Spring Framework

Spring Security Tutorial

❑ 본 교재는 스프링 시큐리티 4.X버전을 기반으로 하며, 스프링 MVC 모듈에 관해 선수 과정을 수강한 학생들을 대상으로 작성되었음을 참고하시기 바랍니다.

대덕인재개발원 : 최희연

❑ 보안의 이해

인가 : 인정하여 허가 함

- Authentication(인증) + Authorization(인가) : 검증과 권한 부여
  (권한 부여를 위해서는 최소 1차례의 인증과정이 필요함을 의미)
- Authentication(인증)이란 : 해당 사용자가 본인이 맞는지 확인하는 절차.
- Authentication(인증) 종류
  - **Credential(자격) 기반 인증** : Principle(아이디) 와 Credential(비밀번호) 에 기반한 인증.
  - **Two-factor authentication (이중인증)** : 로그인 후 보안 인증을 통해 2차례 인증 진행.
  - 물리적 인증 : 물리적 기계 장치를 이용한 인증방식, ex) 지문 인식이나 키 입력.
- Authorization(인가) 란 : 인증된 사용자(principal)가 요청된 자원을 접근할 수 있는지 결정하는 절차.
- Authorization(인가) 종류
  - Granted Authority(부여된 권한) : 적절한 절차로 사용자가 인증되었음을 가정하여 부여된 권한은 persistence 한 영역에 저장되어 관리함.
  - Resource Authority(리소스 권한) : 리소스에 대한 권한이 없는 사용자로부터의 리퀘스트를 가로채는(intercept) 방식(웹 보안의 핵심)
- Security Service 라 함은 사용자 정보를 관리하는 영속계층으로부터 확인된 인증 정보를 통해 신원검증을 거치는 Authentication 과 특정 리소스및 서비스에 대한 권한 정보를 계층화 시켜 화면 및 페이지 또는 메소드에 접근가능 여부를 확인하는 Authorization 을 포함.

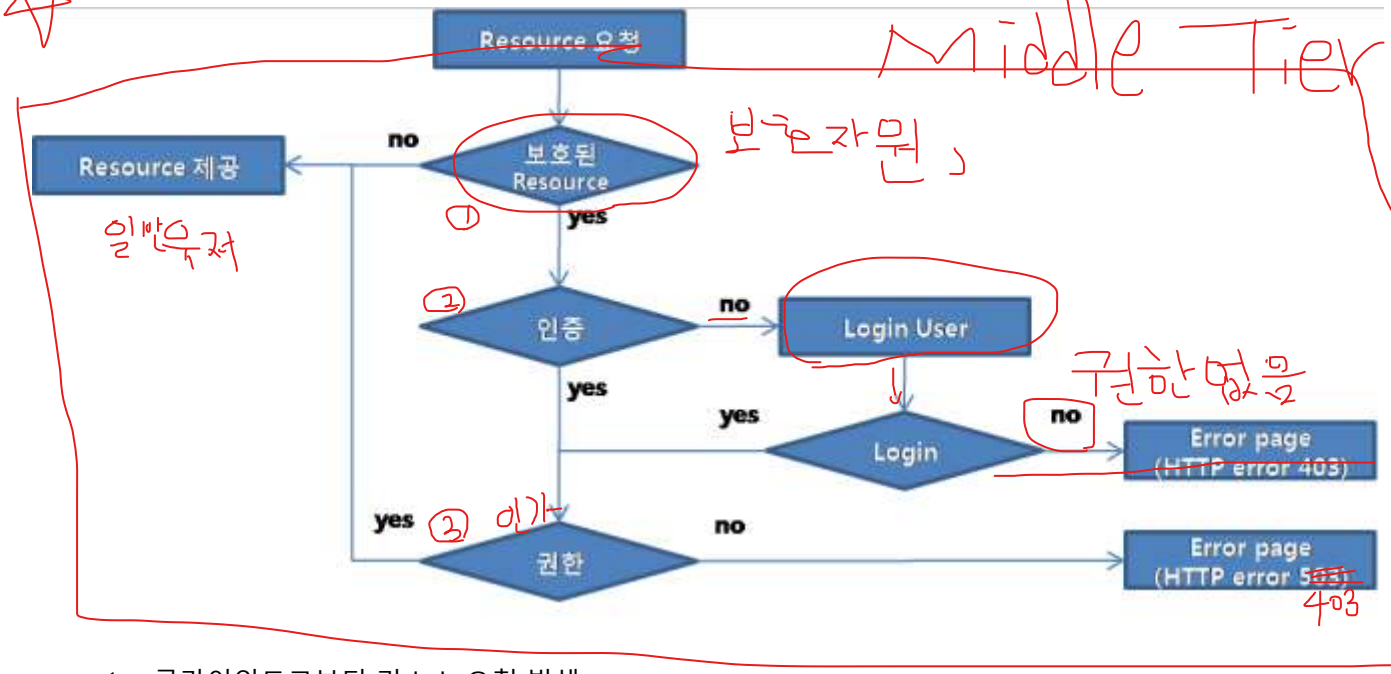
❑ Spring Security

- 엔터프라이즈 어플리케이션을 위한 인증, 권한처리 서비스를 제공하는 보안 솔루션.
- Servlet Filter와 AOP 를 통한 보안을 강제하여 Spring IoC 의 lifecycle 기반으로 동작.
- Authentication, Web URL authorization, Method 호출 authorization, 도메인 객체 기반의 security 처리, 채널보안(https 강제)등의 기능.
- Web request 보안과 더불어 Service layer 및 인스턴스 수준의 보안 제공으로 layering issue 해결 및 웹 클라이언트 외의 다양한 rich 클라이언트/웹서비스에 대한 보안 제어 지원.
- 재사용성, 이식성, 코드 품질, 레퍼런스(정부,은행,대학,기업 등 많은 business field), 다양한 타 프레임워크를 지원하고, 커뮤니티가 활성화되어있음.

❑ Spring Security 주요 기능

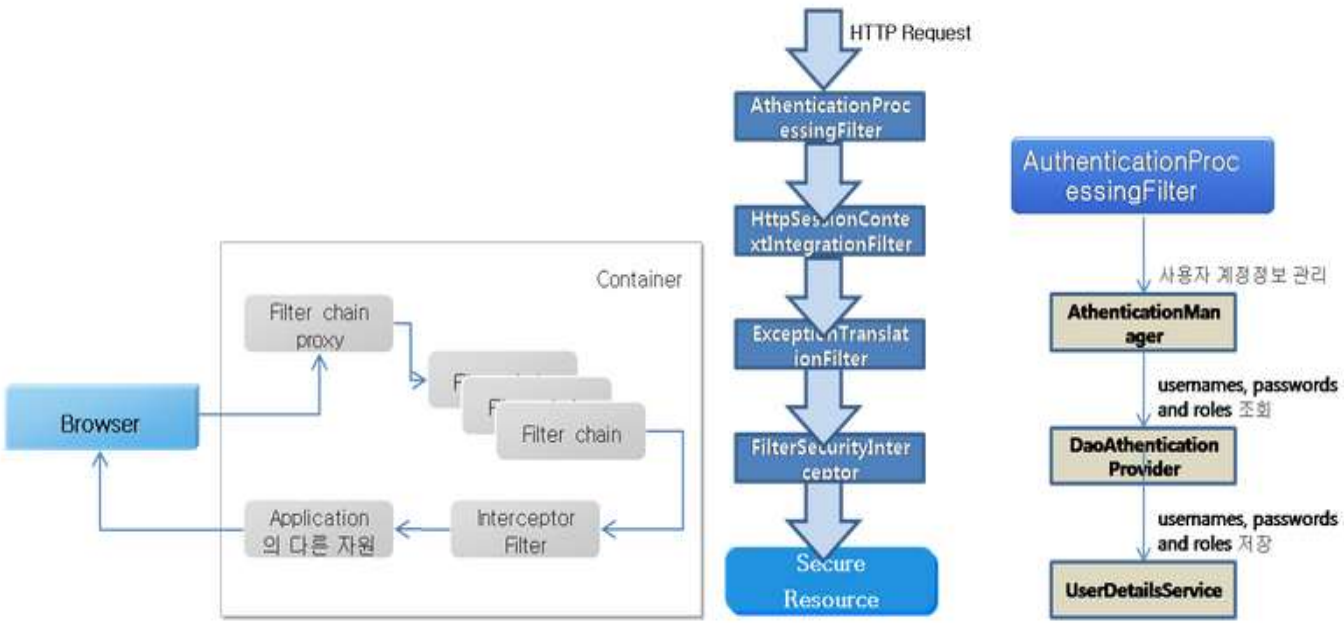
- 자원(url, method 등의 resource) 접근 제한
- 사용자 인증 확인
- 미인증시 인증확인 요청
- 계층적 권한(Hierarchical Role) 설정 및 사용자 권한(Customized Role) 사용

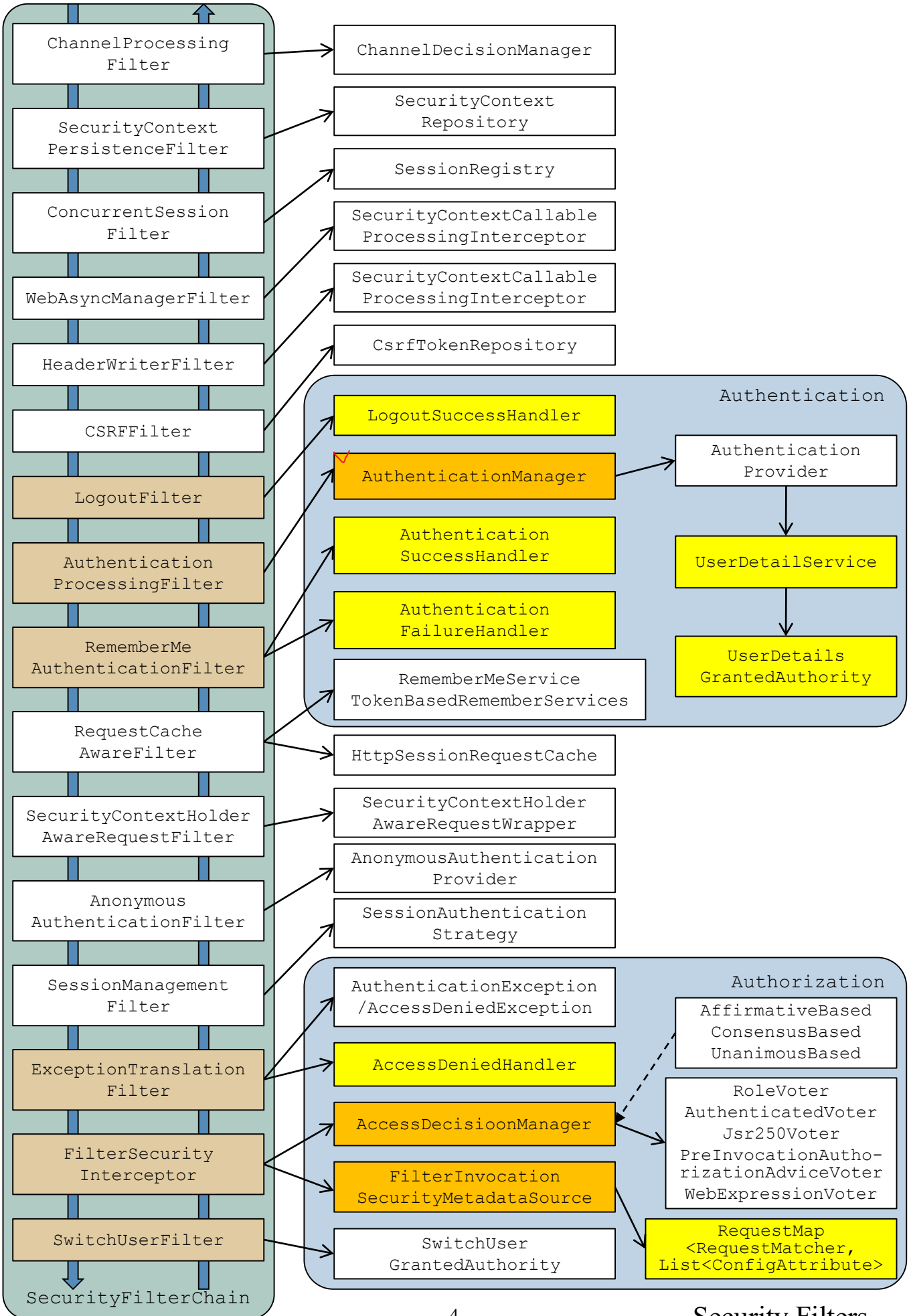
❑ Spring Security Architecture



1. 클라이언트로부터 리소스 요청 발생
2. 요청 자원의 보호 여부 판단
3. 아직 인증이 안되었으므로 HTTP 응답코드() 또는 특정 페이지로 redirect
4. 인증 메커니즘에 따라 웹 페이지 로그인 폼 또는 X509 인증서(브라우저의 login form) 제공
5. 입력 폼의 내용을 HTTP post 또는 인증 세부사항을 포함하는 HTTP 헤더를 서버로 요청
6. 신원정보(credential)가 유효한지 판단
    1. 유효한 경우 다음 단계 진행
    2. 유효하지 않을 경우 신원정보 재요청(되돌아감)
7. 보호자원의 접근 권한이 있을 경우 요청 성공 / 권한이 없을 경우 forbidden 403 HTTP 오류

❑ Spring Security Filter Chain(인증 및 인가 처리는 필터를 기반으로 함.)





