PAU CASESNOVES
Centre Integrat de Formació Professional

Desenvolupament d'aplicacions web

Desenvolupament web a l'entorn client

UT 7.4: Apis segures

Joan Pons Tugores

Índex de continguts

Apis amb control d'accés

Fins ara em treballat amb APIs sense cap tipus de seguretat. Això vol dir que qualsevol pot utilitzar aquestes APIs sense cap control.

Evidentment no serà el més habitual. Normalment una aplicació web tendrà una part per al públic general, però certes parts de l'aplicació estaran restringides a un usuari autoritzat. Depenent de la complexitat de l'aplicació es podran definir rols, on cada rol permetrà certes accions i s'establiran relacions n-m entre els rols i els usuaris.

Autenticar i autoritzar

Són dos conceptes que sovint es confonen:

- **Autenticar** significa provar la identitat. En el nostre context vol dir comprovar que un usuari és realment qui diu ser, normalment validant el parell nom d'usuari - contrasenya contra qualque magatzem de dades.
- **Autoritzar** en canvi significa donar permís per realitzar alguna cosa, per noltros normalment significarà permetre que l'usuari accedeixi a algun recurs de la nostra aplicació web.

Configuració

Quan el nostre projecte implementa una API REST normalment no s'utilitzen les sessions per guardar l'estat de l'aplicació, sinó que és el client qui el manté.

En aquests casos, una vegada tenim l'usuari autenticat, el servidor ens envia un token que inclou informació de la sessió, normalment el nom d'usuari però pot incloure altres dades.

En cada petició que faci el client s'ha d'incloure aquest token. Si el token que rep el servidor és vàlid, la signatura digital confirma que no s'ha modificat i no ha expirat, el

servidor utilitza el nom d'usuari que conté el token per autoritzar o no l'accés al recurs que demana el client.

JWT

Json Web Token és un estàndard obert que defineix una manera compacta i autocontinguda de transmetre informació entre client i servidor com un objecte JSON signat digitalment. La signatura es pot fer amb una paraula secreta (algorisme HMAC) o amb un parell de claus RSA o ECDSA

Com que estan signats digitalment, poden garantir la integritat del contingut del token.

Quan s'utilitza com a mètode d'autorització, el primer que hem de fer és autenticar l'usuari. Després es genera un token que l'usuari ha d'enviar a totes les peticions que faci al servidor.

Estructura d'un token

Un token JWT està format per tres parts:

- **Capçalera**: Indica el tipus de token (jwt) i l'algorisme utilitza per signar-lo, per exemple HMAC HS256 o RSA. Es codifica en BASE64.

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

- **Payload**: Conté els *claims*: informació sobre una entitat, normalment l'usuari, a més d'informació addicional, cap d'ells és obligatori. N'hi ha de tres tipus:
  - Registrats: Registrats a la IANA, de tres lletres:
    - iss: de *issuer* identifica l'emissor. Pt ser una cadena de texta o una URI
    - sub: de *subject*. Identifica el *receptor*.

- aud: de *audience*. El domini o dominis als que va dirigit el token.
- exp: de *expiration date*. La data a partir de la qual el token no serà acceptat. Segons a partir del 01/01/1970
- nbf: de *not before*. La data abans de la qual el token no serà acceptat. Segons a partir del 01/01/1970
- iat: de *issued at*. La data d'emissió del token. Es pot utilitzar per calcular la seva edat. Segons a partir del 01/01/1970
- jti: de *JWT ID*. L'identificador únic del token.

◦ Públics: Definits per el productor. S'ha d'assegurar que no hi haurà col·lisions amb altres clames, per exemple controlant l'espai de noms.

◦ Privats: Acordats entre el productor i el consumidor del token. Hem d'anar alerta a no provocar col·lisions.

