2022-10-26 2반 실습

학습 포인트

- JWT
  1. JWT가 무엇인지 알아본다.
  2. JWT를 왜 사용하는지 알아본다.
  3. JWT를 어떻게 사용하는지 알아본다.
- Passport 미들웨어
  1. Passport 미들웨어의 기능을 이해하고 사용하는 방법을 알아본다.

JWT(JSON Web Token) 개요

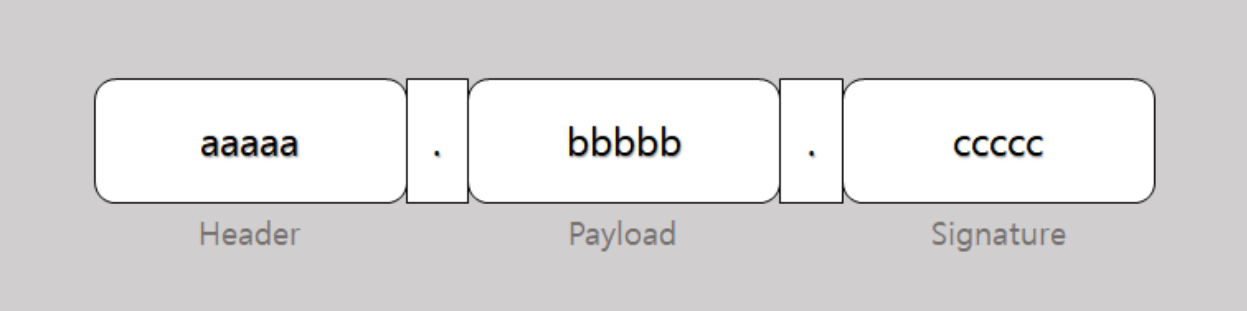
- `JWT`는 유저를 인증(`authentication`)하고 식별(`identification`)하기 위한 `Token`기반 인증 방식
- RFC 7519 참고
- `Token`은 `Session`과 달리 서버가 아닌 클라이언트에 저장되기 때문에 메모리나 스토리지 등을 통해 세션을 관리했던 서버의 부담 감소
- `JWT`는 Token 자체에 사용자의 권한 정보나 서비스를 사용하기 위한 정보가 포함(Self-contained)
- 데이터가 많아지면 토큰도 커지며, 토큰이 한 번 발급된 이후 사용자의 정보가 변경되어도 토큰을 재발급 하지 않는 이상 반영되지 않음
- JWT를 사용하면 RESTful과 같은 Stateless인 환경에서 사용자 데이터 통신 가능
- `Session`사용 시 쿠키등을 통해 식별하고 서버에 세션을 저장
- `Token`사용 시 클라이언트에 저장하고 요청시 HTTP 헤더에 토큰을 첨부하는 것만으로 통신 가능
- JWT 사용되는 과정
  1. client가 id, password를 통해 웹서비스 인증
  2. server에 서명된(signed) JWT를 생성하여 client에 응답으로 반환

3. client가 server에 데이터를 추가적으로 요구할 때 JWT를 HTTP Header에 첨부
4. server에서 client로부터 수신한 JWT 검증

- JWT는 `JSON` 데이터를 Base64 URL-safe Encode를 통해 인코딩하여 직렬화한 것이 포함
- Token 내부에는 위변조 방지를 위해 개인키를 통한 `전자서명` 존재
- Base64 URL-safe Encode : 일반적인 Base64 Encode에서 URL에서 오류없이 사용하도록 '+', '/'를 각각 '-', '_'로 표현한 방식

JWT(JSON Web Token) 구조

- `JWT`는 `Header`, `Payload`, `Signature`로 구성
- 각 요소는 `.`으로 구분



- `Header`에는 JWT에서 사용할 `Type`, `Hash 알고리즘 종류`

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

- `Payload`에는 서버에서 첨부한 `사용자 권한 정보와 데이터`

```
{
  "email": "sample@domain.com",
  "profile": "http://domain.com/image.png",
  "http://domain.com/xxx/yyy/is_admin": true
}
```

- `Signature`에는 Header, Payload를 Base64 URL-safe Encode를 한 후 Header에서 명시한 Hash함수를 적용하고, 개인키(Private Key)로 서명한 `전자서명`이 존재

