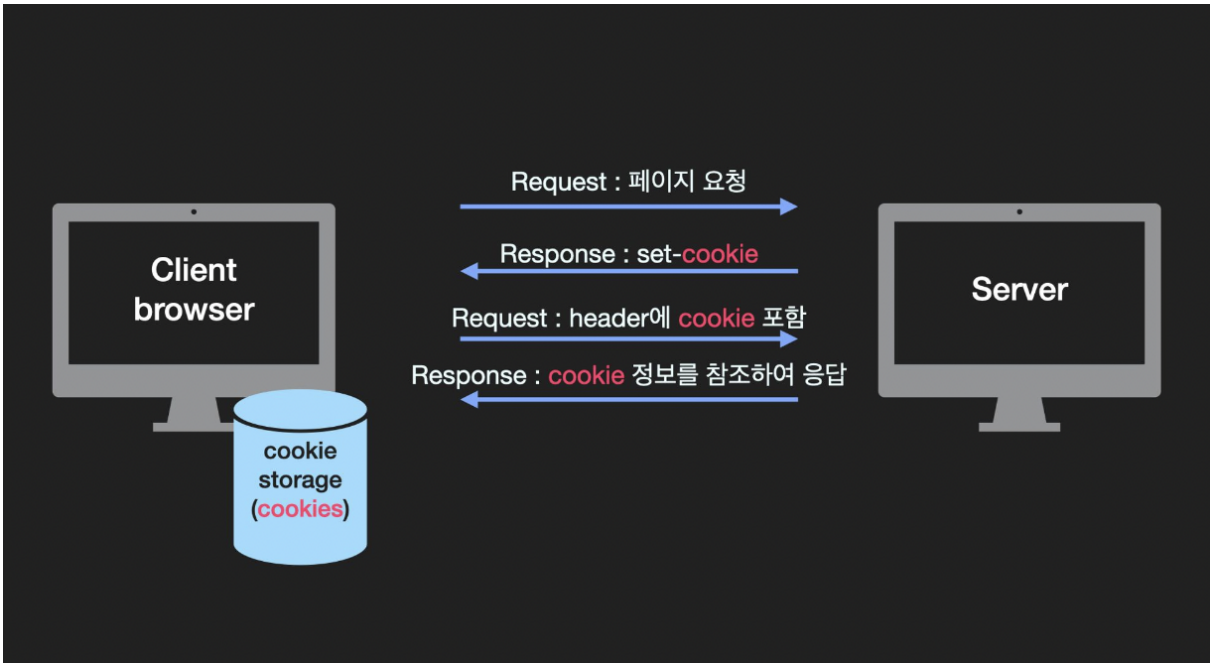
Cookie, Session, JWT

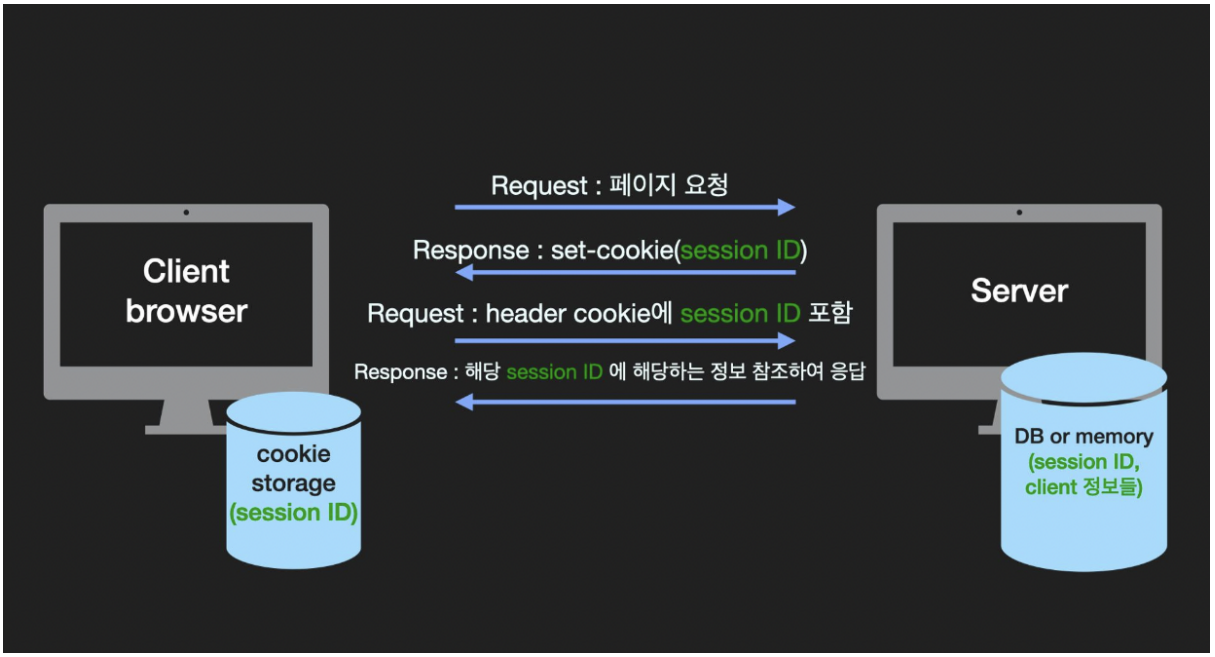
💡 쿠키(Cookie)

