Roles y permisos con JSON Web Tokens

U-TAD

JWT / Auth

- Autenticación/autorización entre el cliente y nuestro server
- Proteger rutas bajo autorización
- Utilizaremos JWT
  - https://jwt.io/
  - Standard RFC 7519 para representar la autorización entre dos partes.
  - Es un Token (cadena de texto) con tres secciones (Header, Payload y Signature):
    i. Tipo de algoritmo de cifrado que se utilizó, normalmente *alg: HS256,* y el token-type
    ii. Data con subject (id), nombre y rol si se quiere. (son datos públicos)
    iii. Firmado con el secret: verifica la integridad del mensaje e identifica al “sender”

JWT / Auth

¿Cómo lo usamos en node?

Instalamos dos bibliotecas, para JWT y para Auth respectivamente:

```
npm i jsonwebtoken bcryptjs
```

En el directorio *routes/* creamos *auth.js* y en *validators/* también un *auth.js*

Para el validator de auth.js tomamos como referencia el modelo de users (UserScheme), donde nos quedaremos con “name”, “age”, “email” y “password”

JWT / Auth (validator/auth.js)

```
const { check } = require("express-validator")
const validateResults = require("../utils/handleValidator")
const validatorRegister = [
    check("name").exists().notEmpty().isLength( {min:3, max: 99} ),
    check("age").exists().notEmpty().isNumeric(), //Puedes aplicarle un min y max también al
número
    check("email").exists().notEmpty().isEmail(),
    check("password").exists().notEmpty().isLength( {min:8, max: 16} ),
    //check("role").optional()  //TODO Actualizar el enum de "models/nosql/users.js"
    (req, res, next) => {
        return validateResults(req, res, next)
    }
]
const validatorLogin = [
    check("email").exists().notEmpty().isEmail(),
    check("password").exists().notEmpty().isLength( {min:8, max: 16} ),
    (req, res, next) => {
        return validateResults(req, res, next)
    }
]
module.exports = { validatorRegister, validatorLogin }
```

JWT / Auth (utils/handlePassword)

```
const bcryptjs = require("bcryptjs")

const encrypt = async (clearPassword) => {
// El número "Salt" otorga aleatoriedad a la función hash al combinarla con la password en claro.
    const hash = await bcryptjs.hash(clearPassword, 10)
    return hash
}

const compare = async (clearPassword, hashedPassword) => {
// Compara entre la password en texto plano y su hash calculado anteriormente para decidir si coincide.
    const result = await bcryptjs.compare(clearPassword, hashedPassword)
    return result
}

module.exports = { encrypt, compare }
```

JWT / Auth (utils/handleJwt.js)

En el fichero **.env** añadimos la entrada **JWT_SECRET=<MasterKey>**

```
const jwt = require("jsonwebtoken")
const JWT_SECRET = process.env.JWT_SECRET

const tokenSign = (user) => {
    const sign = jwt.sign(
        {
            _id: user._id,
            role: user.role
        },
        JWT_SECRET,
        {
            expiresIn: "2h"
        }
    )
    return sign
}

const verifyToken = (tokenJwt) => {
    try {
        return jwt.verify(tokenJwt, JWT_SECRET)
    }catch(err) {
        console.log(err)
    }
}

module.expots = { tokenSign, verifyToken }
```

JWT / Auth (routes/auth.js)

```
const express = require("express")
const { matchedData } = require("express-validator")
const { encrypt, compare } = require("../utils/handlePassword")
const {usersModel} = require("../models")
const router = express.Router()
const {validatorRegister, validatorLogin} = require("../validators/auth")

// Posteriormente, llevaremos la lógica al controller
router.post("/register", validatorRegister, async (req, res) => {
    req = matchedData(req)
    const password = await encrypt(req.password)
    const body = {...req, password} // Con "..." duplicamos el objeto y le añadimos o sobreescribimos una propiedad
    const dataUser = await usersModel.create(body)
    dataUser.set('password', undefined, { strict: false })
    const data = {
       token: await tokenSign(dataUser),
       user: dataUser
    }
    res.send(data)
})
//TODO router.post("/login", (req, res) => {}

module.exports = router

// Pruébalo con una petición POST a http://localhost:3000/api/auth/register con {name, age, email, password}
// Después copia el token de la respuesta, y pégalo en el debugger de https://jwt.io (necesitarás la MasterKey)
```

JWT / Auth (routes/auth.js y controllers/auth.js)

Lleva la lógica del “async (req, res) => { … } a controllers/auth.js

De modo que **routes/auth.js** quedaría tal que

```
const express = require("express")
const { validatorRegister, validatorLogin } = require("../validators/auth")
const { registerCtrl,loginCtrl } = require("../controllers/auth")
const router = express.Router()

router.post("/register", validatorRegister, registerCtrl})
router.post("/login", validatorLogin, loginCtrl}) //TODO: loginCtrl en controllers

module.exports = router
```

```
// TODO: controller/auth.js (intentadlo vosotros dejando vacía la función loginCtrl y usando el handleHttpError)
...
const loginCtrl = asyn (req, res) => {
    //TODO
}
module.exports = { registerCtrl, loginCtrl }
```

JWT / Auth (controllers/auth.js)

