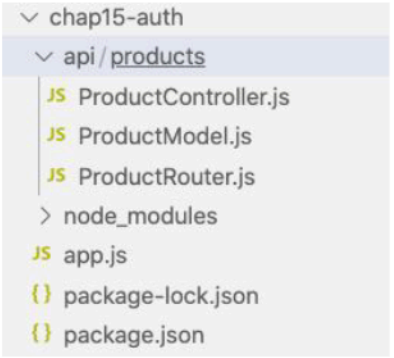
wawiwa

SEGURIDAD DE API

NODE.JS

wawiwa

Seguridad en API



Puedes trabajar en el proyecto pasado o crear una nueva carpeta (project)

Yo escogí crear un nuevo proyecto: entonces cree una carpeta llamada chap15-auth.
Crea una nueva carpeta llamada (selecciona un nombre) y corre npm init en la terminal y después npm install mongoose and npm install express.

Copia el archivo llamado app.js y copia la carpeta "api" del proyecto previo y pégala
Corre el servidor y verifica que todo funcione.
Después corre node app:

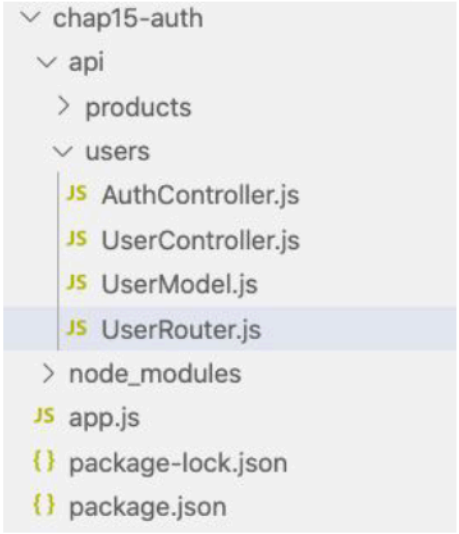
```
Running on port 3000
```

Crea un “User”

En la carpeta “api” crea una carpeta llamada “users”

En “users” crea 4 archivos

- UserModel.js
- UserController.js
- UserRouter.js
- AuthController.js



wawiwa

UserModel.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const UserSchema = new Schema({
   email: {
       type: String,
       unique: [true,'please provide email'],
       required: true,
       lowercase:true
   },
   password: {
       type: String,
       required: [true, 'please provide password'],
       minlength:6,
   },
   firstname: String,
   lastname: String,
   photo:String,
   created: Date,
   modified: Date,
   permission: {}
});
module.exports = mongoose.model('user', UserSchema);
```

Este es el esquema para un “user”

Después vamos a ver como usar el arreglo “permission”

wawiwa

UserController.js

```
const CurrentUser = require('./UserModel')
exports.createUser = async function(req, res, next) {
    try{
        let p1= req.body;
        const newItem = await CurrentUser.create(p1);
        res.status(201).json({
            status:"success",
            data:newItem
        })
    }
    catch(err){
        res.status(400).json({
            status:"fail",
            message:"error:😱" + err
        })
    }
};
```

wawiwa

Leer - UserController.js

```
//Leer por id
exports.getUserById =  function(req, res, next){
    let id = req.params.id
    CurrentUser.find({_id:id}).then(function(data){
        res.status(200).json({
            status:"success",
            data:data
        })
    }).catch(err=>{
        res.status(404).json({
            status:"fail",
            message:"error:😱" + err
        })
    })
}
```

wawiwa

Actualizar UserController.js

```
//actualizar
exports.updateUserById= function(req, res, next){
    let id = req.params.id
    CurrentUser.findByIdAndUpdate(id, req.body,
{new:true,runValidators:true})
    .then(function(data){
        res.status(200).json({
            status:"success",
            data:data
        })
    }).catch(err=>{
        res.status(404).json({
            status:"fail",
            message:"error:😱" + err
        })
    })
}
```

wawiwa

Borrar - UserController.js

```
exports.deleteUserById = function(req, res, next){
    let id = req.params.id
    CurrentUser.findByIdAndDelete(id)
    .then(function(data){
        res.status(404).json({
            status:"success",
            data:null
        })
    }).catch(err=>{
        res.status(404).json({
            status:"fail",
            message:"error:😱" + err
        })
    })
}
```

wawiwa

Obtener - UserController.js

```
//obtener
exports.getUsers = async function(req, res, next){
    //fase 1 - filtrar
    let queryObj = {...req.query}
    let withOutFileds = ['page', 'sort', 'limit', 'fields']
    withOutFileds.forEach(el => {
        delete queryObj[el]
    });
    //fase 2 - filtrado avanzado
    let strQuery = JSON.stringify(queryObj)
    strQuery = strQuery.replace(/\b(gte|gt|lte|lt)\b/g,match =>`$${match}`)
    queryObj = JSON.parse(strQuery)
    console.log(queryObj)
    let sort="";
    let selected = "";
    if(req.query.sort){
        sort = req.query.sort.split(',').join(' ')// add more sorts
    }
```

wawiwa

Obtener - UserController.js

```
    if(req.query.fields){
        selected = req.query.fields.split(',').join(' ')// show fields
    }
    let limit = req.query.limit || 100
    let page = req.query.page || 1
    let skip = (page-1)*limit
    let documents = await CurrentUser.countDocuments()
    if(skip>=documents){
        res.status(404).json({
            status:"fail",
            data:"no data on this page and limit"
        })
        return
    }
    CurrentUser.find(queryObj).skip(skip).limit(limit).select(selected).sort(sort).then(function(data)
    {
        res.status(200).json({
            status:"success",
            data:data
        })
    }).catch(err=>{
        res.status(404).json({
            status:"fail",
            message:"error:😱 :" + err
        })
    })
}
```

wawiwa

Registrar - AuthController.js

```
let User = require('./UserModel')
exports.signup = function(req, res, next) {
    const newUser = new User(req.body);
    newUser.created = new Date();
    newUser.modified = new Date();
    newUser.save().then(function(user) {
    res.status(201).json(
        {
            status:"success",
            user:user
        });
    }).catch(err=>{
        res.status(404).json({
            status:"fail",
            message:"error:😱" + err
    })
    })
}
```

wawiwa

UserRouter.js

```
const express = require('express')
const userRouter = express.Router();
const userController = require('./UserController')
const authController = require('./AuthController')
userRouter.post('/signup', authController.signup);
userRouter.post('/', userController.createUser);
userRouter.get('/:id', userController.getUserById);
userRouter.patch('/:id', userController.updateUserById);
userRouter.delete('/:id', userController.deleteUserById);
module.exports = userRouter;
```

registrar

wawiwa

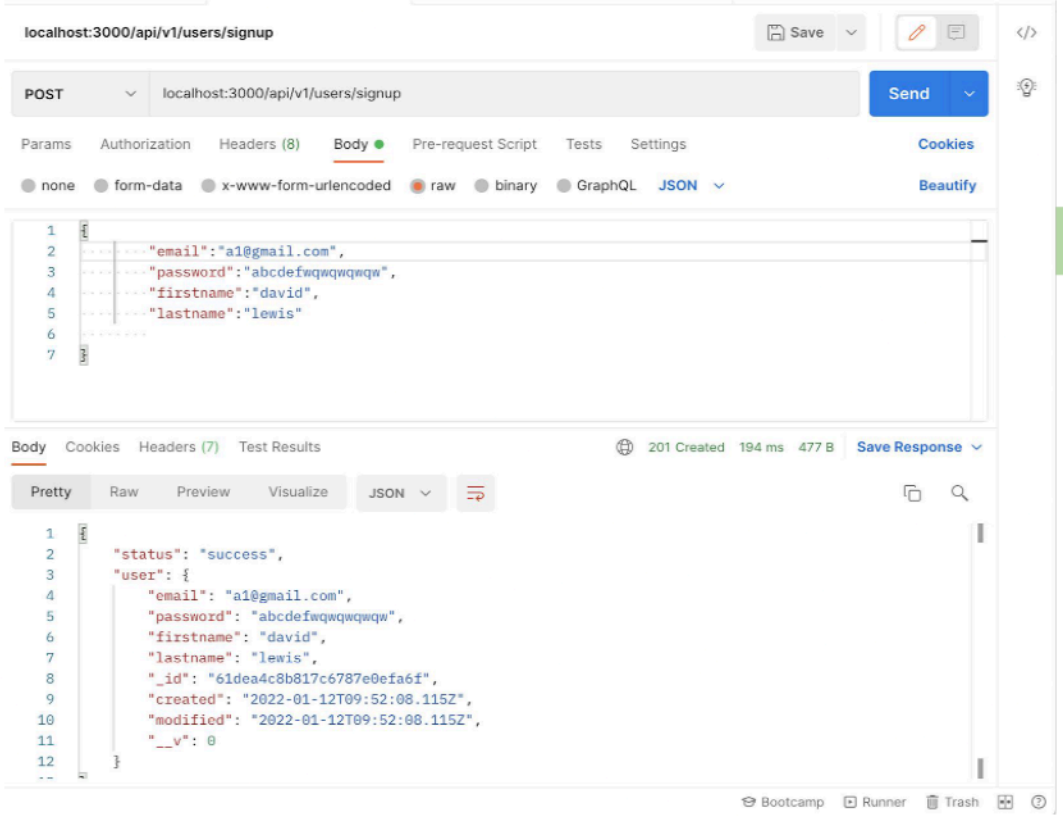
app.js

