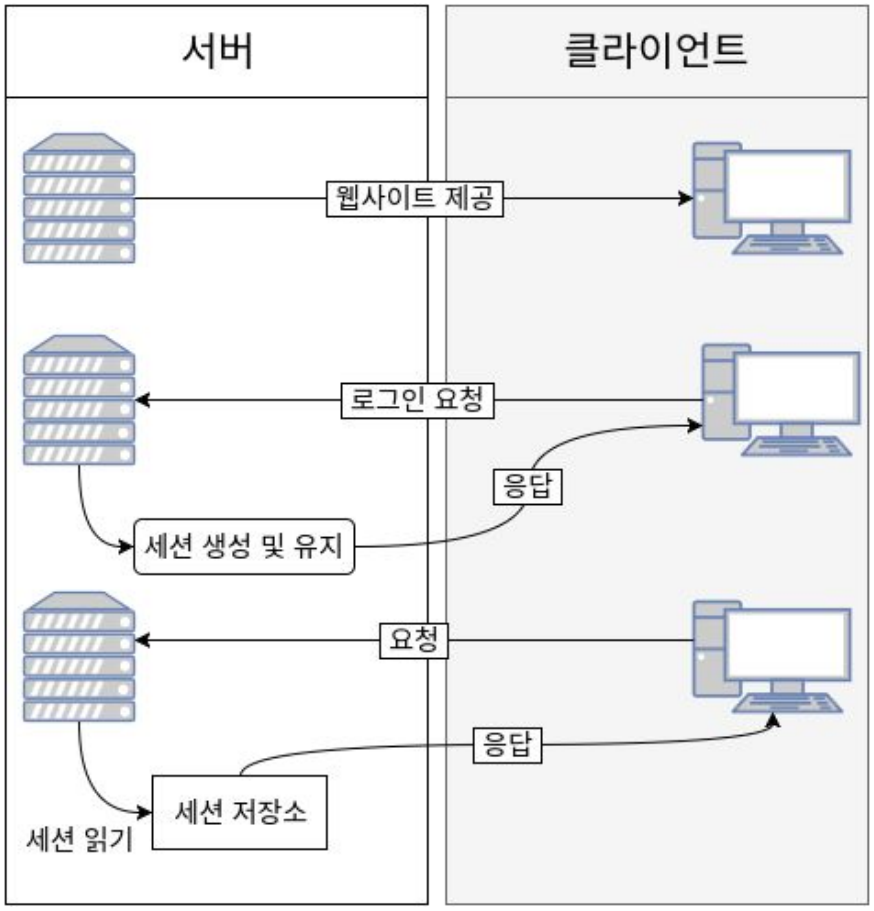
모던 웹 서비스에서는 현재 토큰 기반 인증 시스템이 가장 많이 쓰이고 있음.

이유.

1) Stateless 서버
   Stateful서버는 Client에게 요청을 받을 때마다, Client의 상태를 계속 유지하여 정보를 서비스 제공에 이용함.
   ex) Session을 유지하여, 유지된 Session을 통한 정보를 사용하는 Web Server.
   → 쉽게, 로그인정보를 저장해서 사용. 이때, 서버컴퓨터 메모리 혹은 DB
   <>
   Stateless서버는 상태를 유지하지 않음.
   서버는 Client측에서 들어오는 요청만으로만 작업을 처리함.
   ⇒ Client와 Server의 연결고리가 없기때문에 서버의 확장성(Scalability)가 증가됨.
2) OAuth와 같은 개념으로, 토큰을 넘김으로써 인증 정보를 다른 어플리케이션에 전달하는것이 가능함.
3) 보안수준을 높일 수 있음. 하지만, 무조건적으로 해킹의 위험에서 벗어나는것은 아님.

왜 토큰인가?
:: 기존의 서버기반 인증시스템에서는 메모리, 디스크, DB를 이용하여 서버측에서 유저들의 정보를 가지고 있어야 했음.



이는 아직도 많이 쓰이는 방법중의 하나지만, 서버의 확장에 문제점이 생기게 됨.

유저의 인증시에 서버는 이를 기록, 서버에 저장해야 함.

