

# MSA 로그인 Session 관리는 어떻게 ?

## 일반 인증방식이 MSA에 어려운 경우

1. 서버가 각각 다른 수십 또는 수 백 수 천의 client의 세션을 동기화하고 인증하고 권한을 할당할 필요가 있다 기존의 세션관리하는 방식으로 수용이 가능한 것일까 ?
2. 기존의 session의 방식은 하나의 서버에서 통용되는 방식으로 사실 심플하고 명료하고 편리한 것은 사실이지만 여러대의 서로 다른 서버의 session 동기화의 경우는 SSO를 통하여 공유하는 방식의 한계가 있다

MSA의 세상에서는 내부 뿐만 아니라 외부의 서비스까지도 통합하는 시나리오가 존재한다.

프론트엔드 애플리케이션 또는 백엔드 서비스는 API를 통해 실시간 데이터를 요청하고 이를 사용자 인터페이스에 표시하거나 비즈니스 프로세스를 지원하기 위해 백엔드 서비스를 통해 처리할 수 있습니다. 예를 들어, 지불 처리 업체를 선택한 개발자는 해당 업체의 API를 활용하여 앱이나 웹 사이트에서 PCI(결제 카드 산업) 준수 온라인 결제 기능을 쉽게 활성화할 수 있습니다.

이런 사용의 경우 내부 시스템이 아닐 뿐만 아니라 높은 수준의 보안이 요구됩니다. 이렇듯 내부 또는 외부의 여러가지 서비스를 특별한 SSO 기술이 없이 자연스럽게 통하는 기술이 API Token 기술입니다.

이것도 한데 하루에 최소 100번이상 Basic 인증을 사용하여 API를 호출하는 한 고객을 가정해 보면 , 각각의 요청은(API 호출마다)는 개별적으로 인증되어야 하므로 인증 서버에 대한 총 호출 횟수도 하루에 최소 100번 이상이 될 수 있습니다.

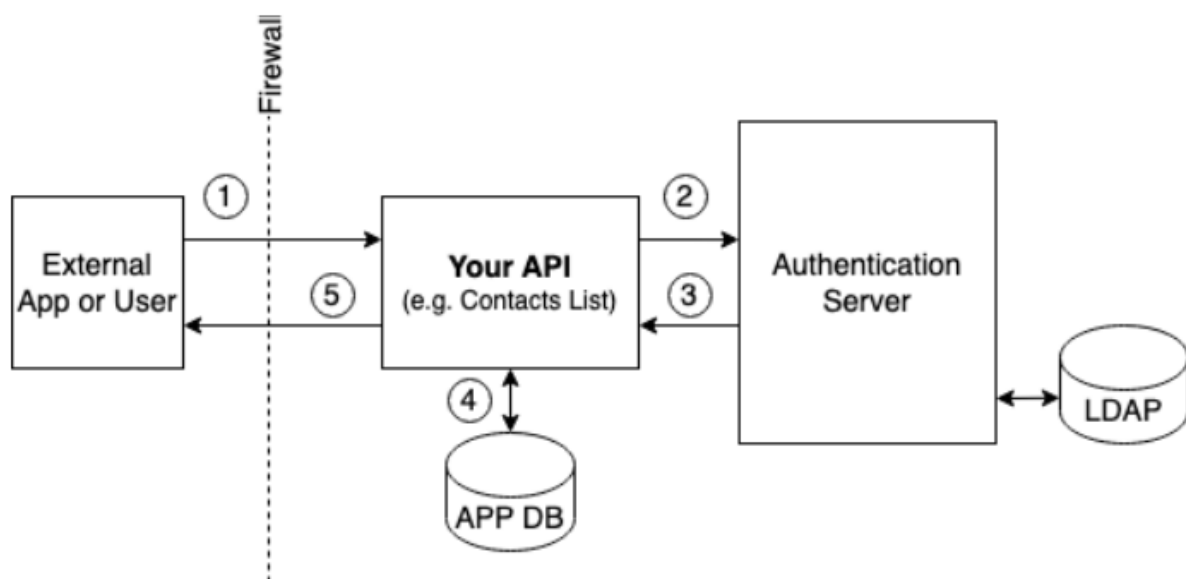


Fig. 1(A): Single API request-response with authentication

만약 우리가 고객 수를 두 배로 확장한다면 (즉, 2명의 고객), API 호출 횟수는 이제 하루에 200번이 될 것이므로, 인증 서버 호출 횟수 역시 두 배로 증가하여 하루에 200번의 호출이 될 것입니다. 따라서 API 인프라의 용량을 늘려야 하는 경우, 인증 서버의 용량도 함께 확장해야 합니다.

