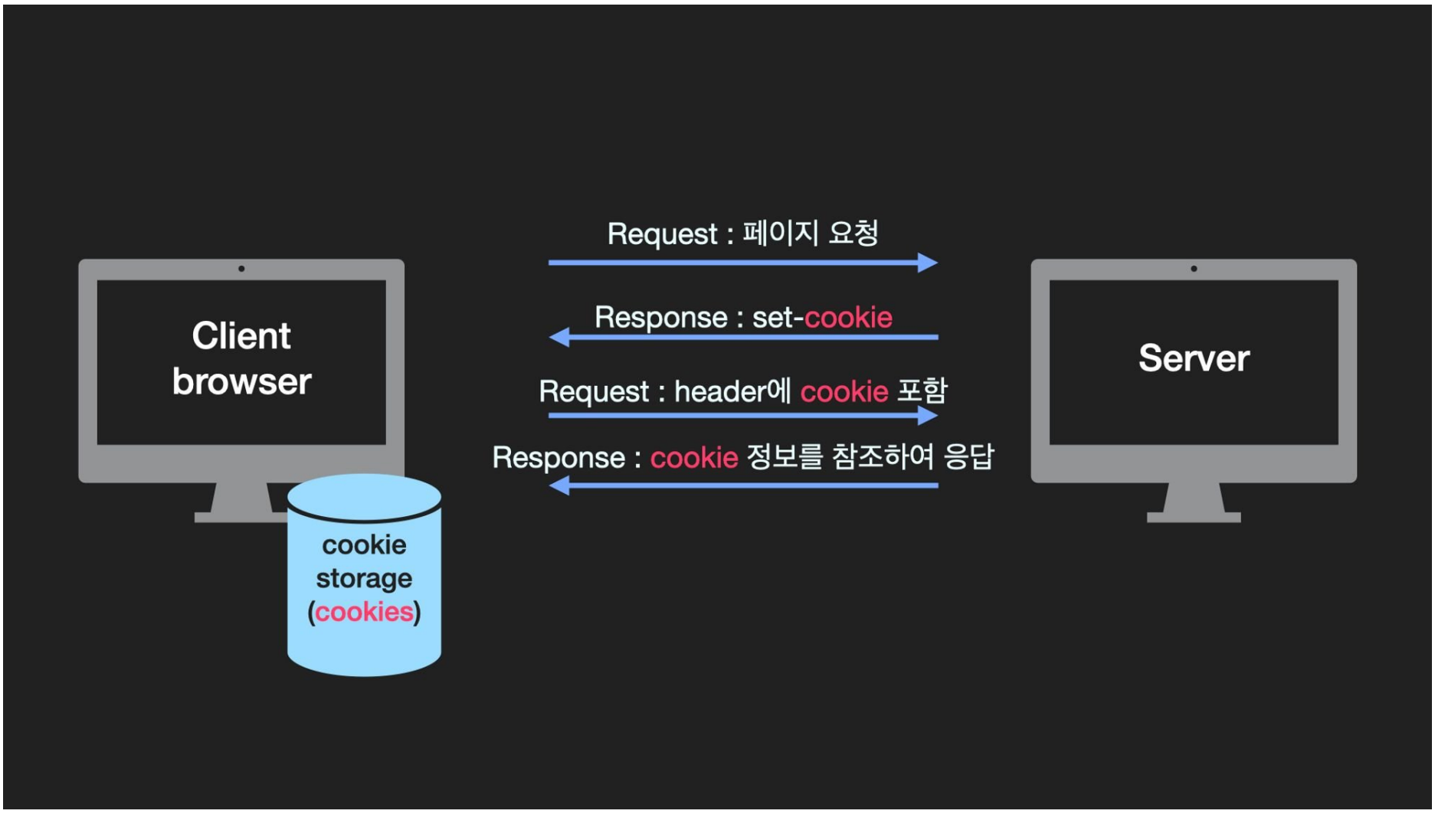
인증, 인가

▼ 등장배경

HTTP 통신에서는 서버와 클라이언트간 통신이 끝나면 연결이 끊어지기 때문에(**비연결성**) 서버는 클라이언트의 상태를 알 수 없다(**무상태성**). 이러한 HTTP의 특성을 보완하기 위해 생겨난 것이 쿠키와 세션이다.

