

Diplomatura en

FullStack Developer

Conexión a Bases de Datos MongoDB y
Gestión de Datos con Express

¿Qué es MongoDB?

MongoDB es una base de datos NoSQL de código abierto, orientada a documentos, diseñada para el almacenamiento de datos a gran escala. Utiliza un formato de documento similar a JSON, llamado BSON, para almacenar datos.

Características Principales

- **Flexible Schema:** Permite almacenar datos con esquemas dinámicos, lo que facilita la integración y actualización de datos.
- **Escalabilidad:** Ofrece una alta escalabilidad horizontal mediante la partición de datos.
- **Alto Rendimiento:** Optimizado para operaciones de lectura y escritura rápidas.
- **Alta Disponibilidad:** Soporta la replicación de datos para alta disponibilidad y recuperación ante desastres.
- **Consultas Poderosas:** Proporciona un lenguaje de consultas enriquecido y potente.

Escalabilidad Horizontal vs Vertical

Escalabilidad Horizontal (Sharding):

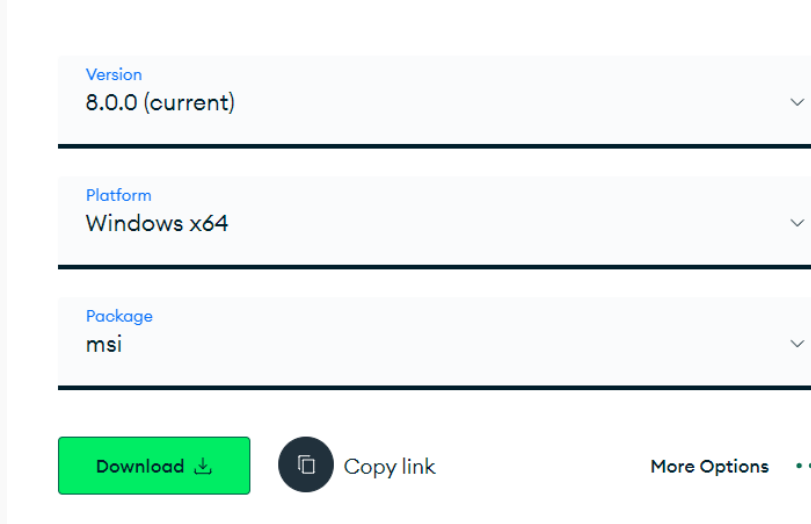
- **Definición:** Añadir más máquinas (nodos) al sistema para distribuir la carga.
- **Ventajas:**
 - Aumenta la capacidad del sistema al añadir más servidores.
 - Mejora la disponibilidad y la tolerancia a fallos.
 - Permite manejar grandes volúmenes de datos y tráfico.
- **Ejemplo en MongoDB:** Dividir la base de datos en partes más pequeñas llamadas shards, cada una alojada en un servidor diferente.

Escalabilidad Vertical (Scaling Up):

- **Definición:** Añadir más recursos (CPU, RAM, almacenamiento) a una única máquina.
- **Ventajas:**
 - Es más sencillo de implementar.
 - No requiere cambios en la aplicación.
 - Inicialmente puede ser más económico.
- **Desventajas:**
 - Tiene un límite físico y económico.
 - No mejora la tolerancia a fallos.
- **Ejemplo en Bases de Datos Relacionales:** Aumentar la capacidad de un servidor de base de datos existente añadiendo más memoria o procesadores.



Instalación de MongoDB en nuestra PC

Visitamos -> <https://www.mongodb.com/try/download/community>



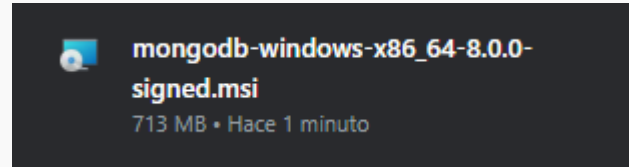
The screenshot shows the MongoDB download interface. It features three dropdown menus for selecting the version, platform, and package type. The version is set to 8.0.0 (current), the platform to Windows x64, and the package to msi. Below these selections are three buttons: a green 'Download' button with a download icon, a 'Copy link' button with a copy icon, and a 'More Options' button with a three-dot menu icon.

Version	8.0.0 (current)	▼
Platform	Windows x64	▼
Package	msi	▼

[Download](#)  [Copy link](#) [More Options](#) 

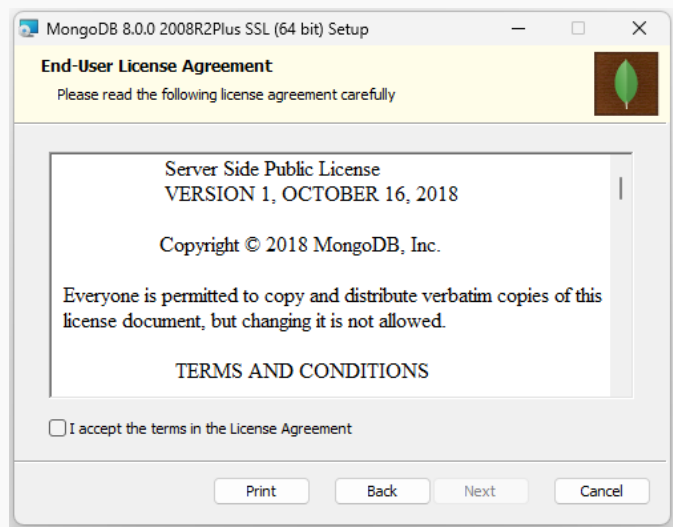
Le damos al botón Download

Una vez que termine instalamos

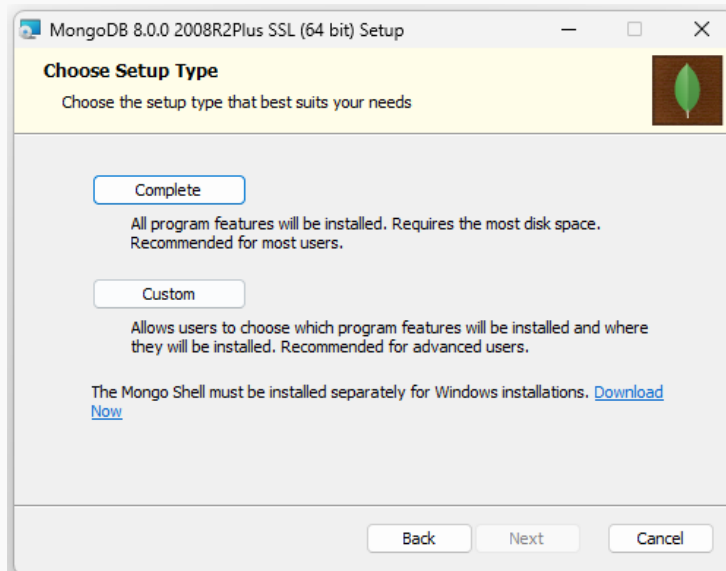


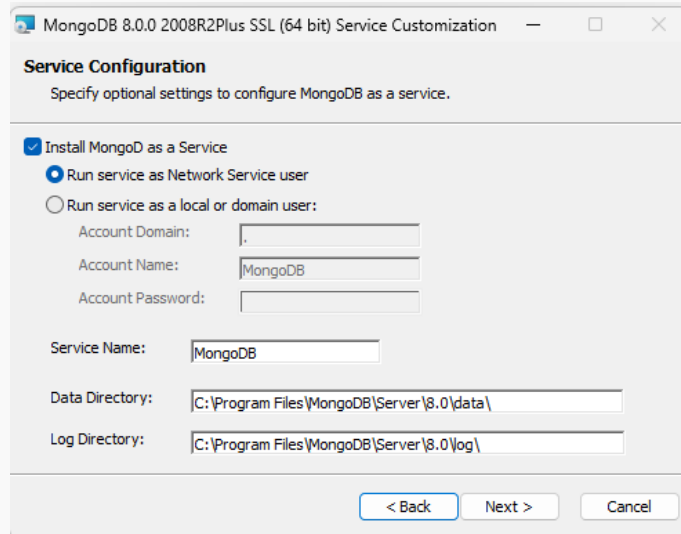
Presionamos next

Aceptamos los términos y luego Next

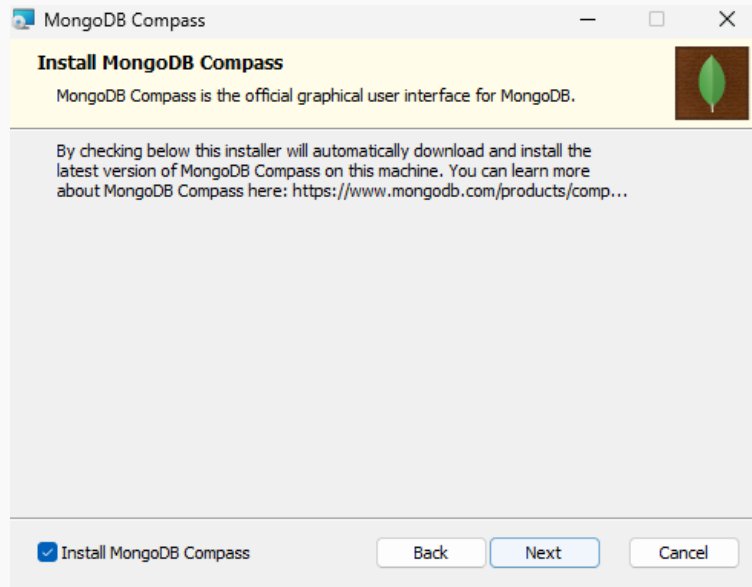


Seleccionamos la instalación completa (presionamos el botón Complete)

