5-2 : 인증, 인가 및 OAuth2

들어가기 전에

이번 시간에는...

인증, 인가
OAuth2

인증 & 인가

인증 & 인가

인증

유저가 누구인지 확인하는 절차 ex) 로그인

인가

유저에 대한 권한을 허락하는 것 ex) 로그인 이후 회원 판별

로그인 방식

로그인 방식 2가지

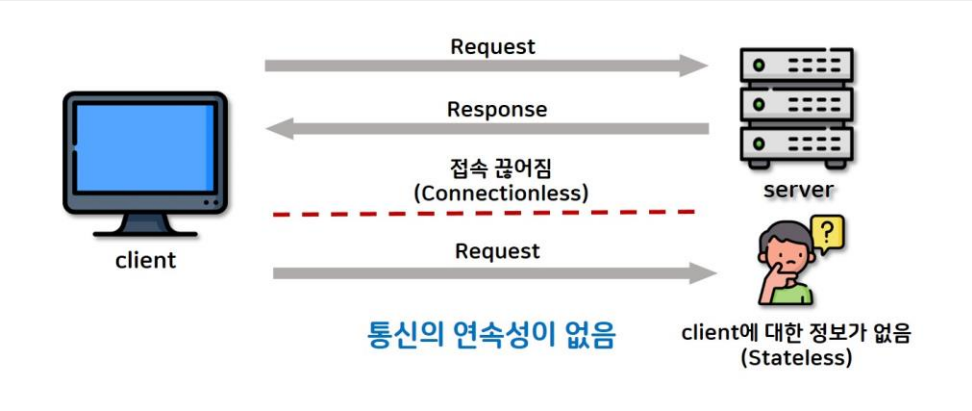
로그인을 하는 방법을 떠올려보자.

1. **ID & PW**
2. **SNS(OAuth)**

인가 방식

인가 방식 3가지

HTTP는 무상태성 특징
클라이언트에 대한 이전 상태 정보 및 현재 통신 상태가 남아있지 않기 때문에
로그인한 유저 정보를 유지하기 위한 여러 방법들이 고안



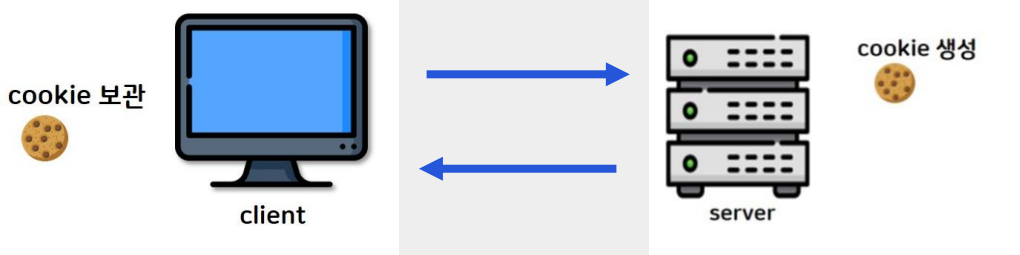
1. 쿠키
2. 세션
3. JWT

쿠키

쿠키(Cookie)

쿠키란 클라이언트가 어떠한 웹사이트를 방문할 경우, 그 사이트가 사용하고 있는 서버를 통해
클라이언트의 브라우저에 설치되는 **작은 기록 정보 파일**