- 클라이언트(브라우저) 로컬에 key-value 쌍으로 저장되는 데이터(Text) 파일
- 유효시간 내에서는 브라우저가 종료 되어도 계속 유지
- 저장위치: 클라이언트의 웹 브라우저가 지정하는 메모리/하드디스크
- 서버에서 Response header에 Set-Cookie 속성을 사용해 클라이언트에 쿠키 생성 가능
- 사용자가 따로 요청하지 않아도 브라우저가 request 시에 쿠키를 request Header를 넣어서 자동으로 서버에 전송
- 사용 예)
  - 아이디, 비밀번호 저장
  - 쇼핑몰 장바구니 기능
  - 팝업창
- 쿠키의 동작 원리
  1. 서버가 클라이언트로부터 request을 받았을 때, 클라이언트에 관한 정보를 토대로 쿠키 구성
  2. 서버는 클라이언트에게 보내는 response header에 쿠키를 담아 전송
  3. 클라이언트가 response를 받으면, 브라우저는 쿠키를 쿠키 디렉터리에 저장
  4. 클라이언트가 같은 request를 보내면 HTTP header에 쿠키를 함께 전송
  5. 서버에서 쿠키를 읽어 이전 상태 정보를 변경할 필요가 있을 때 쿠키를 업데이트하여 변경된 쿠키를 HTTP header에 포함시켜 응답



💡 세션(Session)

- 서버에서 관리하는 데이터(Object 형식)으로 기본적으로 쿠키를 이용해 구현
- 클라이언트가 로그아웃하거나 설정 시간동안 반응이 없으면 무효화되기 때문에 정확한 만료 시점을 알 수 없음
- 쿠키보다 보안성 우수. But 서버 부하가 커질 수 있음 (ex. 동접자 수가 많은 웹 사이트)
- 로그인 같이 보안상 중요한 작업을 수행할 때 주로 사용
- 클라이언트를 구분하기 위해 각 클라이언트에게 Session ID를 부여하고 클라이언트는 이 Session ID를 쿠키에 저장함
- 세션의 동작 원리
  1. 클라이언트가 서버에 요청 시 Session ID 발급
  2. 클라이언트는 Session ID를 쿠키에 저장
  3. 클라이언트가 서버에 요청 시, 쿠키의 Session ID를 함께 전달해서 요청
  4. 서버는 Session IDFH 세션에 있는 클라이언트 정보를 가져와서 사용 및 요청 처리 후 클라이언트에게 응답



💡 쿠키와 세션을 사용하는 이유

- connectionless (비연결성)
  - 클라이언트가 요청을 한 후 응답을 받으면 연결을 끊어버리는 특징
  - HTTP는 클라이언트가 request를 서버에 보내면, 서버는 클라이언트에게 response를 보내고 접속을 끊음
- stateless (비상태성)
  - 통신이 끝나면 상태를 유지하지 않는 특징
  - 연결을 끊는 순간 클라이언트-서버 간 통신이 끝나면 상태 정보는 유지하지 X

➡️ **쿠키와 세션은 이러한 특징을 해결하기 위해 사용함**

- 만약 쿠키와 세션을 사용하지 않으면 페이지를 이동할 때마다 재 로그인을 해야함

💡 쿠키와 세션의 차이

- 쿠키는 서버의 자원을 사용하지 않고, 세션은 서버의 자원을 사용함
- 보안 면에서 세션이 더 우수하며, 요청 속도는 쿠키가 세션보다 더 빠름

- 쿠키는 클라이언트 로컬에 저장되기 때문에 변질되거나 request에서 스니핑 당할 우려가 있어서 보안에 취약하지만 세션은 쿠키를 이용해서 session ID만 저장하고 그것으로 구분해서 서버에서 처리하기 때문에 비교적 보안성이 우수
- 쿠키는 파일로 저장되기 때문에 브라우저를 종료해도 계속해서 정보가 남아있음. 반면 세션은 만료시간을 정할 수 있지만 브라우저가 종료되면 만료시간에 상관없이 삭제됨

❓ 세션도 좋은데 쿠키를 사용하는 이유?

