



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

Lecturer 정수원
<https://github.com/onjsdnjs>

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



I. 강좌 소개

#01. 강의에서 다루는 내용, 개발환경, 선수지식

Lecturer 정수원
<https://github.com/onjsdnjs>

#01. 강의에서 다루는 내용

1. 스프링 시큐리티의 보안 설정 API 와 이와 연계된 각 Filter 들에 대해 학습한다

- 각 API 의 개념과 기본적인 사용법, API 처리 과정, API 동작방식 등 학습
- API 설정 시 생성 및 초기화 되어 사용자의 요청을 처리하는 Filter 학습

2. 스프링 시큐리티 내부 아키텍처와 각 객체의 역할 및 처리과정을 학습한다

- 초기화 과정, 인증 과정, 인가과정 등을 아키텍처적인 관점에서 학습

3. 실전 프로젝트

- 인증 기능 구현 – Form 방식, Ajax 인증 처리
- 인가 기능 구현 – DB 와 연동해서 권한 제어 시스템 구현

#02. 개발환경, 선수지식

- 개발환경

- JDK 1.8 이상
- DB - Postgres
- IDE – IntelliJ or STS

- 선수지식

- Spring Boot
- Spring MVC
- Spring Data JPA
- Thymeleaf
- Postgres
- Lombok

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



II. 스프링 시큐리티 기본 API & Filter 이해

#01. 인증 API – 프로젝트 구성 및 의존성 추가

II. 스프링 시큐리티 기본 API & Filter 이해

#01. 인증 API – 스프링 시큐리티 의존성 추가

pom.xml

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-security</artifactId>
```

```
</dependency>
```

II. 스프링 시큐리티 기본 API & Filter 이해

#01. 인증 API – 스프링 시큐리티 의존성 추가

- 스프링 시큐리티의 의존성 추가 시 일어나는 일들
 - 서버가 기동되면 스프링 시큐리티의 초기화 작업 및 보안 설정이 이루어진다
 - 별도의 설정이나 구현을 하지 않아도 기본적인 웹 보안 기능이 현재 시스템에 연동되어 작동함
 1. 모든 요청은 인증이 되어야 자원에 접근이 가능하다
 2. 인증 방식은 폼 로그인 방식과 httpBasic 로그인 방식을 제공한다
 3. 기본 로그인 페이지 제공한다
 4. 기본 계정 한 개 제공한다 – username : user / password : 랜덤 문자열
- 문제점
 - 계정 추가, 권한 추가, DB 연동 등
 - 기본적인 보안 기능 외에 시스템에서 필요로 하는 더 세부적이고 추가적인 보안기능이 필요

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



II. 스프링 시큐리티 기본 API & Filter 이해

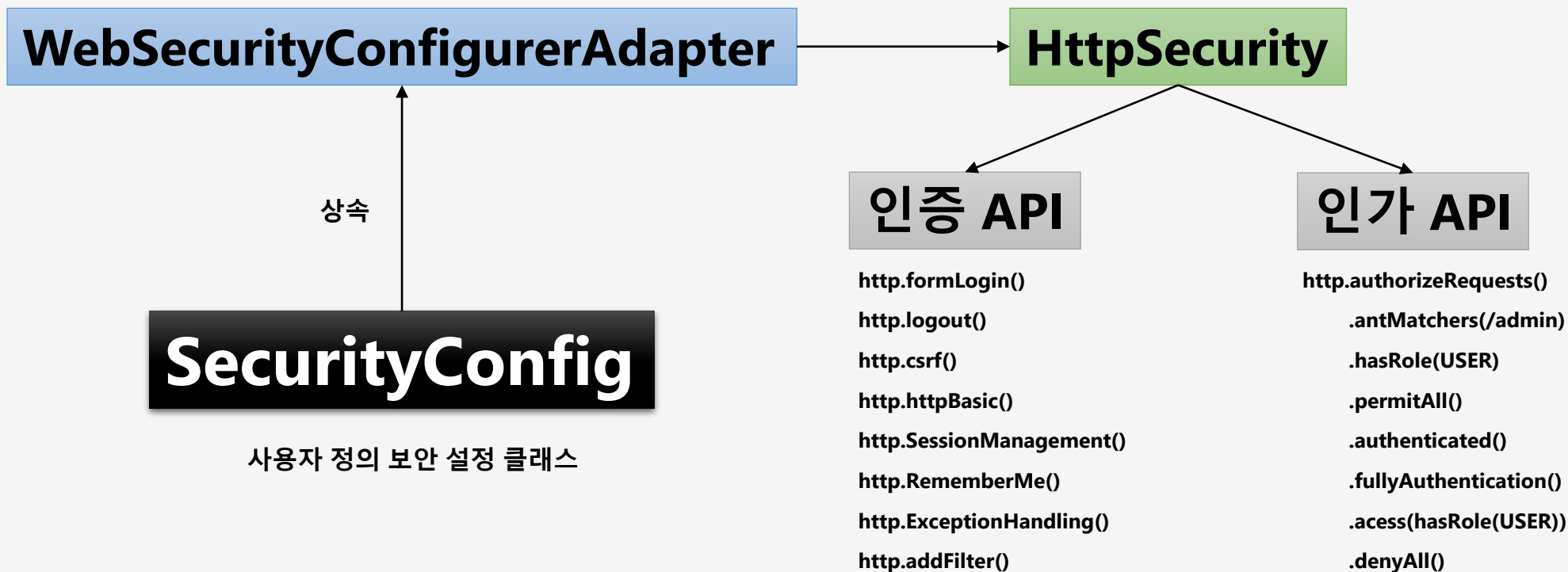
#02. 인증 API – 사용자 정의 보안 기능 구현

II. 스프링 시큐리티 기본 API & Filter 이해

#01. 인증 API – 사용자 정의 보안 기능 구현

스프링 시큐리티의 웹 보안 기능 초기화 및 설정

세부적인 보안 기능을 설정할 수 있는 API 제공



II. 스프링 시큐리티 기본 API & Filter 이해

#01. 인증 API – SecurityConfig 설정

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.formLogin();

}

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



II. 스프링 시큐리티 기본 API & Filter 이해

#03. 인증 API – HTTP Basic 인증, BasicAuthenticationFilter

Lecturer 정수원
<https://github.com/onjsdnjs>

II. 스프링 시큐리티 기본 API & Filter 이해

#04. 인증 API – HTTP Basic 인증



- HTTP는 자체적인 인증 관련 기능을 제공하며 HTTP 표준에 정의된 가장 단순한 인증 기법이다
- 간단한 설정과 Stateless가 장점 - Session Cookie(JSESSIONID) 사용하지 않음
- 보호자원 접근시 서버가 클라이언트에게 401 Unauthorized 응답과 함께 WWW-Authenticate header를 기술해서 인증요구를 보냄
- Client는 ID:Password 값을 Base64로 Encoding한 문자열을 Authorization Header에 추가한 뒤 Server에게 Resource를 요청
 - Authorization: Basic cmVzdDpyZXN0
- ID, Password가 Base64로 Encoding되어 있어 ID, Password가 외부에 쉽게 노출되는 구조이기 때문에 SSL이나 TLS는 필수이다

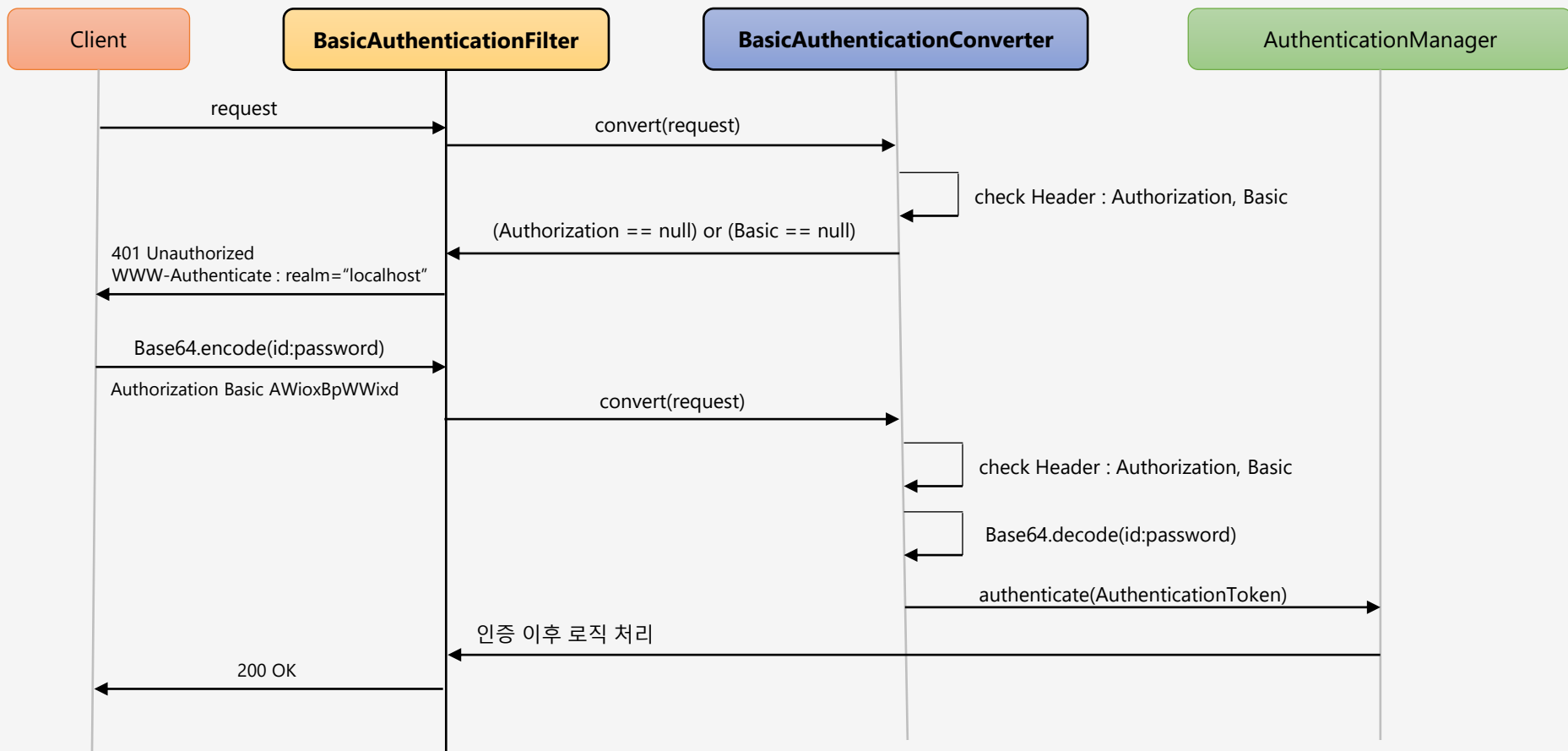
II. 스프링 시큐리티 기본 API & Filter 이해

#03. 인증 API – HTTP Basic 인증

```
protected void configure(HttpSecurity http) throws Exception {  
    http.httpBasic();  
}
```

II. 스프링 시큐리티 기본 API & Filter 이해

#03. 인증 API – BasicAuthenticationFilter



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



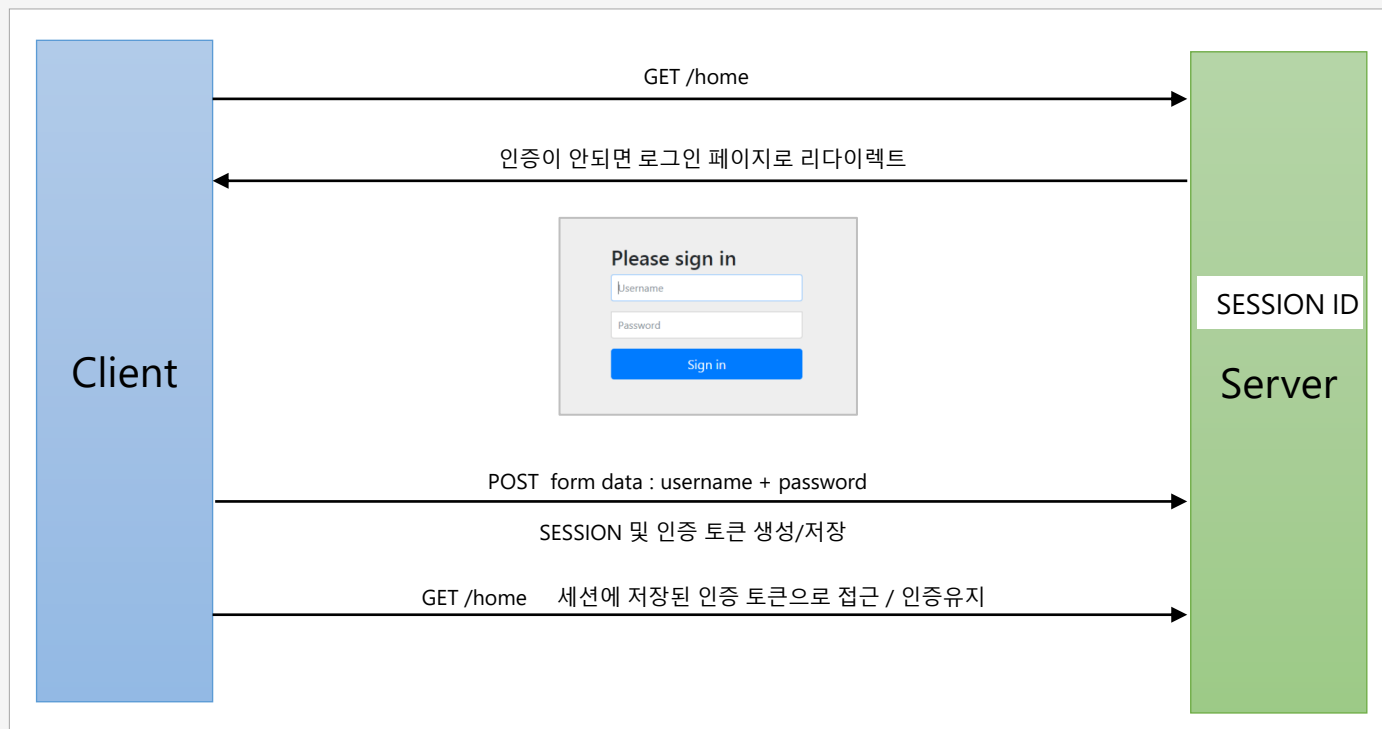
II. 스프링 시큐리티 기본 API & Filter 이해

#04. 인증 API – Form 인증

II. 스프링 시큐리티 기본 API & Filter 이해

#05. 인증 API – Form 인증

Login



II. 스프링 시큐리티 기본 API & Filter 이해

#04. 인증 API – Form Login 인증

http.formLogin() : Form 로그인 인증 기능이 작동함

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

```
    http.formLogin()
```

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

```
        .defaultSuccessUrl("/home")
```

```
        .failureUrl( " /login.html?error=true")
```

```
        .usernameParameter("username")
```

```
        .passwordParameter("password")
```

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

```
        .successHandler(loginSuccessHandler())
```

```
        .failureHandler(loginFailureHandler())
```

```
}
```

```
// 사용자 정의 로그인 페이지
```

```
// 로그인 성공 후 이동 페이지
```

```
// 로그인 실패 후 이동 페이지
```

```
// 아이디 파라미터명 설정
```

```
// 패스워드 파라미터명 설정
```

```
// 로그인 Form Action Url
```

```
// 로그인 성공 후 핸들러
```

```
// 로그인 실패 후 핸들러
```

Core Spring Security



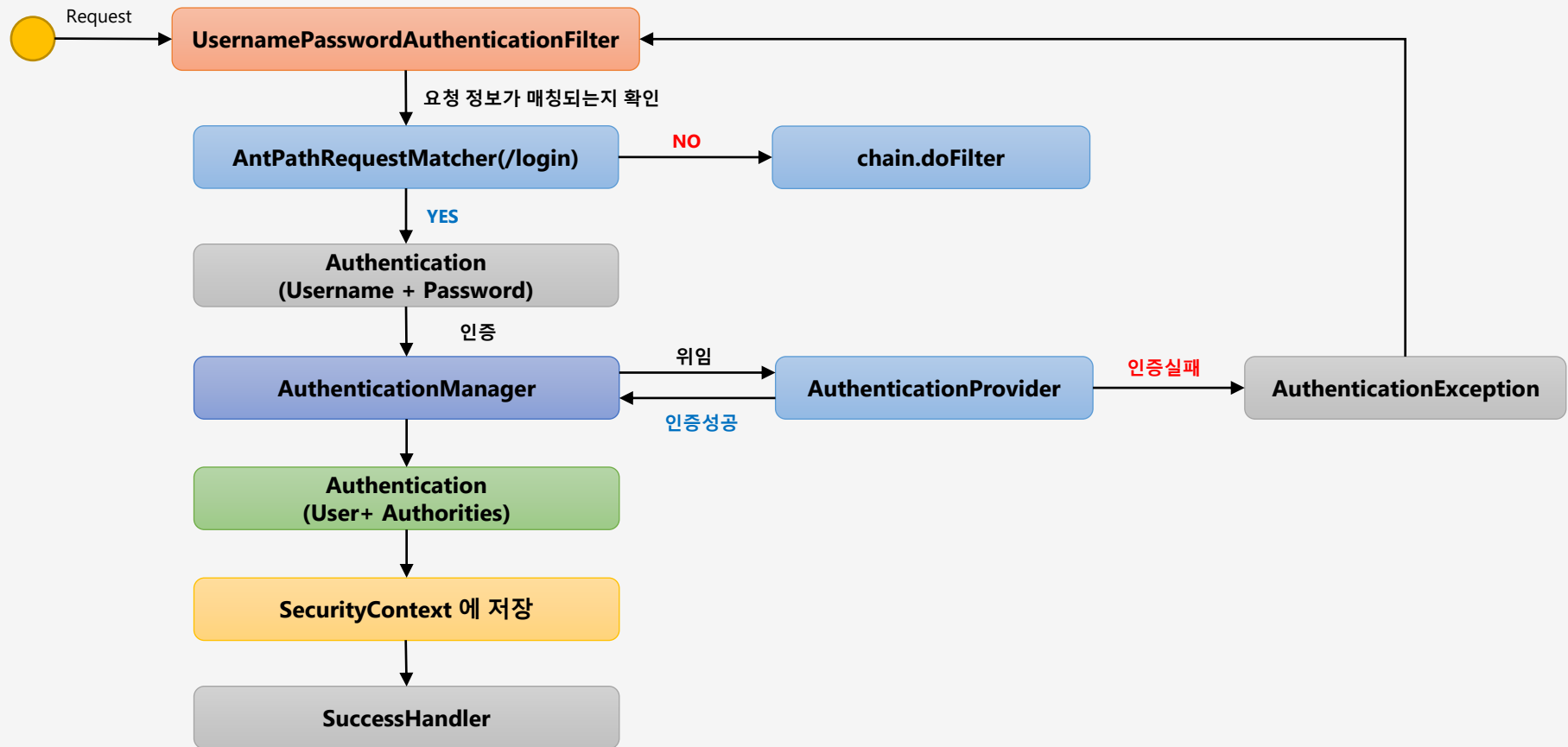
핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

II. 스프링 시큐리티 기본 API & Filter 이해

#05. 인증 API – UsernamePasswordAuthenticationFilter

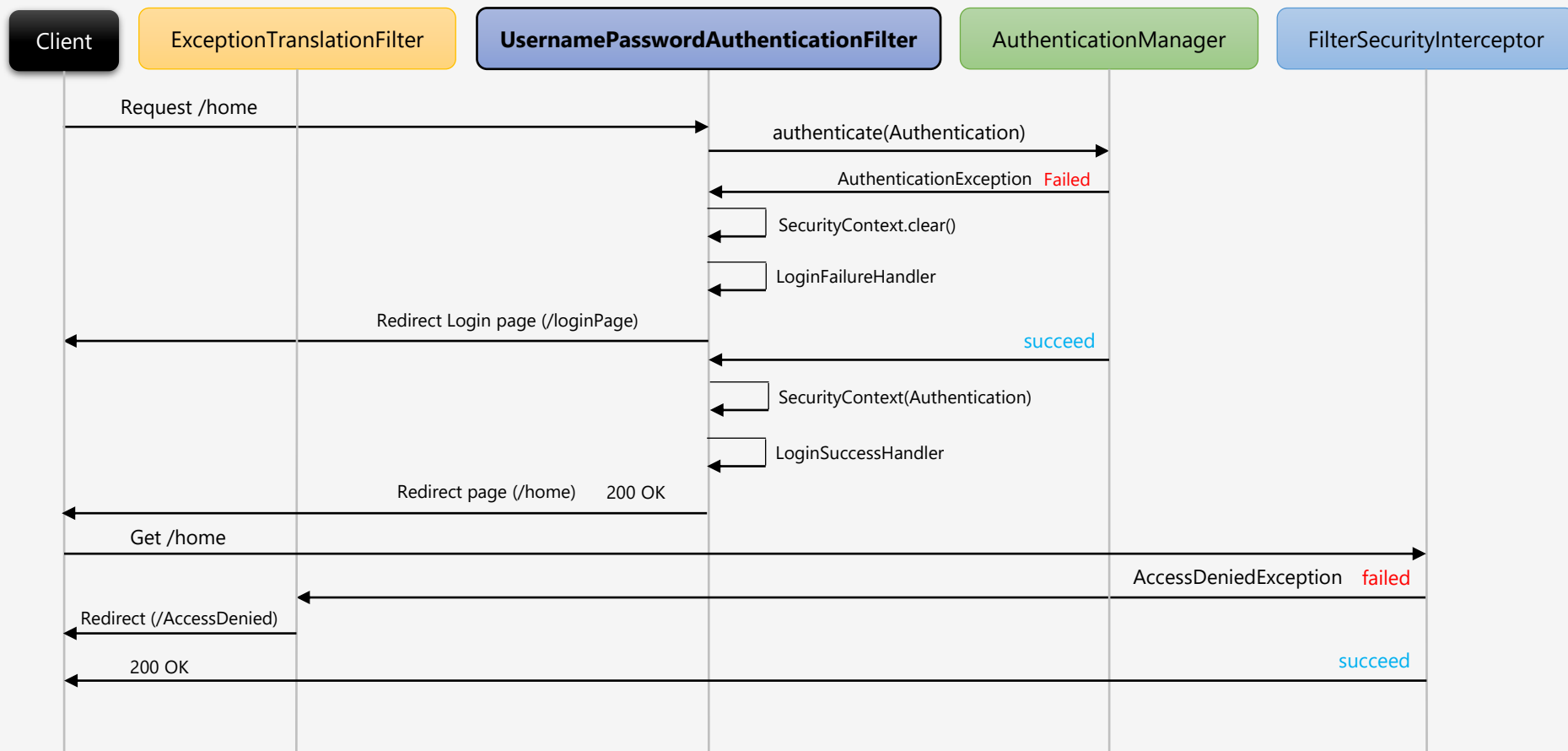
II. 스프링 시큐리티 기본 API & Filter 이해

#05. 인증 API – Login Form 인증



II. 스프링 시큐리티 기본 API & Filter 이해

05. 인증 API – UsernamePasswordAuthenticationFilter



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



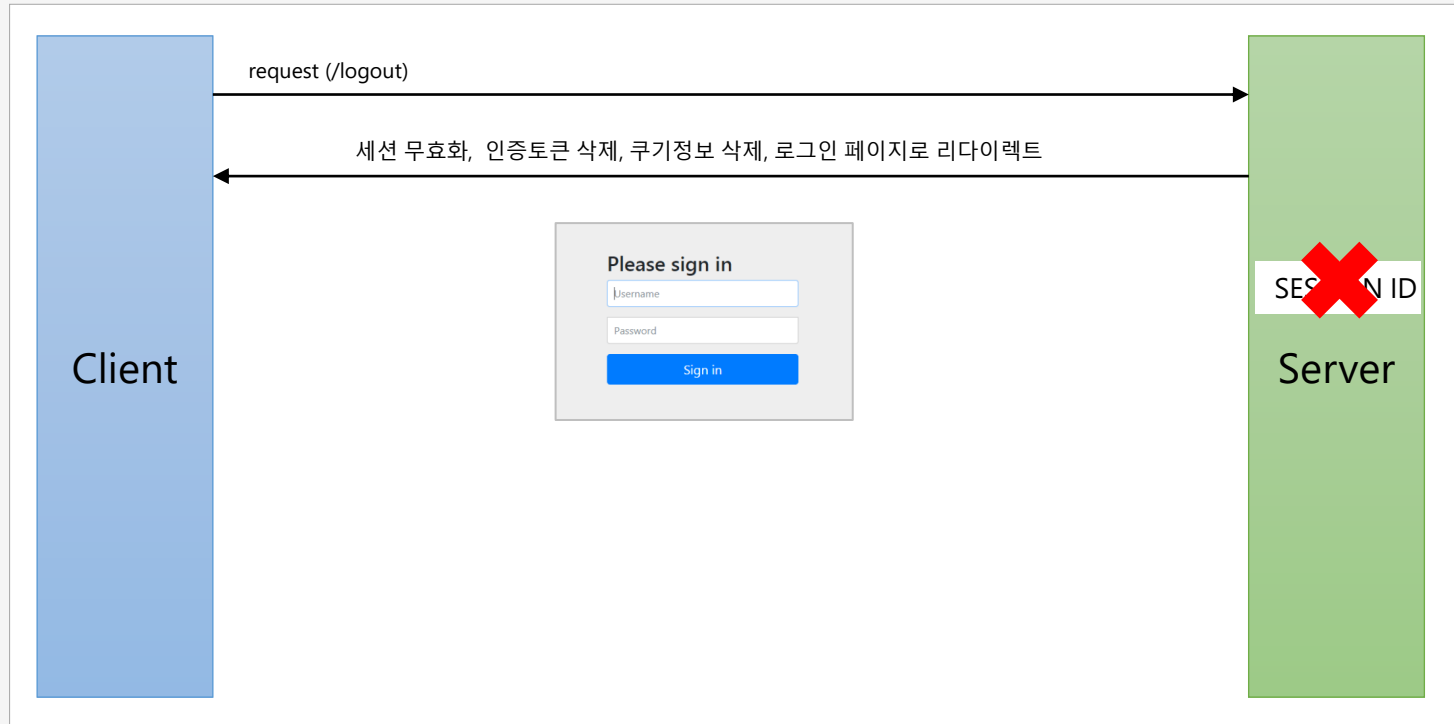
II. 스프링 시큐리티 기본 API & Filter 이해

#06. 인증 API – Logout, LogoutFilter

II. 스프링 시큐리티 기본 API & Filter 이해

#06. 인증 API – Logout

Logout



II. 스프링 시큐리티 기본 API & Filter 이해

#06. 인증 API – Logout

http.logout() : 로그아웃 기능이 작동함

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

```
    http.logout()
```

```
        // 로그아웃 처리
```

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

```
        // 로그아웃 처리 URL
```

```
        .logoutSuccessUrl( " /login " )
```

```
        // 로그아웃 성공 후 이동페이지
```

```
        .deleteCookies( " JSESSIONID", " remember-me " )
```

```
        // 로그아웃 후 쿠키 삭제
```

```
        .addLogoutHandler(logoutHandler())
```

```
        // 로그아웃 핸들러
```

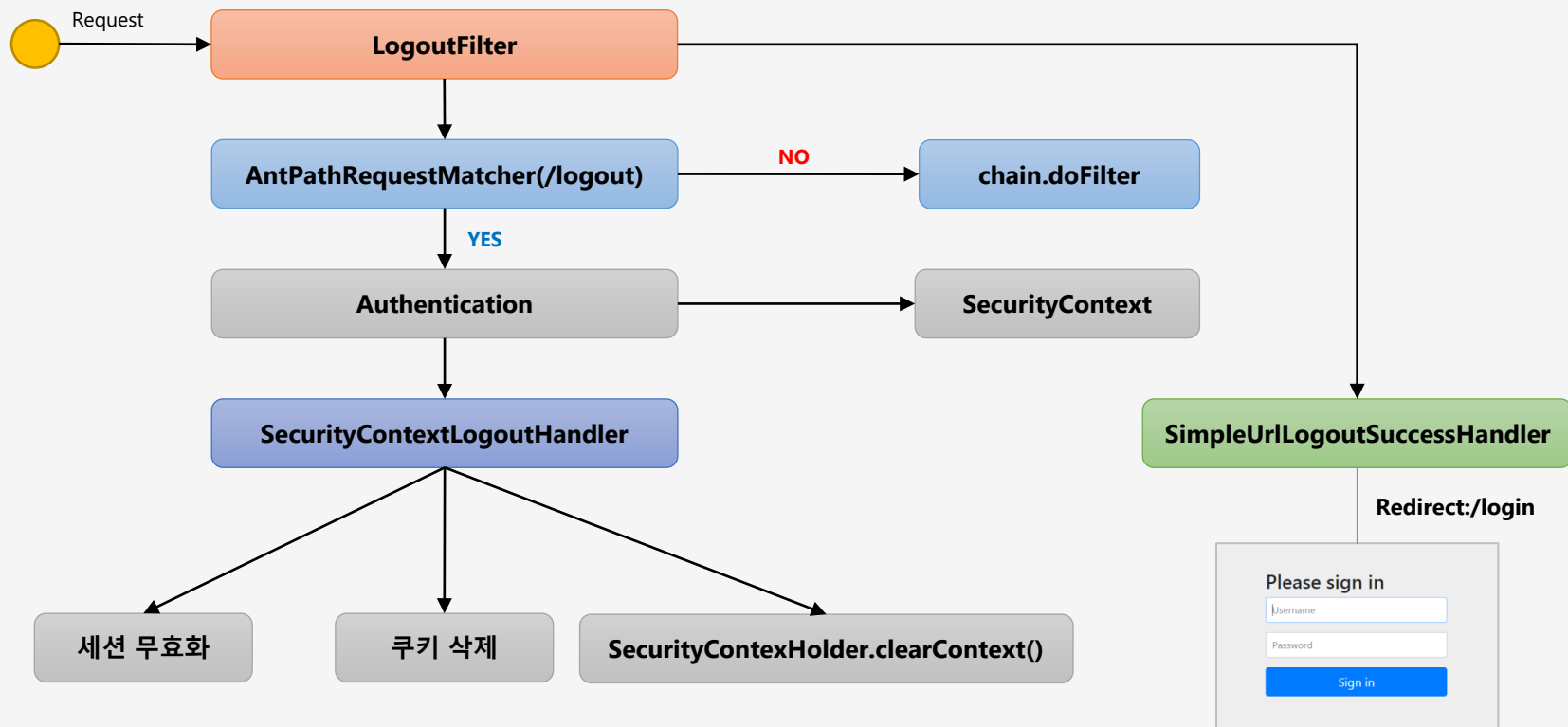
```
        .logoutSuccessHandler(logoutSuccessHandler())
```

```
        // 로그아웃 성공 후 핸들러
```

```
}
```

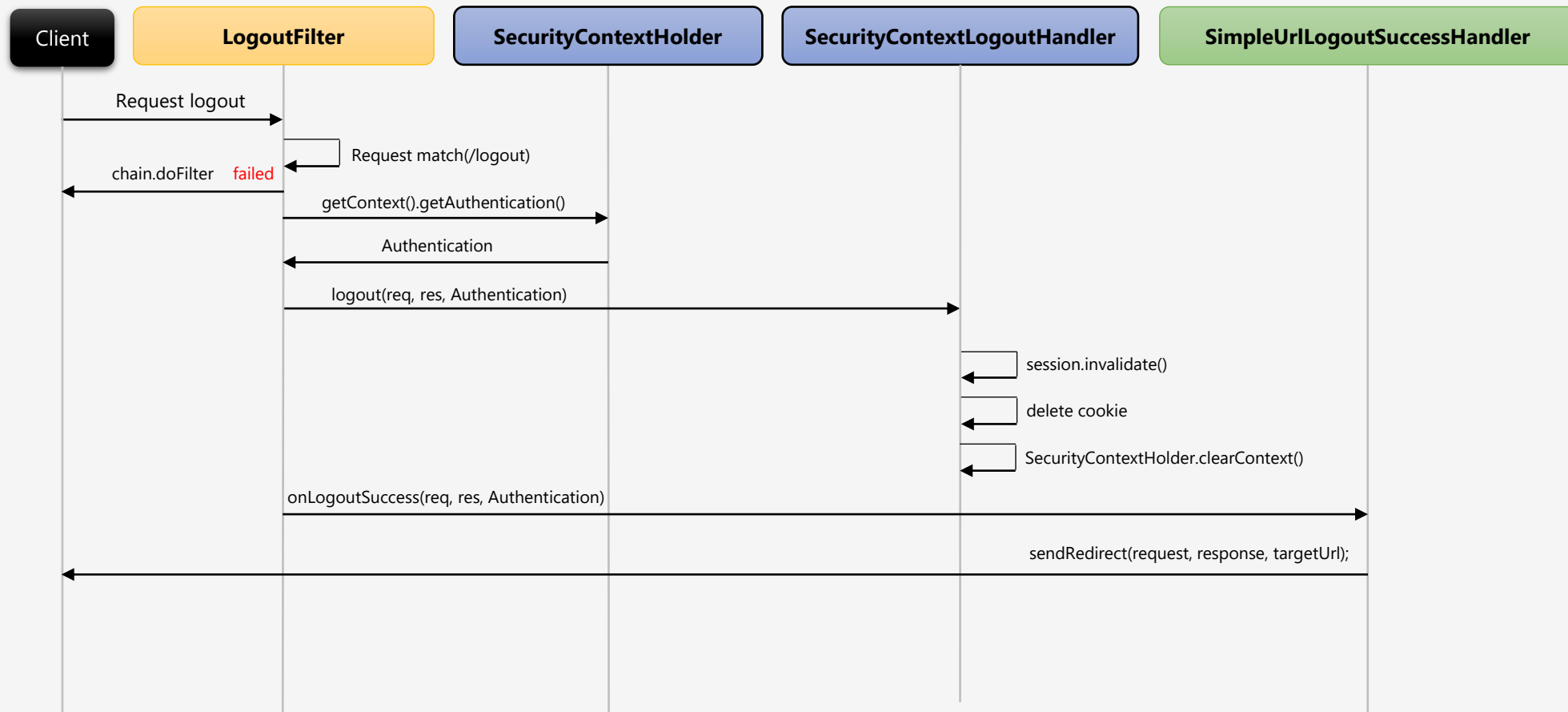
II. 스프링 시큐리티 기본 API & Filter 이해

#06. 인증 API – Logout



II. 스프링 시큐리티 기본 API & Filter 이해

06. 인증 API – LogoutFilter



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

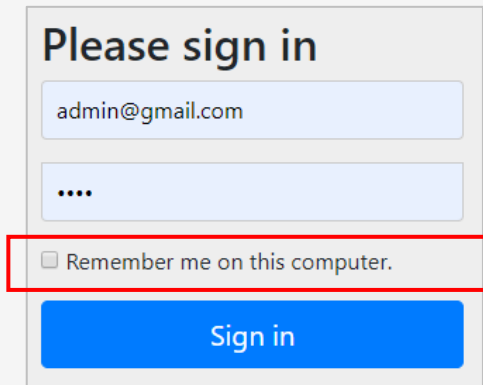


II. 스프링 시큐리티 기본 API & Filter 이해

#07. 인증 API – Remember Me 인증

II. 스프링 시큐리티 기본 API & Filter 이해

#06. 인증 API – Remember Me 인증



Please sign in

admin@gmail.com

....

☐ Remember me on this computer.

Sign in

1. 세션이 만료되고 웹 브라우저가 종료된 후에도 어플리케이션이 사용자를 기억하는 기능
2. **Remember-Me** 쿠키에 대한 **Http** 요청 을 확인한 후 토큰 기반 인증을 사용해 유효성을 검사하고 토큰이 검증되면 사용자는 로그인 된다
3. 사용자 라이프 사이클
 - 인증 성공(**Remember-Me**쿠키 설정)
 - 인증 실패(쿠키가 존재하면 쿠키 무효화)
 - 로그아웃(쿠키가 존재하면 쿠키 무효화)

II. 스프링 시큐리티 기본 API & Filter 이해

#07. 인증 API – Remember Me 인증

http.rememberMe() : rememberMe 기능이 작동함

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

