Desenvolver aplicações Backend para WEB

Prof. Renan Ponick

Senac

Lema

“Não crie demasiada expectativa. É frequente que o muito celebrado, antes de realizar-se, pareça depois menor do que a expectativa que criou.”

Baltasar Gracián

Agenda

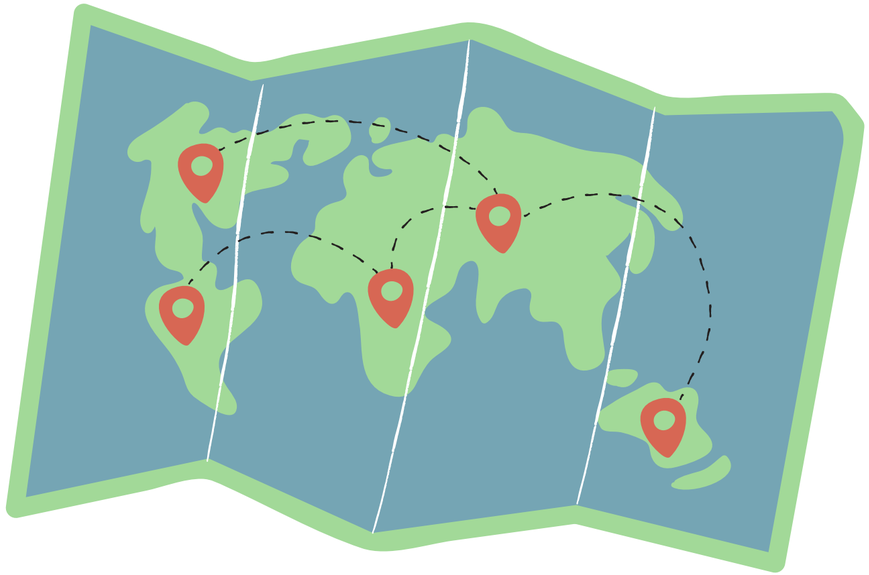
28/09/2023 - ORM - Async - Await
29/09/2023 - Exercícios + Testes
04/10/2023 - Revisão
05/10/2023 - Avaliação
06/10/2023 - Desafios
11/10/2023 - Autenticação
12/10/2023 - FERIADO
13/10/2023 - FERIADO
18/10/2023 - Desafios
19/10/2023 - Avaliação
20/10/2023 - Recuperação/Revisão - **ACABOU**

Conceitos

ORM

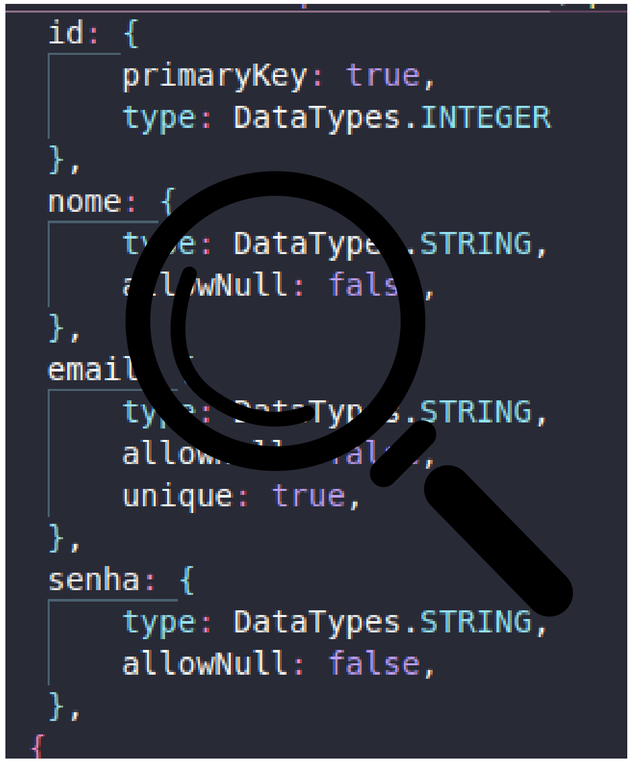
ORM

Object-Relational Mapping
Mapeamento Objeto-Relacional



Conceito

É uma técnica de programação que permite mapear objetos de um sistema de programação orientada a objetos para estruturas de dados em um banco de dados relacional e vice-versa.