이것은 이전의 SSO 방식과 유사, 사실 일반 SSO 솔루션은 로컬에서 SSO Key를 인증하는 라이브러리가 있고 그것을 설치해야 하고 각각의 클라이언트 수 만큼의 함

: 두 개의 외부 노출 API가 하나의 인증 서버를 통해 Basic 인증으로 보호된 상황을 상  
상해보겠습니다. 또한 단일 고객이 각각의 API를 하루에 100번씩 호출한다고 가정해봅시다. 이러한 상황에서 인증 서버 호출은 두 API 간의 인증 합계로 나타날 것이며, 즉 하루에 200회의 인증이 이루어 집니다.

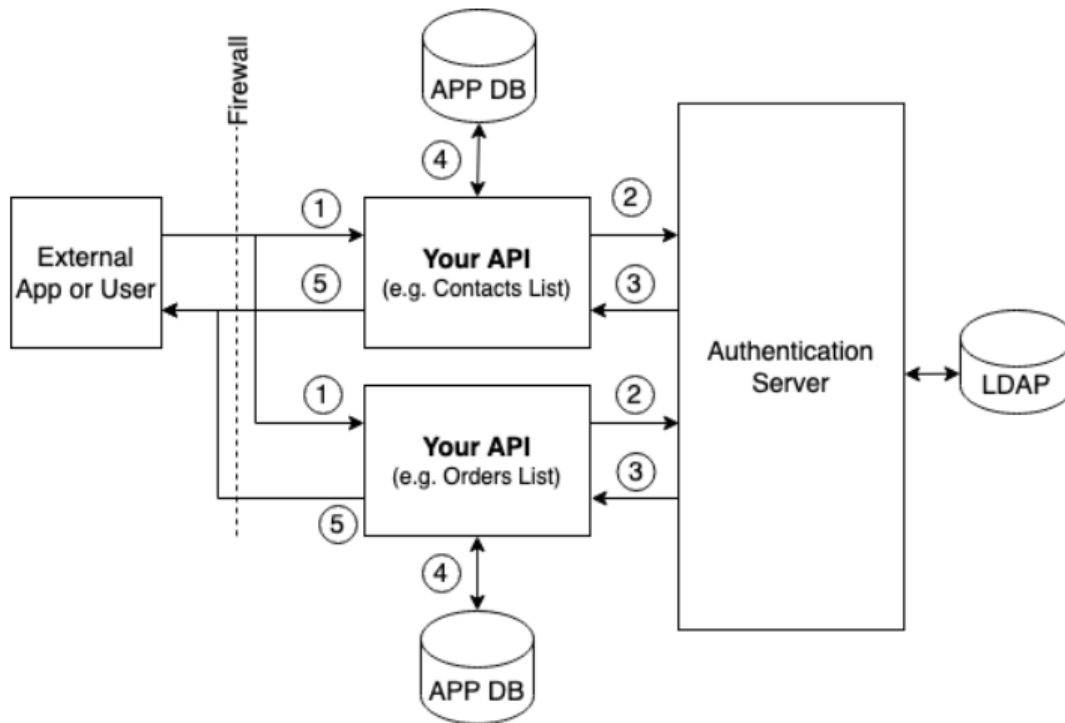


Fig. 1(B): Multiple APIs request-response with authentication

Scenario 1와 마찬가지로 고객 기반이 2배로 늘어나 2명의 고객이 각각의 API를 하루에 100번 호출하는 경우, 총 API 호출 수는 다음과 같은 순서가 될 것입니다.

2명의 고객 x (100회 호출 x 2개의 API) = 하루에 약 400회의 호출

실제로는 이렇게 되어, 인증 서버에 대한 호출도 하루에 400번이 될 것입니다.

요약하면, 각각의 API 서비스는 호출량에 따라 개별적으로 확장될 수 있지만, 총 API 호출량을 고려하여 인증 서버를 확장해야 합니다. 또한 이것은 이 될 가능성이 빠르게 나타납니다.

결국은 입니다.

## Authentication Methods and Challenges

여러 가지 인증 방법이 있습니다. 각 인증 방법은 그 장단점과 사용 사례에 대한 적합성을 가지고 있습니다. 예를 들어, 소셜 미디어 애플리케이션에서 사용자 프로필 이미지를 검색하는 API는 기본 인증 또는 API 키 기반 인증

을 사용할 수 있지만,  
를 악의적인 요청을 방어할 수 있도록 보호해야 합니다.

