OAuth

Tags	
스터디 일자	@2023/12/28

OAuth란

기존 인증 방식과 달리 인증을 중개해주는 방식이다!

→ 이미 사용자 정보를 가지고 있는 **소셜 서비스**(Google, Kakao, Github 등)에서 **인증을 대신 진행**해주고, **특정 접근 권한**에 대해 **토큰을 발급**한 후 이를 기반으로 특정 웹 서버에서 인증을 가능하게 하는 식이다.

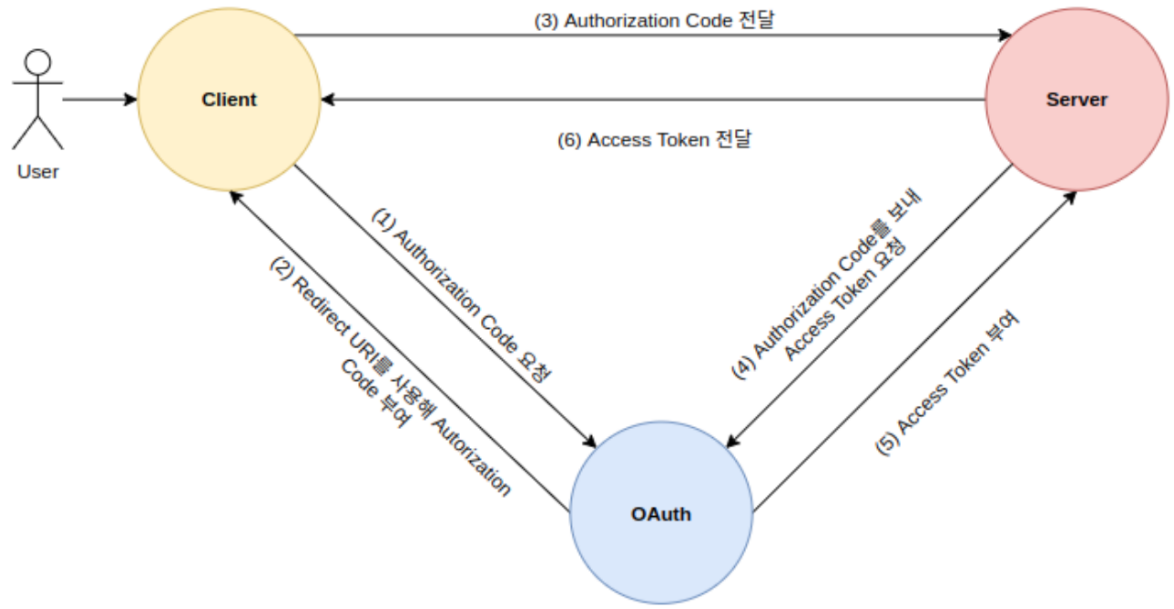
> ☝️ *OAuth 라고 하면 오늘날에는 대부분 **OAuth 2.0 이후의 OAuth**를 말한다.*

OAuth 장점

- 사용자가 아이디, 비밀번호를 일일히 기억할 필요 없이 쉽게 로그인이 가능

- 특정 웹사이트에 로그인한 사용자 정보는 해킹당할 위험이 있는데, 구글, 카카오, 네이버 등 보안이 잘 관리되고 있는 서비스의 로그인 정보를 통해 웹 사이트를 이용하게 함으로써 보안을 보다 강화할 수 있음

