

PROGRAMAÇÃO WEB II

Curso Técnico Integrado em Informática
Lucas Sampaio Leite



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
```

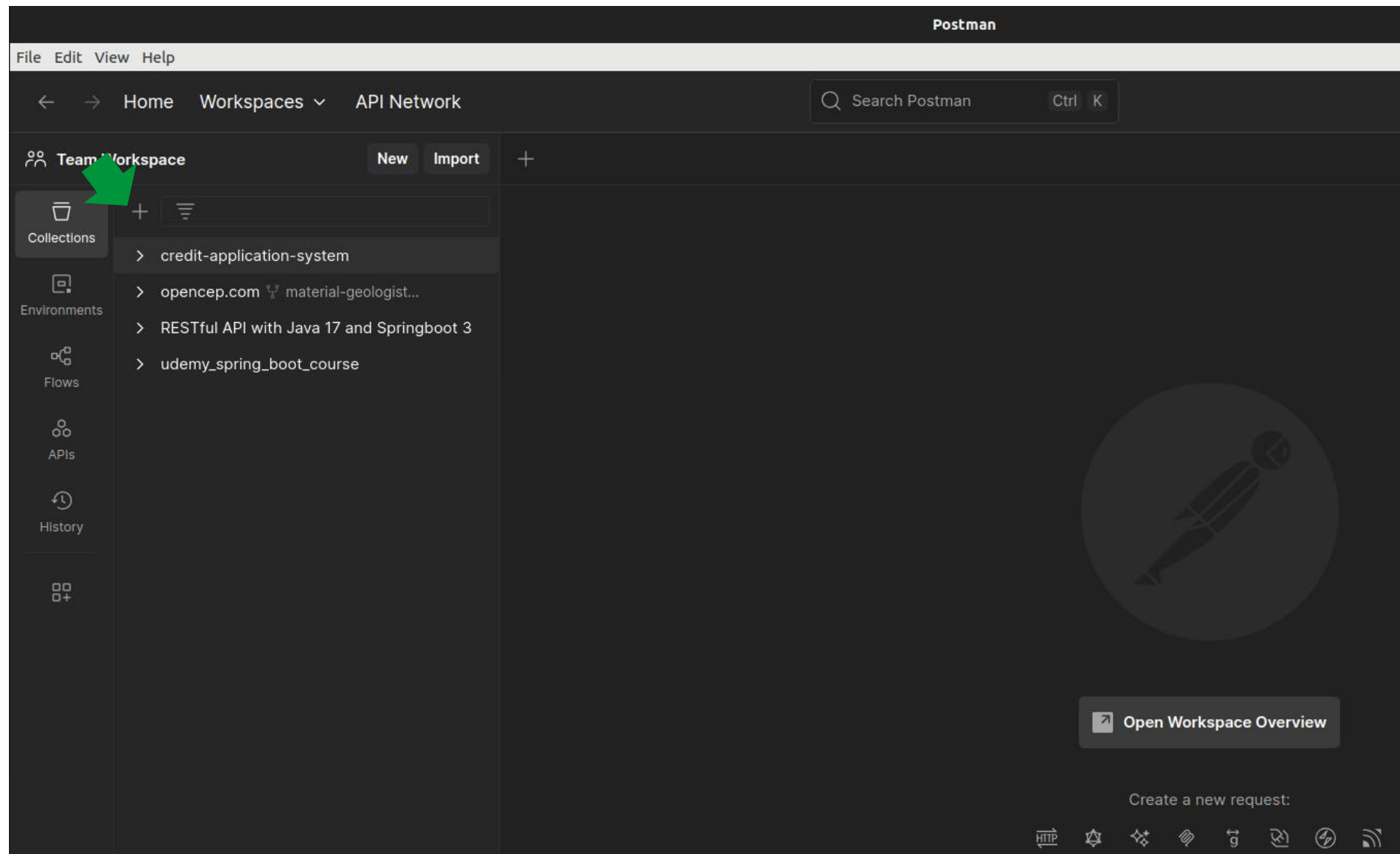


Postman

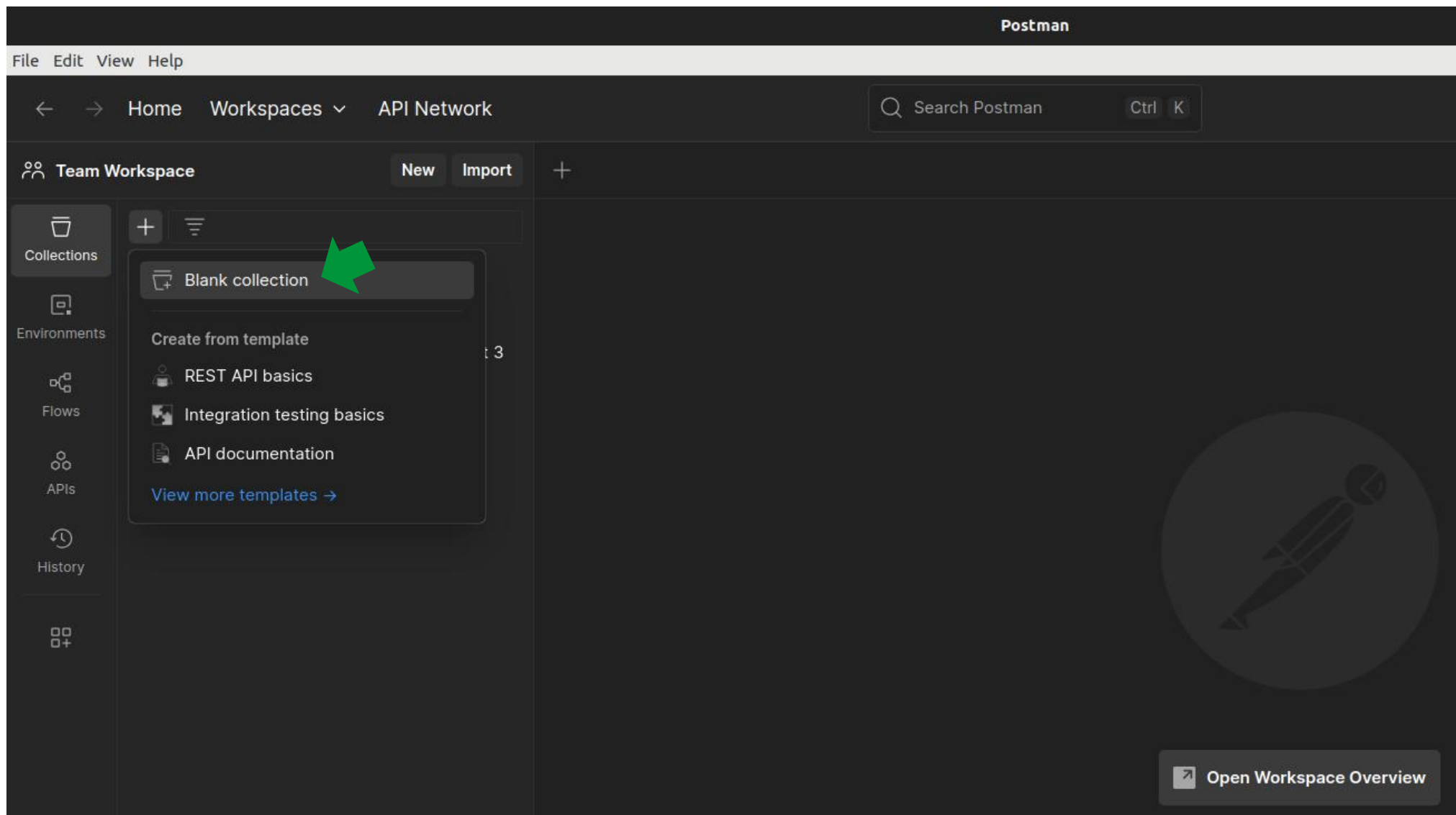
- O Postman é uma ferramenta popular usada para testar APIs RESTful de forma simples e eficiente.
- Com o Postman, é possível enviar requisições HTTP (GET, POST, PUT, DELETE, etc.) e visualizar