1. Instalar express y sequelize

npm i express sequelize pg pg-hstore --S

Consulta para instalación de sequelize en: <https://sequelize.org/docs/v6/getting-started/>

En este caso se instalaron las siguientes versiones

+ pg@8.7.3

+ express@4.18.1

+ pg-hstore@2.3.4

+ [sequelize@6.21.2](mailto:sequelize@6.21.2)

1. Configurar el servidor express en src/app.js
2. Importar app en src/index.js
3. Se agrega “type”:”module, en package.json para usar los import y export sin tener que usar module.export pero los archivos que yo creo se les debe especificar .js para los módulos instalados no es necesario
4. Instalar nodemon (sólo para desarrollo) para no tener que estar reiniciando el servidor con cada cambio manualmente

npm i nodemon -D

1. Añadir un script en package.json para iniciar el servidor

“start”: “nodemon src/index.js”

Ya se puede iniciar el servidor con

npm start

1. Colocar la configuración de sequelize en src/database/database.js poner atención en la **Convención de terminología** indicada en la documentación

**Sequelize** se refiere a la biblioteca misma mientras que **sequelize** se refiere a una instancia de Sequelize, que representa una conexión a una base de datos. Esta es la convención recomendada y se seguirá en toda la documentación.

En la documentación también indica los parámetros que debe recibir.

//Passing parameters separately (other dialects)

const sequelize = new Sequelize('database', 'username', 'password', {

host: 'localhost',

dialect: /\* one of 'mysql' | 'mariadb' | 'postgres' | 'mssql' \*/

});

Se instala dotenv para poner los datos de la base un archivo .env y también se añade .env en .gitignore para que esos datos no sean visibles en caso que se suba a un repositorio git

npm install dotenv --save

1. Importar el objeto sequelize en index y probar que se puede conectar con:

try {

    await sequelize.authenticate();

    console.log('Connection has been established successfully.');

  } catch (error) {

    console.error('Unable to connect to the database:', error);

  }

Que viene en la documentación de sequelize

\*\*\*\*\* This database engine version is not supported, please update your database server. More information <https://github.com/sequelize/sequelize/blob/main/ENGINE.md>

https://github.com/sequelize/sequelize/issues/12186 **se requiere mínimo postgress 9.5**\*\*\*\*\*\*\*\*\*

1. Crear el esquema en la carpeta src/models

import {DataTypes} from 'sequelize'

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

export const Partida = sequelize.define('partida',{

    id: {

        type: DataTypes.INTEGER,

        primaryKey: true,

        autoIncrement: true

    },

    clave: {

        type: DataTypes.INTEGER

    },

    descripcion: {

        type: DataTypes.STRING

    }

})

Para mayor información consultar <https://sequelize.org/docs/v6/core-concepts/model-basics/>

1. El método sync() de sequelize sincroniza tablas, o las crea si no existen, o las borra y las vuelve a crear (<https://sequelize.org/docs/v6/core-concepts/model-basics/#model-synchronization>). Importar Partida en src/index.js

import app from './app.js'

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

import './models/Partida.js'

async function main() {

try {

    await sequelize.sync()

  } catch (error) {

    console.error('Unable to connect to the database:', error);

  }

app.listen(3000)

console.log('Server is listening on port', 3000)

}

main();

**Nota**

Si no se desea que guarde en la tabla los campos de a qué hora se creó y modificó añadir **timestamps**: **false** en el modelo (src\models\Partida.js)

https://sequelize.org/docs/v6/core-concepts/model-basics/#timestamps

En la consola se vería lo siguiente:

[nodemon] starting `node src/index.js`

(node:45908) [SEQUELIZE0006] DeprecationWarning: This database engine version is not supported, please update your database server. More information https://github.com/sequelize/sequelize/blob/main/ENGINE.md

(Use `node --trace-deprecation ...` to show where the warning was created)

Executing (default): SELECT table\_name FROM information\_schema.tables WHERE table\_schema = 'public' AND table\_name = 'partidas'

Executing (default): CREATE TABLE IF NOT EXISTS "partidas" ("id" SERIAL , "clave" INTEGER, "descripcion" VARCHAR(255), "createdAt" TIMESTAMP WITH TIME ZONE NOT NULL, "updatedAt" TIMESTAMP WITH TIME ZONE NOT NULL, PRIMARY KEY ("id"));

Executing (default): SELECT i.relname AS name, ix.indisprimary AS primary, ix.indisunique AS unique, ix.indkey AS indkey, array\_agg(a.attnum) as column\_indexes, array\_agg(a.attname) AS column\_names, pg\_get\_indexdef(ix.indexrelid) AS definition FROM pg\_class t, pg\_class i, pg\_index ix, pg\_attribute a WHERE t.oid = ix.indrelid AND i.oid = ix.indexrelid AND a.attrelid = t.oid AND t.relkind = 'r' and t.relname = 'partidas' GROUP BY i.relname, ix.indexrelid, ix.indisprimary, ix.indisunique, ix.indkey ORDER BY i.relname;

Server is listening on port 3000

1. A continuación se harán las rutas para partidas. Crear el archivo src/routes/partidas.routes.js

import { Router } from "express";

const router = Router()

router.get('/partidas')

router.post('/partidas')

router.put('/partidas/:id')

router.delete('/partidas/:id')

router.get('/partidas/:id')

export default router

1. En app.js importar partidas.routes.js

//configuración del servidor express

import express from 'express'

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

const app = express()

app.use(partidasRoutes)

export default app

1. Ahora se añadirá funcionalidad a las rutas con un controlador; crear el archivo src\controllers\partidas.controller.js

export const getPartidas=(req,res)=>{

    res.send('leyendo partidas')

}

export const addPartida=(req,res)=>{

    res.send('agregando partida')

}

1. En src\routes\partidas.routes.js importar los módulos del controlador y llamarlos en la ruta que les corresponde

import { Router } from "express";

import {getPartidas, addPartida} from "../controllers/partidas.controller.js"

const router = Router()

router.get('/partidas',getPartidas)

router.post('/partidas',addPartida)

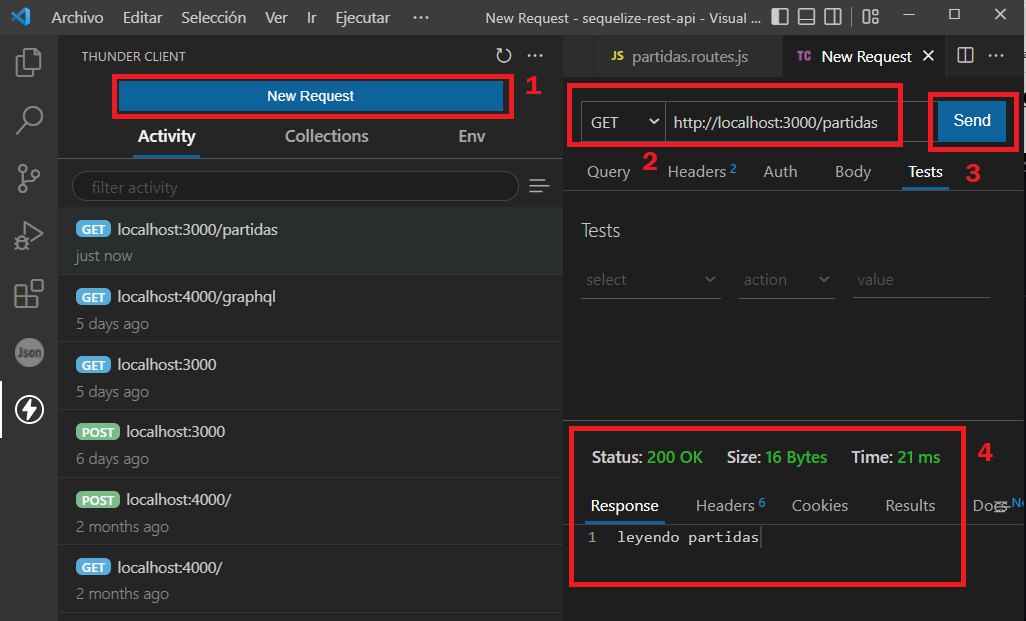
router.put('/partidas/:id')

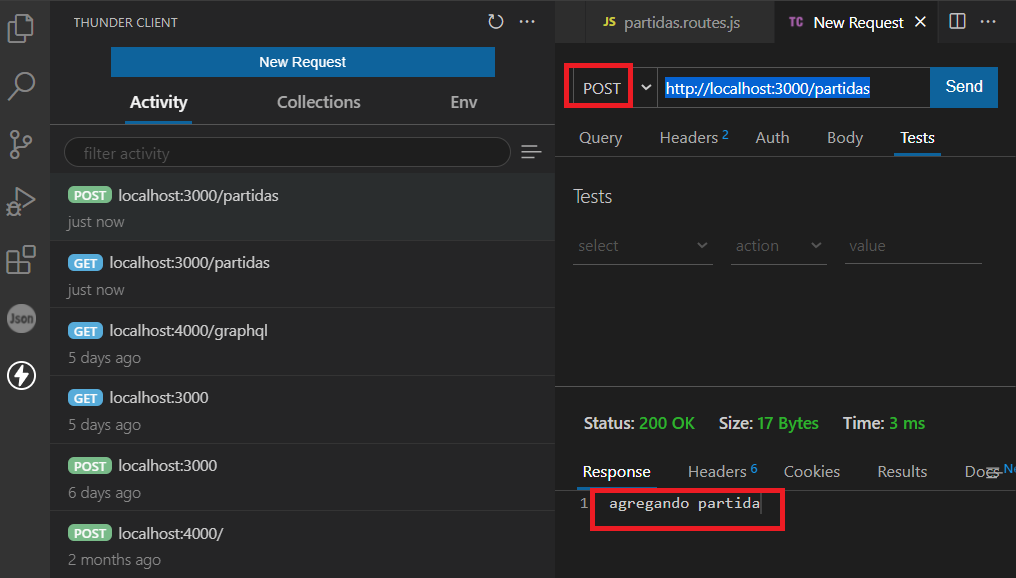
router.delete('/partidas/:id')

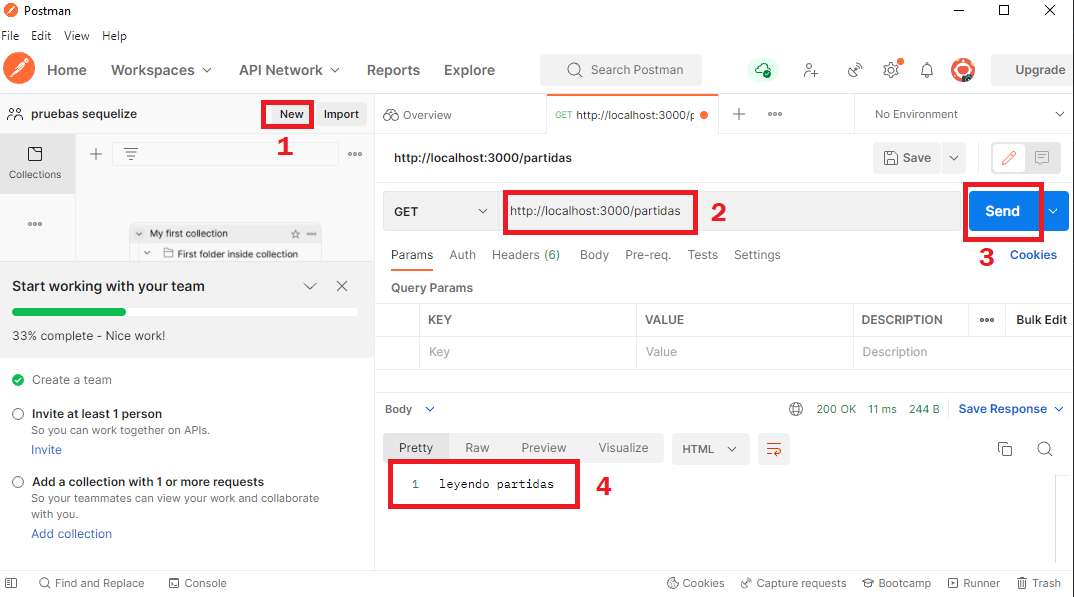
router.get('/partidas/:id')