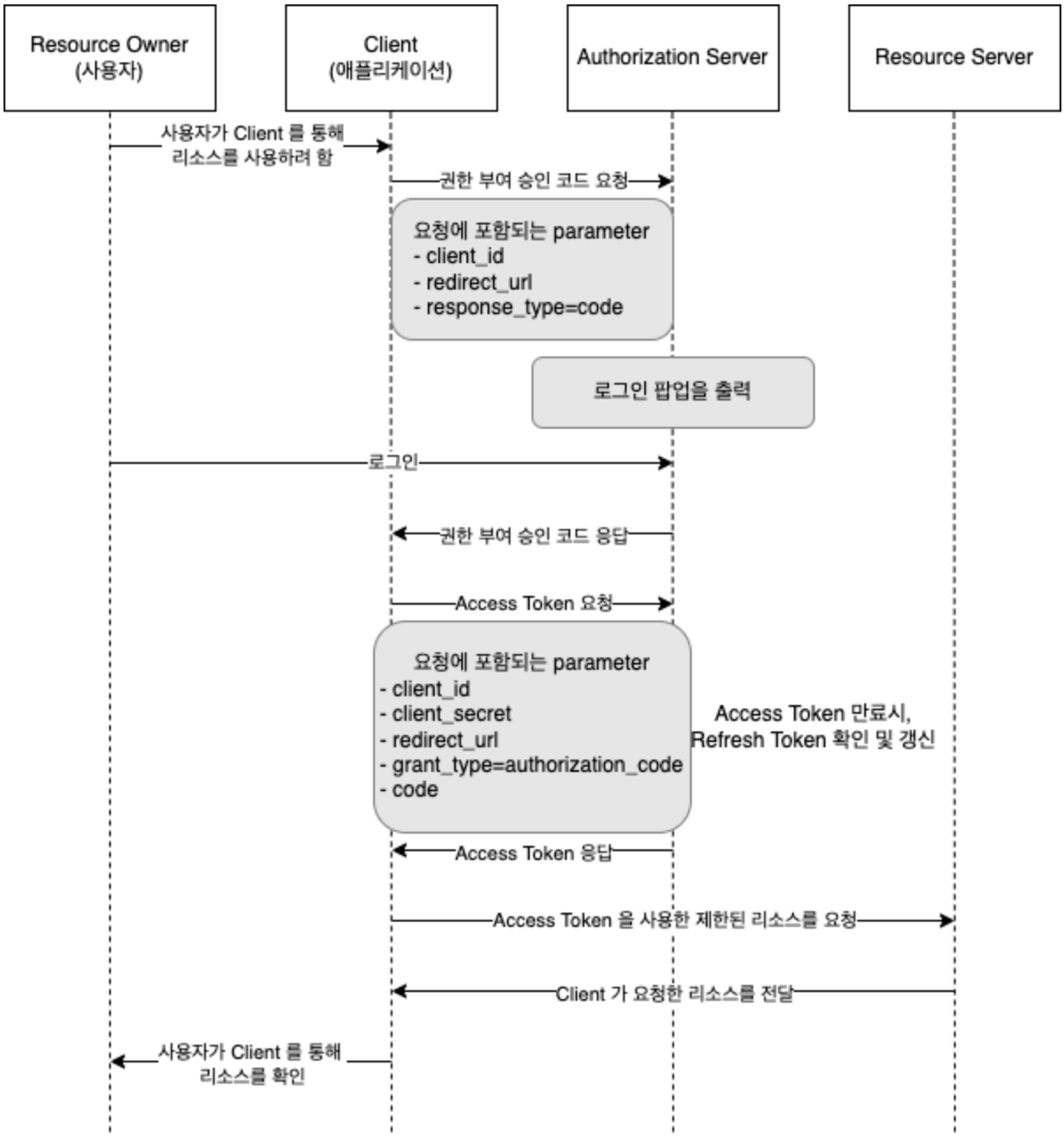
OAuth 인증 과정



1. 사용자가 특정 웹 어플리케이션에서 `OAuth` 서비스를 요청하면
   웹 어플리케이션(Client)에서는
   `OAuth` 서비스에 *Authorization Code*를 요청
2. `OAuth` 서비스는 *Redirect*를 통해 `Client`에게 *Authorization Code*를 부여
3. `Client`는 `Server`에게 OAuth 서비스에서 전달받은 *Authorization Code를 보냄*
4. `Server`는 *Authorization Code*를 다시 `OAuth` 서비스에 전달해, *Access Token*을 전달 받음
5. `Server`는 *Access Token*으로 `Client`를 인증하고 요청에 대한 응답을 반환