```
const express = require('express');
const app = express();
app.use(express.json())
const mongoose = require('mongoose');
const productRouter = require("./api/products/ProductRouter")
const userRouter = require("./api/users/UserRouter")
app.use('/api/v1/products', productRouter);
app.use('/api/v1/users',userRouter);
const strConnect =
"mongodb+srv://asaf:asaf@cluster0.hqfhv.mongodb.net/myFirstDatabase?retr
yWrites=true&w=majority";
const OPT = {
   useNewUrlParser: true
};
mongoose.connect(strConnect, OPT);
const port = process.env.PORT || 3000;
app.listen(port, function() {
   console.log("Running on port " + port);
})
```

Agregar el enrutamiento “user”

wawiwa

Corre el app y registra un “user”



localhost:3000/api/v1/users/signup

wawiwa

UserModel.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const UserSchema = new Schema({
   email: {
        type: String,
        unique: [true,'please provide email'],
        required: true,
        lowercase:true
   },
   password: {
        type: String,
        required: [true, 'please provide password'],
        minlength:6,
   },
   firstname: String,
   lastname: String,
   photo:String,
   created: Date,
   modified: Date,
   permission: {}
});
module.exports = mongoose.model('user', UserSchema);
```

Este es el esquema para un “user”

Después vamos a ver como user el arreglo “permission”

wawiwa

UserModel.js - agrega “hash” al “password”

```
UserSchema.pre('save', function(next) {
    if (!this.isModified('password')) {
        return next();
    }
    const salt = bcrypt.genSaltSync(12);
    this.password = bcrypt.hashSync(this.password, salt);
    next();
});

<!-- Continues in the next slide -->
Don't forget to add const bcrypt = require('bcryptjs')
```

Antes de llamar el método save(), está función será llamada

Verifica que el “password” no haya cambiado

Cifra el “password” (vamos a ver la implementación en la siguiente diapositiva)

For the implementation of encryptPassword() add:

```
const bcrypt = require('bcryptjs');
```

And run:

```
npm install bcryptjs
```

wawiwa

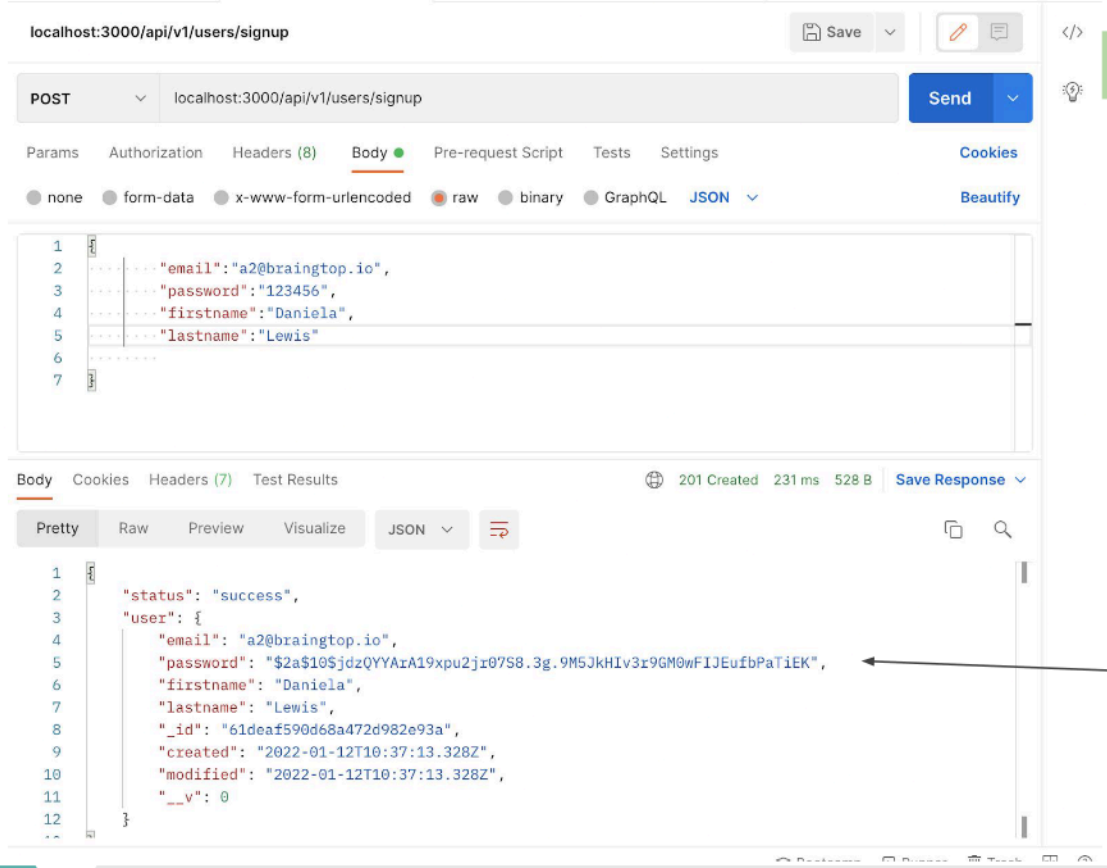
UserModel.js

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const bcrypt = require('bcryptjs')
const UserSchema = new Schema({
    email: {
        type: String,
        unique: [true,'please provide email'],
        required: true,
        lowercase:true
    },
    password: {
        type: String,
        required: [true, 'please provide password'],
        minlength:6,
    },
    firstname: String,
    lastname: String,
    photo:String,
    created: Date,
    modified: Date,
    permission: {}
});
module.exports = mongoose.model('user', UserSchema);
```

```
UserSchema.pre('save', function(next) {
    if (!this.isModified('password')) {
        return next();
    }
    const salt = bcrypt.genSaltSync(12);
    this.password = bcrypt.hashSync(this.password,
salt);
    next();
});
```

wawiwa

Corre el app y trata de registrar un usuario



localhost:3000/api/v1/users/signup

Password encrypted

Hazlo tú mismo

De acuerdo con las diapositivas 323 a la 339, agrega al “project” del capítulo anterior en la diapositiva 321:

En la carpeta api crea la carpeta llamada users y agrega los 4 archivos:

- UserModel.js
- UserController.js
- UserRouter.js
- AuthController.js

Adicionalmente agrega las opciones para registrarse:
api/v1/users/signup

Jwt - json web token https://jwt.io/

.JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre partes como un objeto JSON. Esta información se puede verificar y confiar porque está firmada digitalmente. Los JWT se pueden firmar mediante un secreto (con el algoritmo **HMAC**) o un par de claves pública/privada mediante **RSA** o **ECDSA**.

Aunque los JWT se pueden cifrar para proporcionar también secreto entre las partes, nos centraremos en los tokens *firmados*. Los tokens firmados pueden verificar la *integridad* de los reclamos contenidos en ellos, mientras que los tokens cifrados ocultan esos reclamos a otras partes. Cuando los tokens se firman utilizando pares de claves pública/privada, la firma también certifica que solo la parte que posee la clave privada es la que la firmó.

wawiwa

Jwt - json web token https://jwt.io/

¿Cuándo debes usar JSON Web Tokens?

Aquí hay algunos escenarios donde los JSON Web Tokens son útiles:

- **Autorización**: Este es el escenario más común para user JTW. Una vez que el usuario está conectado, cada solicitud subsecuente va a incluir el JWT, permitiendo al usuario acceder a rutas, servicios y recursos que están permitidos con ese token. “Single Sign On” es una funcionalidad que usa ampliamente JWT estos días, por su pequeña sobrecarga y su habilidad de ser usada en diferentes dominios.
- **Intercambio de información**: JSON Web Tokens son una manera segura de transmitir información entre partes. Porque JWTs pueden ser firmados —por ejemplo, usando pares de claves públicas/privadas—puedes estar seguro que los que mandan son quienes dicen que son. Adicionalmente, como la firma está calculada usando el “header” y el “payload”, también puedes verificar que el contenido no ha sido manipulado.

wawiwa

Encryption (cifrado)

Encryption es un medio **de asegurar datos digitales usando una o más técnicas matemáticas**, junto con la contraseña o la "clave" usada para descifrar la información. El proceso de cifrado traduce información usando un algoritmo que hace que la información original sea legible.

wawiwa

Jwt - json web token - https://jwt.io/

wawiwa

Add config.env

```
const configValues = {
    "uname": "asaf",
    "pwd": "asaf"
};
const config = {
    dev: 'development',
    test: 'testing',
    prod: 'production',
    port: process.env.PORT || 3000,
    //ten days in minutes
    expireTime: 60 * 60 * 1000,
    getDbConnectionString: function() {
        return '';
    },
    secrets: {
        jwt: process.env.JWT || "mysecret"
    },
};
module.exports = config;
```

Valores estáticos a verificar

Modos de ejecución de nuestra aplicación.

Puerto por defecto

Tiempo de expiración del token

Para que no tengamos que buscar el dbConnectionString lo ponemos en app.js cada vez que queramos conectar con la base de datos

El secreto para los tokens - el mío es un muy mal secretot🙃

wawiwa

Signup - AuthController.js

