Tema 5

Uso de Sesiones y Autenticación

Seguridad en nuestra App

- Actualmente, uno de los puntos más importantes que debemos plantearnos a la hora de crear una aplicación web es el tema de la seguridad.

- Diseñar y desarrollar qué diferentes niveles de acceso vamos a permitir en nuestra aplicación o qué tipo de usuarios vamos a poder crear y cómo van a interactuar entre ellos.

- Para esta tarea, podemos servirnos de servicios de autenticación de terceros (Twitter, GitHub), o crear nuestro propio servicio, en el cual seamos los responsables de la autenticación y de la autorización de los diferentes usuarios.

- En este tema vamos a ver diferentes maneras para incluir seguridad en nuestras aplicaciones.

Autorización con Express

- Normalmente, el objetivo que queremos alcanzar implementando un servicio de autenticación y autorización dentro de nuestras aplicaciones es el de limitar el acceso de los usuarios a diferentes funciones.

- Estas funciones pueden ser páginas de nuestro sitio web, métodos en concreto, o accesos a través de algún servicio API que hayamos desarrollado.

- Express posee un Middleware que nos permite aplicar ciertas reglas sobre todas las rutas o sobre grupos de rutas de todas las que tengamos definidas en nuestro proyecto.

Autorización con Express

- Las rutas en Express, podemos definirlas de la siguiente manera:

```
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});
```

- Antes de procesar la petición con la función que especificamos, podemos interactuar con la petición a través de otra función que pasaremos como segundo parámetro a nuestra sentencia:

```
router.get('/', auth, function(req, res, next){
  res.render('index', { title: 'Express' });
});
```

Autorización con Express

- La función **auth** que hemos definido cuenta con los parámetros **req**, **res** y **next**.

- En esta función podemos integrar las sentencias que necesitemos antes de que se procese la petición, devolviendo cualquier respuesta a través de la variable **res** o devolviendo **next** en caso de no tener la necesidad de interactuar.

```
var auth = function(req, res, next){
   console.log ('Entra en auth');
   //Podemos devolver res.send(401), en caso de error por ejemplo
   return next();
}
```

- La llamada a next es importante ya que es la forma que tiene Express de evaluar posibles peticiones posteriores.

Autenticación basada en Sesiones

- La autenticación basada en sesiones se realiza mediante el objeto **session** perteneciente al objeto petición que encontramos en todas las funciones callback que tratan cada una de las peticiones.

- Una sesión web generalmente, es una manera segura de almacenar información del cliente a través de las diferentes peticiones que realiza sobre nuestro sitio web.

- Para trabajar con sesiones en Express tenemos que importar los módulos necesarios, en este caso **cookie-parser** y **express-session**.

- Como cualquier módulo, podemos instalarlo a través de **npm**.

```
npm install cookie-parser
npm install express-session
```

Autenticación basada en Sesiones

- Tenemos que asegurarnos que los módulos estén importados en el proyecto donde queramos usarlo.

```
var cookieParser = require('cookie-parser');
var session = require('express-session');
```

```
app.use(cookieParser());
app.use(session({
  secret: 'frase secreta',
  resave: false,
  saveUninitialized: true
}));
```

- Para la definición de la sesión tenemos que especificar una serie de opciones que nos permiten definirla.

- Por ejemplo, un token secreto que permita codificar las peticiones.

Autenticación basada en Sesiones

- Estos dos módulos instalados son utilizados para:
- **cookieParser ->** nos permite tratar las cookies que encontramos dentro del navegador del cliente o de la petición que estemos tratando.
- **session ->** coloca el objeto **res.session** en cada una de las peticiones que se realicen sobre nuestra aplicación. Almacena datos en la memoria de nuestra aplicación o en cualquier otro sistema persistente como podría ser una base de datos MongoDB.

Autenticación basada en Sesiones

- Para poder trabajar con las variables de sesión, simplemente tenemos que modificar el objeto **session** contenido dentro de cada una de las peticiones de nuestra aplicación:

```
router.get('/usuarios/', function(req, res, next){
    //Guardamos la variable de sesión saludo
    req.session.saludo = 'Hola Mundo';
    res.redirect ('/');
});
```

- En esta primera petición estamos creando o actualizando el valor de una variable de sesión llamada saludo.

