Authorization

▼ 인증, 인가

인증(Authentication)

- 유저가 **누구인지 확인**하는 절차
- 사용자의 신원을 검증하는 프로세스
- ex) ID, PW 로 로그인하는 행위

인가(Authorization)

- 인증 이후의 프로세스로, **인증된 유저가 어떠한 자원에 접근할 수 있는지** 확인하는 절차
- ex) 관리자 페이지 - 관리자 권한

▼ 쿠키

클라이언트에 저장되는 key-value 형태의 데이터 파일

필요 이유

HTTP의 **connectionless(비연결성)**, **stateless(비상태성)** 때문

클라이언트가 요청(request)을 했을 때 그 요청에 맞는 응답(response)을 보낸 후 연결을 끊고, **서버는 클라이언트에 대한 상태 정보를 유지하지 않기** 때문에 사용자가 로그인을 통해 **인증을 거쳐도, 이후 요청에 대해서는 인증된 상태를 유지할 수 없음**

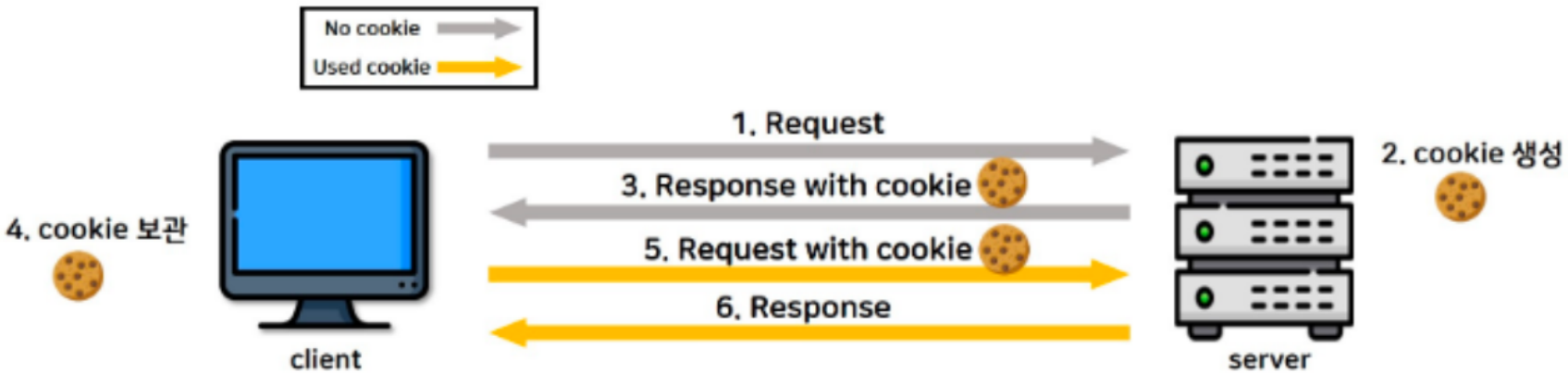
쿠키의 구성요소

- `Name(이름)` : 쿠키를 구별하는 데 사용되는 키 (중복 불가)
- `Value(값)` : 쿠키의 값
- `Domain(도메인)` : 쿠키가 저장된 도메인
- `Path(경로)` : 쿠키가 사용되는 경로
- `Expires(만료기한)` : 쿠키의 만료기한

