INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO

Gustavo Massamichi Nakamura (SP309863X)

Programação Dinâmica para Web (PDWA5)

Projeto de criação de API

SÃO PAULO

2023

Gustavo Massamichi Nakamura (SP309863X)

Sebo Online S.A.

Primeira Entrega

Documentação do trabalho de Programação Dinâmica para Web (5º semestre) no Instituto Federal de Educação, Ciência e Tecnologia de São Paulo ministrado pelo professor Luiz Fernando Postingel Quirino

SÃO PAULO

2023

Sumário

1 - Introdução

Na aula de programação dinâmica para web ministrada pelo professor Quirino no Instituto Federal de Educação, Ciência e Tecnologia de São Paulo foi requerido o projeto que aqui está sendo documentado. Esse projeto tinha como apresentação de dificuldade a seguinte descrição:

"Joana sempre foi apaixonada por livros. Desde pequena, seu lugar preferido era o SEBO da esquina, onde ela podia encontrar preciosidades literárias e histórias esquecidas pelo tempo. Com o avanço da tecnologia, muitos desses SEBOs fecharam as portas, e Joana viu sua paixão se tornar cada vez mais rara nas cidades. Porém, ao invés de lamentar, ela teve uma ideia: por que não trazer o conceito de SEBO para o mundo online?

Assim nasceu o "Sebo Online S.A.", uma plataforma digital onde qualquer pessoa poderia vender ou comprar livros usados, garantindo que as histórias continuassem sendo compartilhadas e relembradas.

Joana acredita que a essência de um SEBO não está apenas nos livros, mas nas pessoas que os leem e os vendem. Por isso, ela quer que o sistema permita a interação entre os usuários, e que cada livro tenha sua própria "história", contada por quem o está vendendo."

Dado a dificuldade anterior, foi necessário desenvolver uma API robusta e eficiente para o Sebo Online S.A. Para essa primeira entrega eu estabeleci a base do sistema, focando principalmente na gestão de usuários.

2 – Objetivo dessa 1ª entrega

-> Objetivo

- Estabelecer base do sistema, focando principalmente na gestão de usuários

-> Requisitos

- Back-End Base

  - Configuração inicial do servidor.
  - Definição e Conexão com banco de dados.
  - Implementação da estrutura básica de roteamento.

- Usuários

  - Cadastro de usuários (compradores e vendedores).
  - Autenticação (login/logout).
  - Edição de perfil.
  - Soft delete para desativar usuários.
  - Sistema de criptografia para senhas.

- Administradores

  - Login de administradores.
  - Visualização básica de relatórios e estatísticas (focar no retorno de um endpoint ao menos, sugestão, pelo endpoint de listagem de usuários, pode ser aprimorado nas próximas entregas).

3 - Tecnologias

- Front-End:
  - HTML;
  - CSS;
- Back-End:
  - Node.js;
- Banco de dados:
  - MySQL Workbench;
- API:
  - Postman (necessário instalação do postman agent no seu desktop);
  - Thunder Client (instalação feita no VS Code);
- IDE:
  - Visual Studio Code;
- Repositório:
  - github - > https://github.com/gmichin/Sebo_Online_S.A

4 – Provas de objetivos cumpridos e requisições com respostas

Back-End Base (configuração inicial do servidor, definição e conexão com banco de dados, implementação da estrutura básica de roteamento):

Código no MySQL Workbench, no repositório estará dentro da pasta “db”:

```sql
CREATE DATABASE sebo_online;

USE sebo_online;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    senha VARCHAR(255) NOT NULL,
    status ENUM('ativo', 'inativo') NOT NULL,
    tipo ENUM('comprador', 'vendedor', 'administrador') NOT NULL,
    area_especializacao VARCHAR(255),
    token VARCHAR(10) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

CREATE TABLE admin (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    senha VARCHAR(255) NOT NULL,
    status ENUM('ativo', 'inativo') NOT NULL,
    tipo ENUM('comprador', 'vendedor', 'administrador') NOT NULL,
    area_especializacao VARCHAR(255),
    token VARCHAR(10) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    dataInicio TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```sql
DELIMITER //
CREATE TRIGGER generate_token_before_insert
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
    SET NEW.token = SUBSTRING(MD5(RAND()), 1, 10);
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER generate_token_before_insert_admin
BEFORE INSERT ON admin
FOR EACH ROW
BEGIN
    SET NEW.token = SUBSTRING(MD5(RAND()), 1, 10);
END;
//
DELIMITER ;
```

DB.JS (olhar instruções comentadas na imagem abaixo):

```
const mysql = require('mysql2');

const db = mysql.createConnection({
  host: '127.0.0.1', //host da sua conexão no MySQL Workbench
  user: 'gustavo', //Seu user da conexão MySQL Workbench
  password: 'juliemei2014', //sua senha da conexão MySQL Workbench
  database: 'sebo_online', //Sua database do MySQL Workbench
});

db.connect((err) => {
  if (err) {
    console.error('Erro ao conectar ao banco de dados:', err);
    return;
  }
  console.log('Conectado ao banco de dados MySQL');
});

module.exports = db;
```

APP.JS:

```
const express = require('express');
const app = express();
const port = process.env.PORT || 3000;
const path = require('path');

app.use(express.json());

app.use(express.urlencoded({ extended: true }));

app.use(express.static(path.join(__dirname, 'public')));

app.use('/path', require('./routes/path'));

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

app.listen(port, () => {
  console.log(`\nServidor rodando: http://localhost:${port}/}`);
});
```

Nota: Para rodar o programa, é necessário escrever no terminal “node app.js”;

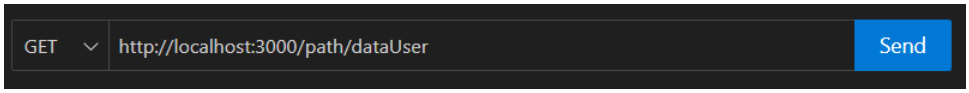
SOBRE TODAS AS REQUISIÇÕES, ELAS ESTÃO CONFIGURADAS EM "ROUTES/PATH.JS" DO ARQUIVO DO GITHUB

Visualização básica de relatórios e estatísticas (focar no retorno de um endpoint ao menos, sugestão, pelo endpoint de listagem de usuários, pode ser aprimorado nas próximas entregas):

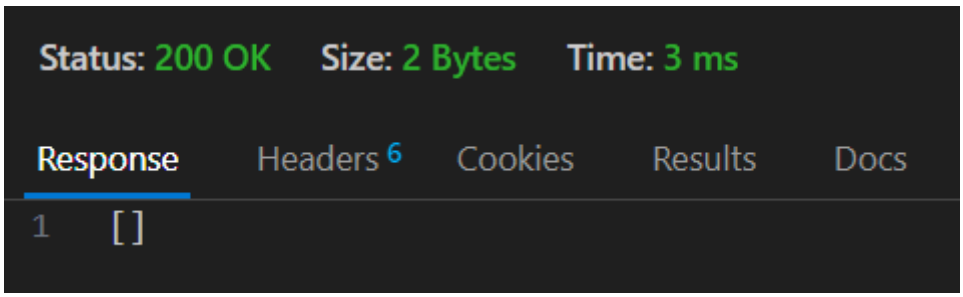
GET – retornar usuários

Link: http://localhost:3000/path/dataUser

THUNDER CLIENT

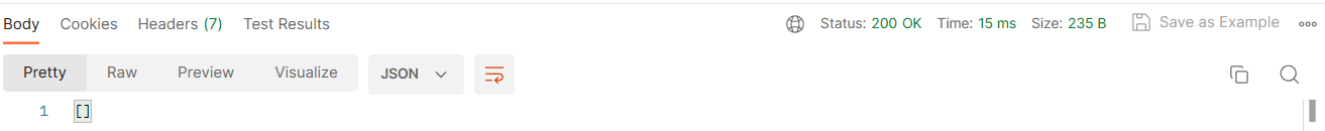


Resposta:



POSTMAN

Resposta:

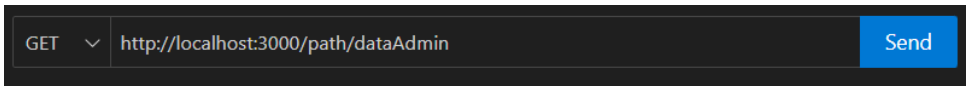


Nota: Essas respostas são de antes de ser feito qualquer cadastro;

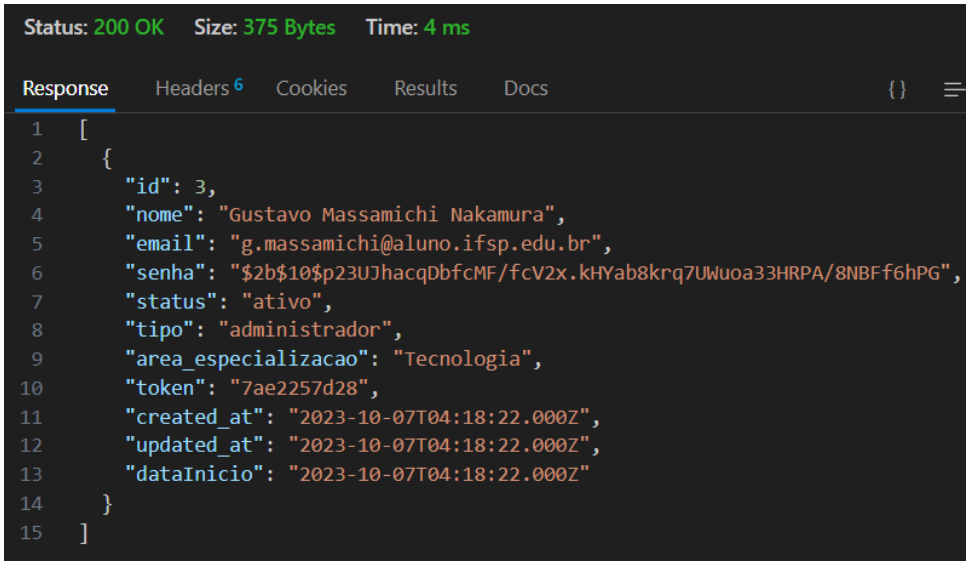
GET – retornar administradores

Link: http://localhost:3000/path/dataAdmin

THUNDER CLIENT

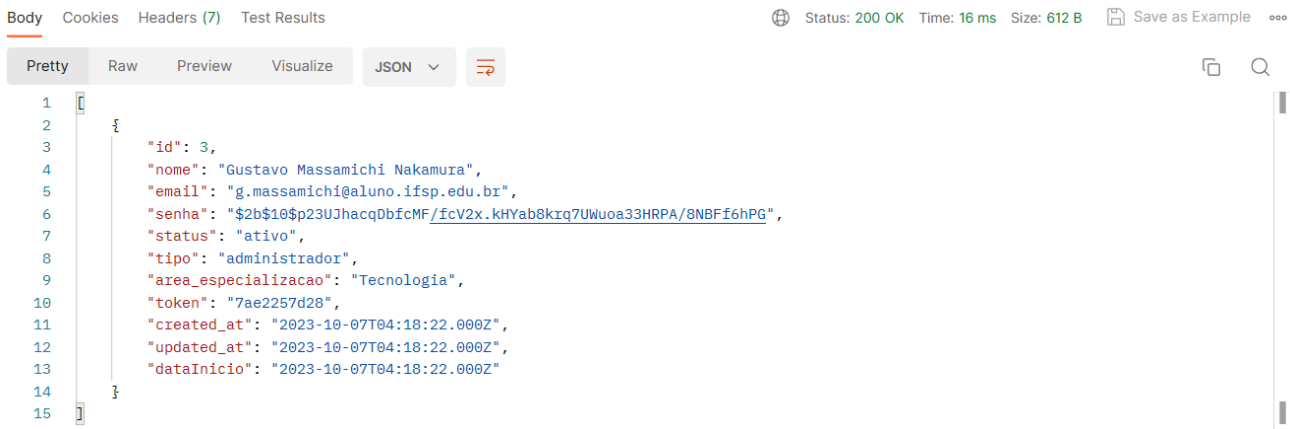


Resposta:



POSTMAN

Resposta:



Nota: Essas respostas são de antes de ser feito qualquer cadastro;

Cadastro de usuários (compradores e vendedores):

POST - cadastro de usuários e administradores:

Link: http://localhost:3000/path/signup

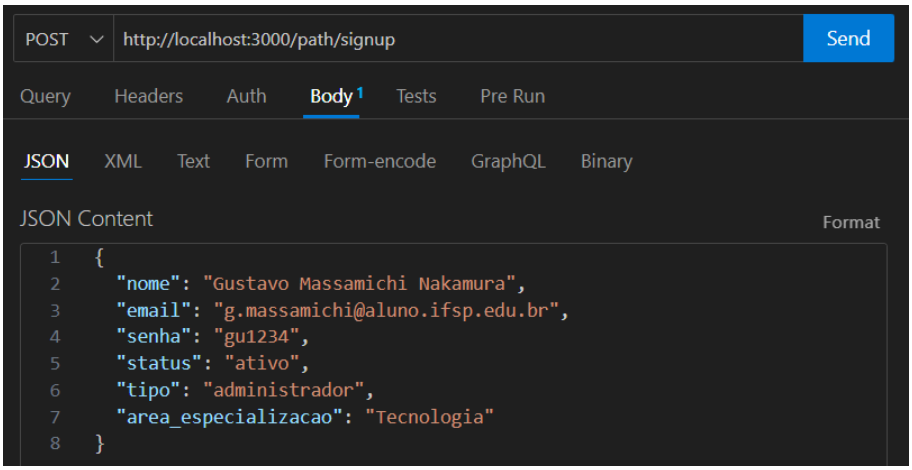
Body (json):

```
{
  "nome": "Inserir seu nome",
  "email": "Inserir seu email",
  "senha": "Inserir sua senha",
  "status": "Escolha entre ativo e inativo",
  "tipo": "Escolha entre comprador, vendedor e administrador",
  "area_especializacao": "Insira sua área de especialização"
}
```

