

Projeto - Pet Shop

Sistema Final

SCC0219 - Introdução ao Desenvolvimento Web - 1o Semestre 2020

Docente: Prof. Dr. Dilvan de Abreu Moreira

Estagiária PAE: Mariany Morais Silva

Integrantes do grupo:

Bruno dos Santos, 10786170

Paulo Henrique da Silva 10734515

Vitor Santana Cordeiro, 10734345

São Carlos, 25 de Julho de 2020

1. Requisitos

Abaixo se encontram os requisitos estipulados para o sistema da Pet Shop.

1.1 Requisitos Funcionais

- O sistema deve acomodar dois tipos de usuários: clientes e administradores.
 - Os administradores são responsáveis por registrar e gerenciar clientes, produtos, serviços e novos administradores.
 - A aplicação começa com um administrador *default*, de *admin-name* “admin” e senha “admin”.
 - Os clientes serão compradores em potencial que acessam nossa aplicação, onde será possível a visualização e compra de produtos e agendamento de serviços para seus pets.
- O sistema deverá permitir que um novo cliente se cadastre no mesmo, fornecendo: CPF, nome, endereço, foto, telefone, e-mail e senha.
- O sistema deverá permitir que um administrador cadastre a um outro, fornecendo: CPF, nome, endereço, foto, telefone, e-mail, admin-name (uma espécie de *username* para os administradores) e senha do novo administrador.
- O sistema deverá possuir um calendário com *slots* de serviço:
 - Esses slots deverão ser de 1 hora de duração, sendo que o calendário em si deve exibir os horários de 08:00 às 18:00 (horário comercial da Pet Shop), totalizando **10 slots** por dia.
 - Cada *slot* poderá possuir apenas um único serviço associado.
 - Cada *slot* poderá estar em três estados diferentes:
 - Vazio: nenhum serviço foi registrado para ele ainda. O *slot* deverá mostrar a palavra “EMPTY”.
 - Disponível: um serviço determinado (e.g. tosa, castração, banho) foi definido, por um dos administradores, para ele. O *slot* deverá mostrar o nome do serviço que foi associado e sua respectiva imagem representativa.

- Agendado: um serviço foi definido para ele e já foi agendado por um usuário. O *slot* deverá mostrar, além do nome do serviço e sua respectiva imagem representativa, o nome e a foto do pet que irá receber o serviço.
- O sistema deverá permitir ao administrador associar serviços (já cadastrados no banco de dados) a um *slot* que esteja vazio.
- O sistema deverá permitir ao administrador editar os serviços marcados como disponíveis nos *slots*, removendo-os (voltando o *slot* para o estado vazio) ou trocando o tipo de serviço que poderá ser prestado naquele *slot*. Contudo, o administrador só poderá editar um *slot* caso nenhum cliente tenha lhe reservado.
- O sistema deverá implementar um carrinho, em que os serviços a serem agendados e os produtos selecionados pelo cliente deverão estar organizados e suas informações (foto, nome e preço) disponíveis. Além disso, deverá ser mostrado o preço total associado a todos os itens de cada uma dessas categorias (produtos e serviços).
 - O cliente só poderá finalizar a compra a partir do carrinho.
 - O carrinho só será limpo após a compra ser finalizada ou se o cliente decidir deliberadamente fazê-lo.
- O sistema deverá permitir que o cliente registre um pet, sendo necessário fornecer seu nome, uma foto, raça e idade. O cliente só poderá agendar um serviço caso tenha o respectivo pet registrado no sistema.
- O sistema deverá permitir ao cliente agendar um serviço em um *slot* marcado como disponível. Assim que a compra for concluída, o calendário deverá ser atualizado para mostrar que o *slot* em questão foi reservado para um dos pets desse cliente.
- O sistema deverá permitir ao administrador cadastrar novos produtos, bem como apagar produtos antigos e atualizar suas informações (e.g. quantidade em estoque).
- O cliente poderá adicionar uma determinada quantidade de um produto ao seu carrinho, contanto que esta não ultrapasse a quantidade em estoque para o mesmo.

