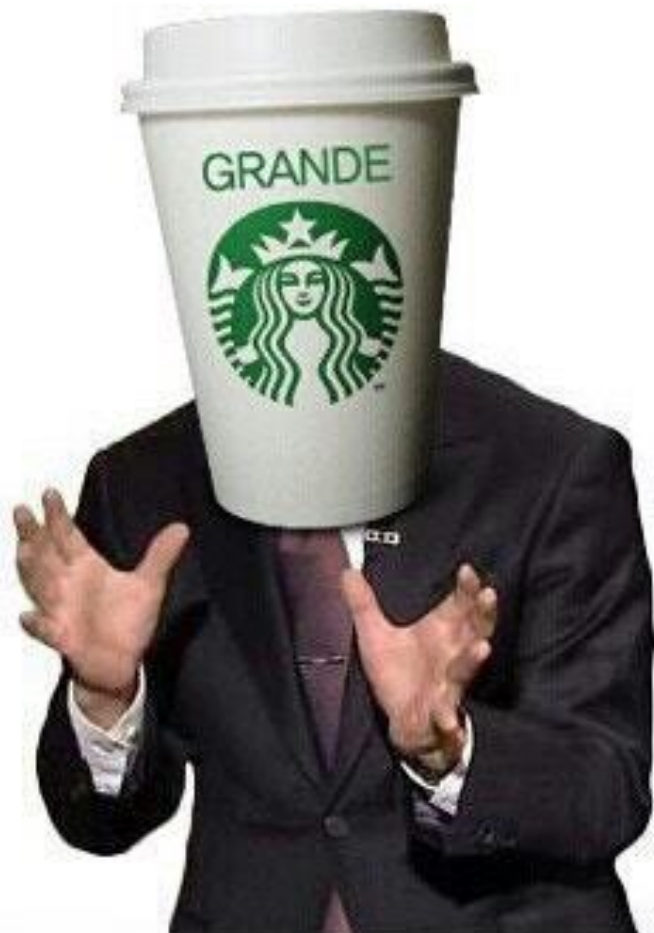




It's Your Life

with





여러분이 생각하는 보안의 이미지는 어떤가요!?

그란데 말입니다





하지만 우리가 보안을 짚다면!?

그란데 말입니다



이쯤되어 다시 올리는 우리 아파트 기적의 보안



💬 16

↕ 4,009

❤ 997



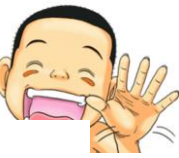


Error

이 비밀번호는 이미 starboy98 사용자가 쓰고
있습니다. 다시 시도해 주십시오.

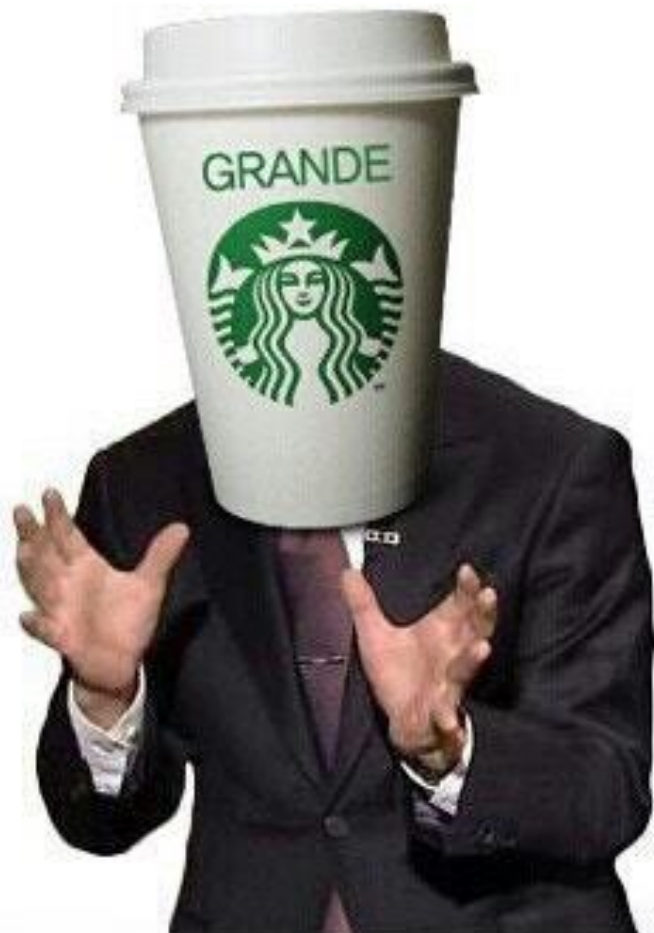
OK







Spring Security



그란데 말입니다

그란데 말입니다

그럼 Spring Security 는 어떤 장점이 있을까요?

1. 다양한 보안 솔루션 제공
2. 유연한 설정이 가능
3. 보안 기능 자동 삽입(헤더, CSRF 등)
4. 다양한 인증 방식 지원, OAuth2 지원
5. 세션을 자동 관리
6. 암호화 기능 내장
7. 보안 테스트를 지원

1. 다양한 보안 솔루션 제공



사장님, 여기
야근
무한 리필이요

웹툰 삼우실

포괄임금제 = 공짜야근?

이비 씨, 이번 주는
야근 좀 해!



2. 유연한 설정이 가능



3. 보안 기능 자동 삽입(헤더, CSRF 등)

5. 세션을 자동 관리



자동이요



4. 다양한 인증 방식 지원



6. 암호화 기능 내장
7. 보안 테스트 제공





CSRF,

XSS



그란데 말입니다

그란데 말입니다

CSFR 이랑 XSS 가 뭘까요!?

CSRF(Cross Site Request Forgery)
→ 사이트간 요청 위조

XSS(Cross Site Scripting)
→ 사이트에 특정 악성 스크립트 삽입

CSRF(Cross Site Request Forgery)



- 특정 서비스의 아이디 & 비밀번호 변경 요청 URL 이 아래와 같다면?
- <http://service.com/user/update/{id}/{password}>

<http://service.com/user/update/admin/1234>



나쁜 X



<https://www.boannews.com/media/view.asp?idx=9481>



XSS(Cross Site Script)



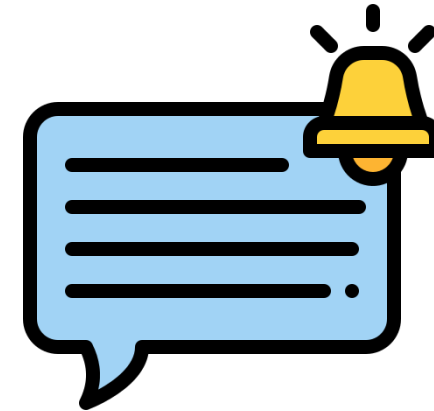
자유게시판

검색어를 입력해주세요.

검색

번호	제목	이름	아이디	작성일	조회수
6	좋은글 공유합니다~~~	hong	홍길동	2022-08-26	40
5	가입인사	hong	홍길동	2022-08-26	10
4	ㅋㅋㅋㅋ 최고예요	.	won	2022-08-25	13
3	안녕하세요~~ ㅎㅎ			2022-08-25	5
2	반갑습니당	심청이	shim	2022-08-16	11
1	안녕하세요	홍길동	hong	2022-08-16	44

이전 1 page / 1 pages 다음



<script />

나쁜 X

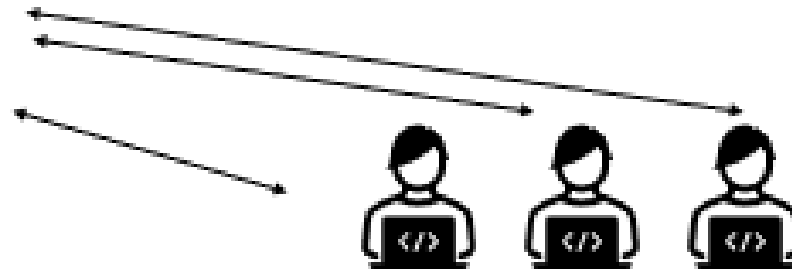
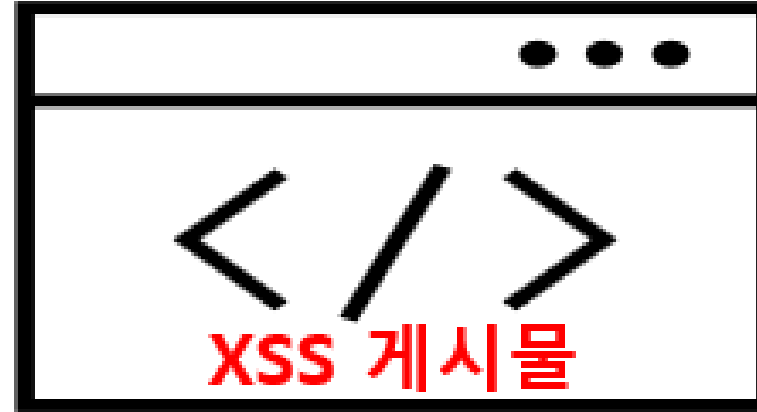