as respostas da API sem precisar criar um frontend ou um cliente separado.
- Alternativas ao Postman: Insomnia,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



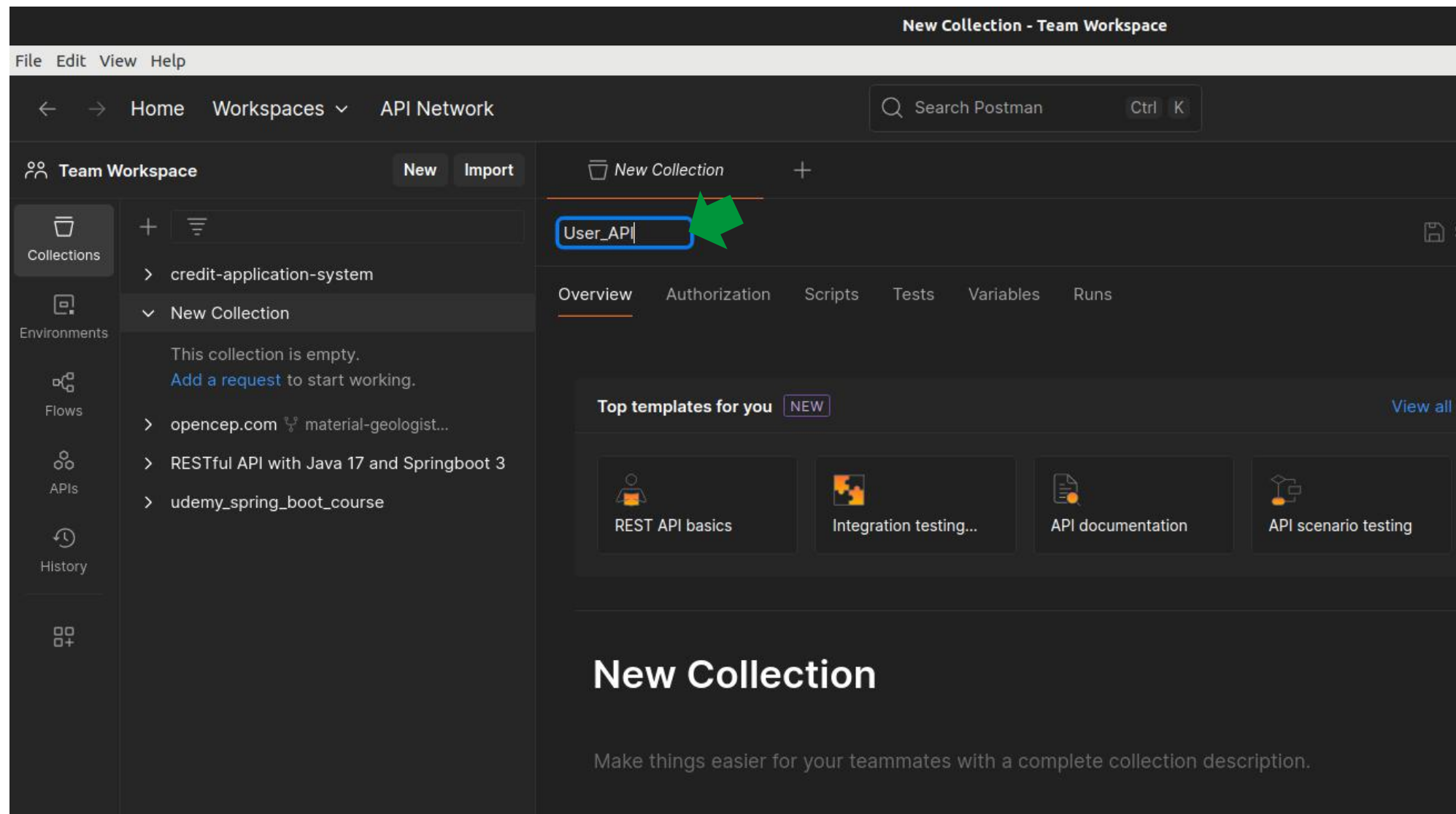
Postman



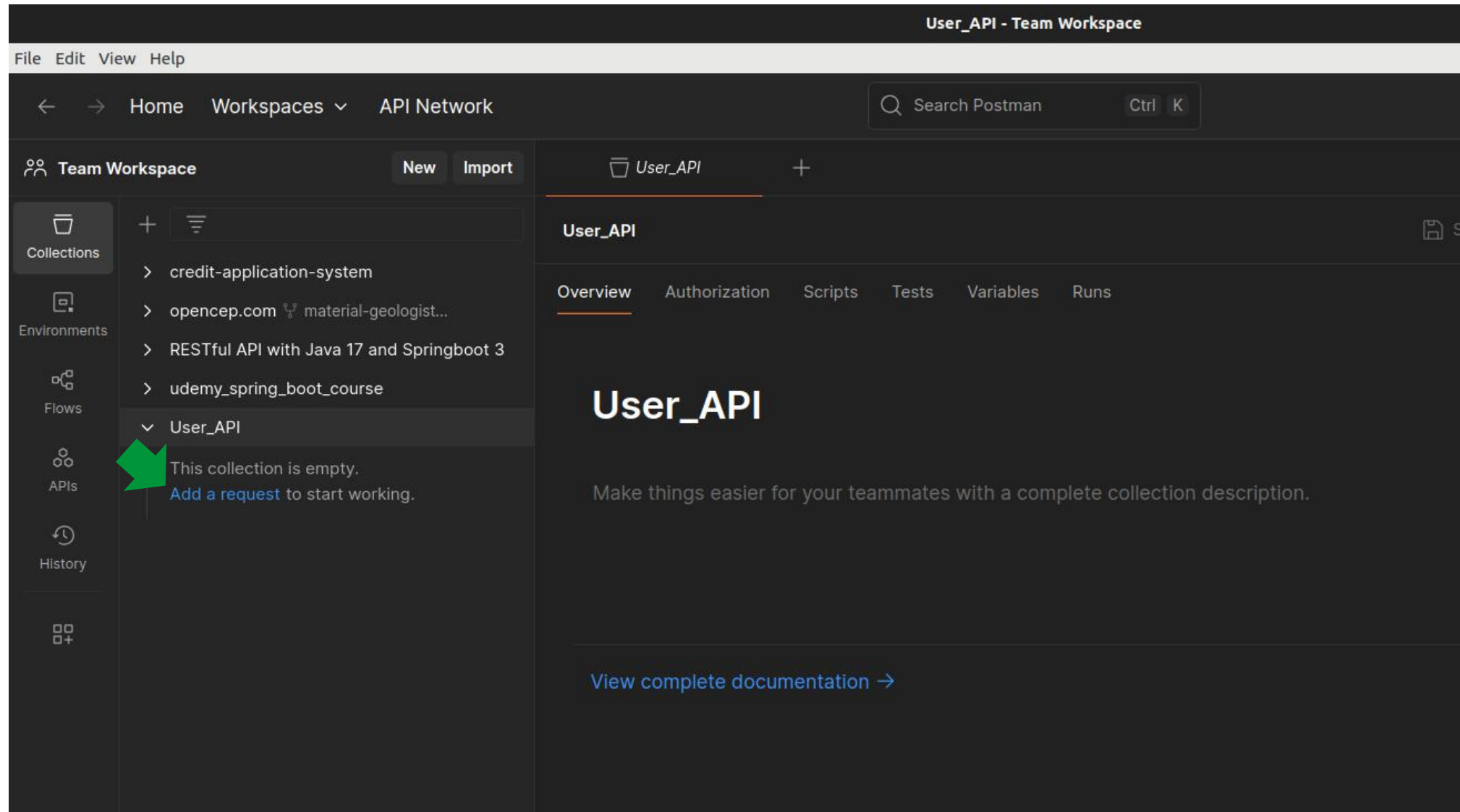
Postman



Postman



Postman



Postman

get-contato - Team Workspace

File Edit View Help

← → Home Workspaces ▾ API Network

Search Postman Ctrl K

Team Workspace New Import

Collections

- credit-application-system
- opencep.com material-geologist...
- RESTful API with Java 17 and Springboot 3
- udemy_spring_boot_course
- ▼ User_API
 - GET get-contato

Environments

Flows

APIs

History

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		

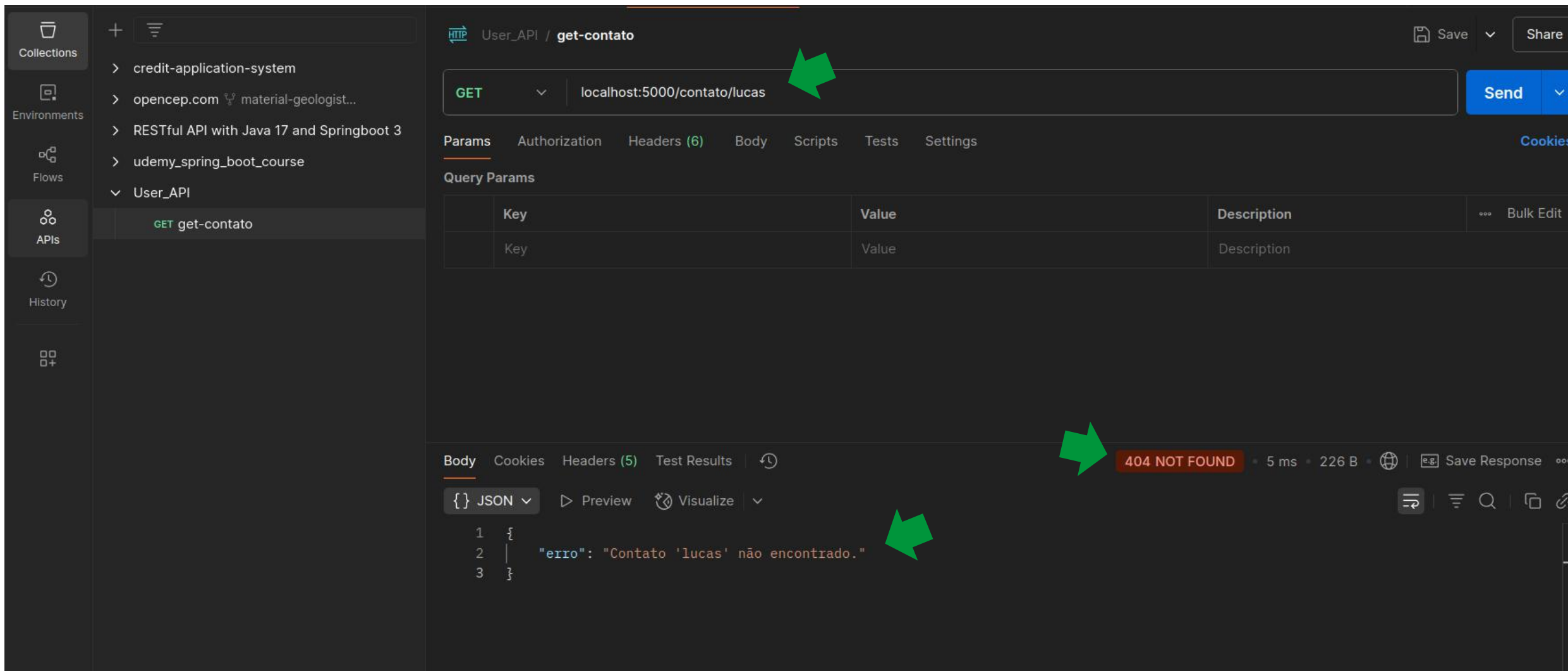
Response History ▾



Postman



Postman



The screenshot displays the Postman application interface. On the left sidebar, the 'Collections' section is expanded, showing a collection named 'User_API' with a sub-collection 'get-contato'. The main workspace shows a GET request to 'localhost:5000/contato/lucas'. The 'Params' tab is selected, showing a table with columns 'Key', 'Value', and 'Description'. The 'Body' tab is also selected, showing a JSON response:

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

. The status bar at the bottom indicates a '404 NOT FOUND' response with a 5 ms duration and 226 B size. Green arrows point to the URL, the 'Send' button, the '404 NOT FOUND' status, and the JSON response body.

HTTP GET localhost:5000/contato/lucas

Params Authorization Headers (6) Body Scripts Tests Settings

Query Params

Key	Value	Description
Key	Value	Description


Body Cookies Headers (5) Test Results

JSON Preview Visualize

```
1 {
2   "erro": "Contato 'lucas' não encontrado."
3 }
```

404 NOT FOUND • 5 ms • 226 B • Save Response

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

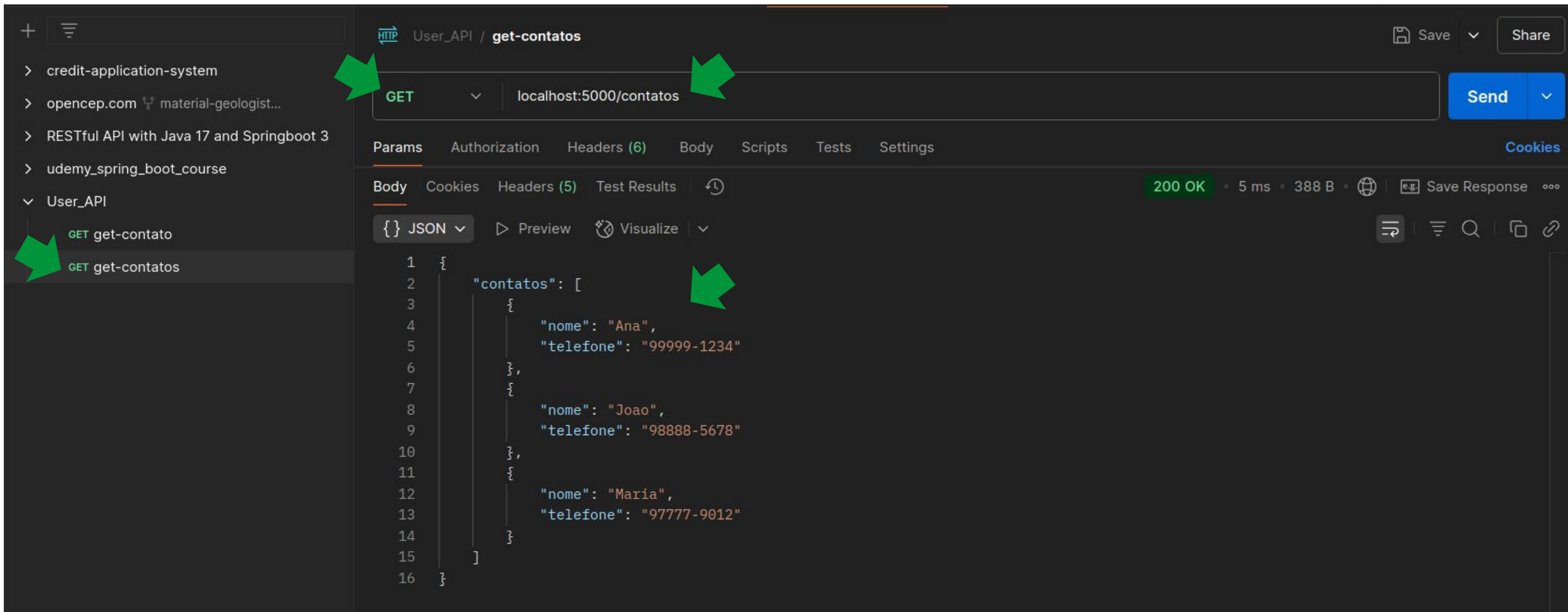


