**M3S20: Full CRUD con Mongoose: Modelos y Middlewares.**

**¡Hola, espero te encuentres muy bien!, nos encontramos en la semana 20 de nuestro Bootcamp.**

**En esta semana damos inicio a un nuevo tema y es Mongoose, es muy importante que prestes atención, toma nota en caso de ser necesario y así puedas continuar sin inconvenientes.**

**A continuación, se muestra un pequeño índice con el contenido que estaremos abordando esta semana. Iniciemos… 😃**

**ÍNDICE**

* [Full CRUD con Mongoose](https://github.com/U-Camp/BOOT-M3-SEM20#full-crud-con-mongoose)
  + [Modelos](https://github.com/U-Camp/BOOT-M3-SEM20#modelos)
  + [Middlewares](https://github.com/U-Camp/BOOT-M3-SEM20#middlewares)
  + [CRUD con Mongoose](https://github.com/U-Camp/BOOT-M3-SEM20#crud-con-mongoose)

**Full CRUD con Mongoose**

Hablemos de la conexión que se realiza entre MongoDB y el Backend.

Cuando trabajamos con ExpressJS, a través de nuestras rutas, podremos conectarnos a MongoDB usando una librería llamada mongoose.

Esta librería será nuestra principal embajadora para comunicarnos con la base de datos y poder realizar transferencias entre ellos.

Comencemos. Establezcamos nuestro ambiente de desarrollo básico.

Recuerda los pasos.

1. Crear nueva carpeta
2. Inicializar Node.js (npm init)
3. Establecer el script con "start" y ejecutando nodemon index.js
4. Instalar dependencia de Express npm install express
5. Crear un archivo llamado index.js donde estableceremos la configuración del servidor.

index.js

// 1. IMPORTACIONES

const express = require('express')

const app = express()

// 2. MIDDLEWARES

app.use(express.json());

// 3. RUTEO

app.get("/", (req, res) => {

})

app.post("/", (req, res) => {

})

app.put("/", (req, res) => {

})

app.delete("/", (req, res) => {

})

// 4. SERVIDOR

app.listen(3000, () => {

console.log("El servidor está de pie")

})

Ok. Crearemos una carpeta llamada models a nivel raiz. Incluiremos dentro un archivo llamado Guitarra.js.

|- models/

| |- Guitarra.js

|- node\_modules/

|- index.js

|- package.json

Como último paso, para poder comunicarnos con la base de datos, necesitaremos instalar una librería llamada mongoose.

En tu terminal: npm install mongoose

Listo, empecemos a construir el modelo.

**Modelos**

De acuerdo a la documentación de [Mongoose](https://mongoosejs.com/docs/models.html), los modelos son constructores que permiten crear un documento, bajo reglas, que se incorporarán posteriormente en una base de datos, puntualmente MongoDB.

Estas reglas se conocen como esquemas o Schema, el cual permiten mantener lineamientos para la creación específica de un documento.

Comencemos escribiendo este código.

Guitarra.js

// 1. IMPORTACIONES

const mongoose = require('mongoose')

// 2. SCHEMA

const guitarraSchema = mongoose.Schema({

nombre: {

type: String,

required: true

},

precio: {

type: Number

},

},

{

timestamps: true // Permite agregar la fecha en el que fue generado el documento.

}

)

// 3. MODELO

const Guitarra = mongoose.model('Guitarra', guitarraSchema)

// 4. EXPORTACIÓN

module.exports = Guitarra

* **Importaciones.** Cada vez que construyamos un archivo nuevo, necesitamos importar las librerías que se usarán para este archivo. En este caso utilizamos mongoosepara la creación del modelo.
* **Schema.** Cuando invocamos mongoose.Schema, necesitamos como primer argumento establecer las diferentes propiedades que deberá tener cada documento Guitarra en MongoDB.
* **Modelo.** Al invocar el modelo, confirmamos cómo se llamará la colección y anidamos el Schema construido anteriormente.
* **Exportación.** Exportamos el modelo para que pueda utilizarse en otros archivos de JavaScript.

Y eso es todo, regresemos a index.js

**Middlewares**

Vamos a instalar dos librerías adicional.

npm install dotenv cors

* **dotenv.** Es una librería que nos permite generar variables de entorno. A través de estas variables, contendremos datos sensibles.
* **cors.** Es una librería que nos permite gestionar el mecanismo de CORS (Cross-origin resource sharing), el cual consiste en restringir o permitir peticiones de dominios externos a la aplicación.

Generamos esta actualización de nuestro código:

// 1. IMPORTACIONES

const express = require('express')

const app = express()

const cors = require('cors') // IMPORTAMOS CORS

const Guitarra = require('./models/Guitarra') // NUESTRO MODELO PARA PERMITIR GENERAR O MODIFICAR GUITARRAS CON LA BASE DE DATOS

const connectDB = require('./config/db') // IMPORTAMOS UNA CARPETA PARA NUESTRA BASE DE DATOS, AÚN NO LA CREAMOS

// 2. MIDDLEWARES

// VARIABLES DE ENTORNO

require('dotenv').config()

// CONEXIÓN A DB

connectDB()

// Habilitar CORS

app.use(cors())

// BODY-PARSER

app.use(express.json());

// 3. RUTEO

app.get("/", (req, res) => {

})

app.post("/", (req, res) => {

})

app.put("/", (req, res) => {

})

app.delete("/", (req, res) => {

})

// 4. SERVIDOR

app.listen(3000, () => {

console.log("El servidor está de pie")

})

Con esto, nuestros middlewares están colocados correctamente.