하며 이

## Basic Authentication은 아래와 같은 단순한 암호화입니다.

그러나 이 정보를 평문으로 네트워크를 통해 전송하는 대신, base64로 인코딩(암호화되지 않음)하여 HTTP 요청의 헤더에 첨부됩니다. 예를 들어, 연락처 목록을 검색하는 데 사용되는 다음 REST API를 기본 인증을 사용하여 호출할 수 있습니다.

사용자 이름이 'hello'이고 비밀번호가 'w0rL%'이라고 가정하면, 다음 명령을 사용하여 API를 호출할 수 있을 것입니다.

```
$ echo 'hello:w0rL%' | base64
aGVsbG86dzByTCUK

$ curl -X GET https://api-examples.com/contacts \
-H 'Authorization: Basic aGVsbG86dzByTCUK'
```

이 접근 방식에는 몇 가지 문제가 있습니다.  
합니다.

```
$ echo 'aGVsbG86dzByTCUK' | base64 --decode
hello:w0rL%
```

가 발생할 수 있습니다. 이로  
인해 API 응답 시간이 느려지거나 시간 초과 또는 인증 서버의 실패로 인한 문제가 발생할 가능성이 있습니다.

## API Key Authentication

이 인증 방법에서는 직접 사용자 이름과 비밀번호에 의존하는 대신, 개발자는 개발을 시작하기 전에 서비스 제공자로부터 고유한 해시된 문자열을 얻습니다. 이 해시된 문자열을 API 키라고 합니다. API 제공자는 개발자와 고유한 API 키를 매핑하고 확인할 수 있습니다. API 제공자에 따라 개발자는 여러 개의 API 키를 생성할 수 있을 수 있습니다. 이렇게 하면 애플리케이션의 특정 영역에 따라 전략적으로 사용하고 일정한 주기로 철회하거나 수정할 수 있는 능력을 제공합니다. 따라서 Basic 인증과 비교하여,

합니다.

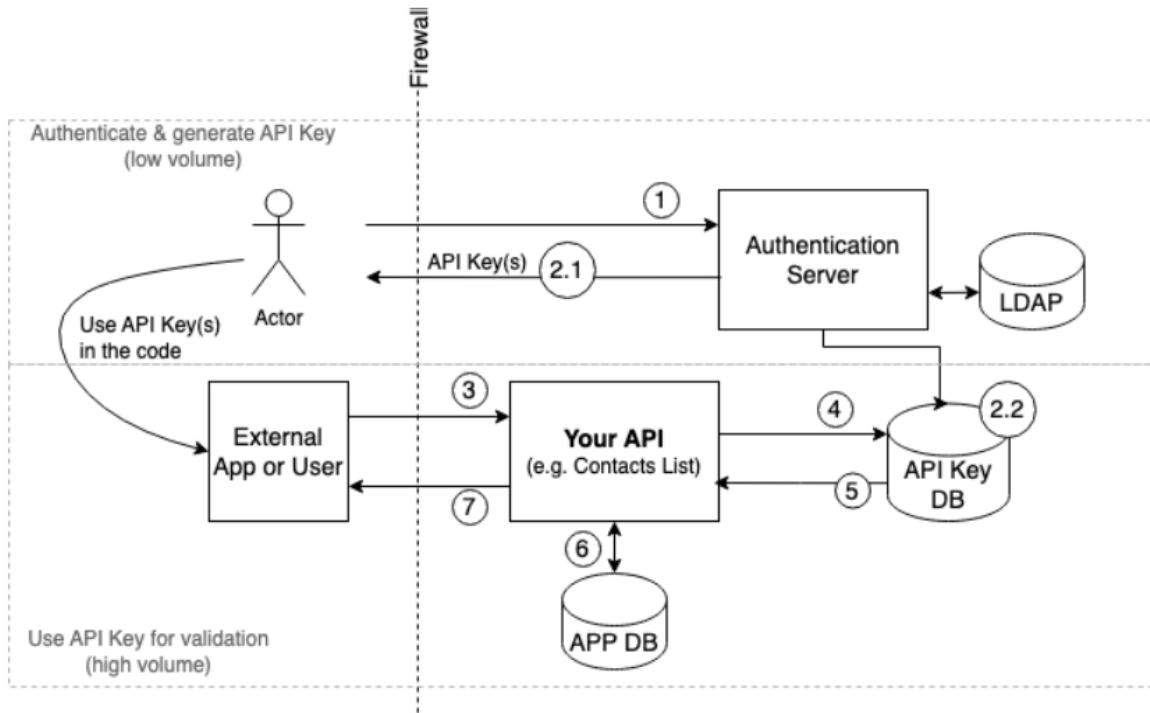


Fig. 2: API Key-based authentication flow

API 키 유효성 검사는 사용자와 API 키 간의 매핑을 저장하는 데이터베이스에 대한 것으로 진행될 수 있습니다. 각 API 호출마다 API 키와 해당 유효성을 이 데이터베이스에서 조회하여, 인증 서버에 대한 빈번한 호출을 제거하고 잠재적으로 단일 장애 지점이 되지 않도록 할 수 있습니다. API 키를 조회하기 위한 응답 시간을 개선하기 위해 키-값 또는 문서 데이터베이스를 활용할 수 있습니다.

