10분 세미나

JWT

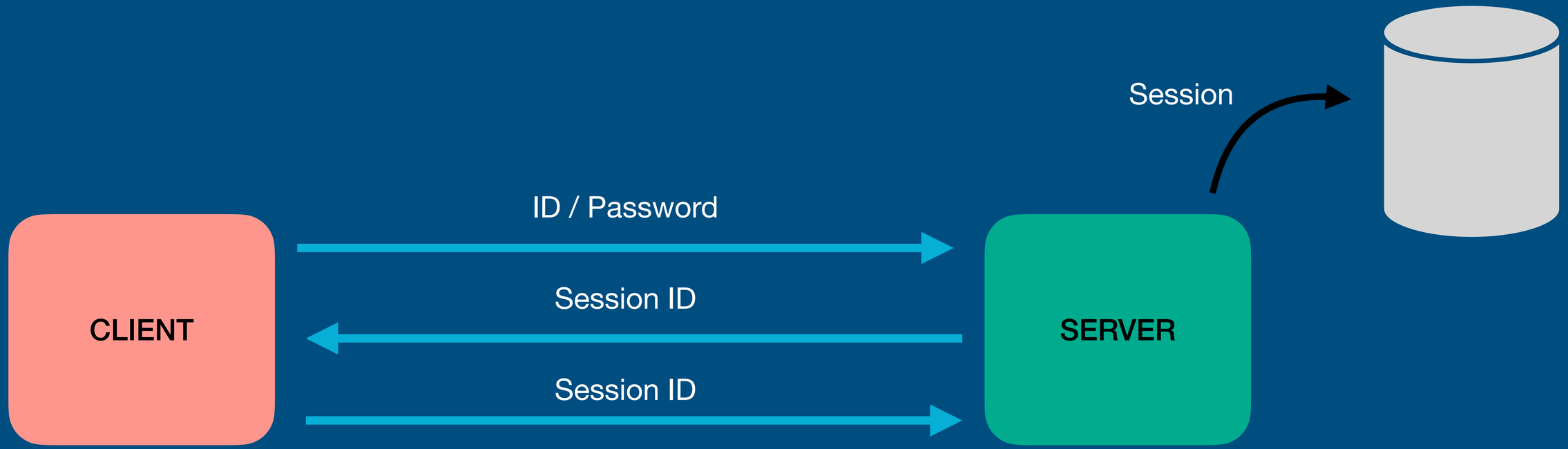
Mash-up 10th 조준형

JWT 란?

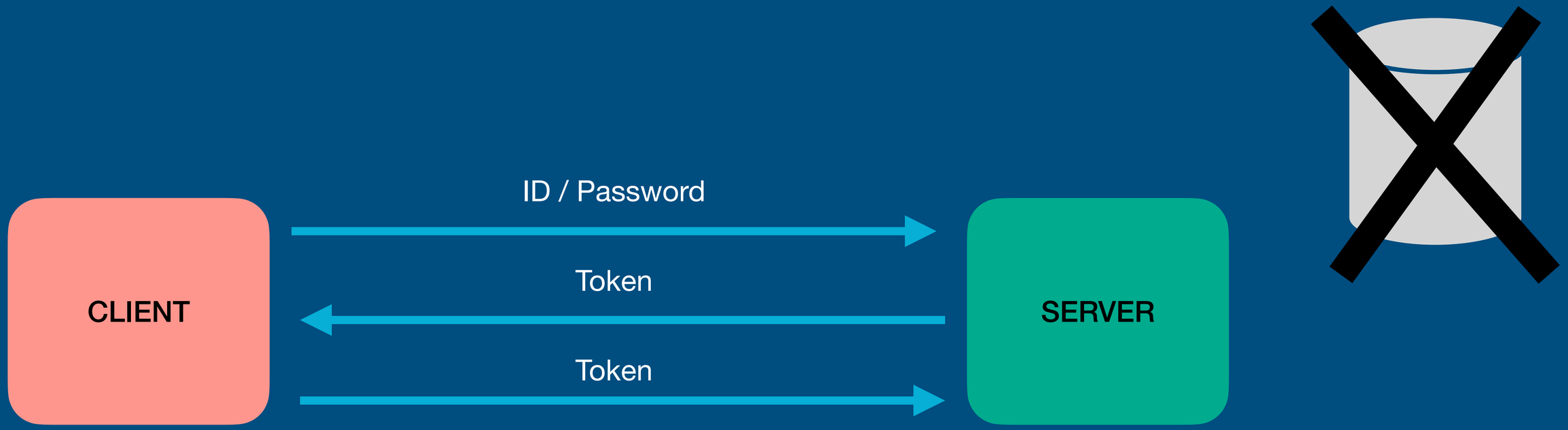
- 일반적으로 클라이언트와 서버 또는 서비스와 서비스 사이에서 권한 인가(Authorization)를 위해 사용하는 토큰