Autenticación basada en Sesiones

- Para acceder al valor almacenado, procederemos de la siguiente manera:

```
router.get('/', function(req, res, next) {
   console.log (req.session.saludo);
   res.render('index', { title: 'Express' });
});
```

- En este caso, mostraremos por consola el valor contenido en la variable saludo (*Hola Mundo*).

- De esta manera, podremos almacenar datos de sesión a lo largo de las peticiones de nuestra aplicación mientras que la sesión siga abierta (normalmente, cuando se mantiene abierto el navegador donde se está ejecutando la aplicación.

PassportJS

- Passport se trata de un middleware de autenticación para Node.
- El objetivo de este módulo es simple: **autenticar las peticiones** de nuestra aplicación.
- Es muy sencillo de utilizar y de integrar en cualquier aplicación escrita para Node. Al trabajar como un middleware nos permite separar la lógica creada para la autenticación de nuestros usuarios de la ya existente propia de la aplicación.
- Aparte, es muy sencillo de integrar con cualquier instalación de **Express**, al igual que hacemos con otros middleware como puede ser *cookie-parsing* o *sessions.*

npm install passport

PassportJS - Authenticate

- Para autenticar a los usuarios dentro de nuestro sistema, simplemente tenemos que hacer una llamada a **passport.authenticate ('local')**.

- En este caso le estamos especificando además, qué sistema de autenticación vamos a utilizar de los que nos ofrece passport. El parámetro *local* indica que la autenticación de usuarios va a ser sobre una instancia de base de datos incorporada en nuestro proyecto.

```
router.post('/login', passport.authenticate('local'), function (req, res){
    //Si esta función se ejecuta, quiere decir que la autenticación
    //ha sido correcta
    //Dentro del objeto req.user tendremos los datos del usuario autenticado
});
```

- Si la autenticación falla, por defecto, Passport devuelve un estado **401 Unauthorized**.

PassportJS - Authenticate

- Podemos especificar qué va a pasar tanto si la autenticación es correcta como si devuelve algún tipo de fallo:

```
router.post('/login', passport.authenticate('local',
                { successRedirect: ‘/‘,
                failureRedirect: '/login'}));
```

- Dependiendo de la respuesta de la autenticación, redireccionaremos la petición hacía una url u otra.

PassportJS - Strategy

- Como ya hemos comentado anteriormente, la manera más sencilla de interactuar con Passport es a través de una base de datos local.

- Para ello, necesitamos instalar un módulo aparte

npm install passport-local

- Este módulo nos permite trabajar con objetos de tipo **Strategy**, mediante los cuales vamos a definir las diferentes formas de actuar dependiendo del paso de la autenticación donde nos encontremos. (login, signup, logout…).

PassportJS - Strategy

- Si trabajamos con ***local*** un ejemplo de Strategy podría ser el siguiente:

```
var passport = require('passport'),
    LocalStrategy = require('passport-local').Strategy;

passport.use(new LocalStrategy (
    function(username, password, done){
        // Dentro de esta función haremos las comprobaciones
        // necesarias contra nuestra base de datos para saber
        // si el usuario existe y además coincide con la password
        return done(null, user);
    })
);
```

- Mediante la llamada final al callback **done** estamos indicando el resultado de la validación. El primer parámetro indica si ha habido error o no (null en este caso) y en el segundo parámetro estamos devolviendo el usuario autenticado para poder trabajar con él.

PassportJS - Strategy

- El formulario que podríamos utilizar para obtener estos datos del usuario podría ser (en HTML), como el siguiente:

```
<form action="/login" method="post">
    <div>
        <label>Username:</label>
        <input type="text" name="username"/>
    </div>
    <div>
        <label>Password:</label>
        <input type="password" name="password"/>
    </div>
    <div>
        <input type="submit" value="Log In"/>
    </div>
</form>
```

PassportJS - Strategy

- Podemos definir nuestros propios objetos Strategy para atacar diferentes acciones dentro de nuestro sistema de autenticación.

```
passport.use('signup', new LocalStrategy(
   function(username, password, done){
      // Desarrollamos las acciones necesarias para
      // registrar al usuario dentro de nuestro sistema
   })
);
```

- La petición podría ser la siguiente:

```
router.post('/signup', passport.authenticate('signup', {
            successRedirect: '/',
            failureRedirect: '/signup'}));
```