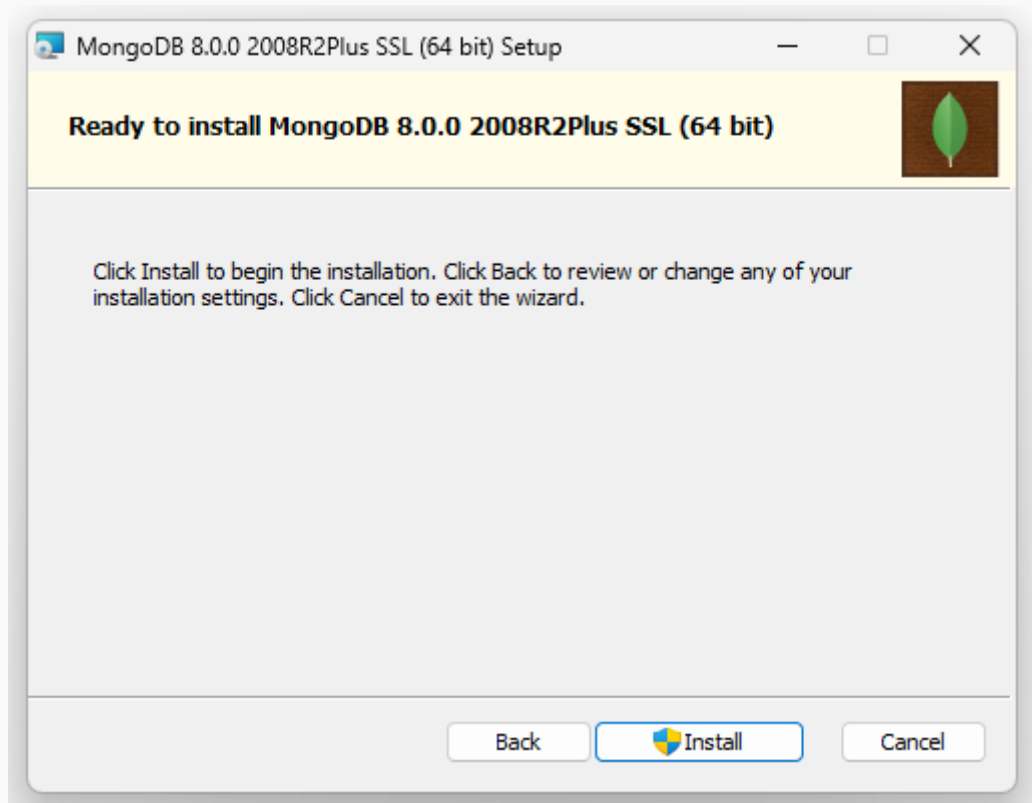




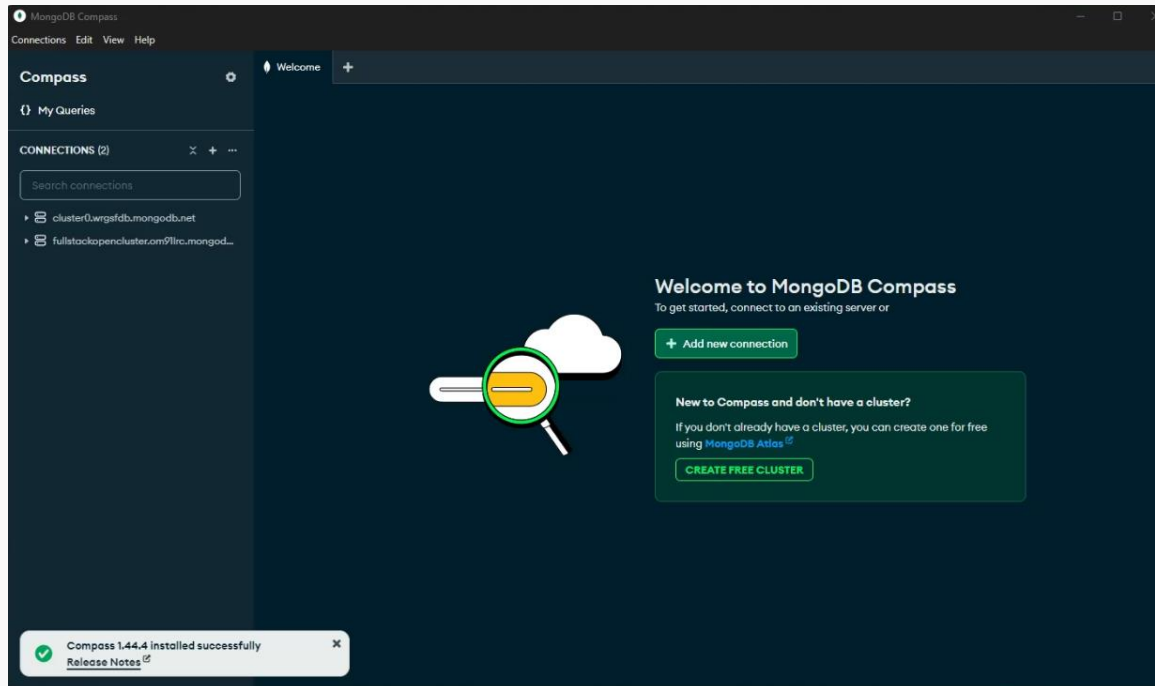
Esta configuración la dejamos así como está y le damos a Next



Nos aseguramos de que la opción de Install MongoDB Compass esté activa y le damos a Next. Esta es una app con interfaz gráfica que nos ayudará a ver los datos que tenemos guardados en la DB



Presionamos el botón Install



1) Vamos a hacer click en Add new Connection

New Connection ✕

Manage your connection settings

URI ⓘ Edit Connection String ☒

`mongodb://localhost:27017/`

Name **Color** No Color ▼

☐ **Favorite this connection**
Favoriting a connection will pin it to the top of your list of connections

> Advanced Connection Options

Cancel Save Connect

How do I find my connection string in Atlas?