```
    http.rememberMe()
```

```
        .rememberMeParameter("remember") // 기본 파라미터명은 remember-me
```

```
        .tokenValiditySeconds(3600) // Default 는 14일
```

```
        .alwaysRemember(true) // 리멤버 미 기능이 활성화되지 않아도 항상 실행
```

```
        .userDetailsService(userDetailsService)
```

```
}
```

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

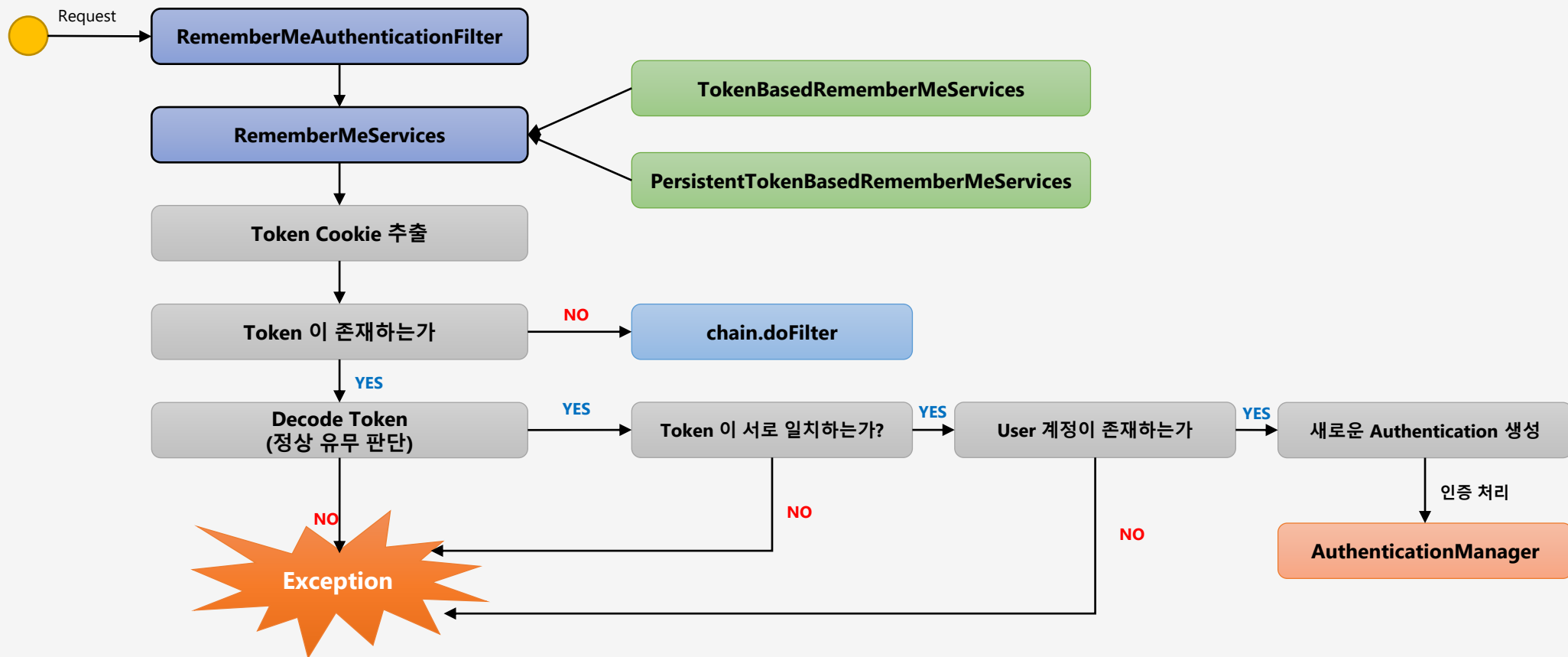


II. 스프링 시큐리티 기본 API & Filter 이해

#08. 인증 API – RememberMeAuthenticationFilter

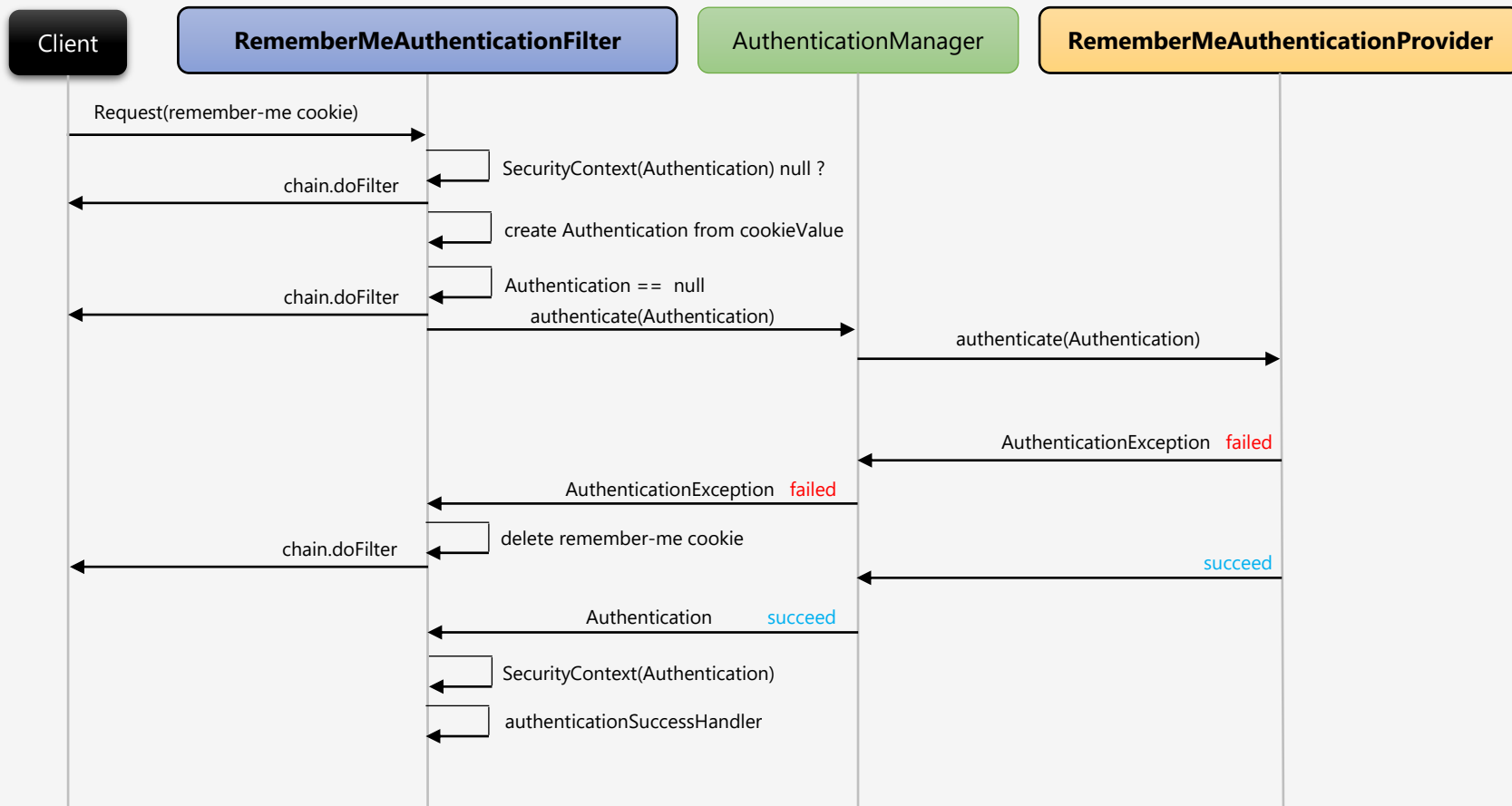
II. 스프링 시큐리티 기본 API & Filter 이해

#08. 인증 API – Remember Me 인증



II. 스프링 시큐리티 기본 API & Filter 이해

#08. 인증 API – RememberMeAuthenticationFilter



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

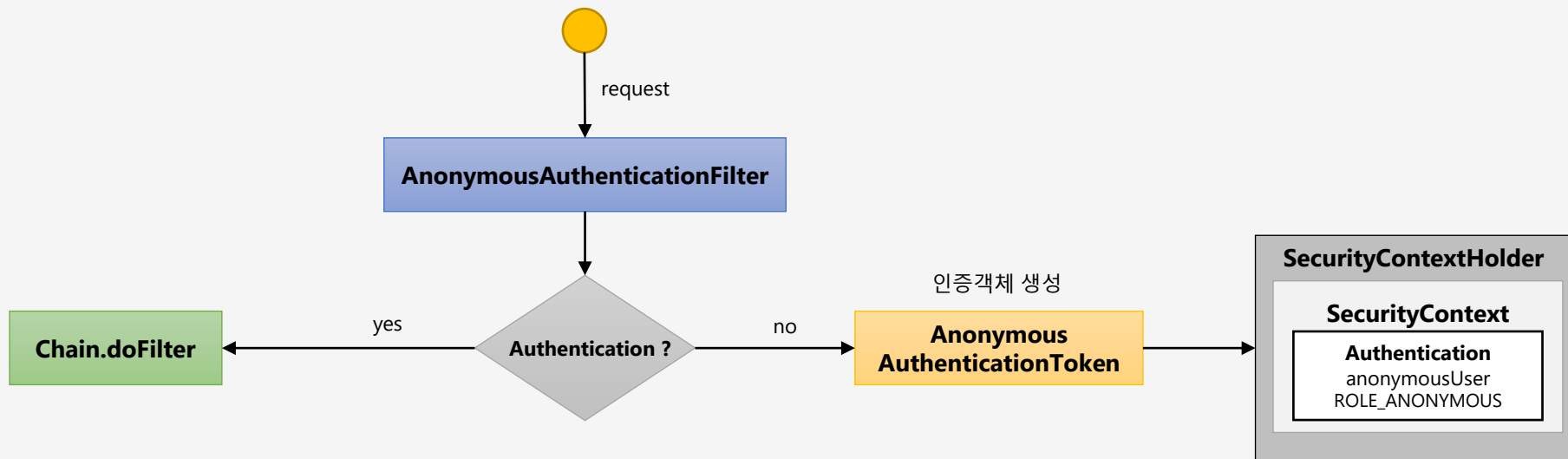


II. 스프링 시큐리티 기본 API & Filter 이해

#09. 인증 API – AnonymousAuthenticationFilter

II. 스프링 시큐리티 기본 API & Filter 이해

#09. 인증 API – AnonymousAuthenticationFilter



- 익명사용자 인증 처리 필터
- 익명사용자와 인증 사용자를 구분해서 처리하기 위한 용도로 사용
- 화면에서 인증 여부를 구현할 때 `isAnonymous()` 와 `isAuthenticated()` 로 구분해서 사용
- 인증객체를 세션에 저장하지 않는다

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

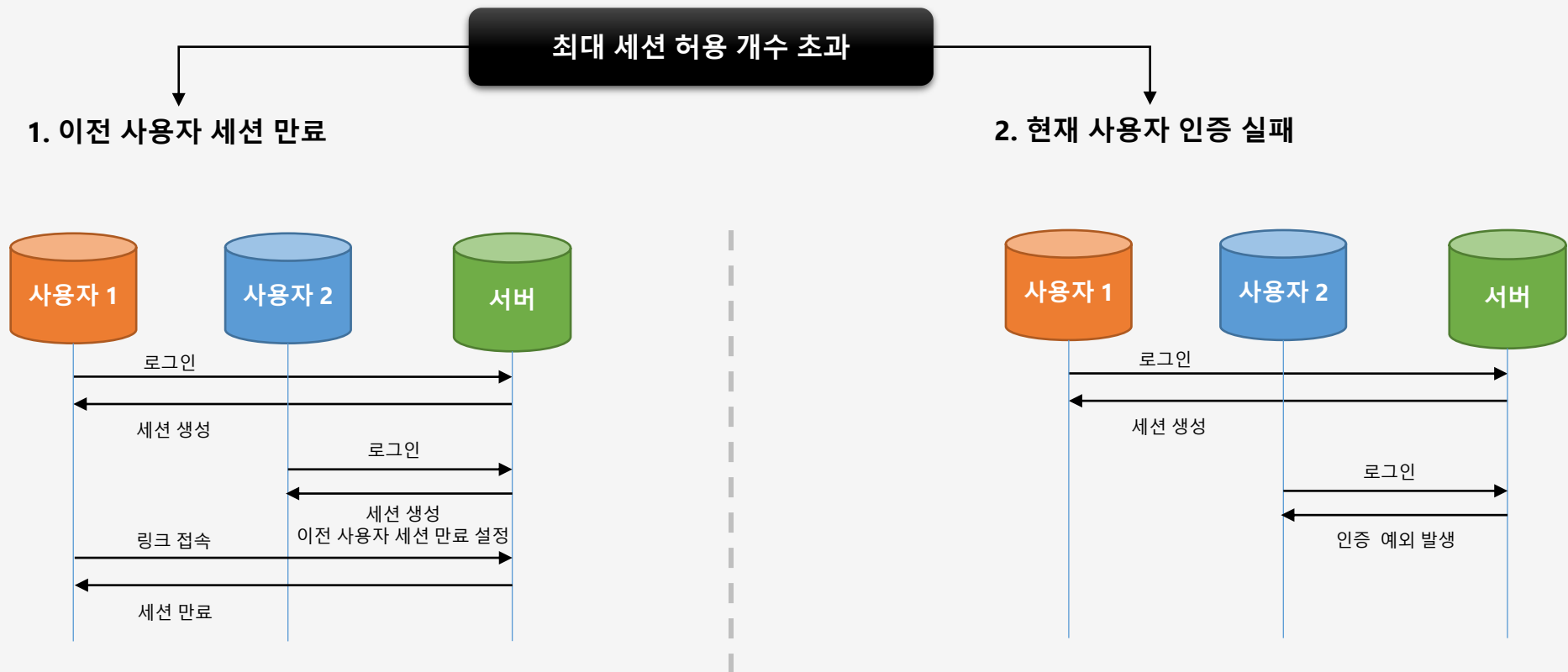


II. 스프링 시큐리티 기본 API & Filter 이해

#10. 인증 API – 동시 세션 제어 / 세션고정보호 / 세션 정책

II. 스프링 시큐리티 기본 API & Filter 이해

#10. 인증 API – 동시 세션 제어



II. 스프링 시큐리티 기본 API & Filter 이해

10. 인증 API – 동시 세션 제어

http.sessionManagement() : 세션 관리 기능이 작동함

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

```
    http.sessionManagement()
```

```
        .maximumSessions(1)           // 최대 허용 가능 세션 수 , -1 : 무제한 로그인 세션 허용
```

```
        .maxSessionsPreventsLogin(true) // 동시 로그인 차단함, false : 기존 세션 만료(default)
```

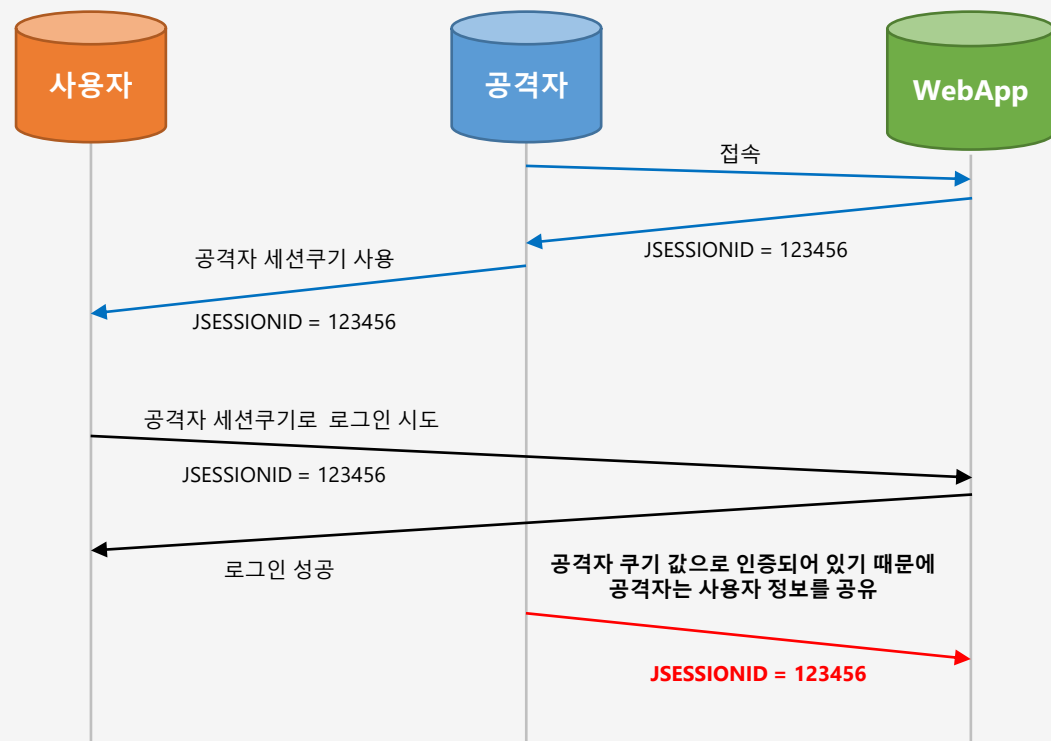
```
        .invalidSessionUrl("/invalid") // 세션이 유효하지 않을 때 이동 할 페이지
```

```
        .expiredUrl("/expired ")      // 세션이 만료된 경우 이동 할 페이지
```

```
}
```

II. 스프링 시큐리티 기본 API & Filter 이해

10. 인증 API – 세션 고정 보호



II. 스프링 시큐리티 기본 API & Filter 이해

10. 인증 API – 세션 고정 보호

http.sessionManagement() : 세션 관리 기능이 작동함

```
protected void configure(HttpSecurity http) throws Exception {  
    http.sessionManagement()  
        .sessionFixation().changeSessionId() // 기본값  
                                           // none, migrateSession, newSession  
}
```

II. 스프링 시큐리티 기본 API & Filter 이해

10. 인증 API – 세션 정책

http.sessionManagement() : 세션 관리 기능이 작동함

```
protected void configure(HttpSecurity http) throws Exception {  
    http.sessionManagement()  
        .sessionCreationPolicy(SessionCreationPolicy. If_Required )  
}
```

SessionCreationPolicy. Always	: 스프링 시큐리티가 항상 세션 생성
SessionCreationPolicy. If_Required	: 스프링 시큐리티가 필요 시 생성(기본값)
SessionCreationPolicy. Never	: 스프링 시큐리티가 생성하지 않지만 이미 존재하면 사용
SessionCreationPolicy. Stateless	: 스프링 시큐리티가 생성하지 않고 존재해도 사용하지 않음

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



II. 스프링 시큐리티 기본 API & Filter 이해

#11. 인증 API – SessionManagementFilter ConcurrentSessionFilter

II. 스프링 시큐리티 기본 API & Filter 이해

11. 인증 API – SessionManagementFilter

1. 세션 관리

- 인증 시 사용자의 세션정보를 등록, 조회, 삭제 등의 세션 이력을 관리

2. 동시적 세션 제어

- 동일 계정으로 접속이 허용되는 최대 세션수를 제한

3. 세션 고정 보호

- 인증 할 때마다 세션쿠키를 새로 발급하여 공격자의 쿠키 조작을 방지

4. 세션 생성 정책

- Always, If_Required, Never, Stateless

II. 스프링 시큐리티 기본 API & Filter 이해

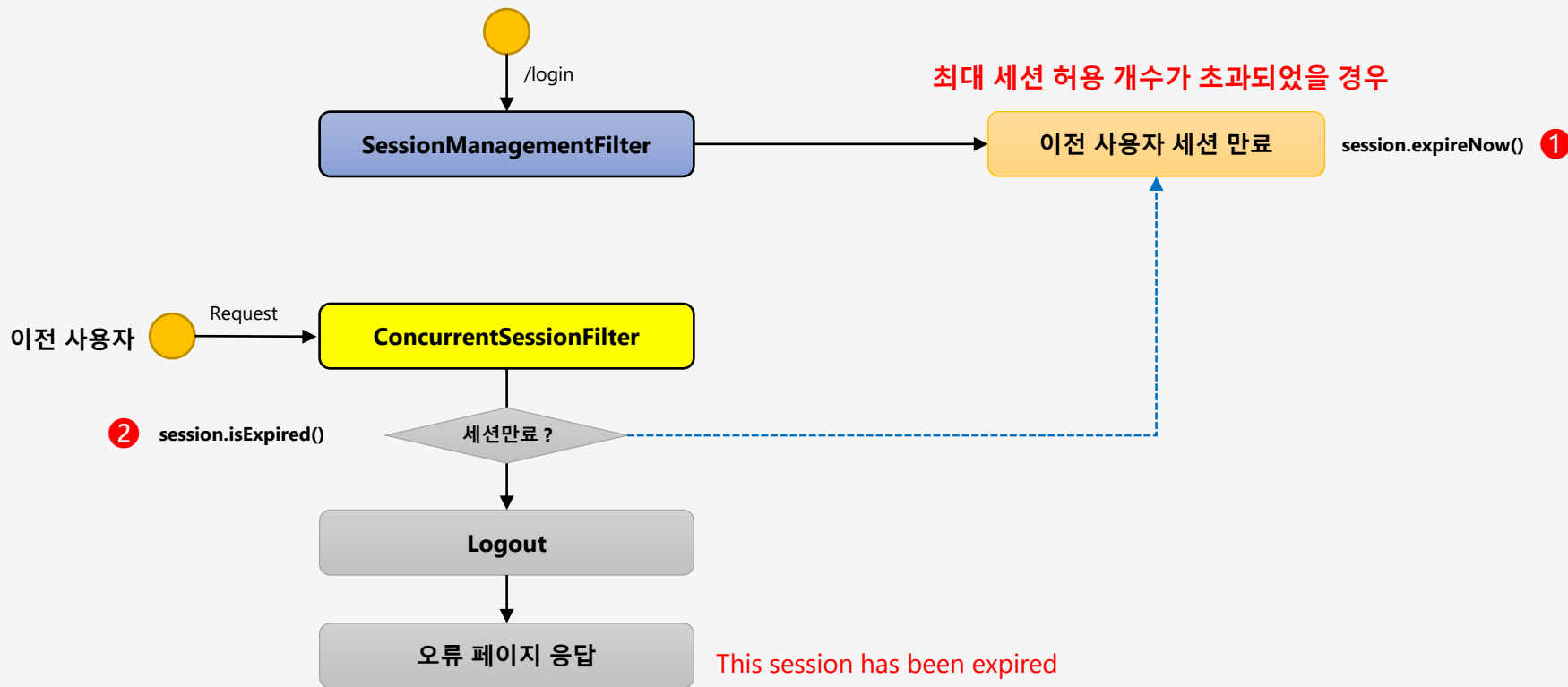
11. 인증 API – ConcurrentSessionFilter

- 매 요청마다 현재 사용자의 세션 만료 여부 체크
- 세션이 만료로 설정되었을 경우 즉시 만료 처리
- **session.isExpired() == true**
 - 로그아웃 처리
 - 즉시 오류 페이지 응답

" This session has been expired "

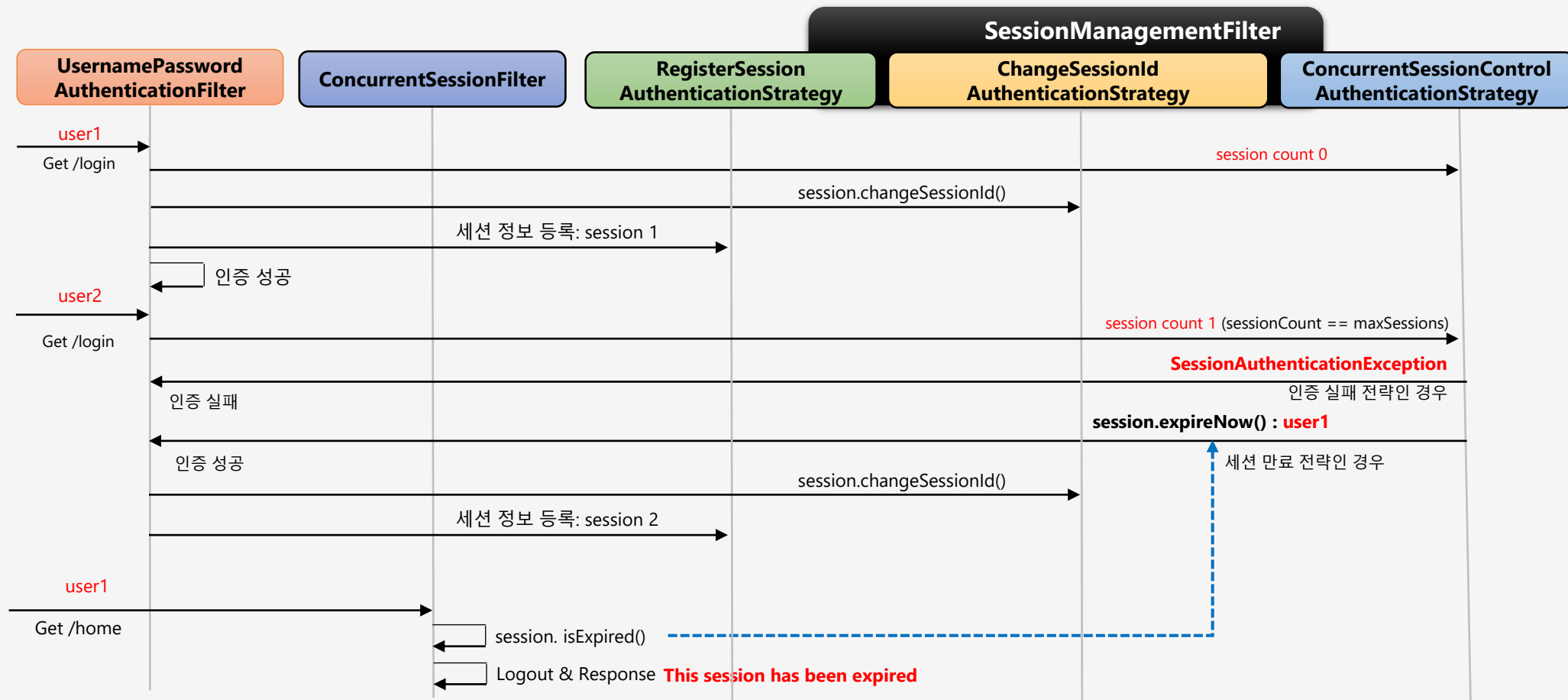
II. 스프링 시큐리티 기본 API & Filter 이해

11. 인증 API – SessionManagementFilter, ConcurrentSessionFilter



II. 스프링 시큐리티 기본 API & Filter 이해

11. 인증 API – ConcurrentSessionFilter, SessionManagementFilter



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



II. 스프링 시큐리티 기본 API & Filter 이해

#12. 인가 API – 권한 설정 및 표현식

#12. 인가 API – 권한 설정

- 선언적 방식
 - URL
 - `http.antMatchers("/users/**").hasRole("USER")`
 - Method
 - `@PreAuthorize("hasRole('USER')")`
`public void user(){ System.out.println("user")}`
- 동적 방식 – DB 연동 프로그래밍
 - URL
 - Method

II. 스프링 시큐리티 기본 API & Filter 이해

#12. 인가 API – 권한 설정

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .antMatcher("/shop/**")
        .authorizeRequests()
            .antMatchers("/shop/login", "/shop/users/**").permitAll()
            .antMatchers("/shop/mypage").hasRole("USER")
            .antMatchers("/shop/admin/pay").access("hasRole('ADMIN')");
            .antMatchers("/shop/admin/**").access("hasRole('ADMIN') or hasRole('SYS ')");
            .anyRequest().authenticated()
        }
}
```

※ 주의 사항 - 설정 시 구체적인 경로가 먼저 오고 그것 보다 큰 범위의 경로가 뒤에 오도록 해야 한다

II. 스프링 시큐리티 기본 API & Filter 이해

#12. 인가 API – 표현식

메소드	동작
authenticated()	인증된 사용자의 접근을 허용
fullyAuthenticated()	인증된 사용자의 접근을 허용, rememberMe 인증 제외
permitAll()	무조건 접근을 허용
denyAll()	무조건 접근을 허용하지 않음
anonymous()	익명사용자의 접근을 허용
rememberMe()	기억하기를 통해 인증된 사용자의 접근을 허용
access(String)	주어진 SpEL 표현식의 평가 결과가 true이면 접근을 허용
hasRole(String)	사용자가 주어진 역할이 있다면 접근을 허용
hasAuthority(String)	사용자가 주어진 권한이 있다면
hasAnyRole(String...)	사용자가 주어진 권한이 있다면 접근을 허용
hasAnyAuthority(String...)	사용자가 주어진 권한 중 어떤 것이라도 있다면 접근을 허용
hasIpAddress(String)	주어진 IP로부터 요청이 왔다면 접근을 허용

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



II. 스프링 시큐리티 기본 API & Filter 이해

#14. 인증/인가 API – `ExceptionTranslationFilter` `RequestCacheAwareFilter`

II. 스프링 시큐리티 기본 API & Filter 이해

#14. 인증/인가 API - ExceptionTranslationFilter

- **AuthenticationException**

- 인증 예외 처리

- 1. **AuthenticationEntryPoint** 호출

- 로그인 페이지 이동, 401 오류 코드 전달 등

- 2. 인증 예외가 발생하기 전의 요청 정보를 저장

- **RequestCache** - 사용자의 이전 요청 정보를 세션에 저장하고 이를 꺼내 오는 캐시 메커니즘
 - **SavedRequest** - 사용자가 요청했던 request 파라미터 값들, 그 당시의 헤더값들 등이 저장

