**Dependencias a instalar**

1. **npm init -y** //inicializar el proyecto con node
2. **npm i -D typescript ts-node** //instalar typescript y typescript node
3. **npm I -D nodemon** //reinicia el servidor automáticamente cuando detecta cambios
4. **npm i -D express** //instalar express en el proyecto
5. **npm i -D @types/express** //instalar los tipos para express
6. **npm install --save sequelize** //paso 1 instalar la dependencias

paso 2 instalar el conector correspondiente

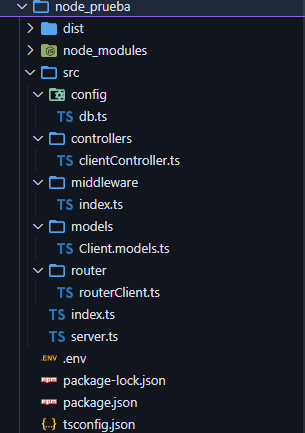
* $ **npm install --save pg pg-hstore** # Postgres
* $ npm install --save mysql2
* $ npm install --save mariadb
* $ npm install --save sqlite3
* $ npm install --save tedious # Microsoft SQL Server
* $ npm install --save oracledb # Oracle Database

1. **npm i dotenv** //instalar dotenv
2. **npm i sequelize-typescript**
3. **npm i express-validator //**instalar la librería para validaciones de express

\*\* npm i -D typescript ts-node nodemon @types/express

\*\* npm i express sequelize dotenv sequelize-typescript express-validator

**El árbol de directorios tiene que quedar así**



\*en este caso la carpeta **node\_prueba** es la raíz del proyecto por cada entidad va un nuevo modelo, por cada modelo va un nuevo controlador, cada ruta debe ser llamada desde server.ts

**Código**

* **node\_prueba/tsconfig.json**

En la raíz del proyecto creo el archivo **tsconfig.json** y agrego lo siguiente:

{

    "compilerOptions": {

        "target": "es6",

        "module": "commonjs",

        "outDir": "./dist",

        "rootDir": "./src",

        "lib": [

            "ESNext"

        ],

        "strict": false,

        "sourceMap": true,

        "esModuleInterop": true,

        "declaration": true,

        "experimentalDecorators": true,

        "emitDecoratorMetadata": true

    },

    "include": [

        "src/\*\*/\*.ts"

    ]

}

* **node\_prueba/package.json**

{

  "name": "node\_prueba",

  "version": "1.0.0",

  "description": "",

  "main": "index.js",

  "scripts": {

"dev": "nodemon --exec ts-node src/index.ts"

  },

  "keywords": [],

  "author": "",

  "license": "ISC",

  "devDependencies": {

    "@types/express": "^4.17.21",

    "nodemon": "^3.1.2",

    "ts-node": "^10.9.2",

    "typescript": "^5.4.5"

  },

  "dependencies": {

    "express": "^4.19.2",

    "dotenv": "^16.4.5",

    "pg": "^8.12.0",

    "pg-hstore": "^2.3.4",

    "sequelize": "^6.37.3",

    "sequelize-typescript": "^2.1.6"

"express-validator": "^7.1.0",

  }

}

* **node\_prueba/.env**

DATABASE\_URL=postgres://root:cnKEM9k8psGBF7IJeNSZ0sQjfYmqijvF@dpg-cpgctiacn0vc73fl5310-a.oregon-postgres.render.com/products\_4hmv?ssl=true

**NOTA**: Para conectar la base de datos al gestor de base de datos el de su preferencia la URL se puede leer asi:

postgres://root:cnKEM9k8psGBF7IJeNSZ0sQjfYmqijvF@dpg-cpgctiacn0vc73fl5310-a.oregon-postgres.render.com/products\_4hmv?ssl=true

* USUARIO
* CONTRASEÑA
* HOST
* NOMBRE DE LA BASE DE DATOS
* **node\_prueba/src/config/db.ts**

import { Sequelize } from 'sequelize-typescript';

import dotenv from 'dotenv';

dotenv.config();

const db = new Sequelize(process.env.DATABASE\_URL, {

    models: [\_\_dirname + '/../models/\*\*/\*.ts']