이를 세션을 이용한다고 하지만, 이 경우에는 메모리를 사용하게 되고, 결국엔 서버의 램에 과부하를 줄 수있음.
이를 피하기 위해 DB를 사용할 수도있지만 이 역시 DB의 성능에 문제를 일으킬 수 있음.

서버의 확장성이란, 더 많은 트래픽을 처리하기 위해 여러 프로세스를 돌리거나, 여러대의 서버 컴퓨터를 추가하는것을 의미함.
세션을 사용하게되면 불가능한것은 아니지만, 세션을 사용한 분산된 시스템을 설계하는것은 매우 복잡함.

CORS(Cross-Origin Resource Sharing)
세션관리시에 사용되는 쿠키는 단일 도메인 및 서브 도메인에서만 작동함.
그렇기 때문에 여러 도메인에서 관리를 할 수 있도록 하는것은 번거로움.

Token기반 시스템은 Stateless함.
이는 상태를 유지하지 않는다는것을 뜻하며, 이 시스템에서는 유저의 인증 정보를 서버 혹은 세션에 담아두지 않음.
→ 위의 Session사용에 의한 확장성등의 문제를 대부분 해겨로할 수 있게 됨.

대략적인 과정.

1) 유저의 Login
2) Login계정정보 검증
3) 정확하다면, 서버에서 유저에게 signed토큰을 발급.
4) (singed란, 정상적으로 발급된 토큰인지 증명하는 signature를 가짐을 의미)
5) Client측에서 해당 토큰을 저장하고, 서버에 요청시마다 토큰을 함께 전송.
   이때, HTTP요청의 헤더에 토큰값을 포함시켜서 전달함.
6) 요청이 들어오면 토큰을 검증하고, 요청에 응답.

무상태(Stateless)와 확장성(Scalability)
토큰기반 인증시스템의 가장 중요한 속성.
Token은 ClientSide에서 저장하기때문에 ServerSide는 완전히 Stateless.
→ 세션기반이 아니기때문에 이제 Token을 사용하게되면 어떤 서버로 요청해도 상관이 없음.
또한, 쿠키를 전달하지 않기때문에, 쿠키사용으로인한 취약점이 사라짐.

Extensibility(확장성)
Scalability와는 조금 다른 개념.
Extensibility는 로그인 정보가 사용되는 분야의 확장을 의미함.
로그인의 결과로 생성되는 토큰을 통해 다른 서비스에서도 권한의 공유가 가능해짐.
추가로, 토큰기반시스템에서는 토큰에 선택적인 권한을 부여하는것이 가능해짐.

CORS와 관련하여, Token을 사용하게되면
디바이스, 도메인 등에 상관없이 Token만 유효하다면 요청이 정상적으로 처리가 됨.
→ Access-Control-Allow-Origin: * //이를 서버측에서 헤더에 포함만 해주면 정상처리됨.

_____
JWT(JSON Web Token)
:: RFC 7519 웹표준.
두 개체사이에서 JSON객체를 사용하여 가볍고 자가수용적(self-contained)방식의, 안전성 있는 정보 전달이 가능하게 해 줍니다.
Java, C++, JavaScript등 수많은 언어를 지원하며, 자체적으로 모든 정보를 가지고있습니다.
이를 self-contained라고하며, JWT의 경우에는 토큰에서는 토큰에 대한 기본정보, 전달정보, 검증정보(signature)를 포함하게 됩니다.
JWT는 간단하게 Header에 포함시키거나 URL의 파라미터로도 전달시킬 수 있습니다.

JWT의 가장 주된 사용처로는 위에서 언급한Login을 예시로 들 수 있음.
__회원인증
1) 로그인
2) 유저정보에 기반한 토큰 발급. 유저에게 반환
3) 유저는 서버에 요청시마다 JWT를 포함하여 요청
4) 요청을 받을때마다 해당 토큰이 유효하고 인증되었는지, 유저가 권한이 있는지 검증.

⇒ 서버는 세션을 유지할 필요도, 유저의 로그인 상태를 확인 할 필요도 없으며, 그저 유저의 요청시마다 토큰을 확인만 하면 되므로 서버자원을 아낄 수 있게됨.

