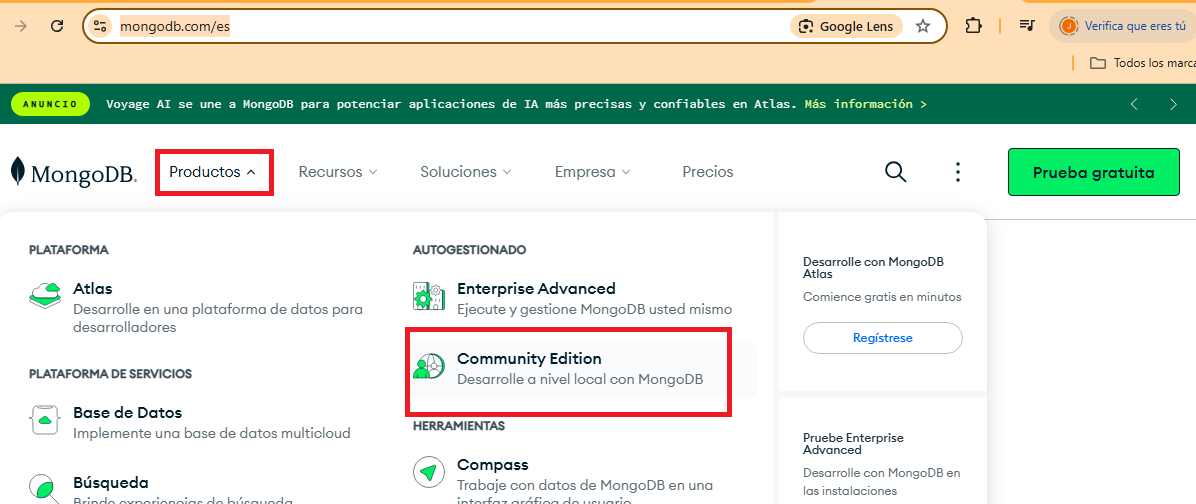
En esta sección vamos a crear un api rest completa para crear un blog con el MERN Stack y vamos a seguir todos los pasos para construir nuestro primer backend con JavaScript, NodeJS, ExpressJS y MongoDB desde cero, sigue todas las clases en orden

Seccion 27 de curso: **Master en React: Aprender ReactJS, Hooks, MERN, NodeJS, JWT+**

Es posible que en ciertas versiones como la 6 de mongodb, no esté instalado por defecto el mongo.exe y por tanto no tengamos una interfaz de comandos para usar mongodb, en ese caso puedes instalar adicionalmente Mongo Shell (mongosh.exe) desde aquí: https://www.mongodb.com/try/download/shell y ya tendrás una consola de comandos para trabajar con mongo.

**Pasos para descargar MongoDb versión 5**

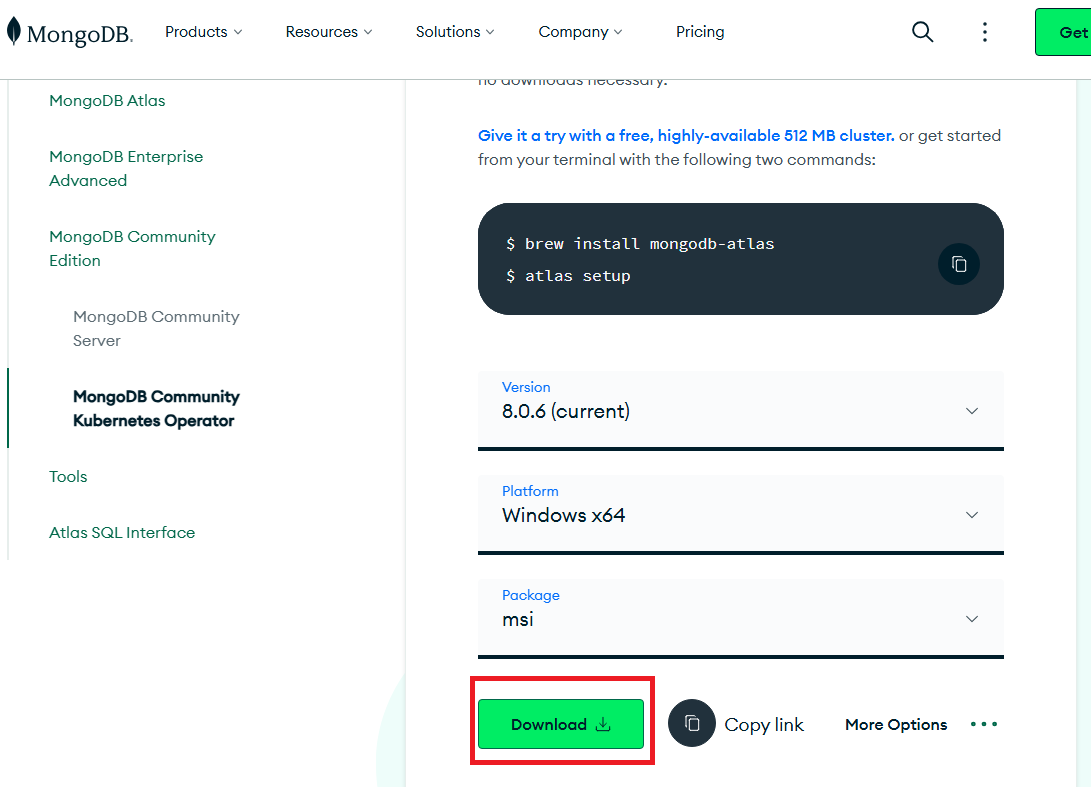
1. Clic en el enlace para instalar MongoDb 🡺 <https://www.mongodb.com/es>
2. Clic en el menú Productos
3. Clic en el recuadro denominado “Comunity Edition”



1. Clic en el botón “Descargar Comunidad”.



1. Seleccionamos la versión de MongoDb deseada (preferible la versión 5)
2. Clic en el botón “Download”



Después de ejecutar el instalador de MongoDB (archivo con extensión msi) se debe de crear la carpeta data preferente en la ruta C:\ y dentro de la carpeta data se debe de crear la carpeta denominada db.