Security Filters

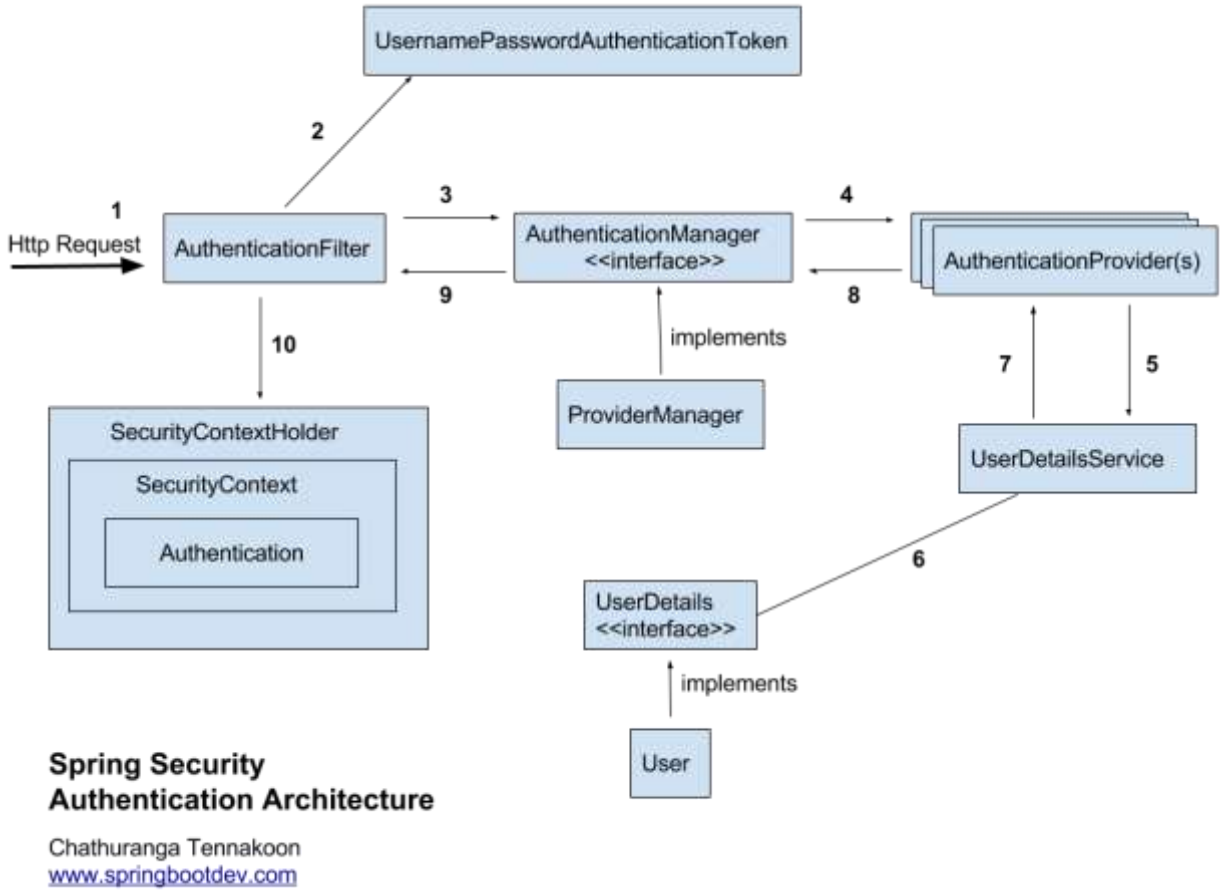
❑ Filter Chain 내 Filter 의 동작순서와 역할

: 필터 체인 내의 필터들은 다음 순서에 따라 동작하며, filter="none"설정이 있는 경우 필터링 생략되고, SecurityContextHolder 내에 SecurityContext 가 관리되지 않는다.

1. ChannelProcessingFilter : HTTPS 보안 채널로 redirection 여부를 결정할 필터.
2. SecurityContextPersistenceFilter : SecurityContextHolder 관리 필터
   ① 요청이 최초 발생시 SecurityContextRepository 로부터 조회된 SecurityContext 를 SecurityContextHolder 에 저장.
   ② 응답 전송 전 SecurityContextHolder 의 변경사항을 HttpSession 에 반영.
3. ConcurrentSessionFilter : 세션 관리 필터
   ① SessionRegistry.refreshLastRequest 를 이용하여 세션의 마지막 요청 갱신
   ② SessionInformation 을 조회하여 만료된 세션인 경우, LogoutHandler 를 동작시키고
4. WebAsyncManagerIntegrationFilter : WebAsyncManager 에 SecurityContext 를 설정하기 위한 SecurityContextCallableProcessingInterceptor 를 등록하는 필터.
5. HeaderWriterFilter : HeaderWriter 를 응답에 기록하기 위한 필터
   - X-Frame-Options, X-XSS-Protection, X-Content-Type-Options 등의 헤더 설정(20p의 headers 참고).
6. CSRFFilter : CsrfTokenRepository 에서 유지되는 토큰을 기반으로 CSRF 공격을 방어하는 필터
7. LogoutFilter : 일련의 LogoutHandler 들을 이용해 로그아웃 요청을 처리하는 필터.
   - SecurityContextLogoutHandler, TokenBasedRememberMeServices, CompositeLogoutHandler 등이 주로 사용되고, LogoutSuccessHandler 나 logoutSuccessUrl 을 이용해 로그아웃 이후 상황이 처리됨.
8. AuthenticationProcessing mechanism : AuthenticationProcessingFilter 계열을 이용해 인증을 처리하고, SecurityContextHolder 내의 SecurityContext 에 Authentication 객체를 갱신함.
   ① UsernamePasswordAuthenticationFilter(FORM_LOGIN_FILTER) : form 입력 아이디와 비밀번호 기반의 인증 처리 필터.
      • 성공 – SecurityContext 에 Authentication 객체를 저장하고, AuthenticationSuccessHandler를 사용하여, 인증 이후의 상황을 처리한 후 InteractiveAuthenticationSuccessEvent 를 발행함.
      • 실패 – AuthenticationFailureHandler 를 호출하거나 401 에러 코드를 전송함.
   ② BasicauthenticationFilter : 브라우저의 기본 인증 메커니즘(Authorization 헤더)으로 전송된 데이터 기반의 인증 처리 필터. Ex) Authorization : BASIC YWxhZGRpbjpvcGVuc2VzYW1l
9. SecurityContextHolderAwareRequestFilter : Security API 를 활용할 수 있는 getAuthentication, isUserInRole 등의 메소드가 정의된 SecurityContextHolderAwareRequest 를 생성하는 필터
10. RememberMeAuthenticationFilter : 이전의 인증 객체가 없고, remember-service 가 활성화된 경우, 쿠키 기반의 remember-me token 으로 SecurityContext 를 갱신함.
11. AnonymousAuthenticationFilter : 이전의 인증 객체가 없으면, 익명 Authentication 를 생성함.
12. SessionManagementFilter : 인증 객체가 존재하는 요청의 경우, SessionAuthenticationStrategy 에 따라 Session-Fixation 공격 방어 등과 같은 처리를 담당.
13. ExceptionTranslationFilter : 필터 체인내에서 발생하는 예외를 처리하고, Http 에러 응답을 전송하거나 AuthenticationEntryPoint 를 실행함.
14. FilterSecurityInterceptor : Voter 기반의 AccessDecisionManager 를 이용하여 자원에 대한 접근을 제어하고, 접근이 거부되면 예외를 발생시킴.
15. SwithUserFilter : 특정 자원에 대해 사용자의 role 을 변경해야 하는 경우 활용되며, 일반적으로 ROLE_ADMIN 사용자가 일반 사용자가 추가 관리자로 지정할때 사용함.

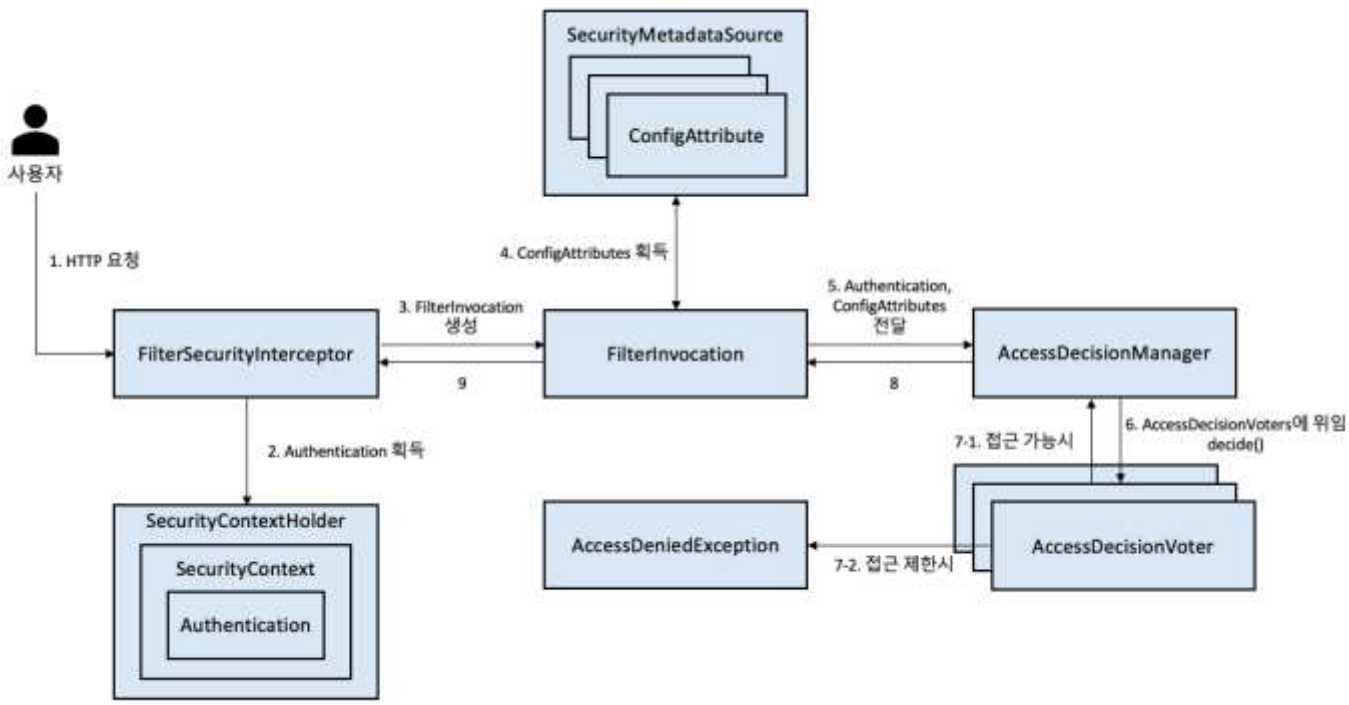
❑ Spring Security Authentication Architecture

https://chathurangat.wordpress.com/2017/08/23/spring-security-authentication-architecture/



❑ Spring Security Authorization Architecture

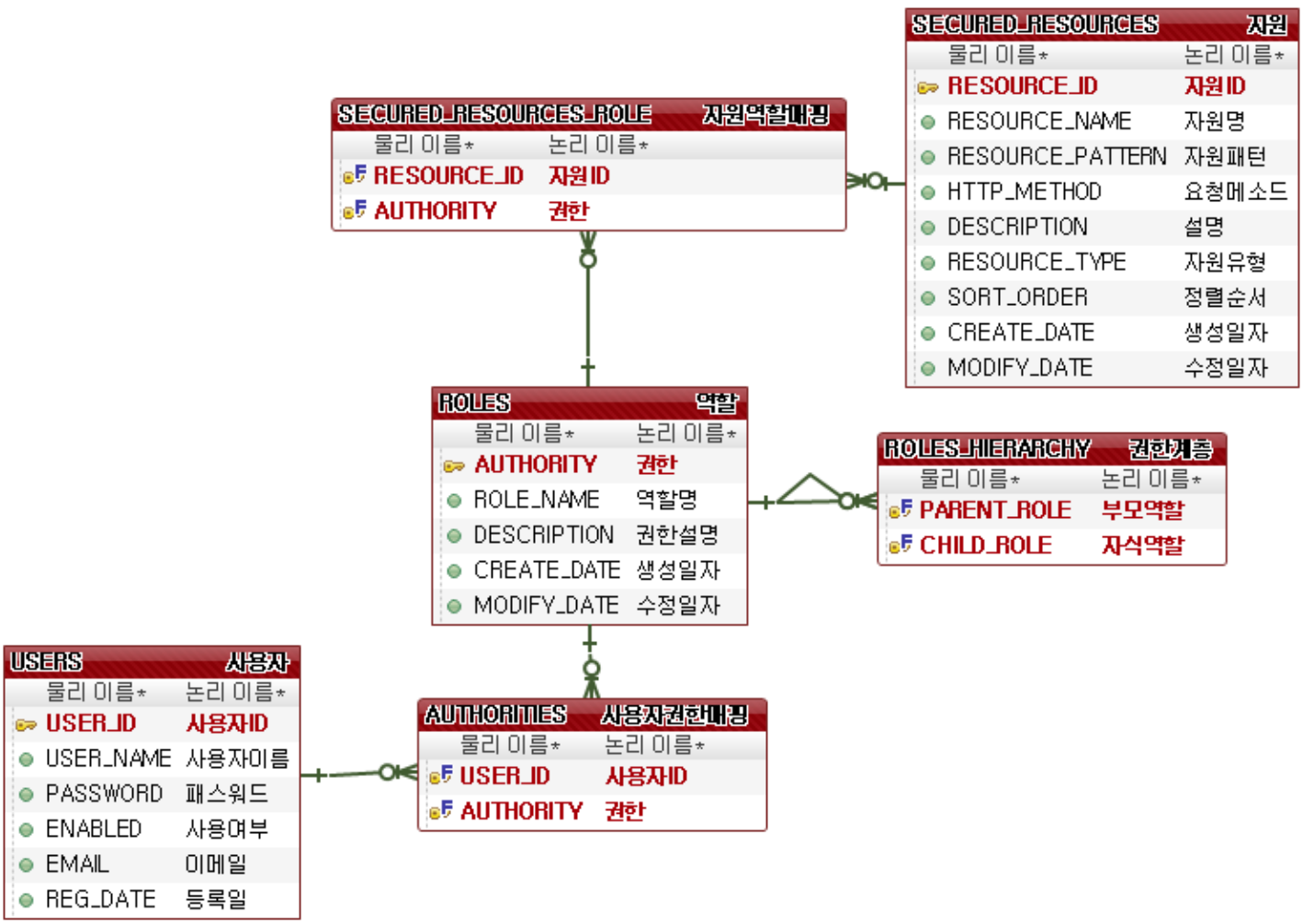
https://velog.io/@minthug94_/Spring-Security-Authorization



