Spring Security

4 Spring Security 적용하기

Spring Security Config 설정하기

필터 Off

```
http.httpBasic().disable();
```

Spring Security 의 특정 필터를 disable하여 동작하지 않게 합니다.

사용하지 않을 필터를 명시적으로 disable하는 것도 좋은 방법입니다.

로그인 & 로그아웃 페이지 관련 기능

```
// login
http.formLogin()
        .loginPage("/login")
        .defaultSuccessUrl("/")
        .permitAll(); // 모두 허용
```

폼 로그인의 로그인 페이지를 지정하고 로그인에 성공했을때 이동하는 *URL*을 지정합니다.

```
// logout
http.logout()
        .logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
        .logoutSuccessUrl("/");
```

로그아웃 *URL*을 지정하고 로그아웃에 성공했을때 이동하는 *URL*을 지정합니다.

Spring Security Config 설정하기

Url Matchers 관련 기능

antMatchers

```
http.authorizeRequests()
        .antMatchers("/signup").permitAll()
```

“/signup” 요청을 모두에게 허용합니다.

mvcMatchers

```
http.authorizeRequests()
        .mvcMatchers("/signup").permitAll()
```

“/signup”, “/signup/“, “/signup.html” 와 같은 유사 *signup* 요청을 모두에게 허용합니다.

regexMatchers

정규표현식으로 매칭합니다.

requestMatchers

antMatchers, mvcMatchers, regexMatchers는 결국에 requestMatchers로 이루어져있습니다.

명확하게 요청 대상을 지정하는 경우에는 requestMatchers를 사용합니다.

```
PathRequest.toStaticResources().atCommonLocations()
```

Spring Security Config 설정하기

인가 관련 설정

authorizeRequests()

```
http.authorizeRequests()
```

인가를 설정합니다.

permitAll()

```
http.authorizeRequests()
        .antMatchers("/home").permitAll()
```

"/home" 요청을 모두에게 허용합니다.

hasRole()

```
http.authorizeRequests()
        .antMatchers(HttpMethod.POST, "/notice").hasRole("ADMIN")
```

권한을 검증합니다.

authenticated()

```
http.authorizeRequests()
        .anyRequest().authenticated()
```

인증이 되었는지를 검증합니다.

Spring Security Config 설정하기

Ignoring

특정 리소스에 대해서 SpringSecurity자체를 적용하지 않고 싶을 때가 있습니다.

우리 프로젝트에서는 아래의 css와 png 파일은 굳이 인증 없이 외부에 공개되어있습니다.

이럴때는 ignoring을 사용하면 됩니다.

http://localhost:8080/css/signin.css
http://localhost:8080/images/spring-security.png

아래 두개의 코드는 결과적으로는 같은 코드입니다.

하지만 ignoring을 사용한 위의 코드는 permitAll을 사용한 아래 코드와 다르게 아예 SpringSecurity의 대상에 포함되지 않습니다.

즉, 어떤 필터도 실행되지 않기 때문에 위의 코드가 아래 코드보다 성능적으로 우수합니다.

```
  @Override
    public void configure(WebSecurity web) {
        // 정적 리소스 spring security 대상에서 제외
//        web.ignoring().antMatchers("/images/**", "/css/**"); // 아래 코드와 같은 코드입니다.
        web.ignoring().requestMatchers(PathRequest.toStaticResources().atCommonLocations());
    }
```

css, javascript, images, web jars, favicon

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

스프링 시큐리티 테스트

SpringSecurity를 사용하는 프로젝트의 테스트는 SpringSecurity가 없는 프로젝트의 테스트와 조금 다른 부분이 있습니다.
SpringSecurity의 테스트에서는 User가 로그인한 상태를 가정하고 테스트해야 하는 경우가 많습니다.
인증을 받지 않은 상태로 테스트를 하면 SpringSecurity에서 요청 자체를 막기 때문에 테스트가 제대로 동작조차 하지 못합니다.
이런 문제는 프로젝트에 spring-security-test를 사용해서 해결할 수 있습니다.
Spring-security-test를 사용하면 테스트 직전에 Mock User를 인증시켜놓고 테스트를 구동시킬수 있습니다.

SpringSecurityTest 의존성 추가

```
testImplementation 'org.springframework.security:spring-security-test'
```

Test 실행 전 MockMvc에 springSecurity (static 메소드)를 설정합니다.

```
@BeforeEach
public void setUp(@Autowired WebApplicationContext applicationContext) {
    this.mockMvc = MockMvcBuilders.webAppContextSetup(applicationContext)
            .apply(springSecurity())
            .build();
}
```

스프링 시큐리티 테스트

@WithMockUser

Mock(가짜) User를 생성하고 Authentication을 만듭니다.

여기서 User는 org.springframework.security.core.userdetails.User를 말합니다.

멤버변수	예시	설명
roles	USER	권한 (ROLE_ 자동으로 붙음)
authorities	ROLE_USER	권한 (사용하면 roles를 무시함)
username	user123	유저명
password	password123	패스워드
setupBefore	TestExecutionEvent.TEST_METHOD TestExecutionEvent.TEST_EXECUTION	언제 유저가 세팅되는지 정함

내부에서 UserDetails를 직접 구현해서 Custom User를 만들어 사용하는 경우에는

WithMockUser를 사용하면 문제가 발생할 수 있습니다.