그러나 이것은 Basic 인증보다는 개선된 방법이지만, API 키에는 여전히 단점이 있습니다.

## Token Based Authentication(구현 내용)

토큰은 이 문맥에서는 인증 서버에서 요청된 길이가 다른 무작위 문자열의 문자열입니다. 토큰 기반 인증 방법을 사용한 API 호출은 다음과 같은 세 단계로 진행됩니다. 아래에서 자세히 설명하겠습니다.

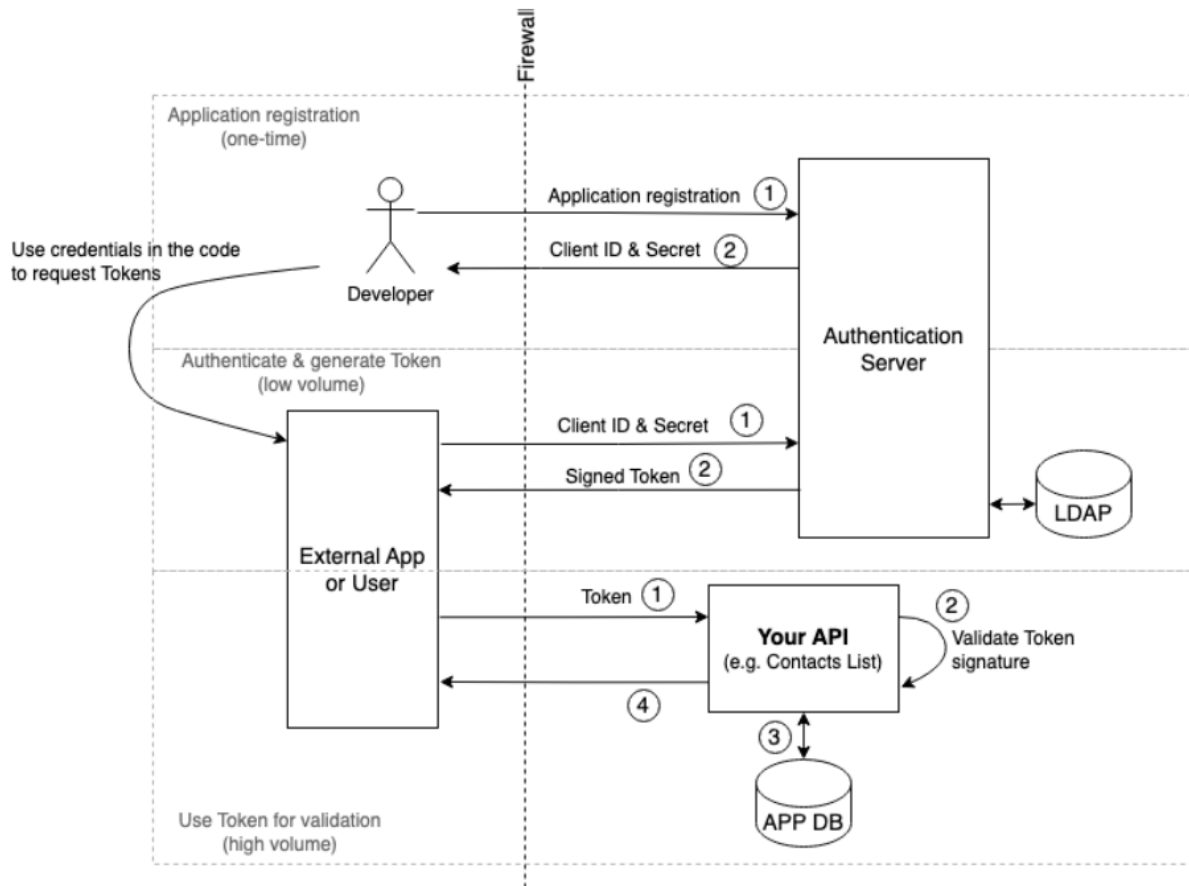


Fig. 3: Token-based authentication flow

에서 인증된 개발자는 비즈니스 API 호출을 위한 자격 증명으로 클라이언트 ID와 시크릿 형태의 자격 증명을 얻기 위해 API 제공업체와 그들의 애플리케이션을 등록해야 합니다.