If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.

[See example](#)

How do I format my connection string?

[See example](#)

Vamos a dejar la URI tal cual está, y podemos definir un nombre que querramos, Luego le damos a Connect

Compass

{ } My Queries

CONNECTIONS (3)

Search connections

- cluster0.wrgsfdb.mongodb.net
- fullstackopencluster.om91lrc.mongodb...
- luis
 - admin
 - config
 - local

Welcome luis +

Sort by Database Name

Database Name	Storage size	Collections	Indexes
admin	20.48 kB	1	1
config	24.58 kB	1	2
local	20.48 kB	1	1

Veremos algo como esto

Mongoose

Mongoose es una herramienta poderosa que se utiliza en Node.js para interactuar con bases de datos MongoDB de manera eficiente y estructurada. Básicamente, Mongoose te permite definir esquemas para tus datos, establecer relaciones entre ellos y realizar validaciones, todo dentro de un entorno basado en objetos. A nivel práctico, te facilita el trabajo porque abstrae muchos de los detalles que tendrías que manejar directamente si usás el driver nativo de MongoDB.

¿Cómo funciona?

MongoDB, por defecto, es bastante flexible en cuanto a la estructura de los documentos que almacena. Esa flexibilidad, si bien tiene sus ventajas, puede convertirse en un problema cuando querés mantener un nivel de consistencia en los datos. Acá es donde Mongoose entra en juego: te permite definir **schemas** (esquemas) que actúan como un modelo a seguir para los documentos en tu base de datos.

Que es un ODM?

Un **ODM** (Object-Document Mapper) es una herramienta que te permite interactuar con una base de datos NoSQL, como MongoDB, utilizando objetos de un lenguaje de programación orientado a objetos. En términos simples, lo que hace un ODM es mapear documentos de la base de datos (como los JSON que almacena MongoDB) a objetos en tu aplicación, facilitando la interacción entre tu código y la base de datos.

En el caso de **Mongoose**, que es uno de los ODM más populares para MongoDB en el ecosistema de Node.js, se encarga de traducir los objetos JavaScript a documentos de MongoDB y viceversa. Esto te permite trabajar con datos de la base de datos como si fueran simples objetos de tu aplicación, con todas las ventajas que eso implica: validaciones, relaciones entre datos, consultas más fáciles, y una estructura definida para mantener consistencia.

Ahora pasemos a crear nuestra app de Express y vamos a instalar las dependencias

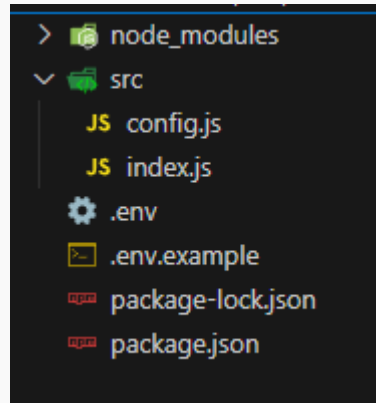
```
npm init -y
```

```
npm install dotenv mongoose express
```

