스프링 시큐리티 (인프런)

대부분의 애플리케이션은 "인증"과 "인가" 없이 제대로 동작할 수 없습니다. 게시판을 만든다고 가정해 봅시다. 익명 게시판이 아닌 이상 여러분이 만든 게시판 애플리케이션은 현재 어떤 사용자가 글을 작성하고 있는지 알아야 합니다. 흔히 '로그인'이라고 부르고 스프링 시큐리티에서는 Authentication 또는 "인증"이라고 말하는 기능이 필요합니다. 또한 글을 수정하는 기능을 구현할 때, 그 글을 수정할 수 있는 사용자는 최소한 그 글을 작성한 사용자 또는 관리자 권한을 가지고 있어야 합니다. 이때 개발자로서 우리는 애플리케이션에 "인가", Authorization 또는 Access Contrl 기능을 적용하여 적절한 권한을 가진 사용자만 해당 글을 수정할 수 있도록 기능을 구현해야 합니다. 또한 웹 애플리케이션인 경우, CSRF, XXS, 세션 변조, Clickjacking 등 다양한 웹 보안 관련 이슈에 대응하는 것도 반드시 필요합니다.

이 강좌는 폼 기반의 웹 애플리케이션에 스프링 시큐리티가 제공하는 다양한 기능을 적용하며 스프링 시큐리티 구조를 파악합니다. 단순히 이런 저런 기능을 적용해 보는 것에 그치지 않고 스프링 시큐리티가 서블릿 기반 웹 애플리케이션에 어떻게 맞물려 동작하는지 그 내부 구조를 학습합니다. 따라서, AuthenticationManager, AccessDecisionManager, FilterChainProxy 등 스프링 시큐리티 내부 구조를 학습하는데 많은 도움이 될 것으로 기대합니다.

이 강좌는 타임리프(Thymeleaf)를 뷰 템플릿으로 사용하는 서블릿 기반 애플리케이션을 주로 다룹니다. OAuth2, Reactive(WebFlux) 그리고 웹소켓을 지원하는 기능에 대해서는 다루지 않았으며 메소드 시큐리티에 대해서는 간략하게 살펴봤습니다. 이번 강좌의 초점을 보다 대중적인 애플리케이션 형태에 집중하려는 선택이었습니다. 하지만, 이 강좌를 충분히 학습하신다면 그 지식을 기반으로 여기서 다루지 않은 기능도 손쉽게 익힐 수 있을 겁니다.

견지망월(見指忘月)이라는 옛말이 있습니다. 물론 손가락도 봐야 하지만 여러분 달도 꼭 보시기 바랍니다. 지금까지 만든 또 앞으로도 만들 제 모든 강좌를 보면서 여러분은 제가 무언가를 학습하는 방법. 그 방법을 익힐 수 있습니다. 저는 항상 테스트를 중요하게 생각하지만 그렇다고 TDD에 집착하지는 않습니다. (가끔 그렇게 보인다는 피드백을 받기도 했지만..) 모든 기능은 직접 코딩하여 검증하며, 이해가 되지 않는 부분 또는 더 자세히 보고 싶은 부분은 디버거를 활용합니다. 여러분은 이 강좌에서도 어김없이 스프링 시큐리티를 적용했을 때 테스트 코드를 작성하는 방법과 디버거를 사용하여 분석하는 방법을 익힐 수 있습니다.

어떤 IDE를 사용하는지는 중요하지 않습니다만 여러분에게 익숙한 IDE를 사용하시기 바랍니다. 이 강좌에서는 IntelliJ IDEA(인텔리J) 유료 버전을 사용합니다. 인텔리J 무료 버전을 사용해도 강좌를 따라 코딩하는 데는 크게 지장은 없지만 일부 과정이 여러분 보다 훨씬 편리해 보일 수는 있습니다. 스프링에서 무료로 제공하는 이클립스 기반의 STS(스프링 툴 스윗)을 사용한다면 비슷한 수준의 편리함을 누릴 수 있을 겁니다.

이 강좌는 수강하시는 분들이 다음과 같은 선수 지식을 갖췄다는 가정하게 만들었습니다. 아직 수강하지 않은 강좌 또는 학습하지 않은 주제가 있다면 꼭 미리 학습하신 뒤에 이번 강좌를 수강하시기 바랍니다.

- 스프링 핵심 기술
- 스프링 부트
- 스프링 웹 MVC
- 스프링 데이터 JPA (optional)

감사합니다.

1부 스프링 시큐리티: 폼 인증

1. 폼 인증 예제 살펴보기

앞으로 차근 차근 만들고 또 다음어 갈 예제가 어떻게 동작하는지 먼저 살펴보겠습니다.

이 애플리케이션에는 다음과 같은 총 4개의 뷰가 있습니다.

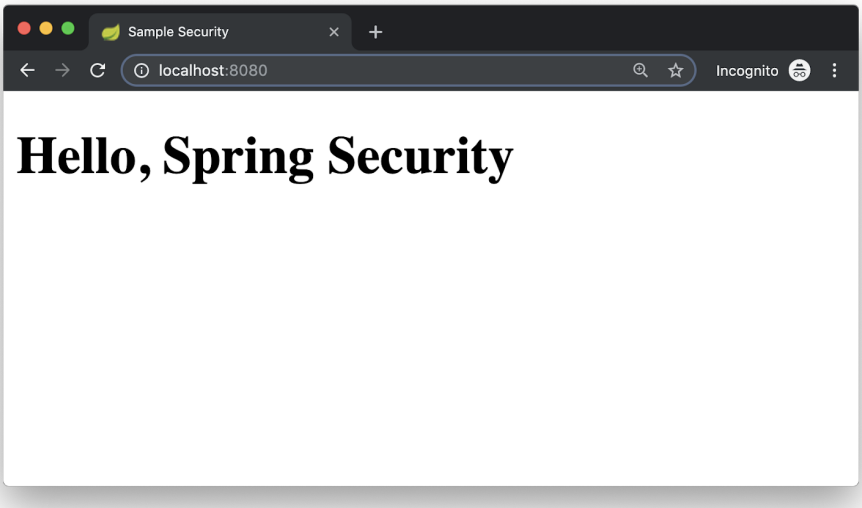
홈 페이지
- /
- 인증된 사용자도 접근할 수 있으며 인증하지 않은 사용자도 접근할 수 있습니다.
- 인증된 사용자가 로그인 한 경우에는 이름을 출력할 것.

정보
- /info
- 이 페이지는 인증을 하지 않고도 접근할 수 있으며, 인증을 한 사용자도 접근할 수 있습니다.

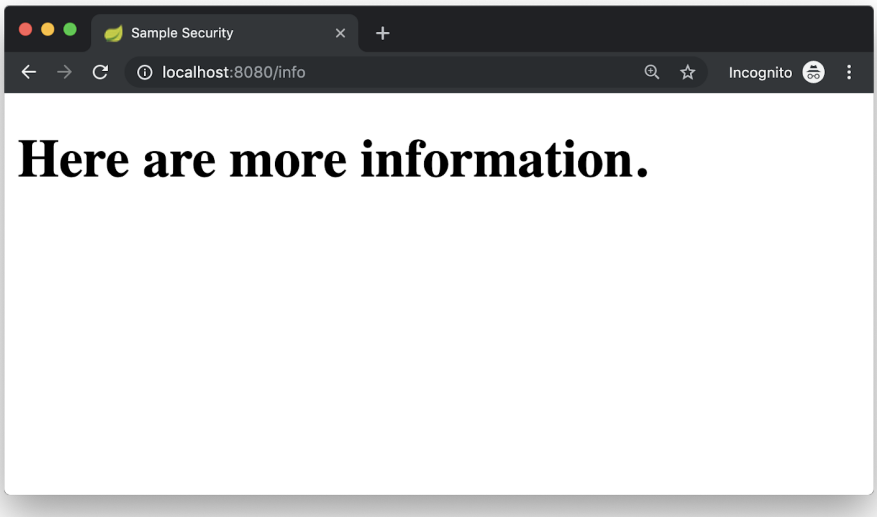
대시보드
- /dashboard
- 이 페이지는 반드시 로그인 한 사용자만 접근할 수 있습니다.
- 인증하지 않은 사용자가 접근할 시 로그인 페이지로 이동합니다.

어드민
- /admin
- 이 페이지는 반드시 ADMIN 권한을 가진 사용자만 접근할 수 있습니다.
- 인증하지 않은 사용자가 접근할 시 로그인 페이지로 이동합니다.
- 인증은 거쳤으나, 권한이 충분하지 않은 경우 에러 메시지를 출력합니다.

