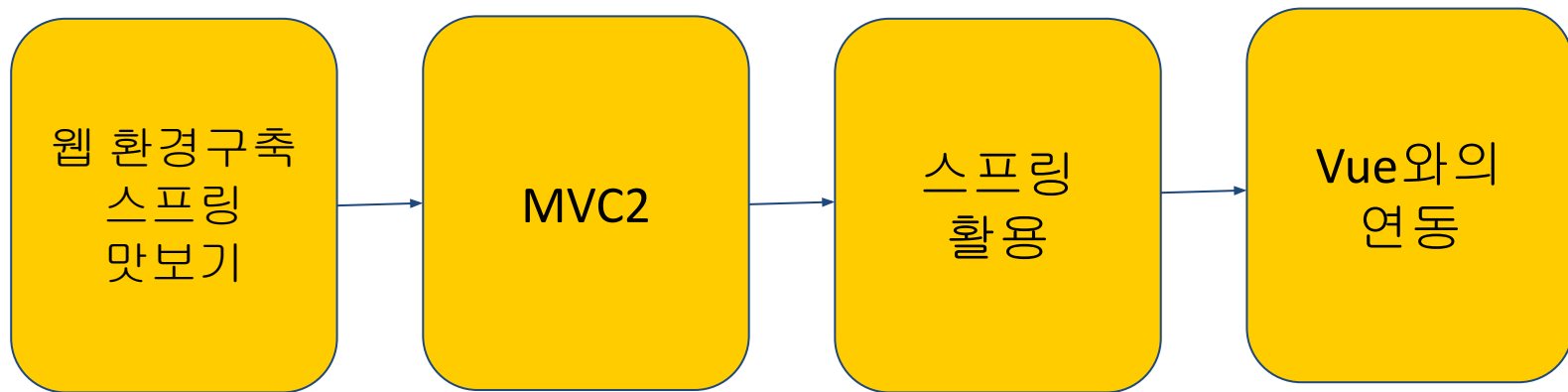


2024년 상반기 K-디지털 트레이닝

SPRING07 - Security

[KB] IT's Your Life



Please sign in

Username

Password

Sign in

환영합니다.

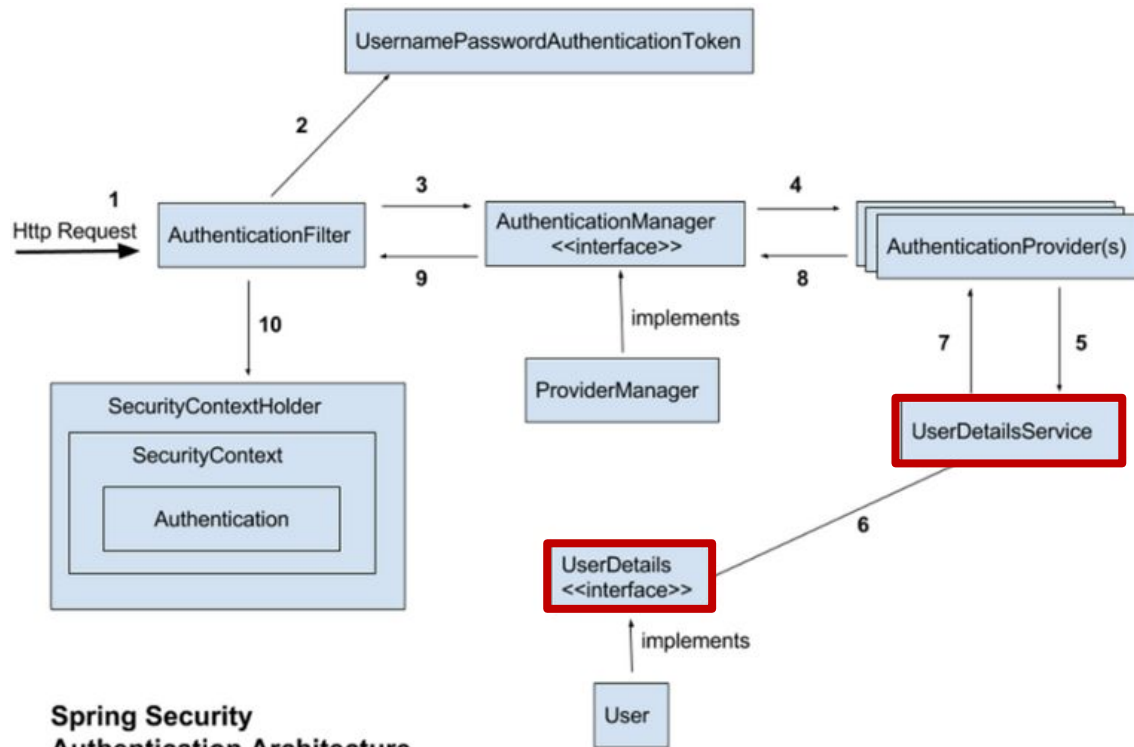
로그인

환영합니다.

admin

로그아웃

Spring Web Security 소개



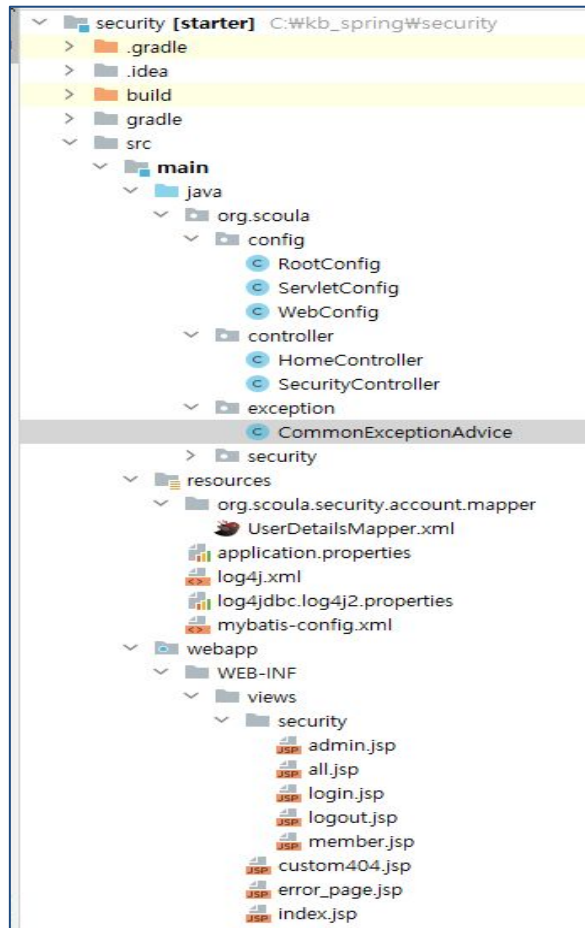
**Spring Security
Authentication Architecture**

스프링 시큐리티 아키텍처

- **HTTP 요청**이 들어오면 **AuthenticationFilter**가 요청을 가로챈.
- 이 필터는 사용자의 인증 정보를 **AuthenticationManager**에게 전달
- **AuthenticationManager**는 **AuthenticationProvider**를 사용해 인증을 처리
- **UserDetailsService**는 사용자 정보를 데이터베이스에서 가져온다.
- 인증이 성공하면 사용자의 정보가 **SecurityContextHolder**에 저장됨.

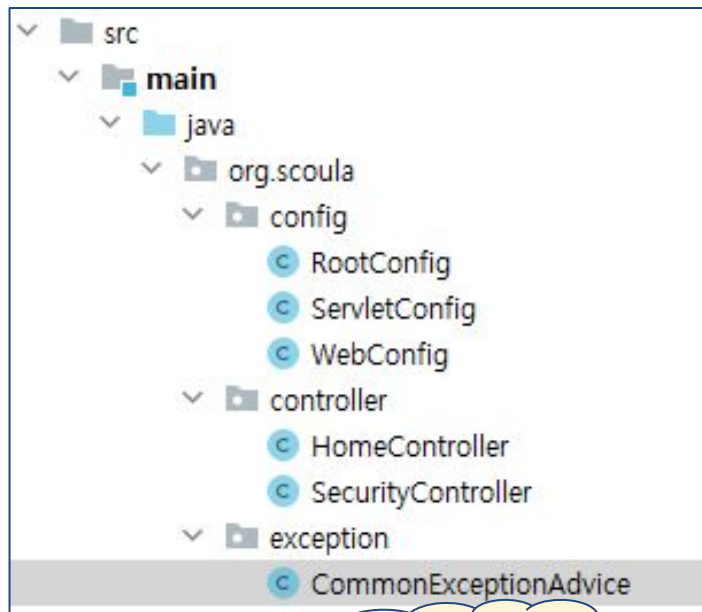
- 프로젝트 생성

- name: security
- 1일차 : 프로젝트 파일 다운로드
- 2일차 : 프로젝트 파일 다운로드



- **spring exception(@ControllerAdvice, @ExceptionHandler, @ResponseStatus)**

- 프로젝트 exception을 일관되게 처리



Advice

```
package org.scoula.exception;

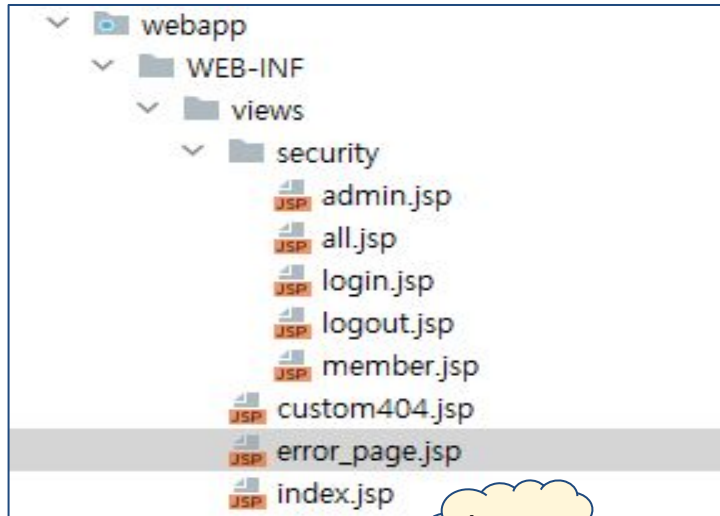
import lombok.extern.log4j.Log4j;
import org.springframework.http.HttpStatus;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.servlet.NoHandlerFoundException;

@ControllerAdvice
@Log4j
public class CommonExceptionHandlerAdvice {
    @ExceptionHandler(Exception.class)
    public String except(Exception ex, Model model) {
        log.error("Exception ..... " + ex.getMessage());
        model.addAttribute("exception", ex);
        log.error(model);
        return "error_page";
    }

    @ExceptionHandler(NoHandlerFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public String handle404(NoHandlerFoundException ex) {
        return "custom404";
    }
}
```


- spring exception

- 프로젝트 exception을 일관되게 처리



jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" import="java.util.*"%>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Insert title here</title>
</head>
<body>
<h4><c:out value="${exception.getMessage()}"></c:out></h4>

<ul>
  <c:forEach items="${exception.getStackTrace()}" var="stack">
    <li><c:out value="${stack}"></c:out></li>
  </c:forEach>
</ul>

