Contenido

[Conexión a múltiples bases de datos desde NodeJS 2](#_Toc103801884)

[index 2](#_Toc103801885)

[Configuración de las bases de datos con variables de entorno. ENV 3](#_Toc103801886)

[.ENV 4](#_Toc103801887)

[primera conexión de base de datos 4](#_Toc103801888)

[segunda conexión de base de datos 5](#_Toc103801889)

[tercera conexión de base de datos 6](#_Toc103801890)

[app.js 7](#_Toc103801891)

[index.js 7](#_Toc103801892)

[students.routes.js creamos las peticiones 8](#_Toc103801893)

[Students.Controllers.js 8](#_Toc103801894)

[postman 10](#_Toc103801895)

## Conexión a múltiples bases de datos desde NodeJS

Para la conexión multiple de bases de datos mysql se consideraron 3 bases de datos, la estructura de la api es la siguiente:

Texto

Descripción generada automáticamente

index para iniciar el servidor:

*import* app *from* './app';

const chalk = require('chalk');

const main=()=>{

    app.listen(app.get("port"));

    console.log(chalk.hex('#0CFFF6').bold("server on port") + chalk.hex("#7851FF").bold(" => ")  + chalk.hex('#00EBAE').bold(app.get("port")));

*// console.log(`server on port ${app.get("port")}`);*

};

main();

## Configuración de las bases de datos con variables de entorno. ENV

Creamos constantes para cada base de datos que luego exportaremos, para la configuración de cada una de las bases de datos.

*import* { config } *from* "dotenv";

config();

const c = {

    database: {

        host: process.env.HOST, (localhost)

        user: process.env.USER, (brayancapetillo)

        password: process.env.PASSWORD, (250214)

        database: process.env.DATABASE (bookssin)

    },

    port: process.env.PORT (4000)

}

const d = {

    database: {

        host: process.env.HOST,

        user: process.env.USER,

        password: process.env.PASSWORD,

        database: process.env.DATABASE2 (datbase2)

    },

    port: process.env.PORT

}

const e = {

    database: {

        host: process.env.HOST,

        user: process.env.USER,

        password: process.env.PASSWORD,

        database: process.env.DATABASE3 (stronglegend)

    },

    port: process.env.PORT

}

module.exports = {c , d , e } ;

## .ENV

Texto

Descripción generada automáticamente

primera conexión de base de datos:

const chalk = require('chalk');

const mysql = require('mysql');

const { promisify } = require('util');

const {c} = require('./../config')

var data = c.database;

const pool = mysql.createPool(data);

pool.getConnection((err, connection) => {

*if* (err) {

*if* (err.code === 'PROTOCOL\_CONNECTION\_LOST') {

            console.error('DATABASE CONNECTION WAS CLOSED');

        }

*if* (err.code === 'ER\_CON\_COUNT\_ERROR') {

            console.error('DATABASE HAS TO MANY CONNECTIONS');

        }

*if* (err.code === 'ECONNREFUSED') {

            console.error('DATABASE CONNECTION WAS REFUSED');

        }

    }

*if* (connection){

        connection.release();

*// console.log('DB is connected ' + `${data.database}`);*

        console.log(chalk.hex('#655DDB').bold("DB ")+chalk.hex('#FA3055').bold(`${data.database}`)+chalk.hex('#655DDB').bold("     is connected "))

    }*else*{

        console.log(chalk.bgHex('#000').hex("#fff").bold("La base de datos ") + chalk.bgHex("#000").hex("#00EBAE").bold(`${data.database} `)+chalk.bgHex('#000').hex("#fff").bold("    no existe o existe un problema de conexion "))}

*return*;

});

*// Promisify Pool Query*

pool.query = promisify(pool.query);

module.exports = pool;

segunda conexión de base de datos:

const mysql = require('mysql');

const { promisify } = require('util');

const chalk = require('chalk');

*// const { database } = require('./../config');*

const {d} = require('./../config')

var data = d.database;

const pool2 = mysql.createPool(data);

pool2.getConnection((err, connection) => {

*if* (err) {

*if* (err.code === 'PROTOCOL\_CONNECTION\_LOST') {

            console.error('DATABASE CONNECTION WAS CLOSED');

        }

*if* (err.code === 'ER\_CON\_COUNT\_ERROR') {

            console.error('DATABASE HAS TO MANY CONNECTIONS');

        }

*if* (err.code === 'ECONNREFUSED') {

            console.error('DATABASE CONNECTION WAS REFUSED');

        }

    }

*if* (connection){

        connection.release();

        console.log(chalk.hex('#655DDB').bold("DB ")+chalk.hex('#FA3055').bold(`${data.database}`)+chalk.hex('#655DDB').bold("     is connected "))

    }*else*{

        console.log(chalk.bgHex('#000').hex("#fff").bold("La base de datos ") + chalk.bgHex("#000").hex("#00EBAE").bold(`${data.database} `)+chalk.bgHex('#000').hex("#fff").bold("    no existe o existe un problema de conexion "))    }