무상태 환경에서 기억하고자 하는 데이터를 쿠키에 담고,
다음 번 요청 시에 **쿠키를 요청에 함께 담아 보냄으로써 무상태를 극복**



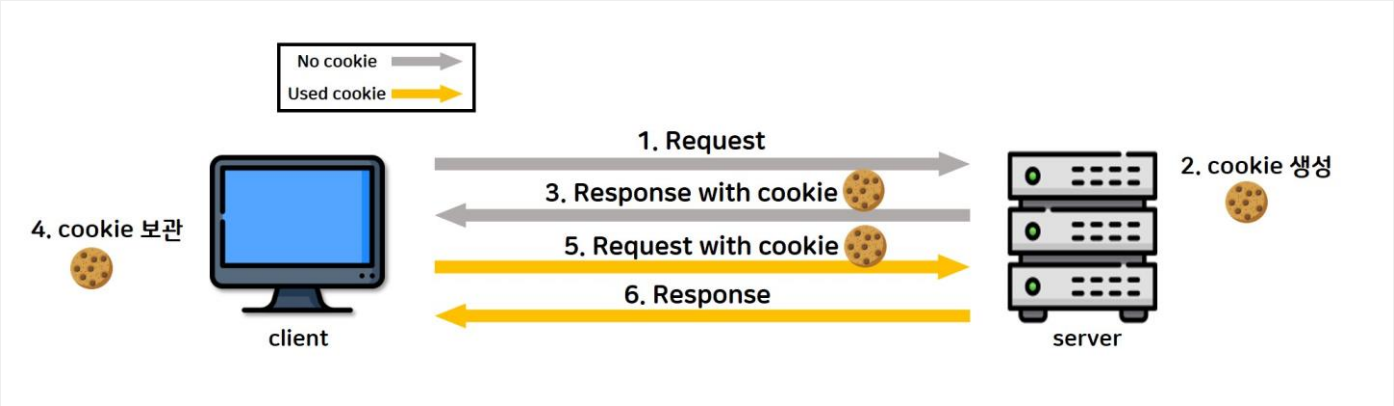

```
MockHttpServletRequest:
      HTTP Method = GET
      Request URI = /user/my/edit
          Headers = [Cookie:"userName=kevin"; "password=abc123"]
```

```
MockHttpServletResponse:
           Status = 200
          Headers = [Set-Cookie:"userName=kevin", "password=abc123"]
```

쿠키

쿠키 인가 과정

1. 사용자의 로그인 요청
2. 로그인 성공 시, 서버는 쿠키를 생성
3. 쿠키에 회원 정보를 담아 응답
4. 사용자는 브라우저 저장소에 쿠키를 저장
5. 다음 번 요청 시 쿠키를 요청에 함께 전달
6. 서버는 쿠키를 보고 유저를 확인하고, 응답



쿠키

쿠키 단점

보안에 취약하다

- 요청 시 쿠키의 값을 그대로 보내기 때문에 개인 정보가 외부에 노출
- 유출 및 조작 당할 위험이 존재

브라우저간 공유 불가능

- 웹 브라우저 마다 쿠키에 대한 지원 형태가 다르기 때문에 브라우저 간에 공유가 불가능

용량 문제

- 쿠키에는 용량이 있어서 많은 정보를 담을 수 없다

쿠키 사이즈

- 쿠키의 사이즈가 커질수록 네트워크 부하가 심해진다

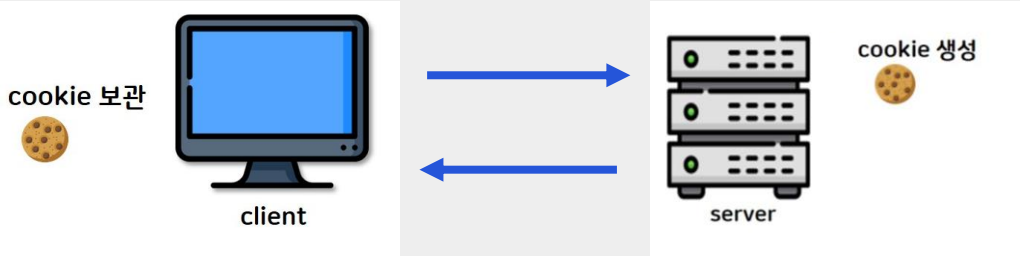
세션

세션(Session)

쿠키를 통해 클라이언트 로그인 상태를 유지 가능
하지만 가장 큰 단점은 쿠키가 유출 및 조작 당할 위험이 존재한다는 것

개인정보를 HTTP로 주고 받는 것은 위험

세션은 이러한 비밀번호 등 클라이언트의 인증 정보를 쿠키가 아닌 **서버 측에 저장하고 관리**합니다.



