Autenticacion JWT NODE

- Quiero un endpoint que sea /api/auth/login
- Hay que especificarlo en el server, en un controlador y una ruta
- Creo el path en el constructor del server

> this.authPath= '/api/auth'

- Donde tengo las rutas creo una nueva (no tengo el authRouter todavía)

```
routes(){

    this.app.use(this.usuariosPath, userRouter)
    this.app.use(this.authPath, authRouter)
}
```

- Creo el authRouter en un nuevo archivo llamado /routes/auth.router.js
- Añado la ruta y el controlador que todavía no he creado

```
import {Router} from 'express'
import { validarCampos } from '../middlewares/validar-campos.js'


const router = Router()

router.post('/login', auth.LoginController)

export default router
```

- auth.controller.js:
- Desestructuro el correo y el password del body

```
const LoginController=(req, res)=>{
      const {correo, password} = req.body

    res.json({
        correo,
        password
    })
}



export default{
    LoginController
}
```

- De esta manera ya tengo todo conectado
- Escribo en el body (en Thunder Client---->body--->json) el json que necesito

```
{
  "correo": "pere@correo.com",
  "password": "123456"
}
```

- Uso el check del express-validator para validar el correo y el password en la ruta
- Coloco el validarCampos para recolectar los errores con la función validationResult del middleware

```
import {Router} from 'express'
import auth from '../controllers/auth.controller.js'
import { check } from 'express-validator'
import { validarCampos } from '../middlewares/validar-campos.js'


const router = Router()

router.post('/login',[
    check('correo', 'El correo es obligatorio').isEmail(),
    check('password', 'El password es obligatorio').not().isEmpty(),
    validarCampos
], auth.LoginController)

export default router
```

- Recordatorio validarCampos

```
import { validationResult } from "express-validator"


export const validarCampos = (req,res, next)=>{

    const errors = validationResult(req)

    if(!errors.isEmpty()){
        return res.status(400).json({
            errors
        })
    }
    next()

}
```

Login

- Coloco el código dentro de un try y un catch por si algo sale mal
- debo verificar si el usuario existe, si está todavía en mi DB (estado en true) y si el password coincide
    - Después generaré el JWT
- auth.controller.js

```
import Usuario from '../models/usuario.js'
import bcryptjs from 'bcryptjs'

const LoginController=async (req, res)=>{
    const {correo, password} = req.body

    try {
       const usuario = await Usuario.findOne({correo});

       if(!usuario) return res.status(400).json({msg: "Usuario / Password no son
correctos -  correo"})

       if(usuario.estado === false) return res.status(400).json({msg: "Usuario /
Password no son correctos - estado"})

       const validPassword = bcryptjs.compareSync(password, usuario.password)
//true o false

       if(!validPassword) return res.status(400).json({msg: "Usuario / Password no
son correctos - password"})

       res.status(200).json({msg: "login ok"})


    } catch (error) {

        console.log(error)
        res.status(400).json({
            msg: "Hable con el administrador"
        })
    }
}

export default{
    LoginController
}
```

Generar JWT

- Instalo con npm i jsonwebtoken
- Actualmente jsonwebtoken no tiene una promesa para generar el jsonwebtoken, con lo cual es un callback que necesito transformar en una promesa

- Lo que yo quiero es algo así en el auth.controller

```
const token = await generarJWT(usuario.id)  //pasarle el id al token
```

- Creo en helpers el archivo generarJWT.js

```
const generarJWT = (uid="")=>{
    //podría ser async pero como trabaja con callbacks voy a tener que generar la promesa manualmente
    return new Promise((resolve,reject)=>{
        const payload = {uid} //voy a grabar el uid en el jwt

        //le paso el payload, la clave secreta, las opciones y el callback donde está el token que voy a tener que resolver
        jwt.sign(payload, process.env.SECRET_KEY, {
            expiresIn: '4h'  //quiero que el token solo viva 4 horas
        }, (err, token)=>{//le paso el callback
            if(err){
                console.log(err)
                reject('No se pudo generar el jwt')
            }else{
                resolve(token)
            }
        })
    })
}
```

- Mando el token en la respuesta para comprobar que todo marcha bien

```
import generarJWT from '../helpers/generarJWT.js';
import Usuario from '../models/usuario.js'
import bcryptjs from 'bcryptjs'

const LoginController=async (req, res)=>{
    const {correo, password} = req.body

    try {
       const usuario = await Usuario.findOne({correo});
                                                    //coloco estos mensajes para identificar el origen del error
       if(!usuario) return res.status(400).json({msg: "Usuario / Password no son correctos -  correo"})

       if(usuario.estado === false) return res.status(400).json({msg: "Usuario / Password no son correctos - estado"})

       const validPassword = bcryptjs.compareSync(password, usuario.password)
```

```
//true o false

       if(!validPassword) return res.status(400).json({msg: "Usuario / Password no
son correctos - password"})
        //generar TOKEN
       const token = await generarJWT(usuario.id)

       res.status(200).json({msg: "login ok", token})

    } catch (error) {

        console.log(error)
        res.status(400).json({
            msg: "Hable con el administrador"
        })
    }

}
```

- Vamos a usar el token para proteger las rutas que necesiten autenticación

---

Cambiar visualmente _id por uid en Mongoose

- Cuando hago un login no quiero que ponga _id, quiero que ponga uid
- Sobreescribo el método en el modelo

```
usuarioSchema.methods.toJSON = function(){
    const {_id,...usuario} = this.toObject()
    usuario.uid = _id
    return usuario
}
```

Proteger rutas mediante el uso del Token - Middlewares

- La ruta de deleteUsuario no debería de ser pública
- Yo no debería poder cambiar su estado a false sin estar autenticado
- Puedo especificarlo por roles, para lo cual debo estar autenticado
- La primera validación que vamos a hacer es que tenga un JWT válido
- Creo el archivo /middlewares/validar-jwt.js
- Normalmente los tokens de acceso van en los headers
- En Headers (Thunder Client), en el campo KEY pongo x-token y en el campo VALUE pego el JWT extraido del login
- Hago pruebas con el endpoint de delete

```
import jwt from 'jsonwebtoken'
```

```
const validarJWT = (req, res, next) =>{
    const token = req.header('x-token')  //x-token, así lo he llamado, puedo
llamarlo como quiera

    console.log(token)
    next()
}

export default validarJWT
```

- Uso el middleware en el user.routes
- Lo coloco el primero
-

```
router.delete('/:id', [
    validarJWT,
    check('id', 'No es un id válido').isMongoId(),
    check('id').custom(userExistsById),
    validarCampos
],
userController.usuariosDelete)
```

- Me imprime en consola el token, lo tengo!
- Hago la verificación del jwt dentro de un try y un catch
- Desestructuro el uid que había introducido en el payload del token
- Paso el uid en el request

```
import jwt from 'jsonwebtoken'

const validarJWT = (req, res, next) =>{
    const token = req.header('x-token')

    if(!token) return res.status(401).json({msg: "No hay token en la petición"})

    try {
        //verifico el jwt y extraigo el uid
        const {uid} = jwt.verify(token, process.env.SECRET_KEY)

        //creo una propiedad nueva dentro de la request para pasar el uid
        req.uid = uid
        next()

    } catch (error) {
        console.log(error)
        return res.status(401).json({msg:"Token no válido"})
    }

    next()
}
```

```
export default validarJWT
```

- Entonces, ahora puedo extraer el uid del req.uid

```
const usuariosDelete =  async (req,res) =>{
   const {id} = req.params
   const uid = req.uid //lo obtengo del middleware validarJWT

    const usuario = await Usuario.findByIdAndUpdate(id, {estado:false}) //si
quisiera borrarlo fisicamente usaría finsByIdAndDelete(id)

    res.json({
        usuario,
        uid
    })

}
```

- Ahora lo que quiero es que, por ejemplo, solo los ADMIN_ROLE puedan borrar

---

Obtener info del usuario autenticado

- En lugar del uid extraigo el usuario desde el middleware validarJWT

```
import jwt from 'jsonwebtoken'
import Usuario from '../models/usuario.js'

const validarJWT = async (req, res, next) =>{
    const token = req.header('x-token')

    if(!token) return res.status(401).json({msg: "No hay token en la petición"})

    try {
        //verifico el jwt y extraigo el uid
        const {uid} = jwt.verify(token, process.env.SECRET_KEY)

        //Encuentro el usuario y se lo paso al request
         const usuario = Usuario.findById(uid)
            req.usuario = usuario

        next()

    } catch (error) {
        console.log(error)
        return res.status(401).json({msg:"Token no válido"})
    }
}
```

```
export default validarJWT
```

