[개발환경] JOSE로 JSON Web Token (JWT) 처리하기

구현 어려움 정도	⭐⭐
구현 완료	☑
기간	@December 13, 2024
정리 완료	☑

JWT

JSON Web Token의 약자로 JSON 형식을 사용해서 정보를 안전하게 전송할 수 있도록 설계된 웹 표준입니다.

JWT는 주로 인증(Authentication)과 권한부여(Authorization) 시스템에서 사용됩니다.

JWT는 크게 세 부분으로 나눠져 있습니다.

- 헤더(Header)
- 페이로드(Payload)
- 서명(Signature)

이 세 부분은 각 `.`(점)으로 구분되어 있습니다. 아래 코드와 같은 형태입니다.

```
header.payload.signature
```

1. 헤더(Header)

헤더는 JWT 알고리즘(HS256, RS256)과 타입(JWT)을 포함합니다. 일반적으로 헤더는 다음과 같은 구조를 가집니다.

```
{
  "alg": "HS256", // alg: 사용되는 알고리즘.
  "typ": "JWT" // typ: 타입
}
```

2. 페이로드(Payload)

페이로드는 실제로 전달하려는 데이터입니다.

사용자의 정보(이메일, 이름, 권한 등)를 포함할 수 있습니다.

페이로드는 클레임(Claims)이라고도 불립니다.

클레임은 2가지 유형이 있습니다.

- 등록된 클레임(Registered Claims): 미리 정의된 필드들
  - 예: `iat`(발행 시간), `exp`(만료 시간), `sub`(주체), `iss`(발행자)
- 공개 클레임(Public Claims): 사용자가 자유롭게 정의할 수 있는 클레임들
- 비공개 클레임(Private Claims): 서버 간에만 사용되는 데이터로 서로 합의돈 규칙에 따라 정의할 수 있음

사용자 id와 이메일을 담은 페이로드는 아래와 같습니다.

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "email": "john.doe@example.com"
}
```

3. 서명(Signature)

서명은 JWT의 무결성을 보장하고 변조를 방지하기 위해 사용됩니다. 서명을 생성하는 방법은 다음과 같습니다.

a. 먼저 헤더와 페이로드를 Base64 url로 인코딩합니다.

b. HMAC SHA-256과 같은 알고리즘을 사용해 `서명키(비밀키 또는 공개키)` 와 함께 헤더, 페이로드를 서명합니다.

서명한 JWT는 아래 코드처럼 생성됩니다.

```
header.payload.signature
```

서명은 HS256 알고리즘을 사용했다면 아래 코드처럼 생성됩니다.

```
HMACSHA256(
    base64UrlEncode(header)
    + "." +
    base64UrlEncode(payload),secret
)
```

서명은 JWT의 무결성을 보장하는 중요한 역할을 합니다. JWT가 생성된 후 서버는 이 서명을 통해 JWT의 데이터가 변조되지 않았는지 확인할 수 있습니다.

Jose 라이브러리로 JWT 처리하기

```
import { SignJWT, jwtVerify } from "jose";
```

jose 라이브러리란?

jose는 JSON Object Signing and Encryption의 약자로 JWT를 생성하고 검증할 수 있는 라이브러리입니다. JWT는 주로 웹 애플리케이션에서 인증과 권한부여를 위해 사용됩니다. jose는 이 JWT를 생성하고 서명하고 검증하는데 필요한 다양한 기능을 제공합니다.

jose에서 제공하는 주요 기능

1. SignJWT

- JWT 생성 및 서명
- 사용자 인증을 위해 JWT를 생성하고 이를 서명하여 안전하게 전송할 수 있게 함

2. jwtVerify

- JWT 검증
- JWT의 유효성을 검증해 잘못된 JWT가 사용되지 않도록 방지

3. 대칭키 및 비대칭키 암호화

- HS256(HMAC SHA-256) 알고리즘이나 RS256 등의 다양한 암호화 알고리즘을 지원

사용 예

1. JWT 생성 (SignJWT)

```
import { SignJWT } from 'jose';

// 페이로드 예시
```

```
const payload = { id: 'eunjee', name: '은지' };

// 비밀 키
const secretKey = new TextEncoder().encode(process.env.SESSION_SECRET); //  보안과 관리를 위해 환경변수를 사용해서

// JWT 서명
const jwt = await new SignJWT(payload)
  .setProtectedHeader({ alg: 'HS256' }) // 알고리즘 설정
  .setIssuedAt() // 발행 시간 설정
  .setExpirationTime('1d') // 만료 시간 설정
  .sign(secretKey); // 서명
console.log(jwt); // 생성된 JWT 출력
```

2. JWT 검증 (jwtVerify)

```
import { jwtVerify } from 'jose';

// 비밀 키
const secretKey = new TextEncoder().encode(process.env.SESSION_SECRET);

// 유효성 검사를 할 JWT
const jwt = 'your-jwt-here';

// JWT 검증
const { payload } = await jwtVerify(jwt, secretKey, {
  algorithms: ['HS256'], // 사용할 알고리즘 설정
});
console.log(payload); // JWT의 페이로드 출력
```

비밀키를 환경변수로 사용해야하는 이유

보안 유지

- 비밀키는 서명된 토큰을 생성할 때 사용되며 비밀키가 외부에 노출되면 보안 위험이 커집니다.
- 비밀키가 유출되면 JWT 토큰을 위조하거나 변조할 수 있기 때문에, 사용자의 토큰을 생성하거나 세션탈취를 할 수 있습니다.
- 이를 위해 환경변수로 설정해 소스 코드에서 직접 접근할 수 없게 해야합니다.

환경별 설정

- 환경별 설정(개발환경, 테스트환경, 운영환경..)도 가능합니다.
- 각각 환경에서 비밀키가 달라야하며 이를 환경 변수로 관리함으로써 각 환경에 맞는 비밀키를 설정할 수 있습니다.

```
# .env 파일 예시 (개발 환경 - local)
SESSION_SECRET=dev-secret-key

# .env.production 파일 예시 (운영 환경 - production)
SESSION_SECRET=prod-secret-key
```

.gitignore에 env 추가

환경변수로 비밀키를 관리하면 버전관리 시스템에서 비밀키가 유출되는 것을 방지할 수 있습니다. .gitignore 파일에 .env를 추가해줘야 합니다!

```
// .gitignore

# env files (can opt-in for committing if needed)
.env*
```

비밀키를 환경변수로 설정하여 보안을 강화하고 환경에 맞게 설정을 관리하며 소스 코드의 민감 정보 유출을 방지해야 합니다.