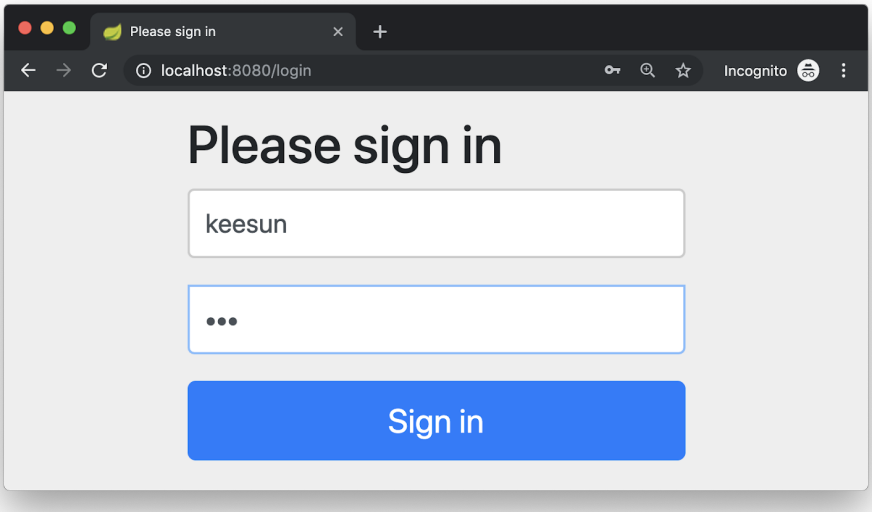
첫 페이지 (/) 에 로그인 하지 않고 접속하면 보이는 메시지 확인.



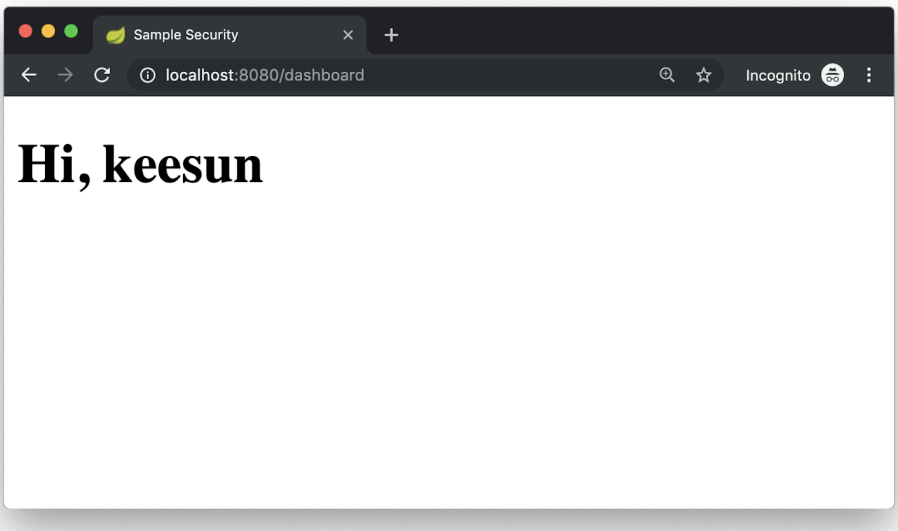
로그인 하지 않고 볼 수 있는 페이지 (/info)에 접근.



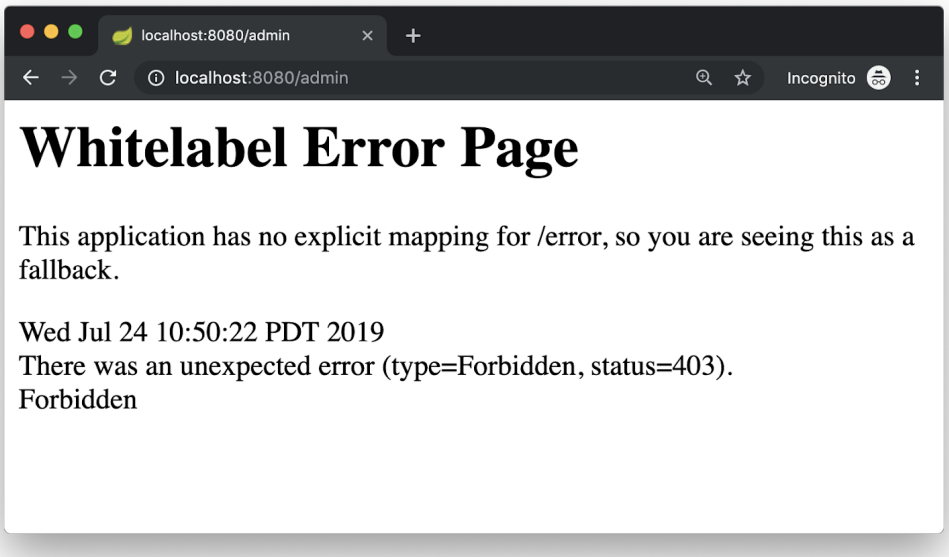
로그인 해야만 볼 수 있는 페이지 (/dashboard)에 접속할 때 로그인 페이지로 이동.



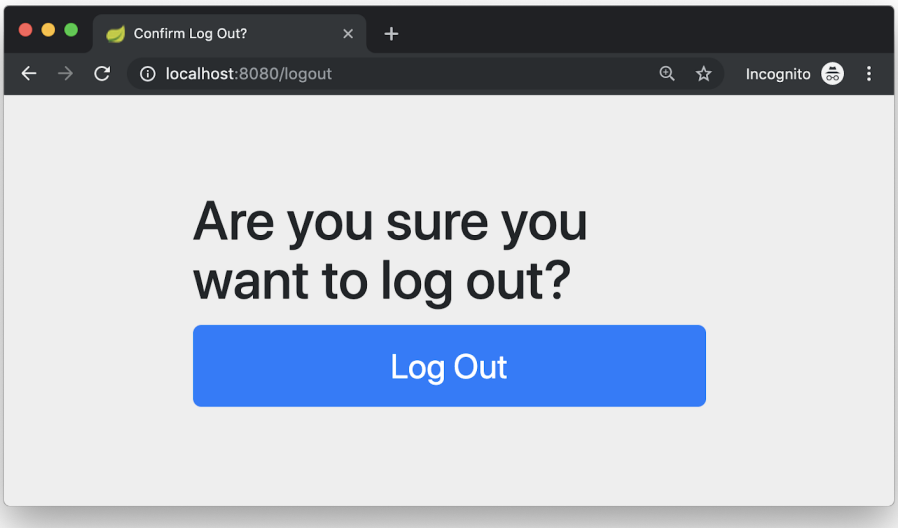
로그인 한 뒤에 가려던 페이지 (/dashboard)로 이동.



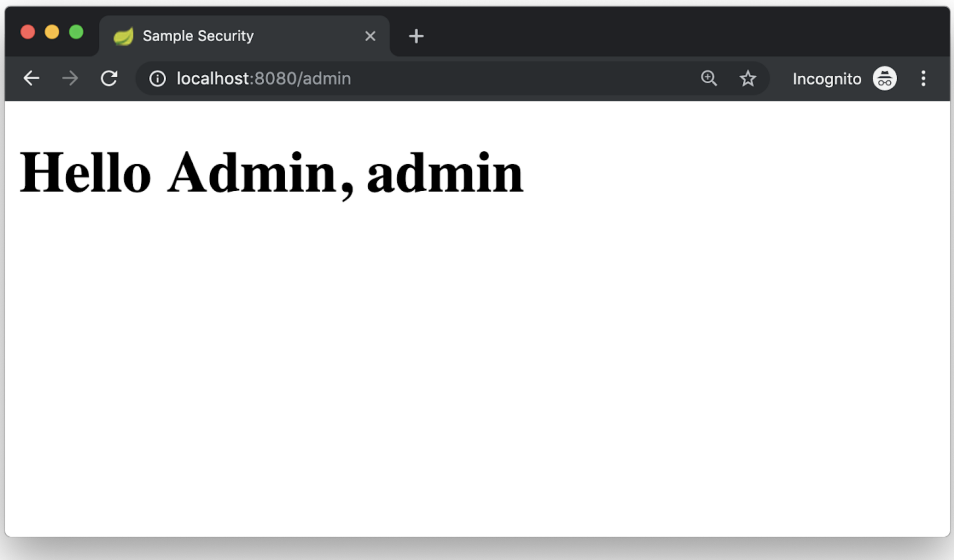
ADMIN 권한이 있는 유저만 접근 가능한 URL (/amdin)로 이동하려는 경우에 보이는 화면



로그아웃 (/logout) 하는 경우에 보이는 화면



ADMIN으로 로그인 한 뒤 (/admin) 접근시 보이는 화면



2. 스프링 웹 프로젝트 만들기

스프링 부트와 타임리프(Thymeleaf)를 사용해서 간단한 웹 애플리케이션 만들기

- https://start.spring.io
- web-start와 thymeleaf 추가
- /, /info, /dashboard, /admin 페이지와 핸들러 만들기

타임리프

- xmlns:th="http://www.thymeleaf.org" 네임스페이스를 html 태그에 추가.
- th:text="${message}" 사용해서 Model에 들어있는 값 출력 가능.