공격자



XSS 공격으로 웹페이지에
악성 게시글 작성



사이트 이용자

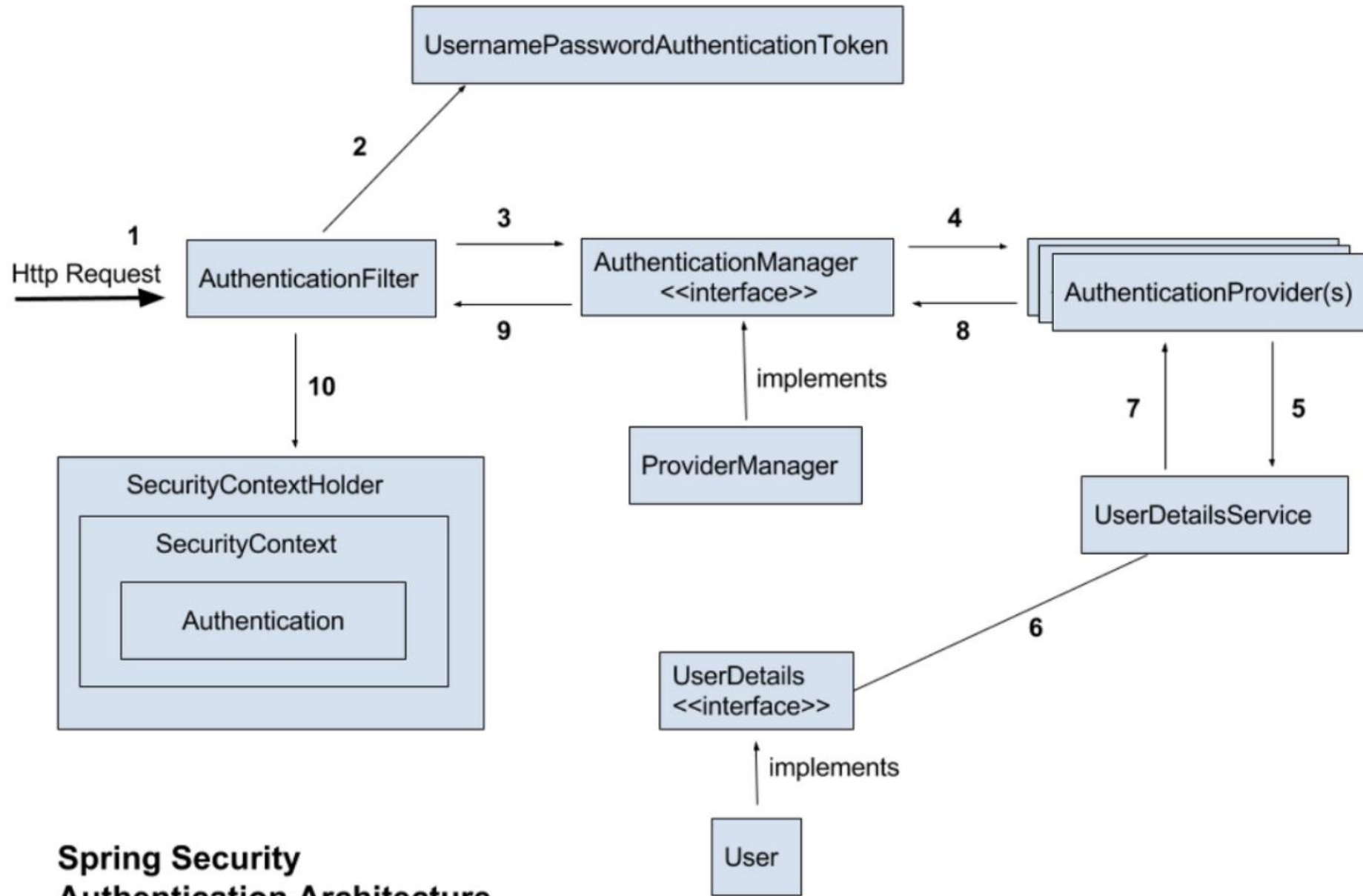
XSS 게시물 접속시,
해당 사이트가 명령한 것으로 해석하고
악성 스크립트 실행

<https://www.boannews.com/media/view.asp?idx=100075>





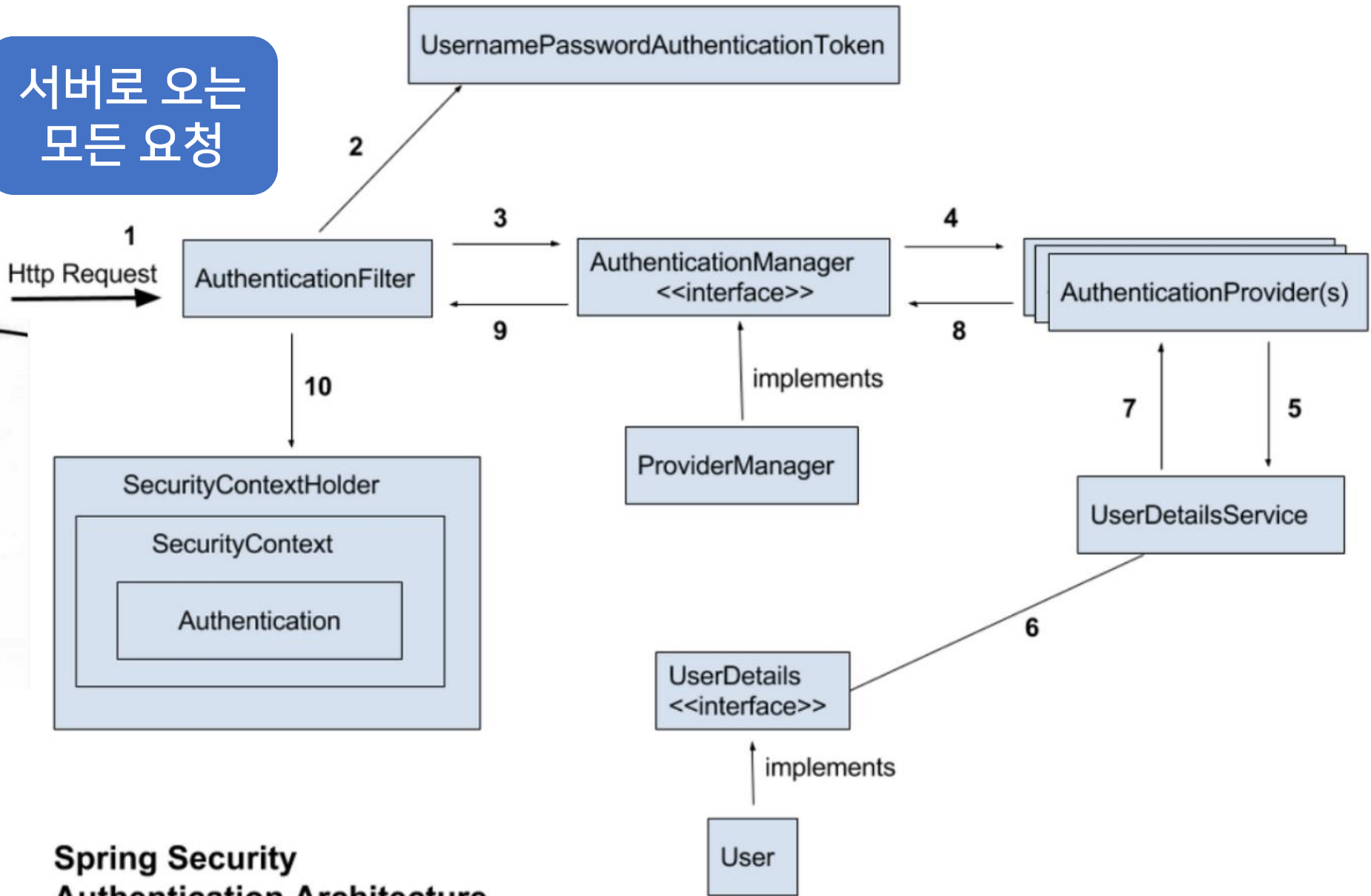
Spring Security



Spring Security Authentication Architecture



서버로 오는
모든 요청



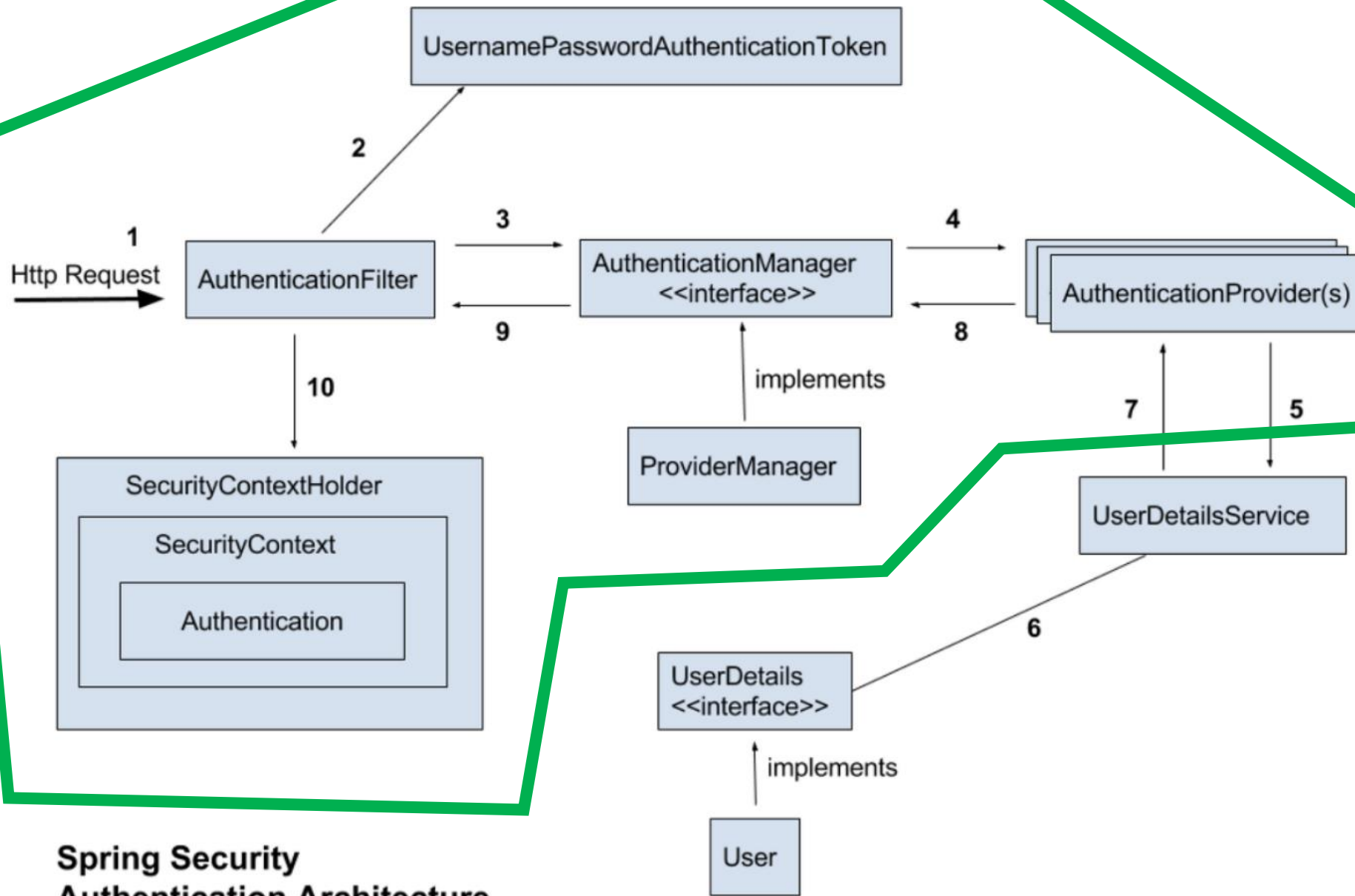
Spring Security Authentication Architecture