- 유저 데이터를 서버가 아닌 클라이언트가 가지고 있는 대신에, 토큰의 수정은 불가능하고 읽기만 가능

Session Based Auth



Token Based Auth



JWT의 구조

https://jwt.io/

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

JWT의 구조

```
HEADER.PAYLOAD.SIGNATURE
```

HEADER : ALGORITHM & TOKEN TYPE
PAYLOAD : DATA
VERIFY SIGNATURE

- 헤더와 페이로드는 암호화 되지 않기 때문에 민감한 정보를 넣으면 안된다.
  - Base64 URL-Safe 인코딩된 문자열
- VERIFY SIGNATURE : 시크릿 키를 이용하여 토큰을 해싱하여 암호화한 결과

Header

```
{
    "alg": "ES256",
    "kid": "Key ID",
    "typ": "JWT"
}
```

- alg는 서명 시 사용하는 알고리즘
- kid는 서명 시 사용하는 키(Public/Private Key)를 식별하는 값
- typ는 토큰의 타입을 지정함

JSON 객체를 문자열로 직렬화하고 UTF-8과 Base64 URL-Safe로 인코딩하여 다음과 같이 헤더를 생성

Base64URLSafe(UTF-8('{"alg": "ES256","kid": "Key ID"}'))

=> **eyJhbGciOiJFUzI1NiIsImtpZCI6IktleSBJRCJ9**

Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

- 권한을 확인할 수 있는 정보를 JSON으로 저장
- iat : 토근 발급 시간

헤더와 마찬가지로, UTF-8과 Base64 URL-Safe로 인코딩

Base64URLSafe('{"name": "John Doe","iat": "1516239022"}')

=> **eyJpYXQiOjE1ODYzNjQzMjcsImlzcyI6ImppbmhvLnNoaW4ifQ**

Signature

{Base64URLSafe(Sign('ES256', '${PRIVATE_KEY}',
'eyJhbGciOiJFUzI1NiIsImtpZCI6IktleSBJRCJ9.eyJpYXQiOjE1ODYzNjQzMjcsImlzcyI6ImppbmhvLnNoaW4ifQ')))

- 점(.)을 구분자로 해서 Base64 URL-Safe로 인코딩한 헤더와 페이로드를 합친 문자열을 서명한 값이다.
- 서명은 헤더의 alg에 정의된 알고리즘과 비밀 키를 이용해 성성하고 Base64 URL-Safe로 인코딩한다.
- Signature를 검증하여 토큰이 조작되었는지 체크함

MEQCIBSOVBBsCeZ_8vHulOvspJVFU3GADhyCHyzMiBFVyS3qAiB7Tm_MEXi2kLusOBpanIrcs2NVq24uuVDgH71M_flQGg

JWT

eyJhbGciOiJFUzI1NiIsImtpZCI6IktleSBJRCJ9.eyJpYXQiOjE1ODYzNjQzMjcsImlzcyI6ImppbmhvLnNoaW4ifQ.MEQCIBSOVBBsCeZ_8vHulOvspJVFU3GADhyCHyzMiBFVyS3qAiB7Tm_MEXi2kLusOBpanIrcs2NVq24uuVDgH71M_flQGg

Data Contamination

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  SECRET_KEY
) ☐ secret base64 encoded
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.4DvTG
7o-6dgSnQWPwYOmPNnM6Kdmv30_WHIk2cXVc_8
```

Data Contamination

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "Fake Name",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  SECRET_KEY
) ☐ secret base64 encoded
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.4DvTG
7o-6dgSnQWPwYOmPNnM6Kdmv30_WHIk2cXVc_8
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkZha
2UgTmFtZSIsImlhdCI6MTUxNjIzOTAyMn0.tf3W
HaAnI2JWeize1IrI6zU-_SZXH6iYg242_ANutg0
```

