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 de entrega

-> Objetivo

- Estabelecer base do sistema, focando principalmente na gestão de usuários.
- Permitir que os usuários (especialmente vendedores) comecem a adicionar, editar e gerenciar itens na plataforma.
- Implementar a funcionalidade de transações e otimizar a experiência do usuário com melhorias baseadas nos feedbacks das fases anteriores.

-> 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).

- Itens

- Adição de novos itens.
- Listagem de itens.
- Edição de itens.
- Busca básica de itens (pode ser otimizada na próxima entrega).
- Listagem dos itens cadastrados, com filtros: titulo, isbn, autor.

- Categorias

- Criação de categorias.
- Listagem de categorias.
- Edição de categorias.
- Soft delete para categorias.
- Registro de novas transações.
- Visualização de transações para um usuário específico.

- Entrega da documentação da API

3 - Tecnologias

- Front-End:
    - HTML;
    - CSS;
- Back-End:
    - Node.js (18.16.0);

- 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 (1.84.1);
- 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”:

```
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
);
```

```sql
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 ;
```

```sql
CREATE TABLE items (
    id INT AUTO_INCREMENT PRIMARY KEY,
    titulo VARCHAR(255) NOT NULL,
    autor VARCHAR(255),
    categoria ENUM('livro', 'revista', 'periodico', 'jornal') NOT NULL,
    preco FLOAT NOT NULL,
    descricao TEXT,
    status ENUM('ativo', 'inativo', 'estoque', 'fora_de_estoque') NOT NULL,
    periodicidade VARCHAR(50),
    id_vendedor INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    data_edicao TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (id_vendedor) REFERENCES users(id)
);
```

```sql
CREATE TABLE transactions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    id_comprador INT NOT NULL,
    id_vendedor INT NOT NULL,
    id_item INT NOT NULL,
    data_transacao TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    valor FLOAT NOT NULL,
    FOREIGN KEY (id_comprador) REFERENCES users(id),
    FOREIGN KEY (id_vendedor) REFERENCES users(id),
    FOREIGN KEY (id_item) REFERENCES items(id)
);

CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(255) NOT NULL,
    descricao TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

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

```js
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: '[redacted]', //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

Cadastro de usuários (compradores e vendedores):

POST - cadastro de usuários:

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

Body (json):

{

"nome": "Inserir seu nome",

"email": "Inserir seu email",

"senha": "Inserir sua senha",

"status": "Escolha entre ativo e inativo",

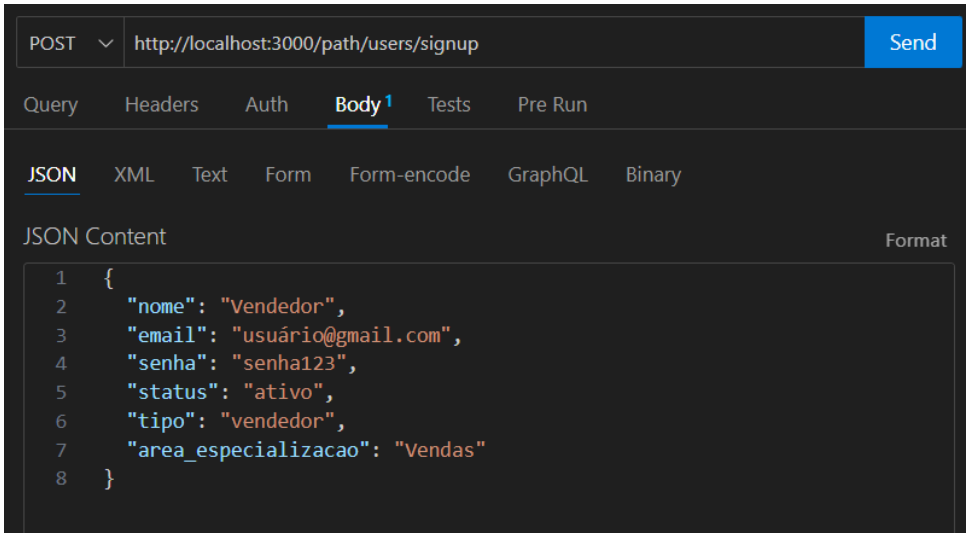
"tipo": "Escolha entre comprador, vendedor e administrador",

"area_especializacao": "Insira sua área de especialização"

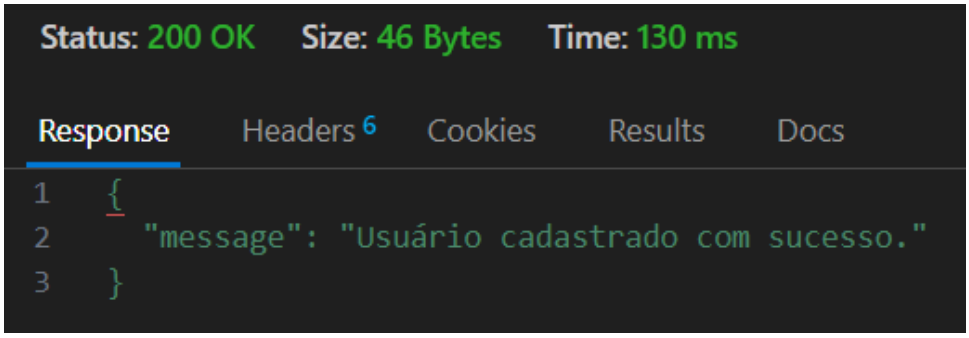
}

Nota: 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.

Requisição:



Resposta:



POST - cadastro de administradores

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

Body (json):

{

"nome": "Inserir seu nome",

"email": "Inserir seu email",

"senha": "Inserir sua senha",

"status": "Escolha entre ativo e inativo",

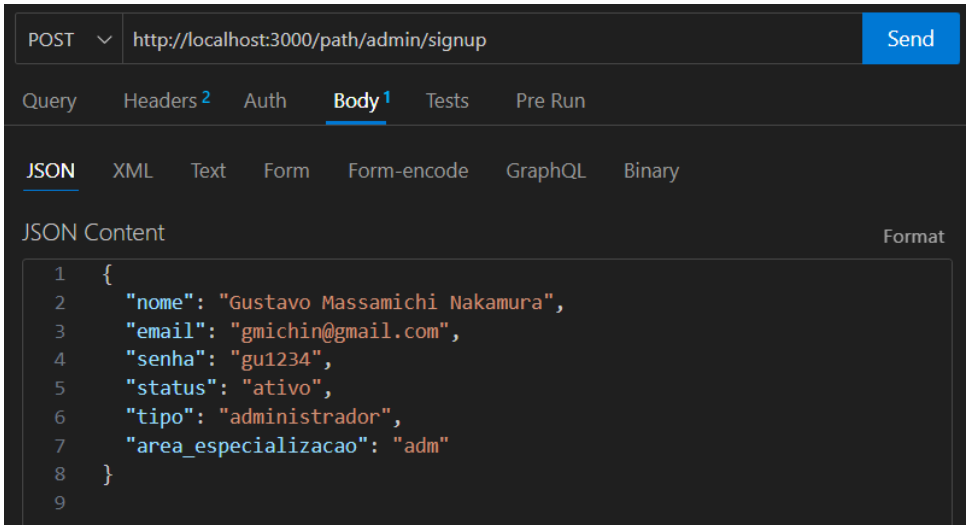
"tipo": "Escolha entre comprador, vendedor e administrador",

"area_especializacao": "Insira sua área de especialização"

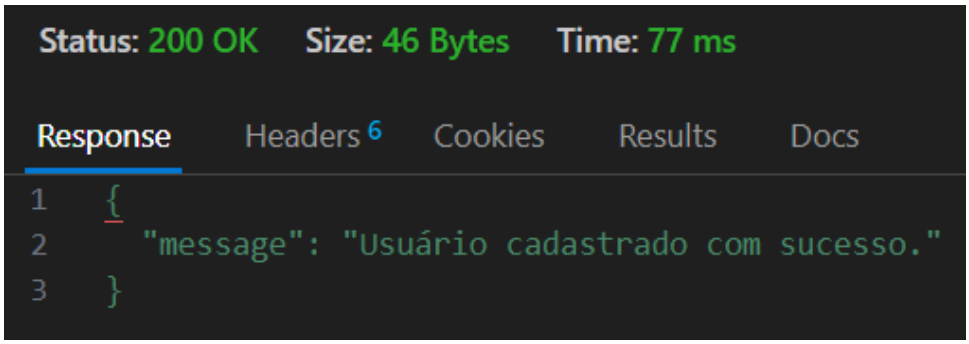
}

Nota: 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.

Requisição:



Resposta:



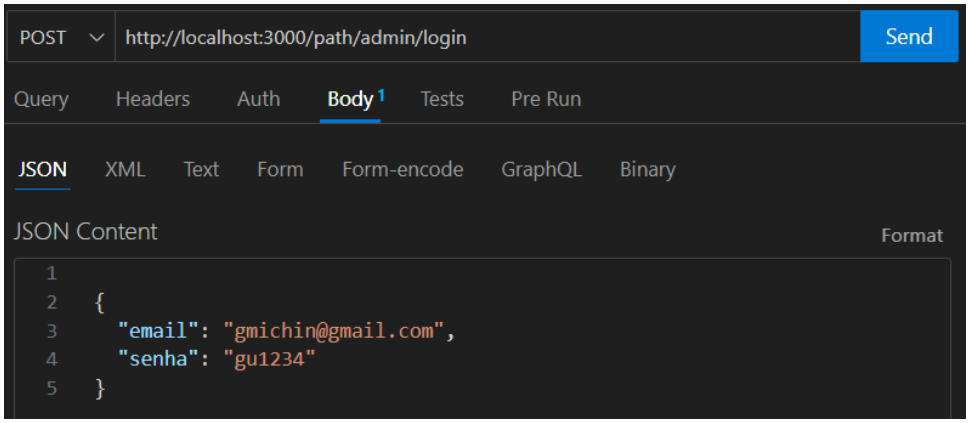
Autenticação (login/logout) de usuários e de administradores:

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

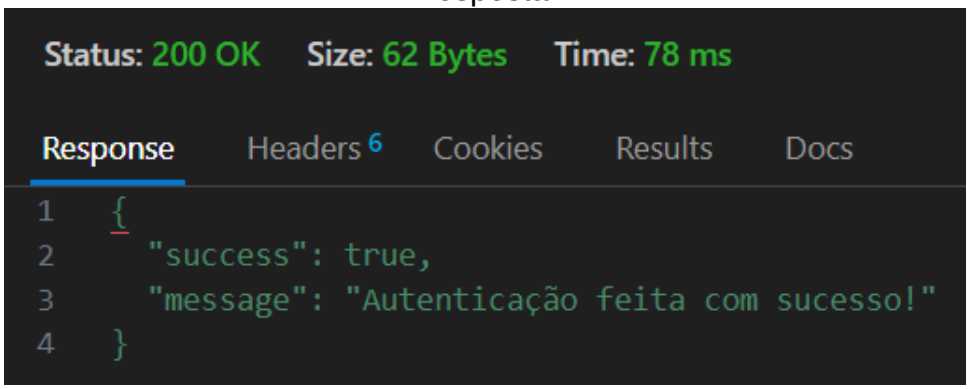
```
{
  "email": "Inserir seu email",
  "senha": "Inserir sua senha"
}
```

Nota: Será necessário o login de adm para o login e retorno de dados de usuários

Requisição:

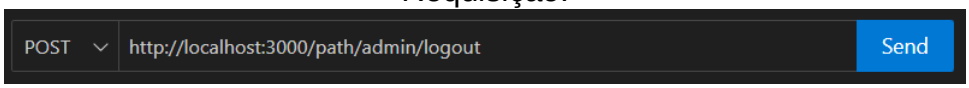


Resposta:

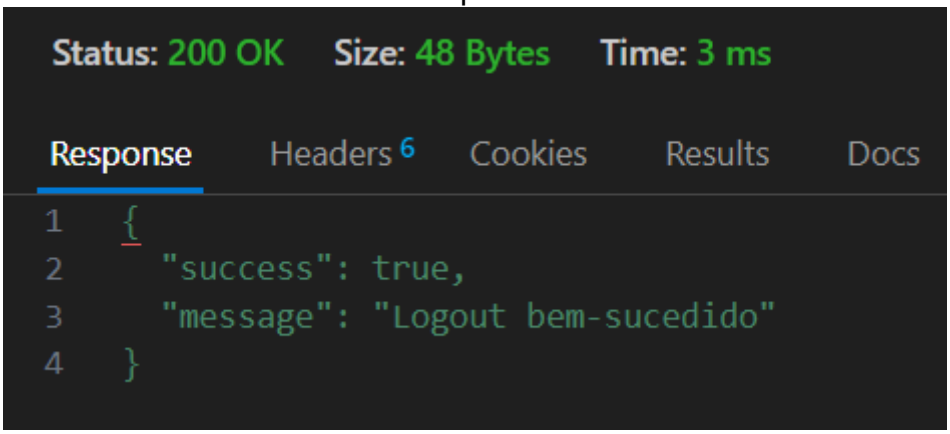


POST – logout de administradores
Link: http://localhost:3000/path/admin/logout

Requisição:



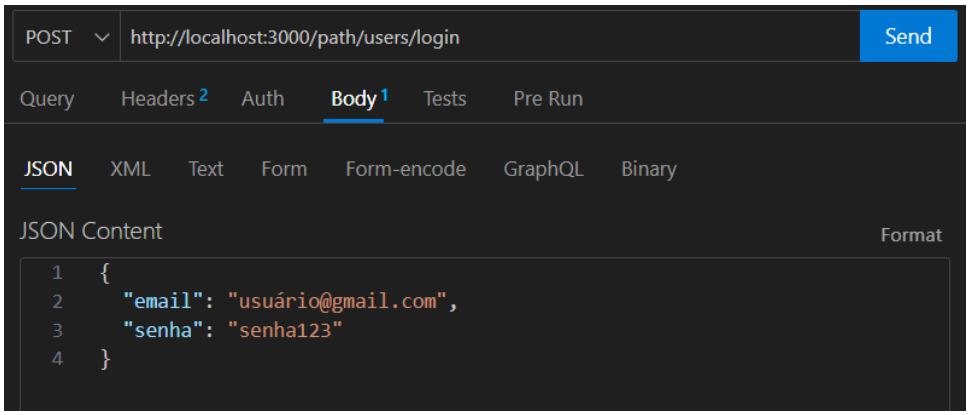
Resposta:



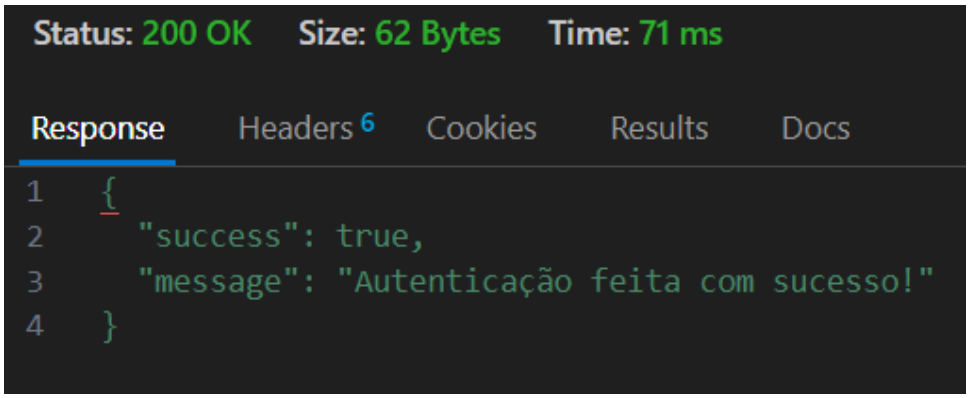
POST - login de usuários
Link: http://localhost:3000/path/users/login
Body (json):

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

Requisição:



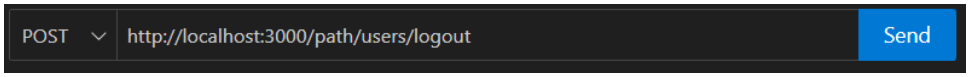
Resposta:



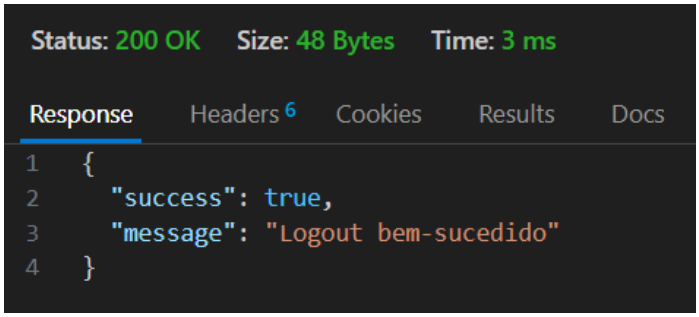
Nota: Só será possível fazer o login de usuário se estiver logado como adm

POST – logout de usuários
Link: http://localhost:3000/path/users/logout

Requisição:



Resposta:

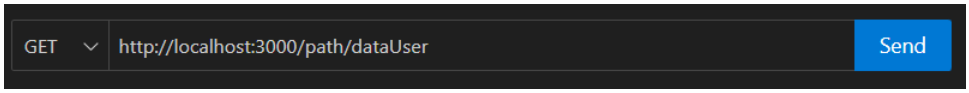


Visualização básica de relatórios e estatísticas):

