Documentação do Projeto 1

Eduardo Tirta e Filipe Borba

https://github.com/BigDataPetShop/PetShopAdmin

São Paulo - Agosto de 2018

Sumário

1. Qual a aplicação?

A aplicação pensada para o projeto é um sistema de gestão para Pet Shops, no qual seria possível cadastrar e gerenciar os animais e donos clientes. O diferencial dessa aplicação é que os cadastros estariam disponíveis para todos os pet shops, não necessitando fazer vários cadastros. Os donos poderiam cadastrar seus pets na plataforma, verificando os serviços consumidos pelo(s) pet(s) e seus gastos. Enquanto isso, os administradores do negócio poderiam verificar os recebimentos pendentes, quem está devendo e anotar novos serviços.

2. Funcionalidades do Sistema

Teremos dois tipos de usuários diferentes para essa plataforma: donos de animais e donos de pet shops.

2.1 Donos de Animais

2.1.1 Cadastro

O usuário fará um cadastro simples de suas informações como nome, RG, telefone, email, etc. para entrar na base de dados do sistema. Após isso, ele pode cadastrar seus pets com informações básicas: nome, data de nascimento, raça, etc.

Um dono pode ter vários pets e um pet pode ter vários donos.

2.1.2 Histórico do Pet

O usuário poderá visualizar todos os seus pets num dashboard. Ao clicar em um animal, será possível ver seu histórico de serviços e produtos consumidos, com o preço e o dia de lançamento. O usuário poderá ver o gasto total do Pet.

2.1.3 Controle de Gastos

O usuário poderá visualizar seus gastos com cada pet com alguns gráficos. Ele poderá ver seus gastos pagos, pendentes e totais para cada mês.

2.2 Pet Shop

2.2.1 Controle financeiro

O usuário poderá fazer o controle financeiro com alguns gráficos. Ele poderá ver seus gastos pagos, pendentes e totais para cada mês. Ainda, poderá ver os donos que estão devendo.

2.2.2 Adicionar serviço à pet

Um Pet Shop possui vários serviços disponíveis para um pet. O Pet Shop deve ser capaz de adicionar um novo serviço e alterar o preço dos existentes, além de relacionar serviços à pets.

2.3 Funcionalidade a serem implementadas no Front

Certas funcionalidades que foram criadas na API, não foram adicionadas no produto, porém podemos utilizar o Postman (guia de uso no índice 8 da documentação), para testar as API críadas que não foram implementadas. Algumas dessas funcionalidades são:

- Criação dos dados do:
    - Serviço
    - Petshop
    - Serviço do Petshop
    - Tipo de animal
    - Tipo da raça
- Atualização dos dados dos:
    - Donos
    - Animais

    - Servicos
    - Petshop
- Deleção do:
    - Dono
    - Animal
    - Servico
- Visualização dos:
    - Serviço dos Petshop e seus valores
    - Animais de um determinado dono
    - Donos de um determinado animal

Estas são algumas informações que conseguimos realizar porém não temos no front, mais informações sobre estas requisições estão no índice 9 da documentação.