Chathuranga Tennakoon
www.springbootdev.com



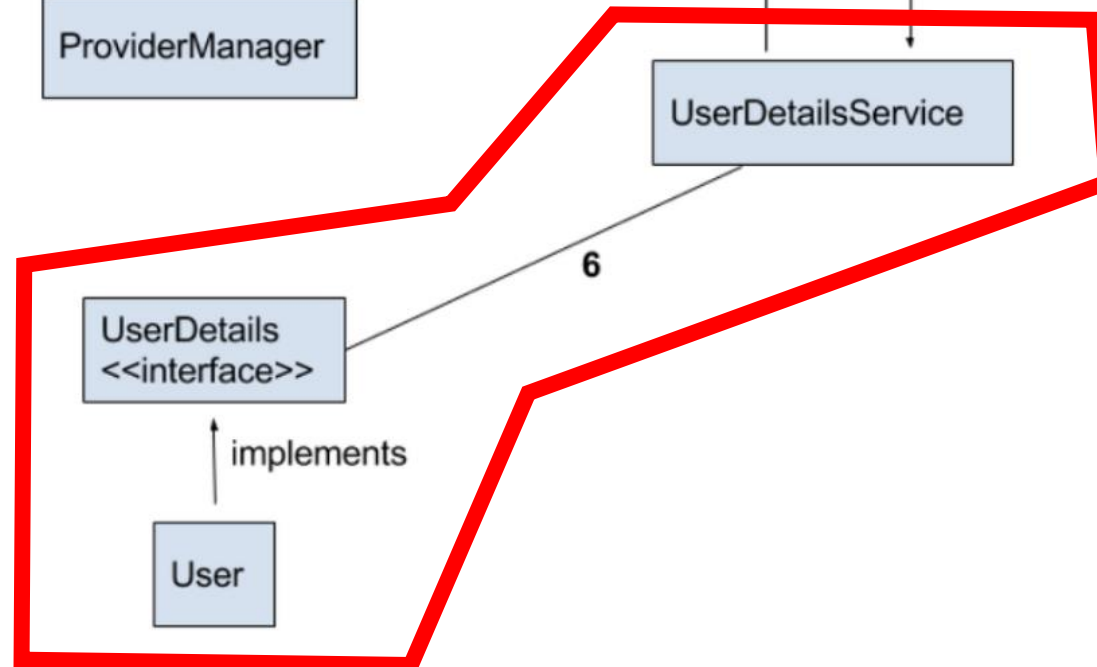
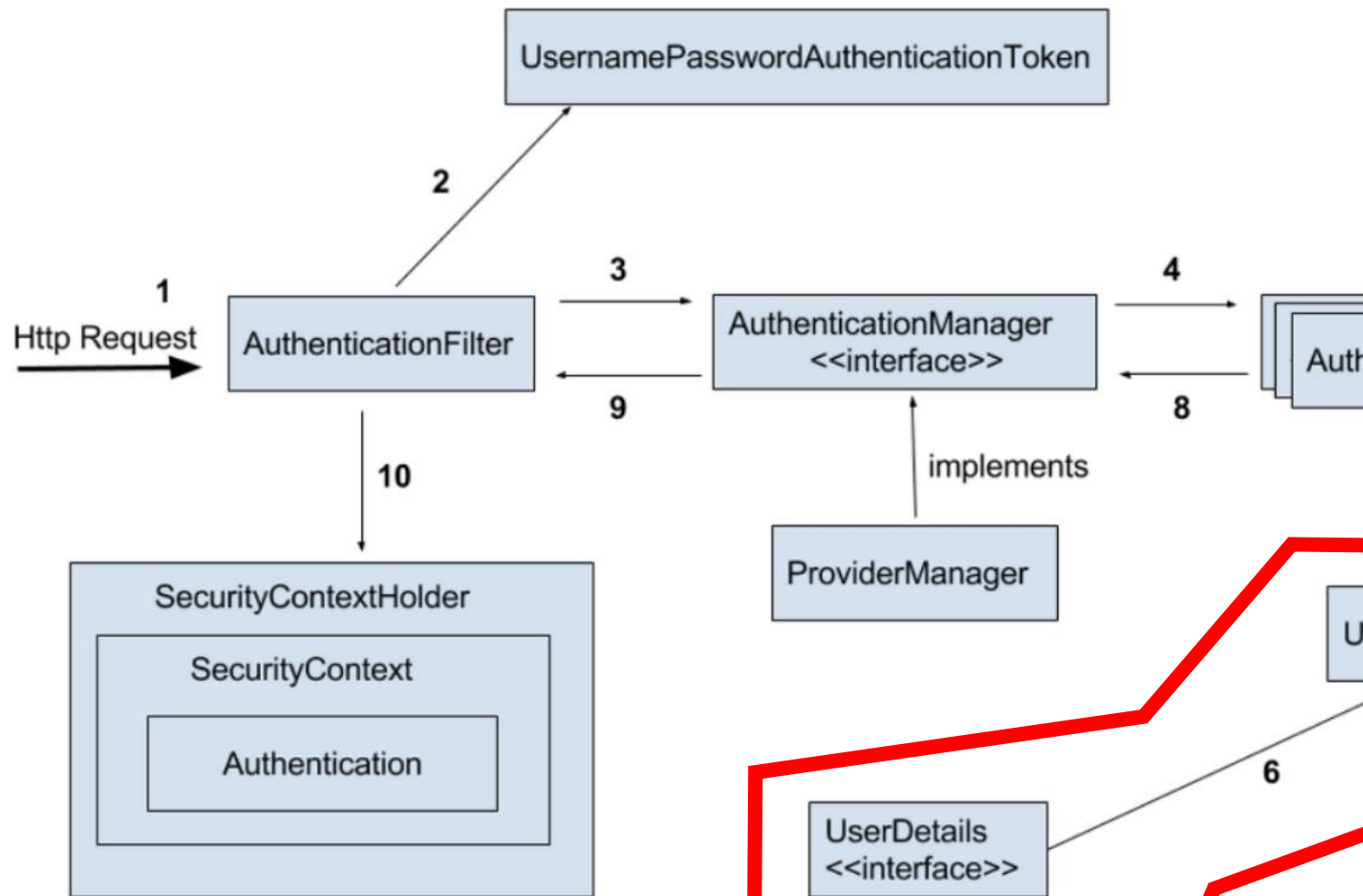


세상은 1등만 기억해



Spring Security Authentication Architecture

Chathuranga Tennakoon
www.springbootdev.com



Spring Security Authentication Architecture



실제 여러분들이 해야할 일

1. 설정 하기
2. 회원 데이터 처리하기



Spring Security



적용하기



build.gradle 에

의존성 추가하기!


```
ext {  
    junitVersion = '5.9.2'  
    springVersion = '5.3.37'  
    lombokVersion = '1.18.30'  
    springSecurityVersion='5.8.13'  
}
```

스프링 시큐리티 버전을 명시해 줍시다!



<https://github.com/xenosign/spring-code-repo/blob/main/gradle/security.gradle>



```
// 스프링 시큐리티
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}")

// 암호화
implementation 'org.springframework.security:spring-security-crypto'
```



```
// 암호화
```

```
implementation 'org.springframework.security:spring-security-crypto'
```

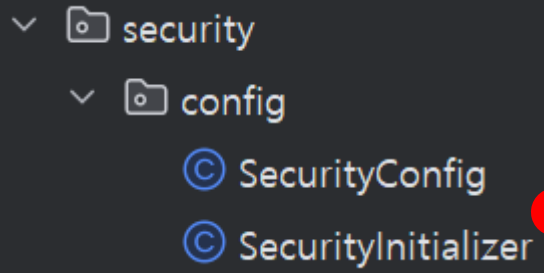
```
implementation 'org.springframework.security:spring-security-core:5.8.0'
```



요 부분은 시큐리티 의존성에 포함이 되어있으므로
해당 부분은 삭제하시면 됩니다!




