

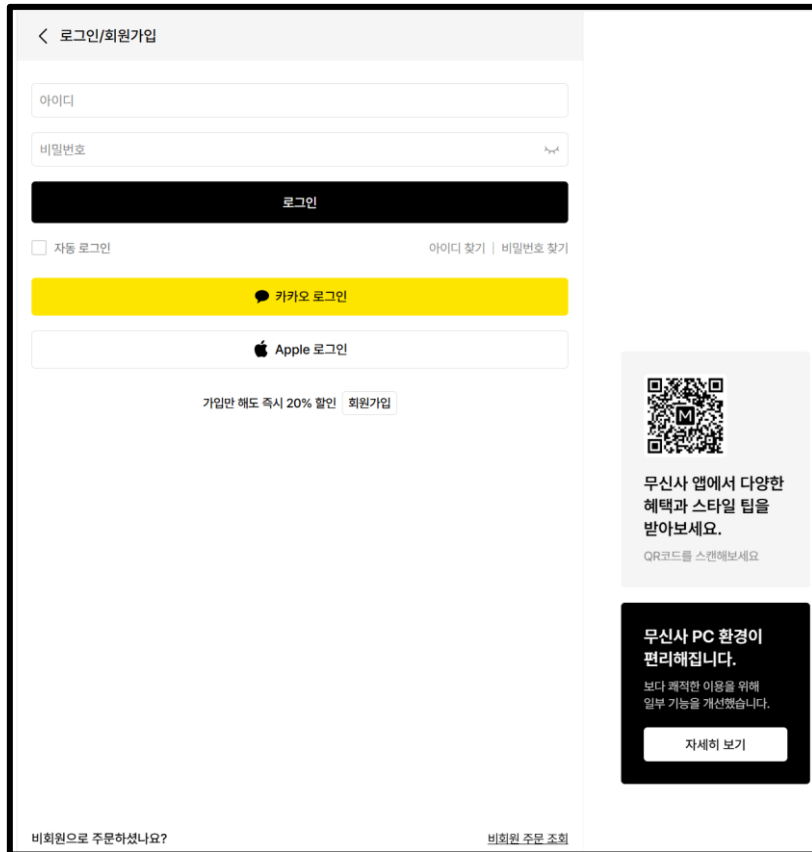
OAuth 2.0과 Social Login

2025.03.18

황지연

1. OAuth Overview (1/1)

- OAuth는 접근 위임을 위한 개방형 표준이다.
 - 인터넷 사용자들이 비밀번호를 제공하지 않고 다른 웹사이트 상의 자신들의 정보에 대해 웹사이트나 애플리케이션의 접근 권한을 부여할 수 있도록 함