Dentro de la carpeta data se guardará la base de datos no relacional de MongoDb

**Crear el demonio para correr MongoDb versión 5**

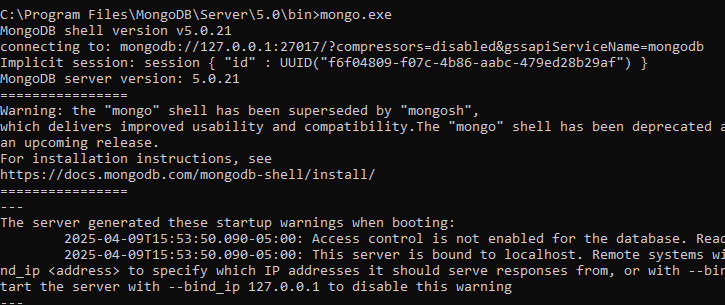
1. Abrimos el cmd donde se haya instalado MongoDb, en este caso es en la sgte ruta: C:\Program Files\MongoDB\Server\5.0\bin
2. Crear carpeta donde se guardarán las colecciones (tablas), en este caso la ruta es: C:\data5
3. Ejecutamos la el sgte comando en CMD: mongod.exe --dbpath C:\data5



**Ejecutar MongoDb después de correr el demonio**

1. Abrimos el cmd donde se haya instalado MongoDb, en este caso es en la sgte ruta: C:\Program Files\MongoDB\Server\5.0\bin
2. Ejecutar el comando: mongo.exe



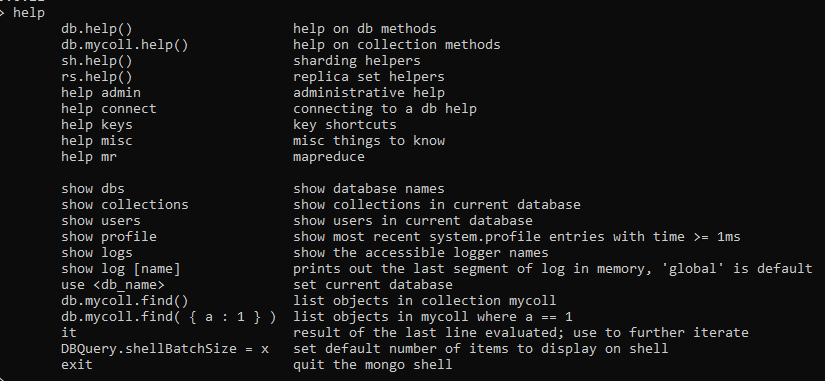


**Comandos de MongoDb**

En la consola donde se ejecutó mongo.exe se pueden usar los sgts comandos:

db.version() : Permite conocer la versión instalada de MongoDb

help : Muestra los comandos disponibles más usados



show dbs : Muestra las base de datos disponibles

use <db\_name>: Si se desea usar la base de datos selecciona, en caso de que la base de datos no exista se crea con el nombre especificado, <db\_name> representa el nombre de la base de datos a usar

exit : Salir de la consola

Colección = tabla

db.ps4.save(<objecto>) : Inserta una coleccion en la tabla especificada que en este caso se denomina “ps4”, antes de usar este comando se debe de usar el comando use para situarse en la base de datos donde se desee crear las coleciones

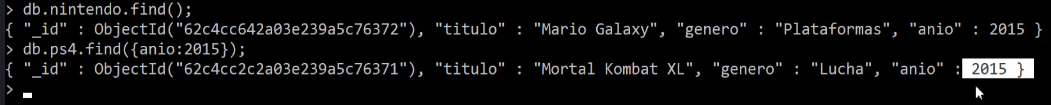


show collections: Muestra las colecciones de la base de datos situada

Método de Consulta a una la colleccion

db.nintendo.find() : selecciona todas los registros de la colección denominada nintendo.

db.nintendo.find({anio:2015}) : selecciona todas los registros de la colección denominada nintendo que tengan la propiedad anio = 2015



db.nintendo.deleteMany({anio:2015}) : Elimina todos los registros de la colección denominada nintendo que tengan la propiedad anio = 2015

db.nintendo.deleteOne({anio:2015}) : Elimina el primer registro de la colección denominada nintendo que tengan la propiedad anio = 2015