*return*;

});

*// Promisify Pool Query*

pool2.query = promisify(pool2.query);

module.exports = pool2;

tercera conexión de base de datos:

*// const { database } = require('./../config');*

const {e} = require('./../config')

var data = e.database;

const pool3 = mysql.createPool(data);

pool3.getConnection((err, connection) => {

*if* (err) {

*if* (err.code === 'PROTOCOL\_CONNECTION\_LOST') {

            console.error('DATABASE CONNECTION WAS CLOSED');

        }

*if* (err.code === 'ER\_CON\_COUNT\_ERROR') {

            console.error('DATABASE HAS TO MANY CONNECTIONS');

        }

*if* (err.code === 'ECONNREFUSED') {

            console.error('DATABASE CONNECTION WAS REFUSED');

        }

    }

*if* (connection){

        connection.release();

        console.log(chalk.hex('#655DDB').bold("DB ")+chalk.hex('#FA3055').bold(`${data.database}`)+chalk.hex('#655DDB').bold(" is connected "))

    }*else*{

        console.log(chalk.bgHex('#000').hex("#fff").bold("La base de datos ") + chalk.bgHex("#000").hex("#00EBAE").bold(`${data.database} `)+chalk.bgHex('#000').hex("#fff").bold("no existe o existe un problema de conexion "))

    }

*return*;

});

*// Promisify Pool Query*

pool3.query = promisify(pool3.query);

module.exports = pool3;

app.js

en app.js tenemos lo siguiente:

Hacemos del uso de Morgan para ver las peticiones que se han estado haciendo, como también usamos rutas para las peticiones, donde este nos enviara al index.js pero de las rutas.

*import* express  *from* "express";

*import* morgan *from* "morgan";

*import* cors *from* "cors";

const app = express();

*//routes*

*// import studentsroutes from "./routes/students.routes";*

*//setings*

app.set('port', c.port || 3000);

*//middlewares*

app.use(morgan("dev"));

app.use(cors());

app.use(express.urlencoded({extended: false}));

app.use(express.json());

*//Routes*

*// app.use(studentsroutes)*

app.use(require('./routes'))

*export* *default* app;

## index.js

*import* { Router } *from* 'express';

const router = Router();

const studentsroutes = require ("./students.routes");

*//students*

router.use('/student',studentsroutes);

module.exports = router;

students.routes.js creamos las peticiones:

*import* { Router } *from* "express";

const router = Router();

const {

    getUsers,

    getUsers2,

    getUsers3

} = require('../controllers/students.controllers')

router.get('/base1',getUsers);

router.get('/base2',getUsers2);

router.get('/base3',getUsers3);

module.exports = router;

## Students.Controllers.js

Aquí obtenemos las conexiones de cada base de datos, en este caso la conexión a la base de datos 1 es con pool , base de datos 2 es con pool2 y así sucesivamente.

const pool = require('../database/database');

const pool2 = require('../database/databasetwo');

const pool3 = require('../database/databasethree');

const chalk = require('chalk');

const students = {

    getUsers : async (req,res)=>{

*try*{

        const result = *await* pool.query("select \* from user");

*if*(!result){

*return* res.json({

                success: false,

                message: "users not found!!"

            })

        }

*return* res.json({

            success: true,

            users: result,

            message: "users found!!"

        })

    }*catch*(e){

        console.log(chalk.red("Existe error en la base de datos 1"))

*return* res.json({

            success: false,

            message: "no se pudo hacer tu consulta fallo la base de datos 1"

        })

    }

    },

    getUsers2 : async (req,res)=>{

*try*{

        const result = *await* pool2.query("select \* from user");

*if*(!result){

*return* res.json({

                success: false,

                message: "users not found!!"

            })

        }

*return* res.json({

            success: true,

            users: result,

            message: "users found!!"

        })

    }*catch*(e){

        console.log(chalk.red("Existe error en la base de datos 2"))

*return* res.json({

            success: false,

            message: "no se pudo hacer tu consulta fallo la base de datos 2"

        })

    }

    },

    getUsers3 : async (req,res)=>{

*try*{

        const result = *await* pool3.query("select \* from nat;");

*if*(!result){

*return* res.json({

                success: false,

                message: "users not found!!"

            })

        }

*return* res.json({

            success: true,

            users: result,

            message: "users found!!"

        })

    }*catch*(e){

        console.log(chalk.red("Existe error en la base de datos 3"))

*return* res.json({

            success: false,

            message: "no se pudo hacer tu consulta fallo la base de datos 3"

        })

    }

    }

}

module.exports = students;

postman

con postman podemos ver que las peticiones se ejecutan correctamente.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En caso de que alguna conexión de base de datos falle se notificara, pero esto no detendrá las demás peticiones de las demás bases de datos que si estén funcionando.

Texto

Descripción generada automáticamente

El siguiente ejemplo es para comprobar lo antes mencionado:

En caso de que la base de datos no exista, lo notificara en la consola del api y al crear una petición también lo notificara.

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente