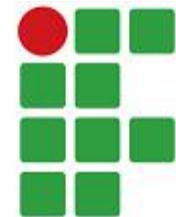


PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática

Lucas Sampaio Leite



**INSTITUTO
FEDERAL**

Baiano

Implementando uma busca de um contato usando @app.get

- O `jsonify` é uma função tradicional do Flask usada para converter objetos Python (como dicionários, listas) em uma resposta JSON válida, configurando automaticamente o cabeçalho `Content-Type: application/json` e serializando os dados para JSON.
- A partir do Flask 1.1 (e versões recentes), pode-se simplesmente retornar um dicionário Python que o Flask:
 - converte automaticamente para JSON
 - adiciona o cabeçalho correto (`Content-Type: application/json`)

Implementando uma busca de um contato usando @app.get

```
@app.get('/contato/<nome>')
def buscar_contato(nome):
    nome = nome.lower()
    if nome in agenda:
        return {
            'nome': nome.capitalize(),
            'telefone': agenda[nome]
        }
    else:
        return {
            'erro': f"Contato '{nome}' não encontrado."
        }
```

HTTP Status Codes

- Status codes HTTP (ou códigos de status HTTP) são números de 3 dígitos que acompanham as respostas enviadas por um servidor web (como Flask) para indicar o resultado da requisição feita por um cliente (navegador, app, frontend etc.).
- Por que usar status codes?
 - Comunicam de forma padronizada o que aconteceu
 - São entendidos universalmente
 - Facilitam a manutenção e testes
 - Boa prática em APIs REST

HTTP Status Codes

Faixa	Significado geral	Exemplos comuns
1xx	Informativo	(raramente usados em APIs)
2xx	Sucesso	200 OK, 201 Created, 204 No Content
3xx	Redirecionamento	301 Moved Permanently, 302 Found
4xx	Erro do cliente	400 Bad Request, 401 Unauthorized, 404 Not Found
5xx	Erro do servidor	500 Internal Server Error, 503 Service Unavailable

Saiba mais: <https://www.devmedia.com.br/http-status-code/41222>

HTTP Status Codes

```
@app.get('/contato/<nome>')
def buscar_contato(nome):
    nome = nome.lower()

    if nome in agenda:
        resposta = {
            'nome': nome.capitalize(),
            'telefone': agenda[nome]
        }
        return resposta, 200
    else:
        erro = {
            'erro': f"Contato '{nome}' não encontrado."
        }
        return erro, 404
```

HTTP Status Codes

```
from http import HTTPStatus
```

```
@app.get('/contato/<nome>')
def buscar_contato(nome):
    nome = nome.lower()

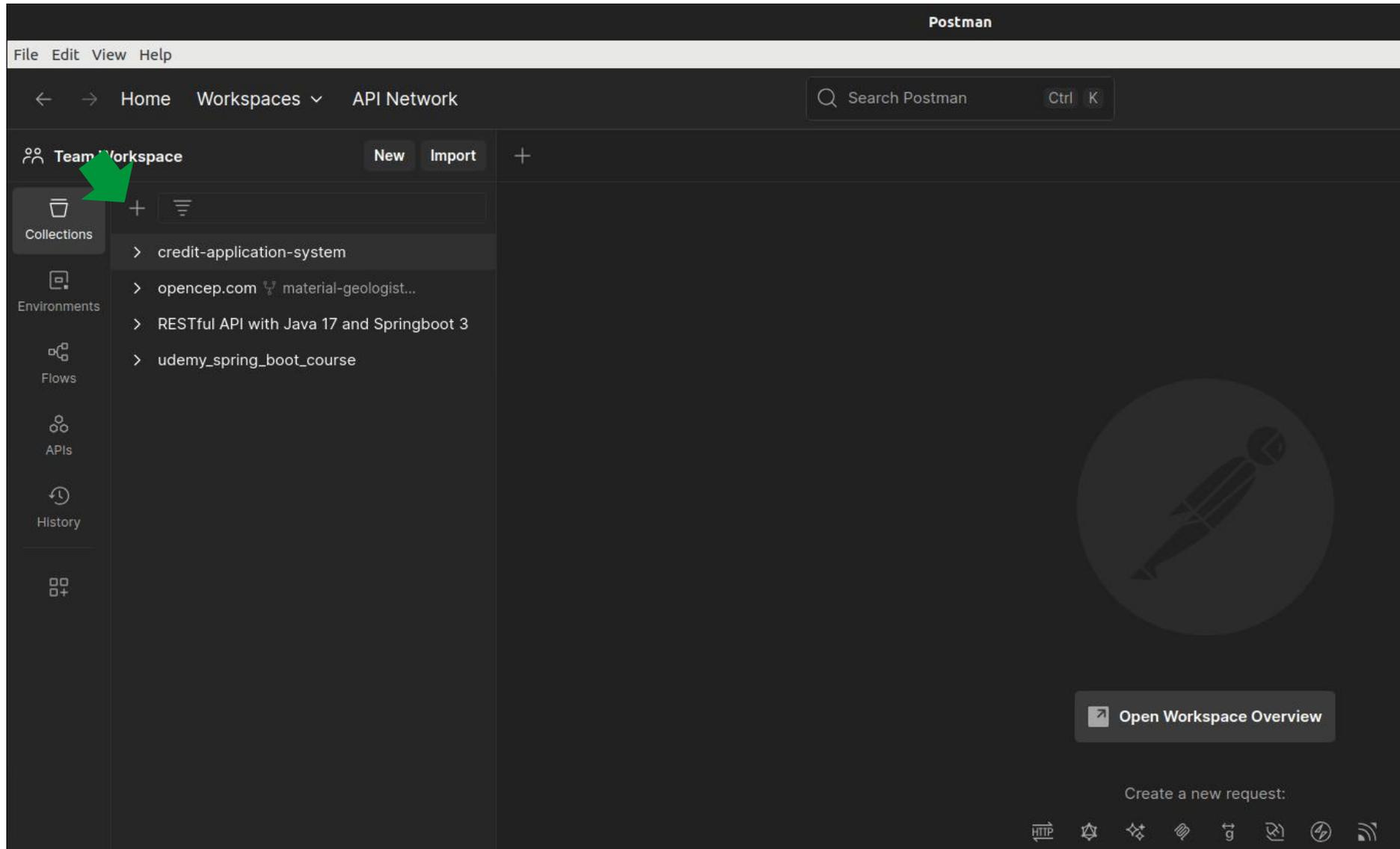
    if nome in agenda:
        resposta = {
            'nome': nome.capitalize(),
            'telefone': agenda[nome]
        }
        return resposta, HTTPStatus.OK
    else:
        erro = {
            'erro': f"Contato '{nome}' não encontrado."
        }
        return erro, HTTPStatus.NOT_FOUND
```