Security 설정 및 적용하기




▼ security

▼ config

- © SecurityConfig
- © SecurityInitializer



Spring Security 관련 설정을 하는 Config 파일과
적용하는 Initializer 파일을 추가해 봅시다!





```
import ...
```

```
@Configuration    Tetz
```

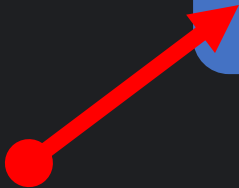
```
@EnableWebSecurity
```

```
@RequiredArgsConstructor
```

```
@Log4j
```

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

일단 아무런 세팅 없이 인터페이스를 받아서
구현만 해봅시다!



```
import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

public class SecurityInitializer extends AbstractSecurityWebApplicationInitializer {  👤 kdtTet

}

```



일단 아무런 세팅 없이 인터페이스를 받아서
구현만 해봅시다!

어제 적용한 UserConfig 는 이제 필요 없으니 주석 처리
+ 새롭게 만든 SecurityConfig 를 적용하기!

© RootConfig

© ServletConfig

© WebConfig

controller

domain

19

20

21

22

23

@Configuration kdtTetz

//@Import(UserConfig.class)

@Import(SecurityConfig.class)

public class WebConfig extends Abst

© AuthenticationIntercept

© UserConfig

service

oauth

© KakaoOauthService

user

© UserService

s

ation.properties

.xml

dbc.log4j2.properties

atis-config.xml

```
10 @Configuration  @kdtTetz
11 public class UserConfig implements WebMvcConfigurer {
12     @Bean  @kdtTetz
13     public BCryptPasswordEncoder passwordEncoder() {
14         return new BCryptPasswordEncoder();
15     }
16
17     // // 인터셉터 추가
18     // @Override
19     // public void addInterceptors(InterceptorRegistry registry) {
20     //     System.out.println("시큐리티 인터셉터가 등록되었습니다.");
21     //     registry.addInterceptor(new AuthenticationInterceptor())
22     //         .addPathPatterns("/**")
23     //         .excludePathPatterns("/", "/user/**", "/resources/**");
24     // }
25
```

UserConfig 에 구현한 Interceptor 와
스프링 시큐리티가 동시에 적용 되므로 인터셉터 코드는 주석 처리!





© AuthenticationInterceptor

© UserConfig

service

oauth

© KakaoOAuthService

user

© UserService

authentication.properties

xml

jdbc.log4j2.properties

atis-config.xml

10

@Component kdtTetz

11

public class AuthenticationInterceptor implements HandlerInterceptor {

12

// @Override

13

// public boolean preHandle(HttpServletRequest request, HttpServletResponse

14

// throws Exception {

15

// String requestUri = request.getRequestURI();

16

// System.out.println("요청 URI: " + requestUri);

17

//

18

// HttpSession session = request.getSession();

19

// if (session.getAttribute("loginUser") == null) {

20

// System.out.println("로그인 안됨. 리다이렉트 중: /user/login");

21

// response.sendRedirect("/user/login");

22

// return false;

23

// }

24

// return true;

25

// }

26

}

Interceptor 구현 코드도 주석 처리 하기



Security 적용

확인하기!



Please sign in

localhost:8080/login

Gmail GDR GitHub NCL YouTube Zoom O365 NAVER INF Claude ChatGPT MLP KB강의자료 KB상세일정 KB상세일정1 KE



Please sign in



서버로 가는 모든 요청을 Spring Security 가 훔치기 때문에
Spring Security 가 로그인이 안되었다고 판단
자신이 제공하는 로그인 페이지로 보내버립니다!



Security

설정 시작!



```
@Configuration new *
@EnableWebSecurity
@RequiredArgsConstructor
@Log4j
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    // 문자셋 필터
    public CharacterEncodingFilter encodingFilter() { 1 usage new *
        CharacterEncodingFilter encodingFilter = new CharacterEncodingFilter();
        encodingFilter.setEncoding("UTF-8");
        encodingFilter.setForceEncoding(true);
        return encodingFilter;
    }
```

스프링 시큐리티 로그인 시 요청을 POST 로 보내도록 권장 되어있습니다!
그 때 한글이 제대로 안보내지는 이슈가 있어서 인코딩 필터를 등록!



그란데 말입니다

그란데 말입니다

왜!? POST 가 권장 될까요!?

GET 방식은 모든 요청 값이
주소 창에 노출이 됩니다!

→ 따라서 기본적인 보안을 위해서 POST 를 권장!

가장 기본 설정이 되는 configure 를 설정해 봅시다!






```
@Override  Tetz *
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers(✓ "/").permitAll()
        .antMatchers(✓ "/user/**").permitAll()
        .antMatchers(✓ "/security/**").permitAll()
        .antMatchers(✓ "/**").access(attribute: "hasRole('ROLE_MEMBER)");

    http.addFilterBefore(encodingFilter(), CsrfFilter.class);
}
```



@Override Tetz *

```
protected void configure(HttpSecurity http) throws Exception {
```

```
    http.authorizeRequests()  
        .antMatchers( "/").permitAll()  
        .antMatchers( "/user/**").permitAll()  
        .antMatchers( "/security/**").permitAll()  
        .antMatchers( "/**").access( attribute: "hasRole('ROLE_MEMBER')");  
  
    http.addFilterBefore(encodingFilter(), CsrfFilter.class);
```

http 리퀘스트에 대해서
인증 여부를 체크하는 메서드 사용

인증 없이 접근이 가능해야 하는 주소들
루트 페이지, 회원 관련 기능, 스프링 시큐리티
관련 요청은 누구나 접속이 가능하도록
permitAll() 처리

```
@Override
protected void configure(HttpSecurity http) {
    http.authorizeRequests()
        .antMatchers("/").permitAll()
        .antMatchers("/user/**").permitAll()
        .antMatchers("/security/**").permitAll()
        .antMatchers("/**").access(attribute: "hasRole('ROLE_MEMBER')");

    http.addFilterBefore(encodingFilter(), CsrfFilter.class);
}
```



```
@Override  Tetz *
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/).permitAll()
        .antMatchers("/user/**").permitAll()
        .antMatchers("/security/**").permitAll()
        .antMatchers("/**").access(attribute: "hasRole('ROLE_MEMBER')");

    http.addFilterBefore(encodingFilter, ...);
}
```

그 외의 주소 요청은
사용자가 특정 role 을 가지고 있을 때에만
접근이 가능하도록 access + hasRole 처리!



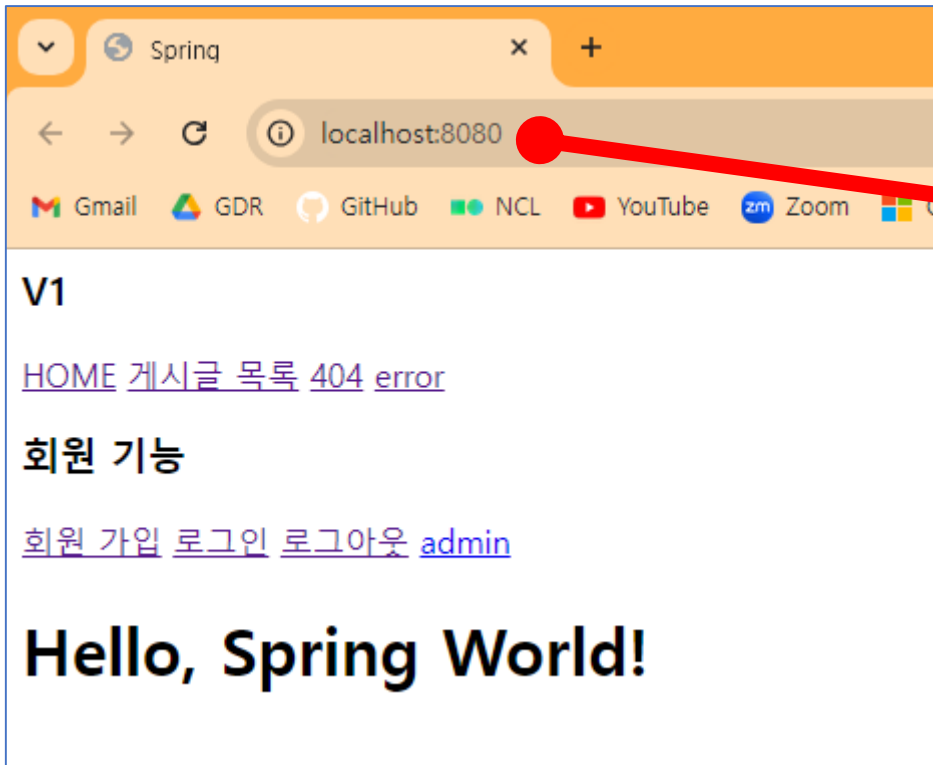
```
@Override  👤 Tetz *  
protected void configure(HttpSecurity http) throws Exception {  
    http.authorizeRequests()  
        .antMatchers(🛡️ "/").permitAll()  
        .antMatchers(🛡️ "/user/**").permitAll()  
        .antMatchers(🛡️ "/security/**").permitAll()  
        .antMatchers(🛡️ "/**").access(attribute: "hasRole('ROLE_MEMBER')");  
  
    http.addFilterBefore(encodingFilter(), CsrfFilter.class);  
}
```

한글에 대한 처리는 CsrfFilter 를 거치기 전에
수행이 되어야 하므로 addFilterBefore 를 통해
한글 처리 필터를 적용 시키는 코드!

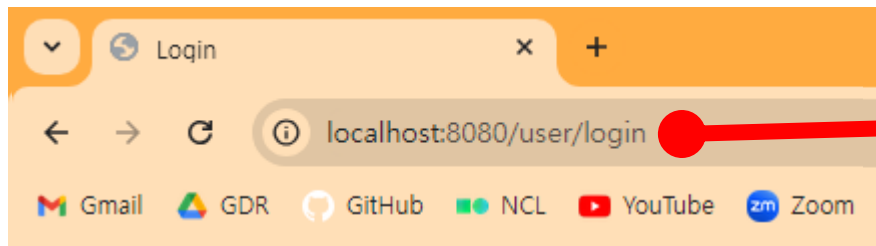


Security

설정 확인



configure 가 적용되어
루트 페이지(/) 에 대한 접근은 누구나
가능한 상태입니다!



