**Nodejs, PostgreSQL & Typescript, REST API CRUD**

Pasos para Crear este Proyecto

M3-L1

1. Crear la carpeta del proyecto
2. Crear un README.md con las userstories en la raíz del proyecto
3. Crear el diagrama E/R, guardarlo como imagen y agregarlo al README.md

M3-L2

1. Crear una carpeta back y una front donde se guardarán todos los archivos del proyecto como dos aplicaciones independientes
2. Crear un archivo .gitignore que guarda carpetas y archivos que no se subirán a github  
    node\_modules/ para ignorar todas las carpetas node\_modules sin importar donde se encuentre  
    \*.env para ignorar todos los archivos que terminen en .env sin importar su nombre  
    dist/ para ignorar todas las carpetas dist sin importar donde se encuentre

BACK

1. Crear una carpeta src
2. Crear el archivo .env para guardar todas las variables de entorno del backend
3. Crear el archivo index.ts que va a contener la conexión a express  
   Por ahora contiene un console.log(“Hola Mundo!!! ”)
4. Ir a la carpeta back en la consola usando cd back
5. Incializar el back con npm init para crear el package.json
6. Instalar TS con npm install typescript -D o npm install --save-dev typescript las dos opciones crean dependencias de desarrollo en el package.json
7. Crear el archivo tsconfig.json con npx tsc --init
8. Configurar tsconfig.json

{

"compilerOptions": {

/\* Visit https://aka.ms/tsconfig to read more about this file \*/

/\* Language and Environment \*/

"target": "ES6" /\* Set the JavaScript language version for emitted JavaScript and include compatible library declarations. \*/,

/\* Modules \*/

"module": "commonjs" /\* Specify what module code is generated. \*/,

"rootDir": "./src" /\* Specify the root folder within your source files. \*/,

/\* Emit \*/

"outDir": "./dist" /\* Specify an output folder for all emitted files. \*/,

"removeComments": true,                           /\* Disable emitting comments. \*/

/\* Type Checking \*/

"strict": true /\* Enable all strict type-checking options. \*/,

},

"include": ["./src/\*\*/\*.ts"],

"exclude": ["node\_modules", ".env"]

}

1. Agregar al package.json  
   "scripts": {

"build": "tsc" para compilar ts a js

},

1. Agregar al package.json otro script

"start": "node ./dist/index.js" para ejecutar el index.js

1. Agregar al package.json otro script  
   "dev": "tsc && node ./dist/index.js" para compilar y ejecutar con un solo comando

M3-L3

1. Instalar dependencias

npm install --save-dev @types/express @types/node nodemon ts-node

npm install express morgan dotenv cors  
Para buscar documentación escribir npm librería en el buscador y nos lleva a la documentación correspondiente

1. Crear archivo nodemon.json

{

"watch": ["src"],

"ext": "ts",

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

}

1. Ajustar package.json  
   "scripts": {

"build": "tsc",

"start": "node ./dist/index.js",

"build:start": "tsc && node ./dist/index.js",

"dev": "nodemon"

},

1. Crear las carpetas necesarias para el backend del proyecto  
   config, controllers, dtos, entities/schemas, interfaces, middlwares, routes, services
2. Agregar las variables de entorno en el archivo .env  
   PORT=3000
3. Crear un archivo de configuración  
   envs.ts que contiene la llamada a dotenv para exportar las variabes de entorno en cualquier lugar donde las necesite  
   import "dotenv/config";

export const PORT = process.env.PORT;

1. Crear archive server.ts  
   import cors from "cors";

import express from "express";

import morgan from "morgan";  
const server = express();

server.use(cors()); // desde que métodos voy a dejar que se ejecuten, desde qué url, qué header, así es para todo tipo de peticiones

server.use(morgan("dev")); // dev = desarrollo

server.use(express.json()); // para que el servidor entienda los json que le llegan por body

server.get("/", (req, res) => {

res.send("Hola Mundo!")

})

export default server;

Instalar por consola  
npm install --save-dev @types/cors @types/Morgan

1. Llamar al servidor desde index.ts  
   import { PORT } from "./config/envs";

import server from "./server";

server.listen(PORT, () => {

console.log(`Server listening on http://localhost:${PORT}`);

});

1. Crear los controladores usersController, appointmentsController
2. Crear los routers para users y appointments y el router principal o indexRouter.ts
3. Actualizar el archivo server.ts

M3-L4

1. Crear las interfaces IUser, IAppointment, ICredential
2. Crear los servicios usersService, appointmentsService, credentialsService
3. Crear los dtos userDto, appointmentDto, credentialDto