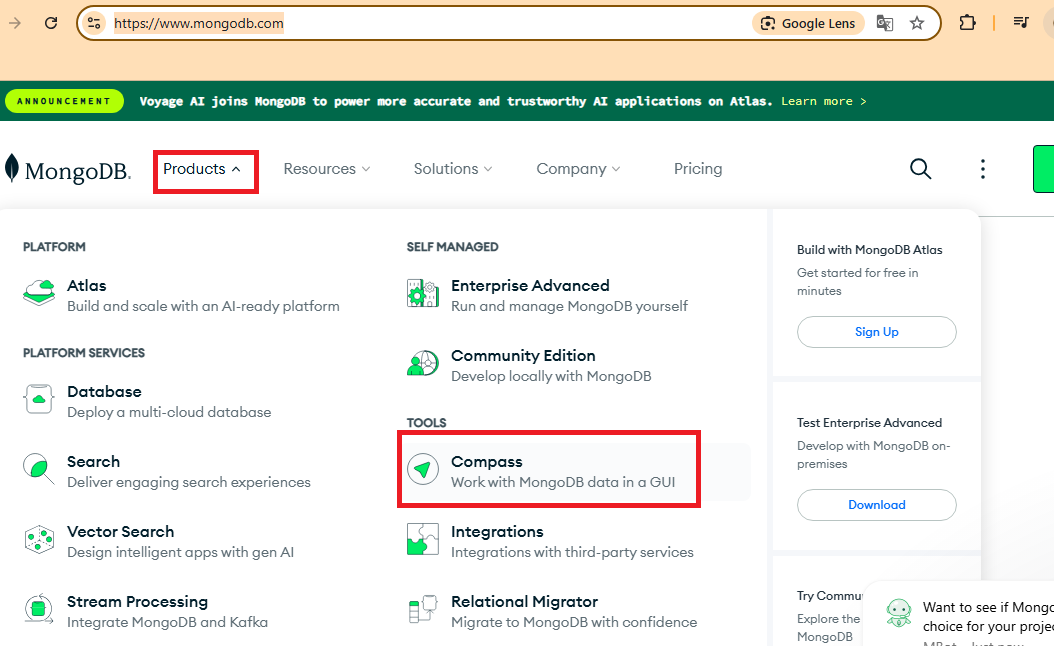
Interfaz de MongoDb: MongoCompass

**Pasos para descargar la interfaz de MongoDb: MongoCompass**

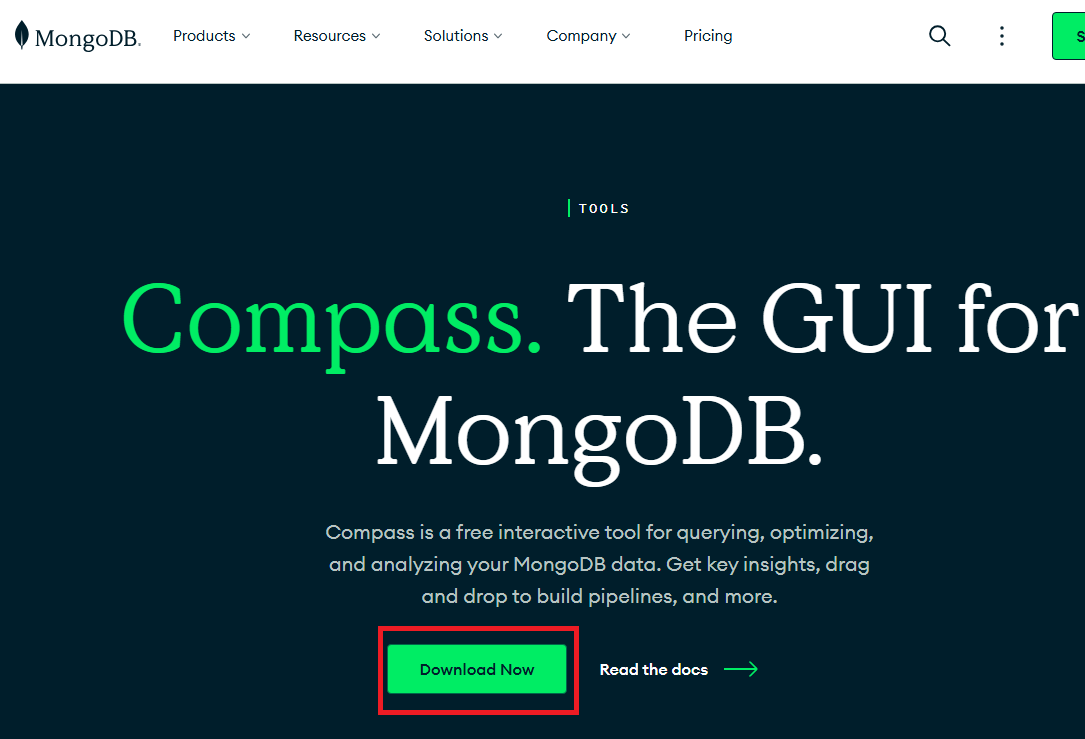
Clic en el sgte enlace <https://www.mongodb.com/>

