UpTASK

Primeros pasos

- Creo el servidor.
- Para ello creo una carpeta Backend y en el directorio escribo en la terminal npm init
- Instalo express (ya tengo instalado nodemon)
- Para habilitar imports en node añadir "type":"module" debajo de "version" en package.json
- El código del servidor inicialmente luce así:

```
import express from 'express'

const app = express()


app.listen(4000, ()=>{
    console.log("Servidor corriendo en el puerto 4000")
})
```

Creando base de datos

- Crear el cluster en mongoDB
- Copiar el enlace de "conexión con MongoCompass"
- Añadirlo a la conexión de Compass con el username y el password

Conectar la base de datos

- Ahora copio el string de conexión de la web de MongoDB para conectar con la aplicación
- Creo la carpeta en backend de config con el archivo db.js
- Instalo mongoose
- En db.js importo mongoose y creo una función async llamada conectarDB con mongoose.connect
    - Le incorporo un try y un catch
    - En el catch, añado un console.log y dentro de un template string imprimo el error.message
    - Termino el proceso con process.exit(1)
    - en el try añado el string de conexión con el name y el password correspondientes
    - Le añado los parametros dentro de un objeto (junto al string de conexión)
    - useNewUrlParser y useUnifiedTopology, ambos en true
    - Creo una constante fuera del try llamada url con untemplate string para imprimir la conexión
    - La pongo dentro e un console.log
    - Para verlo en consola hay que integrar este archivo (db.js) en el index, para ello exporto por default conectarDB

- Importo conectarDB en el index, con el sufijo .js ya que es un archivo local que yo creé
- LLamo a la función dentro del archivo index.js
- index.js

```
import express from 'express'
import conectarDB from './config/db.js'

const app = express()

conectarDB()

app.listen(4000, ()=>{
    console.log("Servidor corriendo en el puerto 4000")
})
```

- db.js

```
import mongoose from "mongoose"

const conectarDB= async()=>{
    try {

        const connection= await
mongoose.connect("mongodb+srv://isma:isma@cluster0.82so450.mongodb.net/?
retryWrites=true&w=majority",{
            useNewUrlParser: true,
            useUnifiedTopology: true
        })

        const url=`${connection.connection.host}: ${connection.connection.port}`
        console.log(`MongoDB conectado en: ${url}`)

    } catch (error) {
        console.log(`error: ${error.message}`)
        process.exit(1)
    }
}


export default conectarDB
```

Ocultar string de conexión

- Instalo en backend dotenv para configurar las variables de entorno

> npm i dotenv

- Lo importo en index.js y escribo dotenv.config()
- Esto va a buscar el archivo .env. Creo el archivo .env en la raíz del backend
- Escribo la variable MONGO_URI con el string de conexión sin comillas como variable de entorno
- Sustituyo el string de conexión en db.js por process.env.MONGO_URI
- Creo otra variable de entorno para el puerto
- index.js

```
import express from 'express'
import conectarDB from './config/db.js'
import dotenv from 'dotenv'

const app = express()

dotenv.config()

conectarDB()

const PORT = process.env.PORT || 4000

app.listen(PORT, ()=>{
    console.log(`Servidor corriendo en el puerto ${PORT}`)
})
```

- db.js

```
import mongoose from "mongoose"

const conectarDB= async()=>{
    try {

        const connection= await mongoose.connect(process.env.MONGO_URI,{
            useNewUrlParser: true,
            useUnifiedTopology: true
        })

        const url=`${connection.connection.host}: ${connection.connection.port}`
        console.log(`MongoDB conectado en: ${url}`)

    } catch (error) {
        console.log(`error: ${error.message}`)
        process.exit(1)
    }
}
```

```
export default conectarDB
```

En el archivo .env en la raíz

```
MONGO_URI=mongodb+srv://isma:isma@cluster0.82so450.mongodb.net/?
retryWrites=true&w=majority
```

Model View Controller

- Creo una nueva carpeta en backend/ llamada models
- Dentro un archivo llamado Usuario.js
    - Importo mongoose
    - Procedo a estructurar el Schema con mongoose.Schema
    - Nombre de tipo String, obligatorio y le quito los espacios con trim
    - Lo mismo con el password, y el mail
    - unique para que el mail sea único
    - Confirmado esta false por defecto porque recibirán un mail de confirmación para confirmar la cuenta
    - El timestamps para crear dos columnas más, una de creado y otra de actualizado
    - Creo el modelo con mongoose.model con el nombre del modelo y el Schema y lo hago disponible con el export default

```
import mongoose from 'mongoose'

const usuarioSchema = mongoose.Schema({
    nombre:{
        type: String,
        required: true,
        trim: true
    },
    password:{
        type: String,
        required: true,
        trim: true
    },
    email:{
        type: String,
        required: true,
        trim: true,
        unique: true
    },
    token:{
        type: String
    },
    confirmado:{
        type: Boolean,
        default: false
    }

},{
    timestamps: true
})

const Usuario = mongoose.model("Usuario", usuarioSchema)

export default Usuario
```

Routing y controladores

- El siguiente paso es crear el routing y los controladores para los diferentes endpoints y los verbos HTTP
- Se hará usando el app (de const app = express()), pues concentra toda la funcionalidad de express
- index.js:

```
app.get('/', (req,res)=>{
    res.send("Hola mundo!")
})
```

- Si coloco app.use , use responde a todos los verbos http
- Puedo enviar la respuesta tipo json

```
app.get('/', (req,res)=>{
    res.json({msg: "OK"})
})
```

- Para no ensuciar el index con tanto código se recomienda ir agrupándolo en rutas y controladores
- Creo la carpeta routes en backend con el archivo usuarioRoutes
- Todo lo relacionado con usuarios se irá agrupando en este endpoint
    - Importo express para usar el Router
- En usuarioRoutes:

```
import express from 'express'

const router = express.Router()

router.get('/', (req,res)=>{
    res.send("Desde API/USUARIOS")
})
router.post('/', (req,res)=>{
    res.send("Desde -POST API/USUARIOS")
})

export default router
```

- Ahora debo importarlo en el index.js
- Gracias al .use va a soportar en este endpoint todos los verbos http

```
import express from 'express'
import conectarDB from './config/db.js'
```

```
import dotenv from 'dotenv'
import usuarioRoutes from './routes/usuarioRoutes.js'

const app = express()

dotenv.config()

conectarDB()

//ROUTING

app.use("/api/usuarios", usuarioRoutes)

const PORT = process.env.PORT || 4000

app.listen(PORT, ()=>{
    console.log(`Servidor corriendo en el puerto ${PORT}`)
})
```

- Al hacerlo de esta forma, todos los request a este endpoint van a estar enun archivo aparte(usuarioRoutes.js)
- Para probar los endpoints se usará POSTMAN
- Puedo simular los endpoints y los tipos de verbos con POSTMAN
- Si lo que quiero es enrutar, puedo usar el primer parametro del router para hacerlo, por ejemplo a confirmar

```
router.get('/confirmar', (req,res)=>{
    res.json({msg: "Okey, ahi lo tienes"})
})
```

- Esto sería igual a /api/usuarios/confirmar

Introduciendo controladores

- Creo la carpeta controllers en Backend/ con el archivo usuarioController.js
- Va a comunicar el routing con los modelos

```
const usuarios = (req,res) =>{
    res.json("Desde API/USUARIOS")
}

export {
    usuarios
}
```

- Ahora en usuarioRoutes importo usuarios y lo añado al endpoint

```
import express from 'express'
import {usuarios} from '../controllers/usuarioController.js'

const router = express.Router()

router.get('/', usuarios)



export default router
```

- Creo un crearUsuario, de tipo POST. Lo exporto
- Lo importo en usuarioRoutes y lo añado
- usuarioController:

```
const usuarios = (req,res) =>{
    res.json({msg: "desde Api Usuarios"})
}

const crearUsuario = (req,res) =>{
    res.json({msg: "creando usuario"})
}



export {
    usuarios,
    crearUsuario
}
```

- usuarioRoutes:

```
import express from 'express'
import {usuarios,crearUsuario} from '../controllers/usuarioController.js'

const router = express.Router()

router.get("/", usuarios)
router.post("/", crearUsuario)



export default router
```

- Uso Postman para controlar que los endpoints devuelven lo que hay en el res.json

---

AHORA BORRAMOS TODO LO PERTENECIENTE A LAS RUTAS Y VERBOS PARA EMPEZAR CON ALGO REALISTA

Autenticación, registro y confirmación de usuarios

- Sigo el procedimiento anterior para crear registrar

```
const registrar = (req,res)=>{

}

export {
    registrar
}
```

- Lo exporto de usuarioController y lo importo en usuarioRoutes

```
import express from 'express'
import {registrar} from '../controllers/usuarioController.js'


const router = express.Router()


router.post("/", registrar) //crea un nuevo usuario



export default router
```

- Ahora toca trabajar el cuerpo de la función registrar
- Usaré los valores del Schema, nombre, email y password

- Para enviar datos desde POSTMAN simulando que vienen de un formulario pulsa body y en raw, cambiar a json (donde pone text)
- Si en el cuerpo de la función registrar pongo un console.log de req.body NO FUNCIONA, no recibe los datos que estoy enviando en formato json desde POSTMAN
- usuarioController.js:

```
const registrar = (req,res)=>{
    console.log(req.body)
    res.json({msg:"OK!"})
}
```

- En el index.js hay que habilitar que pueda procesar info en formato json

```
app.use(express.json())
```

- Ahora si aparece en consola el json que envié desde POSTMAN
- Entonces, ya puedo insertarlo en la base de datos
- Para insertarlo hay que importar el modelo en el controlador
    - Uso el try catch para insertar los datos en la DB dentro de la función registrar
    - Creo una nueva instancia de Usuario (lo importo) y le paso el req.body, el cuerpo de la petición POST de POSTMAN
    - Puedo colocarle el usuario en un console.log para ver si todo marcha bien
    - Si no hay una función para finalizar el proceso POSTMAN se queda dando vueltas

```
const registrar = (req,res)=>{

    try {
        const usuario = new Usuario(req.body)

        console.log(usuario)

    } catch (error) {
        console.log(error)
    }


    res.json({msg:"OK!"})
}
```

- Para almacenarlo (aún no se ha almacenado) usaré async await y usuario.save()

```
const registrar = async (req,res)=>{

    try {
        const usuario = new Usuario(req.body)

        const usuarioAlmacenado= await usuario.save()
        res.json({msg:"OK!"})
```

```
    } catch (error) {
        console.log(error)
    }


}
```

- Ahora , si voy a POSTMAN y envío el json con el nuevo usuario puedo mirar en MongoCompass como se ha almacenado.
- Pero hay cosas que faltan:
    - El password no esta hasheado
    - No generó el token
    - Si lo vuelvo a eviar se queja por ser el mismo mail pero no muestra un mensaje amigable

---

Previniendo usuarios duplicados

- Extraigo con desestructuración el email del req.body en el controller, dentro de la funcion registrar
- Creo una variable llamada existeUsuario.
    - Uso el await
    - Mongoose tiene muchos métodos para interactuar con la DB
    - findOne buscara hasta dar con un match
    - Hago el if, imprimo el error
    - Le paso el objeto de email
- usuarioController

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

const registrar = async (req,res)=>{

    const {email} = req.body
    const existeUsuario =  await Usuario.findOne({email})

    if(existeUsuario){
        const error = new Error('Usuario ya registrado');
        return res.status(400).json({msg:error.message})
    }

    try {
        const usuario = new Usuario(req.body)

        const usuarioAlmacenado= await usuario.save()
        res.json({msg:"Usuario almacenado!"})

    } catch (error) {
        console.log(error)
    }
```

```
}

export {
    registrar
}
```

Hashear los passwords

- Para hashear los passwords instalo bcrypt

> npm i bcrypt

- En Usuario.js, a usuarioSchema le añado la función pre y como primer parametro 'save', para que lo que sea que haga lo haga antes de guardar
- Como segundo parámetro le añado una función anónima con el parámetro next(de express)
    - Este codigo se va a ejecutar antes de salvar el registro en la DB. Se usa function en lugar de una arrow para poder usar el this
- Uso la función de mongoose isModified con el this.password, para que compruebe si el password se ha modificado previamente
    - Si ya se hubiera hasheado antes y se volviera a hashear no daría match
    - Como parámetro le paso el password y en el cuerpo next( ) que es para mandar pasar al siguiente middleware
- A continuación genero la encriptación
    - Necesita un salt. 10 rondas de hasheo como parámetro es el standard, da suficiente seguridad
    - Hago referencia al password con this.password y con la función de bcrypt.hash le paso el password cómo primer parámetro y el salt cómo segundo
    - Lo hago async await para que se tome su tiempo y bloquee el código hasta que tenga una respuesta

```
import mongoose from 'mongoose'
import bcrypt from 'bcrypt'

const usuarioSchema = mongoose.Schema({
    nombre:{
        type: String,
        required: true,
        trim: true
    },
    password:{
        type: String,
        required: true,
        trim: true
    },
    email:{
        type: String,
        required: true,
        trim: true,
        unique: true
    },
    token:{
        type: String
    },
    confirmado:{
        type: Boolean,
        default: false
```

```
    }

},{
    timestamps: true
})


usuarioSchema.pre('save', async function(next){
   if(!this.isModified('password')){
    next()
   }
    const salt = await bcrypt.genSalt(10)
    this.password = await bcrypt.hash(this.password, salt)
})

const Usuario = mongoose.model("Usuario", usuarioSchema)

export default Usuario
```

Generar ID para un token único

- Creo una carpeta llamada helpers en /Backend con un archivo llamado generarId
- Quiero generar un id de manera random.
- El 32 del toString se le conoce cómo el radix
- El substring es para quitarle los dos primeros caracteres
- Concatenando Math.random con toString se obtiene una cadena de caracteres mezclando números y letras
- Junto con Date.now generará un ID lo suficientemente complejo