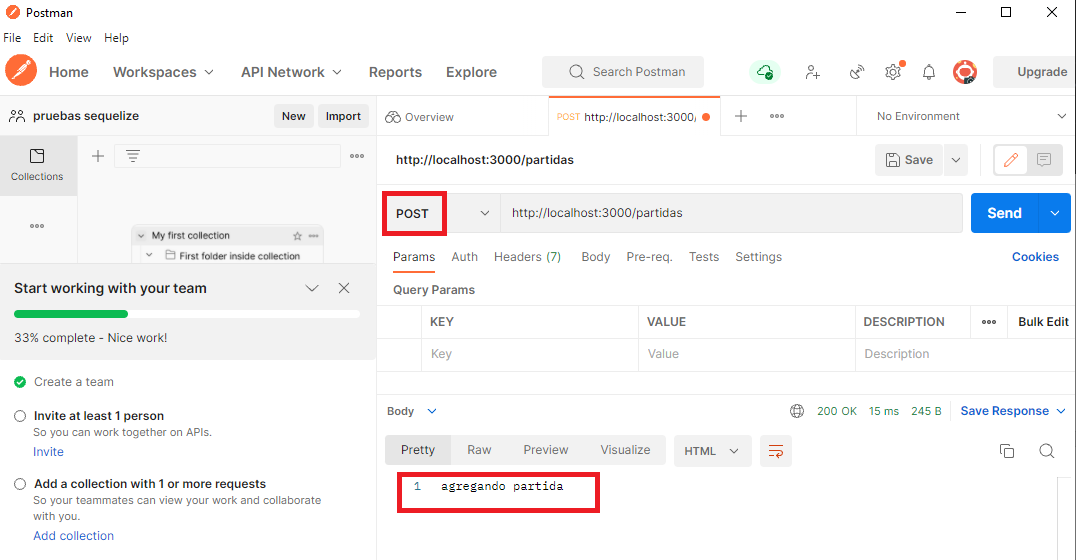
export default router

1. Probar las rutas en Thunder client o Postman









1. Para recibir datos se agregará un middleware cada vez que se envíe un dato en formato json el servidor va a poder interpretarlo y lo va a guardar dentro de un request.body, cada vez que yo llame al request.body voy a poder utilizar los datos que me está enviando. Para más información consultar <http://expressjs.com/en/guide/using-middleware.html>.

src/app.js

//configuración del servidor express

import express from 'express'

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

const app = express()

//middlewares

app.use(express.json())

app.use(partidasRoutes)

export default app

En el controller por el momento sólo imprimir en consola el body y volver a realizar la solicitud, esta vez enviando un objeto json.

export const getPartidas=(req,res)=>{

    res.send('leyendo partidas')

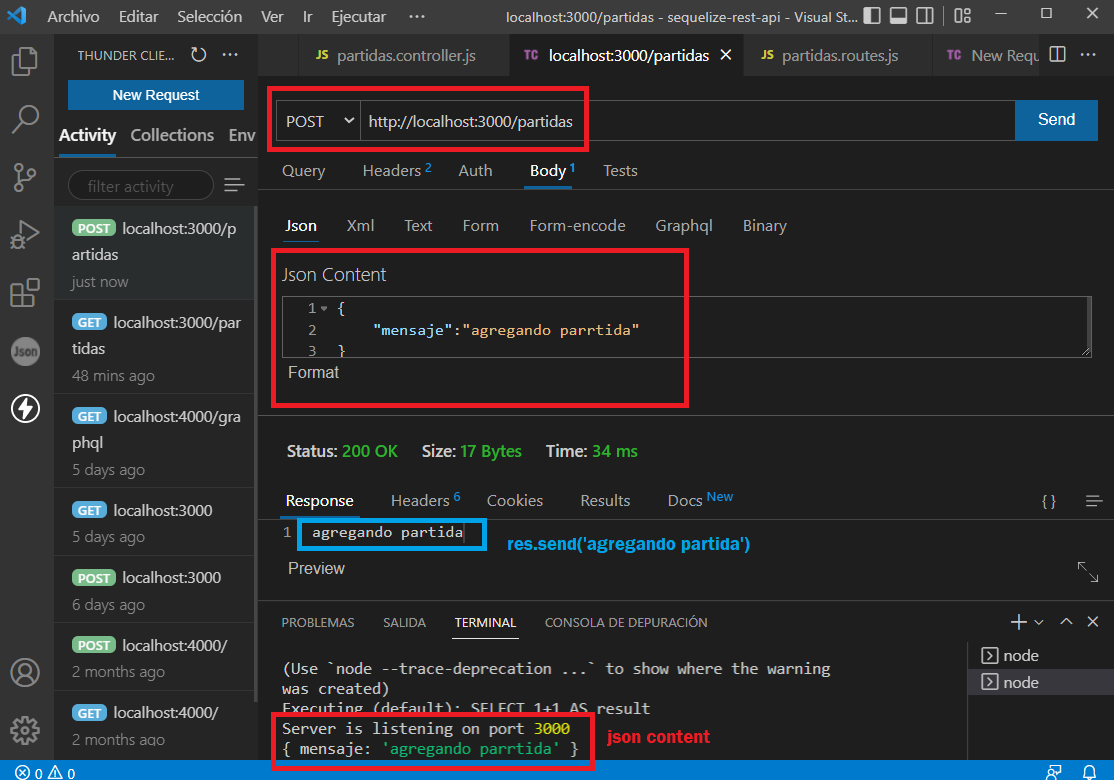
}

export const addPartida=(req,res)=>{

    console.log(req.body)

    res.send('agregando partida')

}



1. Importar el modelo en el controlador y obtener los datos a guardar de request y guardarlos en base de datos (https://sequelize.org/docs/v6/core-concepts/model-querying-basics/)

