Autenticació basada en sessions


Com ja vam comentar anteriorment, la seguretat basada en sessions és, potser, el mecanisme més utilitzat per a definir autenticació en aplicacions web tradicionals, basades en navegador.

Cal tindre en compte, abans de res, que el protocol HTTP que s'empra en les comunicacions client-servidor és un protocol *sense estat*, és a dir, no es guarda cap informació, ni existeix cap relació, entre dues peticions consecutives al mateix servidor. Això dificulta, en principi, el fet que el servidor puga "recordar" qui ha accedit a la web, per a deixar-li continuar fent-ho. Els mecanismes de seguretat basats en sessions afigen un element anomenat *sessió* a la comunicació client-servidor, que permet emmagatzemar informació sobre el client que accedeix, de manera que el servidor emmagatzema aqueixa informació, i quan el client torna a accedir li recorda, i li dona accés.

Veurem en aquest document com configurar les sessions en Express i definir mecanismes d'autenticació i validació d'usuaris.

1. Fonaments de l'autenticació basada en sessions

L'autenticació basada en sessions permet autenticar usuaris en aplicacions web basades en navegadors, i "recordar" l'usuari que es va validar en les seues successives visites. Per a això, utilitzen les **sessions**, que comprenen un conjunt d'interaccions d'un client amb un servidor en un determinat període. Quan obrim un navegador i accedim a una web, automàticament s'inicia la sessió en aquesta web, i mentre no tanquem el navegador o la sessió manualment, l'aplicació recorda (o pot recordar, si vol) que ja hem accedit, i els passos que hem anat donant en l'actual sessió.

Quan intentem accedir a una zona restringida d'unes certes webs, com per exemple la nostra pàgina personal d'una entitat bancària, o els comentaris en un fòrum, l'aplicació ens demana que ens validem. Quan introduïm un login i password, aquesta els acara amb els que tinga emmagatzemats i, si són correctes, emmagatzema en la sessió dades sobre el nostre usuari, com per exemple, i sobretot, nostre *nick* o *login*, i el perfil d'usuari que tenim en la web (és a dir, el rol: si som administradors, editors, visitants, etc). Així, per a cada nova petició que fem en aqueixa mateixa sessió, el servidor comprova en la sessió qui som i quin rol tenim, i en funció d'això, ens permet fer unes coses o altres. En finalitzar, podem tancar la sessió (*logout*), i esborrar les dades que s'hagen guardat en ella de la nostra visita.

2. Definició de sessions en Express

Per a poder treballar amb sessions en Express instal·larem el mòdul *express-session*. És un *middleware* que permet, en cada petició que requerisca una comprovació, determinar si l'usuari ja s'ha validat i amb quines credencials, abans de deixar-li accedir al que cerca o no.

Així que el primer que farem serà instal·lar el mòdul:

```
npm install express-session
```

Després, ho incorporem al nostre servidor Express juntament amb la resta de mòduls:

```
const express = require('express');
const session = require('express-session');
...
```

A continuació, configurem la sessió dins de l'aplicació Express:

```
let app = express();
...
app.use(session({
    secret: '1234',
    resave: true,
    saveUninitialized: false
}));
```

Els paràmetres de configuració que hem emprat són:

- `secret`: una clau de xifratge per a la sessió, que s'emprarà per a enviar-la xifrada entre client i servidor. És una cosa similar a la paraula secreta per a xifrar un token, en l'autenticació basada en tokens.
- `resave`: s'empra per a refrescar la sessió amb cada nou accés, de manera que mentre continuem accedint a l'aplicació dins del temps de caducitat establit per a la sessió, aquest es renova automàticament
- `saveUninitialized`: serveix per a guardar sessions encara que no s'hagen completat. S'utilitza si volem emmagatzemar en sessió dades d'usuaris que no s'hagen validat, per exemple. En el nostre cas no habilitarem aquesta opció.

Existeixen altres paràmetres i opcions de configuració, tal com podem consultar en la web del repositori NPM.

> **NOTA**: la configuració de la sessió haurà de fer-se ABANS de definir els encaminadors, ja que en cas contrari aquest *middleware* s'aplicarà després de processar les rutes, i no tindrà efecte.