등록된 클라이언트 ID와 시크릿을 사용하여 에서 개발자는 비즈니스 API 호출을 시작하기 전에 토큰을 요청합니다. 인증 서버에서는 특정 유효 기간을 가진 서명된 토큰이 발급됩니다.

에서는 토큰이 유효한 동안 두 번째 단계에서 받은 토큰을 전달하면서 적절한 비즈니스 API 엔드포인트를 호출합니다. API 엔드포인트는 토큰을 확인하고 적절한 기능적인 응답을 반환합니다.

주민등록증을 이해해 보면, 주민등록증은 동사무소에서 발급됩니다. 동사무소(인증 서버)은 원본 문서(인증)를 기반으로 여러 점의 확인을 수행한 후 특별한 검증 가능한 도장(지문)이 있는 주민등록증(토큰)을 발급합니다. 그 이후에는 주민등록증이 유효한 동안 비행기 탑승권 발급, 은행 계좌 개설 등과 같은 적절한 장소에서 주민등록증을 제시할 수 있습니다. 공항 및 은행 직원들은 실제로 우리가 누구인지(신원)를 확인할 필요가 없습니다. 즉 국가 기관에 모두 전화,팩스,인터넷 등을 통하여 모두 확인하지 않아도, 주민등록증 자체에 적시된 유효기간 내에 도장(지문)을 확인함으로써 운전면허증이 유효한지 확인하고 그렇다면 해당 서비스를 사용할 권한이 있다고 가정합니다(즉, 비행기를 탑승하거나 은행 계좌를 개설할 수 있다).

토큰 기반 인증 방법은 토큰 검증을 위한 데이터베이스 조회가 없으므로 높은 확장성을 제공합니다. 실제로 증명은 토큰의 진위성에 대한 것이며, 토큰의 서명을 검증함으로써 해결됩니다. 서명에 대해서는 다음 섹션에서 자세히 다루겠지만, 현재는 토큰이 유효로 선언되려면 인증 서버에서 생성한 서명과 호출자로부터 수신한 API 서버의 서명이 일치해야 합니다. 토큰 기반 인증 방법은 또한 만료 및 새로운 토큰 요청(토큰 로테이션)을 강제하여 설계의 추가 보안을 보장합니다.

이제 본론으로 들어와서

JWT (또한 "jot"으로 발음)는 JSON 객체 서명 및 암호화 그룹인 JOSE (JSON Object Signing and Encryption) 그룹에서 2011년부터 2016년까지 등장한 표준입니다 (Peyrott, 2018). JWT 표준은 컴팩트하고 자체 포함된 아티팩트를 사용하여 두 당사자 간에 정보(예: 보안 정보)를 공유할 수 있게 합니다. 따라서 JWT는 내장된 신뢰 메커니즘을 통해 정보를 간결하게 공유할 수 있습니다. 그럼에도 불구하고 이 표준의 가장 일반적인 구현은 API 보안에서 찾을 수 있습니다. JSON(JavaScript Object Notation)은 가벼운 데이터 교환 형식이므로 상태 없는 통신을 통해 클라이언트와 서버 간에 보안 클레임을 교환하는 데 활용되었습니다.

- **JWS** (JSON Web Signatures): 디지털 서명을 사용하여 JSON 콘텐츠에 서명하는 데 사용됩니다.
- **JWK** (JSON Web Keys): JSON 형식의 암호화 키 및 키 세트로 사용됩니다.
- **JWE** (JSON Web Encryption): JSON 콘텐츠를 암호화하거나 해독하는 데 사용됩니다.
- **JWA** (JSON Web Algorithms): JWS, JWE 및 JWK에서 사용되는 암호 알고리즘을 나타내기 위해 사용됩니다.

운전면허증도 유사한내용인데 제 개인적으로 대구 출장 시에 운전면허증을 갱신하여서 지금도 운전면허증의 발급 기관이 대구로 되어 있습니다. 하지만 그 이후 대구에 방문한 적이 없지만 여전히 대구 경찰청(발급자)에서 제 개인 운전면허증을 보증하고 있습니다.