현재 문제

- 로그인 할 방법이 없음
- 현재 사용자를 알아낼 방법이 없음

3. 스프링 시큐리티 연동

스프링 시큐리티 의존성 추가하기

- 스프링 부트 도움 받아 추가하기
    - 스타터(Starter) 사용
    - 버전 생략 - 스프링 부트의 의존성 관리 기능 사용

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

스프링 시큐리티 의존성을 추가하고 나면

- 모든 요청은 인증을 필요로 합니다.
- 기본 유저가 생성됩니다.
    - Username: user
    - Password: 콘솔에 출력된 문자열 확인

```
2019-07-24 11:13:41.245  INFO 10848 --- [           main]
.s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: 114284e0-656a-4fdf-b623-9b552a85b6c8
...
```

해결된 문제

- 인증을 할 수 있다.
- 현재 사용자 정보를 알 수 있다.

새로운 문제

- 인증없이 접근 가능한 URL을 설정하고 싶다.
- 이 애플리케이션을 사용할 수 있는 유저 계정이 그럼 하나 뿐인가?
- 비밀번호가 로그에 남는다고?

4. 스프링 시큐리티 설정하기

스프링 웹 시큐리티 설정 추가

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

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .mvcMatchers("/", "/info").permitAll()
                .mvcMatchers("/admin").hasRole("ADMIN")
                .anyRequest().authenticated();

        http.formLogin();
        http.httpBasic();
    }
}
```

해결한 문제

- 요청 URL별 인증 설정

남아있는 문제

- 여전히 계정은 하나 뿐.
- ADMIN 계정도 없음.
- 비밀번호도 여전히 로그에 남는다.

5. 스프링 시큐리티 커스터마이징: 인메모리 유저 추가

지금까지 스프링 부트가 만들어 주던 유저 정보는?

- UserDetailsServiceAutoConfiguration
- SecurityProperties

SecurityProperties를 사용해서 기본 유저 정보 변경할 수 있긴 하지만...

SecurityConfig에 다음 설정 추가

```
@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.inMemoryAuthentication()
            .withUser("keesun").password("{noop}123").roles("USER").and()
            .withUser("admin").password("{noop}!@#").roles("ADMIN");
}

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

- 인메모리 사용자 추가
- 로컬 AuthenticationManager를 빈으로 노출

해결한 문제

- 계정 여러개 사용할 수 있음.
- ADMIN 계정도 있음

남아있는 문제

- 비밀번호가 코드에 보인다.
- 데이터베이스에 들어있는 유저 정보를 사용하고 싶다.

6. 스프링 시큐리티 커스터마이징: JPA 연동

JPA와 H2 의존성 추가

- 보다 자세한 내용은 “스프링 데이터 JPA” 강좌를 참고하세요.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
</dependency>
```

Account 클래스

```
@Entity
public class Account {

   @Id @GeneratedValue
   private Integer id;

   @Column(unique = true)
   private String username;

   private String password;

   private String role;
```

AccountRepository 인터페이스

```
public interface AccountRepository extends JpaRepository<Account, Integer> {
   Account findByUsername(String username);
}
```

AccountSerivce 클래스 implements UserDetailsService

```
@Service
public class AccountService implements UserDetailsService {

   @Autowired
   AccountRepository accountRepository;
```

```
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        Account account = accountRepository.findByUsername(username);
        if (account == null) {
            throw new UsernameNotFoundException(username);
        }

        return User.builder()
                .username(account.getUsername())
                .password(account.getPassword())
                .roles(account.getRole())
                .build();
    }
}
```

해결한 문제

- 패스워드가 코드에 보이지 않는다.
- DB에 들어있는 계정 정보를 사용할 수 있다.

새로운 문제

- “{noop}”을 없앨 수는 없을까?
- 테스트는 매번 이렇게 해야 하는건가?

7. 스프링 시큐리티 커스터마이징: PasswordEncoder

비밀번호는 반드시 인코딩해서 저장해야 합니다. 단방향 암호화 알고리듬으로.

- 스프링 시큐리티가 제공하는 PasswordEndoer는 특정한 포맷으로 동작함.
- {id}encodedPassword
- 다양한 해싱 전략의 패스워드를 지원할 수 있다는 장점이 있습니다.

// 비추: 비밀번호가 평문 그대로 저장됩니다.

```
@Bean
public PasswordEncoder passwordEncoder() {
        return NoOpPasswordEncoder.getInstance();
}
```

// 추천: 기본 전략인 bcrypt로 암호화 해서 저장하며 비교할 때는 {id}를 확인해서 다양한 인코딩을 지원합니다.

```
@Bean
public PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
}
```

해결한 문제

- “{noop}”을 없앴다. 비밀번호가 좀 더 안전해졌다.

남아있는 문제

- 테스트는 매번 이렇게 해야 하는건가?

8. 스프링 시큐리티 테스트 1부

https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#test-mockmvc

Spring-Security-Test 의존성 추가

```
<dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
</dependency>
```

- 테스트에서 사용할 기능을 제공하기 때문에 Test 스콥이 적절합니다.

RequestPostProcessor를 사용해서 테스트 하는 방법

- with(user("user"))
- with(anonymous())
- with(user("user").password("123").roles("USER", "ADMIN"))
- 자주 사용하는 user 객체는 리팩토리으로 빼내서 재사용 가능.

애노테이션을 사용하는 방법

- @WithMockUser
- @WithMockUser(roles="ADMIN")
- 커스텀 애노테이션을 만들어 재사용 가능.

9. 스프링 시큐리티 테스트 2부

폼 로그인 / 로그아웃 테스트

- perform(formLogin())
- perform(formLogin().user("admin").password("pass"))
- perform(logout())

응답 유형 확인

- authenticated()
- unauthenticated()

해결한 문제

- 스프링 시큐리티 테스트를 작성할 수 있다.

이제부터가 시작입니다.

- 회원 가입 기능 구현 (이번 강좌에서 다루지 않습니다.)
- 로그인/로그아웃 페이지 커스터마이징 (기본 화면 그대로도 괜찮긴 하지만...)
- HTTP BASIC 인증에 대해서 학습
- 뷰에서 인증 정보 참조하는 방법 학습 (뷰의 종류마다 방법이 다릅니다.)
- OAuth 2
- 메소드 시큐리티

하지만 그전에 지금까지 코딩한 스프링 시큐리티 코드 그 내부가 어떻게 생겼는지 살펴보는 시간을 갖겠습니다.

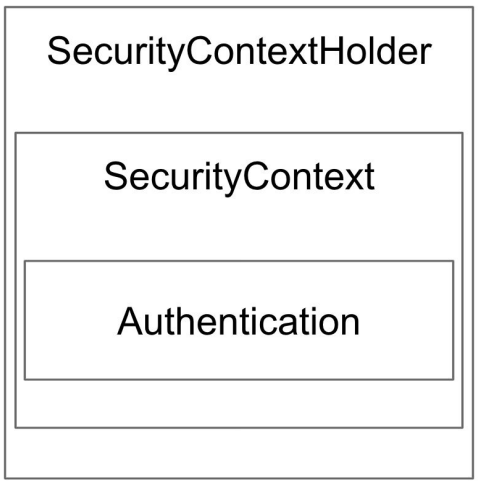
2부 스프링 시큐리티: 아키텍처

10. SecurityContextHolder와 Authentication

https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#core-components

SecurityContextHolder
- SecurityContext 제공, 기본적으로 ThreadLocal을 사용한다.

SecuuryContext
- Authentication 제공.



Authentication
- Principal과 GrantAuthority 제공.

Principal
- “누구"에 해당하는 정보.
- **UserDetailsService에서 리턴한 그 객체.**
- 객체는 UserDetails 타입.

GrantAuthority:
- “ROLE_USER”, “ROLE_ADMIN”등 Principal이 가지고 있는 “권한”을 나타낸다.
- 인증 이후, 인가 및 권한 확인할 때 이 정보를 참조한다.

UserDetails
- 애플리케이션이 가지고 있는 유저 정보와 스프링 시큐리티가 사용하는 Authentication 객체 사이의 어댑터.

UserDetailsService
- 유저 정보를 UserDetails 타입으로 가져오는 DAO (Data Access Object) 인터페이스.

- 구현은 마음대로! (우리는 스프링 데이터 JPA를 사용했습니다.)

11. AuthenticationManager와 Authentication

스프링 시큐리티에서 인증(Authentication)은 AuthenticationManager가 한다.

