### Setting Up Sequelize

**Install Sequelize and PostgreSQL Driver**: Run the following command in your project directory:

npm install sequelize pg pg-hstore

**Initialize Sequelize**: You can set up Sequelize in your application. Create a file named sequelize.js or similar:

const { Sequelize } = require('sequelize');

const sequelize = new Sequelize('prod\_db', 'gteam', 'letsg', {

host: 'localhost', // Change to your Docker host if needed

dialect: 'postgres',

});

module.exports = sequelize;

### Step-by-Step Creating a migration

#### 1. Install Dependencies

Make sure you have Sequelize and the PostgreSQL driver installed:

npm install sequelize pg pg-hstore

You should also have sequelize-cli installed:

npm install --save-dev sequelize-cli

#### 2. Initialize Sequelize

Run this command to set up Sequelize in your project:

npx sequelize-cli init

This will create the necessary folders: config, models, migrations, and seeders.

#### 3. Configure Database Connection

Edit config/config.json to set your database connection details:

{

"development": {

"username": "gteam",

"password": "letsg",

"database": "prod\_db",

"host": "localhost",

"dialect": "postgres"

}}

#### 4. Create a Migration

Generate a migration file to create a new table. For example, to create a users table:

npx sequelize-cli migration:generate --name create-users-table

#### 5. Define the Migration

Open the newly created migration file in the migrations folder and define the table structure:

'use strict';

module.exports = {

up: async (queryInterface, Sequelize) => {

await queryInterface.createTable('users', {

id: {

type: Sequelize.INTEGER,

autoIncrement: true,

primaryKey: true,

},

username: {

type: Sequelize.STRING,

allowNull: false,

},

email: {

type: Sequelize.STRING,

allowNull: false,

unique: true,

},

password: {

type: Sequelize.STRING,

allowNull: false,

},

createdAt: {

type: Sequelize.DATE,

allowNull: false,

defaultValue: Sequelize.literal('CURRENT\_TIMESTAMP'),

},

updatedAt: {

type: Sequelize.DATE,

allowNull: false,

defaultValue: Sequelize.literal('CURRENT\_TIMESTAMP'),

},

});

},

down: async (queryInterface, Sequelize) => {

await queryInterface.dropTable('users');

},

};

#### 6. Run the Migration

Now, you can run the migration to create the users table in your database:

npx sequelize-cli db:migrate

#### 7. Create a Model

To interact with the users table, create a model. Create a file named user.js in the models folder:

'use strict';

const { Model, DataTypes } = require('sequelize');

module.exports = (sequelize) => {

class User extends Model {}

User.init({

id: {

type: DataTypes.INTEGER,

autoIncrement: true,

primaryKey: true,

},

username: {

type: DataTypes.STRING,

allowNull: false,

},

email: {

type: DataTypes.STRING,

allowNull: false,

unique: true,

},

password: {

type: DataTypes.STRING,

allowNull: false,

},

}, {

sequelize,

modelName: 'User',

});

return User;

};

#### 8. Using the Model in Express

Now you can use the model in your Express app to create, read, update, and delete users. Here's an example of how to create a new user:

const express = require('express');const sequelize = require('./sequelize'); // Ensure your sequelize instance is exported from sequelize.jsconst User = require('./models/user')(sequelize); // Import your User model

const app = express();

app.use(express.json());

app.post('/users', async (req, res) => {

try {

const user = await User.create(req.body);

res.status(201).json(user);

} catch (error) {

res.status(400).json({ error: error.message });

}

});

// Sync Sequelize and start the server

sequelize.sync().then(() => {

app.listen(3000, () => {

console.log('Server is running on port 3000');

});