Clic en el menú Productos

Clic en la opción MongCompass

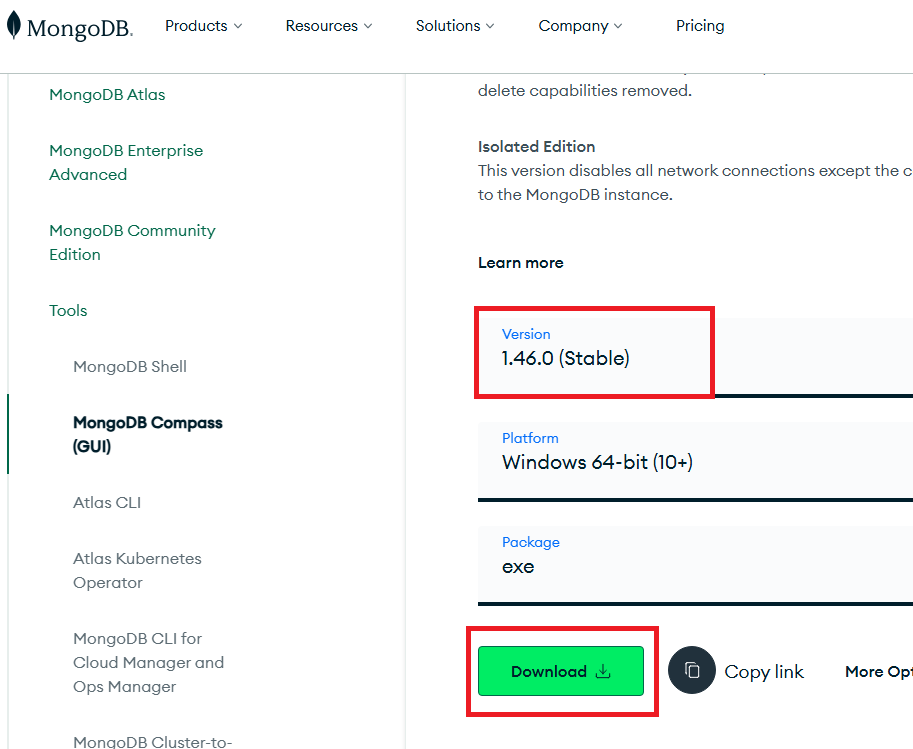


Clic en el botón Download Now

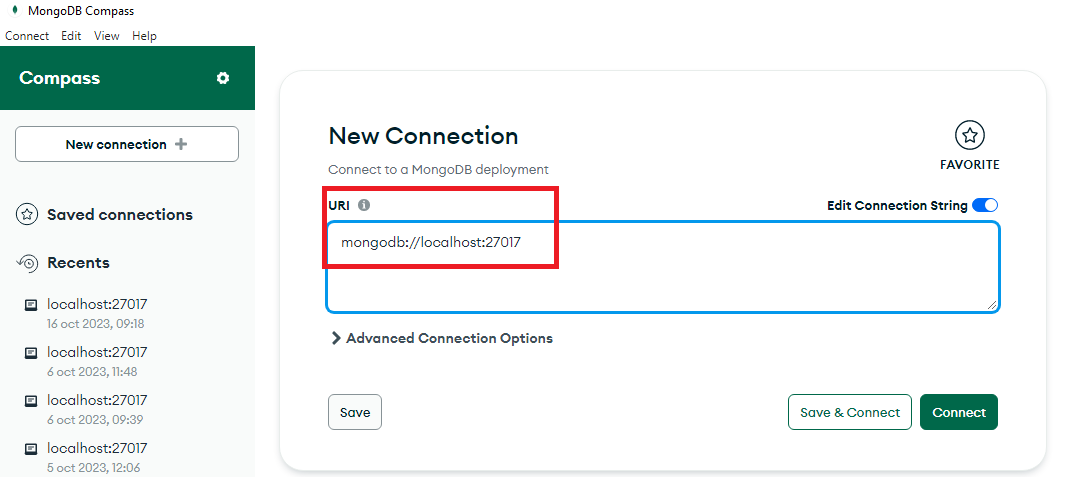


Seleccionar la opción Estable de MongoCompass

Clic en el botón Download



Conectarse a la base de datos de mongodb escribiendo el string de conexión mostrado al ejecutar la línea de comando mongo.exe



**Soluciones a todos los posibles errores de MongoDb y Mongoose: Clase 168**

En algunas versiones (desde la versión 6) no existe el archivo mongo.exe, porque ahora ya esta disponible Mongo Shell para ejecutar líneas de comando de MongoDb

Accediendo al sgte enlace es posible descargar MongoDb Shell <https://www.mongodb.com/try/download/shell>

La documentación aquí:

<https://www.mongodb.com/docs/mongodb-shell/>

**Entorno de desarrollo para crear Api Rest con MongoDb**

**Empaquetador NodeJs**

Comandos a usar después de instalar NodeJs

node --version

**Librería Express**

Express es un framework que se encarga de manejar las peticiones HTTP (GET, POST, UPDATE, DELETE)

npm install express --save Guardar dependencia express dentro de nuestro proyecto, no de manera global

**Mongoose**

Permite tener una librería de funciones variadas (hacer una búsqueda, guardar un dato entre otros) para trabajar de manera más sencilla con monbgodb dentro de nuestro proyecto con node.js

Intalar una versión de mongoose en concreto versión 6.3, una vez ingresado al proyecto

npm install mongoose --save conexion a bd mongodb

Después de instalar mongoose es posible usar la sgte línea de comandos en el CMD:

show mongoose versions: Permite ver las versiones de mongoose disponibles.

npm update mongoose: Actualiza mongoose a la version más estable.

npm uninstall mongoose : Elimina un paquete npm de mi proyecto.

npm install [mongoose@7.2.0](mailto:mongoose@7.2.0): Instalar versión 7.2.0 de mongoose.

Enlace para validar datos de un modelo a través de mongoose: <https://mongoosejs.com/docs/schematypes.html>

const {Schema, model} = require("mongoose");

// https://mongoosejs.com/docs/schematypes.html

const ArticuloSchema = Schema(

    {

        titulo: {

            type: String,

            required: true

        },

        contenido: {

            type: String,

            required: true

        },

        fecha: {

            type: Date,

            default: Date.now

        },

        imagen: {

            type: String,

            default: "default.png"

        },

    }

);

module.exports = model("Articulo", ArticuloSchema, "articulos");// la coleccion va hacer articulos

**Librería multer**

Librería que permite subir archivos

npm install multer --save :Guardar dependencia multi dentro de nuestro proyecto, no de manera global

**Librería validator**

Librería que permite validar datos, su enlace es: https://www.npmjs.com/package/validator

npm install validator --save : Guardar dependencia validator dentro de nuestro proyecto, no de manera global

**Librería core**

Solución a problemas de cors, es decir a problemas de orígenes y dominios cruzados

npm install cors --save Guardar dependencia cors dentro de nuestro proyecto, no de manera global

**Nodemon**

Monitorizar todos los cambios que se hagan a nivel código para actualizar el servidor, evita no tener que cerrar el proyecto y de nuevo arrancar el código.

npm install nodemon --save-dev compila y revisa el código de node, el --save-dev es para que solo se instale en modo desarrollo, no cuando se corra en producción