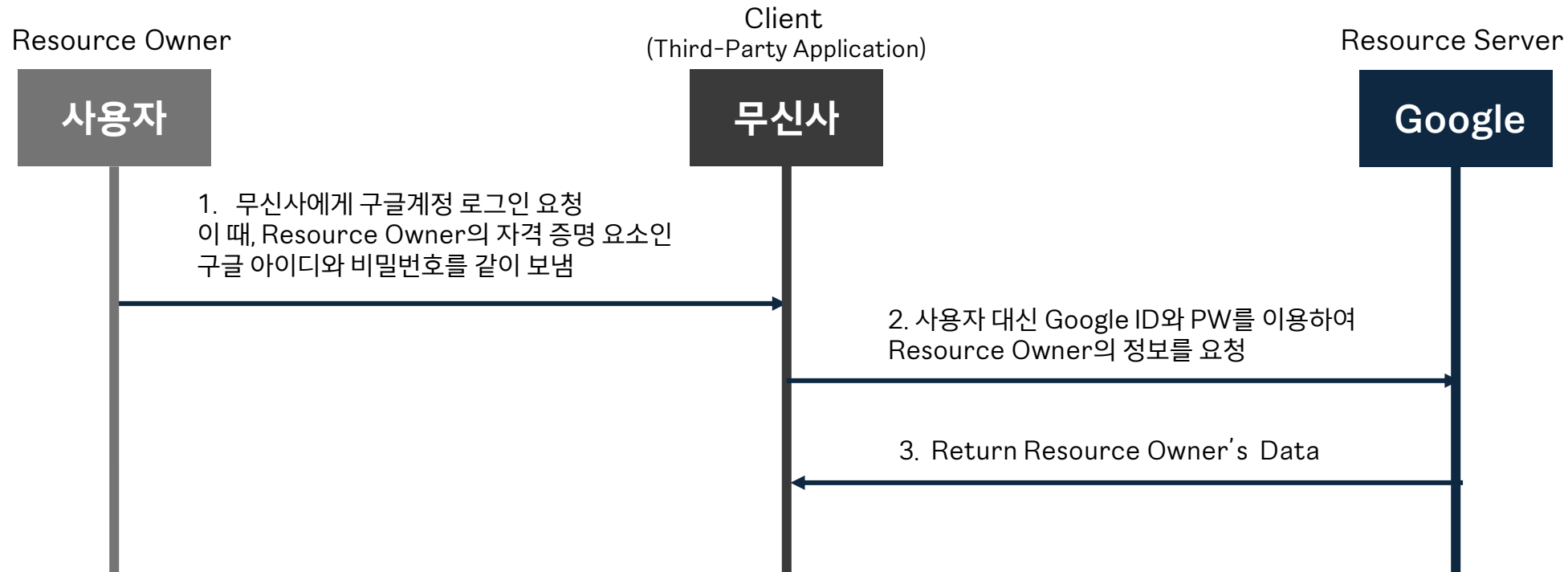
The screenshot shows a mobile app login screen for '무신사' (Minsasa). At the top, there's a back arrow and the text '로그인/회원가입'. Below this are input fields for '아이디' (ID) and '비밀번호' (Password), followed by a black '로그인' (Login) button. Underneath is a checkbox for '자동 로그인' (Auto login) and links for '아이디 찾기' (Find ID) and '비밀번호 찾기' (Find Password). A prominent yellow button labeled '카카오 로그인' (Kakao login) is shown, along with a smaller 'Apple 로그인' (Apple login) button. At the bottom, there's a promotional banner for a 20% discount on new members. On the right side of the screen, there's a QR code with the text '무신사 앱에서 다양한 혜택과 스타일 팁을 받아보세요. QR코드를 스캔해보세요' (Receive various benefits and style tips from the Minsasa app. Scan the QR code). Below the QR code, there's a dark box with white text stating '무신사 PC 환경이 편리해집니다. 보다 쾌적한 이용을 위해 일부 기능을 개선했습니다.' (Minsasa PC environment is more convenient. To provide a more comfortable experience, we have improved some features.) with a '자세히 보기' (View details) button. At the very bottom, there are two small links: '비회원으로도 주문하셨나요?' (Did you order as a non-member?) and '비회원 주문 조회' (Check non-member order).

무신사의 로그인 화면
카카오 로그인, Apple 로그인을 이용할 수 있다.

사용자는 카카오 ID/PW를 무신사에게 넘기지는 않으면서
해당 계정 정보로 카카오에 등록된 이름, 나이, 이메일 정보에는
접근할 수 있도록 한다.

2. OAuth 등장배경 (1/2)

- 기존 클라이언트 - 서버 인증 모델에서는 클라이언트가 서버의 보호된 Resource에 접근하기 위해 Resource Owner의 자격 증명(e.g. ID / PW) 을 직접 사용 해야 했음
- 문제 상황
 - 만약 타사 (Third-Party) 애플리케이션이 특정 리소스에 접근할 필요가 있을 경우, 리소스 소유자는 자신의 계정 정보를 그대로 공유 해야 하는 문제가 발생



2. OAuth 등장배경 (2/2)

- 기존 인증 방식의 경우, 다음과 같은 문제점이 존재한다.
 - Third-Party Application이 직접 사용자의 구글 계정 정보를 보관 (보안 위험)
 - Third-Party Application이 사용자의 모든 리소스에 대한 접근 권한을 가지게 됨 (과도한 권한 부여)
- 이러한 문제를 해결하기 위해 OAuth2.0을 이용하여
Client (Third-Party Application)와 Resource Owner의 자격 증명을 분리하는 인증 및 권한 부여 방식 도입
- 소셜 로그인이 대표적으로 OAuth기반으로 이루어진 기능이다.