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

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

    const { clave, descripcion } = req.body

    const newPartida = await Partida.create({

        clave,

        descripcion

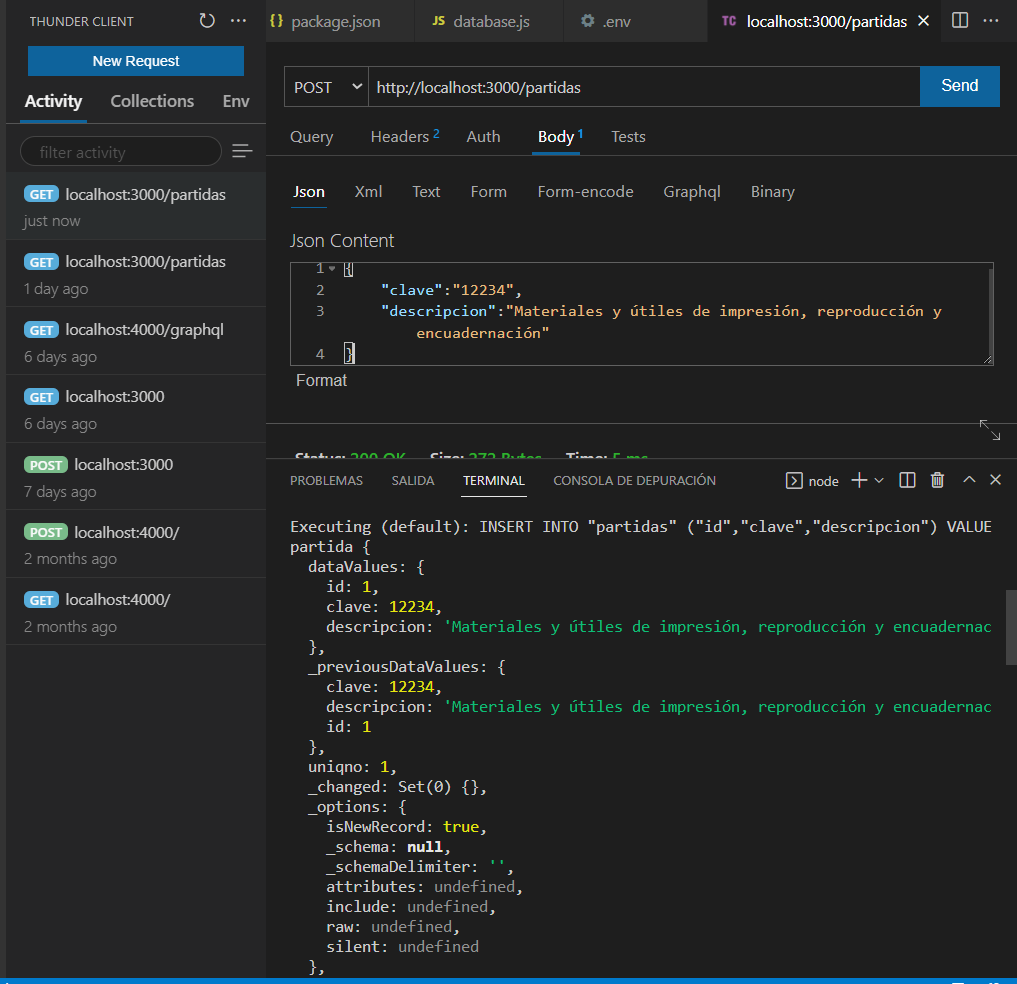
    })

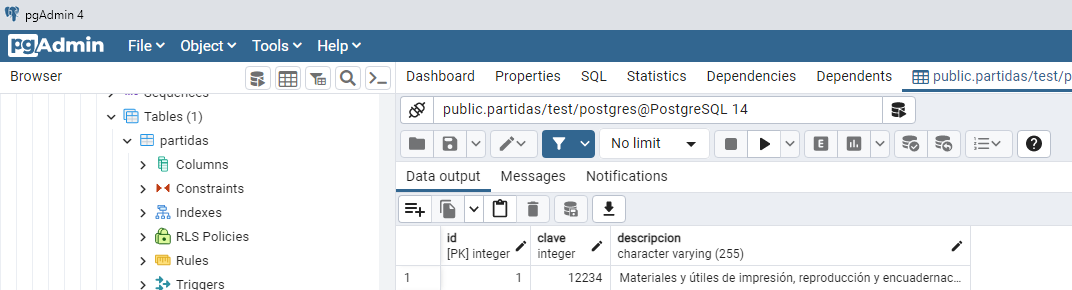
    console.log(newPartida)

    res.send('partida añadida')

}

Ejecutar un request de tipo post para probar que si inserta en base de datos





1. Agregar findAll en el método getPartidas para que me traiga un arreglo de todos los renglones existentes en la base de datos. En lugar de mandarlo a consola se envía al cliente con el método json

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

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

    const partidas= await Partida.findAll()

    res.json(partidas)