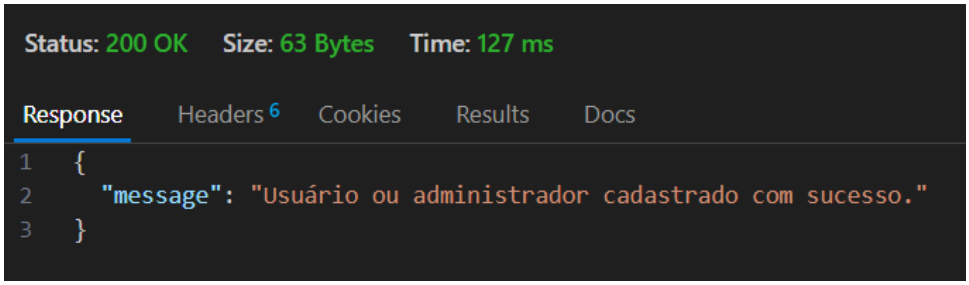
Nota 1: Ao escolher o "tipo" no json acima ele irá cadastrar automaticamente na tabela (table do SQL) de "admin" ou de "users" dependendo se o tipo for vendedor, comprador ou administrador;

Nota 2: Nessa mesma requisição podemos cadastrar tanto usuários quanto administradores;

THUNDER CLIENT



Resposta:



GET após um adm cadastrado - não é o mesmo adm da imagem de requisição POST cadastro anterior

Status: 200 OK Size: 725 Bytes Time: 5 ms

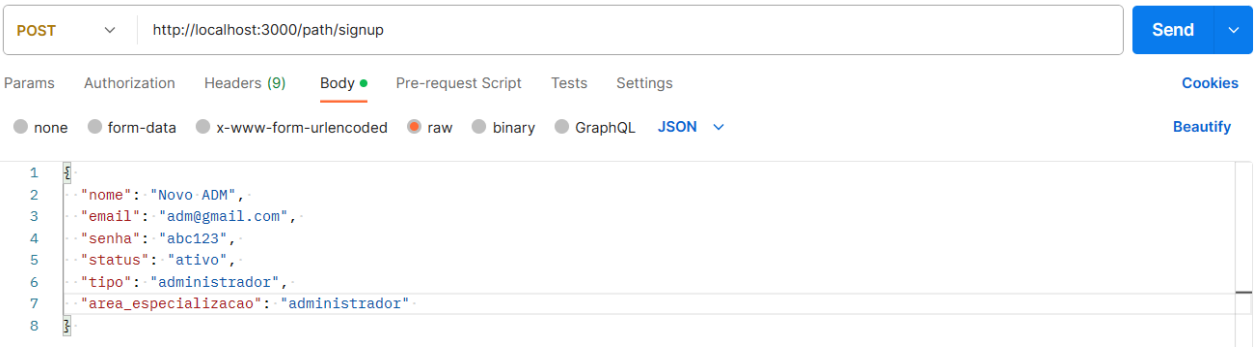
Response Headers 6 Cookies Results Docs { }


```
      "dataInicio": "2023-10-07T04:18:22.000Z"
    },
    {
      "id": 4,
      "nome": "Usuário Adm",
      "email": "usuárioAdm@gmail.com",
      "senha": "$2b$10$.tQ8OVvmypMx/AY/S9/gSuy5Yk7dCVWi0Df6NA7x4MnhC5/zT6yK.",
      "status": "ativo",
      "tipo": "administrador",
      "area_especializacao": "Tecnologia",
      "token": "2b99da0dd1",
      "created_at": "2023-10-07T05:10:22.000Z",
      "updated_at": "2023-10-07T05:10:22.000Z",
      "dataInicio": "2023-10-07T05:10:22.000Z"
    }
  ]
```

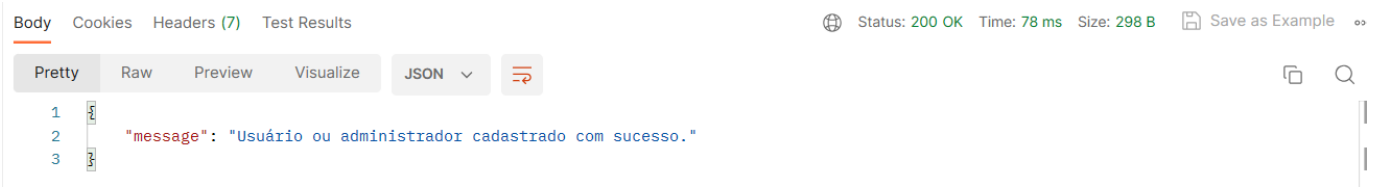

GET após um usuário cadastrado - não é o mesmo usuário da imagem de requisição POST cadastro anterior


```
[
  {
    "id": 6,
    "nome": "Usuário Comum",
    "email": "usuário@gmail.com",
    "senha": "$2b$10$bDsS.Y9rCcwRgby7e61YB.MuT1E20rVJzSvY3n051kOBADQLsSXSm",
    "status": "ativo",
    "tipo": "comprador",
    "area_especializacao": "Tecnologia",
    "token": "7813a45506",
    "created_at": "2023-10-07T05:02:10.000Z",
    "updated_at": "2023-10-07T05:02:10.000Z"
  }
]
```