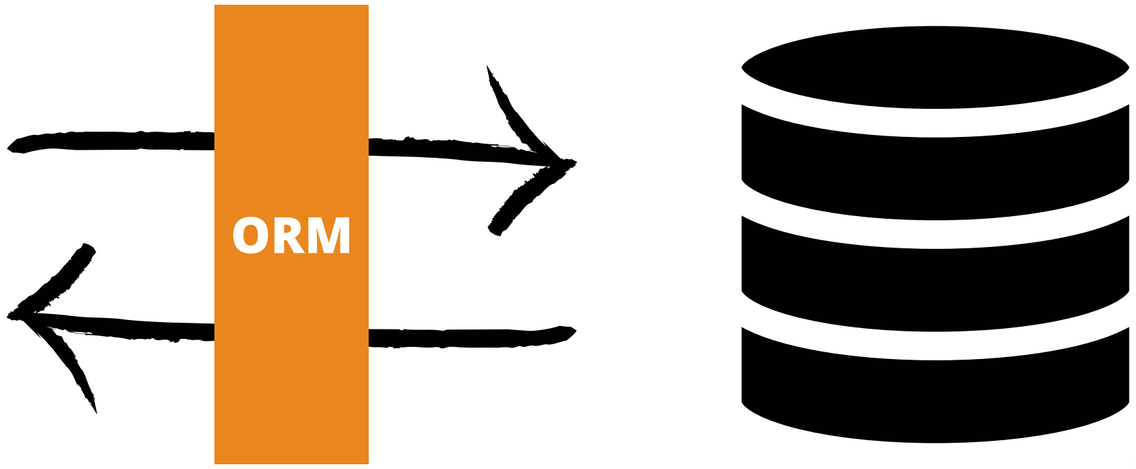
Por que usar?

Ela facilita a interação entre aplicativos escritos em linguagens de programação orientadas a objetos e bancos de dados relacionais.

Como ocorre?

As bibliotecas ou frameworks ORM definem o modo como os dados serão mapeados entre os ambientes, como serão acessados e gravados. Isso diminui o tempo de desenvolvimento, uma vez que não é necessário desenvolver toda essa parte

```
id: {
    primaryKey: true,
    type: DataTypes.INTEGER
},
nome: {
    type: DataTypes.STRING,
    allowNull: false,
},
email: {
    type: DataTypes.STRING,
    allowNull: false,
    unique: true,
},
senha: {
    type: DataTypes.STRING,
    allowNull: false,
},
{
```



Tipos

Exemplos de ORM populares incluem **Sequelize** (para JavaScript/Node.js), Hibernate (para Java), Entity Framework (para C#/.NET) e Django ORM (para Python/Django), entre outros. Cada um desses ORM é projetado para sua linguagem específica, mas todos compartilham o objetivo comum de simplificar a interação entre código orientado a objetos e bancos de dados relacionais.

Async
Await

Async / Await

O async/await simplifica o código assíncrono, tornando-o mais fácil de ler e manter, ao mesmo tempo que mantém a capacidade de controlar o fluxo de execução e lidar com erros de maneira eficaz.

Programação assíncrona

Programação assíncrona

É o ato de executar uma tarefa em "segundo plano", sem nosso controle direto disso

É importante entender que o comportamento do JavaScript é de "executar uma coisa por vez".