3. Diagrama Entidade-Relacionamento

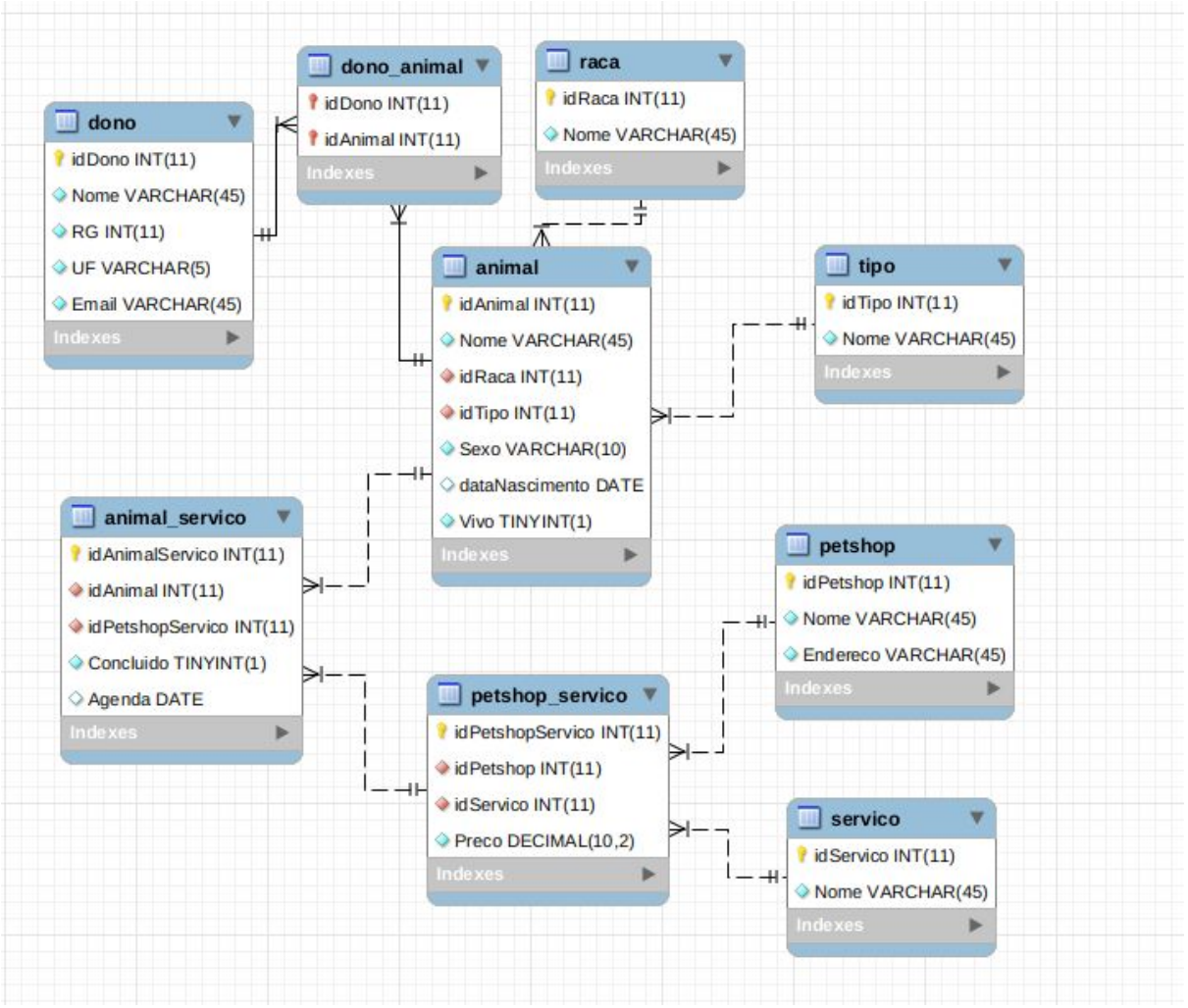


Diagrama Entidade-Relacionamento de nossa aplicação

4. Dicionário de Dados

Animal	
idAnimal	Primary Key - INT(11)
Nome	Nome do animal - VARCHAR(45)
idRaca	Foreign Key, referencia a entidade raça, onde estão o nome das raças - INT(11)

idTipo	Foreign Key, referencia a entidade tipo, onde estão a classificação do animal - INT(11)
Sexo	Sexo do animal - VARCHAR(10)
dataNascimento	Data de quando o animal nasceu - DATE
Vivo	Animal vivo ou não - TINYINT(1)

Tipo	
idTipo	Primary Key - INT(11)
Nome	Classificação do Animal (ex: Gato, Cachorro) - VARCHAR(45)

Raça	
idRaca	Primary Key - INT(11)
Nome	Raça dos animais - VARCHAR(45)

Dono	
idDono	Primary Key - INT(11)
Nome	Nome do dono - VARCHAR(45)
RG	Documento do dono - INT(11)
UF	Estado de emissão do documento do dono - VARCHAR(5)
Email	Email do dono - VARCHAR(45)

PetShop	
idPetshop	Primary Key - INT(11)
Nome	Nome do Estabelecimento - VARCHAR(45)

Servico	
idServico	Primary Key - INT(11)
Nome	Nome do Serviço - VARCHAR(45)

Dono_Animal (Relação dono-animal)	
idDono	Primary Key, Foreign Key - referência à entidade Dono - INT(11)
idAnimal	Primary Key, Foreign Key - referência à entidade Animal - INT(11)

Animal_Servico (Relação animal-servico)	
idAnimalServico	Primary Key - INT(11)
idAnimal	Primary Key, Foreign Key - referência à entidade Animal - INT(11)
idServico	Primary Key, Foreign Key - referência à entidade Servico - INT(11)
Concluido	Indica se o serviço foi terminado ou está pendente - TINYINT(1)
Agenda	Dia em que o serviço vai ser ou foi realizado - DATE

Petshop_Servico (Relação animal-serviço)	
idPetShopServico	Primary Key - INT(11)
idPetshop	Primary Key, Foreign Key - referência à entidade Petshop - INT(11)
idServico	Primary Key, Foreign Key - referência à entidade Servico - INT(11)
Preco	Preço do serviço em cada Petshop - DECIMAL(10,2)