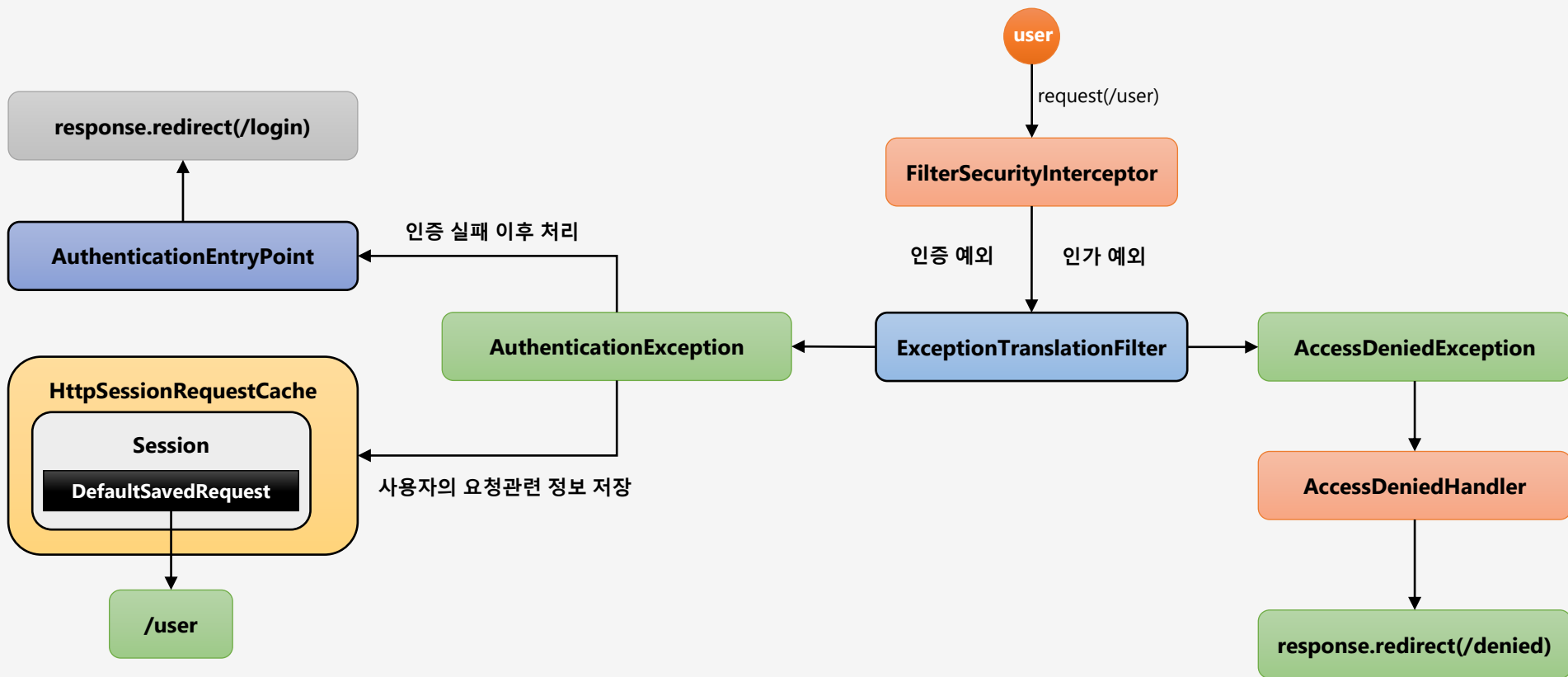
- **AccessDeniedException**

- 인가 예외 처리

- **AccessDeniedHandler** 에서 예외 처리하도록 제공

II. 스프링 시큐리티 기본 API & Filter 이해

#14. 인증/인가 API - ExceptionTranslationFilter



II. 스프링 시큐리티 기본 API & Filter 이해

#13. 인증/인가 API - ExceptionTranslationFilter

http.exceptionHandling() : 예외처리 기능이 작동함

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

```
    http.exceptionHandling()
```

```
        .authenticationEntryPoint(authenticationEntryPoint())
```

// 인증실패 시 처리

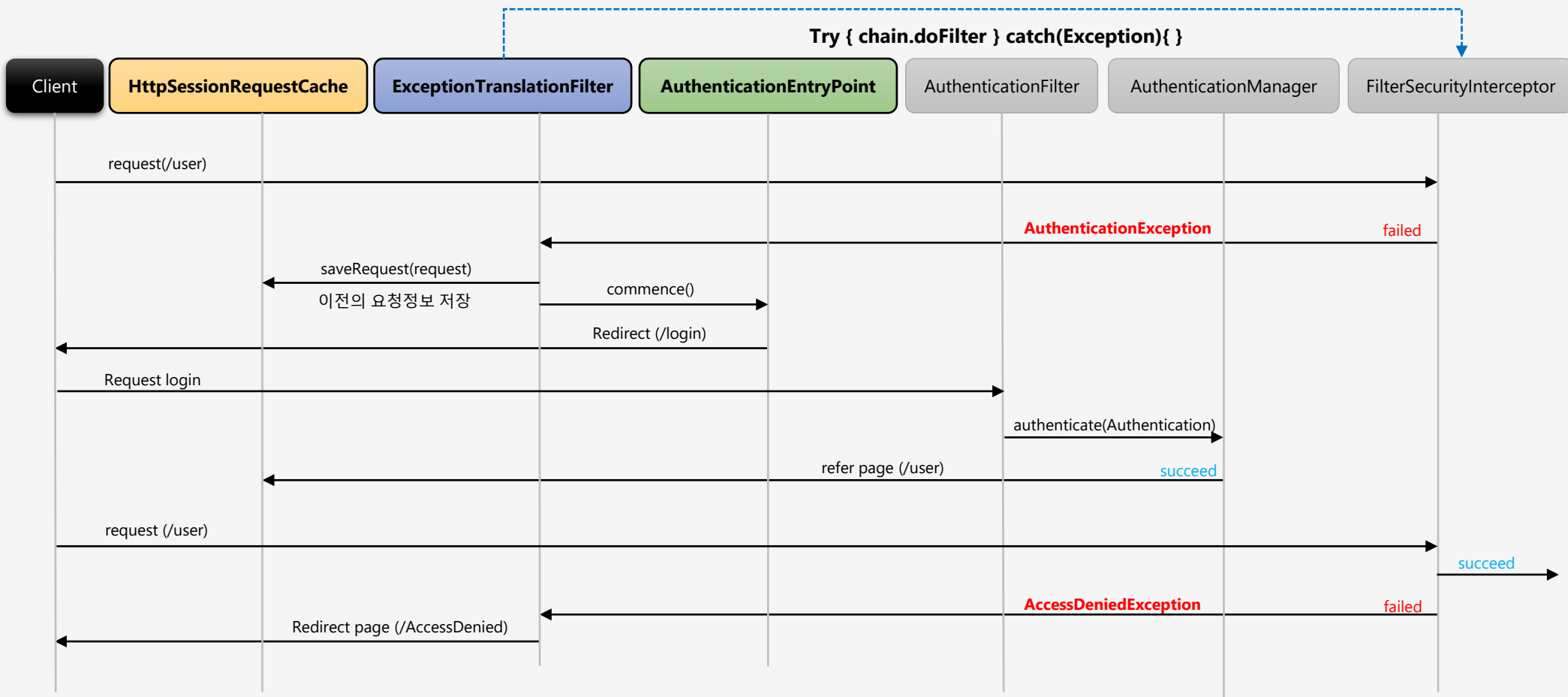
```
        .accessDeniedHandler(accessDeniedHandler())
```

// 인증실패 시 처리

```
}
```

II. 스프링 시큐리티 기본 API & Filter 이해

#14. 인증/인가 API - ExceptionTranslationFilter



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

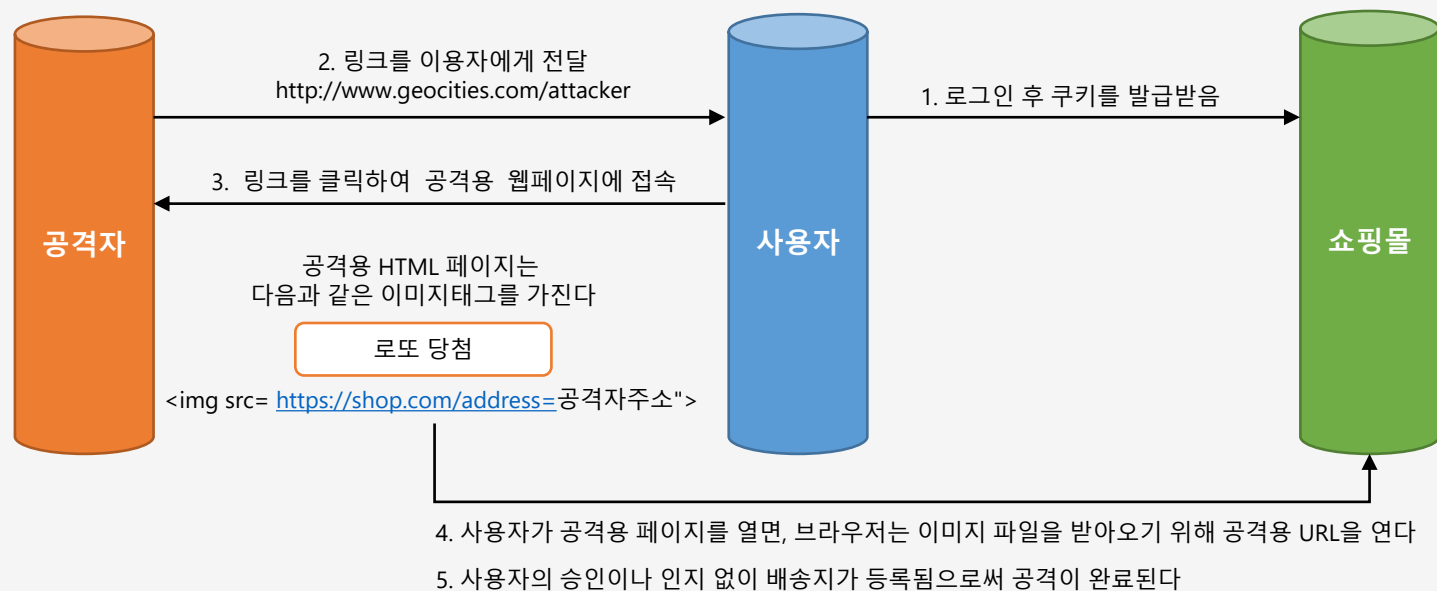


IV. 인증(Authentication) 프로세스 구현

#12. Form 인증 – CSRF, CsrfFilter

IV. 인증(Authentication) 프로세스 구현

#12. Form 인증 - CSRF(사이트 간 요청 위조)



IV. 인증(Authentication) 프로세스 구현

#12. Form 인증 – CsrfFilter

- 모든 요청에 랜덤하게 생성된 토큰을 HTTP 파라미터로 요구
- 요청 시 전달되는 토큰 값과 서버에 저장된 실제 값과 비교한 후 만약 일치하지 않으면 요청은 실패한다

- **Client**

- `<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}" />`
- HTTP 메소드 : PATCH, POST, PUT, DELETE

- **Spring Security**

- `http.csrf()` : 기본 활성화되어 있음
- `http.csrf().disabled()` : 비활성화

Core Spring Security

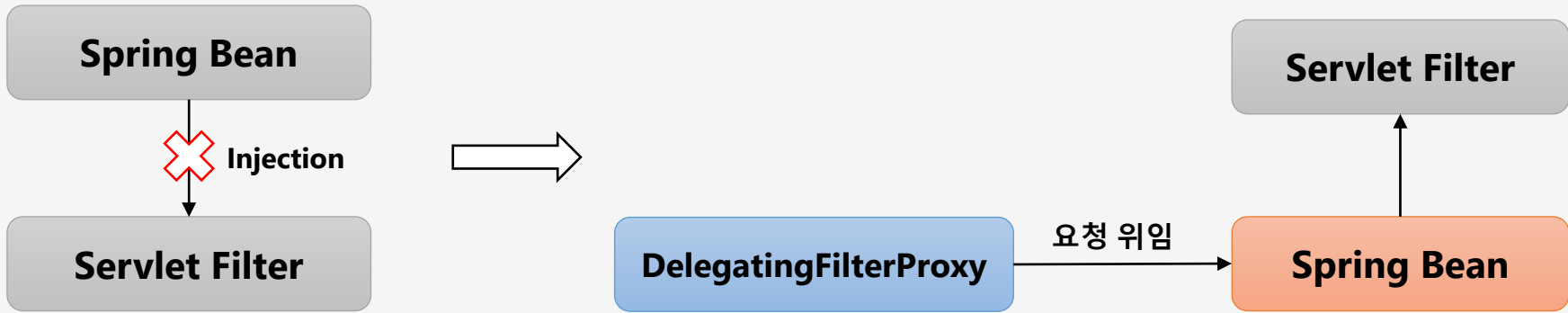
핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#01. DelegatingFilterProxy, FilterChainProxy

#01. DelegatingFilterProxy



- 1. 서블릿 필터는 스프링에서 정의된 빈을 주입해서 사용할 수 없음
- 2. 특정한 이름을 가진 스프링 빈을 찾아 그 빈에게 요청을 위임
 - `springSecurityFilterChain` 이름으로 생성된 빈을 `ApplicationContext` 에서 찾아 요청을 위임
 - 실제 보안처리를 하지 않음

III. 스프링 시큐리티 주요 아키텍처 이해

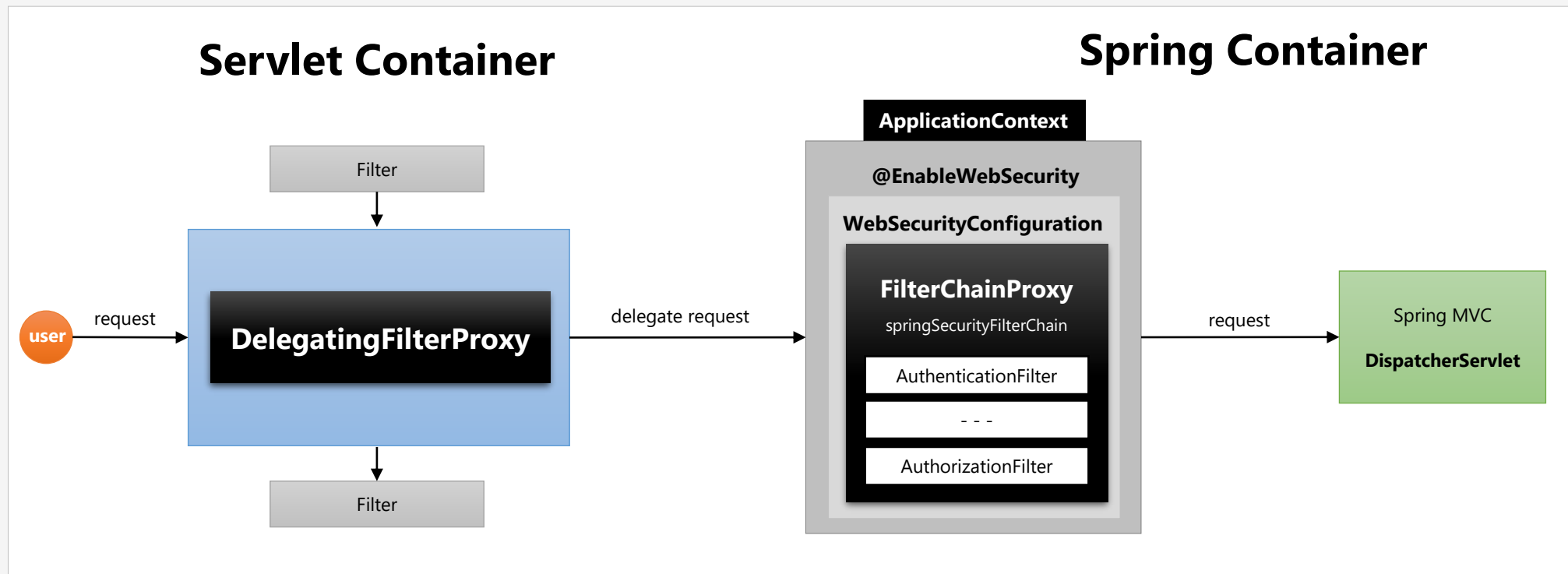
#01. FilterChainProxy

```
▶ 0 = {WebAsyncManagerIntegrationFilter@8791}
▶ 1 = {SecurityContextPersistenceFilter@8795}
▶ 2 = {HeaderWriterFilter@8800}
▶ 3 = {CsrfFilter@8805}
▶ 4 = {LogoutFilter@8808}
▶ 5 = {UsernamePasswordAuthenticationFilter@8811}
▶ 6 = {DefaultLoginPageGeneratingFilter@8837}
▶ 7 = {DefaultLogoutPageGeneratingFilter@8840}
▶ 8 = {ConcurrentSessionFilter@8843}
▶ 9 = {RequestCacheAwareFilter@8846}
▶ 10 = {SecurityContextHolderAwareRequestFilter@8849}
▶ 11 = {AnonymousAuthenticationFilter@8854}
▶ 12 = {SessionManagementFilter@8858}
▶ 13 = {ExceptionTranslationFilter@8862}
▶ 14 = {FilterSecurityInterceptor@8927}
```

1. **springSecurityFilterChain** 의 이름으로 생성되는 필터 빈
2. **DelegatingFilterProxy** 으로 부터 요청을 위임 받고 실제 보안 처리
3. 스프링 시큐리티 초기화 시 생성되는 필터들을 관리하고 제어
 - 스프링 시큐리티가 기본적으로 생성하는 필터
 - 설정 클래스에서 API 추가 시 생성되는 필터
4. 사용자의 요청을 필터 순서대로 호출하여 전달
5. 사용자정의 필터를 생성해서 기존의 필터 전.후로 추가 가능
 - 필터의 순서를 잘 정의
6. 마지막 필터까지 인증 및 인가 예외가 발생하지 않으면 보안 통과

III. 스프링 시큐리티 주요 아키텍처 이해

#01. DelegatingFilterProxy, FilterChainProxy



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

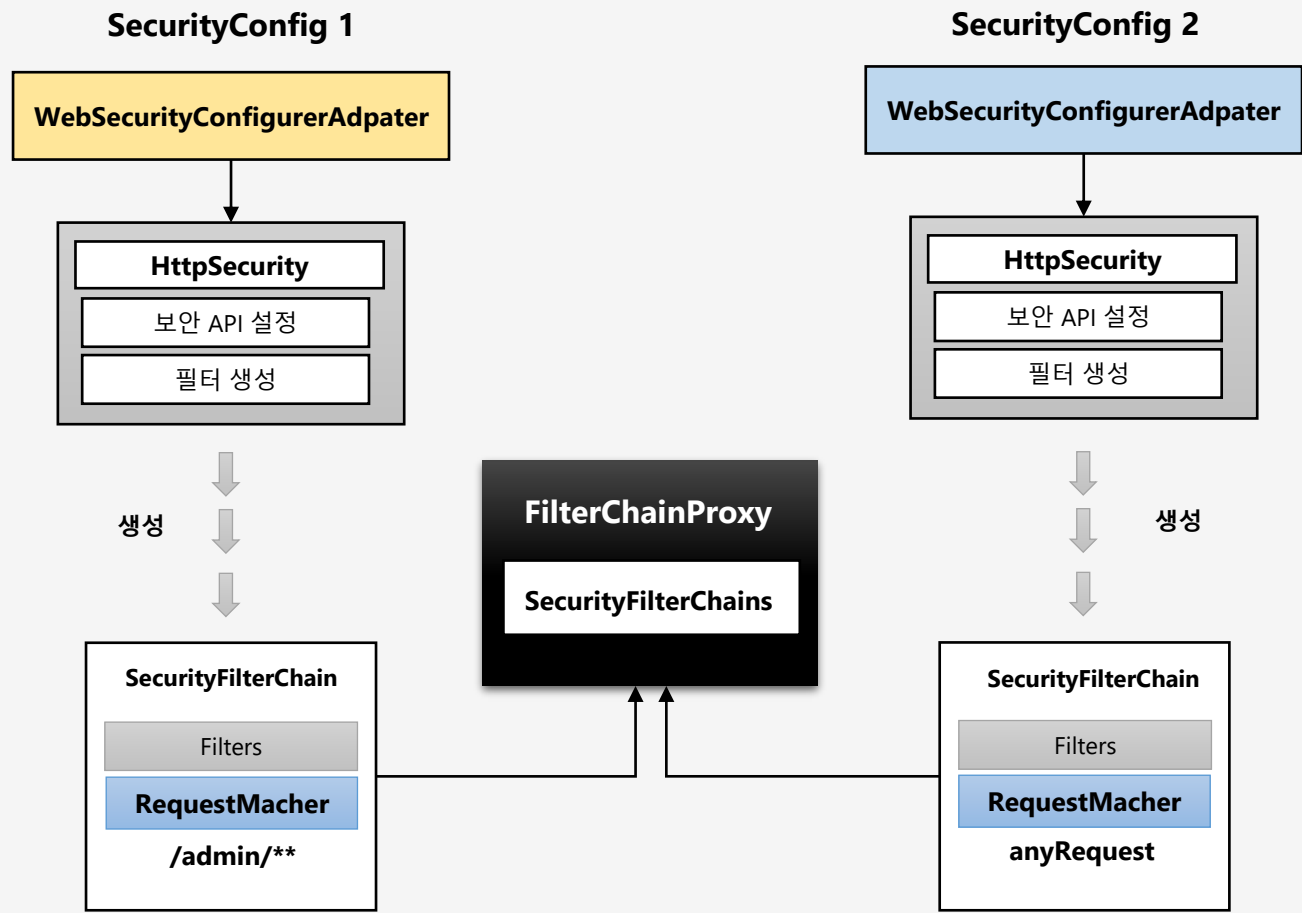


Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#02. 필터 초기화와 다중 보안 설정

III. 스프링 시큐리티 주요 아키텍처 이해

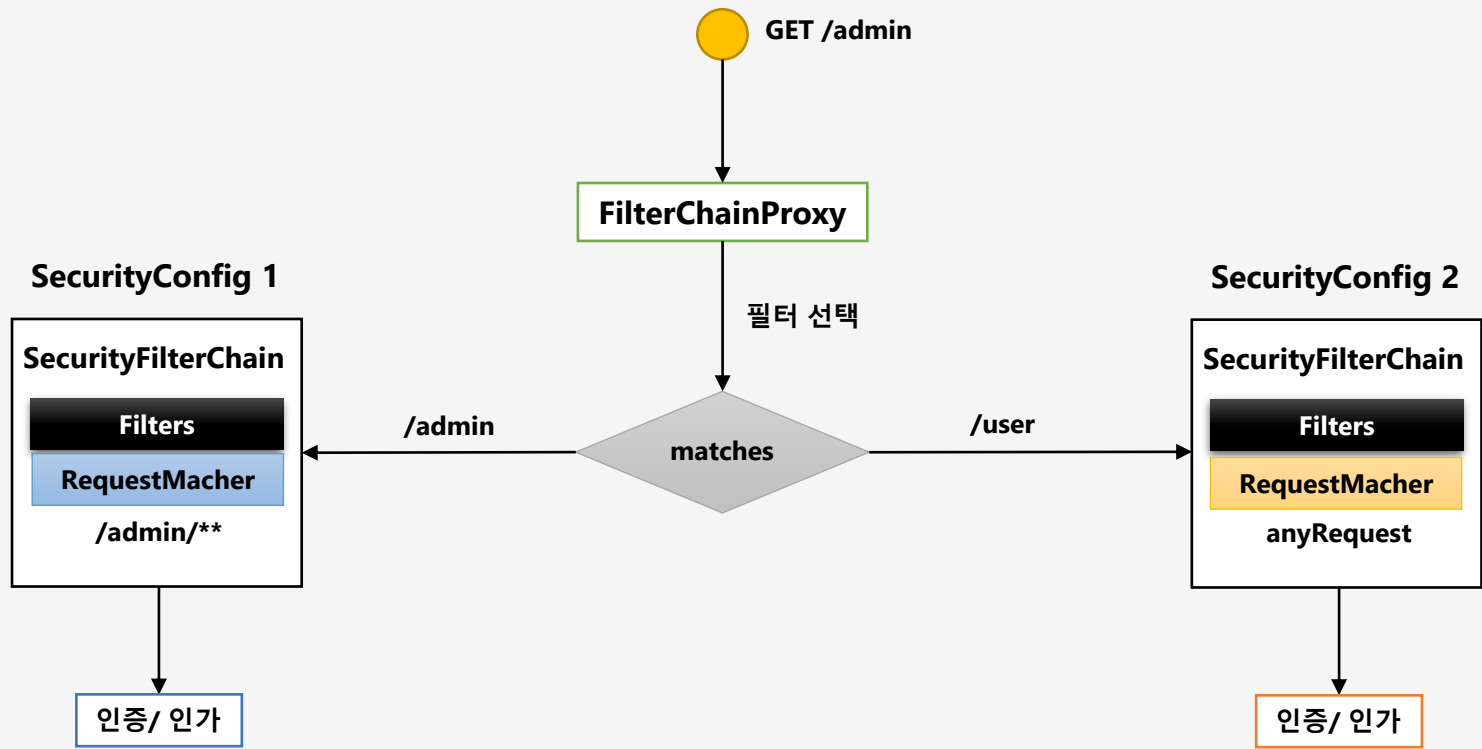
#02. 필터 초기화와 다중 설정 클래스



- 설정클래스 별로 보안 기능이 각각 작동
- 설정클래스 별로 **RequestMatcher** 설정
 - `http.antMatcher("/admin/**")`
- 설정클래스 별로 필터가 생성
- **FilterChainProxy** 가 각 필터들 가지고 있음
- 요청에 따라 **RequestMatcher**와 매칭되는 필터가 작동하도록 함

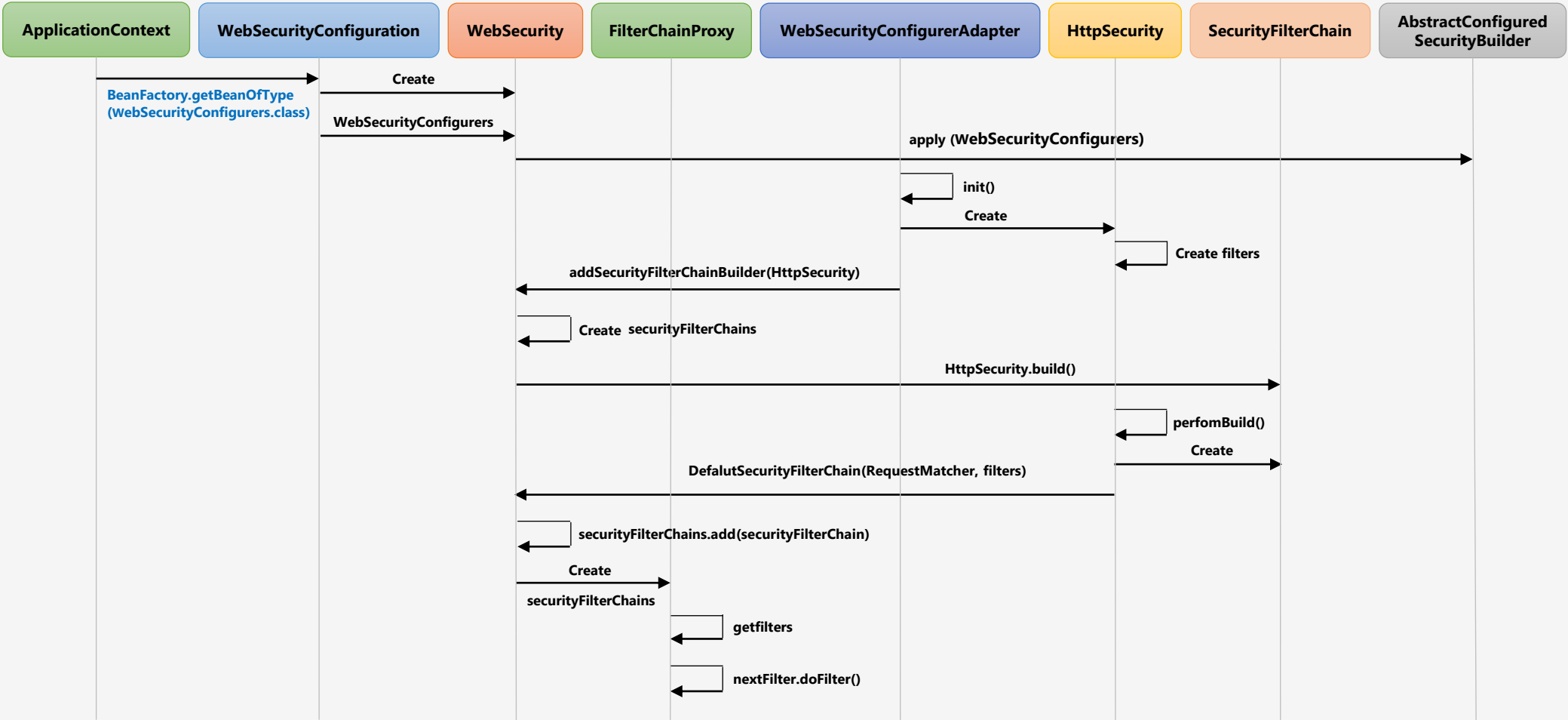
III. 스프링 시큐리티 주요 아키텍처 이해

#02. 필터 초기화와 다중 설정 클래스



III. 스프링 시큐리티 주요 아키텍처 이해

#02. WebSecurity, HttpSecurity, WebSecurityConfigurerAdapter



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

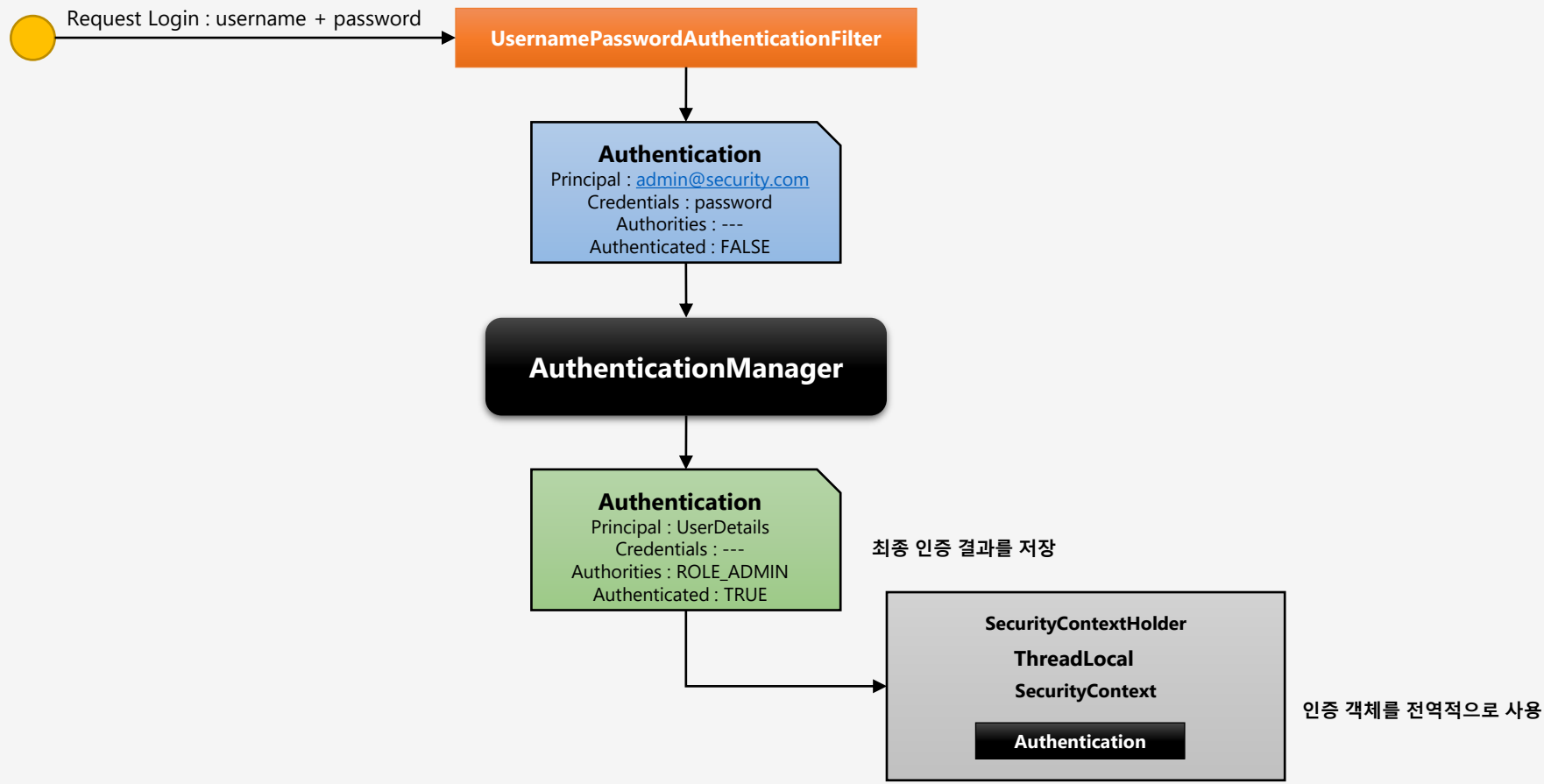
#03. Authentication

III. 스프링 시큐리티 주요 아키텍처 이해

#03. Authentication

- 당신이 누구인지 증명하는 것
 - 사용자의 인증 정보를 저장하는 토큰 개념
 - 인증 시 id 와 password 를 담고 인증 검증을 위해 전달되어 사용된다
 - 인증 후 최종 인증 결과 (user 객체, 권한정보) 를 담고 SecurityContext 에 저장되어 전역적으로 참조가 가능하다
 - **Authentication authentication = SecurityContextHolder.getContext().getAuthentication()**
 - 구조
 - 1) **principal** : 사용자 아이디 혹은 User 객체를 저장
 - 2) **credentials** : 사용자 비밀번호
 - 3) **authorities** : 인증된 사용자의 권한 목록
 - 4) **details** : 인증 부가 정보
 - 5) **Authenticated** : 인증 여부

#03. Authentication



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#04. SecurityContextHolder, SecurityContext

III. 스프링 시큐리티 주요 아키텍처 이해

#04. SecurityContextHolder, SecurityContext

- **SecurityContext**

- Authentication 객체가 저장되는 보관소로 필요 시 언제든지 Authentication 객체를 꺼내어 쓸 수 있도록 제공되는 클래스
- ThreadLocal 에 저장되어 아무 곳에서나 참조가 가능하도록 설계함
- 인증이 완료되면 HttpSession 에 저장되어 어플리케이션 전반에 걸쳐 전역적인 참조가 가능하다

