En la carpeta sequelize-rest-api\src\routes\ crear auth.routes.js y poner dos rutas, una para registrar usuarios y otra para logearse en la app

import { Router } from "express";

const router = Router()

router.post('/registrar')

router.post('/login')

export default router

En sequelize-rest-api\src\models crear el modelo para Empleado.js

import { DataTypes } from 'sequelize'

import { sequelize } from '../database/database.js'

export const Empleado = sequelize.define('empleado', {

        id\_empleado: {

            type: DataTypes.BIGINT,

            primaryKey: true,

            allowNull: false,

            unique: true

        },

        password\_encriptado: {

            type: DataTypes.STRING,

            allowNull: false

        },

        id\_perfil: {

            type: DataTypes.BIGINT,

            allowNull: false,

        }

    },

    {

        timestamps: false

})

Y para Perfil.js

import { DataTypes } from 'sequelize'

import { sequelize } from '../database/database.js'

import { Empleado } from './Empleado.js'

export const Perfil = sequelize.define('sys\_cat\_perfil', {

    id\_sys\_perfil: {

        type: DataTypes.BIGINT,

        primaryKey: true,

        //autoIncrement: true,

        allowNull: false,

        unique: true

    },

    sys\_perfil: {

        type: DataTypes.CHAR(100),

        allowNull: false

    },

    desc\_sys\_perfil: {

        type: DataTypes.CHAR(250),

        allowNull: false,

    },

    autoridad: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_practicas\_campo: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_practicas\_campo: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_encuesta\_equidad\_genero:{

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_becas: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_tutores: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_sal\_retorno\_bienes: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_servicios: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_usuarios: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_encuesta\_habitos\_alimenticios: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_encuesta\_preferencias\_deportivas: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_encuesta\_perfil\_aprendizaje: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_encuesta\_perfil\_pensamiento: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_buzon: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_adquisiciones: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_prestamos: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_bienes: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_prestamos\_bicicletas: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_soporte: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_violencia\_genero: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_servicio\_social: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_informes\_docencia: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

    app\_presupuesto: {

        type: DataTypes.BOOLEAN,

        defaultValue: false

    },

},

{

    timestamps: false

})

Perfil.hasMany( Empleado, { foreignKey: 'id\_perfil',

sourceKey: 'id\_sys\_perfil'})

Empleado.belongsTo(Perfil,{foreignKey: 'id\_perfil',

tergetId: 'id\_sys\_perfil'})

En la carpeta sequelize-rest-api\src\controllers\ crear auth.contoller.js crear dos métodos: uno para login y otro para el registro de usuarios

import {Empleado} from '../models/Empleado.js'

export const registrar = async (req, res) => {res.json('registrar empleado')}

export const login = async (req, res) => {res.json('logear empleado')}

En sequelize-rest-api\src\app.js colocar la ruta para autenticación, agregar /partidas a la de partida

//configuración del servidor express

import express from 'express'

import partidasRoutes from './routes/partidas.routes.js'

import authRoutes from './routes/auth.routes.js'

import cors from 'cors'

const app = express()

//middlewares

app.use(express.json())

app.use(cors())

app.use('/partidas',partidasRoutes)

app.use('/auth',authRoutes)

app.use((err, req, res, next) => {

    return res.json({message: err.message})

})

export default app

corregir en sequelize-rest-api\src\routes\partidas.routes.js eliminando **/partidas** de todas las rutas

import { Router } from "express";

import {getPartidas, addPartida, updatePartida, deletePartida, getPartida, getPartidaByClave} from "../controllers/partidas.controller.js"

const router = Router()

router.get('/',getPartidas)

router.post('/',addPartida)

router.put('/:id',updatePartida)

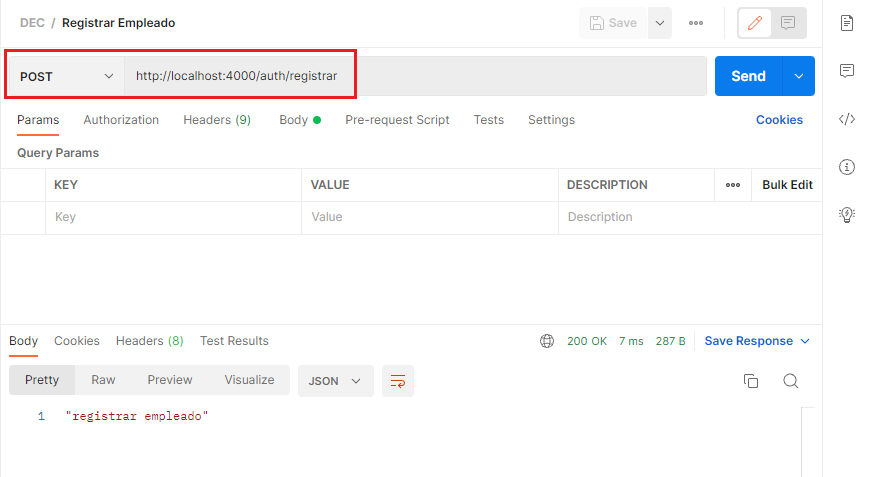
router.delete('/:id',deletePartida)