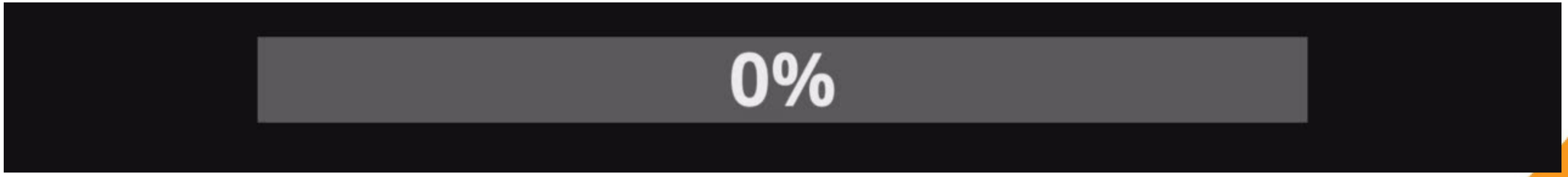
Com isso em mente o assíncrono no JavaScript vai separar seu código em duas partes: coisas que rodam agora, coisas que vão rodar depois de algo acontecer

Programação assíncrona

É para esse tipo de situação que existem as Promises

Programação assíncrona

Quando enviamos uma requisição de dados a uma API, temos uma promessa de que estes dados irão chegar, mas enquanto isso não acontece, o sistema deve continuar rodando



Até aqui tranquilo?

Na pratica

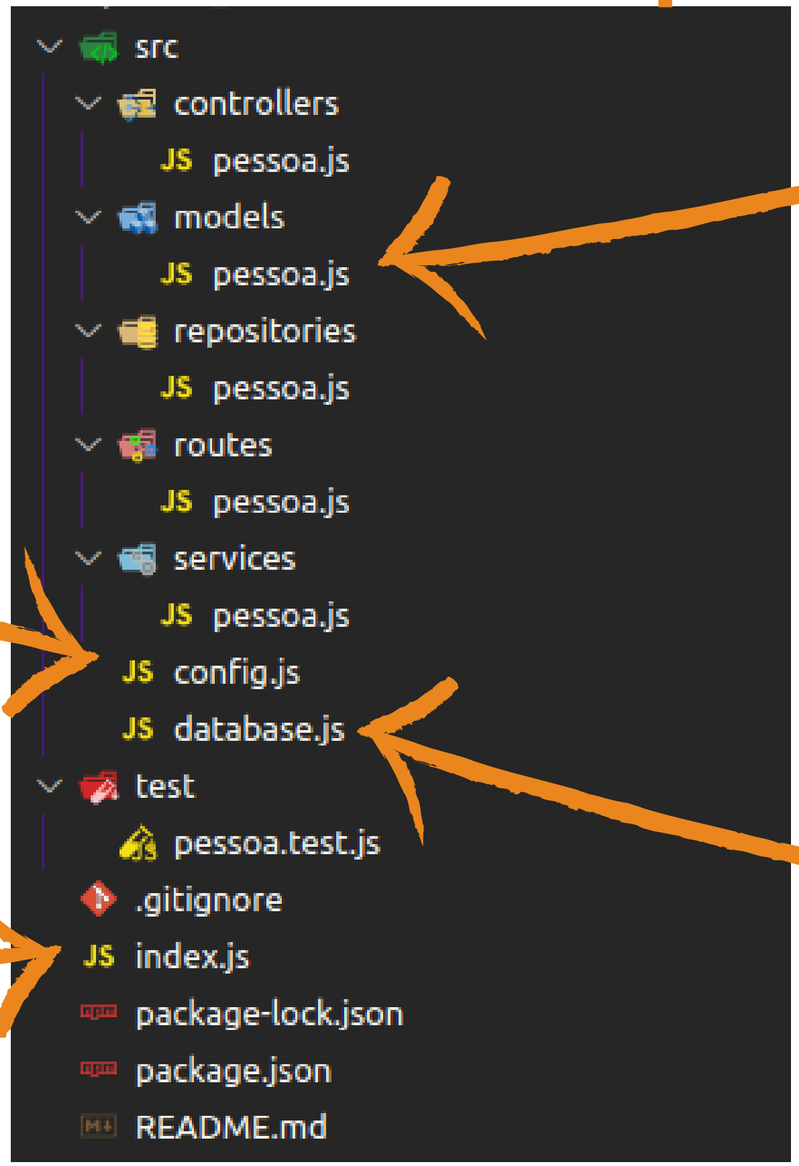
Bora lá

Inicializem o XAMPP, criem um banco novo chamado exemplo, com uma tabela pessoa, e insiram uma pessoa nela.