Postman

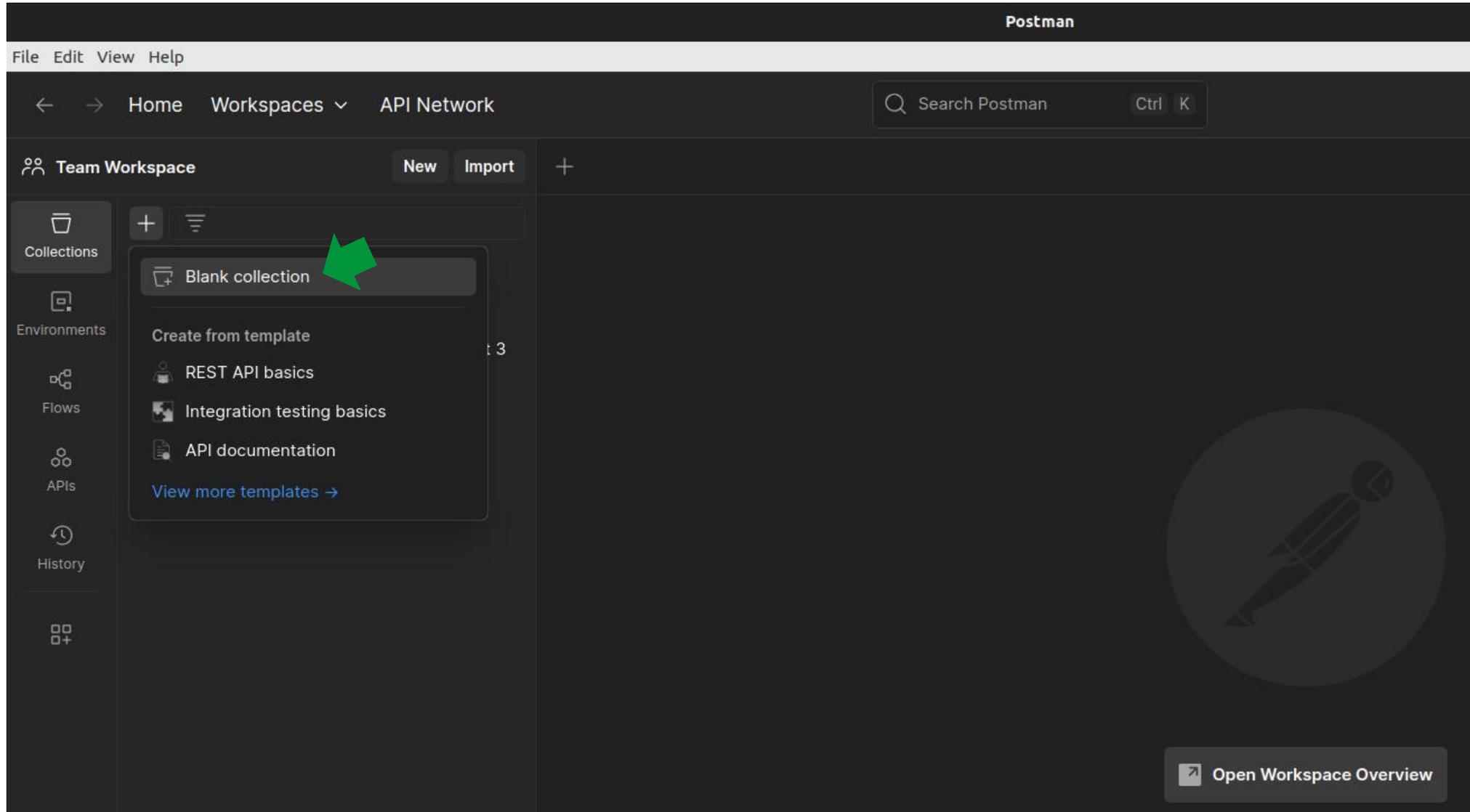
- O Postman é uma ferramenta popular usada para testar APIs RESTful de forma simples e eficiente.
- Com o Postman, é possível enviar requisições HTTP (GET, POST, PUT, DELETE, etc.) e visualizar as respostas da API sem precisar criar um frontend ou um cliente separado.
- Alternativas ao Postman: Insomnia, Hoppscotch, Thunder Client, Paw, etc.



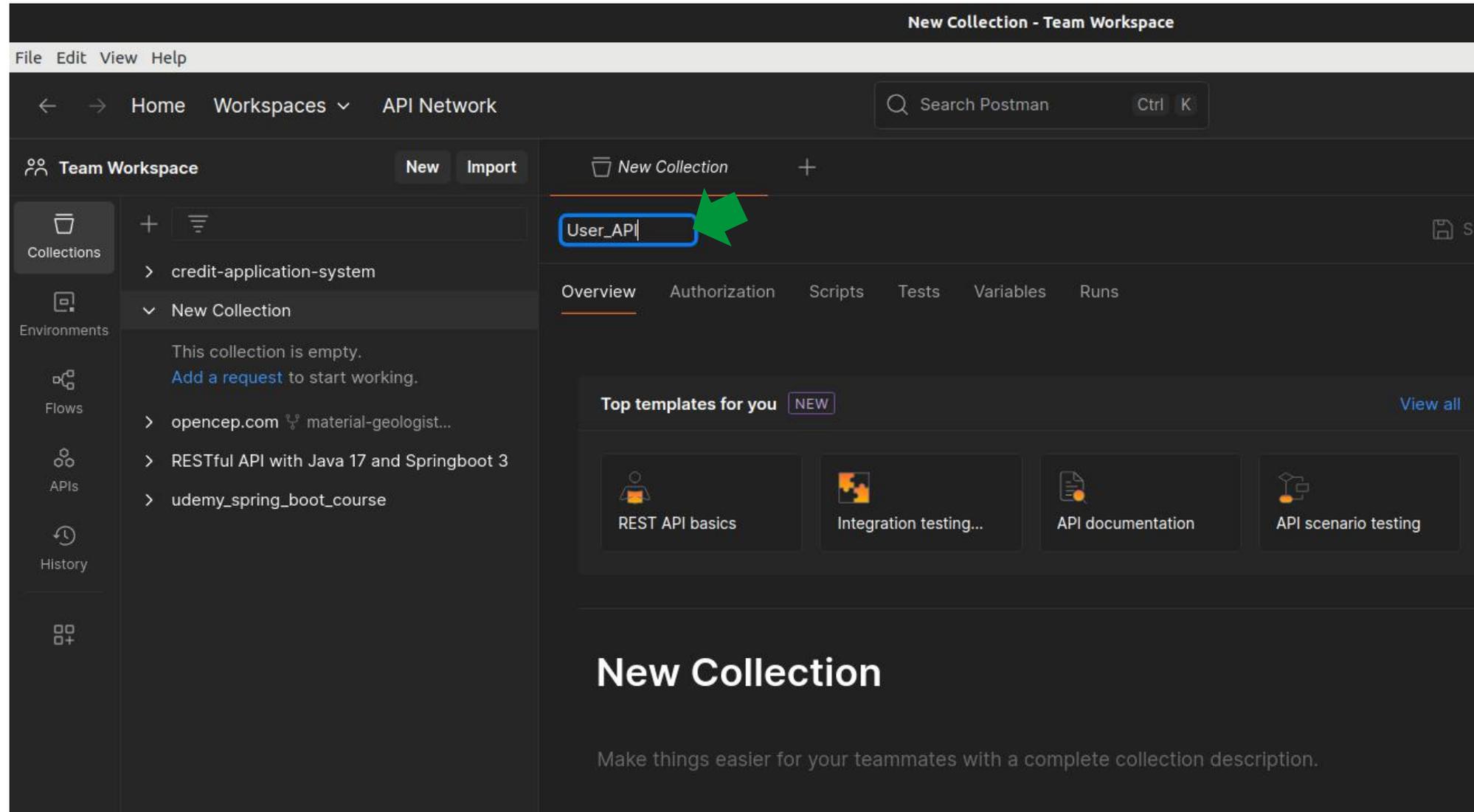
Postman



Postman



Postman



New Collection - Team Workspace

File Edit View Help

← → Home Workspaces API Network

Search Postman Ctrl K

Team Workspace New Import

New Collection

credit-application-system

New Collection

This collection is empty.
[Add a request](#) to start working.

opencep.com material-geologist...