특징

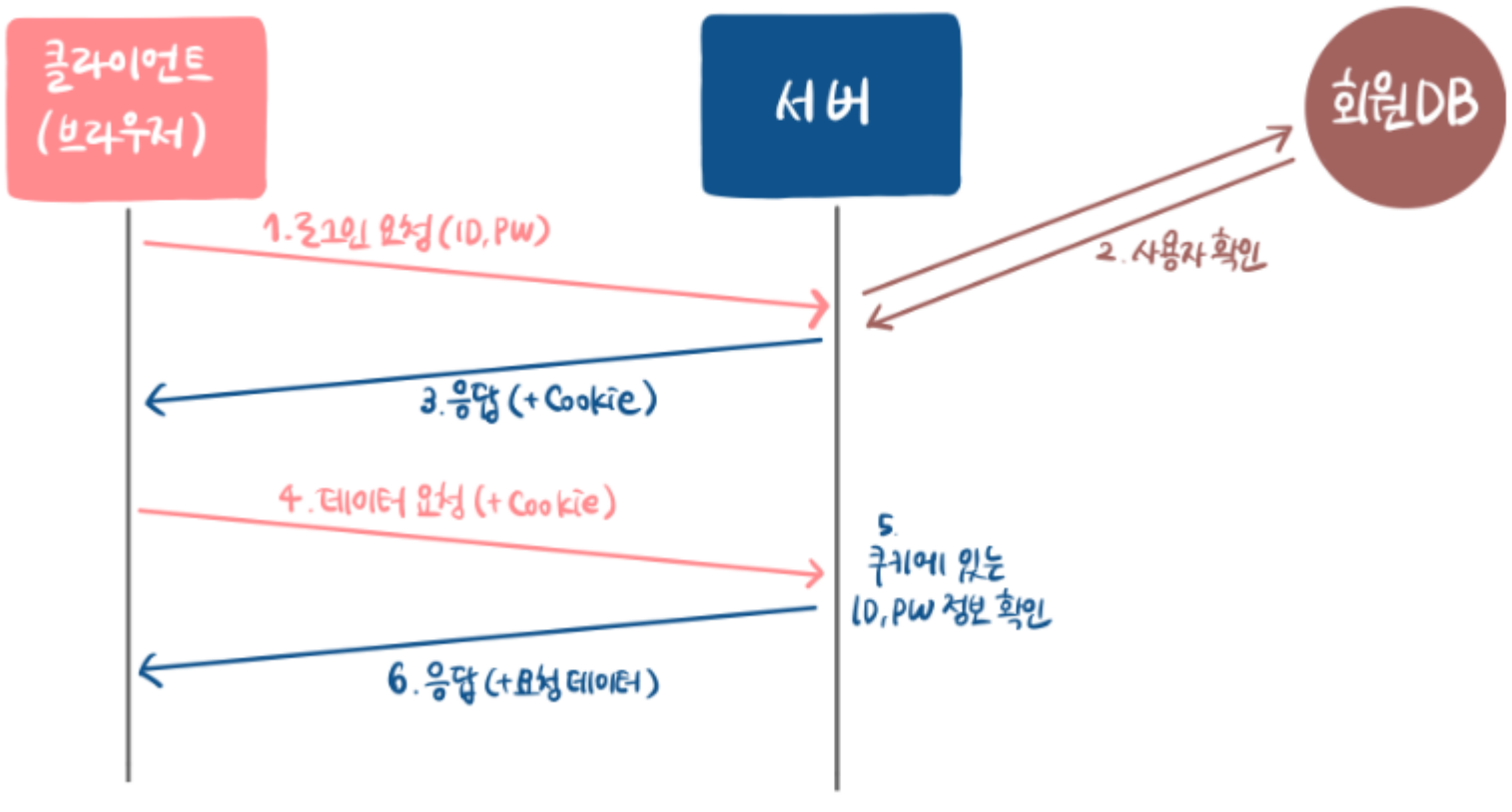
- 클라이언트에 **총 300개의 쿠키를 저장** 가능
- 하나의 **도메인 당 20개의 쿠키**를 가질 수 있음
- 하나의 쿠키는 **4kb까지 저장 가능**
- **브라우저 단위로 쿠키 생성**
    - 같은 기기로 접속해도 크롬으로 접속해서 발행한 쿠키와 safari로 접속해서 발행하는 쿠키는 다름
- 다른 도메인을 대신하여 쿠키 발급 불가
- 만료 시간까지 상태 정보 유지

동작순서



1. 클라이언트가 페이지를 요청
2. 웹 서버는 쿠키 생성
3. 서버는 클라이언트에게 보내는 응답의 header에 쿠키를 담아 보낸다.
4. 넘겨 받은 쿠키는 브라우저 쿠키저장소에 저장
5. 동일 사이트 재방문 시 쿠키저장소에 해당 쿠키가 있는 경우, 요청 페이지와 함께 헤더에 쿠키를 담아 전송한다.

인증/인가 동작순서



1. 클라이언트에서 ID,PW정보를 담아 로그인 요청한다
2. 서버에서는 DB에서 사용자 확인을 한다.
3. 서버는 ID, PW 정보를 쿠키에 담아 브라우저로 다시 보낸다.
4. 이후 브라우저에서는 요청할 때마다 로그인 정보가 담긴 쿠키를 함께 서버로 보낸다.

사용 예시

- **세션관리**
  - 서버에 저장해야할 로그인, 장바구니, 게임 스코어 등의 정보 관리
  - ex) 로그인 유지, 장바구니에 담은 제품 유지
- **개인화**
  - 사용자 선호, 테마 등 사용자의 개인 세팅을 저장, 관리
  - ex) 다크모드 사용, 언어설정, 메뉴 순서 최적화 등
- **트래킹**

    - 분석 및 광고 개제를 위해 웹사이트 내 사용자 행동 기록, 관리
    - ex) 분석데이터 수집, 리타게팅 광고에 기여 등
- 팝업창의 "일주일간 다시 보지 않기" 체크
- 방문했던 사이트에 다시 방문 하였을 때 아이디와 비밀번호 자동 입력

장점

- 기존 로그인 정보를 사용하기 때문에 인증을 위한 추가적인 데이터 저장이 필요 없다.

