

# NODE.JS

# SEGURIDAD DE API

wawiwa

# Seguridad en API

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

```
✓ chap15-auth
  ✓ api / products
    ✓ ProductController.js
    ✓ ProductModel.js
    ✓ ProductRouter.js
  > node_modules
    ✓ app.js
    {} package-lock.json
    {} package.json
```

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



# 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"

# 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
    })
  }
};
```

# 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
    })
  })
}
```

# 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
      })
    })
}
```

# 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
      })
    })
}
```

# Obtener - UserController.js

```
//obtener
exports.getUsers = async function(req, res, next) {
  //fase 1 - filtrar
  let queryObj = { ...req.query }
  let withOutFields = ['page', 'sort', 'limit', 'fields']
  withOutFields.forEach(el => {
    delete queryObj[el]
  });
  //fase 2 - filtrado avanzado
  let strQuery = JSON.stringify(queryObj)
  strQuery = strQuery.replace(/\b(gt|lt|te|llt)\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
  }
}
```

# 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
        })
    })
})
```

# 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
    })
  })
}
```

# UserRouter.js

```
const express = require('express')

const userRouter = express.Router()

const userController = require('./UserController')

const authController = require('./AuthController')

userRouter.post('/signup', authController.signup) ; ← registrar

userRouter.post('/:id', userController.createUser) ;

userRouter.get('/:id', userController.getUserById) ;

userRouter.patch('/:id', userController.updateUserById) ;

userRouter.delete('/:id', userController.deleteUserById) ;

module.exports = userRouter;
```

# 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"

# Corre el app y registra un “user”

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** `localhost:3000/api/v1/users/signup`
- Body (JSON):**

```
1 "email": "a1@gmail.com",
2 "password": "abcdeFgFgFgFgFg",
3 "firstname": "david",
4 "lastname": "lewis"
```
- Headers (18):** Authorization, Headers (18), Body (green dot), Pre-request Script, Tests, Settings, Cookies, x-www-form-urlencoded, raw, binary, GraphQL, JSON (selected), Beauty.
- Response:**

```
1 "status": "success",
2 "user": {
3   "email": "a1@gmail.com",
4   "password": "abcdefghijklmnopqrstuvwxyz",
5   "firstname": "david",
6   "lastname": "lewis",
7   "id": "61de4c8bb317e6787e0efaf1",
8   "created": "2022-01-12T09:52:08.115Z",
9   "modified": "2022-01-12T09:52:08.115Z",
10  "v": 0
11 }
12 }
```
- Request Headers:** Content-Type: application/json
- Response Headers:** Content-Type: application/json, Date: Mon, 13 Mar 2022 09:52:08 GMT, Connection: keep-alive, Transfer-Encoding: chunked, Status: 201 Created, Vary: Accept, Cache-Control: no-store, Pragma: no-cache, X-Powered-By: Express, X-Runtime: 0.000 ms
- Timings:** 194 ms
- Save Response:** A green button labeled "localhost:3000/api/v1/users/signup".

# 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"

# 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();  
});
```

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)

<!-- Continues in the next slide -->  
Don't forget to add `const bcrypt = require('bcryptjs');`  
For the implementation of encryptPassword() add:  
`const bcrypt = require('bcryptjs');`

And run:

```
npm install bcryptjs
```

# UserModel.js

```
UserSchema.pre('save', function(next) {
  if (!this.isModified('password')) {
    return next();
  }
  const salt = bcrypt.genSaltSync(12);
  this.password = bcrypt.hashSync(this.password,
    salt);
  next();
},
{
  type: String,
  unique: [true, 'please provide email'],
  required: true,
  lowercase:true
},
{
  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);
```

# Corre el app y trata de registrar un usuario

The screenshot shows a POST request to `localhost:3000/api/v1/users/signup`. The request body is a JSON object:

```
1 {  
2   "email": "a2@braingtop.io",  
3   "password": "123456",  
4   "firstname": "Daniela",  
5   "lastname": "Lewis"  
6 }  
7
```

The response is a JSON object:

```
1 {  
2   "status": "success",  
3   "user": {  
4     "email": "a2@braingtop.io",  
5     "password": "$2a$05$dzQYArA19xpU2j07SS8..3g.9M5JkHLv3r9GM0mFLJElfbPaIEK",  
6     "firstname": "Daniela",  
7     "lastname": "Lewis",  
8     "id": "61dcaaff90668a472d082293a",  
9     "Created": "2022-01-12T10:37:13.328Z",  
10    "modified": "2022-01-12T10:37:13.328Z",  
11    "v": 0  
12 }
```

A green box highlights the response body with the text "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.

# Jwt - json web token <https://jwt.io/>

## ¿Cuándo debes usar JSON Web Tokens?

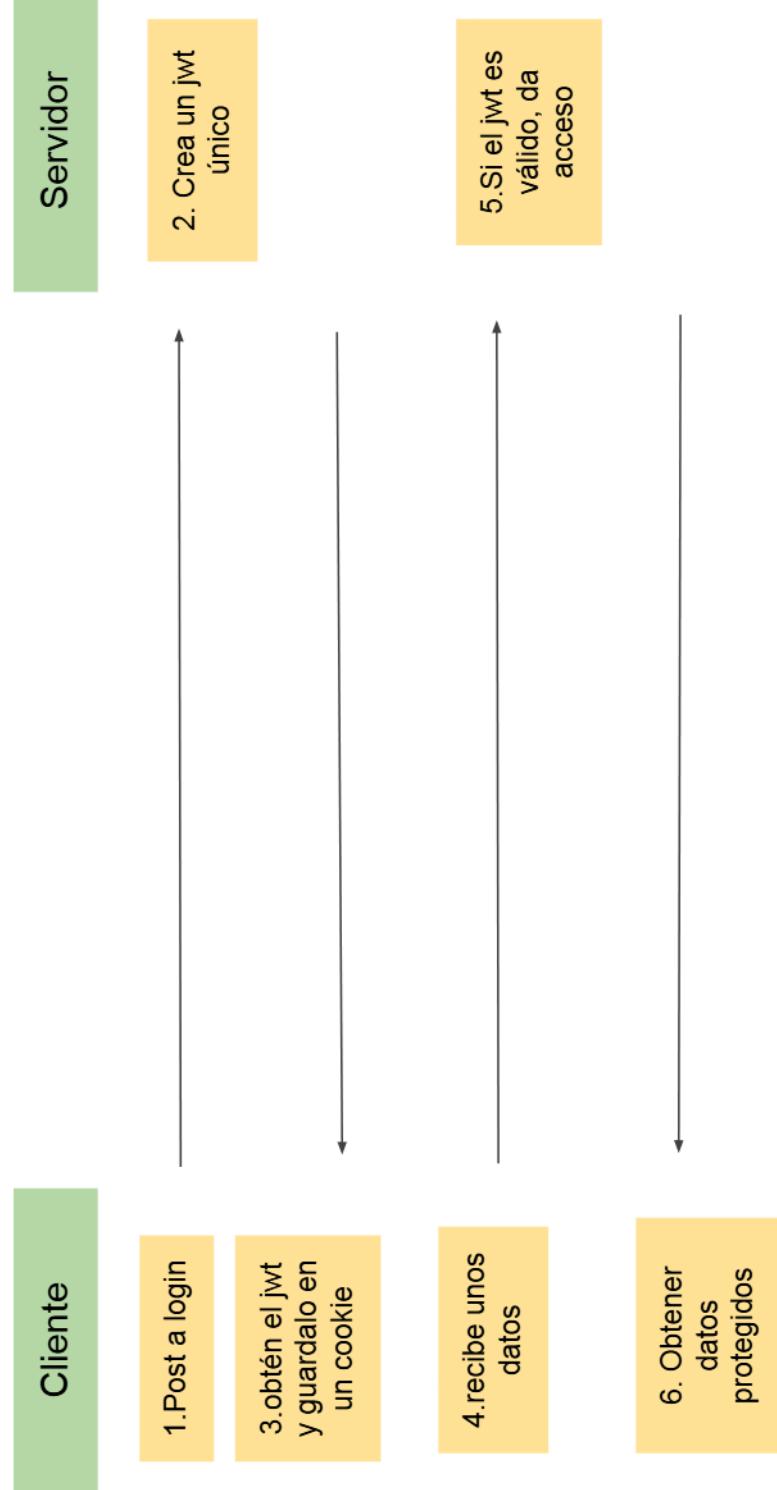
Aquí hay algunos escenarios donde los JSON Web Tokens son útiles:

- **Autorización:** Este es el escenario más común para user JWT. Una vez que el usuario está conectado, cada solicitud sucesiva 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.

# 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.

# Jwt - json web token - <https://jwt.io/>



# 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 secreto 😊

# 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

Resuesta token

# 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

# 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

# Authcontroller Login