```
const generarId =()=>{
    //variable random

    const random = Math.random().toString(32).substring(2)
    const fecha= Date.now().toString(32)

    return random+fecha


}

export default generarId
```

- Importo generarId en el controlador (usuarioController.js)
- En el try, antes de que guarde con el .save( ) pasamos el generarId( ) al usuario.token

```
import Usuario from '../models/Usuario.js'
import generarId from '../helpers/generarId.js'

const registrar = async (req,res)=>{

    const {email} = req.body
    const existeUsuario =  await Usuario.findOne({email})

    if(existeUsuario){
        const error = new Error('Usuario ya registrado');
        return res.status(400).json({msg:error.message})
    }

    try {
        const usuario = new Usuario(req.body)
        usuario.token= generarId()

        const usuarioAlmacenado= await usuario.save()
        res.json({msg:"Usuario almacenado!"})

    } catch (error) {
        console.log(error)
    }


}

export {
    registrar
}
```

Creando el endpoint de autenticación

- Creo un nuevo endpoint con una función aún por definir a la que llamaré autenticar

```
import express from 'express'
import {registrar} from '../controllers/usuarioController.js'
import Usuario from '../models/Usuario.js'


const router = express.Router()



router.post("/", registrar) //crea un nuevo usuario
router.post('/login', autenticar)
```

```
export default router
```

- Este 'login' lo suma al /api/usuarios que tengo en el index.js, quedando /api/usuarios/login como endpoint
- Defino la función autenticar en el controlador (UsuarioController.js)
- Creo el endpoint POST http://localhost:4000/api/usuarios/login en POSTMAN y lo guardo como autenticar usuarios
- En environments (a la izquierda) aprieto + para añadir un nuevo ambiente (me permitirá almacenar variables)
- Lo llamo upTask, ahora a la derecha, cuando voy a colecciones, puedo ver No Environment. Ahí selecciono upTask
- Para definir una variable, clicar en el ojo de arriba a la derecha y en Edit
- Otra opción para guardar una variable es seleccionar la url que quieres guardar y aparece automáticamente set as variable

---

Autenticar

- Primero: comprobar si el usuario existe
- Segundo: comprobar si el usuario esta confirmado
- Tercero: comprobar el password
- Para autenticar el usuario voy a enviar en formato JSON con POSTMAN el email y el password
- En el usuarioController, en la función de autenticar, extraigo con destructuring el email y el password
- Paso a hacer la búsqueda por mail con el método de mongoose findOne
- Añado el posible error en caso de que el usuario no exista

```
const autenticar =async (req, res)=>{
    const {email, password } = req.body
    const usuario = await Usuario.findOne({email})

        if(!usuario){
        const error = new Error("El usuario no existe")
        return res.status(404).json({msg: error.message})
    }
```

```
}
```

- Ahora hay que comprobar si el usuario está confirmado. Es un código muy parecido al anterior. Como es un objeto, uso la notación de punto

```
const autenticar =async (req, res)=>{
    const {email, password } = req.body
    const usuario = await Usuario.findOne({email})
    if(!usuario){
        const error = new Error("El usuario no existe")
        return res.status(404).json({msg: error.message})
    }
    if(!usuario.confirmado){
        const error = new Error("Tu cuenta no ha sido confirmada")
        return res.status(404).json({msg: error.message})
    }
}
```

Comprobando el password

- Crear una función que compruebe el pàssword en Usuario
- de nuevo usaré function porque usaré el this
- En el controlador, dentro de autenticar estoy extrayendo el password, entonces llamaré la función ahi
- Para comparar los passwords tengo bcrypt.compare, y le paso el dato y el password al modelo Usuario.js

```
import mongoose from 'mongoose'
import bcrypt from 'bcrypt'

const usuarioSchema = mongoose.Schema({
    nombre:{
        type: String,
        required: true,
        trim: true
    },
    password:{
        type: String,
        required: true,
        trim: true
    },
    email:{
        type: String,
        required: true,
        trim: true,
```

```
        unique: true
    },
    token:{
        type: String
    },
    confirmado:{
        type: Boolean,
        default: false
    }

},{
    timestamps: true
})


usuarioSchema.pre('save', async function(next){
   if(!this.isModified('password')){
    next()
   }
    const salt = await bcrypt.genSalt(10) //10 rounds de hasheo son suficientes
    this.password = await bcrypt.hash(this.password, salt)
})

usuarioSchema.methods.comprobarPassword = async function(passwordFormulario){
    return await bcrypt.compare(passwordFormulario, this.password)
}


const Usuario = mongoose.model("Usuario", usuarioSchema)

export default Usuario
```

- En el controlador voy a usar el await, para esperar que se ejecute este método que devuelve true o false
- Cómo he creado el metodo, puedo usarlo en la instancia del usuario.
- Cómo el usuario si llega hasta aquí ya está confirmado, entonces tengo acceso a los datos del this.password
- Le paso el password en el controller

```
const autenticar =async (req, res)=>{

   const {email, password } = req.body

    const usuario = await Usuario.findOne({email})

    if(!usuario){
        const error = new Error("El usuario no existe")
        return res.status(404).json({msg: error.message})
    }
```

```
     if(!usuario.confirmado){
         const error = new Error("Tu cuenta no ha sido confirmada")
         return res.status(404).json({msg: error.message})
     }

    if(await usuario.comprobarPassword(password)){
         console.log("Es correcto")

    }else{
         const error = new Error("El password es incorrecto")
         return res.status(404).json({msg: error.message})
     }


}
```

NOTA: debo cambiar la cuenta a confirmada en COMPASS para que el endpoint a login en POSTMAN funcione

- Para obtener una mejor respuesta en postman cuando el password es correcto, la formateo
- El id es con un guión bajo porque así lo maneja Mongo

```
const autenticar =async (req, res)=>{

   const {email, password } = req.body

   const usuario = await Usuario.findOne({email})

   if(!usuario){
         const error = new Error("El usuario no existe")
         return res.status(404).json({msg: error.message})
     }

  if(!usuario.confirmado){
         const error = new Error("Tu cuenta no ha sido confirmada")
         return res.status(404).json({msg: error.message})
     }

   if(await usuario.comprobarPassword(password)){
         res.json({
             _id: usuario._id,
             nombre: usuario.nombre,
             email: usuario.email
         })

     }else{
         const error = new Error("El password es incorrecto")
         return res.status(404).json({msg: error.message})
     }
```

```
}
```

Cómo generar un JSON WEB TOKEN

- Instalar jsonwebtoken

```
npm i jsonwebtoken
```

- En la carpeta helpers creo el archivo generarJWT.js
    - Importo jwt
    - .sign( ) es un método que permite generar un JWT, de forma sincrona va a firmar el payload que le estas enviando al JWT
    - Como primer parámetro tiene un objeto (lo que va a colocar en el JWT), como segundo la llave privada se tiene que almacenar en las variables de entorno
    - Se recomienda poner una cadena compleja. Lo almaceno en .env con JWT_SECRET=??????
    - Como tercer parámetro toma un objeto con opciones
    - expiresIn es cuanto tiempo estará vigente el token. Le pongo 30 días
- Importo generarJWT en el controller
- Lo añado a la respuesta que estoy generando
- .env

```
MONGO_URI=mongodb+srv://isma:</password>@cluster0.82so450.mongodb.net/?
retryWrites=true&w=majority

JWT_SECRET= XXXXXXXXXX
```

- generarJWT

```
import jwt from 'jsonwebtoken'

const generarJWT =()=>{
    return jwt.sign({nombre: "Maik"}, process.env.JWT_SECRET,{
        expiresIn:'30d'
    })

}

export default generarJWT
```

- Uso POSTMAN para visualizar en la respuesta el token. Puedo copiarlo y pegarlo en la web JWT para ver la info en el JWT
- Lo que me interesa es incluir el id.
- Voy al usuario controller y lo meto como parámetro en generarJWT
- usuarioController.js

```
import Usuario from '../models/Usuario.js'
import generarId from '../helpers/generarId.js'
import generarJWT from '../helpers/generarJWT.js'

const registrar = async (req,res)=>{

    const {email} = req.body
    const existeUsuario =  await Usuario.findOne({email})

    if(existeUsuario){
        const error = new Error('Usuario ya registrado');
        return res.status(400).json({msg:error.message})
    }

    try {
        const usuario = new Usuario(req.body)
        usuario.token= generarId()

        const usuarioAlmacenado= await usuario.save()
        res.json({msg:"Usuario almacenado!"})

    } catch (error) {
        console.log(error)
    }


}

const autenticar =async (req, res)=>{

    const {email, password } = req.body
    const usuario = await Usuario.findOne({email})

   if(!usuario){
        const error = new Error("El usuario no existe")
        return res.status(404).json({msg: error.message})
    }

   if(!usuario.confirmado){
        const error = new Error("Tu cuenta no ha sido confirmada")
        return res.status(404).json({msg: error.message})
    }

    if(await usuario.comprobarPassword(password)){
        res.json({
            _id: usuario._id,
```

```
            nombre: usuario.nombre,
            email: usuario.email,
            token: generarJWT(usuario._id)
        })

    }else{
        const error = new Error("El password es incorrecto")
        return res.status(404).json({msg: error.message})
     }


}


export {
    registrar,
    autenticar
}
```

- Debo pasarle el id como parámetro a la función e incluirla en el objeto
- generarJWT.js

```
import jwt from 'jsonwebtoken'

const generarJWT =(id)=>{
    return jwt.sign({id}, process.env.JWT_SECRET,{
        expiresIn:'30d'
    })

}

export default generarJWT
```

Creando un endpoint para confirmar cuentas

- Incluyo el endpoint "/confirmar" con el método get en usuarioRoutes.js. Cómo segundo parámetro la función (que todavía no existe) confirmar
    - Notar que es routing dinámico, ya que el token se expresa cómo comodín
- La inicializo con async en usuarioController, la exporto y l aimporto en usuarioRoutes

```
const confirmar = async(req, res)=>{
    console.log("Routing dinámico")
}


export {
    registrar,
    autenticar,
    confirmar
}
```

- usuarioRoutes.js

```
const router = express.Router()



router.post("/", registrar) //crea un nuevo usuario
router.post('/login', autenticar)
router.get('/confirmar/:token', confirmar)


export default router
```

- Si ahora en POSTMAN pongo el endpoint confirmar/20 me devuelve en consola "Routing dinámico"
- Para extraer los datos de la url uso req.params.token (gracias al routing dinámico)

```
const confirmar = async(req, res)=>{
    console.log(req.params.token)
}
```

- Extraigo con desestructuración el token
- De nuevo uso el método findOne y le paso el token

```
const confirmar = async(req, res)=>{
    const {token} = req.params

    const usuarioConfirmar = await Usuario.findOne({token})
    console.log(usuarioConfirmar)
}
```

- Si ahora pongo el token que copio de la DB de uno de los usuarios me imprime en consola el usuario con su id, token y todo
- Manejo el error en el controller.

```
const confirmar = async(req, res)=>{
    const {token} = req.params

    const usuarioConfirmar = await Usuario.findOne({token})

    if(!usuarioConfirmar){
        const error = new Error("Token no válido")
        return res.status(404).json({msg: error.message})
    }
}
```

- Uso un try catch una vez tengo confirmado el usuario
    - confirmado pasa a ser true
    - Cómo va a ser un token de un solo uso lo elimino con un string vacío
- usuarioController.js

```
const confirmar = async(req, res)=>{
    const {token} = req.params

    const usuarioConfirmar = await Usuario.findOne({token})

    if(!usuarioConfirmar){
        const error = new Error("Token no válido")
        return res.status(404).json({msg: error.message})
    }
    try {
        usuarioConfirmar.confirmado = true
        usuarioConfirmar.token= ""
        await usuarioConfirmar.save();
        res.json({msg: "Usuario confirmado correctamente"})


    } catch (error) {
```

```
        console.log(error)
    }
}
```

Resetear Passwords

- No se puede revertir la cadena hasheada. Para resetear el password habrá que hacer el proceso con un nuevo token
- Añado el endpoint de 'olvide-password'. Es de tipo POST porque el usuario va a enviar su email, y se va a comprobar que ese mail exista y esté confirmado
- usuarioRoutes.js

```
const router = express.Router()



router.post("/", registrar) //crea un nuevo usuario
router.post('/login', autenticar)
router.get('/confirmar/:token', confirmar)
router.post('/olvide-password', olvidePassword )


export default router
```

- Creo la función olvidePassword en usuarioController
- Copio y pego el código (escrito previamente en autenticar) para confirmar que el usuario existe
- Añado el endpoint a POSTMAN
- En caso de que el usuario exista lo manejo con un try catch
- Genero un nuevo token con generarId
- usuarioController.js

```
const olvidePassword = async (req,res)=>{
    const {email} = req.body

   const usuario = await Usuario.findOne({email})

   if(!usuario){
        const error = new Error("El usuario no existe")
        return res.status(404).json({msg: error.message})
    }
    try {
        usuario.token= generarId()
        await usuario.save()
        res.json({msg: "Hemos enviado un email con las instrucciones"})
```

```
    } catch (error) {
        console.log(error)
    }

}
```

Validando el token

- Creo una nueva ruta para validar el token y resetear el password
- Va a comprobar que el token sea válido y el usuario exista. Es lo único que va a hacer. Lo próximo será almacenar un nuevo password
- Extraigo el token de la url con req.params
- Uso de nuevo findOne para buscar si existe en la base de datos
- Si el usuario existe posteriormente se le enviará un formulario para resetear el password
- usuarioController.js