</body>
</html>
```

```
<!DOCTYPE html>
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Insert title here </title>
</head>
<body>
<h1>해당 URL은 존재하지 않습니다.</h1>
</body>
</html>
```

- driver properties & log

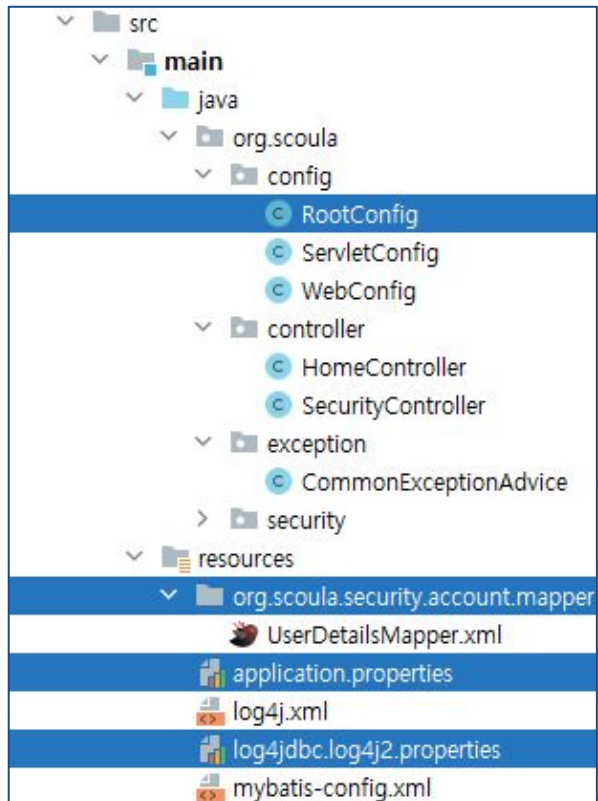
- 연결 정보 분리 & 연결 정보 상세 로그
- MapperScan : mapper xml파일이 있는 자동 설정
- classpath는 resources를 의미

```
@Configuration
@PropertySource({"classpath:/application.properties"})
@Slf4j
@MapperScan(basePackages = {"org.scoula.security.account.mapper"})
public class RootConfig {
    @Value("${jdbc.driver}") String driver;
    @Value("${jdbc.url}") String url;
    @Value("${jdbc.username}") String username;
    @Value("${jdbc.password}") String password;

    @Bean
    public DataSource dataSource() {
        HikariConfig config = new HikariConfig();

        config.setDriverClassName(driver);
        config.setJdbcUrl(url);
        config.setUsername(username);
        config.setPassword(password);

        HikariDataSource dataSource = new HikariDataSource(config);
        return dataSource;
    }
}
```

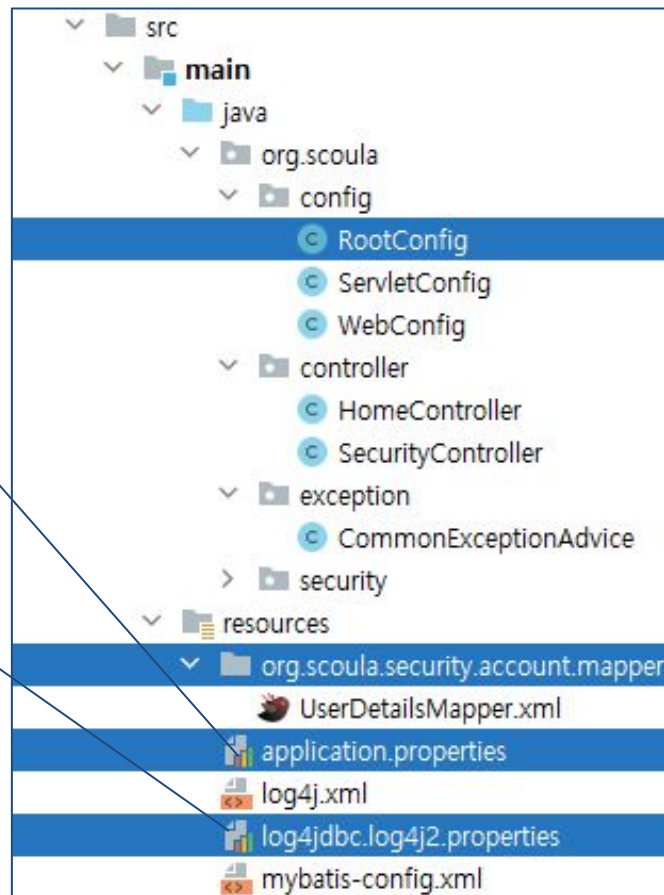


- driver properties & log

```
implementation 'org.bgee.log4jdbc-log4j2:log4jdbc-log4j2-jdbc4:1.16'  
implementation 'org.apache.logging.log4j:log4j-api:2.0.1'  
implementation 'org.apache.logging.log4j:log4j-core:2.0.1'
```

```
jdbc.driver=net.sf.log4jdbc.sql.jdbcapi.DriverSpy  
jdbc.url=jdbc:log4jdbc:mysql://localhost:3306/shop2  
jdbc.username=root  
jdbc.password=1234
```

```
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator  
log4jdbc.auto.load.popular.drivers=false  
log4jdbc.drivers=com.mysql.cj.jdbc.Driver
```



- **build.gradle**

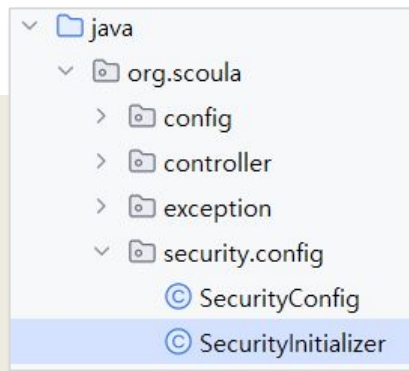
```
...
ext {
    junitVersion = '5.9.2'
    springVersion = '5.3.37'
    ...
    springSecurityVersion='5.8.13'
}

dependencies {
    ...

    // 보안
    implementation("org.springframework.security:spring-security-web:${springSecurityVersion}")
    implementation("org.springframework.security:spring-security-config:${springSecurityVersion}")
    implementation("org.springframework.security:spring-security-core:${springSecurityVersion}")
    implementation("org.springframework.security:spring-security-taglibs:${springSecurityVersion}")
    ...
}
```

- **SecurityInitializer.java**

```
package org.scoula.security.config;  
  
import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;  
  
public class SecurityInitializer extends AbstractSecurityWebApplicationInitializer {  
  
}
```

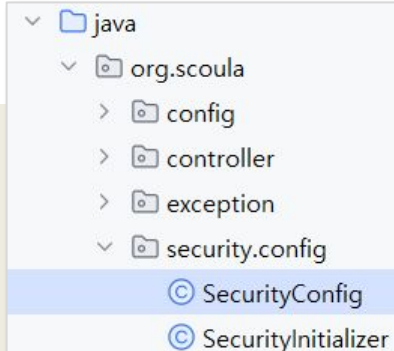


- **SecurityConfig.java**

```
package org.scoula.security.config;

@Configuration
@EnableWebSecurity
@Log4j
public class SecurityConfig extends WebSecurityConfigurerAdapter {

}
```



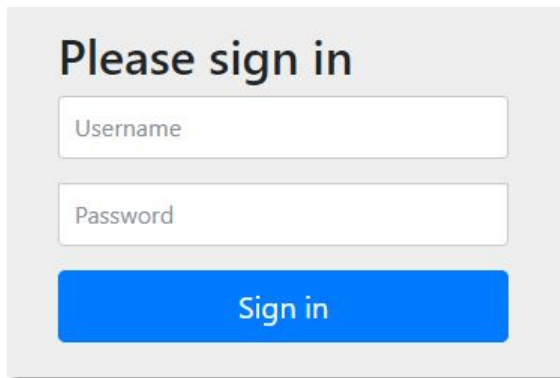
- **config/WebConfig.java**

```
@Override
public Class<?>[] getRootConfigClasses() {
    return new Class[] { RootConfig.class, SecurityConfig.class };
}
```

- 실행

- 모든 페이지는 보안정책에 따라 접근
- 스프링 시큐리티가 설정되면
 - 디폴트 : 모든 페이지는 로그인을 해야 접근 가능
 - /login 페이지로 리다이렉트됨

<http://localhost:8080/login>

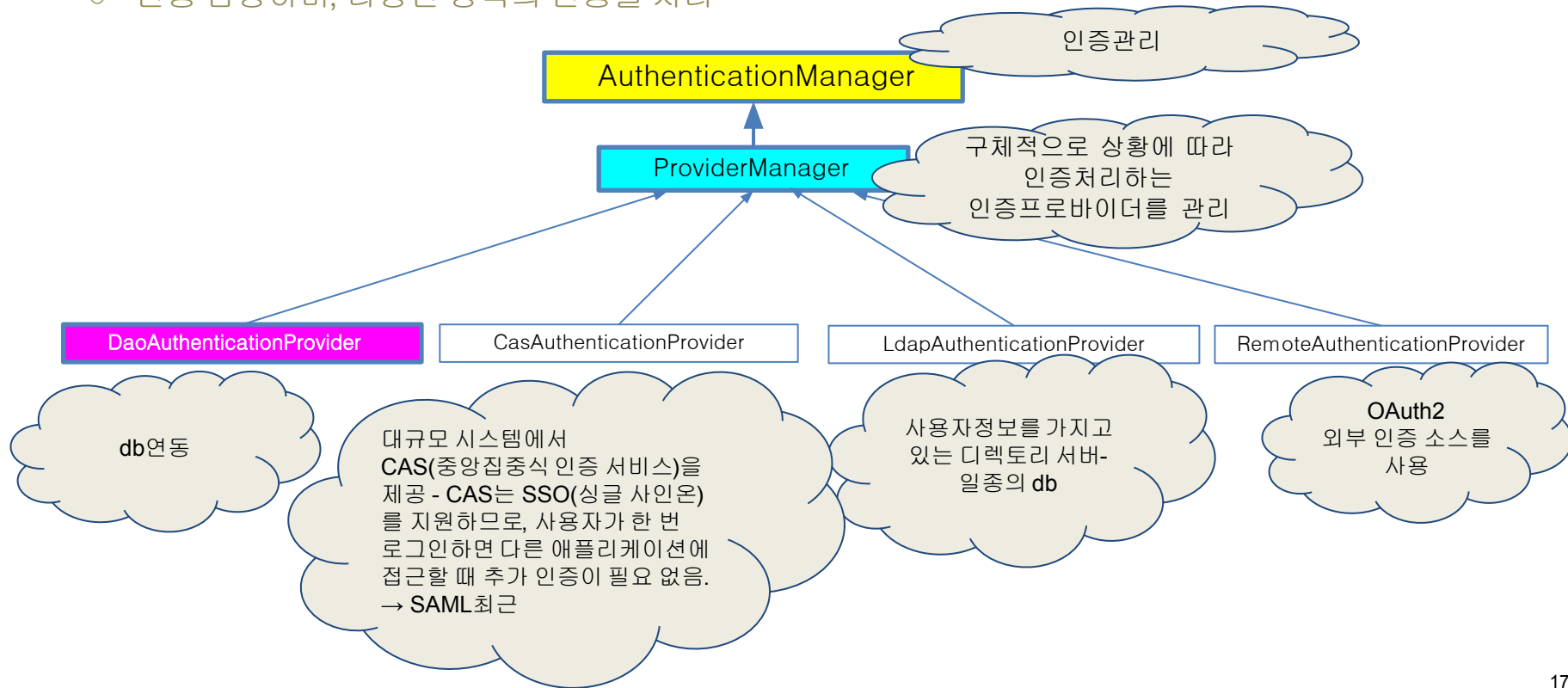


A screenshot of a web browser showing a login form. The form is titled "Please sign in" in bold black text. Below the title are two input fields: "Username" and "Password". At the bottom of the form is a blue button with the text "Sign in". The form is set against a light gray background with a subtle shadow.

- 인증(Authentication): 사용자가 누구인지 확인하는 과정
 - `AuthenticationManager`가 다양한 인증 프로바이더를 통해 인증을 처리
- 권한 부여(Authorization): 인증된 사용자가 무엇을 할 수 있는지 결정하는 과정
 - 인증 이후에 이루어짐

• AuthenticationManager(인증 매니저)

- 인증 담당하며, 다양한 방식의 인증을 처리

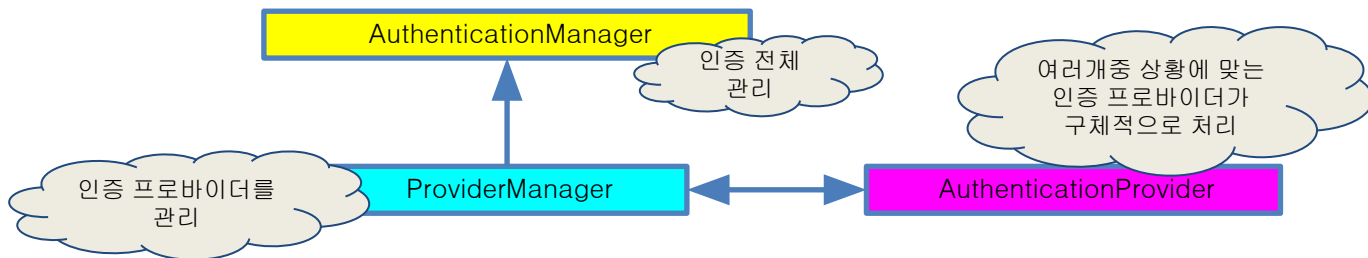


- **AuthenticationManager**: 전체적인 인증 관리 담당
- **ProviderManager**:
 - 실제적인 구체적인 방법으로 인증 처리
 - 상황에 맞는 여러 **provider**를 관리
 - 상황에 맞는 **provider**를 이용해 구체적으로 처리

- **DaoAuthenticationProvider** : 각기 다른 방식으로 사용자 인증을 처리. 주로 데이터베이스에 저장된 사용자 정보를 통해 인증을 처리
 - 주어진 사용자 이름(username)과 비밀번호(password)를 데이터베이스와 비교하여 사용자를 인증이 과정에서 **UserDetailsService** 인터페이스를 사용하여 사용자 정보를 가져오고, 비밀번호를 **PasswordEncoder**로 인코딩하거나 비교하는 방식으로 인증을 수행
 - **UserDetailsService**: 사용자 정보를 불러오는 인터페이스
 - 데이터베이스 또는 다른 영구 저장소에서 사용자 정보를 로드하는 역할
 - 예를 들어, 사용자 이름에 해당하는 비밀번호와 권한 정보를 반환
 - **CustomUser** ← **MemberVO** 시큐리티 적용하여 자동으로 설정하려면, 커스텀해주어야 함.
 - **PasswordEncoder**: 사용자의 입력 비밀번호를 인코딩하거나, 데이터베이스에 저장된 인코딩된 비밀번호와 비교하는 역할, **BCryptPasswordEncoder**와 같은 다양한 인코더 구현체를 사용

• ProviderManager

- 인증에 대한 처리를 AuthenticationProvider라는 타입의 객체를 이용해 처리



• AuthenticationProvider(인증 제공자)

- 실제 인증 작업을 진행
- UserDetailsService**가 인증된 정보에 권한에 대한 정보를 구성



- 스프링 시큐리티를 커스터마이징 하는 방식
 - AuthenticationProvider를 직접 구현하는 방식
 - 실제 처리를 담당하는 **UserDetailsService**를 구현하는 방식

다음주 월요일에
db연동하여 인증
처리 진행 예정

