UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
Departamento de Ciências de Computação

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

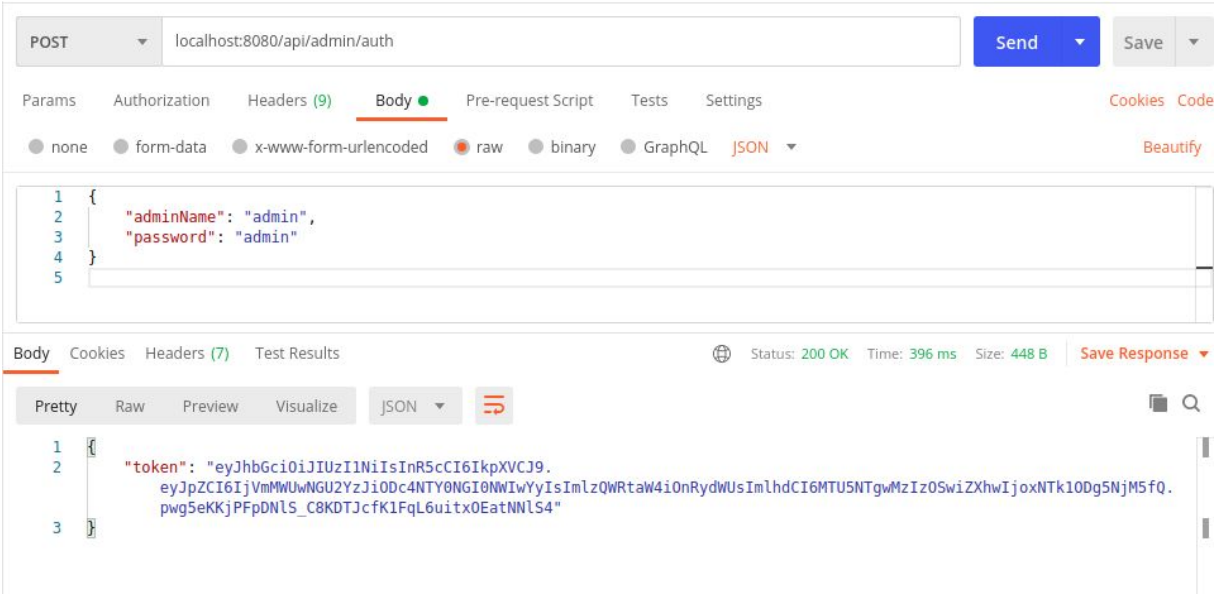
O nosso projeto utiliza do MongoDB para ser a nossa base de dados e o NodeJS, com a biblioteca Express, para a nossa API. Utilizamos de mais algumas bibliotecas/frameworks para automatizar alguns processos, como encriptar senhas, habilitar o CORS, gerar slugs, UUIDs e tokens automaticamente e validações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

Para o Front-end, optamos por utilizar o Vue.js e, como já tinhamos 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).