RESTful API with Java 17 and Springboot 3

udemy_spring_boot_course

Overview Authorization Scripts Tests Variables Runs

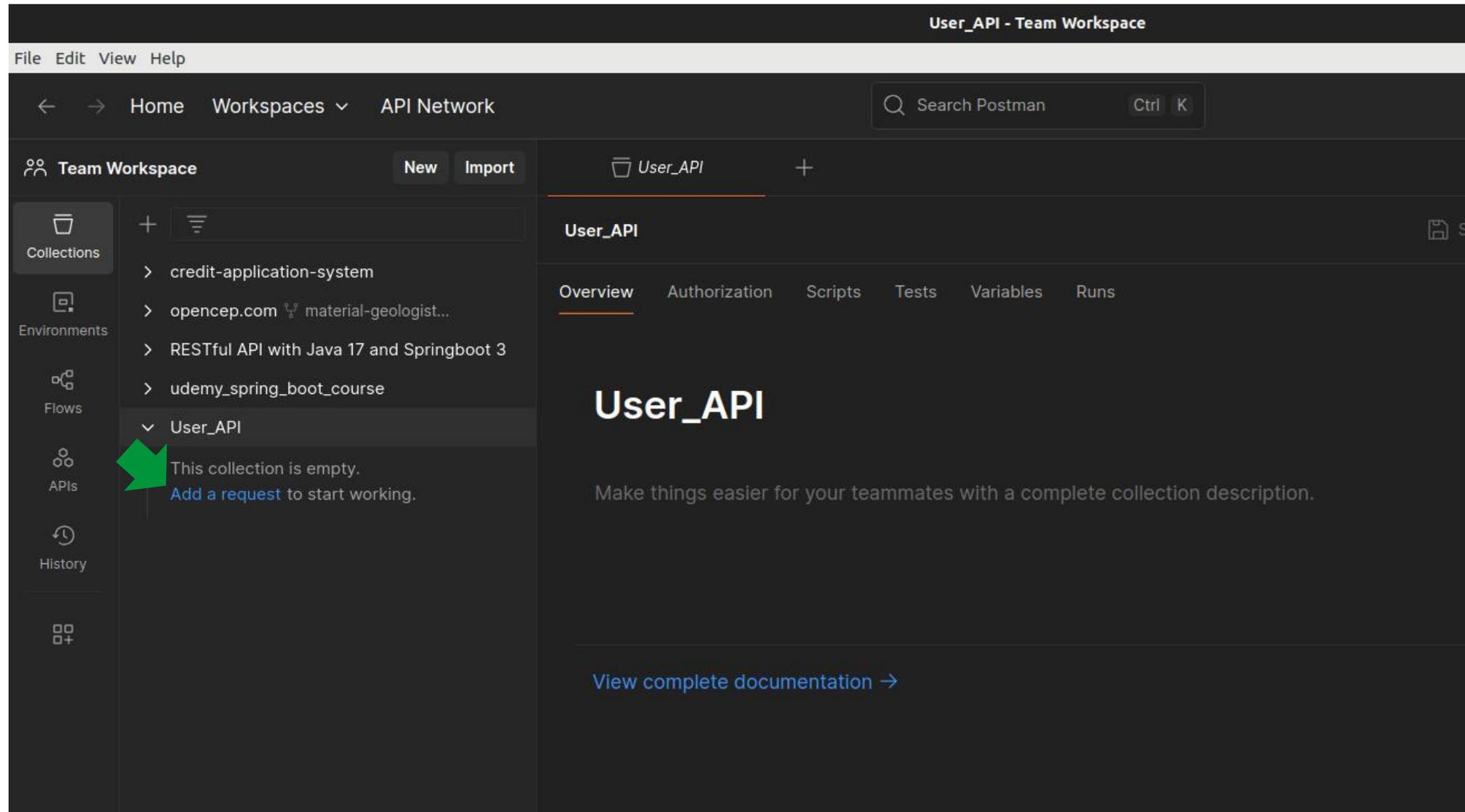
Top templates for you NEW [View all](#)

REST API basics Integration testing... API documentation API scenario testing

New Collection

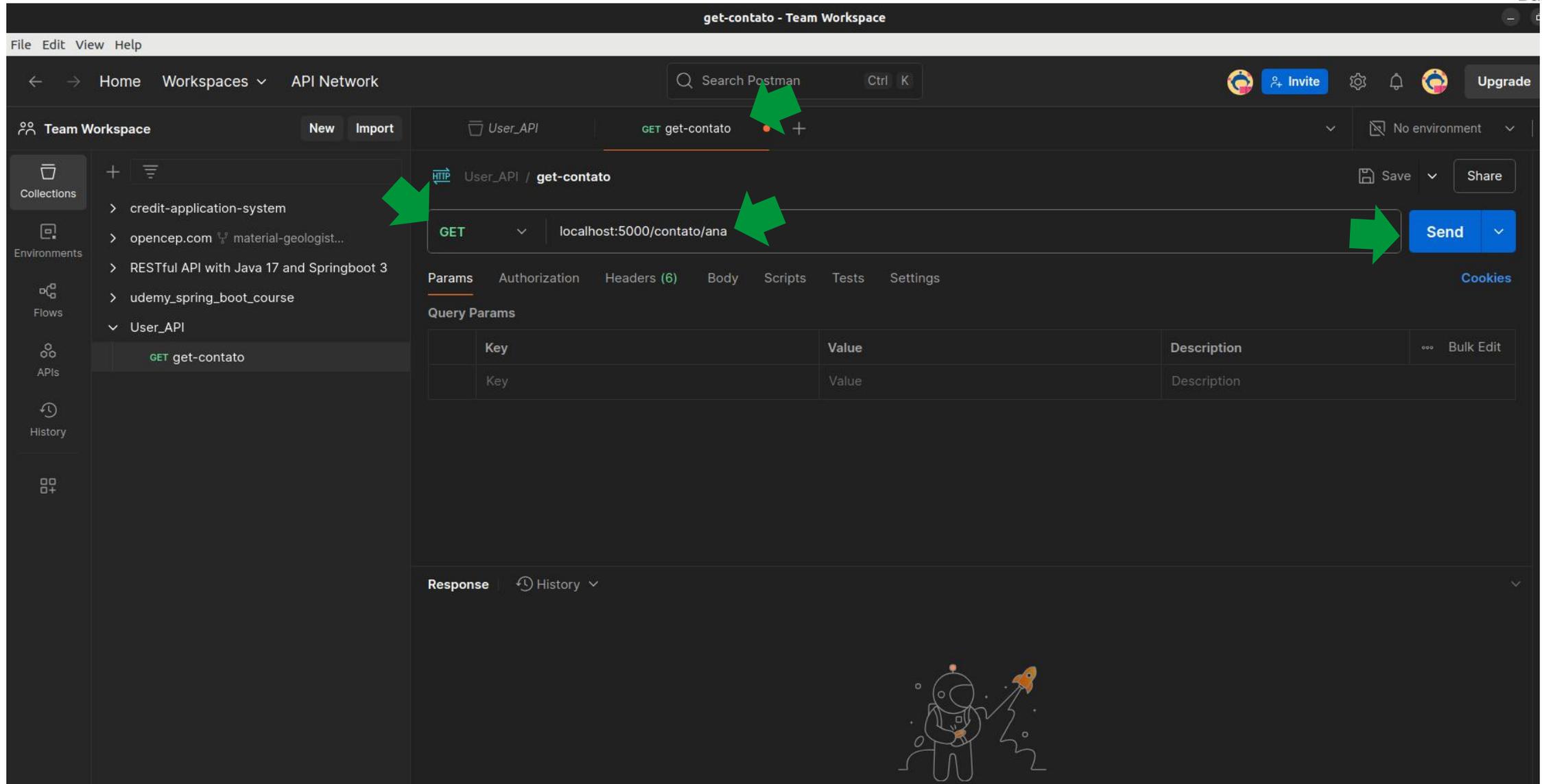
Make things easier for your teammates with a complete collection description.