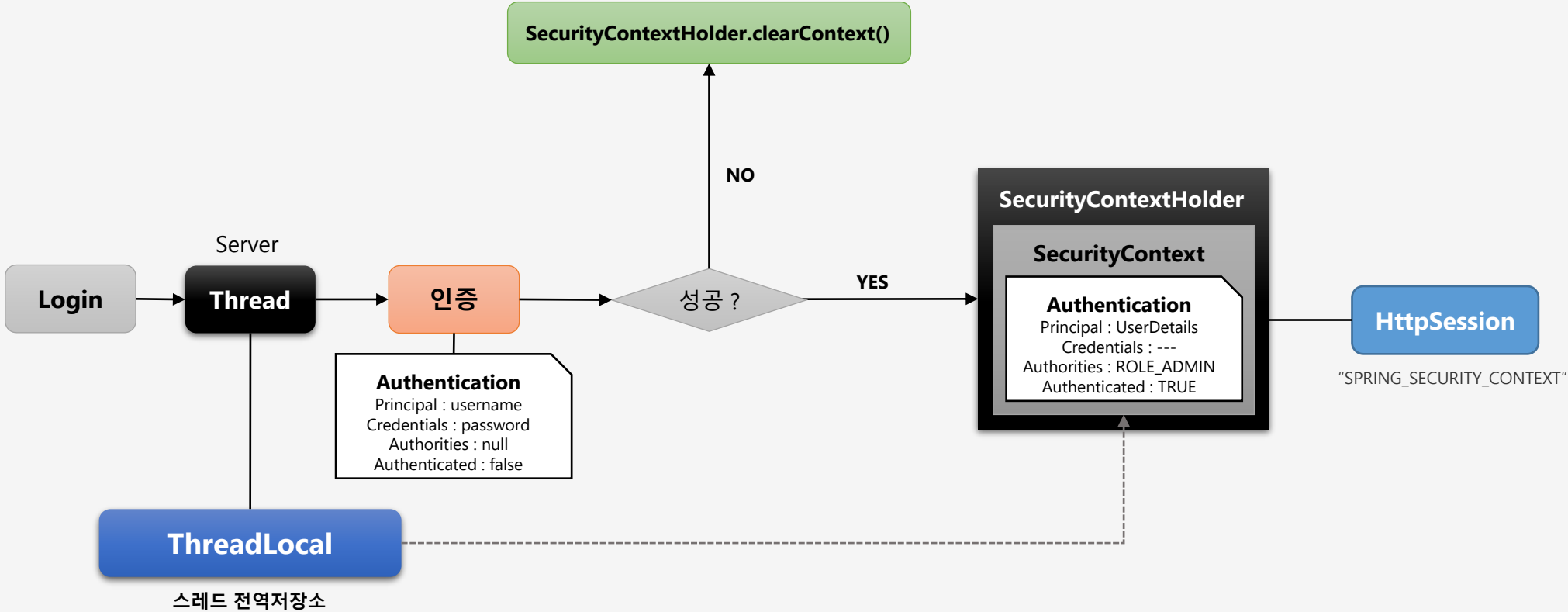
- **SecurityContextHolder**

- SecurityContext 객체 저장 방식
 - MODE_THREADLOCAL : 스레드당 SecurityContext 객체를 할당, 기본값
 - MODE_INHERITABLETHREADLOCAL : 메인 스레드와 자식 스레드에 관하여 동일한 SecurityContext 를 유지
 - MODE_GLOBAL : 응용 프로그램에서 단 하나의 SecurityContext를 저장한다
- SecurityContextHolder.clearContext() : SecurityContext 기존 정보 초기화

- **Authentication authentication = SecurityContextHolder.getContext().getAuthentication()**

III. 스프링 시큐리티 주요 아키텍처 이해

#04. SecurityContextHolder, SecurityContext



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

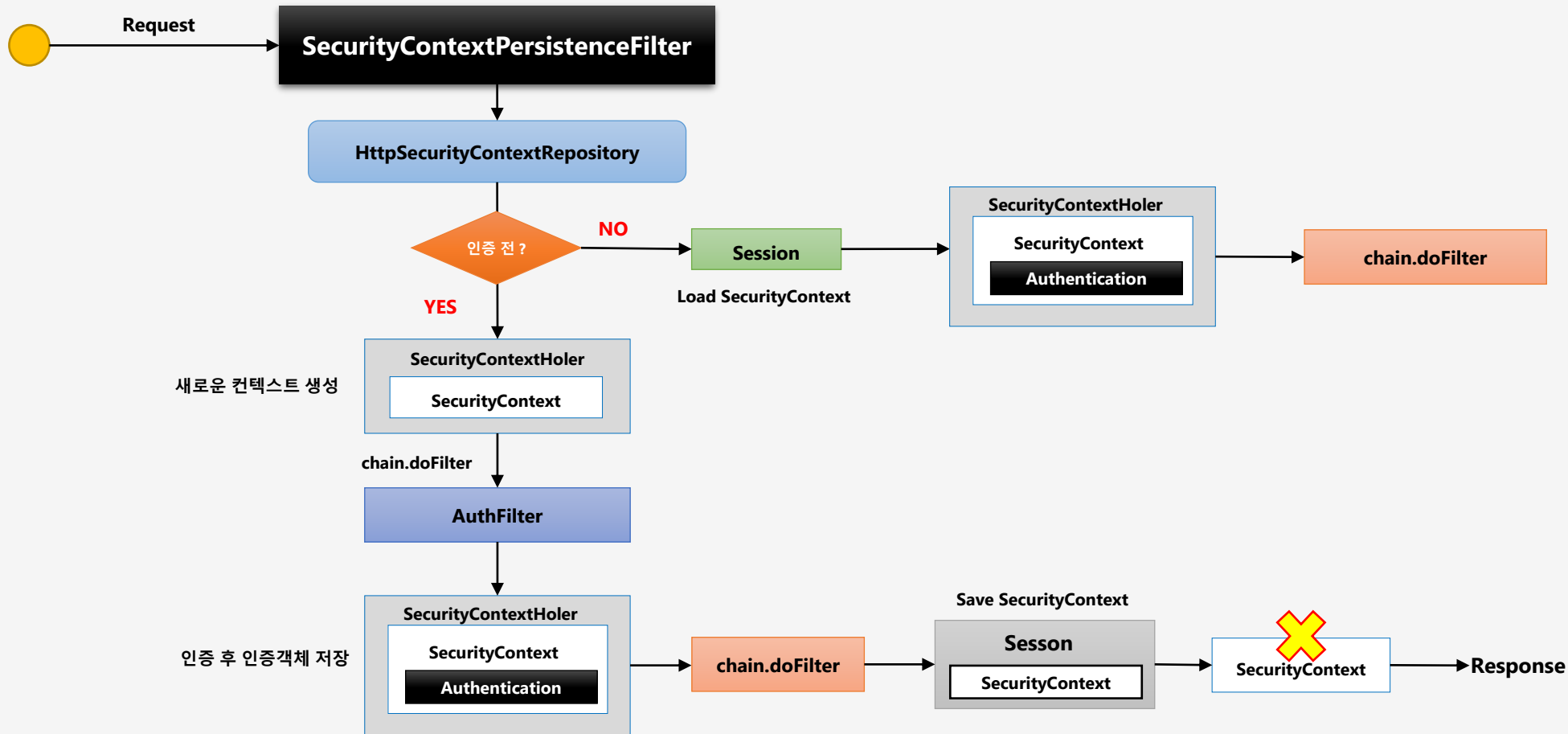
#05. SecurityContextPersistenceFilter

III. 스프링 시큐리티 주요 아키텍처 이해

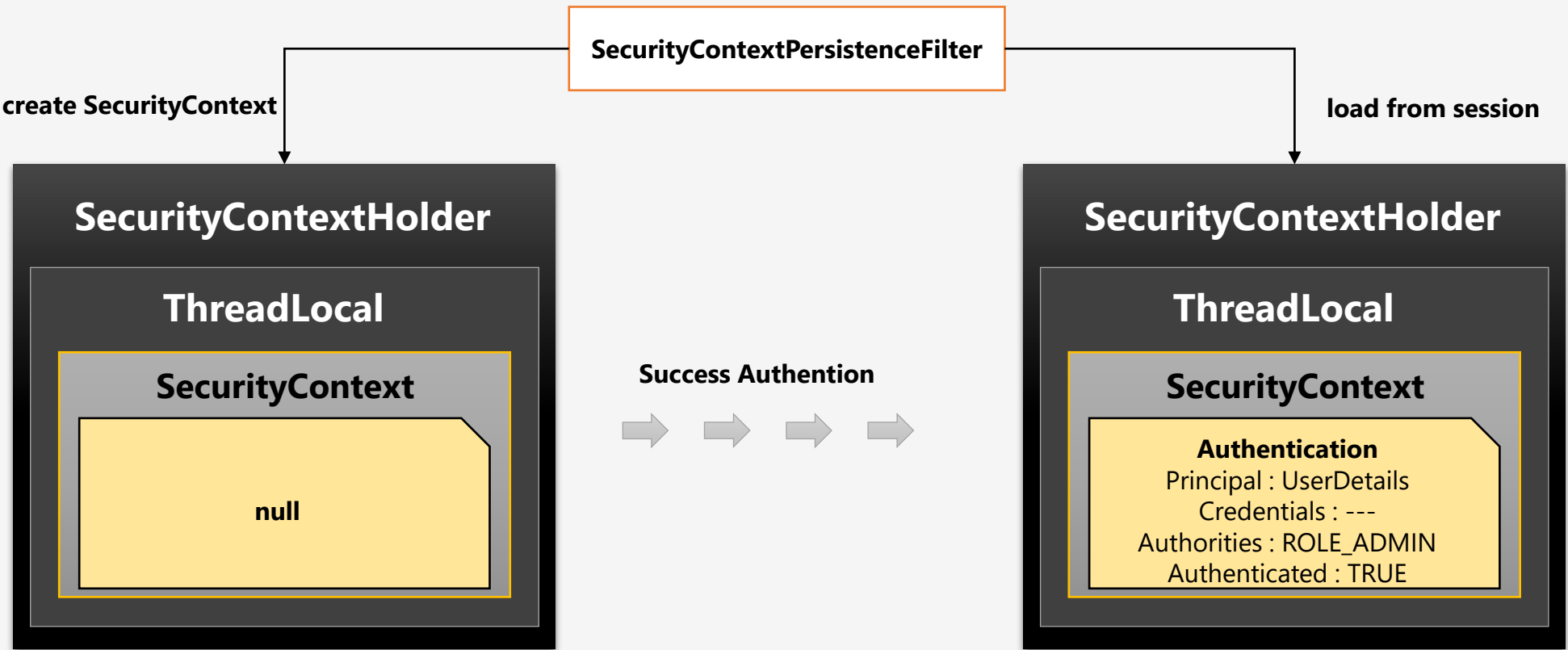
#05. SecurityContextPersistenceFilter

- **SecurityContext 객체의 생성, 저장, 조회**
- **익명 사용자**
 - 새로운 SecurityContext 객체를 생성하여 SecurityContextHolder 에 저장
 - AnonymousAuthenticationFilter 에서 AnonymousAuthenticationToken 객체를 SecurityContext 에 저장
- **인증 시**
 - 새로운 SecurityContext 객체를 생성하여 SecurityContextHolder 에 저장
 - UsernamePasswordAuthenticationFilter 에서 인증 성공 후 SecurityContext 에 UsernamePasswordAuthenticationToken 객체를 SecurityContext 에 저장
 - 인증이 최종 완료되면 Session 에 SecurityContext 를 저장
- **인증 후**
 - Session 에서 SecurityContext 꺼내어 SecurityContextHolder 에서 저장
 - SecurityContext 안에 Authentication 객체가 존재하면 계속 인증을 유지한다
- **최종 응답 시 공통**
 - SecurityContextHolder.clearContext()

#05. SecurityContextPersistenceFilter

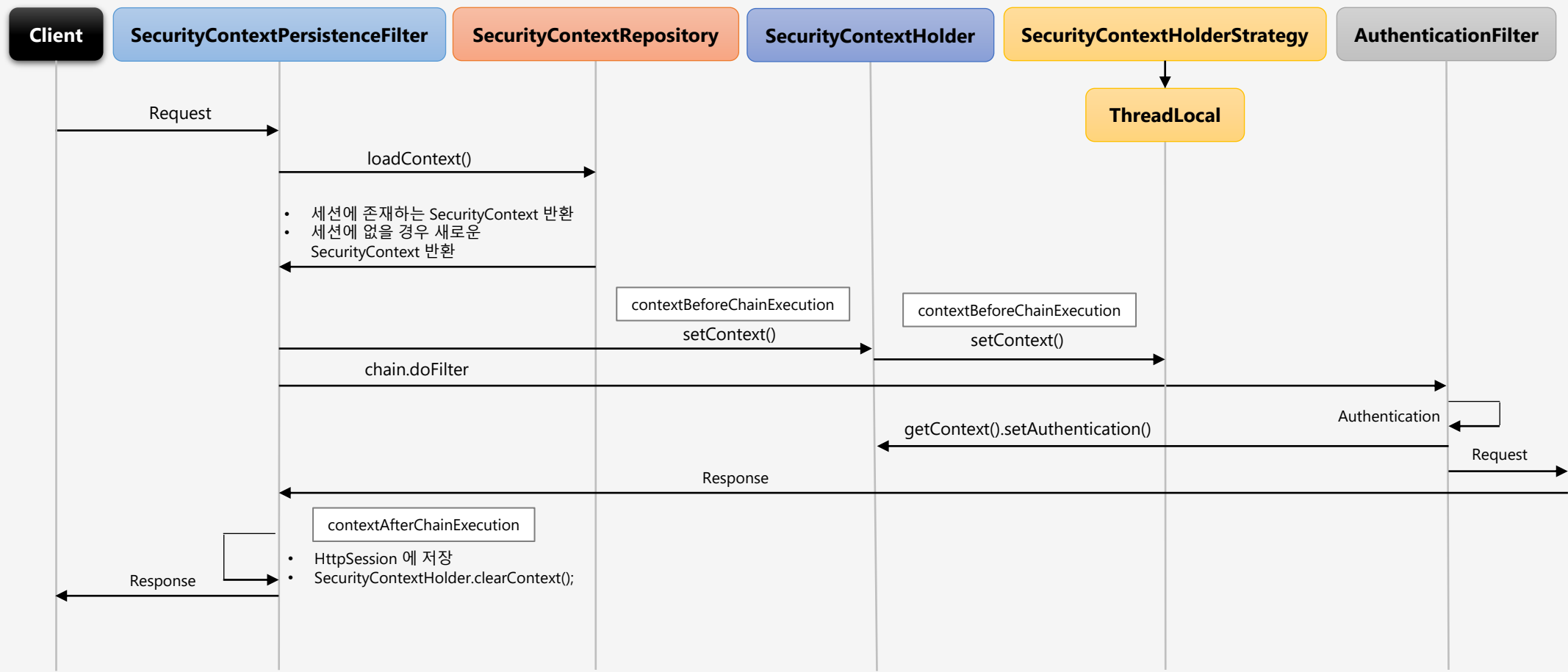


#05. SecurityContextPersistenceFilter



III. 스프링 시큐리티 주요 아키텍처 이해

#05. SecurityContextPersistenceFilter



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

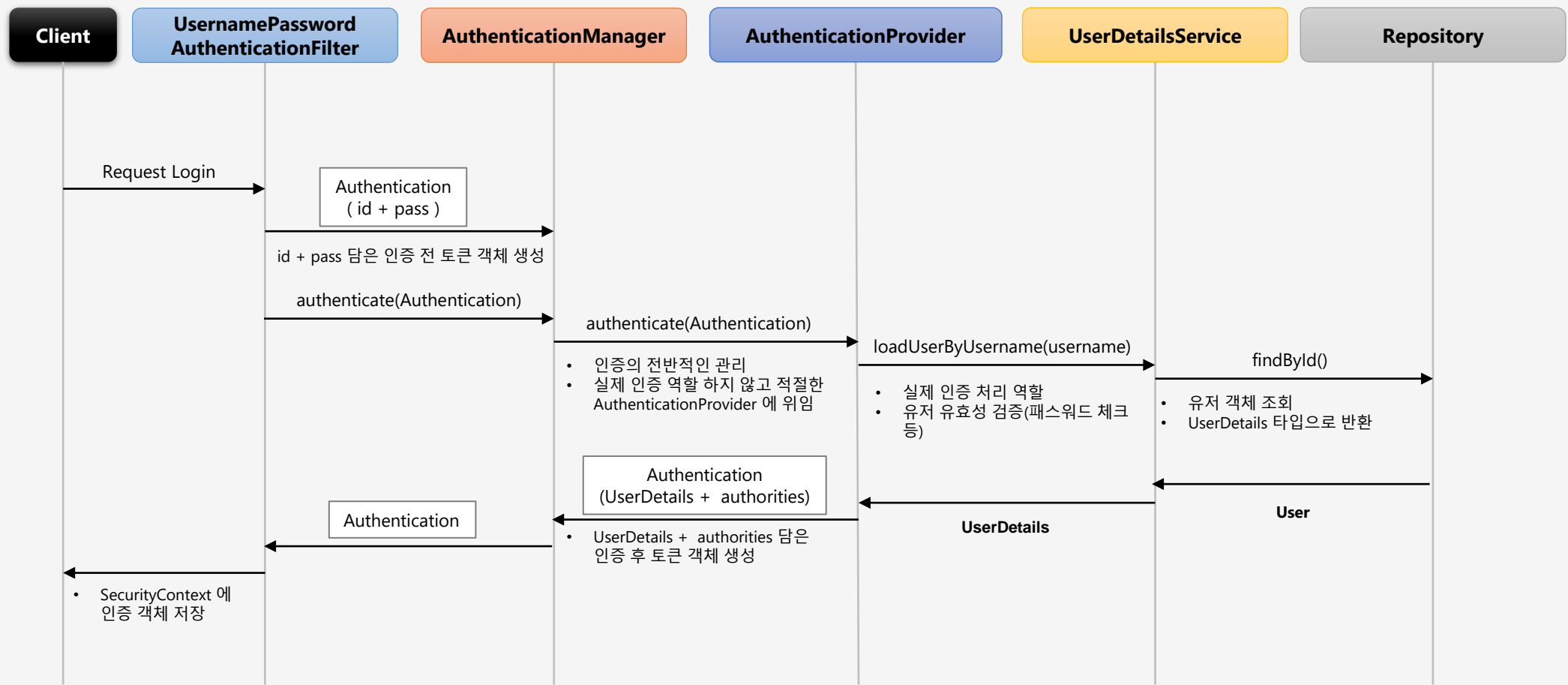


Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#10. Authentication Flow

III. 스프링 시큐리티 주요 아키텍처 이해

#10. Authentication Flow



Core Spring Security

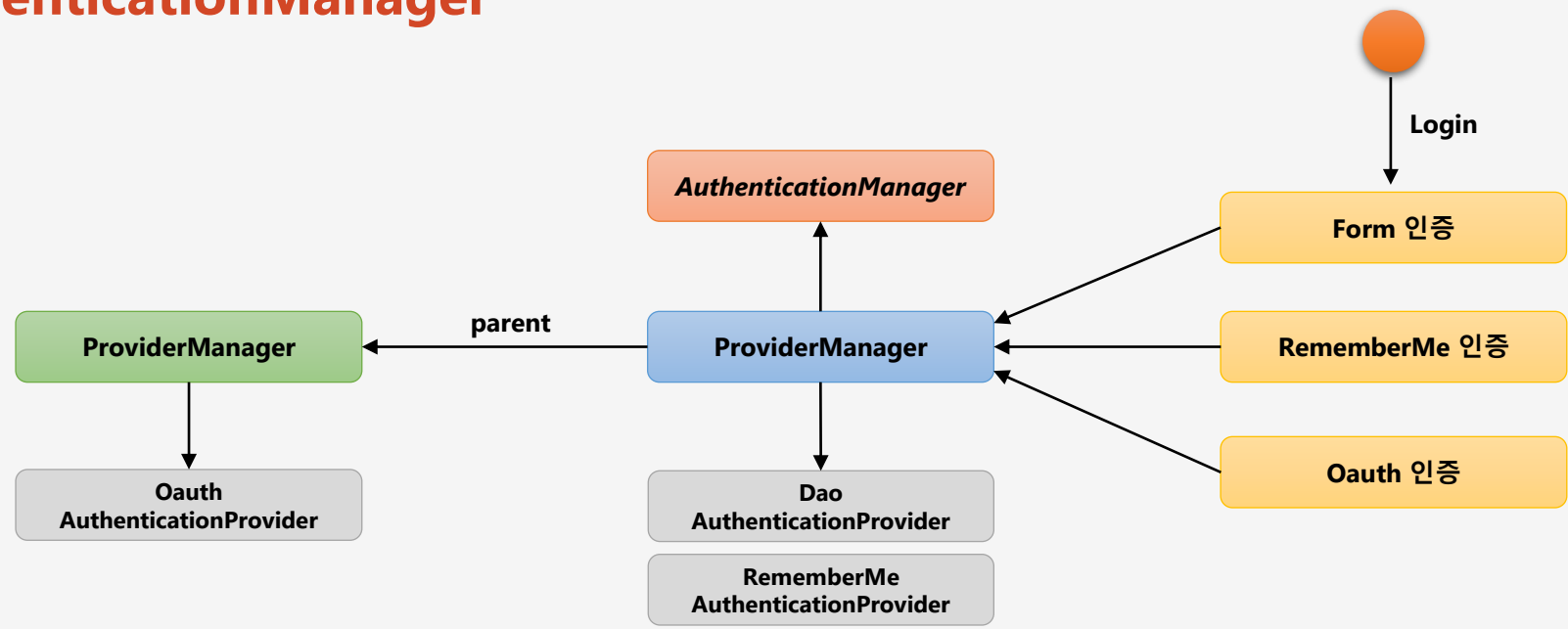
핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#06. AuthenticationManager

#06. AuthenticationManager



- AuthenticationProvider 목록 중에서 인증 처리 요건에 맞는 AuthenticationProvider 를 찾아 인증처리를 위임한다
- 부모 ProviderManager 를 설정하여 AuthenticationProvider 를 계속 탐색 할 수 있다

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

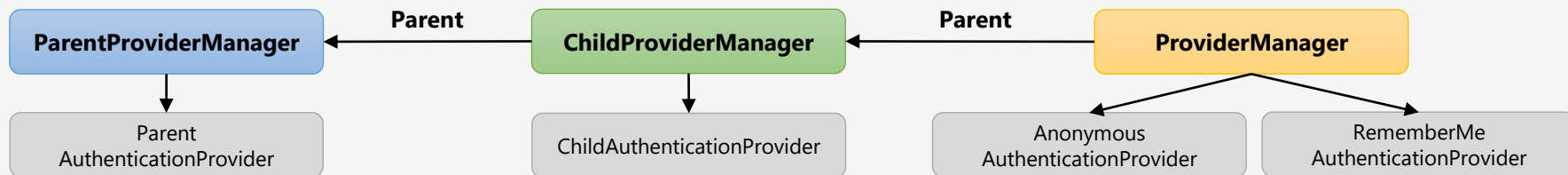
#06. AuthenticationManager 응용편

III. 스프링 시큐리티 주요 아키텍처 이해

#06. AuthenticationManager 응용편

CUSTOM

SECURITY



- **Linked** 형태로 부모와 자식간의 관계를 형성할 수 있다
- 자식에서 적절한 **AuthenticationProvider**를 찾지 못할 경우 계속 부모로 탐색하여 찾는 과정을 반복한다
- **AuthenticationManagerBuilder** 를 사용해서 스프링 시큐리티의 초기화 과정에서 설정한 기본 **Parent** 관계를 변경해야 권한 필터에서 재 인증 시 모든 **AuthenticationProvider** 를 탐색할 수 있다

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

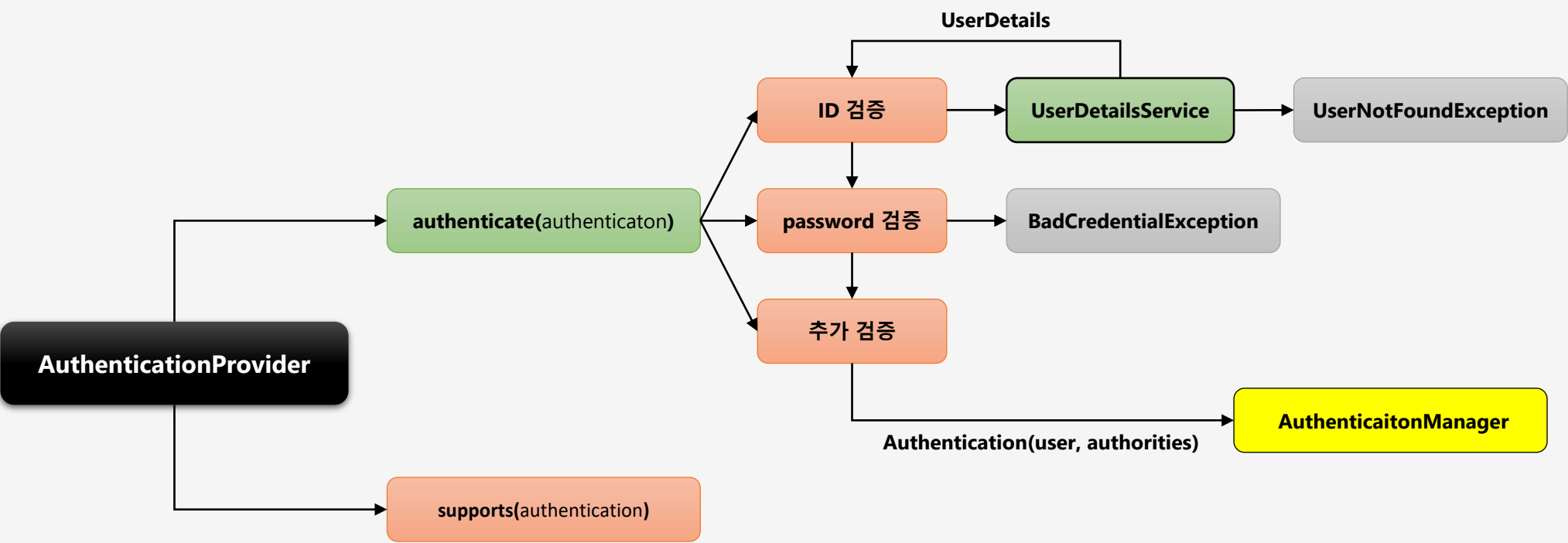


Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#07. AuthenticationProvider

III. 스프링 시큐리티 주요 아키텍처 이해

#07. AuthenticationProvider



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

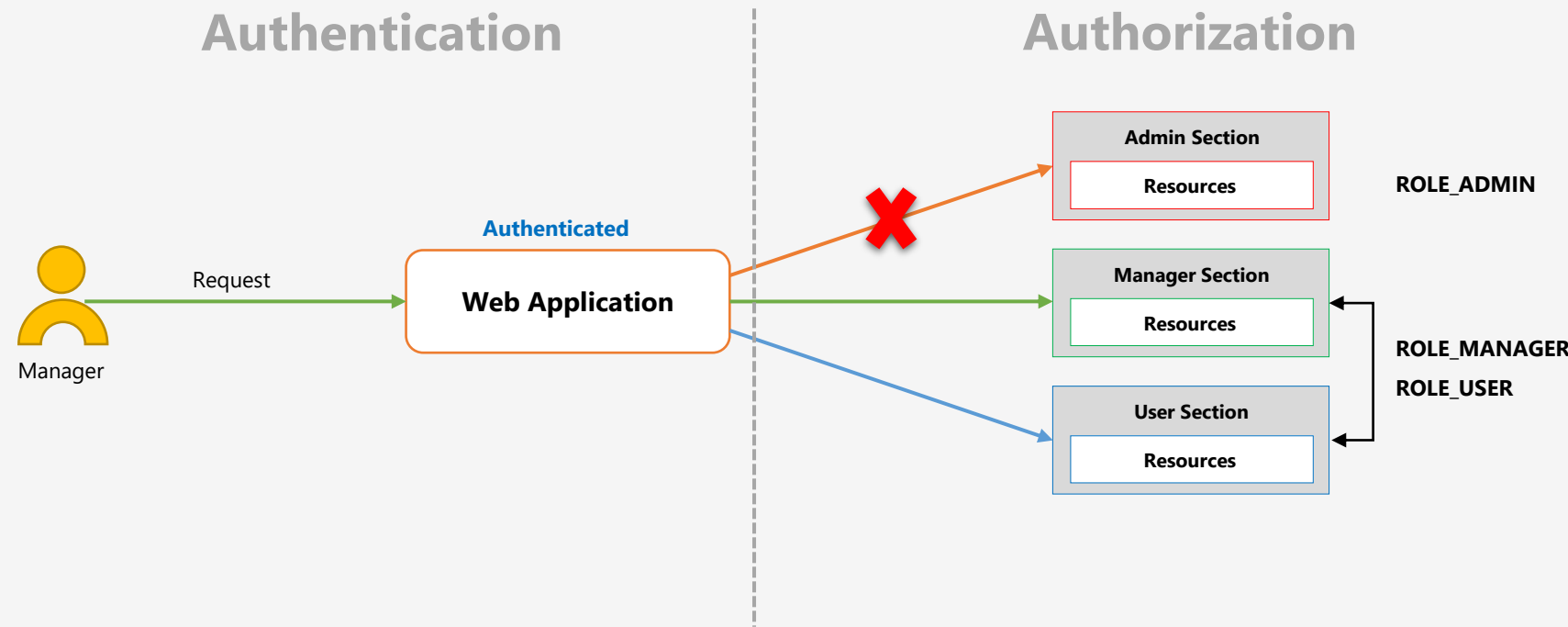


Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#08. Authorization, FilterSecurityInterceptor

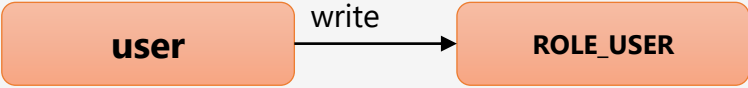
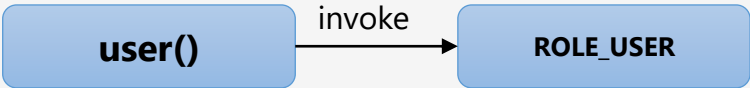
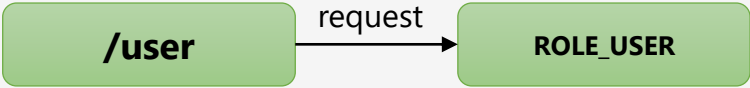
#08. Authorization

- 당신에게 무엇이 허가 되었는지 증명하는 것



#08. Authorization

- 스프링 시큐리티가 지원하는 권한 계층
 - 웹 계층
 - URL 요청에 따른 메뉴 혹은 화면단위의 레벨 보안
 - 서비스 계층
 - 화면 단위가 아닌 메소드 같은 기능 단위의 레벨 보안
 - 도메인 계층(Access Control List, 접근제어목록)
 - 객체 단위의 레벨 보안



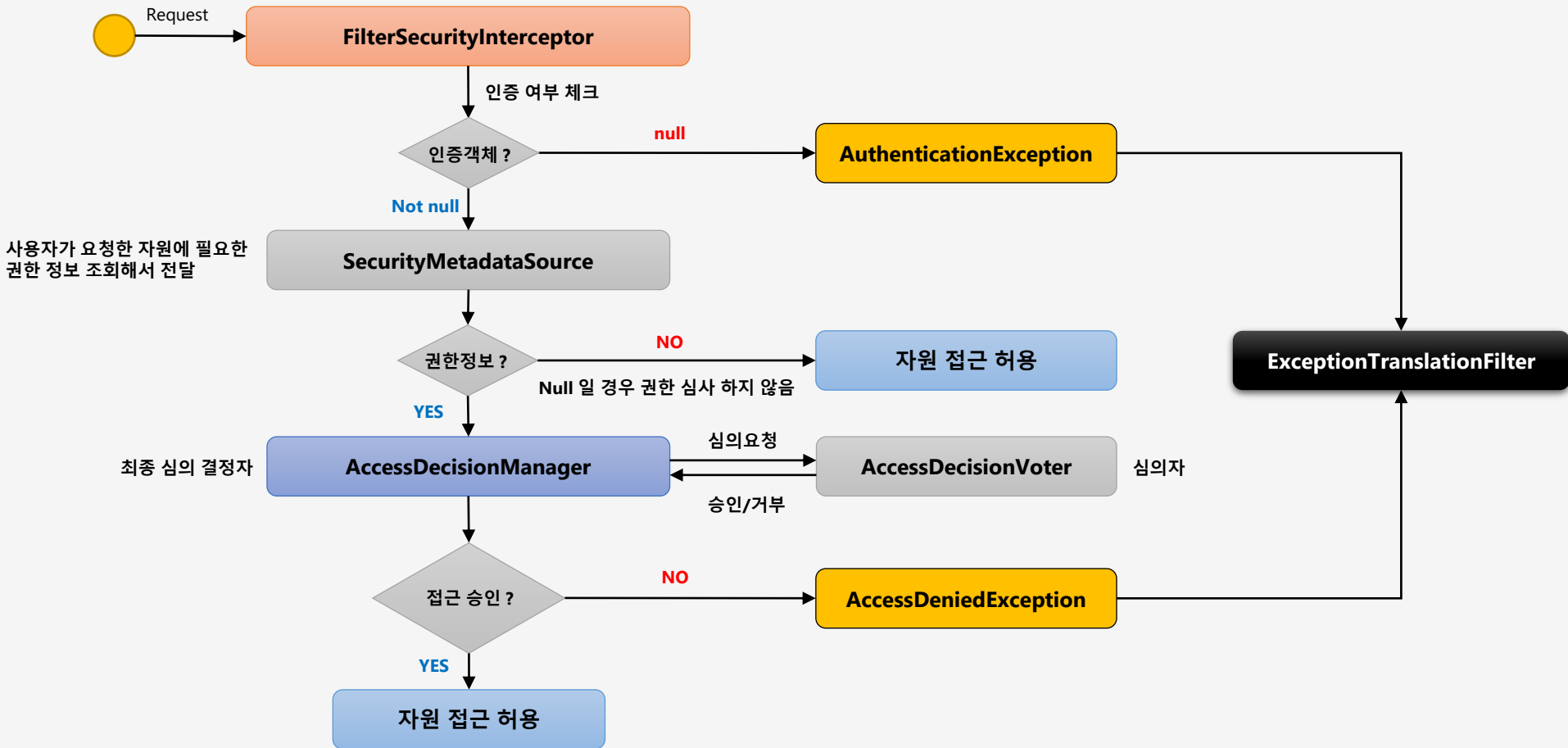
III. 스프링 시큐리티 주요 아키텍처 이해

#08. FilterSecurityInterceptor

- 마지막에 위치한 필터로써 인증된 사용자에게 대하여 특정 요청의 승인/거부 여부를 최종적으로 결정
- 인증객체 없이 보호자원에 접근을 시도할 경우 **AuthenticationException** 을 발생
- 인증 후 자원에 접근 가능한 권한이 존재하지 않을 경우 **AccessDeniedException** 을 발생
- 권한 제어 방식 중 HTTP 자원의 보안을 처리하는 필터
- 권한 처리를 **AccessDecisionManager**에게 맡김