- Caso a venda seja concluída, a quantidade selecionada deverá ser subtraída da quantidade em estoque.
- Cada produto do sistema deverá ter um atributo que indique a quantidade de unidades já vendidas dele, bem como cada serviço deverá possuir um atributo que indique quantas vezes este já foi agendado.
- O sistema deverá possuir uma “tela de ganhos”, uma seção ou página com o intuito de resumir, para o administrador, todos os produtos vendidos e serviços agendados, bem como o ganho total para cada um deles.
 - Para o produto, deverá mostrar, para todos aqueles que já foram vendidos pelo menos uma vez, seu nome, foto, descrição, preço e quantidade total vendida.
 - Para o serviço, deverá mostrar, para todos aqueles que já foram agendados pelo menos uma vez, seu nome, foto, descrição, preço e número total de agendamentos.

1.2 Requisitos Não-funcionais (dados persistentes)

O sistema será implementado utilizando um banco de dados NoSQL orientado a documentos (MongoDB), utilizando das seguintes estruturas lógicas:

- Administrador: id, CPF, nome, foto, fone, email, admin-name, senha e administradores “filhos” (cadastrados por ele).
- Cliente: id, CPF, nome, email, senha, foto, endereço e fone.
- Pet: id, nome, dono (um Cliente), foto, raça, idade e slug.
- Produto: id, nome, slug, tags, foto, descrição, preço, quantidade em estoque, total vendido, ativo (booleano).
- Serviço: id, nome, slug, tags, foto, descrição, preço, responsável, total de vezes que já foi consumido, ativo (booleano).
- Slot: id, cliente, serviço, data, pet, estado (Vazio, Disponível, Agendado).
- Pedido: id, número, cliente, data, produtos e serviços comprados.

2. Descrição do projeto

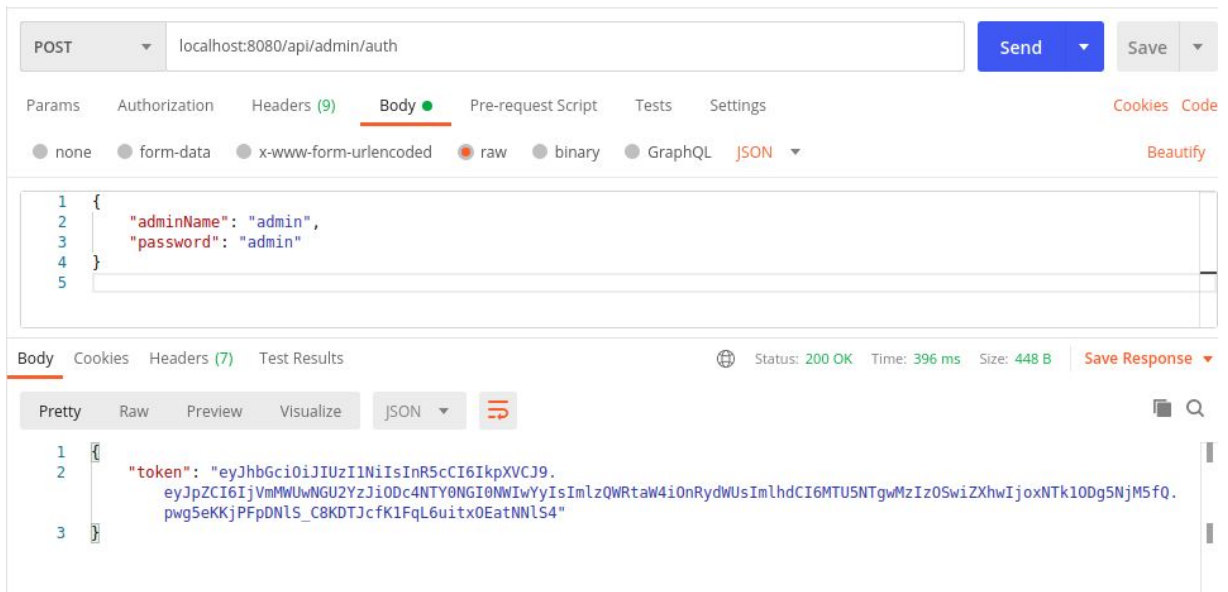
O nosso projeto utiliza do MongoDB para ser a nossa base de dados e o NodeJS, com a biblioteca Express, para a nossa API. Utilizamos de mais algumas bibliotecas/frameworks para automatizar alguns processos, como encriptar senhas, habilitar o CORS, gerar slugs, UUIDs e tokens automaticamente e validações nos campos da requisições.