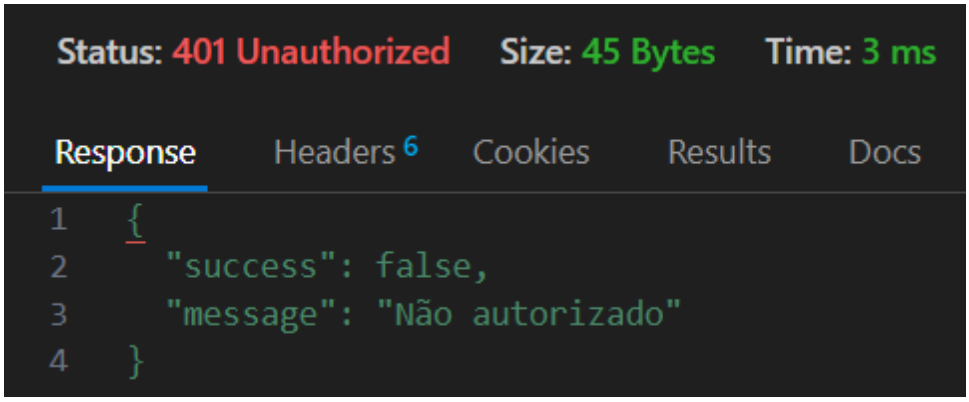
GET – retornar usuários

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

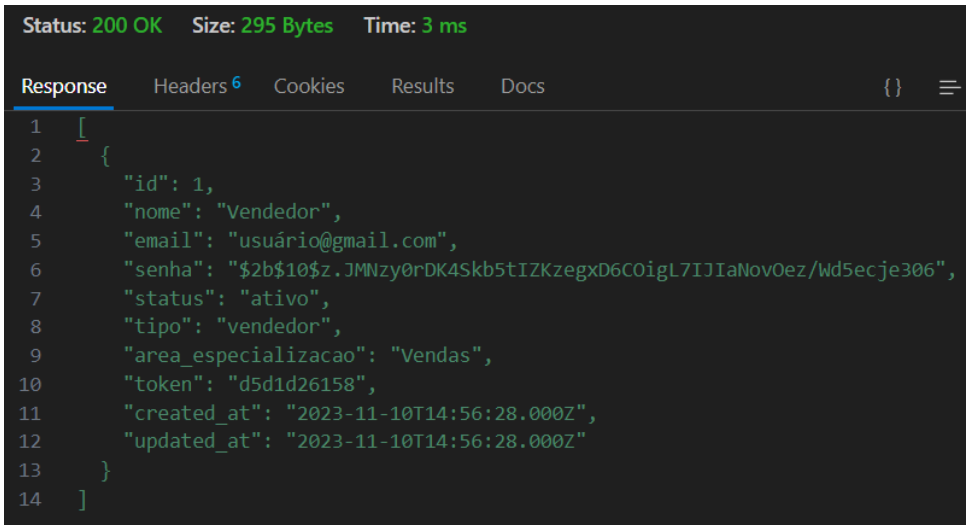
Requisição:



Resposta sem ter feito login de adm:



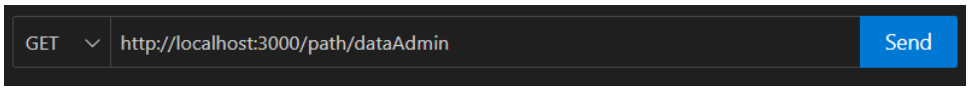
Resposta tendo feito login de adm:



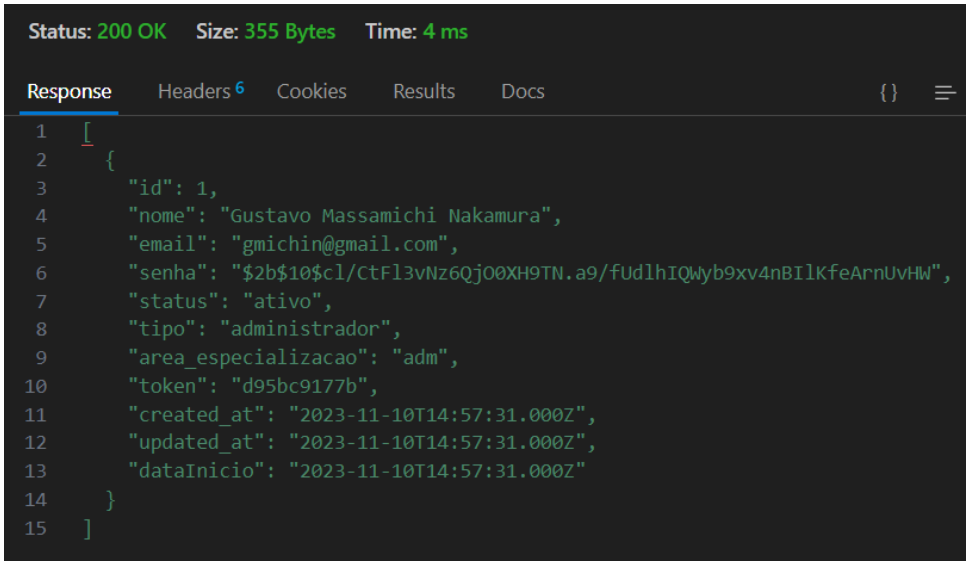
GET – retornar administradores

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

Requisição:



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:

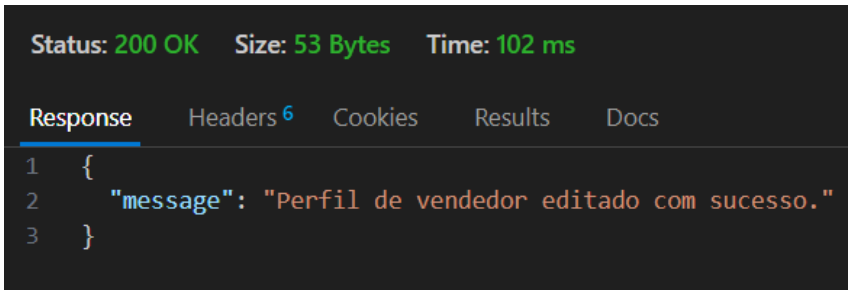
PUT - Edição de usuário comum por token
Link: http://localhost:3000/path/users/(token_usuário)
Body (json):

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

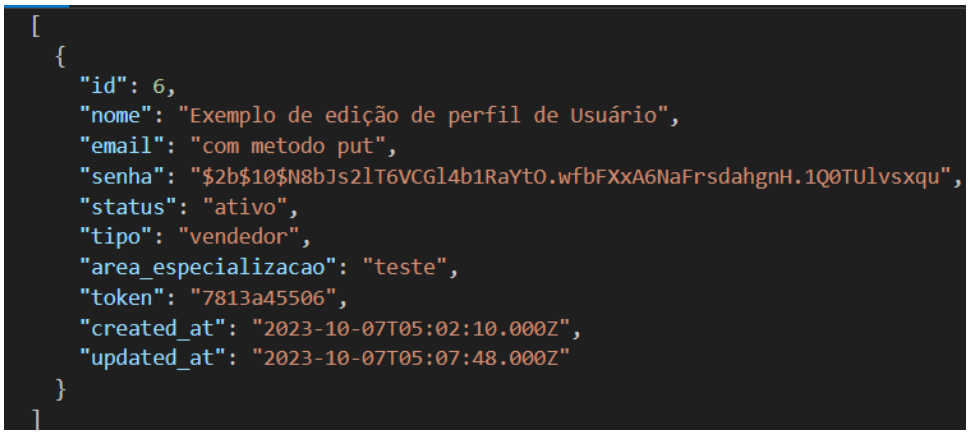
Nota: O token pode ser adquirido com a requisição GET mostrada anteriormente;

Requisição:

Resposta:



GET do usuário após edição:



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

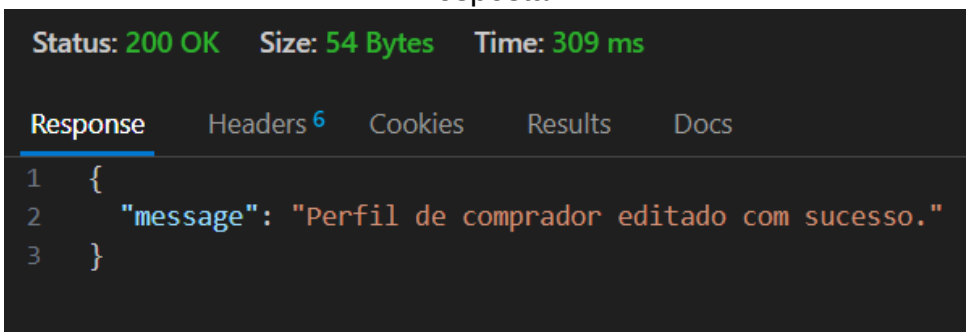
PUT - Edição de administrador para comprador por token
Link: http://localhost:3000/path/admin/(token_adm)
Body (json):

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

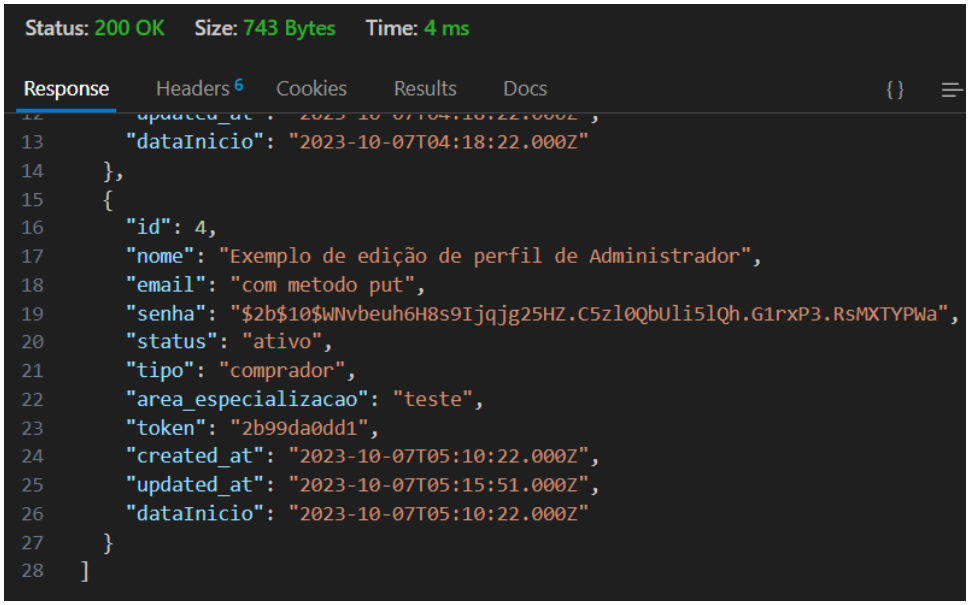

Nota: O token pode ser adquirido com a requisição GET mostrada anteriormente;

