**M3S21: Full CRUD con Mongoose: Autenticación (JWT)**

**¡Hola, nos encontramos nuevamente, pero ahora en nuestra semana 21!. ¿Cómo te sientes?, recuerda siempre por qué estas aquí y lo que te ha costado llegar, siempre podemos seguir adelante.**

**Esta semana, seguiremos trabajando Full CRUD con Mongoose, pero con nuevo contenido: Autenticación y autorización. Para continuar te dejo el índice con el contenido a trabajar, sigamos adelante...**

**ÍNDICE**

* [Full CRUD con Mongoose: Autenticación (JWT)](https://github.com/U-Camp/BOOT-M3-SEM21#full-crud-con-mongoose)
  + [Autenticación y Autorización](https://github.com/U-Camp/BOOT-M3-SEM21#autenticaci%C3%B3n-y-autorizaci%C3%B3n)
  + [JWT](https://github.com/U-Camp/BOOT-M3-SEM21#jwt)
  + [Proceso de Autenticación](https://github.com/U-Camp/BOOT-M3-SEM21#proceso-de-autenticaci%C3%B3n)
  + [Proceso de Autorización](https://github.com/U-Camp/BOOT-M3-SEM21#proceso-de-autorizaci%C3%B3n)

**Full CRUD con Mongoose**

Anteriormente estuvimos trabajando con el manejo de ruteo, a través de nuestro backend.

Sin embargo, deberemos también establecer permisos a los usuarios que nos visiten o pidan información de nuestro servidor.

Para poder identificar a estos usuarios, aprenderemos en esta sección el concepto de autenticación y autorización.

Podrás observar el ejercicio completo en la carpeta ejercicio de este repositorio.

**Autenticación y autorización**

Autenticación y autorización trabajan en conjunto pero, son dos actividades distintas.

**Autenticación.** Se refiere al proceso de verificar la identidad del usuario, revisando sus credenciales y confirmando con nuestro sistema su identidad.

**Autorización.** Es un proceso que, una vez realizada con éxito la autenticación, permite legitimizar la verificación y proceder a mostrar la información privada a este usuario. Dependiendo del usuario, otorgaremos los diferentes privilegios para que pueda interactuar con nuestro servidor.

**JWT**

JSON WEB TOKENS (JWT) es un estándar de seguridad en nuestra industria (también conocida como RFC 7519) que permite representar la interacción entre dos partes.

Dentro de este proceso, nos comunicaremos utilizando un objeto JSON, el cual se otorgará una vez que el usuario logre la autenticación al cliente. Usualmente los JWT se utilizan acompañándose de un texto secreto (encriptado a través de un algoritmo), que nos ayudará a validar que el token proviene del mismo servidor que lo creó/firmó.

Para comenzar, hablaremos de 3 escenarios que debemos diseñar en nuestro Backend:

| **Escenario** | **Proceso** | **Dato que retorna** |
| --- | --- | --- |
| Crear usuario | **Autenticación.** El usuario llena un formulario con sus datos (nombre, email, password), los envía al servidor, los procesamos e insertamos en la base de datos. | Autorización con token. En caso fallido, un mensaje de error |
| Iniciar sesión | **Autenticación.** El usuario llena un formulario con sus datos (email, password) y los envía al servidor. Verificamos que exista en base de datos y su password coincida. | Autorización con token. En caso fallido, un mensaje de error. |
| Verificar usuario | **Autorización.** El usuario entra a páginas privadas, las cuales require que el usuario esté registrado. Al acceder a la misma, se envía automáticamente, por el navegador, el token al servidor. Verificamos que no esté corrupto o expirado. Se descifra y buscamos la información en la base de datos con el id del usuario. | Autorización entregando los datos del usuario al cliente. Si existe un problema, se devuelve un mensaje de error. |

**Preparación de ambiente**

Utilizaremos el mismo ambiente y CRUD que hicimos en la sesión anterior. Lo haremos nuevamente, buscando repasar los conceptos.

Primero, preparemos los archivos nuevos en nuestra aplicación.

|- config/

| |- db.js

|

|- middleware/

| |- authorization.js

|

|- models/

| |- Guitarra.js

| |- Usuario.js

|

|- .env

|- .gitignore

|- index.js

|- package.json

**Instalación y dependencias**

Iniciemos con nuestro package.json

package.json

{

"name": "uclass-postman",

"version": "1.0.0",

"description": "",

"main": "index.js",

"scripts": {

"start": "nodemon index.js"

},

"keywords": [],

"author": "",

"license": "ISC",

"dependencies": {

}

}

Instalaremos estas dependencias, utilizando npm install [LIBRERÍA]:

* bcryptjs
* cors
* dotenv
* express
* jsonwebtoken
* mongoose

Una vez hecho esto, trabajemos en el código de cada archivo.