Postman



The screenshot displays the Postman interface for a workspace named "User_API - Team Workspace". The top navigation bar includes "File", "Edit", "View", and "Help" menus, along with a search bar for "Search Postman" (Ctrl K). The main interface is divided into a left sidebar and a main content area. The sidebar contains sections for "Team Workspace", "Collections", "Environments", "Flows", "APIs", and "History". Under "Collections", several collections are listed, including "credit-application-system", "opencep.com", "RESTful API with Java 17 and Springboot 3", "udemy_spring_boot_course", and "User_API". The "User_API" collection is selected and expanded, showing a message: "This collection is empty. Add a request to start working." A green arrow points to this message. The main content area displays the "User_API" workspace with tabs for "Overview", "Authorization", "Scripts", "Tests", "Variables", and "Runs". The "Overview" tab is active, showing the title "User_API" and a description: "Make things easier for your teammates with a complete collection description." At the bottom, there is a link to "View complete documentation ->".

Postman



get-contato - Team Workspace

File Edit View Help

← → Home Workspaces ▾ API Network

Search Postman Ctrl K

Team Workspace New Import

User_API GET get-contato

HTTP User_API / get-contato

GET localhost:5000/contato/ana

Params Authorization Headers (6) Body Scripts Tests Settings

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

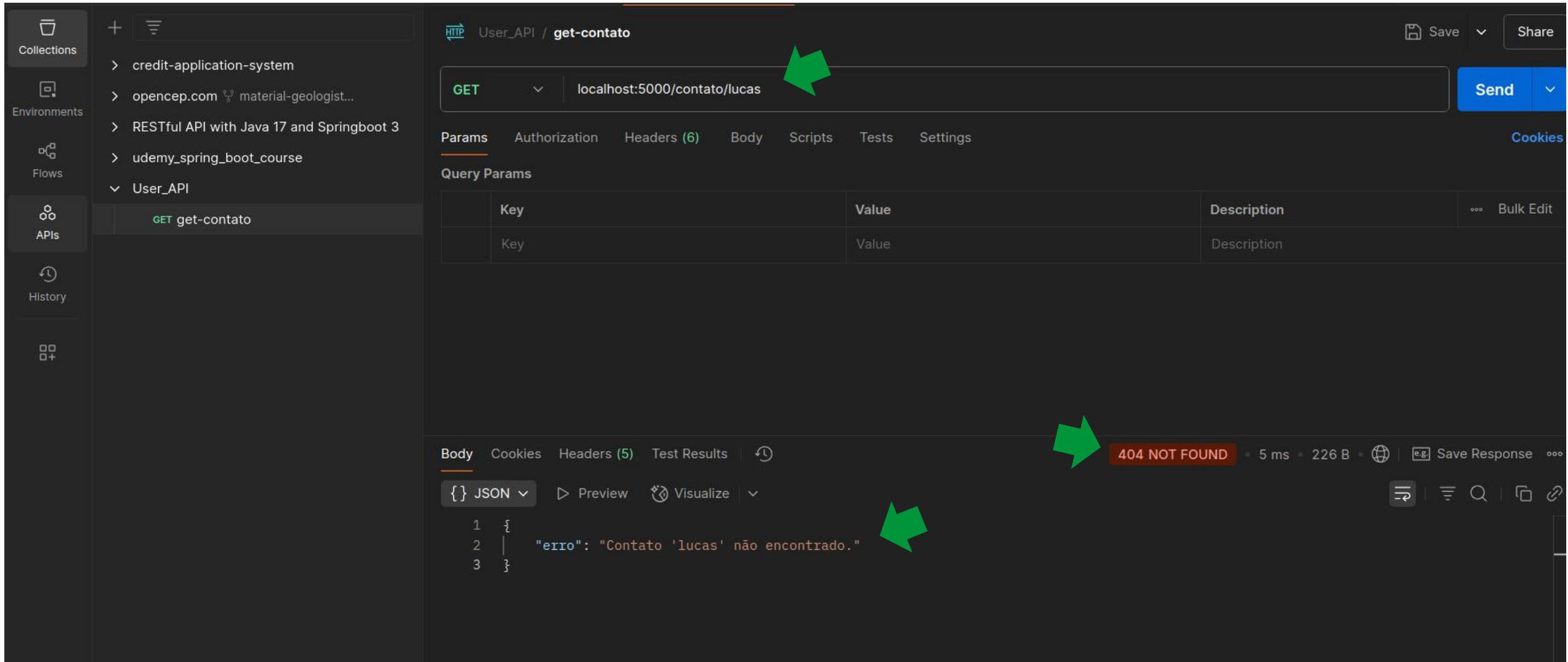
Response History

Send

Postman



Postman



The screenshot displays the Postman interface for a REST client. The left sidebar shows the 'Collections' panel with 'User_API' expanded to 'GET get-contato'. The main workspace shows a GET request to 'localhost:5000/contato/lucas'. The 'Query Params' table is empty. The response is a 404 NOT FOUND error with a 5 ms duration and 226 B of data. The response body is shown in JSON format:

```
{
  "erro": "Contato 'lucas' não encontrado."
}
```

Key	Value	Description
Key	Value	Description

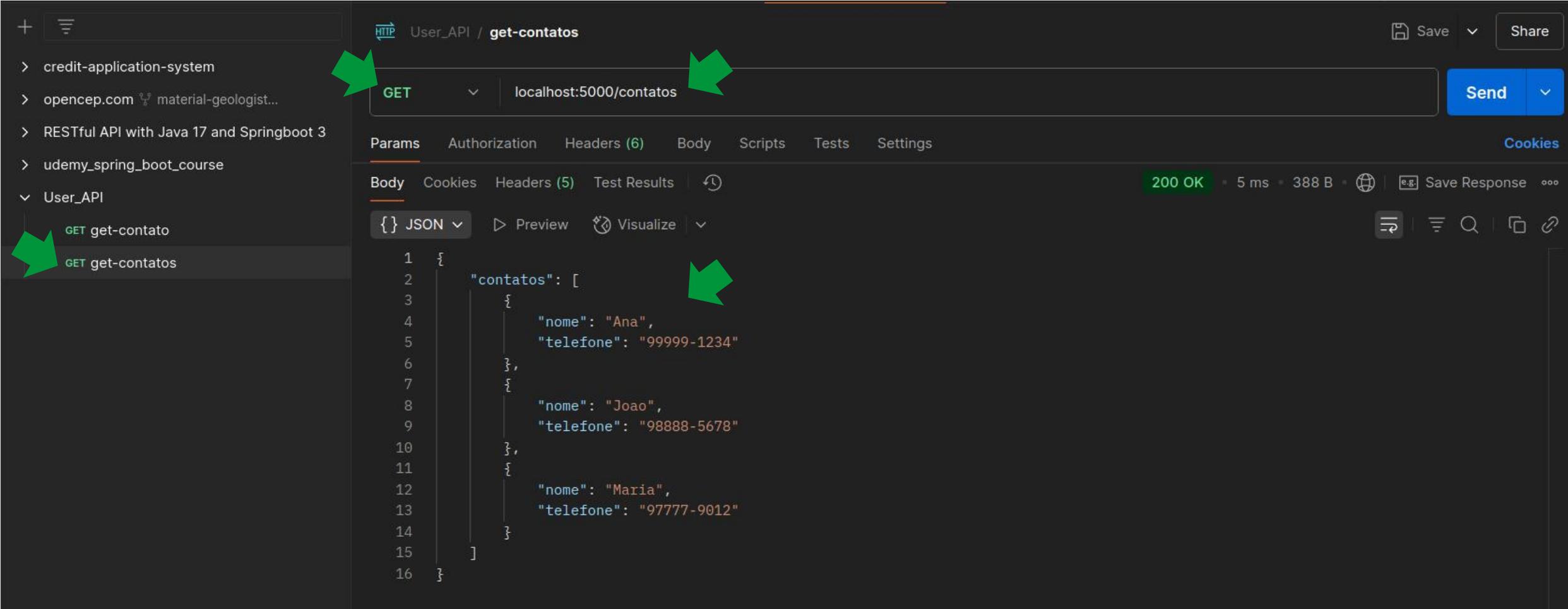
Implementando a listagem de todos os contatos usando @app.get

```
@app.get('/contatos')
def listar_contatos():
    if not agenda:
        resposta = {'mensagem': 'A agenda está vazia.'}
        return resposta, HTTPStatus.OK

    contatos = []
    for nome, telefone in agenda.items():
        contato = {
            'nome': nome.capitalize(),
            'telefone': telefone
        }
        contatos.append(contato)

    return {'contatos': contatos}, HTTPStatus.OK
```

Implementando a listagem de todos os contatos usando @app.get



The screenshot shows a REST client interface with the following details:

- Request:** Method: GET, URL: localhost:5000/contatos
- Response:** Status: 200 OK, Time: 5 ms, Size: 388 B
- Response Body (JSON):**

```
1  {
2    "contatos": [
3      {
4        "nome": "Ana",
5        "telefone": "99999-1234"
6      },
7      {
8        "nome": "Joao",
9        "telefone": "98888-5678"
10     },
11     {
12       "nome": "Maria",
13       "telefone": "97777-9012"
14     }
15   ]
16 }
```

Green arrows in the image point to the GET method, the URL, the selected JSON response format, and the 'contatos' array in the response body.