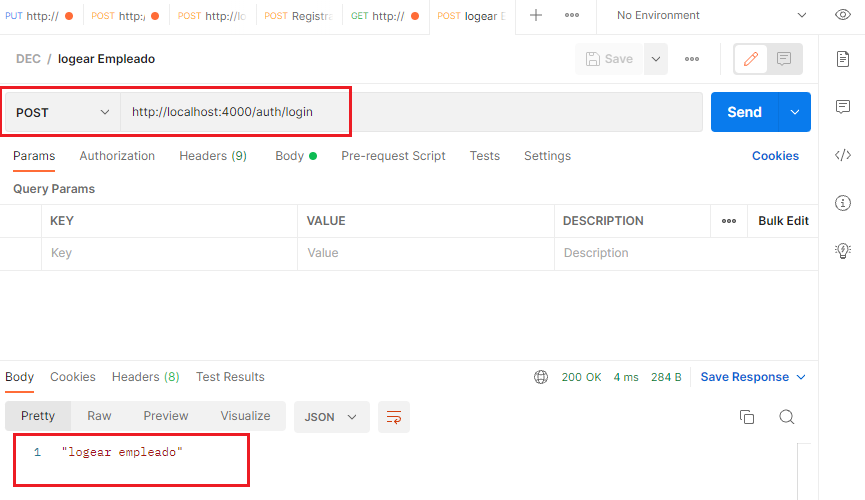
router.get('/:id',getPartida)

router.get('/clave/:clave',getPartidaByClave)

export default router

Probar las dos nuevas rutas para autenticación





Instalar bcrypt y jwt se usarán para cifrar el password del usuario

npm i bcryptjs jsonwebtoken

<https://openbase.com/js/bcryptjs/documentation>

<https://jwt.io/>

Para sequelize-rest-api\src\models\Empleado.js se redefinió el modelo como clase <https://sequelize.org/docs/v6/core-concepts/model-basics/#taking-advantage-of-models-being-classes> Con el propósito de añadir unos métodos para manejar el password

import { DataTypes, Model } from 'sequelize'

import { sequelize } from '../database/database.js'

import bcrypt from 'bcryptjs'

class Empleado extends Model {

    async encriptarPass(password) {

        const salt = await bcrypt.genSalt(10)

        return await bcrypt.hash(password, salt)

     }

     async compararPass(password, passwordEncryptado) {

         return await bcrypt.compare(password, passwordEncryptado)

       }

}

Empleado.init({

    id\_empleado: {

        type: DataTypes.BIGINT,

        primaryKey: true,

        allowNull: false,

        unique: true

    },

    password\_encriptado: {

        type: DataTypes.STRING,

        allowNull: false

    },

    id\_perfil: {

        type: DataTypes.BIGINT,

        allowNull: false,

    }

}, {

  // Other model options go here

  timestamps: false,

  sequelize, // We need to pass the connection instance

  modelName: 'Empleado' // We need to choose the model name

});

export const Usuario= sequelize.models.Empleado

En sequelize-rest-api\src\controllers\auth.contoller.js se modifica el método registrar

import { Usuario } from '../models/Empleado.js'

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

    const {id\_empleado, password, id\_perfil} = req.body

    const usuario=new Usuario({id\_empleado, id\_perfil})

    usuario.password\_encriptado= await usuario.encriptarPass(password)

    console.log(usuario)

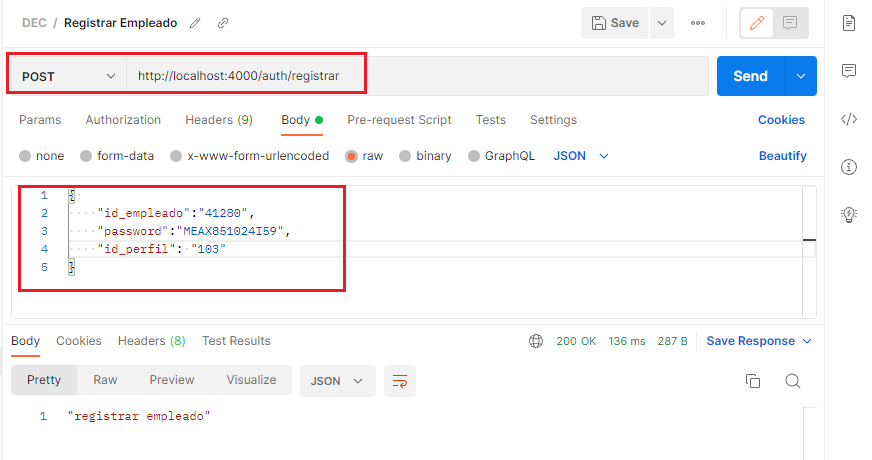
    console.log(usuario.password\_encriptado.length)

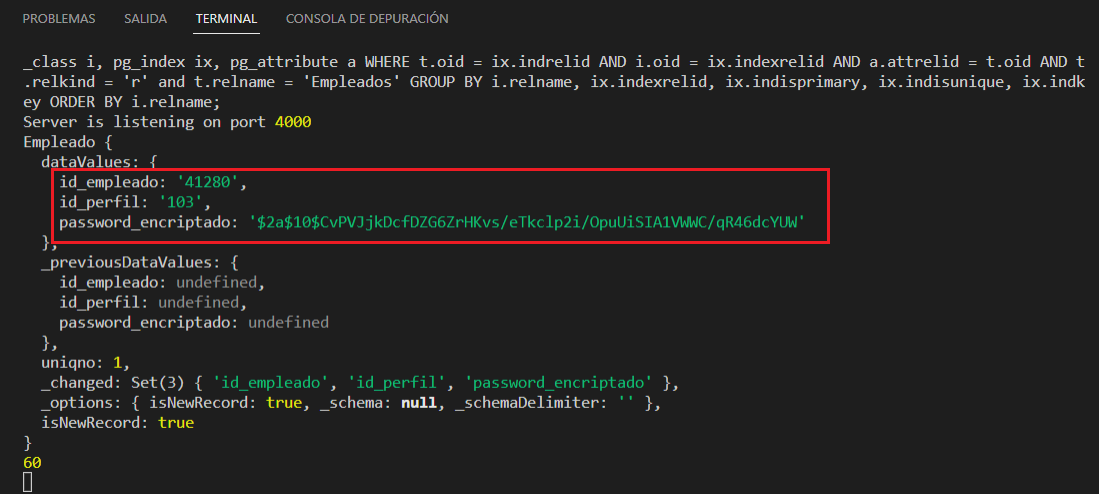
    res.json('registrar empleado')