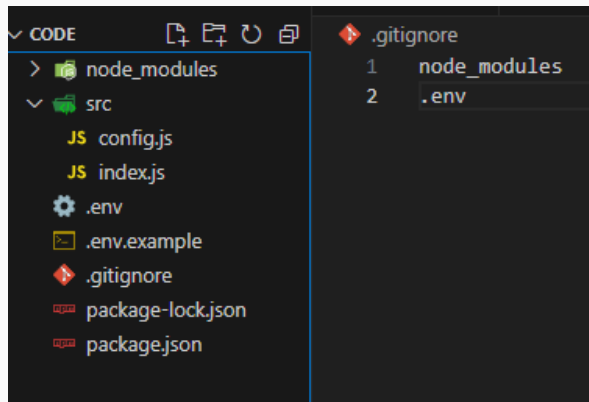
Configuramos nuestro package.json

```
{
  "name": "code",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "dev": "nodemon ./src/index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^16.4.5",
    "express": "^4.19.2",
    "mongoose": "^8.4.0"
  }
}
```

Vamos a crear 4 archivos `index.js` y `config.js` dentro de la carpeta `src` y los otros 2 serán uno con el nombre `.env` y `.env.example`



Ahora vamos a crear un `.gitignore`



Empecemos por el archivo `.env`

```
MONGO_DB_URI="mongodb://localhost:27017/portfolio"  
PORT=3000
```

Vamos a definir estas 2 variables donde la primera indicará el string que generamos en la nube y la otra será el puerto que estaremos utilizando

Ahora en `config.js`

```
import dotenv from "dotenv";  
  
dotenv.config()  
  
export default {  
  MONGO_DB_URI: process.env.MONGO_DB_URI,  
  PORT: process.env.PORT  
}
```

Vamos a exportar un objeto con nuestras variables

Vamos a levantar nuestro servidor para asegurarnos que estamos obteniendo nuestras variables.

En `app.js`

```
import express from 'express';
import config from './config.js';

const app = express();

app.listen(config.PORT, () => {
  console.log(`Listening on ${config.PORT}`)
});
```

Y si todo está OK, deberíamos ver en la consola que nuestra app está corriendo en el puerto 3000.

Ahora hablemos de **mongoose**

Mongoose es una biblioteca de modelado de datos de objetos (ODM) para MongoDB y Node.js. Proporciona una solución basada en esquemas para modelar los datos de la aplicación, ofreciendo una forma sencilla de definir y validar los datos y realizar operaciones CRUD.

Configurar la Conexión a MongoDB:

Vamos a crear un archivo llamado **connect.js** e importaremos **mongoose** y **config** para realizar la conexión a la base de datos en MongoDB:

```
import mongoose from 'mongoose';
import config from './config.js';

export default async function connectToDB() {
  try {
    await mongoose.connect(config.MONGO_DB_URI);
    console.log('Conectado a MongoDB Atlas');
  } catch (err) {
    console.error('Error al conectar a MongoDB Atlas:', err);
  }
}
```

Importamos la función para conectarse a la DB en index.js

```
import express from 'express';
import config from './config.js';
import connectToDB from './connect.js';

const app = express();

connectToDB()

app.listen(config.PORT, () => {
  console.log(`Listening on ${config.PORT}`)
});
```

Y ya tendríamos que ver en la consola el mensaje que pusimos para corroborar que la conexión está establecida

Esquemas y Modelos

Esquema (Schema):

- En Mongoose, un esquema es una estructura de datos que define la forma y los tipos de los documentos en una colección.
- Un esquema también puede definir validaciones, métodos y middlewares.

Modelo (Model):

Un modelo en Mongoose es una clase que se deriva de un esquema. Es responsable de crear y leer documentos de la base de datos MongoDB.

Los modelos son la representación de las colecciones en MongoDB.