쿠키



- 클라이언트 **로컬에** 저장되는 **key-value** 형태의 데이터
- **쿠키 옵션**
    - **Domain** : 쿠키에 저장할 도메인 정보
    - **Path** : 서버가 라우팅할 때 사용하는 경로
    - **MaxAge or Expires** : 쿠키의 유효기간 정하는 옵션
    - **Secure** : 프로토콜에 따른 쿠키 전송 여부 옵션
    - **HttpOnly** : 자바스크립트로 쿠키에 접근 가능한지 결정하는 옵션
    - **SameSite :** Cross-Origin 요청을 받은 경우 요청에서 사용한 메소드와 해당 옵션의 조합을 기준으로 서버의 쿠키 전송 여부를 결정 다양한 옵션이 있음

    ▼ 상세설명

    ### Domain

쿠키는 도메인(domain)에 종속적입니다. 즉, 쿠키는 특정 도메인에서 생성되어 해당 도메인에만 사용됩니다. 도메인 속성을 지정하지 않을 경우에는 생성된 서버로만 전송되며, 지정할 수 있는 값으로는 현재 서버의 주소 혹은 상위 도메인까지만 지정할 수 있습니다.

도메인은 포트 및 서브 도메인 정보, 세부 경로를 포함하지 않습니다.

쿠키는 생성된 도메인에서만 사용 가능하며, 서브 도메인(subdomain)에서는 케이스에 따라 다르게 공유됩니다.

예를 들어, "example.com" 도메인에서 생성된 쿠키는 "example.com" 도메인과 "www.example.com" 서브도메인에서 사용할 수 있지만, "anotherdomain.com" 도메인에서는 사용할 수 없습니다.

그리고 "example.com" 도메인과 "blog.example.com" 서브도메인은 기본적으로 같은 도메인에 속합니다. 따라서, "example.com" 도메인과 "blog.example.com" 서브도메인 간에는 쿠키가 공유되기 때문에 "example.com" 도메인에서 생성된 쿠키는 "blog.example.com" 서브도메인에서도 사용할 수 있습니다. 특정 도메인에서 생성된 쿠키는 그 하위 도메인에 대해서도 유효하기 때문입니다.

하지만, "blog.example.com"에서 생성된 쿠키는 "example.com" 도메인에서 사용할 수 없습니다. 쿠키는 생성된 도메인에서만 유효하며, 다른 도메인에서는 액세스할 수 없습니다.

또한 서브도메인 간에도 쿠키를 공유할 수 있는 것은 아닙니다. 예를 들어, "blog.example.com"에서 생성된 쿠키는 "mail.example.com"과 같은 다른 서브도메인에서는 사용할 수 없습니다. 이것은 서브도메인이 다른 도메인으로 간주되기 때문입니다.

Path

path 옵션의 특징은 설정된 경로를 포함하는 하위 경로로 요청을 하더라도 쿠키를 서버에 전송할 수 있습니다.

즉 path 가 /users이고 요청하는 세부경로가 /users/zansol 인 경우라면 쿠키전송이 가능합니다.

다만 path 옵션을 만족시키지 못하는 요청은 서버로 쿠키를 전송하지 못합니다.

MaxAge or Expires

만약 쿠키가 영원히 남아있으면 탈취되기도 쉬워지기 때문에 유효기간을 설정하는 것은 보안 측면에서 중요합니다.

MaxAge	쿠키가 유효한 시간을 초단위로 설정하는 옵션입니다
Expires	MaxAge와 비슷하지만 언제까지 쿠키가 유효할지 날짜를 지정해줄 수 있습니다. 이때 옵션의 값은 클라이언트의 시간을 기준으로 하며 지정된 시간, 날짜를 초과하면 쿠키는 파괴됩니다.
쿠키는 위 옵션의 여부에 따라 세션 쿠키와 영속성 쿠키로 나누어집니다.	
세션 쿠키	MaxAge 또는 Expires 옵션이 없는 쿠키로 브라우저가 실행중일 때 임시로 사용하는 임시 쿠키입니다. 브라우저를 종료하면 해당 쿠키는 삭제됩니다.
영속성 쿠키	브라우저의 종료 여부와 상관없이 MaxAge 또는 Expires에 지정된 유효시간 만큼 사용가능한 쿠키입니다.