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

JWT(JSON Web Token) 구현

- jwt 모듈 설치

```
npm install jsonwebtoken
```

- jwt 모듈 불러오기

```
const jwt = require('jsonwebtoken');
```

- 토큰생성(`sign`)

```
const token = jwt.sign(payload, secretKey, option)
```

- decode payload(`verify`)

```
const decoded_token = jwt.verify(token, secretKey);
```

Passport

- express 프레임워크 상에서 사용되는 인증 미들웨어 라이브러리
- 기본적인 인증시스템 지원과 더불어 페이스북, 구글과 같은 소셜 로그인을 할 수 있도록 도와주기 때문에 매우 편리
- Passport 공식 문서 참고
- `passport.js`에서는 다양한 인증방법을 `strategie` 라는 이름으로 제공

- 기본적인 사용자 ID 및 PASSWORD를 사용하는 방법부터, OAuth를 사용한 위임 인증 등 다양한 strategies가 존재
- Local Strategy(`passport-local`) : 로컬 DB에서 로그인 인증 방식
- Social Authentication (`passport-google-oauth`, `passport-facebook`, `passport-naver`, `passport-kakao`, `passport-twitter` 등) : 소셜 네트워크 로그인 인증 방식
- passport 모듈(local strategy) 설치

```
npm install passport passport-local bcrypt
```

- passport 처리 과정
  1. 로그인 요청이 라우터(`router`)로 들어옴.
  2. 미들웨어를 거치고, passport.`authenticate()` 호출
  3. authenticate에서 passport/`locaStrategy.js` 호출
  4. done()정보를 토대로, 로그인 성공 시 사용자 정보 객체와 함께 req.login()를 자동으로 호출
  5. req.login 메서드가 passport.`serializeUser()` 호출
  6. req.session에 사용자 아이디 키값만 저장 (메모리 최적화를 위해서)
  7. passport.`deserializeUser()`로 바로 넘어가서 sql조회후 req.user 객체를 등록후, `done()` 반환하여 `req.login` 미들웨어로 다시 되돌아감
  8. 미들웨어 처리후, res.redirect('/')을 응답하면, 세션쿠키를 브라우저에 보내게 된다
  9. 로그인 완료 처리 (이제 세션쿠키를 통해서 통신하며 로그인됨 상태를 알 수 있다.)
- 로그인 이후
  1. 모든 요청에 passport.`session()` 미들웨어가 passport.`deserializeUser()` 메서드를 매번 호출
  2. `deserializeUser`에서 req.session에 저장된 아이디로 데이터베이스에서 사용자 조회
  3. 조회된 사용자 전체 정보를 req.`user` 객체에 저장
  4. 이제부터 라우터에서 req.user를 공용적으로 사용 가능하게 된다.

Passport 주요 함수

- passport.initialize() : passport를 사용한다고 express에게 알림
- passport.session() : session을 사용하여 passport를 동작시킨다고 express에게 알림
- `passport.authenticate( )` : strategy를 호출하고 실행
- `passport.serializeUser()`
  - 로그인에 성공했을 때 작동. 로그인 성공 시 실행되는 `done()`에서 user 객체를 전달받아 세션(정확히는 req.session.passport.user)에 저장
  - 로그인에 성공했을 때 해당 전달받은 user정보를 기반으로 작업을 수행한다. `done`
    함수의 2번째 인자로 user의 식별자를 주고, session 데이터에 해당 정보가 저장된다.
- `passport.deserializeUser()`
  - serializeUser가 처리된 상태에서 사용자가 여러 페이지에 방문할 때마다 실행
  - 실제 서버로 들어오는 요청마다 세션 정보(serializeUser에서 저장됨)를 실제 DB의 데이터와 비교
  - `done`의 두 번째 인자로 user정보 전체를 전달하고, session에 저장된 user식별자를 기반으로 그에 맞는 user를 찾아 활용
  - 그렇게 식별된 user가 `req.user`가 된다. 해당 req의 객체는 passport를 사용했기 때문에 생성되는 객체(해당 객체의 존재여부판단으로 로그인여부 확인 가능)