```
Verifica el "password"  
→  
  
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  
        });  
    }  
}
```

Recupera el token

Responde el token al cliente

# 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”

# Corre el app y trata de conectar un “user”

The screenshot shows a POST request to `localhost:3000/api/v1/users/login`. The request body is JSON with fields `email` and `password`. The response is a successful JSON object containing a token.

**Request Body:**

```
1 "email": "c1@brainstop.io",
2 "password": "123456"
```

**Response Body:**

```
1 "status": "success",
2 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.
3 eyJpZC1oIjyxZCvJYTxNM2YtdjM2E0tVJYiWnZCTsImIhdCI6MTY0MTLSMTASMiwiZXhwIjoxNjQ1MTkxMDkyfQ.
4 fB57C4yahUDUs5saInM1r4wA4UDKoxt_DwCMq7rfF4"
```

## 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

# 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"  
}
```

# Agrega el middleware protectSystem a

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

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

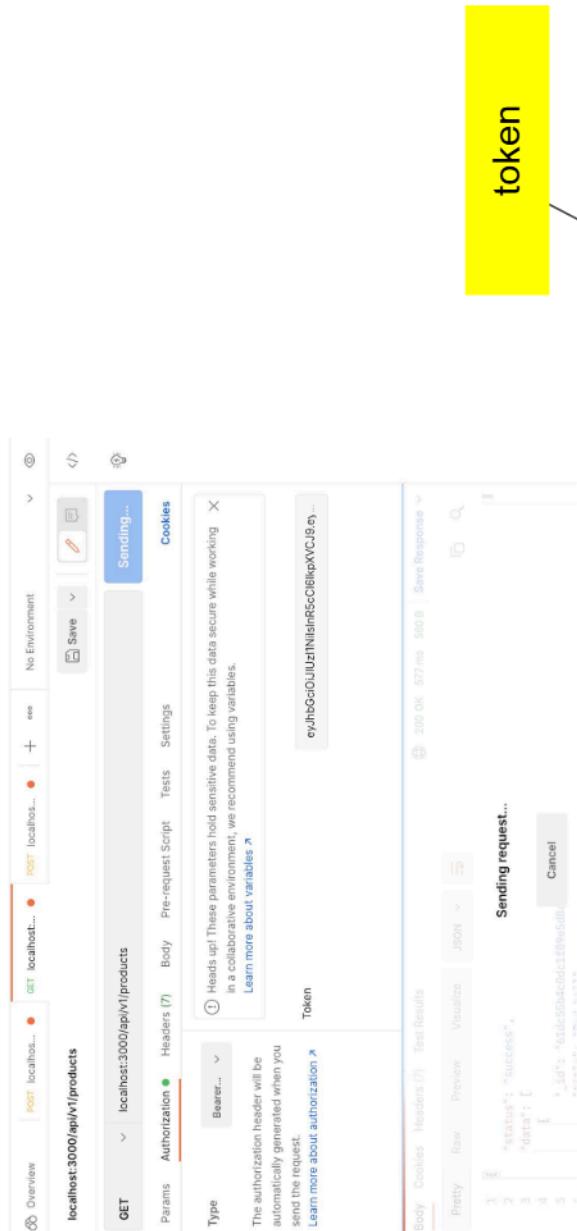
productRouter.get('/', authController.protectSystem, productController.getProducts);

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;
```

# 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  
  }  
}
```

localhost:3000/api/v1/products



Running on port 3000  
eyJhbGciOiJUzI1NiIsInR5cCI6IkpXVCJ9.eyJxZTNkOWZlZGY3Zml5ODh1MDJUNDRmNyIsImlhdcI6MTY0MjMyNDczMCwiZXhwIjoxNjQ1OTI0NzNmfQ.3ZLz5ugvWTU55  
D8d5vbpcKvh0BS8kQjqwR0f6MTegu

# 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.

# 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')
```

# 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  
  }  
}
```

# localhost:3000/api/v1/products

Running on port 3000  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjYxZTNKOWZiGY3mU50Dh1MDJUNDNmNyIsImlhhdCI6MTY0MjMyNDczM(CwiZXhwIjoxNjQ10T10NzlwfQ.3ZLz5ugvVTu55  
{ id: '61e3a9fbdf7fe988b02e44f7' , iat: 1642324730 , exp: 1645924730 }