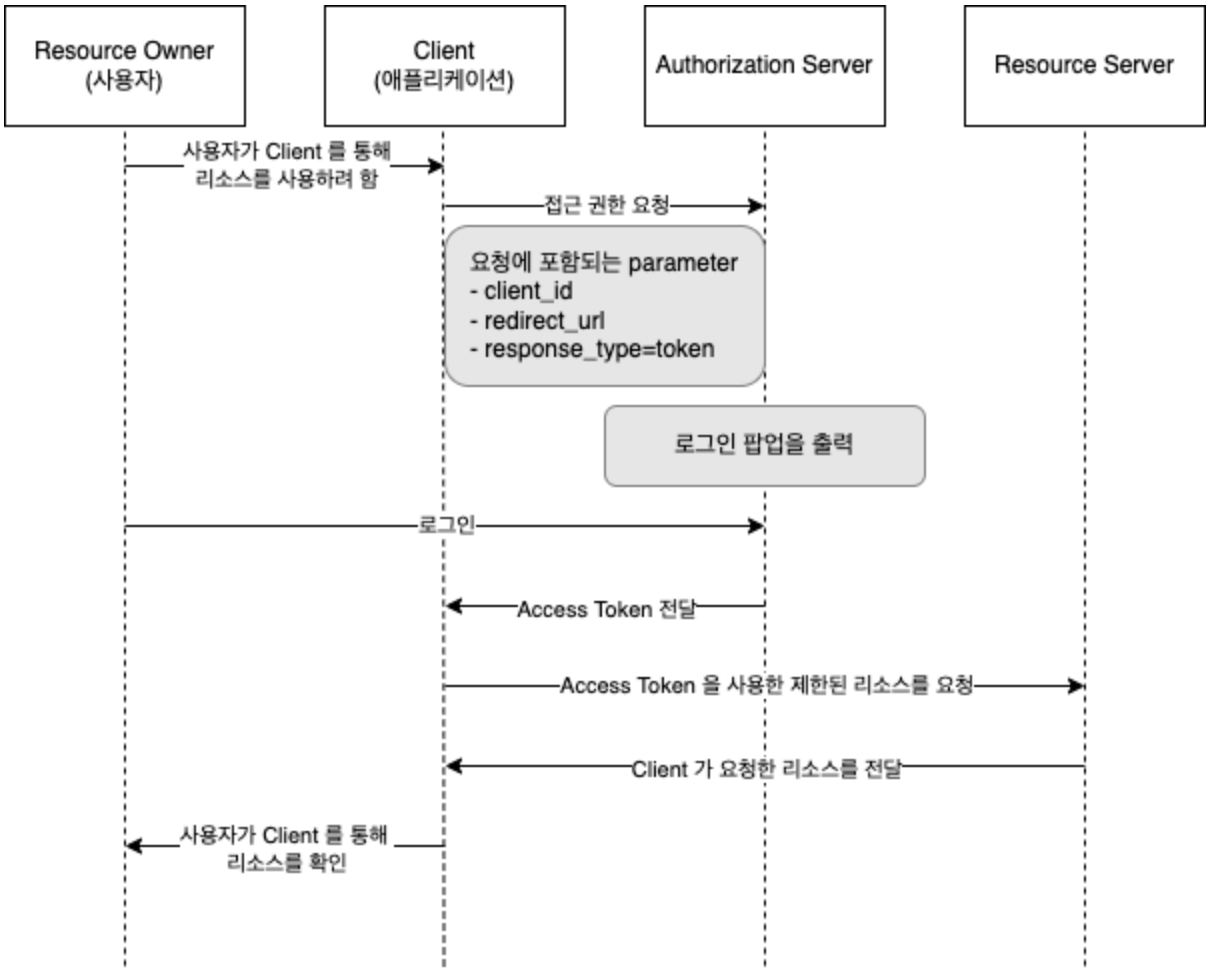
> OAuth2.0에서는 다양한 클라이언트 환경에 적합하도록 **권한 부여 방식에 따른 프로토콜을 4가지 종류로 구분**하여 제공한다.
>
> 1. `Authorization Code Grant`
> 2. `Implicit Grant Type`
> 3. `Resource Owner Password Credentials Grant Type`
> 4. `Client Credentials Grant Type`

