Autenticación con JSON Web Tokens

En este ejemplo veremos como podemos **manejar usuarios**, incluyendo **autenticación** (i.e login),
y como podemos proteger las rutas que nos interese que sólo puedan acceder usuarios registrados y logueados.

Cómo funciona?

Resumidamente, estos son los pasos que se siguen en el proceso de autenticación con **JSON Web Tokens**:

- Primero, el **usuario debe estar registrado**.
- Luego, con la intención de hacer login, el usuario hace una **petición** de tipo **`POST`** a una ruta de nuestra API, típicamente `/token`, `/authentication`, `/login`, etc. aunque podría ser la que quisieramos. En la petición POST nos envía como información al servidor su **nombre de usuario y contraseña**.
- La función que haga el tratamiento de la petición de autenticación:

  ```
  app.post('/authentication', function meReFieroAEstaFuncion() {...}))
  ```

  buscará el usuario en la base de datos, y comprobará que las contraseñas coincidan.
- Si coinciden, entonces procederemos con la *magia arcana del hashing*. **Encriptaremos los datos del usuario** (username, password + lo que queramos) junto con una llave privada, llamada *secret*. Este proceso lo haremos con el módulo **`jsonwebtoken`**.
- El string resultante de la encriptación que nos devuelva jsonwebtoken **lo enviaremos al usuario** como respuesta.
- El usuario usará este token siempre que envie una petición a nuestra API. Concretamente, lo **especificará en el header** de la petición, (ya veremos como hacerlo
  en Postman y cuando veamos AngularJS).
- Cuando el servidor reciba la petición, podrá **asegurarse que el usuario está logueado** mirando el token que este ha enviado con la petición.
- Ahora, dado que al encriptar se ha almacenado la información del usuario en el token, podremos acceder a ella en `req.user`. Por ejemplo, si al encriptar hemos encriptado los atributos `username`, `password` y edad, cuando recibamos una petición con un token correcto, en `req.user.username`, `req.user.password` y `req.user.edad` tendremos los valores que habíamos pasado al encriptarlo. Este paso se lleva a cabo con el módulo **`express-jwt`**, que actua como middleware comprobando el token en aquellas peticiones que queramos que requieran de autenticación, y asignando el valor de req.user con la información que hay codificada en el token. Notad que en el ejemplo se indica como podemos cambiar el atributo del objeto req en el que express-jwt asigna la información, a, por ejemplo, `req.usuario`, para personalizar esta opción si no nos gusta `req.user`.

Sobre el ejemplo

En este ejemplo, usaremos las rutas del usuario para mostrar cómo funciona la autenticación.

Lo importante aquí es dónde ponemos el middleware `express-jwt`. Debemos definir una ruta donde se hará la autenticación, pero a veces no necesitamos que todas las rutas requieran de autenticación. Y es que la ruta para hacer login (i.e autenticación) no puede requerir que el usuario esté autenticado!

Una posible forma de hacerlo es definir todos los caminos que necesitan autenticación en `/api/...` (`/api/users` o `/api/tasks` etc.) añadiendo el middleware de autenticación en `/api`. Otra forma es definir la autenticación llamando a la función unless() y pasandole un array con las rutas que no requieren de autenticación, por ejemplo:

```
// Aquí especificamos que todas los paths excepto authentication y register
requerirán de autenticación
app.use(jwt({ secret: 'some-secret' }).unless( { path: ['/authentication',
'/register'] }))
```

En este ejemplo, las rutas que requieren de autenticación son todas del recurso users excepto la P0ST a `/users`, es decir, el registro. De este modo, el middleware de autenticación se define en las rutas que lo necesitan (ver `routes/user.js`)

Para leer más sobre autenticación

Repositorios de los módulos usados

- https://github.com/auth0/node-jsonwebtoken
- https://www.npmjs.com/package/express-jwt

Herramientas y frameworks para autenticacion más compleja

- https://github.com/GrumpyWizards/DarkLord
- http://passportjs.org/
- https://auth0.com/