}

export const login = async (req, res) => {res.json('logear empleado')}

Probar que está encriptando el password





Ahora se va a generar un token con los datos del usuario que será su pase desde el front end para poder pedirle cosas al backend y comprobar si le puedo proporcionar o no la información solicitada.

Importar jsonwebtoken y se utilizará su método sign que necesita 3 argumentos, el segundo argumento es una palabra secreta que se colocaría en sequelize-rest-api\.env

import { Usuario } from '../models/Empleado.js'

import jwt from 'jsonwebtoken'

import 'dotenv/config'

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

    const {id\_empleado, password, id\_perfil} = req.body

    const usuario=new Usuario({id\_empleado, id\_perfil})

    usuario.password\_encriptado= await usuario.encriptarPass(password)

   try {

    const usuarioGuardado=await usuario.save()

    const token=jwt.sign({id: usuarioGuardado.id\_empleado},process.env.SECRET,{expiresIn: 86400})

    res.json(token)

   } catch (error) {

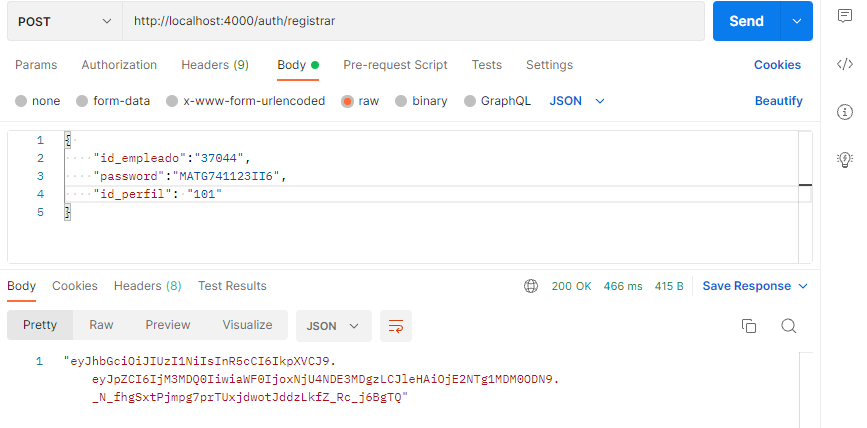
    console.log(error)

   }

}

export const login = async (req, res) => {res.json('logear empleado')}

Probar



Modificar en sequelize-rest-api\src\controllers\auth.contoller.js el método login

import { Usuario } from '../models/Empleado.js'

import { Perfil } from '../models/Perfil.js'

import jwt from 'jsonwebtoken'

import 'dotenv/config'

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

    const {id\_empleado, password, id\_perfil} = req.body

    const usuario=new Usuario({id\_empleado, id\_perfil})

    usuario.password\_encriptado= await usuario.encriptarPass(password)

   try {

    //probablemente no sea necesario verificar el perfil, pues se seleccionará de un select

    const perfil = await Perfil.findByPk(id\_perfil)

    if (!perfil)

        return res.status(404).json({ message: "El perfil indicado no existe" })

    const usuarioGuardado=await usuario.save()

    const token=jwt.sign({id: usuarioGuardado.id\_empleado},process.env.SECRET,{expiresIn: 86400})

    res.json(token)

   } catch (error) {

    console.log(error)

   }

}

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

    const usuarioExistente = await Usuario.findByPk(req.body.id\_empleado)

    if(!usuarioExistente)

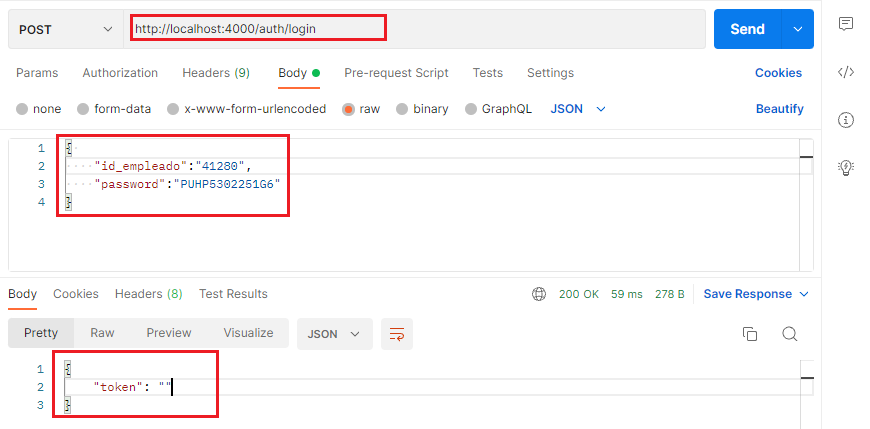
        return res.status(404).json({message: "Usuario o password incorrecto"})