- JWT 구조는 점(.)으로 구분된 세 부분을 포함하고 있습니다. 예를 들어, 다음과 같이 간결한 평문 JWT는 표시된 JSON 객체를 나타냅니다.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG91IiwiaWF0IjOnRydWV9.TjVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
```

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

```
{ "sub": "1234567890", "name": "John Doe", "admin": true }.
```

```
<signautre>
```

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

두 번째 부분은 payload로, 표준 및 사용자 정의 클레임을 포함하는 JSON 객체의 base64 인코딩된 문자열입니다.

- 등록된 클레임 : iss(발행자), exp(만료시간), sub(제목), aud(대상) 등이 있음 => 권장되긴 하지만 필수는 아님
  - 개인 클레임 : 서로 정보를 공유하기 위해 생성된 사용자 지정 클레임 => 원하는 정보들을 넣으면 됨
- ```
{
  // 등록된 클레임
  "iss": "chb2005.tistory.com",
  "sub": "123456789",
  "exp": "1659002265",
  // 개인 클레임
  "userName": "changbum",
  "isAdmin": false
}
```

마지막 부분은 이 토큰을 교환하는 당사자만이 알고 있는 비밀을 사용하여 서명된 헤더와 payload를 연결하여 생성된 서명입니다.

- Signature은 Header, Payload, Secret Key를 합쳐 암호화한 결과값
  - HS256( base64UrlEncode(header) + "." + base64UrlEncode(payload), Secret key)

`signing_algorithm(<header>.<payload>) = <signature>`

따라서 초기 인증 프로세스 중에 발급된 JWT에 포함된 인증 서버(Steps 2 & 3 in Fig. 3)에서 제공한 서명은 API에서 동일한 비밀(Steps 4 in Fig. 3 사용됨)을 사용하여 확인해야 합니다. 서명이 일치하면 API는 토큰의 발급자가 호출자에 의해 성공적으로 인증되었으며 따라서 응답을 신뢰할 수 있음을 확인할 수 있습니다.

## Example of API authentication using JWT

1. 사용자의 정보를 등록함
2. 로그인 : 등록된 이메일과 패스워드를 확인하고 맞으면 사용자에게 특정한 비밀키를 생성
3. 이후로 사용자는 모든 요청이 있을 때 헤더에 해당 비밀키를 전달함
  - ex) 발급받은 Jwt Token이 'xxxx.yyyy.zzzzz'라면
  - A가 B에 요청 전송시 Request Header의 'Authorization'에 'Bearer xxxx.yyyy.zzzzz'를 담아 전송
4. 요청과 토큰을 받은 인증서버는 토큰을 통해 사용자를 인증하고 요청에 대한 응답을 진행
5. 만약 악의적으로 사용자를 바꾸어서 요청허락하고자 하여도 사용자의 비밀키를 알 수 없기 때문에 정확한 Signature을 만들 수 없음

## Endpoints

동일한 방식으로, 코드는 두 가지 서비스로 구성되어 있습니다.

1. Authentication Service, with endpoints for:
  - registering the user : <http://localhost:9090/auth/signup>

Showcase1-Auth

POST 가입

POST 로그인

GET API요청

GET 이메일검증

POST 재발급

POST

http://localhost:9090/auth/signup

Params

Auth

Headers (9)

Body

Pre-req.

Tests

Settings

raw

JSON

```

1  {
2    "email": "jframework@gmail.com",
3    "password": "deloitte01!"
4  }

```

Body

200 OK 1119 ms 38

Pretty

Raw

Preview

Visualize

JSON

```

1  {
2    "email": "jframework@gmail.com"
3  }

```

member\_entity × refresh\_token\_entity

Properties

Data

엔티티 관계도

Showcase1-postgresql

Databases

show

member\_entity

Enter a SQL expression to filter results (use Ctrl+Space)

|   | member_id | authority | email                | password                                                       |
|---|-----------|-----------|----------------------|----------------------------------------------------------------|
| 1 | 2         | ROLE_USER | jframework@gmail.com | \$2a\$10\$n7BIAjrQATjilgAwfEI2.PrSXcgSMqGUws0ZXwUhaV0yW5vUANmO |

• login & generating : <http://localhost:9090/auth/login>

Showcase1-Auth / 로그인

POST

http://localhost:9090/auth/login

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```

1  {
2    "email": "jframework@gmail.com",
3    "password": "deloitte01!"
4  }

```

Body

Cookies

Headers (11)

Test Results

Status: 200 OK Time: 659 ms Size: 749 B Save as Example

Pretty

Raw

Preview

Visualize

JSON

```

1  {
2    "grantType": "Bearer",
3    "accessToken": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIyIiwiaXN0aCI6IjEwPTEVfVWVNFUjIiImV4cCI6MTY5NTk5ODExM30uZ5-tmiOarkWi66fLZite1ZBoilDuhc_Qm0R6m4l8qg5f8B23GpxxcjFjyOJmLOfiyZEPikeFEHPkylAVIYjX4A",
4    "refreshToken": "eyJhbGciOiJIUzUxMiJ9.eyJleHAiOjE2OTY2MDExMTN9.TDai1ko2zIF0f3tKNarzzxA-V0gKRiOXa5_4iIBzBCRgkW_VRugqYILIC9qqgWkbcSuOf3mfhNZ377idgcsyg",
5    "accessTokenExpiresIn": 1695998113455
6  }