**Variables de entorno**

* .env contendrá estas variables, incluyendo una nueva:

PORT=3000

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

SECRET=UCAMP

**Bases de datos**

* db.js. Contendrá nuestra base de datos.

const mongoose = require("mongoose")

const connectDB = async () => {

try {

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

useNewUrlParser: true,

useUnifiedTopology: true

})

console.log("Base de datos conectada")

} catch (error) {

console.log(error)

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

}

}

module.exports = connectDB

**Modelos**

Generaremos dos modelos:

* El usuario contendrá toda la información de las personas que se registren, quieran iniciar sesión o naveguen en las páginas privadas de nuestra aplicación.
* Guitarra será nuestro CRUD previamente armado.

Usuario.js

const mongoose = require('mongoose')

const UsuarioSchema = mongoose.Schema({

username:{

type: String,

required: true

},

email: {

type: String

},

password:{

type: String,

required: true

}

}, {

timestamps: true

})

const Usuario = mongoose.model("Usuario", UsuarioSchema)

module.exports = Usuario

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

**Ignorar archivos (.gitignore)**

Dentro de nuestro archivo .gitignore, agregaremos nuestros archivos y carpetas a ignorar:

.gitignore

.env

node\_modules/

**Ejecución de servidor (index.js)**

Vamos a trabajar con una base de código:

// 1. IMPORTACIONES

const express = require('express')

const app = express()

const cors = require('cors')

const bcryptjs = require('bcryptjs')

const jwt = require('jsonwebtoken')

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

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

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

// 2. MIDDLEWARES

// VARIABLES DE ENTORNO

require('dotenv').config()

// CONEXIÓN A DB

connectDB()

// Habilitar CORS

app.use(cors())

app.use(express.json());

// 3. RUTEO

// A. GUITARRAS

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"

})

}

})

// B. USUARIOS

// CREAR UN USUARIO

// INICIAR SESIÓN

// VERIFICAR USUARIO

// 4. SERVIDOR

app.listen(process.env.PORT, () => console.log("El servidor está de pie"))

Listo. Con esto, comencemos el proceso de autenticación.

**Recuerda que si tienes mucho tiempo en la computadora, es bueno detener unos minutos, ir por agua, al baño y estirar nuestro cuerpo, esto nos ayuda, despeja y aumenta nuestra atención para poder continuar. ¡Tomaté 5 min ⌚ para ir al baño, asomarte a la ventana, estirarte y continuar!**

**¿Te estiraste, tomaste agua?, ¿cómo te sientes? espero que mucho mejor para continuar con nuestro contenido.**

**Proceso de autenticación**

En este proceso, cada vez que el usuario logre crear o mostrar sus credenciales correctamente, le daremos un token.

**Crear usuario**

Vamos a concentranos en index.js sobre las rutas.

Añadiremos esta ruta de crear usuario:

index.js

//...

// B. USUARIOS

// CREAR UN USUARIO

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

// OBTENER USUARIO, EMAIL Y PASSWORD DE LA PETICIÓN

const { username, email, password } = req.body

try {

// GENERAMOS FRAGMENTO ALEATORIO PARA USARSE CON EL PASSWORD

const salt = await bcryptjs.genSalt(10)

const hashedPassword = await bcryptjs.hash(password, salt)

// CREAMOS UN USUARIO CON SU PASSWORD ENCRIPTADO

const respuestaDB = await Usuario.create({

username,

email,

password: hashedPassword

})

// USUARIO CREADO. VAMOS A CREAR EL JSON WEB TOKEN

// 1. EL "PAYLOAD" SERÁ UN OBJETO QUE CONTENDRÁ EL ID DEL USUARIO ENCONTRADO EN BASE DE DATOS.

// POR NINGÚN MOTIVO AGREGUES INFORMACIÓN CONFIDENCIAL DEL USUARIO (SU PASSWORD) EN EL PAYLOAD.

const payload = {

user: {

id: respuestaDB.\_id

}

}

// 2. FIRMAR EL JWT

jwt.sign(

payload, // DATOS QUE SE ACOMPAÑARÁN EN EL TOKEN

process.env.SECRET, // LLAVE PARA DESCIFRAR LA FIRMA ELECTRÓNICA DEL TOKEN,

{

expiresIn: 360000 // EXPIRACIÓN DEL TOKEN

},

(error, token) => { // CALLBACK QUE, EN CASO DE QUE EXISTA UN ERROR, DEVUELVA EL TOKEN

if(error) throw error

res.json({

token

})

}

)

} catch (error) {

return res.status(400).json({

msg: error

})

}

})

Ve leyendo cada línea de código.

Una vez comprendido el algoritmo, pruébalo en Postman.

Lo que vas a obtener de retorno es un token.

**Iniciar sesión**

Vamos a permitir al usuario que pueda, a través de su correo y contraseña, demostrar que esa persona es la que se registro al inicio.