5. Diagrama de Arquitetura do Sistema

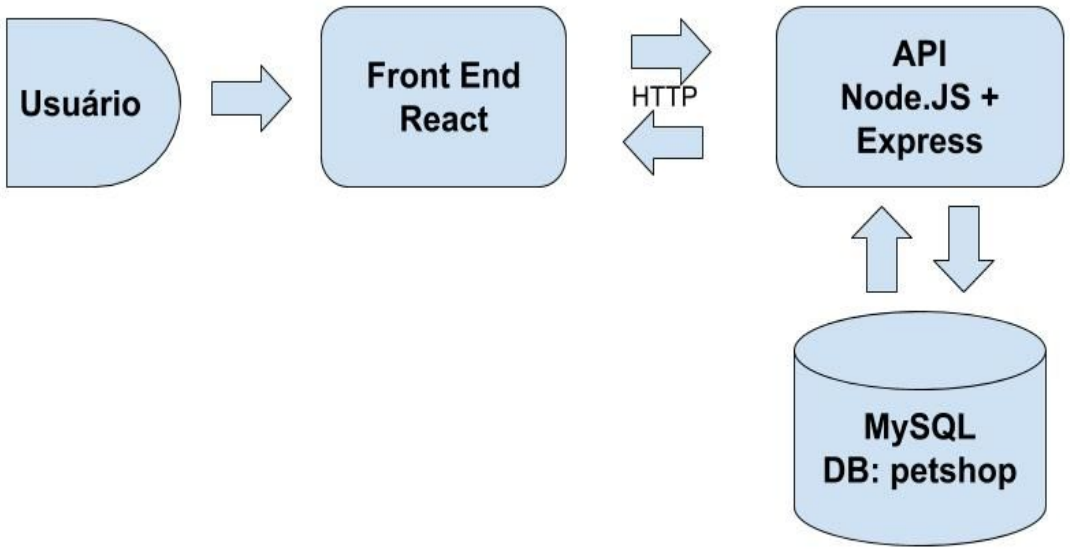


Figura - Infraestrutura da solução, indicando hardware e software

Para entender como a aplicação se comunica com a API, precisamos entender como o React funciona. React é uma biblioteca de javascript para construir UIs. Ele permite que você componha sua view com pedaços de código pequenos e isolados chamados de “componentes”. Vejamos o exemplo a seguir:

```
class ShoppingList extends React.Component {
  render() {
    return (
      <div className="shopping-list">
        <h1>Shopping List for {this.props.name}</h1>
        <ul>
          <li>Instagram</li>
          <li>WhatsApp</li>
          <li>Oculus</li>
        </ul>
      </div>
    );
  }
}

// Example usage: <ShoppingList name="Mark" />
```

Podemos usar componentes para dizer ao React o que queremos olhar na tela. Quando nossos dados mudam, ele irá eficientemente atualizar e re-renderizar nossos componentes. No exemplo da ShoppingList, temos que ela é um componente de React. Um componente de React pode receber parâmetros, chamados de *props* (propriedades), e

retornar a hierarquia de views através do método render(). Este método retorna uma descrição do que terá na tela, assim, o React recebe essa descrição e exibe o resultado.

Assim como as props, os componentes podem receber os chamados *states* (estados). Os states são como props, porém, eles são locais para cada componente e servem para atualizar mudanças de dados do componente. Para o componente ShoppingList, poderíamos mudar o primeiro item da lista “Instagram” para algo como “this.state.item”, que poderia mudar conforme a nossa necessidade. Suponhamos que quando o this.props.name for Mark, precisamos fazer uma requisição numa API, pegar o primeiro item da lista do Mark e renderizar este item. Para isso, podemos criar uma função que faz a requisição nessa API, retorna o resultado e atualiza o this.state.item com o item da compra. Quando nosso state muda, nosso componente será renderizado novamente, permitindo que a lógica e a renderização do nosso código esteja em apenas um componente.

Nossa aplicação utiliza dessa dinâmica do React para se comunicar com a API. Temos um arquivo escrito em Javascript chamado *helper.js* que é utilizado em nossos componentes para fazer a requisição na API e atualizar o estado dos nossos componentes, que, por consequência, irão mostrar as informações que queremos.

6. Documentação de uso do sistema

Pré-requisitos:
Node.js (https://nodejs.org/en/)
Nodemon

Para rodar o Frontend em React:
npm install
npm start

Para rodar o Backend em Node.js:

sudo npm install -g nodemon
npm install
npm start

Observações:

Caso acuse erro de pacote não encontrado, apenas digite o comando “npm install <nome_do_pacote>”. Isso pode ocorrer por causa da diferença do package.json para Linux e Windows.

Antes de iniciar o projeto, crie o banco de dados no MySQL e edite o usuário que irá conectar no arquivo connections.js.

Criar o banco de dados:

1. No diretório do arquivo .sql, abra um terminal.
2. Execute o comando mysql -u {seu_usuario} -p < create.sql
3. Digite sua senha do mysql
4. Repita o comando para o create_func.sql e insert.sql

Conectar ao banco de dados:

1. Na pasta backend, abrir o arquivo connections.js
2. Mude o usuário e senha da instância connection para o do seu MySQL

7. Deploy para produção

Para fazer o deploy do nosso frontend, podemos criar um servidor em Node. Primeiramente, precisamos criar a build do nosso projeto para o ambiente de produção. Para isso, basta rodar o comando npm run build na pasta frontend. Este comando irá gerar um diretório *build* com a build de produção da nossa aplicação. Depois disso, apenas precisamos subir um servidor para que nosso visitante seja servido com o arquivo index.html.

Uma das formas de subir este servidor é utilizando um servidor estático em Node. O jeito mais fácil de resolver isso seria instalar o pacote serve:

```
npm install -g serve
serve -s build
```

Este último comando irá servir seu site estático na porta 5000 (que pode ser alterada utilizando a flag -p ou --port).