```
public interface UserDetailsService {  
    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;  
}
```

로그인과 로그아웃 처리

- 한글 문자 인코딩 발생
 - WebConfig에서 등록한 문자 인코딩 필터보다 먼저 Security Filter가 먼저 동작
 - Security Filter에서 POST body가 resolve됨 - 한글 깨짐
 - Spring Security Filter 체인에서 문자 인코딩 필터를 CsrfFilter 보다 앞에 등록 필요

- SecurityController.java

```
...
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    // 문자셋 필터
    public CharacterEncodingFilter encodingFilter() {
        CharacterEncodingFilter encodingFilter = new CharacterEncodingFilter();
        encodingFilter.setEncoding("UTF-8");
        encodingFilter.setForceEncoding(true);
        return encodingFilter;
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.addFilterBefore(encodingFilter(), CsrfFilter.class);
        ...
    }
}
```


• controller/SecurityController.java

```
package org.scoula.controller;
```

```
...
```

```
@Log4j
```

```
@RequestMapping("/security")
```

```
@Controller
```

```
public class SecurityController {
```

```
    @GetMapping("/all")
```

```
// 모두 접근 가능
```

```
    public void doAll() {
```

```
        log.info("do all can access everybody");
```

```
    }
```

```
    @GetMapping("/member")
```

```
// MEMBER 또는 ADMIN 권한 필요
```

```
    public void doMember() {
```

```
        log.info("logged member");
```

```
    }
```

```
    @GetMapping("/admin")
```

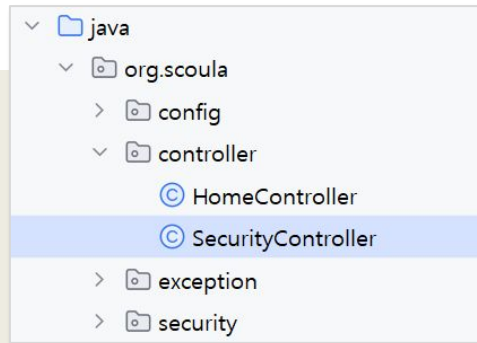
```
// ADMIN 권한 필요
```

```
    public void doAdmin() {
```

```
        log.info("admin only");
```

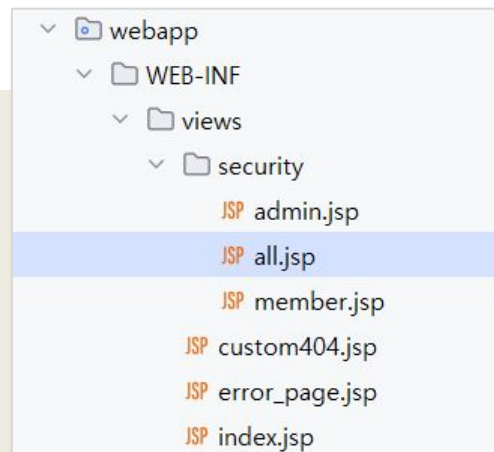
```
    }
```

```
}
```



- views/security 폴더에 all.jsp, member.jsp, admin.jsp 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
    <h1>/security/all page</h1>
</body>
</html>
```



- <h1> 부분은 각 페이지에 맞게 수정

• 경로별 인증/권한 설정

- `public void configure(HttpSecurity http)`이 `http` 인자로 설정
 - 내부적으로 builder 패턴 적용되어 있음.
 - 대부분 메서드의 리턴값이 `HttpSecurity`임 - 메서드 체이닝으로 설정해 나감
 - `ROLE_ADMIN`, `ROLE_MEMBER`는 개발자가 정의, 예약어 아님.

빌더 패턴(Builder Pattern)은 객체 생성 패턴 중 하나로, 복잡한 객체를 단계적으로 생성할 수 있게 해주는 디자인 패턴, 객체를 생성할 때 여러 메서드를 연속적으로 호출하여 설정을 적용

```
http.authorizeRequests()
```

```
// 모두 허용
```

```
.antMatchers("/security/all").permitAll()
```

```
// 특정 역할에게만 허용
```

```
.antMatchers("/security/admin").access("hasRole('ROLE_ADMIN')")
```

```
.antMatchers("/security/member").access("hasRole('ROLE_MEMBER')")
```

```
// 로그인 사용자에게 허용
```

```
.antMatchers( "/board/write",
```

```
"/board/modify",
```

```
"/board/delete").authenticated();
```

1. `configure(HttpSecurity http)` 메서드

- Spring Security에서 HTTP 요청에 대한 보안 설정을 구성할 때 사용됨.
- 주로 경로 별로 접근 권한을 지정할 수 있으며, 대부분의 메서드들은 체이닝(chain) 방식으로 연결되어 설정을 이어감.

2. `authorizeRequests()`

- 보안 필터 체인의 `HttpSecurity` 객체에서 요청 경로에 따라 접근 권한을 설정하는 데 사용됨.

3. `antMatchers()` 메서드

- 특정 URL 패턴에 대한 접근 권한을 설정하는 데 사용
- `/security/all` 경로는 모든 사용자에게 허용(`permitAll()`)
- `/security/admin` 경로는 `ROLE_ADMIN` 권한을 가진 사용자만 접근할 수 있도록 설정(`hasRole('ROLE_ADMIN')`)되어 있음.

4. 특정 권한 부여

- `/security/admin` 경로는 `ROLE_ADMIN` 권한을 가진 사용자에게만 허용
- `/security/member` 경로는 `ROLE_MEMBER` 권한을 가진 사용자에게만 허용

5. 로그인 사용자에게만 허용되는 경로

- `/board/write`, `/board/modify`, `/board/delete`와 같은 경로는 로그인한 사용자만 접근할 수 있도록 설정(`authenticated()`)되어 있음.

- 경로, 접근 허용 기준
 - `"/security/all"`, 모든 사용자 허용 (`permitAll()`)
 - `"/security/admin"`, `ROLE_ADMIN` 권한을 가진 사용자만 허용
 - `"/security/member"`, `ROLE_MEMBER` 권한을 가진 사용자만 허용
 - `"/board/write"`, 로그인한 사용자만 허용 (`authenticated()`)
 - `"/board/modify"`, 로그인한 사용자만 허용 (`authenticated()`)
 - `"/board/delete"`, 로그인한 사용자만 허용 (`authenticated()`)

• SecurityConfig.java

```
...

@Configuration
@EnableWebSecurity
@Log4j
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    public void configure(HttpSecurity http) throws Exception {

        // 경로별 접근 권한 설정
        http.authorizeRequests()
            .antMatchers("/security/all").permitAll()
            .antMatchers("/security/admin").access("hasRole('ROLE_ADMIN')")
            .antMatchers("/security/member").access("hasRole('ROLE_MEMBER')");
    }
}
```

Spring Security에서
antMatchers의 **ant**는 "Ant-Style
Patterns"를 의미합니다. 이는
Apache Ant라는 빌드 도구에서
유래된 패턴 매칭 방식을 사용하는
것을 나타냄.

인증정보를 얻기 위해
login페이지로 자동
포워드됨.

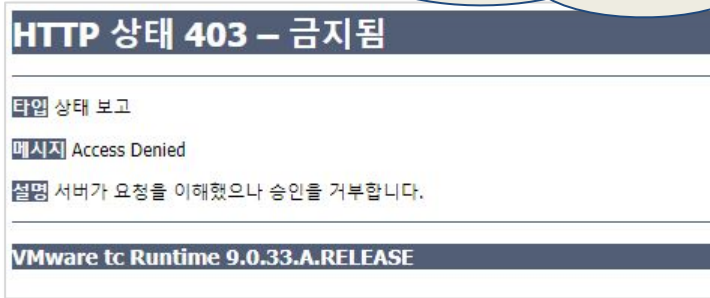
- <http://localhost:8080/security/all>

/security/all page

- <http://localhost:8080/security/member>
- <http://localhost:8080/security/admin>

- 에러 발생보다는 로그인 페이지로 리다이렉트하는 것이 좋음
--> 로그인 설정 필요

member로 로그인 후,
security/admin 접속하면
403에러



- 로그인 설정

- **HttpSecurity http**

- form 기반의 로그인 설정하는 기본으로 제공
- 로그인 화면도 자동으로 생성 가능함.
- 커스텀해서 로그인페이지를 만들어 줄 수도 있음. (*)

http.formLogin()

.loginPage("/customLogin")

.loginProcessingUrl("/login");

// 로그인 폼 설정 시작

// 로그인 요청 폼 GET url 설정

// 로그인 POST 요청 url 설정

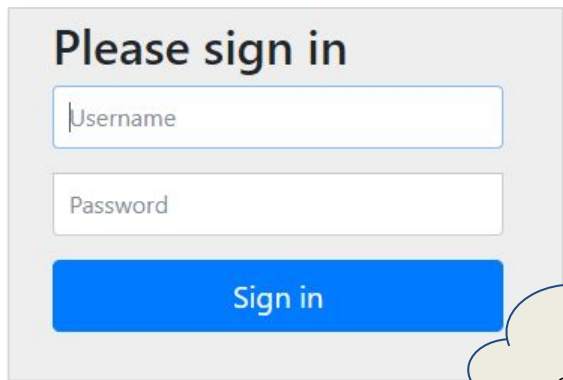
• SecurityConfig.java

```
...  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
    @Override  
    public void configure(HttpSecurity http) throws Exception {  
        http.authorizeRequests()  
            .antMatchers("/security/all").permitAll()  
            .antMatchers("/security/admin").access("hasRole('ROLE_ADMIN')")  
            .antMatchers("/security/member").access("hasAnyRole('ROLE_MEMBER', 'ROLE_ADMIN')");  
  
        http.formLogin();    // form 기반 로그인 활성화, 나머지는 모두 디폴트  
    }  
}
```

자동으로 로그인페이지
만들어주는 기능이 기본
설정되어있음.