});

export default db;

* **node\_prueba/src/index.ts**

import server from "./server";

const PORT = 4000;

server.listen(PORT, () => {

    console.log(`El servidor esta corriendo en el puerto ${PORT}`);

});

* **node\_prueba/src/server.ts**

import Express from "express";

import routerCli from "./router/routerClient";

import db from './config/db';

const server = Express();

server.use(Express.json());

server.use(Express.urlencoded({ extended: true }));

server.use('/api/clients', routerCli);

async function connectDB() {

    try {

        await db.authenticate();

        await db.sync()

        console.log('DB is connected')

    } catch (error) {

        console.log(error)

        console.log('DB is not connected')

    }

}

connectDB()

export default server;

Guardo y desde la terminal ejecuto **npx tsc** para compilar y crear la carpeta dist

* **node\_prueba/src/models/Client.models.ts**

import { Table, Column, Model, DataType, Default } from 'sequelize-typescript';

@Table({

    tableName: 'clients'

})

class Client extends Model {

    @Column({

        type: DataType.STRING(50),

        allowNull: false

    })

    name: string;

    @Column({

        type: DataType.STRING(50),

        allowNull: false

    })

    lastname: string;

    @Column({

        type: DataType.STRING(50),

        allowNull: false

    })

    email: string;

    @Column({

        type: DataType.STRING(50),

        allowNull: false

    })

    phone: string;

}

export default Client;

* **node\_prueba/src/controllers/clientController.ts**

import { Request, Response } from "express";

import Client from "../models/Client.models";

//Creo un nuevo cliente

export const createClient = async (req: Request, res: Response) => {

    try {

        const client = await Client.create(req.body);

        res.status(201).json(client);

    } catch (error) {

        console.error('Error al crear un cliente', error);

        res.status(500).json({ error: error.messaje });

    }

};

//Obtengo todos los clientes

export const getClients = async (req: Request, res: Response) => {

    try {

        const client = await Client.findAll();

        res.status(200).json(client);

    } catch (error) {

        res.status(500).json({ error: error.messaje });

    }

};

//Obtengo un cliente por su id

export const getClientById = async (req: Request, res: Response) => {

    try {

        const client = await Client.findByPk(req.params.id);

        if (client) {

            res.status(200).json(client);

        } else {

            res.status(404).json({ error: "No se encontraron clientes" });

        }

    } catch (error) {

        res.status(500).json({ error: error.messaje });

    }

};

//Actualizo un cliente

export const updateClient = async (req: Request, res: Response) => {

    try {

        const client = await Client.findByPk(req.params.id);

        if (client) {

            await client.update(req.body);

            res.status(200).json(client);

        } else {

            res.status(404).json({ error: "No se encontraron clientes para modificar" });

        }

    } catch (error) {

        res.status(500).json({ error: error.messaje });

    }

};

//Elimino un cliente

export const deleteClient = async (req: Request, res: Response) => {

    try {

        const deleted = await Client.destroy({

            where: { id: req.params.id }

        });

        if (deleted) {

            res.status(204).send();

        } else {

            res.status(404).json({ error: "No se encontraron clientes para eliminar" });

        }

    } catch (error) {

        res.status(500).json({ error: error.messaje });

    }

};

* **node\_prueba/src/router/routerClient.ts**

import { Router } from 'express';

import { createClient, getClients, getClientById, updateClient, deleteClient } from '../controllers/clientController';

import { body, param } from 'express-validator';

import { inputErrors } from '../middleware';

const router = Router();

router.post('/',

    //validación de datos usando ORM

    body('name')

        .isString().notEmpty().withMessage('el nombre del cliente no puede estar vacío'),

    body('lastname')

        .isString().notEmpty().withMessage('el apellido del cliente no puede estar vacío'),

    body('email')

        .isString().notEmpty().withMessage('el mail del cliente no puede estar vacío'),

    body('phone')

