Aula 08: Usando Sequelize com Express para Cadastro de Contatos

---

1. Introdução ao Sequelize

O **Sequelize** é um ORM (Object-Relational Mapping) para Node.js que facilita a interação com bancos de dados relacionais, como **SQLite**, **MySQL**, e **PostgreSQL**. Com ele, podemos manipular tabelas e dados usando modelos JavaScript, o que elimina a necessidade de escrever SQL puro.

Nesta aula, vamos aprender a:

- Configurar o Sequelize em uma aplicação Express.
- Definir um modelo para a tabela de contatos.
- Realizar migrações para criar as tabelas.
- Inserir, listar e validar dados de contatos.

---

2. Configuração Inicial

Instalação do Sequelize e Dependências

1. Crie um projeto Node.js e instale as dependências:

```
mkdir cadastro-contatos
cd cadastro-contatos
npm init -y
npm install express sequelize sqlite3
```

1. Instale o Sequelize CLI (Command Line Interface) globalmente para facilitar a criação de migrações:

```
npm install -g sequelize-cli
```

1. Inicialize o Sequelize no projeto:

```
npx sequelize-cli init
```

Este comando cria uma estrutura de pastas no projeto:

- `models/`: Para os modelos do Sequelize.
- `migrations/`: Para as migrações de banco de dados.
- `config/`: Para as configurações de conexão do banco de dados.

---

3. Configurando o Banco de Dados

Configuração para SQLite

Abra o arquivo `config/config.json` e configure o banco de dados SQLite:

```
{
  "development": {
    "dialect": "sqlite",
    "storage": "./database.sqlite"
  },
  "test": {
    "dialect": "sqlite",
    "storage": "./database-test.sqlite"
  },
  "production": {
    "dialect": "sqlite",
    "storage": "./database-production.sqlite"
  }
```

```
}
```

Isso define que, em ambiente de desenvolvimento, será usado o SQLite com o arquivo `database.sqlite` armazenado no diretório raiz do projeto.

Conexão com MySQL

Caso deseje usar MySQL, altere as configurações da seção `development` para:

```
{
  "development": {
    "username": "root",
    "password": "sua-senha",
    "database": "cadastro_contatos",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

Conexão com PostgreSQL

Para PostgreSQL, a configuração seria assim:

```
{
  "development": {
    "username": "postgres",
    "password": "sua-senha",
    "database": "cadastro_contatos",
    "host": "127.0.0.1",
    "dialect": "postgres"
  }
}
```

4. Criando o Modelo de Contato

Crie o modelo de contato para a tabela no banco de dados. Usaremos o Sequelize CLI para gerar o modelo e a migração:

```
npx sequelize-cli model:generate --name Contato --attributes nome:string,telefone:string,email:string
```

Esse comando gera:

- O arquivo do modelo `models/contato.js`.
- O arquivo de migração `migrations/*-create-contato.js`.

Modelo Gerado (Contato)

Vamos editar o modelo `models/contato.js` para adicionar a restrição de unicidade no campo `email`:

```
module.exports = (sequelize, DataTypes) => {
  const Contato = sequelize.define('Contato', {
    nome: {
      type: DataTypes.STRING,
      allowNull: false
    },
    telefone: {
      type: DataTypes.STRING,
      allowNull: false
    },
    email: {
      type: DataTypes.STRING,
      allowNull: false,
      unique: true
    }
  }, {});

  return Contato;
```

```
};
```

Aqui, definimos que:

- `nome` e `telefone` são campos obrigatórios.
- `email` é obrigatório e deve ser único no banco de dados.

---

5. Realizando Migrações para Criar a Tabela

As migrações garantem que a estrutura do banco de dados esteja atualizada. Para criar a tabela de contatos, basta rodar a migração:

```
npx sequelize-cli db:migrate
```

Esse comando lê o arquivo de migração e cria a tabela `Contatos` no banco de dados.

Exemplo de Migração Gerada

O arquivo de migração `migrations/*-create-contato.js` será similar a este:

```
module.exports = {
  up: async (queryInterface, Sequelize) => {
    await queryInterface.createTable('Contatos', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      nome: {
        type: Sequelize.STRING,
        allowNull: false
      },
      telefone: {
        type: Sequelize.STRING,
```

```
        allowNull: false
      },
      email: {
        type: Sequelize.STRING,
        allowNull: false,
        unique: true
      },
      createdAt: {
        allowNull: false,
        type: Sequelize.DATE
      },
      updatedAt: {
        allowNull: false,
        type: Sequelize.DATE
      }
    });
  },
  down: async (queryInterface, Sequelize) => {
    await queryInterface.dropTable('Contatos');
  }
};
```

---

6. Criando o Servidor Express e Rotas

Agora vamos configurar o Express para gerenciar o cadastro de contatos via formulários HTML.

Configuração do Servidor Express

1. Crie o arquivo `app.js` e configure o Express para receber formulários:

```
const express = require('express');
const { Contato } = require('./models');
const app = express();

// Middleware para processar dados do formulário (URL-encod
ed)
```

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

// Página inicial com formulário de cadastro
app.get('/', (req, res) => {
  res.send(`
    <form action="/contatos" method="POST">
      <label for="nome">Nome:</label>
      <input type="text" name="nome" id="nome" required>

      <label for="telefone">Telefone:</label>
      <input type="text" name="telefone" id="telefone" requ
ired>

      <label for="email">Email:</label>
      <input type="email" name="email" id="email" required>

      <button type="submit">Cadastrar</button>
    </form>
  `);
});

// Rota para cadastrar o contato
app.post('/contatos', async (req, res) => {
  const { nome, telefone, email } = req.body;

  try {
    await Contato.create({ nome, telefone, email });
    res.send('Contato cadastrado com sucesso!');
  } catch (error) {
    if (error.name === 'SequelizeUniqueConstraintError') {
      res.send('Erro: O email já está em uso.');
    } else {
      res.send('Erro ao cadastrar o contato.');
    }
  }
});

// Iniciar servidor
```

```
app.listen(3000, () => {
  console.log('Servidor rodando na porta 3000');
});
```

7. Testando o Cadastro

1. Execute a aplicação:

```
node app.js
```

1. Acesse `http://localhost:3000` no navegador e preencha o formulário. O Sequelize criará automaticamente o registro no banco de dados.

8. Exercícios

1. **Exercício 1 (Fácil)**:

   Adicione uma rota `/contatos/lista` que exiba todos os contatos cadastrados em uma tabela HTML. Liste o `nome`, `telefone` e `email`.

2. **Exercício 2 (Médio)**:

   Adicione validação para garantir que o telefone tenha pelo menos 10 dígitos e seja numérico. Implemente a lógica diretamente no modelo do Sequelize.

3. **Exercício 3 (Difícil)**:

   Crie uma rota `/contatos/editar/:id` que permita a edição de um contato já cadastrado. Adicione também uma rota `/contatos/deletar/:id` para excluir um contato.

Conclusão

O **Sequelize** é uma ferramenta poderosa para interagir com bancos de dados em projetos Node.js. Ele simplifica o mapeamento de dados relacionais e oferece recursos como migrações e validação de dados. No cenário de aplicações monolíticas, o uso de formulários com `application/x-www-form-urlencoded` funciona de forma eficiente junto ao Sequelize.