3. OAuth가 정의하는 4가지 역할 (1/2)

- OAuth 공식 문서에서는 OAuth를 구성하기 위한 역할을 4가지로 분류 하여 설명하고 있다.
<https://datatracker.ietf.org/doc/html/rfc6749>

1.1. Roles

OAuth defines four roles:

resource owner

An entity capable of granting access to a protected resource.
When the resource owner is a person, it is referred to as an end-user.

resource server

The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

client

An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

authorization server

The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

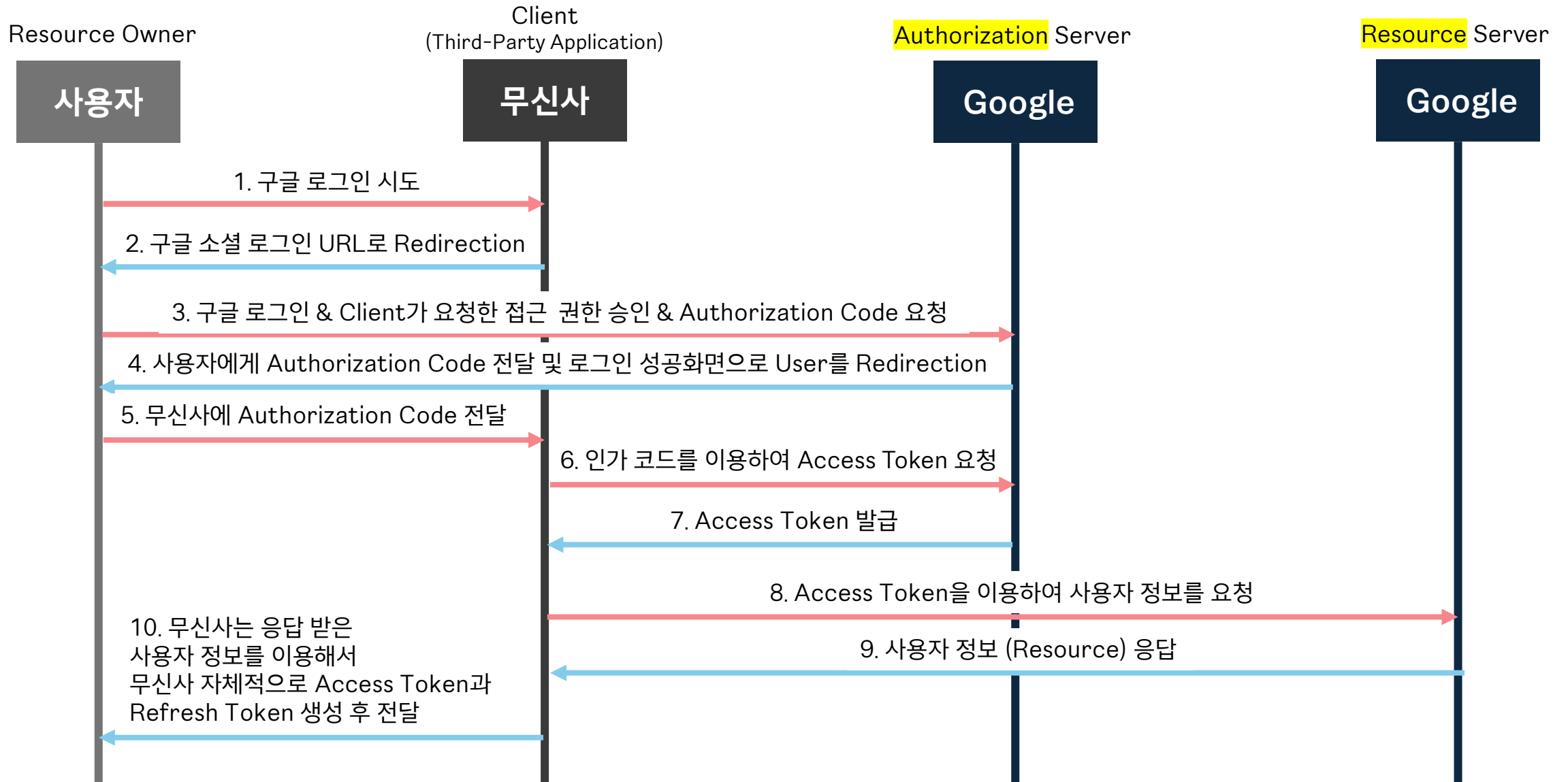
3. OAuth가 정의하는 4가지 역할 (2/2)