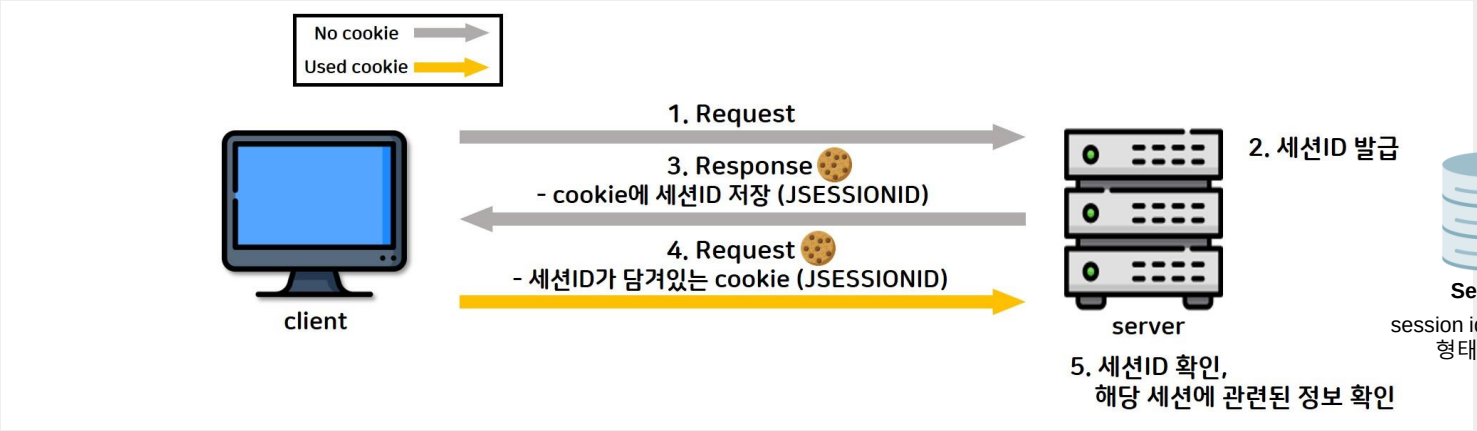

```
HTTP Method = GET
Request URI = /user/my/edit
Headers = [Cookie: "JSESSIONID=FDB5E30BF20045E8A9AAFC788383680"]
```

```
HTTP/1.1 200
Set-Cookie: JSESSIONID=FDB5E30BF20045E8A9AAFC788383680C;
```

세션

세션 인가 과정

1. 사용자의 로그인 요청
2. 로그인 성공 시, 서버에서 세션ID 생성 및 저장
3. 쿠키에 생성한 세션ID 정보를 담아 응답
4. 사용자는 브라우저 저장소에 쿠키를 저장해두고, 다음 요청 시에 쿠키를 함께 전달
5. 서버는 쿠키를 확인하고, 세션ID에 대응되는 유저가 누구인지 확인 후 응답



세션

세션 장단점

장점

- 쿠키를 포함한 요청이 외부에 노출되더라도 유의미한 개인정보가 없기 때문에 안전

단점

- 개인정보가 직접 노출되는 문제 제외한 쿠키의 단점을 그대로 가짐
- 해커가 쿠키를 중간에 탈취하여 마치 해당 유저인 척 위장할 수 있다.
- 서버의 세션 저장소 구축, 관리 부담

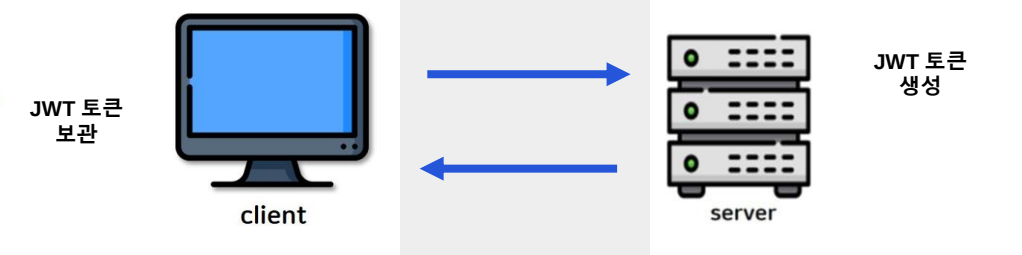
JWT

JWT

JWT(JSON Web Token)란 인증에 필요한 정보들을 암호화 시킨 토큰

JWT 기반 인증은 쿠키/세션 방식과 유사하게
JWT 토큰(Access Token)을 HTTP 헤더에 실어 서버가 클라이언트를 식별합니다.