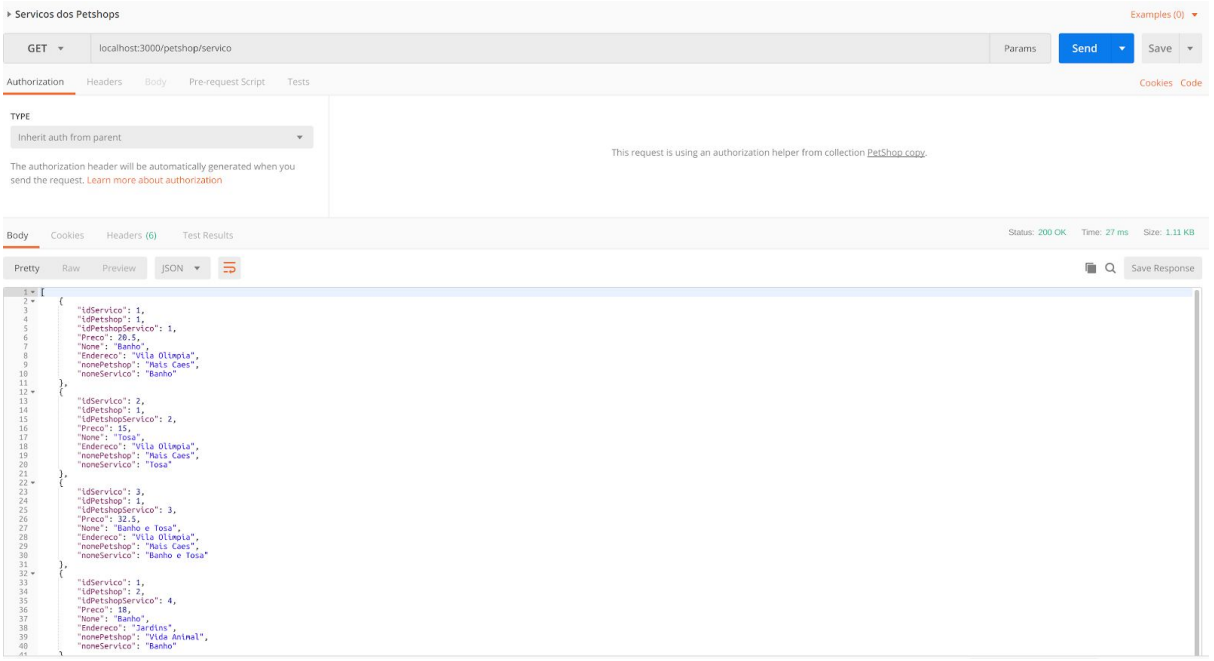
Para realizar outras soluções de deploy, podemos verificar a documentação do Create React App, que explica outras diversas formas de realizar o deploy: https://github.com/facebook/create-react-app/blob/master/packages/react-scripts/template/README.md#deployment

8. Ambiente de testes - Postman

Para testar nossa API, é possível utilizar uma série de requisições que criamos através do Postman. Para isso, faça o download do Postman (https://www.getpostman.com/), abra-o e localize o botão Import no menu. Selecione “Import from Link” e cole o seguinte link:
https://www.getpostman.com/collections/20661d6613a2ec176ef0
Com essa coleção, é possível fazer as requisições com bastante facilidade.

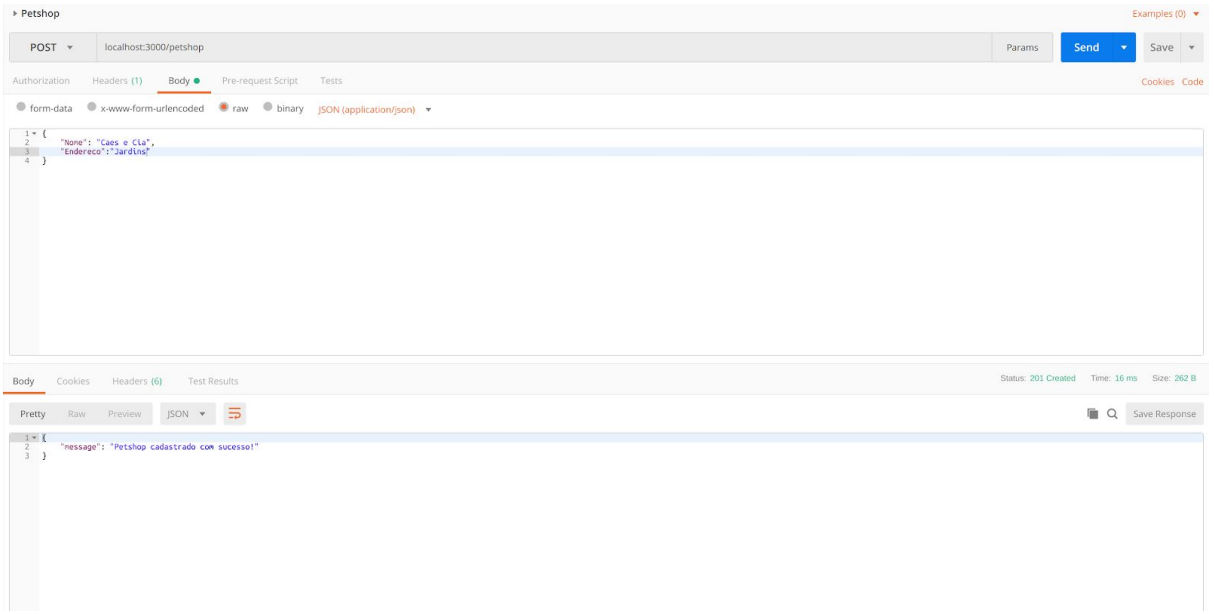
Estas requisições feitas no Postman, possibilita verificar as implementações que não foram críadas no front, como dito no índice 2.3 da documentação. Segue um passo a passo de como faz as requisições get, put, post e delete.