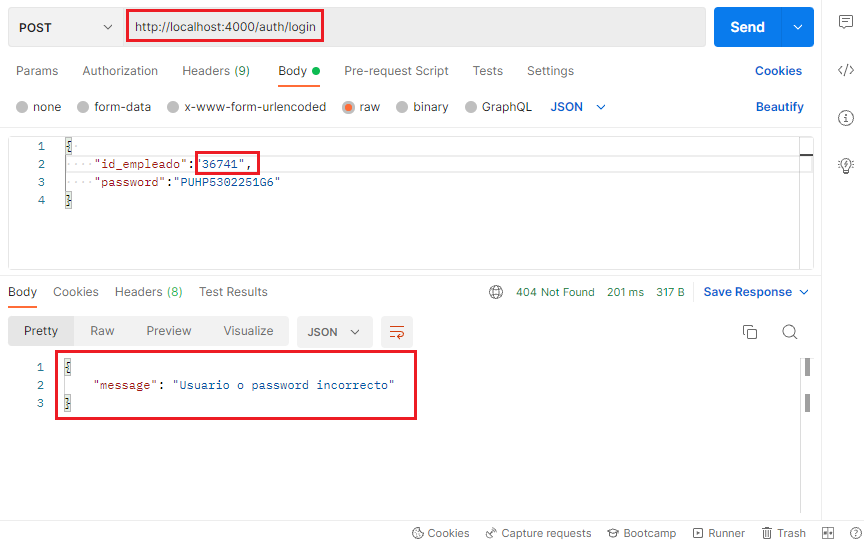
    console.log(usuarioExistente)

    res.json({token:''})

}

Probar





Verificar el password con el método de instancia compararPass creado anteriormente

import { Usuario } from '../models/Empleado.js'

import { Perfil } from '../models/Perfil.js'

import jwt from 'jsonwebtoken'

import 'dotenv/config'

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

    const {id\_empleado, password, id\_perfil} = req.body

    const usuario=new Usuario({id\_empleado, id\_perfil})

    usuario.password\_encriptado= await usuario.encriptarPass(password)

   try {

    //probablemente no sea necesario verificar el perfil, pues se seleccionará de un select

    const perfil = await Perfil.findByPk(id\_perfil)

    if (!perfil)

        return res.status(404).json({ message: "El perfil indicado no existe" })

    const usuarioGuardado=await usuario.save()

    const token=jwt.sign({id: usuarioGuardado.id\_empleado},process.env.SECRET,{expiresIn: 86400})

    res.json(token)

   } catch (error) {

    console.log(error)

   }

}

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

    const usuarioExistente = await Usuario.findByPk(req.body.id\_empleado)

    if (!usuarioExistente)

        return res.status(404).json({ message: "Usuario o password incorrecto" })

    const passwordCorrecto = await usuarioExistente.compararPass(

        req.body.password, usuarioExistente.password\_encriptado)

    if (!passwordCorrecto)

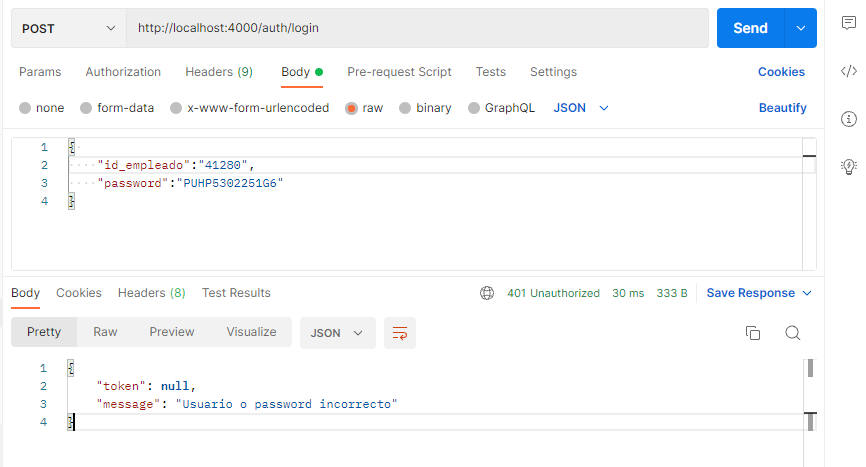
        return res.status(401).json({ token: null, message: "Usuario o password incorrecto" })

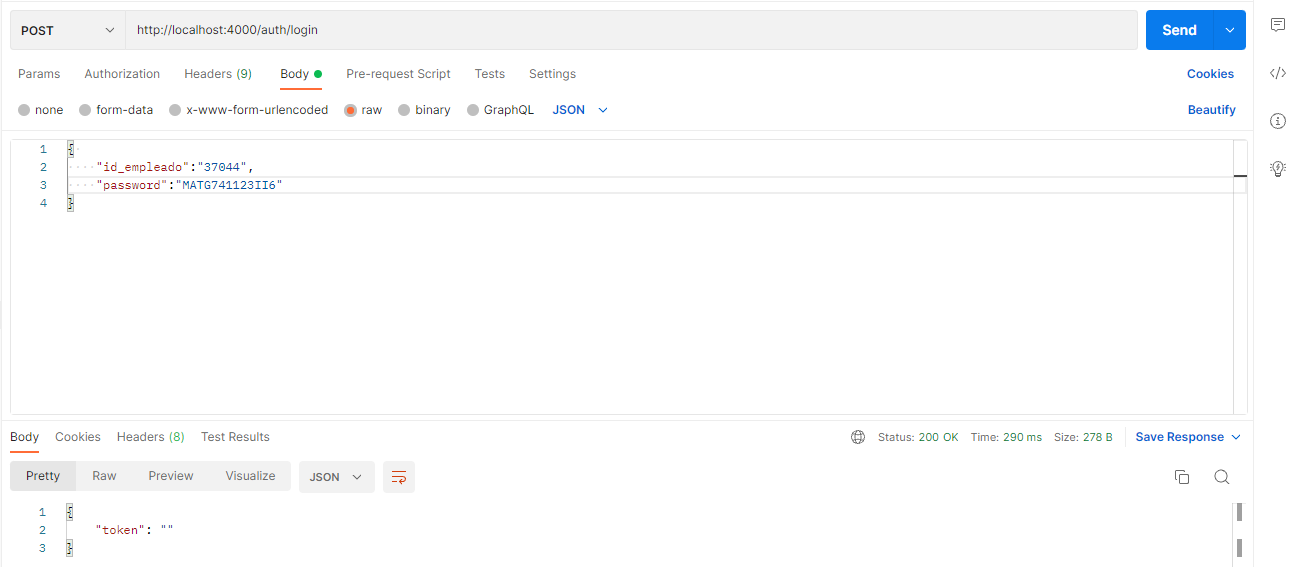
    console.log(usuarioExistente)

