Sping Security

Spring security 를 활용한 Oauth 2 구현

목표

Oauth 2 + Spring security를 활용하여
책임이 분리된 안전한 로그인 기능을 **구현**하는 것

개인 공부를 하면서 정리한 내용으로서 잘못된 부분이 존재할 수 있음
질문은 환영이되 원하는 답변을 듣지 못할 수 있음...

Spring security

스프링에서 지원하는 프레임 워크
인증, 인가 기능 지원

HTTP 요청 → WAS → 필터 → 서블릿 → 스프링 인터셉터 → 컨트롤러

필터에 접근하여 인증 인가 처리가 가능

Spring security

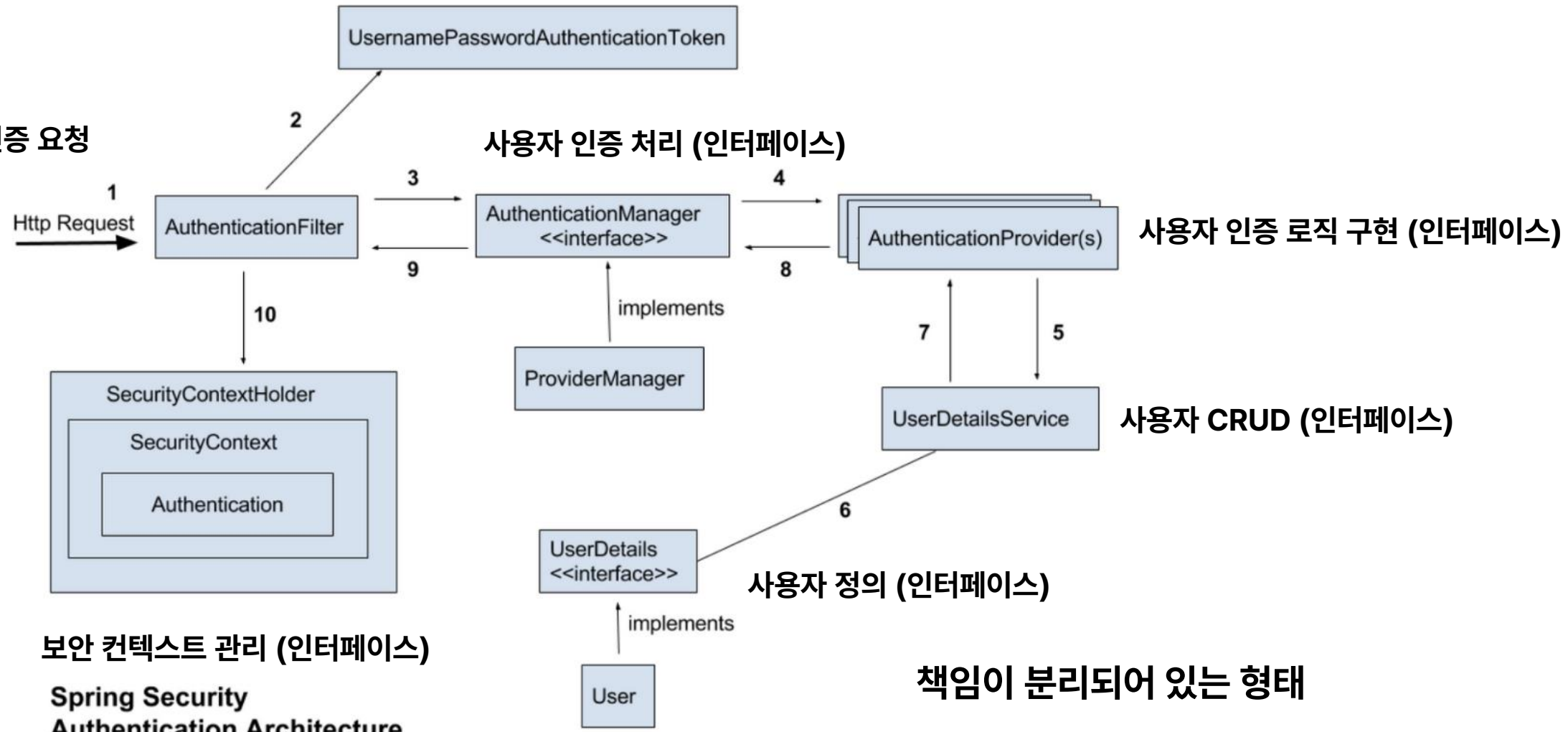
인증, 인가를 어디서 해야할까?