```
Authentication authenticate(Authentication authentication) throws AuthenticationException;
```

- 인자로 받은 Authentication이 유효한 인증인지 확인하고 Authentication 객체를 리턴한다.
- 인증을 확인하는 과정에서 비활성 계정, 잘못된 비번, 잠긴 계정 등의 에러를 던질 수 있다.

인자로 받은 Authentication

- 사용자가 입력한 인증에 필요한 정보(username, password)로 만든 객체. (폼 인증인 경우)
- Authentication
    - Principal: “keesun”
    - Credentials: “123”

유효한 인증인지 확인

- 사용자가 입력한 password가 UserDetailsService를 통해 읽어온 UserDetails 객체에 들어있는 password와 일치하는지 확인
- 해당 사용자 계정이 잠겨 있진 않은지, 비활성 계정은 아닌지 등 확인

Authentication 객체를 리턴

- Authentication
    - Principal: UserDetailsService에서 리턴한 그 객체 (User)
    - Credentials:
    - GrantedAuthorities

12. ThreadLocal

Java.lang 패키지에서 제공하는 쓰레드 범위 변수. 즉, 쓰레드 수준의 데이터 저장소.

- 같은 쓰레드 내에서만 공유.
- 따라서 같은 쓰레드라면 해당 데이터를 메소드 매개변수로 넘겨줄 필요 없음.
- SecurityContextHolder의 기본 전략.

```
public class AccountContext {

    private static final ThreadLocal<Account> ACCOUNT_THREAD_LOCAL
            = new ThreadLocal<>();

    public static void setAccount(Account account) {
        ACCOUNT_THREAD_LOCAL.set(account);
    }

    public static Account getAccount() {
        return ACCOUNT_THREAD_LOCAL.get();
    }

}
```

13. Authencation과 SecurityContextHodler

AuthenticationManager가 인증을 마친 뒤 리턴 받은 Authentication 객체의 행방은?

UsernamePasswordAuthenticationFilter

- 폼 인증을 처리하는 시큐리티 필터
- 인증된 Authentication 객체를 SecurityContextHolder에 넣어주는 필터
- SecurityContextHolder.getContext().setAuthentication(authentication)

SecurityContextPersisenceFilter

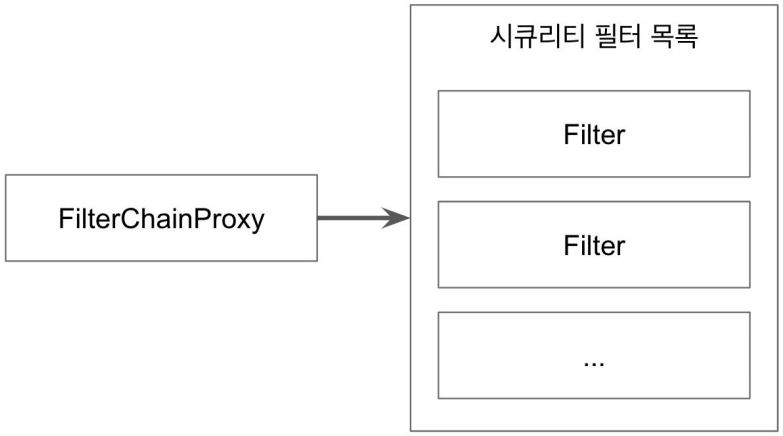
- SecurityContext를 HTTP session에 캐시(기본 전략)하여 여러 요청에서 Authentication을 공유할 수 있 공유하는 필터.
- SecurityContextRepository를 교체하여 세션을 HTTP session이 아닌 다른 곳에 저장하는 것도 가능하다.

14. 스프링 시큐리티 Filter와 FilterChainProxy

스프링 시큐리티가 제공하는 필터들

1. WebAsyncManagerIntergrationFilter
2. **SecurityContextPersistenceFilter**
3. HeaderWriterFilter
4. CsrfFilter
5. LogoutFilter
6. **UsernamePasswordAuthenticationFilter**
7. DefaultLoginPageGeneratingFilter
8. DefaultLogoutPageGeneratingFilter
9. BasicAuthenticationFilter
10. RequestCacheAwareFtiler
11. SecurityContextHolderAwareReqeustFilter
12. AnonymouseAuthenticationFilter
13. SessionManagementFilter
14. ExeptionTranslationFilter
15. FilterSecurityInterceptor

이 모든 필터는 **FilterChainProxy**가 호출한다.



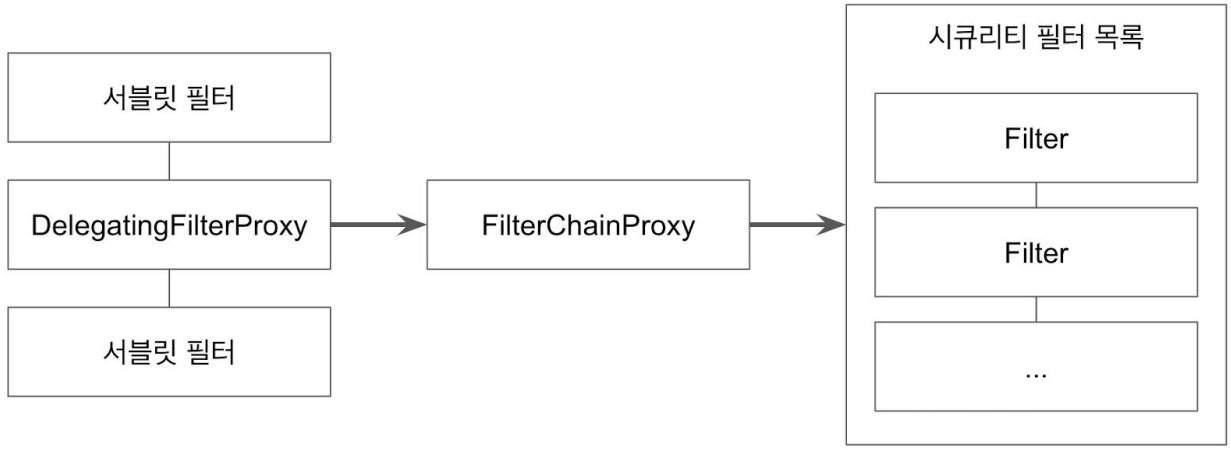
15. DelegatingFilterProxy와 FilterChainProxy

DelegatingFilterProxy

- 일반적인 서블릿 필터.
- 서블릿 필터 처리를 스프링에 들어있는 빈으로 위임하고 싶을 때 사용하는 서블릿 필터.
- 타겟 빈 이름을 설정한다.
- 스프링 부트 없이 스프링 시큐리티 설정할 때는 AbstractSecurityWebApplicationInitializer를 사용해서 등록.
- 스프링 부트를 사용할 때는 자동으로 등록 된다. (SecurityFilterAutoConfiguration)

FilterChainProxy

- 보통 “springSecurityFilterChain” 이라는 이름의 빈으로 등록된다.



16. AccessDecisionManager 1부

Access Control 결정을 내리는 인터페이스로, 구현체 3가지를 기본으로 제공한다.

- **AffirmativeBased**: 여러 Voter중에 한명이라도 허용하면 허용. 기본 전략.
- ConsensusBased: 다수결
- UnanimousBased: 만장일치

AccessDecisionVoter

- 해당 Authentication이 특정한 Object에 접근할 때 필요한 ConfigAttributes를 만족하는지 확인한다.
- **WebExpressionVoter**: 웹 시큐리티에서 사용하는 기본 구현체, ROLE_Xxxx가 매치하는지 확인.
- RoleHierarchyVoter: 계층형 ROLE 지원. ADMIN > MANAGER > USER
- ...

17. AccessDecisionManager 2부

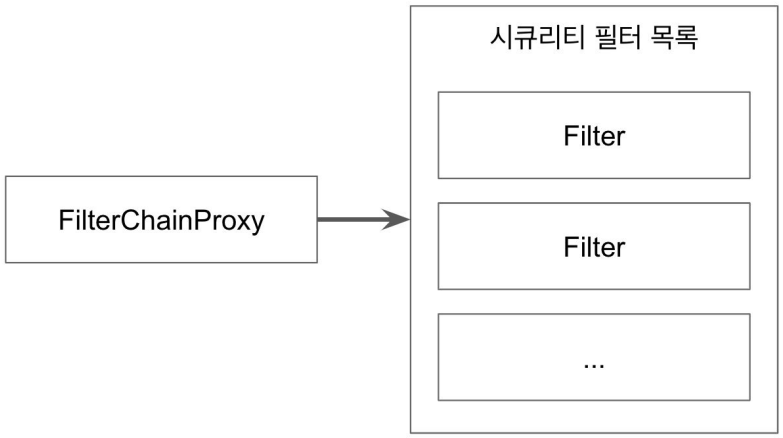
AccessDecisionManager 또는 Voter를 커스터마이징 하는 방법

계층형 ROLE 설정

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

    public SecurityExpressionHandler expressionHandler() {
        RoleHierarchyImpl roleHierarchy = new RoleHierarchyImpl();
        roleHierarchy.setHierarchy("ROLE_ADMIN > ROLE_USER");

        DefaultWebSecurityExpressionHandler handler = new
DefaultWebSecurityExpressionHandler();
        handler.setRoleHierarchy(roleHierarchy);

        return handler;
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
                .mvcMatchers("/", "/info", "/account/**").permitAll()
                .mvcMatchers("/admin").hasRole("ADMIN")
                .mvcMatchers("/user").hasRole("USER")
                .anyRequest().authenticated()
                .expressionHandler(expressionHandler());
        http.formLogin();
        http.httpBasic();
    }

}
```