Requisição:

Resposta:



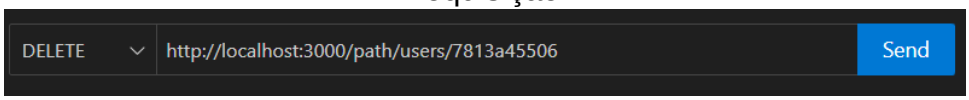
GET do admin após edição:



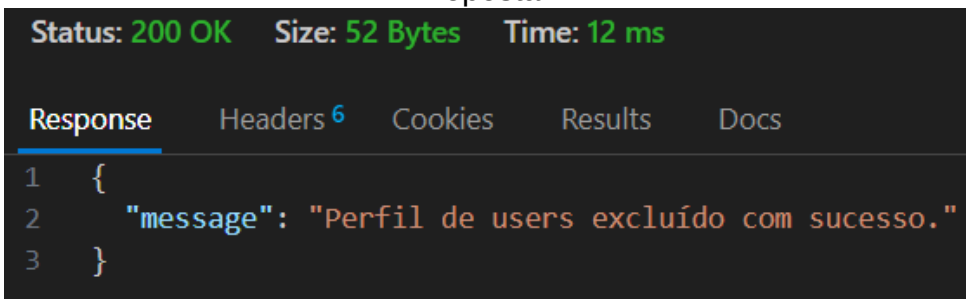
Soft delete para desativar usuários:

DELETE - Deletar usuários por token
Link: http://localhost:3000/path/users/(token_usuário)
Nota: O token pode ser adquirido com a requisição GET mostrada anteriormente;
Nota 2: Não será possível deletar usuários vinculados a transações e itens;

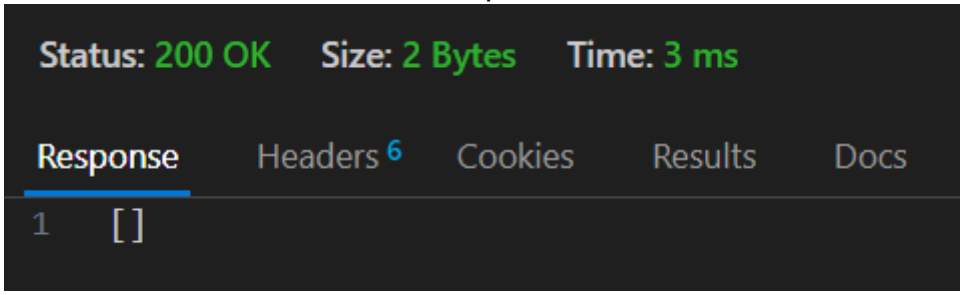
Requisição:



Reposta:

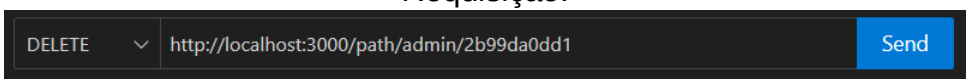


GET user após delete:

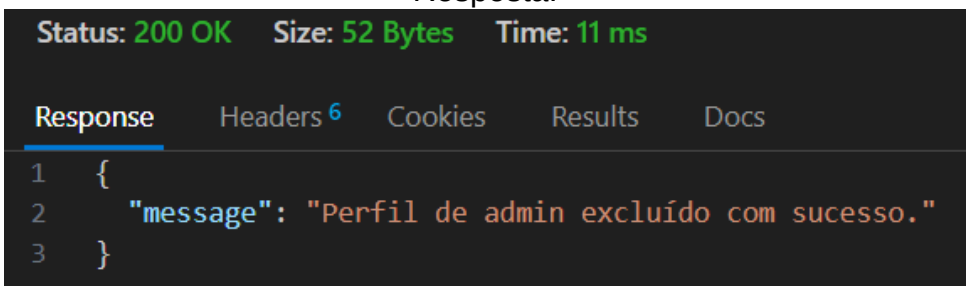


DELETE – Deletar administradores por token
Link: http://localhost:3000/path/admin/(token_admin)
Nota: O token pode ser adquirido com a requisição GET mostrada anteriormente;

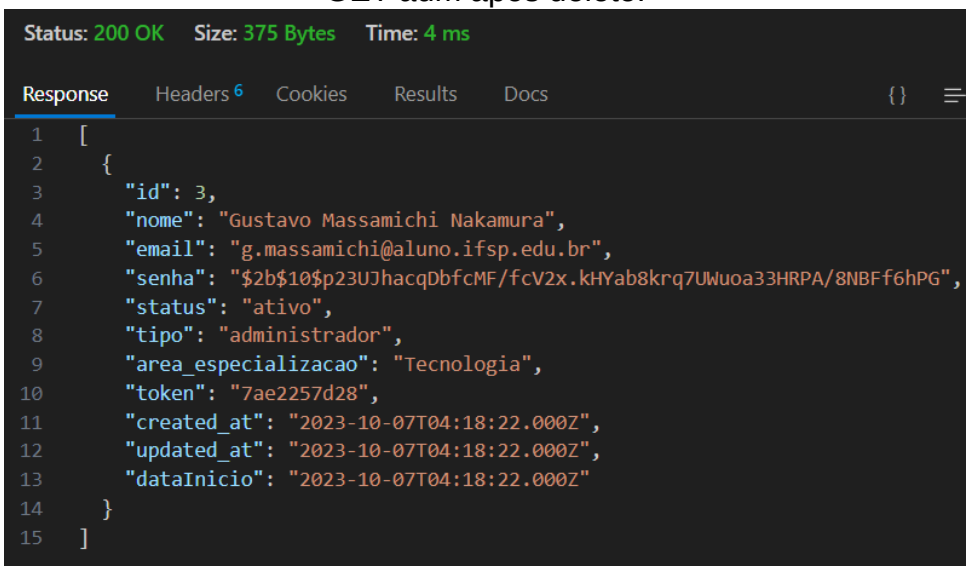
Requisição:



Resposta:



GET adm após delete:

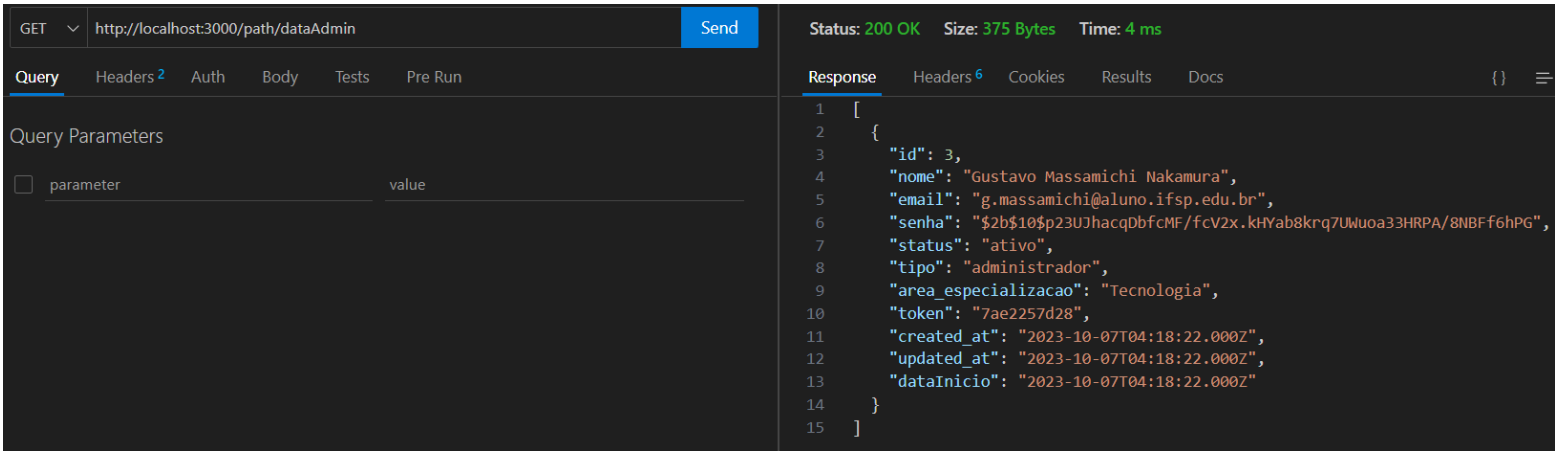


Sistema de criptografia para senhas:

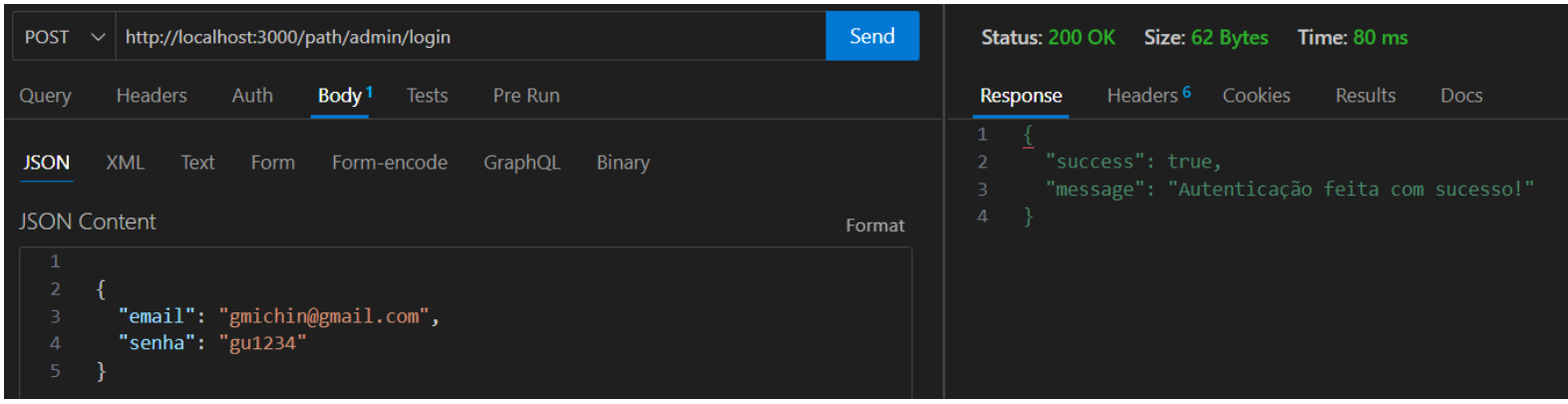
No meu repositório do github, no caminho routes/path.js você poderá ver a seguinte comprovação de criptografia

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

Observe que a senha está criptografada:



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



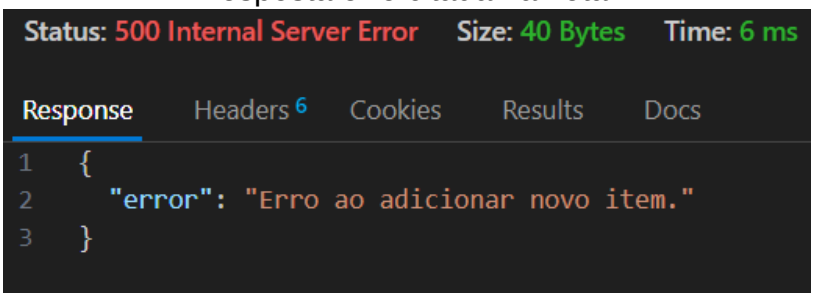
Adição de novos itens:

POST – Adicionar novos itens
Link: http://localhost:3000/path/items
Body (json):

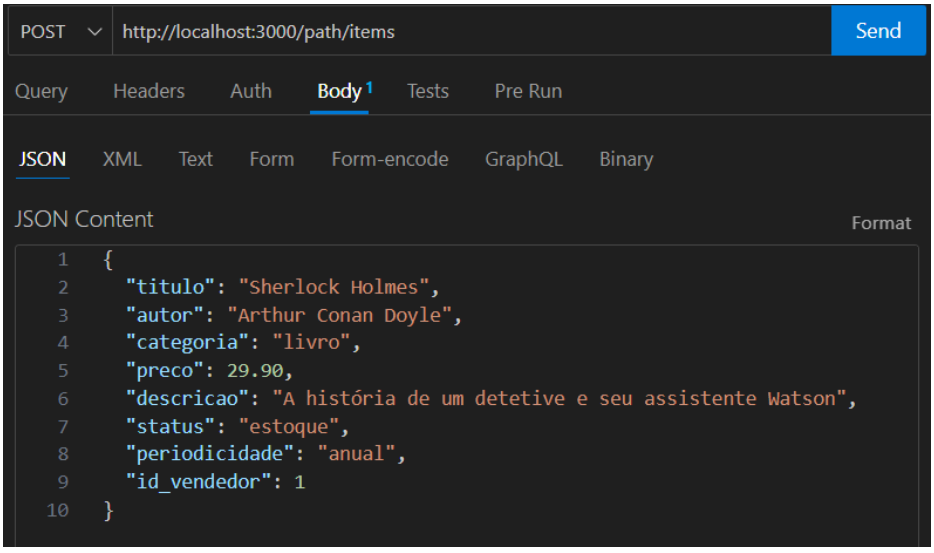
```
{
  "titulo": "Nome do livro",
  "autor": "Autor do livro",
  "categoria": "se é livro, jornal, revista ou periodico",
  "preco": preço
  "descricao": "breve descrição do livro",
  "status": "se está ativo, inativo, estoque ou fora_de_estoque",
  "periodicidade": " periodicidade ",
  "id_vendedor": id_vendedor"
```

}
Nota: os campos de categoria e status só tem as opções citadas, se colocar algo fora dessas opções irá dar erro. Outro possível erro é inserir um id_vendedor não existente.

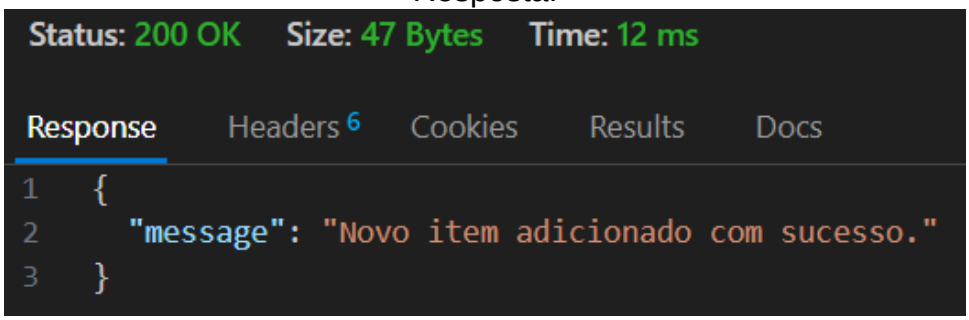
Resposta erro citada na nota:



Requisição:



Resposta:

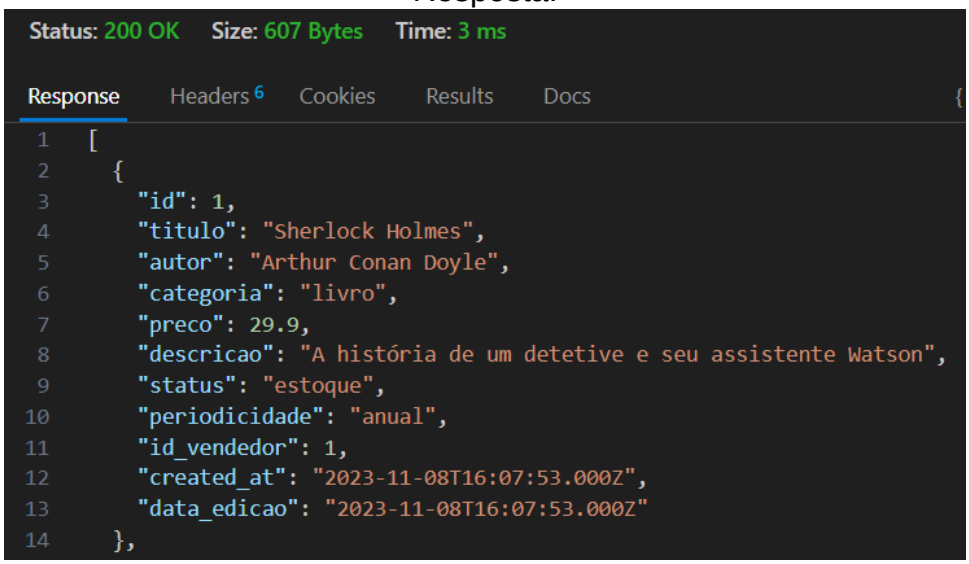


Listagem de itens:

GET – Retorna todos os itens
Link: http://localhost:3000/path/items

Requisição:

Resposta:



Edição de itens:

PUT - Edição de itens específicos por id de item

Link: http://localhost:3000/path/items/edit/(id_do_item)

Body (json):

{

"titulo": "Nome do livro",

"autor": "Autor do livro",

"categoria": "se é livro, jornal, revista ou periodico",

"preco": preço

"descricao": "breve descrição do livro",

"status": "se está ativo, inativo, estoque ou fora_de_estoque",

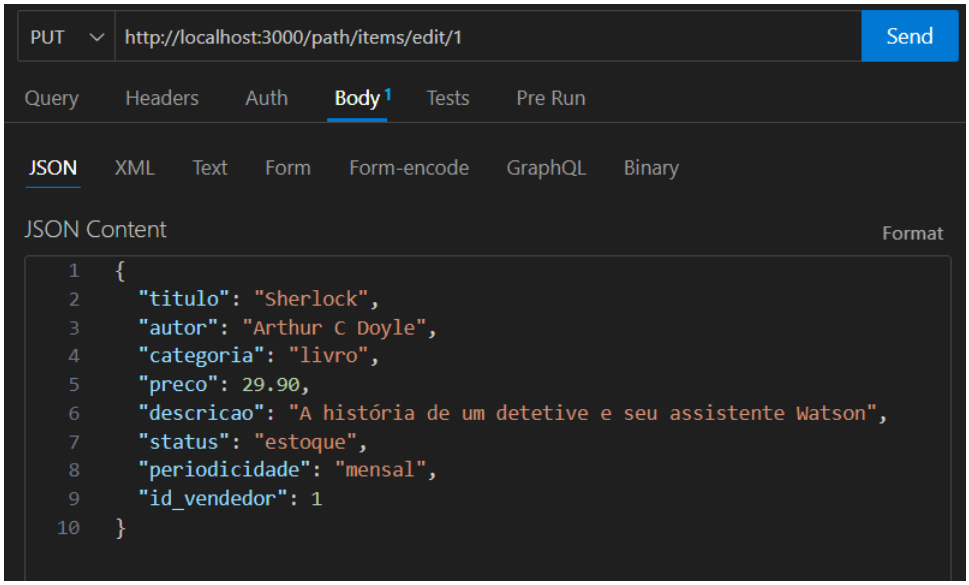
"periodicidade": " periodicidade ",

"id_vendedor": id_vendedor"

