# Spring Security

스프링 기반의 애플리케이션의 **보안**(인증, 권한, 인가 등)을 담당하는 스프링 하위 **프레임워크**

- Filter 기반으로 동작 → Spring MVC와 분리되어 관리 및 동작
- Security 3.2부터는 XML 설정없이 Java bean 설정으로 설정 가능
- Dispatcher Servlet으로 가기 전에 적용 → 가장 먼저 URL 요청 받음

## ⭐ 대표적인 기능

- 사용자 권한에 따른 URI 접근 제어
- DB와 연동하는 Local strategy 로그인
- 쿠키를 이용한 자동 로그인
- 패스워드 암호화

## 📌 용어 정리

- 접근 주체 (Principal)

  보호된 리소스에 접근하는 대상

- 인증 (Authentication)

  증명하다, 보호된 리소스에 접근한 대상에 대해 이 유저가 누구인지, 작업을 수행해도 되는 주체인지 확인하는 과정

  (EX) Login하는 과정

- 인가 (Authorization)

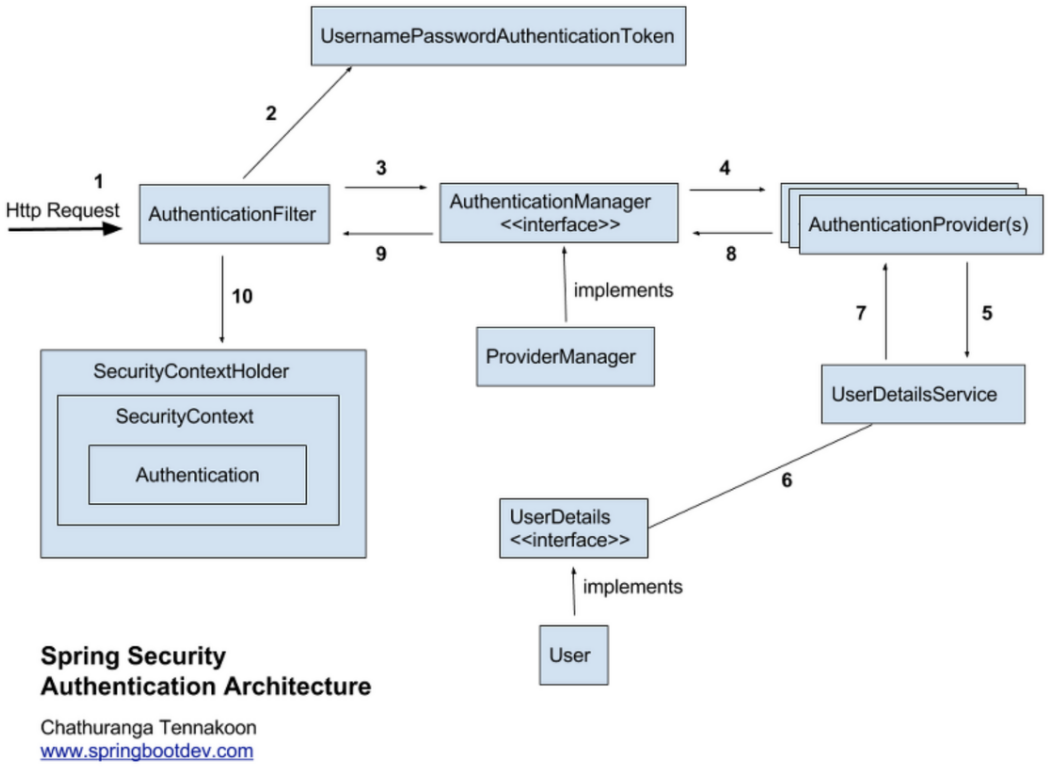
  권한부여 / 허가, 해당 리소스에 대해 접근 가능한 권한을 가지고 있는지 확인하는 과정