JWT 예제

/config/secretkey.js

```
module.exports = {
  secretKey : 'YoUrSeCrEtKeY', // 원하는 시크릿 키
  option : {
      algorithm : "HS256", // 해싱 알고리즘
      expiresIn : "30m",  // 토큰 유효 기간
      issuer : "issuer" // 발행자
  }
}
```

- JWT 토큰 암호화에 사용될 데이터
- 파일이 외부에 공개되지 않도록 해야함

JWT 예제

/jwt.js

```
const randToken = require('rand-token');
const jwt = require('jsonwebtoken');
const secretKey = require('../config/secretKey').secretKey;
const options = require('../config/secretKey').options;
const TOKEN_EXPIRED = -3;
const TOKEN_INVALID = -2;
```

- 토큰 생성 및 검증 함수

```
module.exports = {
    sign: async (user) => {
        const payload = {
            idx: user.userIdx,
            email: user.email,
        };
        const result = {
            token: jwt.sign(payload, secretKey, options),
            refreshToken: randToken.uid(256)
        };
        return result;
    },
    verify: async (token) => {
        let decoded;
        try {
            decoded = jwt.verify(token, secretKey);
        } catch (err) {
            if (err.message === 'jwt expired') {
                console.log('expired token');
                return TOKEN_EXPIRED;
            } else if (err.message === 'invalid token') {
                console.log('invalid token');
                return TOKEN_INVALID;
            } else {
                console.log("invalid token");
                return TOKEN_INVALID;
            }
        }
        return decoded;
    }
}
```

JWT 예제

/controller/user.js

```
const jwt = require('../modules/jwt');

const signin = async (req, res) => {
  const user = await User.getUserByEmail(email);
  const jwtToken = await jwt.sign(user);
  return res.status(statusCode.OK).send(
    util.success(statusCode.OK, responseMsg.LOGIN_SUCCESS, {
      token: jwtToken.token,
    })
  );
};
```

- 로그인 시 토큰을 생성

JWT 예제

/middleware/auth.js

```
const jwt = require('../modules/jwt');
const MSG = require('../modules/responseMessage');
const CODE = require('../modules/statusCode');
const util = require('../modules/util');
const TOKEN_EXPIRED = -3;
const TOKEN_INVALID = -2;

const authUtil = {
  checkToken: async (req, res, next) => {
    var token = req.headers.token;
    // 토큰 없음
    if (!token) return res.json(util.fail(CODE.BAD_REQUEST, MSG.EMPTY_TOKEN));
    // decode
    const user = await jwt.verify(token);
    // 유효기간 만료
    if (user === TOKEN_EXPIRED)
      return res.json(util.fail(CODE.UNAUTHORIZED, MSG.EXPIRED_TOKEN));
    // 유효하지 않는 토큰
    if (user === TOKEN_INVALID)
      return res.json(util.fail(CODE.UNAUTHORIZED, MSG.INVALID_TOKEN));
    if (user.idx === undefined)
      return res.json(util.fail(CODE.UNAUTHORIZED, MSG.INVALID_TOKEN));
    req.idx = user.idx;
    next();
  },
};
```

- 인증을 위한 미들웨어

JWT의 장점

- 세션을 따로 저장하기 위한 저장소가 필요없다.
- 시크릿키만 있으면 유저를 식별할 수 있다.
- MSA 에서 쉽게 사용가능하다.

JWT의 단점

- 시크릿키가 외부에 유출되면 위험하다.
- 토큰이 유출되면 다른사람이 사용할 수 있다.
- 중복 로그인에 대응하기 어렵다.