Para o Front-end, optamos por utilizar o Vue.js e, como já tínhamos despendido muito esforço com a estilização da Mockup, decidimos utilizar seus estilos e htmls para gerar os componentes da aplicação final. Nossa aplicação é uma SPA (Single-Page Application).

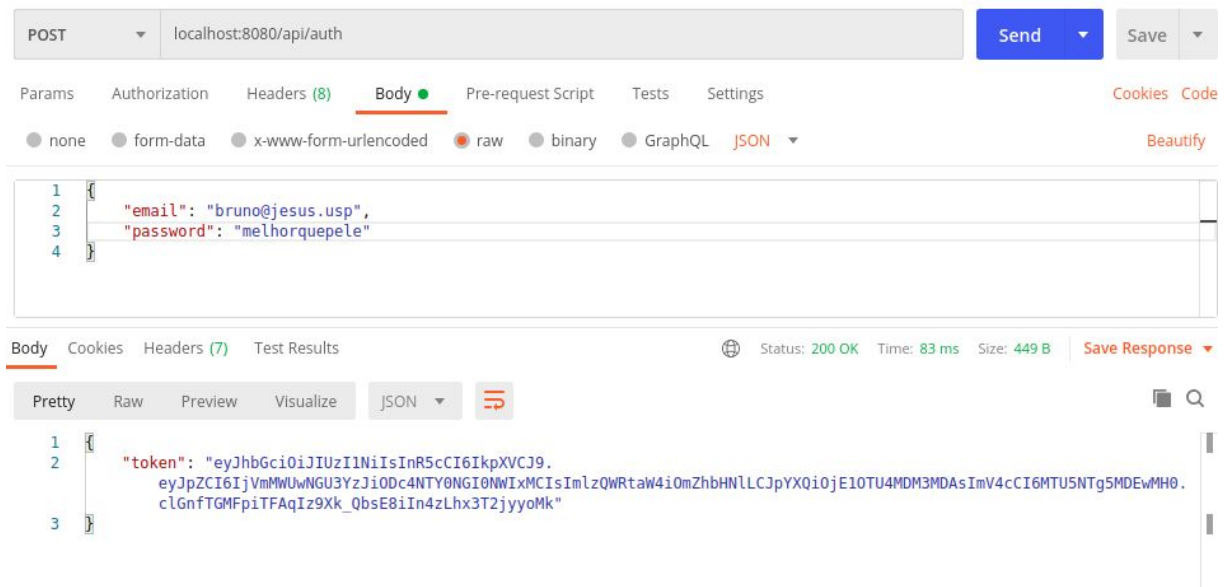
3. Testes

Os testes da API foram feitos usando o Postman. Fizemos uma função para popular o banco com dados iniciais (backend/src/populate.js). Abaixo, seguem os resultados dos testes às principais rotas:

- Login do Admin



- Login do Cliente



- Pegar info dos “filhos” do admin (i.e. os admins criados por ele)