Help...

```
CREATE DATABASE exemplo;

CREATE TABLE pessoas (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL UNIQUE,
  senha VARCHAR(255)
);

INSERT INTO pessoas (nome, email, senha)
 VALUES ('João da Silva', 'joao@example.com', 'senha123');
```

Estruturando as pastas



Instalando as libs

No terminal, dentro do projeto, execute:

npm install sequelize mysql2

Documentação aqui

config.js

```
module.exports = {
    development: {
      username: 'root',
      password: '123456',
      database: 'exemplo',
      host: 'localhost',
      dialect: 'mysql',
    },
  };
```

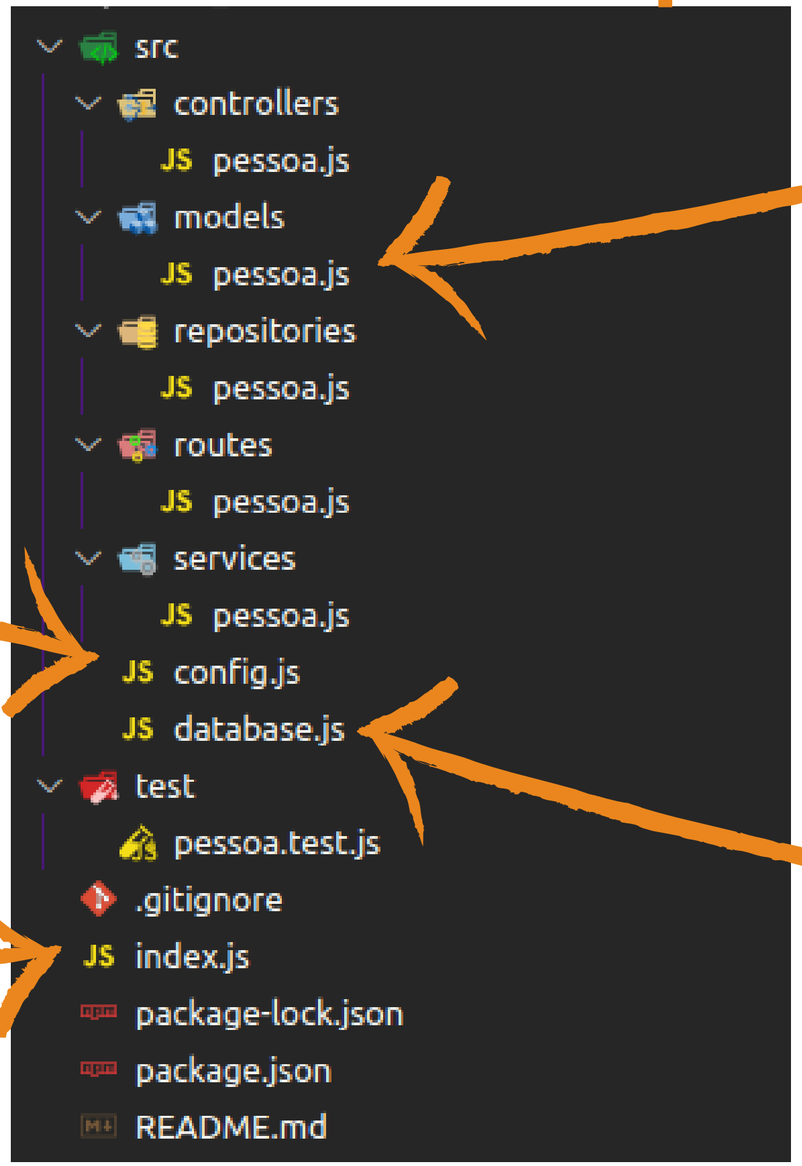
database.js

```
const { Sequelize } = require('sequelize');
const { development } = require('./config');

const sequelize = new Sequelize(development)
sequelize.sync()
  .then(() => {
    console.log('Conexão com o banco de dados estabelecida com sucesso.');
  })
  .catch((error) => {
    console.error('Erro ao conectar ao banco de dados:', error);
  });

module.exports = sequelize
```