HTTP 요청 → WAS → 필터 → 서블릿 → 스프링 인터셉터 → 컨트롤러

전역적인 인증, 인가: **filter**

그렇지 않는 경우: **interceptor**

Spring security architecture



책임이 분리되어 있는 형태

구현체를 만들어 의존성을 주입가능

FilterChain

필터를 여러 개 만들 수 있음

동일 위치에 필터를 여러 개 둘 수 있음

(순서 보장되지 않음)

여러 필터들을 통과하여 최종적으로 서블릿과 연결되는

하나의 흐름을 filterChain 이라고 한다

실제로 다양한 filter들이 사용되고 있음

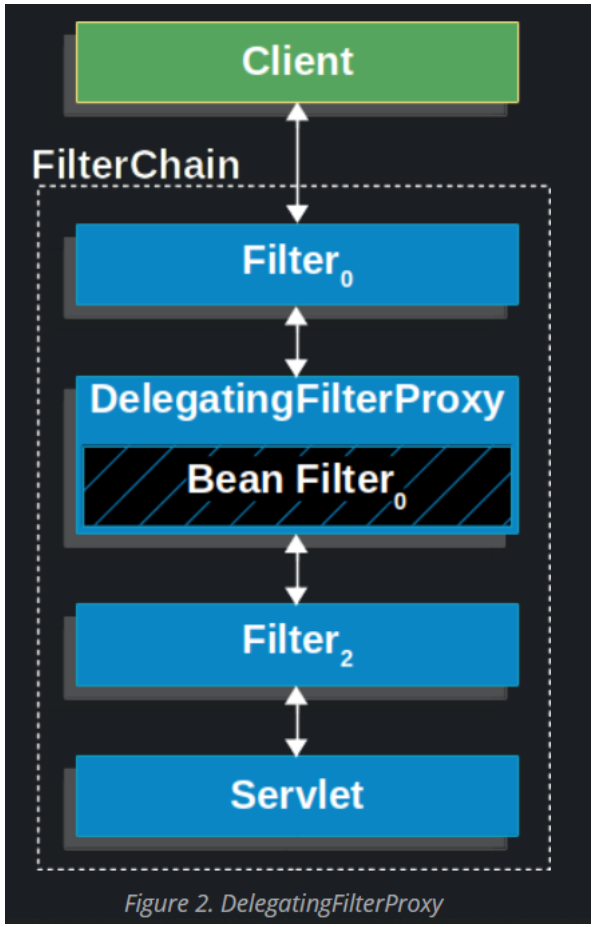


Figure 2. DelegatingFilterProxy

Spring Security 버전마다 filterChain 명세가 다름

Filter는 DelegatingFilterProxy이고 이는 서블릿필터로 내부에 bean filter를

프록시 형태로 가지고 있다고 하는데... 일단 이렇게 이해했습니다.

인가 처리

그러면 인가는 어디서 처리되는가?