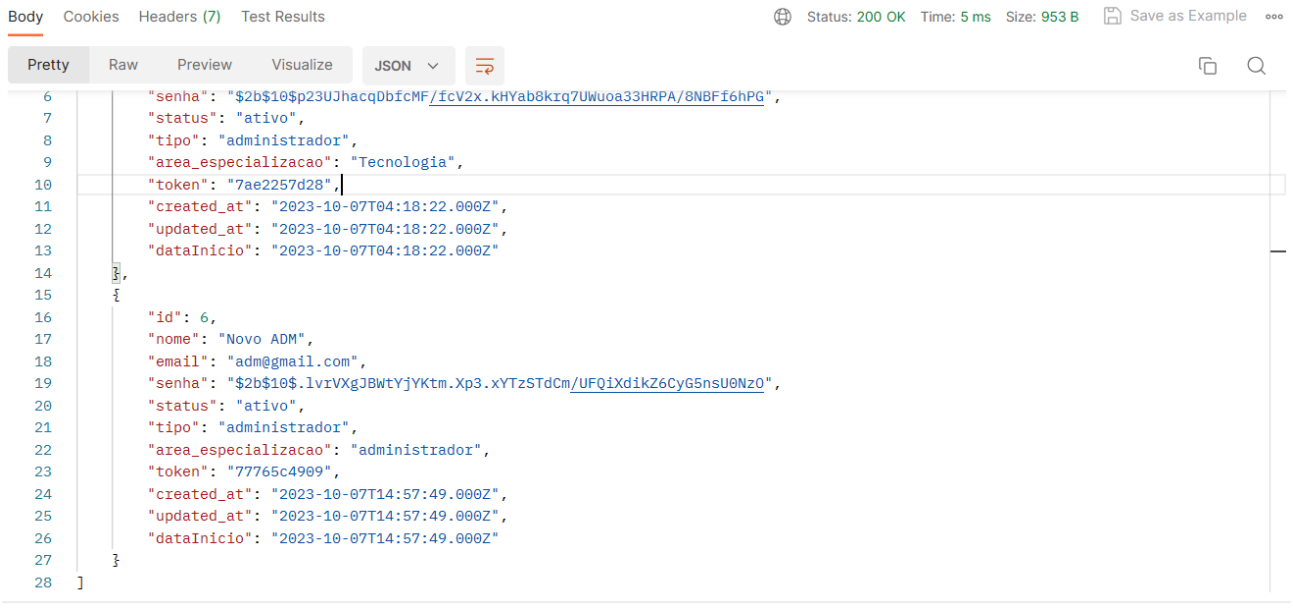

POSTMAN



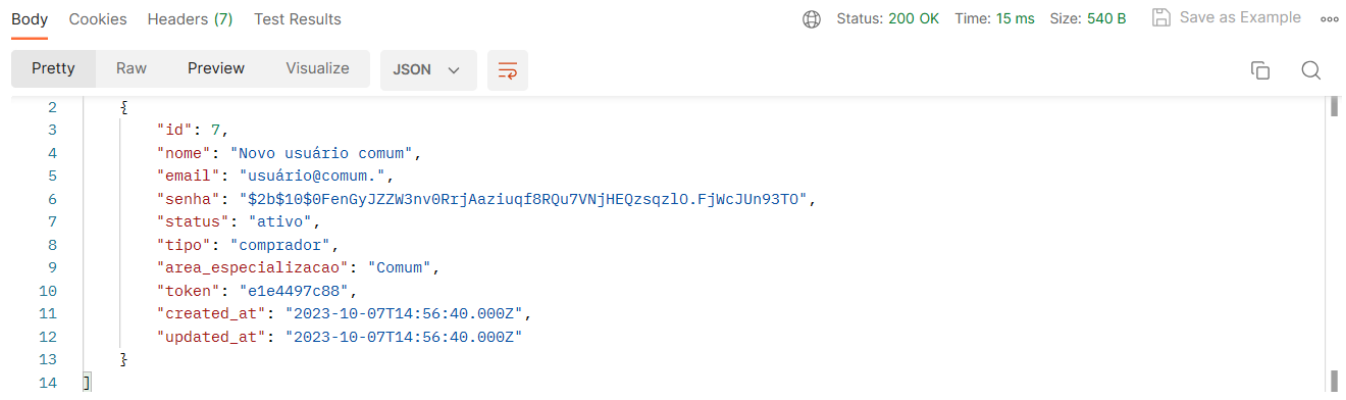
Resposta:



GET após um adm cadastrado



GET após um usuário cadastrado - não é o mesmo usuário da imagem de requisição POST cadastro anterior



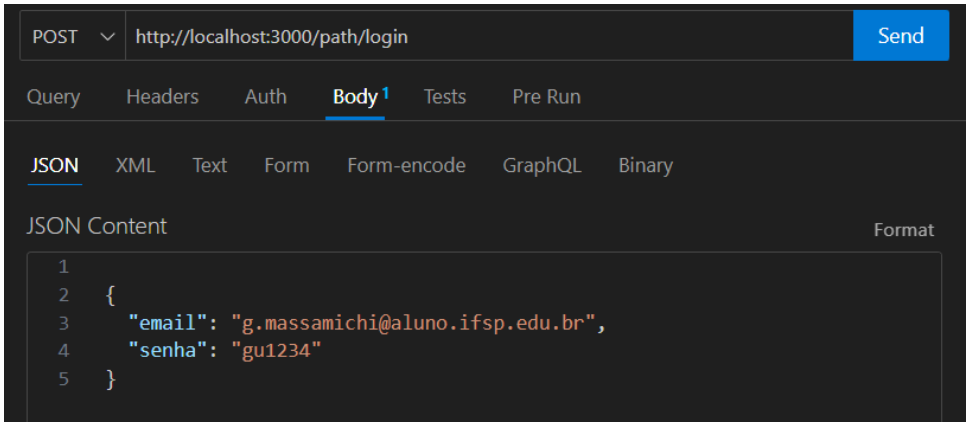
Autenticação (login/logout) de usuários e de administradores:

POST - login de usuários e administradores
Link: http://localhost:3000/path/login
Body (json):

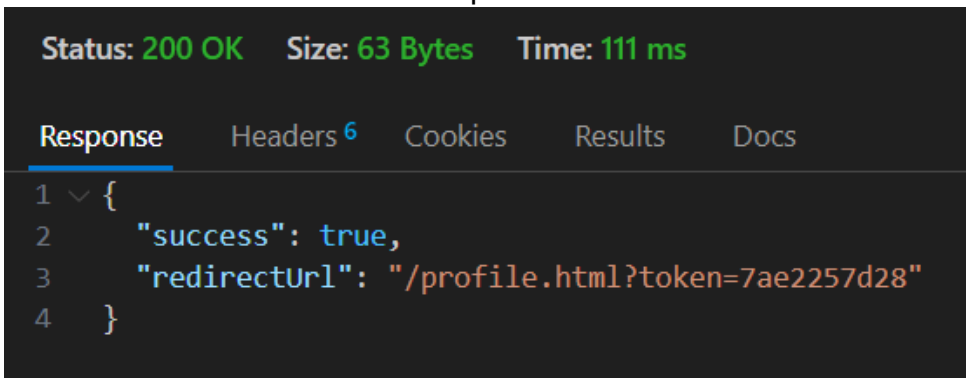
```
{
  "email": "Inserir seu email",
  "senha": "Inserir sua senha"
}
```

Nota: O login de usuários e de adms são feitos nessa mesma requisição;

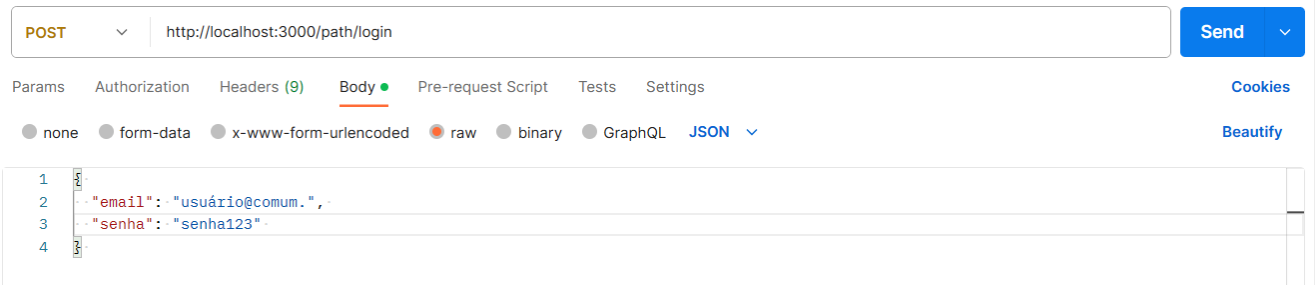
THUNDER CLIENT



Resposta:



POSTMAN



Resposta:

Nota: Como visto nos exemplos de GET anteriores, as senhas estão criptografadas, mas eu posso inserir a senha que escolhi (não criptografada);

Edição de perfil:

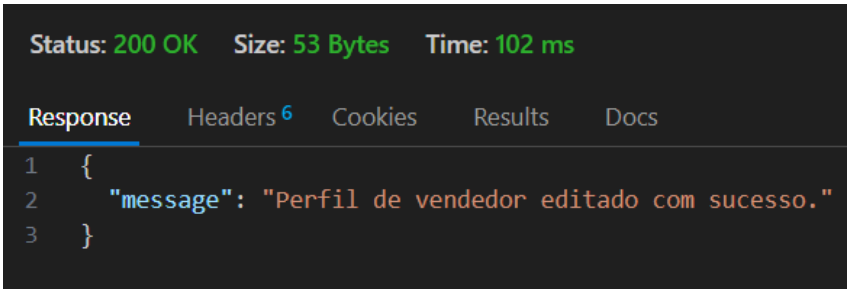
PUT - Edição de usuário comum
Link: http://localhost:3000/path/users/(token_usuário)
Body (json):

```
{
  "nome": "Exemplo de edição de perfil de Usuário",
  "email": "com metodo put",
  "status": "ativo",
  "tipo": "vendedor",
  "area_especializacao": "teste",
  "senha": "senha"
}
```

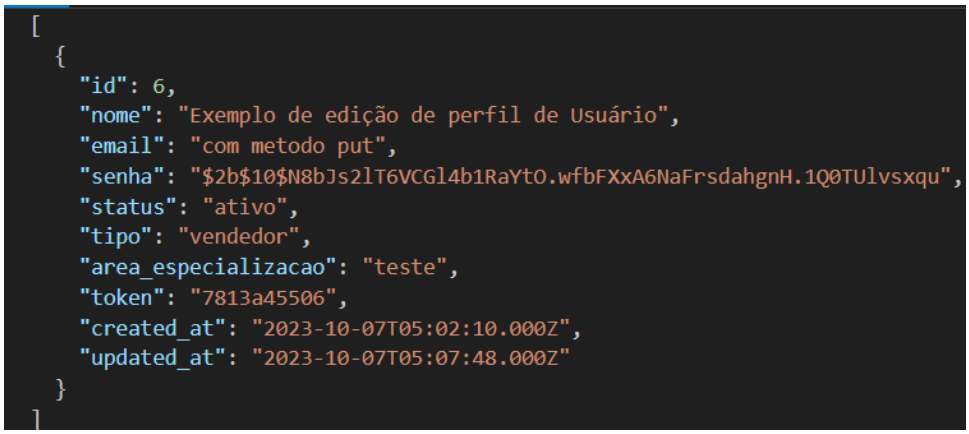
Nota: O token pode ser adquirido com a requisição GET mostrada anteriormente;

THUNDER CLIENT

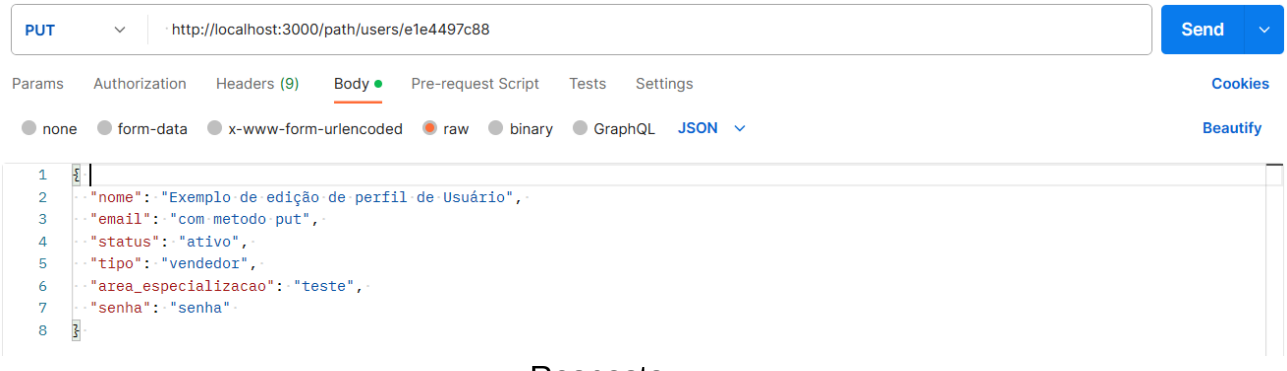
Resposta:



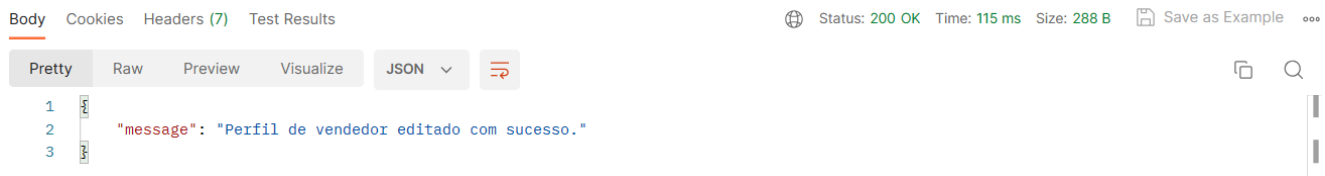
GET do usuário após edição:



POSTMAN



Resposta:



GET do usuário após edição:

Nota: O token pode ser adquirido com a requisição GET mostrada anteriormente;

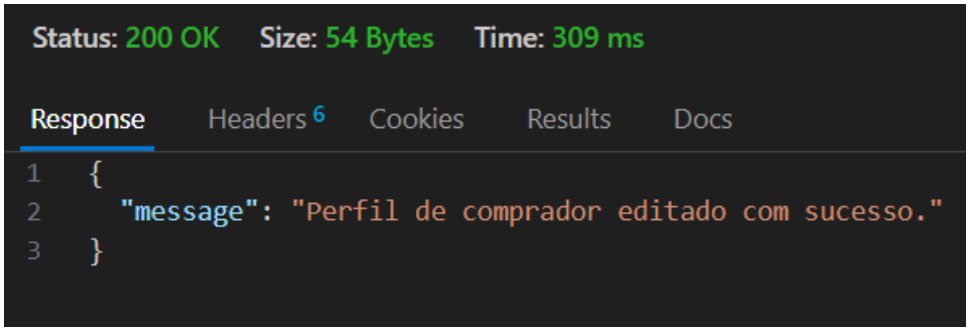
PUT - Edição de administrador para comprador
Link: http://localhost:3000/path/admin/(token_adm)
Body (json):

```
{
  "nome": "Exemplo de edição de perfil de Administrador",
  "email": "com metodo put",
  "status": "ativo",
  "tipo": "comprador",
  "area_especializacao": "teste",
  "senha": "senha"
}
```

