Crear la base de datos: asequelize

Crear un proyecto

npm init -y

Dependencias a instalar:

npm i express

npm i dotenv

npm i sequelize mysql2

Dependencias a instalar de Desarrollo:

npm i sequelize-cli --D

Crear en la raíz del proyecto los siguientes archivos:

.gitignore ( para ignorar los archivos de Git).

.env (Para declarar las variables de entorno)

.sequelizerc

Abrir el archivo.gitignore y en su interior agregar:

/node\_modules/

Abrir el archivo.env y en su interior agregar:

DB\_USERNAME= root

DB\_PASSWORD=

DB\_HOST= localhost

DB\_DATABASE=asequelize

DB\_PORT=3306

DB\_DIALECT=mysql

Abrir el archivo.app.js y en su interior

const express = require('express');

const app = express();

const path = require('path');

const PORT = process.env.PORT || 3000

app.use(express.static(path.resolve(\_\_dirname, '../public')));

app.use(express.json())

//URL encode - Para que nos pueda llegar la información desde el formulario al req.body

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

app.use('/', (req, res) => res.json({ clave: "con el server" }));

app.listen(PORT, () => {

console.log('Servidor corriendo en el puerto' + PORT)

}

);

Abrir el archivo -sequelizerc y en su interior agregar:

const path = require('path')

module.exports = {

config: path.resolve('./src/database/config', 'config.js'),

'models-path': path.resolve('./src/database/models'),

'seeders-path': path.resolve('./src/database/seeders'),

'migrations-path': path.resolve('./src/database/migrations'),

}

Crear la carpeta en la raiz:

public

src

Dentro de la carpeta src:

crear el archivo app.js

crear las carpetas routes y controller

Ejecutar sequelize-cli init para crear las carpetas que menciona .sequelize

Ingresar a src - config y en el interior del archivo config.js reemplazar todo por

// Para tomar lo parametros del env

require('dotenv').config()

module.exports =

{

"username": process.env.DB\_USERNAME,

"password": process.env.DB\_PASSWORD,

"database": process.env.DB\_DATABASE,

"host": process.env.DB\_HOST,

"port": process.env.DB\_PORT,

"dialect": process.env.DB\_DIALECT,

seederStorage: "sequelize",

seederStorageTableName: "seeds",

migrationStorage: "sequelize",

migrationStorageTableName: "migrations"

}

Crear todos los modelos intervinientes

IMPORTANTE VERIFICAR EL ORDEN DE EJECUCION

sequelize model:generate --name Brand --attributes name:string

sequelize model:generate --name Color --attributes name:string

sequelize model:generate --name User –attributes firstName:string,lastName:string,email:string,password:string

sequelize model:generate --name Category --attributes name:string

sequelize model:generate --name Product --attributes name:string,description:text,price:decimal,stock:integer,image:string,keywords:text,categoryId:integer,brandId:integer,colorId:integer,genderId:integer,imageId:Integer,sizeId:integer

sequelize model:generate --name User --attributes name:string,email:string,password:string, userId:integer,keywords:text,roleId:integer

sequelize model:generate --name CategoryProduct --attributes productId:integer,categoryId:integer

sequelize model:generate --name role --attributes name:string

sequelize model:generate --name user --attributes name:string, keyword:text, roleId:integer

Crear todas las relaciones correpondientes

Modelo PRODUCT

static associate(models) {

// belongsTo

Product.belongsTo(models.Brand);

// belongsTo

Product.belongsTo(models.User);

// belongsToMany

Product.belongsToMany(models.Color, {

as: 'colors',

through: 'colorProduct',

});

// belongsToMany

Product.belongsToMany(models.Category, {

as: 'categories',

through: 'CategoryProduct',

});

}

Modelo BRAND

static associate(models) {

// hasMany

Brand.hasMany(models.Product, {

foreignKey: 'brandId',

as: "products"

})

}

Modelo USER

static associate(models) {

// hasMany

User.hasMany(models.Product, {

foreignKey: 'userId',

as: "products"

})

}

Modelo COLOR

static associate(models) {

Color.belongsToMany(models.Product, {

as: 'products',

through: 'colorProduct',

});

}

Modelo CATEGORY

static associate(models) {

Category.belongsToMany(models.Product, {

as: 'products',

through: 'CategoryProduct',

});

}

AHORA HAY QUE AGREGAR LAS CLAVES FORANEAS A LAS MIGRACIONES

MIGRACION PRODUCT

userId: {

type: Sequelize.INTEGER,

references: {

model: 'users',

key: 'id'

}

},

brandId: {

type: Sequelize.INTEGER,

references: {

model: 'brands',

key: 'id'

}

},

MIGRACION COLORPRODUCT

productId: {

type: Sequelize.INTEGER,

references: {

model: 'products',

key: 'id'

}

},

colorId: {

type: Sequelize.INTEGER,

references: {

model: 'colors',

key: 'id'

}

},

MIGRACION CATEGORYPRODUCT

productId: {

type: Sequelize.INTEGER,

references: {

model: 'products',

key: 'id'

}

},

categoryId: {

type: Sequelize.INTEGER,

references: {

model: 'categories',

key: 'id'

}

},

CREAR LA MIGRACION Y CREACION DE LAS TABLAS

sequelize db:migrate

1-Reducir la compeljidad de la base datos

2- podemos precindir la migracion de sequlize

3-Cargar datos en base de datos por work bech para hacer los select y darle info a la vista

4- al momento de exportar hacerlo por separado>

‘