Ejemplo

Creamos una carpeta Database, adentro otra carpeta models y dentro un archivo projectModel.js

```
import mongoose from "mongoose";

const projectSchema = new mongoose.Schema({
  nombre: {
    type: String,
    required: true,
    trim: true
  },
  privado: {
    type: Boolean,
    default: false
  },
  fecha: Date,
  tecnologias: {
    type: [String]
  },
  url: String
}, {
  timestamps: false // Para registrar el momento se deja en true
})

const Project = mongoose.model('Project', projectSchema)

export default Project;
```

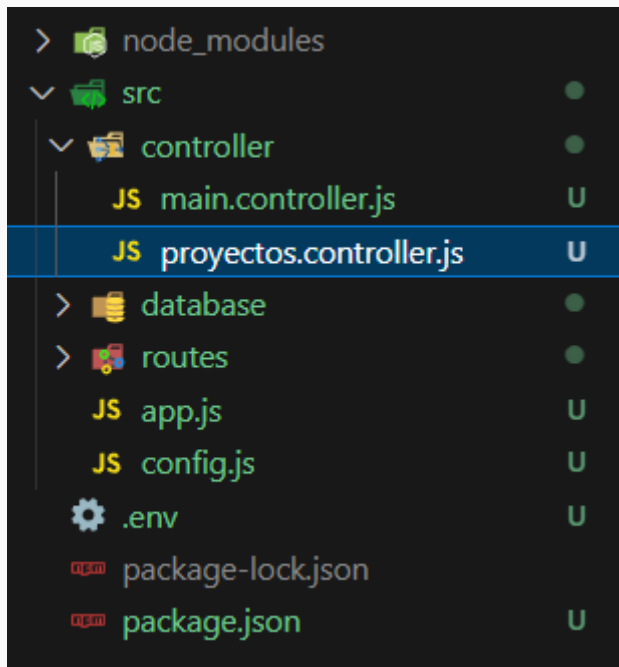

CRUD con mongoose

Ahora con el modelo que definimos en el ejemplo pasemos a ver como realizar un CRUD