    res.json({ token: '' })

}

Probar





Crear el token y enviarlo como respuesta

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

    const usuarioExistente = await Usuario.findByPk(req.body.id\_empleado)

    if (!usuarioExistente)

        return res.status(404).json({ message: "Usuario o password incorrecto" })

    console.log(usuarioExistente.password\_encriptado)

    console.log(req.body.password)

    const passwordCorrecto = await usuarioExistente.compararPass(

        req.body.password, usuarioExistente.password\_encriptado)

    if (!passwordCorrecto)

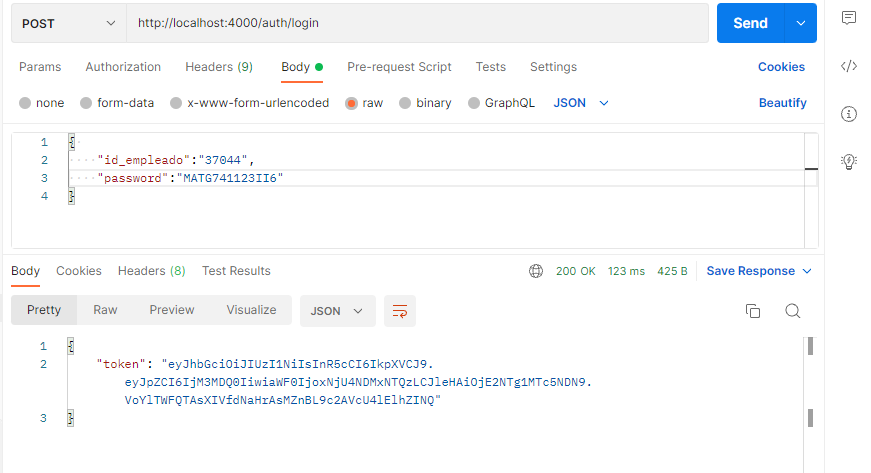
        return res.status(401).json({ token: null, message: "Usuario o password incorrecto" })

    console.log(usuarioExistente)

    const token=jwt.sign({id: usuarioExistente.id\_empleado},process.env.SECRET,{expiresIn: 86400})

    res.json({ token })

}



Crear en sequelize-rest-api\src la carpeta middlewares y dentro el archivo authJWT.js que se usará para verificar que se está enviando un token

export const verificarToken= async(req, res, next) => {

    const token = req.headers["x-access-token"]

    console.log(token)

}

Crear en sequelize-rest-api\src\middlewares el archivo index.js

import {verificarToken} from './authJWT.js'

export {verificarToken}

En sequelize-rest-api\src\routes\partidas.routes.js importar verificarToken e invocarlo antes de añadir una partida

import { Router } from "express";

import {getPartidas, addPartida, updatePartida, deletePartida, getPartida, getPartidaByClave} from "../controllers/partidas.controller.js"

import { verificarToken } from "../middlewares/index.js";

const router = Router()

router.get('/',getPartidas)

router.post('/',verificarToken, addPartida)

router.put('/:id',updatePartida)

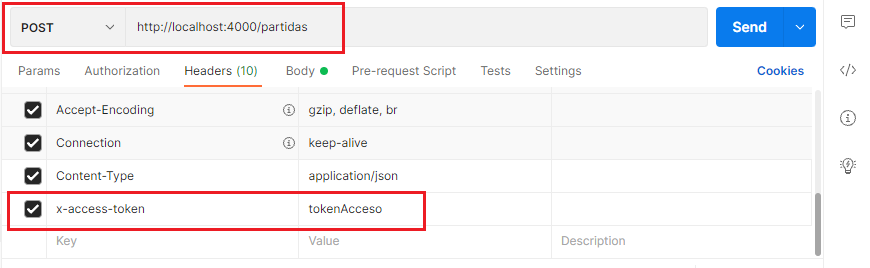
router.delete('/:id',deletePartida)