**Crear un proyecto de node.js**

1. Crear carpeta en cualquier carpeta contenedora
2. Ejecutar comando: npm init
3. Ingresamos los parámetros necesarios para su creación como el nombre, autor, nombre del archivo index

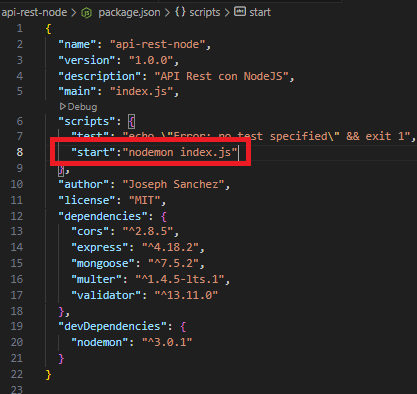
Comandos a usar después de crear el proyecto node.js (Ubicarse por consola dentro de la carpeta principal del proyecto, se debe de tener creado un archivo index.js)

node index.js :Arranca el servidor. Para no tener que typear node index.js para arrancar el servidor se puede escribir dentro del archivo package.json la sgte línea:

"start":"node index.js"

O si se instala la librería Nodemon se puede escribir

"start":"nodemon index.js"



Después se puede escribir el comando npm start para inicializar el servidor

more package.json: Muestra en consola el contenido del archivo package.json

**Conexión a la base de datos**

Con método async-await

const mongoose = require ("mongoose");

const conexion = async()=>{

     try {

        await mongoose.connect('mongodb://127.0.0.1:27017/mi\_blog', { useNewUrlParser: true, useUnifiedTopology: true });

        // Parametros a pasar dentro de un objeto // solo en caso de aviso

        // useNewUrlParser: true

        // useUnifiedTopology: true

        // useCreateIndex: true

        console.log("Conectado correctamente a la base de datos mi\_blog !!");

     } catch (error) {

        console.log(error);

        // throw new Error("No se ha podido conectar a la base de datos");

     }

}

module.exports = {

   conexion

}

Para usar la función exportada denominada conexión se escribe lo sgte en el index.js:

const {conexion} = require("./basedatos/conexion");

**Validación con librería validator**

**Carpeta helpers**

const validator = require("validator");

const validarArticulo =(parametros)=>{

    const validar\_titulo = !validator.isEmpty(parametros.titulo) && validator.isLength(parametros.titulo, {min: 5, max: undefined});

    const validar\_contenido = !validator.isEmpty(parametros.contenido);

    if(!validar\_titulo || !validar\_contenido){

        throw new Error("No se ha validado la informacion");

    }

}

module.exports = {

    validarArticulo

}

La validación en este caso es la sgte:

titulo: No debe de estar vacío.

contenido: No debe de estar vacío, debe de tener mínimo 5 caracteres y no tiene limite en la cantidad máxima de caracteres.

Para usar la función exportada denominada validarArticulo se escribiría en el archivo lo sgte:

const { validarArticulo } = require("../helpers/validar");

Su documentación presenta otros métodos de validación útiles: https://www.npmjs.com/package/validator

**Modelo Articulo**

const {Schema, model} = require("mongoose");

// https://mongoosejs.com/docs/schematypes.html

const ArticuloSchema = Schema(

    {

        titulo: {

            type: String,

            required: true

        },

        contenido: {

            type: String,

            required: true

        },

        fecha: {

            type: Date,

            default: Date.now

        },

        imagen: {

            type: String,

            default: "default.png"

        },

    }

);

module.exports = model("Articulo", ArticuloSchema, "articulos");// la coleccion va hacer articulos

**CRUD - Peticiones HTTP**

Cada URL o ruta de estos endpoints tiene un prefijo definido el archivo index.js, el cual es "/api"

//RUTAS

const rutas\_articulo = require("./rutas/Articulo");

// Cargo las rutas de articulo que se carguen a partir de la palabra /api

app.use("/api",rutas\_articulo);

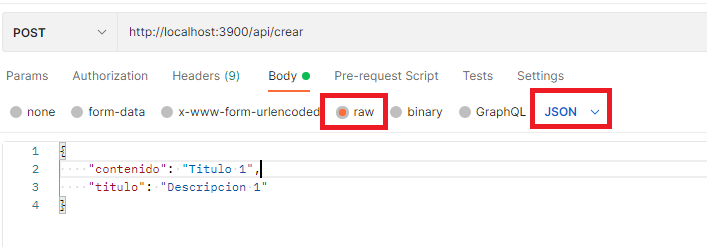
CREATE

**Carpeta ruta**

router.post("/crear",ArticuloControlador.crear);

La ruta "/crear", de tipo POST, nos dirige al método crear del controlador Articulo. La ruta nos indica que se obtiene los datos de un formulario, esto en Postman es en la parte del body, imagen a continuación

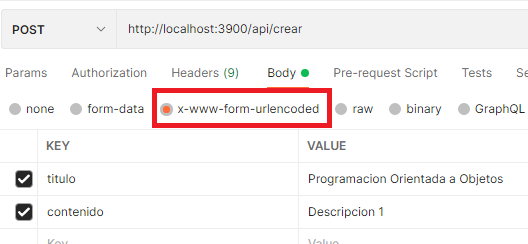
1. Si se envían los parámetros en formato json



En el archivo index.js del proyecto para convertir dicho objeto a javascript se debe de tipear lo sgte:

app.use(express.json()); // recibir datos con content-type app/json, en Postman es la seccion Body - opcion raw

1. Si se envían los parámetros en formato x-www-form-urlencoded (como si se estuviese enviando los datos a través de un formulario)



En el archivo index.js del proyecto para convertir dicho objeto a javascript se debe de tipear lo sgte:

app.use(express.urlencoded({extended: true})); // se recibe datos que llegan en form-url encoded

**Carpeta Controlador**

req: (Request) Contiene los valores de los parámetros enviados en forma de objeto

res: Respuesta.