User id decoficiado

token

The screenshot shows a POSTMAN interface with the following details:

- Method:** GET
- URL:** localhost:3000/api/v1/products
- Headers:** Authorization: Bearer -> Token (highlighted in yellow)
- Body:** (empty)
- Pre-request Script:** (empty)
- Tests:** (empty)
- Settings:** (empty)
- Cookies:** (empty)
- Environment:** (empty)

The response pane displays the JSON data from the previous step:

```
Running
{
  "id": "61e3a9fbdf7fe988b02e44f7",
  "iat": 1642324730,
  "exp": 1645924730
}
```

Below the response, there are status indicators: "Could not get response" (red), "Cancel" (button), and "Error: socket hang up | View in Chrome". A tooltip says "Learn more about troubleshooting API requests".

# Protegiendo datos - fase 3 - comprobar si el usuario existe

```
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
  }
}
```

## 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: {}  
});
```

# 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;
  }
};
```

# 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()

# 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'  
    }  
});
```

# 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'  
    })  
  }  
}
```

# Permisos - Agrega el middleware isAdmin

```
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;
```

Ahora solo los "users" que son "admin" pueden `getProducts`

# Registrar un “user”

The screenshot shows a Postman interface with the following details:

- Method:** POST
- URL:** localhost:3000/api/v1/users/signup
- Headers:** Authorization (B), Body (raw, application/x-www-form-urlencoded)
- Body:**

```
1 {  
2   "email": "c1@gmail.com",  
3   "password": "123456",  
4   "firstname": "as",  
5   "lastname": "last",  
6   "passwordChangedAt": "10/18/2022"  
7 }  
8
```
- Test Results:** Status: 201 Created Time: 276 ms Size: 768 B
- Response Headers:** Status: 201 Created Time: 276 ms Size: 768 B
- Response Body:**

```
1   "status": "success",  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
3   eyJrZC16fjYXZTmVj9K2mQyMhNj5zUTvNWEY1ts.m1hdT16MTY0djh4Ng5myw.iZXhLrjoXhjQ10THW0DRzFQ.  
4   A9GazmzOTIK-qrLVxHLL5OL3udNM7XILWqB8VyrCC2u",  
5   "user": {  
6     "email": "c1@gmail.com",  
7     "password": "2d$105tENNgH9PMlbnPm_L1YCmu5ib75mv2dKE58danZN1shDCh2+HwN",  
8     "firstname": "as",  
9     "lastname": "last",  
10    "passwordChangedAt": "2022-10-09T12:00:00.000Z",  
11    "permission": "user",  
12    "_id": "0103fb0dfdf23cb24395d5ab",  
13    "created": "2022-01-16T11:01:33.471Z",  
14    "modified": "2022-01-1611:01:33.471Z",  
15    "__v": 0  
16 }
```

# Conectar un “user”

Copia el token

The screenshot shows a POST request to `localhost:3000/api/v1/users/login`. The request body contains:

```
1 "email": "r1@gmail.com",
2 "password": "123456"
```

The response status is 200 OK, and the response body is:

```
1 "status": "success",
2 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9,
eyJpczI6IjyX7Bm7jkRkMjQzOTVnMEY1IiwiJhdC16NTY0NjR2MTA3MjwiwZKwIjoxbjQ1OTIwMDUzQ0,
3 a3df48cdmaV75pejkYoFYY17m3m060qjneTj0-Q"
4
```

# Trata de obtener “products”

Pega el token

localhost:3000/api/v1/products/

Authorization: Bearer Token

Token: eyJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJsb2dpbiIsImlhdCI6MTY0MjV2MjA1MywiZXhwIjoxNQ1OTMxMDUzIQ.aidEMABxdmA75peikOyFEYY77NJmRD6QQOfmerjO-Q

Status: 401 Unauthorized Time: 187 ms Size: 283 B Save Response

1. { "message": "you dont have permission" }

2. { "message": "you dont have permission" }

3. { "message": "you dont have permission" }

# Change to admin

```
_id: ObjectId("61e3fb0dff23cb24395a5a5b")
email: "c1@gmail.com"
password: "$2a$10$KENNgh9MwLbtNPm/LiYcmu5iBZ5mv2dKE58danZN1shDhCmH2tWoW"
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
```

Change to admin

# Ahora tu tienes “permission”

The screenshot shows a POSTMAN interface with a yellow callout box pointing to the response body. The callout box contains the text "Respuesta ‘products’".