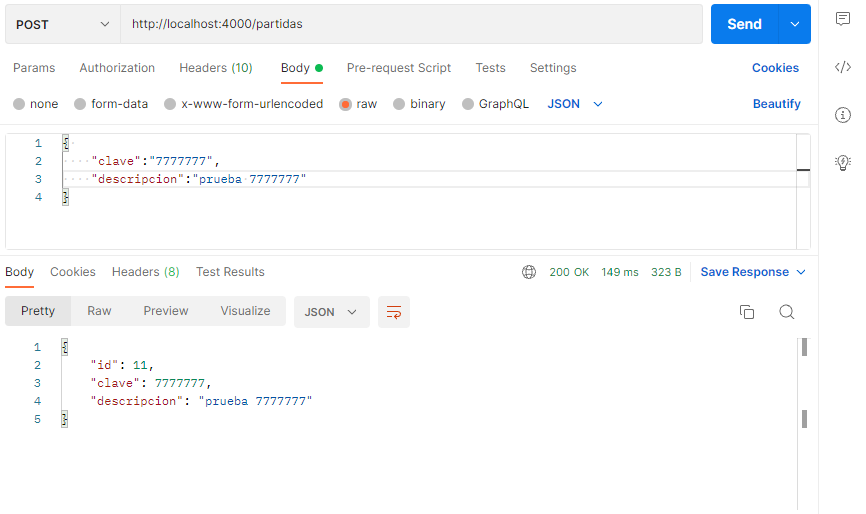
router.get('/:id',getPartida)

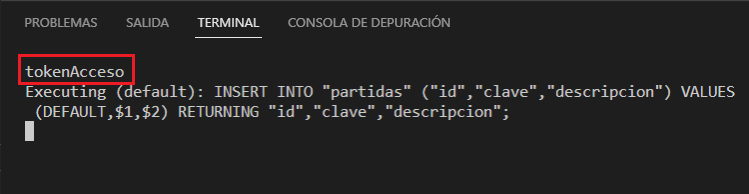
router.get('/clave/:clave',getPartidaByClave)

export default router

Probar añadiendo x-access-token en los headers







Modificar sequelize-rest-api\src\middlewares\authJWT.js para verificar que se está recibiendo el token y decodificarlo para poder ver el valor del id con que se logeó el usuario

import jwt from 'jsonwebtoken'

import 'dotenv/config'

export const verificarToken= async(req, res, next) => {

    const token = req.headers["x-access-token"]

    console.log(token)

    if(!token) return res.status(403).json({message: "No está autorizado para realizar esta operación"})

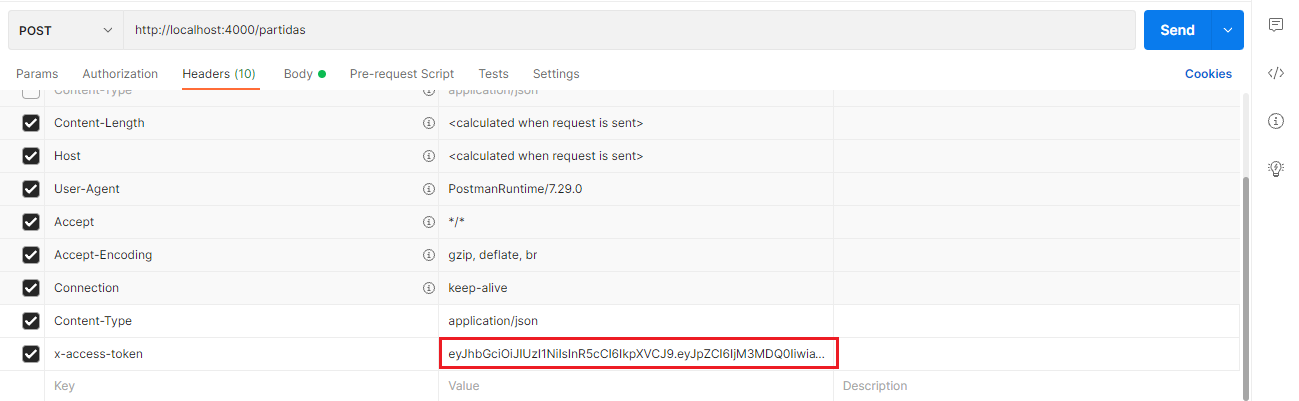
    const decodificado = jwt.verify(token, process.env.SECRET)

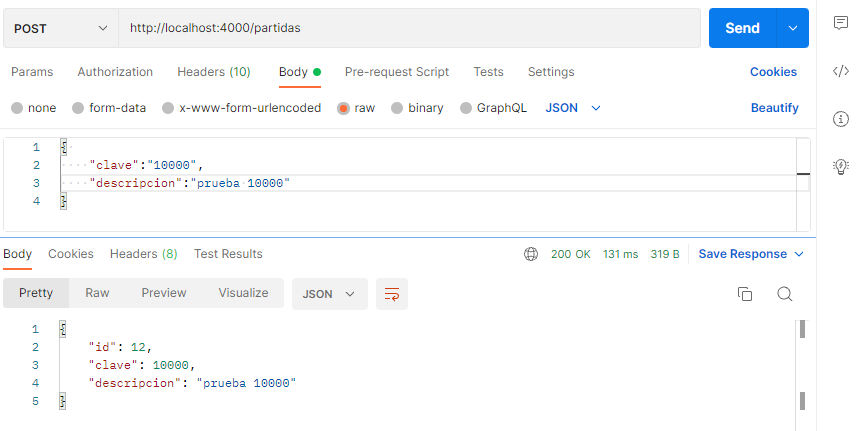
    console.log(decodificado)

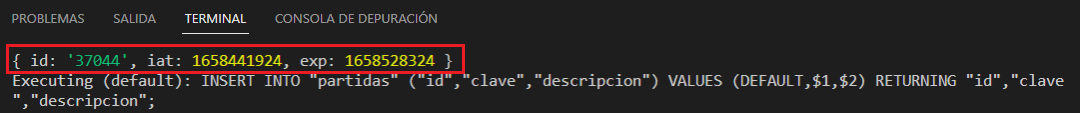
    next()