```

```

{
  "grantType": "Bearer",
  "accessToken":
  "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIyIiwiaXN0aCI6IjEwPTEVfVWVNFUjIiImV4cCI6MTY5NTk5ODExM30uZ5-
  tmiOarkWi66fLZite1ZBoilDuhc_Qm0R6m4l8qg5f8B23GpxxcjFjyOJmLOfiyZEPikeFEHP
  kylAVIYjX4A",
  "refreshToken":
  "eyJhbGciOiJIUzUxMiJ9.eyJleHAiOjE2OTY2MDExMTN9.TDai1ko2zIF0f3tKNarzzxA-
  V0gKRiOXa5_4iIBzBCRgkW_VRugqYILIC9qqgWkbcSuOf3mfhNZ377idgcsyg",
  "accessTokenExpiresIn": 1695998113455
}

```



| refresh_token_entity |                                                                                                                 |
|----------------------|-----------------------------------------------------------------------------------------------------------------|
| rt_key               | rt_value                                                                                                        |
| 1                    |                                                                                                                 |
| 2                    | eyJhbGciOiJIUzUxMiJ9.eyJleHAiOiJlZDY2MDE5MTN9.TDa1koZzIF0f3tKNaRzzxZA~VOgKRIOXa5_4ilBzBCRqkW~VRugqYILIC9qqgWkbc |

## 2. API endpoint with business function

- biz call : <http://localhost:9090/api/member/me>

"accessToken":

"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIyYXV0aCI6IjPTEVfVFNfUiIsImV4cCI6MTY5NTk5ODExM30.Z5-tmiOarkWi66fLZite1ZBoi1Duhc\_Qm0R6m4l8qg5f8B23GpxxcjFjyOJmLOFiyZEPikeFEHPkylAVIYjX4A"

Swagger UI for `http://localhost:9090/api/member/me`. The request is a GET with Bearer Token authorization. The response body is:

```

{
  "email": "jframework@gmail.com"
}
```

Swagger UI for `http://localhost:9090/api/member/jframework@gmail.com`. The request is a GET with Bearer Token authorization. The response body is:

```

{
  "email": "jframework@gmail.com"
}
```

## 3. re generating

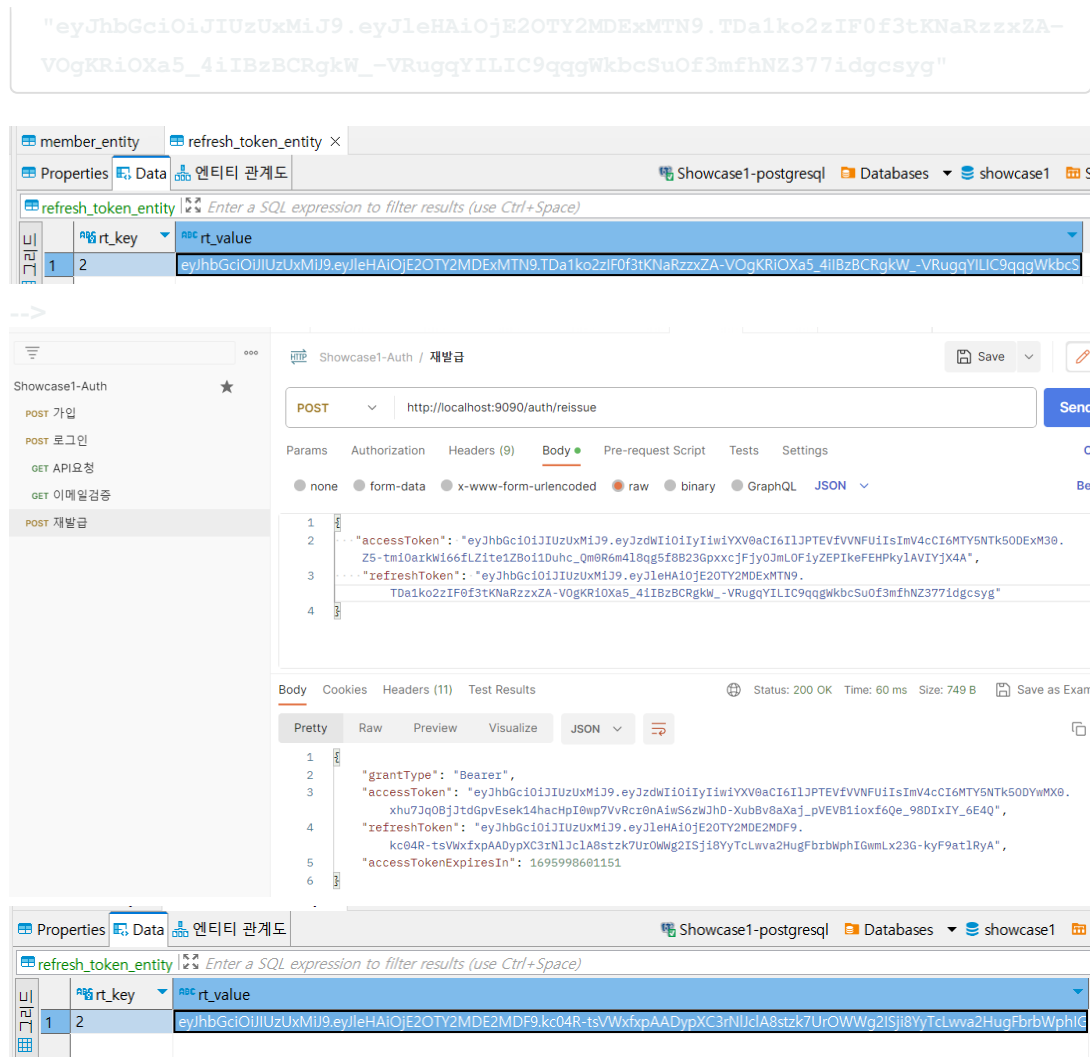
- re generating : <http://localhost:9090/auth/reissue>