**Request URL:** localhost:3000/api/v1/products/

**Method:** GET

**Headers:**

- Authorization: Bearer T...
- Content-Type: application/json

**Body:** (Empty)

**Test Results:**

Status: 200 OK | Time: 605 ms | Size: 413 B | Save Response

**Response Body:**

```
[{"id": "61dc55ce0dc1f89e5d8a7e6", "title": "Blue ball", "description": "This is a blue ball", "price": 500, "created": "2023-10-09T21:00:00.000Z", "__v": 0}]
```

## Hazlo tú mismo

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

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

# Regístrate en <https://mailtrap.io/>

After sign up you can see your Host Port Username Password

Total messages sent: 0

**My Inbox**

[SMTP Settings](#) [Email Address](#) [Auto Forward](#) [Manual Forward](#) [Team Members](#)

**SMTP / POP3** [Reset Credentials](#) [D](#)

Use these settings to send messages directly from your email client or mail transfer agent.

ⓘ Don't disclose your username or password as this may result in your inbox getting filled up with spam.

[Hide Credentials](#) [^](#)

**SMTP**

Host:	smtp.mailtrap.io
Port:	25 or 465 or 587 or 2525
Username:	0e67e85c50f88ec
Password:	0e3b2c5d3bc640
Auth:	PLAIN, LOGIN and CRAM-MD5
TLS:	Optional (STARTTLS on all ports)

**POP3**

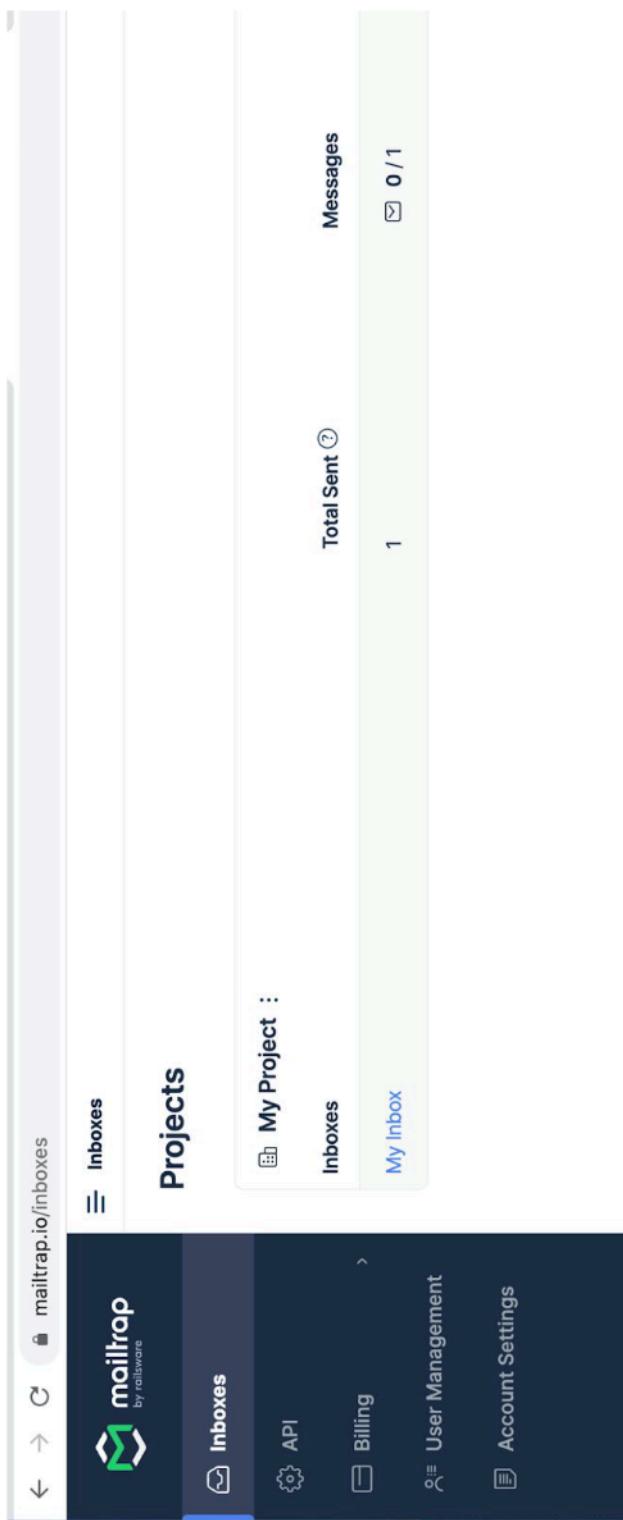
Host:	pop3.mailtrap.io
Port:	1100 or 8950
Username:	0c178850f0f88c
Password:	0e3b2c5d3bc640
Auth:	USER/PASS, PLAIN, LOGIN, APOP and CRAM-MD5
TLS:	Optional (STARTTLS on all ports)