2.1. Validació

En tot procés d'autenticació ha d'haver-hi una validació prèvia, on l'usuari envie les seues credencials i s'acaren amb les existents en la base de dades, abans de deixar-li accedir.

Suposarem, per simplicitat, que tenim els usuaris carregats en un array, amb el seu nom d'usuari i el seu password:

```
const usuaris = [
    { usuari: 'nacho', password: '12345' },
    { usuari: 'pepe', password: 'pepe111' }
];
```

Ara hauríem de definir una ruta que, normalment per POST, recollira les credencials que envia l'usuari i les acarara amb aqueix array. Si concorda amb algun usuari emmagatzemat, es guarda en la sessió el nom de l'usuari que va accedir al sistema, i es pot redirigir a alguna pàgina d'inici. En cas contrari, es pot redirigir a una pàgina de login:

```
app.post('/login', (req, res) => {
    let login = req.body.login;
    let password = req.body.password;

    let existeixUsuari = usuaris.filter(usuari =>
    usuari.usuari == login && usuari.password == password);

    if (existeixUsuari.length > 0)
    {
        req.session.usuari = existeUsuario[0].usuari;
        res.render('index');
    } else {
        res.render('login',
            {error: "Usuari o contrasenya incorrectes"});
    }
});
```

2.2. Autenticació

Una vegada validat l'usuari, hem de definir una funció *middleware* que s'encarregarà d'aplicar-se en cada ruta que vulguem protegir. El que farà serà comprovar si hi ha algun usuari en sessió. En cas afirmatiu, deixarà passar la petició. En cas contrari, enviarà a la pàgina de validació o *login*, per exemple.

```
let autenticacio = (req, res, next) => {
    if (req.session && req.session.usuari)
        return next();
    else
        res.render('login');
};
```

Només ens queda aplicar aquest *middleware* en cada ruta que requerisca validació per part de l'usuari. Això es fa en la mateixa anomenada a *get, post, put* o *delete*:

```
app.get('/protegit', autenticacio, (req, res) => {
    res.render('protegit');
});
```

Notar que passem com a segon paràmetre el *middleware* d'autenticació. Si passa aqueix filtre, s'executarà el codi del `get`. En cas contrari, el *middleware* està configurat per a renderitzar la vista de login.

3. Definint rols

La nostra aplicació també pot tindre diferents rols per als usuaris registrats. Per exemple, podem tindre administradors i usuaris normals. Això se sol definir amb un camp extra en la informació dels usuaris:

```
const usuaris = [
    { usuari: 'nacho', password: '12345', rol: 'admin' },
    { usuari: 'pepe', password: 'pepe111', rol: 'normal' }
];
```

Quan un usuari valide les seues credencials, a més d'emmagatzemar el seu nom d'usuari en sessió, també podem (devem) emmagatzemar el seu rol. Així que la ruta que valida l'usuari es veu modificada per a afegir aquesta nova dada en sessió:

```
app.post('/login', (req, res) => {
    let login = req.body.login;
    let password = req.body.password;

    ...

    if (existeixUsuari.length > 0)
    {
        req.session.usuari = existeUsuario[0].usuari;
        req.session.rol = existeUsuario[0].rol;
        res.render('index');
    } else {
        ...
    }
});
```

Per a poder comprovar si un usuari validat té el rol adequat per a accedir a un recurs, podem definir una altra funció *middleware* que comprove si el rol de l'usuari és el que es necessita (el que se li passa com a paràmetre a la funció):

```
let rol = (rol) => {
    return (req, res, next) => {
        if (rol === req.session.rol)
            next();
        else
            res.render('login');
    }
}
```

> **NOTA**: l'exemple que acabem de veure és una mostra de com podem definir *middleware* que necessite paràmetres addicionals a més dels tres que tot *middleware* ha de tindre (petició, resposta i següent funció a cridar). N'hi ha prou amb definir una funció amb els paràmetres necessaris, i que internament retorne la funció *middleware* amb els tres paràmetres base.