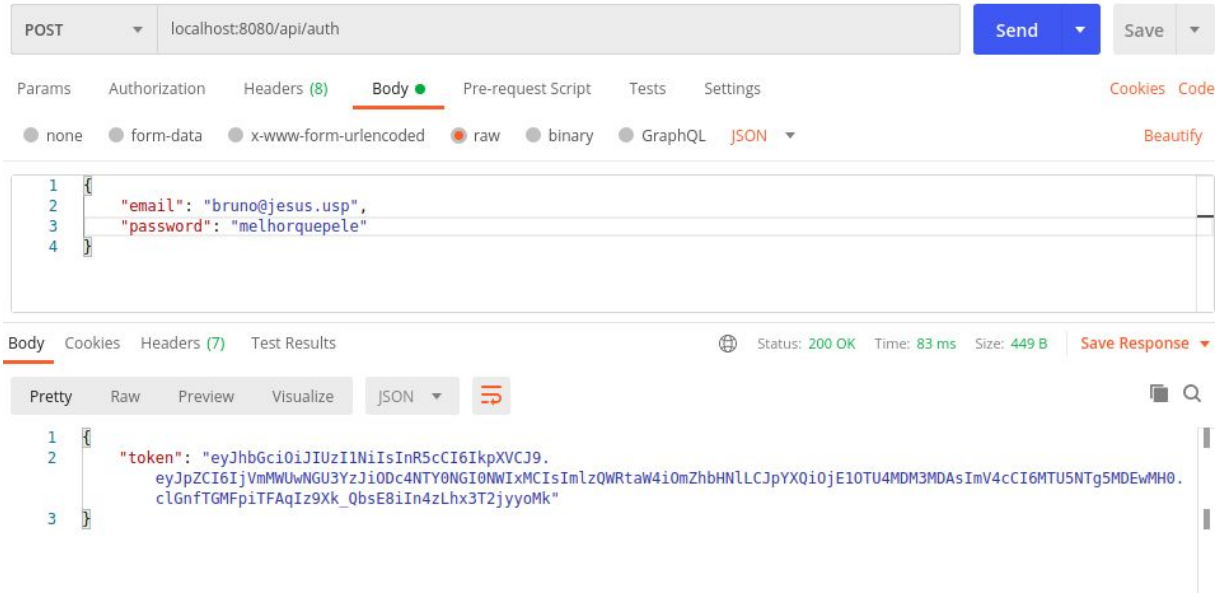
3. Testes

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:

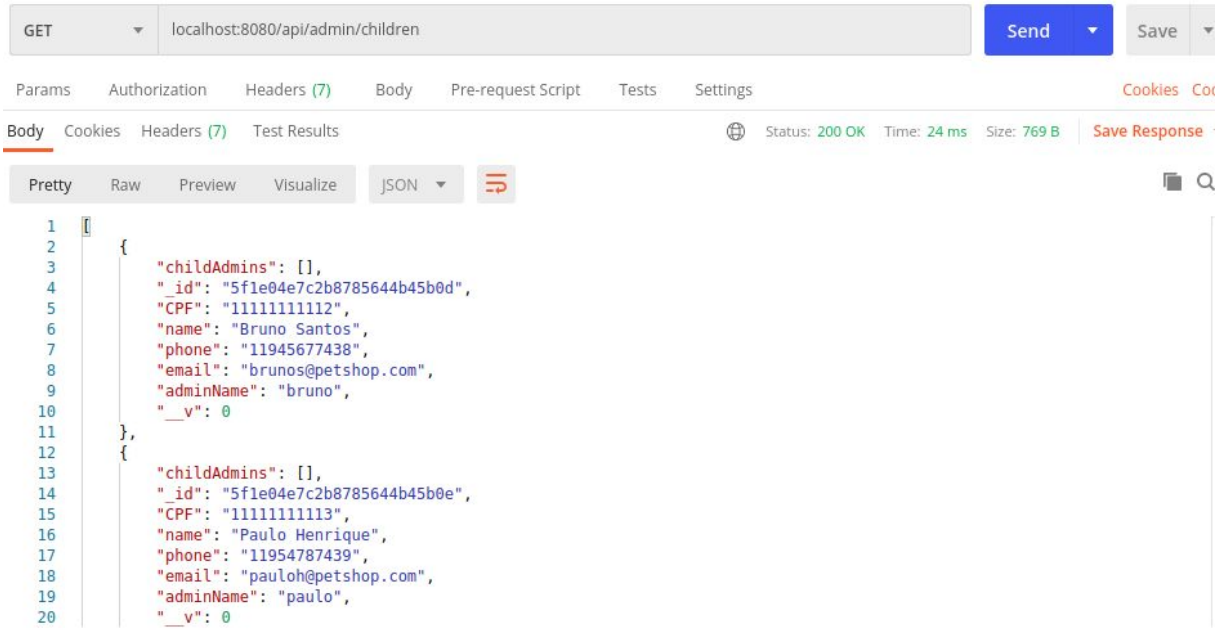
- Login do Admin



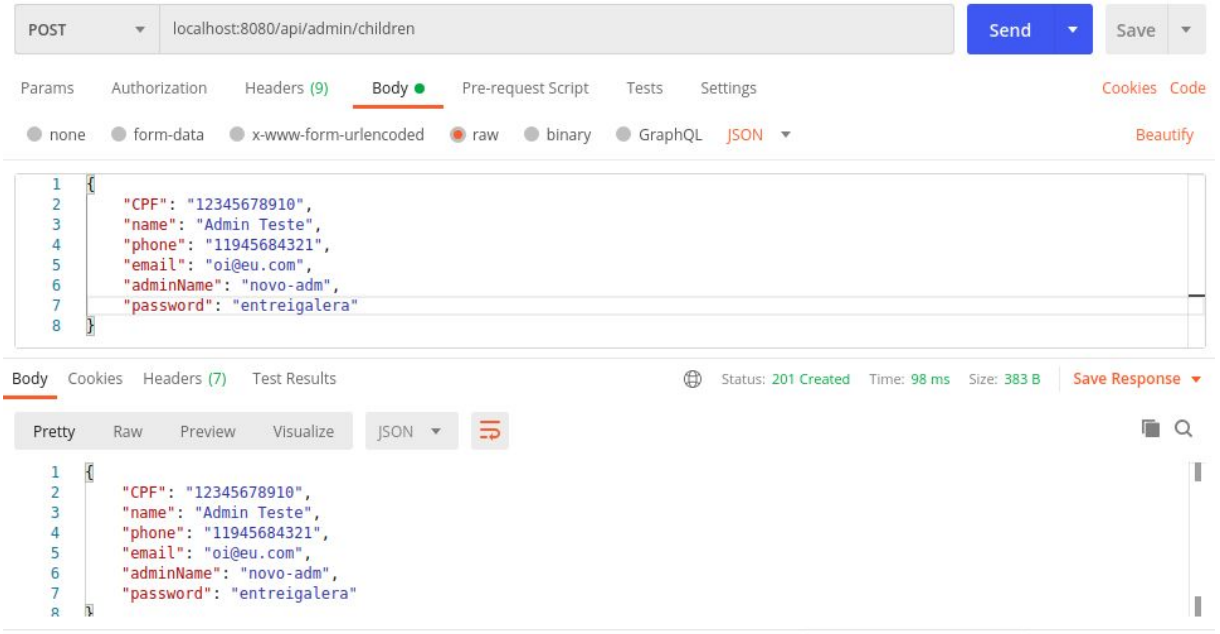
- Login do Cliente