M3-L5

TRABAJANDO CON UNA BASE DE DATOS EN POSTGRESQL

1. Abrir SQL Shell de Postgres  
   Server [localhost]: localhost

Database [postgres]: postgres

Port [5432]: 5432

Username [postgres]: postgres

Contraseña para usuario postgres: Eli09Ate89

1. Crear la base de datos (en el nombre de la DB no se pueden usar mayúsculas, camel case, ni guión medio)

CREATE DATABASE veterinaria (en el ejercicio puso project3\_ft53 como nombre)

1. Conectar a la base de datos con \c y ver si hay tablas \dt
2. Instalar typeORM y sus dependencias  
   Página de la documentación: <https://typeorm.io/>

npm install typeorm --save

npm install reflect-metadata --save

1. Hacer el siguiente import en algún lugar global del proyecto (en el index.ts principal del backend):

import "reflect-metadata"

1. Instalar los types de node:

@types/node

1. Controladores de postgres:

npm install pg --save

1. Habilitar en tsconfig.json:

"emitDecoratorMetadata": true,

"experimentalDecorators": true,

"lib": ["ES6"],

1. Si vamos a compartir el proyecto debemos crear una archivo  
   .example.env

Para que guarde un modelo de las variables de entorno sin los datos verdades, para que sean completados por la persona que necesita ejecutar el proyecto

1. Crear archivo data-source.ts con la siguiente Configuración

import { DataSource } from "typeorm";

export const AppDataSource = new DataSource({

type: "postgres",

host: "localhost",

port: 5432,

username: "postgres",

password: "Eli09Ate89",

database: "veterinaria",

synchronize: true,

logging: true,

entities: [],

subscribers: [],

migrations: [],

});

HASTA AQUÍ DONE, SOLO FALTA AGREGAR LA ENTITIES A DATA-SOURCE.TS

1. Hacerla conexión a la base de datos en el archivo index.ts

AppDataSource.initialize()

.then(() => {

console.log(`Database connected on port ${DB\_PORT}`);

server.listen(PORT, () => {

console.log(`Server listening on http://localhost:${PORT}`);

});

})

.catch((error) => console.log(error));

1. Crear las entidades: User, Appointment, Credential usando los decoradores @entity() y @column() así como las relaciones entre las tablas

Al crear las columnas puedo usar   
! es el operador de assertion de no nulo de TypeScript, que permite decirle al compilador que un valor no será null ni undefined en tiempo de ejecución. Debe usarse con cuidado y solo cuando estás seguro de que el valor nunca será nulo o indefinido. En el contexto de TypeORM, es útil para las columnas con NOT NULL. No usarlo sin razón, es una mala práctica, ya que evitas que TypeScript te de las alertas necesarias en caso de errores de este tipo.

Tener en cuenta que:   
- Entre User y Credential hay una relación de 1:1 (en este caso la relación va en la tabla principal User)

  @OneToOne(() => Credential)

  @JoinColumn({ name: "credentialId" })

  credential!: Credential;

- Entre User y Appointment hay una relación de 1:N

Esta relación va en User

  @OneToMany(() => Appointment, (appointment) => appointment.user)

  appointments!: Appointment[];

- Entre Appointment y User hay una relación de N:1

Esta relación va en Appointment

@ManyToOne(() => User, (user) => user.appointments)

  @JoinColumn({ name: "userId" })

  user!: User;

No puede existir un turno sin un usuario que lo pidió appointment tiene que tener el userId

1. Ajustar usersService y appointmentsService para que tomen los datos de la base de datos y no de un array
2. Crear en Thunder Client de forma organizada las peticiones que vamos a ir haciendo
3. Activar y poner en false la siguiente propiedad de tsconfig.json "strictPropertyInitialization": false,
4. Configuraciones de los servicios, aquí es donde voy a establecer la conexión con la base de datos.

export const getUsersService = async () => {

const users = await AppDataSource.getRepository(User).find();

return users;

};

export const createUserService = async (userData: UserDto) => {

const user = await userModel.create(userData); // crea el registro

const result = userModel.save(user); // guarda el registro

return user;

};

export const getUserByIdService = async (id: number) => {

const user = await userModel.findOneBy({ id });

return user;

};

1. Crear todos los archivo necesarios para vehicles
2. Configurar dropschema en true para vaciar las tablas de la base de datos. Es importante al comenzar un proyecto para eliminar datos si necesitamos hacerlo para cambiar los datos
3. Crear la relación entre las tablas users y vehicles agregando a la entity User

@OneToOne(() => Vehicle)

@JoinColumn()

vehicle: Vehicle;

1. Ajustar el Servicio para establecer la relación y hacer la petición de envío a la base de datos y agregar el find relations para que se muestren los datos relacionados a los users
2. Vamos a trabajar relaciones de uno a varios, para hacerlo más real