1. Spring Security 에서 생성한 정보 : SecurityContextHolder를 이용하여 SecurityContext 확보.
2. 응용프로그램이 분산되어 있는 경우 등 다양한 환경에서 사용 가능토록 SecurityContext를 SecurityContextHolder 내부에 생성한 ThreadLocal 객체를 이용하여 저장.
3. ThreadLocal 객체는 현재 쓰레드에서 필요한 상태 정보만 담을 수 있음.
4. 웹 환경에서 요청이 있을때 마다 동일한 역할을 하는 SecurityContext를 다시 생성하는 것은 맞지않음 → httpSessionContextIntegrationFilter 를 사용하여 SecurityContext 정보를 ThreadLocal 에 기록하고 가져오는 작업을 수행함.(Session 에 저장)

❑ 인증 및 인가 처리를 위한 테이블 스키마

– 사용자 인증과 관련된 테이블은 사용자테이블(Member) 과 사용자권한관련 테이블이며, 사용자권한 관련 테이블은 역할, 자원, 역할계층 등의 테이블로 구성됨.



```
/* 사용자 */
CREATE TABLE USERS (
	USER_ID VARCHAR2(15) NOT NULL, /* 사용자ID */
	USER_NAME varchar2(20) NOT NULL, /* 사용자이름 */
	PASSWORD varchar2(200), /* 패스워드 */
	ENABLED CHAR(1), /* 사용여부 */
	EMAIL VARCHAR2(40) NOT NULL, /* 이메일 */
	REG_DATE DATE NOT NULL, /* 등록일 */
	CONSTRAINT PK_USERS PRIMARY KEY(USER_ID)
);
/* 역할 */
CREATE TABLE ROLES (
	AUTHORITY VARCHAR2(50) NOT NULL, /* 권한 */
	ROLE_NAME VARCHAR2(50), /* 역할명 */
	DESCRIPTION VARCHAR2(100), /* 권한설명 */
	CREATE_DATE DATE, /* 생성일자 */
	MODIFY_DATE DATE, /* 수정일자 */
	CONSTRAINT PK_ROLES PRIMARY KEY(AUTHORITY)
);
/* 사용자권한매핑 */
CREATE TABLE AUTHORITIES (
	USER_ID VARCHAR2(15) NOT NULL, /* 사용자ID */
	AUTHORITY VARCHAR2(50) NOT NULL, /* 권한 */
	CONSTRAINT PK_AUTHORITIES PRIMARY KEY(USER_ID, AUTHORITY),
	CONSTRAINT FK_USERS_AUTHORITIES FOREIGN KEY(USER_ID) REFERENCES USERS(USER_ID),
	CONSTRAINT FK_ROLES_AUTHORITIES FOREIGN KEY(AUTHORITY) REFERENCES ROLES(AUTHORITY)
);

INSERT INTO ROLES
(
	AUTHORITY, ROLE_NAME, DESCRIPTION,
	CREATE_DATE, MODIFY_DATE
) VALUES
(
	'IS_AUTHENTICATED_ANONYMOUSLY', 'ANONYMOUS', '익명사용자',
	SYSDATE, SYSDATE
);
INSERT INTO ROLES
(
	AUTHORITY, ROLE_NAME, DESCRIPTION,
	CREATE_DATE, MODIFY_DATE
) VALUES
(
	'IS_AUTHENTICATED_REMEMBERED', 'REMEMBERED', 'REMEMBERED 사용자',
	SYSDATE, SYSDATE
);
INSERT INTO ROLES
(
	AUTHORITY, ROLE_NAME, DESCRIPTION,
	CREATE_DATE, MODIFY_DATE
) VALUES
(
	'IS_AUTHENTICATED_FULLY', 'AUTHENTICATED', '인증된 사용자',
	SYSDATE, SYSDATE
);
```

```
INSERT INTO ROLES
(
	AUTHORITY, ROLE_NAME, DESCRIPTION,
	CREATE_DATE, MODIFY_DATE
) VALUES
(
	'ROLE_RESTRICTED', 'RESTRICTED', '제한된 사용자',
	SYSDATE, SYSDATE
);
INSERT INTO ROLES
(
	AUTHORITY, ROLE_NAME, DESCRIPTION,
	CREATE_DATE, MODIFY_DATE
) VALUES
(
	'ROLE_USER', 'USER', '일반 사용자',
	SYSDATE, SYSDATE
);
INSERT INTO ROLES
(
	AUTHORITY, ROLE_NAME, DESCRIPTION,
	CREATE_DATE, MODIFY_DATE
) VALUES
(
	'ROLE_ADMIN', 'ADMIN', '관리자',
	SYSDATE, SYSDATE
);
INSERT INTO ROLES
(
	AUTHORITY, ROLE_NAME, DESCRIPTION,
	CREATE_DATE, MODIFY_DATE
) VALUES
(
	'ROLE_A', 'A BIZ', 'A 업무',
	SYSDATE, SYSDATE
);

--역할들 간의 계층구조를 저장하는 테이블
CREATE TABLE ROLES_HIERARCHY (
	PARENT_ROLE VARCHAR(50) NOT NULL,
	CHILD_ROLE VARCHAR(50) NOT NULL,
	CONSTRAINT PK_ROLES_HIERARCHY PRIMARY KEY(PARENT_ROLE,CHILD_ROLE),
	CONSTRAINT FK_ROLES1 FOREIGN KEY(PARENT_ROLE) REFERENCES ROLES(AUTHORITY),
	CONSTRAINT FK_ROLES2 FOREIGN KEY(CHILD_ROLE) REFERENCES ROLES (AUTHORITY)
);

INSERT INTO ROLES_HIERARCHY
(
		CHILD_ROLE, PARENT_ROLE
) VALUES
(
		'ROLE_ADMIN', 'ROLE_USER'
);
```

```
INSERT INTO ROLES_HIERARCHY
(
		CHILD_ROLE, PARENT_ROLE
) VALUES
(
		'ROLE_USER', 'ROLE_RESTRICTED'
);
INSERT INTO ROLES_HIERARCHY
(
		CHILD_ROLE, PARENT_ROLE
) VALUES
(
		'ROLE_RESTRICTED', 'IS_AUTHENTICATED_FULLY'
);
INSERT INTO ROLES_HIERARCHY
(
		CHILD_ROLE, PARENT_ROLE
) VALUES
(
		'IS_AUTHENTICATED_FULLY', 'IS_AUTHENTICATED_REMEMBERED'
);
INSERT INTO ROLES_HIERARCHY
(
		CHILD_ROLE, PARENT_ROLE
) VALUES
(
		'IS_AUTHENTICATED_REMEMBERED', 'IS_AUTHENTICATED_ANONYMOUSLY'
);
INSERT INTO ROLES_HIERARCHY
(
		CHILD_ROLE, PARENT_ROLE
) VALUES
(
		'ROLE_ADMIN', 'ROLE_A'
);

INSERT INTO ROLES_HIERARCHY
(
		CHILD_ROLE, PARENT_ROLE
) VALUES
(
		'ROLE_A', 'ROLE_RESTRICTED'
);
--보호되는 자원에 대한 테이블
CREATE TABLE SECURED_RESOURCES (
	RESOURCE_ID VARCHAR(10) NOT NULL,
	RESOURCE_NAME VARCHAR(50),
	RESOURCE_PATTERN VARCHAR(300) NOT NULL,
	HTTP_METHOD VARCHAR2(20),
	DESCRIPTION VARCHAR(100),
	RESOURCE_TYPE VARCHAR(10),
	SORT_ORDER NUMBER(4) DEFAULT 9999,
	CREATE_DATE DATE,
	MODIFY_DATE DATE,
	CONSTRAINT PK_RECURED_RESOURCES PRIMARY KEY(RESOURCE_ID)
);
```

```
INSERT ALL
INTO SECURED_RESOURCES
(   RESOURCE_ID, RESOURCE_NAME, RESOURCE_PATTERN, RESOURCE_TYPE )
VALUES (  'web-000001', 'exceptUrl', '/**', 'url' )
INTO SECURED_RESOURCES
(   RESOURCE_ID, RESOURCE_NAME, RESOURCE_PATTERN, RESOURCE_TYPE, SORT_ORDER )
VALUES (  'web-000002', 'boardInsertUrl', '/board/*Insert**', 'url', 1 )
INTO SECURED_RESOURCES
(  RESOURCE_ID, RESOURCE_NAME, RESOURCE_PATTERN, RESOURCE_TYPE, SORT_ORDER )
VALUES (  'web-000003', 'boardUpdateUrl', '/board/*Update**',  'url', 1 )
INTO SECURED_RESOURCES
(  RESOURCE_ID, RESOURCE_NAME, RESOURCE_PATTERN, RESOURCE_TYPE, SORT_ORDER )
VALUES (  'web-000004', 'boardDeleteUrl', '/board/*Delete**', 'url', 1 )
INTO SECURED_RESOURCES
(  RESOURCE_ID, RESOURCE_NAME, RESOURCE_PATTERN, HTTP_METHOD, RESOURCE_TYPE, SORT_ORDER )
VALUES (  'web-000005', 'boardRead', '/board/**', 'GET', 'url', 3 )
INTO SECURED_RESOURCES
(  RESOURCE_ID, RESOURCE_NAME, RESOURCE_PATTERN, HTTP_METHOD, RESOURCE_TYPE, SORT_ORDER )
VALUES (  'web-000006', 'boardCreate', '/board/**', 'POST', 'url', 3 )
INTO SECURED_RESOURCES
(  RESOURCE_ID, RESOURCE_NAME, RESOURCE_PATTERN, HTTP_METHOD, RESOURCE_TYPE, SORT_ORDER )
VALUES (  'web-000007', 'boardModify', '/board/**', 'PUT', 'url', 3 )
INTO SECURED_RESOURCES
(  RESOURCE_ID, RESOURCE_NAME, RESOURCE_PATTERN, HTTP_METHOD, RESOURCE_TYPE, SORT_ORDER )
VALUES (  'web-000008', 'boardRemove', '/board/**', 'DELETE', 'url', 3 )
SELECT * FROM DUAL;

--보호된 자원 역할 테이블
CREATE TABLE SECURED_RESOURCES_ROLE (
       RESOURCE_ID VARCHAR(10) NOT NULL,
       AUTHORITY VARCHAR(50) NOT NULL,
       CONSTRAINT FK_SECURED_RESOURCES FOREIGN KEY(RESOURCE_ID) REFERENCES
      SECURED_RESOURCES(RESOURCE_ID),
);
INSERT ALL INTO SECURED_RESOURCES_ROLE (  RESOURCE_ID, AUTHORITY )
VALUES (  'web-000001', 'permitAll' )
INTO SECURED_RESOURCES_ROLE (  RESOURCE_ID, AUTHORITY     )
VALUES (  'web-000002', 'ROLE_ADMIN' )
INTO SECURED_RESOURCES_ROLE (  RESOURCE_ID, AUTHORITY     )
VALUES (  'web-000003', 'ROLE_ADMIN' )
INTO SECURED_RESOURCES_ROLE (  RESOURCE_ID, AUTHORITY     )
VALUES (  'web-000004', 'ROLE_ADMIN' )
INTO SECURED_RESOURCES_ROLE (  RESOURCE_ID, AUTHORITY     )
VALUES (  'web-000005', 'ROLE_USER' )
INTO SECURED_RESOURCES_ROLE (  RESOURCE_ID, AUTHORITY     )
VALUES (  'web-000006', 'ROLE_ADMIN' )
INTO SECURED_RESOURCES_ROLE (  RESOURCE_ID, AUTHORITY     )
VALUES (  'web-000007', 'ROLE_ADMIN' )
INTO SECURED_RESOURCES_ROLE (  RESOURCE_ID, AUTHORITY     )
VALUES (  'web-000008', 'ROLE_ADMIN' )
SELECT * FROM DUAL;
```

❑ Spring security 사용 예제 (Spring 게시판 tutorial 에 연결) – InMemory 방식

1. Spring Security 의존성 추가

```
<properties>
<org.springframework.security-version>5.3.0.RELEASE</org.springframework.security-version>
</properties>
<!-- ..................... -->
<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>
```

2. Deployment Descriptor 설정

1) web.xml 수정

```
<context-param>
	<param-name>contextConfigLocation</param-name>
	<param-value>/WEB-INF/spring/*Context.xml</param-value>
</context-param>

<filter>
	<filter-name>springSecurityFilterChain</filter-name>
	<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
	<init-param>
		<param-name>targetBeanName</param-name>
		<param-value>springSecurityFilterChain</param-value>
	</init-param>
</filter>
<filter-mapping>
	<filter-name>springSecurityFilterChain</filter-name>
	<url-pattern>/*</url-pattern>
</filter-mapping>
```

- DelegatingFilterProxy : 스프링 컨테이너 내부에 등록된 필터 체인에게 intercept한 리퀘스트 필터링을 위임하는 fiter proxy.
- 필터링을 위임할 빈 이름은 filter-name 이나 targetBeanName 파라미터로 지정.
- springSecurityFilterChain 은 상위컨텍스트에 xml 이나 java configuration 형태로 등록함.