Los datos se validan después de haber sido ingresados.

const crear = (req, res) =>{

    // Recoger parametros por post a guardar

    let parametros = req.body;

    // Validar datos

    try {

        validarArticulo(parametros);

    } catch (error) {

        return res.status(400).json(

            {

                status: "error",

                mensaje: "Faltan datos por enviar"

            }

        );

    }

    Articulo.create(parametros).then(response =>{

        res.status(200).send(

            {

                status:"success",

                response,

                message:"Registro creado con exito!",

            }

        );

    }).catch(err=>{

        res.status(500).send(

            {

                status:"error",

                response: err,

            }

        );

    });

    // Crear el objeto a guardar

    // const articulo = new Articulo(parametros);

    // Asignar valores a objeto basado en el modelo (manual o automatico)

    // articulo.titulo = parametros.titulo;

    // Guardar el articulo en la base de datos

    // articulo.save((error, articuloGuardado)=>{

    //     if(error || !articuloGuardado){

    //         return res.status(400).json(

    //             {

    //                 status: "error",

    //                 mensaje: "No se ha guardado el articulo"

    //             }

    //         );

    //     }

    //     // Devolver resultado

    //     return res.status(200).json(

    //         {

    //             status: "success",

    //             articulo: articuloGuardado,

    //             mensaje: "Articulo creado con exito!"

    //         }

    //     );

    // });

}

**READ**

1. **Listar artículos proporcionando un limite opcional**

**Carpeta Ruta**

router.get("/articulos/:ultimos?",ArticuloControlador.listar);

La ruta "/artículos” tiene un parámetro opcional el cual indica el límite de registros a devolver, esta ruta nos dirige al método listar del controlador Articulo

**Carpeta Controlador**

const listar = (req, res)=>{

    setTimeout(() => {

        let consulta = Articulo.find();

        if(req.params.ultimos){

            consulta.limit(2);// limite de registro permitidos

        }

        // ordernar del mas actual al mas antiguo (de manera descendente)

        consulta.sort({fecha: -1}).then(response =>{

            res.status(200).send({

                status:"success",

                params:req.params,

                count:response.length,

                response,

            });

        }).catch(err=>{

            res.status(500).send({

                status:"error",

                response: err,

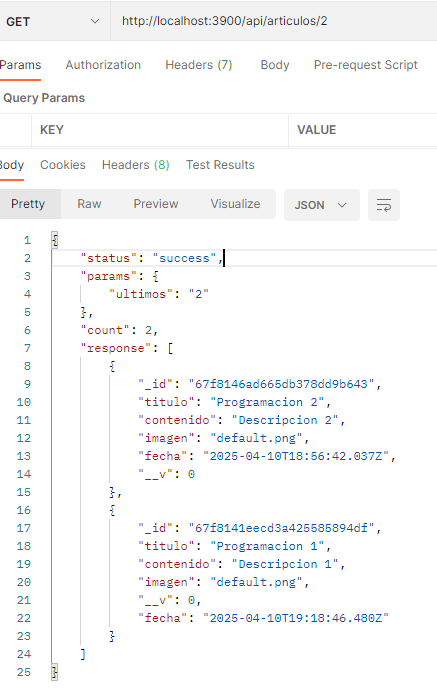
            });

        });

    }, 5000);

}

Testeo de la ruta en Postman



1. **Obtener un artículo concreto en la base de datos mediante su ID**

**Carpeta Ruta**

router.get("/articulo/:id",ArticuloControlador.uno);

La url "/articulo/:id", con método GET tiene el parámetro obligatorio id y apunta al método uno del controlador Articulo.

**Carpeta Controlador**

const uno = (req, res)=>{

    // Recoger un ID por la url

    let id = req.params.id;

    Articulo.findById(id).then(response =>{

        // if(!response.id){

        //     res.status(500).send({

        //         status:"error",

        //         message: "No se ha encontrado el articulo",

        //     });

        // }

        res.status(200).send({status:"success",response});

    }).catch(err=>{

        res.status(500).send(

            {

                status:"error",

                response: err,

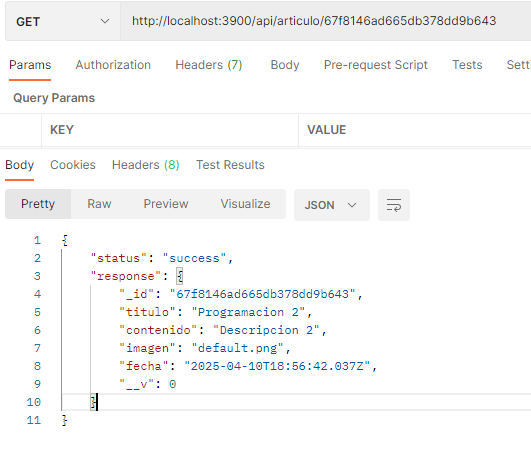
            }

        );

    });

}

Testeo de la ruta en Postman



**DELETE**

1. **Eliminar un registro mediante su ID enviado por parámetro.**

**Carpeta Ruta**

router.delete("/articulo/:id",ArticuloControlador.borrar);

La url "/articulo/:id",, con método DELETE tiene el parámetro obligatorio id y apunta al método borrar del controlador Articulo.