}

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

    const { clave, descripcion } = req.body

    const newPartida = await Partida.create({

        clave,

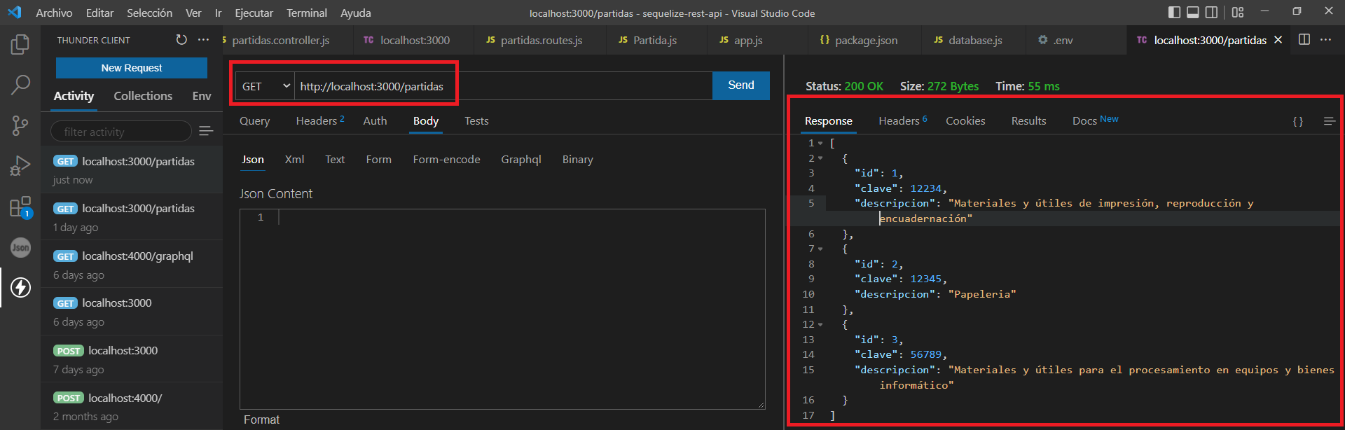
        descripcion

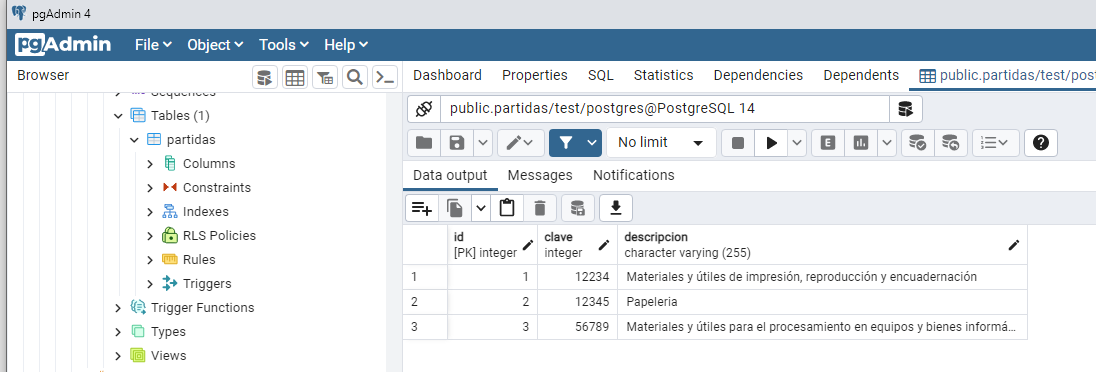
    })

    res.json(newPartida)

}

1. Ejecutar un request de tipo GET para probar el método getPartidas





1. Agregar manejo de errores al controller src\controllers\partidas.controller.js

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

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

    try {

        const partidas = await Partida.findAll()

        res.json(partidas)

} catch (error) {

        return res.status(500).json({ message: error.message })

    }

}

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

    const { clave, descripcion } = req.body

    try {

        const newPartida = await Partida.create({

            clave,

            descripcion

        })

        res.json(newPartida)

    } catch (error) {

        return res.status(500).json({ message: error.message })

    }

}

1. Crear en el controlador src\controllers\partidas.controller.js los métodos updatePartida y deletePartida

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

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

    try {

        const partidas = await Partida.findAll()

        res.json(partidas)

    } catch (error) {

        return res.status(500).json({ message: error.message })

    }

}

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

    const { clave, descripcion } = req.body

    try {

        const newPartida = await Partida.create({

            clave,

            descripcion

        })

        res.json(newPartida)

    } catch (error) {

        return res.status(500).json({ message: error.message })

    }

}

export const updatePartida = async (req, res) => {}

export const deletePartida = async (req, res) => {}

y llamarlos en src\routes\partidas.routes.js

import { Router } from "express";

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

const router = Router()

router.get('/partidas',getPartidas)

router.post('/partidas',addPartida)

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

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

router.get('/partidas/:id')

export default router

en el método deletePartida (src\controllers\partidas.controller.js) borrar el registro que coincida con el id que se obtiene de los parámetros de la petición

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

    try {

        const { id } = req.params;

        await Partida.destroy({

            where: {

                id

            }

        })

        res.sendStatus(204)