- OAuth 공식 문서에서는 OAuth를 구성하기 위한 역할을 4가지로 분류 하여 설명하고 있습니다.
<https://datatracker.ietf.org/doc/html/rfc6749>

	Role	Description
실제 사용자 =	Resource Owner (≡ User)	보호된 리소스에 대한 액세스 권한을 부여할 수 있는 엔티티입니다. 리소스 소유자가 사람인 경우 이를 최종 사용자라고 합니다.
무신사 =	Cilent (Third-Party Application)	리소스 소유자를 대신하여 권한을 가지고 보호된 리소스를 요청하는 애플리케이션입니다.
구글 인증 서버 =	Authorization Sever	리소스 소유자를 성공적으로 인증하고 권한을 획득한 후 클라이언트에 <u>액세스 토큰을 발급하는</u> 서버입니다
구글 리소스 서버 =	Resource Server	보호된 리소스를 호스팅하는 서버로, <u>액세스 토큰</u> 을 사용하여 보호된 리소스 요청을 수락하고 응 답할 수 있습니다.

4. OAuth기반 소셜 로그인 로직 (1/1)

- 만약 사용자가 무신사를 이용하기 위해 구글 로그인을 한다면, 로직은 아래와 같다.



5. 자체 Access Token과 Refresh Token (1/3)

- 이전 페이지의 10번을 보면 다음과 같이 작성되어있다.
 - “응답 받은 사용자 정보를 이용해서 무신사 자체적으로 액세스 토큰과 리프레시 토큰 생성 후 전달”



구글에서 ACCESS TOKEN을 받음에도 불구하고 왜 무신사 자체적으로 Access Token을 생성할까?

- 자체 Access Token을 생성하는 이유
 - 1. 구글 Access Token을 직접 사용하면 보안상 위험이 증가한다.
 - 만약 구글 Access Token을 무신사 API 인증(예시. 로그인 후 장바구니 조회)에 사용하게 된다면
만약 토큰이 유출될 경우, 무신사에서의 유저 데이터 뿐만 아니라 구글에서의 유저 데이터도 가져올 수 있게 된다.
 - 또한, 구글 Access Token의 유효기간은 보통 짧기 때문에 자주 갱신 해줘야 하는 문제가 있다.

5. 자체 Access Token과 Refresh Token (2/3)