Vamos a crear 2 carpetas más que serán controllers y routes

MODELS -> CONTROLLERS -> ROUTES

Esta será nuestra arquitectura básica. Ya definimos un modelo Project, ahora vamos al controlador y este por ahora se encargará de interactuar con la base de datos.



```
import Project from '../database/models/projectModel.js'

export const getProyectos = async (req, res) => {

  const { privado } = req.query
  let fieldPrivado = null
  const dataFromDB = await Project.find()

  if (privado === 'true') {
    fieldPrivado = true
    const filterArrayTrue = dataFromDB.filter(project => project.privado === fieldPrivado)
    return res.status(200).json(
      {
        status: 200,
        message: `Cantidad de proyectos ${filterArrayTrue.length}`,
        payload: filterArrayTrue
      }
    )
  }
  else if (privado === 'false') {
    fieldPrivado = false
    const filterArrayFalse = dataFromDB.filter(project => project.privado === fieldPrivado)
    console.log('first')
    return res.status(200).json(
      {
        status: 200,
        message: `Cantidad de proyectos ${filterArrayFalse.length}`,
        payload: filterArrayFalse
      }
    )
  }
  const cantidadProyectos = dataFromDB.length

  res.status(200).json(
    {
      status: 200,
      message: `Cantidad de proyectos ${cantidadProyectos}`,
      payload: dataFromDB
    }
  )
}
```

```
export const getProyectoById = async (req, res) => {
  const { id } = req.params
  // const proyecto = proyectos.find(project => project.id === parseInt(id))

  const proyecto = await Project.findById(id)

  if (!proyecto) {
    return res.status(404).json({ status: 404, message: `Proyecto de ID: ${id} no se ha encontrado` });
  }

  res.status(200).json({ status: 200, message: `Proyecto de ID: ${id}, se ha encontrado`, payload: proyecto });
}

export const updateProyecto = async (req, res) => {
  const { id } = req.params
  const { url, privado, fecha, tecnologias, nombre } = req.body

  // const proyecto = proyectos.find(project => project.id === parseInt(id))

  const updateProject = await Project.findByIdAndUpdate(id, { url, privado, fecha, tecnologias, nombre }, { new: true })

  if (!updateProject) {