18. FilterSecurityInterceptor

AccessDecisionManager를 사용하여 Access Control 또는 예외 처리 하는 필터.
대부분의 경우 FilterChainProxy에 제일 마지막 필터로 들어있다.



19. ExceptionTranslationFilter

필터 체인에서 발생하는 AccessDeniedException과 AuthenticationException을 처리하는 필터

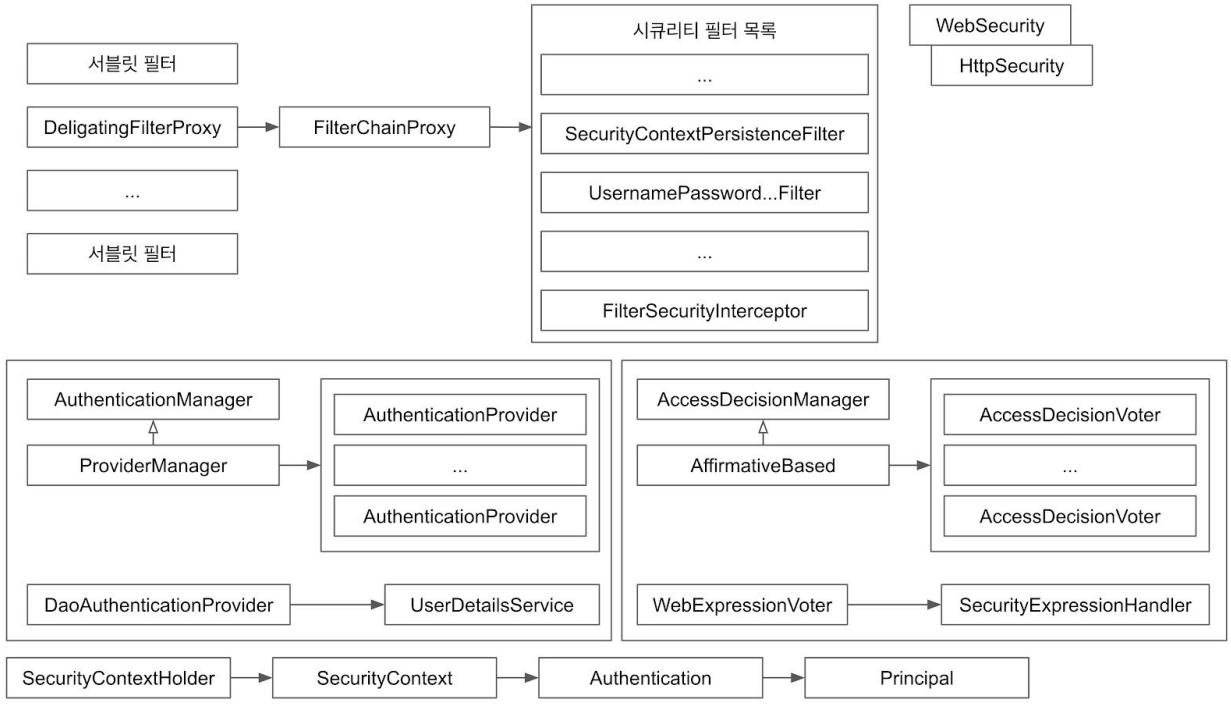
AuthenticationException 발생 시

- AuthenticationEntryPoint 실행
- AbstractSecurityInterceptor 하위 클래스(예, FilterSecurityInterceptor)에서 발생하는 예외만 처리.
- 그렇다면 UsernamePasswordAuthenticationFilter에서 발생한 인증 에러는?

AccessDeniedException 발생 시

- 익명 사용자라면 AuthenticationEntryPoint 실행
- 익명 사용자가 아니면 AccessDeniedHandler에게 위임

20. 스프링 시큐리티 아키텍처 정리



참고

- https://spring.io/guides/topicals/spring-security-architecture
- https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#overall-architecture

3부 웹 애플리케이션 시큐리티

21. 스프링 시큐리티 ignoring() 1부

WebSecurity의 ignoring()을 사용해서 시큐리티 필터 적용을 제외할 요청을 설정할 수 있다.

```
@Override
public void configure(WebSecurity web) throws Exception {
   web.ignoring().requestMatchers(PathRequest.toStaticResources().atCommonLocations());
}
```

- 스프링 부트가 제공하는 PathRequest를 사용해서 정적 자원 요청을 스프링 시큐리티 필터를 적용하지 않도록 설정.

22. 스프링 시큐리티 ignoring() 2부

```
http.authorizeRequests()
.requestMatchers(PathRequest.toStaticResources().atCommonLocations()).permitAll()
```

이런 설정으로도 같은 결과를 볼 수는 있지만 스프링 시큐리티 필터가 적용된다는 차이가 있다.

- 동적 리소스는 http.authorizeRequests()에서 처리하는 것을 권장합니다.
- 정적 리소스는 WebSecurity.ignore()를 권장하며 예외적인 정적 자원 (인증이 필요한 정적자원이 있는 경우)는 http.authorizeRequests()를 사용할 수 있습니다.

23. Async 웹 MVC를 지원하는 필터: WebAsyncManagerIntegrationFilter

스프링 MVC의 Async 기능(핸들러에서 Callable을 리턴할 수 있는 기능)을 사용할 때에도 SecurityContext를 공유하도록 도와주는 필터.

- PreProcess: SecurityContext를 설정한다.
- Callable: 비록 다른 쓰레드지만 그 안에서는 동일한 SecurityContext를 참조할 수 있다.
- PostProcess: SecurityContext를 정리(clean up)한다.

24. 스프링 시큐리티와 @Async

@Async를 사용한 서비스를 호출하는 경우

- 쓰레드가 다르기 때문에 SecurityContext를 공유받지 못한다.

```
SecurityContextHolder.setStrategyName(SecurityContextHolder.MODE_INHERITABLETHREADLOCAL);
```

- SecurityContext를 자식 쓰레드에도 공유하는 전략.
- @Async를 처리하는 쓰레드에서도 SecurityContext를 공유받을 수 있다.

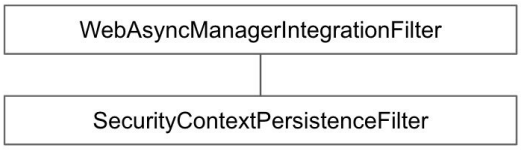
참고

- https://docs.oracle.com/javase/7/docs/api/java/lang/InheritableThreadLocal.html

25. SecurityContext 영속화 필터: SecurityContextPersistenceFilter

SecurityContextRepository를 사용해서 기존의 SecurityContext를 읽어오거나 초기화 한다.

- 기본으로 사용하는 전략은 HTTP Session을 사용한다.
- Spring-Session과 연동하여 세션 클러스터를 구현할 수 있다. (이 강좌에서는 다루지 않습니다.)

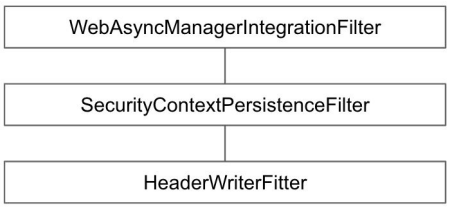


26. 시큐리티 관련 헤더 추가하는 필터: HeaderWriterFilter

응답 헤더에 시큐리티 관련 헤더를 추가해주는 필터

- XContentTypeOptionsHeaderWriter: 마임 타입 스니핑 방어.
- XXssProtectionHeaderWriter: 브라우저에 내장된 XSS 필터 적용.
- CacheControlHeadersWriter: 캐시 히스토리 취약점 방어.
- HstsHeaderWriter: HTTPS로만 소통하도록 강제.
- XFrameOptionsHeaderWriter: clickjacking 방어.

Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Content-Language: en-US
Content-Type: text/html;charset=UTF-8
Date: Sun, 04 Aug 2019 16:25:10 GMT
Expires: 0
Pragma: no-cache
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block



참고

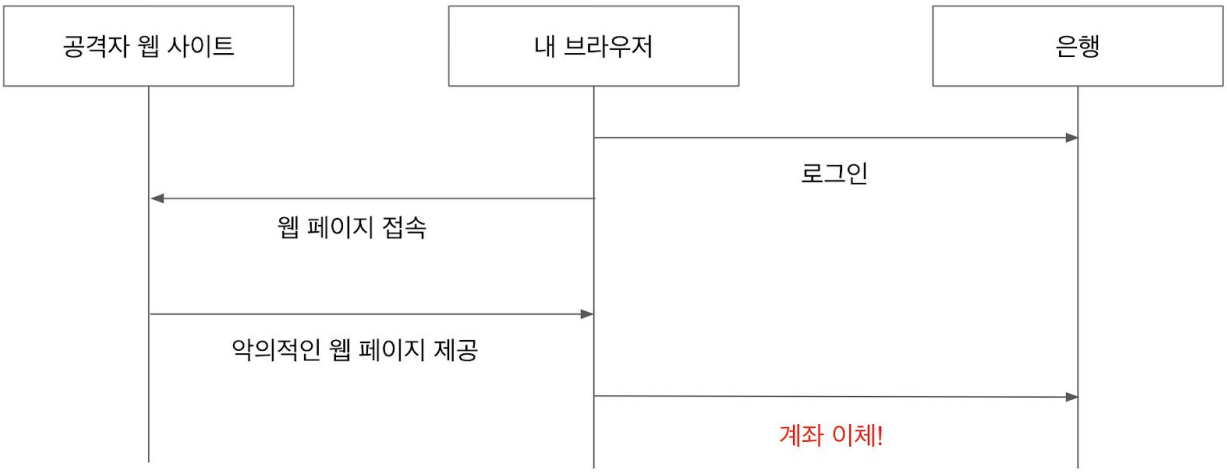
- X-Content-Type-Options:
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options
- Cache-Control:
  - https://www.owasp.org/index.php/Testing_for_Browser_cache_weakness_(OTG-AUTHN-006)
- X-XSS-Protection
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection
  - https://github.com/naver/lucy-xss-filter
- HSTS
  - https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html
- X-Frame-Options

    - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
- https://cyberx.tistory.com/171

27. CSRF 어택 방지 필터: CsrfFilter

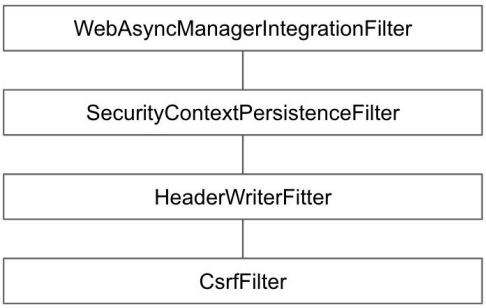
CSRF 어택 방지 필터

- 인증된 유저의 계정을 사용해 악의적인 변경 요청을 만들어 보내는 기법.
- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
- https://namu.wiki/w/CSRF
- CORS를 사용할 때 특히 주의 해야 함.
    - 타 도메인에서 보내오는 요청을 허용하기 때문에...
    - https://en.wikipedia.org/wiki/Cross-origin_resource_sharing



의도한 사용자만 리소스를 변경할 수 있도록 허용하는 필터

- CSRF 토큰을 사용하여 방지.



28. CSRF 토큰 사용 예제

JSP에서 스프링 MVC가 제공하는 <form:form> 태그 또는 타임리프 2.1+ 버전을 사용할 때 폼에 CRSF 히든 필드가 기본으로 생성 됨.

Signup.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>SignUp</title>
</head>
<body>
    <h1>Sign Up</h1>
    <form action="/signup" th:action="@{/signup}" th:object="${account}" method="post">
        <p>Username: <input type="text" th:field="*{username}" /></p>
        <p>Password: <input type="text" th:field="*{password}" /></p>
        <p><input type="submit" value="Submit" /></p>
    </form>
</body>
</html>
```

SignUpController

```
package me.whiteship.demospringsecurityform.account;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class SignUpController {

    @Autowired
    AccountService accountService;

    @GetMapping("/signup")
    public String signUpForm(Model model) {
        model.addAttribute("account", new Account());
        return "signup";
    }
```

```
    @PostMapping("/signup")
    public String processSignUp(@ModelAttribute Account account) {
        account.setRole("USER");
        accountService.createNew(account);
        return "redirect:/";
    }

}
```

SignUpControllerTest

```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class SignUpControllerTest {

    @Autowired
    MockMvc mockMvc;

    @Test
    public void signUpForm() throws Exception {
        mockMvc.perform(get("/signup"))
                .andDo(print())
                .andExpect(content().string(containsString("_csrf")));
    }

    @Test
    public void procesSignUp() throws Exception {
        mockMvc.perform(post("/signup")
            .param("username", "keesun")
            .param("password", "123")
            .with(csrf()))
                .andExpect(status().is3xxRedirection());
    }
}
```

29. 로그아웃 처리 필터: LogoutFilter

여러 LogoutHandler를 사용하여 로그아웃시 필요한 처리를 하며 이후에는 LogoutSuccessHandler를 사용하여 로그아웃 후처리를 한다.

LogoutHandler

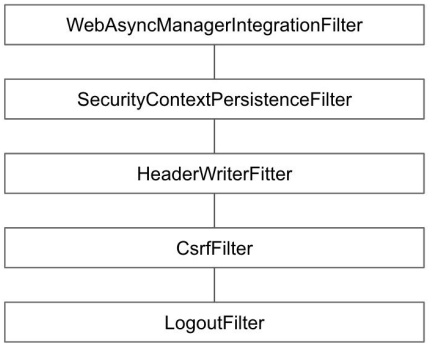
- CsrfLogoutHandler
- SecurityContextLogoutHandler

LogoutSuccessHandler

- SimplUrlLogoutSuccessHandler

로그아웃 필터 설정

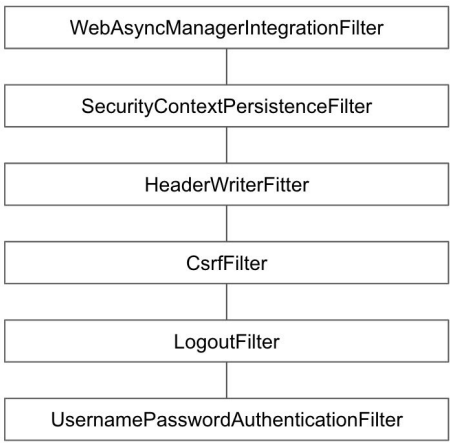
```
http.logout()
        .logoutUrl("/logout")
        .logoutSuccessUrl("/")
        .logoutRequestMatcher()
        .invalidateHttpSession(true)
        .deleteCookies()
        .addLogoutHandler()
        .logoutSuccessHandler();
```



30. 폼 인증 처리 필터: UsernamePasswordAuthenticationFilter

폼 로그인을 처리하는 인증 필터

- 사용자가 폼에 입력한 username과 password로 Authentcation을 만들고 AuthenticationManager를 사용하여 인증을 시도한다.
- AuthenticationManager (ProviderManager)는 여러 AuthenticationProvider를 사용하여 인증을 시도하는데, 그 중에 DaoAuthenticationProvider는 UserDetailsServivce를 사용하여 UserDetails 정보를 가져와 사용자가 입력한 password와 비교한다.



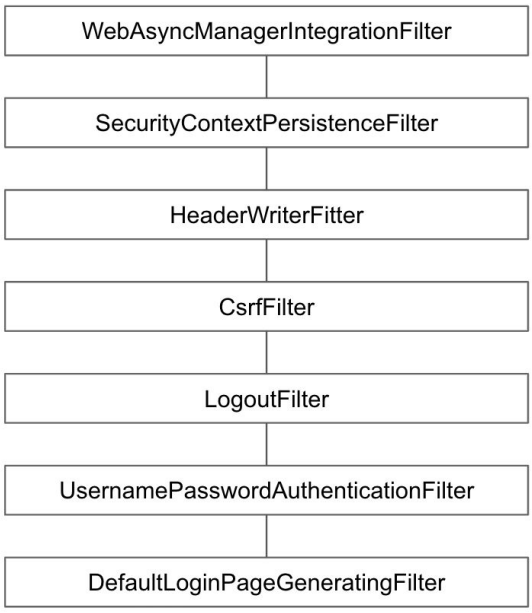
31. DefaultLoginPageGeneratingFilter

기본 로그인 폼 페이지를 생성해주는 필터

- GET /login 요청을 처리하는 필터.

로그인 폼 커스터마이징