8.1 GET



-Na lista posicionada a esquerda selecionar GET
-digitar a URL do endpoint que faz a requisição desejada
-Apertar Send
As respostas serão fornecidas no campo em branco posicionados no canto inferior

8.2 POST

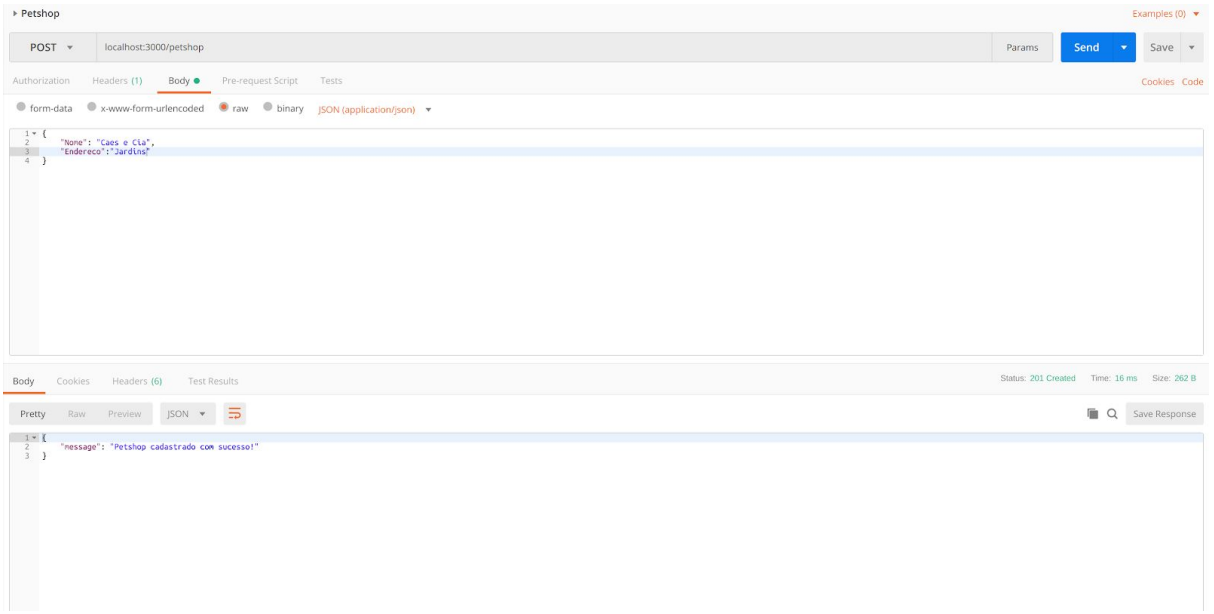


-Na lista posicionada a esquerda selecionar POST
-digitar a URL do endpoint que faz a requisição desejada
-Selecionar Body abaixo do local onde insere o endpoint
-Alterar para Raw
-Trocar text para JSON(application/json)
-Escrever os parâmetro a ser inseridos(para ver quais parâmetros necessário em cada endpoint, ver o índice 9 da documentação).Ex:

```
{
        "Nome": "Caes e Cia",
        "Endereco":"Jardins"
}
```

-Apertar Send
As respostas serão fornecidas no campo em branco posicionados no canto inferior

