pulsati

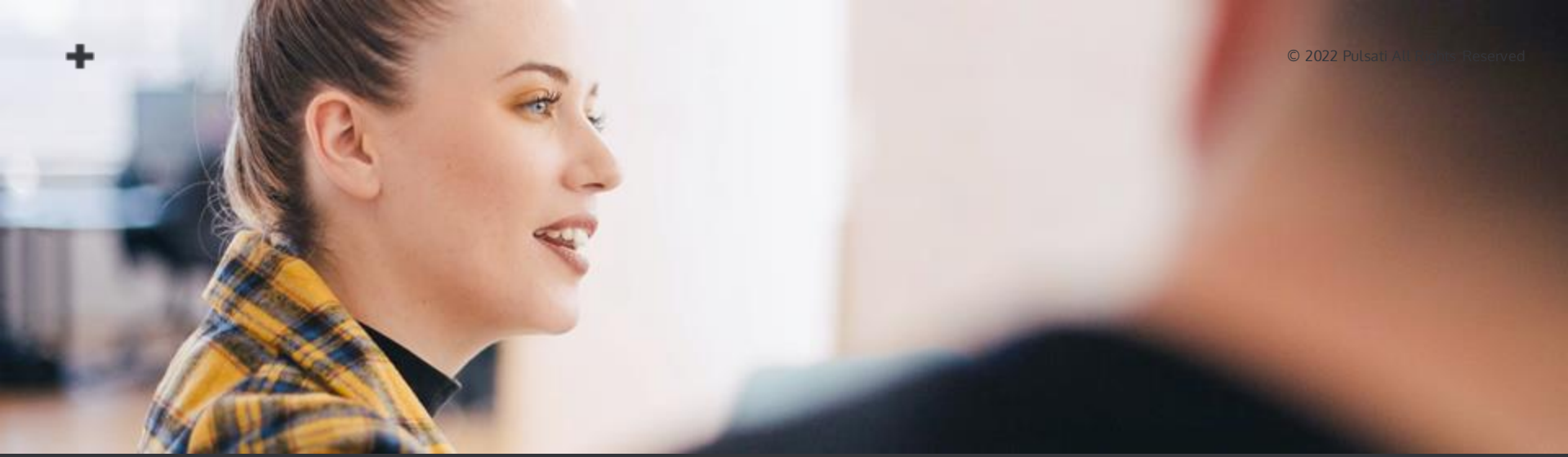
Banco de Dados + Express

Como trabalhar com Banco de Dados e express?

+

require('sequelize')

O Sequelize é uma biblioteca do Node.js que é usada para gerenciar bancos de dados relacionais, permitindo a comunicação do Node.js com uma variedade de bancos de dados, como Oracle, MySQL, PostgreSQL, SQLite e MSSQL, por exemplo usando ORM.



ORM??

É uma técnica que permite mapear os dados entre o modelo de dados do banco de dados relacional e a estrutura de objetos da linguagem de programação.

Como declarar uma classe com ORM

A intenção do ORM é que você possa criar uma classe onde os atributos desta classe possam fazer relação com os campos de uma tabela, por exemplo, digamos que temos uma classe de usuário e como seria essa relação com a tabela de usuários:

```
class Usuario {
    nome;
    sobrenome;
}
```

Com a classe declarada à esquerda, eu poderia esperar que uma biblioteca ORM fizesse uma relação com uma tabela da seguinte forma:

```
table "usuarios" {
    nome: varchar;
    sobrenome: varchar;
}
```

Sendo que, cada biblioteca ORM tem sua forma de trabalhar para definir como exatamente relacionar cada atributo

Como sequelize trabalha o ORM?

+

Sequelize

Para instalar a dependência você precisa executar:

$ npm install sequelize oracledb

Para configurar no banco Oracle:

```
const Sequelize = require('sequelize');
const sequelize = new Sequelize({
    dialect: 'oracle',
    database: 'nome_do_banco',
    username: 'usuario_do_banco',
    password: 'senha_do_banco',
    host: 'endereco_do_banco',
    port: 1521, // porta padrao do oracle
    dialectOptions: { connectTimeout: 30000 }
});
```

Configurando a ORM para tabela "usuarios":

```
const { Model, DataTypes } = require('sequelize');
class Usuario extends Model {}
Usuario.init({
    codigo: {
        type: DataTypes.INTEGER,
        primaryKey: true,
        autoIncrement: true
    },
    nome: {
        type: DataTypes.STRING,
        allowNull: false
    },
    sobrenome: {
        type: DataTypes.STRING
    },
}, {
    sequelize,
    modelName: 'usuarios',
    timestamps: false
});
sequelize.sync();
```

Como funciona o ORM com sequelize?