__정보교류
JWT는 정보가 검증되어있어, 정보를 보낸이가 바뀌진않았는지, 혹은 중간에 조작되지는 않았는지를 검증할 수 있기때문에 JWT를 이용하면 두개체 사이에서 안정성있는 정보를 교환할 수 있게됨.

___JWT
‘ . ‘을 구분자로 하여 3가지 문자열로 구성되어있음.

(보통은 JWT제작시 담당 라이브러리를 사용함.)
header, payload등의 내용은 “key” : “value”형태로 지정.

__헤더(Header)
typ과 alg를 가짐.
typ :: 토큰의 타입을 지정. JWT.
alg :: 해싱 알고리즘을 지정함. 보통 HMAC SHA256 또는 RSA. → 검증. signature에서 사용

{
“typ” : “JWT”,
“alg” : “HS256”
}
과 같은 JSON형식을 사용하여 구성하며, 이 해싱을 base64로 인코딩을 진행함.
이때, base64형식으로 인코딩하게되면 ‘=’ 이 기호가 포함 될 수 있음.
JWT의 경우에는 헤더로도 전송되지만 url로도 갈 수 있기때문에 이 해당 기호를 없애줌.
ex) replace(‘=’,’’)
디코딩에는 문제가 없음.
(누군가는 base64url 형식으로 인코딩하여 = + 등의 기호를 애초에 배제하기도…)
그 결과로 base64형식으로 인코딩된 header값을 얻을 수 있음.
위의 header.payload,signature의 header부분에 이 값이 들어가게 됨.

__정보(payload)
토큰에 담을 정보가 들어감. 이때, 정보 한조각을 클레임(claim)이라는 단위로 표현함.
claim은 name/value의 쌍으로 이루어져있으며, 토큰에는 여러개의 클레임을 담을 수 있음.
크게 3가지의 분류로 나뉘어짐.

1) 등록된(Registered)클레임
2) 공개(Public)클레임
3) 비공개(private)클레임

__1. Registered Claim(Reserved claims)
서비스에서 필요한 정보들이 아닌,
토큰에 대한 정보들을 담기위해 이름이 이미 정해진 Claim.
Registered Claim의 모든 Claim의 사용은 선택적(Optional)이며, 아래와 같음.

- iss :: 토큰 발급자(issuer)
- sub :: 토큰 제목(subject)
- aud :: 토큰 대상자(audience)
- exp :: 토큰 만료시간(expiration)
  :: NumericDate형식(ex. 1480849147370). 언제나 현재시간보다 이후로설정.
- nbf :: Not Before. 이 날짜가 지나기 전까지는 토큰이 처리되지 않음.
- iat :: 토큰 발급 시간(issued at). 이것으로 토큰의 age를 판단.
- jti :: JWT고유 식별자. 중복처리를 방지하기 위함. 일회성 토큰에 유용함.

__2. Public Claim
충돌이 방지된 이름을 가지고 있어야 함.
충돌 방지를 위해서 보통 클레임 이름을 URI 형식으로 지음.
ex) “https://velopert.com/jwt_claims/is_admin” : true
__3. Private Claim
등록되지도, 공개되지도 않은 클레임.
양 측(보통 Server-Client)간 협의하에 사용되는 클레임.
public처럼 URI형식이 아니기때문에 이름의 중복으로 인한 충돌가능성에 유의.
ex) “username” : “velopert”

__서명(signature)
인코딩된 header값과, 인코딩된 payload의 값을 합친 뒤(header+ “.” +payload)
임의의 비밀키값을 통해 해싱함.

이 해싱된 값을 똑같이 base64로 인코딩.