단점

- 사용자의 주요 정보를 매번 요청에 담기 때문에 **보안상 문제**가 있다.
- **클라이언트에서 쿠키 정보를 쉽게 변경, 삭제**할 수 있고, **가로채기** 당할 수도 있다.
- 쿠키에는 용량 제한이 있어 많은 정보를 담을 수 없다.
- 쿠키 사이즈가 커질수록 네트워크 부하가 심해진다.

▼ 세션

서버에서 일정시간 동안 클라이언트 상태를 유지하기 위해 사용

- 일정 시간: 방문자가 웹 브라우저를 통해 웹 서버에 접속한 시점부터 웹 브라우저를 종료하여 연결을 끝내는 시점

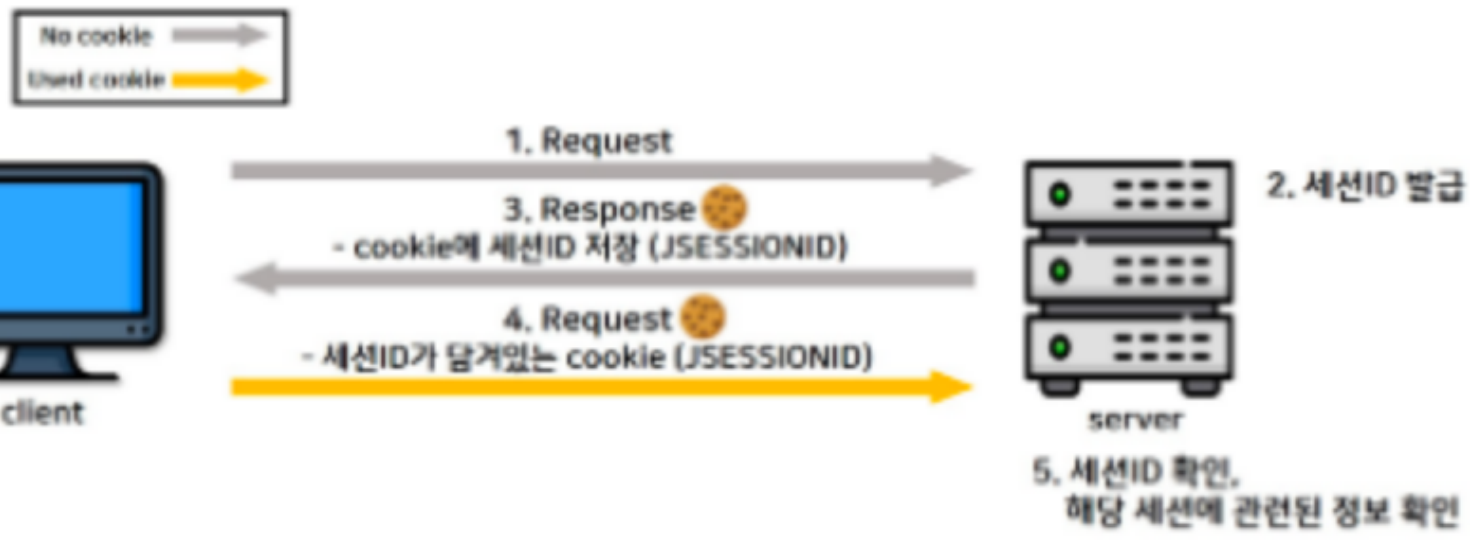
필요 이유

쿠키는 클라이언트측에서 쉽게 변경할 수 있고, 정보가 바로 보이기 때문에 **보안에 취약**하기 때문

특징

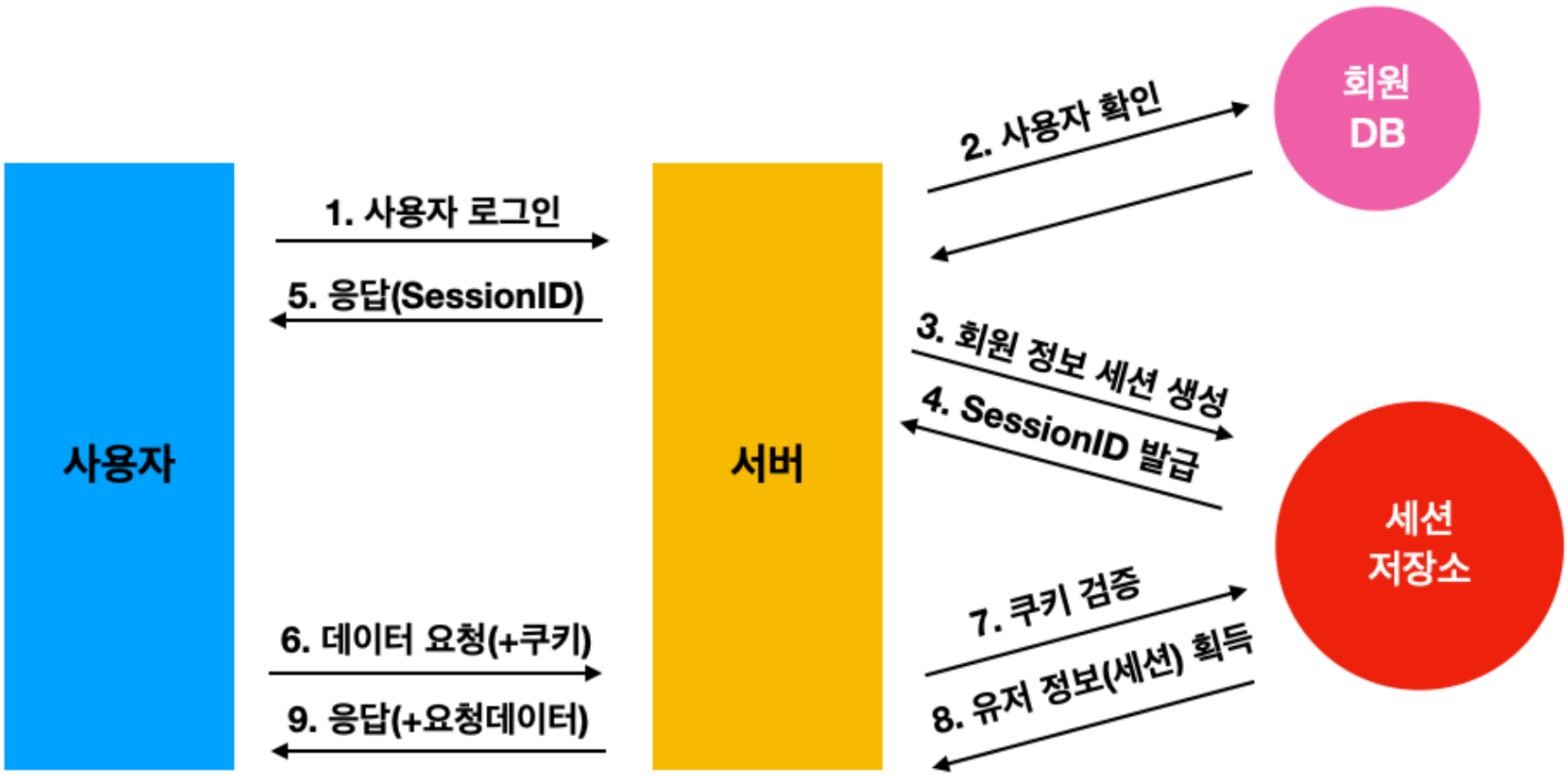
- **웹 서버**에 웹 컨테이너의 상태를 유지하기 위한 정보**(세션정보)를 저장**한다.
- 서버에서 클라이언트별 유일한 **세션 ID를 부여**한다.
    - 세션 ID : 사용자의 주요 정보가 아닌, 단지 사용자를 식별할 수 있는 값을 생성한다. → **보안 강화**
- **브라우저를 닫거나, 서버에서 세션을 삭제했을때만 삭제**가 되므로, 쿠키보다 비교적 보안이 좋다.
- 저장 데이터에 제한이 없다.

동작순서



1. 클라이언트가 페이지를 요청한다.
2. 서버는 접근한 클라이언트의 Request-Header 필드인 Cookie를 확인하여, 클라이언트가 해당 session-id를 보냈는지 확인한다. session-id가 존재하지 않는다면 서버는 session-id를 발급하고, 세션스토리지에 저장한다.