- <http://localhost:8080/security/member>
- <http://localhost:8080/security/admin>

— /login으로 리다이렉트
<http://localhost:8080/login>



Please sign in

Username

Password

Sign in

기본값 설정에
의해 자동 생성됨

- 인증 정보 설정
 - `protected void configure(AuthenticationManagerBuilder auth)`
 - 사용자 정보를 어디서(메모리, 파일, db 등) 얻을지 설정
 - 메모리에 사용자 정보 구축하는 경우
 - 주로 테스트용

`auth.inMemoryAuthentication()` // 메모리에서 사용자 정보 설정

```
.withUser("admin")           // username, 사용자 id
.password("{noop}1234")      // 비밀번호, {noop}는 암호화 없음 의미
.roles("ADMIN","MEMBER");    // ROLE_ADMIN 역할 설정
```

• SecurityConfig.java

```
@Override
protected void configure(AuthenticationManagerBuilder auth)
    throws Exception {
    log.info("configure .....");

    auth.inMemoryAuthentication()
        .withUser("admin")
        .password("{noop}1234")
        .roles("ADMIN","MEMBER");           // ROLE_ADMIN

    auth.inMemoryAuthentication()
        .withUser("member")
        .password("{noop}1234")
        .roles("MEMBER");                   // ROLE_MEMBER
    }
}
```

- admin 또는 member로 로그인 가능

- 로그인 페이지 커스텀마이징

- 기본적으로 제공되는 localhost:8080/login 대신 새로운 로그인 페이지 운영

- 로그인 설정

```
http.formLogin()
```

```
.loginPage("/security/login")
```

```
.loginProcessingUrl("/security/login")
```

```
.defaultSuccessUrl("/");
```

```
// 로그인 설정 시작
```

```
// 로그인 페이지 GET URL — security/login 뷰(jsp) 정의
```

```
// 로그인 POST URL — login form의 action에 지정
```

```
// 로그인 성공 시 이동(redirect)할 페이지
```

- **config/SecurityConfig.java**

```
...
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/security/all").permitAll()
            .antMatchers("/security/admin").access("hasRole('ROLE_ADMIN')")
            .antMatchers("/security/member").access("hasAnyRole('ROLE_MEMBER', 'ROLE_ADMIN')");

        http.formLogin()
            .loginPage("/security/login")
            .loginProcessingUrl("/security/login")
            .defaultSuccessUrl("/");
    }
}
```

- **controller/SecurityController.java**

```
...
@RequestMapping("/security")
public class SecurityController {
    ...

    @GetMapping("/login")
    public void login() {
        log.info("login page");
    }
}
```

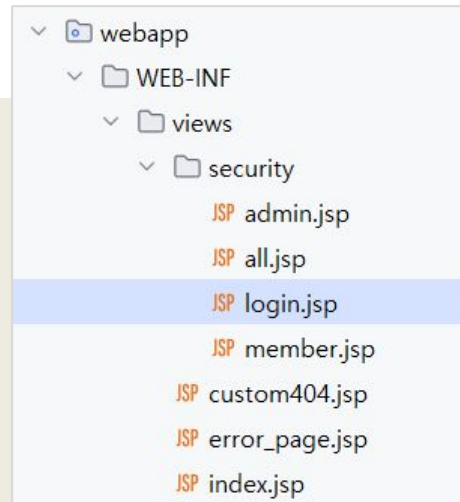
- **CSRF(Cross Site Request Forgery) 공격**

- 인터넷 사용자(희생자)가 자신의 의지와는 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)를 특정 웹사이트에 요청하게 만드는 공격
- POST 요청을 위조하여 전송하는 것
- 방어책
 - CSRF 토큰 운영
 - form 페이지 GET요청 시 form 내에 인증 토큰을 심어서 전송
 - 인증 토큰이 있는 경우에만 정당한 POST 요청으로 인식
 - 해당 토큰이 없으면 에러를 발생시킴
- spring-security 사용시 디폴트로 사용하는 것으로 설정됨

- 로그인 form
 - 요청 파라미터의 이름이 정해져 있음
 - **username**: 사용자 id
 - **password**: 비밀번호

• security/login.jsp

```
<body>
  <h1>login</h1>
  <form name='f' action='/security/login' method='POST'>
    <input type="hidden" name="_csrf.parameterName" value="_csrf.token" />
    <table>
      <tr>
        <td>User:</td>
        <td><input type='text' name='username' value="" /></td>
      </tr>
      <tr>
        <td>Password:</td>
        <td><input type='password' name='password' /></td>
      </tr>
      <tr>
        <td colspan='2'>
          <input name="submit" type="submit" value="Login" />
        </td>
      </tr>
    </table>
  </form>
</body>
</html>
```



login

User:

Password:

```
<form name='f' action='/security/login' method='POST'>
  <input type="hidden" name="_csrf"
  value="d5bab99f-8a5d-46ad-990e-8b5504bce41b" />
  <table>
```

- 확인

- <http://localhost:8080/security/member> 요청

- 로그인하지 않은 상태이므로 /security/login으로 리다이렉트 됨
 - member로 로그인



A screenshot of a web login form. It has a title 'login' in bold. Below it are two input fields: 'User:' and 'Password:'. At the bottom is a 'Login' button.

- 로그인 성공 시

- 원래 요청 url로 리다이렉트됨
 - <http://localhost:8080/security/member>



- <http://localhost:8080/security/admin>을 요청하고, member로 로그인 하면 — 403 에러
 - member로 로그인한 상태에서 <http://localhost:8080/security/admin>을 요청하면 — 403 에러

- 접근 권한 설정 페이지 접근 시
 - 권한이 맞으면 요청한 페이지로 진입
 - 권한이 맞지 않으면
 - 로그인 하지 않은 경우 login 페이지로 이동
 - 로그인 이후 해당 페이지로 들어감
 - 로그인 된 상태라면 403에러 발생

- 로그아웃 설정

- 로그아웃 시 해야 할 일

- 세션 무효화(invalidate)
- 쿠키 제거: JSESSION-ID, remember-me

`http.logout()`

`.logoutUrl("/security/logout")`

`.invalidateHttpSession(true)`

`.deleteCookies("remember-me", "JSESSION-ID")`

`.logoutSuccessUrl("/");`

// 로그아웃 설정 시작

// 로그아웃 호출 url

// 세션 invalidate

// 삭제할 쿠키 목록

// 로그아웃 이후 이동할 페이지

- logout url을 post로 요청해야 적용됨. !!

• config/SecurityConfig.java

```
...
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    ...
    @Override
    public void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/security/all").permitAll()
            .antMatchers("/security/admin").access("hasRole('ROLE_ADMIN')")
            .antMatchers("/security/member").access("hasAnyRole('ROLE_MEMBER', 'ROLE_ADMIN')");

        http.formLogin()
            .loginPage("/security/login")
            .loginProcessingUrl("/security/login")
            .defaultSuccessUrl("/");

        http.logout()
            .logoutUrl("/security/logout")
            .invalidateHttpSession(true)
            .deleteCookies("remember-me", "JSESSION-ID")
            .logoutSuccessUrl("/security/logout");

        ...
    }
    ...
}
```

// 로그아웃 설정 시작
 // POST: 로그아웃 호출 url
 // 세션 invalidate
 // 삭제할 쿠키 목록
 // GET: 로그아웃 후 이동할 페이지

- **controller.SecurityController.java**

```
@RequestMapping("/security")
public class SecurityController {

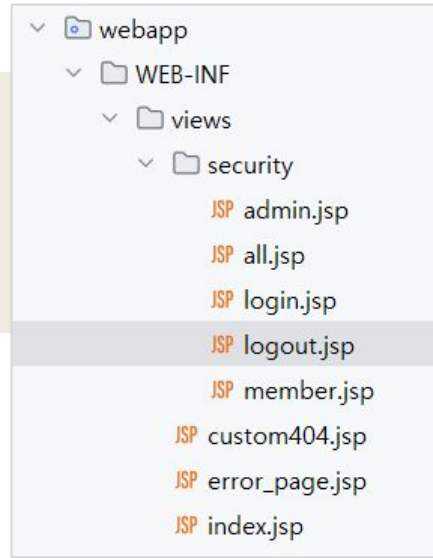
    ...

    @GetMapping("/logout")
    public void logout() {
        log.info("logout page");
    }
}
```

- security/logout.jsp

```
<body>
  <h1>
    로그 아웃됨
  </h1>

  <a href="/">홈으로</a>
</body>
```



- member.jsp, admin.jsp

```
<body>
  <h1>/security/member page</h1>

  <form action="/security/logout" method="post">
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
    <input type="submit" value="로그아웃"/>
  </form>
</body>
</html>
```

http://localhost:8080/security/logout

/security/member page

로그아웃

redirect:
/security/logout

로그 아웃됨

홈으로

- PasswordEncoder 인터페이스

- 비밀번호는 반드시 암호화해서 처리해야 함

메서드	설명
<code>String encode(String rawPassword)</code>	<ul style="list-style-type: none">- 암호화되지 않은 비밀번호(<code>rawPassword</code>)를 암호화해서 리턴함- 같은 값을 암호화해도 매번 다른 값을 리턴- <code>equals()</code>로 비교할 수 없음
<code>boolean matches(String rawPassword, String encodedPassword)</code>	<ul style="list-style-type: none">- 사용자가 입력한 암호화되지 않은 비밀번호(<code>rawPassword</code>)와 암호화된 비밀번호 (DB. 저장값)가 일치하는지 검사- 같으면 <code>true</code>, 다르면 <code>false</code> 리턴

- 구현체

- `BCryptPasswordEncoder`

- config/SecurityConfig.java

```
...  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
    @Bean  
    public PasswordEncoder passwordEncoder() {  
        return new BCryptPasswordEncoder();  
    }  
  
    ...  
  
}
```

- test : PasswordEncoderTest.java

```
@ExtendWith(SpringExtension.class)
```

```
@ContextConfiguration(classes = {  
    RootConfig.class,  
    SecurityConfig.class  
})
```

```
}
```

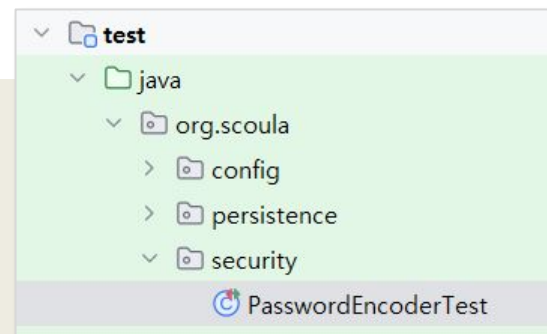
```
@Log4j
```

```
public class PasswordEncoderTest {
```

```
    @Autowired
```

```
    private PasswordEncoder pwEncoder;
```

```
}
```



- test: PasswordEncoderTests.java

```
@Test
public void testEncode() {
    String str = "1234";

    String enStr = pwEncoder.encode(str);           // 암호화
    log.info("password: " + enStr);

    String enStr2 = pwEncoder.encode(str);           // 암호화
    log.info("password: " + enStr2);

    log.info("match : " + pwEncoder.matches(str, enStr)); // 비밀번호 일치 여부 검사
    log.info("match : " + pwEncoder.matches(str, enStr2)); // 비밀번호 일치 여부 검사
}
```