Authorization Code Grant (권한 부여 승인 코드 방식)



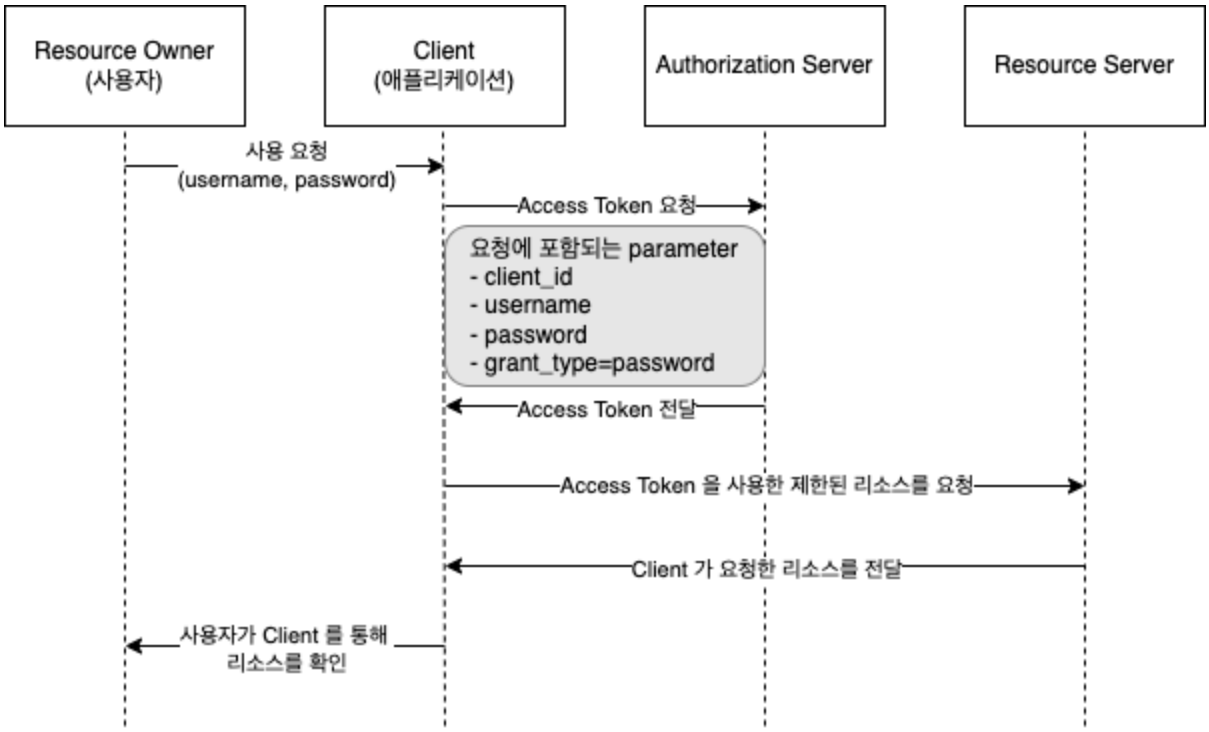
- 권한 부여 승인을 위해 자체 생성한 ***Authorization Code*를 전달하는 방식**으로 많이 쓰임
- 다른 방식보다 좀 더 신뢰성이 있는 방식이라 발급되는 액세스 토큰의 유효시간이 좀 더 김
- *Refresh Token* 사용이 가능

Implicit Grant (암묵적 승인 방식)