8.3 PUT

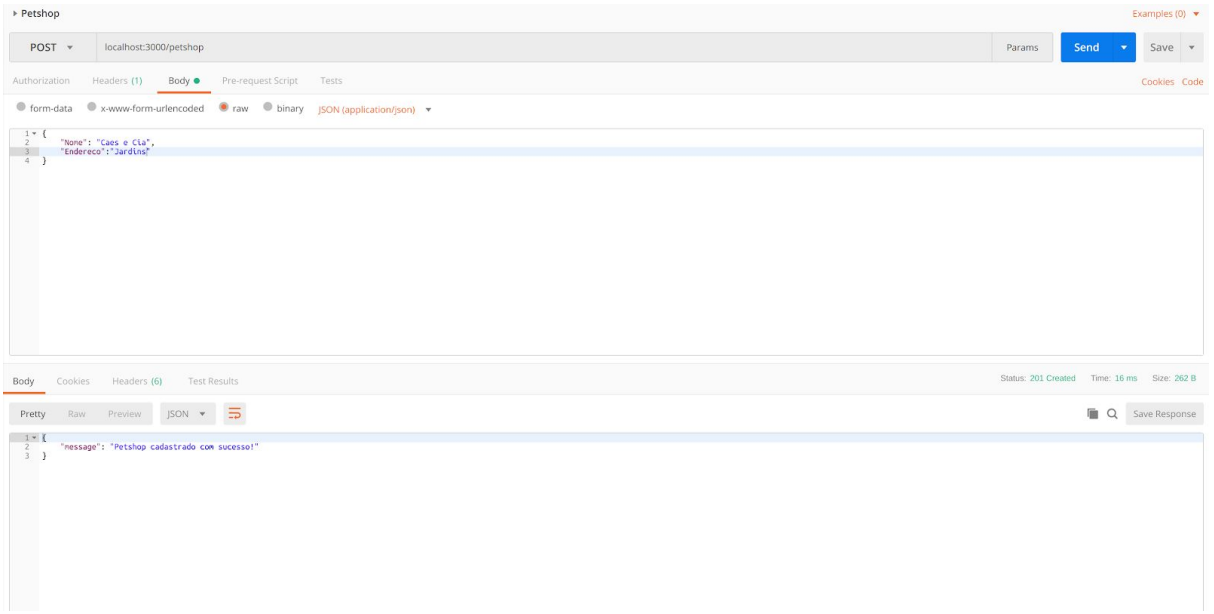


-Na lista posicionada a esquerda selecionar PUT
-digitar a URL do endpoint que faz a requisição desejada
-Selecionar Body abaixo do local onde insere o endpoint
-Alterar para Raw
-Trocar text para JSON(application/json)
-Escrever os parâmetro a ser inseridos(para ver quais parâmetros necessário em cada endpoint, ver o índice 9 da documentação).Ex:

```
{
        "idPetshop": 3,
        "Nome": "Pet Pet Shopper ATUALIZADO",
        "Endereco": "Rua Quatá, 400"
}
```

-Apertar Send
As respostas serão fornecidas no campo em branco posicionados no canto inferior

8.4 DELETE



-Na lista posicionada a esquerda selecionar DELETE
-digitar a URL do endpoint que faz a requisição desejada
-Selecionar Body abaixo do local onde insere o endpoint
-Alterar para Raw
-Trocar text para JSON(application/json)
-Escrever os parâmetro a ser inseridos(para ver quais parâmetros necessário em cada endpoint, ver o índice 9 da documentação).Ex:

```
{
        "idPetshop": 3
}
```

-Apertar Send
As respostas serão fornecidas no campo em branco posicionados no canto inferior

Para muitas requisições, já tem um exemplo feito, bastando selecionar a endpoint e requisição desejada.

9. Especificações REST API

Raca

Raca - Leitura dos dados da Raça

GET
/raca

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idRaca	int	ID da Raca
Nome	string	Nome da Raca

Raca - Criação da Raça

POST
/raca

Parametros

Campo	Tipo	Descrição
Nome	string	Nome da Raca

Raca - Atualização da Raça

PUT
/raca

Parametros

Campo	Tipo	Descrição
Nome	string	Nome da Raca atual
novoNome	string	Nome da Raca desejada

Raca - Deletar Raça

DELETE
/raca

Parametros

Campo	Tipo	Descrição
Nome	string	Nome da Raca

Tipo

Tipo - Leitura dos dados do Tipo

GET
/tipo

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idTipo	int	ID do Tipo
Nome	string	Nome do Tipo

Tipo - Criação do Tipo

POST
/tipo

Parametros

Campo	Tipo	Descrição
Nome	string	Nome do Tipo

Tipo - Atualização do Tipo

PUT
/tipo

Parametros

Campo	Tipo	Descrição
Nome	string	Nome do Tipo atual
novoNome	string	Nome do Tipo desejada

Tipo - Deletar Tipo

DELETE
/tipo

Parametros

Campo	Tipo	Descrição
Nome	string	Nome do Tipo

Dono

Dono - Leitura dos dados do Dono

GET
/dono

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idDono	int	ID do Dono
Nome	string	Nome do Dono
RG	int	Documento do Dono
UF	string	Estado do Dono

Email	string	Email utilizado para identificar o Dono

Dono - Leitura dos dados específicos do dono

GET
/dono/email

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idDono	int	ID do Dono
Nome	string	Nome do Dono
RG	int	Documento do Dono
UF	string	Estado do Dono
Email	string	Email utilizado para identificar o Dono

Dono - Quantidade de Donos na Base

GET
/dono/count

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
num	int	Soma da quantidade do Donos

Dono - Criação de novo Dono

POST
/dono

Parametros

Campo	Tipo	Descrição
Nome	string	Nome do Dono
RG	int	Documento do Dono
UF	string	Estado do Dono
Email	string	Email utilizado para identificar o Dono