    return res.status(404).json({ status: 404, message: `Proyecto de ID: ${id} no se ha encontrado` });
  }
  // Actualizando las propiedades que vengan por medio del body

  res.status(200).json({ status: 200, message: 'Proyecto actualizado', payload: updateProject })
}
```

```
export const crearProyecto = async (req, res) => {
  const { url, privado, fecha, tecnologias, nombre } = req.body;

  const nuevoProyecto = {
    url,
    privado,
    fecha,
    tecnologias,
    nombre
  };

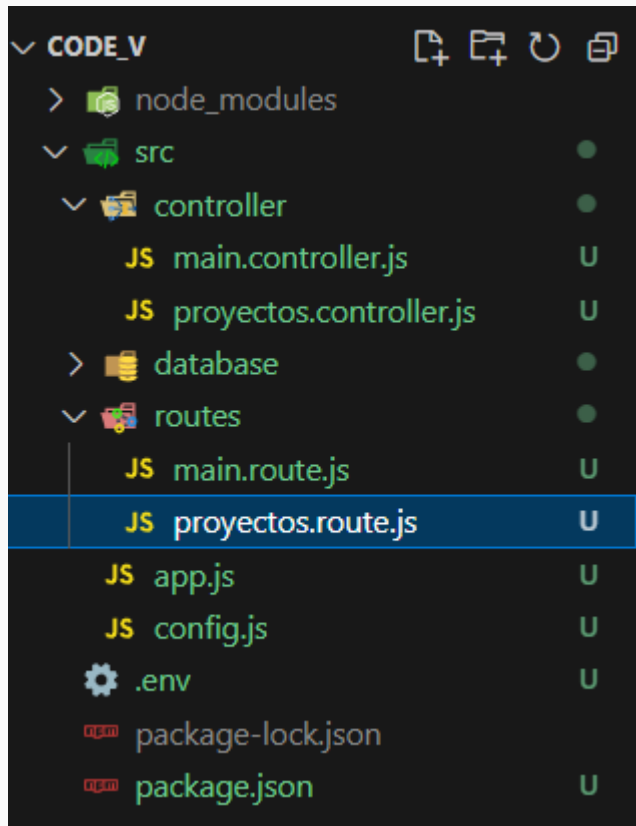
  const newProjectToDB = await Project.create(nuevoProyecto)

  // Enviamos una respuesta indicando que el proyecto fue creado exitosamente
  res.status(201).json({
    status: 201,
    message: 'Proyecto creado exitosamente',
    payload: newProjectToDB
  });
}

export const deleteProyecto = async (req, res) => {
  const { id } = req.params
  const projectDelete = await Project.findByIdAndDelete(id)

  if (!projectDelete){
    return res.status(404).json({ status: 404, message: `Proyecto de ID: ${id} no se ha encontrado` });
  }

  // Enviamos una respuesta indicando que el proyecto fue creado exitosamente
  res.status(201).json({
    status: 201,
    message: 'Proyecto eliminado exitosamente',
    payload: projectDelete
  });
}
```



```
import { Router } from "express";
import {
  crearProyecto,
  deleteProyecto,
  getProyectoById,
  getProyectos,
  updateProyecto
} from
"../controller/projectos.controller.js";

const router = Router();

router.get('/', getProyectos)
router.post('/', crearProyecto)
router.get('/:id', getProyectoById)
router.put('/:id', updateProyecto)
router.delete('/:id', deleteProyecto)

export default router
```

```
// app.js
import express from 'express';
import cors from 'cors';
import mainRoutes from './routes/main.route.js'
import proyectosRoutes from './routes/proyectos.route.js'
import { connectToDB } from './database/connect.js';

const app = express();

// Conectar a la DB
connectToDB()

app.use(express.json());
app.use(cors());

// Importar rutas
app.use('/', mainRoutes)
app.use('/proyectos', proyectosRoutes)

const PORT = 3000;

app.listen(PORT, ()=>console.log(`Listening on ${PORT}`));
```