**Carpeta Controlador**

const borrar = (req, res)=>{

    const { id } = req.params;// desestructuracion

    let articuloId = id;

    Articulo.findOneAndDelete({\_id: articuloId}).then(response =>{

        res.status(200).send({status:"success",response});

    }).catch(err=>{

        res.status(500).send({status:"error",

        response: err,});

    });

    // Articulo.findByIdAndDelete(id).then(response =>{

    //     if(response.img) unlinkFile(response.img);

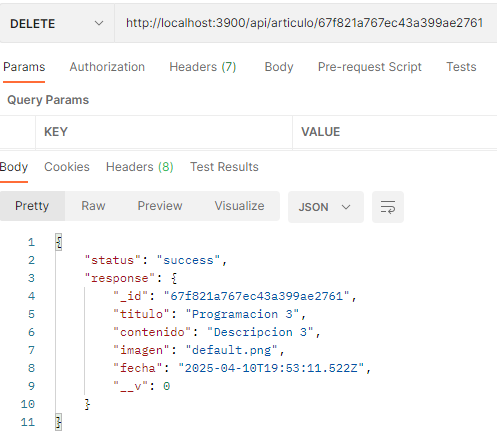
    //     res.status(200).send({response});

    // }).catch(err=>{

    //     res.status(500).send({response: err});

    // });

}

Testeo de la ruta en Postman

**PUT - UPDATE**

1. **Actualizar el registro Articulo mediante su ID enviado como parámetro.**

Carpeta Ruta

router.put("/articulo/:id",ArticuloControlador.editar);

La url ("/articulo/:id”, con método PUT tiene el parámetro obligatorio id y apunta al método editar del controlador Articulo.

**Carpeta Controlador**

const editar = (req, res)=>{

    // Recoger parametros por PUT

    let parametros = req.body;

    const { id } = req.params;

    try {

        validarArticulo(parametros);

    } catch (error) {

        return res.status(400).json(

            {

                status: "error",

                mensaje: "Faltan datos por enviar"

            }

        );

    }

    Articulo.findOneAndUpdate({\_id: id},parametros,{new: true}).then(response =>{ // new: true -> devuelve el objeto tal cual queda despues de actualizar

        res.status(200).send({status:"success",response});

    }).catch(err=>{

        res.status(500).send({status:"error",

        response: err,});

    });

    // Articulo.findByIdAndUpdate(id, payload).then(response =>{

    //     res.status(200).send({response});

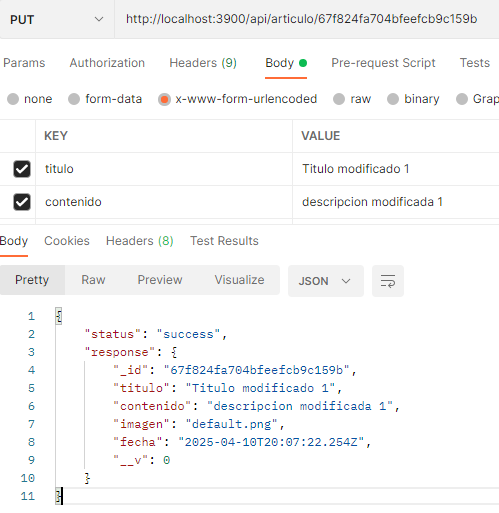
    // }).catch(err=>{

    //     if(payload.img) unlinkFile(payload.img);

    //     res.status(500).send({response: err});

    // });

}

Testeo en Postman

**Subir ficheros o imágenes con Nodejs**

Objetivo: Subir archivo de tipo jpg, jpeg, gig, png al servidor y actualizar la imagen del articulo mediante el envió de ID

**Carpeta Ruta**

Para esto se usa la librería 'multer', el cual es un middleware que se aplica a una ruta en concreto, en este caso el middleware se lo pasa en la ruta en forma de arreglo

[subidas.single("file0")]

Y se ejecuta antes del método que apunta la ruta que en este caso es el método subir

Importación de la librería:

const multer = require('multer');

Configuración donde se guardarán los ficheros o imágenes

const almacenamiento = multer.diskStorage({

    destination:(req, file, cb)=>{

        // esto se sube a la raiz del proyecto por eso se puso el punto adelante de la ruta

        cb(null, './imagenes/articulos/');

    },

    filename:(req, file, cb)=>{

        cb(null,"articulo"+Date.now()+file.originalname);// cambio de nombre del archivo

    }

});

const subidas = multer({storage: almacenamiento});

;

Ruta donde se guardaran las imágenes: './imagenes/articulos/'

const subidas = multer({storage: almacenamiento});

URL para subir fichero "/subir-imagen/:id", tiene un parámetro obligatorio denominado id y antes de que se ejecute dicho método se ejecuta el middleware multer (variable subidas), se usa el medoto single porque solo se va a subir un solo archivo y el nombre del campo (atributo name del formulario) se denomina file0

router.post("/subir-imagen/:id",[subidas.single("file0")],ArticuloControlador.subir);

**Carpeta Controlador**

Librería que tiene método para borrar fichero

const fs = require('fs');

const subir =(req, res)=>{

    // Comprueba si se paso un archivo

    if(!req.file || !req.file){

        return res.status(404).json(

            {

                status: "error",

                mensaje: "Peticion invalida",

            }

        );

    }

    // Nombre del archivo

    let archivo = req.file.originalname;

    // Extension del archivo

    let archivo\_split = archivo.split("\.");

    let archivo\_extension = archivo\_split[1];

    // return res.status(200).json(

    //     {

    //         status: "success",

    //         mensaje: "Enviado",

    //         archivo\_split,

    //         fichero: req.file

    //     }

    // );

    // Comprobar extension correcta

    if(archivo\_extension !="png" && archivo\_extension !="jpg" && archivo\_extension !="jpeg" && archivo\_extension !="gif"){