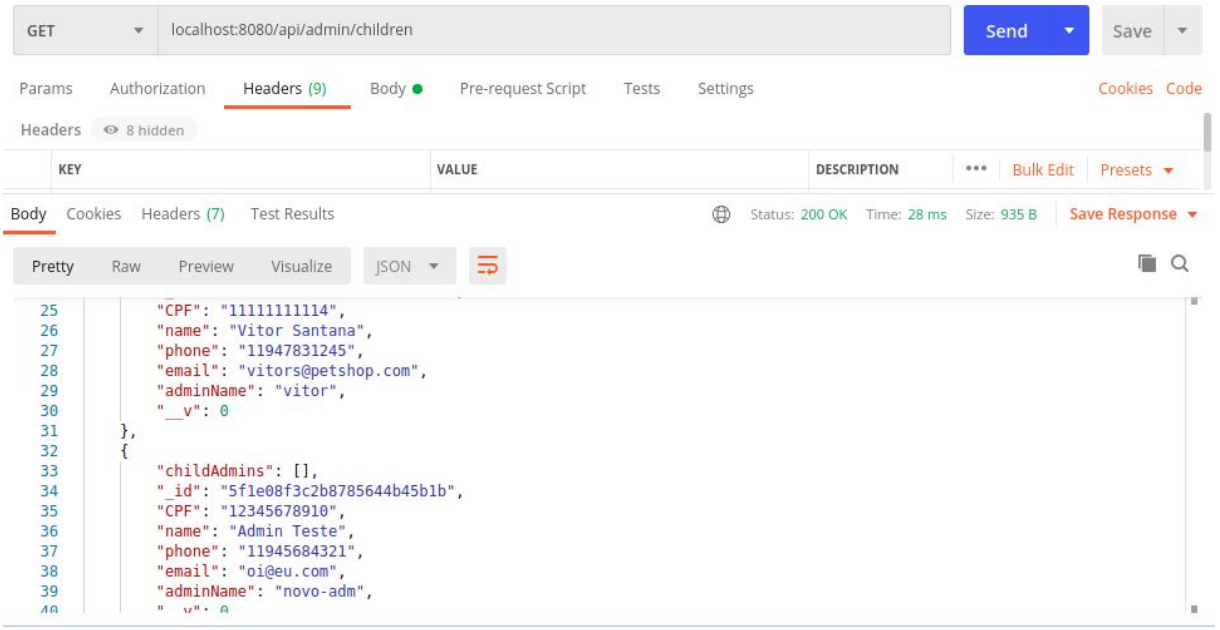


- Pegar info dos “filhos” do admin (i.e. os admins criados por ele)