        .isString().notEmpty().withMessage('el telefono del cliente no puede estar vacío'),

    inputErrors, createClient)

router.get('/', getClients);

router.get('/:id',

    param('id')

        .isNumeric()

        .withMessage('El id ingresado no es válido'),

    inputErrors, getClientById);

router.put('/:id',

    param('id')

        .isNumeric()

        .withMessage('El id ingresado no es válido'),

    body('name')

        .isString().notEmpty().withMessage('el nombre del cliente no puede estar vacío'),

    body('lastname')

        .isString().notEmpty().withMessage('el apellido del cliente no puede estar vacío'),

    body('email')

        .isString().notEmpty().withMessage('el mail del cliente no puede estar vacío'),

    body('phone')

        .isString().notEmpty().withMessage('el telefono del cliente no puede estar vacío'),

    inputErrors, updateClient);

router.delete('/:id',

    param('id')

        .isNumeric()

        .withMessage('El id ingresado no es válido'),

    inputErrors, deleteClient)

export default router;

**Nota**: En el caso que las validaciones fuesen para datos de tipo numéricos o boleanos seria asi ejemplo:

router.put('/:id',

    param('id')

        .isNumeric()

        .withMessage('El id ingresado no es válido'),

    body('name')

        .notEmpty().withMessage('el nombre del producto no puede estar vacío'),

    body('price')

        .notEmpty().withMessage('el nombre del producto no puede estar vacío')

        .isNumeric().withMessage('El valor ingresado debe ser numerico')

        .custom(value => value > 0).withMessage('El valor ingresado no es válido'),

    body('availability')

        .isBoolean()

        .withMessage('El valor ingresado no es válido'),

    inputErrors,

    upDateProd)

* **node\_prueba/src/middelware/index.ts**

código para manejo de errores

import { Request, Response, NextFunction } from 'express'

import { validationResult } from 'express-validator'

export const inputErrors = (req: Request, res: Response, next: NextFunction) => {

    let errors = validationResult(req) // para recuperar los mensajes de error

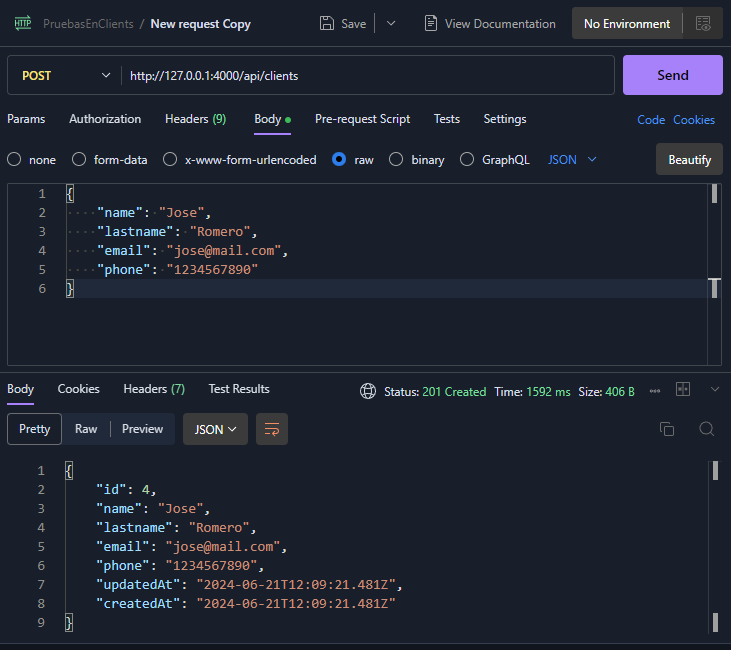
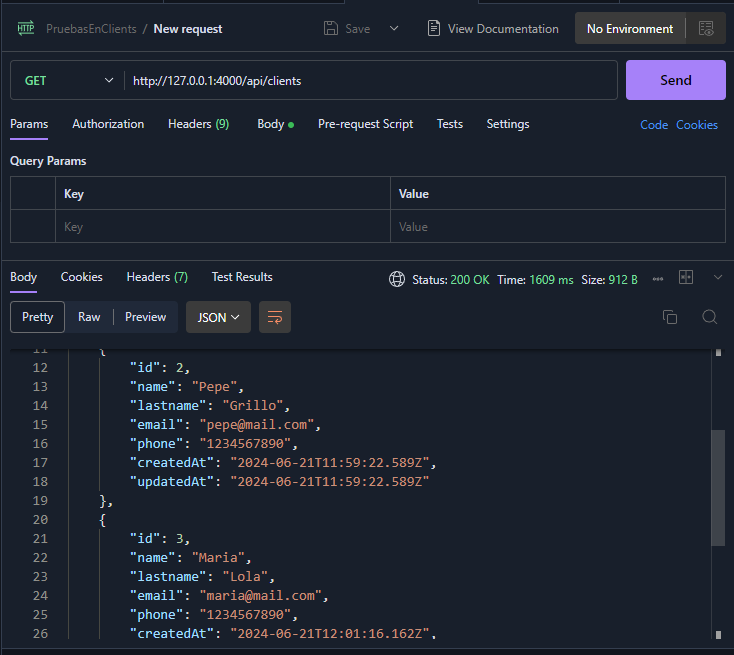
    if (!errors.isEmpty()) {