- 권한

  어떠한 리소스에 대한 접근 제한. 제한된 최소한의 권한을 가졌는지 확인

## 🔷 Form 기반 Login Flow

**2-1. 인증관련 architecture**




## 전통적으로 **세션-쿠키** 방식으로 인증

1. 로그인 정보 입력 및 인증 요청
2. **AuthenticationFilter**가 로그인 정보를 이용해 **UsernamePasswordAuthenticationToken**(인증용 객체 == Authentacation) 생성
   (JSESSIONID가 Security Context에 있는지 판단)
3. **AuthenticationManager**( `implements` **ProviderManager** )에게  인증용 객체 전달
4. 실제 인증을 할 **AuthencationProvider**(AuthencationProviders 중 하나로 커스텀 가능)에게 **Authentication**(UsernamePasswordAuthenticationToken) 객체 전달
5. **UserDetailsService**에게 사용자 아이디 전달
6. UserDetailsService는 DB에서 인증에 사용할 사용자 정보(아이디, 암호화된 패스워드, 권한 등)을 **UserDetails**객체 형태로 전달 받음
   (인증용 객체와 도메인 객체를 분리하지 않기 위해 도메인 객체에 UserDetails를 상속)
7. **AuthenticationProvider**는 UserDetails 객체를 전달 받아 로그인 입력정보와 UserDetails 객체를 가지고 인증 시도
8. 인증 완료 후, AuthenticationManager는 Authentication객체(인증 성공 유무 저장)를 반환하여 **SecurityContextHolder**에 사용자 인증 정보 저장

9. AuthenticationFiter에게 인증 성공 유무 전달
   성공 시, **AuthenticationSuccessHandler** 호출
   실패 시, **AuthenticationFailureHandler** 호출

> ❓ **Authentication** 객체와 **UserDetails** 객체의 차이
> Authentication : 인증 요청정보 + 인증 후, **인증 유무 저장** 및 성공시 UserDetails 객체 저장
> UserDetails : 사용자 정보 저장

[ **SecurityContextHolder** ]

보안 주체의 세부 정보 포함. 응용프로그램의 현재 보안 컨텍스트에 대한 세부 정보 저장

[ **SecurityContext** ]

Authentication을 보관 및 사용

[ **Authentication** ]

현재 접근하는 주체의 정보와 권한을 저장한 인터페이스. Security Context에 저장

- 접근 : SecurityContextHolder ⇒ SecurityContext ⇒ Authentication

```
public interface Authentication extends Principal, Serializable {
    // 현재 사용자의 권한 목록을 가져옴
    Collection<? extends GrantedAuthority> getAuthorities();

    // credentials(주로 비밀번호)을 가져옴
    Object getCredentials();

    // 사용자 상세정보
    Object getDetails();

    // Principal(주로 ID) 객체를 가져옴.
    Object getPrincipal();

    // 인증 여부를 가져옴
    boolean isAuthenticated();

    // 인증 여부를 설정함
    void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException;
}
```

[ **UsernamePasswordAuthenticationToken** ]

Authentication을 implements한 AbstactAuthenticationToken의 하위 클래스

User의 ID가 Principal 역할, Password가 Credential 역할 수행

```
public class UsernamePasswordAuthenticationToken extends AbstractAuthenticationToken {
    // 주로 사용자의 ID에 해당함
    private final Object principal;
    // 주로 사용자의 PW에 해당함
    private Object credentials;

    // 인증 완료 전의 객체 생성
    public UsernamePasswordAuthenticationToken(Object principal, Object credentials) {
    super(null);
    this.principal = principal;
    this.credentials = credentials;
    setAuthenticated(false);
  }

    // 인증 완료 후의 객체 생성
    public UsernamePasswordAuthenticationToken(Object principal, Object credentials,
      Collection<? extends GrantedAuthority> authorities) {
    super(authorities);
    this.principal = principal;
    this.credentials = credentials;
    super.setAuthenticated(true); // must use super, as we override
  }
}


public abstract class AbstractAuthenticationToken implements Authentication, CredentialsContainer {
}
```

[ **AuthenticaitionProvider** ]

실제 인증에 대한 부분을 처리. 인증 전의 Authentication객체를 받아 인증이 완료된 객체를 반환하는 역할

AuthenticationProvider 인터페이스를 구현 후, Custom한 AuthenticaionProvider을 AuthenticaionManager에 등록

```
public interface AuthenticationProvider {

  // 인증 전의 Authenticaion 객체를 받아서 인증된 Authentication 객체를 반환
    Authentication authenticate(Authentication var1) throws AuthenticationException;

    boolean supports(Class<?> var1);

}
```

[ **AuthenticationManager** ]

실질적으로 인증에 대한 처리는 AuthenticationManager에 등록된 AuthenticaionProvider에 의해 처리.