Es codifica en BASE64

```
{
  "sub": "9876543210",
 "name": "Jo Mateix",
  "iat": 15162567895,
  "nbf": 15162568000,
  "exp": 15162570000
}
```

- **Signatura**: Per calcular la signatura s'utilitzen els valors codificats de la capçalera i el payload, l'algorisme i el secret o les claus, i es signa. La signatura serveix per comprovar que el token no ha canviat i si està signat amb una clau privada, per comprovar que l'emissor és qui diu ser.

```
HMACSHA256(
  base64UrlEncode(capçalera) + "." +
  base64UrlEncode(payload),
  paucasesnoves
)
```

El token es forma unint els valors en BASE64 de la capçalera i el payload i la signatura amb '.' El valor del token que hem anant posat d'exemple seria:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI5ODc2NTQzMjEwIiwi
bmFtZSI6IkpvIE1hdGVpeCIsImlhdCI6MTUxNjI1Njc4OTUsIm5iZiI6MTUxNjI
1NjgwMDAsImV4cCI6MTUxNjI1NzAwMDB9.ONU09HgVXgabRuO6Ug0y_
WKY2xr254uyUEP3sVJ0fGk
```

A la url https://jwt.io/#debugger-io teniu un formulari per poder especificar les dades de la capçalera i del payload i l'algorisme de signatura.

Exemple pràctic

Ja hem comentat que per accedir a la part restringida d'una aplicació web necessitarem efectuar dues passes: autentificació, demostrar que som qui deim que som, i autorització, demanar permís per accedir als recursos restringits.

Autenticació

L'api haurà de tenir un endpoint, obert evidentment, que permeti que l'usuari s'identifiqui.

El típic formulari d'usuari i password. Ja depèn de cada cas, però el més normal és enviar un post amb les dades del formulari a aquest endpoint. Al següent exemple enviam les dades en format JSON al servidor.

```
document.getElementById("formulariLogin").onsubmit = (e) => {
    e.preventDefault();
    const dades = {
        login: document.getElementById('username').value,
        password: document.getElementById('password').value
    }
    fetch(`${rutaAplicacio}authenticate`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Accept': 'application/json'
        },
        body: JSON.stringify(dades)
    })
        .then(resposta => {
            if (!resposta.ok) {
                if (resposta.status === 401) {
```

```
                throw new Error("Usuari o contrasenya incorrectes.");
            } else {
                throw new Error(resposta.status + ": " + resposta.statusText);
            }
        }
        return resposta.json();
    })
    .then(dades => {
        token = dades.accessToken;
    })
    .catch(error => {
        alert(error);
    });
};
```

En aquest cas poden passar dues coses:

- Ens arriba una resposta amb el codi *401 Unauthorized*. L'usuari no s'ha aconseguit autenticar, o el nom d'usuari o la contrasenya no són correctes.
- Ens arriba una resposta amb el codi 200. L'usuari s'ha identificat satisfactòriament. En aquest cas al cos de la resposta ens arriba el token.

```
{
"accessToken":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImlzQW
RtaW4iOnRydWUsImV4cCI6MTcwNTc3MTQ5NH0.oRSvLuSjGx_KLIO04f0
ah0wwiqgRQiYWRC4TmGoLzX6--
Nakm4siUQamMn2_9FtilYI7D0x7bpu4AFd9pJWNbA"
}
```

Haurem de guardar aquest token, per exemple al *session storage*, perquè l'haurem d'enviar amb totes les peticions que facem a l'àrea restringida de l'aplicació.

Autorització

Si el servidor ens ha autenticat correctament estam en possessió d'un token. Cada vegada que necessitem fer una petició a l'àrea restringida de l'API haurem d'enviar aquest token dins la capçalera *Authorization* de la petició:

```
fetch(url, {
        method: 'GET',
        headers: {
            accept: 'application/json'
            , Authorization: 'Bearer ' + token
        }
    })
```

Normalment s'inclou la paraula *Bearer* seguida d'un espai abans del token.

Com a resultat podem rebre el codi *403 Forbidden.* Això vol dir que:

- No hem enviat el token.
- El token ha expirat
- L'usuari intenta accedir a una part de l'aplicació a la que no té accés.

Si el token és correcte i l'usuari té accés al recurs que demana llavors el resultat serà exactament el mateix que si l'API estàs oberta.