```
const comprobarToken = async(req, res)=>{
    const {token} = req.params

   const tokenValido = await Usuario.findOne({token})

    if(tokenValido){
        res.json({msg:"Token válido, el usuario existe"})
    }else{
        const error = new Error("Token no válido")

        return res.status(404).json({msg: error.message})
    }
}
```

Almacenando el nuevo password

- Creo el nuevo endpoint de tipo POST con la función nuevoPassword
- Como apuntan al mismo sitio se puede usar router.route()
- usuarioRoutes.js

```
const router = express.Router()



router.post("/", registrar) //crea un nuevo usuario
router.post('/login', autenticar)
router.get('/confirmar/:token', confirmar)
```

```
router.post('/olvide-password', olvidePassword )
router.route('/olvide-password/:token').get(comprobarToken).post(nuevoPassword)

export default router
```

- Extraigo el token de la url con req.params y el password del body con req.body
- usuarioController.js

```
const nuevoPassword= async (req,res)=>{
    const {token} = req.params
    const {password}= req.body

    console.log(token)
    console.log(password)
}
```

- Ahora si voy a POSTMAN con el endpoint con un token válido me aparece en consola el token y el password que envio desde el body en POSTMAN
- Copio y pego el mismo código de tokenValido pero lo cambio por la variable usuario

```
const nuevoPassword= async (req,res)=>{
    const {token} = req.params
    const {password}= req.body

    const usuario = await Usuario.findOne({token})

    if(usuario){
        console.log(usuario)
    }else{
        const error = new Error("Token no válido")
        return res.status(404).json({msg: error.message})
    }

}
```

- Ahora si le doy al send en POSTMAN me aparece el usuario en consola (por el console.log). Está todo bien
- Entonces, reescribo el password con el req.body
- Como el password no está modificado, pasa a lo siguiente (en UsuarioSchema, el pre('save')) con el next( ), que es generar un nuevo salt para hashear el password
- Hay que eliminar el token

```

```

```
const nuevoPassword= async (req,res)=>{
    const {token} = req.params
    const {password}= req.body

    const usuario = await Usuario.findOne({token})

    if(usuario){
        usuario.password = req.body
        usuario.token=""
        await usuario.save()
        res.json({msg:"Password almacenado correctamente"})
    }else{
        const error = new Error("Token no válido")
        return res.status(404).json({msg: error.message})

    }

}
```

- Es mejor ponerlo dentro de un try catch

```
const nuevoPassword= async (req,res)=>{
    const {token} = req.params
    const {password}= req.body

    const usuario = await Usuario.findOne({token})

    if(usuario){
        usuario.password = password
        usuario.token=""

    try {
        await usuario.save()
        res.json({msg:"Password almacenado correctamente"})

    } catch (error) {
       console.log(error)
    }
    }else{
        const error = new Error("Token no válido")
        return res.status(404).json({msg: error.message})

    }

}
```

Comenzando un custom middleware

- Hay zonas que son públicas pero hay zonas que requieren estar autenticado
- Creo una carpeta llamada middleware en /backend
- Un middleware es cada linea del archivo index.js, por ejemplo. Va a una, pasa a la siguiente, y a la siguiente
- Creo el archivo checkAuth.js y dentro del archivo una función que llamaré checkAuth
- Lo importo en usuarioRoutes
- Creo el endpoint /perfil con get para pasarle el JWT y retorne el perfil del usuario
- Primero entra al endpoint, después ejecuta el middleware y después la otra función (perfil)
- Tanto en la función perfil(en usuarioController) como en la de checkAuth he puesto un console.log "desde X"
- usuarioRoutes

```
const router = express.Router()

router.post("/", registrar) //crea un nuevo usuario
router.post('/login', autenticar)
router.get('/confirmar/:token', confirmar)
router.post('/olvide-password', olvidePassword )
router.route('/olvide-password/:token').get(comprobarToken).post(nuevoPassword)
router.get('/perfil', checkAuth, perfil)

export default router
```

- Si ahora hago una petición get a /perfil desde POSTMAN me devuelve "desde checkAuth" pero no aparece lo del controlador que es "desde perfil"
- checkAuth toma req, res pero también next, que permite pasar al siguiente middleware
- checkAuth

```
const checkAuth = (req, res, next)=>{
    console.log("desde checkAuth.js")
    next()
}

export default checkAuth
```

- Ahora, si hago la petición GET a /perfil desde POSTMAN me imprime los dos console.log, el de checkAuth y el de perfil
- En este custom middleware voy a estar revisando que el usuario esté autenticado y que el JSONWebToken sea válido

Custom middleware

- checkAuth.js

```
const checkAuth = (req, res, next)=>{
    console.log(req.headers.authorization)
    next()
}

export default checkAuth
```

- Este console.log da undefined. Usualmente es en los headers dónde se va a enviar el JWT. Los headers es lo que se envia primero. Al enviar el JWT ahi se puede confrimar que todo esté correctamente y se le da acceso al usuario
- EN POSTMAN hay una pestañita que dice autorization. Añado Bearer Token y en la pestañita añado el token que consigo en POSTMAN haciendo un request a /login
- Ahora puedo ver en consola que tengo Bearer a la izquierda y el token a la derecha. Me interesa solo la derecha (el token)
- Para ello uso split( ) para dividir por espacios en un arreglo y le indico que el token esta en la posición 1

```
const checkAuth = (req, res, next)=>{
    let token;

    if(req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')){

       try {
            token = req.headers.authorization.split(' ')[1]

            console.log(token)

       } catch (error) {
            console.log(error)

        }
    }

    next()
}

export default checkAuth
```

- Importo la librería de jwt que me permite verificar(descifrar) el JSONWebToken
- Como segundo parametro le paso la misma variable de entorno que usé para generar el salt
- checkAuth.js

```
import jwt from 'jsonwebtoken'

const checkAuth = (req, res, next)=>{
    let token;

   if(req.headers.authorization && req.headers.authorization.startsWith('Bearer'))
{

       try {
            token = req.headers.authorization.split(' ')[1]

            const decoded= jwt.verify(token, process.env.JWT_SECRET)
            console.log(decoded)

        } catch (error) {
              return res.status(404).json({msg: "Hubo un error"})

        }
    }

    next()
}

export default checkAuth
```

- Transformo en async la función para usar el await
- El JWT tiene el id del usuario,
- Importo el Usuario de models.
- Agrego una nueva variable en el req. y le paso al método findById decoded.id, porque es lo que está extrayendo del token en decoded

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

const checkAuth = async (req, res, next)=>{
    let token;
    if(req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')){
        try {

            token = req.headers.authorization.split(' ')[1]

            console.log(token)

           const decoded= jwt.verify(token, process.env.JWT_SECRET)

            req.usuario= await Usuario.findById(decoded.id)

            console.log(req.usuario)
```

```
        } catch (error) {

            return res.status(404).json({msg: "Hubo un error"})

        }
    }

    next()
}

export default checkAuth
```

- Si hago la petición GET con el TOKEN de autenticación el console.log de req.usuario me devuelve algo asi

```
{
  _id: new ObjectId("6303ab39029bae8dff989ff5"),
  nombre: 'Pepito',
  password: '$2b$10$GHvRXh9Mf5CczsSHFPe3lOIM7sy.z3tXdcDAJWrZtEnz78EOG.0CG',
  email: 'correo@correo.com',
  confirmado: true,
  token: 'os5uj9uohbo1gb5gg83f',
  createdAt: 2022-08-22T16:13:45.953Z,
  updatedAt: 2022-08-23T14:08:03.706Z,
  __v: 0
}
```

- Para eliminar el password y otros campos uso select en req.usuario
- Importante el next( ) para pasar al siguiente middleware

```
const checkAuth = async (req, res, next)=>{
    let token;

    if(req.headers.authorization &&
req.headers.authorization.startsWith('Bearer')){

        try {
            token = req.headers.authorization.split(' ')[1]

            console.log(token)

            const decoded= jwt.verify(token, process.env.JWT_SECRET)

           req.usuario= await Usuario.findById(decoded.id).select("-password -
confirmado -token -createdAt -updatedAt  -__v")
```

```
            return next();

        } catch (error) {

            return res.status(404).json({msg: "Hubo un error"})

         }
    }
        if(!token){
         const error = new Error("Token no válido")
        return res.status(404).json({msg: error.message})
    }

    next()
}
```

- Le coloco return next( ) porque una vez que se verificó el JWT y se le asignó al req paso al siguiente middleware
- Si no hay un token pasaré un nuevo error
- Este middleware verifica el token y da paso a perfil si todo esta bien
- En el controlador configuro perfil para que me devuelva el usuario en pantalla en fromato json

```
const perfil =async (req,res)=>{
    const {usuario} = req

   res.json(usuario)
}
```

Creando el modelo para proyectos

- Lo primero importo mongoose
- Añado el Schema
- En creador establezco una relación con el id del usuario
- Lo que hay en ref es de donde va a obtener esta referencia, en este caso es de Usuario= mongoose.model(**"Usuario"**, usuarioSchema)
- Colaboradores es un arreglo de usuarios
- Al final, el timestamps en true crea las dos filas de cuando fue creado y actualizado

```
import mongoose from 'mongoose'

const proyectoSchema = mongoose.Schema({
    nombre:{
        type: String,
        trim: true,
        required: true
    },
    descripcion:{
        type: String,
        trim: true,
        required: true
    },
    fechaEntrega:{
        type: Date,
        default: Date.now()
    },
    cliente:{
        type: String,
        trim: true,
        required: true
    },
    creador:{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Usuario'
    },
    colaboradores:[
        {
            type: mongoose.Schema.Types.ObjectId,
            ref: 'Usuario'
        }
    ]
}, {
    timestamps: true
})

const Proyecto= mongoose.model('Proyecto', proyectoSchema)

export default Proyecto
```

- Creo proyectoController.js y creo una función con req, res que es la que traerá los proyectos de la persona autenticada
- Creo el cascaron de las funciones que usaré. Exporto todo

```
const obtenerProyectos= async(req,res)=>{

}

const nuevoProyecto= async (req,res)=>{
    console.log(req.body)
}

const obtenerProyecto=async(req,res)=>{

}

const editarProyecto = async(req, res)=>{

}

const eliminarproyecto = async(req,res)=>{

}

const agregarColaborador = (req,res)=>{

}

const eliminarColaborador = (req,res)=>{

}

const obtenerTareas = async(req,res)=>{

}


export{
    obtenerProyectos,
    nuevoProyecto,
    obtenerProyecto,
    editarProyecto,
    eliminarproyecto,
    agregarColaborador,
    eliminarColaborador,
    obtenerTareas
}
```

- En /routes creo proyectoRoutes.js y las importo todas
- Importo también checkAuth y express. Declaro el router

```
import {obtenerProyectos,
nuevoProyecto,
obtenerProyecto,
editarProyecto,
eliminarProyecto,
agregarColaborador,
eliminarColaborador,
obtenerTareas} from '../controllers/proyectoController.js'
import checkAuth from '../middleware/checkAuth.js'
import express from 'express'

const router = express.Router()




export default router;
```

- Añado el endpoint de proyectos en el index.js que está en la raiz

```
import express from 'express'
import conectarDB from './config/db.js'
import dotenv from 'dotenv'
import usuarioRoutes from './routes/usuarioRoutes.js'

const app = express()

dotenv.config()
app.use(express.json())

conectarDB()

//ROUTING

app.use("/api/usuarios", usuarioRoutes)
app.use("/api/proyectos", usuarioRoutes) //aquí irá proyectoRoutes




const PORT = process.env.PORT || 4000

app.listen(PORT, ()=>{
    console.log(`Servidor corriendo en el puerto ${PORT}`)
})
```

- Como inicio, en proyectoRoutes.js hago una petición GET de obtenerProyectos. Le añado el middleWare de autenticación
- Como comparten la raíz con la petición post los incluyo en router.route

```
const router = express.Router()


router.route('/')
    .get(checkAuth, obtenerProyectos)
    .post(checkAuth, nuevoProyecto)

export default router;
```

- ObtenerProyecto, editar y eliminar requieren el id.

```
router.route('/:id')
    .get(checkAuth,obtenerProyecto)
    .put(checkAuth, editarProyecto)
    .delete(checkAuth, eliminarProyecto)
```

- Añado los endpoints que faltan a proyectoRoutes.js

```
import {obtenerProyectos,
nuevoProyecto,
obtenerProyecto,
editarProyecto,
eliminarProyecto,
agregarColaborador,
eliminarColaborador,
obtenerTareas} from '../controllers/proyectoController.js'
import checkAuth from '../middleware/checkAuth.js'
import express from 'express'

const router = express.Router()


router.route('/')
    .get(checkAuth, obtenerProyectos)
    .post(checkAuth, nuevoProyecto)

router.route('/:id')
    .get(checkAuth,obtenerProyecto)
    .put(checkAuth, editarProyecto)
```

```
    .delete(checkAuth, eliminarProyecto)

router.get('/tareas/:id', checkAuth, obtenerTareas)
router.post('/agregar-colaborador/:id', agregarColaborador)
router.delete('/eliminar-colaborador/:id', checkAuth, eliminarColaborador)


export default router;
```

Creando Proyectos

- Falta importar proyectoRoutes en el index.js y añadirlo al endpoint
- Como es un export default lo puedo llamar como quiera

```
import express from 'express'
import conectarDB from './config/db.js'
import dotenv from 'dotenv'
import usuarioRoutes from './routes/usuarioRoutes.js'
import proyectoRoutes from './routes/proyectoRoutes.js'

const app = express()

dotenv.config()
app.use(express.json())

conectarDB()

//ROUTING

app.use("/api/usuarios", usuarioRoutes)
app.use("/api/proyectos", proyectoRoutes)




const PORT = process.env.PORT || 4000

app.listen(PORT, ()=>{
    console.log(`Servidor corriendo en el puerto ${PORT}`)
})
```