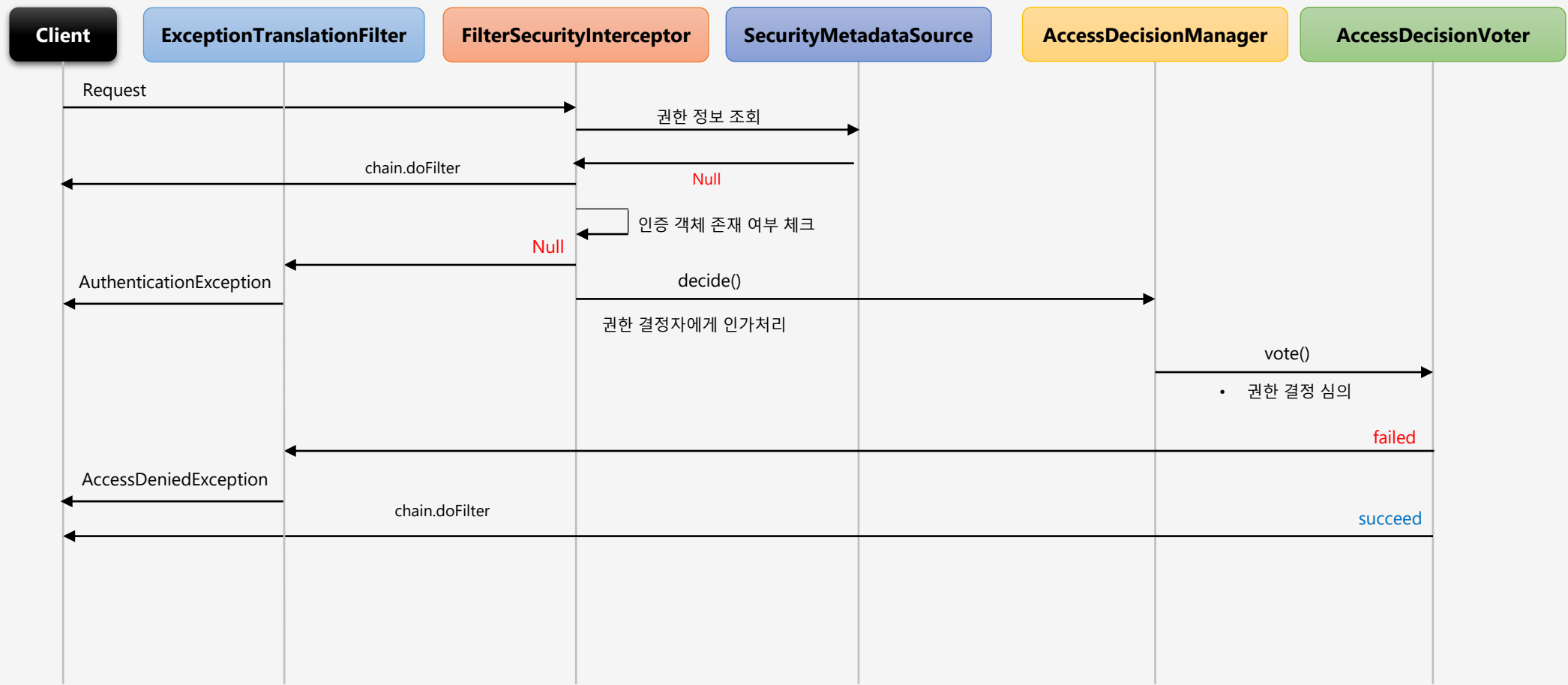
III. 스프링 시큐리티 주요 아키텍처 이해

#08. FilterSecurityInterceptor



III. 스프링 시큐리티 주요 아키텍처 이해

08. FilterSecurityInterceptor



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#09. AccessDecisionManager, AccessDecisionVoter

III. 스프링 시큐리티 주요 아키텍처 이해

#09. AccessDecisionManager

- 인증 정보, 요청정보, 권한정보를 이용해서 사용자의 자원접근을 허용할 것인지 거부할 것인지를 최종 결정하는 주체
- 여러 개의 Voter 들을 가질 수있으며 Voter 들로부터 접근허용, 거부, 보류에 해당하는 각각의 값을 리턴받고 판단 및 결정
- 최종 접근 거부 시 예외 발생

- 접근결정의 세가지 유형

- **AffirmativeBased :**

- 여러개의 Voter 클래스 중 하나라도 접근 허가로 결론을 내면 접근 허가로 판단한다



- **ConsensusBased :**

- 다수표(승인 및 거부)에 의해 최종 결정을 판단한다
 - 동수일경우 기본은 접근허가이나 allowIfEqualGrantedDeniedDecisions 을 false 로 설정할 경우 접근거부로 결정된다



- **UnanimousBased :**

- 모든 보터가 만장일치로 접근을 승인해야 하며 그렇지 않은 경우 접근을 거부한다

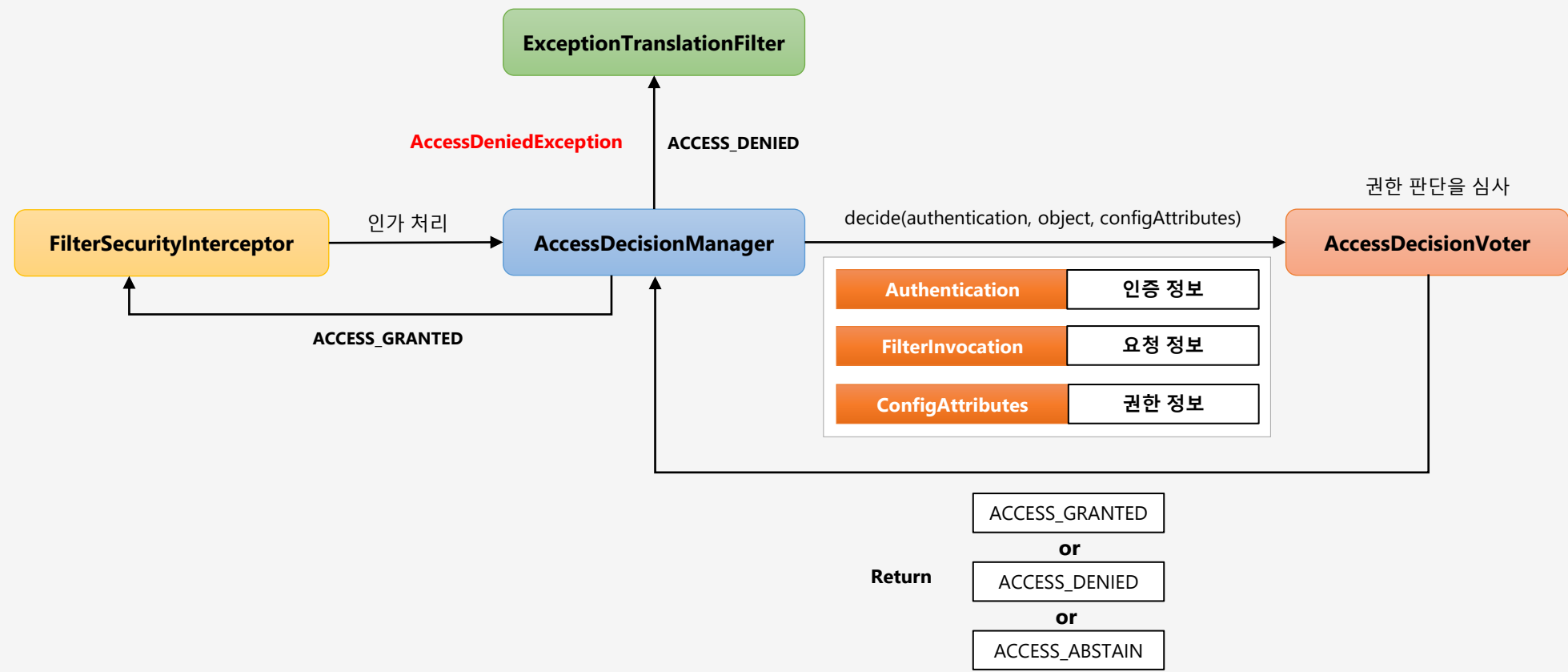


III. 스프링 시큐리티 주요 아키텍처 이해

#09. AccessDecisionVoter

- 판단을 심사하는 것(위원)
- **Voter** 가 권한 부여 과정에서 판단하는 자료
 - **Authentication** - 인증 정보(user)
 - **FilterInvocation** - 요청 정보 (antMatcher("/user"))
 - **ConfigAttributes** - 권한 정보 (hasRole("USER"))
- 결정 방식
 - **ACCESS_GRANTED** : 접근허용(1)
 - **ACCESS_DENIED** : 접근 거부(0)
 - **ACCESS_ABSTAIN** : 접근 보류(-1)
 - Voter 가 해당 타입의 요청에 대해 결정을 내릴 수 없는 경우

#09. AccessDecisionManager, AccessDecisionVoter



Core Spring Security

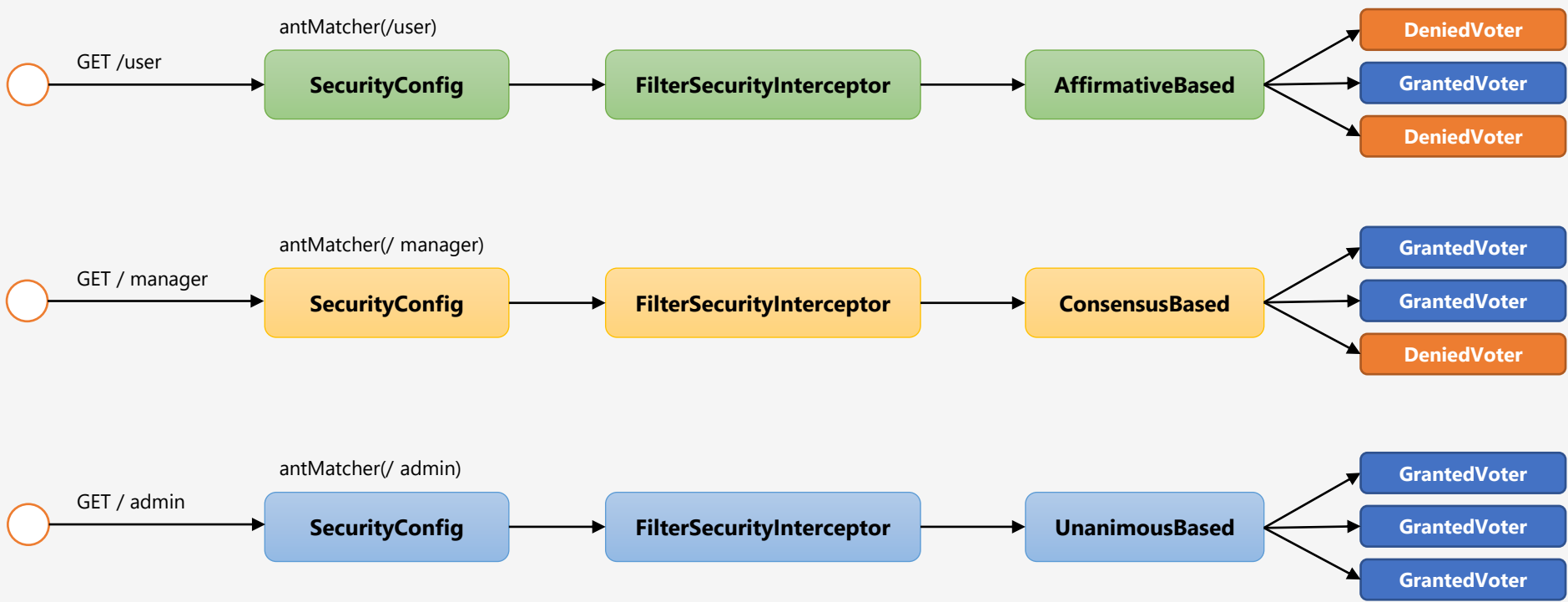


핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#10. AccessDecisionManager, AccessDecisionVoter 응용편

#10. AccessDecisionManager, AccessDecisionVoter 응용편



- 고려 사항
 - 초기화 시 생성된 특정 필터를 참조하는 방법

Core Spring Security

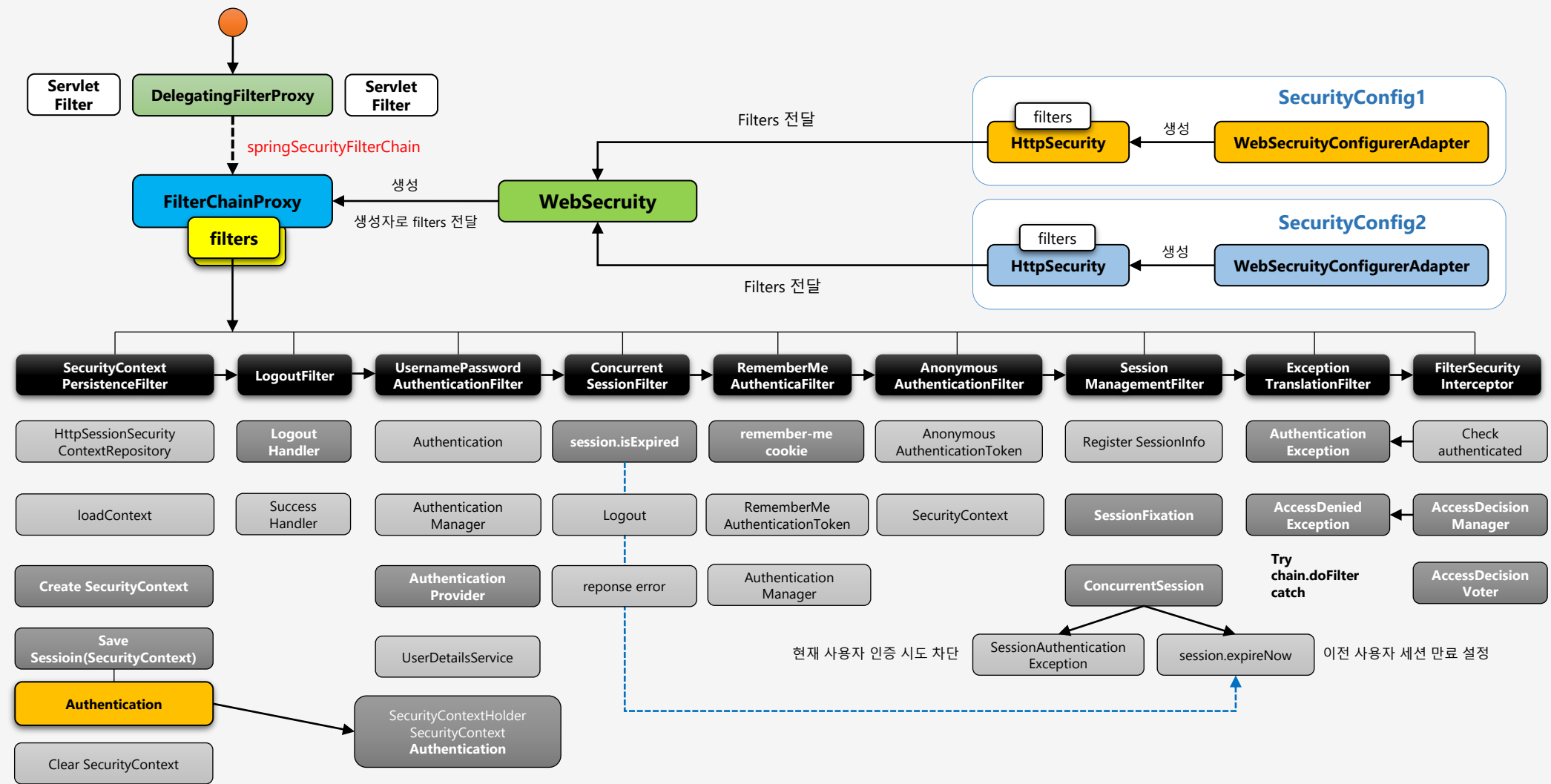
핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



Ⅲ. 스프링 시큐리티 주요 아키텍처 이해

#11. 스프링 시큐리티 필터 및 아키텍처 정리

III. 스프링 시큐리티 주요 아키텍처 이해



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#01. 실전 프로젝트 구성

II. 스프링 시큐리티 기본 API & Filter 이해

#01. 실전 프로젝트 구성

1. 프로젝트 명 : **core-spring-security**
2. 기본 의존관계 설정 - **pom.xml**
3. 기본 패키지 및 폴더 구성
4. 기본 View Template 생성
5. 기본 정적 자원 설정

IV. 인증(Authentication) 프로세스 구현

#02. 시큐리티 의존성 추가 및 기본 환경설정

- pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

IV. 인증(Authentication) 프로세스 구현

#03. 시큐리티 의존성 추가 및 기본 환경설정

- **application.properties**

```
spring.datasource.url=jdbc:postgresql://localhost:5432/springboot
spring.datasource.username=postgres
spring.datasource.password=pass

spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true

spring.thymeleaf.cache=false

spring.devtools.livereload.enabled=true
spring.devtools.restart.enabled=true
```

IV. 인증(Authentication) 프로세스 구현

#03. SecurityConfig 설정하기

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

 http

 .authorizeRequests()

 .anyRequest().authenticated()

 .and()

 .formLogin();

 }

}

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#02. 메뉴 권한 및 WebIgnore 설정

IV. 인증(Authentication) 프로세스 구현

#02. WebIgnore 설정

- js / css / image 파일 등 보안 필터를 적용할 필요가 없는 리소스를 설정

@Override

```
public void configure(WebSecurity web) throws Exception {
```

```
    web.ignoring().requestMatchers(PathRequest.toStaticResources().atCommonLocations());
```

```
}
```


Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#04. Form 인증 – User 등록 / PasswordEncoder

IV. 인증(Authentication) 프로세스 구현

#04. Form 인증 - PasswordEncoder

- Spring Security 5.0 이전에는 기본 PasswordEncoder 가 평문을 지원하는 NoOpPasswordEncoder
- 암호화 포맷 : {id}encodedPassword
 - bcrypt, noop, pbkdf2, scrypt, sha256 (기본 포맷은 Bcrypt : {bcrypt}\$2a\$10\$dXJ3SW6G7P50IGmMkkmwe.20cQQubK3.HZWzG3YB1tIRy.fqvM/BG)
- 생성
 - PasswordEncoder passwordEncoder = PasswordEncoderFactories.createDelegatingPasswordEncoder()
- 인터페이스
 - encode(password)
 - 패스워드 암호화
 - matches(rawPassword, encodedPassword)
 - 패스워드 비교

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

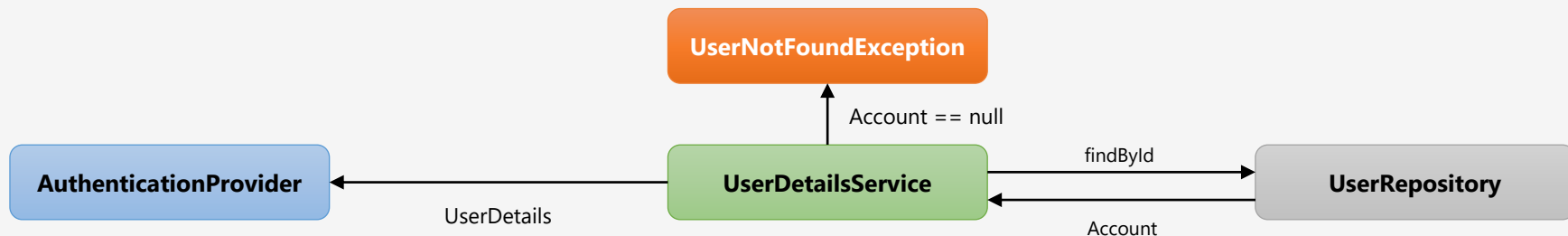


IV. 인증(Authentication) 프로세스 구현

#06. Form 인증 – CustomUserService

IV. 인증(Authentication) 프로세스 구현

#05. CustomUserService



```
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
    Account account = userRepository.findByUsername(username);  
  
    if (account == null) {  
        throw new UsernameNotFoundException("No user found with username: " + username);  
    }  
  
    // 권한  
    ArrayList<GrantedAuthority> roles = new ArrayList<GrantedAuthority>();  
    roles.add(new SimpleGrantedAuthority(account.getRole()));  
  
    return new AccountContext(account, roles);  
}
```

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

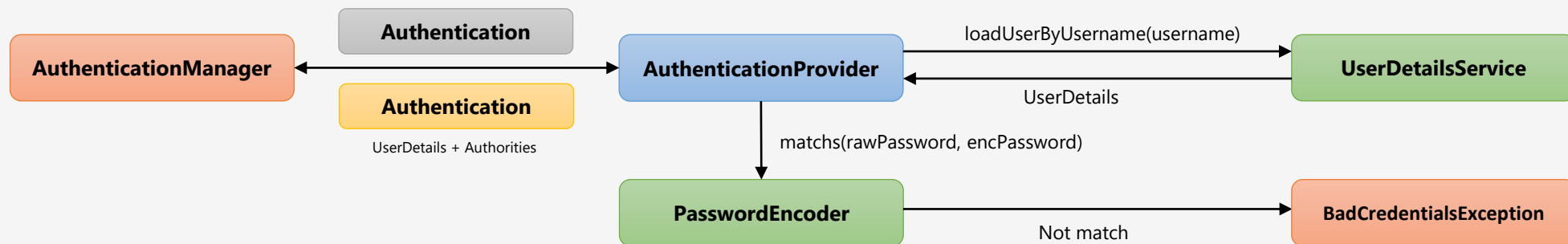


IV. 인증(Authentication) 프로세스 구현

#06. Form 인증 – CustomAuthenticationProvider

IV. 인증(Authentication) 프로세스 구현

#06. CustomAuthenticationProvider



```
public Authentication authenticate(Authentication auth) throws AuthenticationException {
```

```
    String loginId = auth.getName();
    String passwd = (String) auth.getCredentials();
```

```
    UserDetails userDetails = userDetailsService.loadUserByUsername(loginId);
```

```
    if (userDetails == null || !passwordEncoder.matches(passwd, userDetails.getPassword())) {
        throw new BadCredentialsException("Invalid password");
    }
```

```
    return new UsernamePasswordAuthenticationToken(userDetails.getUser(), null, userDetails.getAuthorities());
```

```
}
```

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

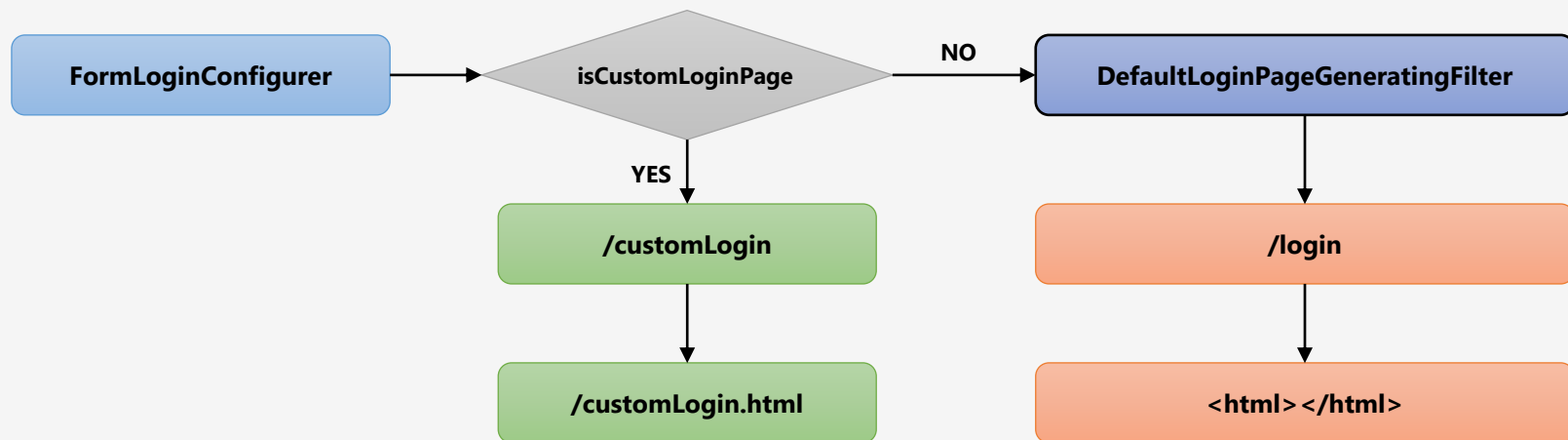


IV. 인증(Authentication) 프로세스 구현

#07. Form 인증 – Custom Login Form Page

IV. 인증(Authentication) 프로세스 구현

#07. Form 인증 - Custom Login Form Page



@Override

```
public void configure(HttpSecurity http) throws Exception {  
    http.formLogin().loginPage("/customLogin")  
}
```


Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#08. Form 인증 - 로그아웃 및 화면 보안 처리

IV. 인증(Authentication) 프로세스 구현

#08. Form 인증 - 로그아웃 및 화면 보안 처리

- 로그아웃 방법
 - <form> 태그를 사용해서 POST로 요청
 - <a> 태그를 사용해서 GET 으로 요청 – **SecurityContextLogoutHandler** 활용
- 인증 여부에 따라 로그인/로그아웃 표현
 - <li sec:authorize="isAnonymous()"><a th:href="@{/login}">로그인
 - <li sec:authorize="isAuthenticated()"><a th:href="@{/logout}">로그아웃

```
@GetMapping(value = "/logout")
public String logout(HttpServletRequest request, HttpServletResponse response) {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    if (auth != null) {
        new SecurityContextLogoutHandler().logout(request, response, auth);
    }
    return "redirect:/login";
}
```

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

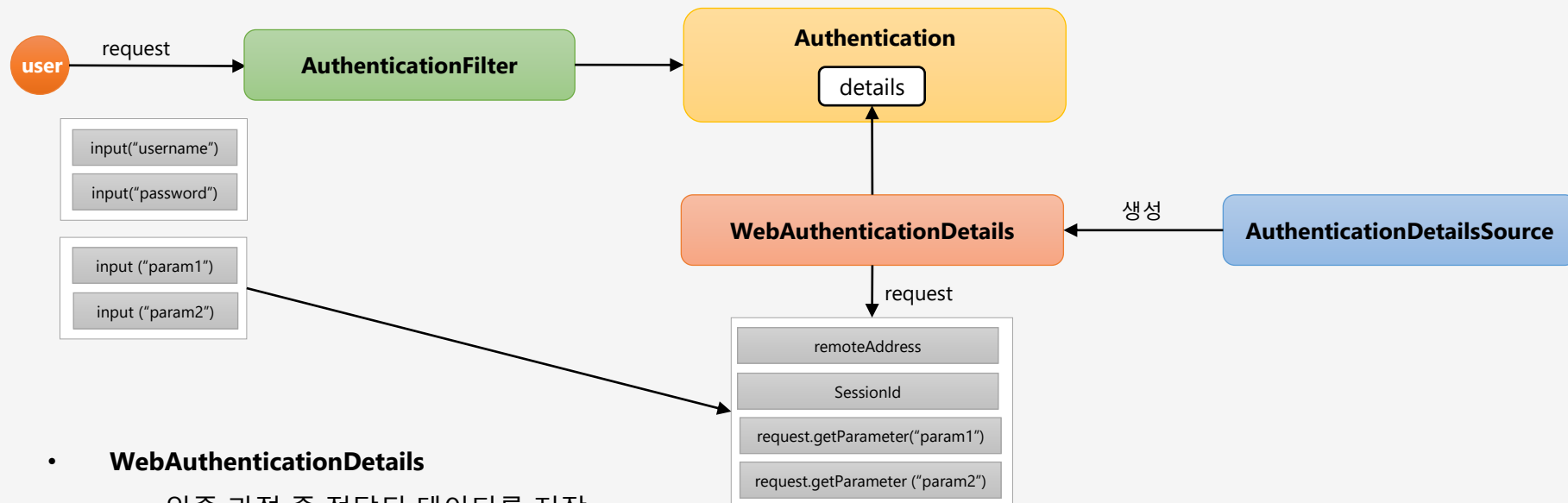


IV. 인증(Authentication) 프로세스 구현

#08. Form 인증 – WebAuthenticationDetails AuthenticationDetailsSource

IV. 인증(Authentication) 프로세스 구현

#08. Form 인증 – WebAuthenticationDetails, AuthenticationDetailsSource



- **WebAuthenticationDetails**
 - 인증 과정 중 전달된 데이터를 저장
 - Authentication의 details 속성에 저장
- **AuthenticationDetailsSource**
 - WebAuthenticationDetails 객체를 생성

Core Spring Security



핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

IV. 인증(Authentication) 프로세스 구현

#09. Form 인증 – CustomAuthenticationSuccessHandler

IV. 인증(Authentication) 프로세스 구현

#09. Form 인증 - CustomAuthenticationSuccessHandler

SecurityConfig

```
@Override
public void configure(HttpSecurity http) throws Exception {
    http.formLogin().successHandler(CustomAuthenticationSuccessHandler())
}
```

CustomAuthentication SuccessHandler

```
@Override
public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response,
Authentication authentication) throws IOException {
    RequestCache requestCache = new HttpSessionRequestCache(); // 요청 캐시와 관련된 작업
    final HttpSession session = request.getSession(false); // 세션 관련 작업
    Object principal = authentication.getPrincipal() // 인증된 사용자 관련작업
    redirectStrategy.sendRedirect(request, response, targetUrl); // 인증 성공 후 이동
}
}
```

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

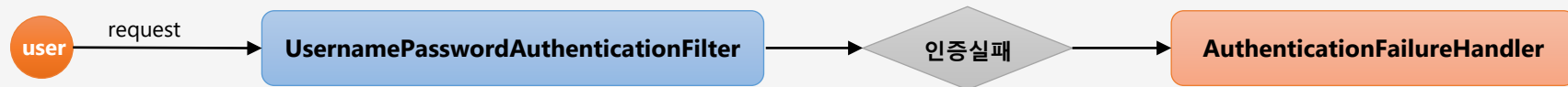


IV. 인증(Authentication) 프로세스 구현

#09. Form 인증 –CustomAuthenticationFailureHandler

IV. 인증(Authentication) 프로세스 구현

#10. Form 인증 - CustomAuthenticationFailureHandler



SecurityConfig

```
@Override
public void configure(HttpSecurity http) throws Exception {
    http.formLogin().failureHandler(CustomAuthenticationFailureHandler())
}
}
```

CustomAuthentication
FailureHandler

```
@Override
public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response,
AuthenticationException exception) throws IOException {
    if (exception instanceof UsernameNotFoundException) {
        errorMessage = messages.getMessage("사용자가 존재하지 않습니다. ", null, locale);
    } else if (exception instanceof BadCredentialsException) {
        errorMessage = messages.getMessage("아이디 혹은 비밀번호가 일치하지 않습니다.", null, locale);
    } else {
        errorMessage = "인증에 실패했습니다. 웹 마스터에게 문의하십시오. ! ";
    }
}
}
```


Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#11. Form 인증 - Access Denied

IV. 인증(Authentication) 프로세스 구현

#11. Form 인증 - Access Denied

SecurityConfig

```
@Override
public void configure(HttpSecurity http) throws Exception {
    http.exceptionHandling().accessDeniedPage("/accessDenied")
        .accessDeniedHandler(accessDeniedHandler)
}
```

AccessDeniedHandler

```
@Override
public void handle(HttpServletRequest request, HttpServletResponse response, AccessDeniedException
accessDeniedException) throws IOException, ServletException {

    String deniedUrl = errorPage + "?exception=" + accessDeniedException.getMessage();
    response.sendRedirect(request, response, deniedUrl);
}

public void setErrorPage(String errorPage) {
    this.errorPage = errorPage;
}
```

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#13. Form 인증 – 인증 사용자 정보 구하기

IV. 인증(Authentication) 프로세스 구현

#13. Form 인증 – 인증 사용자 정보 구하기

1. 어플리케이션 전역

- Account account = (Account) **SecurityContextHolder.getContext().getAuthentication().getPrincipal()**;
String username = account.getUsername();

2. Spring MVC

- Authentication, Principal**

```
public String getUsername(Authentication authentication, Principal principal)
```

```
Account account = (Account) authentication.getPrincipal();
```

```
Account account = (Account)((UsernamePasswordAuthenticationToken)principal).getPrincipal();
```

```
String username = account.getUsername();
```

- @AuthenticationPrincipal**

- 인증 성공 이후 생성된 Authentication 객체의 principal 속성에 저장되어 있는 객체

```
public ModelAndView getUsername(@AuthenticationPrincipal Account account)
```

```
String username = account.getUsername();
```

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

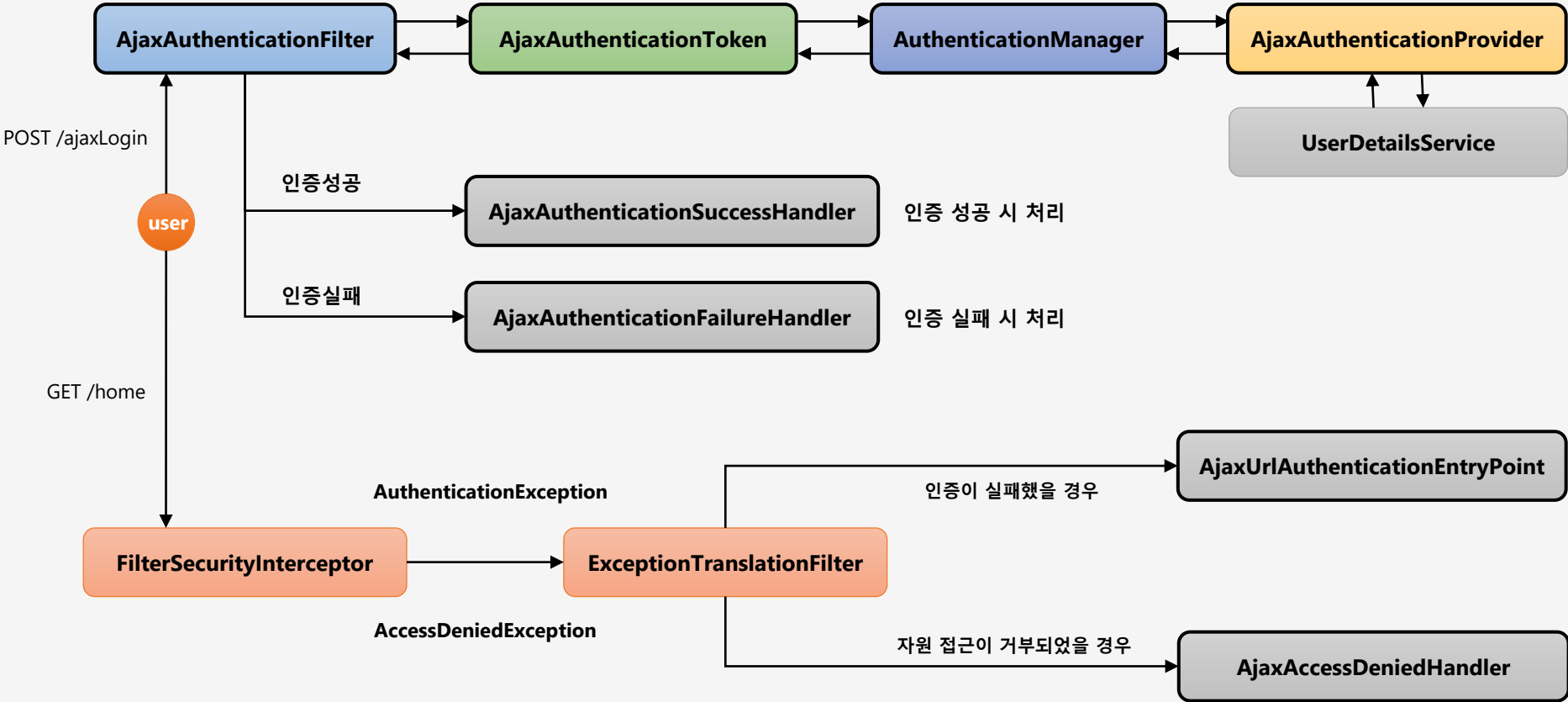


IV. 인증(Authentication) 프로세스 구현

#14. Ajax 인증 – 흐름 및 개요

IV. 인증(Authentication) 프로세스 구현

#14. Ajax 인증 - 흐름 및 개요



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#14. Ajax 인증 – AjaxAuthenticationFilter

IV. 인증(Authentication) 프로세스 구현

#14. Ajax 인증 – AjaxAuthenticationFilter

- **AbstractAuthenticationProcessingFilter** 상속
- **필터 작동 조건**
 - AntPathRequestMatcher("/api/login") 로 요청정보와 매칭하고 요청 방식이 Ajax 이면 필터 작동
- **AjaxAuthenticationToken** 생성하여 **AuthenticationManager** 에게 전달하여 인증처리
- **Filter** 추가
 - `http.addFilterBefore(AjaxAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class)`

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#15. Ajax 인증 – AjaxAuthenticationProvider

IV. 인증(Authentication) 프로세스 구현

#15. Ajax 인증 – AjaxAuthenticationProvider

- **AuthenticationProvider** 인터페이스 구현
- 인증 작동 조건
 - supports(Class<?> authentication)
 - ProviderManager로부터 넘어온 인증객체가 AjaxAuthenticationToken 타입이면 작동
- 인증 검증이 완료되면 **AjaxAuthenticationToken** 생성하여 최종 인증 객체 반환

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#16. Ajax 인증 – AjaxAuthenticationSuccessHandler AjaxAuthenticationFailureHandler

IV. 인증(Authentication) 프로세스 구현

#16. Ajax 인증 – AjaxAuthenticationSuccessHandler, AjaxAuthenticationFailureHandler

- **AjaxAuthenticationSuccessHandler**

- **AuthenticationSuccessHandler** 인터페이스 구현
- **Response Header** 설정
 - `response.setStatus(HttpStatus.OK.value());`
 - `response.setContentType(MediaType.APPLICATION_JSON_VALUE);`
- **JSON** 형식으로 변환하여 인증 객체 리턴 함
 - `objectMapper.writeValue(response.getWriter(), ResponseBody.ok(userDto));`

- **AjaxAuthenticationFailureHandler**

- **AuthenticationFailureHandler** 인터페이스 구현
- **Response Header** 설정
 - `response.setStatus(HttpStatus.UNAUTHORIZED.value());`
 - `response.setContentType(MediaType.APPLICATION_JSON_VALUE);`
- **JSON** 형식으로 변환하여 오류 메시지 리턴 함
 - `objectMapper.writeValue(response.getWriter(), ResponseBody.error(message));`

Core Spring Security



핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

IV. 인증(Authentication) 프로세스 구현

#17. Ajax 인증 – AjaxLoginUrlAuthenticationEntryPoint AjaxAccessDeniedHandler

Lecturer 정수원
<https://github.com/onjsdnjs>

IV. 인증(Authentication) 프로세스 구현

#17. Ajax 인증 – AjaxLoginUrlAuthenticationEntryPoint, AjaxAccessDeniedHandler

- **AjaxLoginUrlAuthenticationEntryPoint**

- `ExceptionHandlerFilter` 에서 인증 예외 시 호출
- `AuthenticationEntryPoint` 인터페이스 구현
- 인증 오류 메시지와 401 상태 코드 반환
 - `response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized");`

- **AjaxAccessDeniedHandler**

- `ExceptionHandlerFilter` 에서 인가 예외 시 호출
- `AccessDeniedHandler` 인터페이스 구현
- 인가 오류 메시지와 403 상태 코드 반환
 - `response.sendError(HttpServletResponse.SC_FORBIDDEN, "forbidden");`

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#18. Ajax 인증 – DSL 로 Config 설정하기

IV. 인증(Authentication) 프로세스 구현

#18. Ajax 인증 – DSL 로 Config 설정하기

- **Custom DSLs**
 - **AbstractHttpConfigurer**
 - 스프링 시큐리티 초기화 설정 클래스
 - 필터, 핸들러, 메서드, 속성 등을 한 곳에 정의하여 처리할 수 있는 편리함 제공
 - **public void init(H http) throws Exception** - 초기화
 - **public void configure(H http)** – 설정
 - **HttpSecurity** 의 **apply(C configurer)** 메서드 사용

IV. 인증(Authentication) 프로세스 구현

#18. Ajax 인증 – DSL 로 Config 설정하기

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

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .exceptionHandling()
            .authenticationEntryPoint(new AjaxLoginUrlAuthenticationEntryPoint())
            .and()
            .apply(new AjaxLoginConfigurer<>())
            .successHandlerAjax ajaxAuthenticationSuccessHandler)
            .failureHandlerAjax ajaxAuthenticationFailureHandler)
            .loginProcessingUrl(/ajaxLogin)
            .setAuthenticationManager ajaxAuthenticationManager())
            .readAndWriteMapper(objectMapper);
    }
}
```

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



IV. 인증(Authentication) 프로세스 구현

#19. Ajax 인증 - 로그인 Ajax 구현 & CSRF

IV. 인증(Authentication) 프로세스 구현

#19. Ajax 인증 – 로그인 Ajax 구현 & CSRF

- **헤더 설정**

- 전송 방식이 Ajax 인지의 여부를 위한 헤더설정

- `xhr.setRequestHeader("X-Requested-With", "XMLHttpRequest");`

- **CSRF 헤더 설정**

- `<meta id="_csrf" name="_csrf" th:content="${_csrf.token}"/>`
 - `<meta id="_csrf_header" name="_csrf_header" th:content="${_csrf.headerName}"/>`
 - `var csrfHeader = $('meta[name="_csrf_header"]').attr('content')`
 - `var csrfToken = $('meta[name="_csrf"]').attr('content')`
 - `xhr.setRequestHeader(csrfHeader, csrfToken);`

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



V. 인가(Authorization) 프로세스 구현 - DB 연동

#01. 개요

V. 인가(Authorization) 프로세스 구현 - DB 연동

#01. 개요

- DB와 연동하여 자원 및 권한을 설정하고 제어함으로 동적 권한 관리가 가능하도록 한다
- 설정 클래스 소스에서 권한 관련 코드 모두 제거
 - ex) `antMatcher("/user").hasRole("USER ")`
- 관리자 시스템 구축
 - 회원 관리 - 권한 부여
 - 권한 관리 - 권한 생성, 삭제
 - 자원 관리 - 자원 생성, 삭제, 수정, 권한 매핑
- 권한 계층 구현
 - **URL** - Url 요청 시 인가 처리
 - **Method** - 메소드 호출 시 인가 처리
 - Method
 - Pointcut

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



V. 인가(Authorization) 프로세스 구현 - DB 연동

#02. Url 방식 - 주요 아키텍처 이해

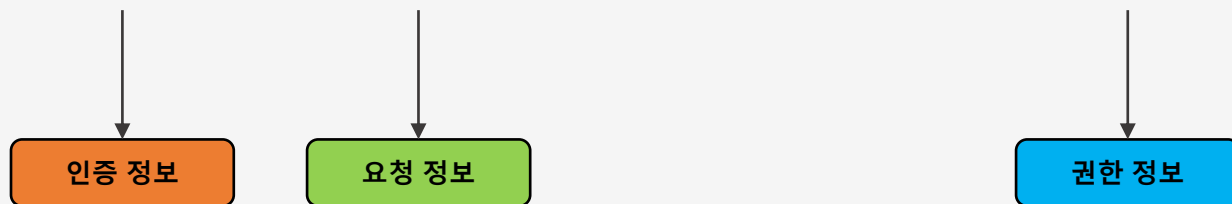
V. 인가(Authorization) 프로세스 구현 - DB 연동

#03. 주요 아키텍처 이해

- 스프링 시큐리티의 인가처리

`http.antMatchers("/user").access("hasRole('USER')")`

사용자가 `/user` 자원에 접근하기 위해서는 `ROLE_USER` 권한이 필요하다



V. 인가(Authorization) 프로세스 구현 - DB 연동

#03. 주요 아키텍처 이해

Authentication

user

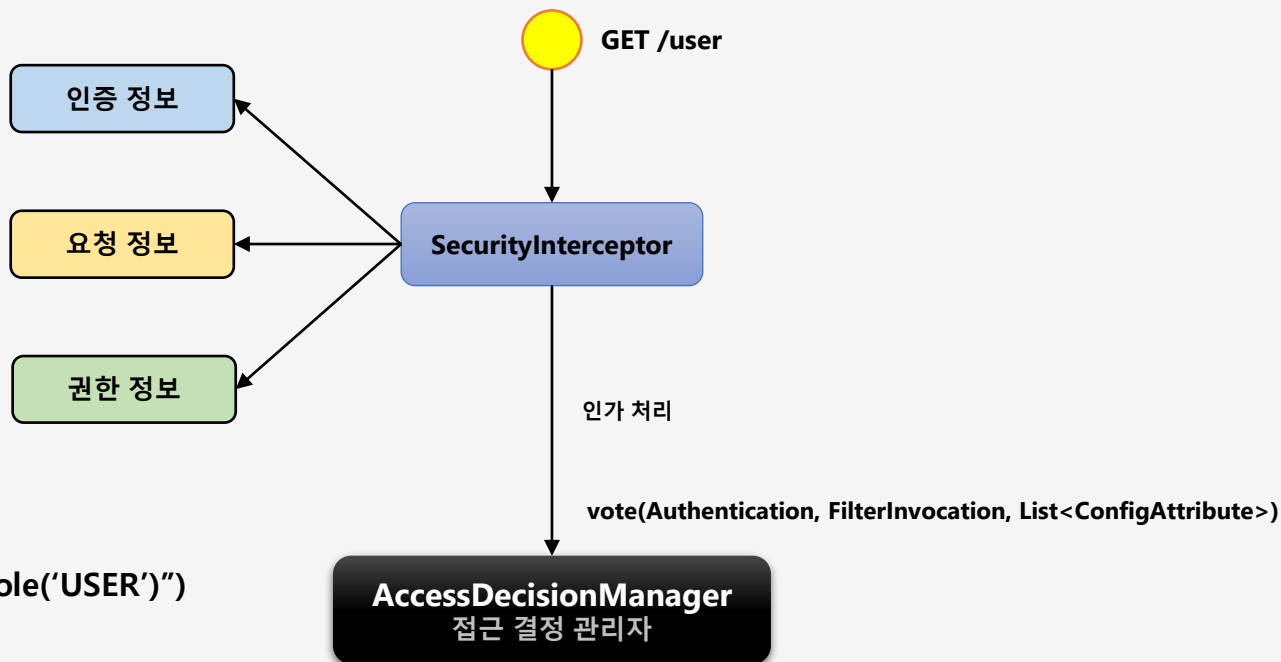
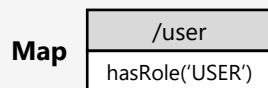
FilterInvocation

request(/user)

List<ConfigAttribute>

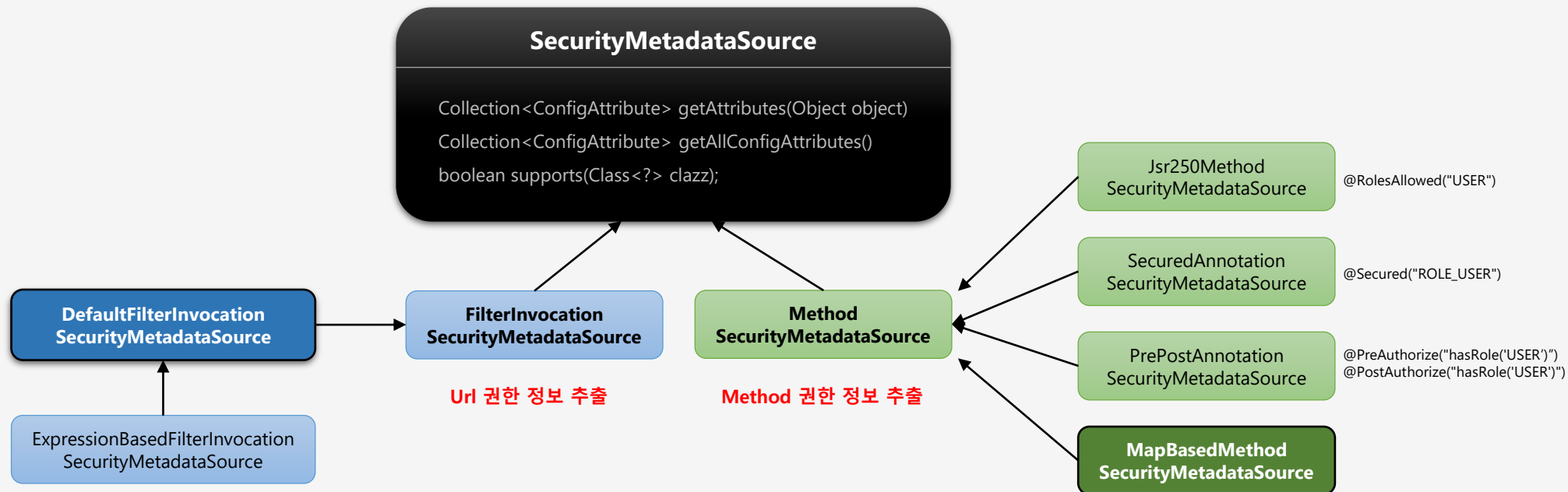
"hasRole('USER')"

`http.antMatchers("/user").access("hasRole('USER')")`



V. 인가(Authorization) 프로세스 구현 - DB 연동

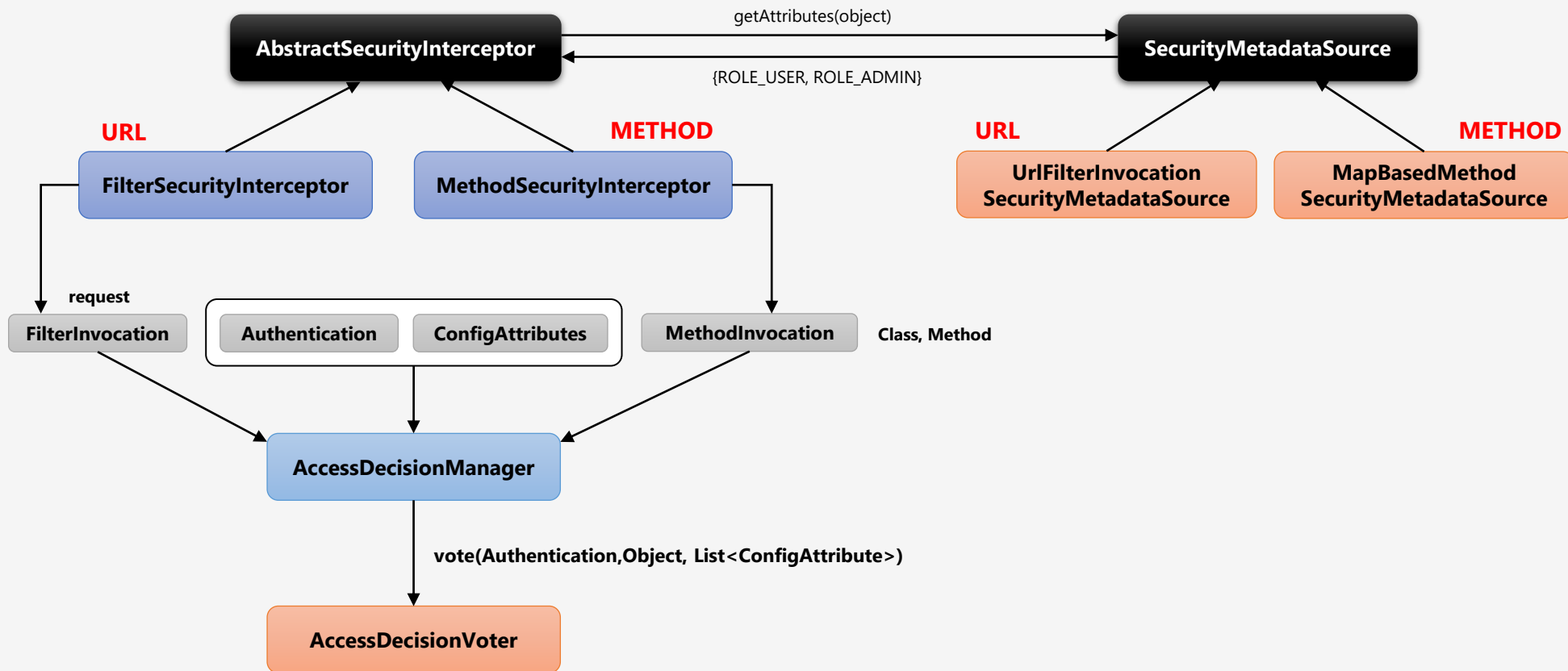
#02. 주요 아키텍처 이해



❖ 자원에 설정된 권한정보를 추출하도록 구현

V. 인가(Authorization) 프로세스 구현 - DB 연동

#02.주요 아키텍처 이해



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

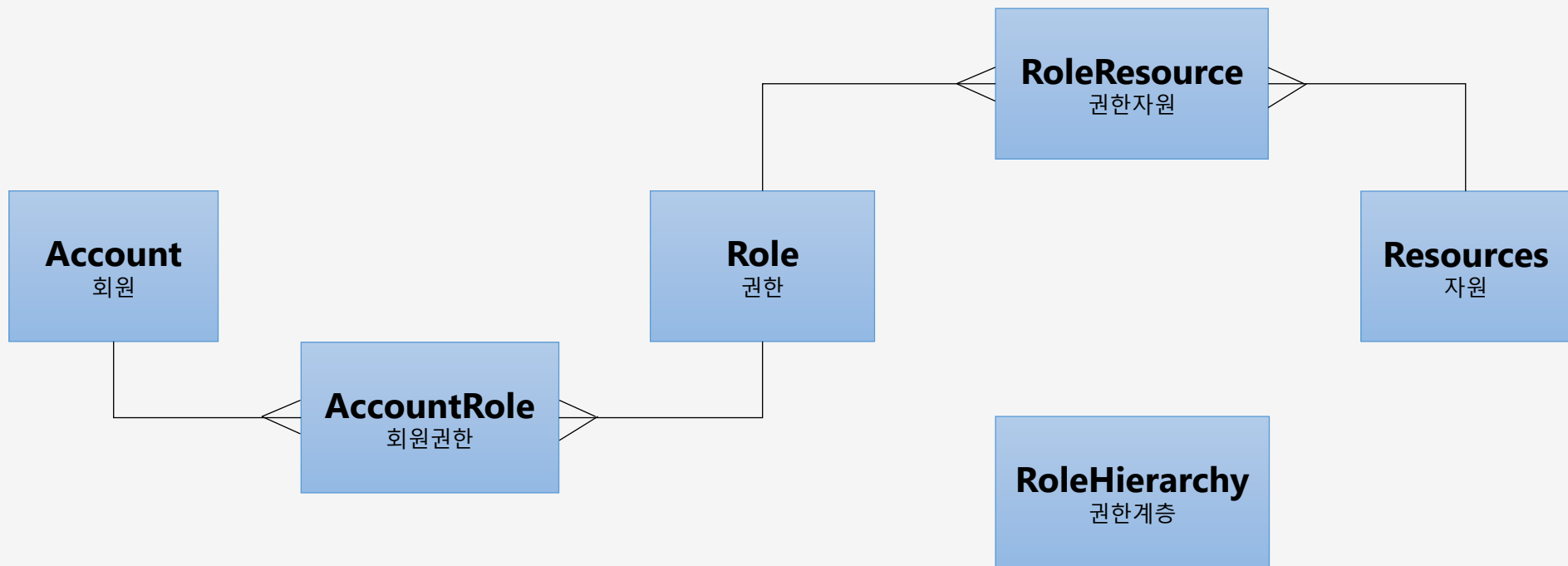


V. 인가(Authorization) 프로세스 구현 - DB 연동

#03. Url 방식 - 관리자 시스템 구성

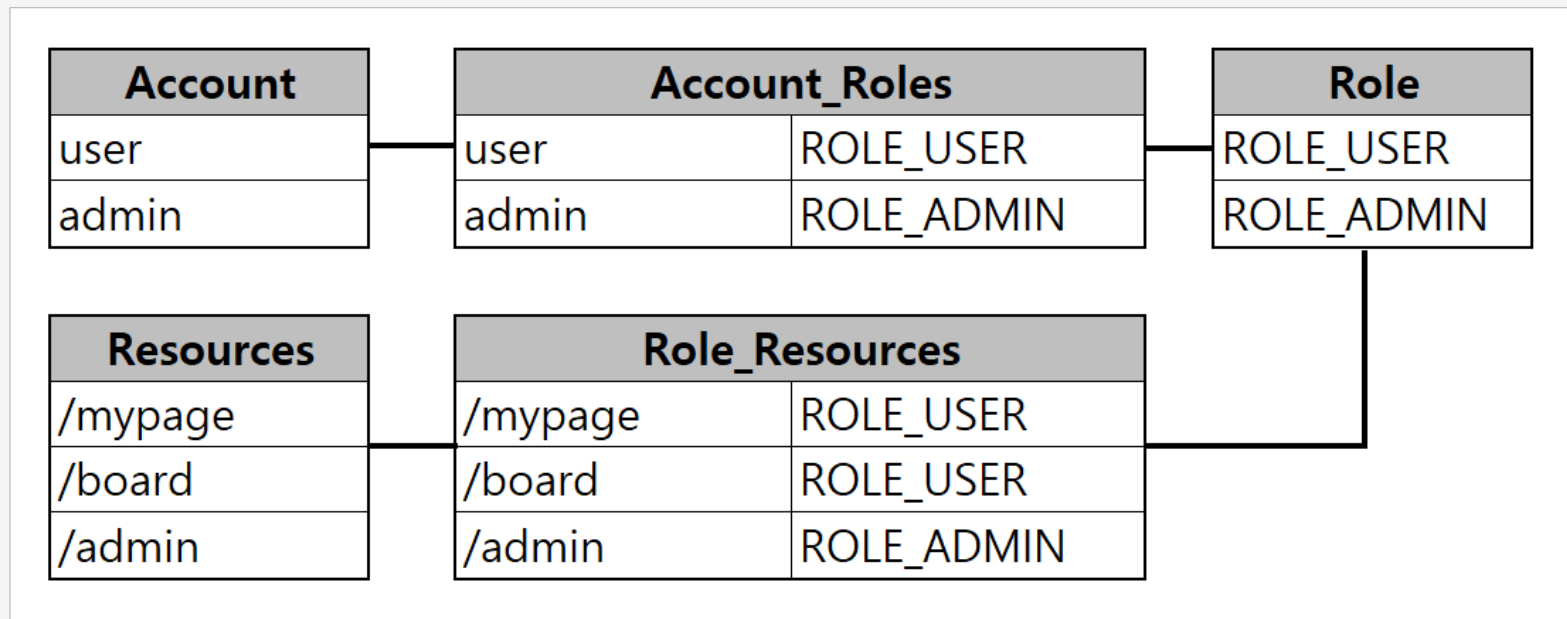
V. 인가(Authorization) 프로세스 구현 - DB 연동

#03. Url 방식 - 도메인 관계도



V. 인가(Authorization) 프로세스 구현 - DB 연동

#03. Url 방식 - 테이블 관계도



Core Spring Security

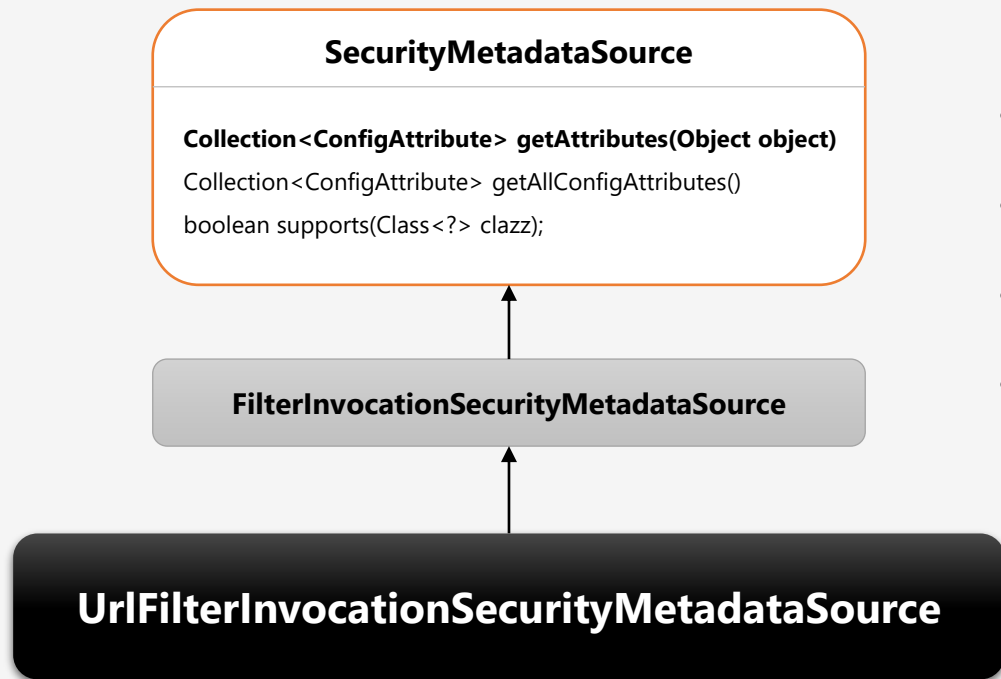
핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



V. 인가(Authorization) 프로세스 구현 - DB 연동

#04. Url 방식 – FilterInvocationSecurityMetadataSource

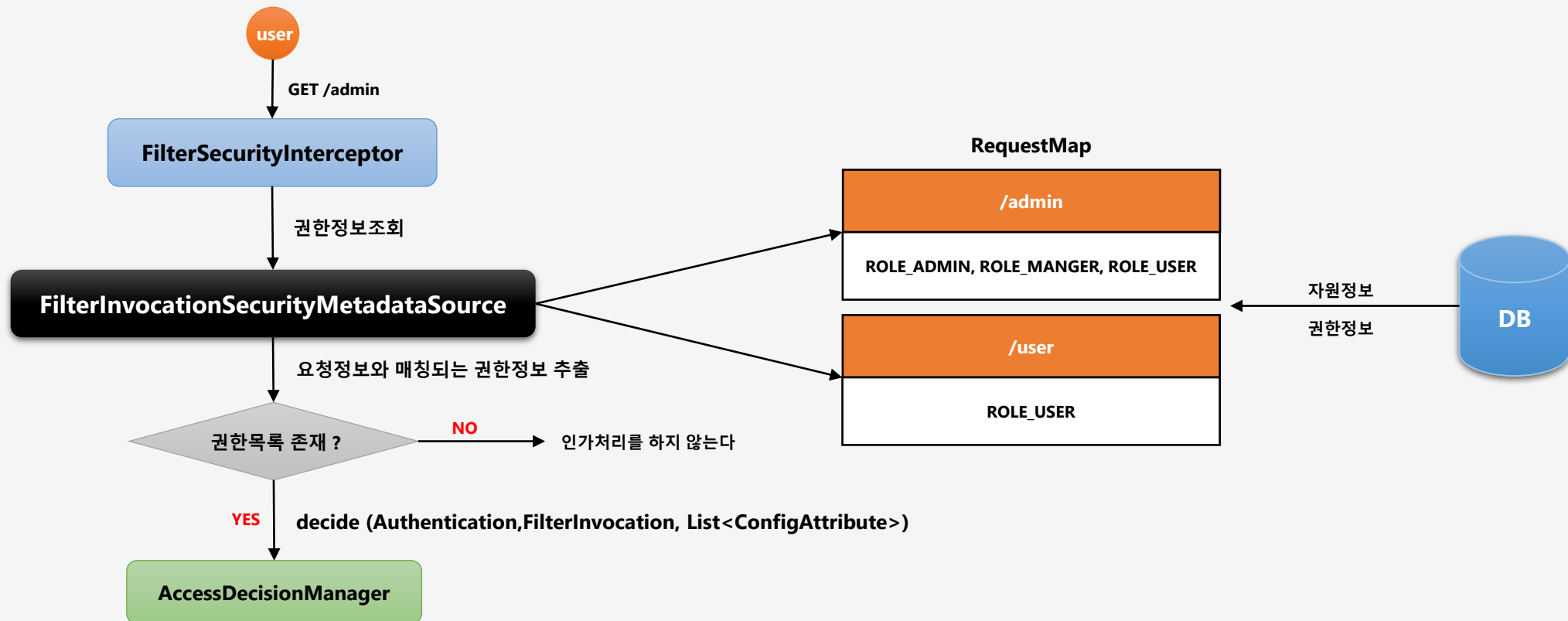
#04. FilterInvocationSecurityMetadataSource



- 사용자가 접근하고자 하는 Url 자원에 대한 권한 정보 추출
- AccessDecisionManager 에게 전달하여 인가처리 수행
- DB 로부터 자원 및 권한 정보를 매핑하여 맵으로 관리
- 사용자의 매 요청마다 요청정보에 매핑된 권한 정보 확인

V. 인가(Authorization) 프로세스 구현 - DB 연동

#04. FilterInvocationSecurityMetadataSource



V. 인가(Authorization) 프로세스 구현 - DB 연동

#02. Url 방식 - FilterInvocationSecurityMetadataSource

http.addFilterAt(filterSecurityInterceptor(), FilterSecurityInterceptor.class)

@Bean

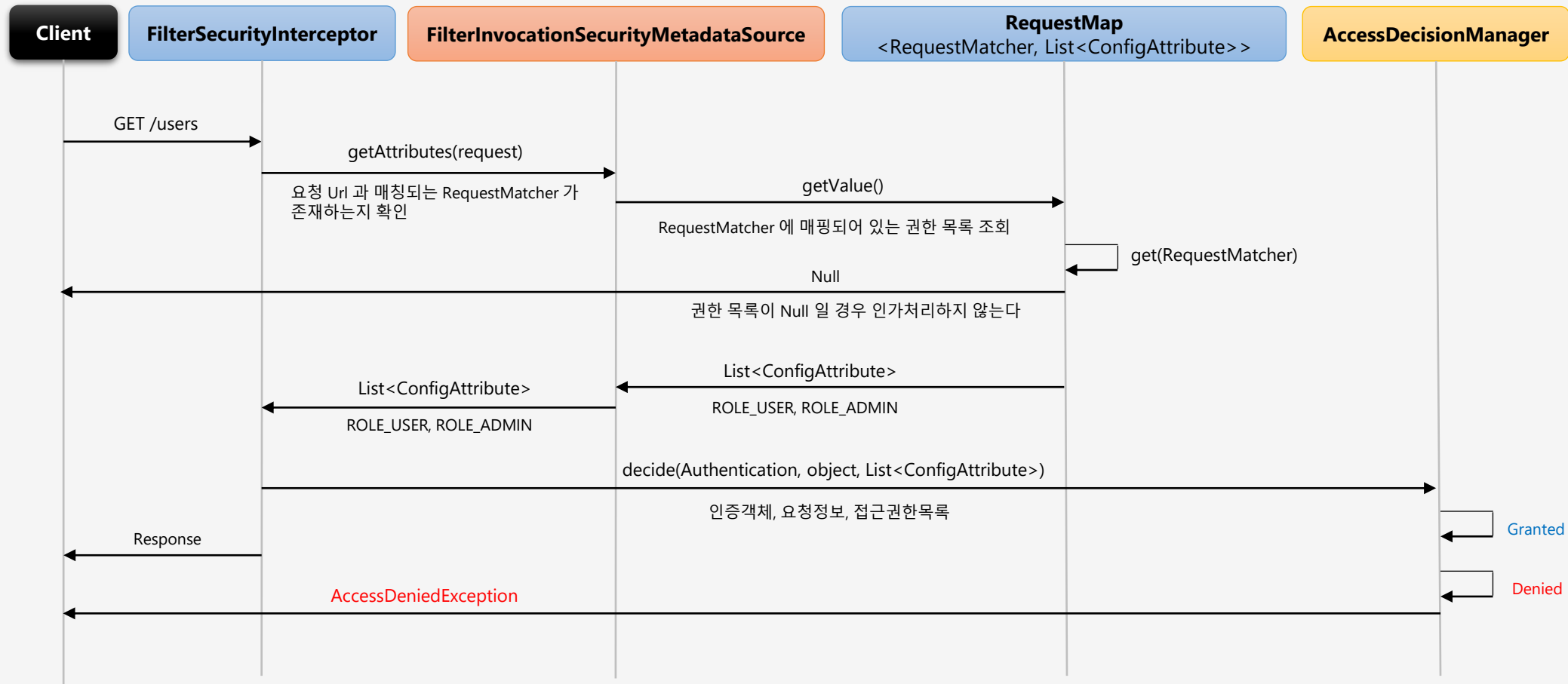
```
public FilterSecurityInterceptor filterSecurityInterceptor() {  
    FilterSecurityInterceptor filterSecurityInterceptor = new FilterSecurityInterceptor();  
    filterSecurityInterceptor.setAuthenticationManager(authenticationManager);  
    filterSecurityInterceptor.setSecurityMetadataSource(urlFilterInvocationSecurityMetadataSource());  
    filterSecurityInterceptor.setAccessDecisionManager(accessDecisionManager);  
    return filterSecurityInterceptor;  
}
```

@Bean