Dono - Atualização do Dono

PUT
/dono

Parametros

Campo	Tipo	Descrição
Email	string	Email utilizado para identificar o Dono

Dono - Deletar Dono

DELETE
/dono

Parametros

Campo	Tipo	Descrição
Email	string	Email utilizado para identificar o Dono

Animal

Animal - Leitura dos dados do Animal

GET
/animal

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idAnimal	int	ID do Animal
Nome	string	Nome do Animal
Raca	string	Raca do Animal
Tipo	string	Qual tipo do animal
Sexo	string	Sexo do Animal
dataNascimento	date	Data de nascimento do animal
Vivo	boolean	Status se o animal esta vivo ou morto

Animal - Leitura dos dados dos Animais vivos

GET
/animal/vivo

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idAnimal	int	ID do Animal

Nome	string	Nome do Animal
Raca	string	Raca do Animal
Tipo	string	Qual tipo do animal
Sexo	string	Sexo do Animal
dataNascimento	date	Data de nascimento do animal
Vivo	boolean	Status se o animal esta vivo ou morto

Animal - Criação de novo Animal

POST
/animal

Parametros

Campo	**Tipo**	**Descrição**
Nome	string	Nome do Animal
Raca	string	Raca do Animal
Tipo	string	Qual tipo do animal
Sexo	string	Sexo do Animal
dataNascimento	date	Data de nascimento do animal
Vivo	boolean	Status se o animal esta vivo ou morto
Email	string	Email utilizado para identificar o Dono

Animal - Atualização do Animal

PUT
/animal

Parametros

Campo	**Tipo**	**Descrição**
idAnimal	int	ID do Animal

Animal - Obito do Animal

PUT
/animal/morto

Parametros

Campo	Tipo	Descrição
idAnimal	int	ID do animal
Vivo	boolean	Status se o animal esta vivo ou morto

Petshop

Petshop - Leitura dos dados do Petshop

GET
/petshop

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idPetshop	int	ID do Petshop
Nome	string	Nome do Petshop

Petshop - Criação do Petshop

POST
/petshop

Parametros

Campo	Tipo	Descrição
Nome	string	Nome do Petshop

Petshop - Atualização do Petshop

PUT
/petshop

Parametros

Campo	Tipo	Descrição
Nome	string	Nome do Petshop atual
novoNome	string	Nome do Petshop desejada

Petshop - Deletar Petshop

DELETE
/petshop

Parametros

Campo	Tipo	Descrição
Nome	string	Nome do Petshop

Servico

Servico - Leitura dos dados do Servico

GET
/petshop

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição

idServico	int	ID do Servico
Nome	string	Nome do Servico

Servico - Atualização do Servico

PUT
/petshop

Parametros

Campo	Tipo	Descrição
Nome	string	Nome do Servico atual
novoNome	string	Nome do Servico desejada

Dono_Animal

Dono_Animal - Listagens do animal de um dono

GET
/dono/animal/:Email

Parametros

Campo	Tipo	Descrição
Email	string	Email utilizado para identificar o Dono

Success

Campo	Tipo	Descrição
idAnimal	int	ID do Animal
Nome	string	Nome do Animal
racaAnimal	string	Raca do Animal
tipoAnimal	int	Tipo do Animal
Sexo	string	Sexo do Animal
dataNascimento	date	Data de Nascimento do Animal

Vivo	boolean	Status do Animal

Dono_Animal - Listagens dos donos de um animal

GET
/animal/dono/:idAnimal

Parametros

Campo	Tipo	Descrição
idAnimal	int	ID do Animal

Success

Campo	Tipo	Descrição
idDono	int	ID do Dono
Nome	string	Nome do Dono
RG	int	Documento do Dono
UF	string	Estado do Dono
Email	string	Email do Dono

Dono_Animal - Atualizar relação do dono com animal

POST
/dono/animal

Parametros

Campo	Tipo	Descrição
Nome	string	Nome do Animal
Email	string	Email do dono

Petshop_Servico

Petshop_Servico - Leitura dos serviços em cada petshop

GET
/petshop/servico

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idPetshop	int	ID do Petshop
idServico	int	Id do Servico
Preco	int	Preco do Servico no petshop
nomePetshop	string	Nome do petshop
Endereco	string	Endereco do Petshop
nomeServico	date	Nome do Servico

Petshop_Servico - Leitura dos serviços em um petshop

GET
/petshop/servico/:idPetshop

Parametros

Campo	Tipo	Descrição
idPetshop	int	ID do Petshop

Success

Campo	Tipo	Descrição
idPetshop	int	ID do Petshop
idServico	int	Id do Servico
Preco	int	Preco do Servico no petshop
nomePetshop	string	Nome do petshop
Endereco	string	Endereco do Petshop

nomeServico	date	Nome do Servico

Petshop_Servico - Criação dos serviços em um petshop

- Foi criada um stored procedure chamada createPetshopService que compara se o nome do serviço já existe, caso exista, cria vincula ao petshop, caso contrário, cria o serviço antes de vincular ao petshop.