3. 인증 정보 관리를 위한 데이터 베이스(앞의 스키마 참조)나 파일 생성

```
#username(id)=password,authority[,authority],enabled[disabled]
a001=7c5c579cb7849489ce44e274da4d17bbfa75cc6834325f53603c34d9a6c916c7,ROLE_ADMIN,1
b001=bfe7cdc14e6d787ce39e5fd42302594f63069d27a4628bbccd3d0b2417e97bc5,ROLE_USER,1
c001=6d200af5f8e58282859b11fca1db849fc771d873ead0f78fb9b53939351d20e5,ROLE_USER,1
```

In memory 방식의 계정 관리를 위해 properties 파일 생성

4. securityContext.xml 을 통해 springSecurityFilterChain 을 형성할, 인증 및 인가와 관련된 빈 등록 (/WEB-INF/spring/securityContext.xml).

1) In Memory User Service를 사용하는 경우.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
	xmlns:beans="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:c="http://www.springframework.org/schema/c"
	xmlns:p="http://www.springframework.org/schema/p"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
	http://www.springframework.org/schema/beans/spring-beans.xsd
	http://www.springframework.org/schema/security
	http://www.springframework.org/schema/security/spring-security.xsd">
<!-- properties 객체나 메모리내에서 인증 기반 정보를 유지하는 경우,InMemory User Service 를 사용함. -->
<!-- properties : 인증 기반 정보를 가진 properties 파일의 위치 -->
	<user-service id="inMemoryUserService"
		properties="/WEB-INF/spring/security/user.properties" />
<!-- user service 를 기반으로 인증을 수행하는 authentication manager(=ProviderManager) 의 등록. -->
<!-- user service 의 형태에 따라 사용되는 provider 의 종류가 달라지며, 이는 security 네임스페이스를
사용하는 경우, 여러구현체중 자동 선택됨. -->
	<authentication-manager>
<!-- SHA256 단방향 함수를 사용한 비밀번호 암호화, salt 객체를 사용해 좀 더 정교한 암호화 가능. -->
		<authentication-provider user-service-ref="inMemoryUserService">
			<password-encoder hash="sha-256">
				<salt-source user-property="username"/>
			</password-encoder>
		</authentication-provider>
	</authentication-manager>
<!-- springSecurityFilterChain 을 타지않을 리퀘스트 지정. -->
	<http security="none" pattern="/resources/**" />
	<http security="none" pattern="/board/boardList.do" />
<!-- 등록된 authenticationManager와 accessDecisionManager 를 사용한  -->
<!-- 리퀘스트에 대한 시큐리티 설정을 하기 위한 엘리먼트 -->
<!-- hasRole()나 hasAnyRole()등의 spEL 형식의 SecurityExpressionRoot 의  -->
<!-- built-in 표현식을 사용하기 위해서는 use-expressions="true" 설정이 반드시 필요함. -->
	<http auto-config="true"
		use-expressions="true"
		access-denied-page="/WEB-INF/views/error/accessDenied.jsp">
<!--Spring Security 의 기본 로그인 페이지를 사용함. "/spring_security_login" -->
		<form-login/>
<!-- 기본 로그아웃 주소인 "j_spring_security_logout" 대신 커스텀 url 을 사용하기 위한 설정. -->
		<logout logout-url="/logout.do" logout-success-url="/"/>
		<anonymous/>
<!-- 리소스 패턴별 인증 및 인가 설정을 위한 엘리먼트 등록. -->
<!-- access 속성으로 ConfigAttribute 를 등록하게 되는데, 여기서 securityEL 을 쓰기 위해  -->
<!-- use-expressions="true" 설정이 필요함. -->
<!-- 이때 접근 가능 여부 결정은 AccessDesictionManager 내의 AccessDecisionVoter 중  -->
<!-- WebExpressionVoter 가 판단함. -->
		<intercept-url pattern="/board/boardView/*"
				access="hasAnyRole('ROLE_USER', 'ROLE_ADMIN')" />
		<intercept-url pattern="/board/*" access="hasRole('ROLE_ADMIN')" />
		<csrf disabled="true"/>
	</http>
</beans:beans>
```

- SecurityExpressionRoot 의 common built-in expression 의 종류.
  - Intercept-url 의 access 설정이나 security의 authentication 이나 authority 태그에서 사용할수 있는 ConfigAttribute expression 들.

Expression	Desription
hasRole([role]	현재 user가 특정 role 가지고 있는지 여부
hasAnyRole([role1,role2])	현재 user가 제시된 role 중 하나를 가지고 있는지 여부
hasAuthority([authority])	현재 user가 특정 권한을 가지고 있는지 여부
hasAnyAuthority([authority1,authority2])	현재 user가 제시된 권한중 하나를 가지고 있는지 여부
Principal	현재 user 의 principal(신원정보)
Authentication	시큐리티 컨텍스트에서 관리되는 현재 인증 객체
permitAll	모두에게 허가된 리소스
denyAll	접근 불가
isAnonymous()	익명사용자 인지 여부
isRememberMe()	RememberMe를 사용한 user 인지 여부
isAuthenticated()	익명사용자만 아니면 true
isFullyAuthenticated()	익명사용자도 아니고 RememberMe user 도 아닌 경우 true
hasPermission(Object target, Object permission)	현재 user가 target 객체에 대해 제시된 권한을 가졌다면 true
hasPermission(Object targetId, String targetType, Object permission)	현재 user가 제시된 target에 대해 제시된 권한을 가지면 true
hasIpAddress(String ip)	WebSecurityExpressionRoot 에서 추가됨.

- Role : Sprint Security는 일련의 AccessDecisionVoter 들이 "Configuration attribute"에 따라 접근허용 여부를 결정하는 투표기반 아키텍처를 갖고 있는데, 가장 일반적인 RoleVoter 는 "ROLE_" 접두어로 시작되는 속성을 찾아, 지정된 role 과 현재 사용자의 role이 일치하면 접근허용에 투표하는 단순 Voter 이다.
- Authority/Privilege/Permission : 자원에 대해 voter 에 의해 접근이 허용되기 위한 role 을 설정했다면, 해당 role 을 가진 유저는 자원에 대한 접근 authority를 갖는다고 표현한다.

5. security 인증객체 사용 혹은 인가에 따른 선택적 랜더링을 위한 view 수정

1) index.jsp 수정

```
<%@taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
<security:authorize access="isAuthenticated()">
	<security:authentication property="principal" var="user"/>
	<security:authentication property="details" var="details"/>
	<h2>${user.username }님
	<a href="${pageContext.request.contextPath }/logout.do">로그아웃</a></h2>
	<div style="border:1px solid black;">${user} <hr /> ${details}</div>
</security:authorize>
```

<security:authencation> 태그를 통해 security 인증객체에 접근 가능.
Principal : 신원정보(id를 비롯한 기타 로그인시 조회된 개인정보를 가진 객체)
Credential : 자격증명, Details : IP 나 세션아이디등의 정보를 가진 객체

2) 접근 거부시 사용할 view 작성(/WEB-INF/views/error/accessDenied.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
	<head>
	<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
	<title>Insert title here</title>
	</head>
	<body>
		<h4>접근 거부 상황(Access Denied)</h4>
		<h2>세션 스코프 상세</h2>
		<div style="border: 1px solid black; color: red;">
			<c:forEach items="${sessionScope }" var="entry">
				${entry.key }====${entry.value }<br />
			</c:forEach>
		</div>
		<h2>리퀘스트 스코프 상세</h2>
		<div style="border: 1px solid black; color: green;">
			<c:forEach items="${requestScope }" var="entry">
				${entry.key }====${entry.value }<br />
			</c:forEach>
		</div>
	</body>
</html>
```

3) 인증 정보 랜더링을 위한 tiles attribute 추가.

- /WEB-INF/views/template.jsp 수정

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles"%>
<html>
	<head>
		<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
		<title><tiles:getAsString name="title" /></title>
		<tiles:insertAttribute name="scriptTags" />
		<tiles:insertAttribute name="styleTags" />
	</head>
	<body>
		<div style="border: 1px solid black; background-color: aqua; font-size: 20px;">
			<tiles:insertAttribute name="header" />
		</div>
		<tiles:insertAttribute name="content" />
	</body>
</html>
```

- /WEB-INF/views/frags/header.jsp 작성 : 선택적 랜더링을 위해 security 태그 사용

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
<security:authorize access="!isAuthenticated()">
	<a href="<c:url value="/resources/loginForm.jsp"/>">로그인</a>
</security:authorize>
<security:authorize access="isAuthenticated()">
	<security:authentication property="principal" var="user"/>
	<c:url value="/logout.do" var="logout"/>
	<security:authorize access="isRememberMe()">
		RememberMe 로 다시 로그인한  ${user.user_name }님 <a href="${logout }">로그아웃</a>
		<security:authentication property="details.remoteAddress"/>
	</security:authorize>
	<security:authorize access="isFullyAuthenticated()">
		Full Authenticated ${user.user_name }님 <a href="${logout }">로그아웃</a>
		<security:authentication property="details.remoteAddress"/>
	</security:authorize>
</security:authorize>
```

- Programmatic logout (LogoutHandler 구현체 활용)

```
new CompositeLogoutHandler(
	new SecurityContextLogoutHandler(),
	new CookieClearingLogoutHandler("sampleCookie")
).logout(request, response, authentication);
```

- /src/main/resources/kr/co/sample/tiles/defs.xml 수정

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC
	"-//Apache Software Foundation//DTD Tiles Configuration 2.1//EN"
	"http://tiles.apache.org/dtds/tiles-config_2_1.dtd">
<tiles-definitions>
	<definition name="parent" template="/WEB-INF/views/template.jsp">
		<put-attribute name="title" value="Spring Board Tutorial" />
		<put-attribute name="header" value="/WEB-INF/views/frags/header.jsp" />
		<put-attribute name="scriptTags" value="/WEB-INF/views/frags/scripts.jsp" />
		<put-attribute name="styleTags" value="/WEB-INF/views/frags/styles.jsp" />
	</definition>