인증 필터와 동일하게 인가 필터를 만들 수 있음

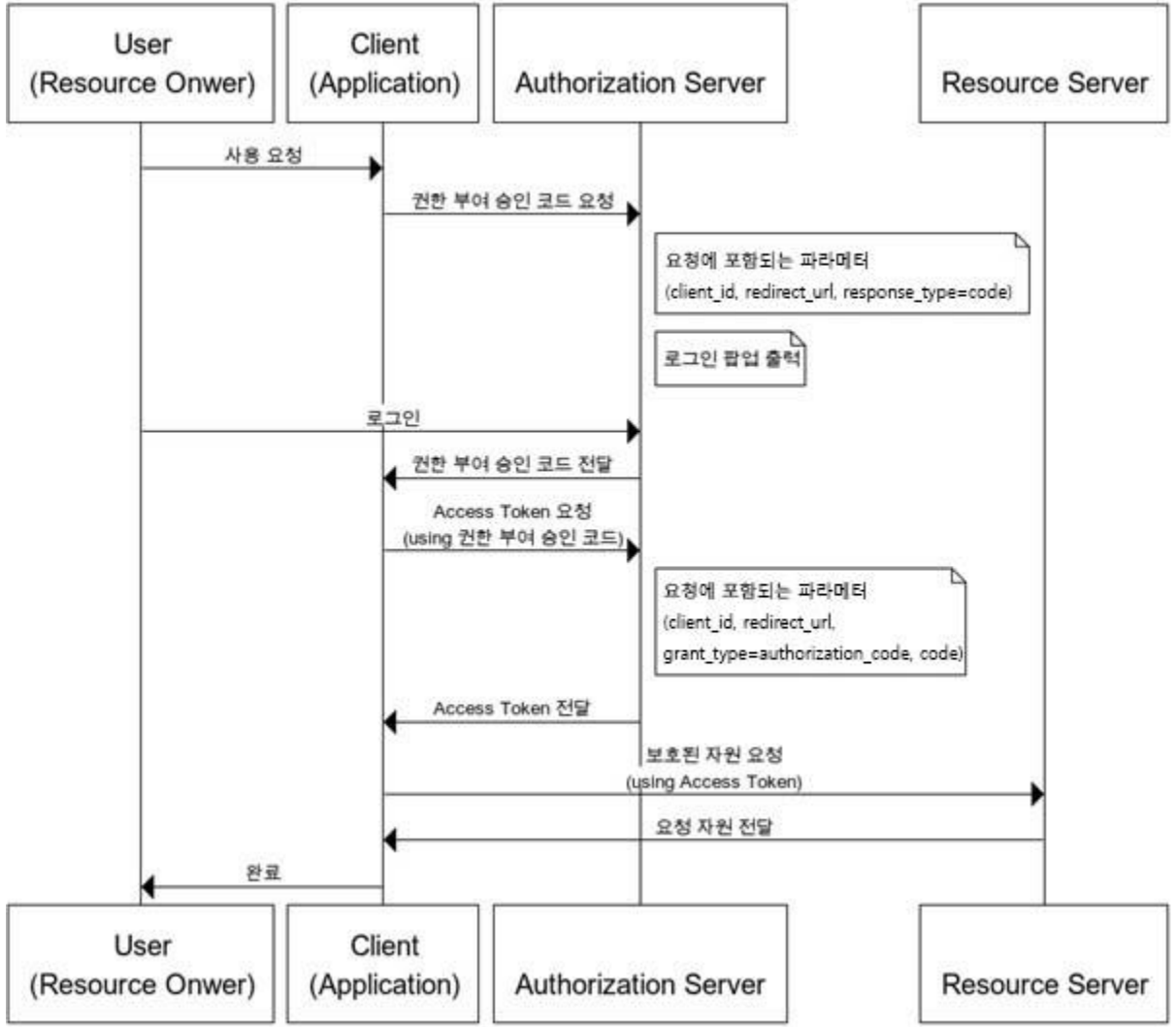
구현체를 만들어서 사용하는 것보다 필터 체인에 직접 추가하는 형태로 대부분 사용하고 있음

```
http
    .authorizeHttpRequests((authorize) -> authorize
        .anyRequest().authenticated()
    )
```

```
@Bean
SecurityFilterChain web(HttpSecurity http) throws Exception {
    http
        .authorizeHttpRequests((authorize) -> authorize
            .requestMatchers("/endpoint").hasAuthority('USER')
            .anyRequest().authenticated()
        )
        // ...

    return http.build();
}
```