Implementando a criação de um contato usando @app.post

```
from flask import Flask, request
```

```
@app.post('/contato')
def adicionar_contato():
    dados = request.get_json()

    if not dados or 'nome' not in dados or 'telefone' not in dados:
        return {'erro': "Campos 'nome' e 'telefone' são obrigatórios."}, HTTPStatus.BAD_REQUEST

    nome = dados['nome'].lower()
    telefone = dados['telefone']

    if nome in agenda:
        return {'erro': f"Contato '{nome}' já existe."}, HTTPStatus.CONFLICT

    agenda[nome] = telefone
    return {
        'mensagem': f"Contato '{nome.capitalize()}' adicionado com sucesso.",
        'nome': nome.capitalize(),
        'telefone': telefone
    }, HTTPStatus.CREATED
```

Implementando a criação de um contato usando @app.post

```
@app.post('/contato')
def adicionar_contato():
    dados = request.get_json()

    if not dados or 'nome' not in dados or 'telefone' not in dados:
        return {'erro': "Campos 'nome' e"}

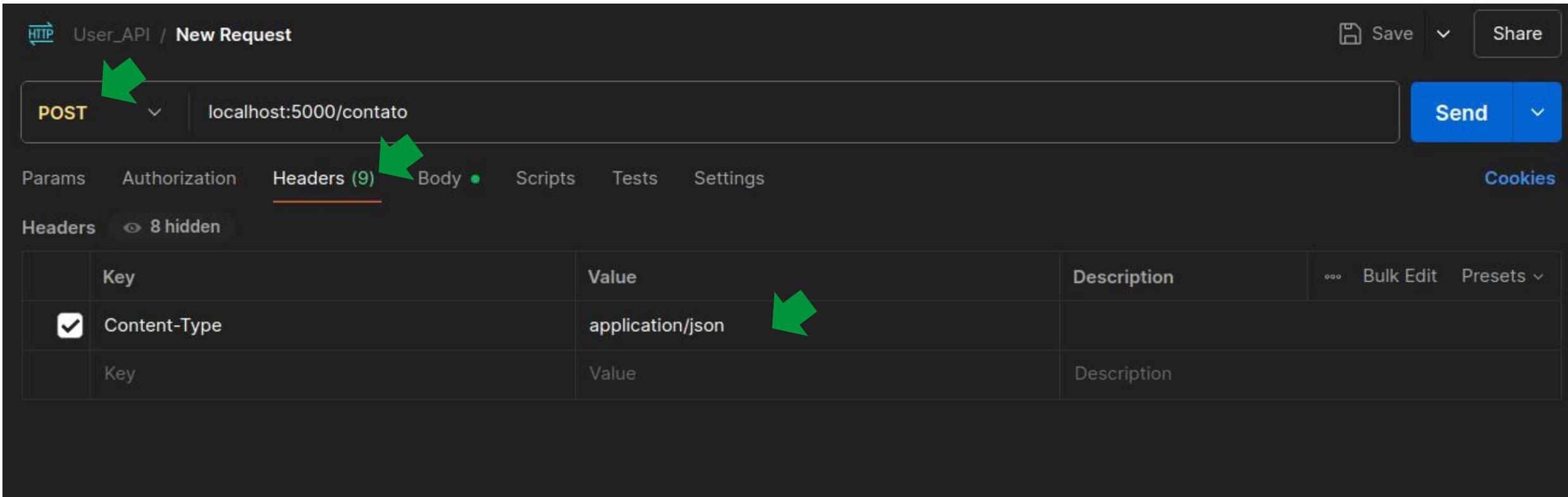
    nome = dados['nome'].lower()
    telefone = dados['telefone']

    if nome in agenda:
        return {'erro': f"Contato '{nome}' já existe."}, HTTPStatus.CONFLICT

    agenda[nome] = telefone
    return {
        'mensagem': f"Contato '{nome.capitalize()}' adicionado com sucesso.",
        'nome': nome.capitalize(),
        'telefone': telefone
    }, HTTPStatus.CREATED
```

No Flask, a função `request.get_json()` é usada para acessar os dados enviados no corpo da requisição HTTP quando o formato é JSON (geralmente usado em POST, PUT e PATCH).

Implementando a criação de um contato usando @app.post



User_API / New Request

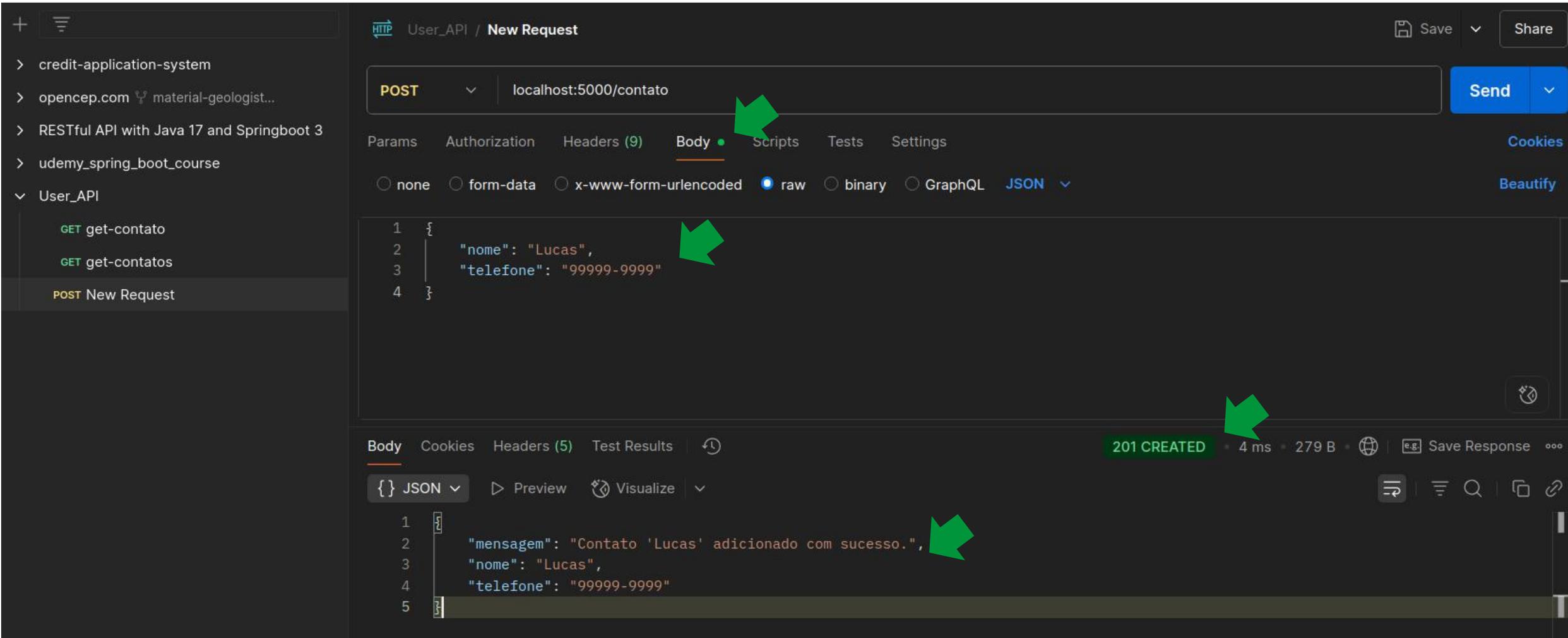
POST localhost:5000/contato

Params Authorization Headers (9) Body Scripts Tests Settings Cookies

Headers 8 hidden

	Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Content-Type	application/json				
	Key	Value	Description			