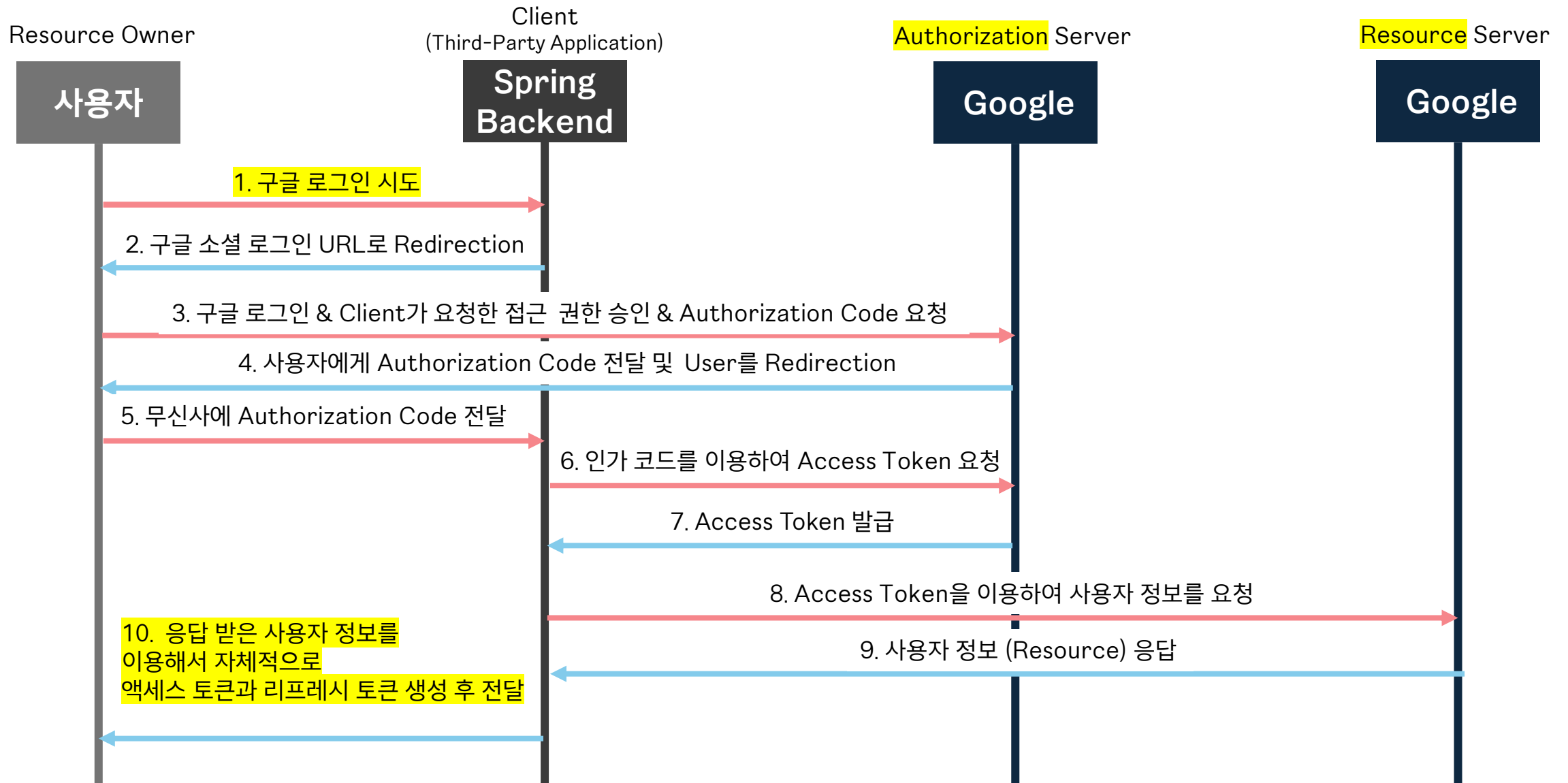
- 자체 Access Token을 생성하는 이유
 - 2. 다양한 OAuth2.0 Provider와 연동 가능
 - 백엔드가 자체적인 JWT 인증을 사용하면, OAuth Provider와 관계없이 일관된 인증 로직을 유지할 수 있다.
 - 3. Stateless (무상태) 인증 방식 적용 가능
 - JWT 기반 인증을 하면, User(= FrontEnd)는 Access Token을 포함하여 API 요청을 보내고, 백엔드는 이 토큰만 검증하면 됨
=> 서버의 부담 감소

5. 자체 Access Token과 Refresh Token (3/3)