- 세션은 서버의 자원을 사용하기 때문에 서버가 과부화 되고 속도가 느려질 수 있어서 보안이 중요하지 않은 작업에는 쿠키를 주로 사용함

💡 JWT(Json Web Token)

- 정보를 비밀리에 전달하거나 인증할 때 주로 사용하는 토큰으로, Json 객체를 이용함
- 일반적으로 클라이언트-서버 사이에서 통신할 때 권한을 위해 사용하는 토큰으로 정보를 안전성 있게 전달함
- 웹 상에서 정보를 Json 형태로 주고받기 위해 표준규약에 따라 생성한 암호화된 토큰
- 복잡하고 읽을 수 없는 string 형태로 저장

📌 JWT의 구성 요소

- .를 구분자로 헤더(header), 페이로드(payload), 서명(signature) 세 파트로 나누어져 있음
- 헤더(header)
  - typ: 토큰의 타입 지정 (JWT이므로 “JWT”라는 값 저장)
  - alg: 해싱 알고리즘 지정 (기본적으로 HMAC, SHA256, RSA가 사용되며 토큰을 검증할 때 사용되는 signature 부분에서 사용됨)

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

- 정보(payload)
  - 전달하려는 정보(사용자 id 등) 즉, claim(클레임)이라고 하는 것이 저장되어 있음

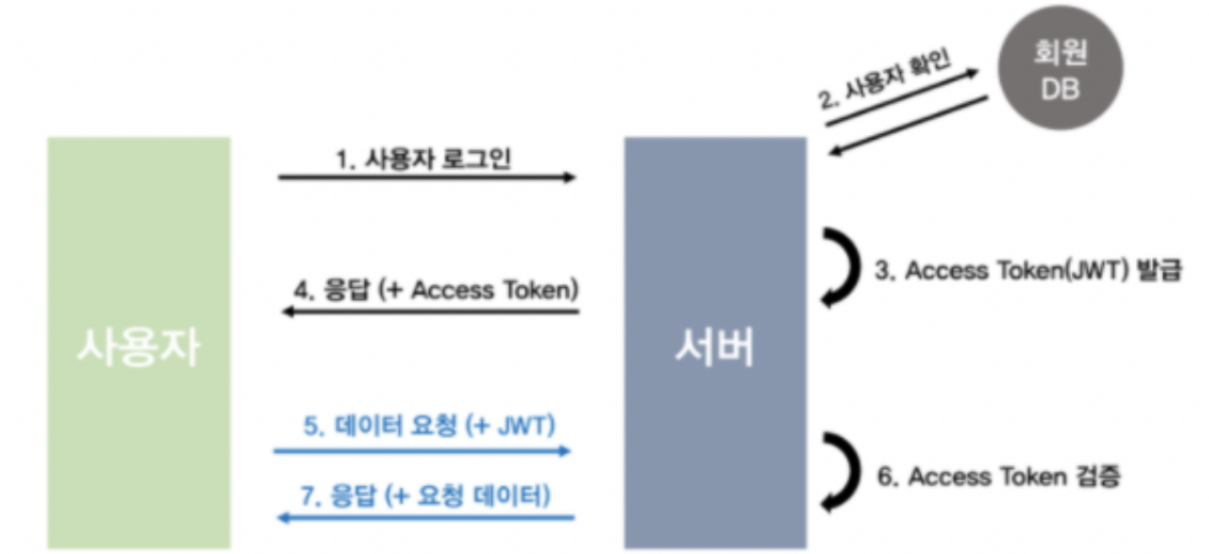
        - claim: 정보의 한 조각, name/value 한 쌍
    - 토큰에는 여러 개의 클레임들을 넣을 수 있지만 너무 많아질 경우 토큰의 길이가 길어질 수 있음
    - 또한 노출과 수정이 가능한 지점이기 때문에 인증에 필요한 최소한의 정보만을 담아야 함
    - 클레임의 종류
        1. 등록된(registered) 클레임
            - 서비스에서 필요한 정보들이 아닌, 토큰에 대한 정보들을 담기위하여 이름이 이미 정해진 클레임들. 등록된 클레임의 사용은 모두 선택적(optional)임
                - `iss` : 토큰 발급자 (issuer)
                - `sub` : 토큰 제목 (subject)
                - `aud` : 토큰 대상자 (audience)
                - `exp` : 토큰의 만료시간(expiration), 시간은 NumericDate 형식으로 되어있어야 하며 언제나 현재 시간보다 이후로 설정되어 있어야 함
                - `nbf` : Not before을 의미하며, 토큰의 활성 날짜와 비슷한 개념. 마찬가지로 NumericDate형식으로 날짜를 지정하며, 이 날짜가 지나기 전까지는 토큰이 처리되지 X
                - `iat` : 토큰이 발급된 시간(issued at), 이 값을 사용하여 토큰의 age가 얼마나 되었는지 판단
                - `jti` : JWT의 고유 식별자로서, 주로 중복적인 처리를 방지하기 위하여 사용되며, 일회용 토큰에 사용하면 유용
        2. 공개(public) 클레임
            - 공개 클레임들은 충돌이 방지된(collision-resistant)이름을 가지고 있어야 함. 충돌을 방지하기 위해서는, 클레임 이름을 URI형식으로 지음

```
{
    "https://chup.tistory.com/jwt_claims/is_admin" :
}
```