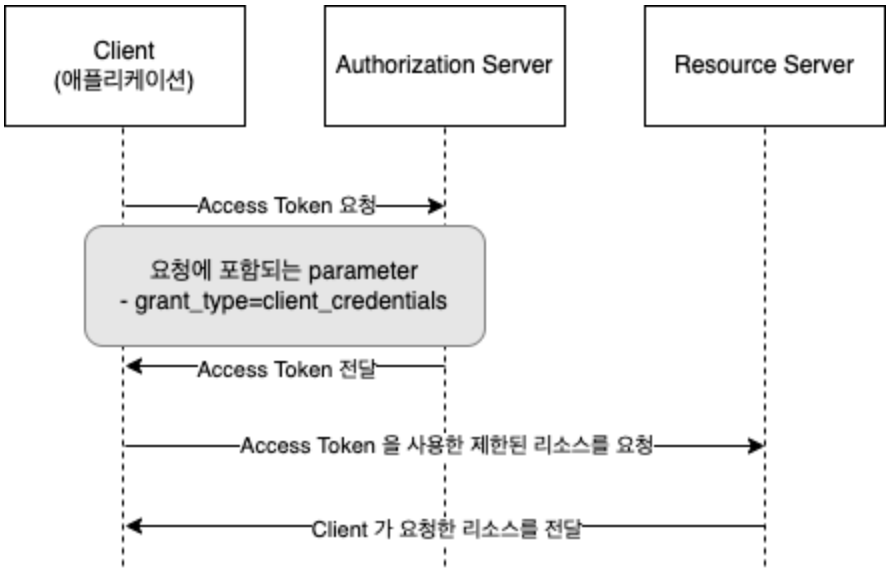
- 자격 증명을 안전하게 저장하기 힘든 **클라이언트에게 최적화된 방식**
- 앞에서 나온 권한 부여 승인 코드 없이 바로 *Access Token*이 발급됨
- *Refresh Token* 사용이 불가능
- *Access Token*이 **URL을 통해 전달되므로 위험**
    - 그래서 *Access Token* 의 만료 기간을 되도록 짧게 설정하여 위험을 줄여야 함

Resource Owner Password Credentials Grant (자원 소유자 자격증명 승인 방식)



- **OAuth 제공자가 서비스하는 애플리케이션에서만 사용 가능한 방식**
  - 페북에 로그인하기, 구글 앱에 로그인하기 등 (자사 앱)
- 단순히 username과 password로 자격을 증명하는 방식
- *Refresh Token* 사용 가능

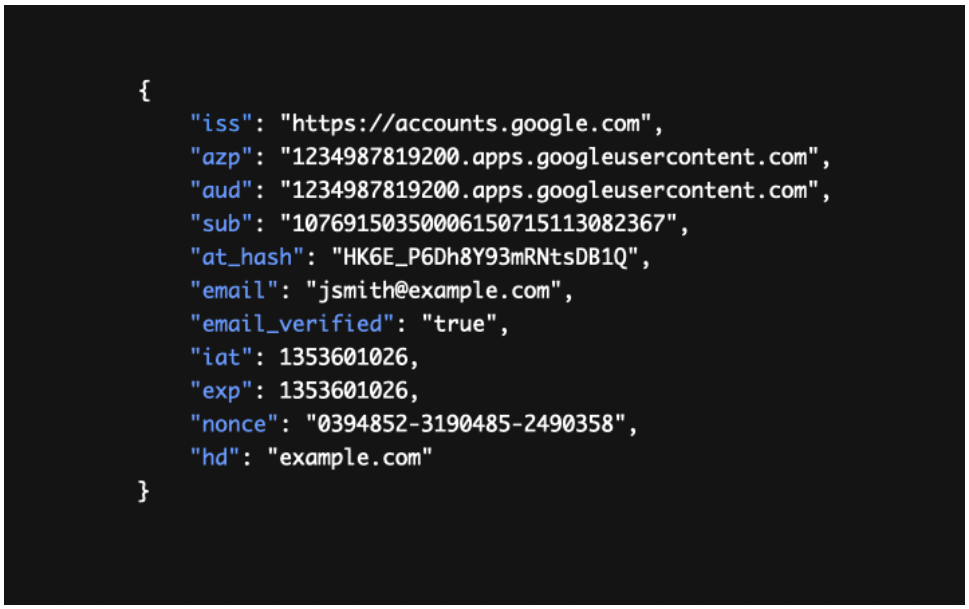
Client Credentials Grant Type (클라이언트 자격증명 승인 방식)