GET localhost:8080/api/admin/children Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (7) Test Results Status: 200 OK Time: 24 ms Size: 769 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "childAdmins": [],
4     "id": "5f1e04e7c2b8785644b45b0d",
5     "CPF": "11111111112",
6     "name": "Bruno Santos",
7     "phone": "11945677438",
8     "email": "brunos@petshop.com",
9     "adminName": "bruno",
10    "_v": 0
11  },
12  {
13    "childAdmins": [],
14    "id": "5f1e04e7c2b8785644b45b0e",
15    "CPF": "11111111113",
16    "name": "Paulo Henrique",
17    "phone": "11954787439",
18    "email": "pauloh@petshop.com",
19    "adminName": "paulo",
20    "_v": 0
21  }
22 ]
```

- Criar novo admin (se tornará filho daquele que o criou). Além do status de retorno (201 - CREATED), na segunda imagem é possível ver que o admin que registramos volta na resposta dos filhos do admin root.

POST localhost:8080/api/admin/children Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "CPF": "12345678910",
3   "name": "Admin Teste",
4   "phone": "11945684321",
5   "email": "oi@eu.com",
6   "adminName": "novo-adm",
7   "password": "entreigalera"
8 }
```

Body Cookies Headers (7) Test Results Status: 201 Created Time: 98 ms Size: 383 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "CPF": "12345678910",
3   "name": "Admin Teste",
4   "phone": "11945684321",
5   "email": "oi@eu.com",
6   "adminName": "novo-adm",
7   "password": "entreigalera"
8 }
```

GET localhost:8080/api/admin/children Send Save

Params Authorization **Headers (9)** Body ● Pre-request Script Tests Settings Cookies Code

Headers 8 hidden

KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
<div> <div>Body</div> <div>Cookies</div> <div>Headers (7)</div> <div>Test Results</div> </div> <div> Status: 200 OK Time: 28 ms Size: 935 B Save Response </div>					

Pretty
Raw
Preview
Visualize
JSON
≡

```

25     "CPF": "11111111114",
26     "name": "Vitor Santana",
27     "phone": "11947831245",
28     "email": "vitors@petshop.com",
29     "adminName": "vitor",
30     "__v": 0
31   },
32   {
33     "childAdmins": [],
34     "id": "5f1e08f3c2b8785644b45b1b",
35     "CPF": "12345678910",
36     "name": "Admin Teste",
37     "phone": "11945684321",
38     "email": "oi@eu.com",
39     "adminName": "novo-adm",
40     "__v": 0
  }

```

- Pegar todos os produtos (lado do cliente)

GET localhost:8080/api/products Send Save

Params Authorization **Headers (9)** Body ● Pre-request Script Tests Settings Cookies Code

Body Cookies Headers (7) Test Results Status: 200 OK Time: 33 ms Size: 11.96 KB Save Response

Pretty
Raw
Preview
Visualize
JSON
≡

```

1  [
2    {
3      "tags": [
4        "foods"
5      ],
6      "totalSold": 17,
7      "active": true,
8      "id": "5f1e38d16290b1119305e431",
9      "slug": "racaio-para-gatos-1",
10     "name": "Ração para gatos 1",
11     "photo": "https://raw.githubusercontent.com/paulo-hs/Web-PetShop/master/real-deal/frontend/public/images/racoes/racao-gato.jpg",
12     "description": "Ração deliciosa e nutritiva para o seu gatinho.",
13     "price": 9.99,
14     "inStock": 42,
15     "__v": 0
16   },
17   {
18     "tags": [
19       "foods"
  
```

- Pegar todos os serviços (lado do cliente)

GET localhost:8080/api/services Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

Body Cookies Headers (7) Test Results Status: 200 OK Time: 14 ms Size: 2.35 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "tags": [
4       "tosa",
5       "banho"
6     ],
7     "totalBooked": 10,
8     "active": true,
9     "id": "5f1e38d16290b119305e42b",
10    "slug": "tosa",
11    "name": "Tosa",
12    "photo": "https://raw.githubusercontent.com/paulo-hs/Web-PetShop/master/real-deal/frontend/public/images/
13      categorias/banhotosa.png",
14    "description": "Uma tosa bem suave para seu bichinho",
15    "responsible": "Paulo Henrique",
16    "price": 50,
17    "__v": 0
18  },
19  {
20    "tags": [
```