```
let User = require('./UserModel')
let jwt = require('jsonwebtoken')
let config = require('../../config')
exports.signup = function(req, res, next) {
    const newUser = new User(req.body);
    newUser.created = new Date();
    newUser.modified = new Date();
    newUser.save().then(function(user) {
    let token = jwt.sign({id:user._id},config.secrets.jwt,{expiresIn:config.expireTime})
    res.status(201).json(
        {
            status:"success",
            token,
            user:user
        });
    }).catch(err=>{
        res.status(404).json({
            status:"fail",
            message:"error:😱" + err
    })
    })
}
```

Agrega require jsonwebtoken

Agrega require config

Retorna en nuevo token por el dado id. En sign-in el usuario va a recibir un nuevo token con el tiempo de expiración

Respuesta token

wawiwa

UserModel autenticación en Login

Agrega estos `UserSchema.methods a UserModel`

```
UserSchema.methods = {
    authenticate: function(plainTextPword) {
        console.log("this password is " + this.password);
        return bcrypt.compareSync(plainTextPword, this.password);
    },
    toJson: function() {
        const obj = this.toObject();
        delete obj.password;
        return obj;
    }
};
```

Borrar el “password” del “user” antes de la respuesta

wawiwa

Authcontroller Login

Agrega esta `función login a AuthController.js`

```
exports.login = function(req, res, next){
    let {email, password} = req.body;
    if(!email || !password){
        res.status(400).send('you need email and password');
        return;
    }
    User.findOne({ email: email })
    .then(function(user) {
        if (!user) {
            res.status(401).send('No user with the given username');
            return
        }
```

Para si no existe un “user” con el “email” o “password” dados

Encuentra el “user” con el “email” dado en la base de datos

wawiwa

Authcontroller Login

```
else {
    if (!user || !user.authenticate(password)) {
            res.status(401).send('Wrong password');
            return
    }
    else{
            let token = signToken(user._id)
            res.status(200).json({
                status:"success",
                token:token
            });
        }
}
```

Verifica el "password"

Recupera el token

Responde el token al cliente

wawiwa

Authcontroller Login

Agrega esta `función signToken a AuthController.js`

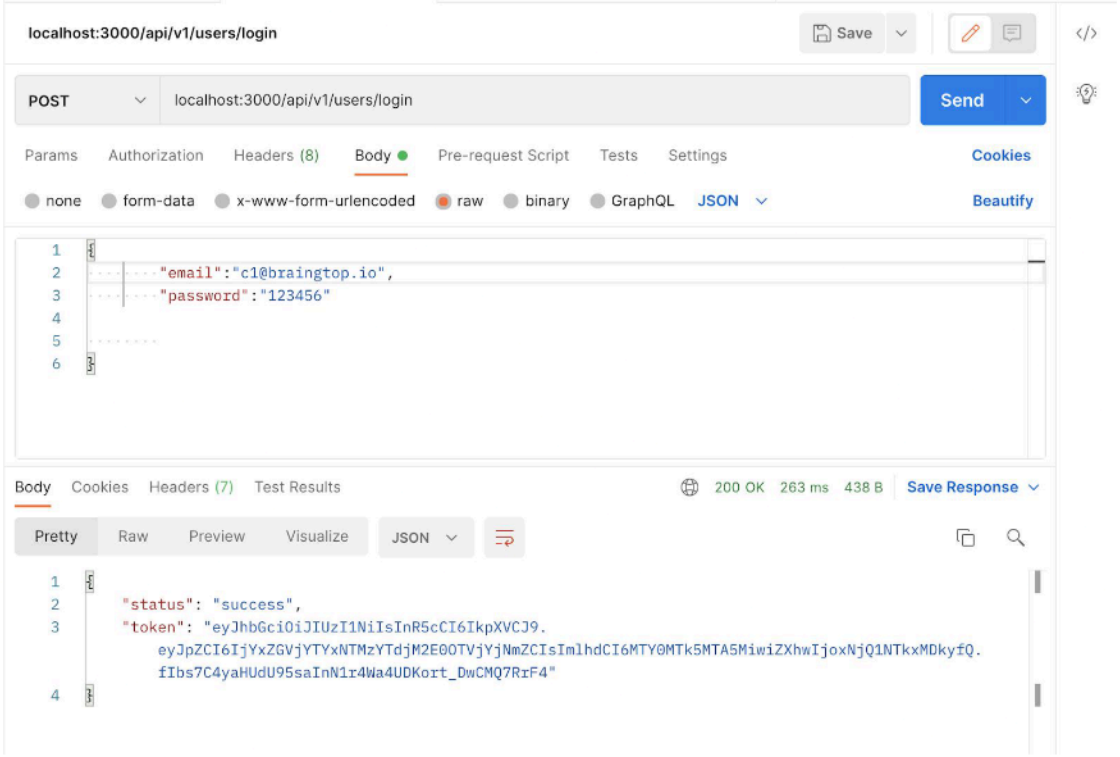
```
let signToken = id =>  {
    return jwt.sign({
            id: id
        },
        config.secrets.jwt, {
            expiresIn: config.expireTime
        }
    )
}
```

Fase 1 encuentra el “user”

wawiwa

Corre el app y trata de conectar un “user”

localhost:3000/api/v1/users/login



Hazlo tú mismo

De acuerdo con las diapositivas 340 hasta 351: Agrega a “project”

- config.js
- Login + token - api/v1/users/login

Adicionalmente, agrega las opciones para registrarse:
api/v1/users/signup

wawiwa

Proteger los datos - Agrega la función exports.protectSystem a AuthController.js

```
exports.protectSystem = async function(req, res, next){
    //fase 1 - obtén el token
    //fase 2 - verificación de token
    //fase 3 - comprobar si el "user" existe
    //fase 4 - comprobar si el "user" no cambió su "password"
}
```

wawiwa

Agrega el middleware protectSystem a productRouter.js

```
const express = require('express')
const productRouter = express.Router();
const productController = require('./ProductController')

const authController =
require('../users/AuthController')

productRouter.get('/',authController.protectSystem, productController.getProducts);

productRouter.post('/',productController.createProduct);
productRouter.get('/:id',productController.getProductById);
productRouter.patch('/:id',productController.updateProductById);
productRouter.delete('/:id',productController.deleteProductById);
productRouter.get('/get/statistic',productController.getStatistic);
module.exports = productRouter;
```

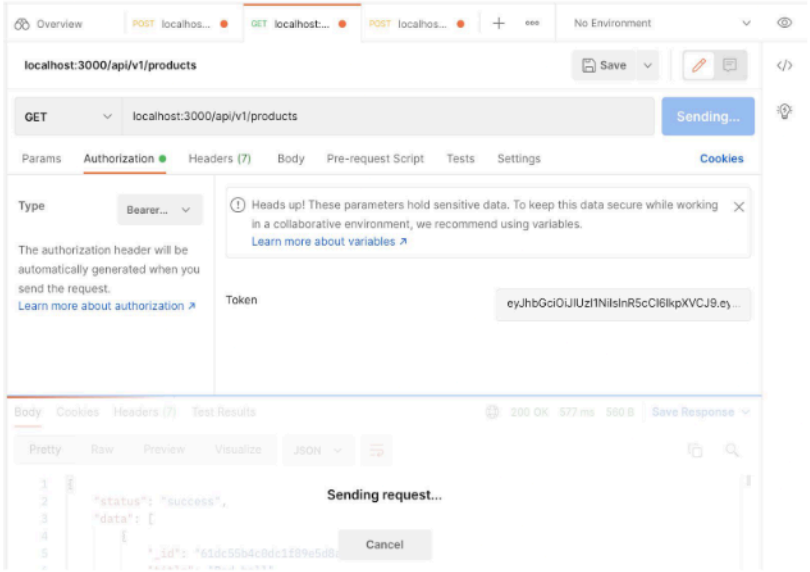
wawiwa

Proteger datos - Fase 1 obtener el Token

```
exports.protectSystem = async function(req, res, next){
    //fase 1 - get token
    let token = ""
    arrAuthorization = req.headers.authorization.split(' ')
    if(arrAuthorization[0]=='Bearer' && arrAuthorization[1])
        token = arrAuthorization[1]
    console.log(token)
    if(!token){
        res.status(401).json({fail:"You are not login again"})
        return
    }
}
```

wawiwa

localhost:3000/api/v1/products



token

```
Running on port 3000
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYxZTNkOWZiZGY3ZmU5ODhiMDJlNDRmNyIsImlhdCI6MTY0MjMyNDczMCwiZXhwIjoxNjQ1OTI0NzMwfQ.3ZLz5ugvVTu55
DBd5vbpcKVH0BS8kQJqwR076MTEgWU
```

wawiwa

Agrega “promisify” a AuthController.js - enlace

El método util.promisify() básicamente toma una función como input que sigue el estilo de callback de Node.js, en otras palabras, con un (err, value) y retorna una versión de lo mismo que retorna una promesa en lugar de un callback.

```
// Importar los módulos fs y util
const fs = require('fs');
const util = require('util');

// Leer el archivo usando una promesa e imprimiendo el texto
let file = util.promisify(fs.readFile);
file('./promisify.js', 'utf8') // Leyendo el mismo archivo
   .then((txt) => {
      console.log(txt);
   })

// Printing error if any
   .catch((err) => {
      console.log('Error', err);

});
```

En el ejemplo, hemos usado el módulo **fs** para leer archivos. Hemos usado el método **util.promisify()** para convertir el **fs.readFile** en un método basado en promesa. Ahora, el método arriba nos da una promesa en lugar de un callback.

wawiwa

Agrega “promisify” a AuthController.js - enlace

El método util.promisify() básicamente toma una función como input que sigue el estilo de callback de Node.js, en otras palabras, con un (err, value) y retorna una versión de lo mismo que retorna una promesa en lugar de un callback.