/user 요청도 허용을 시켰으므로
문제 없이 접근이 가능한 것 확인 가능!



V1

[HOME](#) [게시글 목록](#) [404 error](#)

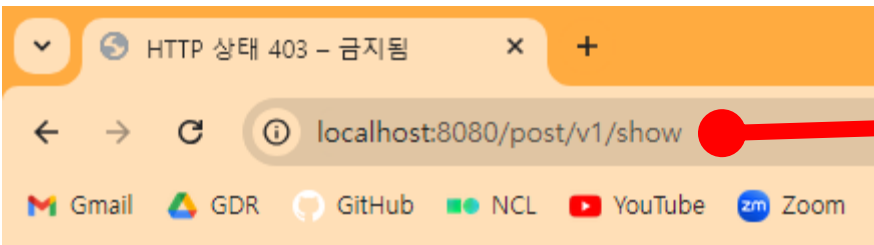
회원 기능

[회원 가입](#) [로그인](#) [로그아웃](#) [admin](#)

로그인

아이디:

비밀번호:



HTTP 상태 403 – 금지됨

타입 상태 보고

메시지 Access Denied

설명 서버가 요청을 이해했으나 승인을 거부합니다.

Apache Tomcat/9.0.91

permitAll() 처리가 안된 요청 주소로 접근을 시도하면 Spring Security 가 403(UnAuthorized) 페이지를 노출 시킵니다!





자, 그럼 이제...

로그인을
구현하자!





로그인 관련

설정하기!





@Override Tetz *


```
protected void configure(HttpSecurity http) throws Exception {
```

```
    http.authorizeRequests()
```

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

```
        .antMatchers( "/user/**).permitAll()
```

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

```
        .antMatchers( "/**).access( attribute )
```

```
    http.formLogin()
```

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

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

```
        .defaultSuccessUrl("/user/member)
```

```
        .failureUrl( authenticationFailureUrl: "/user/login-failed);
```

기존 설정 아래 부분에
formLogin 형태의 설정을 추가!





@Override Tetz *


```
protected void configure(HttpSecurity http) throws Exception {
```

```
    http.authorizeRequests()
```

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

```
        .antMatchers( "/user/**").permitAll()
```

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

```
        .antMatchers( "/**").access( attributeHasAnyRoles(
```

인증이 안된 사용자가 인증이 필요한
요청을 보내면 설정한 주소로 요청을 보냅니다!


→ 우리가 만든 login 페이지로 이동

```
    http.formLogin()
```

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

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

```
        .defaultSuccessUrl("/user/member")
```

```
        .failureUrl( authenticationFailureUrl: "/user/login-failed");
```

```
http.formLogin()  
    .loginPage("/user/login")  
    .loginProcessingUrl("/user/login")  
    .defaultSuccessUrl("/user/member")  
    .failureUrl(authenticationFailureUrl: "/user/login-failed");
```

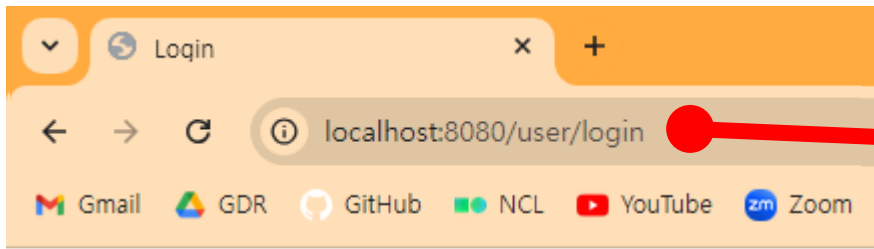
해당 url 로 POST 방식 요청이 들어오면
스프링 시큐리티가 직접
로그인 처리를 합니다

로그인이 성공한 경우와 실패한 경우에
redirect 할 주소 설정 하기!



로그인 관련 설정

확인!



V1

[HOME](#) [게시글 목록](#) [404 error](#)

회원 기능

[회원 가입](#) [로그인](#) [로그아웃](#) [admin](#)

로그인

아이디:

비밀번호:

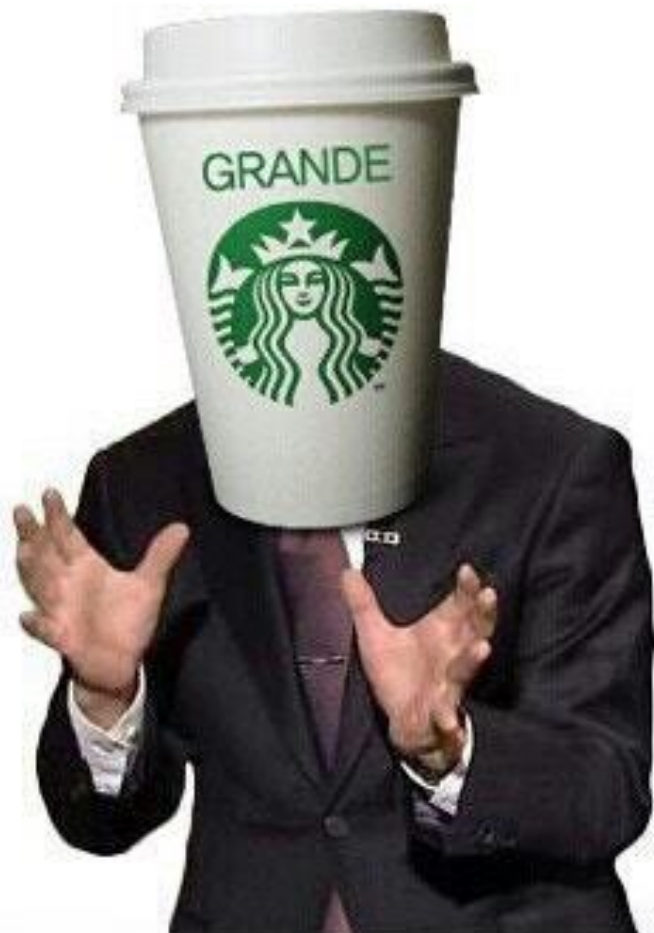
허용이 안된 다른 곳으로 요청을 보내면
자동으로 설정한 /user/login 주소로
리다이렉트가 되어

로그인 페이지가 뜨게 됩니다!



이제는 Security

컨트롤러를 만들 때!



그란데 말입니다

그란데 말입니다

어제 구현한 직접 로그인과 Spring Security 로그인을
하나의 컨트롤러에서 처리하는게 맞을까요!?

서비스 레이어를 나누는 것처럼
두 개는 비슷한 과정이나
처리 과정이 다르기 때문에

컨트롤러를 분리(= 요청 시작 주소를 분리) 하는 것이
정신 건강에 이롭습니다 ☺

@Override Tetz *

```
protected void configure(HttpSecurity http) throws Exception {
```

```
    http.authorizeRequests()
```

```
        .antMatchers(✓ "/").permitAll()
```

```
        .antMatchers(✓ "/user/**").permitAll()
```

```
        .antMatchers(✓ "/security/**").permitAll()
```

```
        .antMatchers(✓ "/**").access( attribute: "hasRole('ADMIN')");
```

```
    http.formLogin()
```

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

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

```
        .defaultSuccessUrl("/security/member")
```

```
        .failureUrl( authenticationFailureUrl: "/security/login-failed");
```



이제 Security 관련 요청은
요청 주소 자체를 /security 로
분리할 예정이므로

로그인 설정의 주소도 전부 변경하기!



controller

> board

> book

> member

> oauth

> post

> todo

> user

© SecurityController

© UserController

security 관련 처리를 위해
SecurityController 만들기!



```
@Controller kdtTetz *
```

```
@RequiredArgsConstructor
```

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

```
public class SecurityController {
```

```
    private final UserService userService;
```

```
    private final String context = "/security";
```

컨트롤러 시작 주소를
/security 주소로 매핑!

jsp 파일도 따로 다를 것이므로
context 도 /security 로 설정

```
@GetMapping(🌐"/login")  👤 kdtTetz  
public String loginPage() {  
    return context + "/login";  
}
```

로그인 페이지를 보여주는 컨트롤러
메서드



```
@GetMapping(🌐"/login-failed")  new *  
public String loginFailPage() {  
    return context + "/login-failed";  
}
```

로그인이 실패하면
로그인 실패 페이지로 연결하기!



Security

프론트 페이지 작업



```
▼ webapp
  ▼ WEB-INF
    ▼ views
      > exception
      > member
      > post
      ▼ security
        JSP admin.jsp
        JSP login.jsp
        JSP login-failed.jsp
        JSP member.jsp
```

각각 필요한 jsp 파일 만들기

<https://github.com/xenosign/spring-code-repo/blob/main/jsp/security/login.jsp>



```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
<%@include file="../header2.jsp"%>
<h1>SECURITY 로그인</h1>
<form action="/security/login" method="post">
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />
    아이디: <input type="text" name="username"><br>
    비밀번호: <input type="password" name="password"><br>
    <input type="submit" value="로그인"/>
</form>
</body>
</html>
```

CSRF 공격을 막기 위해 위와 같은
안보이는 데이터를 서버에 전송하여
해당 요청이 제대로 된 클라이언트에서
왔는지를 체크 합니다!