INFO : org.galapagos.security.SecurityTest -
password: **\$2a\$10\$VJK.3K/W3PhSu53.FVm7W0EzFZPIGTw5.iiCZXgKTHPhK419Jdz2**
INFO : org.galapagos.security.SecurityTest -
password: **\$2a\$10\$ME3YMFVYP.Wi1YTL5ghU9.yPyuEkEBXpQI9FHBsZ9BpP0tef8hj72**
INFO : org.galapagos.security.SecurityTest - match :true
INFO : org.galapagos.security.SecurityTest - match :true

같은 문자열이어도
매번 다르게 암호화됨

- config/SecurityConfig.java

```
...
public class SecurityConfig extends WebSecurityConfigurerAdapter {
...
    @Override
    protected void configure(AuthenticationManagerBuilder auth)
        throws Exception {
        auth.inMemoryAuthentication()
            .withUser("admin")
            .password("{noop}1234")
            .password("$2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/IddCyn0nic5dFo3VfJYrXYC")
            .roles("ADMIN","MEMBER");           // ROLE_ADMIN, ROLE_MEMBER

        auth.inMemoryAuthentication()
            .withUser("member")
            .password("{noop}1234")
            .password("$2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/IddCyn0nic5dFo3VfJYrXYC")
            .roles("MEMBER");           // ROLE_MEMBER
    }
...
}
```

id	pw	name	tel
1	apple	apple	016
2	com	com	com
3	computer	test55	oracle
4	gardenpro	take	oracle
5	take1	\$2a\$10\$v8rnQF3y19ofkrJKPCU4Vul9tG3TvtbCdMKE8dsRgQishYwGR.QKy	hong
6	www	www	www

아이디:

패스워드:

이름:

전화:

아이디:

패스워드:

암호화된 로그인 처리 결과

암호화된 로그인 처리 결과 >> 암호화된 로그인 성공!!

```

at java.lang.Thread.run(Unknown Source)

** END NESTED EXCEPTION **

Tue Nov 17 10:14:53 KST 2020 WARN: Establishing SSL connection without server's identity verification.
1: $2a$10$1nRLr817Dv12iSF763nscOtQlXH.Bd7lgm80a/9ugk89DnCgrN9ii
2: $2a$10$1nRLr817Dv12iSF763nscOtQlXH.Bd7lgm80a/9ugk89DnCgrN9ii
3: true
1: $2a$10$1nRLr817Dv12iSF763nscOtQlXH.Bd7lgm80a/9ugk89DnCgrN9ii
2: $2a$10$1nRLr817Dv12iSF763nscOtQlXH.Bd7lgm80a/9ugk89DnCgrN9ii
3: false
    
```

jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<form action="sInsert.do">
아이디: <input type="text" name="id"><br>
패스워드: <input type="text" name="pw"><br>
이름: <input type="text" name="name" value="hong"><br>
전화: <input type="text" name="tel" value="011"><br>
<input type="submit" value="회원가입">
</form>
<hr color="red"><br>
<form action="sLogin.do">
아이디: <input type="text" name="id"><br>
패스워드: <input type="text" name="pw"><br>
<input type="submit" value="회원로그인">
</form>
</body>
</html>
```

Controller

```
@Controller
public class SecurityController {

    @Autowired
    MemberService service;

    @RequestMapping("sInsert.do")
    public String create2(MemberBag memberBag) {
        service.create(memberBag);
        return "redirect:security-test.jsp";
    }

    @RequestMapping("sLogin.do")
    public String login(MemberBag bag, Model model) {
        boolean result = service.login(bag);
        if (result) {
            model.addAttribute("result", ">> 암호화된 로그인 성공!!");
        } else {
            model.addAttribute("result", ">> 암호화된 로그인 실패@@@@@@@@");
        }
        return "result";
    }
}
```


Service

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

@Service
public class MemberService {

    //암호화를 해서 db에 넣는 작업
    @Autowired
    MemberDAO dao;

    @Autowired
    BCryptPasswordEncoder pwEncoder;

    //암호화해서 회원가입
    public void create(MemberBag bag) {
        //bag에는 입력한 pw가 들어있다.
        //이 pw를 꺼내서 암호화한 후, bag에 다시 넣어야 한다.
        bag.setPw(pwEncoder.encode(bag.getPw()));
        dao.create(bag);
    }

    //로그인처리
    public boolean login(MemberBag bag) {
        String getPw = dao.login(bag);
        System.out.println("서비스단에서 받은 getPw: " + getPw);
        if (pwEncoder.matches(bag.getPw(), getPw)) {
            return true;
        }
        return false;
    }
}
```

mapper

```
<mapper namespace="member">
    <insert id="create" parameterType="mbag">
        insert into member values
        ({id}, {pw}, {name}, {tel})
    </insert>
    <!-- orm(object, rdb) -->
    <select id="read" parameterType="mbag" resultMap="rMap">
        select * from
        member where id = #{id}
    </select>

    <select id="one" parameterType="mbag" resultType="String">
        select pw from member where id = #{id}
    </select>

    <!-- mbag의 멤버변수이름과 항목명이 동일해야함. -->
    <!-- 결과는 테이블형태(항목들, 레코드들의 리스트) -->
    <resultMap type="mbag" id="rMap">
        <result property="id" column="id" />
        <result property="pw" column="pw" />
        <result property="name" column="name" />
        <result property="tel" column="tel" />
    </resultMap>
</mapper>
```

```
<?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"
```

```
    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/
```

```
    <beans:bean id="bcryptPasswordEncoder"  
        class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder
```

```
</beans:beans>
```

빈설정

DAO

```
@Repository
```

```
public class MemberDAO {
```

```
    @Autowired
```

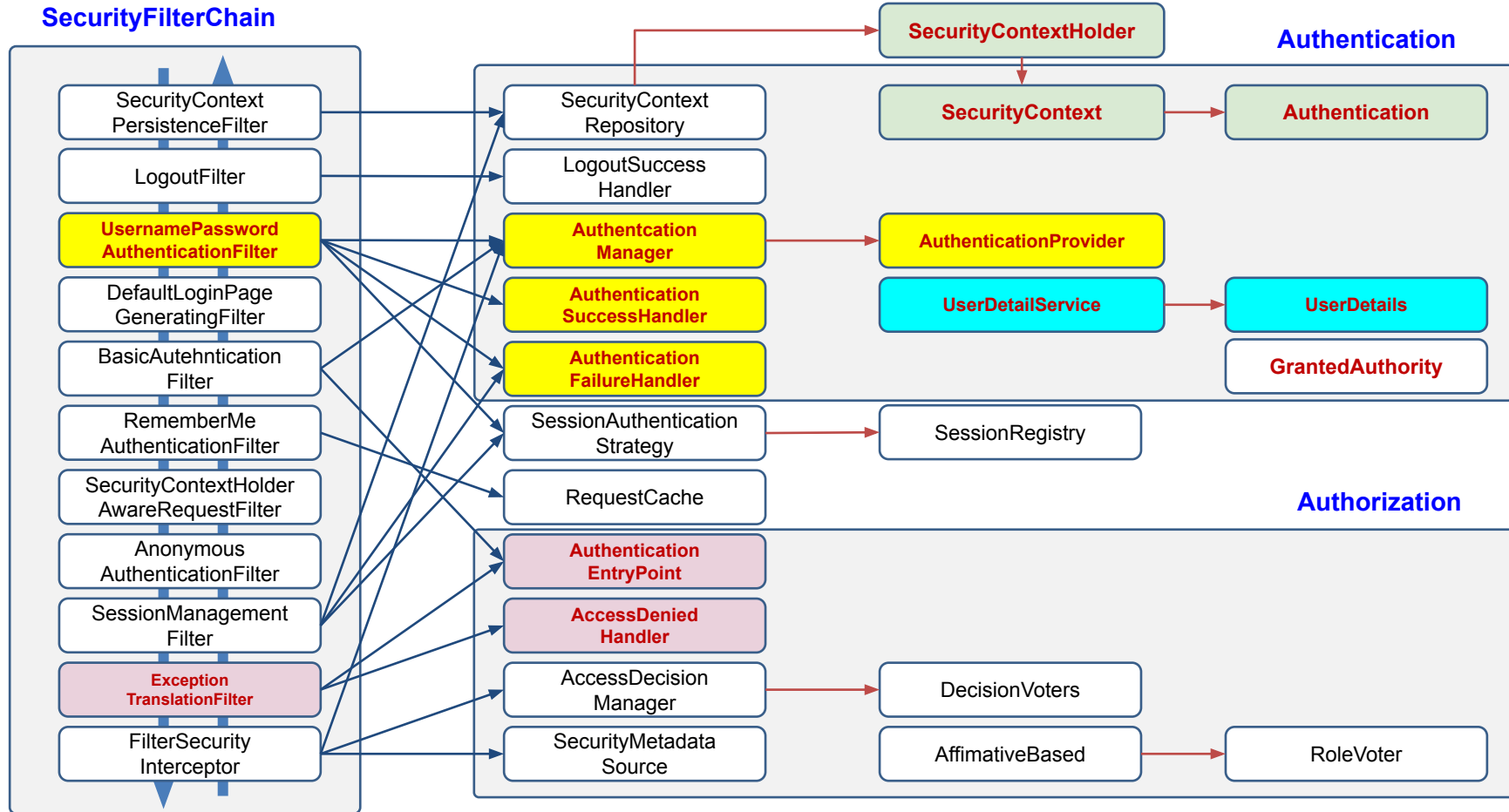
```
    SqlSessionTemplate mybatis;
```

```
    public int insert(MemberVO vo) {  
        return mybatis.insert("member.create", vo);  
    }
```

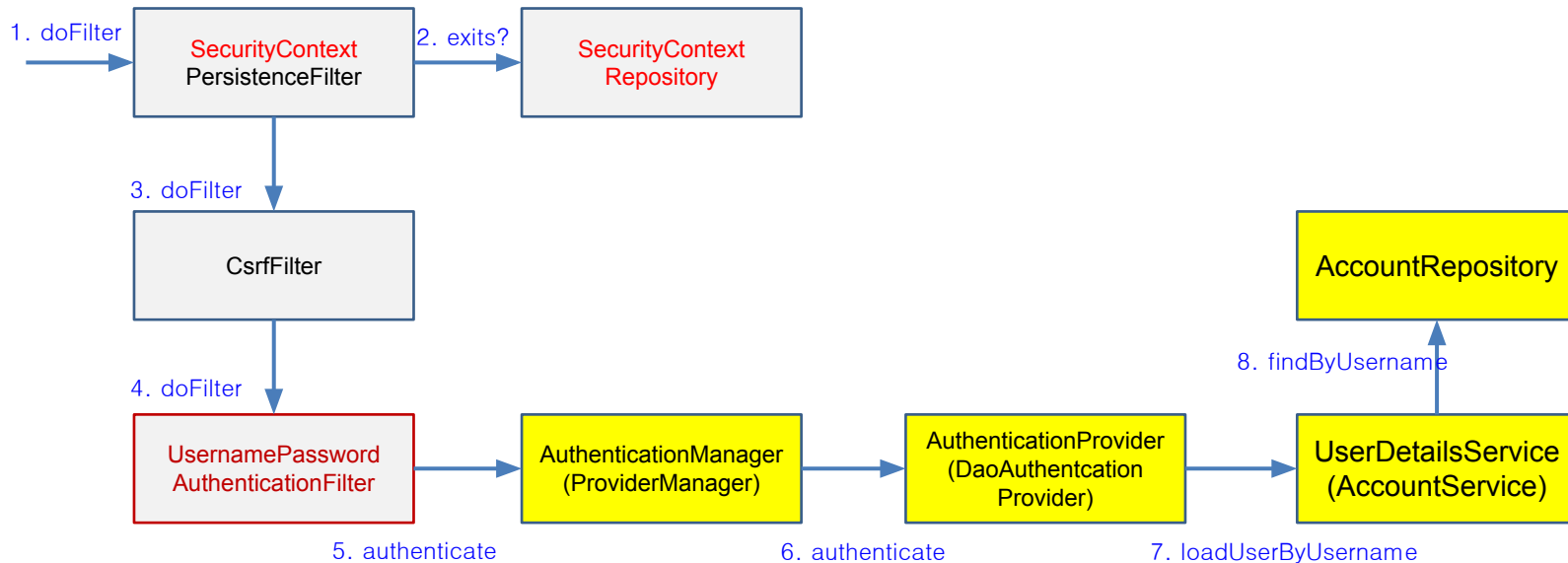
```
    public String slogin(MemberVO vo) {  
        String getPw = mybatis.selectOne("member.one", vo);  
        System.out.println("1: " + getPw);  
        if(getPw != null) {  
            return getPw;  
        }  
        return "null";  
    }
```

```
}
```

