Aula 10 – Sequelize

Prof.: Álvaro Pagliari

UNOESC

Sequelize

- ORM baseado em Promises para Node.js
- Suporta transações, relacionaments, eager e lazy loading, read replication e múltiplos dialetos
- Suporta Postgres, MySQL, MariaDB, SQLite, Sql Server, Oracle, Amazon Redshift e Snowflake's Data Cloud

UNOESC

Example

```
const { Sequelize, Model, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');

class User extends Model {}
User.init({
  username: DataTypes.STRING,
  birthday: DataTypes.DATE
}, { sequelize, modelName: 'user' });

(async () => {
  await sequelize.sync();
  const jane = await User.create({
    username: 'janedoe',
    birthday: new Date(1980, 6, 20)
  });
  console.log(jane.toJSON());
})();
```

UNOESC

Instalação

- `npm install --save sequelize`
- Uma das opções abaixo:
  - `npm install --save pg pg-hstore # Postgres`
  - `npm install --save mysql2`
  - `npm install --save mariadb`
  - `npm install --save sqlite3`
  - `npm install --save tedious # Microsoft SQL Server`
  - `npm install --save oracledb # Oracle Database`

UNOESC

Conexão

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

// Option 1: Passing a connection URI
const sequelize = new Sequelize('sqlite::memory:') // Example for sqlite
const sequelize = new Sequelize('postgres://user:pass@example.com:5432/dbname') // Example for postgres

// Option 2: Passing parameters separately (sqlite)
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: 'path/to/database.sqlite'
});

// Option 3: Passing parameters separately (other dialects)
const sequelize = new Sequelize('database', 'username', 'password', {
  host: 'localhost',
  dialect: /* one of 'mysql' | 'postgres' | 'sqlite' | 'mariadb' | 'mssql' | 'db2' | 'snowflake' | 'oracl
});
```

UNOESC

Models (com define)

```
const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');

const User = sequelize.define('User', {
  // Model attributes are defined here
  firstName: {
    type: DataTypes.STRING,
    allowNull: false
  },
  lastName: {
    type: DataTypes.STRING
    // allowNull defaults to true
  }
}, {
  // Other model options go here
});

// `sequelize.define` also returns the model
console.log(User === sequelize.models.User); // true
```

Models (com extends Model)

```
const { Sequelize, DataTypes, Model } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory:');

class User extends Model {}

User.init({
  // Model attributes are defined here
  firstName: {
    type: DataTypes.STRING,
    allowNull: false
  },
  lastName: {
    type: DataTypes.STRING
    // allowNull defaults to true
  }
}, {
  // Other model options go here
  sequelize, // We need to pass the connection instance
  modelName: 'User' // We need to choose the model name
});

// the defined model is the class itself
console.log(User === sequelize.models.User); // true
```

UNOESC

Migrações

- Para utilizar migrações precisamos instalar o sequelize CLI
  - `npm install --save-dev sequelize-cli`
  - `npx sequelize-cli init`
- Os models devem ser gerados via comando
  - `npx sequelize-cli model:generate --name User --attributes firstName:string,lastName:string,email:string`
- Para realizar a migração, utiliza-se o comando
  - `npx sequelize-cli db:migrate`

UNOESC

CRUD

- Utilize o seguinte métodos para salvar dados

Inserção

```
// Create a new user
const jane = await User.create({ firstName: "Jane", lastName: "Doe" });
console.log("Jane's auto-generated ID:", jane.id);
```

Atualização

```
const jane = await User.create({ name: "Jane" });
jane.favoriteColor = "blue"
await jane.update({ name: "Ada" })
// The database now has "Ada" for name, but still has the default "green" for favorite color
await jane.save()
// Now the database has "Ada" for name and "blue" for favorite color
```

Exclusão

```
const jane = await User.create({ name: "Jane" });
console.log(jane.name); // "Jane"
await jane.destroy();
// Now this entry was removed from the database
```

UNOESC

Seleção

- É possível selecionar os dados via `.findAll()`

```
Post.findAll({
  where: {
    authorId: 12,
    status: 'active'
  }
});
// SELECT * FROM post WHERE authorId = 12 AND status = 'active';
```

- Também pode ser utilizado o método `query()`

```
// Callee is the model definition. This allows you to easily map a query to a predefined model
const projects = await sequelize.query('SELECT * FROM projects', {
  model: Projects,
  mapToModel: true // pass true here if you have any mapped fields
});
// Each element of `projects` is now an instance of Project
```

UNOESC

Validações e Constraints

- Constraints são regras executadas pelo BD e os erros serão repassados do banco de dados ao Sequelize
- Validações são funções executadas pelo Sequelize em Javascript

```
const { Sequelize, Op, Model, DataTypes } = require("sequelize");
const sequelize = new Sequelize("sqlite::memory:");

const User = sequelize.define("user", {
  username: {
    type: DataTypes.TEXT,
    allowNull: false,
    unique: true
  },
  hashedPassword: {
    type: DataTypes.STRING(64),
    validate: {
      is: /^[0-9a-f]{64}$/i
    }
  }
});

(async () => {
  await sequelize.sync({ force: true });
  // Code here
})();
```

Associações

- Sequelize suporta 4 tipos de associações
  - Tem Um
  - Pertence a
  - Tem Vários
  - Pertence à Vários

```
const A = sequelize.define('A', /* ... */);
const B = sequelize.define('B', /* ... */);

A.hasOne(B); // A HasOne B
A.belongsTo(B); // A BelongsTo B
A.hasMany(B); // A HasMany B
A.belongsToMany(B, { through: 'C' }); // A BelongsToMany B through the junction table C
```

Associações

- Relação 1:1
  - Usa-se o hasOne e o belongsTo
- Relação 1:n
  - Usa-se hasMany e belongsTo
- Relação n:n
  - Usa-se duas chamadas belongsToMany

Exemplo

- Criar um novo projeto node.js
- Instalar o sequelize, pg e pg-hstore
  - `npm install --save sequelize pg pg-hstore`
  - `npm install --save-dev sequelize-cli`
- Iniciar o projeto com o sequelize-cli
  - `npx sequelize-cli init`

UNOESC

Exemplo

- Criar o modelo via sequelize-cli
  - `npx sequelize-cli model:generate --name Product --attributes name:string,description:string,price:double`
  - `npx sequelize-cli db:migrate`

Exercícios

- Converta a aplicação dos Produtos que foi desenvolvida na aula anterior de mongodb para Postgresql, reimplementando todas as funções do model

- Migre a aplicação da questão acima para utilizar outro banco de dados que não Posgresql