```
http.formLogin()
        .usernameParameter("my-username")
        .passwordParameter("my-password");
```



32. 로그인/로그아웃 폼 커스터마이징

https://docs.spring.io/spring-security/site/docs/current/reference/html5/#jc-form

Signin.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>SignIn</title>
</head>
<body>
    <h1>Sign In</h1>
    <div th:if="${param.error}">
        <div class="alert alert-danger">
            Invalid username or password.
        </div>
    </div>
    <form action="/signin" th:action="@{/signin}" method="post">
        <p>Username: <input type="text" name="username" /></p>
        <p>Password: <input type="password" name="password" /></p>
        <p><input type="submit" value="SignIn" /></p>
    </form>
</body>
</html>
```

Logout.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>SignIn</title>
</head>
<body>
    <h1>Logout</h1>
    <form action="/logout" th:action="@{/logout}" method="post">
        <p><input type="submit" value="Logout" /></p>
    </form>
</body>
</html>
```

시큐리티 설정

```
http.formLogin()
```

```
        .loginPage("/signin")
        .permitAll();
```

33. Basic 인증 처리 필터: BasicAuthenticationFilter

Basic 인증이란?

- https://tools.ietf.org/html/rfc7617
- 요청 헤더에 username와 password를 실어 보내면 브라우저 또는 서버가 그 값을 읽어서 인증하는 방식. 예) Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l (keesun:123을 BASE 64)
- 보통, 브라우저 기반 요청이 클라이언트의 요청을 처리할 때 자주 사용.
- 보안에 취약하기 때문에 반드시 HTTPS를 사용할 것을 권장.

34. 요청 캐시 필터: RequestCacheAwareFilter

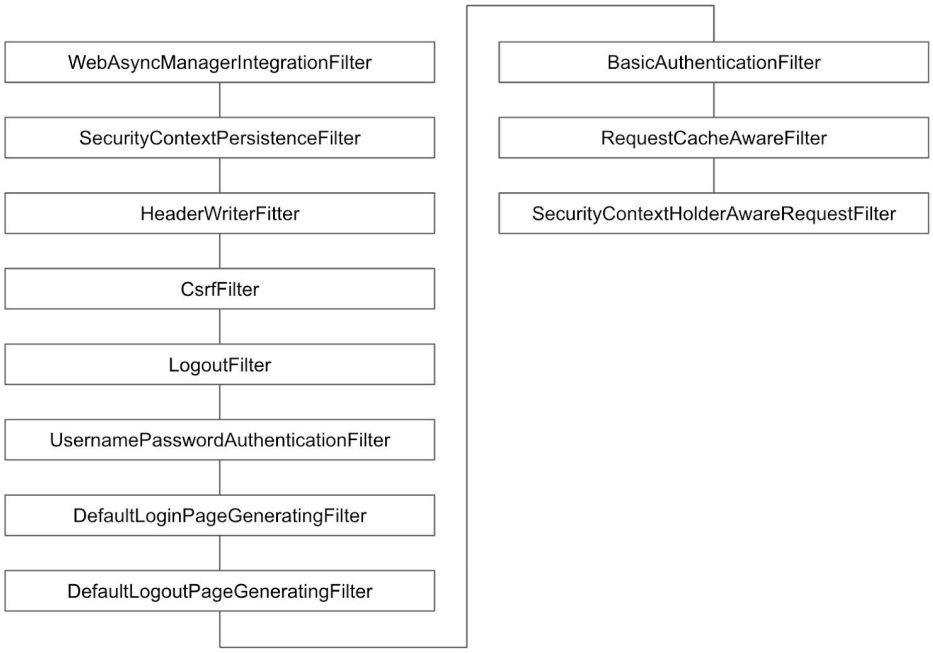
현재 요청과 관련 있는 캐시된 요청이 있는지 찾아서 적용하는 필터.

- 캐시된 요청이 없다면, 현재 요청 처리
- 캐시된 요청이 있다면, 해당 캐시된 요청 처리

35. 시큐리티 관련 서블릿 스팩 구현 필터: SecurityContextHolderAwareRequestFilter

시큐리티 관련 서블릿 API를 구현해주는 필터

- HttpServletRequest#authenticate(HttpServletResponse)
- HttpServletRequest#login(String, String)
- HttpServletRequest#logout()
- AsyncContext#start(Runnable)



36. 익명 인증 필터: AnonymousAuthenticationFilter

https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#anonymous

현재 SecurityContext에 Authentication이 null이면 “익명 Authentication”을 만들어 넣어주고, null이 아니면 아무일도 하지 않는다.

기본으로 만들어 사용할 “익명 Authentication” 객체를 설정할 수 있다.

```
http.anonymous()
        .principal()
        .authorities()
        .key()
```

참고

- https://en.wikipedia.org/wiki/Null_object_pattern

37. 세션 관리 필터: SessionManagementFilter

https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#session-mgmt

세션 변조 방지 전략 설정: sessionFixation

- 세션 변조: https://www.owasp.org/index.php/Session_fixation
- none
- newSession
- migrateSession (서블릿 3.0- 컨테이너 사용시 기본값)
- **changeSessionId** (서브릿 3.1+ 컨테이너 사용시 기본값)
- https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#nsa-session-management-attributes

유효하지 않은 세션을 리다이렉트 시킬 URL 설정

- invalidSessionUrl

동시성 제어: maximumSessions

- 추가 로그인을 막을지 여부 설정 (기본값, false)
- https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#nsa-concurrency-control

세션 생성 전략: sessionCreationPolicy

- **IF_REQUIRED**
- NEVER
- STATELESS
- ALWAYS

38. 인증/인가 예외 처리 필터: ExceptionTranslationFilter

https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#exception-translation-filter

인증, 인가 에러 처리를 담당하는 필터

- AuthenticationEntryPoint
- AccessDeniedHandler

Access-denied.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Access Denied</title>
</head>
<body>
   <h1><span th:text="${name}">Name</span>, you can't access to the resource.</h1>
</body>
</html>
```

ExceptionHandler 설정

```
        http.exceptionHandling()
                .accessDeniedHandler((request, response, accessDeniedException) -> {
                    UserDetails principal = (UserDetails)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
                    String username = principal.getUsername();
                    String servletPath = request.getServletPath();
                    System.out.println(username + " is denied to access to " + servletPath);
                    response.sendRedirect("/access-denied");
                });
```