OAuth 인증 인가 flow 1



User: 사용자

Client: TravelMate 서비스

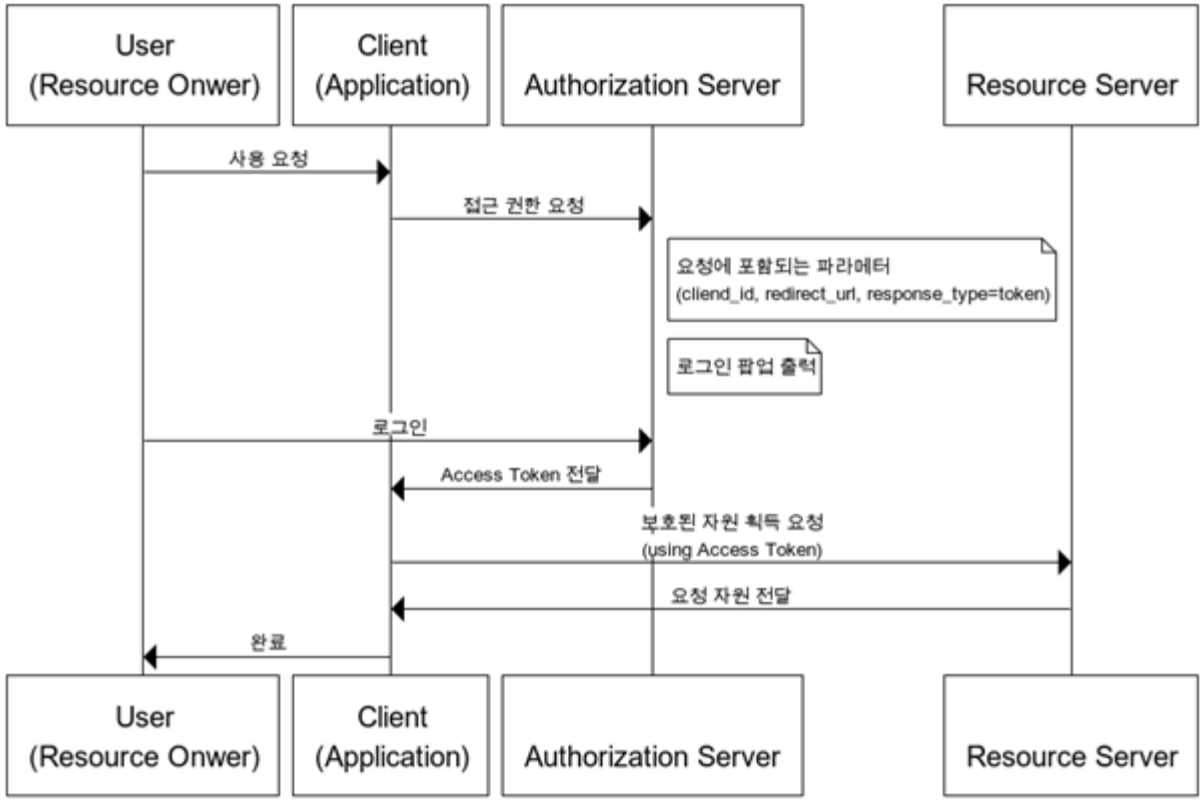
Authorization Server: 구글

Resource Server: 구글

Client는 프론트 엔드가 아닌 요청하는 주체

현재는 TravelMate 서비스가 된다.

OAuth 인증 인가 flow 2



권한 부여 승인 코드를 반환하지 않고

바로 Access Token 반환

Access Token를 바로 반환하는 경우 어떤 취약점이 존재하는가?

"전달된 Access Token를 검증하는 것이 권한 부여 서버 이다" "리소스 서버 이다" 와 같이 혼동을 줄 수 있는 글을 많이 봤는데 너무 이분법적으로 뭐가 맞다 틀리다 보다는 구현하기 나름이다.

다만 Access Token을 검증하기 위해서는 Resource Server를 **반드시(?)** 들러야한다.

암호화 서명을 활용한 토큰 검증

암호화 서명을 사용

권한 부여 서버를 호출할 필요 없음 (직접 승인 코드를 확인)

데이터베이스를 공유할 필요없음 (db에 승인 정보를 저장하여 권한 부여 서버와 리소스 서버 같은 db를 공유)

암호화 서명을 활용한 인증 플로우

1. 클라이언트가 ACCESS_TOKEN 과 인증 수단 (google, kakao) 을 넣어 jwt의 형태로 서버에 전송
2. 서버는 jwt으로 ACCESS_TOKEN 과 인증 수단을 확인하여 해당 리소스 서버에 검증 및 유저 정보 확인
3. DB에 유저 정보가 존재한다면 로그인, 그렇지 않다면 회원 가입