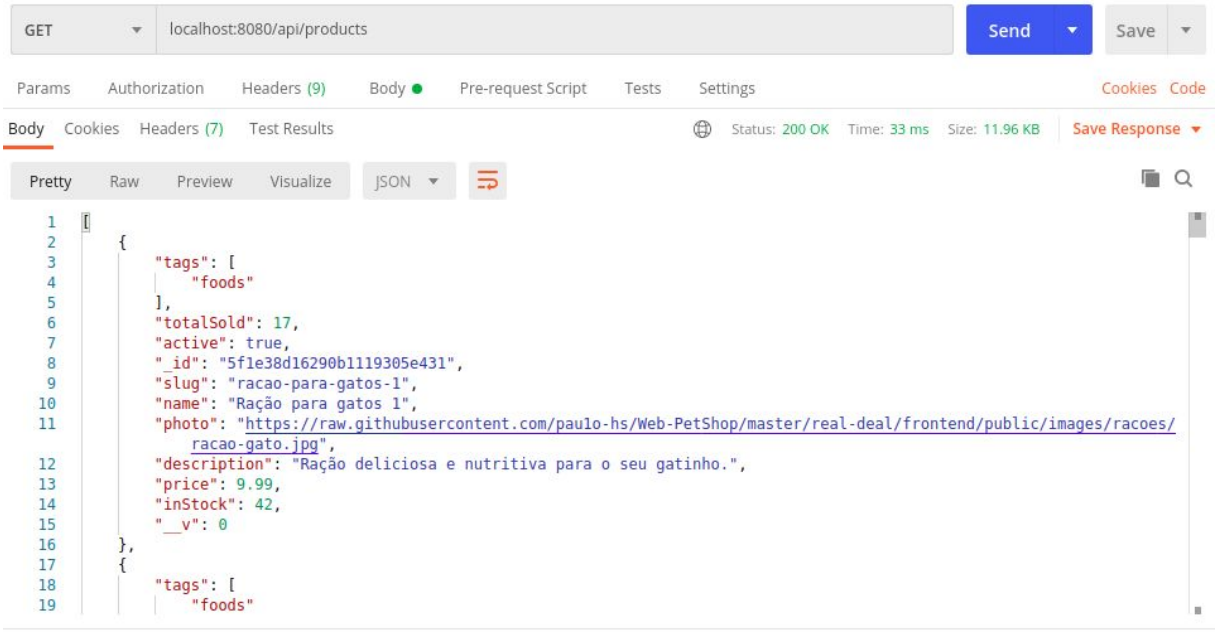


- Criar novo admin (se tornará filho daquele que o criou). Além do status de retorno (201 - CREATED), na segunda imagem é possível ver que o admin que registramos volta na resposta dos filhos do admin root.