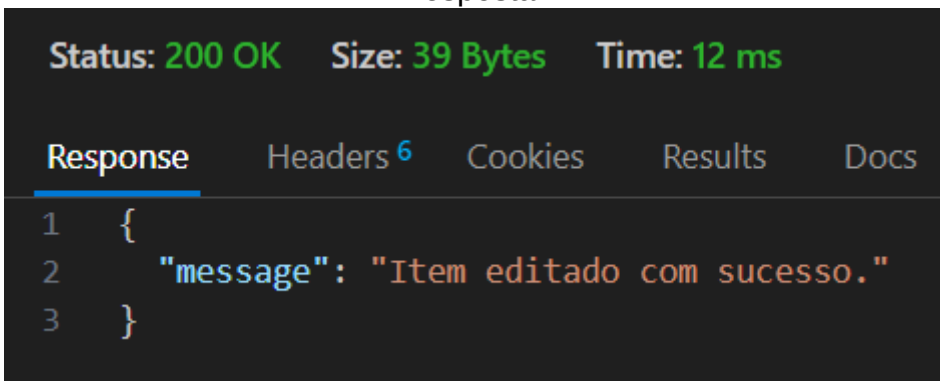
}

Nota: os campos de categoria e status só tem as opções citadas, se colocar algo fora dessas opções irá dar erro

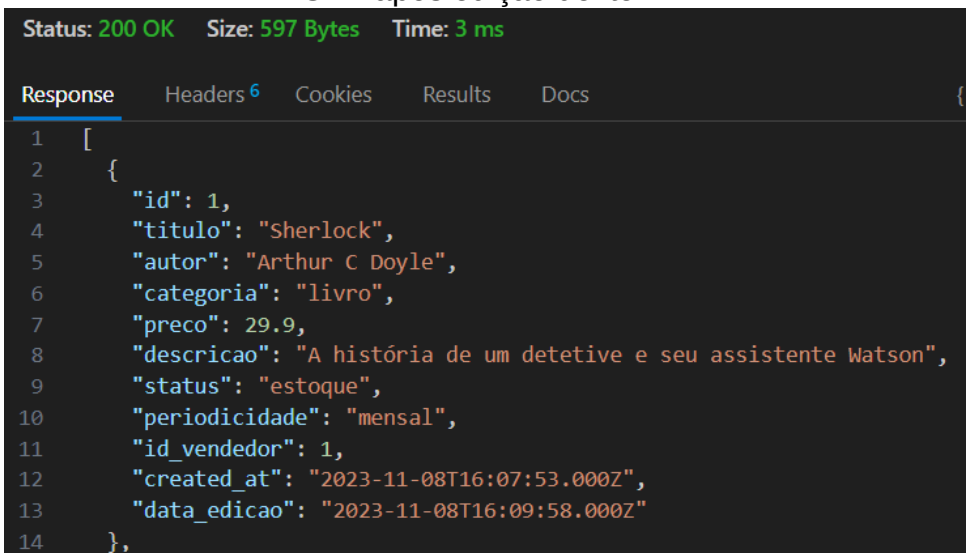
Requisição:



Resposta:



GET após edição de item:

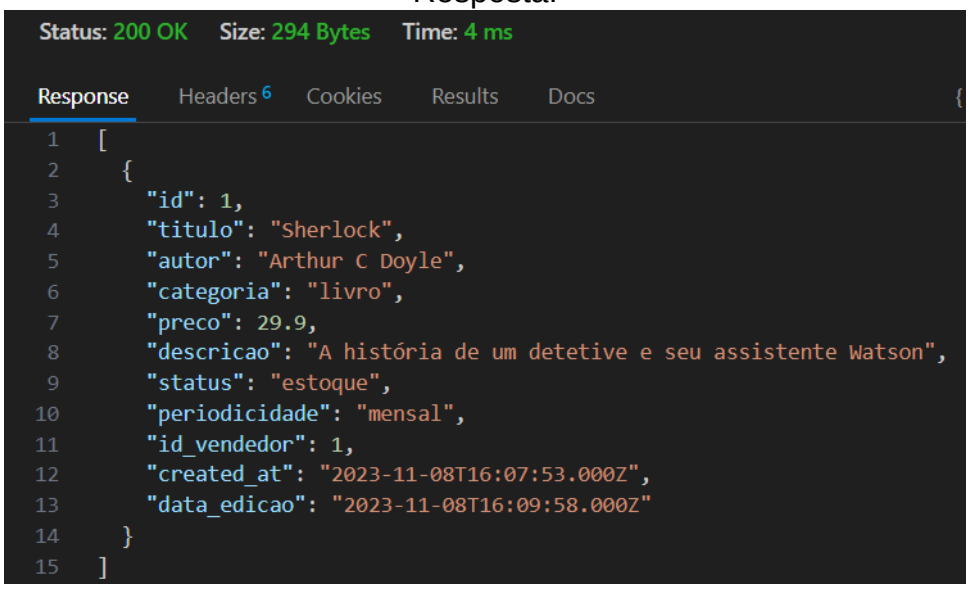


Busca básica de itens:

GET – Busca de itens específicos pelo id do item
Link: http://localhost:3000/path/items/(id_do_item)

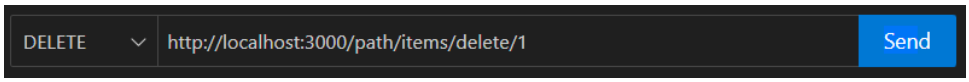
Requisição:

Resposta:

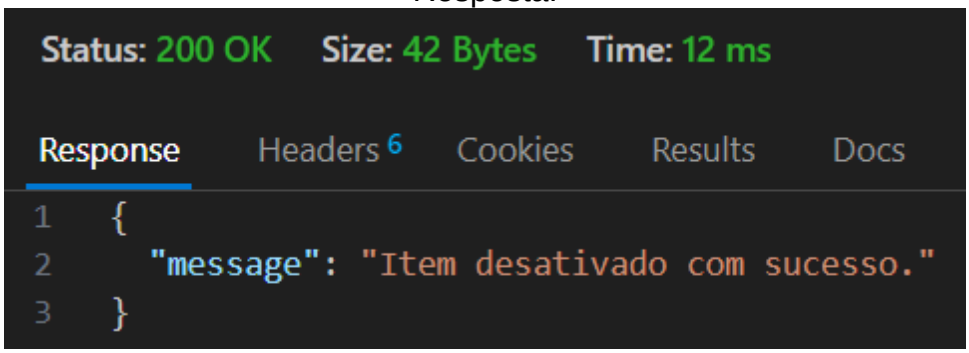


Requisição adicional itens:
DELETE – Deletar itens por id do item
Link: http://localhost:3000/path/items/delete/(id_do_item)
Nota: O item não será deletado antes de transação vinculada.

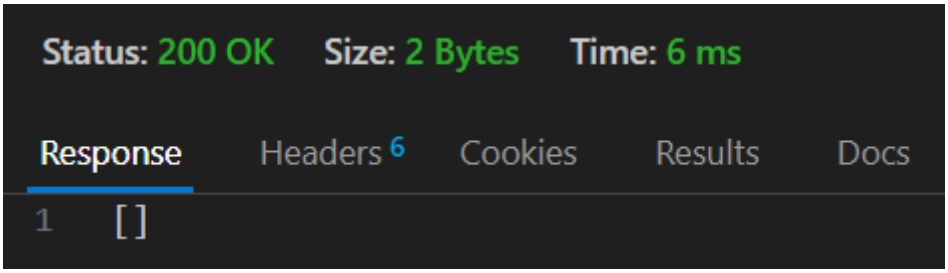
Requisição:



Resposta:



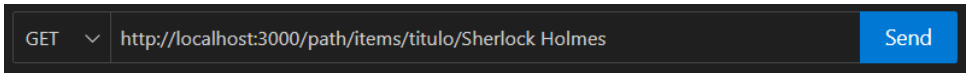
GET após delete do item:



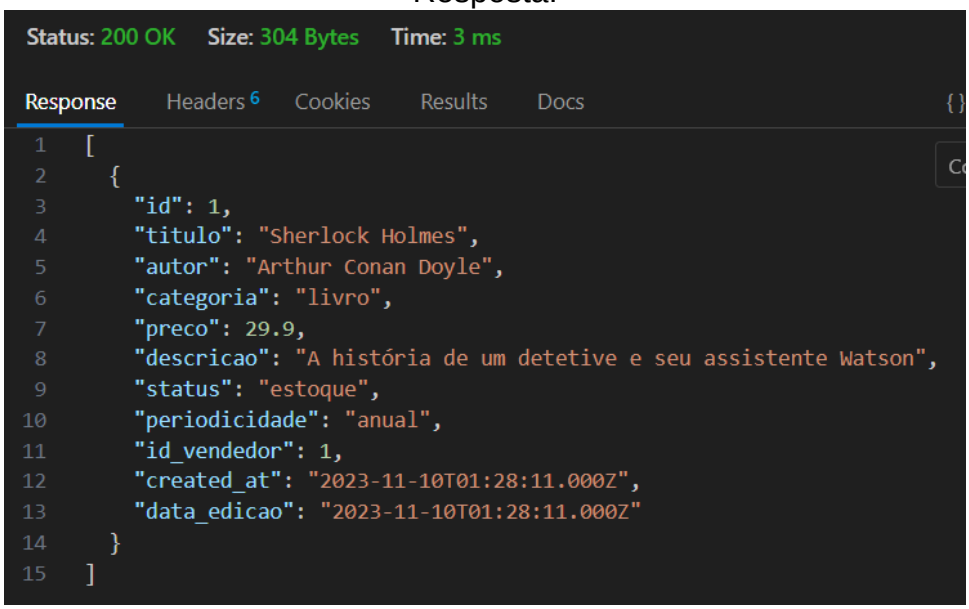
GET – BUSCA DE ITENS POR CRITÉRIOS
Link: http://localhost:3000/path/items/(critério)/(valor_do_critério)

Nota: O critério pode ser qualquer uma das variáveis do json de itens.