쉽게 말해서 **JWT 토큰이 쿠키를 대신**



JWT

JWT 토큰 구조

JWT는 .을 구분자로 나누어지는 세 가지 문자열의 조합

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

암호화할 해싱 알고리즘 및
토큰의 타입에 대한 정보

Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

토큰에 담을 정보.
주로 클라이언트의 고유ID, 유효 기간 등을 포함

Signature

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) secret base64 encoded
```

인코딩된 Header와 Payload를 더한 뒤
서버의 비밀키로 해싱하여 생성.
(위변조 여부 확인 목적)

JWT

JWT 인가 과정

1.클라이언트 로그인 요청

2. 로그인 성공 시, 유저 정보 및 유효기간 등을 Payload에 담고, 비밀키를 사용해 Access Token(JWT)을 발급

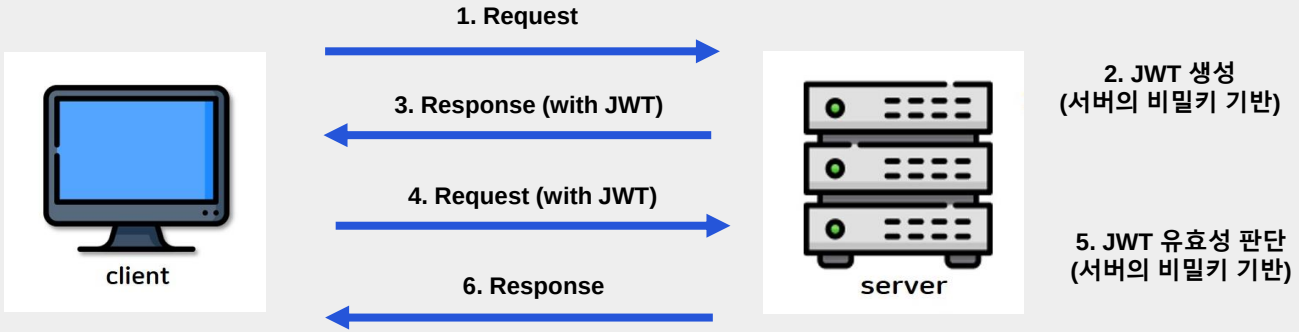
3. 서버는 생성한 JWT 토큰을 클라이언트에게 전달

4. 클라이언트는 전달받은 토큰을 저장해두고, 요청할 때 마다 토큰을 요청 헤더 Authorization에 포함시켜 함께 전달

5. 서버는 받은 JWT의 헤더, 페이로드에 비밀키를 더해 암호화

5. 암호화 값이 JWT의 Signature와 일치하는지 판단 후, 유효 기간 등을 확인하여 유효한 토큰인지 확인

6. 유효한 토큰이라면 요청에 응답