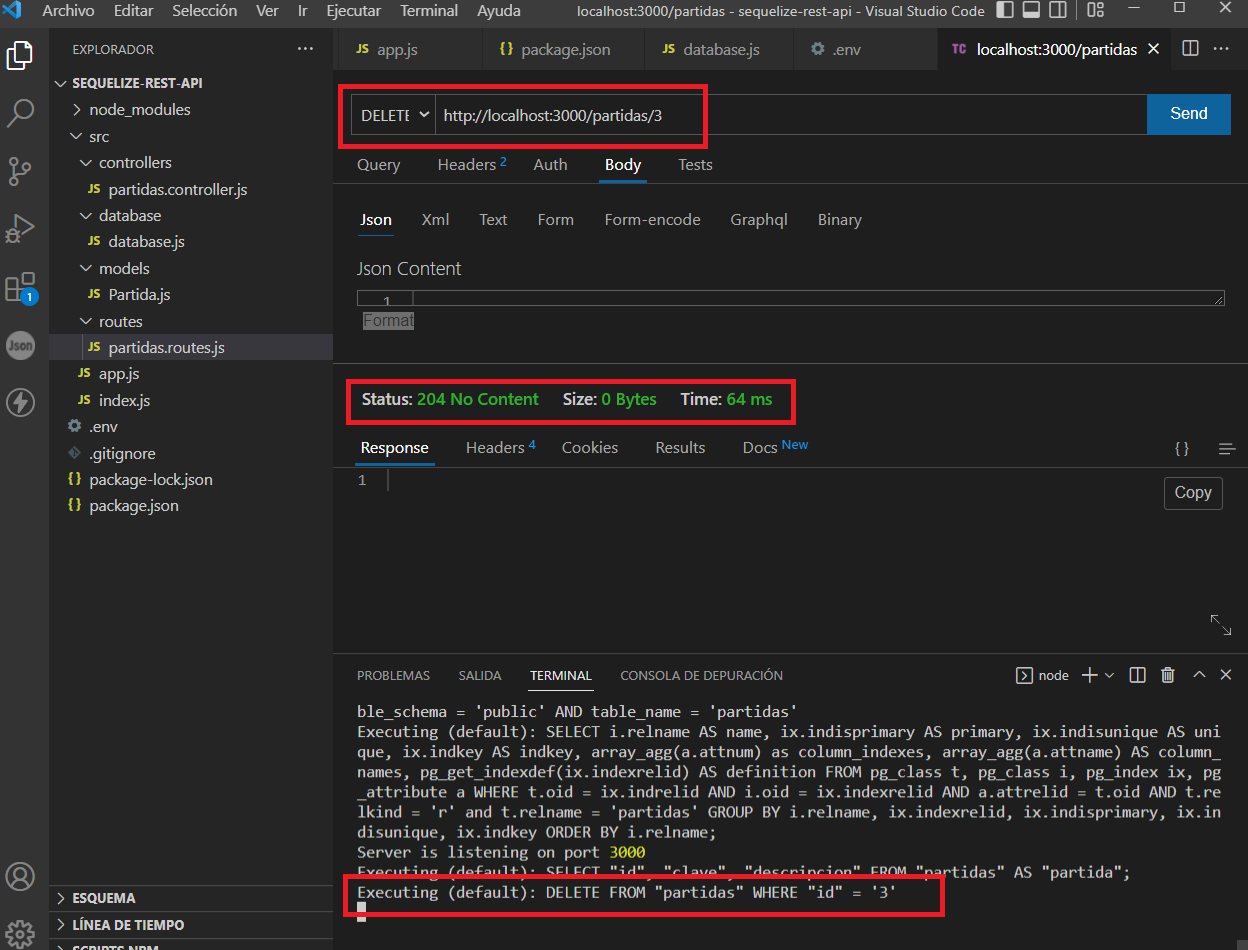
    } catch (error) {

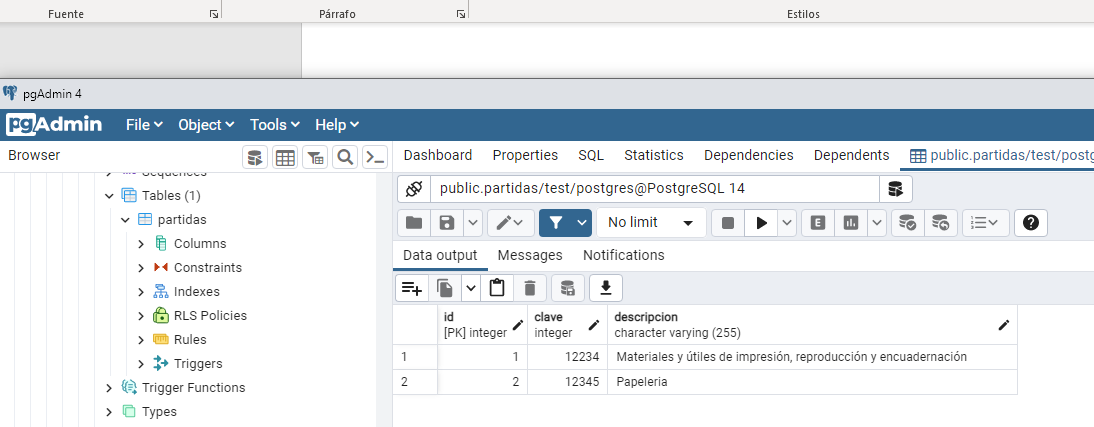
        return res.status(500).json({message: error.message})

    }

}

1. Probar haciendo una petición de tipo delete para algún id existente





1. Modificar el método updatePartida (src\controllers\partidas.controller.js), obteniendo el id de los parámetros de la petición y los datos a modificar del body de la petición

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

    try {

        const { id } = req.params

        const { clave, descripcion } = req.body

        const partida = await Partida.findByPk(id)

        partida.clave = clave

        partida.descripcion = descripcion

        await partida.save()

        res.json(Partida.findByPk(id))

    } catch (error) {

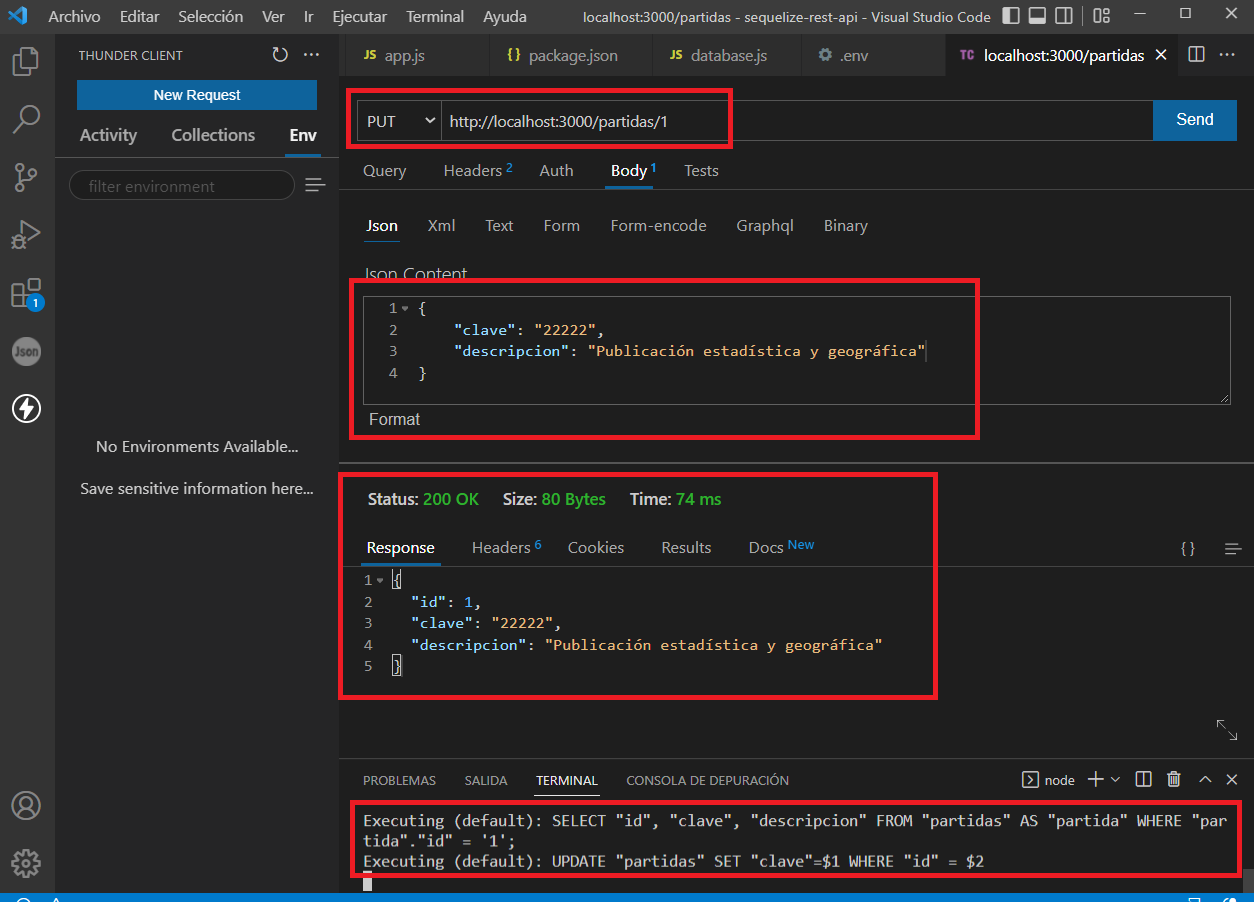
        return res.status(500).json({ message: error.message })