3. 세션 ID를 담은 세션 쿠키와 함께 응답을 전송한다.
4. 클라이언트는 요청할 때마다 세션 ID가 담긴 쿠키를 함께 전송한다.
5. 서버는 세션 ID를 확인하고 이를 통해 클라이언트 정보를 가져와 활용한다.

인증/인가 동작순서



1. 사용자가 로그인을 요청
2. 서버 DB에서 계정 정보를 읽어 사용자를 확인
3. 사용자의 고유한 Id를 부여하여 세션 저장소에 저장한후
4. 이와 연결된 세션Id를 발급한다.
5. 사용자는 서버에서 해당 세션Id를 받아 쿠키에 저장
6. 그 후, 인증이 필요한 요청마다 쿠키를 헤더에 실어 보낸다.
7. 서버는 쿠키를 받아 세션 저장소에서 대조 후
8. 대응되는 정보를 가져온다.
9. 인증이 완료되면 서버는 사용자에 맞는 데이터를 보내준다.

사용 예시

- 화면이 이동해도 로그인이 풀리지않고 로그아웃되기 전까지 유지되는 것

장점

- 사용자의 **로그인 정보를 주고 받지 않기** 때문에 **상대적으로 안전**하다.
- 사용자마다 **고유한 세션 ID가 발급**되기 때문에, 요청이 들어올 때마다 **회원DB를 찾지 않아도 된다.**

단점

- 서버 세션 저장소를 사용하므로 **요청이 많아지면 서버 부하**가 심해진다.
- 서버 자원을 사용하기 때문에 유저가 많아지면 **저장 공간에 대한 비용** 발생한다.
- 사용자를 식별할 수 있는 값인 세션 ID를 생성하고, 서버에 저장해야하는 작업이 생긴다.
- 탈취자가 세션Id 자체를 탈취하여 **클라이언트인척 위장**할 수 있다는 한계가 있다.

쿠키 vs 세션

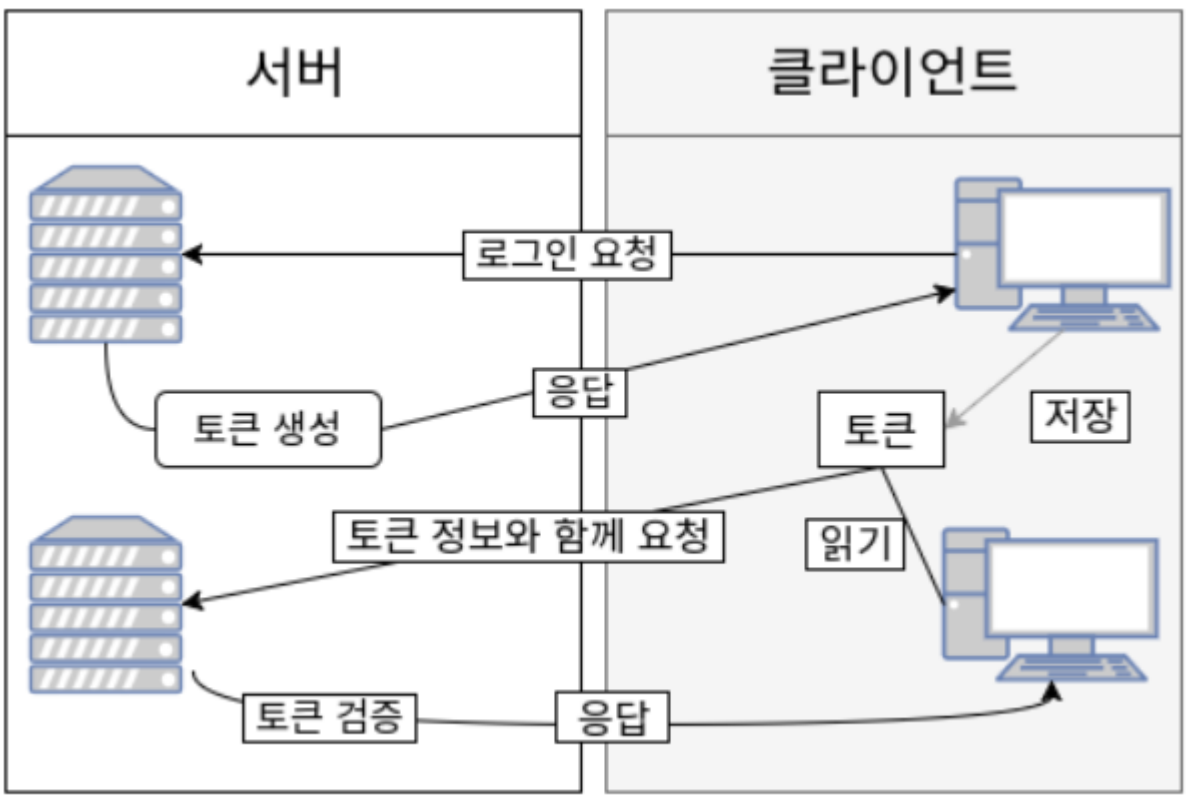
	쿠키(Cookie)	세션(Session)
저장위치	클라이언트(로컬 PC)	웹서버
저장 형식	text	Object
만료시점	쿠키 저장시 설정->(만료시점 전까지는 사라지지 않음)	브라우저 종료시 삭제->(기간 지정 가능)
사용하는 자원	클라이언트 리소스	웹 서버 리소스
용량 제한	총 300개, 하나의 도메인당 20개, 하나의 쿠키당 4KB	서버가 허용하는 한 무제한
속도	세션보다 빠름	쿠키보다느림
보안	세션보다 취약	쿠키보다 좋음

▼ 토큰(JWT)

필요이유

세션 인증은 서버가 세션정보를 가지고 있어야 하고, 이를 조회하는 과정이 필요하기 때문에 많은 오버헤드가 발생했다. 하지만, 토큰은 **클라이언트에 저장되기 때문에 서버의 부담을 덜 수** 있다.

토큰 작동방식



1. 사용자가 아이디와 비밀번호로 로그인을 한다.
2. 서버 측에서 사용자(클라이언트)에게 유일한 토큰을 발급한다.
3. 클라이언트는 서버 측에서 전달받은 토큰을 쿠키나 스토리지에 저장해 둠
4. 서버에 요청을 할 때마다 해당 토큰을 찾아서 HTTP 요청 헤더에 포함시켜 전달한다.
5. 서버는 전달받은 토큰을 검증하고 요청에 응답한다.

토큰 장점

- 세션과 달리 **클라이언트에 저장** -> **서버부담 줄일수있음**
- 토큰 자체에 데이터가 들어가있어서 토큰을 판별만하면됨
- **Stateless함**

토큰 단점

- 쿠키/세션과 다르게 **토큰 자체의 데이터 길이가 길어**, 인증 요청이 많아질수록 네트워크 부하가 심해질수 있다.
- **Payload 자체는 암호화되지 않기** 때문에 유저의 중요한 정보는 담을 수 없다.
- 토큰을 탈취당하면 대처하기 어렵다.

  ⇒ 따라서 사용 기간 제한을 설정하는 식으로 극복

JWT(JSON Web Token)

인증에 필요한 정보들을 암호화시킨 JSON토큰

- JWT 기반인증은 JWT 토큰을 HTTP 헤더에 실어 서버가 클라이언트를 식별하는 방식이다
- JWT는 JSON 데이터를 **Base64 URL-safe Encode** 를 통해 인코딩하여 직렬화한 것이며,
- 토큰 내부에는 위변조 방지를 위해 개인키를 통한 **전자서명**도 들어있다.

구조

JWT는 `.`을 구분자로 나누어지는 세가지 문자열의 조합 (**.** 을 기준으로 Header, Payload, Signature을 의미)

Header

- JWT에서 사용할 타입과 해시 알고리즘의 종류가 담겨있음

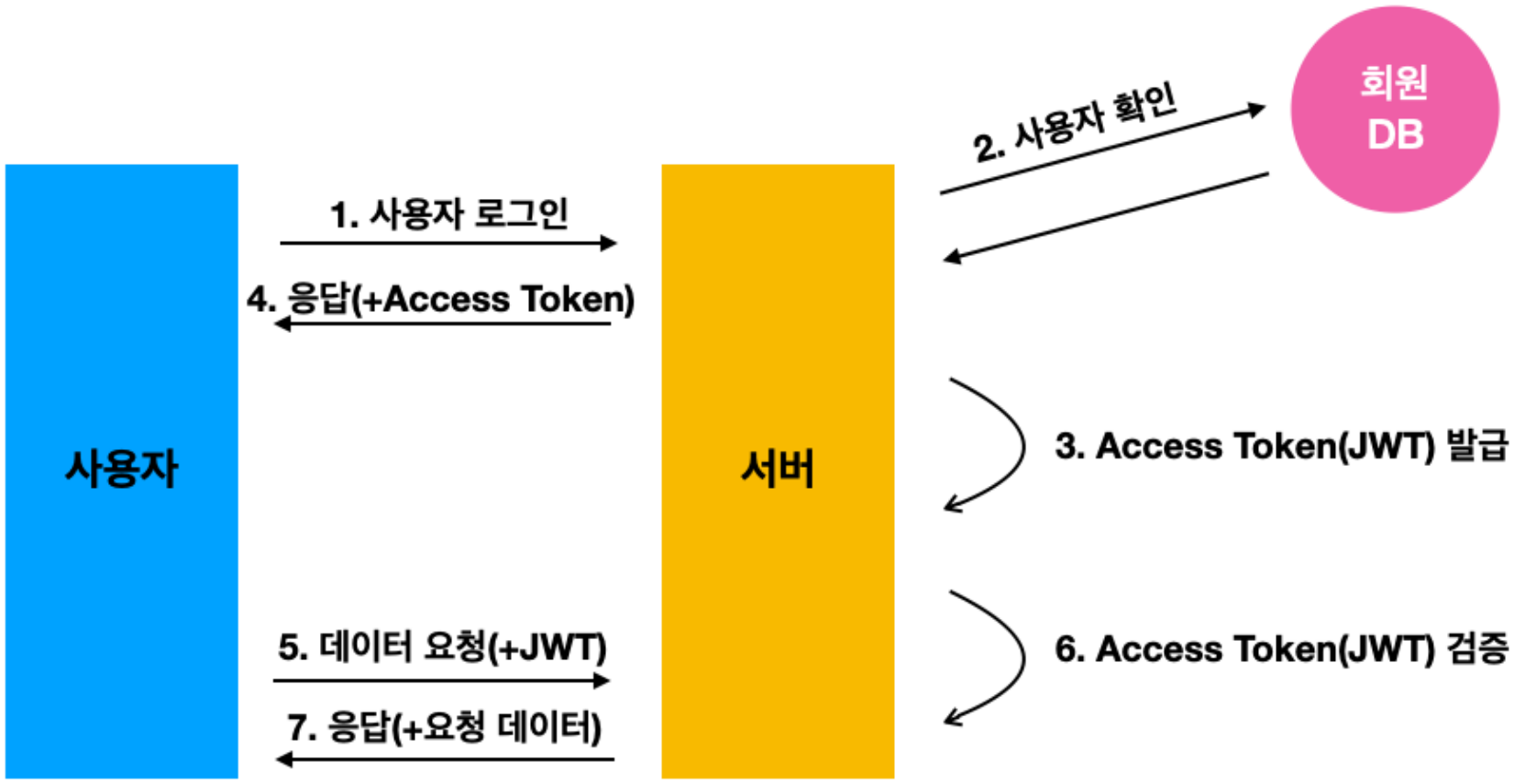
Payload

- 서버에서 첨부된 사용자 권한 정보와 데이터가 담겨있음
- 즉, 서버와 클라이언트가 주고받는 시스템에서 실제로 사용될 정보에 대한 내용을 담고 있는 섹션