OAuth 에서 제공되는 대부분의 ACCESS_TOKEN이 JWT인 것으로 확인됨

1. 클라이언트가 권한 부여 서버에서 획득한 ACCESS_TOKEN을 서버에 전송
2. 서버에서는 ACCESS_TOKEN을 가지고 리소스 서버에 검증 및 유저 정보 확인
3. DB에 유저 정보가 존재한다면 로그인, 그렇지 않다면 회원 가입

https://velog.io/@mdy0102/%EA%B5%AC%EA%B8%80-Oauth-%EB%A1%9C%EA%B7%B8%EC%9D%B8-%EC%A0%81%EC%9A%A9%EA%B8%B0-1

서버사이드 렌더링에서 인증 플로우

앞선 인증 플로우는 프론트엔드와 벡엔드가 분리된 형태에서의 인증 흐름이다.

이제 살펴볼 인증 플로우는 현재 우리가 진행중인 프로젝트의 형태인 서버사이드 렌더링에서의 인증 흐름이다.

Spring Security는 인증이 되었다면 보안 컨텍스트에 유저 정보를 저장한다.

Spring Security는 서버사이드 렌더링에 사용할 수 있는 형태를 Default로 제공하고 있음

전자의 인증 플로우의 경우 프론트가 Oauth 권한 부여 서버에 접근하여 ACCESS TOKEN을 획득하게 되는데

이경우 Spring Security에서 어떻게 인증 처리를 진행하는지 아직 모르겠음...

대부분 블로그도 서버사이드 렌더링으로 진행했음..

서버사이드 렌더링에서 인증 플로우

하나 확인된 방식은 다음과 같음

1. 클라이언트에서 벡엔드 로그인 api에서 접속
2. 해당 api에서 서버사이드 렌더링으로 로그인 진행
3. 로그인 후 벡엔드 api를 호출
4. 해당 api에서 필요 정보 반환 (ACCESS TOKEN)
5. Client 엔드포인드로 redirect
6. 클라이언트는 서버에서 반환한 JWT를 계속 요청 헤더에 포함해서 전송

결국에는 프론트 사이드 서버 사이드 상관없이 프론트 사이드의 경우에도

로그인 로직을 서버사이드로 진행하면 된다!

서버사이드 렌더링에서 인증 플로우

여담으로 서버사이드 렌더링의 경우 JWT를 사용할 필요가 없어진다. (정확히는 token로 저장할 필요가 없어짐)

인증이 되는경우 Spring Security에서 보안 컨텍스트에 유저정보를 저장하게 되는데,

보안 컨텍스트에 접근하는 것 만으로도 이미 인증이 보장되기 때문에 별도의 토큰을 프론트에게 넘겨줄 필요가 없어짐

스프링은 멀티 스레드 통신으로 요청 마다 스레드가 만들어지는데, 만약 비동기 통신을 하는 경우 요청 스레드와 응답 스레드가 달라 별도의 처리가 필요함

OAuth 로그인 필터 추가

OAuth2Login() 으로 OAuthLoginAuthenticationFilter 필터를 필터 체인에 추가

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
            .oauth2Login(c ->
                    c.clientRegistrationRepository(clientRegistrationRepository()))
            //중략
    return http.build();
}
```

만약 OAuth 말고 다른 로그인 방식이 있는 경우 어떻게 해야 할까? (위 코드는 확장성이 좋아 보이지 않다)

OAuth2Login() 에서 OAuthLoginConfigurer 객체를 반환하는데 OAuthLoginConfigurer내부에 OAuthLoginAuthenticationFilter 필터가 존재

ClientRegistration

권한부여 서버에 접근하기 위한 세부정보 인스턴스

```java
public final class ClientRegistration {
    private String registrationId;
    private String clientId;
    private String clientSecret;
    private ClientAuthenticationMethod clientAuthenticationMethod;
    private AuthorizationGrantType authorizationGrantType;
    //중략
    public class ProviderDetails {
        private String authorizationUri;
        private String jwkSetUri;
        private String issuerUri;
        private Map<String, Object> configurationMetadata;
        //중략
        }
    }
}
```

```java
private ClientRegistration oauthClientRegistration() {
    return CommonOAuth2Provider.GOOGLE
            .getBuilder("google")
            .clientId("secret")
            .clientSecret("secret")
            .build();
}
```

ClientId 부터 OAuth 에 필요한 세부 정보를 모두 담고 있다. (yml에 명시해도 된다)

CommonOAuthProvider에서 자동으로 세부정보를 초기화 시킨다.

ClientRegistrationRepositoty

ClientRegistration을 조회할 수 있는 Repository