Implementando a criação de um contato usando @app.post



The screenshot displays the Postman interface for a new request. The request is a POST to `localhost:5000/contato`. The body is raw JSON: `{ "nome": "Lucas", "telefone": "99999-9999" }`. The response is `201 CREATED` with a message: `"mensagem": "Contato 'Lucas' adicionado com sucesso.", "nome": "Lucas", "telefone": "99999-9999"`.

```
POST localhost:5000/contato
```

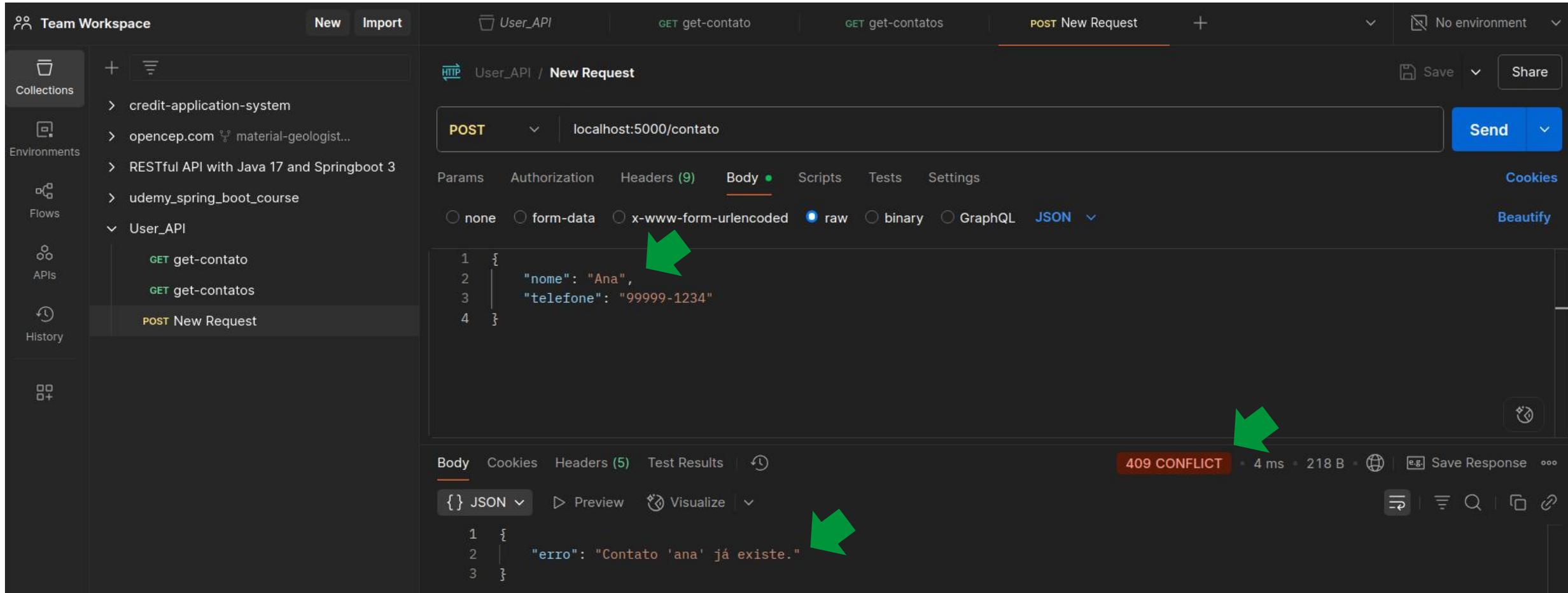
Body (raw)

```
1 {
2   "nome": "Lucas",
3   "telefone": "99999-9999"
4 }
```

Response (JSON)

```
1 {}
2 "mensagem": "Contato 'Lucas' adicionado com sucesso.",
3 "nome": "Lucas",
4 "telefone": "99999-9999"
5 {}
```

Implementando a criação de um contato usando @app.post



The screenshot displays the Postman interface for a REST client. The workspace is named "Team Workspace" and contains a collection named "User_API". The selected request is a POST method to "localhost:5000/contato". The request body is set to "raw" JSON format with the following content:

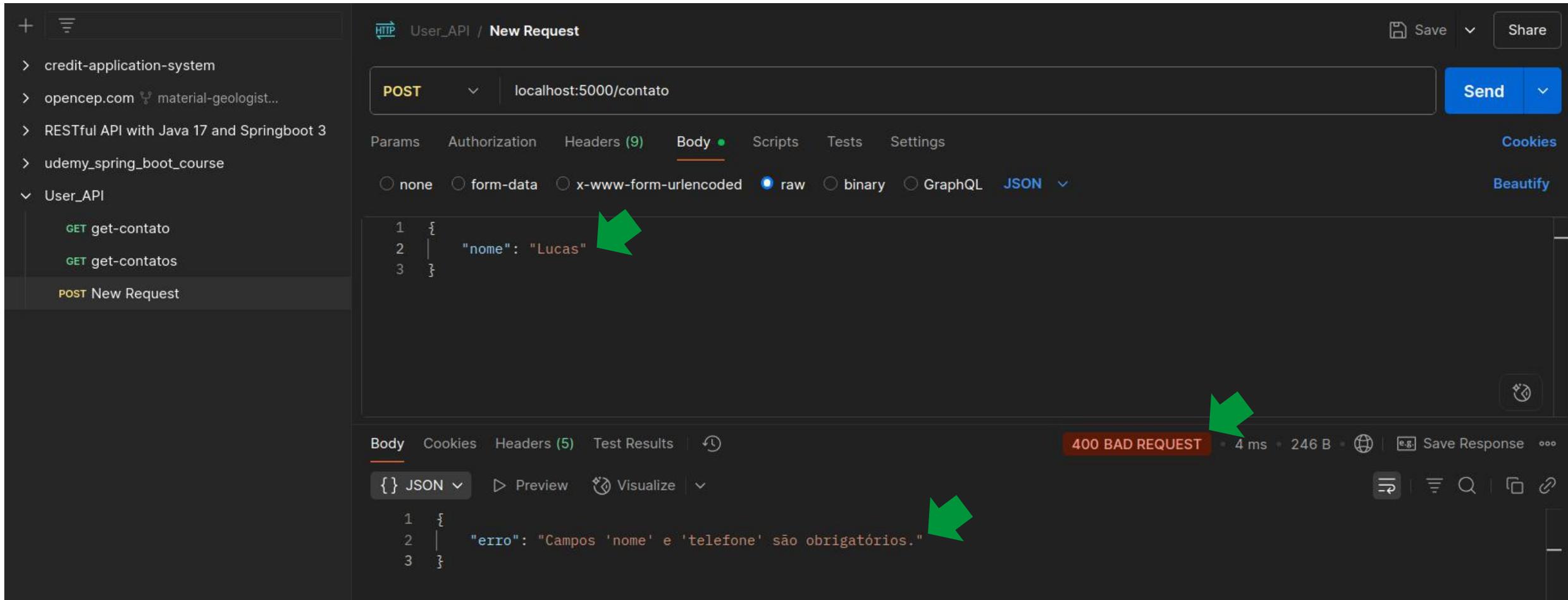
```
1 {  
2   "nome": "Ana",  
3   "telefone": "99999-1234"  
4 }
```

The response shows a "409 CONFLICT" status with a response time of 4 ms and a size of 218 B. The response body is:

```
1 {  
2   "erro": "Contato 'ana' já existe."  
3 }
```

Green arrows in the image point to the request body, the response status, and the error message in the response body.