```
let User = require('./UserModel')
const {promisify} = require('util')
let jwt = require('jsonwebtoken')
let config = require('../../config')
```

wawiwa

Protegiendo datos - Agrega la función `exports.protectSystem a AuthController.js`

```
exports.protectSystem = async function(req, res, next){
    .
    .
     //fase 2 - verification token
    let decoded = ""
    try{
        decoded = await promisify(jwt.verify)(token, config.secrets.jwt)
        console.log(decoded)
    }
    catch(err){
        console.log(err)
        res.status(401).json({fail:"verification token failed please
login again:" + err})
        return
    }
}
```

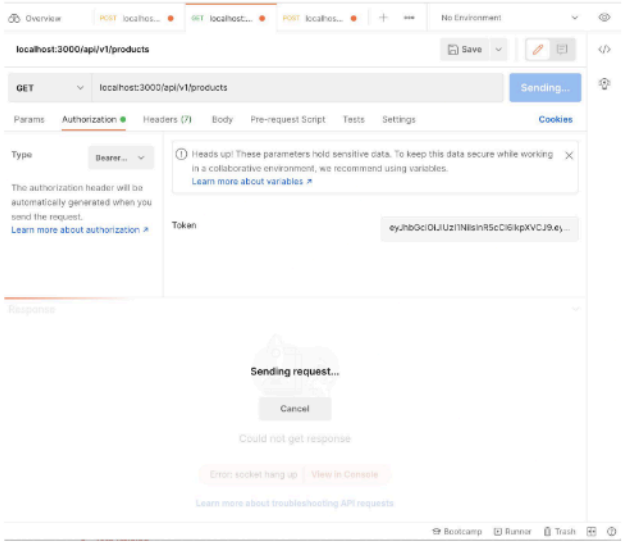
wawiwa

localhost:3000/api/v1/products

```
Running on port 3000
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYxZTNkOWZiZGY3ZmU5ODhiMDJlNDRmNyIsImlhdCI6MTY0MjMyNDczMCwiZXhwIjoxNjQ1OTI0NzMwfQ.3ZLz5ugvVTu55
DBd5vbpcKVH0BS8kQJqwR076MTEgWU
{ id: '61e3d9fbdf7fe988b02e44f7', iat: 1642324730, exp: 1645924730 }
```

User id decoficiado

token

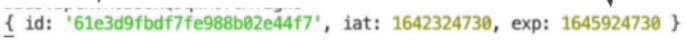


wawiwa

Protegiendo datos - fase 3 - comprobar si el usuario existe

User _id

User _id




```
exports.protectSystem = async function(req, res, next){
    .
    .
    .
    // fase 3 check if user exist
    const currentUser = await User.findById(decoded.id)
    console.log(currentUser)
    if(!currentUser){
        res.status(401).json({fail:"User not login please login
again"})
        return
    }
}
```

wawiwa

Protegiendo datos - fase 4 - Agrega **passwordChangedAt** a userModel.js

```
const UserSchema = new Schema({
    email: {
        type: String,
        unique: [true,'please provide email'],
        required: true,
        lowercase:true
    },
    password: {
        type: String,
        required: [true, 'please provide password'],
        minlength:6,
    },
    firstname: String,
    lastname: String,
    photo:String,
    created: Date,
    modified: Date,
    passwordChangedAt:Date,
    permission: {}
});
```

wawiwa

Protegiendo datos - fase 4 - Agrega userModel.js

```
UserSchema.methods = {
    authenticate: function(plainTextPword) {
        console.log("this password is " + this.password);
        return bcrypt.compareSync(plainTextPword, this.password);
    },
    changePasswordAfter:function(JWTTimeStamp){
        if(this.changePasswordAt){
            let changedTimeStamp = parseInt(this.passwordChangedAt.getTime()/1000)
            return JWTTimeStamp < changedTimeStamp
        }
        return false; //password didn't change
    },

    toJson: function() {
        const obj = this.toObject();
        delete obj.password;
        return obj;
    }
};
```

wawiwa

Protegiendo datos - protectSystem Fase 4

```
exports.protectSystem = async function(req, res, next){
    .
    .
    .

//fase 4 - check if user changed passwords
   if(currentUser.changePasswordAfter(decoded.iat))
   {
        res.status(401).json({fail:"User changed passwords, please login
again"})
        return
   }
   req.user = currentUser;
   next();
}
```

Adjunta user a req y llama next()

wawiwa

Permisos - Agrega “permissions” a UserModel.js

```
const UserSchema = new Schema({
    email: {
        type: String,
        unique: [true,'please provide email'],
        required: true,
        lowercase:true
    },
    password: {
        type: String,
        required: [true, 'please provide password'],
        minlength:6,
    },
    firstname: String,
    lastname: String,
    photo:String,
    created: Date,
    modified: Date,
    passwordChangedAt:Date,
    permission: {
        type:String,
        enum:['user', 'author', 'admin'],
        default:'user'
    }
});
```

wawiwa

Permisos - Agrega el middleware isAdmin