	<definition extends="parent" name="board/*">
		<put-attribute name="content" value="/WEB-INF/views/board/{1}.jsp" />
	</definition>
	<definition extends="parent" name="matrix/*">
		<put-attribute name="content" value="/WEB-INF/views/matrix/{1}.jsp" />
	</definition>
	<definition extends="parent" name="error/*">
		<put-attribute name="content" value="/WEB-INF/views/error/{1}.jsp" />
	</definition>
</tiles-definitions>
```

❑ Spring security 사용 예제 – DataBase 방식(xml config)

1. Spring Security 의존성 추가
2. web.xml 수정
   1) Root WebApplicationContext 에 security-context 설정 파일 등록
   2) DelegatingFilterProxy 등록 : filter-name="springSecurityFilterChain "
3. UserDetails 구현 : SecurityContext 내에서 Authentication 의 principal 로 캡슐화될 사용자 정보를 저장한 객체 정의
   SecurityContext 를 통해 사용자 정보를 조회할 시 다음과 같은 코드가 사용됨.
   - SecurityContextHolder.getContext().getAuthentication().getPrincipal() : 비인증 상태시 AnonymousAuthentication 객체 반환
   - HttpServletRequest.getUserPrincipal() : 비인증 상태시 null 반환
4. UserDetailsService 나 UserDetailsManager 구현 : 데이터베이스로부터 사용자 정보 조회하거나 사용자 정보 CRUD 를 수행할 객체 정의
5. Security-context 설정 파일 작성
   1) 인증 처리를 위한 AuthenticationManager 등록
      HttpSecurity(UsernamePasswordAuthenticationFilter),
      GlobalMethodSecurityConfiguration 에 의해 사용됨.
      (참고, session fixation attack 방어를 위해 SessionFixationProtectionStrategy의 전략에 따라 인증 성공시 새로운 세션을 생성하도록 기본 설정됨.
      session-management 엘리먼트의 session-fixation-protection 속성으로 전략 변경 – none, newSession, migrateSession, changeSessionId 등)
      ① 4.의 UserDetailsService 등록
      ② AuthenticationProvider 등록 : PasswordEncoder 를 통해 해쉬암호화 가능.
   2) 인가 처리를 위한 AccessDecisionManager 등록
      HttpSecurity(FilterSecurityInterceptor), GlobalMethodSecurityConfiguration 에 의해 사용되며, SecurityContext 자동 설정으로 필요한 accessDecisionManager 들이 자동 등록되나, 계층형 권한 관리 등이 필요할 경우, 명시적으로 등록하여 사용하며, 다음과 같은 종류가 있다.
      ① AffirmativeBased : 여러 AccessDecisionVoter 중 하나 이상 승인시 인가
      경우에 따라, WebExpressionVoter 기반의 AffirmativeBased#0 과 MethodInvocation를 지원하는 AffirmativeBased#1 이 자동 등록됨.
      ② UnanimousBased : 모든 voter 들이 만장일치로 승인시 인가
      ③ ConsensusBased : 다수결에 따라 인가
      AccessDecisionManager 에서 사용되는 AccessDecisionVoter 들의 종류
      a. RoleVoter : 사용자가 소유한 GrantedAuthority 가 RolePrefix 로 시작하는 ConfigAttribute 와 일치할 경우 승인.
      b. RoleHierarchyVoter : 권한별 계층 구조에 따라 승인 여부를 판단함.
      "ROLE_A > ROLE_B" 와 같은 형태로 계층 구조를 표현하며, ROLE_A 가 ROLE_B 의 권한을 포함한다는 의미임.

c. AuthenticatedVoter : IS_AUTHENTICATED_FULLY, IS_AUTHENTICATED _REMEMBER, IS_AUTHENTICATED_ANONYMOUS 등의 ConfigAttribute에 대한 승인 여부 판단.

d. Jsr250Voter : @RolesAllowed, @PermitAll, @DenyAll 등의 어노테이션 기반으로 승인 여부 판단. MethodInvocation 에서 주로 사용됨.

e. PreInvocationAuthorizationAdviceVoter : @PreAuthorize, @PostAuthorize 등의 어노테이션 기반으로 승인 여부 판단, MethodInvocation 에 대한 인터셉터에서 사용되므로, global-method-security 설정시 사용됨.

f. WebExpressionVoter : spEL로 빌트인 함수를 사용한 WebExpressionConfigAttribute 에 대해 승인 여부를 판단, FilterInvocation 에서 사용되므로, http 설정시 자동 등록됨.

3) 보호 자원과 접근 권한 정보를 조회하여 RequestMap을 생성할 FactoryBean 등록
requestMap : 접근 제어시 사용자의 요청과 매칭할 패턴을 가진 Matcher 를 키로, 매칭되는 자원에 대한 허가정보를 가진 ConfigAttribute 들을 값으로 가진 Map<RequestMatcher, List<ConfigAttribute>> 을 생성함.

4) 3) requestMap 을 주입하여 SecurityMetadataSource 등록
자원과 권한에 대한 정보를 관리하며, security context 내에서 보호된 자원과 자원에 대한 ConfigAttribute 의 식별을 담당함.

① FilterInvocationSecurityMetadataSource : 필터 기반의 접근 제어시 사용

a. DefaultFilterInvocationSecurityMetadataSource : RolePrefix 로 시작하는 권한명이나 "IS_AUTHENTICATED_FULLY" 등의 ConfigAttribute 식별

b. ExpressionBasedFilterInvocationMetadataSource : spEL 기반의 빌트인 함수를 사용하는 WebExpressionConfigAttribute 식별
WebSecurityExpressionHandler 객체가 필요함.

② MethodSecurityMetadataSource : AOP 방식으로 method 호출 기반의 접근 제어시 사용

a. SecuredAnnocationSecurityMetadataSource : @Secured

b. Jsr250MethodSecurityMetadataSource : @RolesAllowed, @PermitAll 등

c. PrePostAnnotationSecurityMetadataSource : @Pre/PostAuthorize

5) 1), 2), 4) 의 객체를 사용하여 AbstractSecurityInterceptor 등록
(ex. FilterSecurityInterceptor)
보호 자원에 대한 접근 제어를 수행하며, 다음과 같은 작업을 수행함

① SecurityContextHolder 로부터 Authentication 객체 획득

② 현재 요청 자원의 보호 여부를 확인

③ 보호된 자원인 경우,

a. Authentication.isAuthenticated() 가 false 를 반환하는 경우, 등록된 AuthenticationManager 를 이용하여 인증하고, SecurityContextHolder 의 Authentication 객체를 변경함.

b. AccessDecisionManger 를 이용하여 접근 권한 확인

c. RunAsManager 로 run-as-replacement 처리

d. 실구현체로 제어를 넘겨, 보호 자원에 대한 처리가 끝난 후, Interceptor-StatusToken이 반환하여 AbstractSecurityInterceptor 재 호출 여부를 결정
e. afterInvocation 에서 AbstractSecurityInterceptor 재호출
f. RunAsManager 가 Authentication 객체를 바꾼 경우, SecurityContextHolder 의 Authentication 객체를 rusAs 객체로 바꿈
g. AfterInvocationManager 가 정의된 경우 호출하여, 호출자로부터 반환된 보안 객체를 대체함.

③ 공개 자원인 경우, 실 구현체로 자원에 대한 처리 이후에 afterInvocation 을 호출하지 않음

④ 호출자로부터 반환된 객체와 함께 실구현체로 다시 제어를 반환하면, 실구현체에서 호출자에게 결과를 반환하거나 예외를 발생시킴.

6) 5) 의 Interceptor 객체를 사용하여, WebInvocationPrivilegeEvaluator 등록 <security:authorize url=""> 의 형태로 URL 기준의 선택적 랜더링에 사용됨.

7) http 보안 설정

- 속성정보

① auto-config : 폼로그인/아웃, HTTP Basic auth, 익명 로그인, remember-me, 서블릿 API 들과의 연계를 지원

② use-expressions : spEL 기반의 빌트인 함수를 이용한 접근 제어 지원

③ access-decision-manager-ref : AccessDecisionManager 객체

④ authentication-manager-ref : AuthenticationManager 객체

⑤ pattern : 해당 http 엘리먼트에 의해 필터 체인을 거치게될 URL 패턴

⑥ security="none" : 시큐리티 컨텍스트 예외 URL 패턴과 함께 사용

⑦ disable-url-rewriting : jsessionid 파라미터의 사용을 해제(기본 true)

- 하위 엘리먼트

① <http-basic> 설정 : 서블릿 컨테이너의 basic 인증방식 적용

② <form-login> 설정 : spring security 기본 폼 인증 페이지를 사용한 userpassword 기반의 인증 , <http-basic> 과 선택 적용.

a. login-page : 커스텀 로그인 페이지 주소 (기본, /login GET)
b. username-parameter : id 전송 파라미터명 (기본, username)
c. password-parameter : password 전송 파라미터명 (기본, password)
d. login-processing-url : 인증 처리 주소 (기본, /login POST)
e. default-target-url : 인증 성공시 리다이렉트 주소
f. always-use-default-target : target url 사용 여부
g. authentication-failure-url : 인증 실패시 리다이렉트 주소, 실패시 SPRING_SECURITY_LAST_EXCEPTION 속성으로 예외 및 메시지 확인 가능.
h. authentication-failure-ref : AuthenticationFailureHandler 객체
i. authentication-success-handler-ref : AuthenticationSuccessHandler 객체

③ <logout> 설정: LogoutHandler 자동 추가(SecurityContextLogoutHandler 기본 등록)

a. logout-url : 로그아웃 처리 주소 (기본, /logout)
b. logout-success-url : 로그아웃 후 리다이렉트 주소 (기본, /login?logout)
c. delete-cookies : 로그아웃 시 삭제할 쿠키명
d. invalidate-session : 세션 만료 여부
e. success-handler-ref : LogoutSuccessHandler 객체

④ <remember-me> 설정

a. remember-me-cookie : rememberMe 쿠키명(기본, remember-me)
b. remember-me-parameter : RememberMeFilter 동작을 위한 파라미터명
c. services-alias : 현재 RememberMeServices 객체 빈을 export할 이름
d. services-ref : Custom RememberMeServices 객체
e. data-source-ref : remember-me token 을 관리할 테이블(PERSISTENT_LOGINS)에 액세스할 dataSource
f. token-repository-ref : rememer-me token 객체에 대한 관리 전략을 커스터마이징한 객체, 기본 제공 객체는 다음과 같다
   - InMemoryTokenRepositoryImpl : 인메모리 방식의 토큰 관리 객체
   - JdbcTokenRepositoryImpl : PERSISTENT_LOGINS 테이블을 이용한 토큰 관리 객체로, 커스터마이징하여 테이블의 구조를 변경할 수 있음.
g. token-validity-seconds : remember-me cookie 의 유효기간 설정
h. use-secure-cookie : secure cookie 사용 여부
i. user-service-ref : UserDetailsService 객체

⑤ <custom-filter> 설정 : ref, before, position, after 등의 속성으로 필터 체인내의 위치 결정, **5) 번에서 등록한 securityInterceptor 를 적절한 필터링 위치에 배치함.**

⑥ <anonymous> 설정 : 익명 유저 설정

a. enabled : 허용 여부 (기본, true)
b. granted-authority : 익명 유저에게 부여할 역할 (기본, ROLE_ANONYMOUS)
c. username : 익명유저의 아이디 (기본, anonymousUser)

⑦ <session-management> 설정 : 세션의 동시성 제어, SessionManagementFilter 에 의해 사용됨.

- 속성종류

a. session-fixation-protection : 인증 성공시 session fixation attack 방어를 위한 정책 설정, SessionFixationProtectionStrategy 의 객체 사용.
   none(기존 세션을 그대로 사용, SFA 방어 없음),
   newSession(새로운 세션을 생성함),
   migrateSession(새로운 세션을 생성하고, 기존 세션의 속성을 복사함)
   changeSessionId(서블릿 3.1 스펙부터 지원되는 SFA 방어 전략을 이용하여, 기존 세션을 그대로 사용하되, 세션의 id 만 HttpServletRequest.changeSessionId() 를 사용하여 변경함)
   MSIE 에서 지원되지 않기때문에, none 설정이 필요함.

b. invalid-session-strategy-ref : InvalidSessionStrategy 객체
   - SimpleRedirectInvalidSessionStrategy : SessionManagementFilter 에 의해 유효하지 않은 세션이 감지될때, 특정 주소리 리다이렉션하는 전략

c. invalid-session-url : 세션이 invalid 할때 리다이렉트 주소

d. session-authentication-starategy-ref : 세션 인증시 사용할 전략으로, session-fixation-protection 속성의 값에 따라 자동 설정
   - NullAuthenticatedSessionStrategy : none
   - SessionFixationProtectionStrategy : migrateSession
   - ChangeSessionIdAuthenticationStrategy : changeSessionId
   - ConcurrentSessionControlAuthenticationStrategy : 동시성 제어시 사용
   - RegisterSessionAuthenticationStrategy : 동시성 제어시 사용
   - CompositeSessionAuthenticaitonStrategy : 복합 전략 적용
   - CsrfAuthenticationStrategy : CSRF 방어에 사용되며, 세션 인증시 기존 csrfToken 을 제거하고, 새로운 csrfToken 을 생성

e. session-authentication-error-url : 세션 인증 실패시 리다이렉트 주소로 미설정시 401이 발생하고, form-login 의 authentication-failure-url 로 연결됨.

- 하위 엘리먼트

a. concurrency-control : 동시 세션 제어에 사용됨
   max-session : 동시에 생성 가능한 세션수
   error-if-maximum-exceeded : 최대 세션치를 초과한 경우, 만료시킬 세션을 선택할때 사용되는 속성으로 미설정시 기존의 세션 중에서 만료가 되고, true로 설정한 경우 현재 인증을 요청한 세션에 대해 authentication error를 발생시킴.
   expired-url : 이미 만료된 세션을 사용한 경우 리다이렉트 주소

⑧ <headers> 설정 : 컨텐츠 보안과 관련된 헤더를 지원

-속성 종류

a. disabled : headers 엘리먼트를 비활성화, true 인 경우 헤더 설정을 위한 하위 엘리먼트를 사용할 수 없음.

b. default-disabled : headers 로 등록되는 기본 헤더들의 사용 여부

-기본 헤더와 설정 방법

a. Cache-Control : 캐시 제어를 위한 헤더
   <cache-control /> (no-cache, no-store, max-age=0, must-revalidate)

b. X-Content-Type-Options : nosniff – style 태그나 script 태그를 이용한 mime 기반의 공격을 방어하기 위한 헤더
   <content-type-options />

c. X-Frame-Options : [deny|sameorigin|allow-from URI] - frame 을 이용한 컨텐츠 랜더링을 제어하기 위한 헤더
   - deny(기본값) : frame 사용 금지
   - sameorigin : 동일 출처의 외부 프레임에서만 허용
   - allow-from URI : 특정 도메인에 대해 허용

ex) X-Frame-Options : ALLOW-FROM http://www.test.com
<frame-options policy="DENY|SAMEORIGIN|ALLOW-FROM"
from-parameter="from" strategy="whitelist|regexp"
value="domainURL|regular_expression" />
- strategy 속성에 따라 value 속성에 whilelist 나 정규식을 사용하면 iframe 등의 src 속성으로 from 파라미터가 전달되고, 매칭되는 경우, ALLOW-FROM 헤더값이 설정되고, 이외에는 DENY 가 설정됨.

d. X-XSS-Protections : 브라우저의 XSS 방어 옵션을 사용하기 위한 헤더
- 0 : XSS 필터링을 사용하지 않음.
- 1 : XSS 필터링을 사용함.
- 1; mode=block : XSS 필터링을 사용하고, XSS 공격이 감지되시 브라우저의 페이지 랜더링이 중지됨.

e. Strict-Transport-Security (HSTS): 브라우저의 https 사이트 목록에 해당 도메인을 추가하여 http 를 사용하는 경우에도 https 를 강제하도록 하기 위한 헤더
ex) Strict-Transport-Security : max-age=31536000; includeSubDomains

f. Content-Security-Policy : XSS 방어용 헤더로 script/style/img 등의 태그로 로딩되는 리소스를 제한할때 사용
- none : 어떤 리소스도 허용하지 않음.
- self : 동일 출처에서는 허용하나, 하위 도메인에서는 허용하지 않음.
- unsafe-inline : 인라인 스크립트나 인라인 스타일을 허용함.
- unsafe-eval : eval 과 같은 텍스트-스크립트 변환 매커니즘을 허용함.
상기 값들을 사용할 수 있는 지시자의 종류
defalut-src|script-src|frame-src|img-src|style-src 등
ex) Content-Security-Policy: default-src https: 'unsafe-eval' 'unsafe-inline';object-src 'none'
<content-security-policy policy-directives="default-src https: 'unsafe-eval' 'unsafe-inline';object-src 'none'" report-only="true"/>
- 모든 자원은 https 프로토콜로 요청하고, 인라인 코드는 허용하되, object 태그를 이용한 자원의 접근은 금지함.

⑨ <cors> 설정 : spring-webmvc 모듈의 mvc:cors 설정으로 대체할 수 있음.
Cross-Origin Request : 자바스크립트에서 제한하고 있는 XmlHttpRequest 의 SOP (Same Origin Policy)에 위반하여 요청의 origin 도메인과 target 도메인이 서로 다른 경우의 요청을 정의하며, 이러한 요청을 처리하기 위해 서버에는 CORS 설정이 추가되어야 함. CORS 설정이란 자원에 대한 접근을 허가할 수 있는 요청에 origin 에 대한 정보를 포함시키고, 응답에는 자원에 대한 접근을 허가할 cross-origin 집합에 대한 정보를 자원과 함께 전송할 수 있는 헤더들을 사용하는 구조임.
이 경우, 웹브라우저 혹은 클라이언트는 현재 서버에서 특정 MIME 이나 method (GET/POST 를 제외한) 가 지원되는지 여부를 확인하기 위해 preFlight 요청을 OPTIONS 메소드로 전송.

a. 요청헤더
   - Access-Control-Request-Method : 특정 method 를 대상으로 접근 요청
   - Access-Control-Request-Headers : 특정 header 를 대상으로 접근 요청

b. 응답헤더
   - Access-Control-Allow-Origin : 접근을 허가할 origin ex) *
   - Access-Control-Allow-Methods : 접근을 허가할 method list. ex) *
   - Access-Control-Allow-Headers : 접근을 허가할 header list ex) *
   - Access-Control-Max-Age : preflight 요청에 대해 저장된 캐시 만료시간

spring-webmvc 모듈의 xml config 에도 CORS 와 관련된 설정이 있음.