-- URL-safe
-- JSON의 변조를 체크
-- Server-Client간 정보교환시, HttpRequest Header에 JSON토큰을 넣어주고, Server에서는 별도의 인증과정없이 헤더에 포함되어있는 JWT정보를 통해 인증함
-- HMAC알고리즘을 사용하여, 비밀키 또는 RSA를 이용한 public/private키쌍으로 서명.
-- JWS(JSON Web Signature) :: 서명시 사용한 키를통해 JSON손상을 확인
-- JWE(JSON Web Encryption) :: JSON을 암호화하여 URL-safe문자열로 표현한 것

-- JWT Process
1) C.사용자가 Id와 password를 입력하여 로그인
2) S.서버는 요청을 확인하고 Sercret Key를 통해 AccessToken발급
3) S.Client에 JWT전달
4) C.서비스요청과 권한확인을 위해 헤더에 JWT전달
   API요청시 Client에서 Authorization header에 Access Token을 담아서 전송.
5) S.JWT서명을 체크하고 JWT에서 사용자 정보를 확인
   JWT Signature를 체크, Payload에서 사용자 정보를 확인해 데이터를 반환
6) S.Client요청에 대한 응답 전달

⇒ 로그인 정보를 서버에서 메모리에 보관하지 않는, 토큰기반 인증 메커니즘을 제공
⇒ JWT에는 필요한 모든 정보를 토큰에 포함. DB서버와의 오버헤드 최소화
⇒ CORS(Cross-Origin Resource Sharing)에는 쿠키를 사용하는게 아니므로, JWT를 채용한 인증매커니즘은 두 도메인에서 API를 제공하더라도 문제가 발생하지 않음.

JWT가 독립적이기때문에 같은 JWT로 여러 서비스에 사용할 수 있다던가, 트래픽에 대한 부담이 적다던가, URL파라미터로 사용할 수 있다던가 하는 장점이 있음.
서버측에서는 사용자의 로그인관련사항, 인증의 유무등을 세션을 통해 관리할 필요가 없음. 이는 서버자원의 비용 절감을 의미함.

대표적인 단점으로는 토큰기반이기때문에, 토큰이 만료될 때 까지는 사용자 정보를 서버에서 강제로 변경할 수 없음.
즉, DB에서 사용자 정보를 조작하더라도 토큰에 직접 적용할수는 없게됨.
필드의 증가는 토큰 자체의 크기에 지속적으로 영향을 미치게되고, 이는 데이터 트래픽 크기에 영향을 미칠 수 있게 됨.

Authorization: Bearer <token>

1) POST/login
2) Create Token with secret key
3) Return Token
4) Request With Token on header
5) Check Token Signature

6) Response

JSON 객체를 이용해 Self-contained 방식으로 정보를 안전한게 전달

Server의 확장시 Scale Out문제.
각각의 서버마다 로그인/새로고침 등의 기능에서 세션정보가 저장.
서버1에서 로그인하고 서버2를 통해 새로고침하게되면
서버는 CORS에 의해 인증이 안됐다고 판단함.

서버의 메모리에 세션을 다 저장하는것도 서버의 메모리를 너무
사용하게됨.
그렇다고 DB에 모든 정보를 넣고 사용하자니 DB서버의 부하를 야기함.