```
exports.isAdmin = function(req, res, next){
    if(req.user && req.user.permission &&
req.user.permission=='admin')
        next()
    else{
        res.status(401).json({
            message:'you don't have permission'
        })
    }
}
```

wawiwa

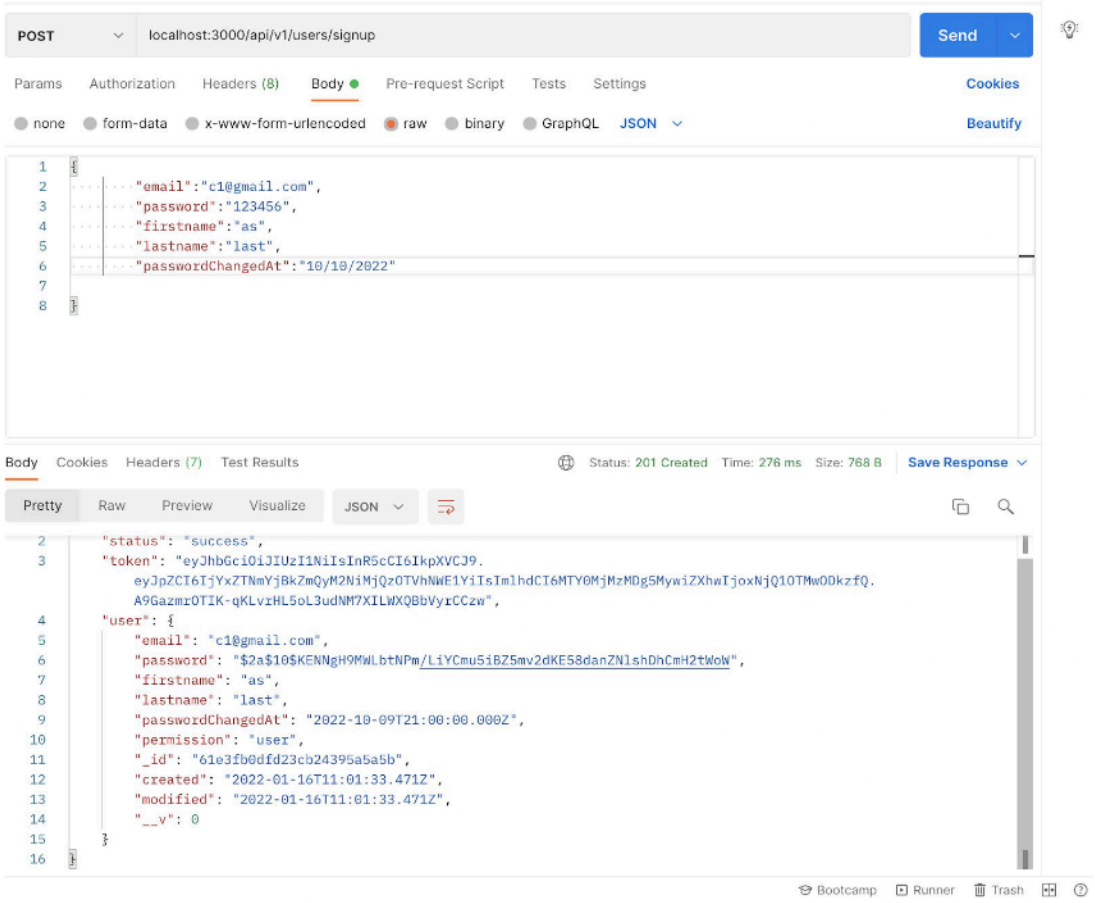
Permisos - Agrega el middleware isAdmin

Ahora solo los “users” que son “admin” pueden `getProducts`

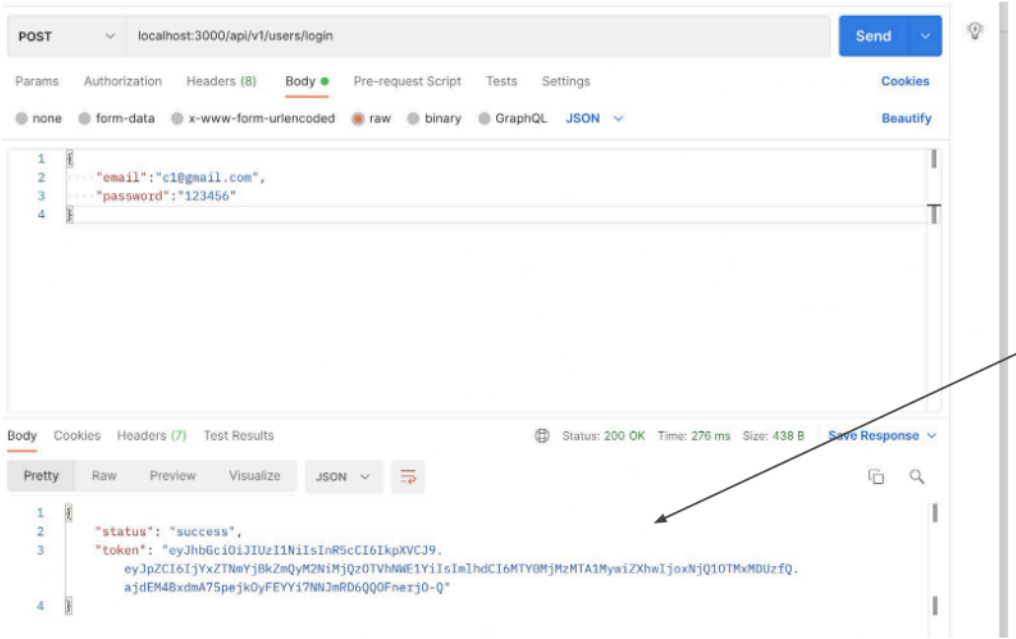
```
const express = require('express')
const productRouter = express.Router();
const productController = require('./ProductController')
const authController = require('../users/AuthController')
productRouter.get('/',authController.protectSystem,authController.isAdmin,
productController.getProducts);
productRouter.post('/',productController.createProduct);
productRouter.get('/:id',productController.getProductById);
productRouter.patch('/:id',productController.updateProductById);
productRouter.delete('/:id',productController.deleteProductById);
productRouter.get('/get/statistic',productController.getStatistic);
module.exports = productRouter;
```

wawiwa

Registrar un “user”



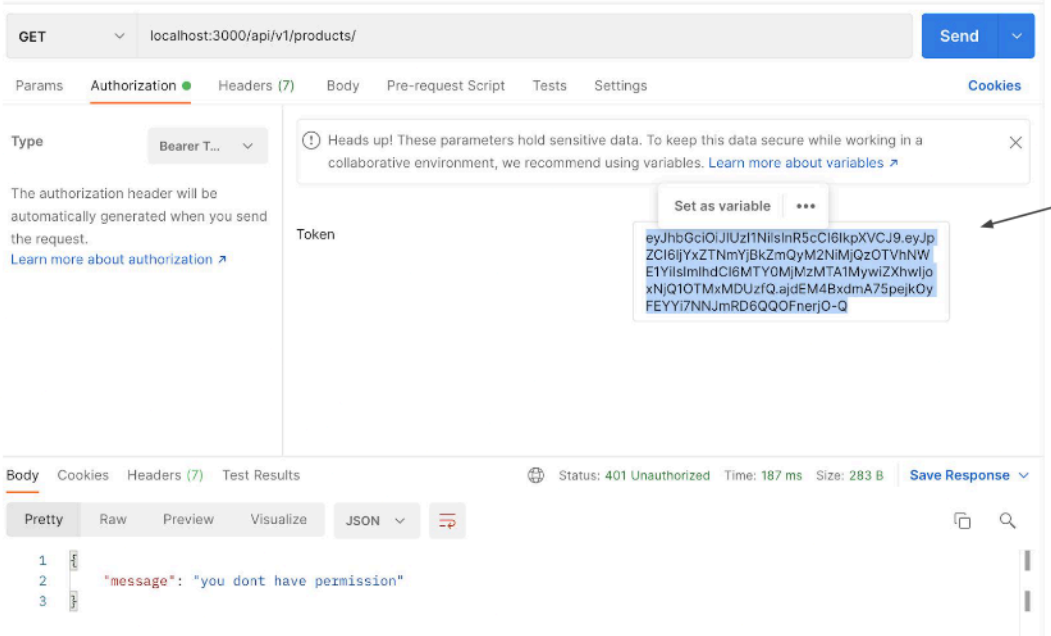
Conectar un “user”



Copia el token

wawiwa

Trata de obtener “products”



Pega el token

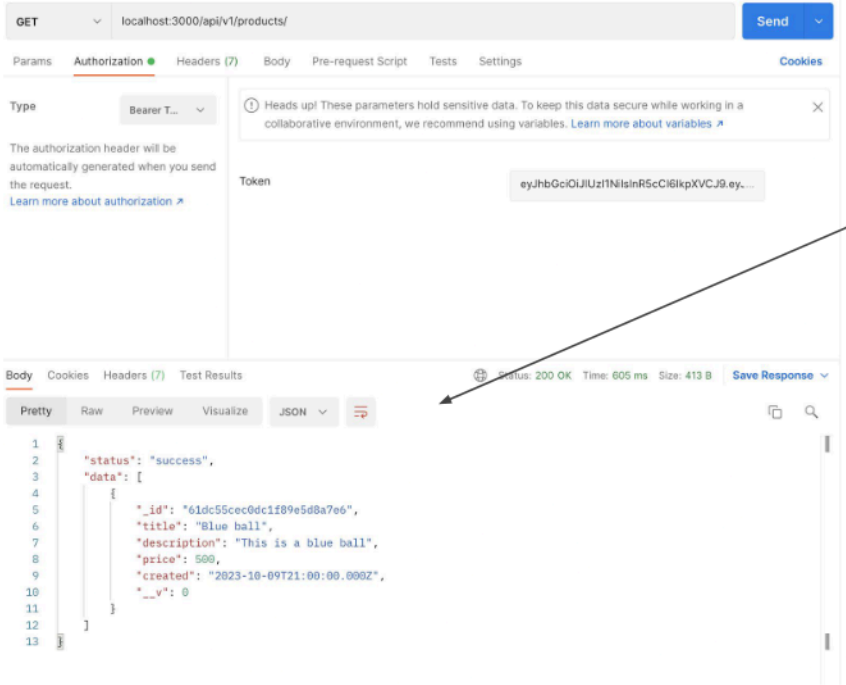
wawiwa

Change to admin

```
_id: ObjectId("61e3fb0dfd23cb24395a5a5b")
email: "c1@gmail.com"
password: "$2a$10$KENNgH9MWLbtNPm/LiYCmu5iBZ5mv2dKE58danZNlshDhCmH2tWoW"
firstname: "as"
lastname: "last"
passwordChangedAt: 2022-10-09T21:00:00.000+00:00
permission: "admin"
created: 2022-01-16T11:01:33.471+00:00
modified: 2022-01-16T11:01:33.471+00:00
__v: 0
```

wawiwa

Ahora tu tienes “permission”



Respuesta “products”

wawiwa

Hazlo tú mismo

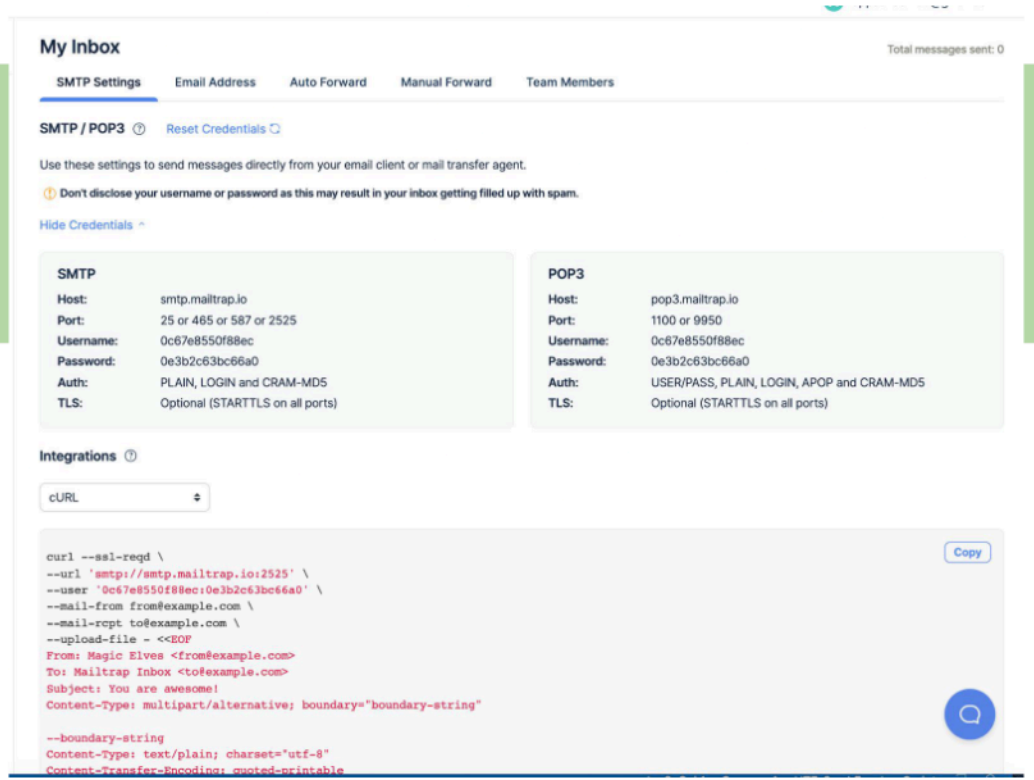
De acuerdo con las diapositivas 352 a 374. Agrega a “project” protect

- Solamente los “users” que son “admin” pueden borrar datos

wawiwa

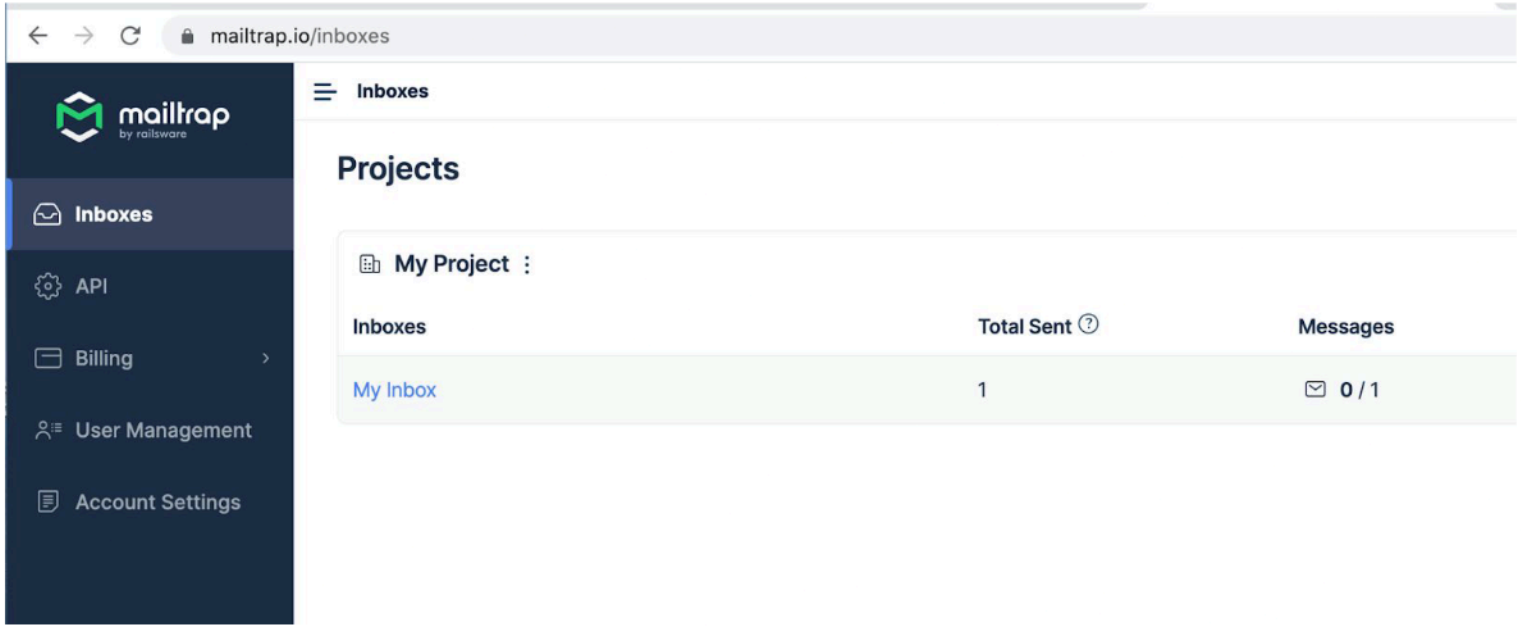
Regístrate en https://mailtrap.io/

After sign up you can see your
Host
Port
Username
Password



wawiwa

Ve a tu inbox



wawiwa

Contraseña olvidada

```
const config = {
    configValues : {
        "uname": "asaf",
        "pwd": "asaf"
    },
    EMAIL_USERNAME : "0c67e8550f88ec",
    EMAIL_PASSWORD : "0e3b2c63bc66a0",
    EMAIL_HOST : "smtp.mailtrap.io",
    EMAIL_PORT : 25,
    dev: 'development',
    test: 'testing',
    prod: 'production',
    port: process.env.PORT || 3000,
    //ten days in minutes
    expireTime: 60 * 60 * 1000,
    getDbConnectionString: function() {
        return '';
    },
    secrets: {
        jwt: process.env.JWT || "mysecret"
    },
};
module.exports = config;
```

Agrega a config.js

EMAIL_USERNAME,EMAIL_PASSWORD,EMAIL_HOST,EMAIL_PORT

wawiwa

Módulo Email.js

```
let nodemailer = require('nodemailer')
let config = require('../../config')
const sendEmail = async options=>{
   let transporter = nodemailer.createTransport({
       host:config.EMAIL_HOST,
       port:config.EMAIL_PORT,
       auth:{
           user:config.EMAIL_USERNAME,
           pass:config.EMAIL_PASSWORD
       }
   })
   const mailOptions = {
       from:'<test mail>no-reply',
       to:options.email,
       subject:options.subject,
       text:options.message
   }
  await transporter.sendMail(mailOptions)
}
module.exports = sendEmail
```

Agrega “send email” a este módulo y expórtalo.

wawiwa

AuthController.js export.forgotPassword

```
exports.forgotPassword = async function(req, res, next){
    //fase 1 - obtén user por su email
    let user = await User.findOne({email:req.body.email})
    if(!user){
        res.status(404).json({
            message:'please send email'
        })
        return
    }
```

wawiwa

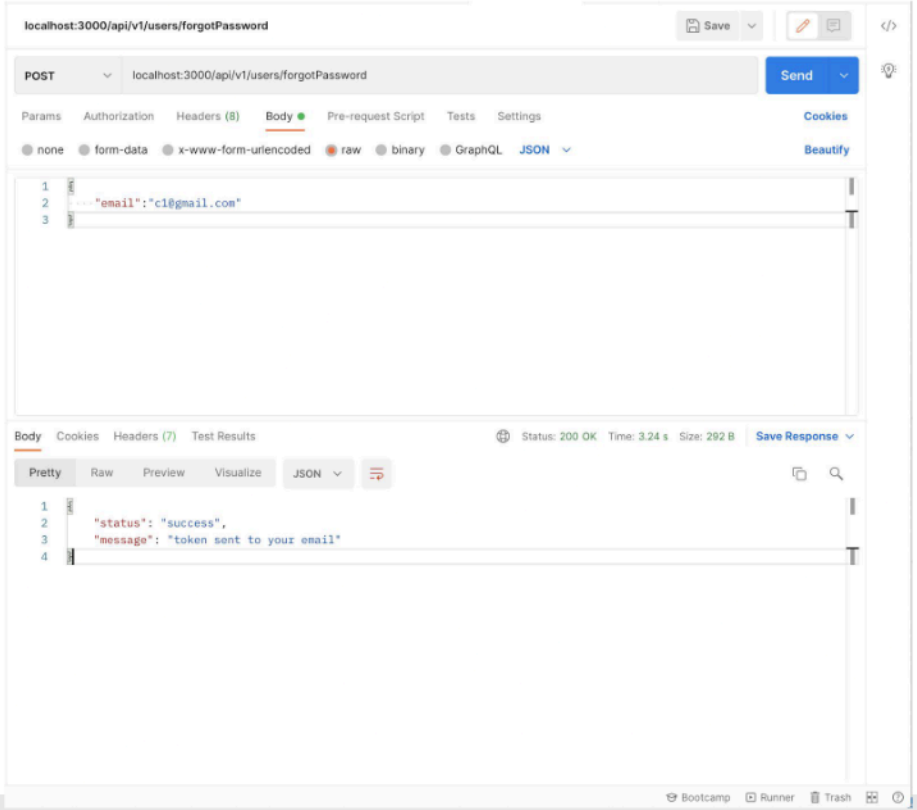
AuthController.js export.forgotPassword