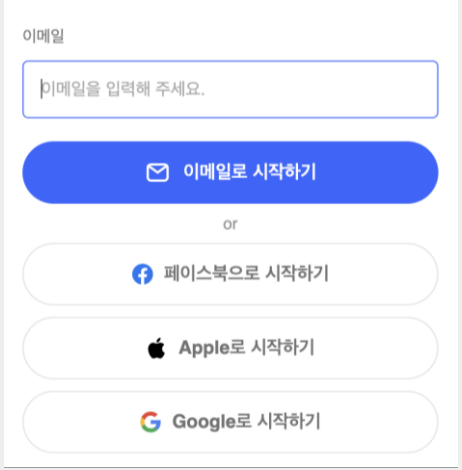


OAuth

OAuth

OAuth는 인터넷 사용자들이 비밀번호를 제공하지 않고
다른 웹사이트 상의 자신들의 정보에 대해 웹사이트나 애플리케이션의 접근 권한을 부여할 수 있는
공통적인 수단으로서 사용되는, 접근 위임을 위한 개방형 표준_위키백과

쉽게 말해서 **페이스북, 구글, 카카오 로그인 등의 외부 서비스의 로그인 방식으로 인증하는 방법**



OAuth

OAuth2 참여자

OAuth2 동작에 관여하는 참여자는 크게 3가지로 구분

1. Resource Server : Client가 제어하고자 하는 자원을 보유하고 있는 서버입니다.

ex) Facebook, Google, Twitter

2. Resource Owner : 자원의 소유자(유저)입니다

ex) Client 서비스를 통해 로그인하는 실제 유저

3. Client : Resource Server에 접속해서 정보를 가져오고자 하는 클라이언트(웹 어플리케이션)입니다.

ex) 내 서비스

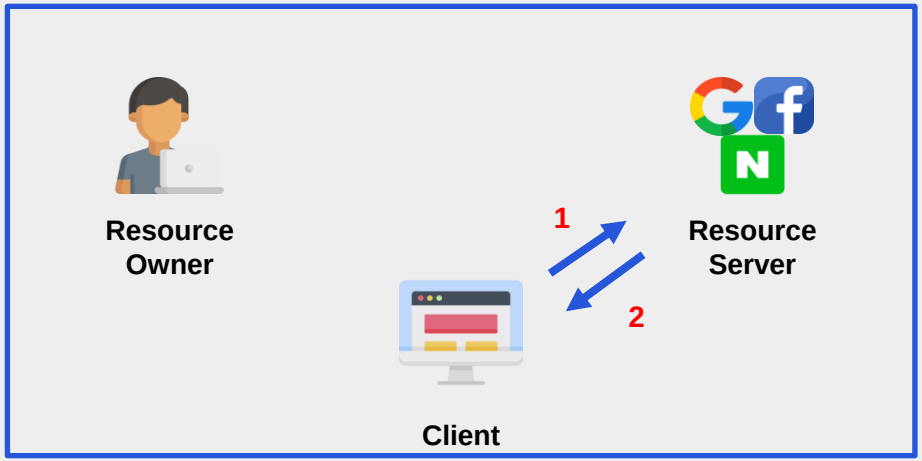
OAuth

OAuth2 인증 과정

Client 등록 및 설정

1. Client는 Resource Server를 이용하기 위해 자신의 서비스를 등록
2. 등록을 마치면, Client는 Client ID, Client Secret 등의 정보를 Resource Server로 부터 제공 받는다.

(Resource Server는 Client ID, Client Secret 등의 정보로 Client를 구분)