Implementando a criação de um contato usando @app.post



The screenshot displays the Postman interface for a new request. The request is a POST to `localhost:5000/contato` with a JSON body: `{ "nome": "Lucas" }`. The response is a `400 BAD REQUEST` with a message: `"erro": "Campos 'nome' e 'telefone' são obrigatórios."`. Green arrows highlight the request body, the error status, and the error message.

```
1 {
2   "nome": "Lucas"
3 }
```

400 BAD REQUEST • 4 ms • 246 B • Save Response

```
1 {
2   "erro": "Campos 'nome' e 'telefone' são obrigatórios."
3 }
```

curl (Client URL)

- curl (Client URL) é uma ferramenta de linha de comando usada para fazer requisições HTTP (e também outros protocolos como FTP, SMTP, etc.) a servidores.

```
lucas@lucas-Inspiron-15-3520: ~  
lucas@lucas-Inspiron-15-3520:~$ curl -X POST http://localhost:5000/contato \  
-H "Content-Type: application/json" \  
-d '{"nome": "ana", "telefone": "8888-0000"}'  
{  
  "mensagem": "Contato 'Ana' adicionado com sucesso.",  
  "nome": "Ana",  
  "telefone": "8888-0000"  
}  
lucas@lucas-Inspiron-15-3520:~$
```

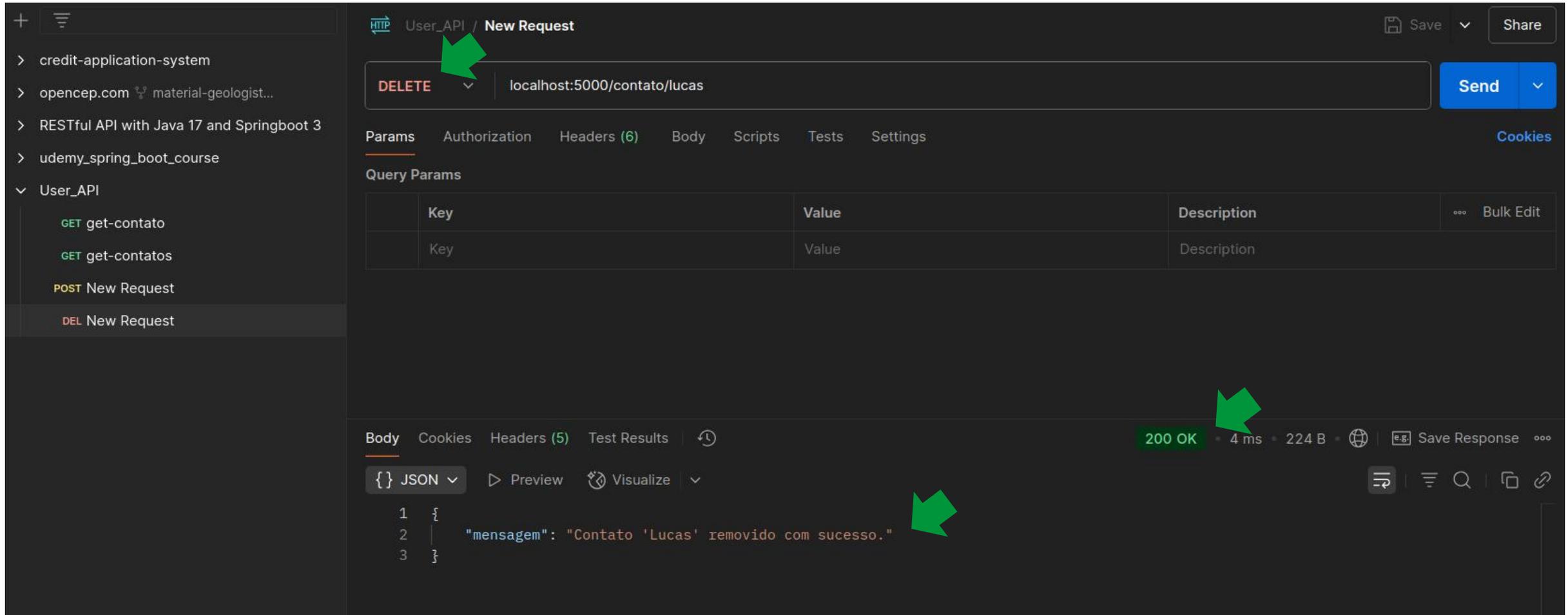
Removendo um contato usando @app.delete



```
@app.delete('/contato/<nome>')
def deletar_contato(nome):
    nome = nome.lower()

    if nome in agenda:
        del agenda[nome]
        resposta = {
            'mensagem': f"Contato '{nome.capitalize()}' removido com sucesso."
        }
        return resposta, HTTPStatus.OK
    else:
        erro = {
            'erro': f"Contato '{nome}' não encontrado."
        }
        return erro, HTTPStatus.NOT_FOUND
```

Removendo um contato usando @app.delete



The screenshot shows the Postman interface for a REST client. On the left sidebar, the 'User_API' folder is expanded, showing a 'DEL New Request' item. The main workspace displays a 'New Request' for the 'User_API' folder. The HTTP method is set to 'DELETE' and the URL is 'localhost:5000/contato/lucas'. The 'Send' button is visible. Below the URL bar, the 'Query Params' section is empty. The 'Body' tab is selected, showing a JSON response:

```
{  "mensagem": "Contato 'Lucas' removido com sucesso."}
```

 The status bar at the bottom indicates a '200 OK' response with a response time of 4 ms and a size of 224 B. Three green arrows point to the 'DELETE' method, the '200 OK' status, and the JSON response.