Secure

만약 Secure 옵션이 true로 설정되면 HTTPS를 사용하는 경우에만 쿠키를 전송할 수 있습니다.

반대로 Secure가 false면 HTTP에도 쿠키를 전송할 수 있습니다.

단 도메인이 localhost인 경우에는 HTTPS가 아니여도 쿠키 전송이 가능합니다. (개발 단계에서 localhost를 많이 사용하기에 생긴 예외)

HttpOnly

true로 설정된 경우 자바스크립트로 쿠키에 설정이 불가능하게 됩니다.

옵션을 명시하지 않은 경우에는 기본적으로 false가 지정됩니다.

이 옵션이 false인 경우에는 document.cookie를 이용해

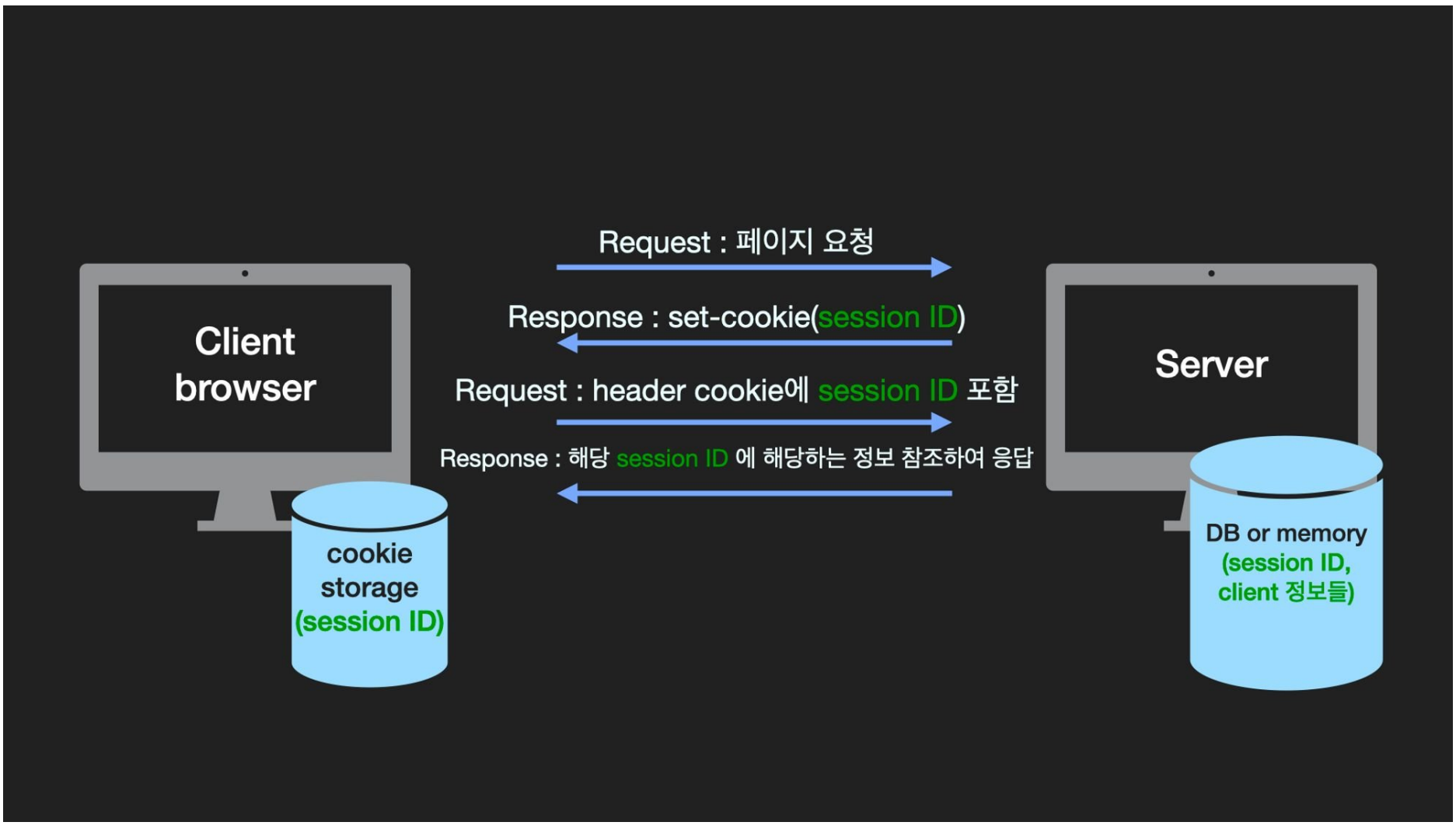
자바스크립트로 쿠키에 접근할 수 있으므로 쿠키가 탈취될 수 있습니다.

SameSite

Lax	Cross-Origin 요청이라면 GET 메소드에 대해서만 쿠키를 전송할 수 있습니다.
Strict	단어 그대로 가장 엄격한 옵션으로 Cross-Origin이 아닌 same-site인 경우에만 쿠키를 전송할 수 있습니다.
None	Cross-Origin에 대해 가장 관대한 옵션으로 항상 쿠키를 보내줄 수 있습니다. 다만 쿠키 옵션 중 Secure 옵션이 필요합니다.

- **장점**
  - 사용자 입장에서 별도의 인증 과정을 거치지 않고 서버가 나를 쉽게 기억하도록 할 수 있다.
  - ID-PW를 굳이 입력하지 않아도 되는 자동완성 기능, 지난번 읽었던 글이 다시 새로운 글에 업데이트 되지 않도록 하는 기능 등 무상태성인 HTTP에게 상태 정보를 기억하게 하는 가벼운 일에 쿠키가 쓰인다.
- **단점**
  - 쿠키값은 **쉽게 수정 가능**하다. - **악의적으로 변조**될 가능성이 크다.
  - **보안적 약점**이 있다보니 그다지 중요하지 않은 정보만 저장해야 한다.

세션



- 쿠키를 이용하는 방식으로 쿠키에 세션 ID만 저장하고 정보는 **서버 측**에서 저장하고 관리한다.
- 웹 브라우저가 서버에 접속해 **브라우저를 종료할 때까지 세션을 유지**한다
- **장점**

    - 세션 id를 제외한 나머지 상태 데이터들이 서버에 저장되기 때문에, 클라이언트에 저장되는 **쿠키보다 보안성이 높다.**
    - 로그인 정보를 유지하여 자동 로그인 가능
- **단점**
    - 상태 데이터를 서버에서 가지고 있는 만큼 **부하가 늘어난다**는 단점이 있다. 즉, 서버에 세션이 많이 존재하면 서버가 터질 수도 있다.
    - 서버에서 데이터를 바로 읽어 처리할 수 있는 쿠키에 반해, 세션은 세션 id를 통해 세션을 읽어온 다음 또 데이터를 읽어야 하기 때문에 **속도가 상대적으로 느리다.**
    - stateless를 위배한다. 서버에 상태를 저장해야 하기 때문에 해당 서버를 늘리면(scale out) 따로 세션 ID를 또 저장해야 한다.
    - 세션 저장소에 문제가 생기면 이전에 저장된 유저들의 인정이 불가해진다.

쿠키 VS 세션

쿠키

- 클라이언트에 저장
- 만료시간 동안 파일로 저장되므로 브라우저를 종료해도 정보가 남아있다.

세션

- 서버에 저장 > 서버 자원 사용 > 서버에 요청을 보내는 사용자가 많을 경우 부하가 심할 수 있음
- 브라우저가 종료되면 만료시간에 상관없이 삭제된다.
- 쿠키를 이용해 쿠키에 세션 ID만 저장하고 서버에서 세션을 처리하기 때문에 **비교적 보안성이 좋다.**

	쿠키	세션
저장 위치	클라이언트	서버
보안	안좋음	좋음
서버 부하	없음	높음
생명주기	브라우저 종료해도 유지됨	브라우저 종료시 소멸
속도	빠름	느림

JWT(Json Web Token)