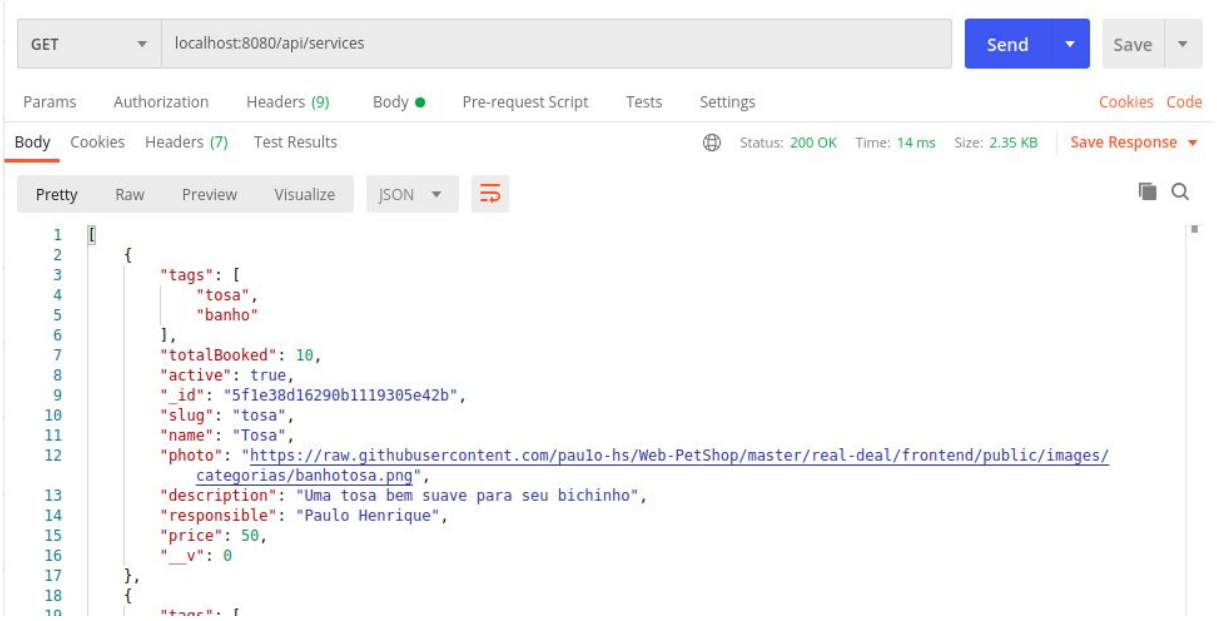




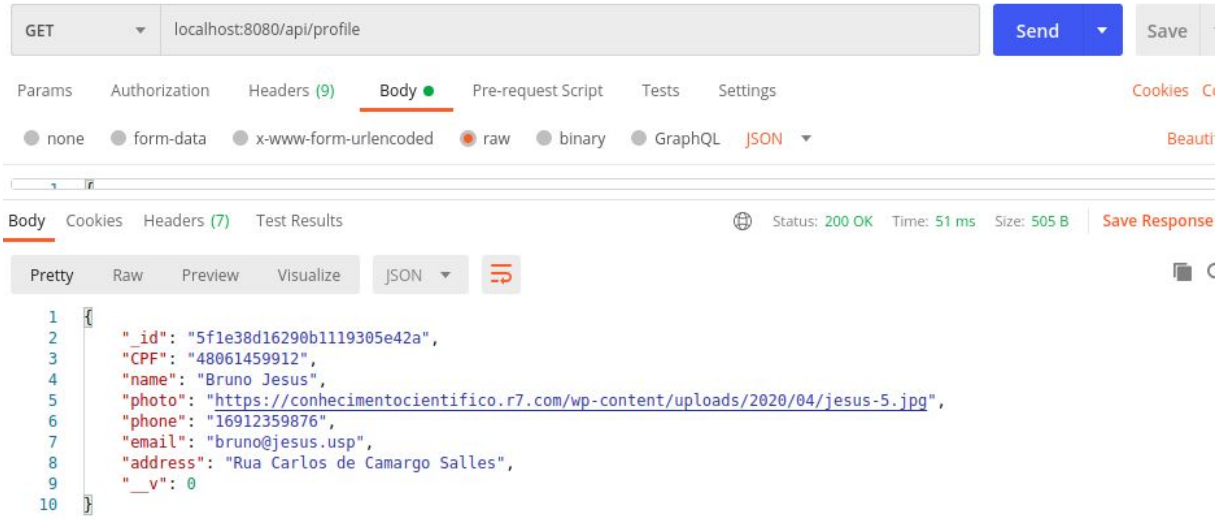
- Pegar todos os produtos (lado do cliente)



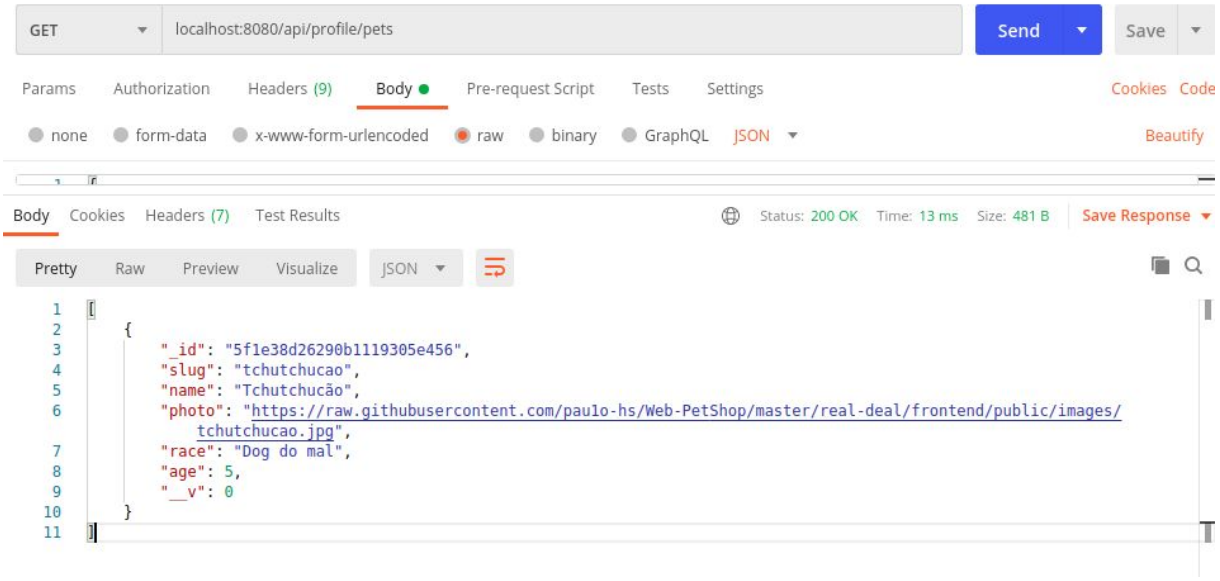
- Pegar todos os serviços (lado do cliente)