인증 성공 후, UsernamePasswordAuthenticationToken의 2번째 생성자를 이용해 인증이 성공한 객체를 생성하여 Security Context에 저장. 인증 상태를 유지하기 위해 세센에 보관.

인증 실패 시, AuthenticaionException 발생

```
public interface AuthenticationManager {
  Authentication authenticate(Authentication authentication)
    throws AuthenticationException;
}
```

[ **ProviderManager** ]

AuthenticationManager를 implements한 메서드

AuthenticationProvider의 List 저장. for문을 통해 모든 provider를 조회 및 Authenticate 처리

```
public class ProviderManager implements AuthenticationManager, MessageSourceAware,
InitializingBean {
    public List<AuthenticationProvider> getProviders() {
    return providers;
  }
    public Authentication authenticate(Authentication authentication)
      throws AuthenticationException {
    Class<? extends Authentication> toTest = authentication.getClass();
    AuthenticationException lastException = null;
    Authentication result = null;
    boolean debug = logger.isDebugEnabled();
        //for문으로 모든 provider를 순회하여 처리하고 result가 나올 때까지 반복한다.
    for (AuthenticationProvider provider : getProviders()) {
            ....
      try {
        result = provider.authenticate(authentication);

        if (result != null) {
          copyDetails(authentication, result);
          break;
        }
      }
      catch (AccountStatusException e) {
        prepareException(e, authentication);
        // SEC-546: Avoid polling additional providers if auth failure is due to
        // invalid account status
        throw e;
      }
          ....
    }
    throw lastException;
  }
}
```

[ **CustomAuthenticationProvider 등록 방법** ]

WebSecurityConfigurerAdapter(AuthenticationManager의 하위 클래스)를 상속한 SecurityConfig에서 가능

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

    @Bean
    public AuthenticationManager getAuthenticationManager() throws Exception {
        return super.authenticationManagerBean();
```

```
    }

    @Bean
    public CustomAuthenticationProvider customAuthenticationProvider() throws Exception {
        return new CustomAuthenticationProvider();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(customAuthenticationProvider());
    }
}
```

[ **UserDetails** ]

인증에 성공하여 생성된 Class

UsernamePasswordAuthenticationToken을 생성하기 위해 사용

UserVO 모델에 UserDetails를 implements하여 처리

- Spring Security에서 org.springframework.security.core.userdetails.User 클래스 제공

  BUT. 이메일, 프로필 이미지 경로와 같은 부가적인 정보 저장 불가능⇒ 상속받아 구현

```
public interface UserDetails extends Serializable {

    Collection<? extends GrantedAuthority> getAuthorities();

    String getPassword();

    String getUsername();

    boolean isAccountNonExpired();

    boolean isAccountNonLocked();

    boolean isCredentialsNonExpired();

    boolean isEnabled();

}
```

[ **UserDetailsService** ]

UserDetails 객체를 반환하는 인터페이스.

일반적으로, UserDetailsService를 구현한 클래스 내부에 UserRepository를 주입받아 DB와 연결하여 처리

```
public interface UserDetailsService {

    UserDetails loadUserByUsername(String var1) throws UsernameNotFoundException;

}
```

[ **Password Encoding** ]

패스워드 암호화에 사용될 PasswordEncoder 구현체 지정

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
  // TODO Auto-generated method stub
  auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());
}

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

[ **GrantedAuthority** ]

사용자(principal)가 가지고 있는 권한

ROLE_ADMIN, ROLE_USER와 같은 ROLE_*의 형태로 사용

UserDetailsService에 의해 호출. 특정 자원에 대한 권한이 있는지 검사하여 접근 허용 여부를 결정

---

# Spring Security 적용

## 1. 의존성 추가 (Maven)

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-security -->

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

## 2. Sprinf Security 설정 활성화 및 커스텀

SecurityConfig 클래스 구성 및 WebSecurityConfigurerAdapter 상속

`@Configuration`

⇒ 해당 클래스를 Configuration으로 등록

`@EnableWebSecurity`

⇒ Spring Security를 활성화

`@EnableGlobalMethodSecurity(prePostEnabled=true)` *-선택사항*

⇒ Controller에서 특정 페이지에 특정 권한이 있는 유저만 접근을 허용하는 @PreAuthorize 어노테이션을 활성화시키는 어노테이션

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

### ◆ 인증 제외

**[ configure(WebSecurity web) 메서드 ]**

- **WebSecurity** : FilterChainProxy를 생성하는 필터
- Spring Security가 무시할 수 있도록 설정
- 파일 기준 : resources/static 디렉터리