```
@Bean
public ClientRegistrationRepository clientRegistrationRepository() {
    ClientRegistration clientRegistration = oauthClientRegistration();
    return new InMemoryClientRegistrationRepository(clientRegistration);
}
```

spring security architecture에서 userDetails 와 userDetailsService와 동일한 구성

Spring security

CSRF 및 CORS 처리

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
            .csrf(CsrfConfigurer::disable)
            .cors(CorsConfigurer::disable)
}
```

왜 우리는 지난 학기에 CORS 문제가 없었는가...

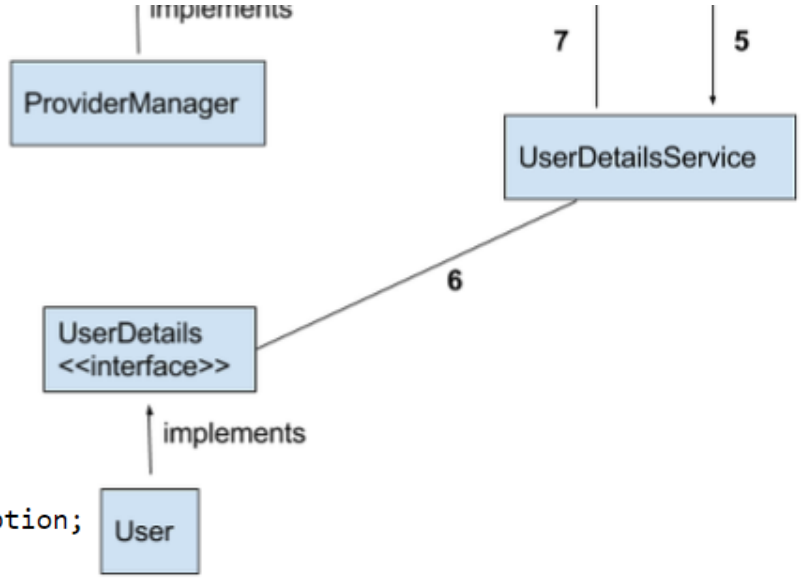
선택기(requestMatchers)를 통한 로그인만 접근 허용

```
.authorizeHttpRequests(o-> o
                        .requestMatchers("**oauth**").permitAll()
```

Spring security

로그인 후 사용자 정보 받아 오기

1. OAuth2UserService 직접 구현
2. DefaultOAuth2UserService 사용




```java
public interface UserDetailsService {

    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;

}
```

```java
@FunctionalInterface
public interface OAuth2UserService<R extends OAuth2UserRequest, U extends OAuth2User> {

    U loadUser(R userRequest) throws OAuth2AuthenticationException;

}
```

Spring security

일반적으로 loadUser에 db에 select 쿼리를 날려 유저가 존재하지 않으면 새로 회원가입 시키는 로직으로 구현

그러나 우리의 경우 기본 정보 이외에도 다른 정보를 추가해야함 (Health)

심지어 user의 Health 외래키가 not null

```
@GetMapping("/gateway")
public String gateway(@AuthenticationPrincipal OAuth2User oauth) {
    Optional<User> user = userService.findByEmail(oauth.getAttribute("email"));
}
```

인증된 OAuth2User에 접근가능

출처

Spring Security docs

https://docs.spring.io/spring-security/reference/servlet/oauth2/index.html

Spring Security docs에서 소개한 방식과 가장 유사

https://velog.io/@goat_hoon/Spring-Security%EB%A5%BC-%ED%99%9C%EC%9A%A9%ED%95%9C-OAuth-%EC%A0%81%EC%9A%A9%EA%B8%B0-Google

다양한 Oauth 동시 사용 예제

https://junuuu.tistory.com/415

OCP 원칙을 지킨 구현

https://junhyunny.github.io/java/design-pattern/spring-boot/multiple-sns-login-with-spring-security-oauth2-client/

그밖에 무수한 포스팅

다음

JWT을 활용한 TOKEN 발급

OCP 원칙을 지킨 다중 Oauth 구현