```
    try{
    //fase 2 - crea un "reset token" aleatorio
    let resetToken = user.createNewPasswordToken()
    await user.save({validateBeforeSave:false})
    //fase 3 - mandalo al email del "user"
    let resetUrl = req.protocol + ":/" + req.get('host')
+"/api/v1/users/resetPassword/" + resetToken
    let message = "click here to make new password " + resetUrl
    await sendEmail({email:user.email, subject:'your password reset token ',
message})
    res.status(200).json({status:"success", message:'token sent to your email'})
    }
    catch(err){
        user.passwordResetToken=undefined
        user.passwordResetExpires = undefined
        await user.save({validateBeforeSave:false})
        res.status(500).json({status:"failed", message:'error sending email'})
    }
}
```

wawiwa

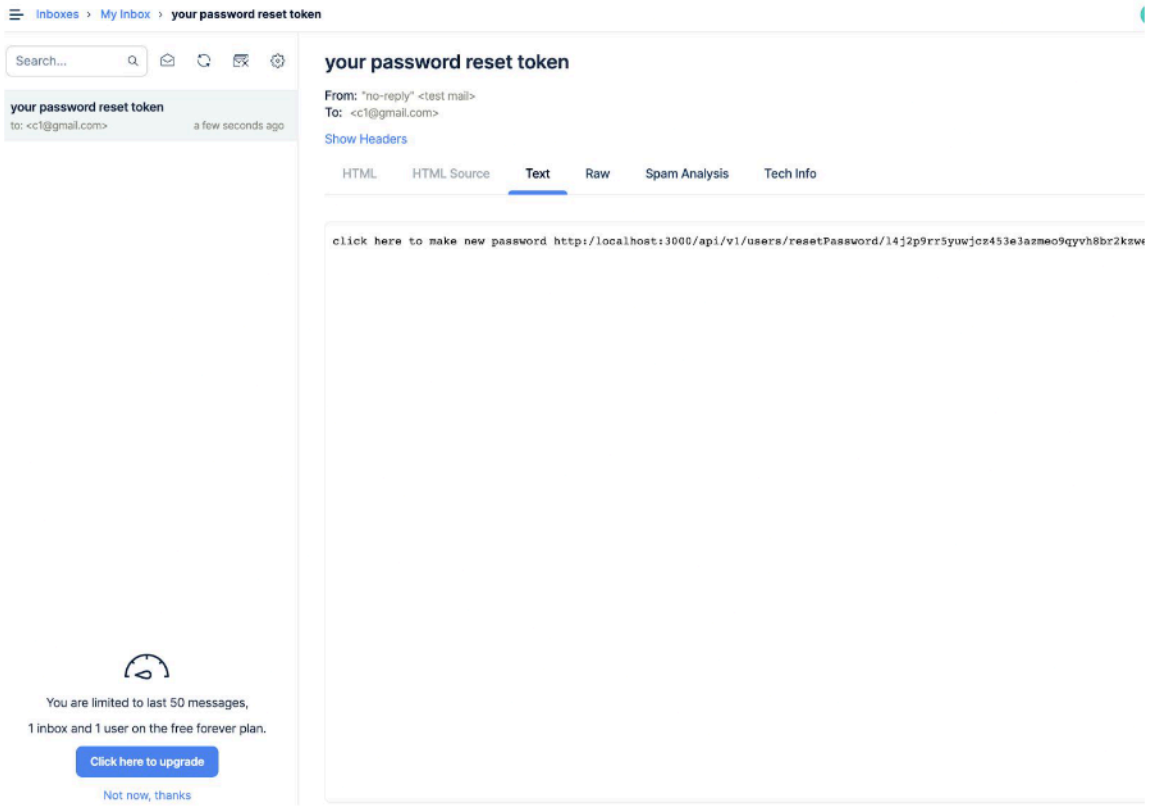
localhost:3000/api/v1/users/forgotPassword

Manda el token al email



wawiwa

Acepta el email



wawiwa

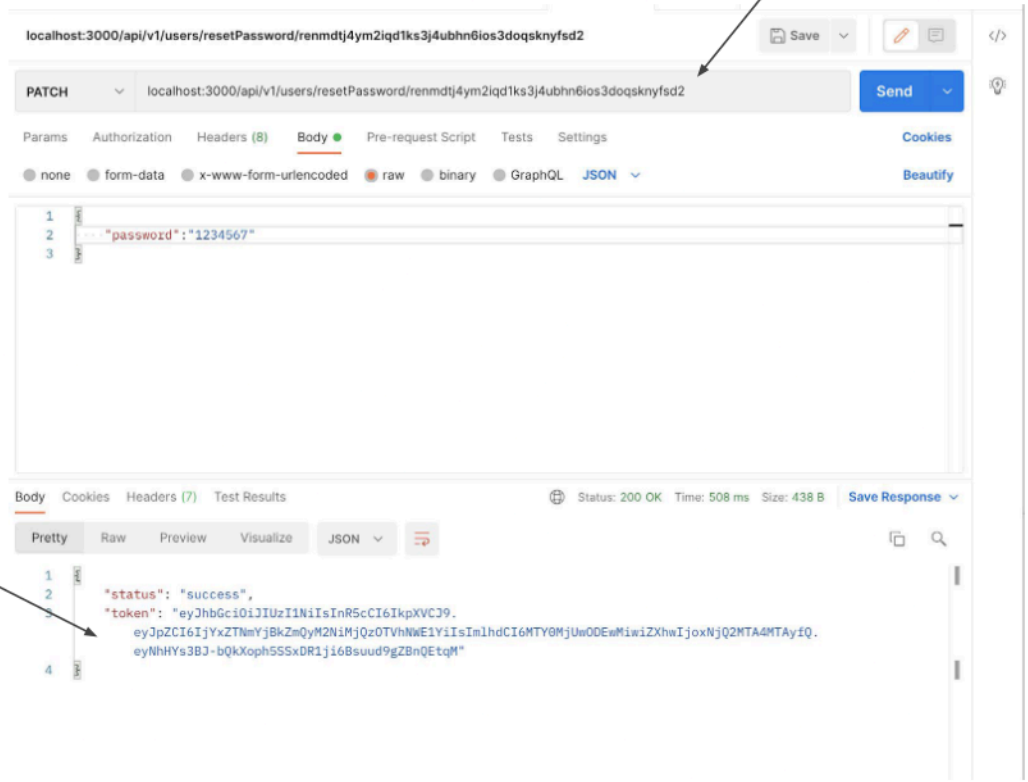
Reestablecer el password

```
exports.resetPassword= async function(req, res, next){
   //fase 1 - obtener user
    let user = await User.findOne(
        {
            passwordResetToken:req.params.token,
            passwordResetExpires:{$gt:Date.now()}
        })
    //fase 2 - verificar si el token no ha expirado
    if(!user){
        res.status(404).json({status:"failed", message:'invalid token'})
    }
    user.password = req.body.password
    //fase 3
    user.passwordResetToken=undefined
    user.passwordResetExpires=undefined
    await user.save()
    let token = signToken(user._id)
    res.status(200).json({
        status:"success",
        token:token
    });
}
```

wawiwa

Mandar reestablecer contraseña

localhost:3000/api/v1/users/resetPassword/renmdtj4ym2iqd1ks3j4ubhn6ios3doqsknyfsd2



Nuevo token

wawiwa

Actualizar mi “password”

```
exports.updatePassword =async function (req, res, next){
    //obtener user
    let user = await User.findById(req.user._id);
    //verficiar password actual
    if(!(await user.matchPassword(req.body.currentPassword)))
    {
        req.status(401).json({
            status:"failed",
            message:"password not correct"
        })
        return
    }
    //si el password es correcto, actualizalo
    user.password = req.body.password
    await user.save()
    //conecta el usuario mandando el token jwt
    let token = signToken(user._id)
    res.status(200).json({
        status:"success",
        token:token
    });
}
```

wawiwa

Actualizar mi “password”

```
const express = require('express')
const userRouter = express.Router();
const userController = require('./UserController')
const authController = require('./AuthController')

userRouter.post('/signup', authController.signup);
userRouter.post('/login', authController.login);
userRouter.post('/forgotPassword', authController.forgotPassword);
userRouter.patch('/resetPassword/:token', authController.resetPassword);
userRouter.patch('/updateMyPassword',authController.protectSystem,authController.updatePasswo
rd);


userRouter.post('/',userController.createUser);
userRouter.get('/:id',userController.getUserById);
userRouter.patch('/:id',userController.updateUserById);
userRouter.delete('/:id',authController.isAdmin,userController.deleteUserById);
module.exports = userRouter;
```

wawiwa

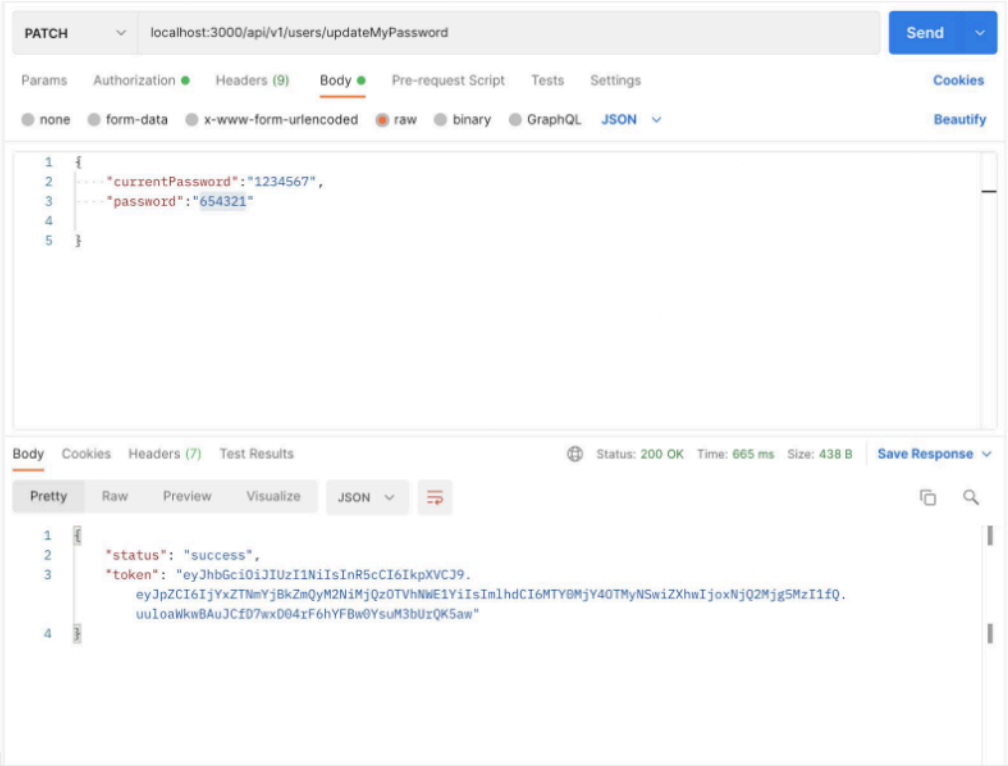
Agrega esta función a los métodos “model”