**Integrations** [D](#)

CURL [Copy](#)

```
curl --ssl-reqd \ 
--url 'https://smtp.mailtrap.io:2525' \
--user '0e67e85c50f88ec@be3bec640' \
--mail-from from@example.com \
--mail-to to@example.com \
--upload-file - <>EOF
From: Magic Elves <from@example.com>
To: Mailtrap Inbox <to@example.com>
Subject: You are awesome!
Content-Type: multipart/alternative; boundary="boundary-string"
--boundary-string
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable
```

# Ve a tu inbox



# Contraeña 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_USERNAME, EMAIL_PASSWORD, EMAIL_HOST, EMAIL_PORT
```

# 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.

# 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  
    }  
}
```

# 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'})
}
```

# Manda el token al email

localhost:3000/api/v1/users/forgotPassword

The screenshot shows a Postman interface with the following details:

- Request URL:** localhost:3000/api/v1/users/forgotPassword
- Method:** POST
- Headers:** Authorization (none), Headers (8)
- Body:** Body (green dot) selected, JSON selected, raw (disabled), binary (disabled), GraphQL (disabled). Body content: { "email": "c1@gmail.com" }
- Response Status:** 200 OK
- Response Body:**

```
1: {  
2:   "status": "success",  
3:   "message": "token sent to your email"  
4: }
```

# Acepta el email

The screenshot shows an email inbox interface with the following details:

- Header:** Inboxes > My inbox > your password reset token
- Search Bar:** Search..., Q, Filter icon, Refresh icon, Help icon
- Email Preview:** Subject: your password reset token, To: <c@gmail.com>, Sent: a few seconds ago
- Email Content:**
  - Text Body:** Click here to make new password <http://localhost:1000/api/v1/users/resetPassword/> A123qrPswdJc453e13ameo9ApbShBz2kmt4
  - Headers:** From: "no-reply" <test mail>, To: <c@gmail.com>
  - Options:** Show Headers, HTML, Source, Text (selected), Raw, Spam Analysis, Tech Info
- Footer:** You are limited to last 30 messages, 1 inbox and 1 user on the Free forever plan. Click here to upgrade. Not now, thanks.

# 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
  });
}
```

# Mandar reestablecer contraseña

The screenshot shows a POST request in Postman to the URL `localhost:3000/api/v1/users/resetPassword/renmtjt4ym2lqd1ks3j4ubhn6ios3d0qscknysd2`. The request method is **PATCH**. The body contains the JSON object `{"passwordId": "1234567"}`. The response status is **200 OK**, time **508 ms**, size **438 B**. The response body is:

```
1 {"status": "success",  
2 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkoXVC9J  
eyJ0ZTI6IjYxTThmJjBzQyMNMjQzOTWhMkI1IiItImhC16NTY0MjUwDEmiwiizXhmljoxNjQ2NTA4MTAyfQ.  
eyNHYs3Bj-bqjkophbSSScDR1jig8suud9gZBnd0Etqm"
```

Nuevo token

# Actualizar mi “password”

```
exports.updatePassword =async function (req, res, next) {
  //obtener user
  let user = await User.findById(req.user._id);
  //verificar 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
  });
}
```

# 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;
```

# Agrega esta función a los métodos “model”

```
UsersSchema.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 = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
    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.