```
<cors>
        <mapping path="/**" allow-credentials="true" allowed-methods="*"
                                allowed-origins="*" allowed-headers="*" />
</cors>
```

⑩ <csrf> 설정

CSRF(Cross-Site-Request-Forgery) 공격 방어를 목적으로 POST 요청에 csrf 토큰이 포함되어있는지를 확인하기 위한 설정

a. disabled : csrf 토큰 확인 해제 속성. Security 4.x 부터 기본 false 로 설정됨.

b. token-repository-ref : csrf 토큰 생성/저장/조회를 위한 저장소.
   - HttpSessionCsrfTokenRepository(default)
   - CookieCsrfTokenRepository

c. csrf 토큰 전송 구조를 위해 security 태그 라이브러리 활용
   : csrf 토큰을 요청 파라미터와 헤더의 형태로 서버로 전달하기 위해 사용됨.
   - csrf 토큰 전송 방법 : X-CSRF-TOKEN(헤더), _csrf(파라미터)

```
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
<security:csrfMetaTags/>
<form method="post">
        <security:csrfInput/>
```

   - <security:csrfMetaTags /> : "_csrf_parameter", "_csrf_header", "_csrf" 세개의 meta 태그가 생성, AJAX 요청에서 헤더와 파라미터의 형태로 토큰을 전송하기 위해 활용됨.
   - <security:csrfInput /> : name="_csrf" input 태그를 생성해 줌.

```
<meta name="_csrf_parameter" content="_csrf" />
<meta name="_csrf_header" content="X-CSRF-TOKEN" />
<meta name="_csrf" content="0e162703-7e62-4fe1-8ac2-bba983e5bd2a" />
<script type="text/javascript">
        var csrfHeader = $("meta[name='_csrf_header']").attr("content");
        var csrfParameter = $("meta[name='_csrf_parameter']").attr("content");
        var csrfToken = $("meta[name='_csrf']").attr("content");
</script>
<form method="post">
        <input value="0e162703-7e62-4fe1-8ac2-bba983e5bd2a" name="_csrf"
                type="hidden" />
</form>
```

```java
package kr.co.sample.security;
/**
 * Principal(신원정보)에 다양한 정보를 담기위한  UserDetails 구현체
 */
public class CustomUserDetails implements UserDetails, Serializable {
	private String user_id; //사용자ID
	private String user_name; //사용자이름
	private String password; //패스워드
	private String email; //이메일
	private String reg_date; //등록일
	private boolean enabled; // 계정 활성화여부
	private List<GrantedAuthority> authorities;

	@Override
	public Collection<? extends GrantedAuthority> getAuthorities() {
		return authorities;
	}

	@Override
	public String getPassword() {
		return password;
	}

	@Override
	public String getUsername() {
		return user_id;
	}

	@Override
	public boolean isAccountNonExpired() {
		return enabled;
	}

	@Override
	public boolean isAccountNonLocked() {
		return enabled;
	}

	@Override
	public boolean isCredentialsNonExpired() {
		return enabled;
	}

	@Override
	public boolean isEnabled() {
		return enabled;
	}

	//getter, setter..
```

```
// 동시 세션을 제어하기 위해 UserDetails 내에서는 반드시 hashcode 와 equals 메소드를 명확하게 재정의해야 함.
// 그렇지 않은 경우, 동일 아이디로 다른 기기에서 접속시 동일 principal 임을 확인하지 못하는 경우가 발생함.
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result
		+ ((password == null) ? 0 : password.hashCode());
		result = prime * result + ((user_id == null) ? 0 : user_id.hashCode());
		result = prime * result
		+ ((user_name == null) ? 0 : user_name.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
		return true;
		if (obj == null)
		return false;
		if (getClass() != obj.getClass())
		return false;
		CustomUserDetails other = (CustomUserDetails) obj;
		if (password == null) {
		if (other.password != null)
		return false;
		} else if (!password.equals(other.password))
		return false;
		if (user_id == null) {
		if (other.user_id != null)
		return false;
		} else if (!user_id.equals(other.user_id))
		return false;
		if (user_name == null) {
		if (other.user_name != null)
		return false;
		} else if (!user_name.equals(other.user_name))
		return false;
		return true;
	}
}
```

```
/**
* 레거시 시스템의 사용자 객체의 형태를 변형하지 않고, Security 내의 Authtication 객체의 principal 객체로 활용할 wrapper.
*  UserDetailService 나 UserDetailManager 에서 UserDetails 의 구현체로 CustomUser 대신 wrapper 를 구현체로 사용함.
*/
public class CustomUserWrapper extends User{
	public CustomUserWrapper (CustomUser customUser) {
	super(customUser.getUsername(),  customUser.getPassword(), customUser.getAuthorities());
	this.user = user;
	}
	private CustomUser user;
	public CustomUser getUser(){
		return this.user;
	 }
}
```

```
package kr.co.sample.security;
/**
 * 데이터베이스의 사용자 신원정보와 권한 정보를 조회하기 위한 UserDetailsService 구현체
 */
public class CustomUserDetailsService extends JdbcDaoImpl{
	@Override
	public UserDetails loadUserByUsername(String username)
	throws UsernameNotFoundException {
		String message = null;
		List<UserDetails> users = loadUsersByUsername(username);
		if (users.size() == 0) {
			message = username+"에 해당하는 정보가 존재하지 않습니다.";
			logger.debug(message);
			throw new UsernameNotFoundException(message);
		 }
		 UserDetails user = users.get(0);
		 Set<GrantedAuthority> dbAuthsSet = new HashSet<GrantedAuthority>();
		 if (getEnableAuthorities()) {
		     dbAuthsSet.addAll(loadUserAuthorities(user.getUsername()));
		 }
		 if (getEnableGroups()) {
		     dbAuthsSet.addAll(loadGroupAuthorities(user.getUsername()));
		 }
		 List<GrantedAuthority> dbAuths = new ArrayList<GrantedAuthority>(dbAuthsSet);
		 addCustomAuthorities(user.getUsername(), dbAuths);
		 if (dbAuths.size() == 0) {
		 message = username + "에 부여된 권한이 없습니다. ";
		     logger.debug(message);
		     throw new UsernameNotFoundException(message);
		 }
		 ((CustomUserDetails)user).setAuthorities(dbAuths);
		 return user;
		// wrapper 사용시
		// return new CustomUserWrapper(customUser);
	}
	/*
	 * userByUsername 쿼리를 사용해 데이터베이스내의 사용자 정보를 조회하는 메소드
	 */
	@Override
	protected List<UserDetails> loadUsersByUsername(String username) {
	JdbcTemplate template = getJdbcTemplate();
		return template.query(getUsersByUsernameQuery(), new String[]{username}, new
				RowMapper<UserDetails>(){
			@Override
			public UserDetails mapRow(ResultSet rs, int rowNum)
			throws SQLException {
				CustomUserDetails userDetails = new CustomUserDetails();
				userDetails.setUser_id(rs.getString("USER_ID"));
				userDetails.setUser_name(rs.getString("USER_NAME"));
				userDetails.setPassword(rs.getString("PASSWORD"));
				userDetails.setEnabled(rs.getBoolean("ENABLED"));
				userDetails.setEmail(rs.getString("EMAIL"));
				userDetails.setReg_date(rs.getString("REG_DATE"));
				userDetails.setAuthorities(AuthorityUtils.NO_AUTHORITIES);
				return userDetails;
			}
		});
	}
```

```
	/*
	 * authoritiesByUsername 쿼리를 통해 사용자의 권한 정보를 조회하는 메소드
	 */
	@Override
	protected List<GrantedAuthority> loadUserAuthorities(String username) {
		JdbcTemplate template = getJdbcTemplate();
		return template.query(getAuthoritiesByUsernameQuery(), new String[]{username}, new
				RowMapper<GrantedAuthority>(){
			@Override
			public GrantedAuthority mapRow(ResultSet rs, int rowNum)
			throws SQLException {
				String role = rs.getString("AUTHORITY");
				return new SimpleGrantedAuthority(role);
			}
		});
	}
}
```

```
package kr.co.sample.security.hierarchicalRole;
/**
 * 데이터베이스의 권한 계층 구조를 조회하여 계층구조 문자열을 생성하는 FactoryBean.<br />
 * 권한계층구조 문자열의 형태는 다음과 같다.
 * <pre>
 * ROLE_ADMIN > ROLE_USER
 * ROLE_USER > ROLE_RESTRICTED
 * ROLE_RESTRICTED > IS_AUTHENTICATED_FULLY
 * </pre>
 */
public class RoleHierarchyStringFactoryBean implements FactoryBean<String> {
	private DataSource dataSource;
	@Required
	public void setDataSource(DataSource dataSource) {
		this.dataSource = dataSource;
	}

	String ROLE_HIERARCHY_QUERY =
				" SELECT CHILD_ROLE || ' > ' || PARENT_ROLE  HIERARYSTRING "
			   +" FROM ROLES_HIERARCHY                    "
			   +" START WITH CHILD_ROLE = 'ROLE_ADMIN'     "
			   +" CONNECT BY PRIOR PARENT_ROLE = CHILD_ROLE ";

	@Override
	public String getObject() throws Exception {
		NamedParameterJdbcTemplate template = new NamedParameterJdbcTemplate(dataSource);
		List<Map<String, Object>> hierarchy = template.queryForList(ROLE_HIERARCHY_QUERY, new
				HashMap<String, String>());
		StringBuffer buffer = new StringBuffer();
		for(Map<String, Object> record : hierarchy){
			buffer.append(record.get("HIERARYSTRING")+"\n");
		}
		return buffer.toString();
	}
```

```
	@Override
	public Class<?> getObjectType() {
	return String.class;
	}

	@Override
	public boolean isSingleton() {
	return true;
	}
}
```

```
package kr.co.sample.security;
/**
 * 데이터베이스로부터 보호된 자원(Request URI)-접근권한에 대한 정보를 조회하여 <br />
 * matcher-config attribute 의 형태로 관리하는 map 을 생성할 FactoryBean.
 */
public class RequestMapFactoryBean implements FactoryBean<LinkedHashMap<RequestMatcher,
    List<ConfigAttribute>>>{

	private DataSource dataSource;

	@Required
	public void setDataSource(DataSource dataSource) {
	    this.dataSource = dataSource;
	}

	LinkedHashMap<RequestMatcher, List<ConfigAttribute>> requestMap;

	@PostConstruct
	public void init(){
		requestMap = new LinkedHashMap<RequestMatcher, List<ConfigAttribute>>();
		NamedParameterJdbcTemplate template = new NamedParameterJdbcTemplate(dataSource);
		StringBuffer sql = new StringBuffer("");
		sql.append(" SELECT B.RESOURCE_PATTERN pattern, A.AUTHORITY  authority       ");
		sql.append(" FROM SECURED_RESOURCES_ROLE A INNER JOIN SECURED_RESOURCES B ON ");
		sql.append(" (A.RESOURCE_ID = B.RESOURCE_ID)                                 ");
		List<Map<String, Object>> resourcesList = template.queryForList(sql.toString(), new
                                    MapSqlParameterSource());
		if(resourcesList.size() > 0){
			for(Map<String, Object> mappingInfo : resourcesList){
				String pattern = (String) mappingInfo.get("pattern");
				String authority = (String) mappingInfo.get("authority");
				RequestMatcher matcher = new AntPathRequestMatcher(pattern);
				List<ConfigAttribute> attributes = null;
				attributes = requestMap.get(matcher);
				if(attributes==null){
				    attributes = new LinkedList<ConfigAttribute>();
				}
				attributes.add(new SecurityConfig(authority));
				requestMap.put(matcher, attributes);
			}
		}
	}
```