}

Probar enviando un token válido de los que se obtuvieron en anteriores pruebas







Ahora que se verificó el token, se procede a guardar el id del usuario en la petición

import jwt from 'jsonwebtoken'

import 'dotenv/config'

import { Usuario } from '../models/Empleado'

export const verificarToken= async(req, res, next) => {

    const token = req.headers["x-access-token"]

    console.log(token)

    if(!token) return res.status(403).json({message: "No está autorizado para realizar esta operación"})

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

    req.idUsuario=decodificado.id

    const usuario = await Usuario.findById(req.idUsuario, {password: 0})

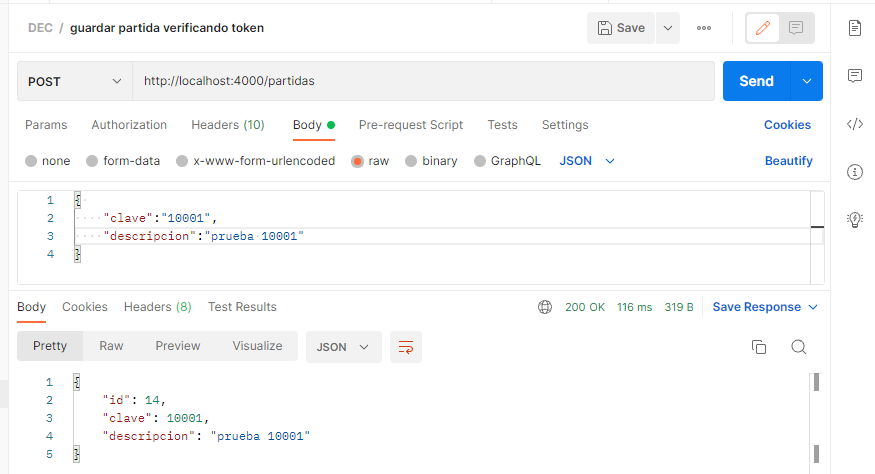
    if(!usuario)

        return res.status(404).json({message: 'No está autorizado para realizar esta operación'})

    next()

}

Probar



Añadir la verificación a el borrado y modificación de las partidas en sequelize-rest-api\src\routes\partidas.routes.js.

import { Router } from "express";

import {getPartidas, addPartida, updatePartida, deletePartida, getPartida, getPartidaByClave} from "../controllers/partidas.controller.js"

import { verificarToken } from "../middlewares/index.js";

const router = Router()

router.get('/',getPartidas)

router.post('/',verificarToken, addPartida)

router.put('/:id',verificarToken,updatePartida)

router.delete('/:id',verificarToken,deletePartida)

router.get('/:id',getPartida)

router.get('/clave/:clave',getPartidaByClave)

export default router

En sequelize-rest-api\src\middlewares\authJWT.js meter el código de la validación en un try y enviar un mensaje de error en caso que falle.

import jwt from 'jsonwebtoken'

import 'dotenv/config'

import { Usuario } from '../models/Empleado.js'

export const verificarToken= async(req, res, next) => {

    try {

        const token = req.headers["x-access-token"]

        if(!token) return res.status(403).json({message: "No está autorizado para realizar esta operación"})

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

        req.idUsuario=decodificado.id

        const usuario = await Usuario.findByPk(req.idUsuario, {password: 0})

        if(!usuario)

            return res.status(403).json({message: 'No está autorizado para realizar esta operación'})

        next()

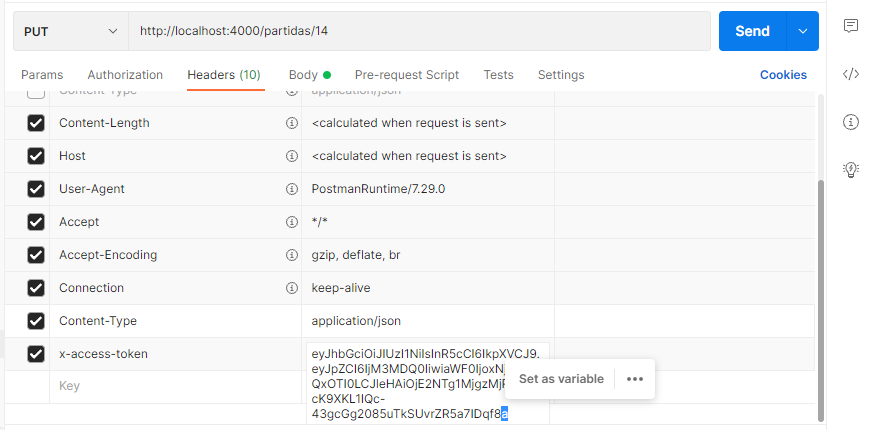
    } catch (error) {

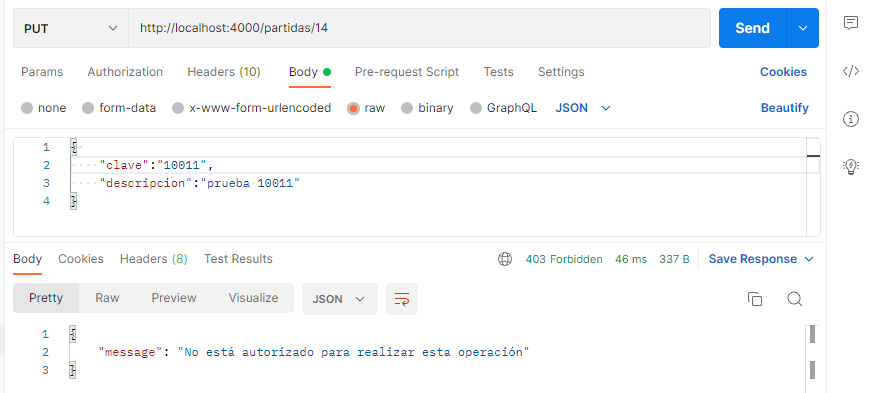
        return res.status(403).json({message: 'No está autorizado para realizar esta operación'})