- O cliente poder ver suas informações

GET localhost:8080/api/profile Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Body Cookies Headers (7) Test Results Status: 200 OK Time: 51 ms Size: 505 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "5f1e38d16290b119305e42a",
3   "CPF": "48061459912",
4   "name": "Bruno Jesus",
5   "photo": "https://conhecimentocientifico.r7.com/wp-content/uploads/2020/04/jesus-5.jpg",
6   "phone": "16912359876",
7   "email": "bruno@jesus.usp",
8   "address": "Rua Carlos de Camargo Salles",
9   "__v": 0
10 }
```

- Pegar todos os pets do cliente

GET localhost:8080/api/profile/pets Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Body Cookies Headers (7) Test Results Status: 200 OK Time: 13 ms Size: 481 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "_id": "5f1e38d26290b119305e456",
4     "slug": "tchutchucacao",
5     "name": "Tchutchucão",
6     "photo": "https://raw.githubusercontent.com/paulo-hs/Web-PetShop/master/real-deal/frontend/public/images/tchutchucacao.jpg",
7     "race": "Dog do mal",
8     "age": 5,
9     "_v": 0
10  }
11 }
```

- Cliente criar um pet (após logado)

POST localhost:8080/api/profile/pets Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

Body Cookies Headers (7) Test Results Status: 201 Created Time: 24 ms Size: 313 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "name": "Uauuau",
3   "race": "Dog belga",
4   "age": 17,
5   "photo": "anyphoto"
6 }
```

- Criar um pedido

POST localhost:8080/api/orders Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
1 {
2   "services": [
3     {
4       "service": "5f1e38d16290b1119305e42b",
5       "pet": "5f1e38d26290b1119305e456",
6       "date": 1595851200000
7     }
8   ]
9 }
```

Body Cookies Headers (7) Test Results Status: 201 Created Time: 14 ms Size: 357 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "services": [
3     {
4       "service": "5f1e38d16290b1119305e42b",
5       "pet": "5f1e38d26290b1119305e456",
6       "date": 1595851200000
7     }
8   ]
9 }
```

- Ver todos os pedidos realizados e suas informações (somente para admins)

GET localhost:8080/api/admin/orders Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code

Body Cookies Headers (7) Test Results Status: 200 OK Time: 253 ms Size: 1.65 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "_id": "5f1e38d26290b1119305e457",
4     "number": "d6e6483c-587f-4aaf-9254-b8293b6a7ae7",
5     "customer": {
6       "_id": "5f1e38d16290b1119305e42a",
7       "CPF": "48061459912",
8       "name": "Bruno Jesus",
9       "email": "bruno@jesus.usp"
10    },
11    "services": [
12      {
13        "_id": "5f1e38d26290b1119305e458",
14        "service": {
15          "tags": [
16            "tosa",
17            "banho"
18          ],
19        "_id": "5f1e38d16290b1119305e42b",
20        "description": "Uma tosa bem suave para seu bichinho",

```

4. Para rodar

Para rodar a nossa aplicação, você deverá ter o Docker instalado em sua máquina (<https://docs.docker.com/get-docker>). Ele será utilizado para isolar o nosso banco de dados MongoDB e conseguir fazê-lo portátil e fácil de usar. Você também poderá, utilizando o Docker, acessar ao localhost:8081 para conseguir ver as informações do banco (usuário: admin, senha: admin).

Para rodar o servidor, você precisará do NPM (Node Package Manager), e para rodar o frontend você precisará do YARN (outro package manager para o Node.js).

Para iniciar o servidor, basta rodar, na sequência:

- `docker-compose up -d` (caso você tenha baixado o Docker agora, esse processo irá demorar um pouco pois ele irá baixar as imagens dos containers)
- `npm install`
- `npm start`

Para o frontend, rode:

- `yarn serve`

Essas mesmas instruções também se encontram nos arquivos README.md das pastas frontend e backend.

5. Problemas

Apesar de conseguir terminar completamente o backend, não conseguimos resultados semelhantes no frontend devido ao tempo curto. Alguns dos requisitos propostos na seção 1 não puderam ser integrados ao front (apesar de estarem funcionando no back).