```
	@Override
	public LinkedHashMap<RequestMatcher, List<ConfigAttribute>> getObject()
	throws Exception {
		if(requestMap==null){
		    init();
		}
		return requestMap;
	}
	@Override
	public Class<?> getObjectType() {
	    return LinkedHashMap.class;
	}
	@Override
	public boolean isSingleton() {
	    return true;
	}
}
```

• In memory 설정 파일을 securityContextInMemory.xml 로 변경하고, securityConext.xml 신규 작성.

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

<!-- JdbcUserDetailsManager 를 사용해 데이터베이스 인증정보를 바탕으로 인증 여부를 확인. -->
<!-- Authentication.principal(신원정보)로 UserDetails 가 사용됨. -->
<!-- <jdbc-user-service data-source-ref="dataSource" -->
<!-- id="jdbcUserService" -->
<!-- users-by-username-query="SELECT USER_ID USERNAME, PASSWORD, ENABLED FROM USERS WHERE USER_ID
= ?" -->
<!-- authorities-by-username-query="SELECT USER_ID USERNAME, AUTHORITY FROM AUTHORITIES WHERE USER_ID
= ?" -->
<!-- role-prefix="" -->
<!-- /> -->

<!-- principal 내에 기본 인증 정보 이외의 정보를 세팅하고 싶다면, 별도의 UserDetails 와  -->
<!-- UserDetailsService 를 구현 및 등록해 사용할수 있음. -->
	<beans:bean id="userDetailsService" class="kr.co.sample.security.CustomUserDetailsService"
		p:dataSource-ref="dataSource"
		p:enableAuthorities="true"
		p:usersByUsernameQuery="SELECT * FROM USERS WHERE USER_ID = ?"
		p:authoritiesByUsernameQuery="SELECT * FROM AUTHORITIES WHERE USER_ID = ?"
	/>

	<authentication-manager id="authenticationManager">
		<authentication-provider user-service-ref="userDetailsService" />
	</authentication-manager>
```

```
<!-- 데이터베이스내의 권한 계층 정보를 조회하여 권한 계층 문자열을 생성하는 FactoryBean  -->
        <beans:bean id="hierarchyString"
                class="kr.co.sample.security.hierarchicalRole.RoleHierarchyStringFactoryBean"
                p:dataSource-ref="dataSource"
        />

<!-- 권한 계층 문자열에 따라 특정 계정의 유효 권한 정보를 추출하는 메소드. -->
        <beans:bean id="roleHierarchy"
                class="org.springframework.security.access.hierarchicalroles.RoleHierarchyImpl"
                p:hierarchy-ref="hierarchyString"
        />

<!-- AffirmativeBased 의 access decision 정책은 여러개의 AccessDecisionVoter 중 단 하나라도 -->
<!-- ACCESS_DENIED 를 던지면 접근 거부, ACCESS_ABSTAIN 과 하나 이상의 ACCESS_GRANTED로 구성된 경우 접근 허용.   -->
<!-- 다수결의 원칙에 따라 모든 Voter 의 투표결과를 합산하여 최종적으로 ACCESS_SUBTAIN(0) 보다 큰 값일 지닌 경우 접근 허용.
-->
<!-- 모두 ACCESS_ABSTAIN 을 던졌을 경우, 접근 거부를 위해 allowIfAllabstainDecisions="false" 를 설정. -->
        <beans:bean id="accessDecisionManager"
                class="org.springframework.security.access.vote.AffirmativeBased"
                p:allowIfAllAbstainDecisions="false" >
                <!--SecurityExpressionRoot 의 securityEL 을 사용한 접근제어를 위해 필요한 AccessDecisionVoter -->
                <beans:constructor-arg name="decisionVoters">
                        <beans:list>
                                <!-- IS_AUTHENTICATED_ANONYMOUS, IS_REMEMBERME 등의 접근에 대한 결정자 -->
                                <beans:bean
                                class="org.springframework.security.access.vote.AuthenticatedVoter" />
                                <!-- 계층형 권한 처리를 위한 결정자 -->
                                <beans:bean
                                class="org.springframework.security.access.vote.RoleHierarchyVoter"
                                c:roleHierarchy-ref="roleHierarchy"
                                />
                                <!-- SpringExpressionRoot 의 securityEL(built-in) 메소드를 사용하는 결정자 -->
                                <beans:bean
                                class="org.springframework.security.web.access.expression.WebExpressionVoter" />
                        </beans:list>
                </beans:constructor-arg>
        </beans:bean>

<!-- 데이터베이스내의 보호된 리소스-권한 매핑 정보를 조회한후 관리할 map 을 생성하는 factory bean -->
        <beans:bean id="requestMap" class="kr.co.sample.security.RequestMapFactoryBean"
                p:dataSource-ref="dataSource"
        />

<!-- 데이터베이스 기반으로 현재 시점의 url 보호자원-권한의 매핑 정보를 runtimne 에 동적으로 변경 반영하기 위해 관리되는
metadatasource -->
        <beans:bean id="filterInvocationSecurityMetadataSource"
                class="org.springframework.security.web.access.intercept.DefaultFilterInvocationSecurityMetadataSou
               rce"
                c:requestMap-ref="requestMap"
        />

<!-- 리퀘스트를 가로챈 interceptor 가 Http 리소스에 대한 보호를 수행하기 위해 요청url-권한에 대한 정보를 가진
SecurityMetadataSource 가 필요함 -->
        <beans:bean id="filterSecurityInterceptor"
                class="org.springframework.security.web.access.intercept.FilterSecurityInterceptor"
                p:authenticationManager-ref="authenticationManager"
                p:accessDecisionManager-ref="accessDecisionManager"
                p:securityMetadataSource-ref="filterInvocationSecurityMetadataSource"
        />
```

```
	<http security="none" pattern="/resources/**" />
	<http auto-config="true"
		use-expressions="true"
		access-denied-page="/WEB-INF/views/error/accessDenied.jsp"
		authentication-manager-ref="authenticationManager">
	<!-- 커스텀 로그인페이지와 커스텀 인증 파라미터를 사용하기 위한 설정 -->
		<form-login
			login-page="/resources/loginForm.jsp"
			authentication-failure-url="/resources/loginForm.jsp?error=true"
			username-parameter="user_id"
			password-parameter="password"
			login-processing-url="/login.do"
			default-target-url="/"/>
		<!-- 커스텀 로그아웃페이지 설정 -->
		<logout logout-url="/logout.do" logout-success-url="/" delete-cookies="JSESSIONID"/>
		<!-- 세션 종료 후에도 인증 상태를 유지하기 위해 쿠키등으로 rememberMe 기능 이용을 위한 설정.
		-->
		<remember-me remember-me-parameter="rememberMe"
				token-validity-seconds="#{60*60*24*7}" />
		<anonymous/>
		<!-- 별도의 custom filterSecurityIntercptor 를 등록한 경우 반드시, -->
		<!-- SpringSecurityFilterChain 내에서 필터링 순서를 결정해야함. -->
		<custom-filter ref="filterSecurityInterceptor" before="FILTER_SECURITY_INTERCEPTOR" />
		<session-management >
		<!-- 특정 아이디에 대해 동시 생성가능한 세션수 지정.  -->
		<!-- 동시 세션 기능을 지원하기 위해 HttpSessionEventPublisher 를 리스너로 등록함.  -->
		<!-- error-if-maximum-exceeded="true" 를 사용하는 경우 최대허용수를 넘는 세션에서
		expire발생. -->
		<!-- false(default) 인 경우는 기본적으로 가장 오래된 세션이 expire됨. -->
		<!-- 동시 세션을 지원하기 위해서는 반드시 UserDetails의 hashcode, equals 를 명확하게
		정의하여 객체 상태 비교가 가능토록 해야함. -->
			<concurrency-control max-sessions="1"
				expired-url="/resources/loginForm.jsp?error=true" />
		</session-management>
	</http>
</beans:beans>
```

•동시 세션 지원을 위한 리스너 등록(web.xml)

```
<listener>
<listener-class>org.springframework.security.web.session.HttpSessionEventPublisher</listener-class>
</listener>
```

Spring Security 설정 파일 작성 과정.

1. 데이터베이스로 관리되는 인증정보 조회하여 인증정보를 가진 userDetails 생성하기 위한 jdbc-user-detailsService 등록
2. 1번의 userDetailsService 를 통해 인증정보를 생성하는 인증 관리자 등록(authenticationManager) 등록
3. 권한이 계층 구조를 갖는 경우(계층구조 표현 문자열을 생성하는 빈 등)
4. 인가 정보에 따른 접근결정 등록(accessDecisionManager) 등록(2번의 권한계층표현문등을 사용하는 각 Decision Voter 들의 사용)
5. 자원에 대한 인가정보를 갖고있는 Map<RequestMatcher, List<ConfigAttribute>> 생성 빈 등록
6. 4번의 인가정보를 바탕으로 보안설정들을 관리하는 저장소인 FilterInvocationSecurityMetadataSource 등록
7. 1번의 인증관리자와 3번의 접근결정자 5번의 인가정보 저장소를 바탕으로 SecurityFilter 사이에 리퀘스트를 가로채기위한 interceptor 등록
8. SpringSecurityFilterChain 을 구성하기 위한 http 설정.
9. 인증 정보 획득을 위한 폼을 구성하기위해 http form-login 과 logout 의 설정.
10. 등록된 필터체인 내에 6번의 interceptor 를 등록하기 위해 custom-filter등

▪ **CSRF(Cross Site Request Forgery) 공격 방어**를 위한 설정 : CSRF 토큰 데이터를 사용.

✓폼 화면 구성시 hidden 태그로 _csrf.token 값을 기억해두고, 동시에 세션에 해당 토큰을 기억해둠.

✓폼 전송시 _csrf.parameterName 으로 전송된 값이 세션에 저장된 값과 동일한지 여부로 위변조 체크.

1) CSRF 전략 설정 : SecurityContext 내에 CSRFTokenRepository 생성 및 유지(securityContext.xml)

```
<!-- CSRF 방어 단계 -->
<!-- 1) 세션 내에 CSRFTokenRepository 로 CSRFToken 유지를 위한 설정 추가("securityContext.xml") -->
<!-- 2) CSRFToken 을 폼태그에 추가("view jsp form tag") -->
<!-- 특정 리퀘스트에 대해서만 CSRF 토큰을 사용하는 경우, 패턴매쳐를 사용해 리퀘스트 매핑 설정.  -->
        <csrf request-matcher-ref="antPathRequestMatcher" />
        </http>
        <beans:bean id="antPathRequestMatcher"
                class="org.springframework.security.web.util.matcher.AntPathRequestMatcher"
                c:pattern="/**/*update*/**"
                c:httpMethod="POST"
                c:caseSensitive="false" />
</beans:beans>
```

2) 폼태그 내의 hidden 태그로 csrf 토큰 추가.(boardEdit.jsp or boardForm.jsp)

-Spring form 태그를 사용하는 경우 자동으로 csrf 토큰용 hidden 태그가 추가됨.

```
<form:form action="${pageContext.request.contextPath }/board/boardInsert.do"
                method="post" enctype="multipart/form-data" modelAttribute="board">
```

-Multipart request 인 경우, CSRF 토큰 사용 방법

a)web.xml 에 MultipartFilter(servlet 2.3 이하 버전에서 multipart request 처리하던 방식) 등록

```
<!-- 상위 WebApplicationContext 내에 등록된 MultipartResolver 를 사용해 multipart request를 파싱하기위한
필터, filterMultipartResolver 라는 빈을 자동으로 찾도록 설정되어있음. -->
<filter>
        <filter-name>MultipartFilter</filter-name>
        <filter-class>org.springframework.web.multipart.support.MultipartFilter</filter-class>
</filter>
<filter-mapping>
        <filter-name>MultipartFilter</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

b) 상위 컨텍스트내에 multipartResolver 를 등록해 관리함(ApplicationContext.xml).

```
<!-- multipart 업로드 데이터의 경우 MultipartFilter 를 사용해 springSecurityFilterChain 이전에 멀티파트
파싱이 이뤄져야함. 이 경우 실제 파싱 작업을 수행하는 multipartResolver 를 사용할수 있어야 하고, 이를 위해서
상위 컨텍스트에 multipartResolver 를 등록해야 함. -->
<bean id="filterMultipartResolver"
        class="org.springframework.web.multipart.commons.CommonsMultipartResolver"
        p:defaultEncoding="#{appInfo.pdsDefaultEncoding}"
        p:maxInMemorySize="#{appInfo.pdsMaxInMemorySize}"
        p:maxUploadSize="#{appInfo.pdsMaxUploadSize}"
        p:uploadTempDir="file:#{appInfo.pdsTempDir}" />
```

```
<form action="${pageContext.request.contextPath }/board/boardUpdate.do"
        method="post" enctype="multipart/form-data" id="boardUpdateForm">
        <input type="hidden" name="bo_no" value="${board.bo_no }" />
        <input type="hidden" name="${_csrf.parameterName }" value="${_csrf.token }" />
```

3) 컨트롤러에서 Authentication 객체 받기.(ex. BoardInsertController)

```
@Controller
public class BoardInsertController{
        @PostMapping("/boardInsert")
        public String insert( @AuthenticationPrincipal CustomUserDetails user , BoardVO board) {
        // @AuthenticationPrincipal("user") CustomUser user // wrapper 활용시
```

: 위의 예와 같이 컨트롤러 단에서 @AuthenticationPrincipal (since 4.0) 을 사용하여 인증 객체를 받을 수 있음.
단, DispatcherServlet 컨텍스트에 어노테이션 해결을 위한 ArgumentResolver 를 추가해야 함.