- 가장 간단한 방식으로, 클라이언트 자신이 관리하는 리소스 혹은 권한 서버에 해당 클라이언트를 위한 제한된 리소스 접근 권한이 설정되어 있는 경우 사용
    - 간단하게 그냥 **클라이언트 자체가 Resource Owner 인 경우**를 뜻한다고 생각하면 편함
- *Refresh Token* 사용 불가능

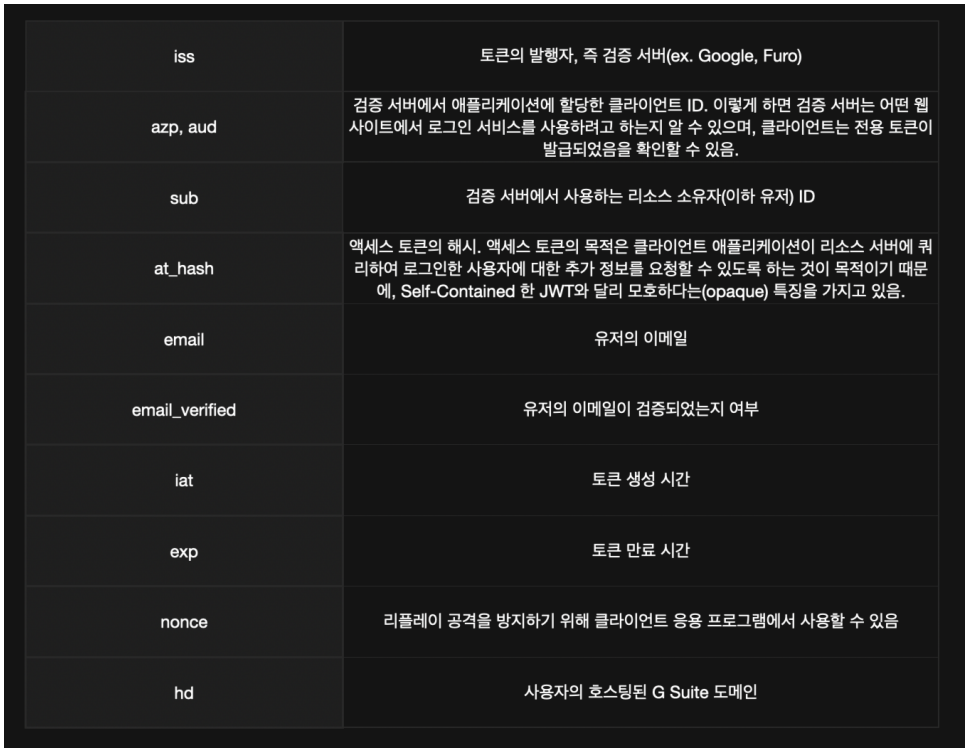
엑세스 토큰 (jwt 아님)

클라이언트가 검증 서버로부터 성공적으로 액세스 토큰 정보를 받아오면, 사용자의 여러 정보를 알아낼 수 있다.



```
{
    "iss": "https://accounts.google.com",
    "azp": "1234987819200.apps.googleusercontent.com",
    "aud": "1234987819200.apps.googleusercontent.com",
    "sub": "10769150350006150715113082367",
    "at_hash": "HK6E_P6Dh8Y93mRNtsDB1Q",
    "email": "jsmith@example.com",
    "email_verified": "true",
    "iat": 1353601026,
    "exp": 1353601026,
    "nonce": "0394852-3190485-2490358",
    "hd": "example.com"
}
```