```
UserSchema.methods = {
    authenticate: function(plainTextPword) {
        console.log("this password is " + this.password);
        return bcrypt.compareSync(plainTextPword, this.password);
    },
    changePasswordAfter :function(JWTTimeStamp){
        if(this.changePasswordAt){
            let changedTimeStamp = parseInt(this.passwordChangedAt.getTime()/1000)
            return JWTTimeStamp < changedTimeStamp
        }
        return false; //password dont changed
    },
    createNewPasswordToken :function(){
        let str ="abcdefghijklmnopqrstuvwzyz1234567890"
        this.passwordResetToken =""
        for(const i=0;i<40;i++){
            let n = Math.floor(Math.random() * (str.length - 1));
            this.passwordResetToken +=str[n]
        }
        this.passwordResetExpires = Date.now() + 30*60*1000 // half hour
        return this.passwordResetToken;
    },
    matchPassword:function(pass){
        if(bcrypt.compareSync(pass, this.password))
            return true
        return false
    },
    toJson: function() {
        const obj = this.toObject();
        delete obj.password;
        return obj;
    }
```

Verifica si el usuario manda su “password”, antes de cambiarlo.

wawiwa

localhost:3000/api/v1/users/updateMypassword



wawiwa

Actualiza la información de “user”

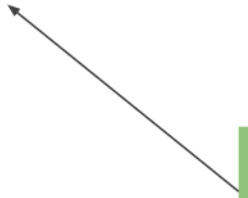
```
exports.updateMe = async function(req, res, next){
    // crea error si el “user” manda el password
    if(req.body.password)
    {
        res.status(400).json({
            status:"failed",
            message:"can't update password from here"
        });
        return
    }
    //actualizar “user”
    let filterBody = allowedObj(req.body, 'firstname', 'lastname','email')
    let user = await User.findByIdAndUpdate(req.user._id,filterBody,
        {new:true, runValidators:true})
    res.status(200).json({
        status:"success",
        user:user
    });
}
```

El llenado que permitiste actualizar. Por ejemplo, no quieres que el usuario actualice el token o el permiso.

wawiwa

Actualiza la información de “user”

```
let allowedObj = function(obj, ...allowedFields){
    let newobj = {}
    Object.keys(obj).forEach(el =>{
        if(allowedFields.includes(el))
            newobj[el] = obj[el]
    });
    return newobj
}
```



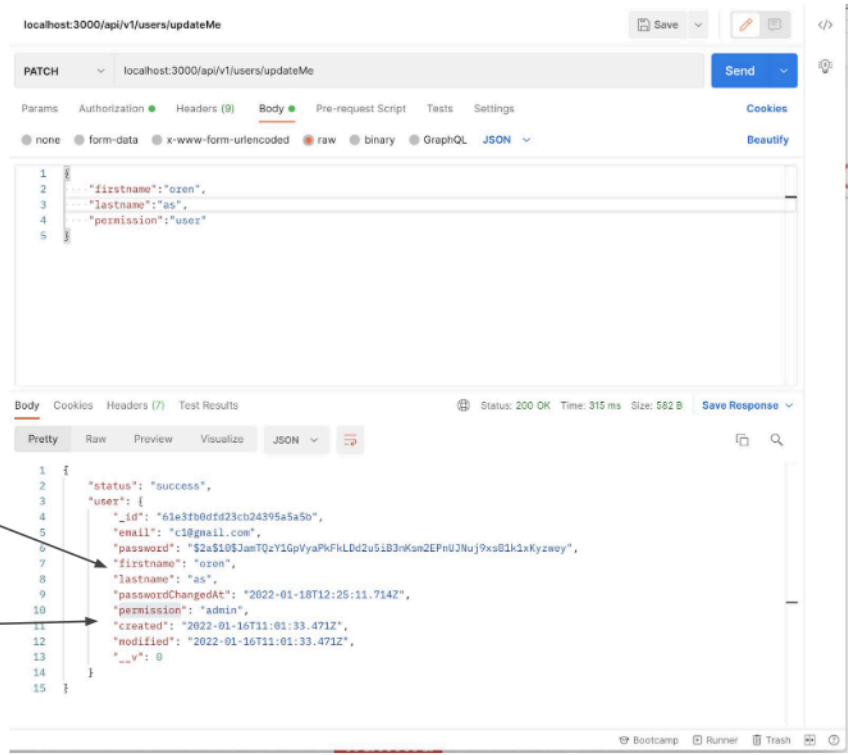
Retorna objeto permitido actualizado.

wawiwa

localhost:3000/api/v1/users/updateMe

Firstname y last name cambiaron

Permission no se actualiza



wawiwa

Borrar “user” - configura “active” a false

```
const UserSchema = new Schema({
    email: {
        type: String,
        unique: [true,'please provide email'],
        required: true,
        lowercase:true
    },
    password: {
        type: String,
        required: [true, 'please provide password'],
        minlength:6,
    },
    firstname: String,
    lastname: String,
    photo:String,
    created: Date,
    modified: Date,
    passwordChangedAt:Date,
    permission: {
        type:String,
        enum:['user', 'author', 'admin'],
        default:'user'
    },
    passwordResetToken:String,
    passwordResetExpires:Date,
    active:{
        type:Boolean,
        default:true
    }
});
```

Agrega la propiedad “active” al “user”

wawiwa

Agrgea la función “pre” para el esquema del “user”

```
UserSchema.pre(/^find/, function(next){
    this.find({active:true})
    next()
})
```

wawiwa

Agrega la función deleteMe a authcontroller.js

```
exports.deleteMe = async function(req, res, next){
    //borra user
    let user = await
User.findByIdAndUpdate(req.user._id,{active:false})
    res.status(204).json({
        status:"success",
        user:null
    });
}
```

wawiwa

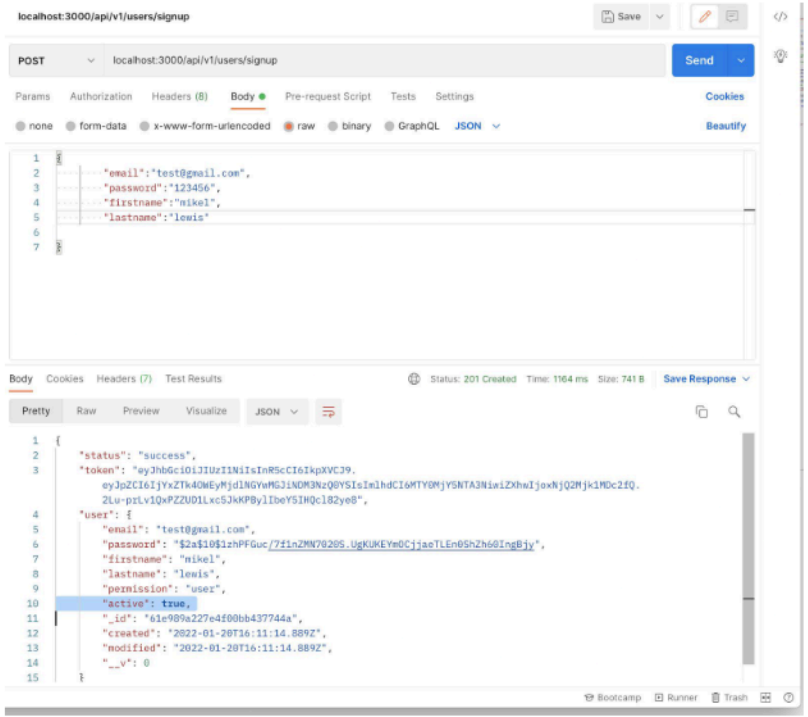
Agrega la función deleteMe a UserRouter.js

```
const express = require('express')
const userRouter = express.Router();
const userController = require('./UserController')
const authController = require('./AuthController')

userRouter.post('/signup', authController.signup);
userRouter.post('/login', authController.login);
userRouter.post('/forgotPassword', authController.forgotPassword);
userRouter.patch('/resetPassword/:token', authController.resetPassword);
userRouter.patch('/updateMyPassword',authController.protectSystem,authController.updatePassword);
userRouter.patch('/updateMe',authController.protectSystem,authController.updateMe);
userRouter.patch('/deleteMe',authController.protectSystem,authController.deleteMe);
userRouter.post('/',userController.createUser);
userRouter.get('/:id',userController.getUserById);
userRouter.patch('/:id',userController.updateUserById);
userRouter.delete('/:id',authController.isAdmin,userController.deleteUserById);
module.exports = userRouter;
```

wawiwa

localhost:3000/api/v1/users/signup



wawiwa

localhost:3000/api/v1/users/deleteMe


```
_id: ObjectId("61e989a227e4f00bb437744a")
email: "test@gmail.com"
password: "$2a$10$1zhPFGuc/7f1nZMN7020S.UgKUKEYmOCjjaeTLEn0ShZh60IngBjy"
firstname: "mikel"
lastname: "lewis"
permission: "user"
active: false
created: 2022-01-20T16:11:14.889+00:00
modified: 2022-01-20T16:11:14.889+00:00
__v: 0
```


wawiwa
¿Preguntas?