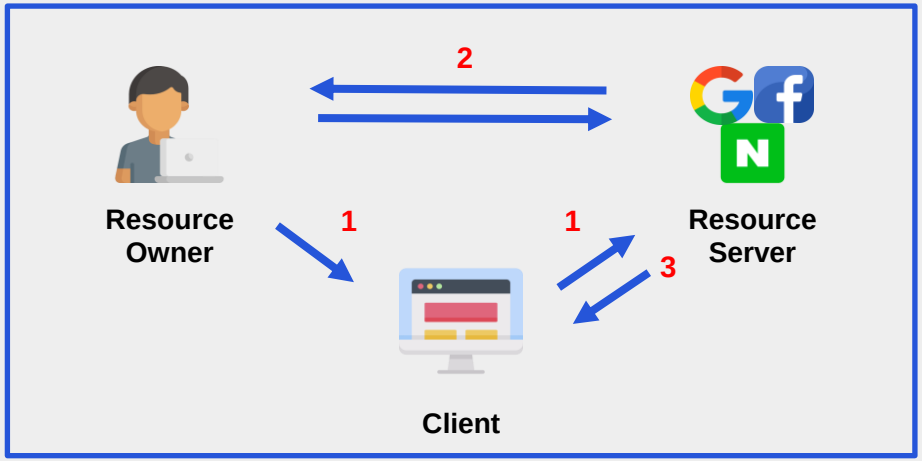


OAuth

OAuth2 인증 과정

Authorization Code 발급

1. Resource Owner가 소셜 로그인 버튼을 누르면, Client를 통해 Resource Server 로그인 페이지로 리다이렉션
2. Resource Server에서는 요청한 Client의 ID, Secret 값 등을 확인하고 인증이 되면 로그인을 승인
   로그인 승인 후, Resource Owner에게 Client로 정보 위임 권한을 줄 건지 질의
3. Resource Owner가 권한 동의를 마치면, Resource Server는 Client가 등록한 리다이렉션 URL로 Authorization code를 담아 요청

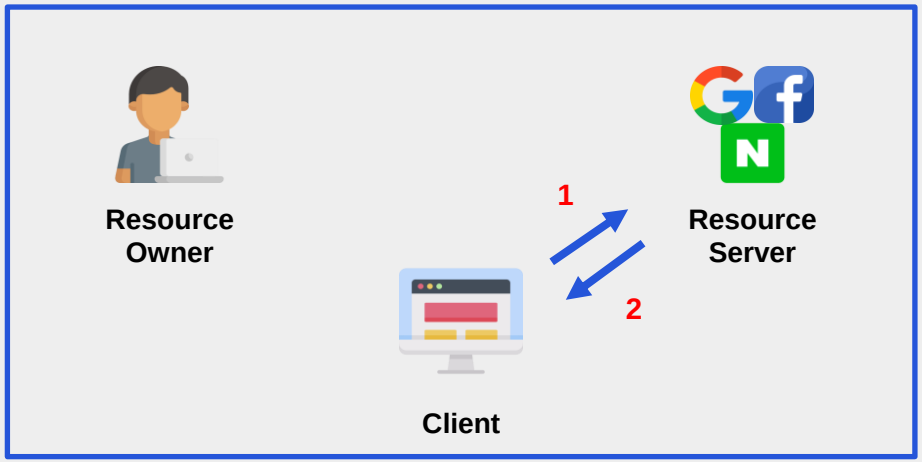


OAuth

OAuth2 인증 과정

Access Token 발급

1. Client는 ID와 비밀키 및 발급받은 Authorization Code 값을 Resource Server에게 직접 전달
2. Resource Server는 정보를 검사한 다음, 유효한 요청이면 Access Token을 발급

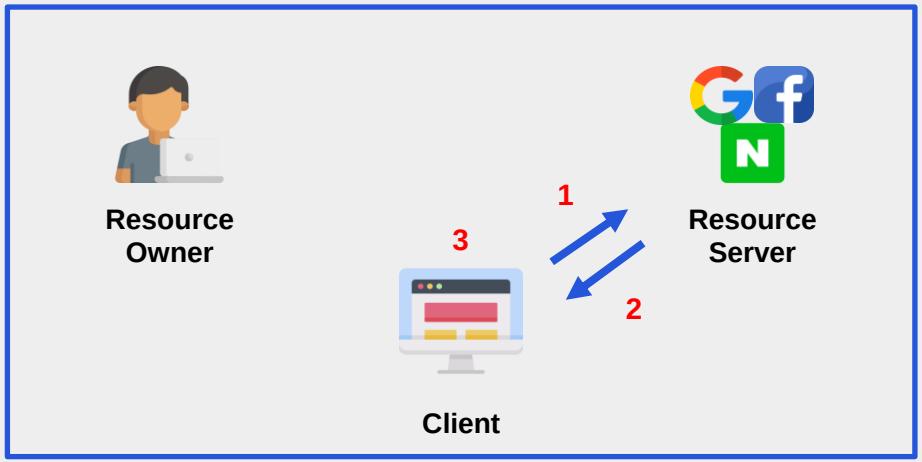


OAuth

OAuth2 인증 과정

유저 정보 조회

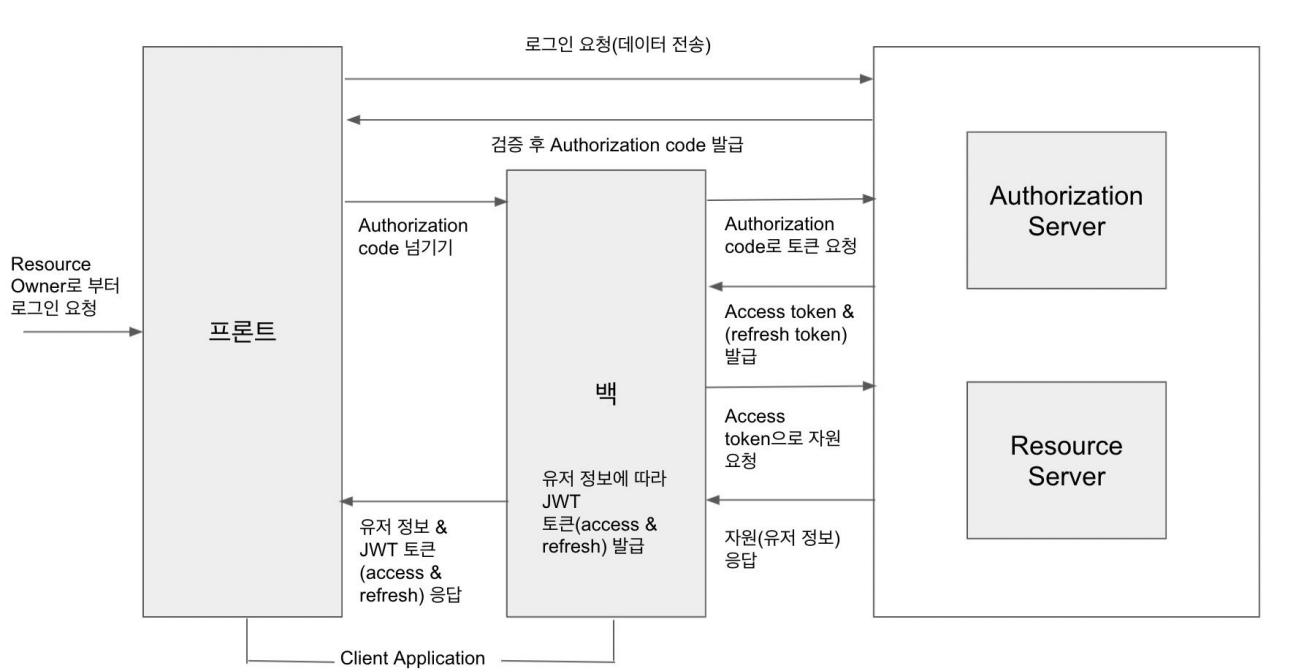
1. Client는 유저 정보 조회를 위해 AccessToken을 헤더에 담아 보냄
2. Resource Server는 Access Token을 검증하고 유저 정보를 내려 줌