Estos se encargarán de que, cuando llegue una petición de un cliente, evalúen si se tiene todo lo necesario para avanzar a las rutas.

Si todo es correcto, el siguiente paso es el ruteo.

**CRUD con Mongoose**

Crearemos primero un archivo llamado db.js que contendrá la conexión con la base de datos de MongoDB.

La generaremos en una carpeta nueva llamada config.

`config/db.js``

// IMPORTACIÓN DE LA LIBRERÍA DE MONGOOSE

const mongoose = require("mongoose")

// FUNCIÓN PARA REALIZAR UNA CONEXIÓN CON BASE DE DATOS

const connectDB = async () => {

try {

// CONEXIÓN A BASE DE DATOS

// IMPORTANTE UTILIZAR NUESTRA VARIABLE DE ENTORNO CON PROCESS.ENV

await mongoose.connect(process.env.MONGODB\_URI, {

useNewUrlParser: true,

useUnifiedTopology: true

})

// EXPRESAR EN TERMINAL QUE NUESTRA BASE DE DATOS FUE CONECTADA CORRECTAMENTE

console.log("Base de datos conectada")

} catch (error) {

console.log(error)

process.exit(1) // DETIENE LA APP POR COMPLETO

}

}

// EXPORTACIÓN DE LA FUNCIÓN PARA ACCESO

module.exports = connectDB

Posteriormente, crearemos un archivo llamado .env, el cual contendrá nuestras variables sensibles.

.env

PORT=3000

MONGODB\_URI=mongodb://localhost:27017/ucamp-guitarras

Observa que nuestra conexión MONGODB\_URI contiene al final el nombre de la base de datos ucamp-guitarras.

Excelente. Regresemos a nuestro archivo principal index.js y generemos el proceso CRUD completo.

Explóralo con calma.

index.js�

// 1. IMPORTACIONES

const express = require('express')

const app = express()

const cors = require('cors')

const connectDB = require('./config/db')

const Guitarra = require('./models/Guitarra')

// 2. MIDDLEWARES

// VARIABLES DE ENTORNO

require('dotenv').config()

// CONEXIÓN A DB

connectDB()

// Habilitar CORS

app.use(cors())

app.use(express.json());

// 3. RUTEO

app.get("/obtener-guitarras", async (req, res) => {

try {

const guitarras = await Guitarra.find({})

res.json({

guitarras

})

} catch (error) {

res.status(500).json({

msg: "Hubo un error obteniendo los datos"

})

}

})

app.post("/crear-guitarra", async(req, res) => {

const { nombre, precio } = req.body

try {

const nuevaGuitarra = await Guitarra.create({ nombre, precio })

res.json(nuevaGuitarra)

} catch (error) {

res.status(500).json({

msg: "Hubo un error creando la guitarra"

})

}

})

app.put("/actualizar-guitarra", async (req, res) => {

const { id, nombre, precio } = req.body

try {

const actualizacionGuitarra = await Guitarra.findByIdAndUpdate(id, { nombre, precio }, { new: true })

res.json(actualizacionGuitarra)

} catch (error) {

res.status(500).json({

msg: "Hubo un error actualizando la guitarra"

})

}

})

app.delete("/borrar-guitarra", async (req, res) => {

const { id } = req.body

try {

const guitarraBorrada = await Guitarra.findByIdAndRemove({\_id: id })

res.json(guitarraBorrada)

} catch (error) {

res.status(500).json({

msg: "Hubo un error borrando la guitarra especificada"

})

}

})

// 4. SERVIDOR

app.listen(process.env.PORT, () => {

console.log("El servidor está de pie")

})

Una vez que hayas analizado con calma cada ruta, es importante abrir POSTMAN y realizar las pruebas correspondientes (crear, leer, actualizar o borrar guitarras).

Veamos crear una guitarra.

Leer la guitarra creada.

Actualizar una guitarra.

Borrar una guitarra.

Haz diferentes pruebas y revisa en MongoDB Compass que se van creando y modificando.

Es un proceso que, conforme a la práctica, será más fácil de gestionar.

**Hemos llegado al final del contenido. ¿Qué te pareció?, ¿pudiste realizar el paso a paso indicado?, espero que todas tud respuestas sean positivas, recuerda que puedes consultarle a tus coaches en caso de ser necesario.**

**Te dejo por acá un podcast por si deseas escuchar sobre diversas maneras de sentirse feliz, [escuchalo aquí](https://open.spotify.com/episode/5R8pJTCsIx9DmAjgfCbaBm) 😃**

**¡Nos vemos pronto!**