WithMockUser는 org.springframework.security.core.userdetails.User를 만들어 주지만

우리가 필요한 User는 Custom User이기 때문입니다. (class cast 에러가 발생할 수 있습니다.)

스프링 시큐리티 테스트

@WithUserDetails

WithMockUser와 마찬가지로 Mock(가짜) User를 생성하고 Authentication을 만듭니다.

WithMockUser와 다른점은

가짜 User를 가져올 때 UserDetailsService의 Bean 이름을 넣어줘서

userDetailsService.loadUserByUsername(String username)을 통해 User를 가져옵니다.

멤버변수	예시	설명
value	user123	가져올 user의 username
userDetailsServiceBeanName	userDetailsService	UserDetailsService를 구현한 Bean의 이름
setupBefore	TestExecutionEvent.TEST_EXECUTION	언제 유저가 세팅되는지 정함

스프링 시큐리티 테스트

@WithAnonymousUser

WithMockUser와 동일하지만 인증된 유저 대신에
익명(Anonymous)유저를 Authentication에서 사용합니다.
익명이기 때문에 멤버변수에 유저와 관련된 값이 없습니다.

멤버변수	예시	설명
setupBefore	TestExecutionEvent.TEST_EXECUTION	언제 유저가 세팅되는지 정함

스프링 시큐리티 테스트

@WithSecurityContext

다른 방식들은 Authentication을 가짜로 만들었다고 한다면

WithSecurityContext는 아예 SecurityContext를 만듭니다.

멤버변수	설명

factory	WithSecurityContextFactory를 Implement한 Class를 넣어줍니다.
	`public interface WithSecurityContextFactory<A extends Annotation> {`
	`  SecurityContext createSecurityContext(A annotation);`
	`}`

setupBefore	언제 유저가 세팅되는지 정함

스프링 시큐리티 테스트

with(user( ))

다른 방식은 어노테이션 기반인 반면에 이 방식은 직접 User를 MockMvc에 주입하는 방법입니다.

WithMockUser와 마찬가지로 유저를 생성해서 Principal에 넣고 Authentication을 생성해줍니다.

org.springframework.security.test.web.servlet.request.user를 사용합니다.

```
mockMvc.perform(get("/admin")
                .with(user(user))) // 유저 추가
```

Custom Filter 만들기

프로젝트에 SpringSecurity를 포함시켜 개발하다 보면 SpringSecurity에서 기본으로 제공하는 필터뿐만 아니라
개발자가 원하는 방식대로 동작하는 필터가 필요할때가 많습니다.
이럴때 우리는 커스텀 필터를 구현하면 됩니다.
커스텀 필터를 구현하기 위해서는 다른 필터와 마찬가지로 Filter Interface를 구현해야 합니다.
그러나 Filter Interface를 직접 구현하게 되면 중복실행 문제가 있어서
대부분의 경우에는 OncePerRequestFilter를 구현하기를 권장합니다.

Custom Filter 만들기

예시1)

1개의 요청이 들어온 시점부터 끝날 때 까지 걸린 시간을 Log로 기록합니다.

Stopwatch로 시작을 기록하고 모든 필터체인과 요청이 끝난뒤에 작업한 시간을 기록합니다.

Ex) StopWatch '/login': running time = 150545041 ns

```
@Slf4j
public class StopwatchFilter extends OncePerRequestFilter {
    protected void doFilterInternal(
            HttpServletRequest request,
            HttpServletResponse response,
            FilterChain filterChain
    ) throws ServletException, IOException {
        StopWatch stopWatch = new StopWatch(request.getServletPath());
        stopWatch.start(); // stop watch 시작
        filterChain.doFilter(request, response);
        stopWatch.stop(); // stop watch 종료
        log.info(stopWatch.shortSummary());
    }
}
```

Custom Filter 만들기

예시2)

개인보안노트서비스는 1명의 유저는 1개의 권한을 갖도록 되어있습니다.

권한이 나뉘어져있다보니 테스트하는데 어려움을 느꼈습니다.

그래서 tester 유저인 경우에는 모든 권한을 한번에 갖는 필터를 만들기로 하였습니다.

```
public class TesterAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) {
        Authentication authentication = super.attemptAuthentication(request, response);
        User user = (User) authentication.getPrincipal();
        if (user.getUsername().startsWith("tester")) {
            return new UsernamePasswordAuthenticationToken(
                    user,
                    null,
                    Stream.of("ROLE_ADMIN", "ROLE_USER")
                            .map(authority -> (GrantedAuthority) () -> authority)
                            .collect(Collectors.toList())
            );
        }
        return authentication;
```

Custom Filter 만들기

필터가 구현이 되었다면 필터를 추가해야합니다.

필터를 추가하는 작업은 SpringSecurityConfig(WebSecurityConfigurerAdapter)에서 할수 있습니다.

이때 순서를 정해야 하는데 이미등록된 필터를 하나 정하고 그 앞에 위치할 것인지 뒤에 위치할 것인지 정하면됩니다.

http.addFilterBefore(새로운필터, 이미존재하고앞에있는필터)

http.addFilterAfter(새로운필터, 이미존재하고뒤에있는필터)

```
// stopwath filter
http.addFilterBefore(
        new StopwatchFilter(),
        WebAsyncManagerIntegrationFilter.class
);
http.addFilterBefore(
        new TesterAuthenticationFilter(this.authenticationManager()),
        UsernamePasswordAuthenticationFilter.class
);
```