- 백엔드는 자체 Access Token 뿐만 아니라 Refresh Token도 만들어서 프론트 엔드에 넘겨준다.
그렇다면 Refresh Token의 역할은 무엇일까?
- Refresh Token은 새로운 Access Token을 발급받기 위한 용도로 사용된다.
 - Access Token이 만료되면, 프론트엔드는 Refresh Token을 다시 백엔드로 보내 새로운 액세스 토큰을 발급받는다.
 - Refresh Token의 유효기간이 다 되었다면, 사용자는 다시 로그인 해야 한다.
 - 마찬가지로 사용자가 로그아웃 한다면, Refresh Token은 사라진다.

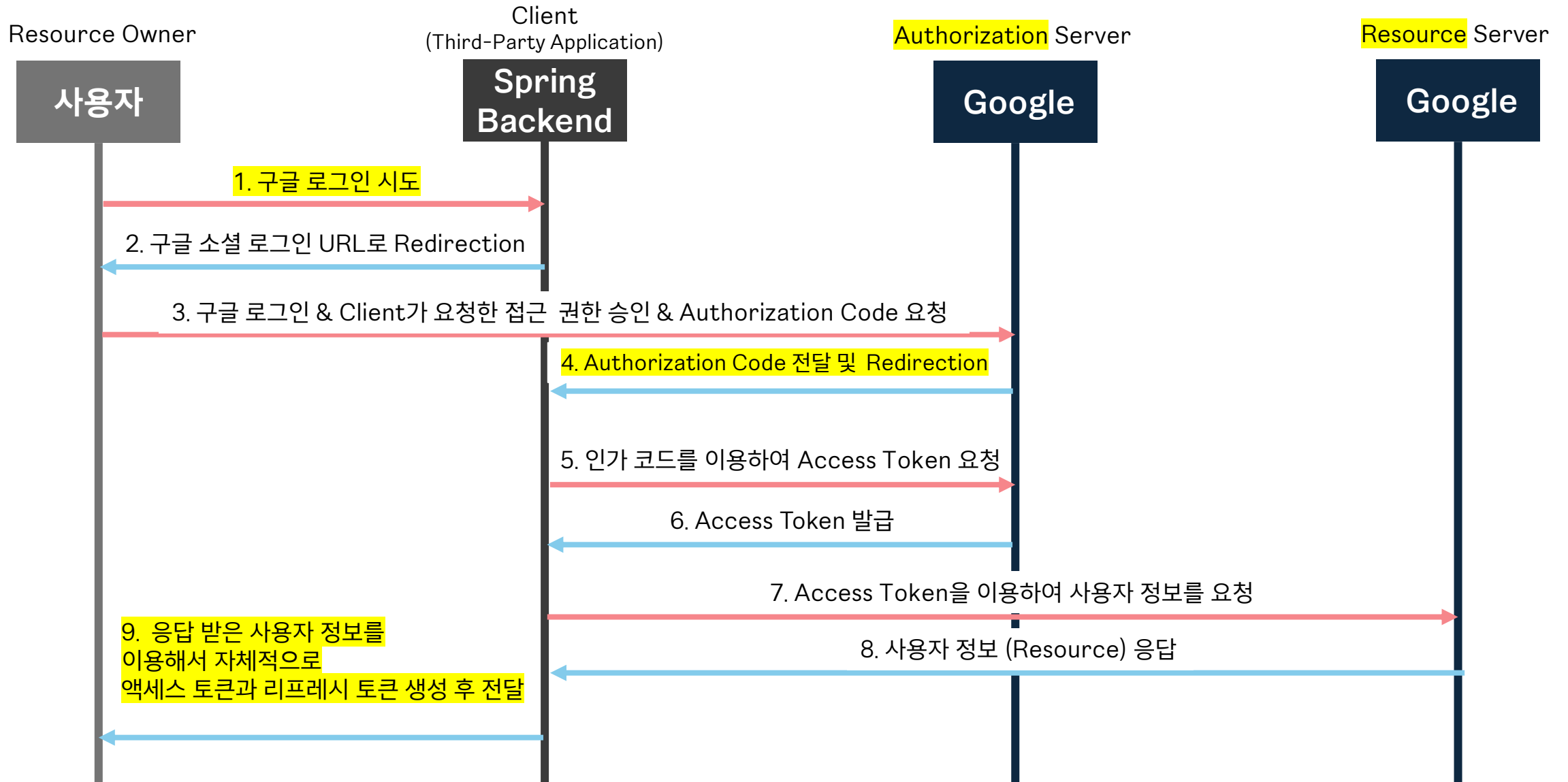
6. Spring Security와 소셜 로그인 (1/5)