3. Client는 받아온 유저 정보를 가지고 회원가입 및 로그인 기능 수행
   이후 JWT 토큰을 발급하여 인가 처리 수행



OAuth

OAuth2 인증 과정

시퀀스 다이어그램 예시



OAuth 구현 방법

OAuth2 구현 방법

1. Resource Server에 내 Application을 Client로 등록
2. Resource Server 로그인 페이지로 리다이렉션 하는 API 제작(Authorization code 발급)
3. Resource Server 로그인 인증 후, Access Code를 담은 리다이렉션 요청을 받는 API 제작
4. 3번 API 로직에서 Access Code를 Header에 담아 Access Token을 받아오고, 이를 다시 담아 유저 정보를 받아오는 로직 구현

(RestTemplate, WebClient, Apache Client 등 Server to Server 통신 라이브러리 활용)

5. 받아온 정보를 활용하여 회원가입, 로그인 로직 구현
6. 받아온 정보로 JWT 토큰을 만들어 프론트에게 응답

과제 안내

Spring Boot 강의 수강(총 약 3시간 분량)

Section16. Spring Security로 Spring Boot 앱 보호하기

과제 안내

인증, 인가 구현

4주차에 개발한 프로젝트에 인증, 인가 구현

OAuth2를 활용한 인증 구현

JWT를 활용한 토큰 기반 인가 구현

과제 유의사항

- 반드시 스프링 시큐리티를 사용하지 않으셔도 됩니다.
- OAuth2가 익숙하지 않다면, 우선 ID & PW 기반 인증으로 구현하기

Hello World!