**SEQULIZE AND POSTGRES**

1. Install node
2. npm i express sequelize pg pg-hstore
3. Create a server and a db in the postgres.
4. Have these credentials stored in .env
5. DATABASE = bookBridge
6. DATABASE\_HOST = localhost
7. DATABASE\_USER = postgres
8. DATABASE\_PASSWORD = skdv@299

4. In init/db.js create an instance of sequlize and initialize it with the creditionals of the database

import dotenv from "dotenv";

import {Sequelize} from "sequelize";

dotenv.config({ path: "./.env" });

const sequelize = new Sequelize(

    process.env.DATABASE,

    process.env.DATABASE\_USER,

    process.env.DATABASE\_PASSWORD,

    {

    host: process.env.DATABASE\_HOST,

    dialect: "postgres",

});

export default sequelize;

5. create a app.js file where we need to authenticate and run the sequlize connection.

import express from "express";

import sequelize from "./init/db.js";

const app = express();

app.use(express.json());

sequelize.authenticate()

  .then(() => console.log("Database connected!"))

  .catch((err) => console.error("DB Error:", err));

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {

  console.log(`Server running on port ${PORT}`);

});

export default app;

6. Now run node app.js to connect the database , if you got the message as db connected then it was successful else check for the connection credientials.

7. To create tables in the db , create a folder named models and create a file for each table and define each table like this

import { DataTypes } from "sequelize";

import sequelize from "../init/db.js";

const User = sequelize.define("User", {

  user\_id: {

    type: DataTypes.INTEGER,

    primaryKey: true,

    allowNull: false

  },

  user\_name: {

    type: DataTypes.STRING(100),

    allowNull: false,

  },

  registration\_number: {

    type: DataTypes.INTEGER,

    unique: true,

    allowNull: false,

  },

  email\_id: {

    type: DataTypes.STRING(100),

    unique: true,

    allowNull: false,

  },

  password: {

    type: DataTypes.STRING(300),

    allowNull: false,

  },

  is\_active: {

    type: DataTypes.BOOLEAN,

    defaultValue: true,

  },

  created\_at: {

    type: DataTypes.DATE,

    defaultValue: DataTypes.NOW,

  },

}, {

  tableName: "users",

  timestamps: false,

});

export default User;

1. Create all the other models and provide all the necessary associations
   1. **One – to – one**

Example: A user has one profile.

User.hasOne(Profile, { foreignKey: "userId" });

Profile.belongsTo(User, { foreignKey: "userId" });

**9.2. One-to-Many**

Example: A user has **many books**.

User.hasMany(Book, { foreignKey: "userId" });

Book.belongsTo(User, { foreignKey: "userId" });

**9.3 Many – to - Many**

Example: A book can belong to many users, and users can own many books.

User.belongsToMany(Book, { through: "UserBooks", foreignKey: "userId" });

Book.belongsToMany(User, { through: "UserBooks", foreignKey: "bookId" });

**9.4 Self Association**

Example: A user can have a referrer (another user).

User.hasMany(User, { as: "Referrals", foreignKey: "referrerId" });

User.belongsTo(User, { as: "Referrer", foreignKey: "referrerId" });

**CRUD OPERATIONS USING SEQULIZE**

**1. Create (Insert)**

**// Single record**

const user = await User.create({

id: 1,

name: "John Doe",

email: "john@example.com",

age: 25

});

// Multiple records

await User.bulkCreate([

{ name: "Alice", email: "alice@example.com", age: 30 },

{ name: "Bob", email: "bob@example.com", age: 28 }

]);

**2. Read (Select)**

// Find all users

const users = await User.findAll();

// Find by primary key

const user = await User.findByPk(1);

// Find one user with condition

const user = await User.findOne({ where: { email: "john@example.com" } });

// Where + Operators

import { Op } from "sequelize";

// Age > 25

const users = await User.findAll({

where: { age: { [Op.gt]: 25 } }

});

// Age between 20 and 30

const users = await User.findAll({

where: { age: { [Op.between]: [20, 30] } }

});

// Select specific attributes (columns)

const users = await User.findAll({

attributes: ["id", "name"]

});

// Ordering & Limit

const users = await User.findAll({

order: [["age", "DESC"]],

limit: 5,

offset: 10

});

3. Update

// Update specific fields

await User.update(

{ age: 26 },

{ where: { id: 1 } }

);

// Increment field

await User.increment("age", { by: 1, where: { id: 1 } });

**4. Delete**

// Delete specific user

await User.destroy({ where: { id: 1 } });

// Delete all users with age < 18

await User.destroy({ where: { age: { [Op.lt]: 18 } } });

: 20 }

});