 설정(pom.xml)

```
<properties>
  <org.springframework-version>5.2.8.RELEASE</org.springframework-version>
  <org.springframework.security-version>5.2.5.RELEASE</org.springframework.security-version>
</properties>

<!-- Spring Security -->
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>${org.springframework.security-version}</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>${org.springframework.security-version}</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-taglibs</artifactId>
  <version>${org.springframework.security-version}</version>
</dependency>
```

3) 스프링 설정파일(security-context.xml)

▣ 메모리에서 인증정보를 관리

```

<security:http pattern="/css/**" security="none"/>

<security:http>
  <security:intercept-url pattern="/top.jsp" access="permitAll()"/>
  <security:intercept-url pattern="/admin/**" access="hasAuthority('ROLE_ADMIN')"/>
  <security:intercept-url pattern="/**" access="isAuthenticated()"/>
  <security:form-login default-target-url="/top.jsp"/>
  <security:logout logout-url="/logout" logout-success-url="/top.jsp"/>
  <security:csrf disabled="true"/>
</security:http>

<security:authentication-manager>
  <security:authentication-provider>
    <security:user-service>
      <security:user name="user" password="1234" authorities="ROLE_USER"/>
      <security:user name="admin" password="1234" authorities="ROLE_ADMIN"/>
    </security:user-service>
  </security:authentication-provider>
</security:authentication-manager>

```

spring security에 의한 제한을 하고 싶지 않은 경우 security="none"을 설정하면 제어 대상에서 제외한다.

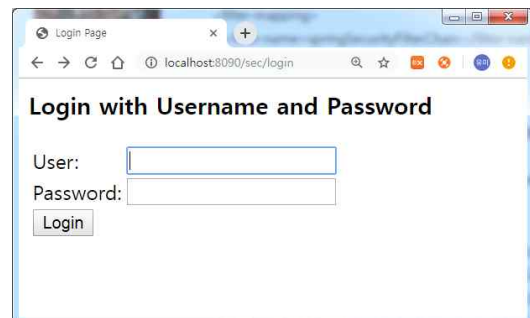
4) Spring Security 활성화(web.xml)

```

<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

```

5) 동작확인



마. 데이터베이스에 인증정보를 저장

1) 테이블 생성

```

CREATE TABLE T_ROLE (
    ID          number primary key,
    ROLE_NAME   varchar2(100) not null,
    DESCRIPTION varchar2(100)
);

CREATE TABLE T_USER (
    ID          number primary key,
    LOGIN_ID    varchar2(20)  not null,
    PASSWORD    varchar2(200) not null,
    FULL_NAME   varchar2(100) not null,
    DEPT_NAME   varchar2(100) not null
);

CREATE TABLE T_USER_ROLE (
    ROLE_ID number,
    USER_ID number,
    foreign key (role_id) references t_role(id),
    foreign key (user_id) references t_user(id),
    primary key(role_id, user_id)
);

--///데이터 입력
insert into t_user values( 1,'user', '1234','사용자','인사');
insert into t_user values( 2,'admin', '1234','관리자','기획');

insert into t_role values(1, 'ROLE_USER','일반사용자');
insert into t_role values(2, 'ROLE_ADMIN','시스템관리자');

insert into t_user_role values(1, 1);
insert into t_user_role values(2, 2);

```

2) <authentication-manager> 설정에서 <authentication-provider> 수정

```

<authentication-manager>
  <authentication-provider>
    <jdbc-user-service data-source-ref="authDataSource"
      users-by-username-query
        = "select LOGIN_ID, '{noop}' || PASSWORD, 1
          from T_USER
          where LOGIN_ID = ?"
      authorities-by-username-query
        = "select LOGIN_ID, ROLE_NAME
          from T_ROLE
          inner join T_USER_ROLE on T_ROLE.ID = T_USER_ROLE.ROLE_ID
          inner join T_USER on T_USER_ROLE.USER_ID = T_USER.ID
          where LOGIN_ID = ?" />
    </authentication-provider>
  </authentication-manager>

```

바. 패스워드 암호화

1) 로그인 암호화

```

<authentication-provider>
  <password-encoder hash="bcrypt"/>

```

- bcrypt
- plaintext
- sha
- sha-256
- md5
- md4
- {sha}
- {sha}

2) 회원가입 시 패스워드 암호화

- 회원가입시 패스워드를 암호화하여 저장해야 한다.

```
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

public class EncryptTest {

    @Test
    public void test() {
        BCryptPasswordEncoder scpwd = new BCryptPasswordEncoder();
        String password = scpwd.encode("1111");
        System.out.println(password);
    }
}
```

사. CSRF protection 활성화

1) CSRF(Cross Site Request Forgery)

CSRF(Cross Site Request Forgery)란 사이트 간 요청 위조 방지하기 위해서 Spring Security가 설정되면 기본적으로 활성화 됨.

Spring Security 의 CSRF protection은 Http 세션과 동일한 생명주기를 가지는 토큰을 발행한 후 Http 요청(PATCH, POST, PUT, DELETE 메서드인 경우)마다 발행된 토큰이 요청에 포함되어 (Http헤더 혹은 파라미터) 있는지 검사하는 가장 일반적으로 알려진 방식의 구현이 설정되어 있다.

2) 미설정 시 에러



Expected CSRF token not found. Has your session expired?

The server understood the request but refuses to authorize it.

3) csrf token을 파라미터에 지정

- 프론트단에서 csrf토큰값을 같이 보내줘야 한다.

```
<form action="login" method="post">
  <sec:csrfinput/>
</form>
```

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

아. spring security 태그 라이브러리

1) SpEL을 사용한 액세스 제어

Expression	Description
hasRole([role])	현재 로그인한 주체(principal)가 권한을 가지고 있으면 true 반환
hasAnyRole([role1,role2])	현재 로그인한 주체가 제공된 권한 중 하나라도 가지고 있으면 true 반환 (선택표로 여러 개의 권한 목록 지정 가능)
principal	현재 로그인한 사용자를 나타내는 객체에 직접 액세스할 수 있다.
authentication	SecurityContext에서 얻은 현재 Authentication 객체에 직접 액세스할 수 있다.
permitAll()	모든 사용자 접근 가능 (Always evaluates to true)
denyAll()	모든 사용자 접근 불가능 (Always evaluates to false)
isAnonymous()	현재 로그인한 사용자가 익명 사용자(anonymous user)인 경우 true 반환
isRememberMe()	현재 로그인한 사용자가 기억하고 있는 사용자(remember-me user)인 경우 true 반환
isAuthenticated()	현재 로그인한 사용자가 익명이 아닌 경우 true 반환 (인증만 되어 있으면 접근 허용)
isFullyAuthenticated()	현재 로그인한 사용자가 익명 또는 기억하고 있는 사용자가 아닌 경우 true 반환

2) authorize 태그(권한을 체크하는 태그)

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

* 로그인 되지 않았다면 참입니다.

<sec:authorize access="isAnonymous()"></sec:authorize>

* 로그인 했다면 참입니다.

<sec:authorize access="isAuthenticated()"></sec:authorize>

* 인자로 주어진 롤을 가지고 있다면 참입니다.

<sec:authorize access="hasRole('ROLE_ADMIN')"></sec:authorize>

* 인자로 주어진 롤을 가지고 있지 않다면 참입니다.

<sec:authorize access="!hasRole('ROLE_ADMIN')"></sec:authorize>

* 인자로 주어진 롤들 중 하나라도 가지고 있다면 참입니다.

<sec:authorize access="hasAnyRole('ROLE_ADMIN','ROLE_MANAGER')"></sec:authorize>

3) authentication 태그

* 사용자 정보 조회.

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

자. 세션정보 조회

```
@RequestMapping(value = "/", method = RequestMethod.GET)
public String home(Locale locale,
                  Model model,
                  @AuthenticationPrincipal UserDetails user) {
```

```
    UserDetails userDetails =
        (UserDetails)SecurityContextHolder.getContext().getAuthentication().getPrincipal();
```

```
// 시큐리티 컨텍스트 객체를 얻습니다.
SecurityContext context = SecurityContextHolder.getContext();

// 인증 객체를 얻습니다.
Authentication authentication = context.getAuthentication();

// 로그인한 사용자정보를 가진 객체를 얻습니다.
Principal principal = authentication.getPrincipal();

// 사용자가 가진 모든 롤 정보를 얻습니다.
Collection<? extends GrantedAuthority> authorities = authentication.getAuthorities(); Iterator<?
extends GrantedAuthority> iter = authorities.iterator();
while (iter.hasNext()) {
    GrantedAuthority auth = iter.next();
    System.out.println(auth.getAuthority());
}
```

차. 로그인 처리 후 추가작업

```
public class CustomLoginSuccessHandler implements AuthenticationSuccessHandler {
    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse
        response, Authentication authentication) throws IOException, ServletException {

        //로그인 후 추가작업

        response.sendRedirect("top.jsp")
    }
}
```

```
<bean id="loginHandler" class="com.yedam.app.CustomLoginSuccessHandler"/>

<security:http>

    <security:form-login default-target-url="/top.jsp"
        authentication-success-handler-ref="loginHandler" />

</security:http>
```

카. 로그아웃 처리 후 추가작업

타. 독자적으로 인증하는 방식

1) DetailUser 구현

```
public class UserVO implements UserDetails {
    @Override
    public void getPassword(){
        return password;
    }
    @Override
    public void getUsername(){
        return id;
    }
    @Override
    public void setPassword(){
        return password;
    }
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities(){
        List<GrantAuthority> auth = new ArrayList<>();
        auth.add(new SimpleGrantedAuthority(this.role)
        return auth;
    }
}
```

2) UserDetailsService 구현

```
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class CustomUserDetailsService implements UserDetailsService{

    @Autowired
    UserDAO dao;

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

        UsersVO vo = new UsersVO();
        vo.setName(username);
        vo = dao.getUser(vo);

        //사용자가 존재하지 않는 경우
        if(vo == null) {
            throw new UsernameNotFoundException("no user");
        }

        //권한 지정
        List<GrantedAuthority> auth = new ArrayList<>();
        auth.add(new SimpleGrantedAuthority(vo.getRole()));

        //User 객체 return
        return new SampleUser(username, vo.getPassword(), auth);
    }

    public Collection<GrantedAuthority> getAuthorities(String username) {
        List<Authority> authList = dao.getRoles(username);
        List<GrantedAuthority> authorities = new ArrayList<>();
        for (Authority authority : authList) {
            authorities.add(new SimpleGrantedAuthority(authority.getAuthority()));
        }
        return authorities;
    }
}
```


3) security-context.xml에서 provider에 서비스 교체

```
<beans:bean id="userDetailsService" class="com.lec.security.CustomUserDetailsService"/>

<authentication-manager>
  <authentication-provider user-service-ref="userDetailsService">
    <password-encoder hash="bcrypt"/>
  </authentication-provider>
</authentication-manager>
```

4) provider 커스터마이징

```
package co.company.spring.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import co.company.spring.users.service.impl.CustomUserDetailsService;

@Configuration
@EnableWebSecurity // 스프링시큐리티 사용을 위한 어노테이션선언
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    // @Autowired
    // private AuthenticationProvider authenticationProvider;

    @Autowired
    CustomUserDetailsService userService;

    /*
     * @Bean public CustomUserDetailsService detailsService() { return new
     * CustomUserDetailsService(); }
     */

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    /* * 스프링 시큐리티가 사용자를 인증하는 방법이 담긴 객체. */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        /* * AuthenticationProvider 구현체 */
        // auth.authenticationProvider(authenticationProvider);
        auth.userDetailsService(userService)
            .passwordEncoder(passwordEncoder());
    }

    /* * 스프링 시큐리티 규칙 */
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests() // 보호된 리소스 URI에 접근할 수 있는 권한을 설정
        // .antMatchers("/").permitAll() // 전체 접근 허용
    }
}
```

```
.antMatchers("/top.jsp").permitAll() // 전체 접근 허용
.antMatchers("/admin/**").hasRole("ADMIN")// admin이라는 롤을 가진 사용자만 접근 허용
// .anyRequest().authenticated()
.antMatchers("/**").authenticated().and().formLogin()
// .loginPage("/login")
// .loginProcessingUrl("loginprocess")
.defaultSuccessUrl("/top.jsp")
// .usernameParameter("id")
// .passwordParameter("password")
.and().logout().logoutUrl("/logout").logoutSuccessUrl("/top.jsp")
.and().csrf().disable(); // csrf 보안 설정을 비활성화
}
}
```

2. kakao login

kakao developers

내 애플리케이션

제품

문서

내 애플리케이션 > 앱 설정 > 앱 키

앱 설정

요약 정보

일반

비즈니스

앱 키

플랫폼

팀 관리

제품 설정

카카오 로그인

동의항목

간편가입

카카오톡 채널

개인정보 국외이전

연결 끊기

사용자 프로퍼티

APP

예담

ID 524658

OWNER

Web

앱 키

- 네이티브 앱 키: Android, iOS SDK에서 API를 호출할 때 사용합니다.
- JavaScript 키: JavaScript SDK에서 API를 호출할 때 사용합니다.
- REST API 키: REST API를 호출할 때 사용합니다.
- Admin 키: 모든 권한을 갖고 있는 키입니다. 노출이 되지 않도록 주의가 필요합니다.

kakao developers

내 애플리케이션

제품

문서

내 애플리케이션 > 앱 설정 > 플랫폼

앱 설정

요약 정보

일반

비즈니스

앱 키

플랫폼

팀 관리

제품 설정

카카오 로그인

동의항목

간편가입

카카오톡 채널

개인정보 국외이전

연결 끊기

사용자 프로퍼티

보안

고급

카카오링크

APP

예담

ID 524658

OWNER

Web

Android

Android 플랫폼 등록

iOS

iOS 플랫폼 등록

Web

삭제

수정

- 카카오 로그인 사용 시 Redirect URI를 등록해야 합니다. [등록하러 가기](#)

앱 설정
요약 정보
일반
비즈니스
앱 키
플랫폼
팀 관리

제품 설정

카카오 로그인

동의함목
간편가입
카카오톡 채널
개인정보 국외이전
연결 끊기
사용자 프로퍼티
보안

카카오 로그인 **ON**

[동의함 미리보기](#)

활성화 설정

상태

ON

카카오 로그인 API를 활용하면 사용자들이 번거로운 회원 가입 절차 대신, 카카오로 서비스 시작할 수 있습니다.
상태가 OFF일 때도 카카오 로그인 설정 항목을 변경하고 서버에 저장할 수 있습니다.
상태가 ON일 때만 실제 서비스에서 카카오 로그인 화면이 연결됩니다.

Redirect URI

삭제

수정

Redirect URI	
http://localhost:81/springTemplate/kakaologin	

- 카카오 로그인에서 사용할 OAuth Redirect URI를 설정합니다. (최대 10개)
- REST API로 개발하는 경우 필수로 설정해야 합니다.

```
<!-- httpclient -->
<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.13</version>
</dependency>

<!-- jackson-databind -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.12.0</version>
</dependency>
```