Completamos la función loginController:

```
const loginCtrl = async (req, res) => {
    try {
        req = matchedData(req)
        const user = await usersModel.findOne({ email: req.email }).select("password name role email")
        if(!user){
            handleHttpError(res, "USER_NOT_EXISTS", 404)
            return
        }
        const hashPassword = user.password;
        const check = await compare(req.password, hashPassword)
        if(!check){
            handleHttpError(res, "INVALID_PASSWORD", 401)
            return
        }
        user.set("password", undefined, {strict:false}) //Si no queremos que se muestre el hash en la respuesta
        const data = {
            token: await tokenSign(user),
            user
        }
        res.send(data)
    }catch(err){
        console.log(err)
        handleHttpError(res, "ERROR_LOGIN_USER")
    }
}
```

JWT / Auth (test)

En el index.http prueba con la clave correcta, **y otra que no lo sea**, respecto a cuando haces el “register”.

```
POST http://localhost:3000/api/auth/register HTTP/1.1
Content-Type: application/json

{
    "name": "Menganito",
    "age": 20,
    "email": "test10@test.com",
    "password": "HolaMundo.01"
}

###

POST http://localhost:3000/api/auth/login HTTP/1.1
Content-Type: application/json

{
    "email": "test10@test.com",
    "password": "HolaMundo.01"
}
```

JWT (Cómo utilizar el token de sesión, protege rutas)

En el directorio *middleware/* creamos el fichero *session.js.* Y en la llamada, requeriremos una header *Authorization = Bearer <token>*

```
const { handleHttpError } = require("../utils/handleError")
const { verifyToken } = require("../utils/handleJwt")

const authMiddleware = async (req, res, next) => {
    try{
        if (!req.headers.authorization) {
            handleHttpError(res, "NOT_TOKEN", 401)
            return
        }
// Nos llega la palabra reservada Bearer (es un estándar) y el Token, así que me quedo con la última parte
        const token = req.headers.authorization.split(' ').pop()
//Del token, miramos en Payload (revisar verifyToken de utils/handleJwt)
        const dataToken = await verifyToken(token)
        if(!dataToken._id) {
            handleHttpError(res, "ERROR_ID_TOKEN", 401)
            return
        }
        next()
    }catch(err){
        handleHttpError(res, "NOT_SESSION", 401)
    }
}

module.exports = authMiddleware
```

JWT (Cómo utilizar el token de sesión, protege rutas)

Vamos a la ruta que queramos proteger, por ejemplo, en routes/track.js

```
const authMiddleware = require("../middleware/session")
...
router.get("/", authMiddleware, getItems)
```

Y ahora lo probamos desde el cliente:

Haz login en `POST http://localhost:3000/api/auth/login` Y obtén el Token.

Copialo en la siguiente llamada tal que:

```
GET http://localhost:3000/api/tracks
Authorization: Bearer <tuToken>
```

Prueba sin la cabecera Authorization y/o cambiando el valor del Token, debería fallar.

JWT (Cómo utilizar el token de sesión, protege rutas)

Para conocer los datos del cliente en el controlador, en **middleware/session.js**

```
const { usersModel } = require("../models")
const authMiddleware = async (req, res, next) => {
    try{
…
      const user = await usersModel.findById(dataToken._id)
      req.user = user // Inyecto al user en la petición
      next()
…
```

Y ahora en controllers/tracks.js

```
const getItems = async (req, res) => {
    try{
        const user = req.user
        const data = await tracksModel.find({})
        res.send({data, user})// Tengo todos los datos el cliente
…
```

Roles

Además de la autenticación vista anteriormente con JWT, a través del rol definiremos la **autorización**. Por ejemplo, solo el rol *admin* podrá hacer POST.

En routes/tracks.js, añadimos ***authMiddleware*** a todas las rutas.

…

```
router.post("/", authMiddleware, validatorCreateItem, createItem)
```

…

Roles

Ahora en el directorio **middleware/** creamos el fichero **rol.js**

```
const { handleHttpError } = require("../utils/handleError")
const checkRol = (roles) => (req, res, next) => { // Doble argumento
    try{
        const {user} = req
        const userRol = user.role
        const checkValueRol = roles.includes(userRol)//Comprobamos que el rol del usuario esté en roles
        if (!checkValueRol) {
            handleHttpError(res, "NOT_ALLOWED", 403)
            return
        }
        next()
    }catch(err){
        handleHttpError(res, "ERROR_PERMISSIONS", 403)
    }
}
module.exports = checkRol
```

Importa y añade este middleware a la ruta POST de tracks.js pasándole la lista de roles aceptados, en este caso solo “admin”

```
router.post("/", authMiddleware, checkRol(["admin"]), validatorCreateItem, createItem)
```

NOTA: Para cambiar el rol por defecto “user” a “admin”, deberás hacerlo en Base de Datos desde Mongo Atlas.

Roles

Ejercicio

Habilita una ruta PUT en users o auth, para que un admin pueda modificar un usuario y hacerle admin.

Comprueba que solo un usuario autenticado y con rol admin puede cambiar los datos de un usuario. Una vez cambiado, comprueba en Base de Datos que ahora el usuario modificado tiene rol “admin”