Si volem aplicar els dos *middleware* a una ruta determinada (és a dir, comprovar si l'usuari està autenticat i, a més, si té el rol adequat), podem passar-los un darrere l'altre, separats per comes, en la definició de la ruta. Per exemple, a aquesta ruta només han de poder accedir usuaris validats que tinguen rol d'administrador:

```
app.get('/protegitAdmin', autenticacio,
    rol('admin'), (req, res) => {
    res.render('protegit_admin');
})
```

4. Altres opcions

A més de les opcions vistes anteriorment, hi ha algunes operacions més que, si bé poden ser secundàries, convé tindre presents quan treballem amb autenticació basada en sessions.

4.1. Tancament de sessió o *logout*

D'una banda, està la possibilitat de fer **logout** i eixir de la sessió. Per a això, podem definir una ruta que responga a aquesta petició, i destruïsca les dades de sessió de l'usuari, redirigint després a un altre recurs:

```
app.get('/logout', (req, res) => {
    req.session.destroy();
    res.redirect('/');
});
```

4.2. Accedir a la sessió des de les vistes

Per a poder accedir a la sessió des de les vistes, hem de definir un *middleware* que associe la sessió amb els recursos de la vista:

```
app.use((req, res, next) => {
    res.locals.session = req.session;
    next();
});
```

> **NOTA**: aquest middleware ha de definir-se després del middleware que configura la sessió i abans dels encaminadors, perquè tinga efecte en renderitzar les vistes.

Després, podem accedir a aquesta sessió des de les vistes, a través de la variable `session` que hem definit en la resposta (`res.locals`). Per exemple, així podríem veure si un usuari està ja logueado, per a mostrar o no el botó de "Login":

```
{% if (session and session.usuari) %}
    <a class="btn btn-dark" href="/logout">Logout</a>
{% else %}
    <a class="btn btn-dark" href="/login">Login</a>
{% endif %}
```

4.3. Temps de vida de la sessió

A més, podem establir el **temps de vida** de la sessió, quan la configurem. Podem fer-ho utilitzant indistintament l'atribut `expires` o l'atribut `maxAge` , encara que amb una sintaxi una mica diferent segons quin utilitzem. Hem d'indicar el nombre de mil·lisegons de vida, comptant des del moment actual, per la qual cosa se sol utilitzar `Date.now()` en aquests càlculs. Així definiríem, per exemple, una sessió de 30 minuts:

```
app.use(session({
    secret: '1234',
    resave: true,
    saveUninitialized: false,
    expires: new Date(Date.now() + (30 * 60 * 1000))
}));
```

Ací pots descarregar un exemple complet per a provar aquests mecanismes. Es té una pàgina d'inici pública, una restringida per a usuaris validats i una altra restringida per a usuaris administradors. Es disposa també d'un formulari de *login* i d'una ruta de *logout*.

> **Exercici 1:**

> Crea una còpia de l'exercici *LlibresWeb_v4* i crida-la **LlibresWebSessions**. A partir d'aqueixa base, afegirem ara autenticació basada en sessions. Instal·la el *middleware express-session* en el projecte, i configura'l com en l'exemple vist abans. Defineix a mà en el servidor principal un array amb noms i passwords d'usuaris autoritzats, i protegeix les rutes que permeten fer qualsevol modificació sobre el catàleg de llibres. En concret, només els usuaris validats podran:
>
> - Veure el formulari d'inserció de llibres i inserir llibres (enviar el formulari anterior)
> - Esborrar llibres
> - Veure el formulari d'edició de llibres i editar llibres (enviar el formulari)
>
> Afig per a això una vista `login.njk` al conjunt de vistes de l'aplicació. Pots emprar el mateix formulari de login que en l'exemple, i també afig les dues rutes per a mostrar el formulari i per a recollir les dades i validar l'usuari. En cas de validació reeixida, es renderitzarà la vista del llistat de llibres. En cas contrari, el formulari de login amb un missatge d'error, com en l'exemple proporcionat.
>
> Afig també una funció de *logout* al menú de l'aplicació, que només serà visible si l'usuari ja està validat, i que permetrà destruir la seua sessió i redirigir al llistat de llibres.