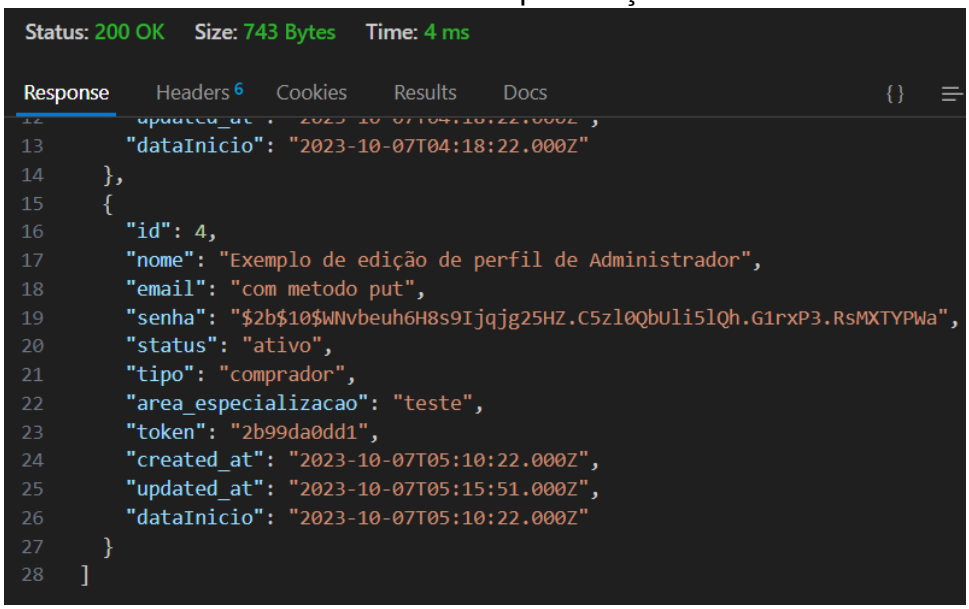
Nota: O token pode ser adquirido com a requisição GET mostrada anteriormente;

THUNDER CLIENT

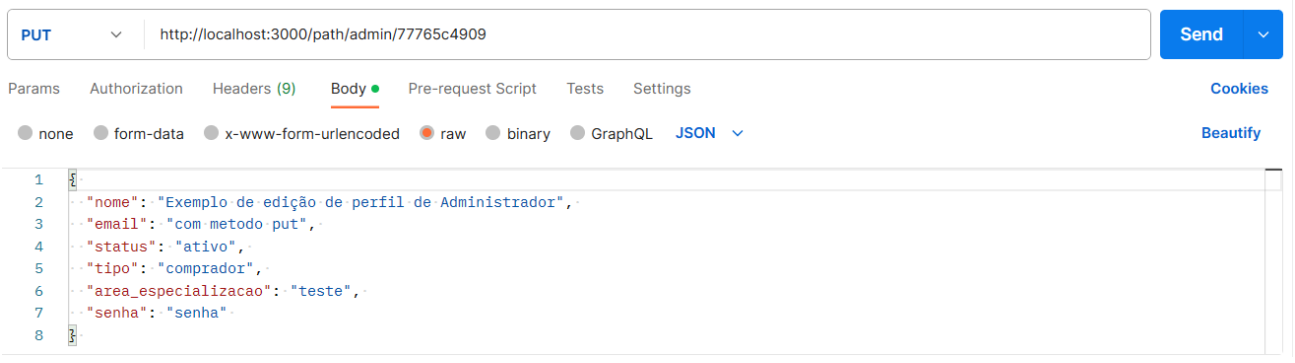
Resposta:



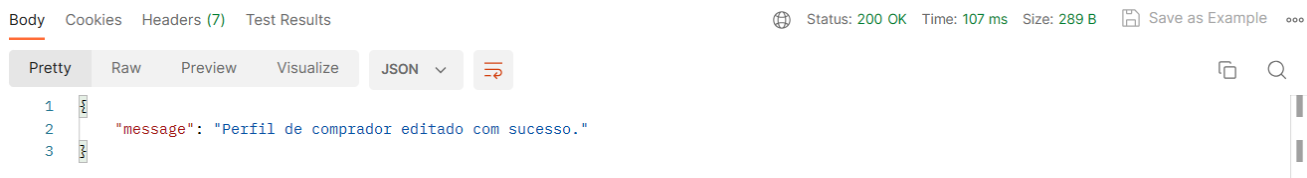
GET do admin após edição:



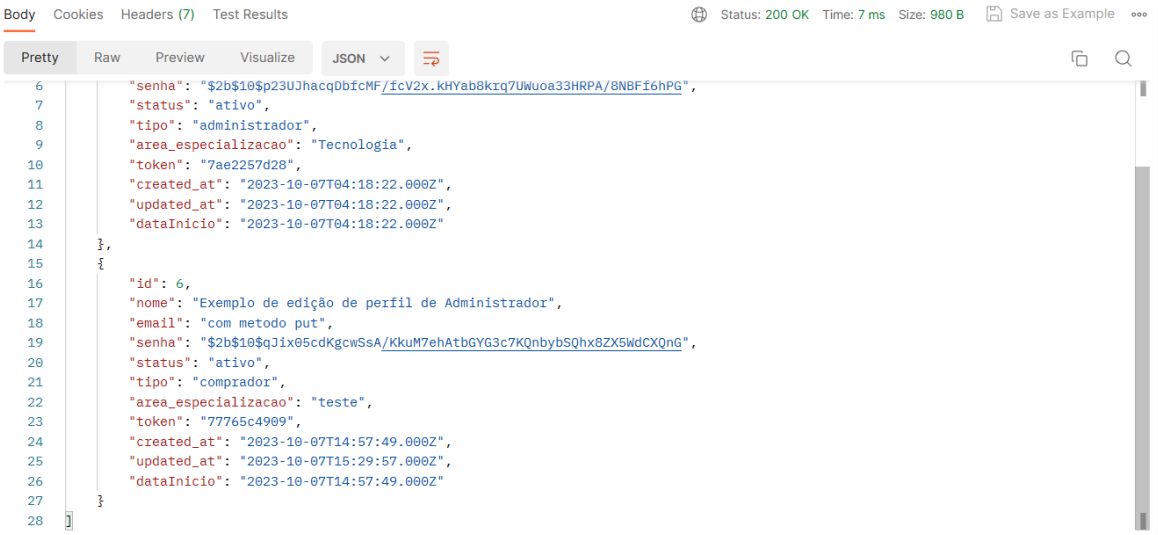
POSTMAN



Resposta:



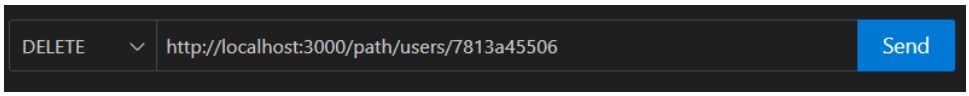
GET do admin após edição:



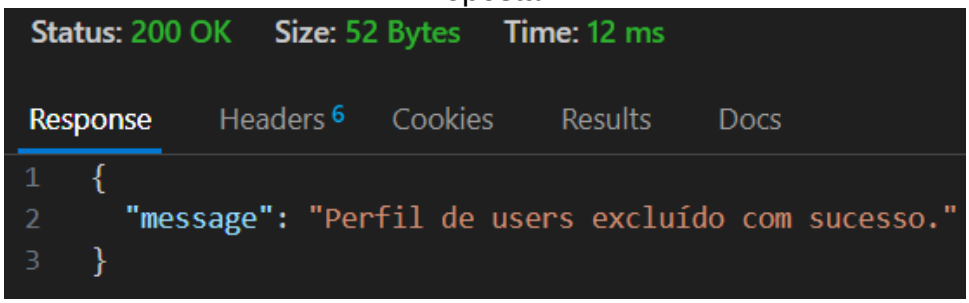
Soft delete para desativar usuários:

DELETE - Deletar usuários
Link: http://localhost:3000/path/users/(token_usuário)
Nota: O token pode ser adquirido com a requisição GET mostrada anteriormente;

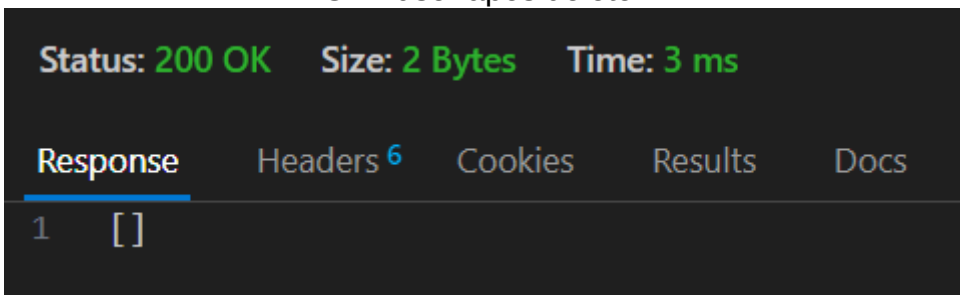
THUNDER CLIENT



Reposta:



GET user após delete:



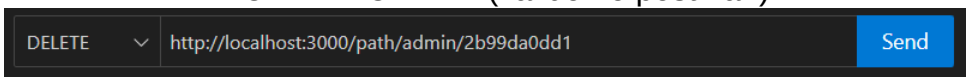
POSTMAN

Resposta:

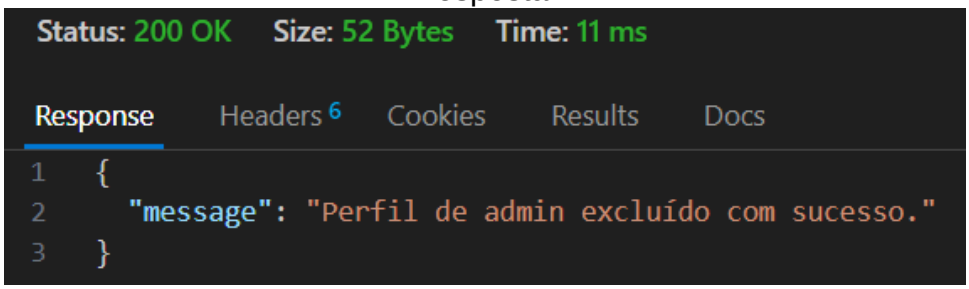
GET user após delete:

DELETE – Deletar administradores
Link: http://localhost:3000/path/admin/(token_admin)
Nota: O token pode ser adquirido com a requisição GET mostrada anteriormente;

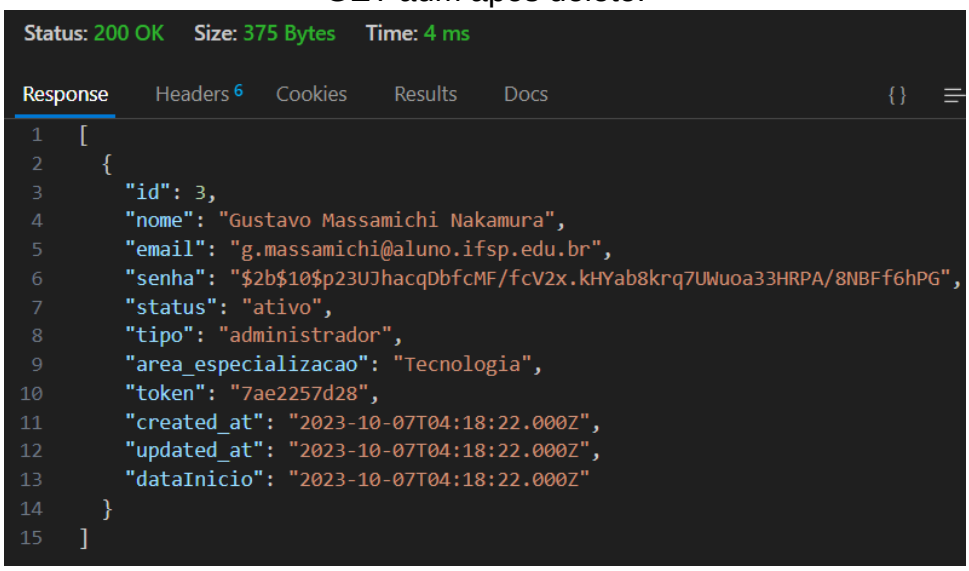
THUNDER CLIENT (válido no postman)



Resposta:

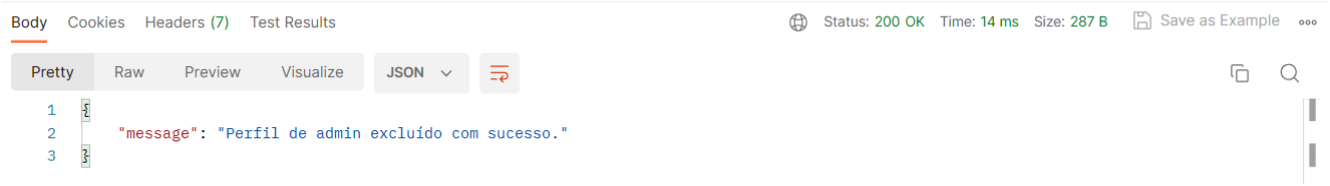


GET adm após delete:

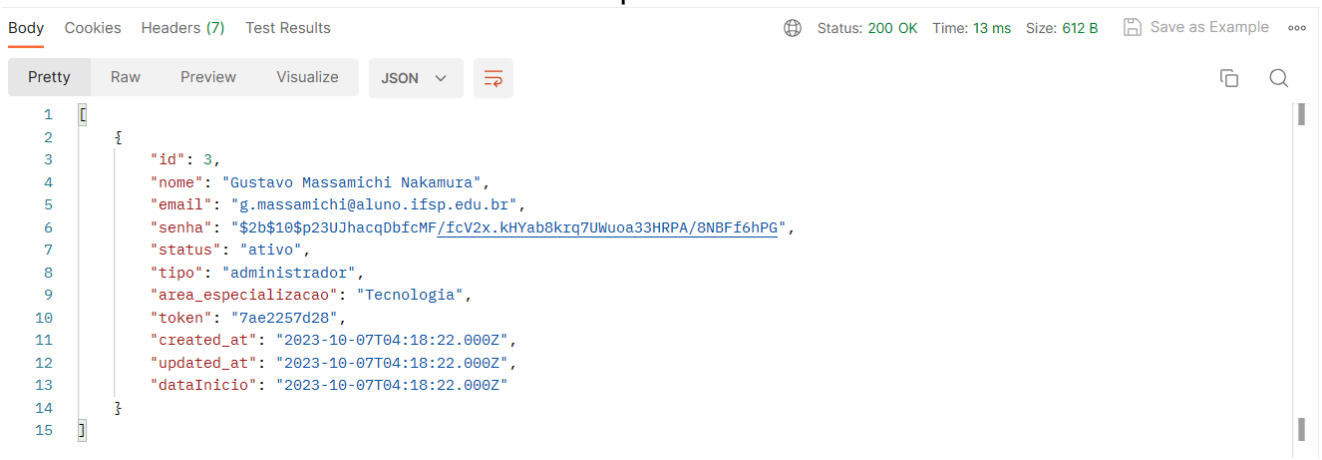


POSTMAN

Resposta:



GET admin após delete:

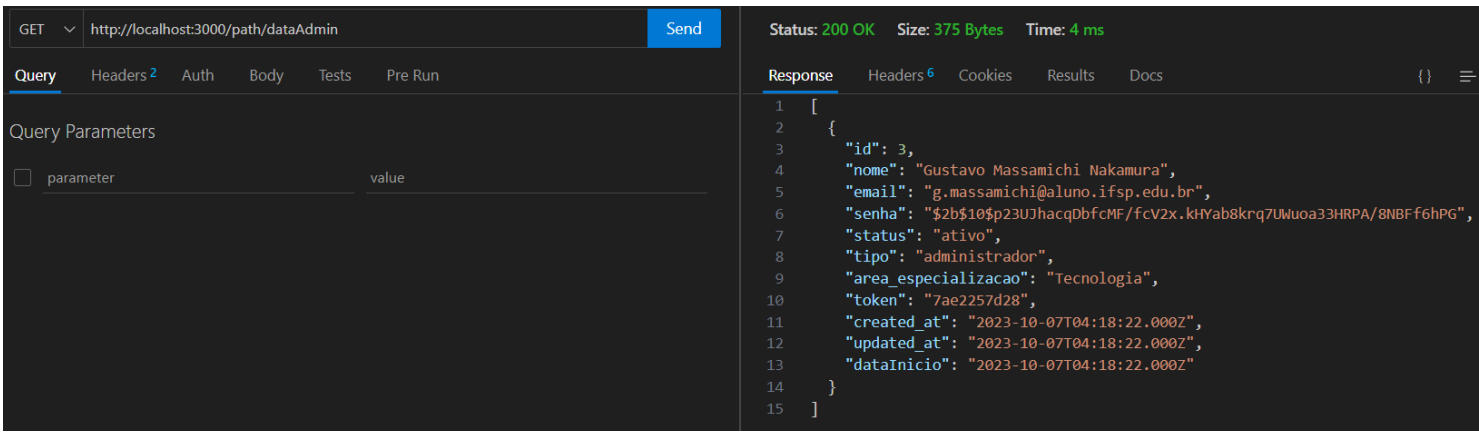


Sistema de criptografia para senhas:

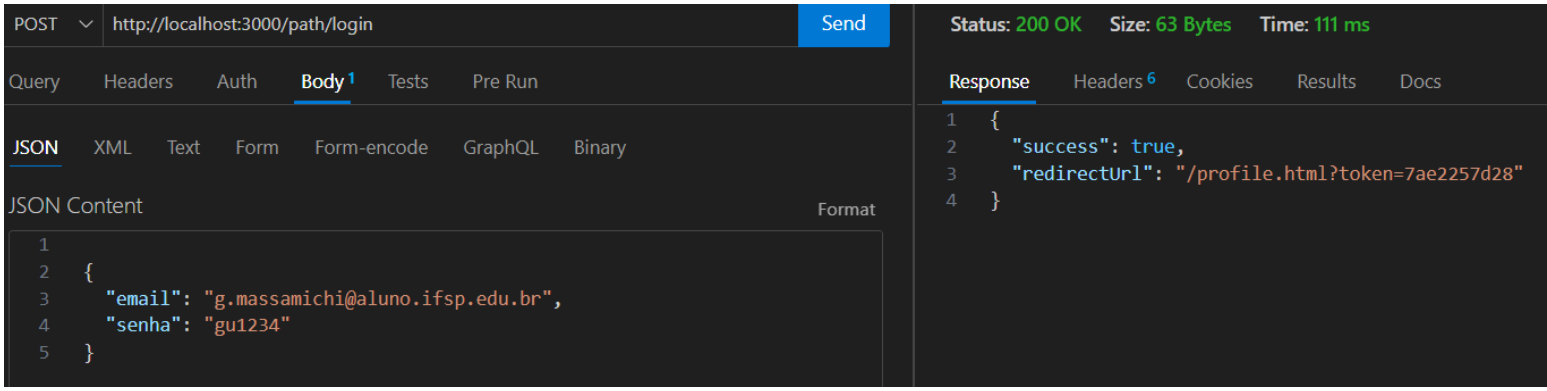
No meu repositório do github, no caminho routes/path.js você poderá ver a seguinte comprovação de criptografia

```
const bcrypt = require('bcrypt');
```

Observe que a senha está criptografada:

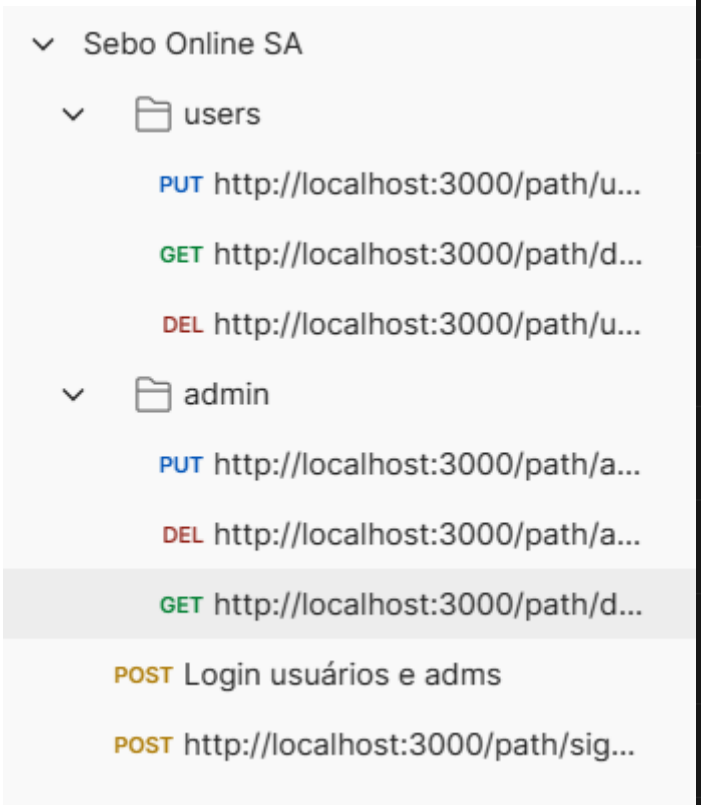


Mesmo assim, posso colocar a senha normalmente como escolhi inicialmente:



BREVE RESUMO DAS RESQUISIÇÕES

POSTMAN



Thunder Client