- Le coloco un console.log(req.body) a nuevoProyecto en proyectoController.js
- Abro POSTMAN y apunto a proyectos con una petición POST. Abro body, raw, json y escribo el body
- Relleno los campos según el schema. El de la fecha no porque ya tiene Date.now(). Creador y colaboradores no los voy a mandar

- Esto es lo que escribo en el body/raw/json en POSTMAN

```
{
    "nombre": "tienda virtual",
    "descripcion": "Tienda virtual para un cliente",
    "cliente": "códigonuevo"
}
```

- El usuario debe de estar autenticado para la petición, asi que debo abrir auth, bearer Token y colocarle el token
- Ahora puedo ver gracias al console.log de req.body en NuevoProyecto(), el json que he escrito en POSTMAN en la consola
- A NuevoProyecto le puedo añadir un console.log(req.usuario) porque tengo un usuario autenticado con el checkAuth
- Importo Proyecto de /models
- Creo una nueva instancia de proyecto con el req.body

```
const nuevoProyecto= async (req,res)=>{

    const proyecto= new Proyecto(req.body)

    proyecto.creador= req.usuario._id
}
```

- Una vez almacenado voy a regresar el proyecto almacenado. Uso un try y un catch

```
const nuevoProyecto= async (req,res)=>{

   const proyecto= new Proyecto(req.body)

   proyecto.creador= req.usuario._id

    try {

        const proyectoAlmacenado = await proyecto.save()

        res.json(proyectoAlmacenado)

    } catch (error) {

       console.log(error)
    }
}
```

- Esto crea el proyecto en la DB. El resto de info que no agregué en el body de POSTMAN lo agrega express por su cuenta
- Genero un nuevo token de otro usuario con autenticar usuario y lo uso como auth en POSTMAN para publicar un nuevo proyecto que pongo en formato json en el body/raw/json de POSTMAN. Añado nombre, descripción y cliente

Obtener los proyectos de los usuarios autenticados

- En POSTMAN hago una petición GET a /api/proyectos, el endpoint de obtenerProyectos en proyectosRouter. Debo hacerlo con la autorización TOKEN
- Escribo en la función del controller obtenerProyectos.js

```
const obtenerProyectos= async(req,res)=>{

   const proyectos = await Proyecto.find()

   res.json(proyectos)
}
```

- Me trae todos los proyectos. Tiene que traer sólo los del usuario que ha realizado la petición (autenticado)
- En todos los endpoints que tienen el checkAuth como middleware dispongo de req.usuario para identificar que usuario esta autenticado
- Puedo hacer una consulta con mongoose algo más avanzada

```
const obtenerProyectos= async(req,res)=>{

  const proyectos = await Proyecto.find().where('creador').equals(req.usuario)

  res.json(proyectos)
}
```

Obtener y validar un proyecto por su ID

- Con el id de proyecto (que puedo consultar en Compass) es con lo que voy a hacer la consulta desde el endpoint
- Cómo tiene router dinámico('/:id'), accedo con req.params haciendo desestructuración

```
const obtenerProyecto= async(req,res)=>{

    const {id} = req.params
```

```
    console.log(id)
}
```

- Le añado un console.log para comprobar que extraigo el id. Hago la consulta GET en POSTMAN con el id de una tarea en el endpoint
- Uso el método findById y manejo el error

```
const obtenerProyecto=async(req,res)=>{
    const {id} = req.params

    const proyecto = await Proyecto.findById(id)

    if(!proyecto){

        return res.status(404).json({msg:"No encontrado"})
    }

    res.json(proyecto)
}
```

NOTA: el código de if(!proyecto) da error. Se corrige más adelante en la parte del FRONTEND

- Si comparo proyecto.creador === req.usuario._id me da false. Incluso si uso solo ==. Debo transformar los dos a string
- Hago una comparación para confirmar si el creador coincide con el id del usuario

```
const obtenerProyecto = async(req,res)=>{
    const {id} = req.params

    const proyecto = await Proyecto.findById(id)

    if(!proyecto){

         const error= new Error("No encontrado")

       return res.status(404).json({msg: error.message})
    }//DA ERROR

   if(proyecto.creador.toString() !== req.usuario._id.toString()){

        const error = new Error("No tienes los permisos. Acción no válida")

        return res.status(401).json({msg: error.message})
    }
```

```
    res.json(proyecto)
}
```

Editar un proyecto

- Ahora apunto con POSTMAN con un PUT al proyectos/id_del_proyecto
- En la función editarProyecto tengo las mismas medidas de seguridad que en obtenerProyecto. Copio y pego
- Voy al body/raw/json para escribir el cuerpo de la actualización en POSTMAN

```
const editarProyecto = async(req, res)=>{
    const {id} = req.params
    const proyecto = await Proyecto.findById(id)

    if(!proyecto){
        const error= new Error("No encontrado")
        return res.status(404).json({msg: error.message})
    }
    if(proyecto.creador.toString() !== req.usuario._id.toString()){
        const error = new Error("No tienes los permisos. Acción no válida")
        return res.status(401).json({msg: error.message})
    }

}
```

- Si llego hasta proyecto es que pasó todas las validaciones
- Compongo para editar

```
const editarProyecto = async(req, res)=>{
    const {id} = req.params
    const proyecto = await Proyecto.findById(id)

    if(!proyecto){
        const error= new Error("No encontrado")
        return res.status(404).json({msg: error.message})
    }
    if(proyecto.creador.toString() !== req.usuario._id.toString()){
        const error = new Error("No tienes los permisos. Acción no válida")
        return res.status(401).json({msg: error.message})
    }

    proyecto.nombre      = req.body.nombre || proyecto.nombre
    proyecto.descripcion = req.body.descripcion || proyecto.descripcion
    proyecto.fechaEntrega= req.body.fechaEntrega || proyecto.fechaEntrega
    proyecto.cliente     = req.body.cliente || proyecto.cliente
```

```
}
```

- Después uso un try catch para el producto almacenado

```
const editarProyecto = async(req, res)=>{

   const {id} = req.params

   const proyecto = await Proyecto.findById(id)

    if(!proyecto){

       const error= new Error("No encontrado")

        return res.status(404).json({msg: error.message})
    }
    if(proyecto.creador.toString() !== req.usuario._id.toString()){

       const error = new Error("No tienes los permisos. Acción no válida")

       return res.status(401).json({msg: error.message})
    }

    proyecto.nombre = req.body.nombre || proyecto.nombre
    proyecto.descripcion = req.body.descripcion || proyecto.descripcion
    proyecto.fechaEntrega= req.body.fechaEntrega || proyecto.fechaEntrega
    proyecto.cliente = req.body.cliente || proyecto.cliente

    try {
        const proyectoAlmacenado = await proyecto.save()

        res.json(proyectoAlmacenado)

   } catch (error) {

    }
}
```

Eliminar un proyecto

- Las consideraciones son las mismas. Leer el parámetro, identificar el proyecto por id y que el creador coincida con el usuario que loquiere borrar
- Uso el metodo deleteOne( ) de mongoose en un try catch

```
const eliminarProyecto = async(req,res)=>{
    const {id} = req.params

   const proyecto = await Proyecto.findById(id)

    if(!proyecto){

       const error= new Error("No encontrado")

        return res.status(404).json({msg: error.message})
    }
    if(proyecto.creador.toString() !== req.usuario._id.toString()){

        const error = new Error("No tienes los permisos. Acción no válida")

        return res.status(401).json({msg: error.message})
    }

    try {
        await proyecto.deleteOne()

       res.json({msg: "Proyecto eliminado"})

    } catch (error) {

    }

}
```

Crear Modelo Tareas

- Cada tarea va a estar asociada a un proyecto. Por eso los relaciono y le paso de referencia el modelo de Proyecto
- Finalmente declaro el modelo y lo exporto por default

```
import mongoose from 'mongoose'



const tareaSchema = mongoose.Schema({

   nombre:{
        type: String,
        trim: true,
        required: true
    },
    descripcion:{
        type: String,
        trim: true,
        required: true
    },
    estado:{
        type: Boolean,
        default: false
    },
    fechaEntrega:{
        type: Date,
        required: true,
        default: Date.now()
    },
    prioridad:{
        type: String,
        required: true,
        enum:["Baja", "Media", "Alta"]
    },
    proyecto:{
        type:  mongoose.Schema.Types.ObjectId,
        ref: "Proyecto"
    }
},{
    timestamps: true
})

const  Tarea = mongoose.model("Tarea", tareaSchema)

export default Tarea
```

- Creo tareaController.js y tareaRoutes.js

- Declaro las funciones en el controlador

```
const agregarTarea = async (req,res)=>{

}

const obtenerTarea = async (req,res)=>{

}

const actualizarTarea = async (req,res)=>{

}
const eliminarTarea = async (req,res)=>{

}

const cambiarEstado = async (req,res)=>{

}

export{
    agregarTarea,
    obtenerTarea,
    actualizarTarea,
    eliminarTarea,
    cambiarEstado
}
```

- Las importo en tareaRoutes.js. (Recuerda poner la extensión .js!!!)
    - Importo express y checkAuth para la autenticación

```
import{
    agregarTarea,
    obtenerTarea,
    actualizarTarea,
    eliminarTarea,
    cambiarEstado
} from '../controllers/tareaController.js'

import express from 'express'
import checkAuth from '../middleware/checkAuth.js'

const router = express.Router()

router.post('/', checkAuth, agregarTarea)

router.route('/:id')
.get(checkAuth, obtenerTarea)
```

```
.put(checkAuth, actualizarTarea)
.delete(checkAuth, eliminarTarea)

router.post('/estado/:id', checkAuth, cambiarEstado)

export default router
```

- Importo el archivo proyectoRoutes en el index.js

```
import express from 'express'
import conectarDB from './config/db.js'
import dotenv from 'dotenv'
import usuarioRoutes from './routes/usuarioRoutes.js'
import proyectoRoutes from './routes/proyectoRoutes.js'
import tareaRoutes from './routes/tareaRoutes.js'

const app = express()

dotenv.config()
app.use(express.json())

conectarDB()

//ROUTING

app.use("/api/usuarios", usuarioRoutes)
app.use("/api/proyectos", proyectoRoutes)
app.use("/api/tareas", tareaRoutes)




const PORT = process.env.PORT || 4000

app.listen(PORT, ()=>{
    console.log(`Servidor corriendo en el puerto ${PORT}`)
})
```

- Empiezo por el endpoint de agregarTarea. En POSTMAN, tipo PUT, me aseguro de que el TOKEN del usuario esté relacionado a algun proyecto existente suyo
- Pongo esto en el cuerpo, siguiendo el modelo creado previamente

```
{
    "nombre": "Elegir colores",
    "descripcion": "Elegir colores acordes a la reunión",
    "prioridad": "Media",
```

```
    "proyecto": "630507d6e8132e9b17d31421"
}
```

- Le pongo un console.log(req.body) a agregarTarea( ) para comprobar que se comunique bien con el endpoint
- Si todo esta correcto, lo siguiente es importar el modelo de Proyecto para comprobar que el proyecto exista
- Extraigo proyecto de req.body y uso el método findById. Lo imprimo en consola

```
const agregarTarea = async (req,res)=>{

    const {proyecto} = (req.body)

    const existeProyecto = await Proyecto.findById(proyecto)

    console.log(existeProyecto)
}
```

- Manejo el error

```
const agregarTarea = async (req,res)=>{

    const {proyecto} = (req.body)

    const existeProyecto = await Proyecto.findById(proyecto)

    console.log(existeProyecto)

    if(!existeProyecto){

       const error = new Error('El proyecto no existe')

       await res.status(404).json({msg: error.message})
    }
}
```

- Añado la comprobación de que el creador es el mismo que el usuario que solicita

```
const agregarTarea = async (req,res)=>{

    const {proyecto} = (req.body)
```

```
    const existeProyecto = await Proyecto.findById(proyecto)

    console.log(existeProyecto)

    if(!existeProyecto){

        const error = new Error('El proyecto no existe')

        await res.status(404).json({msg: error.message})
    }

    if(existeProyecto.creador.toString() !== req.usuario._id.toString()){

       const error = new Error("No tienes los permisos adecuados")

       return res.status(404).json({msg: error.message})
    }
}
```

- Si pasa las validaciones, puedo pasar a almacenarlo con un try catch
- Lo puedo hacer con new tarea o con mongoose se puede hacer de la siguiente forma
- Debo importar Tarea de models

```
const agregarTarea = async (req,res)=>{

    const {proyecto} = (req.body)

    const existeProyecto = await Proyecto.findById(proyecto)

    console.log(existeProyecto)

    if(!existeProyecto){
        const error = new Error('El proyecto no existe')
        await res.status(404).json({msg: error.message})
    }

    if(existeProyecto.creador.toString() !== req.usuario._id.toString()){
        const error = new Error("No tienes los permisos adecuados")
        return res.status(404).json({msg: error.message})
    }

    try {
        const tareaAlmacenada = await Tarea.create(req.body)
        res.json(tareaAlmacenada)

    } catch (error) {
        console.log(error)
    }
}
```

- Sólo quien creó el proyecto es quien puede añadir tareas. Los colaboradores pueden cambiar el estado de las tareas