Configurando a ORM para tabela "usuarios":

```
 1 const{ Model, DataTypes } = require('sequelize');
 2 class Usuario extends Model {}
 3 Usuario.init({
 4     codigo: {
 5         type: DataTypes.INTEGER,
 6         primaryKey: true,
 7         autoIncrement: true },
 8     nome: {
 9          type: DataTypes.STRING,
10          allowNull: false },
11     sobrenome: {
12          type: DataTypes.STRING },
13 }, { sequelize, modelName: 'usuarios', timestamps: false });
14 sequelize.sync();
```

Na linha 1, estamos importando Model e DataTypes

Model vai nos ajudar a fazer a relação entre uma classe que descrevemos para uma tabela de banco de dados

DataTypes nos ajuda a fazer a relação das informações de um campo com o campo da tabela do banco de dados. Por exemplo, type nos diz que tipo de dado aquele campo suporta (número, texto...)

Como funciona o ORM com sequelize?

Configurando a ORM para tabela "usuarios":

```
 1 const{ Model, DataTypes } = require('sequelize');
 2 class Usuario extends Model {}
 3 Usuario.init({
 4     codigo: {
 5         type: DataTypes.INTEGER,
 6         primaryKey: true,
 7         autoIncrement: true },
 8     nome: {
 9          type: DataTypes.STRING,
10          allowNull: false },
11     sobrenome: {
12          type: DataTypes.STRING },
13 }, { sequelize, modelName: 'usuarios', timestamps:  false });
14 sequelize.sync();
```

Na linha 2, estamos defininfo a classe usuário, que iremos usar posteriormente para fazer relação com uma tabela do banco de dados.

Na linha 3, é onde chamamos a função 'init', que irá precisar de informações, indicando qual tabela e campos existem na tabela para fazer a relação da classe com a tabela do banco de dados

Como funciona o ORM com sequelize?

Configurando a ORM para tabela "usuarios":

```
 1 const{ Model, DataTypes } = require('sequelize');
 2 class Usuario extends Model {}
 3 Usuario.init({
 4     codigo: {
 5         type: DataTypes.INTEGER,
 6         primaryKey: true,
 7         autoIncrement: true },
 8     nome: {
 9         type: DataTypes.STRING,
10         allowNull: false },
11     sobrenome: {
12         type: DataTypes.STRING },
13 }, { sequelize, modelName: 'usuarios', timestamps: false });
14 sequelize.sync();
```

Na linha 4, eu defino um campo que existe na tabela, que neste caso é o campo 'codigo'.

Na linha 5, eu informo qual o tipo desse campo, para este campo vai ser inteiro, pois eu informo o DataTypes.INTEGER

Na linha 6, eu indico que ele é uma primary key

Na linha 7, que esse campo vai autoincrementar

Como funciona o ORM com sequelize?

Configurando a ORM para tabela "usuarios":

```
 1 const{ Model, DataTypes } = require('sequelize');
 2 class Usuario extends Model {}
 3 Usuario.init({
 4     codigo: {
 5         type: DataTypes.INTEGER,
 6         primaryKey: true,
 7         autoIncrement: true },
 8     nome: {
 9         type: DataTypes.STRING,
10         allowNull: false },
11     sobrenome: {
12         type: DataTypes.STRING },
13 }, { sequelize, modelName: 'usuarios', timestamps: false });
14 sequelize.sync();
```

Na linha 13, eu informo as configurações que eu quero interagir com o banco de dados, enviando 3 informações:

- Sequelize: eu indico a instância do sequelize, que eu configurei anteriormente, que vai ter os dados para se conectar ao banco
- ModelName: o nome da tabela
- Timestamps: eu não quero que sejam criados campos automáticos para salvar a data de criação e alteração

Como funciona o ORM com sequelize?

Configurando a ORM para tabela "usuarios":

```
 1 const{ Model, DataTypes } = require('sequelize');
 2 class Usuario extends Model {}
 3 Usuario.init({
 4     codigo: {
 5         type: DataTypes.INTEGER,
 6         primaryKey: true,
 7         autoIncrement: true },
 8     nome: {
 9          type: DataTypes.STRING,
10          allowNull: false },
11     sobrenome: {
12           type: DataTypes.STRING },
13 }, { sequelize, modelName: 'usuarios', timestamps:  false });
14 sequelize.sync();
```

Na linha 14 é executada uma sincronização entre o que está no banco de dados e o que foi definido pela aplicação utilizando o sequelize

Prática

Crie um servidor express que:

- Adicione um usuário
- Leia um usuário
- Atualize um usuário
- Delete um usuário

O/
Exercícios
+

Persistir no banco de dados

Usando a aplicação desenvolvida na aula anterior.

Atividade 1: Utilizando o sequelize, definir a entidade Aeroporto e seus campos

Atividade 2: Alterar as rotas POST/PUT/GET/DELETE para trabalharem os dados do banco de dados

Referências

Tutorial de CRUD com Node.js, Sequelize e SQLite
https://www.luiztools.com.br/post/tutorial-de-crud-com-node-js-sequelize-e-sqlite/

SQLite 3 e SQLiteBrowser, como instalá-los no Ubuntu
https://ubunlog.com/pt/sqlite-3-e-sqlitebrowser-como-instal%C3%A1-los-no-ubuntu/

Obrigado!

pulsati

pulsati.com.br