- Header
    - 토큰의 타입과 해시 알고리즘(서명 생성에 사용되는 알고리즘) 정보가 들어간다.
    - BASE64 방식으로 인코딩해서 JWT의 가장 첫 부분에 기록된다.
- Payload
    - exp와 같이 만료시간을 나타내는 공개 클레임
    - 클라이언트와 서버간 협의하에 사용하는 비공개 클레임
    - 그 외 어떤 내용이던 상관없이 추가 가능
    - 위의 두가지 요소를 조합하여 작성한뒤 BASE64 인코딩하여 두번째 요소로 위치한다.
    - 예시 - {'userId' : 1, 'iat': 15395134325 }
- Signature
    - JWT가 원본 그대로라는 것을 확인할 때 사용하는 부분
    - 시그니쳐는 BASE64URL 인코드된 **header**와 **payload** 그리고 JWT secret(별도 생성)을 헤더에 지정된 암호 알고리즘으로 암호화하여 전송한다. (복호화 가능)
- **장점**
    - 이미 인증된 정보이기 때문에 세션 저장소와 같은 별도의 인증 저장소가 필수적이지는 않다
    - 세션과 다르게 클라이언트의 상태를 서버가 저장해두지 않아도 된다. → stateless 보장
    - Header와 Payload를 가지고 Signature를 생성하므로 데이터 위변조를 막을 수 있다.

    - 확장성이 우수하다
        - 토큰 기반으로 다른 로그인 시스템에 접근 및 권한 공유가 가능
- **단점**
    - base64 인코딩을 통한 정보를 전달하기 때문에 전달량이 많다. → 네트워크 전달 시 많은 데이터양으로 부하가 생길 수 있다.
    - payload에는 암호화가 되어있지 않기 때문에 민감 정보를 저장할 수 없다.
    - **토큰이 탈취당하면 만료될 때까지 대처가 불가능하다 → 유효시간을 줄여서 해결**