// INICIAR SESIÓN

app.post("/usuario/iniciar-sesion", async(req, res) => {

// OBTENEMOS EL EMAIL Y EL PASSWORD DE LA PETICIÓN

const {email, password} = req.body

try {

// ENCONTRAMOS UN USUARIO

let foundUser = await Usuario.findOne({email})

// SI NO HUBO UN USUARIO ENCONTRADO, DEVOLVEMOS UN ERROR

if(!foundUser){

return res.status(400).json({msg: "El usuario no existe"})

}

// SI TODO OK, HACEMOS LA EVALUACIÓN DE LA CONTRASEÑA ENVIADA CONTRA LA BASE DE DATOS

const passCorrecto = await bcryptjs.compare(password, foundUser.password)

// SI EL PASSWORD ES INCORRECTO, REGRESAMOS UN MENSAJE SOBRE ESTO

if(!passCorrecto){

return await res.status(400).json({msg: "Password incorrecto"})

}

// SI TODO CORRECTO, GENERAMOS UN JSON WEB TOKEN

// 1. DATOS DE ACOMPAÑAMIENTO AL JWT

const payload = {

user: {

id: foundUser.id

}

}

// 2. FIRMA DEL JWT

jwt.sign(

payload,

process.env.SECRET,

{

expiresIn: 3600000

},

(error, token) => {

if(error) throw error;

//SI TODO SUCEDIÓ CORRECTAMENTE, RETORNAR EL TOKEN

res.json({token})

})

} catch (error) {

res.json({

msg: "Hubo un error",

error

})

}

})

Revisa el algoritmo. Prueba en Postman.

Con estas dos rutas, ya permitimos al usuario poder obtener un token.

Más adelante, cuando se trabaje desde un navegador, lo guardaremos. Por ahora, lo importante es que la respuesta devuelva el token.

Avancemos con la autorización.

**Proceso de autorización**

Para este proceso, necesitamos primero generar un middleware que verificará el token que el cliente debe enviar a través de Headers.

Cada petición que suceda, se revisará ese token y pasará por un proceso de validación, a través de este archivo.

El archivo se llama authorization.js y lo anclaremos dentro de la nueva ruta que crearemos más adelante.

authorization.js

const jwt = require('jsonwebtoken')

module.exports = (req, res, next) => {

// EXTRAEMOS EL TOKEN QUE VIENE DESDE LA PETICIÓN

const token = req.header('x-auth-token')

// SI NO HAY TOKEN, RETORNAMOS UN ERROR

if(!token) {

return res.status(401).json({

msg: "No hay token, permiso no válido"

})

}

try {

// CONFIRMAMOS LA VERIFICACIÓN DEL TOKEN A TRAVÉS DE LA LIBRERÍA DE JWT

const openToken = jwt.verify(token, process.env.SECRET)

// SI TODO ESTÁ CORRECTO, A LA PETICIÓN LE ANCLAMOS UNA PROPIEDAD ADICIONAL CON EL TOKEN DESCIFRADO

req.user = openToken.user

// NEXT, AL INVOCARSE, PERMITE AVANZAR A LA SIGUIENTE FUNCIÓN

next()

} catch (error) {

res.json({

msg: "Hubo un error",

error

})

}

}

Una vez hecho esto, anclemos a la ruta nueva este middleware.

index.js

// 1. IMPORTACIONES

// ...

// AGREGAMOS EL ARCHIVO DE AUTHORIZATION.JS

const auth = require('./middleware/authorization')

// ...

// VERIFICAR USUARIO

// COMO OBSERVACIÓN, ESTAMOS EJECUTANDO EL MIDDLEWARE DE AUTH (AUTORIZACIÓN) ANTES DE ACCEDER

// A LA RUTA PRINCIPAL

app.post("/usuario/verificar-usuario", auth, async (req, res) => {

// ... AQUÍ DESARROLLAREMOS LA RUTA

})

**Verificar usuario**

Comencemos con el desarrollo de la ruta que permita la verificación.

app.get("/usuario/verificar-usuario", auth, async (req, res) => {

try {

// CONFIRMAMOS QUE EL USUARIO EXISTA EN BASE DE DATOS Y RETORNAMOS SUS DATOS, EXCLUYENDO EL PASSWORD

const usuario = await Usuario.findById(req.user.id).select('-password')

res.json({usuario})

} catch (error) {

// EN CASO DE HERROR DEVOLVEMOS UN MENSAJE CON EL ERROR

res.status(500).json({

msg: "Hubo un error",

error

})

}

})

Para poder observar que lo hicimos correctamente, revisemos en Postman.

Es importante que establezcas los Headers para que podamos verificar la identidad desde el token.

Y con esto, terminamos la autenticación y autorización de un Backend a través de JWT.

**¡Muy bien, llegamos al final del contenido de esta semana! recuerda practicar lo que trabajamos en las semanas anteriores y en esta para que repases el contenido visto. Recuerda mientras más practiques más rápido serás y mejor desempeño tendrás.**

**¡Nos vemos pronto!**