구글의 액세스 토큰 간소화 예시



iss	토큰의 발행자, 즉 검증 서버(ex. Google, Furo)
azp, aud	검증 서버에서 애플리케이션에 할당한 클라이언트 ID. 이렇게 하면 검증 서버는 어떤 웹 사이트에서 로그인 서비스를 사용하려고 하는지 알 수 있으며, 클라이언트는 전용 토큰이 발급되었음을 확인할 수 있음.
sub	검증 서버에서 사용하는 리소스 소유자(이하 유저) ID
at_hash	액세스 토큰의 해시. 액세스 토큰의 목적은 클라이언트 애플리케이션이 리소스 서버에 쿼리하여 로그인한 사용자에 대한 추가 정보를 요청할 수 있도록 하는 것이 목적이기 때문에, Self-Contained 한 JWT와 달리 모호하다는(opaque) 특징을 가지고 있음.
email	유저의 이메일
email_verified	유저의 이메일이 검증되었는지 여부
iat	토큰 생성 시간
exp	토큰 만료 시간
nonce	리플레이 공격을 방지하기 위해 클라이언트 응용 프로그램에서 사용할 수 있음
hd	사용자의 호스팅된 G Suite 도메인

각 필드가 의미하는 바

OAuth의 문제와 한계

OAuth 인터페이스에는 **유저 정보에 대한 표준이 없어**, 각 서비스 제공자마다 유저데이터를 다른 방식으로 넘겨주기 때문에 클라이언트가 각각의 인터페이스에 개별적으로 대응해야 한다.


```
{
    "iss": "https://accounts.google.com",
    "azp": "1234987819200.apps.googleusercontent.com",
    "aud": "1234987819200.apps.googleusercontent.com",
    "sub": "10769150350006150715113082367",
    "at_hash": "HK6E_P6Dh8Y93mRNtsDB1Q",
    "email": "jsmith@example.com",
    "email_verified": "true",
    "iat": 1353601026,
    "exp": 1353601026,
    "nonce": "0394852-3190485-2490358",
    "hd": "example.com"
}
```


구글


```
{
    "aud": "APP_KEY",
    "sub": "USER_ID",
    "auth_time": 1661967952,
    "iss": "https://kauth.kakao.com",
    "exp": 1661967972,
    "iat": 1661967952,
    "nickname": "John Doe",
    "picture": "http://yyy.kakao.com/.../img_110x110.jpg",
    "email": "johndoe@kakao.com"
}
```


카카오

네이버

Spring Project with OAuth

> `build.gradle`
>
> ```
> implementation 'org.springframework.boot:spring-b
> ```
>
> - 소셜 로그인 등 소셜 기능 구현 시 필요한 의존성
>
> `SecurityConfig` - Spring Security 관련 설정을 하는 곳
>
> ```
> // OAuth 로그인 코드 추가
> http.oauth2Login((oauth2Login) ->
>         oauth2Login
> ```

```
                // 로그인 페이지 설정
        .loginPage("/security-login/login")
                // 로그인 성공 시 이동할 URL 설정
        .defaultSuccessUrl("/security-login")
                // oauth2Login에 성공하면 추후 작업을
                // OAuth2UserService 인터페이스의 구
        .userInfoEndpoint(userInfoEndpoint -> use
                .userService(principalOauth2UserS
                )
);
```

사용자 정보 매핑

```
private static userAuthDto ofGoogle(String userNa

       return userAuthDto.builder()
               .name((String) attributes.get("nam
               .email((String) attributes.get("em
               .picture((String) attributes.get('
               .build();
   }
```

`OAuth2User` 라는 객체로 사용자 정보를 받아올 수 있는데, 이 때 `OAuth2User` 가 반환하는 사용자 정보는 Map 형태이기 때문에 DTO를 통해 값 하나하나를 변환해주어야 한다.

OAuth와 JWT의 차이?

- **JWT**는 토큰의 종류
- **OAuth**는 토큰을 발급하고 인증하는 오픈 스탠다드 프로토콜

☝️ **OAuth Token**은 사용자의 중요한 정보를 담고 있지 않고 일종의 랜덤한 문자열이다. 단지 **OAuth**를 사용하는 인증 서버에서 사용자 정보를 가리키고 있는 포인터라고 생각하면 된다.