JWT 보완

- 대칭키 사용
  - Server에서 대칭키를 사용
    - Client가 토큰 내용을 읽는 것도 못하게 할 수 있음
  - Client에서 대칭키를 사용
    - 토큰이 유출되어도 제 3자는 토큰의 내용을 보지못함

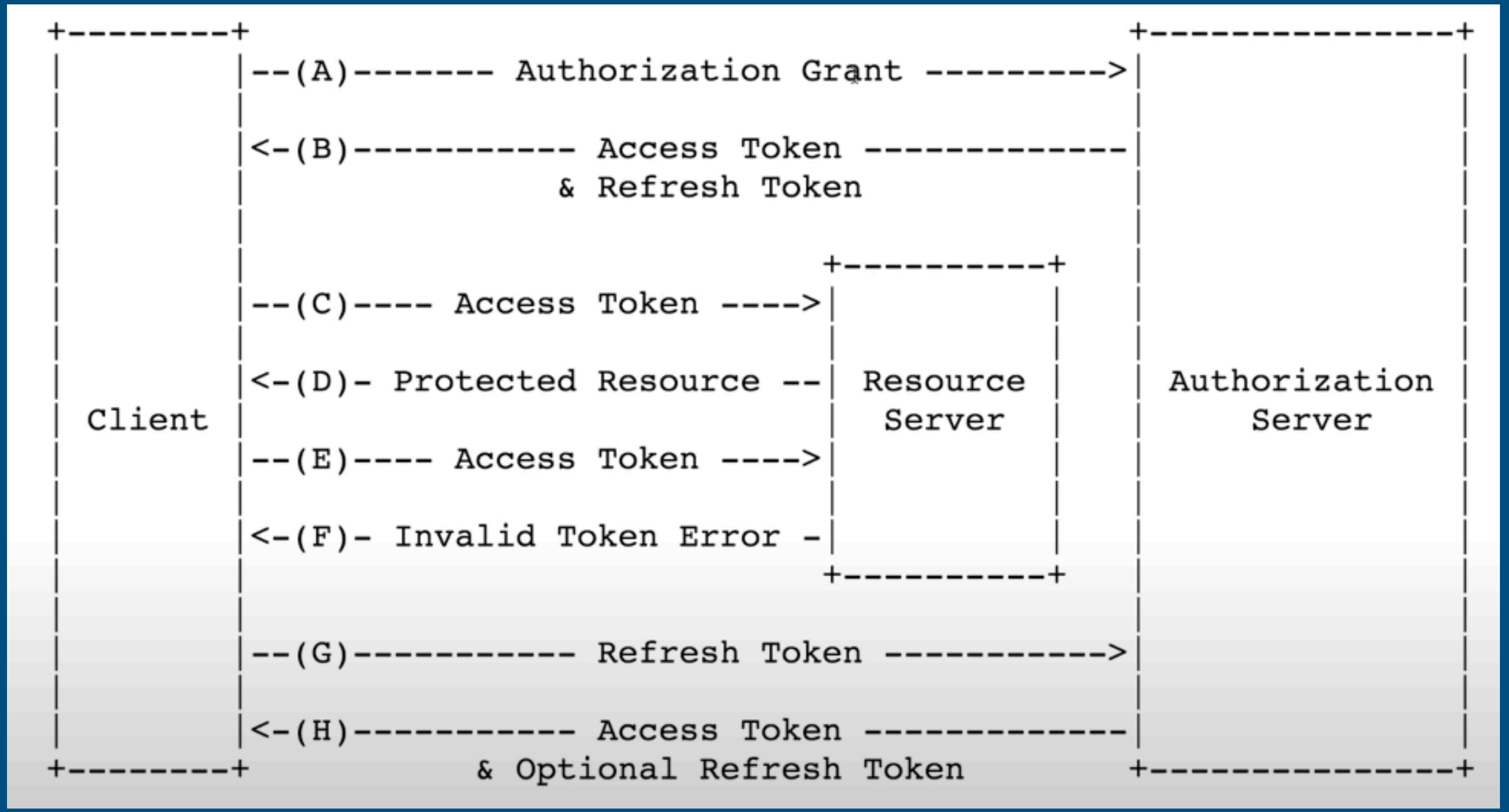
- 만료 기한 설정
  - 토큰에 만료기한을 넣어서 기한이 지난 토큰은 무시하게 만든다.

JWT 보완

- Sliding Session
  - 특정 요청 시 마다 토큰의 만료 기한을 연장시켜 토큰을 새로 발급

- Refresh Token
  - 비교적 긴 만료시간을 가지는 Refresh Token을 함께 발급하여 클라이언트에게 전달함
  - Refresh Token은 서버에서 별도의 Storage에 보관되기 때문에 JWT의 장점이 사라짐

Refresh Token



감사합니다