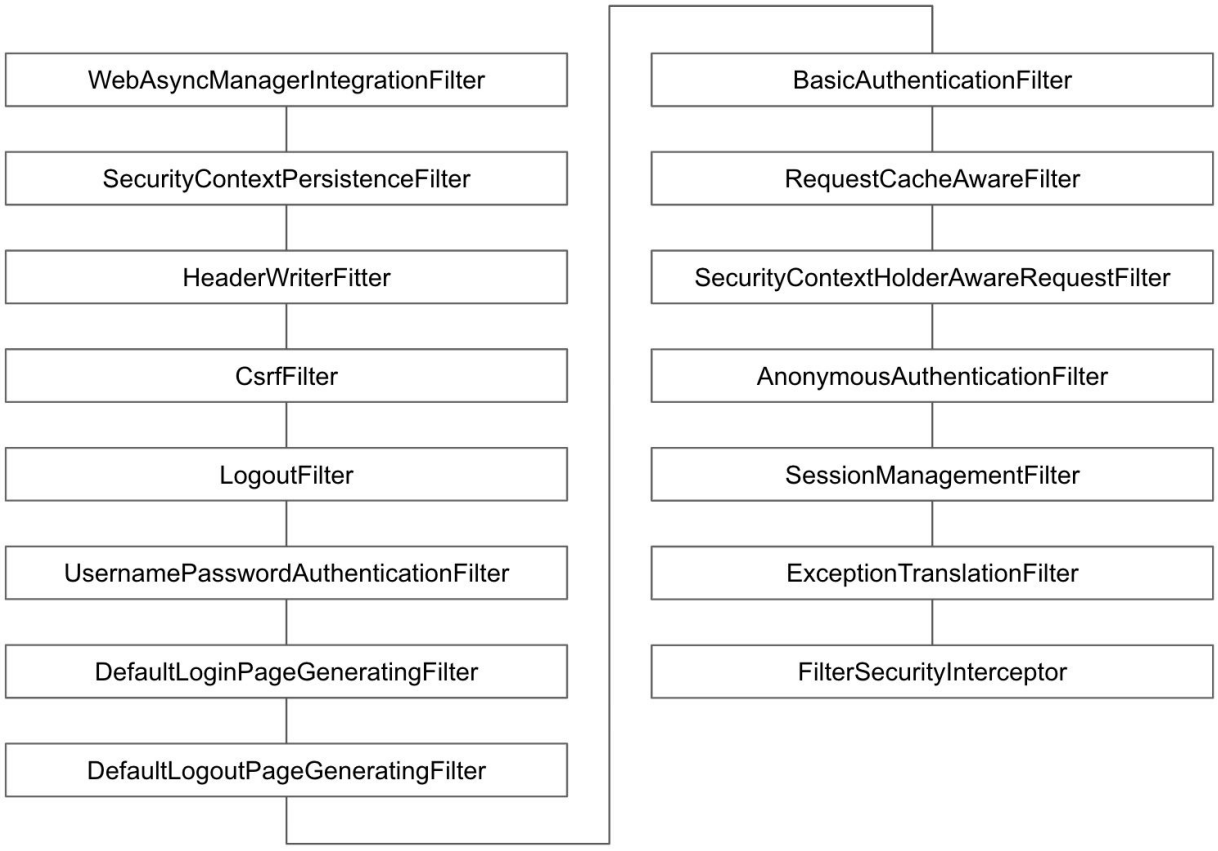
39. 인가 처리 필터: FilterSecurityInterceptor

https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#filter-security-interceptor

HTTP 리소스 시큐리티 처리를 담당하는 필터. AccessDecisionManager를 사용하여 인가를 처리한다.

HTTP 리로스 시큐리티 설정

```
http.authorizeRequests()
        .mvcMatchers("/", "/info", "/account/**", "/signup").permitAll()
        .mvcMatchers("/admin").hasAuthority("ROLE_ADMIN")
        .mvcMatchers("/user").hasRole("USER")
        .anyRequest().authenticated()
        .expressionHandler(expressionHandler());
```



40. 토큰 기반 인증 필터 : RememberMeAuthenticationFilter

세션이 사라지거나 만료가 되더라도 쿠키 또는 DB를 사용하여 저장된 토큰 기반으로 인증을 지원하는 필터

RememberMe 설정

```
http.rememberMe()
        .userDetailsService(accountService)
        .key("remember-me-sample");
```

쿠키 플러그인

- https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnpilhplaeedifhccceomclgfbg?hl=en

41. 커스텀 필터 추가하기

LoggingFilter.java

```
public class LoggingFilter extends GenericFilterBean {

    private Logger logger = LoggerFactory.getLogger(this.getClass());

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {
        StopWatch stopWatch = new StopWatch();
        stopWatch.start(((HttpServletRequest)request).getRequestURI());
        chain.doFilter(request, response);
        stopWatch.stop();
        logger.info(stopWatch.prettyPrint());
    }
}
```

커스텀 필터 추가 설정

```
http.addFilterAfter(new LoggingFilter(), UsernamePasswordAuthenticationFilter.class);
```

5부 스프링 시큐리티 그밖에

https://github.com/thymeleaf/thymeleaf-extras-springsecurity/blob/3.0-master/README.markdown

42. 타임리프 스프링 시큐리티 확장팩

의존성 추가

```
<dependency>
        <groupId>org.thymeleaf.extras</groupId>
        <artifactId>thymeleaf-extras-springsecurity5</artifactId>
</dependency>
```

Authentication과 Authorization참조

```
<div th:if="${#authorization.expr('isAuthenticated()')}">
   <h2 th:text="${#authentication.name}"></h2>
   <a href="/logout" th:href="@{/logout}">Logout</a>
</div>
<div th:unless="${#authorization.expr('isAuthenticated()')}">
   <a href="/login" th:href="@{/login}">Login</a>
</div>
```

43. sec 네임스페이스

Sec 네임스페이스 등록

```
xmlns:sec="http://www.thymeleaf.org/extras/spring-security"
```

Sec 네임스페이스 사용하기

```
<div sec:authorize="isAuthenticated()">
    <h2 sec:authentication="name">Name</h2>
    <a href="/logout" th:href="@{/logout}">Logout</a>
</div>
<div sec:authorize="!isAuthenticated()">
    <a href="/login" th:href="@{/login}">Login</a>
</div>
```

44. 메소드 시큐리티

https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#jc-method
https://www.baeldung.com/spring-security-method-security

@EnableGlobalMethodSecurity

```
@EnableGlobalMethodSecurity(jsr250Enabled = true, prePostEnabled = true,
securedEnabled = true)
```

@Secured와 @RollAllowed

- 메소드 호출 이전에 권한을 확인한다.
- 스프링 EL을 사용하지 못한다.

@PreAuthorize와 @PostAuthorize

- 메소드 호출 이전 @있다.

MethodSecurityConfig.java

```
@Configuration
@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true,
jsr250Enabled = true)
public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {

    @Override
    protected AccessDecisionManager accessDecisionManager() {
        RoleHierarchyImpl roleHierarchy = new RoleHierarchyImpl();
        roleHierarchy.setHierarchy("ROLE_ADMIN > ROLE_USER");
        AffirmativeBased accessDecisionManager = (AffirmativeBased)
super.accessDecisionManager();
        accessDecisionManager.getDecisionVoters().add(new
RoleHierarchyVoter(roleHierarchy));
        return accessDecisionManager;
    }
}
```

45. @AuthenticationPrincipal

https://docs.spring.io/spring-security/site/docs/5.1.5.RELEASE/reference/htmlsingle/#mvc-authentication-principal

웹 MVC 핸들러 아규먼트로 Principal 객체를 받을 수 있다.

커스텀 유저 클래스 구현하기

```
public class UserAccount extends User {

    private Account account;

    public UserAccount(Account account) {
        super(account.getUsername(), account.getPassword(), List.of(new
SimpleGrantedAuthority("ROLE_" + account.getRole())));
        this.account = account;
    }

    public Account getAccount() {
        return account;
    }
}
```

AccountService 수정

```
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        Account account = accountRepository.findByUsername(username);
        if (account == null) {
            throw new UsernameNotFoundException(username);
        }

        return new UserAccount(account);
    }
```

@AuthenticationPrincipal 애노테이션 적용 예제 1

```
@AuthenticationPrincipal UserAccount userAccount
```

- UserDetailsService 구현체에서 리턴하는 객체를 매개변수로 받을 수 있다.
- 그 안에 들어있는 Account객체를 getter를 통해 참조할 수 있다.

@AuthenticationPrincipal 애노테이션 적용 예제 2

```
@AuthenticationPrincipal(expression = "#this == 'anonymousUser' ? null : account") Account account
```

- 익명 Authentication인 경우 (“anonymousUser”)에는 null 아닌 경우에는 account 필드를 사용한다.
- Account를 바로 참조할 수 있다.

@AuthenticationPrincipal 애노테이션 적용 예제 3

```
@CurrentUser Account account
```

- @AP를 메타 애노테이션으로 사용하여 커스텀 애노테이션을 만들어 쓸 수 있다.

@CurrentUser

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.PARAMETER)
@AuthenticationPrincipal(expression = "#this == 'anonymousUser' ? null : account")
public @interface CurrentUser {
}
```

46. 스프링 데이터 연동

https://docs.spring.io/spring-security/site/docs/current/reference/html5/#data

@Query 애노테이션에서 SpEL로 principal 참조할 수 있는 기능 제공.

스프링 시큐리티 데이터 의존성 추가

```
<dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-data</artifactId>
        <version>${spring-security.version}</version>
</dependency>
```

@Query에서 principal 사용하기

```
@Query("select b from Book b where b.author.id = ?#{principal.account.id}")
List<Book> findCurrentUserBooks();
```

타임리프 리스트 참조

```
<tr th:each="book : ${books}">
    <td><span th:text="${book.title}"> Title </span></td>
</tr>
```

47. 스프링 시큐리티 마무리

이번 강좌에서 다룬 내용

- 스프링 시큐리티 아키텍처
- 폼 기반 웹 애플리케이션 인증 기능
- 로그인/로그아웃 페이지 커스터마이징
- 여러 인증 관련 응답 헤더
- CSRF
- 세션 관리
- 타임리프 연동
- 스프링 데이터 연동

다루지 않은 내용

- ACL
- WebSocket
- OAuth 2.0
- Reactive