        // Borrar archivo y dar respuesta

        fs.unlink(req.file.path, (error)=>{

            return res.status(400).json(

                {

                    status: "error",

                    mensaje: "Imagen invalida",

                }

            );

        });

    }else{

        const { id } = req.params;

        // new: true -> devuelve el objeto tal cual queda después de actualizar

        Articulo.findOneAndUpdate({\_id: id},{imagen: req.file.filename},{new: true}).then(response =>{

            res.status(200).send({status:"success",response,fichero: req.file});

        }).catch(err=>{

            res.status(500).send({status:"error",

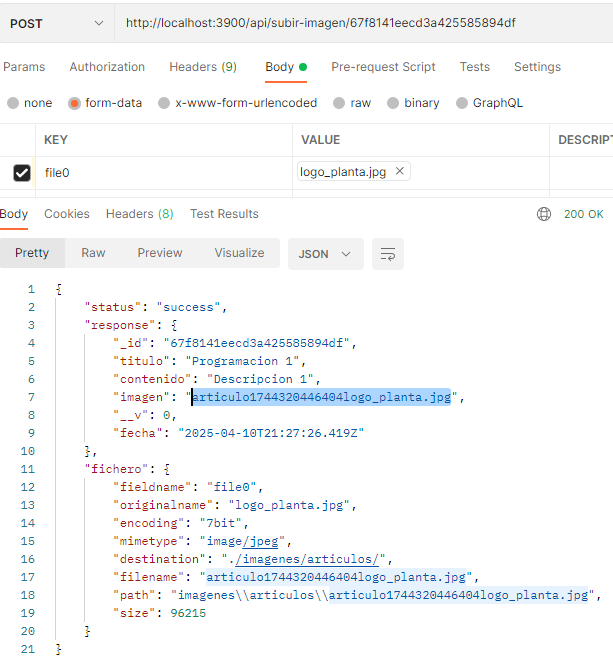
            response: err,});

        });

    }

}

Testeo de la ruta en postman



**Actualizar imagen enviando su nombre como parámetro con Nodejs**

**Carpeta Ruta**

router.get("/imagen/:fichero",ArticuloControlador.imagen);

La URL "/imagen/:fichero", envía un parámetro obligatorio denominado fichero y apunta al método imagen del controlador Articulo.

**Carpeta Controlador**

Dentro de nodeJs existe la librería path que permite cargar archivos, a continuación se muestra la importación de la librería

const path = require('path');

const imagen =(req, res)=>{

    // parametro enviado por url (nombre del fichero)

    let fichero = req.params.fichero;

    // ruta de las imagenes en el servidor (proyecto nodejs)

    let ruta\_fisica = "./imagenes/articulos/"+fichero;

    fs.stat(ruta\_fisica, (error, existe)=>{

        if(existe){

            // uso de la libreria path

            return res.sendFile(path.resolve(ruta\_fisica));

        }else{

            res.status(404).send(

                {

                    status:"error",

                    message: "La imagen no existe",

                }

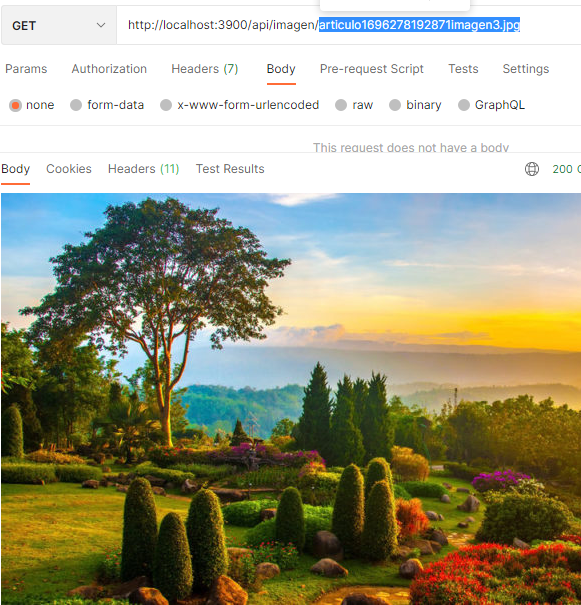
            );

        }

    });

}

Testeo en postman



**Buscador de artículo (registro) que coincida con el texto ingresado en su título o en su contenido.**

**Carpeta Ruta**

router.get("/buscar/:busqueda",ArticuloControlador.buscador);

La URL "/buscar/:busqueda", envía un parámetro obligatorio denominado busqueda y apunta al método buscador del controlador Articulo.

**Carpeta Controlador**

const buscador =(req, res)=>{

    // Sacar el string de busqueda

    let busqueda = req.params.busqueda;

    // Find OR

    Articulo.find(

        {

            // Interpretacion de la busqueda: Si el titulo o el contenido del articulo incluye el texto de busqueda seleccionamelo

            "$or":

            [

                {"titulo":{"$regex":busqueda,"$options":"i"}},

                {"contenido":{"$regex":busqueda,"$options":"i"}}

            ]

        }

    )

    .sort({fecha: -1}) // ordernar del mas actual al mas antiguo (de manera descendente)

    .then(response =>{

        if(!response){

            return res.status(500).send({

                status:"error",

                message: "No se ha encontrado el articulo",

            });

        }

        res.status(200).send({status:"success",response});

    }).catch(err=>{

        res.status(500).send({status:"error",

        message: "No se han encontrado articulos",});

    });

    // Articulo.findOneAndUpdate({\_id: id},{imagen: req.file.filename},{new: true}).then(response =>{ // new: true -> devuelve el objeto tal cual queda despues de actualizar

    //     res.status(200).send({status:"success",response,fichero: req.file});

    // }).catch(err=>{

    //     res.status(500).send({status:"error",

    //     response: err,});

    // });

}

Testeo en postman