O resto é com vocês, vejam a documentação do sequelize e tentem criar a model.

https://sequelize.org/docs/v6/

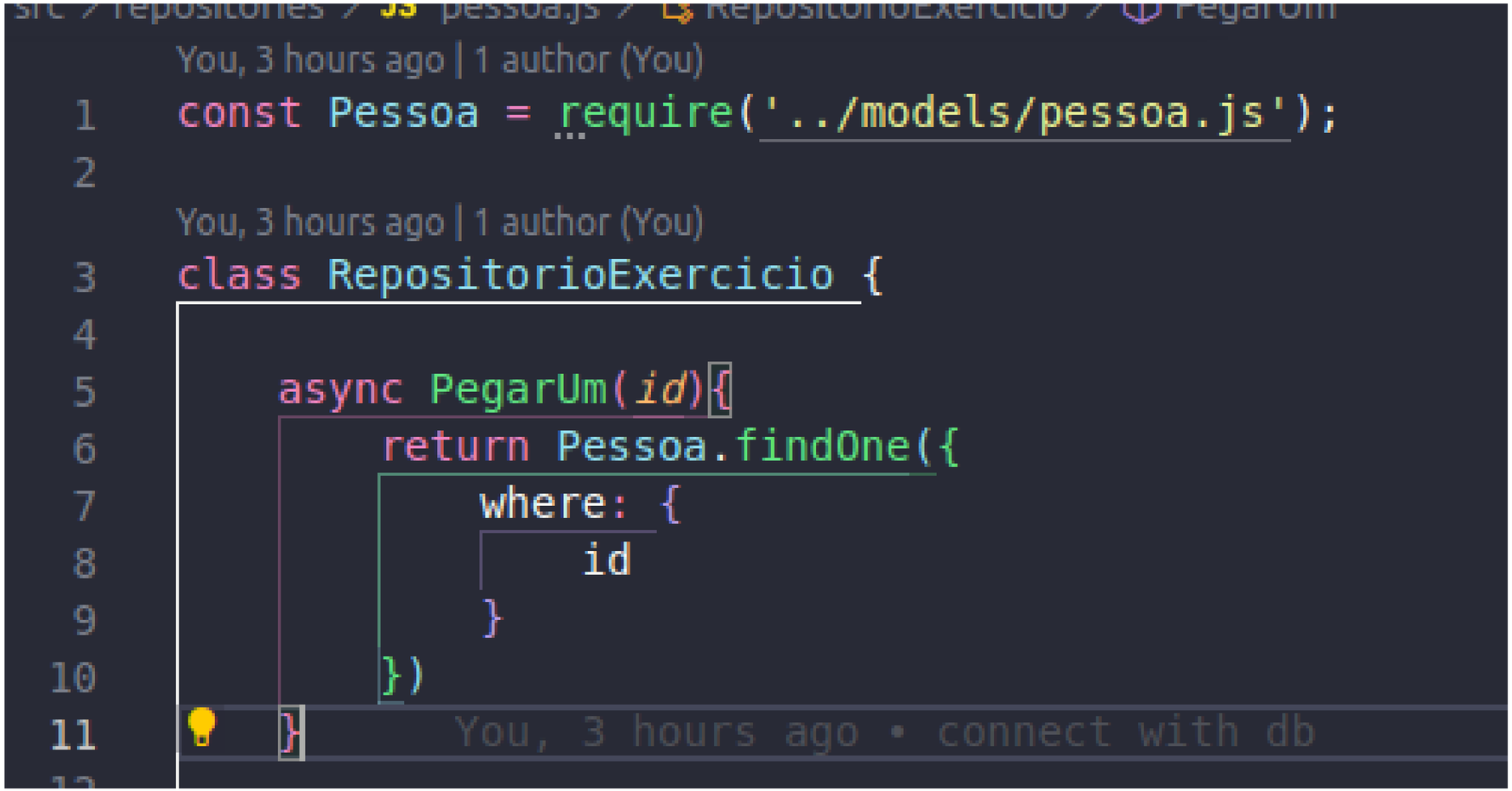
Estruturando as pastas



models/pessoa.js

```
src > models > JS pessoa.js > ...
       You, 3 hours ago | 1 author (You)
 1   const { DataTypes } = require('sequelize');
 2   const sequelize = require('../database.js');
 3
 4   const Pessoa = sequelize.define('pessoas', {
 5       id: {
 6           primaryKey: true,
 7           type: DataTypes.INTEGER
 8       },
 9       nome: {
10           type: DataTypes.STRING,
11           allowNull: false,
12       },
13       email: {
14           type: DataTypes.STRING,
15           allowNull: false,
16           unique: true,
17       },
18       senha: {
19           type: DataTypes.STRING,
20           allowNull: false,
21       },
22   }, {…