Security Filter Chain



- **UsernamePasswordAuthenticationFilter**: 사용자가 입력한 사용자 이름과 비밀번호를 처리하여 인증을 시도하는 필터
- **AuthenticationManager**: 인증을 처리하는 핵심 관리자. 다양한 인증 프로바이더 (**AuthenticationProvider**)를 통해 인증을 시도
- **AuthenticationProvider**: 실제로 인증을 수행하는 컴포넌트. 예를 들어, **DaoAuthenticationProvider**는 데이터베이스에서 사용자 정보를 조회하여 인증을 수행
- **UserDetailsService**: 사용자 정보를 로드하는 서비스. 주로 데이터베이스에서 사용자 정보를 가져와 **UserDetails** 객체로 반환
- **UserDetails**: 인증에 필요한 사용자 정보를 담고 있는 객체. 사용자 이름, 비밀번호, 권한 등의 정보를 포함
- **GrantedAuthority**: 사용자의 권한(예: **ROLE_USER**, **ROLE_ADMIN**)을 나타내는 객체
- **AuthenticationSuccessHandler**: 인증이 성공했을 때 실행되는 핸들러로, 보통 로그인 후 리다이렉션 등을 처리
- **AuthenticationFailureHandler**: 인증이 실패했을 때 실행되는 핸들러로, 로그인 실패 후의 처리를 담당
- **ExceptionTranslationFilter**: 인증 또는 권한 부여 과정에서 발생하는 예외를 처리하는 필터
- **AccessDeniedHandler**: 사용자가 접근할 권한이 없을 때(**Authorization 실패**) 처리하는 핸들러.



- **SecurityContextPersistenceFilter**

- request가 발생하면 **SecurityContext** 객체의 생성, 저장, 조회를 담당하는 필터
- 새로운 **SecurityContext**를 생성하여 **SecurityContextHolder**에 저장
- 익명의 사용자의 경우
 - **AnonymousAuthenticationFilter**에서 **AnonymousAuthenticationToken**객체를 **SecurityContext**에 저장
- 인증 시
 - **UsernamePasswordAuthenticationFilter**에서 인증 성공후 **SecurityContext**에 **UsernamePasswordAuthentication**객체를 **Authentication**객체와 함께 저장
 - 인증이 완료되면 **Session**에 **SecurityContext**를 저장하고 **Response**
- 인증 후
 - **Session**에서 **SecurityContext**를 꺼내 **SecurityContextHolder**에 저장
 - **SecurityContext**내 **Authentication**객체가 있으면 인증을 유지

- **LogoutFilter**

- 유저의 로그아웃을 진행
- 설정된 로그아웃 **URL**로 오는 요청을 감시하여 , 해당 유저를 로그아웃 처리

- **UsernamePasswordAuthenticationFilter**

- 설정된 로그인 URL로 오는 요청을 감시하며, 유저 인증을 처리
- 인증 실패시, `AuthenticationFailureHandler`를 실행

- **DefaultLoginPageGenerationFilter**

- 사용자가 별도의 로그인 페이지를 구현하지 않은 경우, 기본적으로 설정한 로그인 페이지를 처리

- **BasicAuthenticationFilter**

- HTTP요청의 (BASIC)인증 헤더를 처리하여 결과를 `SecurityContextHolder`에 저장

- **RememberMeAuthenticationFilter**

- `SecurityContext`에 인증(Authentication) 객체가 있는지 확인
- `RememberMeServices`를 구현한 객체의 요청이 있을 경우, Remember-Me인증 토큰으로 컨텍스트에 주입

- **AnonymousAuthenticationFilter**

- SecurityContextHolder에 인증(Authentication)객체가 있는지 확인
- 필요한 경우 Authentication 객체를 주입

- **SessionManagementFilter**

- 요청이 시작된 이후 인증된 사용자인지 확인하고, 인증된 사용자 일 경우, SessionAuthenticaitonStrategy를 호출하여 세션 고정 보호 메커니즘을 활성화하거나 여러 동시 로그인을 확인하는 것과 같은 세션 관련 활동을 수행

- **ExceptionTranslationFilter**

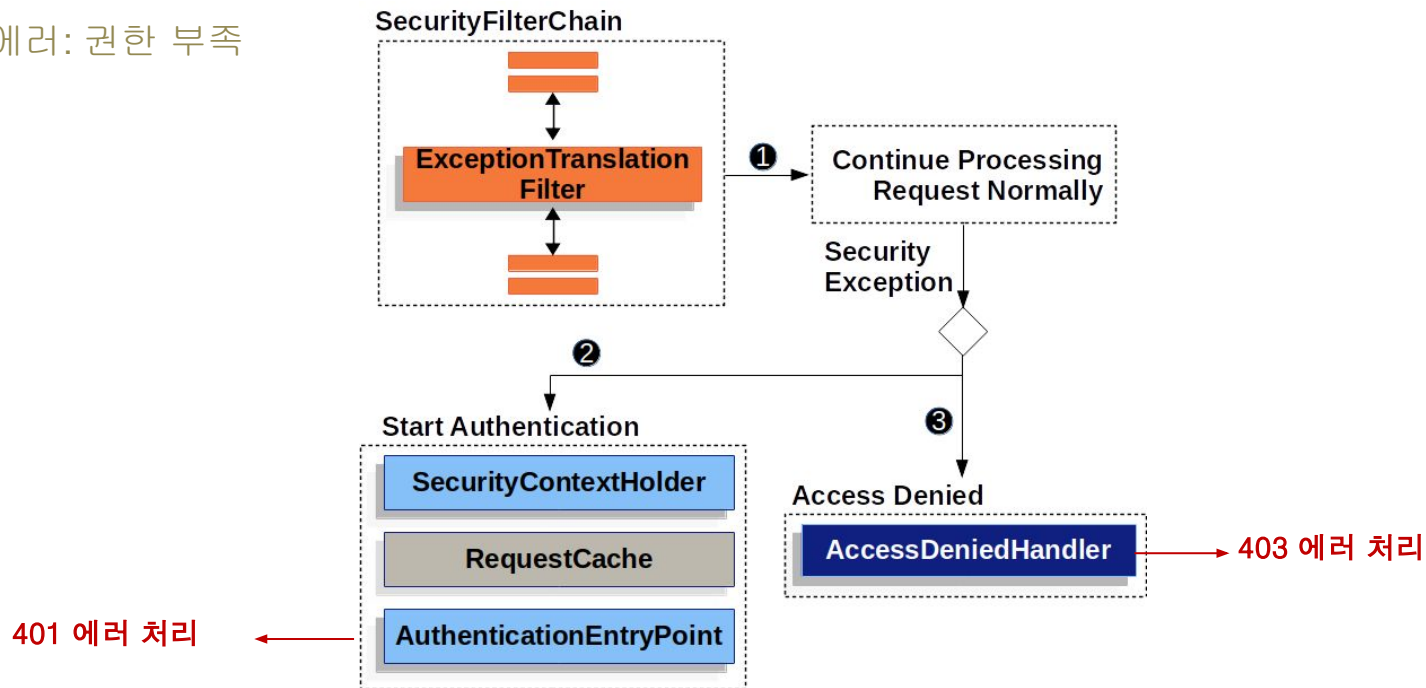
- 필터체인 내에서 발생하는 모든 예외(AccessDeniedException, AuthenticationException)를 처리

- **FilterSecurityInterceptor**

- HTTP 리소스의 보안처리를 수행

- 보안 에러

- 401 에러: 로그인 없이 접근한 경우
- 403 에러: 권한 부족



member 테이블

• 테이블 생성

-- 사용자 정보 테이블

```
drop table if exists tbl_member;
```

```
create table tbl_member
```

```
(  
    username          varchar(50) primary key,      -- 사용자 id  
    password          varchar(128) not null,         -- 암호화된 비밀번호  
    email             varchar(50) not null,  
    reg_date          datetime default now(),  
    update_date        datetime default now()  
);
```

-- 사용자 권한 테이블

```
drop table if exists tbl_member_auth;
```

```
create table tbl_member_auth
```

```
(  
    username          varchar(50) not null,          -- 사용자 id  
    auth              varchar(50) not null,          -- 권한 문자열 ROLE_ADMIN, ROLE_MANAGER, ROLE_MEMBER 등  
    primary key(username, auth),                    -- 복합키  
    constraint fk_authorities_users foreign key (username) references tbl_member(username)  
);
```

• 데이터 추가

-- 테스트 사용자 추가

```
insert into tbl_member(username, password, email)
```

```
values
```

```
('admin', '$2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/IddCyn0nic5dFo3VfJYrXYC', 'admin@galapgos.org'),  
('user0', '$2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/IddCyn0nic5dFo3VfJYrXYC', 'user0@galapgos.org'),  
('user1', '$2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/IddCyn0nic5dFo3VfJYrXYC', 'user1@galapgos.org'),  
('user2', '$2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/IddCyn0nic5dFo3VfJYrXYC', 'user2@galapgos.org'),  
('user3', '$2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/IddCyn0nic5dFo3VfJYrXYC', 'user3@galapgos.org'),  
('user4', '$2a$10$EsIMfxbJ6NuvwX7MDj4WqOYFzLU9U/IddCyn0nic5dFo3VfJYrXYC', 'user4@galapgos.org');
```

```
select * from tbl_member;
```

- 권한 데이터 추가

```
insert into tbl_member_auth(username, auth)
values
```

```
    ('admin','ROLE_ADMIN'),
    ('admin','ROLE_MANAGER'),
    ('admin','ROLE_MEMBER'),
    ('user0','ROLE_MANAGER'),
    ('user0','ROLE_MEMBER'),
    ('user1','ROLE_MEMBER'),
    ('user2','ROLE_MEMBER'),
    ('user3','ROLE_MEMBER'),
    ('user4','ROLE_MEMBER');
```

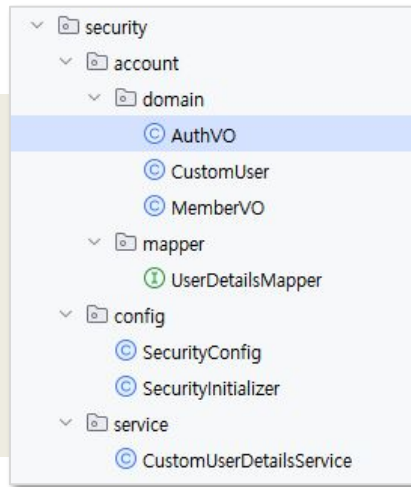
```
select * from tbl_member_auth order by auth;
```

- ADMIN(최고 관리자)
 - ROLE_ADMIN, ROLE_MANAGER, ROLE_MEMBER
- MANAGER(일반 관리자)
 - ROLE_MANAGER, ROLE_MEMBER
- MEMBER(일반 회원)
 - ROLE_MEMBER

- AuthVO.java

```
@Data
public class AuthVO implements GrantedAuthority {
    private String username;
    private String auth;

    @Override
    public String getAuthority() {
        return auth;
    }
}
```



- GrantedAuthority 인터페이스

- 권한 정보 추출

```
public interface GrantedAuthority extends Serializable {
    String getAuthority();
}
```

- MemberVO.java

@Data

@NoArgsConstructor

@AllArgsConstructor

public class MemberVO {

private String username;

private String password;

private String email;

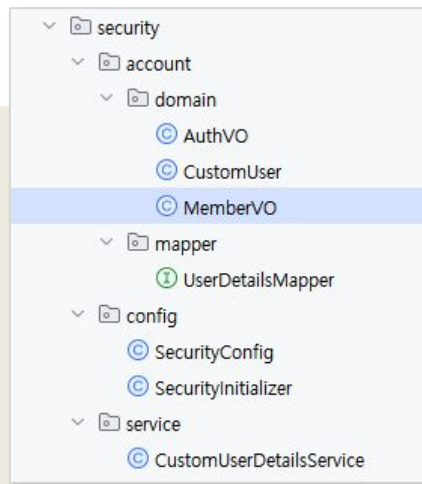
private Date regDate;

private Date updateDate;

private List<AuthVO> authList;

// 권한 목록, join 처리 필요

}

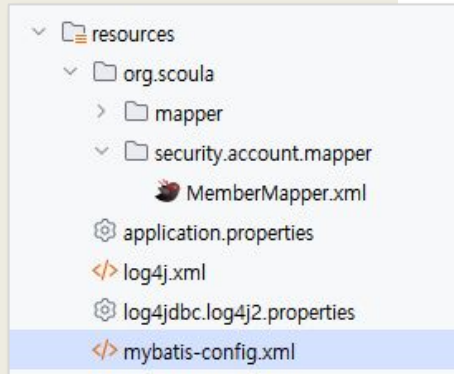


- resources::mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
  <settings>
    <setting name="mapUnderscoreToCamelCase" value="true" />
  </settings>

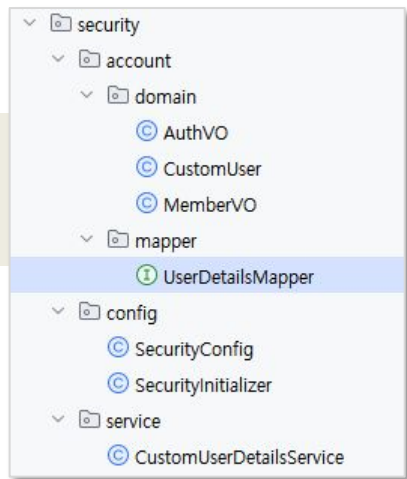
  <typeAliases>
    <package name="org.scoula.security.account.domain" />
  </typeAliases>
</configuration>
```



- UserDetailsMapper.java

```
public interface UserDetailsMapper {  
    public MemberVO get(String username);  
}
```

- tbl_membe와 tbl_member_auth를 Join처리
 - MemberVO의 authList를 설정
 - MyBatis의 resultMap으로 Join 쿼리 결과를 MemberVO로 맵핑



• MyBatis의 <resultMap>

```
select m.username, password, email, reg_date, update_date, auth
from
    tbl_member m left outer join tbl_member_auth a
        on m.username = a.username
where m.username = 'admin';
```

List<AuthVO>로 구성

	username	password	email	reg_date	update_date	auth
1	admin	\$2a\$10\$EsIMfxbJ6NuvwX7l	admin@galapgos.org	2023-11-24 17:24:04	2023-11-24 17:24:04	ROLE_ADMIN
2	admin	\$2a\$10\$EsIMfxbJ6NuvwX7l	admin@galapgos.org	2023-11-24 17:24:04	2023-11-24 17:24:04	ROLE_MANAGER
3	admin	\$2a\$10\$EsIMfxbJ6NuvwX7l	admin@galapgos.org	2023-11-24 17:24:04	2023-11-24 17:24:04	ROLE_USER

1개의 MemberVO 인스턴스로 구성

@Data

```
public class MemberVO {
    private String username;
    private String password;
    private String email;
    private Date regDate;
    private Date updateDate;

    private List<AuthVO> authList;
}
```

- **ResultMap**

- select 쿼리 결과를 VO객체로 맵핑하는 규칙 정의

```
<ResultMap type="VO 객체 클래스명" id="식별자">
  <id property="VO 속성명" column="컬럼명"/>
  <result property="VO 속성명" column="컬럼명"/>
  ...
</ResultMap>
```

- 1:N의 Join 결과 맵핑

```
<ResultMap type="VO 객체 클래스명" id="식별자">
  <id property="VO 속성명" column="컬럼명"/>
  <result property="VO 속성명" column="컬럼명"/>
  ...
  <collection property="부모객체의 컬렉션 속성명" resultMap="자식 테이블 맵핑 resultMap ID">
</ResultMap>
```

- MyBatis의 <resultMap>

```
<resultMap id="memberMap" type="MemberVO">
  <id property="username" column="username" />
  <result property="password" column="password" />
  <result property="email" column="email" />
  <result property="regDate" column="reg_date" />
  <result property="updateDate" column="update_date" />

  <collection property="authList" resultMap="authMap" />
</resultMap>

<resultMap id="authMap" type="AuthVO">
  <result property="username" column="username" />
  <result property="auth" column="auth" />
</resultMap>
```

List<AuthVO>로 구성해 줌

• resources :: UserDetailsMapper.xml

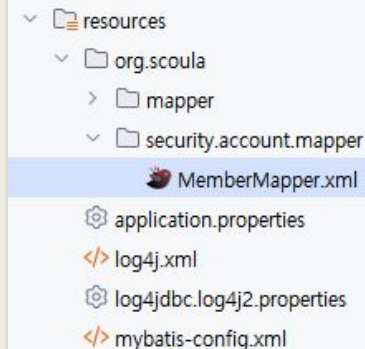
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="org.scoula.security.account.mapper.UserDetailsMapper">

    <resultMap id="authMap" type="AuthVO">
        <result property="username" column="username" />
        <result property="auth" column="auth" />
    </resultMap>

    <resultMap id="memberMap" type="MemberVO">
        <id property="username" column="username" />
        <result property="username" column="username" />
        <result property="password" column="password" />
        <result property="email" column="email" />
        <result property="regDate" column="reg_date" />
        <result property="updateDate" column="update_date" />

        <collection property="authList" resultMap="authMap" />
    </resultMap>
```



- resources :: UserDetailsMapper.xml

```
<select id="get" resultMap="memberMap">
    select m.username, password, email, reg_date, update_date, auth
    from
        tbl_member m left outer join tbl_member_auth a
            on m.username = a.username
    where m.username = #{username}
</select>
</mapper>
```

- SecurityConfig.java

```
@Configuration
```

```
@EnableWebSecurity
```

```
@Log4j
```

```
@MapperScan(basePackages = {"org.scoula.security.account.mapper"})
```

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {
```

```
...
```

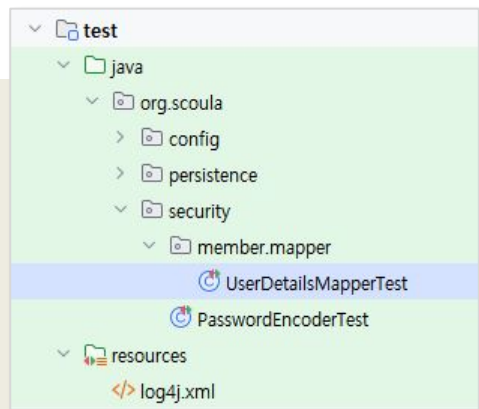
```
}
```


- test :: UserDetailsMapperTest.java

```
@ExtendWith(SpringExtension.class)
@ContextConfiguration(classes = { RootConfig.class, SecurityConfig.class })
@Log4j
public class UserDetailsMapperTest {
    @Autowired
    private UserDetailsMapper mapper;

    @Test
    public void testGet() {
        MemberVO member = mapper.get("admin");
        log.info(member);

        for(AuthVO auth : member.getAuthList()) {
            log.info(auth);
        }
    }
}
```



```
INFO : org.scoula.security.account.mapper.UserDetailsMapperTest - MemberVO(username=admin,
password=$2a$10$EsImfbJ6NuvwX7MDj4WqOYFzLU9U/lldCyn0nic5dFo3VfJYrXYC, email=admin@galapgos.org, regDate=Wed Jul 24 12:01:52 KST 2024,
updateDate=Wed Jul 24 12:01:52 KST 2024, authList=[AuthVO(username=admin, auth=ROLE_ADMIN), AuthVO(username=admin, auth=ROLE_MANAGER),
AuthVO(username=admin, auth=ROLE_MEMBER)])
INFO : org.scoula.security.account.mapper.UserDetailsMapperTest - AuthVO(username=admin, auth=ROLE_ADMIN)
INFO : org.scoula.security.account.mapper.UserDetailsMapperTest - AuthVO(username=admin, auth=ROLE_MANAGER)
INFO : org.scoula.security.account.mapper.UserDetailsMapperTest - AuthVO(username=admin, auth=ROLE_MEMBER)
```

UserDetails 사용하기

- UserDetails 인터페이스

- Spring 인증에 필요한 필수 항목 정의 규약

UserDetails

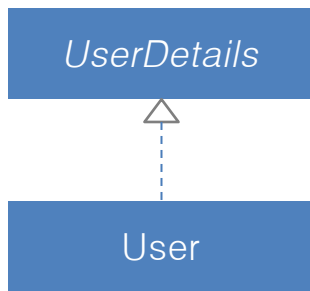
리턴값	메서드	설명
String	getUsername()	사용자 id 리턴
String	getPassword()	비밀번호 리턴
Collection <? extends GrantedAuthority>	getAuthorities()	권한 목록 리턴
boolean	isAccountNonExpired	계정 만료 여부 리턴. true: 유효, false: 만료
boolean	isAccountNonLocked	계정 잠김 여부 리턴. true: 유효, false: 잠김
boolean	isCredentialsNonExpired	신임장 만기 만료 여부 리턴. true: 유효, false: 만료
boolean	isEnabled	계정 사용가능 여부 리턴. true: 사용가능, false: 사용불가능

- UserDetails.java

```
public interface UserDetails extends Serializable {  
    Collection<? extends GrantedAuthority> getAuthorities();  
  
    String getPassword();  
  
    String getUsername();  
  
    boolean isAccountNonExpired();  
  
    boolean isAccountNonLocked();  
  
    boolean isCredentialsNonExpired();  
  
    boolean isEnabled();  
}
```

- **User**

- 스프링에서 정의된 **UserDetails** 구현체
- 스프링 시큐리티에서 요구하는 최소 정보만 유지



• User.java

```
public class User implements UserDetails, CredentialsContainer {  
    private static final long serialVersionUID = 580L;  
    private static final Log logger = LoggerFactory.getLog(User.class);  
  
    private String password;  
    private final String username;  
    private final Set<GrantedAuthority> authorities;  
  
    private final boolean accountNonExpired;  
    private final boolean accountNonLocked;  
    private final boolean credentialsNonExpired;  
    private final boolean enabled;
```

- **User.java**

```
public User(String username, String password, Collection<? extends GrantedAuthority> authorities) {  
    this(username, password, true, true, true, true, authorities);  
}  
  
public User(String username, String password, boolean enabled, boolean accountNonExpired,  
    boolean credentialsNonExpired, boolean accountNonLocked,  
    Collection<? extends GrantedAuthority> authorities) {  
    Assert.isTrue(username != null && !"".equals(username) && password != null,  
        "Cannot pass null or empty values to constructor");  
    this.username = username;  
    this.password = password;  
    this.enabled = enabled;  
    this.accountNonExpired = accountNonExpired;  
    this.credentialsNonExpired = credentialsNonExpired;  
    this.accountNonLocked = accountNonLocked;  
    this.authorities = Collections.unmodifiableSet(sortAuthorities(authorities));  
}
```

- **User.java**