        return res.status(400).json({ errors: errors.array() })

    }

    next()

}

* **Cargando un cliente mediante request en postman**
* **Traer datos desde la BD mediante request en postman**
* **Conexión a BD local en .env**

DATABASE\_URL=mysql://root:mi\_contraseña@localhost:3306/mi\_base\_de\_datos

* **Modelo relacional en BD**

import { Table, Column, DataType, Model, Default, Unique, ForeignKey, BelongsTo } from 'sequelize-typescript';

import Persona from './Persona.models';

import Categoria from './Categoria.models';

@Table({

    tableName: 'libros'

})

class Libro extends Model {

    @Column({

        type: DataType.INTEGER(),

        primaryKey: true,

        autoIncrement: true,

        allowNull: false

    })

    id\_libro: number;

    @Unique

    @Column({

        type: DataType.STRING(50),

        allowNull: false

    })

    nombre: string;

    @Column({

        type: DataType.STRING(50),

        allowNull: false

    })

    descripcion: string;

    @ForeignKey(() => Persona)

    @Column({

        type: DataType.INTEGER(),

        allowNull: false

    })

    id\_persona: number;

    @BelongsTo(() => Persona)

    persona: Persona;

    @ForeignKey(() => Categoria)

    @Column({

        type: DataType.INTEGER(),

        allowNull: false

    })

    id\_categoria: number;

    @BelongsTo(() => Categoria)

    categoria: Categoria;

}

export default Libro;

**Paso a paso**

1. Crear la carpeta del proyecto e inicializarla **npm init -y**
2. **Instalar todas las dependencias**
3. Crear todo el árbol de directorios y los archivos y completar con el código correspondiente a **.env** y **tsconfig.json** y modificar el srcipt en **package.json ("dev": "nodemon --exec ts-node src/index.ts")**
4. Completar el código en **src/config/db.ts**
5. Completar el código en **server.ts**
6. Completar el código en **index.ts**
7. Ejecutar **npx tsc** para compilar todo y que se cree la carpeta **dist**
8. Completar el código en **src/middeleware/index.ts**
9. Crear el **modelo** solicitado por el enuciado del parcial
10. Crear el **cotroller** por el enuciado del parcial
11. Crear el **router** por el enuciado del parcial
12. En **server.ts** importarel **router** y agregar el **server.use** que corresponda según la ruta
13. En postman la url es:[**http://127.0.0.1:PUERTO/api/loquesea**](http://127.0.0.1:PUERTO/api/loquesea)
14. Para hacer un **POST** poner la url correspondiente, elegir **BODY** opción **RAW** opción **JSON** y el json seria asi:

{

“nombreColumna”: “valor”,

“nombreColumna”: “valor”,

“nombreColumna”: “valor”

…..etc

}