        3. 비공개(private) 클레임

        - 등록된 클레임도 아니고, 공개된 클레임들도 아닌 클레임들로, 양 측간에 (보통 클라이언트 <-> 서버) 합의하에 사용되는 클레임 이름들. 공개 클레임과는 달리 이름이 중복되어 충돌할 수 있으니 사용할 때 유의해야 함.
- 서명(signature)
    - 헤더의 인코딩값과 정보의 인코딩값을 합친 후 발급해준 서버가 지정한 비밀키로 암호화 시켜 토큰을 변조하기 어렵게 생성
    - ex) 토큰이 발급된 후 누군가가 payload의 정보를 수정하면 payload에는 다른 누군가가 조작된 정보가 들어가지만 signature에는 수정되기 전의 payload 내용을 기반으로 이미 암호화 되어있는 결과가 저장되어 있기 때문에 조작된 payload와 다른 결과값을 출력함
    - 서버는 토큰이 조작 되었는지 여부를 쉽게 알 수 있고, 조작된 토큰을 악용하기 어려워짐

💡 JWT의 동작 원리



1. 사용자가 id와 password를 입력하여 로그인 요청
2. 서버는 회원 DB에 들어가 있는 사용자인지 확인
3. 확인 후 서버는 로그인 요청 확인 후, secret key를 통해 토큰을 발급

4. 클라이언트에 전달
5. 서비스 요청과 권환을 확인하기 위해 헤더에 데이터(JWT) 요청
6. 데이터를 확인하고 JWT에서 사용자 정보 확인
7. 클라이언트 요청에 대한 응답과 요청한 데이터를 전달

📌 Refresh Token

- 만약 유효기간이 짧은 Token을 발급하게되면 사용자 입장에서 자주 로그인을 해야하기 때문에 번거롭고 반대로 유효기간이 긴 Token을 발급하게되면 제 3자에게 토큰을 탈취 당할 경우 보안에 취약하다는 약점이 있음.
- 이러한 점들을 보완하기 위해 **Refresh Token** 을 사용.
- Refresh Token은 Access Token과 똑같은 JWT로, Access Token의 유효기간이 만료되었을 때, Refresh Token이 새로 발급해주는 열쇠가 됨.
- 예) Refresh Token의 유효기간은 1주, Access Token의 유효기간은 1시간이라고 한다면, 사용자는 Access Token으로 1시간동안 API요청을 하다가 시간이 만료되면 Refresh Token을 이용하여 새롭게 발급함. 이 방법또한 Access Token이 탈취 당한다 해도 정보가 유출이 되는걸 막을 수 없지만, 더 짧은 유효기간때문에 탈취되는 가능성이 적다는 점을 이용한 것.
- Refresh Token또한 유효기간이 만료됐다면, 사용자는 재 로그인 필요.
- Refresh Token도 탈취 될 가능성이 있기 때문에 적절한 유효기간 설정이 필요.

Access Token + Refresh Token 인증 과정