<https://github.com/xenosign/spring-code-repo/blob/main/jsp/security/login-failed.jsp>



```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
<%@include file="../header2.jsp"%>
    <h1>SECURITY 로그인 실패</h1>
    <a href="/security/login">로그인 페이지로 이동</a>
</body>
</html>
```



Header

업데이트



```
<h3>회원 기능</h3>
<a href="/user/register">회원가입</a>
<a href="/user/login">로그인</a>
<a href="/user/logout">로그아웃</a>
<h3>시큐리티 회원 기능</h3>
<a href="/security/login">로그인</a>
```

Security 로그인 링크를 추가!



Security 컨트롤러

작동 확인



시큐리티 회원 기능

[로그인](#)

SECURITY 로그인

아이디:

비밀번호:

로그인



로그인 링크를 클릭하거나
인가가 안된 요청을 보내면

스프링 시큐리티가 /security/login 으로
리다이렉트를 시키므로

방금 작업한 시큐리티 로그인 페이지가 뜹니다!



그란데 말입니다

그란데 말입니다

실제로 로그인을 해보면 될까요!?

아쉽게도 안됩니다!

왜냐!?

로그인 파트를 전혀 구현한 적이 없으니까요 T-T;;

시큐리티 회원 기능

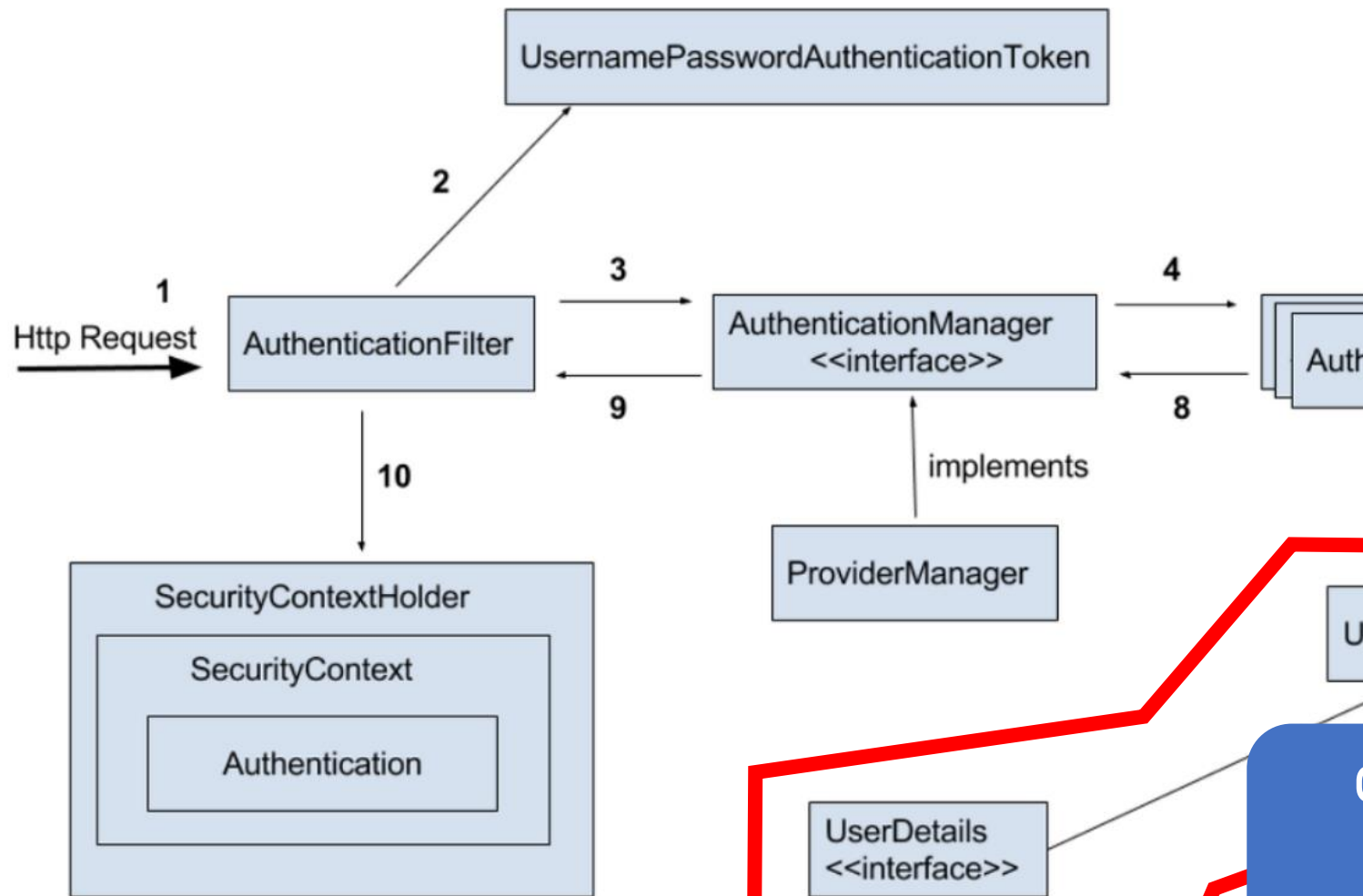
[로그인](#)

SECURITY 로그인 실패

[로그인 페이지로 이동](#)

아직은 로그인 파트가 구현이 안되어 있으므로
로그인이 당연히 안됩니다!



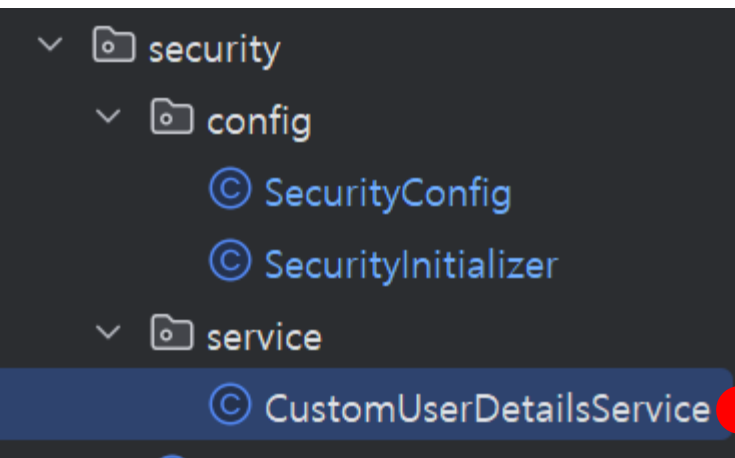


이제 로그인 처리를 담당하는 UserDetailService 를 구현할 차례 입니다!



UserDetailService

구현하기



회원 로그인 처리를 담당하게 될
CustomUserDetailsService
클래스를 만들어 봅시다



이 친구도 서비스 레이어 이므로
@Service 어노테이션 추가!
의존성 주입용 어노테이션 추가!

```
@Service  👤 kdtTetz  
@RequiredArgsConstructor  
public class CustomUserDetailsService implements UserDetailsService {  
  
    private final UserRepository userRepository;
```

회원 DB 와 통신을 해야 하므로
DB 처리 담당 UserRepository 주입 받기



@Override no usages kdtTetz *

```
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    User user = userRepository.findByUsername(username);

    if (user == null) {
        throw new UsernameNotFoundException(username);
    }

    List<SimpleGrantedAuthority> authorities = new ArrayList<>();
    String[] roles = user.getRoles().split(regex: ",");
    for (String role : roles) {
        authorities.add(new SimpleGrantedAuthority(role.trim()));
    }

    return new org.springframework.security.core.userdetails.User(
        user.getUsername(),
        user.getPassword(),
        authorities
    );
}
```

스프링 시큐리티의 로그인을 담당하는
loadUserByUsername 메서드를
오버라이드하여 구현합니다!



@Override no usages kdtTetz *

```
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
    User user = userRepository.findByUsername(username);  
  
    if (user == null) {  
        throw new UsernameNotFoundException(username);  
    }  
  
    List<SimpleGrantedAuthority> authorities = new ArrayList<>();  
    String[] roles = user.getRoles().split(regex: ",");  
    for (String role : roles) {  
        authorities.add(new SimpleGrantedAuthority(role.trim()));  
    }  
  
    return new org.springframework.security.core.userdetails.User(  
        user.getUsername(),  
        user.getPassword(),  
        authorities  
    );  
}
```

일단 Repository 를 사용하여
DB 에서 회원 정보를 찾아옵니다

회원 정보가 없으면
에러를 발생 시켜서 로그인
실패 처리



@Override no usages kdtTetz *

```
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
    User user = userRepository.findByUsername(username);  
  
    if (user == null) {  
        throw new UsernameNotFoundException(username);  
    }  
  
    List<SimpleGrantedAuthority> authorities = new ArrayList<>();  
    String[] roles = user.getRoles().split(regex: ",");  
    for (String role : roles) {  
        authorities.add(new SimpleGrantedAuthority(role.trim()));  
    }  
  
    return new org.springframework.security.core.userdetails.User(  
        user.getUsername(),  
        user.getPassword(),  
        authorities  
    );  
}
```

드디어 어제 만들었던 roles 컬럼을 씁니다!!

roles 에 등록 된 회원 권한을
배열로 만들어서 시큐리티에
전달 → 권한이 여러 개면
여러 권한을 동시에 전달 합니다!