```
<annotation-driven>
        <argument-resolvers>
        <beans:bean
       class="org.springframework.security.web.method.annotation.AuthenticationPrincipalArgumentResolver" />
        </argument-resolvers>
</annotation-driven>
```

4.x 이전 버전에서는 아래와 같이 인증 객체를 받을 수 있음.

```
@PostMapping("/boardInsert")
public String insert( Authentication  authentication, BoardVO board) {
    CustomUserDetails user =  (CustomUserDetails)authentication.getPrincipal();
```

- **CSRF 방어 설정시 주의**! : 로그아웃을 위한 리퀘스트는 반드시 POST 방식을 사용해야함.
(/WEB-INF/views/frags/header.jsp 수정)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
<security:csrfMetaTags />
<script>
       function logoutFunc(event){
       event.returnValue=false;
       var frm = document.getElementById("logoutFrm");
       frm.submit();
}
</script>
<c:url value='/logout.do' var="logoutUrl"/>
<form id="logoutFrm" action="${logoutUrl }" method="post">
     <security:csrfInput />
</form>
<security:authorize access="!isAuthenticated()">
              <a href="<c:url value="/resources/loginForm.jsp"/>">로그인</a>
</security:authorize>
<security:authorize access="isAuthenticated()">
       <security:authentication property="principal" var="user"/>
       <c:url value="/logout.do" var="logout"/>
       <security:authorize access="isRememberMe()">
              RememberMe 로 다시 로그인한  ${user.user_name }님
       </security:authorize>
       <security:authorize access="isFullyAuthenticated()">
              Full Authenticated ${user.user_name }님
       </security:authorize>
       <a onclick="logoutFunc(event)"  href="${logout }">로그아웃</a>
       접속 IP :<security:authentication property="details.remoteAddress"/>
</security:authorize>
```

▪ XSS(Cross Site Script) 공격 방어를 위한 설정

1) X-XSS-protection header 의 사용 (securityContext.xml)

```
        <headers>
<!-- 비표준 헤더로 브라우져 자체의 XSS 필터링 기능을 사용하기 위한 헤더의 추가,  -->
<!-- 그러나 이 헤더가 스크립트 공격 자체를 방어하는 기능을 갖는것은 아님. -->
        <xss-protection block="true" enabled="true"/>
        </headers>
</http>
```

2) 파라미터로 전송되는 저장형 XSS 공격에 대한 방어를 위해 필터 사용.

a) script 를 비롯한 몇몇 공격 가능 형 태그의 escape 를 위한 유틸리티 작성.

```
package kr.co.sample.util;
// imports
public class CustomStringUtils {
        private static Logger log = LoggerFactory.getLogger(CustomStringUtils.class);

        /**
         * XSS공격의 일환으로 관리자가 아닌 사용자에 의해 악성 스크립트가 웹페이지에 삽입되는 경우를
         * 방어하기 위해 script 를 비롯한 실행 코드 삽입형 태그를 escape 시키기 위한 메소드.
         * @param original escape 전의 script|object|embed|applet 태그를 가진 원본
         * @return escape 된 문자열
         */
        public static String escapeForXSSDefend(String original){
                if(original!=null){
                        String regex = "</?(script|object|embed|applet)\\s*[\\w=/'\"]*\\s*>";
                        Pattern pattern = Pattern.compile(regex, Pattern.CASE_INSENSITIVE);
                        Matcher matcher = pattern.matcher(original);
                        StringBuffer sb = new StringBuffer("");
                        while(matcher.find()){
                                String replacement = matcher.group().replaceAll("<", "&lt;");
                                replacement = replacement.replaceAll(">", "&gt;");
                                matcher.appendReplacement(sb, replacement);
                        }
                        matcher.appendTail(sb);
                        log.debug("escape execute : [{}]", sb.toString());
                        return sb.toString();
                }else{
                        return null;
                }
        }
}
```

b) 파라미터를 escape 하기 위한 필터 작성

```
package kr.co.sample.web.filter;
// imports
/**
 * XSS 공격 발생 가능한 리퀘스트에 대해 악성 스크립트를 무력화시키기위한 필터.
 * 스크립트 포함 가능성이 있는 모든 클라이언트측 전송 데이터를 escape 시킴.
 */
public class XSSDefendFilter implements Filter{
	Logger log = LoggerFactory.getLogger(XSSDefendFilter.class);
	@Override
	public void init(FilterConfig filterConfig) throws ServletException {
		log.debug(this.getClass().getSimpleName()+"'s instance construct!");
	}
	@Override
	public void doFilter(ServletRequest request, ServletResponse response,
		FilterChain chain) throws IOException, ServletException {
		HttpServletRequest req = WebUtils.getNativeRequest(request, HttpServletRequest.class);
		Map<String, String[]> parameterMap = req.getParameterMap();
		for(Entry<String, String[]> entry : parameterMap.entrySet()){
			String[] paramValues = entry.getValue();
			if(paramValues!=null){
				for(int i=0; i<paramValues.length; i++){
					paramValues[i] =
					CustomStringUtils.escapeForXSSDefend(paramValues[i]);
				}
			}
		}
		chain.doFilter(request, response);
	}
	@Override
	public void destroy() {
		log.debug(this.getClass().getSimpleName()+"'s instance destroy!");
	}
}
```

c) 상위 필터의 등록 및 매핑 (스프링 컨텍스트 내에 등록하기위해 DelegateFilterProxy를 사용)

- securityContext.xml 에 등록

```
<beans:bean id="XSSDefendFilter" class="kr.co.sample.web.filter.XSSDefendFilter" />
```

- web.xml 에서 매핑(**주의! MultipartFilter 와 springSecurityFilterChain 사이에 등록**)

```
<filter>
	<filter-name>xssFilter</filter-name>
	<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
	<init-param>
		<param-name>targetBeanName</param-name>
		<param-value>XSSDefendFilter</param-value>
	</init-param>
</filter>
<filter-mapping>
	<filter-name>xssFilter</filter-name>
	<url-pattern>/*</url-pattern>
</filter-mapping>
```

3) Lucy 필터 사용 : https://github.com/naver/lucy-xss-servlet-filter

a) maven dependency 등록.

```
<dependency>
        <groupId>com.navercorp.lucy</groupId>
        <artifactId>lucy-xss-servlet</artifactId>
        <version>2.0.0</version>
</dependency>
```

b) Lucy servlet 필터 등록 및 매핑 (characterEncodingFilter 다음에 등록)

```
<filter>
        <filter-name>xssEscapeServletFilter</filter-name>
        <filter-
class>com.navercorp.lucy.security.xss.servletfilter.XssEscapeServletFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>xssEscapeServletFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

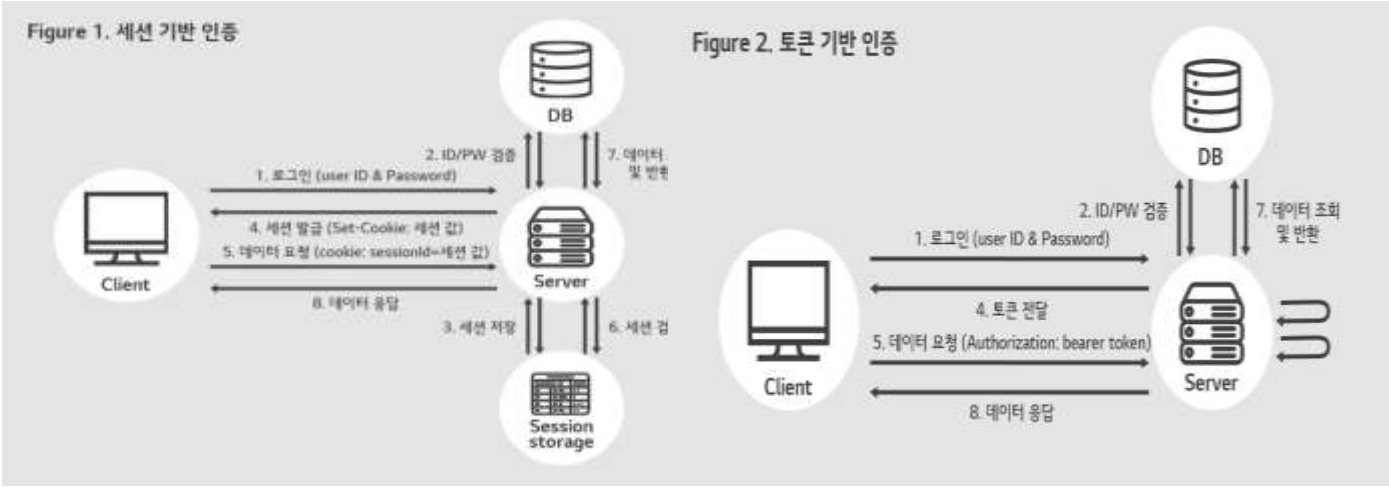
c) classpath 에 lucy-xss-servlet-filter-rule.xml 작성 : https://github.com/naver/lucy-xss-servlet-filter 참고

d) classpath 에 화이트리스트 설정 파일 작성 : https://github.com/naver/lucy-xss-filter/tree/master/conf 참고

▪ JWT (Json Web Token) 기반 인증

• 사용자의 인증을 위한 정보를 서명한 토큰을 기반으로 인증을 처리하므로, 서버에 사용자 정보를 저장하지 않고, 전달받은 토큰의 서명과 데이터를 검증하는 것만으로 인증이 가능한 방식.

• 세션 기반 인증 vs 토큰 기반 인증



✓ Self-Contained : 클라이언트로 전송되는 세션값(아이디)에는 정보가 포함되지 않으나, 토큰에는 정보가 포함되어 길이가 더 길어짐.

✓ Stateless : 세션 기반 인증은 서버 내에 세션에 대한 상태 정보를 저장하지만, 토큰 방식은 저장되는 데이터가 없는 무상태성 방식으로 서버 확장에 제약이 없고, 토큰의 유효성 검증만으로 요청을 인증하기 때문에 멀티 플랫폼 운영이 쉬어짐.

• JSON 으로 데이터을 포현하며 토큰에 포함된 데이터로 인가(Authorites)와 사용자 정보(Claims) 교환

✓비밀키나 공개/개인 키로 데이터를 서명하여 보호함.

✓ 전자서명으로 데이터의 송신자를 확인하고, 데이터의 무결성을 보장함.

1. JWT 구조
   1) Header :
      ① typ : 토큰의 타입
      ② alg : 서명 알고리즘

```
{
      "typ": "JWT",
      "alg": "HS256"
}
```

   2) Payload : 토큰에 담을 데이터(Claim)들의 표현 영역으로 key/value 형태를 가짐.
      ① Claim 의 종류
         A. Registered Claims (선택) : 서비스 필요 정보가 아닌 기 등록된 claim
         B. Public Claims (사용자 정의) : 충돌 방지를 위해 주로 URI 를 키로 사용함.
         C. Private Claims : 토큰 사용자(서버-클라이언트) 간 정보 공유를 위해 사용함.
            a. iss(issuer) : 토큰 발행 주체
            b. sub(subject) : 토큰의 제목
            c. aud(audience) : 토큰의 발급 대상자(사용자)
            d. exp(expiration time) : 토큰 만료 시간
            e. nbf(not before) : 토큰 사용 시작 시간
            f. iat(issued at) : 토큰 발행일
            g. jti(JWT id) : 토큰에 대한 고유 식별자

```
{
      "username":"testUser",
      "role":"USER",
      "iat":1660784568092,
      "exp":1660786411232,
      "issu":"VariousTechPractice"
}
```

   3) Signature : 헤더와 페이로드를 서명한 값으로 비밀키를 이용해 base64url 로 인코딩한 헤더와 페이로드를 alg 로 설정한 해싱 알고리즘으로 서명하여 토큰의 유효성 검증에 사용함.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  scretekey)
```

2. JWT 취약점
    1) 정보노출 : 페이로드의 데이터는 암호화하지 않고, base64 로 인코딩한 값이므로 민감 정보를 포함하지 않아야함 -> HTTPS 채널 사용.
    2) None 알고리즘 : alg="none" 인 경우 서명 값 검증을 수행하지 않으므로, 작성자 확인도 데이터 무결성도 보장할 수 없음 -> None 알고리즘 불허 설정.
    3) 서명 키 관리 : 토큰 기반 인증은 토큰을 저장하고 인증하는 방식이 아닌, 토큰의 헤더와 데이터 서명을 이용해 검증을 수행하므로, 서명 키가 노출된 경우, 위변조를 통한 인증 우회 권한 탈취 등이 가능해짐.
    4) 토큰 유효기간 및 파기 : 토큰 만료시점까지 사용이 가능하므로 별도의 토큰 파기 절차가 필요함.
    -> access/refresh token 구조를 활용하되 만료시간을 적절히 설정하고, refresh 토큰은 access 토큰 발급에만 사용하도록 제한함.
3. Spring Security + JWT 기반 인증 예제