Obtener una tarea y validación

- Obtener tarea requiere un id
- En POSTMAN, petición GET al id de la tarea (recordar el TOKEN válido)
- Puedo poner un console.log en obtenerTarea para ver si se comunican satisfactoriamente
- Extraigo el id con desestructuración. Hay que asegurarse que la tarea pertenece a un proyecto de la persona que está autenticada

```
const obtenerTarea = async (req,res)=>{

    const {id} = req.params

    const tarea = await  Tarea.findById(id)

     console.log(tarea)

}
```

- Puedo usar .populate para cruzar la información con el proyecto asociado en el Schema. De esta forma hago solo una consulta y me devuelve la tarea y el proyecto.

```
const obtenerTarea = async (req,res)=>{

    const {id} = req.params

     const tarea = await  Tarea.findById(id).populate("proyecto")

     console.log(tarea)

}
```

- Se requiere comprobar quien es el creador.

```
const obtenerTarea = async (req,res)=>{

    const {id} = req.params

     const tarea = await  Tarea.findById(id).populate("proyecto")
```

```
     if(tarea.proyecto.creador.toString() !==  req.usuario._id.toString()){

        const error = new Error("No tienes los permisos para acceder a la tarea")
        return res.status(404).json({msg: error.message})
     }
     res.json(tarea)
}
```

- Ahora POSTMAN me devuelve algo así


```
{
    "_id": "630622278533b7da996f8ae0",
    "nombre": "Elegir colores",
    "descripcion": "Elegir colores acordes a la reunión",
    "estado": false,
    "fechaEntrega": "2022-08-24T13:05:39.341Z",
    "prioridad": "Media",
    "proyecto": {
        "_id": "630507d6e8132e9b17d31421",
        "nombre": "tienda virtual",
        "descripcion": "Tienda virtual para un cliente",
        "fechaEntrega": "2022-08-23T17:01:07.369Z",
        "cliente": "códigonuevo",
        "colaboradores": [],
        "creador": "6303ab39029bae8dff989ff5",
        "createdAt": "2022-08-23T17:01:10.624Z",
        "updatedAt": "2022-08-23T17:01:10.624Z",
        "__v": 0
    },
    "createdAt": "2022-08-24T13:05:43.779Z",
    "updatedAt": "2022-08-24T13:05:43.779Z",
    "__v": 0
}
```

- Manejo del error si la tarea no existe

```
const obtenerTarea = async (req,res)=>{

    const {id} = req.params

    const tarea = await  Tarea.findById(id).populate("proyecto")

     if(!tarea){
         const error = new Error("La tarea no existe o no se encuentra")
         return res.status(404).json({msg: error.message})
     }
```

```
    if(tarea.proyecto.creador.toString() !==  req.usuario._id.toString()){
       const error = new Error("No tienes los permisos para acceder a la tarea")
       return res.status(403).json({msg: error.message})
    }
    res.json(tarea)
}
```

Actualizar Tarea

- Hay que verificar la tarea, ver si existe, y verificar que el usuario tenga los permisos necesarios
- En un try catch guardo los cambios de la petición PUT

```
const actualizarTarea = async (req,res)=>{

        const {id} = req.params

        const tarea = await  Tarea.findById(id).populate("proyecto")

        if(!tarea){
            const error = new Error("La tarea no existe o no se encuentra")
            return res.status(404).json({msg: error.message})
        }

        if(tarea.proyecto.creador.toString() !==  req.usuario._id.toString()){
           const error = new Error("No tienes los permisos para acceder a la
tarea")
           return res.status(403).json({msg: error.message})
        }

        tarea.nombre = req.body.nombre || tarea.nombre;
        tarea.descripcion = req.body.descripcion || tarea.descripcion;
        tarea.fechaEntrega = req.body.fechaEntrega || tarea.fechaEntrega;
        tarea.prioridad = req.body.prioridad || tarea.prioridad;

        try {

           const tareaAlmacenada = await tarea.save()
           res.json(tareaAlmacenada)

        } catch (error) {
           console.log(error)
        }

}
```

- Retorno del servidor la tareaAlmacenada porque luego en el frontEnd lo voy a sincronizar con el state

Eliminar Tarea

- Las mismas comprobaciones pero se hace la petición DELETE

```
const eliminarTarea = async (req,res)=>{
    const {id} = req.params
    const tarea = await  Tarea.findById(id).populate("proyecto")

    if(!tarea){
        const error = new Error("La tarea no existe o no se encuentra")
        return res.status(404).json({msg: error.message})
    }

    if(tarea.proyecto.creador.toString() !==  req.usuario._id.toString()){
       const error = new Error("No tienes los permisos para acceder a la tarea")
       return res.status(403).json({msg: error.message})
    }

    try {
        await tarea.deleteOne()
        res.json({msg: "Tarea eliminada"})
    } catch (error) {
        console.log(error)
    }
}
```

- Si ahora en POSTMAN voy al endpoint tareas/id_de_una_tarea con DELETE, borra dicha tarea
- Cambiar estado es algo que los colaboradores si van a poder hacer, por lo que se hará más adelante

---

Obtener tareas

- Listar las tareas de un proyecto puede estar en proyectoController pero tambien en tareaController
- Tengo el endpoint de tareas/:id, este id es del proyecto
- Monto el endpoint en POSTMAN
- Para obtener las tareas tienes que ser el creador del proyecto o colaborador/a
- Se podría hacer así

```
const obtenerTareas = async(req,res)=>{
    const {id} = req.params

    const existeProyecto = await Proyecto.findById(id)

    if(!existeProyecto){
        const error = new Error("El proyecto no existe o no se encuentra")
        return res.status(404).json({msg: error.message})
    }

    const tareas = await Tarea.find().where("proyecto").equals(id)
    res.json(tareas)
}
```

- Para no hacer dos llamados http, me puedo traer un campo extra en proyecto con las tareas asociadas y filtrarlas en el controlador
    - En obtenerProyecto de proyectoController
    - Es correcto que cuando me traiga el proyecto me traiga también las tareas asociadas a ese proyecto

```
const obtenerProyecto=async(req,res)=>{

   const {id} = req.params

   const proyecto = await Proyecto.findById(id)

    if(!proyecto){
        const error= new Error("No encontrado")
        return res.status(404).json({msg: error.message})
    }

    if(proyecto.creador.toString() !== req.usuario._id.toString()){
        const error = new Error("No tienes los permisos. Acción no válida")
        return res.status(401).json({msg: error.message})
    }

    const tareas = await Tarea.find().where("proyecto").equals(proyecto._id)

    res.json({
        proyecto,
        tareas
    })


}//MARCA ERROR el if(!proyecto)
```

- Manejándolo desde aquí ya no es necesario el obtenerTareas.
- Elimino el endpoint y las importaciones
- El arreglo de colaboradores en el schema de proyecto va a ser un arreglo que almacene el id de cada colaborador

FRONTEND

- En la carpeta donde se encuentra el backend creo el proyecto de react con nombre frontend

```
npm create vite@latest
```

- Instalo dos dependencias:

```
npm i axios react-router-dom
```

- Voy a tener dos terminales, una para el backend ejecutándose y otra para el frontend
- Hago limpieza de los archivos que no necesito en react y limpio App.jsx (dejo el index.css pero borro el contenido)

Instalar Tailwind

```
npm i -D tailwindcss postcss autoprefixer
```

- Creo el archivo de configuración

```
npx tailwindcss init -p
```

- En el archivo tailwind.config.cjs

```
module.exports = {
  content: ["index.html", "src/**/*.jsx"],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

- En index.css incluyo las directivas de tailwind

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

- Le añado un color al body en el html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>UpTask</title>
  </head>
  <body class="bg-gray-100 min-h-screen">
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Creando ROUTING con React Router Dom 6

- Importo BrowserRouter, Routes y Route de react-router-dom en App.jsx
- Voy a tenr dos áreas, una pública donde registrar, recuperar el password o iniciar sesión, y una privada de manegement de los proyectos
- Creo las carpetas en /src layouts, paginas, components
- En layouts creo AuthLayout.jsx
- En App.jsx incorporo el layout que englobe las rutas

```
import {BrowserRouter, Routes, Route} from 'react-router-dom'
import AuthLayout from './layouts/AuthLayout'


function App() {


  return (
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<AuthLayout/>} >

          </Route>

        </Routes>
      </BrowserRouter>
  )
}

export default App
```

- Creo el Login.jsx en /src/paginas y lo añado como index (es decir, la página que muestra en "/") al BrowserRouter
- No se visualiza el Login porque hay que definir el Outlet en el componente de Layout.jsx

```
import {Outlet} from 'react-router-dom'
```

```
const AuthLayout = () => {
  return (
    <>
      <div>AuthLayout</div>


      <Outlet />

    </>


  )
}

export default AuthLayout
```

- Ahora ya está inyectando el contenido y visualizo Login
- Creo otra ruta llamada registrar con su componente en paginas
- Creo dos páginas más: OlvidePassword y NuevoPassword

```
import {BrowserRouter, Routes, Route} from 'react-router-dom'
import AuthLayout from './layouts/AuthLayout'
import Login from './paginas/Login'
import NuevoPassword from './paginas/NuevoPassword'
import OlvidePassword from './paginas/OlvidePassword'
import Registrar from './paginas/Registrar'


function App() {


  return (
      <BrowserRouter>
        <Routes>
          <Route path="/" element={<AuthLayout/>} >
            <Route index element={<Login />} />
            <Route path="registrar" element={<Registrar />} />
            <Route path="olvide-password" element={<OlvidePassword/>} />
            <Route path="olvide-password/:token" element={<NuevoPassword />} />

          </Route>
        </Routes>
      </BrowserRouter>
  )
}

export default App
```

- En olvidé password el usuario va a colocar su email y se le enviará el token via email con el enlace, se leerá el token de la url y se hará la validación
- Creo otra página, ConfirmarCuenta.jsx

```
    <Route path="confirmar/:id" element={<ConfirmarCuenta />} />
```

- Una vez que le envie el email para confirmar la cuenta presiona el boton, leo el id de la url y se hace la validación

Creando el Layout principal para el área pública

- Coloco un main con el outlet, centro el contenido con mx-auto y añado un mediaquery. Tambien un display flex en tamaño mediano, y lo centro
- Añado un div con un mediaquery de tamaño mediano que tome 2 terceras partes y en uno más grande 2 partes de 5 (el 40%)
- Coloco el Outlet dentro
- AuthLayout.jsx

```
import {Outlet} from 'react-router-dom'

const AuthLayout = () => {
  return (
    <>


      <main className="container mx-auto mt-5 md:mt-20 p-5 md:flex md:justify-
center">
        <div className="md:w-2/3 lg:w-2/5">
          <Outlet />
        </div>
      </main>


    </>


  )
}

export default AuthLayout
```

- De esta manera aplica a todos los hijos de AuthLayout

Creando formulario Login

- Añado los estilos. Le pongo un htmlFor para comunicar con el id el label y el input, así cuando clico en el label resalta el input
- Copio este div para el password.
- Añado el input de tipo submit antes de cerrar el form
- Queda así el componente (con algunos retoques de margins):

```
import React from 'react'

const Login = () => {
  return (
    <>
      <h1 className="text-sky-600 font-black text-6xl">Inicia sesión y administra
tus
      <span className="text-slate-700"> Proyectos</span></h1>


      <form className="bg-white my-10 shadow rounded-lg p-10">

       <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="email">Email</label>

          <input
           id="email"
           type="email"
           placeholder="Email de registro"
           className="w-full mt-3 p-2 border rounded-xl bg-gray-50"/>
       </div>

       <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="password">Password</label>

          <input
           id="password"
           type="password"
           placeholder="Tu password"
           className="w-full mt-3 p-2 border rounded-xl bg-gray-50 mb-3"/>
       </div>

       <input type="submit"
       value="Iniciar Sesión"
       className="bg-sky-700 mb-5 text-white w-full font-bold p-2 rounded-xl
uppercase hover:cursor-pointer hover:bg-sky-800 transition-colors"/>
      </form>
    </>
  )
}

export default Login
```

Añadiendo el Routing

- Añado un nav despues del form (dentro del div) en Login.jsx

```
  <nav className="lg:flex lg:justify-between">
      <Link to="/registrar" className="block text-slate-500 text-center my-5">¿No
tienes cuenta? Registrate</Link>

      <Link to="/olvide-password" className="block text-slate-500 text-center my-
5">¿Olvidaste tu password?</Link>
  </nav>
```

Registrar

- Copio todo lo que hay en el fragment y lo incorporo en Registrar.jsx
- Le añado un campo con el nombre y otro con repetir password

Nuevo Password

- Copio el fragment de registrar y dejo solo el campo del password. Cambio el titulo, el input y el routing
- Copio el fragment de NuevoPassword y lo copio en Confirmar cuenta. Dejo solo el titulo y lo cambio por confirma tu cuenta
    - No hay routing a otras páginas
    - No hay formulario
    - Desde aqui se confirmará la cuenta

NOTA: Adecuar estas páginas al gusto, OlvidePassword, NuevoPassword, etc

Añadiendo state al formulario

- Borro todo lo que hay en la base de datos
- Creo el state para los 4 campos de Registrar.jsx. Lo hago por separado (un state para cada campo)
    - con el value y el onChange de siempre

Validación

- En el onSubmit del form añado una nueva función llamada handleSubmit
- Pongo todos los states en un arreglo y verifico que no haya ningún espacio en blanco con .includes
- Creo un nuevo componente llamado Alerta para mostrarlo en caso de que hayan campos vacíos

```
const Alerta = ({alerta}) => {
  return (
    <div className={`${alerta.error ? 'from-red-400 to-red-600': 'from-sky-400 to-
sky-600'} bg-gradient-to-br
    text-center p-3 rounded-xl text-white font-bold text-sm my-10 uppercase`}>
        {alerta.msg}
    </div>
  )
}

export default Alerta
```

- Creo un nuevo state para manejar la alerta, con un objeto vacío ya que tiene un objeto con error y otro con mensaje
- Extraigo el mensaje de la alerta. En caso de que no esté vacío muestro la alerta

```
import { useState } from 'react'
import {Link} from 'react-router-dom'
import Alerta from '../components/Alerta'



const Registrar = () => {

  const [nombre, setNombre] = useState('')
  const [email, setEmail] = useState('')
  const [password, setPassword]= useState('')
  const [password2, setPassword2]= useState('')
  const [alerta, setAlerta]= useState({})
const handleSubmit= (e)=>{
  e.preventDefault()

  if([nombre,email,password,password2].includes('')){

    setAlerta({
      msg: "Todos los campos son obligatorios",
      error: true
    })
    return
  }
}

const {msg} = alerta;

  return (
    <>
    <h1 className="text-sky-600 font-black text-6xl">Crea tu cuenta y administra
tus
    <span className="text-slate-700"> Proyectos</span></h1>
```

```
      {msg && <Alerta alerta={alerta}/>}

      <form className="bg-white my-10 shadow rounded-lg p-10"
        onSubmit={handleSubmit}>
        <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="nombre">Nombre</label>

          <input
           id="nombre"
           type="text"
           value={nombre}
           onChange={e=>setNombre(e.target.value)}
           placeholder="Tu nombre"
           className="w-full mt-3 p-2 border rounded-xl bg-gray-50"/>
        </div>
        <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="email">Email</label>

          <input
           id="email"
           type="email"
           value={email}
           onChange={e=>setEmail(e.target.value)}
           placeholder="Email de registro"
           className="w-full mt-3 p-2 border rounded-xl bg-gray-50"/>
        </div>
        <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="password">Password</label>

          <input
           id="password"
           type="password"
           value={password}
           onChange={e=> setPassword(e.target.value)}
           placeholder="Tu password"
           className="w-full mt-3 p-2 border rounded-xl bg-gray-50 mb-3"/>
        </div>
        <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="repetir-password">Repite tu password</label>

          <input
           id="password2"
           type="password"
           value={password2}
           onChange={e=>setPassword2(e.target.value)}
```

```
          placeholder="Repite tu password"
          className="w-full mt-3 p-2 border rounded-xl bg-gray-50 mb-3"/>
        </div>

        <input type="submit"
        value="Registrate"
        className="bg-sky-700 mb-5 text-white w-full font-bold p-2 rounded-xl
uppercase hover:cursor-pointer hover:bg-sky-800 transition-colors"/>
      </form>

      <nav className="lg:flex lg:justify-between">
      <Link to="/" className="block text-slate-500 text-center my-5">¿Ya tienes
cuenta? Inicia sesión</Link>
      <Link to="/olvide-password" className="block text-slate-500 text-center my-
5">¿Olvidaste tu password?</Link>


      </nav>
    </>
    )
}

export default Registrar
```

- Ahora falta validar que los dos passwords sean iguales

Validando password y password2

- Primero verifico que matchean
- Tambien verifico que sea más largo de 6 caracteres

```
const handleSubmit= (e)=>{
  e.preventDefault()

  if([nombre,email,password,password2].includes('')){

    setAlerta({
      msg: "Todos los campos son obligatorios",
      error: true
    })
    return
  }
  if(password !== password2){
    setAlerta({
      msg: "Los passwords no coinciden",
      error: true
    })
    return
  }
```

```
  if(password.length < 6){
    setAlerta({
      msg: "El password es muy corto, mínimo 6 caracteres",
      error: true
    })
    return
  }
  setAlerta({})
  console.log("creando registro...")
}
```

- Ahora que ya valida, lo próximo es llamar a la API

Enviando request a la API

- Usaré un try catch en lugar de ese console.log de creando registro
- Lo mismo que incluia en POSTMAN en el body lo haré ahora con código.
- Para ello usaré axios

```
  const handleSubmit= async (e)=>{
  e.preventDefault()

  if([nombre,email,password,password2].includes('')){

    setAlerta({
      msg: "Todos los campos son obligatorios",
      error: true
    })
    return
  }
  if(password !== password2){
    setAlerta({
      msg: "Los passwords no coinciden",
      error: true
    })
    return
  }
  if(password.length < 6){
    setAlerta({
      msg: "El password es muy corto, mínimo 6 caracteres",
      error: true
    })
    return
  }
  setAlerta({})

  try {
    const respuesta = await axios.post('http://localhost:4000/api/usuarios',{
```

```
      nombre,
      email,
      password
    })

    console.log(respuesta)
  } catch (error) {
    console.log(error)
  }
}
```

- Esto solo no funciona, por un problema de CORS ya los dos están en localhost

Habilitar peticiones por CORS

- Se configura en el backend (en el index)
- Instalo el paquete cors en la carpeta raíz del backend

> npm i cors

- Importo el paquete en el index.js
- Creo una lista blanca, compruebo si en origins esta la dirección de la lista blanca (mi localhost del frontend)
  - Si esta, le doy acceso con el callback en true, el primer parámetro null(de error)
  - Si no esta (else) le niego el acceso y mando un error
- Tengo que añadirle los headers
- Este es el código en el index

```
const whitelist = ["http://127.0.0.1:5173"];

const corsOptions ={
    origin: function(origin, callback){
        if(whitelist.includes(origin)){
            callback(null, true)
        }else{
            callback(new Error("Error de cors"))
        }
    }
}

app.use(function(req, res, next) {
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-
Type, Accept");
    next();
  });
```

- Puedo desestructurar la data de la petición axios, que es lo que interesa
- Si voy al backend, al usuarioController, ahí tenia el res.json(usuarioAlmacenado) en la funcion registrar

```
const registrar = async (req,res)=>{

    const {email} = req.body
    const existeUsuario =  await Usuario.findOne({email})

    if(existeUsuario){
        const error = new Error('Usuario ya registrado');
        return res.status(400).json({msg:error.message})
    }

    try {
        const usuario = new Usuario(req.body)
        usuario.token= generarId()

        const usuarioAlmacenado= await usuario.save()
        res.json({usuarioAlmacenado})


    } catch (error) {
        console.log(error)
    }


}
```

- Registrar.jsx (frontend)

```
 const handleSubmit= async (e)=>{

 e.preventDefault()

  if([nombre,email,password,password2].includes('')){

    setAlerta({
      msg: "Todos los campos son obligatorios",
      error: true
    })
    return
  }
  if(password !== password2){

   setAlerta({
      msg: "Los passwords no coinciden",
      error: true
    })
    return
```

```
    }
    if(password.length < 6){

     setAlerta({
        msg: "El password es muy corto, mínimo 6 caracteres",
        error: true
      })
      return
    }

    setAlerta({})

    try {
      const {data} = await axios.post('http://localhost:4000/api/usuarios',{
        nombre,
        email,
        password
      })
      console.log(data) //esto me devuelve solo lo que necesito, los datos del
usuario


    } catch (error) {
      console.log(error)
    }
}
```

- En el archivo usuarioController del backend, en la función registrar, en el try, cuando sale bien, en lugar de devolver usuarioAlmacenado, devolveré un mensaje

```
const registrar = async (req,res)=>{

    const {email} = req.body
    const existeUsuario =  await Usuario.findOne({email})

    if(existeUsuario){
        const error = new Error('Usuario ya registrado, revisa tu mail');
        return res.status(400).json({msg:error.message})
    }

    try {
        const usuario = new Usuario(req.body)
        usuario.token= generarId()

        const usuarioAlmacenado= await usuario.save()
        res.json({msg:"Usuario creado corectamente!"})


    } catch (error) {
        console.log(error)
```

```
    }


}
```

- Para acceder al mensaje del servidor y mostrarlo en pantalla uso el setAlerta
- Para mostrar el error del servidor si intento registrar un correo existente, caigo en el catch
- Para mostrar la info de este mensaje es usa error.response (documentación de axios)
    - Ahi tengo data, la cual puedo mostrar en el setAlerta con el msg (así lo nombré en el controlador del backend)

```
const handleSubmit= async (e)=>{

 e.preventDefault()

  if([nombre,email,password,password2].includes('')){

    setAlerta({
      msg: "Todos los campos son obligatorios",
      error: true
    })
    return
  }
  if(password !== password2){

   setAlerta({
      msg: "Los passwords no coinciden",
      error: true
    })
    return
  }
  if(password.length < 6){

   setAlerta({
      msg: "El password es muy corto, mínimo 6 caracteres",
      error: true
    })
    return
  }
  setAlerta({})

  try {
    const {data} = await axios.post('http://localhost:4000/api/usuarios',{
      nombre,
      email,
      password
    })

    setAlerta({
      msg: data.msg,
```

```
        error: false    //para que muestre el mensaje en azul de correcto
      })


    } catch (error) {

      setAlerta({
        msg: error.response.data.msg,
        error: true
      })
    }
  }
```

- Las rutas deben de estar ocultas en variables de entorno

Instalando y configurando NodeMailer

> npm i nodemailer

- Usaré mailtrap. En integrations pongo nodeMailer y copio la configuración de smtp (no la pongo aqui por seguridad)
- Creo un archivo en helpers llamado emails.js, con una función de flecha llamada emailRegistro
- emails.js:

```
import nodemailer from 'nodemailer'

export const emailRegistro = (datos)=>{

}
```

- Importo esta función en el controlador
- Voy a la función de registrar, que es donde tengo usuario con token e email
- Le coloco un console.log(usuarioAlmacenado)

```
const registrar = async (req,res)=>{

    const {email} = req.body

    const existeUsuario =  await Usuario.findOne({email})

    if(existeUsuario){
        const error = new Error('Usuario ya registrado');
        return res.status(400).json({msg:error.message})
    }

    try {
```

```
        const usuario = new Usuario(req.body)
        usuario.token= generarId()

        const usuarioAlmacenado= await usuario.save()

        //Enviar email de confirmacion

        console.log(usuarioAlmacenado)

        res.json({msg:"Usuario creado correctamente! Revisa tu mail"})



    } catch (error) {
        console.log(error)
    }


}
```

- Puedo ver por el console.log que me devuelve el usuario completo. Necesito el nombre y el token
- En lugar de un console.log mando los datos en un objeto como parametro de emailRegistro

```
const registrar = async (req,res)=>{

    const {email} = req.body
    const existeUsuario =  await Usuario.findOne({email})

    if(existeUsuario){
        const error = new Error('Usuario ya registrado');
        return res.status(400).json({msg:error.message})
    }

    try {
        const usuario = new Usuario(req.body)
        usuario.token= generarId()

        const usuarioAlmacenado= await usuario.save()

        //Enviar email de confirmacion
        emailRegistro({
            nombre: usuario.nombre,
            email: usuario.email,
            token: usuario.token
        })

        res.json({msg:"Usuario creado correctamente! Revisa tu mail"})
```

```
    } catch (error) {
        console.log(error)
    }


}
```

- Para comprobar que todo va bien, puedo ponerle un console.log(datos) en emailRegistro
- Ahora puedo ver nombre, email y token

Enviando el email de bienvenida

- Extraigo el email, el nombre y el token en la funcion emailregistro
- Copio y pego la configuración de mailtrap que hay en el inbox como SMTP Settings, con el Integrations en nodeMailer, dentro de la función
    - A esto se le conoce como transport. Habrá que guardar en variables de entorno el user y el pass
- Creo la función para enviar el email, y con un template literal armo el cuerpo del mensaje
    - Primero el from (de quién) , luego el to(a quien), el subject (el asunto), el text y el html
    - en el html añado la url del frontend, el endpoint de confirmar y le paso el token en la url para poder leerlo despues
- En la variable de entorno FRONTEND_URL en .env tengo la direccion del server del front-end (lo que puse en la whitelist del cors)
- Hay que guardarlo en una variable de entrono, igual que la del backend (CAPITULO 416)
- asi queda emailRegistro

```
import nodemailer from 'nodemailer'

export const emailRegistro = async(datos)=>{
    const {email, nombre, token} = datos;

      //copy past de mailtrap, lo he pasado a variables de entorno
    const transport = nodemailer.createTransport({
        host: process.env.HOST_SMTP,
        port: process.env.PORT_SMTP,
        auth: {
          user: process.env.USER_SMTP,
          pass: process.env.PASS_SMTP
        }
      });

      //información del email

      const info = await transport.sendMail({
        from:"UpTask -  Administrador de Proyectos",
        to: email,
        subject: "UpTask - Comprueba tu cuenta",
        text: "Comprueba tu cuenta en UpTask",
        html:` <p>Hola, ${nombre}. Comprueba tu cuenta en UpTask</p>
        <p>Tu cuenta ya esta casi lista. Sólo debes comprobarla en el siguiente
```

```
enlace</p>
        <a href="${process.env.FRONTEND_URL}/confirmar/${token}">Comprobar
Cuenta</a>
        <p>Si tu no creaste esta cuenta puedes eliminar este mensaje </p>
        `


      })
}
```

Confirmar Cuenta

- Importo useEffect y useState en confirmarCuenta.jsx
- useEffect pq voy a leer la url y se requiere que el código se ejecute una vez y enviar una petición
- Importo también useParams y Link de react-router-dom.
- Importo axios
- Importo la alerta

- Gracias al id dinámico en la ruta de confirmar/:id se puede extraer con useParams y hacer el match

```
import {useState, useEffect} from 'react'
import {useParams, Link} from 'react-router-dom'
import axios from 'axios'
import Alerta from '../components/Alerta'

const ConfirmarCuenta = () => {

  const params = useParams()
  console.log(params)

  return (
    <>
    <h1 className="text-sky-600 font-black text-6xl">Confirma tu cuenta y crea tus
    <span className="text-slate-700"> Proyectos</span></h1>
  </>
  )
}

export default ConfirmarCuenta
```

- Puedo observar gracias al console.log que me devuelve el :id (el token) en un objeto
- Hago destructuring para extraer el id
- Hay que enviar el mismo endpoint al backend, para eso usaré el useEffect, porque requiero comprobar eso cuando el componente cargue y una sola vez

```
import {useState, useEffect} from 'react'
import {useParams, Link} from 'react-router-dom'
import axios from 'axios'
import Alerta from '../components/Alerta'

const ConfirmarCuenta = () => {

  const params = useParams()

  const {id} = params

  useEffect(()=>{

    const confirmarCuenta= async()=>{
      try {
        const url=`http://localhost:4000/api/usuarios/confirmar/${id}` //habría
que guardar esta ruta en una variable de entorno
        const {data}= await axios(url)

        console.log(data)

      } catch (error) {
        console.log(error)
      }
    }
  }, [])

  return (
    <>
    <h1 className="text-sky-600 font-black text-6xl">Confirma tu cuenta y crea tus
    <span className="text-slate-700"> Proyectos</span></h1>
  </>
  )
}

export default ConfirmarCuenta
```

- Cómo es una petición GET pongo solo axios (default). En el error habrá que debuggearlo mejor que con un console.log
- Uso async await porque consulto la API. Es de tipo GET porque sólo debo visitar la url
- Si miro en Compass ha hecho lo que se detalla en el usuarioController, que es borrar el token y volver la cuenta a true. Da error porque lo hace dos veces (error de código) pero está funcionando
- Creo el state en confirmar cuenta para mostrar la alerta de forma condicional extrayendo msg

```
import {useState, useEffect} from 'react'
import {useParams, Link} from 'react-router-dom'
import axios from 'axios'
import Alerta from '../components/Alerta'
```

```
const ConfirmarCuenta = () => {

  const params = useParams()

  const {id} = params

  const [alerta,setAlerta] = useState({})

  useEffect(()=>{

    const confirmarCuenta= async()=>{
      try {
        const url=`http://localhost:4000/api/usuarios/confirmar/${id}`
        const {data}= await axios(url)

        setAlerta({
          msg: data.msg,
          error: false
        })

      } catch (error) {

        setAlerta({
          msg: error.response.data.msg,
          error: true
        })

      }
    }
    confirmarCuenta()
  }, [])

  const {msg} = alerta

  return (
    <>
    <h1 className="text-sky-600 font-black text-6xl">Confirma tu cuenta y crea tus
    <span className="text-slate-700"> Proyectos</span></h1>

    <div>
      {msg && <Alerta alerta={alerta} />}
    </div>
  </>
  )
}

export default ConfirmarCuenta
```

- Creo otro state para renderizar un link de forma condicional para iniciar sesión

```
import {useState, useEffect} from 'react'
import {useParams, Link} from 'react-router-dom'
import axios from 'axios'
import Alerta from '../components/Alerta'

const ConfirmarCuenta = () => {

  const params = useParams()

  const {id} = params

  const [alerta,setAlerta] = useState({})

  const [cuentaConfirmada, setCuentaConfirmada] = useState(false)

  useEffect(()=>{

    const confirmarCuenta= async()=>{
      try {
        const url=`http://localhost:4000/api/usuarios/confirmar/${id}`
        const {data}= await axios(url)

        setAlerta({
          msg: data.msg,
          error: false
        })
        setCuentaConfirmada(true)

      } catch (error) {

        setAlerta({
          msg: error.response.data.msg,
          error: true
        })

      }
    }
    confirmarCuenta()
  }, [])

  const {msg} = alerta

  return (
    <>
    <h1 className="text-sky-600 font-black text-6xl">Confirma tu cuenta y crea tus
    <span className="text-slate-700"> Proyectos</span></h1>

    <div>
      {msg && <Alerta alerta={alerta} />}

      {cuentaConfirmada &&
       <Link to="/" className="block text-slate-500 text-center my-5">Iniciar
sesión</Link>}
```

```
      </div>
    </>
    )
}

export default ConfirmarCuenta
```

- Método de confirmar en usuarioController.js, en el backend:

```
const confirmar = async(req, res)=>{
    const {token} = req.params

    const usuarioConfirmar = await Usuario.findOne({token})

    if(!usuarioConfirmar){
        const error = new Error("Token no válido")
        return res.status(404).json({msg: error.message})
    }

    try {
        usuarioConfirmar.confirmado = true
        usuarioConfirmar.token= ""

        await usuarioConfirmar.save();
        res.json({msg: "Usuario confirmado correctamente"})


    } catch (error) {
        console.log(error)
    }
}
```

Primeros pasos para reestablecer contraseña

- NOTA: mailtrap asociado a miguenovuelve
- Voy al componente OlvidePassword, en el front
- Ahi copio el header y el campo de email del componente Registrar.jsx pongo un texto de Recuperar acceso
- Coloco tambien un link, un input tipo submit, el state del email, etc
- OlvidePassword.js:

```
import {Link} from 'react-router-dom'
import {useState} from 'react'

const OlvidePassword = () => {

  const [email, setEmail]= useState('')

  return (
    <>
     <h1 className="text-sky-600 font-black text-6xl">Recupera tu acceso y no
pierdas tus
    <span className="text-slate-700"> Proyectos</span></h1>
    <form className="my-5 bg-white rounded-xl shadow-lg p-10">

        <label className="uppercase text-gray-600 block font-bold"
        htmlFor="email">Email</label>

        <input
         id="email"
         type="email"
         value={email}
         onChange={e=>setEmail(e.target.value)}
         placeholder="Email de registro"
         className="w-full mt-3 p-2 border rounded-xl bg-gray-50"/>


        <input type="submit"
          value="Recupera tu cuenta"
          className="bg-sky-700 mb-3 text-white w-full font-bold p-2 rounded-lg
mt-5 uppercase hover:cursor-pointer hover:bg-sky-800 transition-colors"/>

      <div className="flex justify-between">
      <Link to="/" className="block text-slate-500 text-center my-5">¿Ya tienes
cuenta? Inicia sesión</Link>
      <Link to="/registrar" className="block text-slate-500 text-center my-5">¿No
tienes cuenta? Crea una!</Link>

      </div>
    </form>
    </>
  )
```

```
}

export default OlvidePassword
```

- Coloco el onSubmit en el form, con una función llamada handleSubmit
- Coloco dentro el e.preventDefault y hago una validación sencilla
- Extraigo el mensaje de alerta y lo muestro de forma condicional encima del form

```
import {Link} from 'react-router-dom'
import {useState} from 'react'
import Alerta from '../components/Alerta'

const OlvidePassword = () => {

  const [email, setEmail]= useState('')
  const [alerta, setAlerta] = useState({})

const handleSubmit = async(e)=>{

 e.preventDefault()

  if(email === '' || email.length < 6){

    setAlerta({
      msg: "El email es obligatorio",
      error: true
    })

    return
  }
}

const {msg} = alerta

  return (
    <>
     <h1 className="text-sky-600 font-black text-6xl">Recupera tu acceso y no
pierdas tus
    <span className="text-slate-700"> Proyectos</span></h1>

      {msg && <Alerta alerta={alerta} />}
    <form onSubmit={handleSubmit}className="my-5 bg-white rounded-xl shadow-lg p-
10">

        <label className="uppercase text-gray-600 block font-bold"
        htmlFor="email">Email</label>

        <input
         id="email"
         type="email"
```

```
          value={email}
          onChange={e=>setEmail(e.target.value)}
          placeholder="Email de registro"
          className="w-full mt-3 p-2 border rounded-xl bg-gray-50"/>


        <input type="submit"
          value="Recupera tu cuenta"
          className="bg-sky-700 mb-3 text-white w-full font-bold p-2 rounded-lg
mt-5 uppercase hover:cursor-pointer hover:bg-sky-800 transition-colors"/>

      <div className="flex justify-between">
      <Link to="/" className="block text-slate-500 text-center my-5">¿Ya tienes
cuenta? Inicia sesión</Link>
      <Link to="/registrar" className="block text-slate-500 text-center my-5">¿No
tienes cuenta? Crea una!</Link>

      </div>
    </form>
    </>
  )
}

export default OlvidePassword
```

Enviando email y token para reestablecer password

- Coloco un try catch en el handleSubmit después de la validación
- Le paso el endpoint de tipo post (COMO EN POSTMAN) usando axios y le paso un objeto con el mail
- Es de tipo post porque debo pasarle el mail, mirar OlvidePassword del controller más abajo

---

NOTA: la url debería estar en una variable de entorno

```
import {Link} from 'react-router-dom'
import {useState} from 'react'
import Alerta from '../components/Alerta'
import axios from 'axios'

const OlvidePassword = () => {

  const [email, setEmail]= useState('')
  const [alerta, setAlerta] = useState({})

const handleSubmit = async(e)=>{
  e.preventDefault()

  if(email === '' || email.length < 6){
```

```
    setAlerta({
       msg: "El email es obligatorio",
       error: true
     })
     return
   }

   try {
     const {data} = await  axios.post(`http://localhost:4000/api/usuarios/olvide-
password`, {email})

     console.log(data)

   } catch (error) {

     console.log(error.response)
   }
}

const {msg} = alerta

   return (
     <>
      <h1 className="text-sky-600 font-black text-6xl">Recupera tu acceso y no
pierdas tus
     <span className="text-slate-700"> Proyectos</span></h1>
       {msg && <Alerta alerta={alerta} />}

     <form onSubmit={handleSubmit}className="my-5 bg-white rounded-xl shadow-lg p-
10">

         <label className="uppercase text-gray-600 block font-bold"
         htmlFor="email">Email</label>

         <input
          id="email"
          type="email"
          value={email}
          onChange={e=>setEmail(e.target.value)}
          placeholder="Email de registro"
          className="w-full mt-3 p-2 border rounded-xl bg-gray-50"/>


         <input type="submit"
           value="Recupera tu cuenta"
           className="bg-sky-700 mb-3 text-white w-full font-bold p-2 rounded-lg
mt-5 uppercase hover:cursor-pointer hover:bg-sky-800 transition-colors"/>

       <div className="flex justify-between">
       <Link to="/" className="block text-slate-500 text-center my-5">¿Ya tienes
cuenta? Inicia sesión</Link>
       <Link to="/registrar" className="block text-slate-500 text-center my-5">¿No
tienes cuenta? Crea una!</Link>
```

```
      </div>
    </form>
    </>
  )
}

export default OlvidePassword
```

- Si mando un mail que no existe aparece en consola dentro de data.msg "Usuario no existe"
- Manejo el error con setAlerta

```
const handleSubmit = async(e)=>{
  e.preventDefault()

  if(email === '' || email.length < 6){

   setAlerta({
      msg: "El email es obligatorio",
      error: true
    })
    return
  }

  try {
    const {data} = await  axios.post(`http://localhost:4000/api/usuarios/olvide-
password`, {email})

    console.log(data)

  } catch (error) {

    setAlerta({
      msg: error.response.data.msg,
      error: true
    })
  }
}
```

- Si coloco un mail que existe me muestra un mensaje en pantalla: "Hemos enviado un mail con las instrucciones" y me genera un nuevo token
- Manejo el mensaje con setAlerta

```
const handleSubmit = async(e)=>{

 e.preventDefault()

  if(email === '' || email.length < 6){
```

```
      setAlerta({
        msg: "El email es obligatorio",
        error: true
      })
      return
    }

    try {
      const {data} = await  axios.post(`http://localhost:4000/api/usuarios/olvide-
password`, {email})

      setAlerta({
        msg: data.msg,
        error: false
      })

    } catch (error) {
      setAlerta({
        msg: error.response.data.msg,
        error: true
      })
    }
}
```

- Paso el OlvidePassword del controller por aqui:

```
const olvidePassword = async (req,res)=>{
    const {email} = req.body
    const usuario = await Usuario.findOne({email})
    if(!usuario){
        const error = new Error("El usuario no existe")
        return res.status(404).json({msg: error.message})
    }
    try {
        usuario.token= generarId()
        await usuario.save()
        res.json({msg: "Hemos enviado un email con las instrucciones"})

    } catch (error) {

    }

}
```

- Notar que poner msg siempre facilita las cosas a la hora de reutilizar la Alerta
- Es en el try del controlador desde dónde voy a enviar el email
- Copio y pego del mail en helpers (emailRegistro) y le cambio el nombre por emailOlvidePassword.
- Lo exporto y lo importo en el controlador

```
 export const emailOlvidePassword = async(datos)=>{
      const {email, nombre, token} = datos;

      const transport = nodemailer.createTransport({
          host: process.env.HOST_SMTP,
          port: process.env.PORT_SMTP,
          auth: {
            user: process.env.USER_SMTP,
            pass: process.env.PASS_SMTP
          }
        });

        //información del email

        const info = await transport.sendMail({
          from:"UpTask -  Administrador de Proyectos",
          to: email,
          subject: "UpTask - Reestablece tu password",
          text: "Reestablece tu password en UpTask",
          html:` <p>Hola, ${nombre}. Has solicitado resetear tu password</p>
          <p>Sigue el siguiente enlace para reestablecer tu password</p>
          <a href="${process.env.FRONTEND_URL}/olvide-
password/${token}">Reestablecer password</a>
          <p>Si este mensaje no es para ti puedes eliminarlo </p>
          `

        })
      }
```

- Lo añado en el try y lleno el cuerpo con el usuario (que ya tengo disponible) y sus valores dentro de un objeto