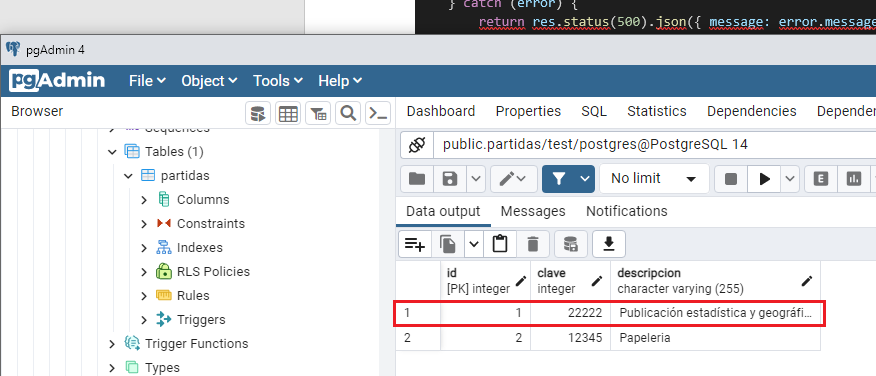
    }

}

https://sequelize.org/docs/v6/core-concepts/model-querying-finders/

Probar una petición put, enviando los datos a modificar en formato json para in id existente





1. Crear el método getPartida en src\controllers\partidas.controller.js y añadirlo a \src\routes\partidas.routes.js

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

    try {

        const { id } = req.params

        const partida = await Partida.findByPk(id)

        if (!partida)

            return res.status(404).json({ message: "La partida solicitada no existe" })

        res.json(partida)