Signature

- Payload를 Base 64 URL-safe Encode를 한 이후 HEader에 명시된 해시함수를 적용, 개인키로 서명한 전자서명이 담겨있음
- 전자서명에는 비대칭 암호화 알고리즘을 사용하므로 암호화를 위한 키와 복호화를 위한 키가 다르다. 암호화(전자서명)에는 개인키를, 복호화(검증)에는 공개키를 사용

인증/인가 동작순서



1. 사용자가 로그인을 한다.
2. 서버에서는 계정 정보를 읽어 사용자를 확인 후
3. 사용자 고유ID값을 부여한 후, 기타 정보와 함께 Payload에 넣는다. JWT의 유효기간 설정하고 암호화할 SECRET KEY를 이용하여 Access Token을 발급 한다.
4. 사용자는 Access Token을 받아 로컬 스토리지(혹은 쿠키)에 저장한 후,
5. 인증이 필요한 요청마다 토큰을 헤더에 실어서 보낸다.
6. 서버에서는 해당 토큰의 Verify Signature을 SECRET KEY로 복호화한 후, 조작여부, 유효기간을 확인한다.
7. 검증이 완료되면, Payload를 디코딩하여 사용자의 ID에 맞는 데이터를 가져온다.

장점

- Header 와 Payload를 가지고 Signature를 생성함으로 **데이터 위변조를 막을 수 있음**
- 인증 정보에대한 **별도의 저장소가 필요없음**
- JWT는 토큰에 대한 기본정보와 전달할 정보 및 토큰이 검증되었음을 증명하는 서명 등 필요한 모든 정보를 자체적으로 가짐
- **무상태**
- **확장성 우수**
- 토큰 기반으로 **다른 로그인 시스템에 접근 및 권한 공유가 가능** (쿠키와 차이)
    - OAuth의 경우 Facebook, Google 등 소셜계정을 이용하여 다른 웹서비스에서도 로그인 가능
- **모바일 어플리케이션 환경에서도** 잘 작동

단점

- **구현 복잡도가 증가**한다.
- JWT에 담는 내용이 커질수록 네트워크 비용이 증가한다.
- **이미 발급된 JWT에 대해서는 돌이킬 수 없다.**
    - JWT는 한번 발급되면 유효기간이 완료될 때까지 계속 사용이 가능하다. (토큰의 유효기간을 너무 길게 잡으면 안된다.)
- Secret Key 유출 시 JWT 조작이 가능하다.
- Payload 자체는 암호화되지 않기 때문에 **사용자의 중요한 정보는 담을 수 없다**.
- Access token 탈취당하면 사용자와 똑같은 지위를 가질 수 있다.