25   });
26
27   module.exports = Pessoa;
28
```

repos/pessoa.js



Até aqui tranquilo?

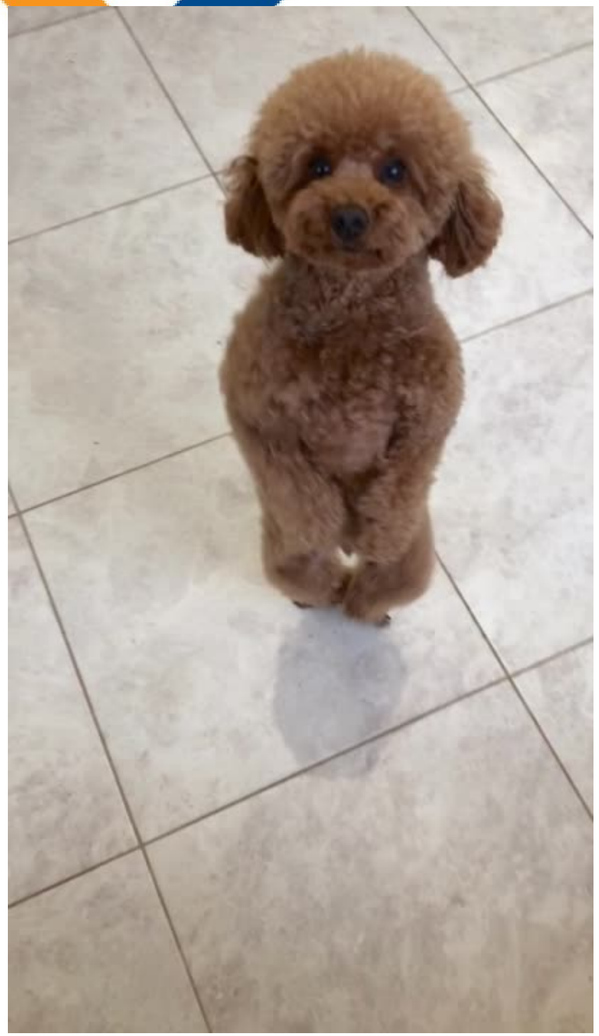
Exercício

Crie um repositório no GitHub de nome APIPetShopNodeJs, clone e Inicialize um projeto NOVO.

Instale as dependencias que vimos até aqui e crie a estrutura de pastas corretamente.

Crie um banco de dados novo chamado PetShop, que contenha uma tabela chamada **Clientes,** armazene nesta tabela, id, nome, telefone (único).

Realize o CRUD deste cliente, assim como fizemos com a Pessoa



Desafio

Crie uma tabela nova dentro do banco PetShop
A classe será o nosso querido - Cachorros
Faça com que a classe tenha relação com a classe Cliente, onde
Um Cachorro pertence a Um Cliente
E
Um Cliente pode ter Vários Cachorros

Feito isso, realize o CRUD do Cachorro.
Faça um metodo para listar todos os cachorros de um Cliente