

스프링 시큐리티란

스프링 시큐리티(Spring Security) 는

☞ Spring 기반 웹 애플리케이션에서 인증(Authentication)과 인가(Authorization)를 쉽게 구현하도록 도와주는 보안 프레임워크입니다.

웹 애플리케이션에서 자주 등장하는 다음과 같은 문제를 해결해 줍니다.

- 로그인 처리
- 로그아웃 처리
- 접근 권한 제어
- 세션 / 토큰 관리
- 소셜 로그인(OAuth2)

즉,

☞ “누가 접근했는지”,

☞ “무엇을 할 수 있는지”

를 관리하는 표준 도구입니다.

☞ 왜 시큐리티가 필요한가?

웹사이트는 크게 두 가지를 보호해야 합니다.

① 서버 리소스

관리자 페이지

게시글 등록/삭제 API

결제, 주문, 회원 관리 기능

☞ 아무나 접근하면 안 되는 영역

② 유저의 개인정보

아이디 / 비밀번호

이메일, 전화번호

결제 정보

☞ 유출되면 심각한 문제 발생

→ 그래서 웹 애플리케이션에는 반드시 보안 정책이 필요합니다.

☞ 인증(Authentication)

인증이란?

“이 사람이 누구인가?”를 확인하는 과정

인증이 필요한 이유

개인화된 서비스 제공

특정 기능 제한

사용자 식별

◇ 익명 사용자(Anonymous User)
로그인하지 않은 사용자
일부 페이지는 접근 허용 가능 (예: 메인 페이지)

◇ Username / Password 인증 (기본 로그인)
가장 일반적인 로그인 방식
아이디 + 비밀번호 → 서버 검증 → 로그인 성공

✓ 인증 이후 관리 방식

① 세션(Session) 방식

- 서버가 로그인 정보를 기억
- 전통적인 방식
- Spring Security 기본 방식

② 토큰(Token) 방식 (Sessionless)

- JWT 사용
- REST API, 모바일 환경에 적합

◇ SNS 로그인 (소셜 로그인)

네이버 / 카카오 / 구글 로그인

인증을 외부 서비스에 위임

OAuth2 기반

☞ 우리는 **"이 사람이 인증되었다"**는 결과만 신뢰

🔒 인가(Authorization, 권한)

인가란?

"이 사용자가 무엇을 할 수 있는가?"

✓ 예시

| 사용자 | 접근 가능 |
|---------|----------|
| USER | /user |
| MANAGER | /manager |
| ADMIN | /admin |
| 익명 사용자 | / |

☞ 인증 후 권한(Role) 을 기준으로 접근 제어

◇ 스프링 시큐리티의 인가 방식

✗ @Secured

예전 방식

deprecated (사용 권장 X)

@PreAuthorize / @PostAuthorize

```
@PreAuthorize("hasRole('ADMIN')")
public void adminOnly() {
    // 관리자만 접근 가능
}
```

메서드 실행 전/후 권한 체크

현재 가장 많이 사용

↳ AOP 기반

시큐리티는 내부적으로 AOP를 활용

컨트롤러/서비스에 공통 보안 로직 적용

⌚ 메모리 사용자 인증 (In-Memory Authentication)

개념

- DB 없이 메모리에 사용자 정보 저장
- 실습 / 테스트 용도
- ✓ 언제 사용?
- 시큐리티 구조 학습
- 빠른 테스트
- 프로토타입

⌚ 기본 사용자 로그인 방식들

① 기본 사용자 로그인 (Spring Boot 기본)

✓ 특징

- 아이디: user
- 비밀번호: 서버 실행 시 콘솔에 출력

② application.yml 에 사용자 등록

```
spring:
  security:
    user:
      name: user1
      password: 1234
      roles: USER
```

✓ 설정만으로 로그인 가능 ✓ 실습에 매우 적합

③ UserDetailsService 사용

✓ 핵심 인터페이스

UserDetailsService

↳ “아이디로 사용자를 조회하는 역할”

```
@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(String username) {
        return User.builder()
            .username("user1")
            .password("{noop}1234")
            .roles("USER")
            .build();
    }
}
```

✓ 실무에서 가장 많이 사용

✓ DB 연동 시 필수

④ SecurityConfig 설정

Spring Security의 핵심 설정 파일

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/admin/**").hasRole("ADMIN")
                .requestMatchers("/user/**").hasRole("USER")
                .anyRequest().authenticated()
            )
            .formLogin();

        return http.build();
    }
}
```

✓ URL 접근 권한 제어

✓ 로그인 방식 정의

⌚ 정리 요약

개념

설명

| 개념 | 설명 |
|--------------------|---------------------|
| 인증 | 사용자가 누구인지 확인 |
| 인가 | 사용자가 무엇을 할 수 있는지 결정 |
| Spring Security | 인증 + 인가 표준 프레임워크 |
| In-Memory | 실습/테스트용 사용자 |
| UserDetailsService | 실무 핵심 인터페이스 |
| SecurityConfig | 보안 정책의 중심 |

↳ 다음 실습 단계

DB 연동 로그인

BCryptPasswordEncoder

JWT 인증

OAuth2 소셜 로그인