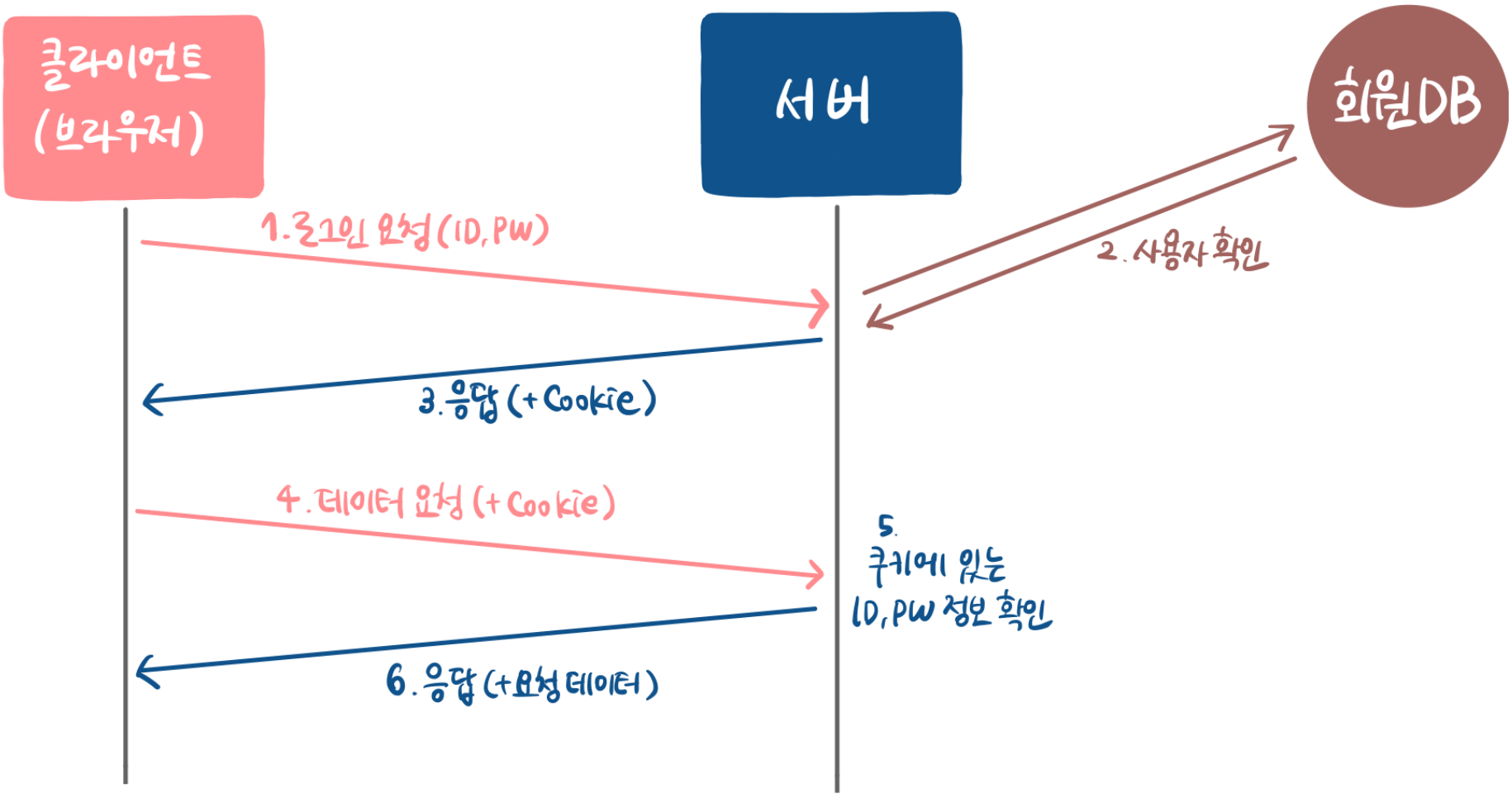
        ▼ 짧은 유효시간을 보완하는 방법

        1. sliding session
            - 특정한 서비스를 계속 사용하고 있는 특정 유저에 대한 유효시간을 연장시켜주는 방법
            - 접속이 단발성이라면 이 방법이 통하지 않음
            - 너무 긴 access token을 발급시켜 준 상황에서 이 방법을 쓰면 무한정 사용하는 상황이 발생 가능
        2. refresh token
            - access token과 함께 비교적 긴 시간(7일, 30일 등)의 유효시간을 가진 refresh token이라는 토큰을 발급하는 방법
            - 클라이언트가 access token이 만료됨을 알게 되거나 서버로부터 만료됨을 확인받으면 refresh token으로 서버에게 새로운 access token을 발급하도록 요청하여 발급받는 방식

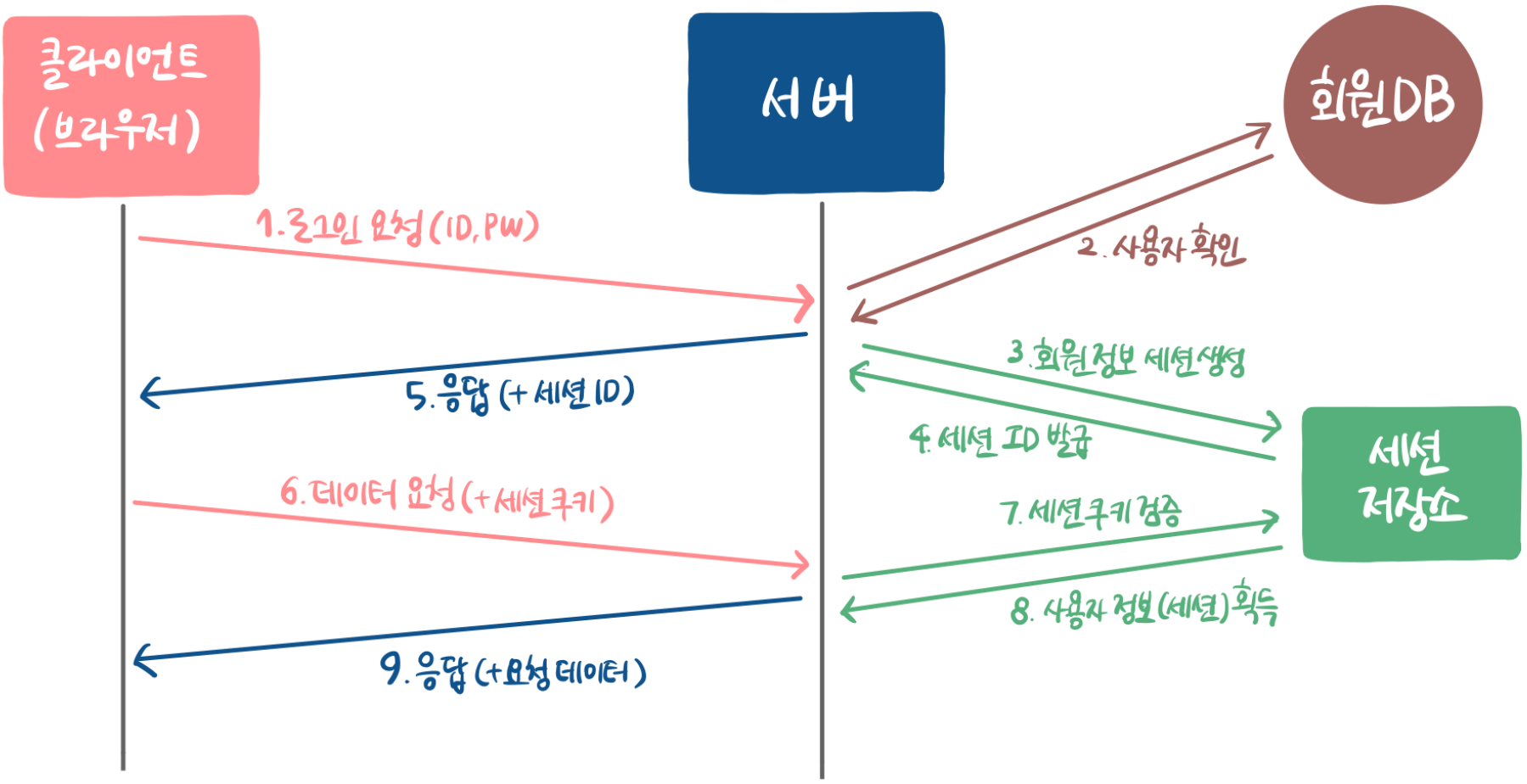
인증/인가

- 인증(authentication) : 사용자가 누구인지 확인하는 절차 ex) 회원가입과 로그인
- 인가(authorization) : 사용자가 요청하는 것에 대한 권한이 있는지를 확인하는 절차