```
@app.get('/contatos')
def listar_contatos():
    if not agenda:
        resposta = {'mensagem': 'A agenda está vazia.'}
        return resposta, HTTPStatus.OK

    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



HTTP User_API / **get-contatos** Save Share

GET localhost:5000/contatos Send

Params Authorization Headers (6) Body Scripts Tests Settings Cookies

Body Cookies Headers (5) Test Results 200 OK 5 ms 388 B Save Response

{} JSON Preview Visualize

```
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 }
```

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"}

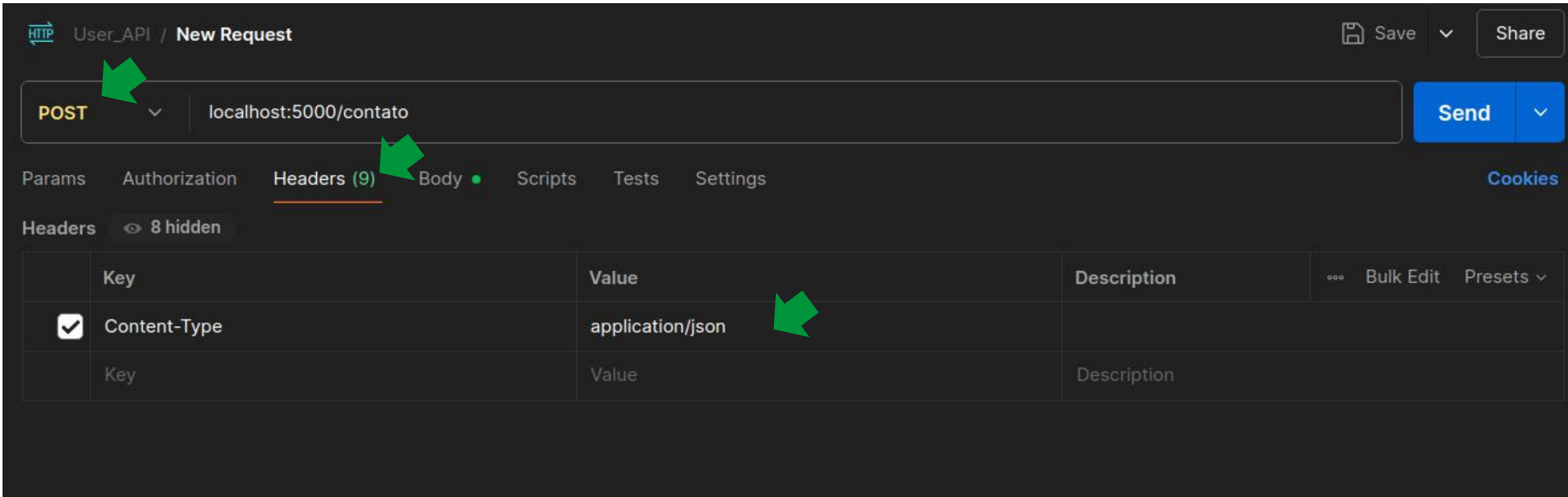
    nome = dados['nome'].lower()
    telefone = dados['telefone']

    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



The screenshot shows the Postman interface for creating a new request. The top bar indicates the request is for 'User_API' and is a 'New Request'. The method is set to 'POST' and the URL is 'localhost:5000/contato'. The 'Headers' tab is selected, showing a table with headers. The 'Content-Type' header is set to 'application/json'. The 'Body' tab is also visible, indicating 8 hidden headers.