```
public Collection<GrantedAuthority> getAuthorities() { return this.authorities; }

public String getPassword() { return this.password; }

public String getUsername() { return this.username; }

public boolean isEnabled() { return this.enabled; }

public boolean isAccountNonExpired() { return this.accountNonExpired; }

public boolean isAccountNonLocked() { return this.accountNonLocked; }

public boolean isCredentialsNonExpired() { return this.credentialsNonExpired; }

public void eraseCredentials() {this.password = null; }
```


- **User.java**

```
private static SortedSet<GrantedAuthority> sortAuthorities(Collection<? extends GrantedAuthority> authorities) {
    Assert.notNull(authorities, "Cannot pass a null GrantedAuthority collection");
    SortedSet<GrantedAuthority> sortedAuthorities = new TreeSet(new AuthorityComparator());
    Iterator var2 = authorities.iterator();

    while(var2.hasNext()) {
        GrantedAuthority grantedAuthority = (GrantedAuthority)var2.next();
        Assert.notNull(grantedAuthority, "GrantedAuthority list cannot contain any null elements");
        sortedAuthorities.add(grantedAuthority);
    }

    return sortedAuthorities;
}
```

• User.java

```
public static UserBuilder builder() {  
    return new UserBuilder();  
}  
  
public static UserBuilder withUsername(String username) {  
    return builder().username(username);  
}  
  
public static UserBuilder withUserDetails(UserDetails userDetails) {  
    return  
withUsername(userDetails.getUsername()).password(userDetails.getPassword()).accountExpired(!userDetails.isAccountNonExpired()).account  
Locked(!userDetails.isAccountNonLocked()).authorities(userDetails.getAuthorities()).credentialsExpired(!userDetails.isCredentialsNonExpired())  
.disabled(!userDetails.isEnabled());  
}
```

• User.java

```
public static final class UserBuilder {  
    private String username;  
    private String password;  
    private List<GrantedAuthority> authorities;  
    private boolean accountExpired;  
    private boolean accountLocked;  
    private boolean credentialsExpired;  
    private boolean disabled;  
    private Function<String, String> passwordEncoder;  
  
    private UserBuilder() {  
        this.passwordEncoder = (password) -> {  
            return password;  
        };  
    }  
    ...  
}
```

- **User.java**

```
public UserDetails build() {  
    String encodedPassword = (String)this.passwordEncoder.apply(this.password);  
    return new User(this.username, encodedPassword,  
        !this.disabled, !this.accountExpired, !this.credentialsExpired, !this.accountLocked,  
        this.authorities);  
}  
}  
  
private static class AuthorityComparator implements Comparator<GrantedAuthority>, Serializable {  
    ...  
}  
}
```

- **UserDetailsService** 인터페이스

- 로그인 사용자의 상세 내역을 제공하는 규약 정의

```
public interface UserDetailsService {  
    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;  
}
```

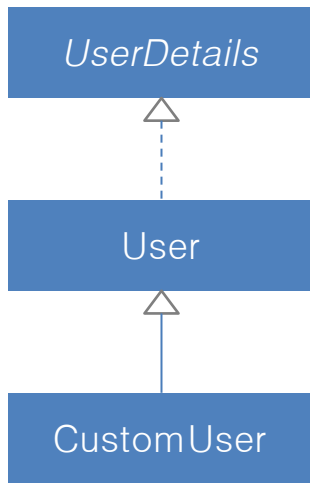
- 매개변수: 찾고자 하는 **username** 문자열
- 리턴값: 찾은 **UserDetails** 객체
- 예외: 찾고자 하는 **username**이 없는 경우 **UsernameNotFoundException** 발생

- **UserDetailsService** 인터페이스 구현

- **UserDetailsMapper** 주입으로 멤버 정보 추출

- CustomUser

- User 클래스 상속
- 애플리케이션에서 필요한 사용자 추가 정보(MemberVO) 관리

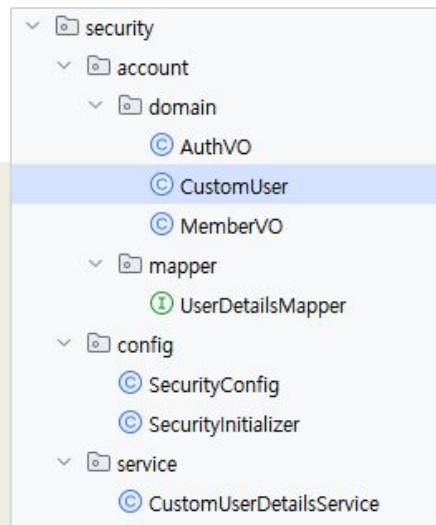


• CustomUser.java

@Getter

@Setter

```
public class CustomUser extends User {  
    private MemberVO member;           // 실질적인 사용자 데이터  
  
    public CustomUser(String username, String password,  
        Collection<? extends GrantedAuthority> authorities) {  
        super(username, password, authorities);  
    }  
  
    public CustomUser(MemberVO vo) {  
        super(vo.getUsername(), vo.getPassword(), vo.getAuthList());  
        this.member = vo;  
    }  
}
```



• CustomUserDetailsService.java

```
package org.scoula.security.service;
```

```
...
```

```
@Log4j
```

```
@Component
```

```
@RequiredArgsConstructor
```

```
public class CustomUserDetailsService implements UserDetailsService {
```

```
    private final UserDetailsMapper mapper;
```

```
    @Override
```

```
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
```

```
        MemberVO vo = mapper.get(username);
```

```
        if(vo == null) {
```

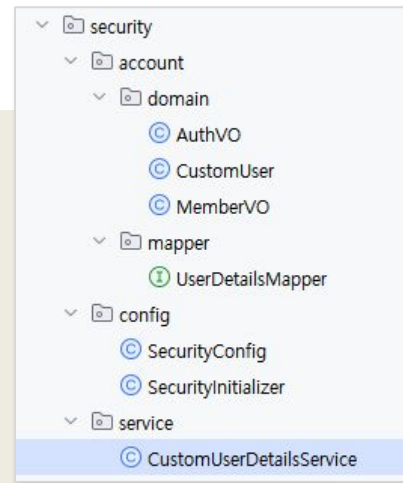
```
            throw new UsernameNotFoundException(username + "은 없는 id입니다.");
```

```
        }
```

```
        return new CustomUser(vo);
```

```
    }
```

```
}
```



- 스프링 시큐리티 설정
 - CustomUserService를 Bean으로 등록
 - AuthenticationManagerBuilder에서 CustomUserService를 설정

• SecurityConfig.java

```
@Configuration
@EnableWebSecurity
@Log4j
@MapperScan(basePackages = {"org.scoula.security.account.mapper"})
@ComponentScan(basePackages = {"org.scoula.security"})
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private final UserDetailsService userDetailsService;

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

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // in memory user 정보 삭제 → UserDetailsService와 같이 사용 불가
        auth
            .userDetailsService(userDetailsService)
            .passwordEncoder(passwordEncoder());
    }
}
```

- 컨트롤러에서 UserDetails 얻기

- Principal 주입

- getName() 메서드를 통해 username 얻을 수 있음

```
...
import java.security.Principal;

...

@GetMapping("/member")
public void doMember(Principal principal) {
    log.info("username = " + principal.getName());
}
```

INFO : org.scoula.controller.SecurityController - username =
user0

- 컨트롤러에서 UserDetails 얻기

- Authentication 주입

- getPrincipal()을 통해 UserDetails 추출

```
...
import org.springframework.security.core.Authentication;
...

@GetMapping("/member")
public void doMember(Authentication authentication) {
    UserDetails userDetails = (UserDetails)authentication.getPrincipal();

    log.info("username = " + userDetails.getUsername());
}
```

INFO : org.scoula.controller.SecurityController - username = **user0**

- 컨트롤러에서 UserDetails 얻기
 - @AuthenticationPrincipal 애노테이션
 - CustomUser 주입을 요구할 수 있음

```
...
import org.springframework.security.core.annotation.AuthenticationPrincipal;
...

@GetMapping("/admin")
public void doAdmin(@AuthenticationPrincipal CustomUser customUser) {
    MemberVO member = customUser.getMember();
    log.info("username = " + member);
    ...
}
```

INFO : org.scoula.controller.SecurityController - username ~~M~~emberVO(username=admin, password=\$2a\$10\$EsImfxbJ6NuvwX7MDj4WqOYFzLU9U/IddCyn0nic5dFo3VfJYrXYC, email=admin@galapgos.org, regDate=Wed Jul 24 12:01:52 KST 2024, updateDate=Wed Jul 24 12:01:52 KST 2024, authList=[AuthVO(username=admin, auth=ROLE_ADMIN), AuthVO(username=admin, auth=ROLE_MANAGER), AuthVO(username=admin, auth=ROLE_MEMBER)])

- **spring security tag lib 사용**

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

- **<sec:authentication>**

- 인증이 된 경우 사용자 정보 출력

- `<sec:authentication property="principal">`
 - principal : UserDetailsService가 리턴한 객체
--> loadUserByUsername()에서 반환한 User 객체
 - 로그인 사용자명 출력

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

- **<sec:authorize access="">**

- 인증 여부 판단 및 접근 권한 체크

- **isAnonymous()** 로그인하지 않은 경우 **true**
 - **isAuthenticated()** 로그인한 경우 **true**
 - **isFullyAuthenticated()** remember-me 이외의 경로로 인증된 경우 **true**
 - **isRememberMe()** remember-me로 인증된 사용자인 경우 **true**
 - **hasRole([role])** 현재 주체에 지정된 역할이 있는지 여부.
역할이 'ROLE_'로 시작하지 않으면 자동 추가
 - **hasAnyRole([role1,role2])** 제공된 역할 목록에 있는지 여부
 - **hasAuthority([authority])** 현재 주체에 지정된 권한.
 - **hasAnyAuthority([authority1,authority2])**
 - **principal** 현재 사용자를 나타내는 주체 개체
 - **authentication** SecurityContext Authentication.
 - **permitAll** 항상 허용
 - **denyAll** 항상 불허

- index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="sec" %>
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
<h1>환영합니다.</h1>

<sec:authorize access="isAnonymous()"> <!-- 로그인 안한 경우 -->
  <a href="/security/login">로그인</a>
</sec:authorize>

<sec:authorize access="isAuthenticated()"> <!-- 로그인 한 경우 -->
  <sec:authentication property="principal.username"/>
  <form action="/security/logout" method="post">
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
    <input type="submit" value="로그아웃"/>
  </form>
</sec:authorize>

</body>
</html>
```

환영합니다.

[로그인](#)

환영합니다.

admin

로그아웃

SecurityConfig 전체 코드

- **SecurityConfig.java**

```
package org.scoula.security.config;

...
@Configuration
@EnableWebSecurity
@Log4j
@MapperScan(basePackages = {"org.scoula.security.account.mapper"})
@ComponentScan(basePackages = {"org.scoula.security"})
@RequiredArgsConstructor
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private final UserDetailsService userDetailsService;

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

• SecurityConfig.java

```
// 문자셋 필터
public CharacterEncodingFilter encodingFilter() {
    CharacterEncodingFilter encodingFilter = new CharacterEncodingFilter();
    encodingFilter.setEncoding("UTF-8");
    encodingFilter.setForceEncoding(true);
    return encodingFilter;
}

@Override
public void configure(HttpSecurity http) throws Exception {
    http.addFilterBefore(encodingFilter(), CsrfFilter.class);

    // 경로별 접근 권한 설정
    http.authorizeRequests()
        .antMatchers("/security/all").permitAll()
        .antMatchers("/security/admin").access("hasRole('ROLE_ADMIN')")
        .antMatchers("/security/member").access("hasRole('ROLE_MEMBER')");
}
```

• SecurityConfig.java

```
http.formLogin()
    .loginPage("/security/login")
    .loginProcessingUrl("/security/login")
    .defaultSuccessUrl("/");

http.logout()
    .logoutUrl("/security/logout")           // POST: 로그아웃 호출 url
    .invalidateHttpSession(true)             // 세션 invalidate
    .deleteCookies("remember-me", "JSESSION-ID") // 삭제할 쿠키 목록
    .logoutSuccessUrl("/security/logout");     // GET: 로그아웃 이후 이동할 페이지
}

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    log.info("configure .....");
    auth
        .userService(userDetailsService)
        .passwordEncoder(passwordEncoder());
}
}
```

// 로그아웃 설정 시작

1. PasswordEncoder
2. Spring Security config
3. UserDetails
4. Security Filter Chain
5. 접근 제한 설정
6. 인증, 인가