```
@Override
public void configure(WebSecurity web) throws Exception
{
   // static 디렉터리의 하위 파일 목록은 인증 무시 ( = 항상통과 )
   web.ignoring().antMatchers("/css/**", "/js/**", "/img/**", "/lib/**");
}
```

### ◆ 로그인 / 로그아웃

**[ configure(HttpSecurity http) 메서드 ]**

- HttpSecurity : HTTP 요청에 대한 웹 기반 보안을 구성
-
- `authorizeRequests()` : HttpServletRequest 따라 접근(Access) 제한

  ✅ **antMatchers()** : 특정 경로 지정

  ✅ **permitAll(), hasRole()** : Role(권한)에 따른 접근 설정

- `formlogin()` : Form 기반 로그인
  - 기본적으로 HttpSession 사용
  - /login 경로로 접근

  ✅ **.loginPage("/user/login")** : 커스텀 로그인 Form 사용시 경로 지정. 로그인 Form의 action 경로와 일치해야함

  ✅ **.loginProcessingUrl("/경로명")** : 로그인 Form의 action 경로 커스텀

✅ **.defaultSuccessUrl("/user/login/result")** : 로그인 성공 시 이동되는 페이지. Controller URL 매핑 필수

✅ **.usernameParameter("파라미터명")** : 로그인 Form에서 name = username인 input을 기본으로 인식. usernameParameter() 메서드를 통해 커스텀 가능

- `logout()` : 로그아웃 지원
  - WebSecurityConfigurerAdapter 사용 시 자동 적용
  - /logout 경로로 접근

✅ **.logoutRequestMatcher(new AntPathRequestMatcher("/user/logout"))** : 다른 URL로 재정의

✅ **.invalidateHttpSession(true)** : HTTP 세션 초기화

✅ **.deleteCookies("KEY명")** : 로그아웃 시, 특정 쿠키를 제거하는 메서드

```
@Override
protected void configure(HttpSecurity http) throws Exception {
   http
       // csrf(Cross site request forgery, 사이트간 요청 위조) 보안 설정을 비활성화
       .csrf().disable()
       .headers().frameOptions().disable()
       .and()
       .authorizeRequests()
       // 페이지 권한 설정
            .antMatchers("/admin/**").hasRole("ADMIN")
            .antMatchers("/user/myinfo").hasRole("MEMBER")
            .antMatchers("/**").permitAll()
       .and() // 로그인 설정
            .formLogin()
            .loginPage("/user/login")
            .defaultSuccessUrl("/user/login/result")
            .permitAll()
       .and() // 로그아웃 설정
            .logout()
            .logoutRequestMatcher(new AntPathRequestMatcher("/user/logout"))
            .logoutSuccessUrl("/user/logout/result")
            .invalidateHttpSession(true)
       .and()
             // 403 예외처리 핸들링
             .exceptionHandling().accessDeniedPage("/user/denied");

          // .authenticationEntryPoint(new LoginUrlAuthenticationEntryPoint("/"));
    }
```

🔶 **패스워드 암호화**

[ **configure(AuthenticationManagerBuilder auth) 메서드** ]

- AuthenticationManager를 생성하기 위해 AuthenticationManagerBuilder 사용

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

@Override
public void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(memberService).passwordEncoder(passwordEncoder());
}
// memberService == UserDetailService
```

**[ 참고사이트 ]**

`Speing Security 정의`

- https://mangkyu.tistory.com/76
- https://bamdule.tistory.com/52
- https://velog.io/@jayjay28/2019-09-04-1109-작성됨

`Spring Security 구현`

- https://dailyheumsi.tistory.com/185
- https://xmfpes.github.io/spring/spring-security/
- https://victorydntmd.tistory.com/328