만료인 경우 재발급

"accessToken":

"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIyYXV0aCI6IjPTEVfVFNfUiIsImV4cCI6MTY5NTk5ODExM30.Z5-tmiOarkWi66fLZite1ZBoi1Duhc\_Qm0R6m4l8qg5f8B23GpxxcjFjyOJmLOFiyZEPikeFEHPkylAVIYjX4A",

"refreshToken":



## JWT의 장단점

### 장점

- 서버는 비밀키만 알고 있으면 되기 때문에 세션 방식과 같이 별도의 인증 저장소가 필요하지 않음 => 서버측 부하 감소
- 여러개의 서버를 사용하는 대형 서비스 같은 경우에 접근 권한 관리가 매우 효율적임 => 확장성이 좋음
- Refresh Token까지 활용한다면 더 높은 보안성을 가질 수 있음

### 단점

- Payload의 정보(Claim)가 많아질 수록 토큰이 커짐
- 중요한 데이터는 넣을 수 없음
- 토큰 자체를 탈취당하면 대처가 어려움
- 로그아웃 시 JWT 방식은 세션이 없는 stateless 방식이기 때문에 토큰 관리가 어려움

## Conclusion

토큰 기반 인증은 API를 무단 액세스로부터 보호하는 보안 수준을 높이는 것뿐만 아니라 토큰 검증 프로세스를 인증 프로세스와 분리함으로써 API 기반 데이터 교환 아키텍처를 확장하는 데 기여했습니다.

JWT 기반 토큰은 API 인증 영역에서 널리 채택된 표준이 되었습니다. 사실, SAP BTP의 SAP Credential Store 서비스 및 Google Identity 인증 서비스를 활용한 예제에서 동일한 접근 방식이 필요했음을 알 수 있습니다.

원문 : <https://blogs.sap.com/2023/02/05/secure-your-apis-using-json-web-token-jwt/> 을 약간 수정함

## References

Palmer. (2006, November 3). Data is the New Oil. ANA Marketing Maestros: Data Is the New Oil. Retrieved January 5, 2023, from [https://ana.blogs.com/maestros/2006/11/data\\_is\\_the\\_new.html](https://ana.blogs.com/maestros/2006/11/data_is_the_new.html)

Jones, M., Bradley, J., & Sakimura, N. (2015, May). RFC 7519: JSON Web Token (JWT). RFC 7519: JSON Web Token (JWT). Retrieved January 9, 2023, from <https://www.rfc-editor.org/rfc/rfc7519>

JWT.IO. (n.d.). JSON Web Tokens – jwt.io. Retrieved January 10, 2023, from <http://jwt.io/>

Peyroth. (2018). JWT Handbook. Auth0 Inc. <https://auth0.com/resources/ebooks/jwt-handbook>

Barnes, R. (2014, April). RFC 7165: Use Cases and Requirements for JSON Object Signing and Encryption (JOSE). RFC 7165: Use Cases and Requirements for JSON Object Signing and Encryption (JOSE). Retrieved January 11, 2023, from <https://www.rfc-editor.org/rfc/rfc7165>

---

🔄Revision #10

★Created 29 September 2023 08:40:15 by Admin

✎Updated 14 December 2023 08:45:04 by Admin