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

2022 © Wawiwa Tech | Confidential

# localhost:3000/api/v1/users/updateMyPassword

The screenshot shows a POST request to `localhost:3000/api/v1/users/updateMyPassword`. The request body is:

```
1 {  
2   "currentPassword": "1234567",  
3   "password": "654321"  
4 }  
5
```

The response status is 200 OK, with a response body:

```
1 {  
2   "status": "success",  
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCIkVXCY9.  
4 eyJpZC16IjixZTNeYjB0ZQyM2iBjQ-0TVNNE1vi5ImhdC16MTy0Mj40TMyNSwiZXhwIjoxMjg5Mz14q.  
5 uploadKwBAuJC1D7wx004fP6HFBN0ySuH3BuTC5aw"
```

# 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.

# 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  
}  
  
// Returns updated object.  
// actualizado.
```

# localhost:3000/api/v1/users/updateMe

The screenshot shows a POST request to `localhost:3000/api/v1/users/updateMe`. The request body is:

```
1 "firstname": "Oren",
2 "lastname": "Bambauer",
3 "password": "asf",
4 "permission": "user"
5 }
```

The response body is:

```
1 {
2   "status": "success",
3   "user": {
4     "_id": "61a3bd0df23c124395a5a3b0",
5     "email": "cl@gmail.com",
6     "password": "$2b$10$J3anQz1OpIyapPKFLDg2u15i93KcneZEPnUj9s82113Kyzmey",
7     "firstname": "Oren",
8     "lastname": "Bambauer",
9     "passwordChageDate": "2022-01-07T12:25:41.714Z",
10    "permission": "admin",
11    "created": "2022-01-16T11:01:33.471Z",
12    "modified": "2022-01-16T11:01:33.471Z",
13    "v": 0
14  }
15 }
```

Annotations on the right side of the response pane:

- A green box highlights the first two lines of the response body with the text: "Firstname y last name cambiaron".
- A green box highlights the last two lines of the response body with the text: "Permission no se actualiza".

# 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”

# Agrege la función “pre” para el esquema del “user”

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

# 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  
    });  
}
```

# Agrega la función deleteMe a UserRouter.js

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

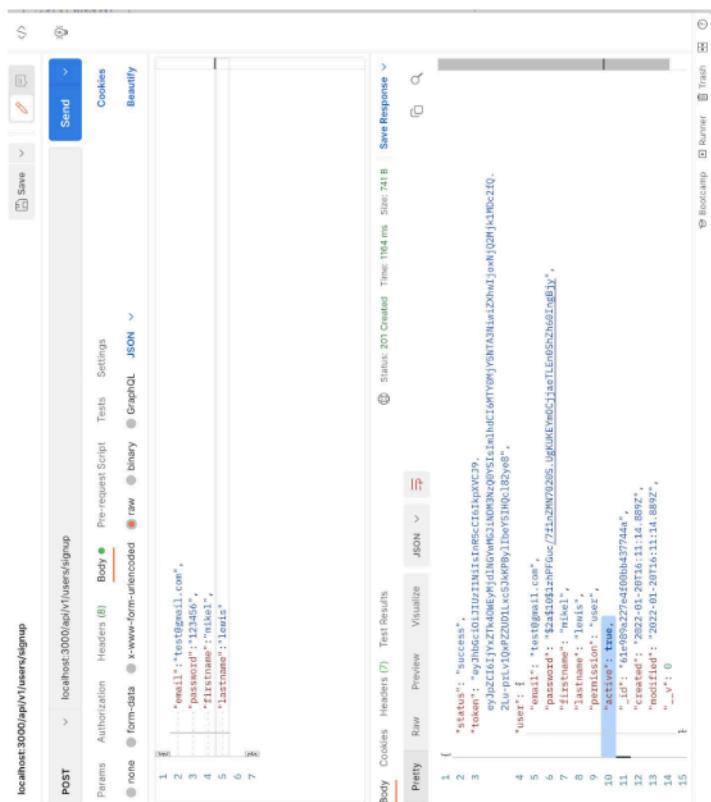
userRouter.post('/signup', authController.signup);
userRouter.post('/login', authController.login);
userRouter.post('/forgotPassword', authController.forgotPassword);
userRouter.patch('/resetPassword/:token', authController.resetPassword);
userRouter.patch('/updateMyPassword', authController.protectSystem,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

localhost:3000/api/v1/users/signup



# localhost:3000/api/v1/users/deleteMe

```
_id: ObjectId("61e989a227e4f00bb437744a")
email: "test@gmail.com"
password: "$2a$10$1zhPFGuc/7f1nZMw7020S.UgJKUKEYm0CjjaetTLEn0ShZh60IngBjy"
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?