Requisição:



Resposta:



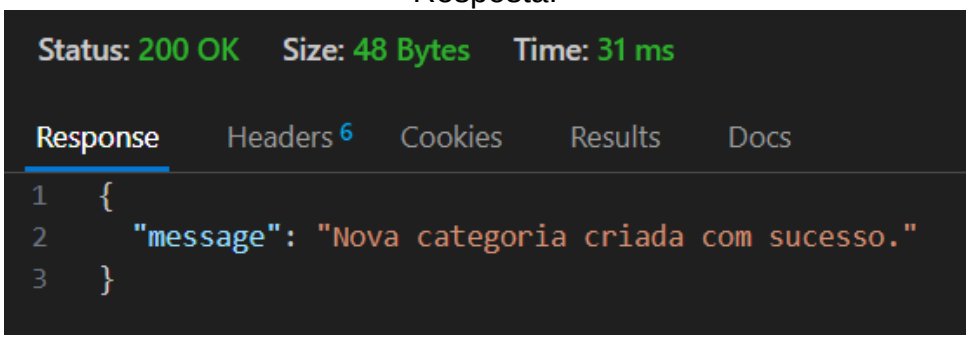
Criação de categorias:

POST – Criar categorias
Link: http://localhost:3000/path/categories
Body (json):

```
{
  "nome": "nome da categoria",
  "descricao": "Descrição da categoria"
}
```

Requisição:

Resposta:



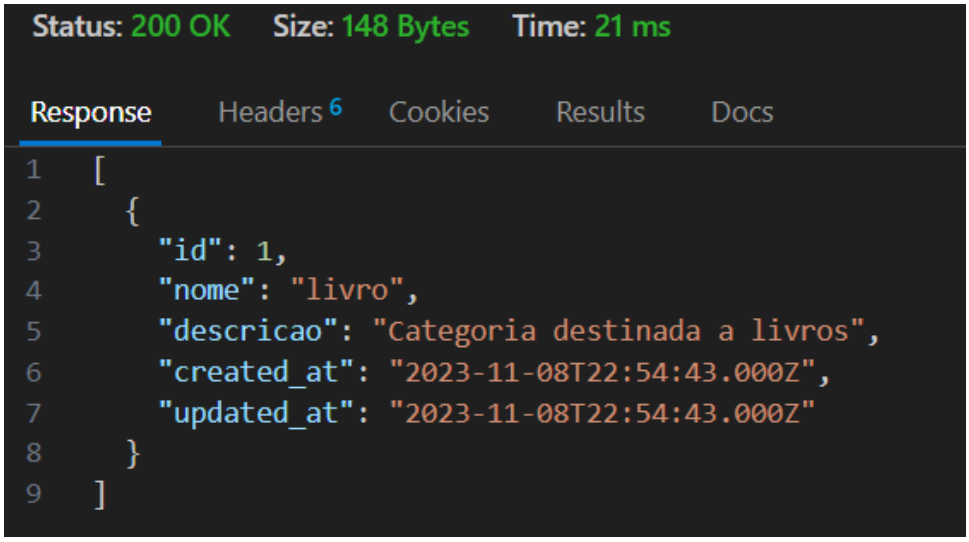
Listagem de categorias:

GET – Listar categorias

Link: http://localhost:3000/path/categories

Requisição:

Resposta:



Edição de categorias:

PUT – Editar categorias por id de categoria

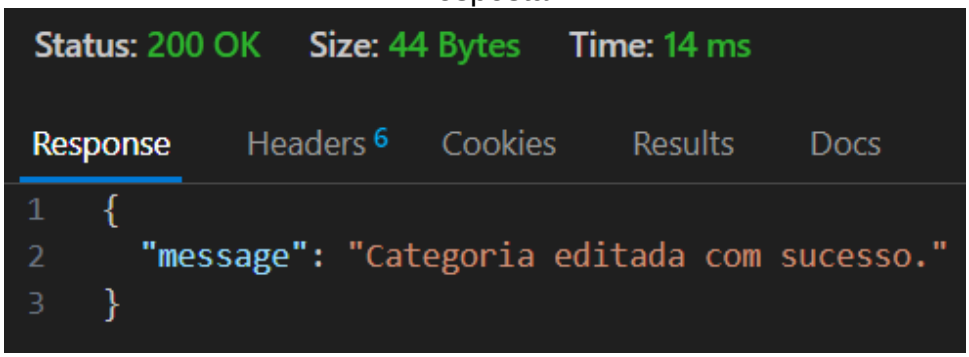
Link: http://localhost:3000/path/categories/edit/(id_da_categoria)

Body (json):

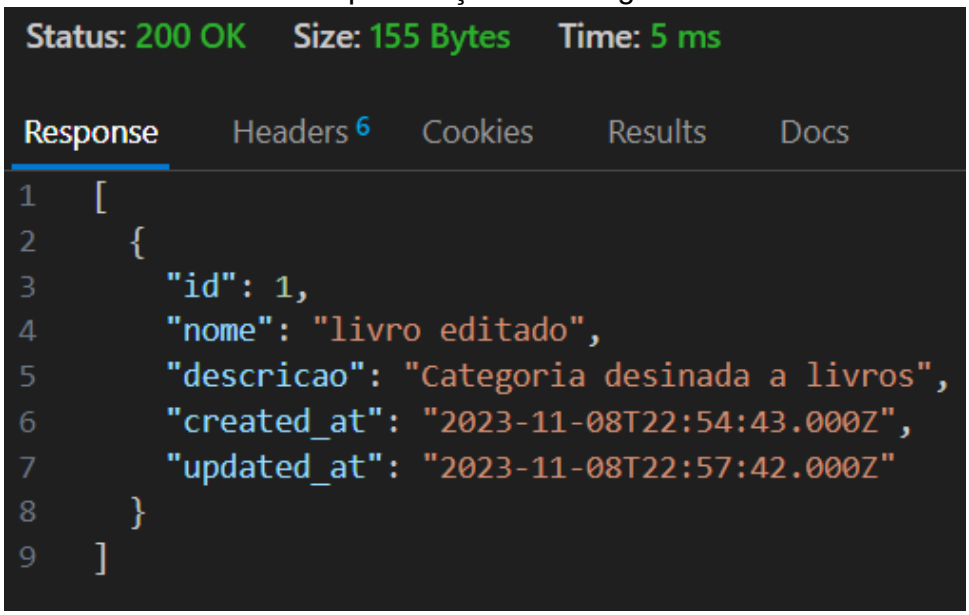
```
{
  "nome": "nome da categoria",
  "descricao": "Descrição da categoria"
}
```

Requisição:

Resposta:



GET após edição de categorias:

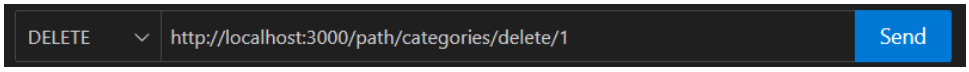


Soft delete para categorias:

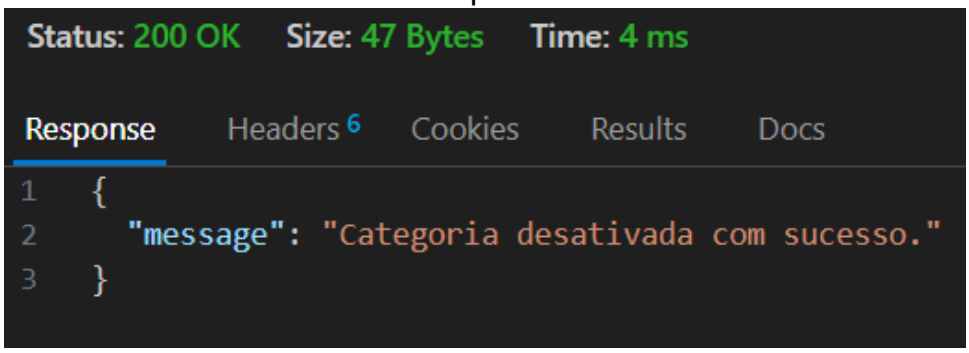
DELETE – Deletar as categorias por id de categoria

Link: http://localhost:3000/path/categories/delete/(id_da_categoria)