```
public FilterInvocationSecurityMetadataSource urlFilterInvocationSecurityMetadataSource() {  
    return new UrlFilterInvocationSecurityMetadataSource();  
}
```

V. 인가(Authorization) 프로세스 구현 - DB 연동

#04. Url 방식 - FilterInvocationSecurityMetadataSource



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

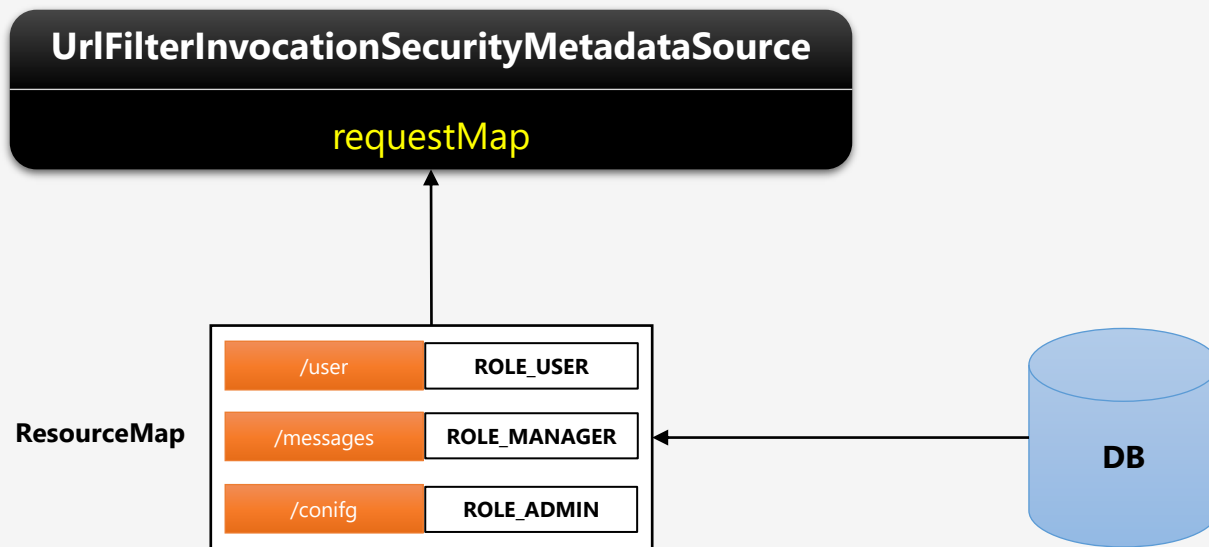


V. 인가(Authorization) 프로세스 구현 - DB 연동

#05. Url 방식 – Map 기반 DB 연동

V. 인가(Authorization) 프로세스 구현 - DB 연동

#05. Url 방식 - Map 기반 DB 연동



- **UrlResourceMapFactoryBean**

- DB로 부터 얻은 권한/자원 정보를 ResourceMap 을 빈으로 생성해서 UrlFilterInvocationSecurityMetadataSource 에 전달

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

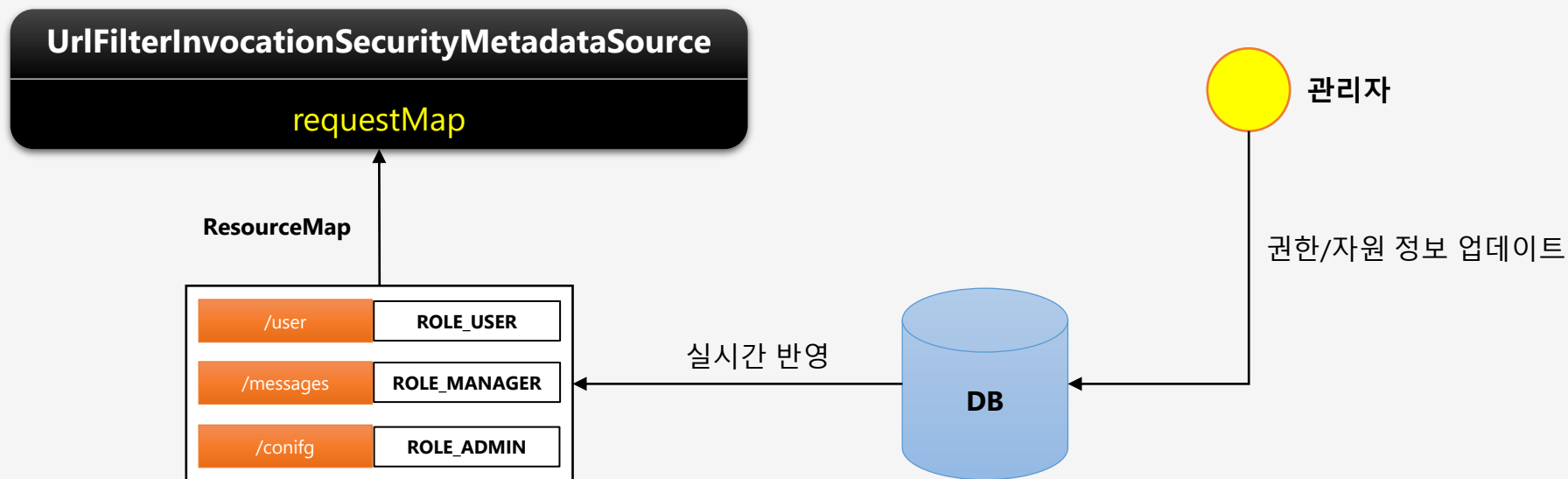


V. 인가(Authorization) 프로세스 구현 - DB 연동

#06. Url 방식 - 인가처리 실시간 반영하기

V. 인가(Authorization) 프로세스 구현 - DB 연동

#06. Url 방식 - 인가처리 실시간 반영하기



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



V. 인가(Authorization) 프로세스 구현 - DB 연동

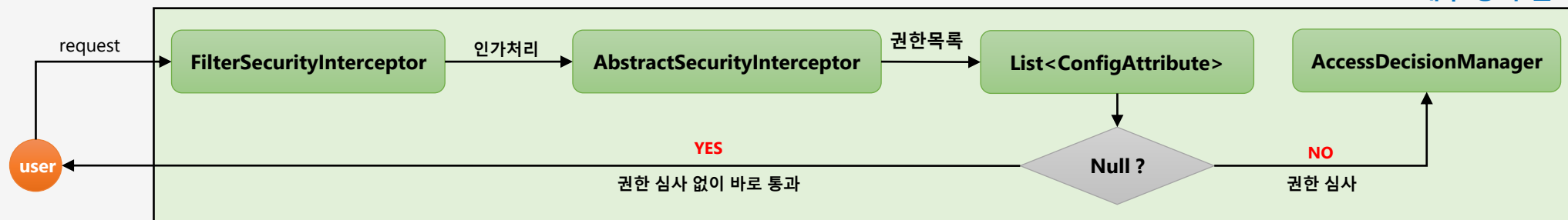
#05. Url 방식 - PermitAllFilter 구현

V. 인가(Authorization) 프로세스 구현 - DB 연동

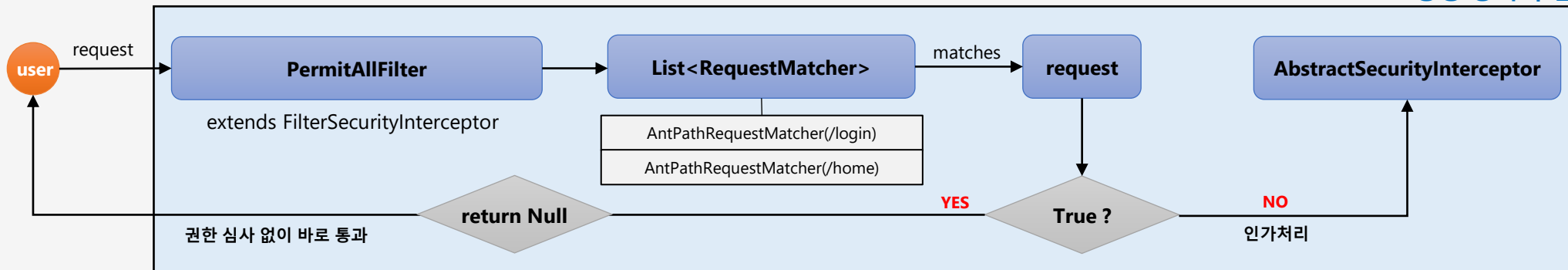
#07. Url 방식 - PermitAllFilter 구현

- 인증 및 권한심사를 할 필요가 없는 자원(/, /home, /login ..)들을 미리 설정해서 바로 리소스 접근이 가능하게 하는 필터

내부 동작 원리



응용 동작 구현



#05. Url 방식 - PermitAllFilter

http.addFilterBefore(permitAllFilter(), FilterSecurityInterceptor.class)

@Bean