+ 웹/모바일간 쿠키-세션의 처리가 다름. Mobile First시대임.
+ not only web

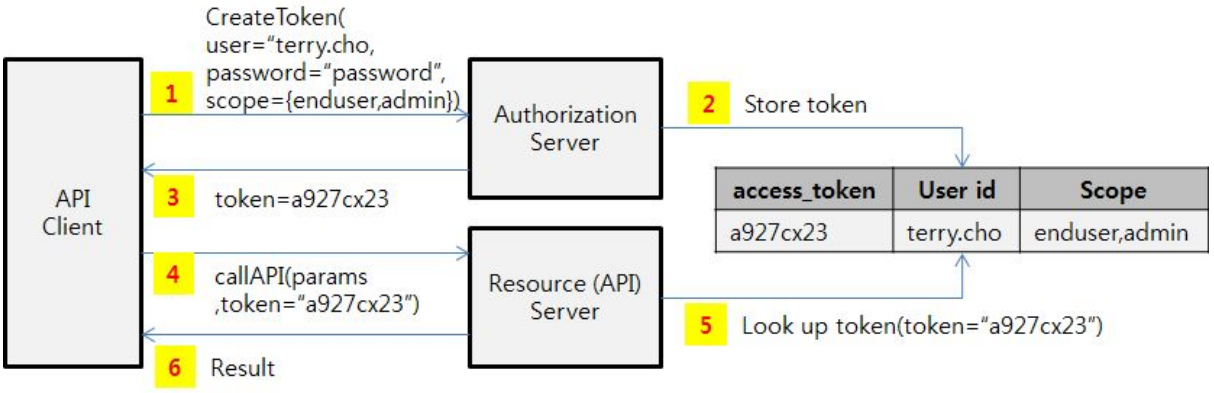
⇒ Self-contained & Stateless

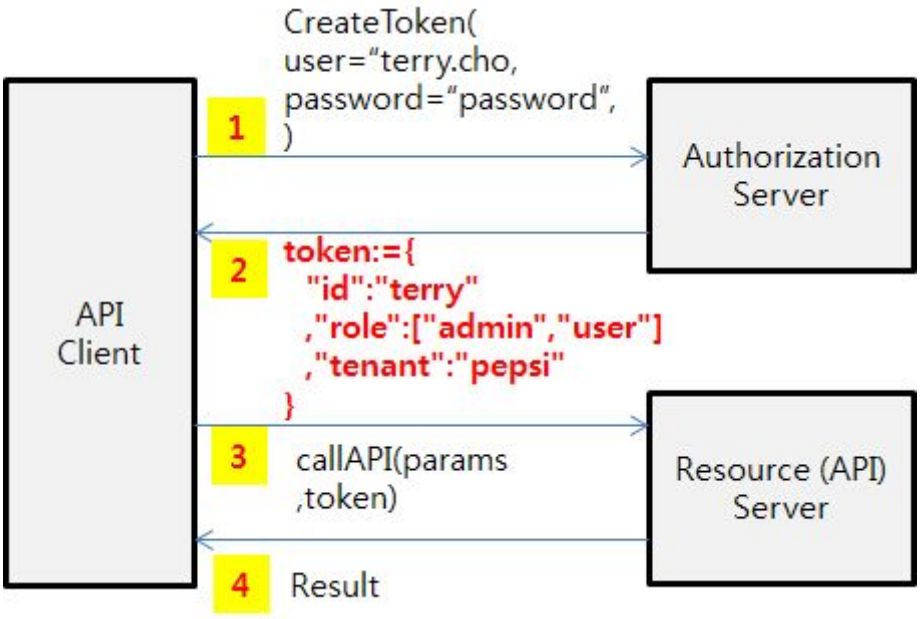
해싱 알고리즘. (HMAC SHA256 or RSA).

payload 는 암호화되지않는, base64로 인코딩된 데이터임.
JWT를 탈취당하면 디코딩을 통해 데이터를 볼 수 있게됨.
JWE를 통해 암호화하거나, payload에 중요데이터를 넣지 않아야 함.

기존의 OAuth와 같은 방식에서는 서비스를 제공해주는 입장에서 토큰을가지고, 해당 토큰에 관련된 정보를 서버쪽에서 찾아서 제공해 줬어야 함.
즉, 서버는 클라이언트에 관련된 정보를 가지고있어야하며,
동시에 요청이 들어오면 해당 정보를 다시한번 서버사이드에서 찾아야 했었음.

JWT는 Claim기반으로, 토큰 자체가 정보를 가짐.





앞서, JWT를 탈취하게되면 payload의 정보를 알 수 있고, 이를 통한 변조역시도 가능하게 됨.
변조 방지를 위해 JWE를 사용함.
→ 이를통해 메세지가 변조되지 않았음을 증명할 수 있음(무결성)
→ 서명(Signature) // HMAC방식 사용.
HMAC는 비밀키를 사용하여 암호화시키며, 해당 값을 토큰의 뒤에 붙여서 표현함.
탈취되었더라도 해당 비밀키를 모르기때문에 값의 변조시, 메세지가 변조되었음을 알 수 있게됨.

토큰방식에서는 서버에서 해당 토큰의 사용을 막을 방도가 없음.
실제 사용시에는 해당 토큰의 유일한값을 매개로하여, 해당 토큰을 blacklist에 등록하여 검증필요.
로그아웃과 같은 기능 수행시에 해당 토큰의 사용을 막을 목적