    }

}

Probar cambiando algún caracter del token que se envía





Finalmente se verificarán los perfiles (adicional a eso se debe también revisar que el perfil tenga permiso para la app), añadir un método en sequelize-rest-api\src\middlewares\authJWT.js para verificar que el perfil es de Asistente

import jwt from 'jsonwebtoken'

import 'dotenv/config'

import { Usuario } from '../models/Empleado.js'

export const verificarToken= async(req, res, next) => {

    try {

        const token = req.headers["x-access-token"]

        if(!token) return res.status(403).json({message: "No está autorizado para realizar esta operación"})

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

        req.idUsuario=decodificado.id

        const usuario = await Usuario.findByPk(req.idUsuario, {password: 0})

        if(!usuario)

            return res.status(403).json({message: 'No está autorizado para realizar esta operación'})

        next()

    } catch (error) {

        return res.status(403).json({message: 'No está autorizado para realizar esta operación'})

    }

}

export const isAsistente = async (req,res,next) =>{

    const usuario=await Usuario.findByPk(req.idUsuario)

    if(usuario.id\_perfil==121){

        next()

        return

    }

    return res.status(403).json({message: 'No está autorizado para realizar esta operación'})

}

export const isJefatura = async (req,res,next) =>{

}

Modificar \sequelize-rest-api\src\middlewares\index.js

import {verificarToken, isAsistente} from './authJWT.js'

export {verificarToken, isAsistente}

Modificar \sequelize-rest-api\src\routes\partidas.routes.js para que verifique si el usuario tiene el perfil de asistente antes de agregar, modificar o borrar una partida.

import { Router } from "express";

import {getPartidas, addPartida, updatePartida, deletePartida, getPartida, getPartidaByClave} from "../controllers/partidas.controller.js"

import { verificarToken, isAsistente } from "../middlewares/index.js";

const router = Router()

router.get('/',getPartidas)

router.post('/',[verificarToken, isAsistente], addPartida)

router.put('/:id',[verificarToken, isAsistente],updatePartida)

router.delete('/:id',[verificarToken, isAsistente], deletePartida)

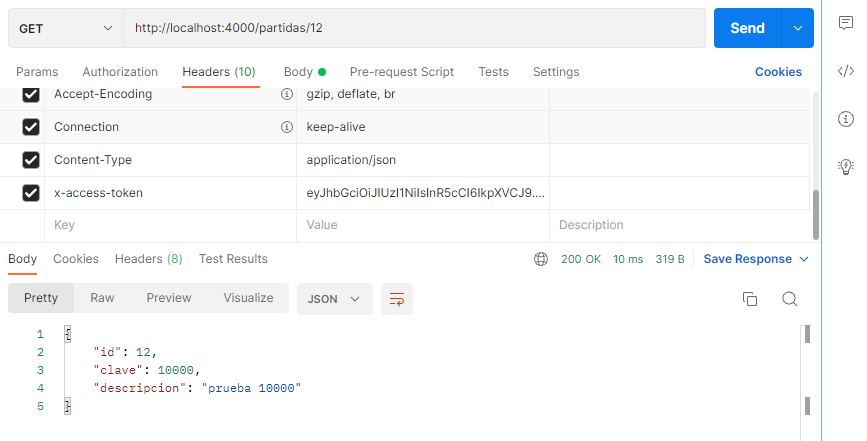
router.get('/:id',getPartida)

router.get('/clave/:clave',getPartidaByClave)

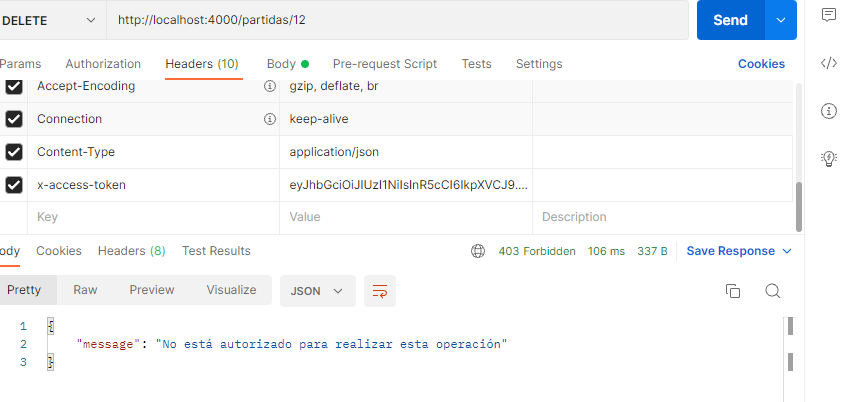
export default router

Probar

Cualquier usuario puede usar los métodos GET



Pero sólo un asistente puede eliminar, agregar o actualizar



Registrar un usuario con perfil 121 para obtener su token y poder repetir la solicitud pero esta vez con el token de un usuario que cuenta con un perfil autorizado a borrar