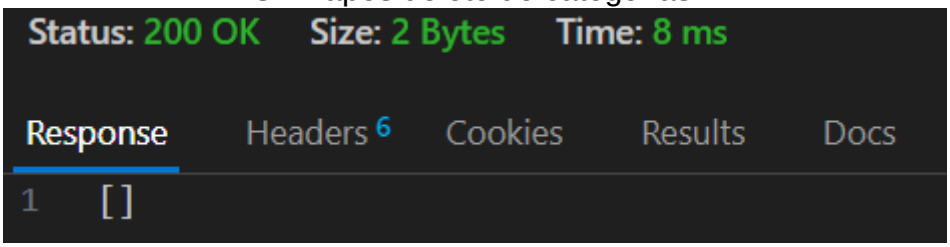
Requisição:



Resposta:



GET após delete de categorias:

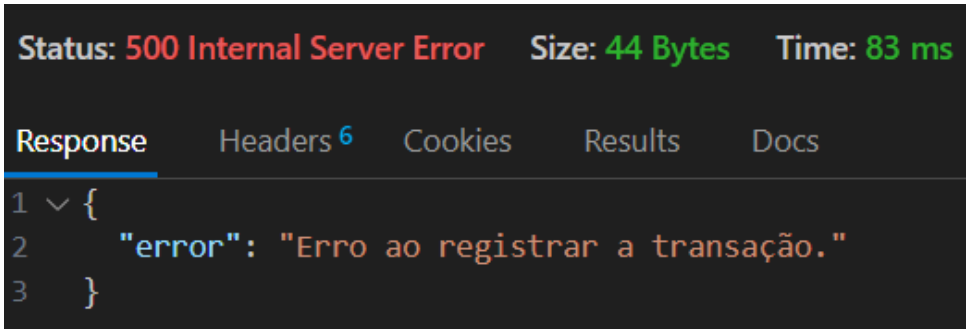


Registro de novas transações:

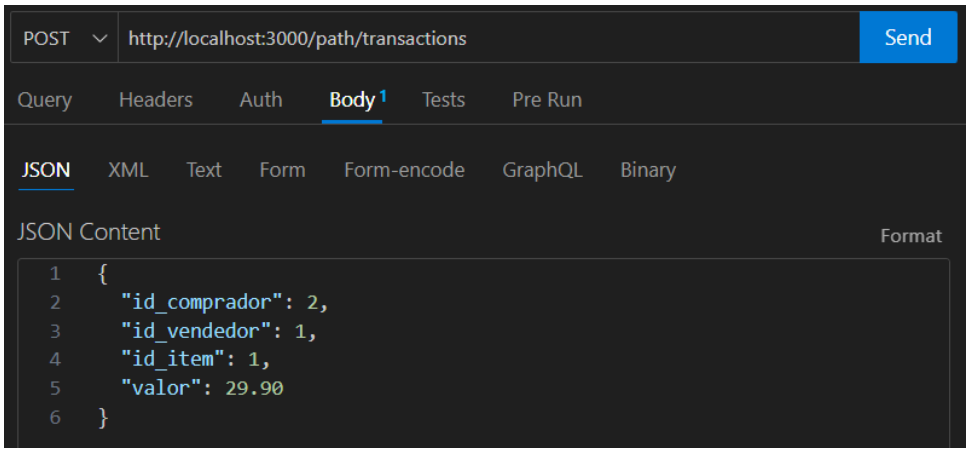
POST – Registro de novas transações

Link: http://localhost:3000/path/transactions
Body (json):
{
  "id_comprador": id de comprador existente em users,
  "id_vendedor": id de vendedor existente em users,
  "id_item": id de itens existentes em items,
  "valor": valor da transação
}
Nota: Não vai ser possível registrar transações com id_comprador, id_vendedor ou id_item inexistente;

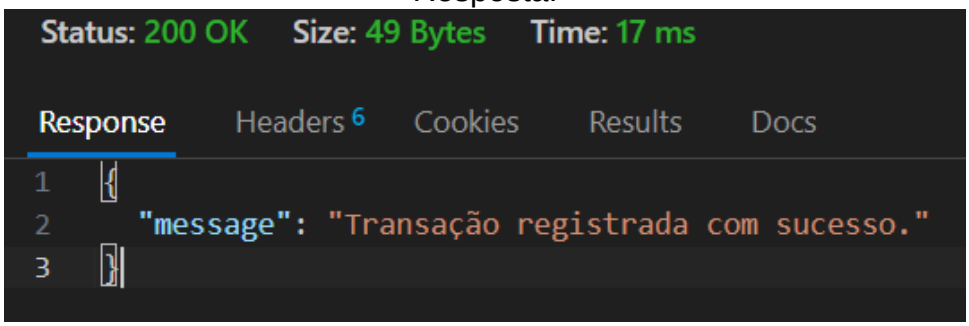
Erro citado na nota:



Requisição:

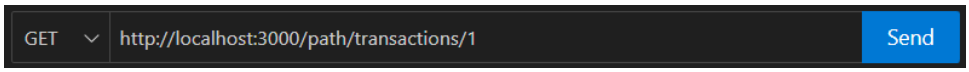


Resposta:

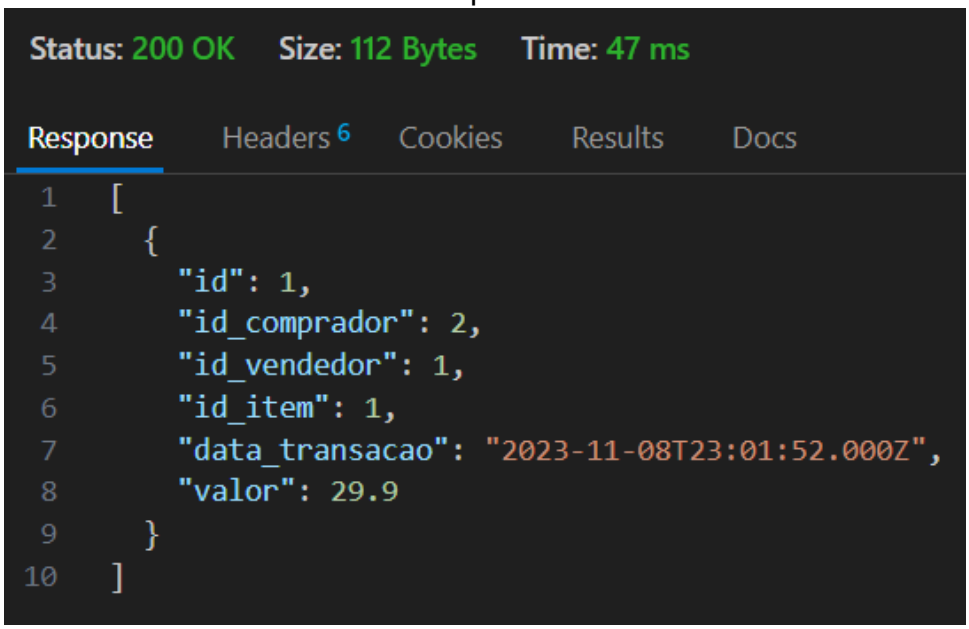


Visualização de transações para um usuário específico:
GET – Vizualizar transação por id de comprador ou vendedor
Link: http://localhost:3000/path/transactions/(id_do_comprador_ou_vendedor)

Requisição:



Resposta:

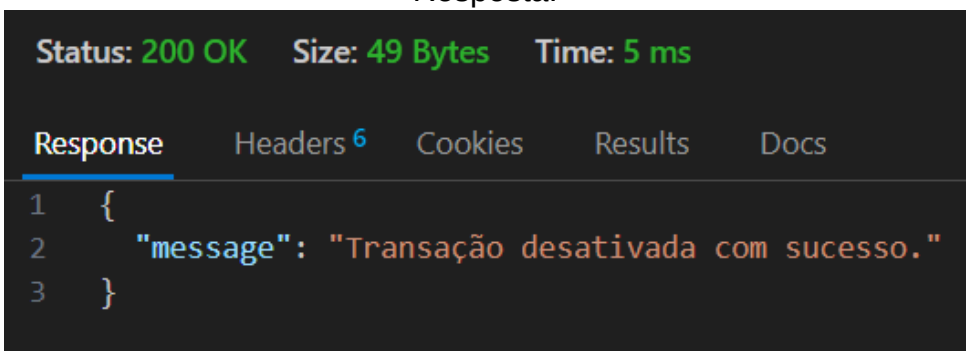


Requisição adicional de transação:

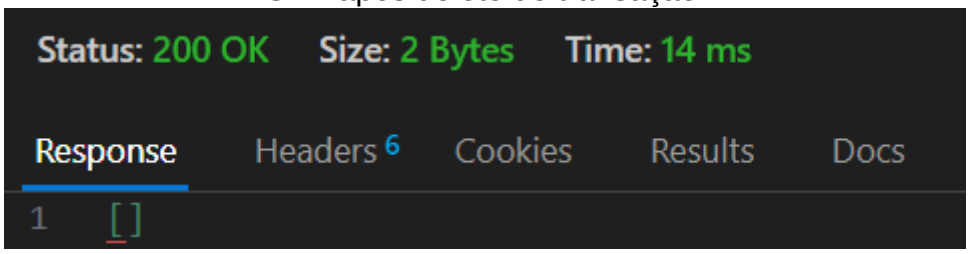
DELETE – Deletar transação por id de transação
Link: http://localhost:3000/path/transactions/delete/(id_da_transação)

Requisição:

Resposta:



GET após delete de transação:



BREVE RESUMO DAS REQUISIÇÕES