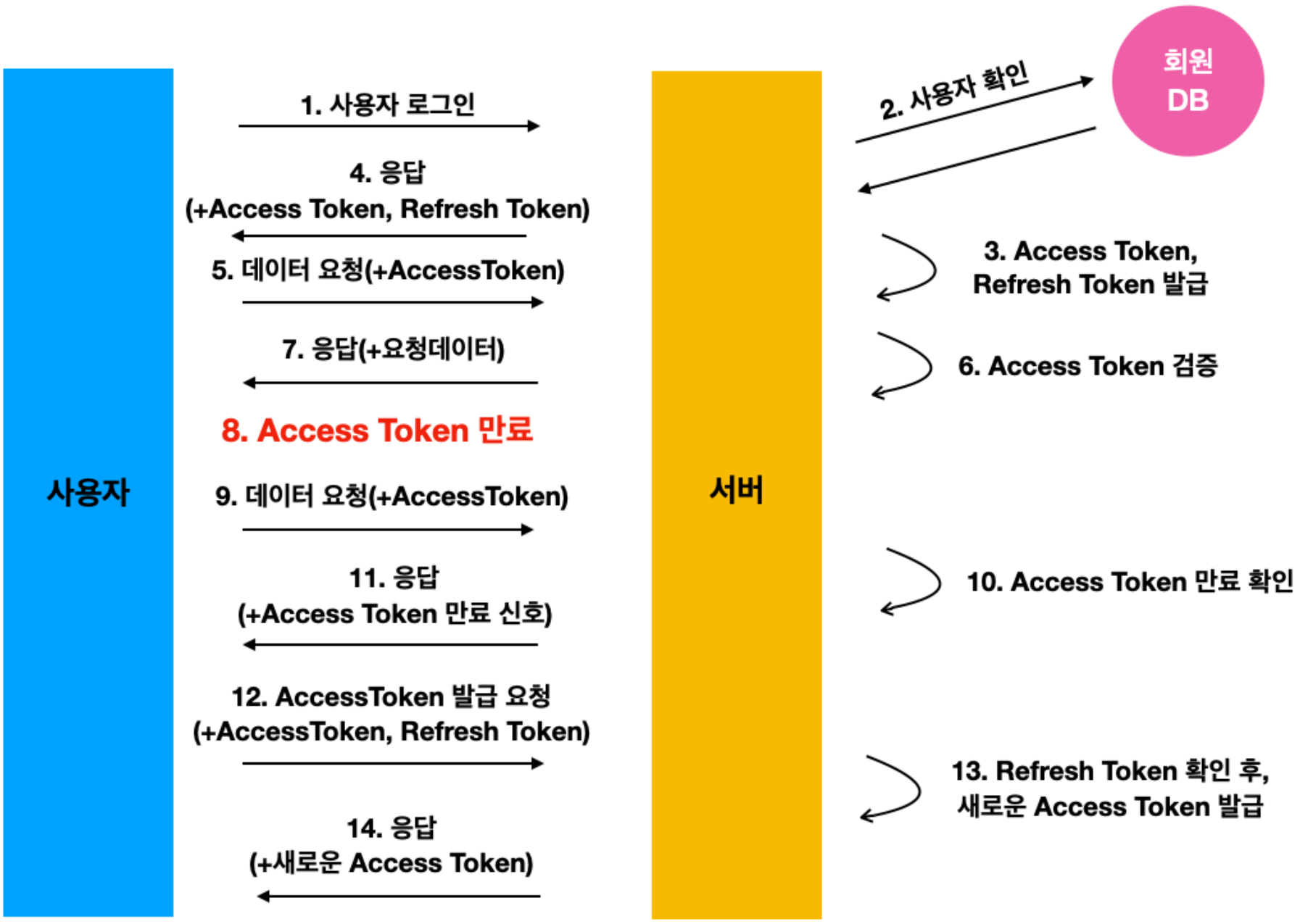
⇒ 만료기간 짧게 설정

Refresh Token

- Access Token을 탈취 당해도 상대적으로 피해를 줄일 수 있게, 기존의 Access Token의 유효기간을 짧게 하고 **Refresh Token**이라는 새로운 토큰을 발급한다.

  (유효기간 짧게 설정하면 사용자도 사용못하기 때문)

인증/인가 동작방식 (Access Token + Refresh Token)



1. 사용자가 ID, PW를 통해 로그인
2. 서버에서는 회원 DB에서 값을 비교한다. (보통 PW는 암호화해서 들어간다.)
3. 사용자 인증이 되면 서버에서 Access Token, Refresh Token을 발급, 보통 회원 DB에 Refresh Token을 저장
4. 서버는 사용자에게 Access Token, Refresh Token을 보낸다.
5. 사용자는 Refresh Token을 안전한 저장소에 저장 후, Access Token을 헤더에 실어 요청을 보낸다.
6. 서버는 Access Token을 검증 후
7. 이에 맞는 데이터를 사용자에게 보내준다.
8. 시간이 흘러 Access Token이 만료
9. 사용자는 만료된 Access Token을 헤더에 실어 요청을 보낸다.
10. 서버는 Access Token이 만료됐음을 확인
11. 만료된 토큰임을 알리고 권한없음을 신호로 보낸다.
    - Access Token이 만료될때 마다 9~11 과정을 거칠 필요는 없다.
    - Access Token의 Payload를 통해 유효기간을 알 수 있다.
    - 따라서 프론트엔드 단에서 API 요청전에 토큰이 만료 됐다면 바로 재발급 요청 가능

12. 사용자는 Refresh Token 과 Access Token을 함께 서버로 보낸다.
13. 서버는 받은 Access Token이 조작되지 않았는지 확인하고, Refresh Token과 사용자의 DB에 저장되어 있던 Refresh Token을 비교한다.

    서버는 Refresh Token이 동일하고 유효기간도 지나지 않았다면 Access Token을 사용자에게 보내준다.
14. 새로운 Access Token을 헤더에 실어 API 요청을 한다.

쿠키&세션 vs JWT

	장점	단점
Cookie & Session	Cookie만 사용하는 방식보다 보안 향상 서버쪽에서 Session 통제 가능 네트워크 부하 낮음	세션 저장소 사용으로 인한 서버 부하
JWT	인증을 위한 별도의 저장소가 필요 없음 별도의 I/O 작업 없는 빠른 인증 처리 확장성이 우수함	토큰의 길이가 늘어날수록 네트워크 부하 특정 토큰을 강제로 만료시키기 어려움