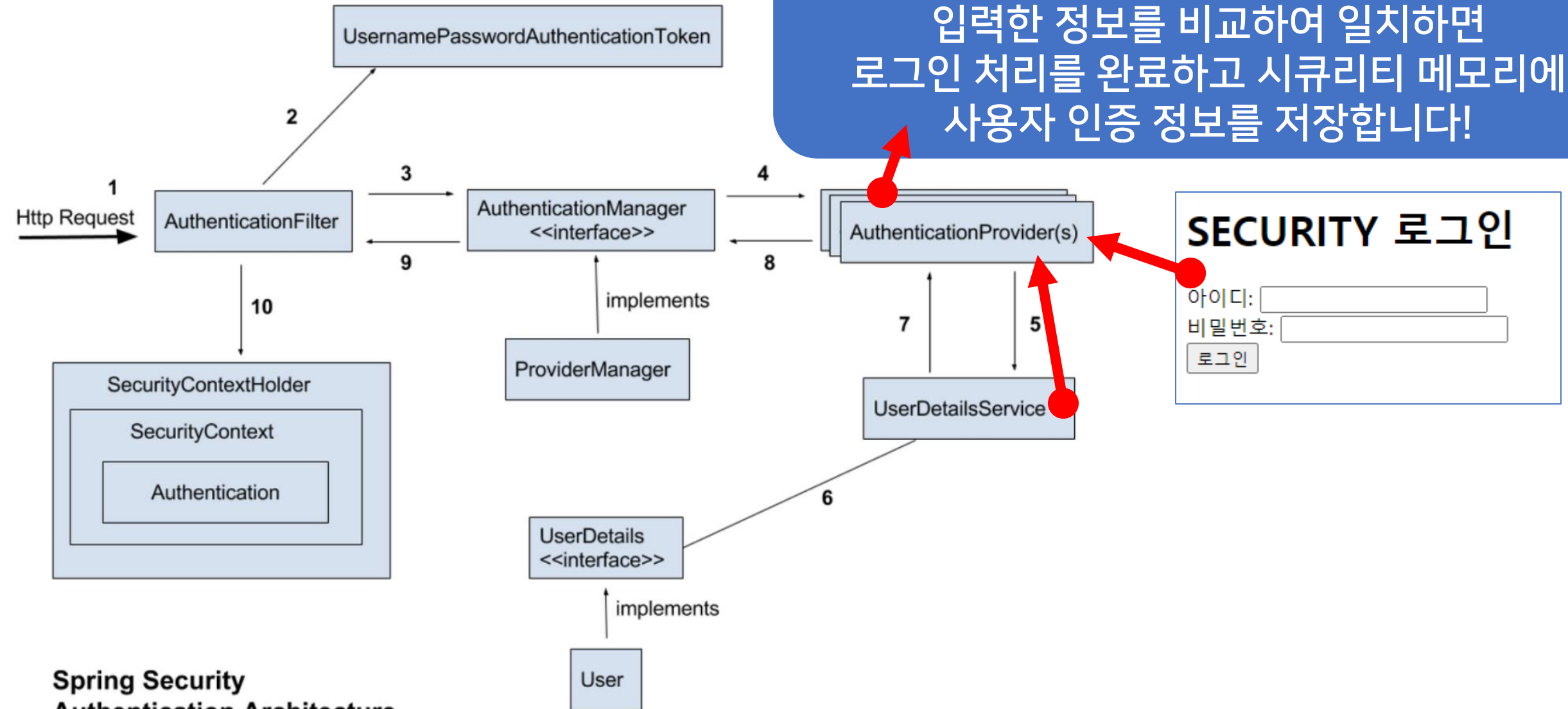


@Override no usages kdtTetz *

```
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
    User user = userRepository.findByUsername(username);  
  
    if (user == null) {  
        throw new UsernameNotFoundException(username);  
    }  
  
    List<SimpleGrantedAuthority> authorities = new ArrayList<>();  
    String[] roles = user.getRoles().split(regex: ",");  
    for (String role : roles) {  
        authorities.add(new SimpleGrantedAuthority(role.trim()));  
    }  
  
    return new org.springframework.security.core.userdetails.User(  
        user.getUsername(),  
        user.getPassword(),  
        authorities  
    );  
}
```

시큐리티에 DB 로 부터 받은
id, password, 권한 정보를
전달

이렇게 UserDetailsService 로 부터 전달받은 정보와 사용자가 Form 통해 입력한 정보를 비교하여 일치하면 로그인 처리를 완료하고 시큐리티 메모리에 사용자 인증 정보를 저장합니다!



Spring Security Authentication Architecture



UserDetailsService

등록 하기



```
@Configuration  👤 Tetz *  
@EnableWebSecurity  
@RequiredArgsConstructor  
@Log4j  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
    private final CustomUserDetailsService customUserDetailsService;  
    private final BCryptPasswordEncoder bCryptPasswordEncoder;
```

↓
방금 작성한
CustomUserDetailService 와
암호화를 위한 Bcrypt 주입 받기!

```
@Override new *  
protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
    auth.userDetailsService(customUserDetailsService).passwordEncoder(bCryptPasswordEncoder);  
}
```



매개 변수가 다른 configure 메서드를 오버라이딩 합니다!

방금 만든 서비스와 암호화 도구를 기본으로 사용자 인증에 사용하겠다고 설정



로그인

테스트!!

| | id | username | password | roles |
|---|------|----------|-------------------------|-------------|
| ▶ | 1 | 12 | 12 | ROLE_MEMBER |
| | 2 | tetz | 12 | ROLE_MEMBER |
| | 3 | siwan | 12 | ROLE_MEMBER |
| | 4 | 22 | 222 | ROLE_MEMBER |
| | 5 | 33 | 33 | ROLE_MEMBER |
| | 6 | 55 | \$2a\$10\$j9BpQ1EaNU... | ROLE_MEMBER |
| ● | NULL | NULL | NULL | NULL |

스프링 시큐리티는 기본적으로
암호화된 password 를 확인하기로 설정 했으므로
암호화되어서 저장 된 회원으로만 테스트가 가능!

저는 55 를 택하겠습니다!!



이제는 로그인이 되었기 때문에
우리가 이전에 설정한 로그인 성공 시의
리다이렉트 주소인
/security/member 로 리다이트 됩니다!

회원 기능

[회원가입](#) [로그인](#) [로그아웃](#)

시큐리티 회원 기능

[로그인](#)

404, 존재하지 않는 페이지 입니다

[홈 페이지로 돌아가기](#)

하지만 /security/member 에 대한
컨트롤러 처리는 아직 안되어 있으므로
404 페이지가 출력 된 상태!



회원 페이지

컨트롤러 작업!

스프링 시큐리티에 의해 로그인 된
사용자의 정보는 Principal 매개 변수를 통해
접근이 가능합니다!!

```
@GetMapping(🌐✓"/member") new *
public String loginSuccessPage(Model model, Principal principal) {
    if (principal == null) {
        return "redirect:/security/login";
    }
    UserDetails userDetails = customUserDetailsService.loadUserByUsername(principal.getName());
    model.addAttribute(attributeName: "user", userDetails);
    return context + "/member";
}
```




principal 이 null 이면 로그인이 안되었으므로
로그인 페이지로 리다이렉트!

```
@GetMapping(🌐✓"/member") new *
public String loginSuccessPage(Model model,
    if (principal == null) {
        return "redirect:/security/login";
    }
    UserDetails userDetails = customUserDetailsService.loadUserByUsername(principal.getName());
    model.addAttribute(attributeName: "user", userDetails);
    return context + "/member";
}
```



```
@GetMapping(🌐✓"/member") new *  
public String loginSuccessPage(Model model, Principal principal) {  
    if (principal == null) {  
        return "redirect:/security/login";  
    }  
    UserDetails userDetails = customUserDetailsService.loadUserByUsername(principal.getName());  
    model.addAttribute(attributeName: "user", userDetails);  
    return context + "/member";  
}
```

principal 에서 이름을 가져온 다른
시큐리티의 메모리에 저장 된 회원 정보를
UserDetailsService 에서 꺼내오기

→ 꺼내온 사용자 정보를 model 에 담아서
member.jsp 로 전송



회원 페이지

만들기!

<https://github.com/xenosign/spring-code-repo/blob/main/jsp/security/member.jsp>



```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
<!DOCTYPE html>
<html>
<head>
    <title>회원 페이지</title>
</head>
<body>
<%@include file="../header2.jsp"%>

<sec:authorize access="isAuthenticated()">
    <h1>SECURITY 로그인 성공</h1>
    <h2>사용자 정보</h2>
    <p>사용자명: <sec:authentication property="name"/></p>
    <p>권한:
        <sec:authentication property="authorities" var="authorities" />
        <c:forEach items="${authorities}" var="authority" varStatus="vs">
            ${authority}<c:if test="${!vs.last}">, </c:if>
        </c:forEach>
    </p>
</sec:authorize>
```

스프링 시큐리티에 대한 접근은
sec 라는 키워드로 시작하겠다는 선언

로그인 여부에 따라 화면을 다르게
출력하는 코드 + 스프링 시큐리티로 부터
받은 값을 출력



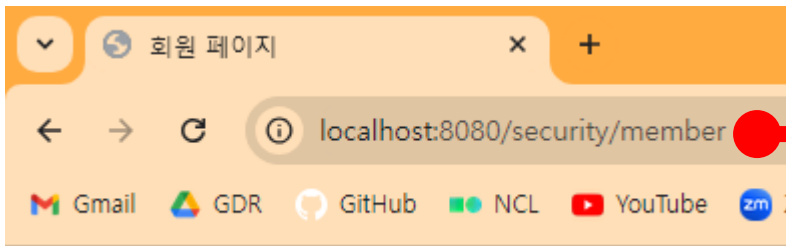
```
<sec:authorize access="!isAuthenticated()">
  <h1>SECURITY 로그인 실패</h1>
  <p>로그인 정보를 불러올 수 없습니다. 다시 로그인해주세요.</p>
</sec:authorize>
</body>
</html>
```

로그인에 실패한 케이스에 대한 출력!



회원 페이지

테스트



이제는 /security/member 요청에 대한
페이지 설정이 되었으므로 페이지가 뜹니다!