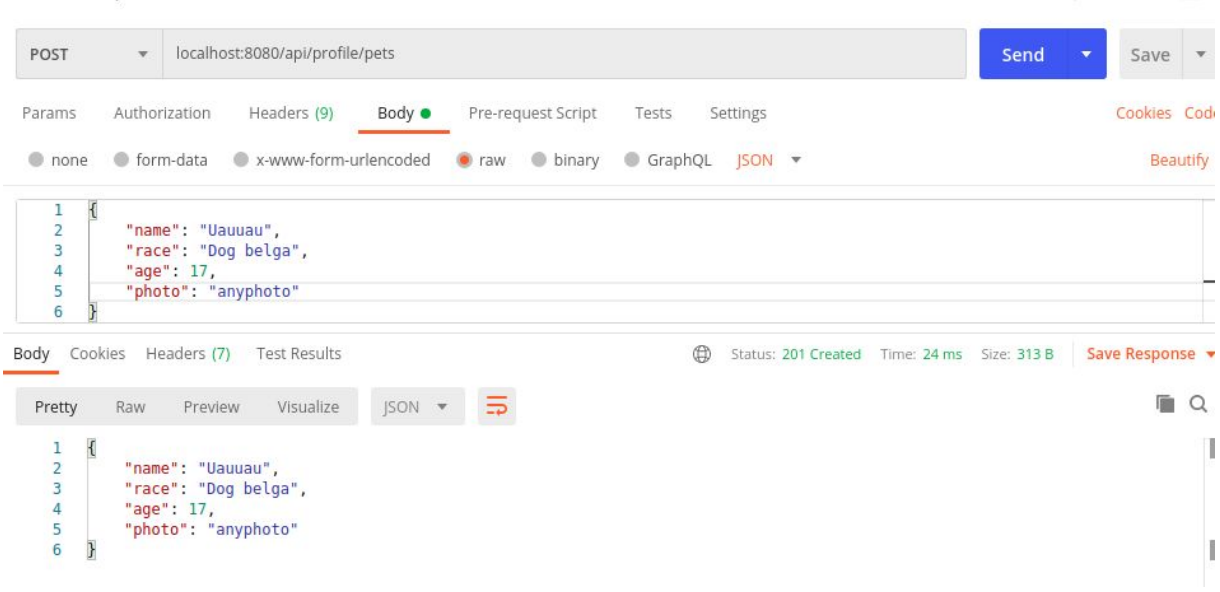
- O cliente poder ver suas informações



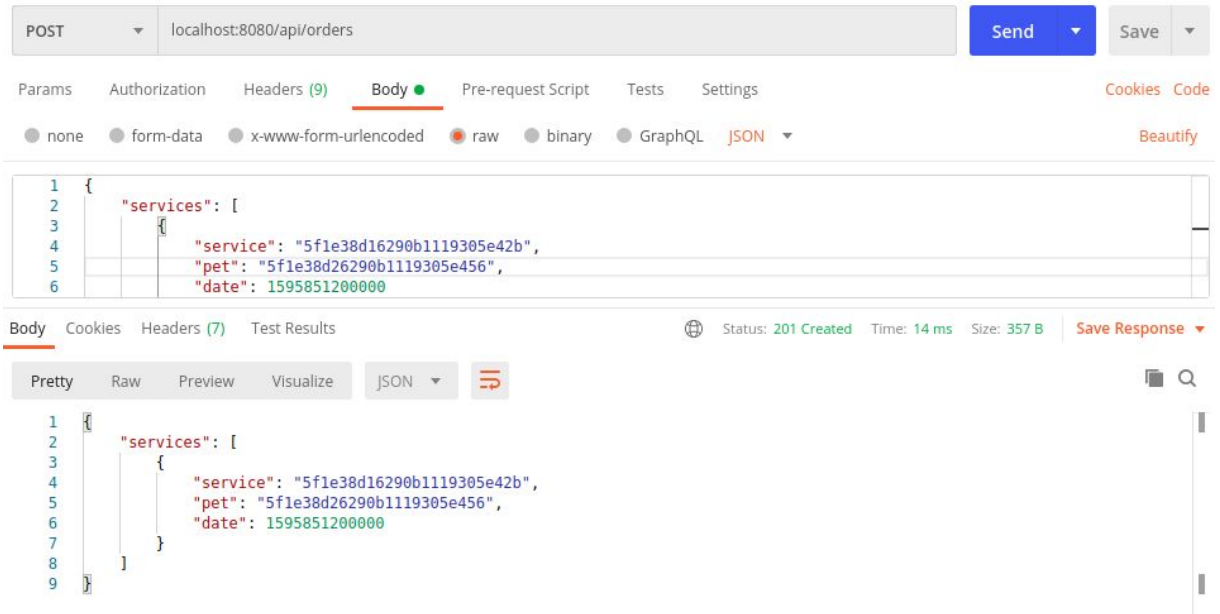
- Pegar todos os pets do cliente



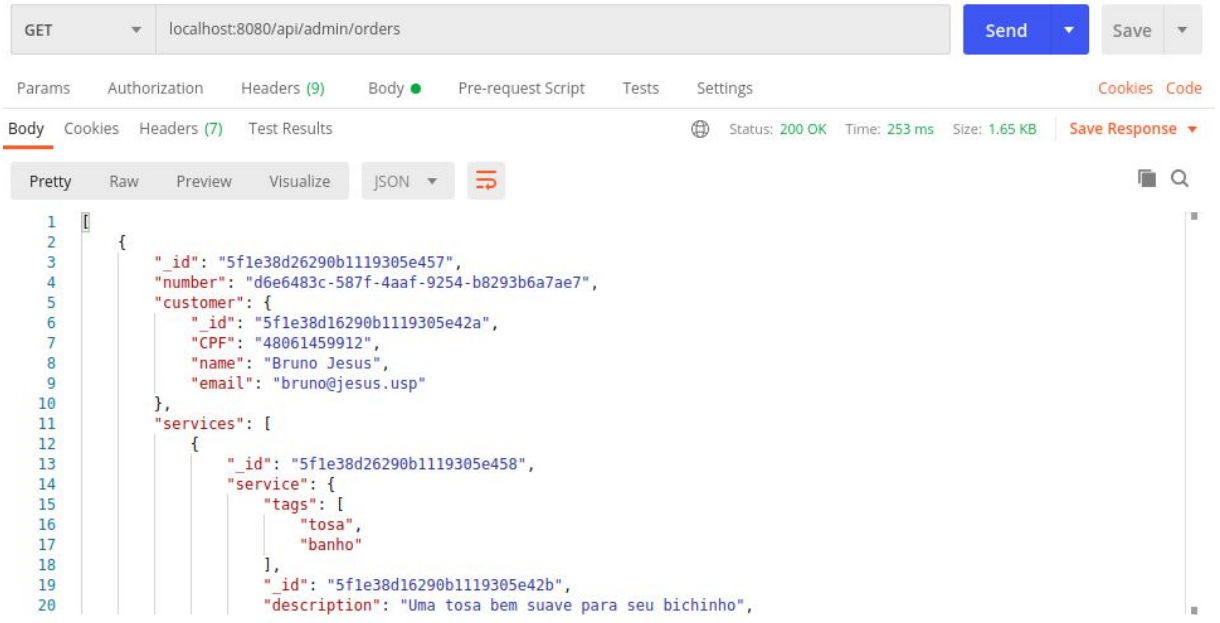
- Cliente criar um pet (após logado)



- Criar um pedido



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



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).