- 우리가 만드는 백엔드 서버에서 소셜 로그인 기능을 구현하게 되면, 우리의 서버는 Client 역할에 해당한다.



6. Spring Security와 소셜 로그인 (2/5)

- 우리가 만드는 백엔드 서버에서 소셜 로그인 기능을 구현하게 되면, 우리의 서버는 Client 역할에 해당한다.



6. Spring Security와 소셜 로그인 (3/5)

- 우리가 만드는 백엔드 서버에서 소셜 로그인 기능을 구현하게 되면, 우리의 서버는 Client 역할에 해당한다. (1, 4, 9)
 - 구현해야 할 기능이 많아보이지만 기본적으로 **Spring Security**에 구현이 거의 다 되어있다.
- Spring Security를 사용하면
 - OAuth 2.0 인증 자동 처리, OAuth2.0 Provider와 연동 지원
 - 백엔드 서버 자체 Access Token 발급 과정을 간단하게 처리 가능 (JWT 사용)
- 사용하지 않는다면 ...
 - 1 OAuth2 Authorization Code 요청 처리
 - 2 Authorization Code를 이용한 Access Token 요청
 - 3 Access Token을 이용한 유저 정보 요청
 - 4 Access Token의 유효성 검증 및 보안 처리
 - 5 세션 또는 JWT 기반 인증 방식 적용
 - 6 로그아웃 및 Refresh Token 처리

등을 모두 처리 해야함!! 🤔 🧐 😞

6. Spring Security와 소셜 로그인 (4/5)

- 즉, 우리가 구현 해야 하는 1,4,10번 기능 중에 10번 기능만 구현하면 된다!
 - (1) 구글 로그인 요청
 - : Spring Security에서 기본적으로 제공하는 URL (`http://{domain}/oauth2/authorization/{registrationId}`)을 사용하면 따로 컨트롤러를 만들지 않아도 된다.
 - (4) 로그인 성공 후 Redirection
 - : Spring Security에서 기본적으로 제공하는 URL (`http://{domain}/login/oauth2/code/{registrationId}`)을 사용하면 따로 컨트롤러를 만들지 않아도 된다.

6. Spring Security와 소셜 로그인 (5/5)

- 10번 기능을 좀 더 자세히 설명 하자면, 일반적으로 아래와 같은 로직을 의미한다.
 - ✓ User 검증 (우리 서비스에 이미 등록 되어있는 사용자인가? 처음 등록하는 사용자인가?)
 - ✓ 사용자 정보 기반 JWT 발급
 - : 사용자 인증 정보 (e.g. ID, 이름, 권한 정보) 등을 포함한 Access Token 발급 및 Refresh Token 발급

이 외에도

- ✓ Login Success Handler, Failure Handler
- ✓ Refresh Token을 이용한 Access Token 재발행 로직

을 구현 해주면 된다.

7. 소셜 로그인 구현

- 이제 실제로 구현 해보자! 는 다음 시간에
 - To Be Continue ...
-

- References

- [\[Spring Security\] 백엔드에서 소셜 로그인 구현하기](#)
- [스프링 부트 OAuth2 소셜 로그인 구현하기](#)
- [OAuth 공식문서](#)
- [\[Spring\] Spring Security + OAuth2 + JWT](#)
- [소셜 로그인 DB 설계](#)