회원 기능

[회원가입](#) [로그인](#) [로그아웃](#)

시큐리티 회원 기능

[로그인](#)

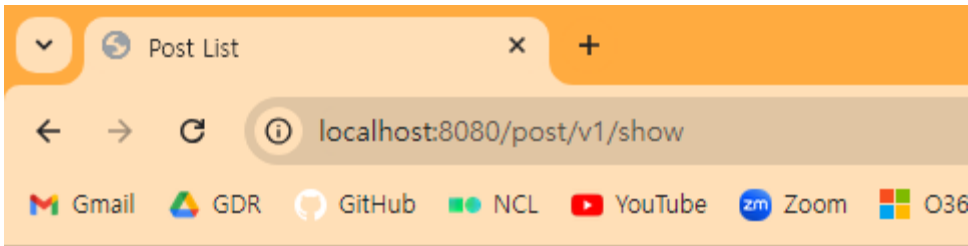
SECURITY 로그인 성공

스프링 시큐리티와 model 로 부터 받은 정보를
정상적으로 출력 하는 것 확인 가능!

사용자 정보

사용자명: 55

권한: ROLE_MEMBER



로그인이 되었으므로 다른 페이지에도
접근 가능!

회원 기능

[회원가입](#) [로그인](#) [로그아웃](#)

시큐리티 회원 기능

[로그인](#)

글 목록

제목 검색 / 내용 검색

| ID | Title | Content | Actions |
|----|----------|------------------------|---|
| 1 | 첫 번째 게시물 | 이것은 첫 번째 게시물의 내용입니다. a | <input type="button" value="수정"/> <input type="button" value="삭제"/> |
| 2 | 두 번째 게시물 | 이것은 두 번째 게시물의 내용입니다. b | <input type="button" value="수정"/> <input type="button" value="삭제"/> |
| 3 | 세 번째 게시물 | 이것은 세 번째 게시물의 내용입니다. c | <input type="button" value="수정"/> <input type="button" value="삭제"/> |
| 4 | 네 번째 게시물 | 이것은 네 번째 게시물의 내용입니다. d | <input type="button" value="수정"/> <input type="button" value="삭제"/> |




시큐리티 권한에 따른




차이 확인하기!

ROLE_ADMIN 권한만 접근이 가능한
요청 주소 만들기!

@Override Tetz *

```
protected void configure(HttpSecurity http) throws Exception {  
    http.authorizeRequests()  
        .antMatchers( "/").permitAll()  
        .antMatchers( "/user/**").permitAll()  
        .antMatchers( "/security/admin").access( attribute: "hasRole('ROLE_ADMIN')")  
        .antMatchers( "/security/**").permitAll()  
        .antMatchers( "/**").access( attribute: "hasRole('ROLE_MEMBER')");  
}
```





```
@GetMapping(🌐"/admin")  👤 kdtTetz  
public String admin() {  
    return context + "/admin";  
}
```

해당 요청에 admin.jsp 를 보여주는
컨트롤러 만들기!

<https://github.com/xenosign/spring-code-repo/blob/main/jsp/security/admin.jsp>



```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
<%@include file="../header2.jsp"%>
<h1>SECURITY ADMIN 페이지 입니다</h1>
<a href="/security/login">로그인 페이지로 이동</a>
</body>
</html>
```

정말 간단한 admin 페이지!



```
<h3>시큐리티 회원 기능</h3>
```

```
<a href="/security/login">로그인</a>
```

```
<a href="/security/admin">admin</a>
```



admin 테스트를 위한
Header도 업데이트!



특정 아이디어에

권한 부여!



| | id | username | password | roles |
|---|------|----------|-------------------------|-------------|
| ▶ | 1 | 12 | 12 | ROLE_MEMBER |
| | 2 | tetz | 12 | ROLE_MEMBER |
| | 3 | siwan | 12 | ROLE_MEMBER |
| | 4 | 22 | 222 | ROLE_MEMBER |
| | 5 | 33 | 33 | ROLE_MEMBER |
| | 6 | 55 | \$2a\$10\$j9BpQ1EaNU... | ROLE_MEMBER |
| | 7 | 66 | \$2a\$10\$QPpt0Ms23r... | ROLE_MEMBER |
| ✱ | NULL | NULL | NULL | NULL |

방금 새롭게 회원 가입을 해서
66 계정을 만들었습니다!

그리고 55 계정에는 ROLE_ADMIN 권한을
추가해서 시큐리티 권한에 따른 차이를
확인해 보겠습니다!



| | | | |
|---|----|-------------------------|------------------------|
| 6 | 55 | \$2a\$10\$j9BpQ1EaNU... | ROLE_MEMBER,ROLE_ADMIN |
| 7 | 66 | \$2a\$10\$QPpt0Ms23r... | ROLE_MEMBER |

55 계정에는 ROLE_ADMIN 권한을 추가!

시큐리티 권한에 따른



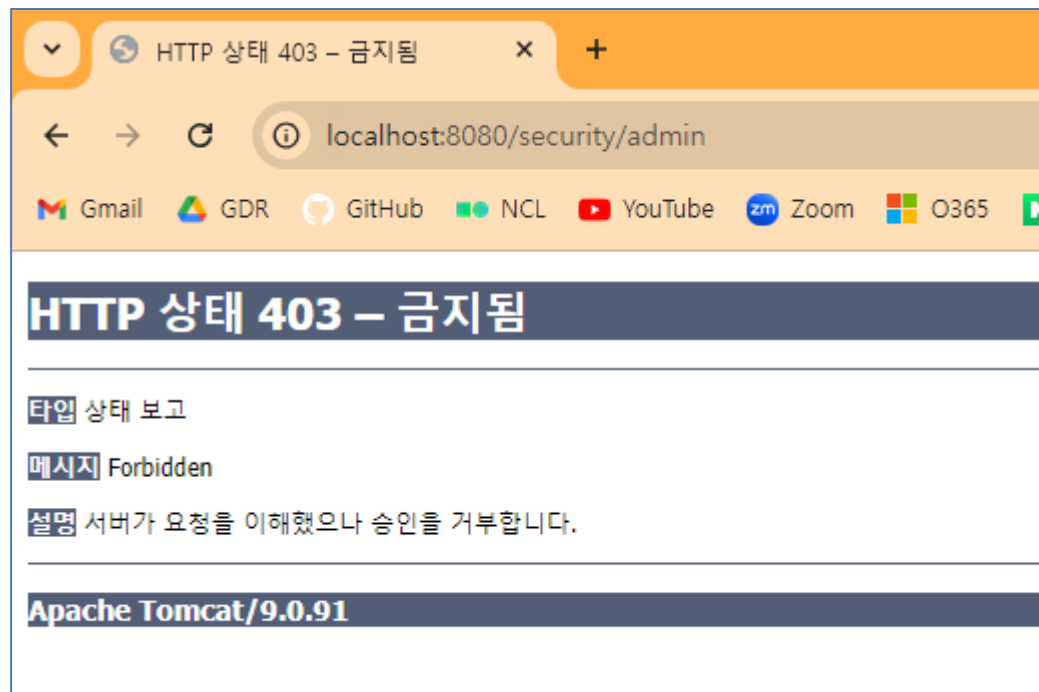
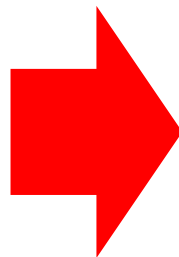
차이 테스트!

SECURITY 로그인 성공

사용자 정보

사용자명: 66

권한: ROLE_MEMBER

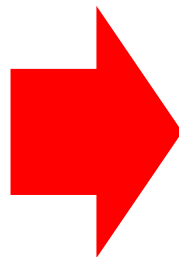


SECURITY 로그인 성공

사용자 정보

사용자명: 55

권한: ROLE_ADMIN, ROLE_MEMBER



SECURITY ADMIN 페이지 입니다

[로그인 페이지로 이동](#)





로그 아웃

구현하기!



```
http.logout()  
    .logoutUrl("/security/logout")  
    .invalidateHttpSession(true)  
    .deleteCookies(...cookieNamesToClear: "remember-me", "JSESSIONID")  
    .logoutSuccessUrl("/security/login")  
    .permitAll();
```

로그 아웃 요청을 받을 주소와
각종 설정을 설정하기!

Cookie 는 해당 이름으로 저장이 되므로
저걸로 지정 하시면 됩니다!

로그아웃이 성공하면 리다이렉트 될 주소와
로그 아웃에 접근 가능한 권한 까지 설정!



그란데 말입니다

로그 아웃할 때도 CSRF 에서 안전할까요?

당연하게도 안전하지 않습니다!
따라서 로그아웃 요청을 보낼 때에도
CSRF 방지를 위한 코드를 통해 보내 주어야 합니다!

그란데 말입니다



헤더에 로그아웃

버튼 추가하기

<https://github.com/xenosign/spring-code-repo/blob/main/jsp/security/header.jsp>



다른 기능을 위한 링크 추가하기

```
<h3>시큐리티 회원 기능</h3>
<a href="/security/admin">admin</a>
<a href="/security/member">member</a>
<a href="/security/login">로그인</a>
<a href="#" onclick="document.getElementById('logout-form').submit();">로그아웃</a>
<form id="logout-form" action="/security/logout" method="post" style="display: none;">
  <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
</form>
```

이제 로그아웃 요청은 간단한 GET 방식 주소 요청이 아니라
input 요소에 CSRF 방지 정보를 담아서 해당 form 을
전달하는 방식으로 변경 됩니다!

로그 아웃 테스트





회원 기능

[회원가입](#) [로그인](#) [로그아웃](#)

시큐리티 회원 기능

[admin](#) [member](#) [로그인](#) [로그아웃](#)

SECURITY 로그인

아이디:

비밀번호:

로그 아웃을 누르면 바로 로그 아웃이 성공하고
성공 시 리다이렉트 주소인
/security/login 으로 리다이렉트 됩니다!





JWT



Login with Kakao



the face reader
lee jung jae, dipsy