```
const olvidePassword = async (req,res)=>{

   const {email} = req.body
   const usuario = await Usuario.findOne({email})

   if(!usuario){
        const error = new Error("El usuario no existe")
        return res.status(404).json({msg: error.message})
    }
    try {
        usuario.token= generarId()
        await usuario.save()

        emailOlvidePassword({
            nombre: usuario.nombre,
```

```
            email: usuario.email,
            token: usuario.token
        })
        res.json({msg: "Hemos enviado un email con las instrucciones"})

    } catch (error) {
        console.log(error)
    }

}
```

Añadiendo el token para resetear password

- Hay que leer el token, extraerlo y hacer un llamado a la base de datos para hacer match
- Voy al componente (y ruta) NuevoPassword.jsx
- Hago una serie de importaciones: Link, useParams, etc (se vé más abajo)
- Hago uso del useEffect, va a ejecutarse una sola vez por lo que le coloco el arreglo de dependencias vacío
- Extraigo el token con useParams, lo llamo así porque así lo puse en el enrutado
- Es un get porque lo estoy consultando en la base de datos

```
import {Link, useParams} from 'react-router-dom'
import {useState, useEffect} from 'react'
import axios from 'axios'
import Alerta from '../components/Alerta'

const NuevoPassword = () => {

  const {token} = useParams()

  useEffect(()=>{

     const comprobarToken= async()=>{
       try {
        const {data}=await axios(`http://localhost:4000/api/usuarios/olvide-
password/${token}`)

        console.log(data)

       } catch (error) {
        console.log(error.response)
       }
    }
    comprobarToken()

  },[])

  return("<componente NuevoPassword.JSX...>")
```

- A estas alturas, en la pantalla de Nuevo Password el console.log(data) me devuelve "usuario existe" (tengo el token en la URL)
- Paso los datos a usuarioController.js:

```
const comprobarToken = async(req, res)=>{
    const {token} = req.params
    const tokenValido = await Usuario.findOne({token})

    if(tokenValido){
        res.json({msg:"El usuario existe"})
    }else{
        const error = new Error("Token no válido")
        return res.status(404).json({msg: error.message})
    }
}

const nuevoPassword= async (req,res)=>{
    const {token} = req.params
    const {password}= req.body

    const usuario = await Usuario.findOne({token})

    if(usuario){
        usuario.password = password
        usuario.token=""

    try {
        await usuario.save()
        res.json({msg:"Password almacenado correctamente"})

    } catch (error) {
       console.log(error)
    }
    }else{
        const error = new Error("Token no válido")
        return res.status(404).json({msg: error.message})

    }

}
```

- Creo un nuevo estado con tokenValido (borro lo de la data, era sólo para el console.log)
- Será apara mostrar el formulario en el caso de que el token sea válido o no
- También creo el state de Alerta, extraigo el msg y lo muestro de manera condicional

```
import {Link, useParams} from 'react-router-dom'
import {useState, useEffect} from 'react'
import axios from 'axios'
```

```
import Alerta from '../components/Alerta'

const NuevoPassword = () => {

  const {token} = useParams()
  const [tokenValido, setTokenValido] = useState(false)
  const [alerta,setAlerta] = useState({})

  useEffect(()=>{
    const comprobarToken= async()=>{
      try {
         await axios(`http://localhost:4000/api/usuarios/olvide-
password/${token}`)

        setTokenValido(true)

      } catch (error) {

        setAlerta({
          msg: error.response.data.msg,
          error: true
        })
      }
    }
    comprobarToken()

  },[])

const {msg} = alerta

  return (
    <>
    <h1 className="text-sky-600 font-black text-6xl">Recupera tu password y
administra tus
    <span className="text-slate-700"> Proyectos</span></h1>

    {msg && <Alerta alerta={alerta} />}

    {tokenValido &&(

    <form className="bg-white my-10 shadow rounded-lg p-10">

      <div className="my-5">

        <label className="uppercase text-gray-600 block font-bold"
        htmlFor="password">Nuevo Password</label>

        <input
         id="password"
         type="password"
         placeholder="Escribe tu nuevo password"
         className="w-full mt-3 p-2 border rounded-xl bg-gray-50 mb-3"/>
      </div>
```

```
      <input type="submit"
      value="Guardar password"
      className="bg-sky-700 mb-5 text-white w-full font-bold p-2 rounded-xl
uppercase hover:cursor-pointer hover:bg-sky-800 transition-colors"/>
    </form>


    )}


    <nav className="lg:flex lg:justify-between">
      <Link to="/" className="block text-slate-500 text-center my-5">¿Ya tienes
cuenta? Inicia sesión</Link>
    </nav>
  </>
  )
}

export default NuevoPassword
```

Almacenando el nuevo Password

- Hay que validar el password. Creo un state para captar el input
- Tambien añado un onSubmit en el form con la función handleSubmit, le añado e.preventDefault( )
- Valido la extensión del password con password.length
- Para hacer la llamada a la base de datos tipo POST usaré un try catch, con await y axios
- Mostraré una alerta para notificar que el password se ha modificado correctamente

```
 const handleSubmit = async (e)=>{
    e.preventDefault()

   if(password.length <6){

      setAlerta({
        msg: "El password debe ser mínimo de 6 caracteres",
        error: true
      })
    }

    try {
      const url = `http://localhost:4000/api/usuarios/olvide-password/${token}`
      const {data} = await axios.post(url, {password})

      setAlerta({
        msg: data.msg,
        error: false
      })
```

```
    } catch (error) {
      setAlerta({
        msg: error.response.data.msg,
        error: true
      })
    }

  }
```

- Muestro el link de forma condicional con el state passwordModificado

```
  import {Link, useParams} from 'react-router-dom'
import {useState, useEffect} from 'react'
import axios from 'axios'
import Alerta from '../components/Alerta'

const NuevoPassword = () => {

  const {token} = useParams()

  const [password, setPassword]= useState('')
  const [passwordModificado, setPasswordModificado]= useState(false)
  const [tokenValido, setTokenValido] = useState(false)
  const [alerta,setAlerta] = useState({})


  useEffect(()=>{
    const comprobarToken= async()=>{
      try {
         await axios(`http://localhost:4000/api/usuarios/olvide-
password/${token}`)

        setTokenValido(true)
        setPasswordModificado(true)

      } catch (error) {

        setAlerta({
          msg: error.response.data.msg,
          error: true
        })
      }
    }
    comprobarToken()

  },[])

  const handleSubmit = async (e)=>{
    e.preventDefault()
```

```
    if(password.length <6){

      setAlerta({
        msg: "El password debe ser mínimo de 6 caracteres",
        error: true
      })
    }

    try {

      const url = `http://localhost:4000/api/usuarios/olvide-password/${token}`
      const {data} = await axios.post(url, {password})

      setAlerta({
        msg: data.msg,
        error: false
      })

    } catch (error) {
      setAlerta({
        msg: error.response.data.msg,
        error: true
      })
    }

  }

const {msg} = alerta

  return (
    <>
    <h1 className="text-sky-600 font-black text-6xl">Recupera tu password y
administra tus
    <span className="text-slate-700"> Proyectos</span></h1>

    {msg && <Alerta alerta={alerta} />}

    {tokenValido &&(

    <form className="bg-white my-10 shadow rounded-lg p-10" onSubmit=
{handleSubmit}>

      <div className="my-5">

        <label className="uppercase text-gray-600 block font-bold"
        htmlFor="password">Nuevo Password</label>

        <input
         id="password"
         type="password"
         value={password}
         onChange={(e)=>setPassword(e.target.value)}
         placeholder="Escribe tu nuevo password"
         className="w-full mt-3 p-2 border rounded-xl bg-gray-50 mb-3"/>
```

```
      </div>


      <input type="submit"
      value="Guardar password"
      className="bg-sky-700 mb-5 text-white w-full font-bold p-2 rounded-xl
uppercase hover:cursor-pointer hover:bg-sky-800 transition-colors"/>
    </form>


    )}


{passwordModificado &&
       <Link to="/" className="block text-slate-500 text-center my-5">Iniciar
sesión</Link>}
  </>
  )
}

export default NuevoPassword
```

NOTA: puedes configurar axios para no escribir una url tan larga todo el rato (opcional)

- creo una carpeta config en /src con clienteAxios.js

```
import axios from 'axios'

const clienteAxios = axios.create({
    baseUrl: "http://localhost:4000/api"
})

export default clienteAxios
```

Validando la autenticación

- Hago unas importaciones en el Login.jsx: Link, useState...(se ve mas abajo)
- Creo el state para el email, password y alerta. Asigno un handleSubmit al form
- Hago la validación y extraigo de la alerta con desestructuración el msg para mostrarla de forma condicional

```
import {Link} from 'react-router-dom'
import {useState} from 'react'
import {useNavigate} from 'react-router-dom'
```

```
import Alerta from '../components/Alerta'

const Login = () => {

  const [email, setEmail]= useState('')
  const [password, setPassword]= useState('')
  const [alerta, setAlerta]= useState({})

  const handleSubmit= async (e)=>{
    e.preventDefault()
    if([email, password].includes("")){
      setAlerta({
        msg: "Todos los campos son obligatorios",
        error: true
      })
      return
    }

  }

  const {msg} = alerta
  return (
    <>
      <h1 className="text-sky-600 font-black text-6xl">Inicia sesión y administra
tus
      <span className="text-slate-700"> Proyectos</span></h1>

      {msg && <Alerta alerta={alerta} />}

      <form className="bg-white my-10 shadow rounded-lg p-10"
        onSubmit={handleSubmit}>
        <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="email">Email</label>

          <input
           id="email"
           value={email}
           onChange={e=>setEmail(e.target.value)}
           type="email"
           placeholder="Email de registro"
           className="w-full mt-3 p-2 border rounded-xl bg-gray-50"/>
        </div>
        <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="password">Password</label>

          <input
           id="password"
           value={password}
           onChange={e=>setPassword(e.target.value)}
           type="password"
```

```
            placeholder="Tu password"
            className="w-full mt-3 p-2 border rounded-xl bg-gray-50 mb-3"/>
          </div>

          <input type="submit"
          value="Iniciar Sesión"
          className="bg-sky-700 mb-5 text-white w-full font-bold p-2 rounded-xl
uppercase hover:cursor-pointer hover:bg-sky-800 transition-colors"/>
        </form>

        <nav className="lg:flex lg:justify-between">
        <Link to="/registrar" className="block text-slate-500 text-center my-5">¿No
tienes cuenta? Registrate</Link>
        <Link to="/olvide-password" className="block text-slate-500 text-center my-
5">¿Olvidaste tu password?</Link>


        </nav>
      </>
    )
}

export default Login
```

- Si pasa la validación pasa al try catch donde se hará la llamada a la base de datos

Autenticando al usuario

- Una petición de tipo POST a usuarios/login
- Importo axios! Le paso los states cómo objeto, como segundo parametro despues de la url a axios

```
 const handleSubmit= async (e)=>{
    e.preventDefault()

    if([email, password].includes("")){

      setAlerta({
        msg: "Todos los campos son obligatorios",
        error: true
      })
      return
    }
    try {

      const {data} = await axios.post('http://localhost:4000/api/usuarios/login',
{email, password})

      console.log(data)
```

```
    } catch (error) {
      console.log(error.response.data.msg)
    }

  }
```

- console.log(data) me devuelve el usuario con su id y toda la info
- Si el usuario no existe me aparece en consola el mensaje de que no existe
- Creo el setAlerta en el error
- Voy a almacenar el token del inicio de sesion en el localStorage

```
import {Link} from 'react-router-dom'
import {useState} from 'react'
import {useNavigate} from 'react-router-dom'
import Alerta from '../components/Alerta'
import axios from 'axios'

const Login = () => {

  const [email, setEmail]= useState('')
  const [password, setPassword]= useState('')
  const [alerta, setAlerta]= useState({})

  const handleSubmit= async (e)=>{
    e.preventDefault()
    if([email, password].includes("")){
      setAlerta({
        msg: "Todos los campos son obligatorios",
        error: true
      })
      return
    }
    try {

      const {data} = await axios.post('http://localhost:4000/api/usuarios/login',
{email, password})

      localStorage.setItem('token', data.token)
    } catch (error) {
      setAlerta({
       msg: error.response.data.msg,
       error: true

      })
    }

  }

  const {msg} = alerta
  return (
    <>
```

```
      <h1 className="text-sky-600 font-black text-6xl">Inicia sesión y administra
tus
      <span className="text-slate-700"> Proyectos</span></h1>

      {msg && <Alerta alerta={alerta} />}

      <form className="bg-white my-10 shadow rounded-lg p-10"
        onSubmit={handleSubmit}>
        <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="email">Email</label>

          <input
           id="email"
           value={email}
           onChange={e=>setEmail(e.target.value)}
           type="email"
           placeholder="Email de registro"
           className="w-full mt-3 p-2 border rounded-xl bg-gray-50"/>
        </div>
        <div className="my-5">

          <label className="uppercase text-gray-600 block font-bold"
          htmlFor="password">Password</label>

          <input
           id="password"
           value={password}
           onChange={e=>setPassword(e.target.value)}
           type="password"
           placeholder="Tu password"
           className="w-full mt-3 p-2 border rounded-xl bg-gray-50 mb-3"/>
        </div>

        <input type="submit"
        value="Iniciar Sesión"
        className="bg-sky-700 mb-5 text-white w-full font-bold p-2 rounded-xl
uppercase hover:cursor-pointer hover:bg-sky-800 transition-colors"/>
      </form>

      <nav className="lg:flex lg:justify-between">
      <Link to="/registrar" className="block text-slate-500 text-center my-5">¿No
tienes cuenta? Registrate</Link>
      <Link to="/olvide-password" className="block text-slate-500 text-center my-
5">¿Olvidaste tu password?</Link>


      </nav>
    </>
  )
}

export default Login
```

- Lo próximo es CONTEXT