HTTP User_API / New Request

DELETE localhost:5000/contato/lucas

Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

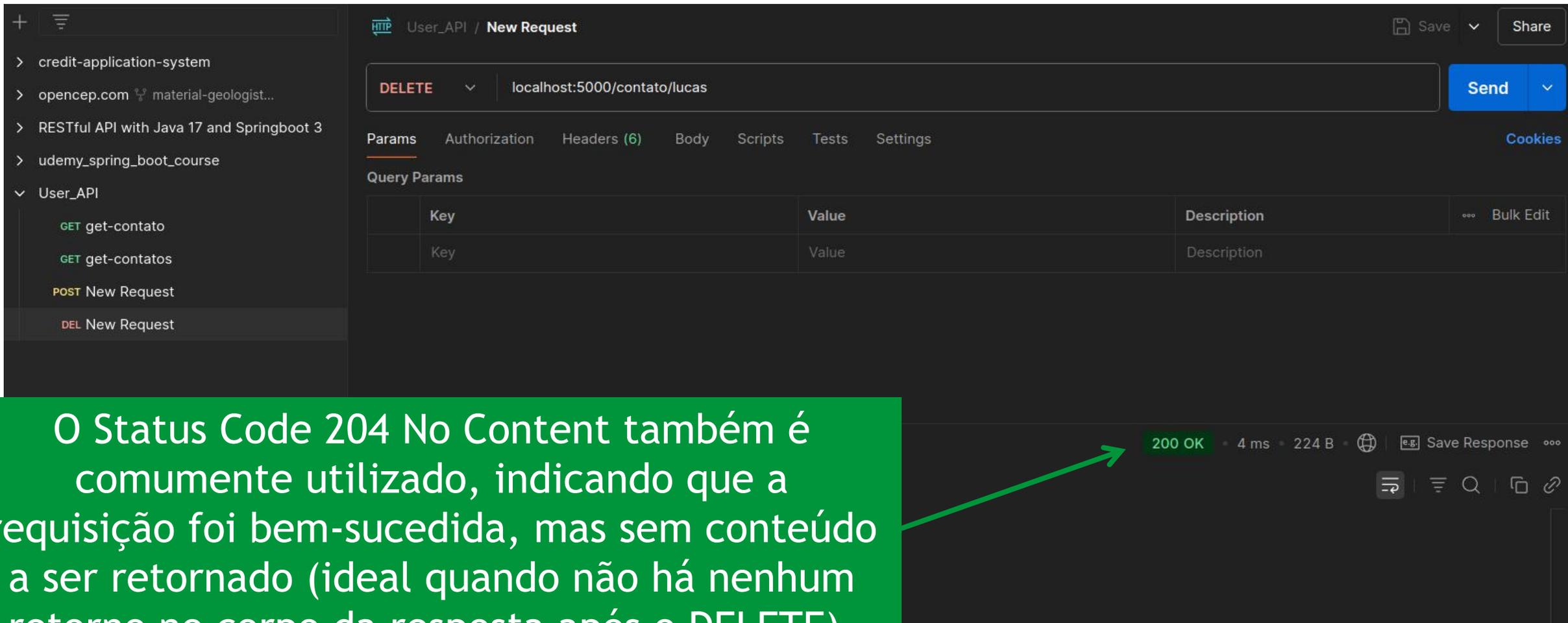
Body Cookies Headers (5) Test Results

200 OK • 4 ms • 224 B • Save Response

{ } JSON Preview Visualize

```
1 {
2   "mensagem": "Contato 'Lucas' removido com sucesso."
3 }
```

Removendo um contato usando @app.delete

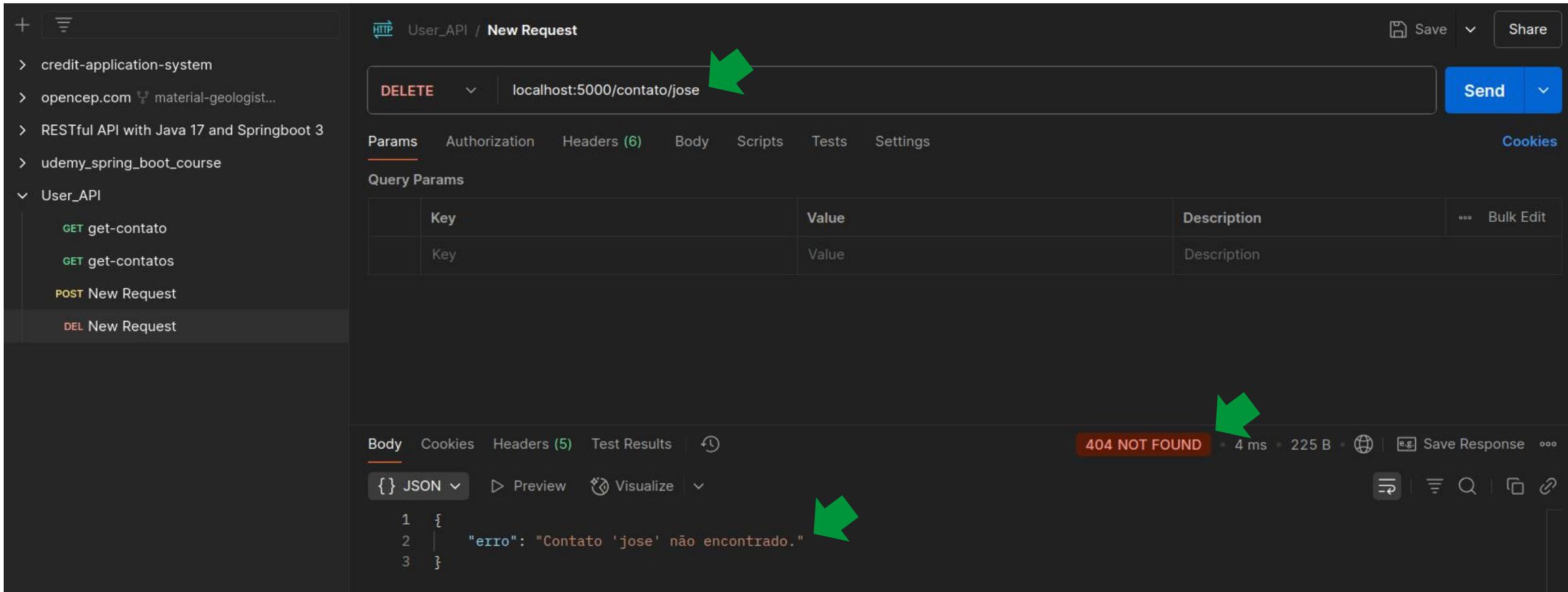


The screenshot shows a REST client interface with a sidebar on the left containing a tree view of collections. The main area displays a 'New Request' for a DELETE method to the URL 'localhost:5000/contato/lucas'. Below the URL bar are tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Scripts', 'Tests', and 'Settings'. The 'Query Params' section is currently empty. At the bottom, the response status is shown as '200 OK' with a response time of '4 ms' and a size of '224 B'. A green arrow points from the text box to the '200 OK' status.

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

O Status Code 204 No Content também é comumente utilizado, indicando que a requisição foi bem-sucedida, mas sem conteúdo a ser retornado (ideal quando não há nenhum retorno no corpo da resposta após o DELETE).

Removendo um contato usando @app.delete



The screenshot shows the Postman interface for a new request. The request method is **DELETE** and the URL is `localhost:5000/contato/jose`. The response status is **404 NOT FOUND** with a response time of 4 ms and a size of 225 B. The response body is a JSON object:

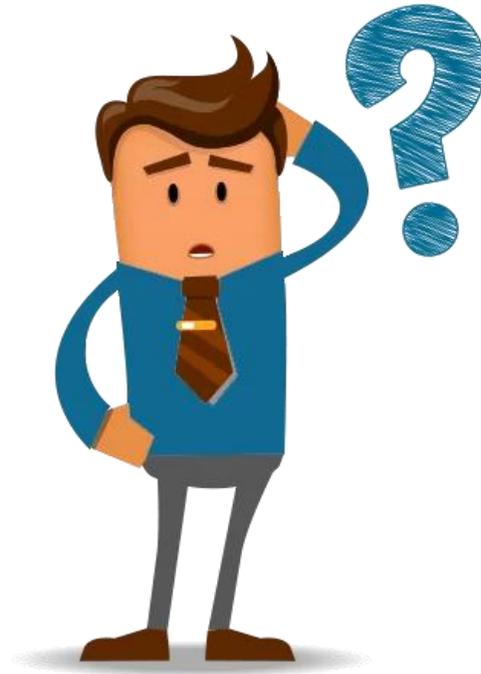
```
{
  "erro": "Contato 'jose' não encontrado."
}
```

Green arrows in the image point to the URL, the 404 status, and the error message in the JSON response.

Exercícios

- Implemente duas rotas de atualização:
 - Uma rota usando `@app.put` para realizar a atualização total de um contato, substituindo completamente os dados (nome e telefone).
 - Outra rota usando `@app.patch` para realizar a atualização parcial, permitindo alterar apenas o telefone de um contato existente.
 - Utilize `request.get_json()` para acessar os dados enviados no corpo da requisição. Lembre-se de tratar casos de erro, como dados ausentes ou contato inexistente.

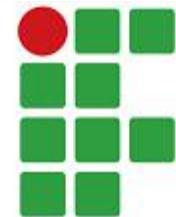
Dúvidas



PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática

Lucas Sampaio Leite



**INSTITUTO
FEDERAL**

Baiano