```
public PermitAllFilter permitAllFilter () {  
    String[] permitAllPattern = [/,/index,/home,/login,/errorpage/**];  
    PermitAllFilter permitAllFilter = new PermitAllFilter (permitAllPattern);  
    permitAllFilter.setAccessDecisionManager(accessDecisionManager);  
    permitAllFilter.setSecurityMetadataSource(filterInvocationSecurityMetadataSource);  
    permitAllFilter.setRejectPublicInvocations(false);  
    return commonFilterSecurityInterceptor;  
}
```

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

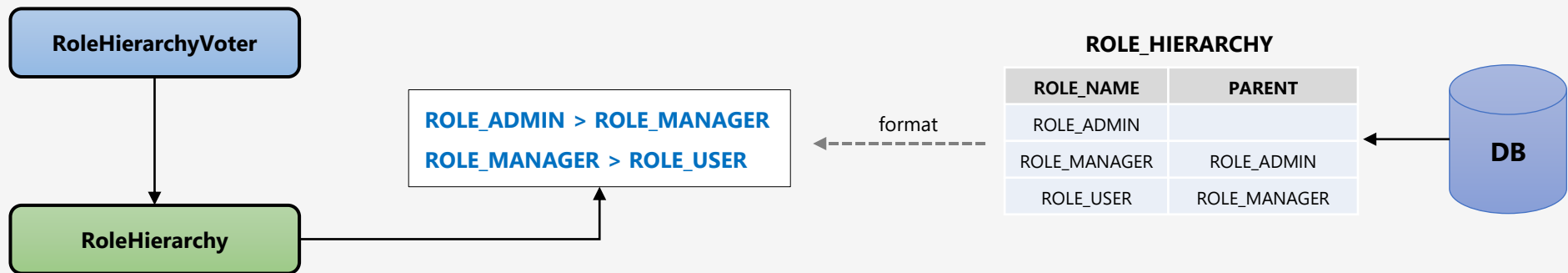


V. 인가(Authorization) 프로세스 구현 - DB 연동

#06. Url 방식 - 계층 권한 적용하기

V. 인가(Authorization) 프로세스 구현 - DB 연동

#08. Url 방식 - 계층 권한 적용하기



- **RoleHierarchy**

- 상위 계층 Role은 하위 계층 Role의 자원에 접근 가능함
- ROLE_ADMIN > ROLE_MANAGER > ROLE_USER 일 경우 ROLE_ADMIN 만 있으면 하위 ROLE 의 권한을 모두 포함한다

- **RoleHierarchyVoter**

- RoleHierarchy 를 생성자로 받으며 이 클래스에서 설정한 규칙이 적용되어 심사함

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

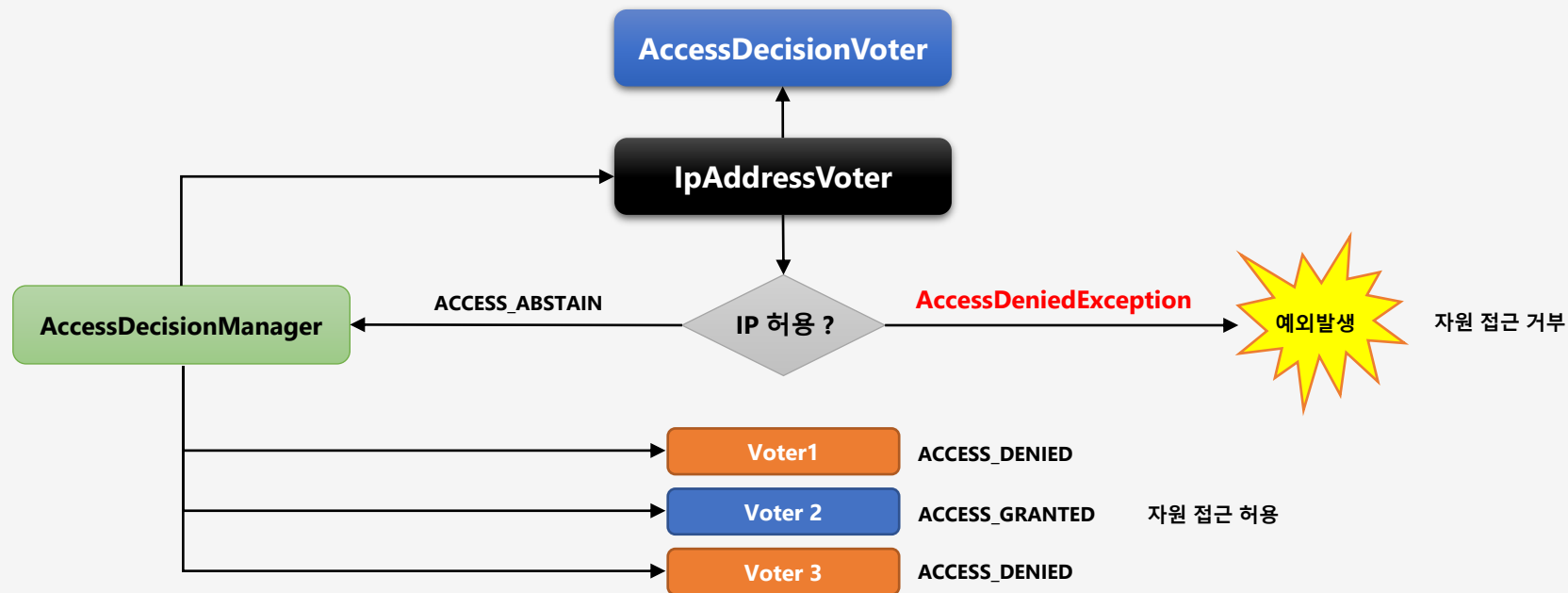


V. 인가(Authorization) 프로세스 구현 - DB 연동

#07. Url 방식 – 아이피 접속 제한하기

V. 인가(Authorization) 프로세스 구현 - DB 연동

#07. Url 방식 - 아이피 접속 제한하기



• 심의 기준

- 특정한 IP 만 접근이 가능하도록 심의하는 Voter 추가
- Voter 중에서 가장 먼저 심사하도록 하여 허용된 IP 일 경우에만 최종 승인 및 거부 결정을 하도록 한다
- 허용된 IP 이면 ACCESS_GRANTED 가 아닌 ACCESS_ABSTAIN 을 리턴해서 추가 심의를 계속 진행하도록 한다
- 허용된 IP 가 아니면 ACCESS_DENIED 를 리턴하지 않고 즉시 예외 발생하여 최종 자원 접근 거부

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



V. 인가(Authorization) 프로세스 구현 - DB 연동

#08. Method 방식 – 개요

#08. Method 방식 - 개요

- 서비스 계층의 인가처리 방식
 - 화면, 메뉴 단위가 아닌 기능 단위로 인가처리
 - 메소드 처리 전.후로 보안 검사 수행하여 인가처리
- **AOP 기반으로 동작**
 - 프록시와 어드바이스로 메소드 인가처리 수행
- **보안 설정 방식**
 - 어노테이션 권한 설정 방식
 - @PreAuthorize("hasRole('USER')"), @PostAuthorize("hasRole('USER')"), @Secured("ROLE_USER")
 - 맵 기반 권한 설정 방식
 - 맵 기반 방식으로 외부와 연동하여 메소드 보안 설정 구현

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

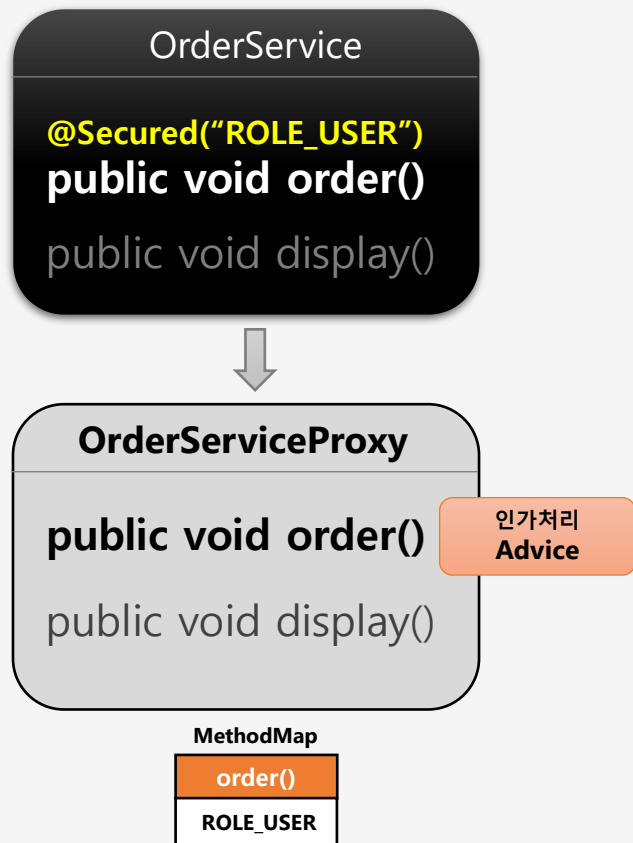


V. 인가(Authorization) 프로세스 구현 - DB 연동

#09. Method 방식 – 주요 아키텍처

V. 인가(Authorization) 프로세스 구현 - DB 연동

#09. Method 방식 - 주요 아키텍처



• 인가 처리를 위한 초기화 과정과 진행

• 초기화 과정

- ① 초기화 시 전체 빈을 검사하면서 보안이 설정된 메소드가 있는지 탐색
- ② 빈의 프록시 객체를 생성
- ③ 보안 메소드에 인가처리(권한심사) 기능을 하는 Advice 를 등록
- ④ 빈 참조시 실제 빈이 아닌 프록시 빈 객체를 참조

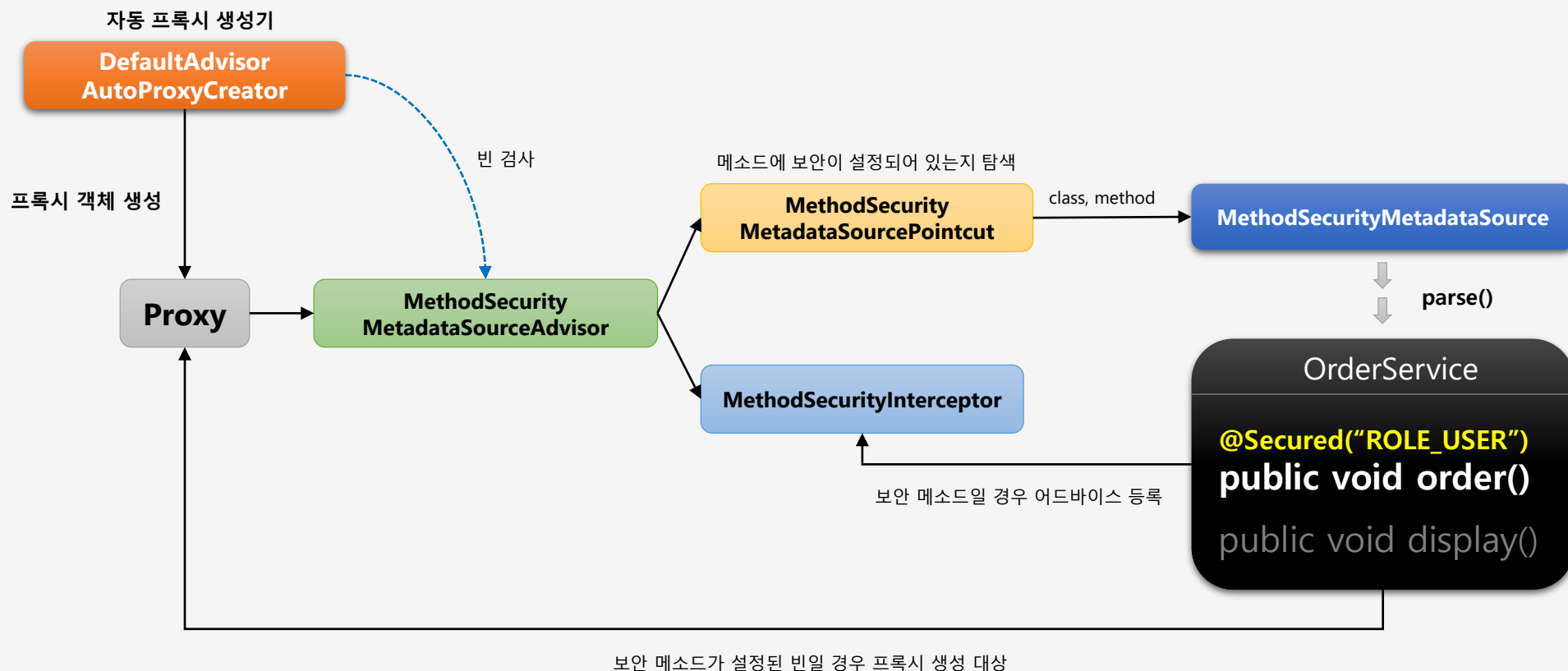
• 진행과정

- ① 메소드 호출 시 프록시 객체를 통해 메소드를 호출
- ② Advice가 등록되어 있다면 Advice를 작동하게 하여 인가 처리
- ③ 권한 심사 통과하면 실제 빈의 메소드를 호출한다

V. 인가(Authorization) 프로세스 구현 - DB 연동

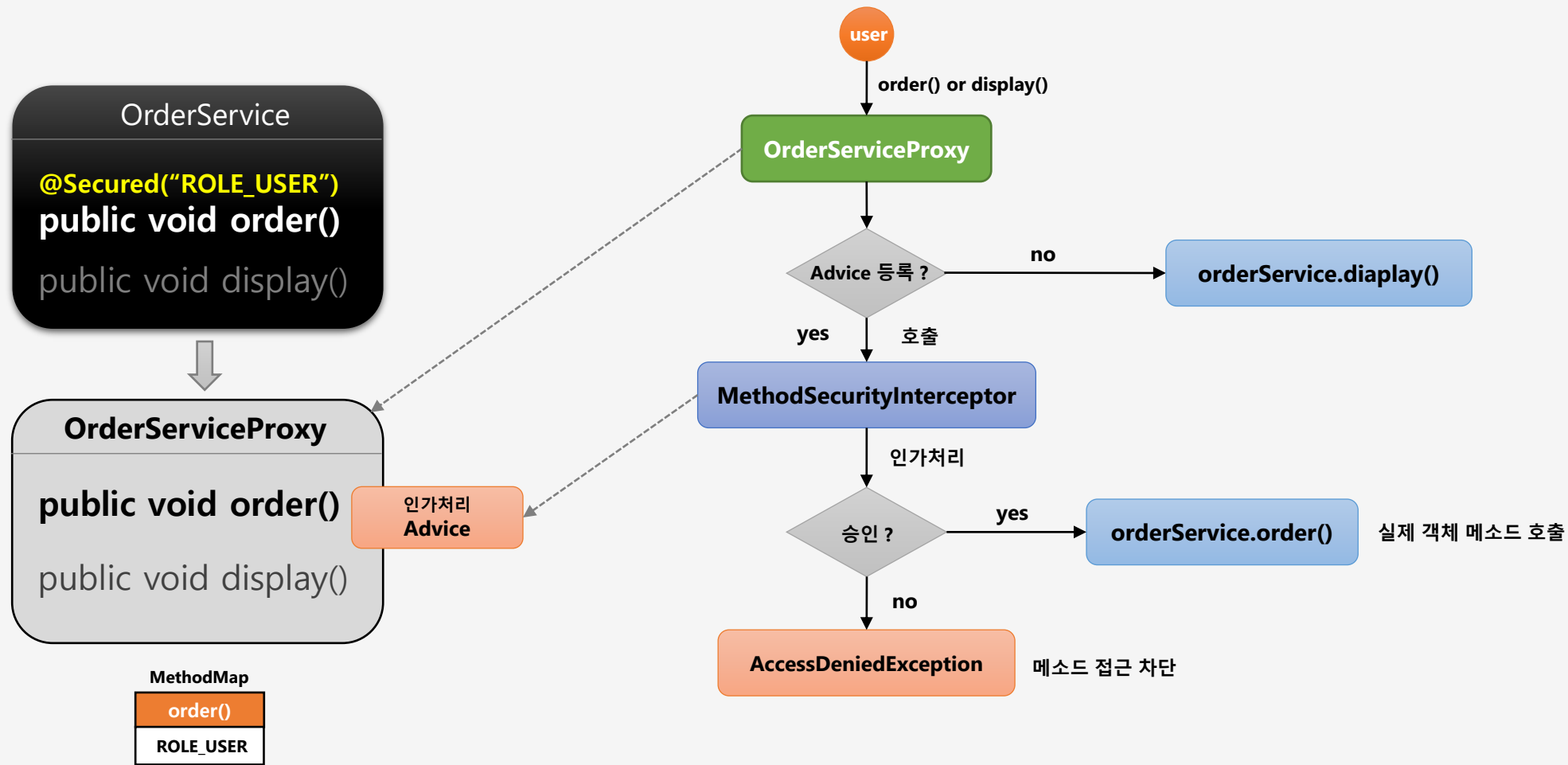
#09. Method 방식 - 주요 아키텍처

- 인가 처리를 위한 초기화 과정



V. 인가(Authorization) 프로세스 구현 - DB 연동

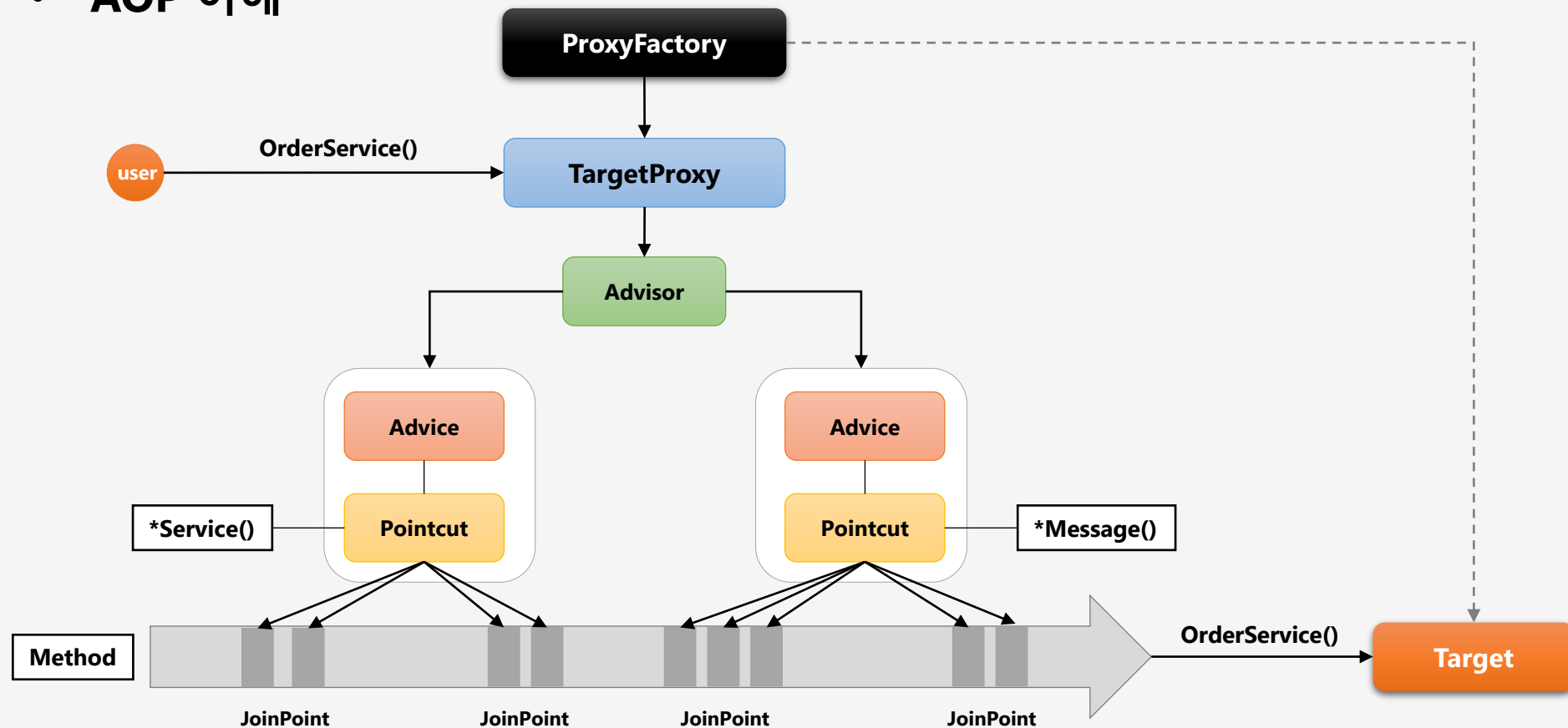
#09. Method 방식 - 주요 아키텍처



V. 인가(Authorization) 프로세스 구현 - DB 연동

#09. Method 방식 - 주요 아키텍처 이해

- AOP 이해



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



V. 인가(Authorization) 프로세스 구현 - DB 연동

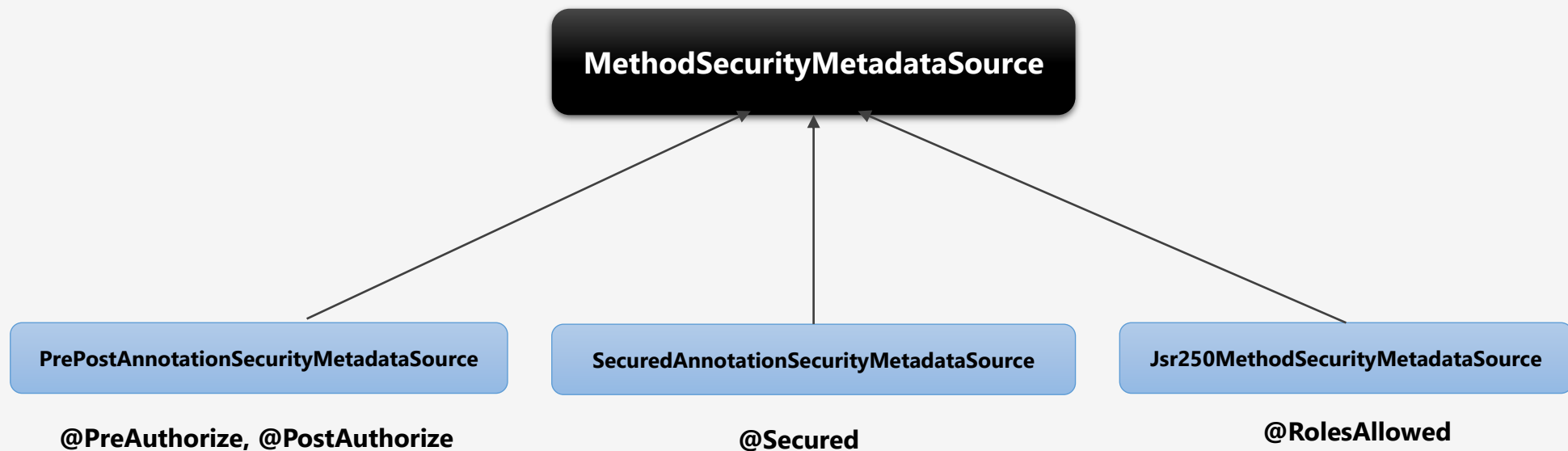
#09. Method 방식 – 어노테이션 권한 설정

II. 스프링 시큐리티 기본 API & Filter 이해

#09. Method 방식 – 어노테이션 권한 설정

- 보안이 필요한 메소드에 설정한다
- **@PreAuthorize, @PostAuthorize**
 - SpEL 지원
 - **@PreAuthorize("hasRole('ROLE_USER') and (#account.username == principal.username)")**
 - PrePostAnnotationSecurityMetadataSource 가 담당
- **@Secured, @RolesAllowed**
 - SpEL 미지원
 - **@Secured ("ROLE_USER"), @RolesAllowed("ROLE_USER")**
 - SecuredAnnotationSecurityMetadataSource, Jsr250MethodSecurityMetadataSource 가 담당
- **@EnableGlobalMethodSecurity**(prePostEnabled = true, securedEnabled = true)

#04. FilterInvocationSecurityMetadataSource



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



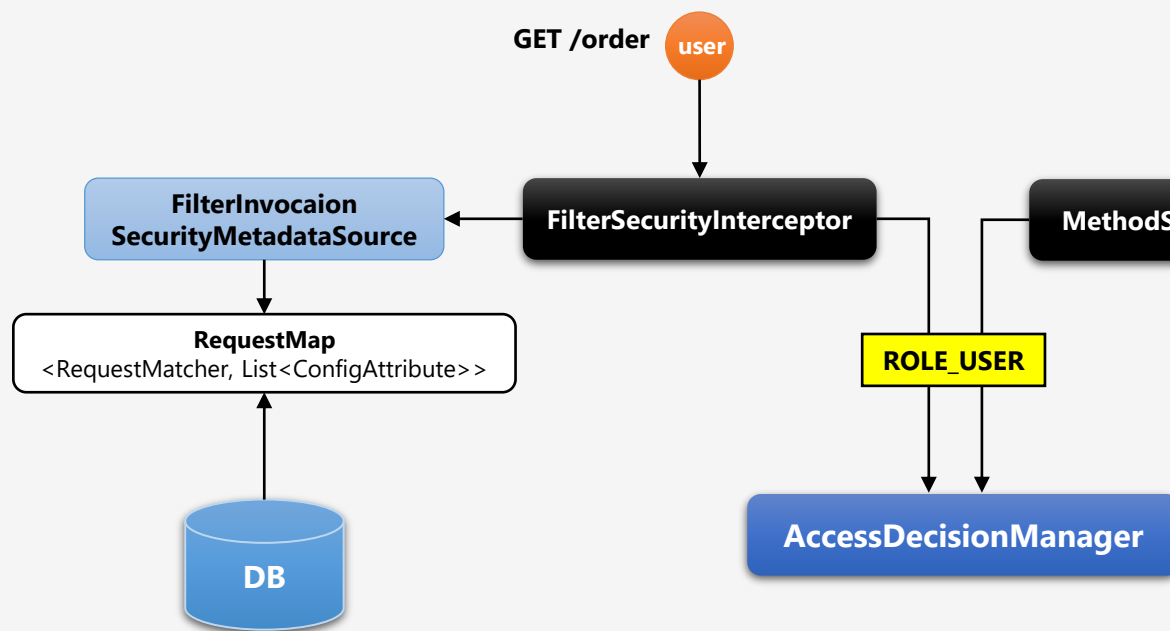
V. 인가(Authorization) 프로세스 구현 - DB 연동

#10. Method 방식 – Map 기반 DB 연동(1)

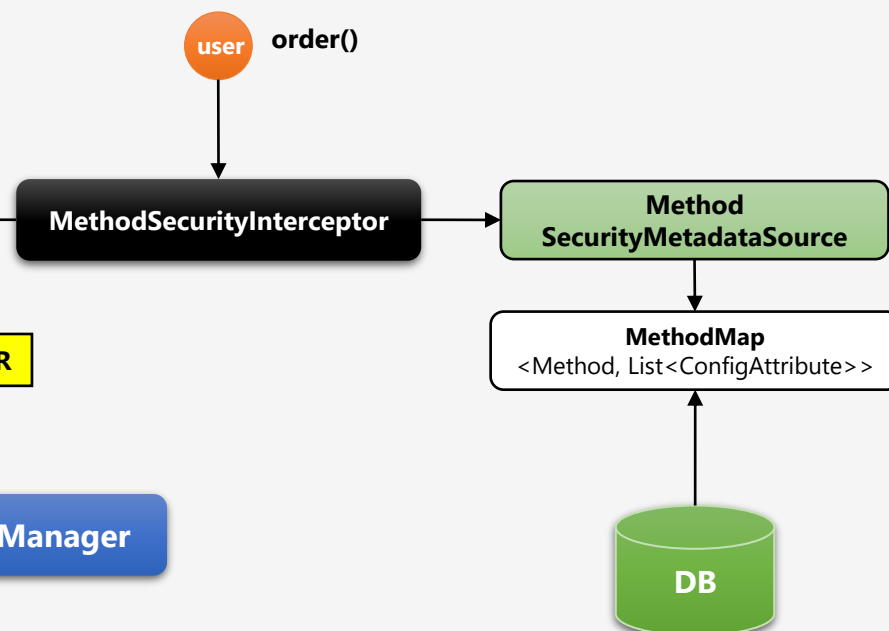
V. 인가(Authorization) 프로세스 구현 - DB 연동

#08. Method 방식 - Map 기반 DB 연동

Filter 기반 URL 방식



AOP 기반 Method 방식



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

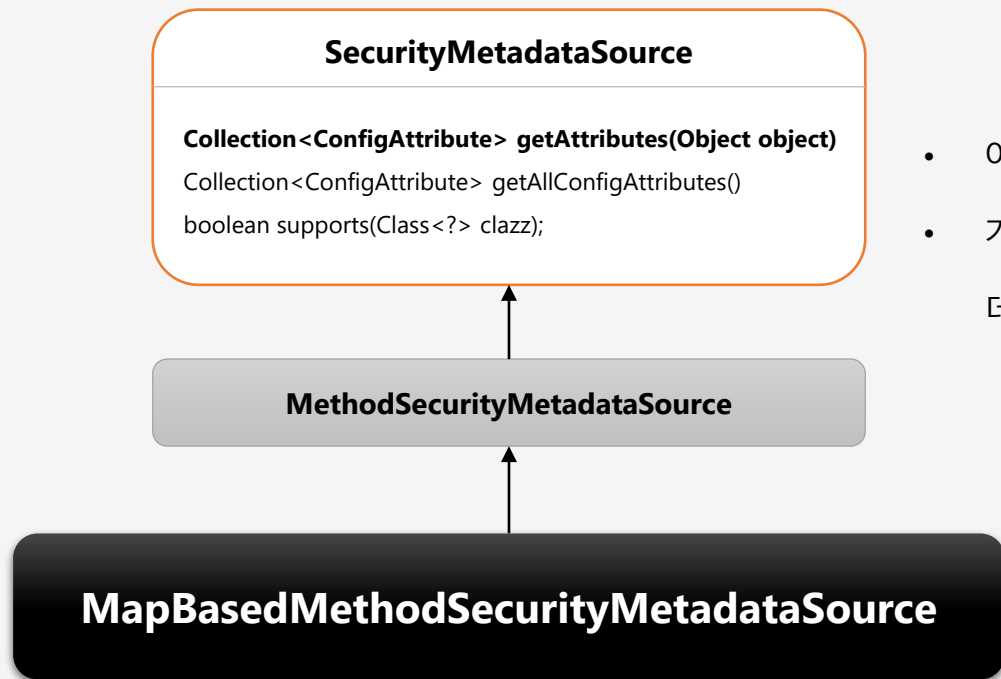


V. 인가(Authorization) 프로세스 구현 - DB 연동

#10. Method 방식 – Map 기반 DB 연동(2)

V. 인가(Authorization) 프로세스 구현 - DB 연동

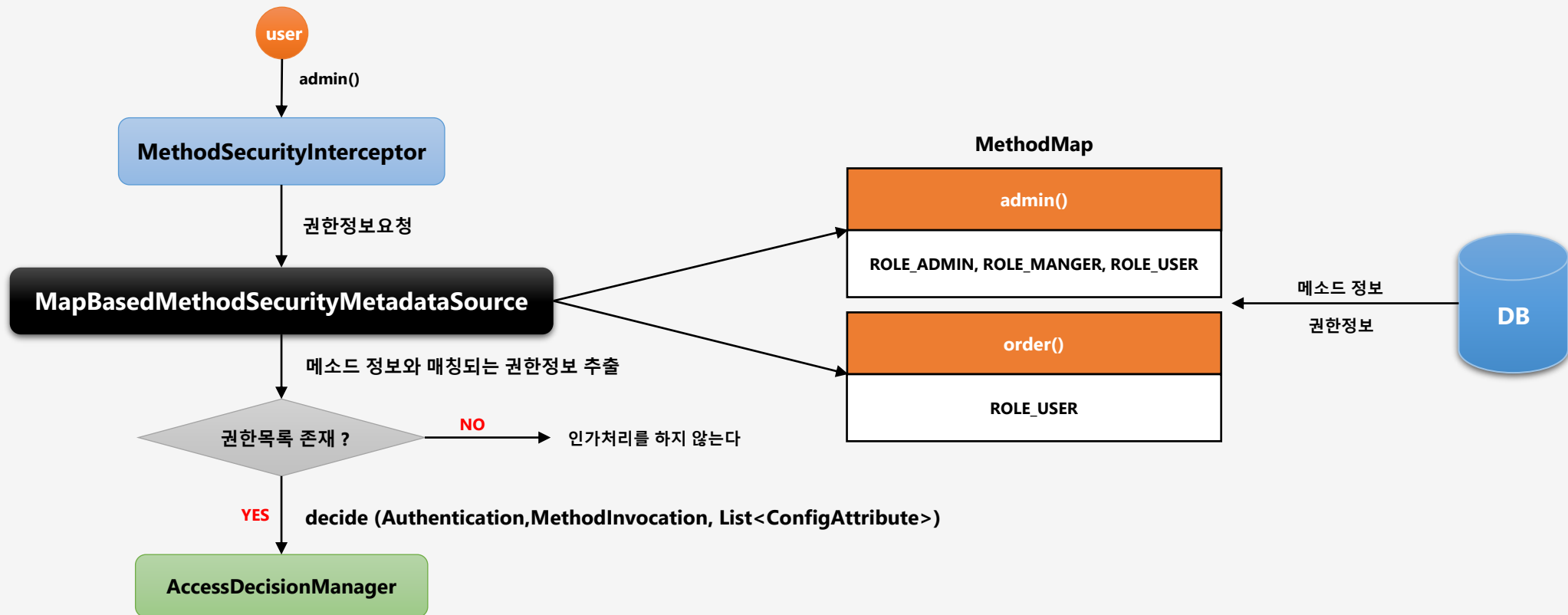
#10. Method 방식 - Map 기반 DB 연동



- 어노테이션 설정 방식이 아닌 맵 기반으로 권한 설정
- 기본적인 구현이 완성되어 있고 DB로부터 자원과 권한정보를 매핑한 데이터를 전달하면 메소드 방식의 인가처리가 이루어지는 클래스

V. 인가(Authorization) 프로세스 구현 - DB 연동

#10. Method 방식 - Map 기반 DB 연동



II. 스프링 시큐리티 기본 API & Filter 이해

#10. Method 방식 – Map 기반 DB 연동

@Configuration

@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)

class MethodSecurityConfig extends **GlobalMethodSecurityConfiguration** {

 @Override

 protected MethodSecurityMetadataSource **customMethodSecurityMetadataSource()** {

return new MapBasedMethodSecurityMetadataSource();

 }

}

Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍

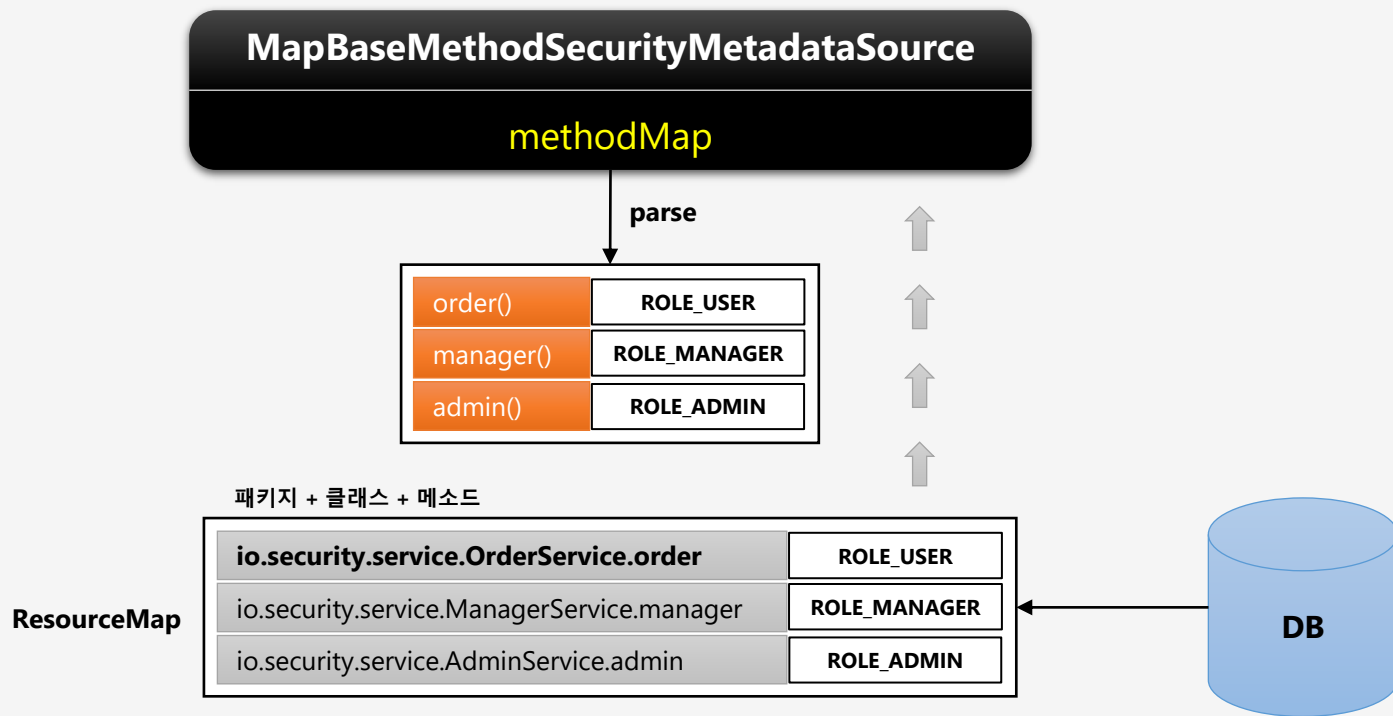


V. 인가(Authorization) 프로세스 구현 - DB 연동

#11. Method 방식 – Map 기반 DB 연동(3)

V. 인가(Authorization) 프로세스 구현 - DB 연동

#11. Method 방식 - Map 기반 DB 연동

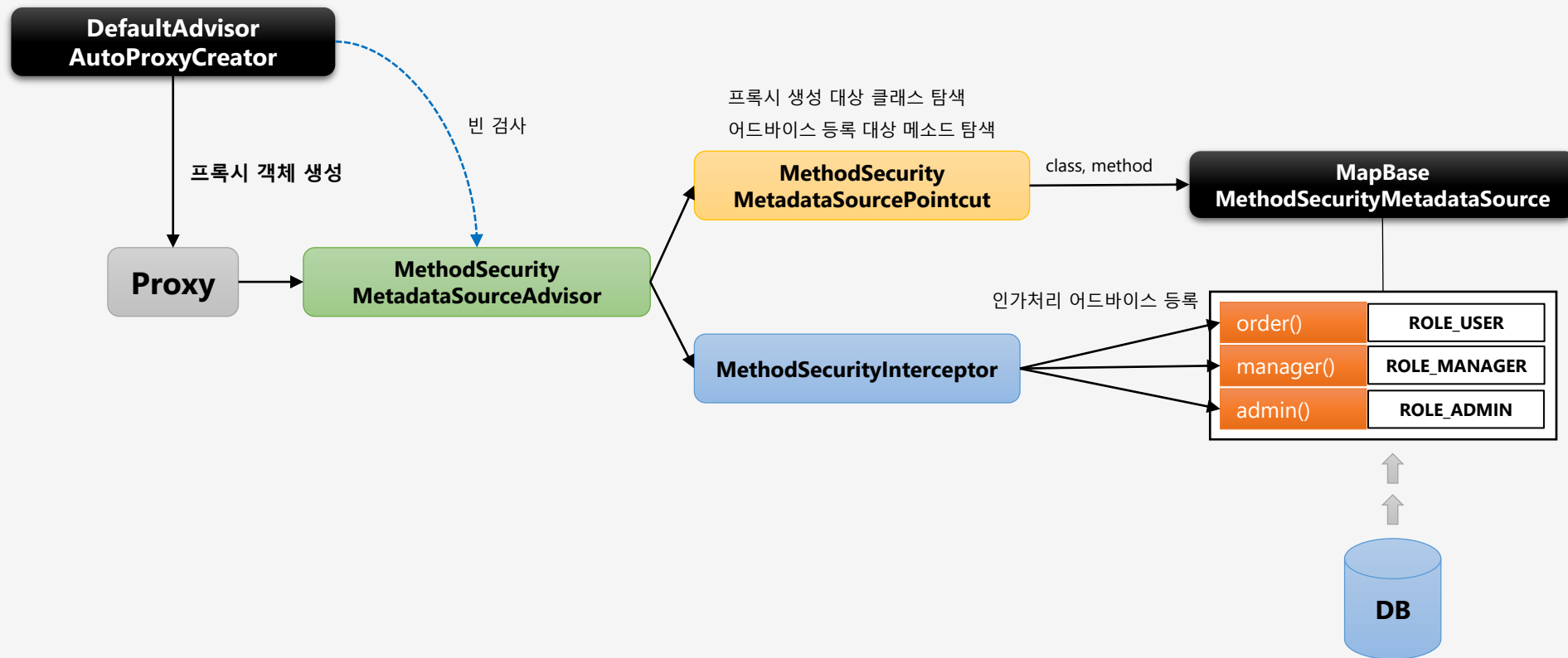


- **MethodResourcesMapFactoryBean**

- DB로 부터 얻은 권한/자원 정보를 ResourceMap 을 빈으로 생성해서 MapBasedMethodSecurityMetadataSource 에 전달

V. 인가(Authorization) 프로세스 구현 - DB 연동

#11. Method 방식 - Map 기반 DB 연동



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



V. 인가(Authorization) 프로세스 구현 - DB 연동

#12. Method 방식 - ProtectPointcutPostProcessor

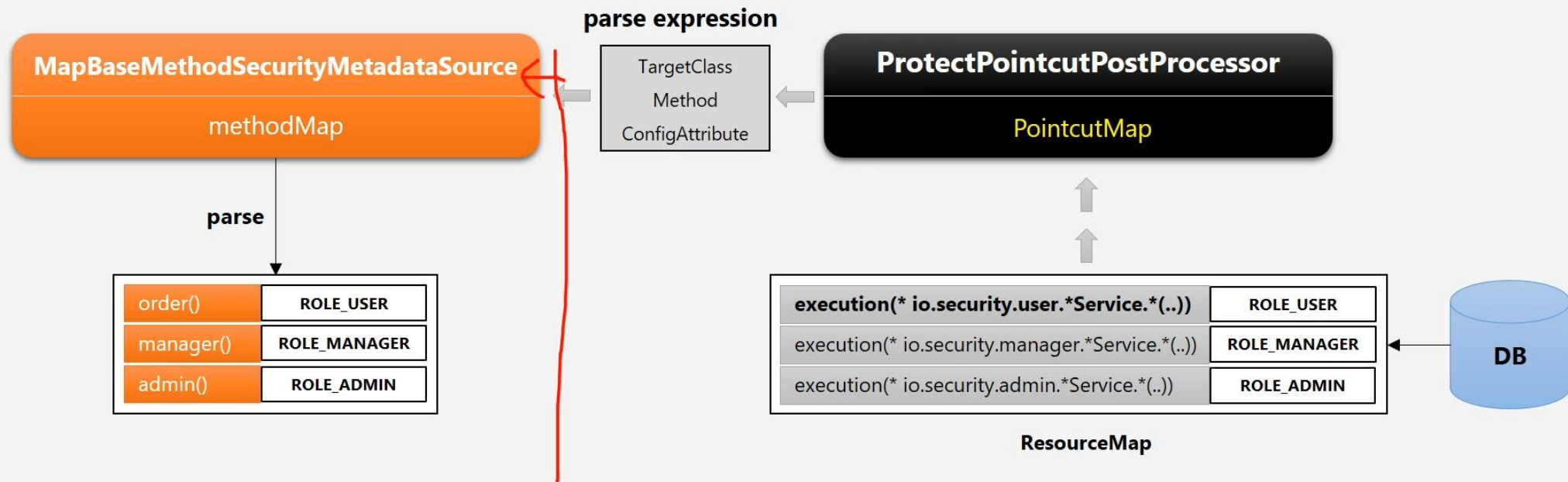
V. 인가(Authorization) 프로세스 구현 - DB 연동

#12. Method 방식 - ProtectPointcutPostProcessor

- 메소드 방식의 인가처리를 위한 자원 및 권한정보 설정 시 자원에 포인트 컷 표현식을 사용할 수 있도록 지원하는 클래스
- 빈 후처리기로써 스프링 초기화 과정에서 빈 들을 검사하여 빈이 가진 메소드 중에서 포인트 컷 표현식과 **matching** 되는 클래스 ,메소드, 권한 정보를 **MapBasedMethodSecurityMetadataSource** 에 전달하여 인가처리가 되도록 제공되는 클래스
- DB 저장 방식
 - **Method 방식**
 - `io.security.service.OrderService.order : ROLE_USER`
 - **Pointcut 방식**
 - `execution(* io.security.service.*Service.*(..)) : ROLE_USER`
- 설정 클래스에서 빈 생성시 접근제한자가 **package** 범위로 되어 있기 때문에 리플렉션을 이용해 생성한다

V. 인가(Authorization) 프로세스 구현 - DB 연동

#12. Method 방식 - ProtectPointcutPostProcessor

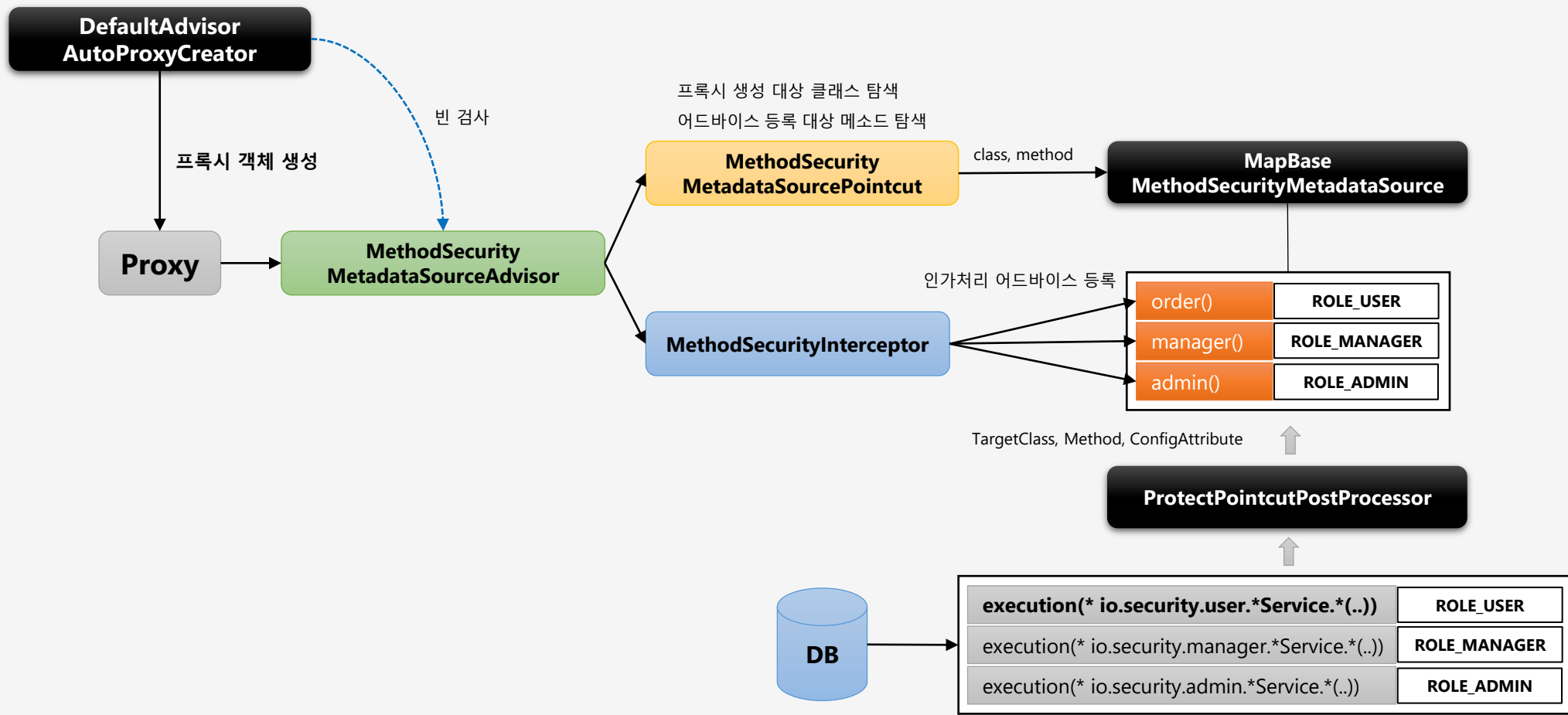


• MethodResourceMapFactoryBean

- DB로 부터 얻은 권한/자원 정보를 ResourceMap 빈으로 생성해서 ProtectPointcutPostProcessor 에 전달

V. 인가(Authorization) 프로세스 구현 - DB 연동

#12. Method 방식 - ProtectPointcutPostProcessor



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



V. 인가(Authorization) 프로세스 구현 - DB 연동

#12. 번외편 – 실시간 메소드 보안 구현

#12. 번외편 – 실시간 메소드 보안 구현

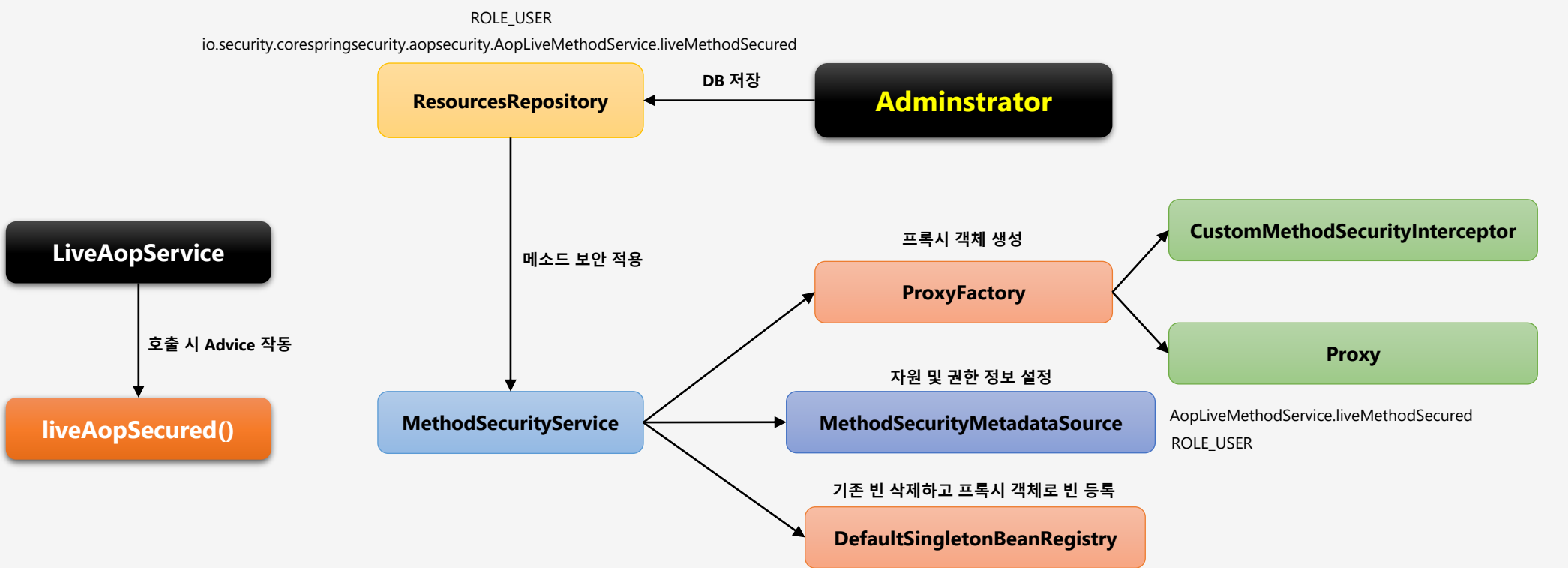
- **개선점**

- 메소드 보안은 스프링 시큐리티 초기화 시점에 보안 적용 대상 빈의 프록시 생성 및 어드바이스 적용이 이루어짐
- DB 에 자원을 실시간으로 업데이트 하더라도 AOP 가 바로 적용되지 않음

- **보안 메소드 실시간 적용 처리 과정**

1. 메소드 보안 최초 설정 시 대상 빈의 프록시 객체 생성하고 메소드에 Advice 등록하여 AOP 적용
2. MapBasedMethodSecurityMetadataSource 에 자원 및 권한 정보 전달
3. DefaultSingletonBeanRegistry 로 실제 빈을 삭제하고 프록시 객체를 빈 참조로 등록한다
4. 보안이 적용된 메소드 호출 시 Advice 가 작동한다
5. 메소드 보안 해제 시 메소드에 등록된 Advice 를 제거한다
6. 메소드 보안 재 설정 시 메소드에 등록된 Advice 를 다시 등록한다

#12. 번외편 - 실시간 메소드 보안 구현



Core Spring Security

핵심 개념 및 아키텍처 이해와 실전 예제로 완성하는 스프링 시큐리티 프로그래밍



VI. 강좌 마무리

Lecturer 정수원
<https://github.com/onjsdnjs>

VI. 강좌 마무리

#01. 학습한 내용

1. 보안 API

- 인증, 세션, 예외, 인가 등의 API 설정, 사용법, 기능 이해

2. Filter

- 요청에 따른 각 Filter 들의 처리과정과 동작 방식 이해

3. 내부 아키텍처

- 스프링 시큐리티 초기화 과정
- 요청에 대한 시작과 마지막 처리과정의 아키텍처 이해

4. 실전 프로젝트

- DB 연동 인증 - Form, Ajax
- DB 연동 인가 - Url, Method

VI. 강좌 마무리

#02. 더 많은 주제들

- 인증

- OAuth 2 인증
- JWT 인증
- 마이크로 서비스 인증 - Spring Cloud Security

- 인가

- Access Control List (ACL)

Core