쿠키를 통한 인증/인가

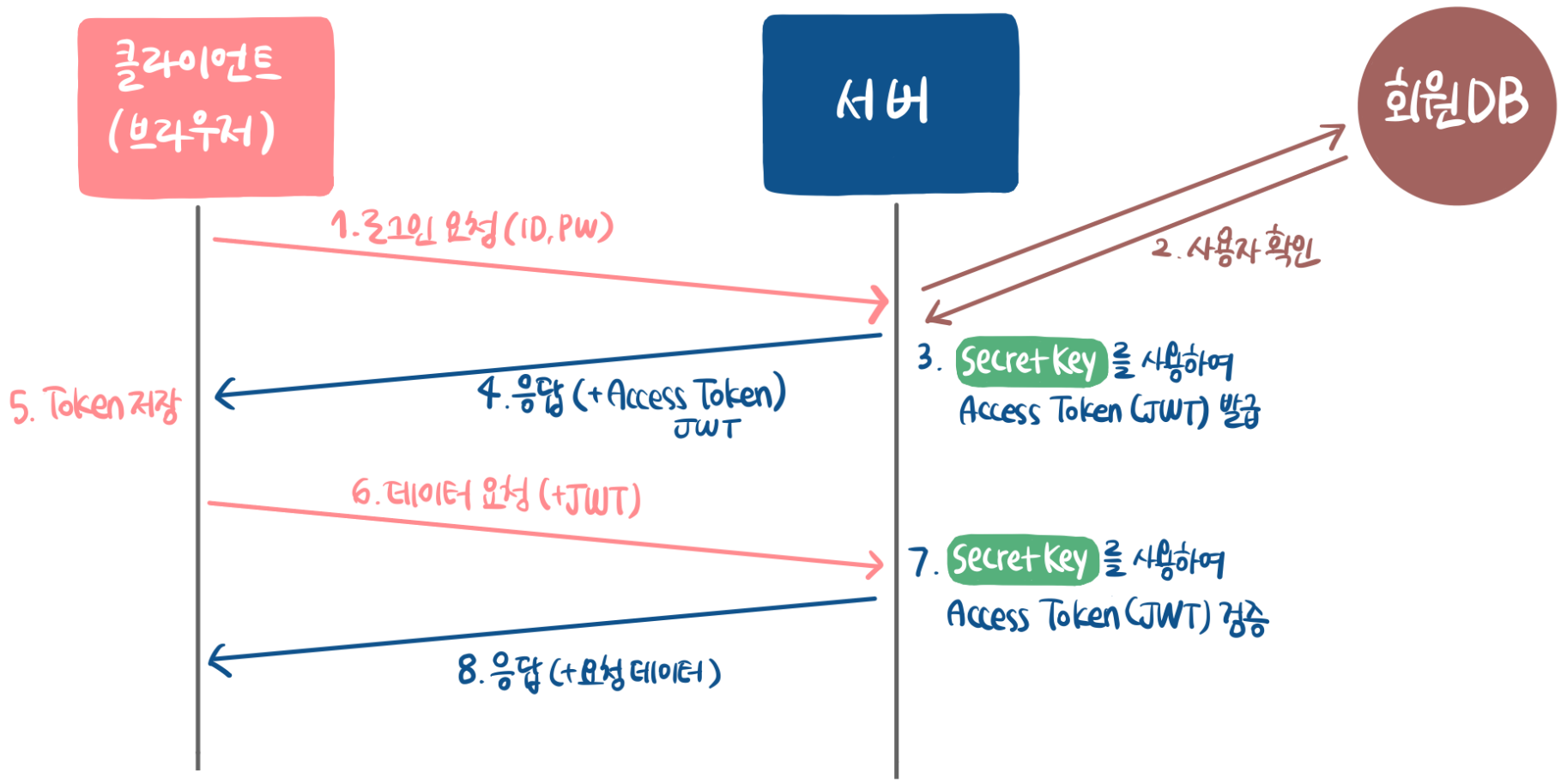


세션을 통한 인증/인가



1. 클라이언트가 로그인 하면 서버는 회원정보를 대조하여 인증한다. (인증)
2. 회원(클라이언트) 정보를 세션 저장소(서버에 있음)에 생성하고 session ID를 발급한다.
3. http response header 쿠키 부분에 발급한 session ID를 담아서 보낸다.
4. 클라이언트에서는 session ID를 쿠키 저장소에 저장하고 이후에 http 요청을 보낼 때마다 쿠키에 session ID를 담아서 보낸다.
5. 서버에서 쿠키에 담겨 온 session ID에 해당하는 회원 정보를 세션 저장소에서 가져온다. (인가)
6. 회원 정보를 바탕으로 처리된 데이터를 응답 메시지에 담아서 클라이언트에 보낸다.

JWT를 통한 인증/인가



1. 클라이언트가 ID, PW 로그인 요청을 보낸다.
2. 회원 DB에서 사용자를 확인한다.

3. 로그인 성공 시, 서버는 로그인 정보를 Payload에 담고, Secret Key를 사용해서 Access Token(JWT)을 발급한다.
4. 서버가 JWT를 Client에 전달한다. 이 때 전달방법은 개발자가 정한다.

   ex) 응답 Header에 `Authorization: BEARER <JWT>` 형태로 전달
5. 클라이언트는 전달받은 토큰(JWT)을 저장한다.(쿠키, local storage 등)
6. 클라이언트가 서버에 요청할 때마다 토큰(JWT)을 요청 Header의 `Authorization` 에 포함시켜 함께 전달한다.
7. 서버는 클라이언트가 전달한 토큰의 Signature를 secret Key로 복호화한 후, 위변조 여부 및 유효기간을 검증한다.
8. 검증에 성공하면, JWT에서 사용자 정보를 확인하고 요청에 응답한다.