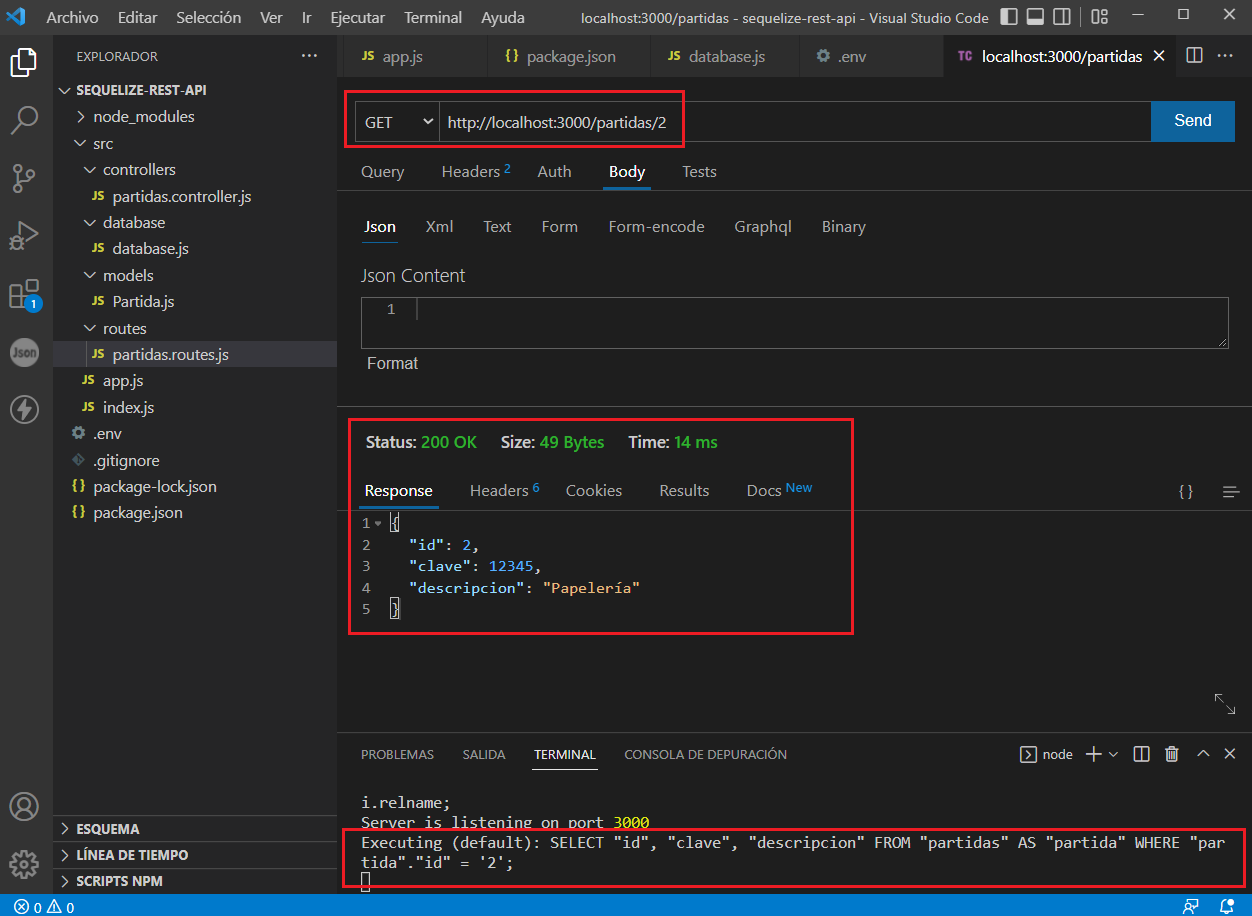
    } catch (error) {

        return res.status(500).json({ message: error.message })

    }

}

Probar la petición get colocando un id



1. Ejemplo con findOne: Se usará el método findOne de sequelize para encontrar una partida por clave

En src\controllers\partidas.controller.js se crea el método getPartidasByClave

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

    try {

        const { clave } = req.params

        const partida = await Partida.findOne({ where: { clave: clave } })

        if (!partida)

            return res.status(404).json({ message: "La partida solicitada no existe" })

        res.json(partida)

    } catch (error) {

        return res.status(500).json({ message: error.message })

    }

}

Y se añade en src\routes\partidas.routes.js

import { Router } from "express";

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

const router = Router()

router.get('/partidas',getPartidas)

router.post('/partidas',addPartida)

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

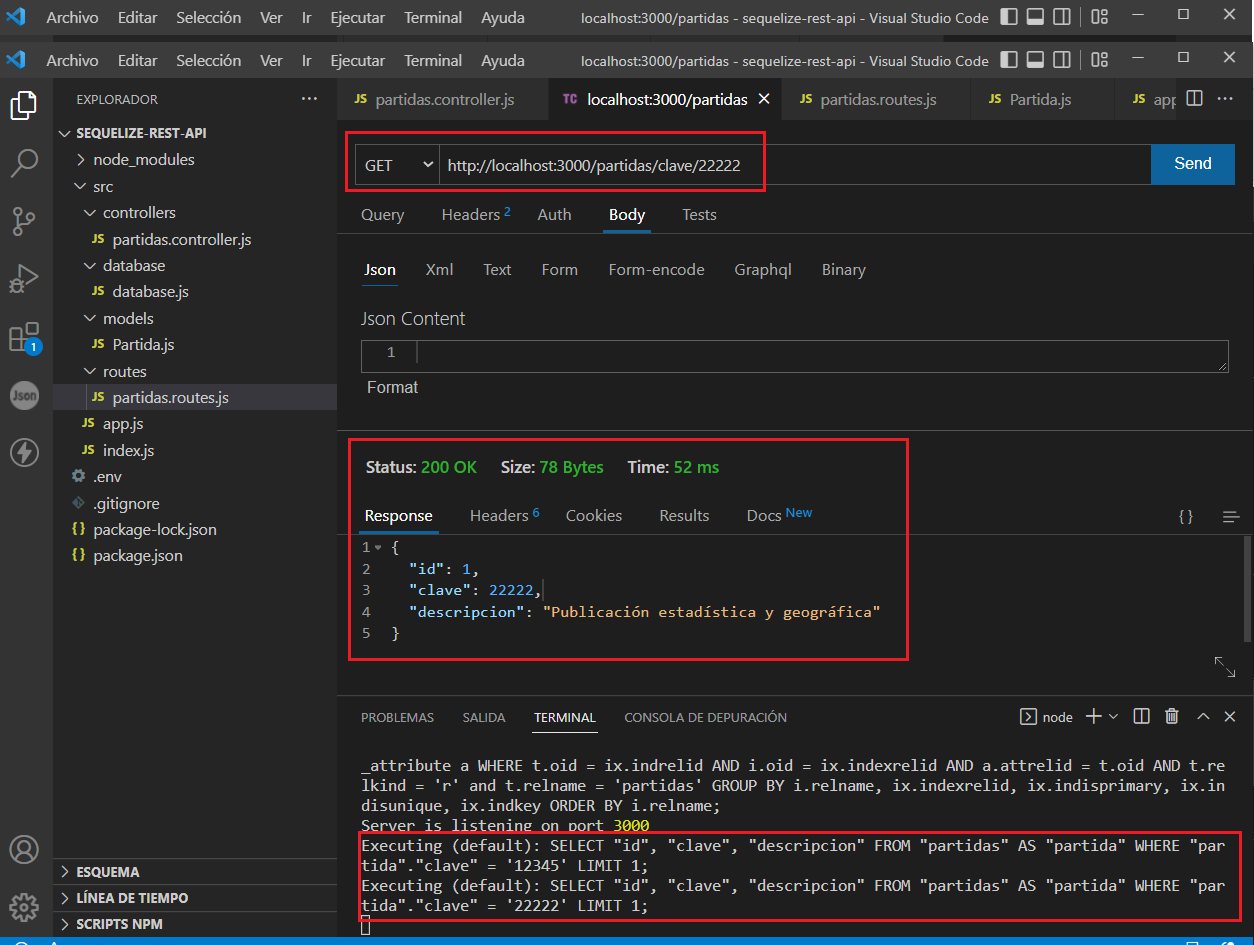
router.delete('/partidas/:id',deletePartida)

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

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

export default router

Probar la petición GET



Minuto 1:18:16 método set se utiliza para actualizar sólo las propiedades que se mandan en el req si no coincide con ninguna no lo toma en cuenta para actualizar

A findOne se le pueden especificar atributos para indicarle sólo lo que me voy a traer de la bd

https://www.youtube.com/watch?v=3xiIOgYdbiE