POST
/petshop/servico/

Parametros

Campo	**Tipo**	**Descrição**
idPetshop	int	ID do Petshop
nomeServico	string	Nome da serviço a ser criado
Preco	int	Preço do serviço no Petshop

Petshop_Servico - Atualizando serviços de um petshop

PUT
/petshop/servico/

Parametros

Campo	**Tipo**	**Descrição**
idPetshop	int	ID do Petshop
nomeServico	string	Nome da serviço a ser criado
Preco	int	Novo Preço do serviço no Petshop

Petshop_Servico - Deletar serviços de um petshop

- Foi criado uma trigger que antes de deletar um servico, o servico que está vinculado ao animal será removido também, não é o ideal para uma aplicação, já que geralmente queremos ter um histórico do que foi feito anteriormente, porém, criamos esta trigger para exercitar nosso conhecimento sobre o MySQL

DELETE
/petshop/servico/

Parametros

Campo	Tipo	Descrição
idPetshopServico	int	ID do Petshop

Animal_Servico

Animal_Servico - Leitura dos serviços em cada animal

GET
/animal/servico

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idAnimal	int	ID do Animal
Concluido	boolean	Status se o servico foi feito
Agenda	date	Data a ser feito o Servico
idPetshop	int	ID do Petshop
idServico	int	Id do Servico
Preco	int	Preco do Servico no petshop
nomePetshop	string	Nome do petshop
Endereco	string	Endereco do Petshop
nomeServico	date	Nome do Servico

Animal_Servico - Leitura dos serviços em aberto

GET
/animal/servico/pending

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idAnimal	int	ID do Animal
Concluido	boolean	Status se o servico foi feito
Agenda	date	Data a ser feito o Servico
idPetshop	int	ID do Petshop
idServico	int	Id do Servico
Preco	int	Preco do Servico no petshop
nomePetshop	string	Nome do petshop
Endereco	string	Endereco do Petshop
nomeServico	date	Nome do Servico

Animal_Servico - Leitura dos serviços em aberto de um animal

GET
/animal/servico/pending/:idAnimal

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
idAnimal	int	ID do Animal
Concluido	boolean	Status se o servico foi feito
Agenda	date	Data a ser feito o Servico

idPetshop	int	ID do Petshop
idServico	int	Id do Servico
Preco	int	Preco do Servico no petshop
nomePetshop	string	Nome do petshop
Endereco	string	Endereco do Petshop
nomeServico	date	Nome do Servico

Animal_Servico - Leitura dos animais que geram maior receita

GET
/animal/servico/top5

Parametros

Campo	**Tipo**	**Descrição**
none	-	-

Success

Campo	**Tipo**	**Descrição**
idAnimal	int	ID do Animal
nomeAnimal	string	Nome do Animal
Receita	int	Gasto total do animal
nomeRaca	string	Raca do animal
nomeTipo	string	Tipo de Animal

Animal_Servico - Leitura dos serviços concluidos de um animal

GET
/animal/servico/complete/:idAnimal

Parametros

Campo	**Tipo**	**Descrição**
none	-	-

Success

Campo	Tipo	Descrição
idAnimal	int	ID do Animal
Concluido	boolean	Status se o servico foi feito
Agenda	date	Data a ser feito o Servico
idPetshop	int	ID do Petshop
idServico	int	Id do Servico
Preco	int	Preco do Servico no petshop
nomePetshop	string	Nome do petshop
Endereco	string	Endereco do Petshop
nomeServico	date	Nome do Servico

Animal_Servico - Leitura do gasto de um animal

GET
/animal/servico/sum/:idAnimal

Parametros

Campo	Tipo	Descrição
idAnimal	int	ID do Animal

Success

Campo	Tipo	Descrição
num	int	Soma dos servicos feitos pelo animal

Animal_Servico - Leitura da receita de dos petshops por petshop

GET
/animal/servico/sum/petshop/all

Parametros

Campo	Tipo	Descrição
none	-	-

Success

Campo	Tipo	Descrição
num	int	Soma dos servicos
Nome	string	Nome do Petshop

Animal_Servico - Leitura da receita de um petshop

GET
/animal/servico/sum/petshop/:idPetshop

Parametros

Campo	Tipo	Descrição
idPetshop	int	ID do Petshop

Success

Campo	Tipo	Descrição
num	int	Soma dos servicos

Animal_Servico - Criação de um serviço para um animal

POST
/animal/servico/

Parametros

Campo	Tipo	Descrição
idAnimal	int	ID do Animal
nomePetshop	string	Nome do Petshop em que o animal faz o serviço
nomePetshop	string	Nome do Servico que o animal fara
Agenda	date	Dia em que o serviço está agendado

Animal_Servico - Conclui um serviço de um animal

PUT
/animal/servico/close

Parametros

Campo	Tipo	Descrição
idAnimalServico	int	ID do Animal

Animal_Servico - Deleta um serviço de um animal

DELETE
/animal/servico/

Parametros

Campo	Tipo	Descrição
idAnimalServico	int	ID do Animal