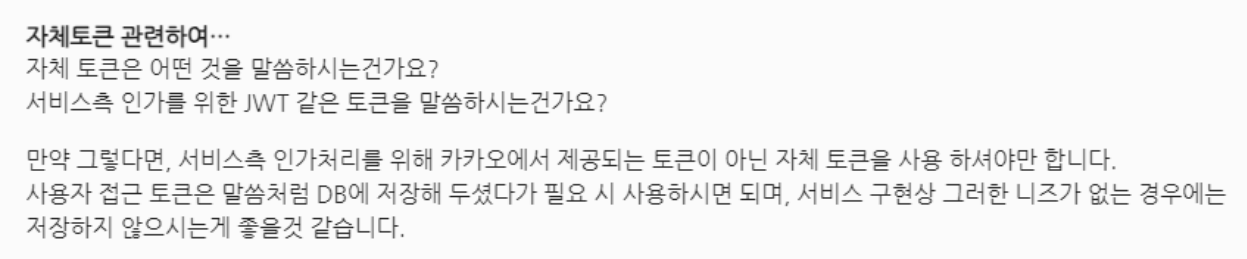
반면, **JWT Token**은 헤더, 페이로드, 서명으로 구성되어 있고 페이로드에는 사용자 정보 등 다양한 정보가 담겨 있는 토큰이다.

> **JWT**는 **클라이언트와 서버 간의 인증**을 처리하기 위한 토큰이고, 토큰 자체에 정보를 포함한다.
>
> **OAuth**는 **사용자의 인증과 권한 부여**를 위한 프로토콜로, 클라이언트가 사용자의 동의를 얻어 제한된 액세스 토큰을 발급 받아 자원에 접근한다!

하지만, OAuth 토큰만으로 인증 / 인가를 구현하는 방법은 **하지 말란다.**

왜인지는 정확하게 모르겠으나, 카카오 문의 답변에도 그러라고 하고.. **암튼 안좋답니다!** (그대로 쓰면 고소 당한다는 말도 잇음ㅋㅋㅋ!!!)

(유력한건 로그인 한 후에도 계속 소셜에다 토큰으로 요청을 해야한다는점? 인듯)



GoodGood

32 · 7개월 전

만들지 않고도 사용은 가능하죠. 다만, 말씀하신대로 해당 토큰에 대해서 validation은 항상 카카오에 요청해야 할거에요
만약 jwt를 직접 구성한다면 validation은 백엔드에서 직접 수행할 수 있겠죠

저는 가입 및 로그인시에만 kakao access token을 사용했고, 해당 token에서 가져올 수 있는 유저 id를 jwt에 넣어서 사용자 클라이언트에 저장해뒀습니다.
그럼 사용자는 요청시마다 jwt를 담아서 보내고 저는 백엔드에서 jwt를 검증하기만 하면 되죠

보통은 소셜 플랫폼에서 Access Token을 발급받고, 토큰에 포함된 사용자 정보를 토대로 자신의 서비스에 맞는 JWT를 만드는 편입니다.

이유는 여러가지가 있겠지만, 플랫폼이 준 Access Token의 경우 서비스가 능동적으로 다루기 어렵다는 특징이 있습니다. (페이로드, 만료시간 등)

그럼 질문자분의 두 번째 질문처럼, 그냥 플랫폼의 토큰체계를 그대로 써도 되냐는 궁금증이 생기는데..

일단 가능은 합니다만, 소셜 플랫폼마다 정책이 조금씩 다르니 확인해보시기 바랍니다. Access Token이 일회성인 곳도 있어서, 제 기억엔 한 번 토큰 받아서 정보 취득하면 다시 발급받아야하는 경우도 있었습니다.

소셜 로그인을 구현할 때는 OAuth + JWT을 사용해보아요~~!

참고한 곳

https://velog.io/@on5949/SpringSecurity-OAuth-와-일반로그인을-동일하게-처리하기

소셜 로그인 토큰을 그대로 써도 되나요?

https://velog.io/@jinony/왜-OAuth-JWT일까

https://velog.io/@chanyoung1998/JWT-와-OAuth-의-차이

https://velog.io/@wooyoung-tom/oauth