JWT

Como primer paso para empezar aplicar este mecanismo de autenticación, descargaremos la libreria jsonwebtoken a través de npm.

npm i jsonwebtoken --save

Haremos un breve recorrido por la aplicación para entender cómo funciona una autenticación con JWT. Es recomendable leer la documentación oficial antes de seguir este manual para la implementación de este mecanismo de seguridad.

Antes de continuar, vamos a utilizar como mecanismo para encriptar el token una clave privada y, para verificar el contenido del token, una clave pública. Este tipo de claves pueden generarse de forma online desde :
https://travistidwell.com/jsencrypt/demo/

Una vez generadas, se creará la carpeta claves dentro de nuestro rest (a nivel del app.js) y se crearán dos archivos. En privada.pem se pegará la clave privada que generamos en travistidwell y en el publica.pem la clave pública generada en el sitio.

El mecanismo de autenticación por token se valida de la siguiente forma :

Un usuario se loguea con un mecanismo tradicional de usuario y contraseña. Esta información (que se envía al servidor a través de un formulario) se dirige a la ruta /auth/login.

Auth está declarada dentro del app.js y auth es un controller que tiene la siguiente estructura :

```
const express = require('express');
const router = express.Router();
const jwt = require('jsonwebtoken');
const fs = require('fs'); // file system (operaciones de archivos)
const md5 = require('md5');
const usuariosModel = require('../models/usuariosModel');

router.post('/login', async(req,res,next) => {
   try {

      let usuario_ok = await usuariosModel.getUser(req.body.mail, md5(req.body.password));

         var signOptions = {
            expiresIn : "2h",
            algorithm : "RS256"
         }
```

```
            const privateKey = fs.readFileSync('./claves/privada.pem','utf-8');
            const payload = {id : usuario_ok[0].id_u, nombre : usuario_ok[0].nombre_u};
            const usuario = {nombre : usuario_ok[0].nombre_u}
            const token = jwt.sign(payload, privateKey, signOptions);
            res.json({usuario, JWT : token});



        } else {
            res.json({status : "invalid"});

        }


    } catch(error) {
        console.log(error);
        res.status(500).json({status : 'error'})

    }
});


module.exports = router;
```

Lo importante a tener en cuenta es que cuando los usuarios entran por este controller, lo primer que hacemos es una petición al model para verificar si efectivamente el usuario existe dentro de nuestra tabla. También comprobamos que la cuenta esté validada (correo verificado)

Si estas condiciones se cumplen, comienza a armarse el JWT. Este está compuesto por 3 partes. **Header, payload y firma digital.**
Para esto, configuramos las **signOptions** determinando el tiempo de expiración y el cifrado que tendrá este token. Cargamos **la clave privada** que usaremos para encriptar este token y generamos el **payload** enviando el id y los permisos que tendrá el usuario que generó el token. Una vez realizado esto, se genera y luego se envía en un JSON al cliente.

Una vez recibido, éste se guardará dentro del navegador del cliente (dentro de un storage) y el cliente enviará peticiones al servidor autenticandose con este token.

A modo de validar estas rutas, se deberá definir cual son rutas verificadas y cuales serán públicas sin validación previa. Para esto, dentro del app.js se verificará si el usuario envía un token y si el cliente está accediendo a una ruta privada o no.

Para esto, definiremos un middleware llamado secured :
secured = (req,res,next) => {

```
  let token = req.headers.authorization;

  token = token.replace('JWT ','');
  const publicKey = fs.readFileSync('./claves/publica.pem');
  let decoded = jwt.verify(token, publicKey);
    req.user_id = decoded.id;
    next();

}
```

Cabe destacar que por diagrama de API REST el cliente siempre enviará el token con el siguiente formato :
JWT token o bien Bearer token.
Por eso usamos el método replace. Lo que le interesa al server para autenticar el usuario es el contenido. JWT O Bearer simplemente es una convención que se usa para enviar el token del cliente al servidor. Dentro de este middleware utilizaremos la clave pública para verificar el acceso y dentro del objeto req crearemos la propiedad user_id y le daremos el valor que llega del cliente.
Otro dato importante es que dentro de req.headers vemos la propiedad authorization.
Es importante entender que el protocolo http se comunica entre cliente y servidor a través de cabeceras o headers. Además de definir los verbos http, en estas se envían las autorizaciones. Authorization es un parámetro en el que se envían las autenticaciones basadas en token.

Por último, pasamos este middleware dentro del app.use donde definimos las rutas para determinar si son de acceso público o no.