- En el controlador extraigo el usuario de req.usuario

```
const usuariosDelete =  async (req,res) =>{
   const {id} = req.params
   const usuarioAutenticado = req.usuario

    const usuario= await Usuario.findByIdAndUpdate(id, {estado:false}) //si
quisiera borrarlo fisicamente usaría finsByIdAndDelete(id)

   res.json({
        usuarioAutenticado,
        usuario
    })

}
```

- Debo agregar otra validación, porque aunque el estado sea false se puede logear
- La hago en el middleware
- también debo validar que venga el usuario porque si no dará undefined

```
import jwt from 'jsonwebtoken'
import Usuario from '../models/usuario.js'

const validarJWT = async (req, res, next) =>{
    const token = req.header('x-token')

    if(!token)
     return res.status(401).json({msg: "No hay token en la petición"})

    try {
        //verifico el jwt y extraigo el uid
        const {uid} = jwt.verify(token, process.env.SECRET_KEY)

        //creo una propiedad nueva dentro de la request para pasar el uid
        const usuario = await Usuario.findById(uid)

        //verificar que venga usuario por el uid
        if(!usuario) return res.status(401).json({msg:"Token no válido - usuario
no existe en BD"})

        //verificar si el usuario tiene estado en true
        if(!usuario.estado) return res.status(401).json({msg:"Token no válido -
usuario con estado false"})

       req.usuario = usuario
        next()
```

```
    } catch (error) {
        console.log(error)
        return res.status(401).json({msg:"Token no válido"})
    }


}

export default validarJWT
```

- Si yo ahora borro el usuario de la DB del cual es el token, aunque pase la verificación, se firme, etc, no pasa la validación "usuario no existe en la db"
- En la siguiente lección se hará la validación mediante el rol (solo los ADMIN pueden borrar)

Verificar ROL de administrador

- Quiero crear un middleware que me fuerce a que para llamar el deleteUsuario tiene que ser admin
- Creo un nuevo middleware /middlewares/validar-roles.js
- Tengo el usuario en el request

```
const esAdminRole = (req, res, next) =>{

    if(!req.usuario) return res.status(500).json({msg: "Se quiere validar el role
sin obtener el token"})

    const {rol, nombre} = req.usuario

    if(rol !== 'ADMIN_ROLE') return res.status(401).json({msg: `${nombre} no es
administrador`})
    next()
}

export default esAdminRole
```

- Coloco el middleware en la ruta

```
router.delete('/:id', [
    validarJWT,
    esAdminRole,
    check('id', 'No es un id válido').isMongoId(),
    check('id').custom(userExistsById),
    validarCampos
],
userController.usuariosDelete)
```

Middleware - Tiene rol

- Voy a generar un middleware que me permita verificar varios roles de manera simultanea
- Quiero pasarle los roles válidos
- Cómo estoy ejecutando una función, el middleware tiene que ejecutar una función
- Le paso el req, res, next
- Al usar el rest en los parámetros, este me devuelve un arreglo con los parametros que le pase (los roles en este caso)

```
const tieneRole = (...roles)=>{

    return (req,res,next)=>{
        if(!req.usuario) return res.status(500).json({msg: "Se quiere validar el role sin obtener el token"})

        const {rol} = req.usuario
        if(!roles.includes(rol)) return res.status(401).json({msg: `El servicio requiere uno de estos roles ${roles}`})
        next()
    }

}

export default tieneRole
```

- user.routes.js

```
router.delete('/:id', [
    validarJWT,
   // esAdminRole,
   tieneRole('ADMIN_ROLE','VENTAS_ROLE','USER_ROLE'), //solo deja pasar si tiene uno de estos roles
    check('id', 'No es un id válido').isMongoId(),
    check('id').custom(userExistsById),
    validarCampos
],
userController.usuariosDelete)
```

Guia de uso

- Hacer el login con correo y passsword válidos
- Copiar el token que aperece en consola sin las comillas y añadirla a los headers (en Thunder Client) con el nombre de x-token
- Copiar el id de un usuario que esté activo y copiarlo en la url para borrar con el método DELETE