HTTP User_API / New Request Save Share

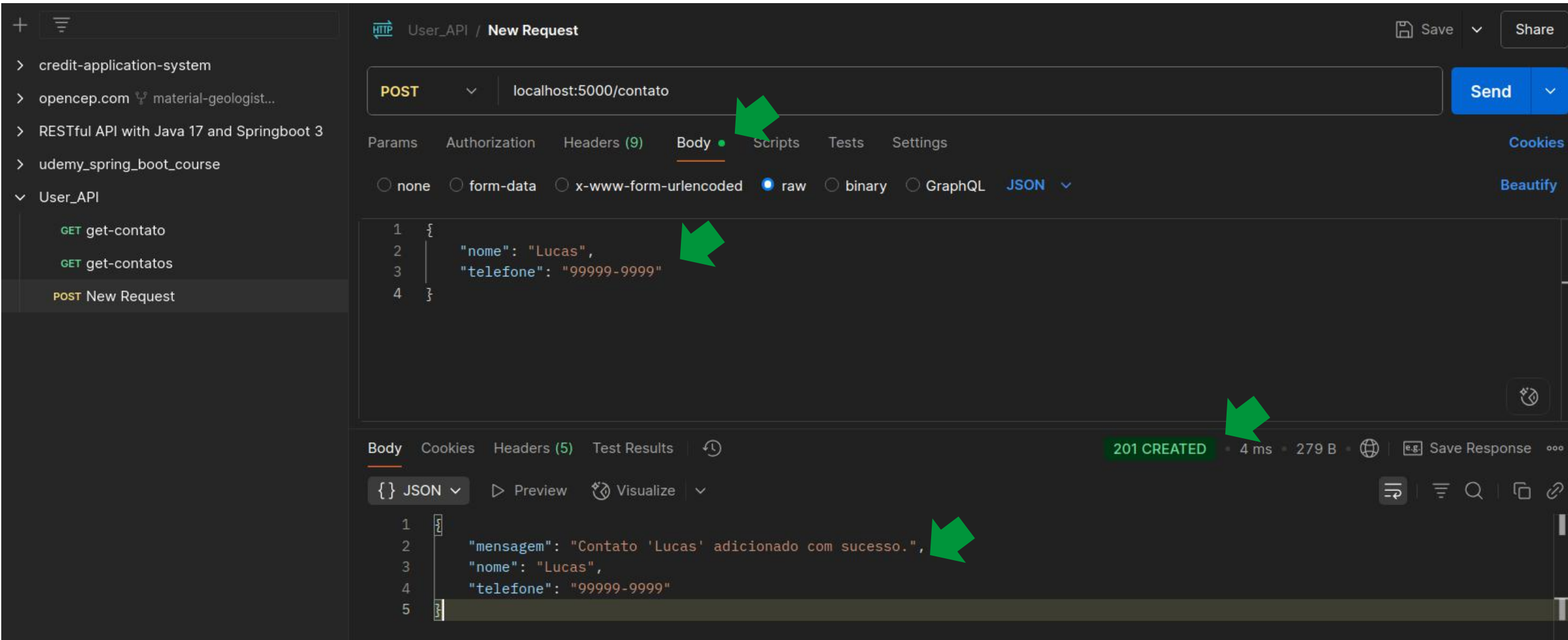
POST localhost:5000/contato Send

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 left sidebar shows a collection named 'User_API' with a 'New Request' entry selected. The main panel shows a POST request to 'localhost:5000/contato'. The 'Body' tab is active, showing a raw JSON body:

```
{  "nome": "Lucas",  "telefone": "99999-9999"}
```

. The 'Send' button is visible. Below the request, the response is shown with a status of '201 CREATED', a time of '4 ms', and a size of '279 B'. The response body is a JSON object:

```
{  "mensagem": "Contato 'Lucas' adicionado com sucesso.",  "nome": "Lucas",  "telefone": "99999-9999"}
```

. Green arrows highlight the 'Body' tab, the JSON body, the '201 CREATED' status, and the response body.

HTTP User_API / New Request

POST localhost:5000/contato

Params Authorization Headers (9) Body Scripts Tests Settings

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

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

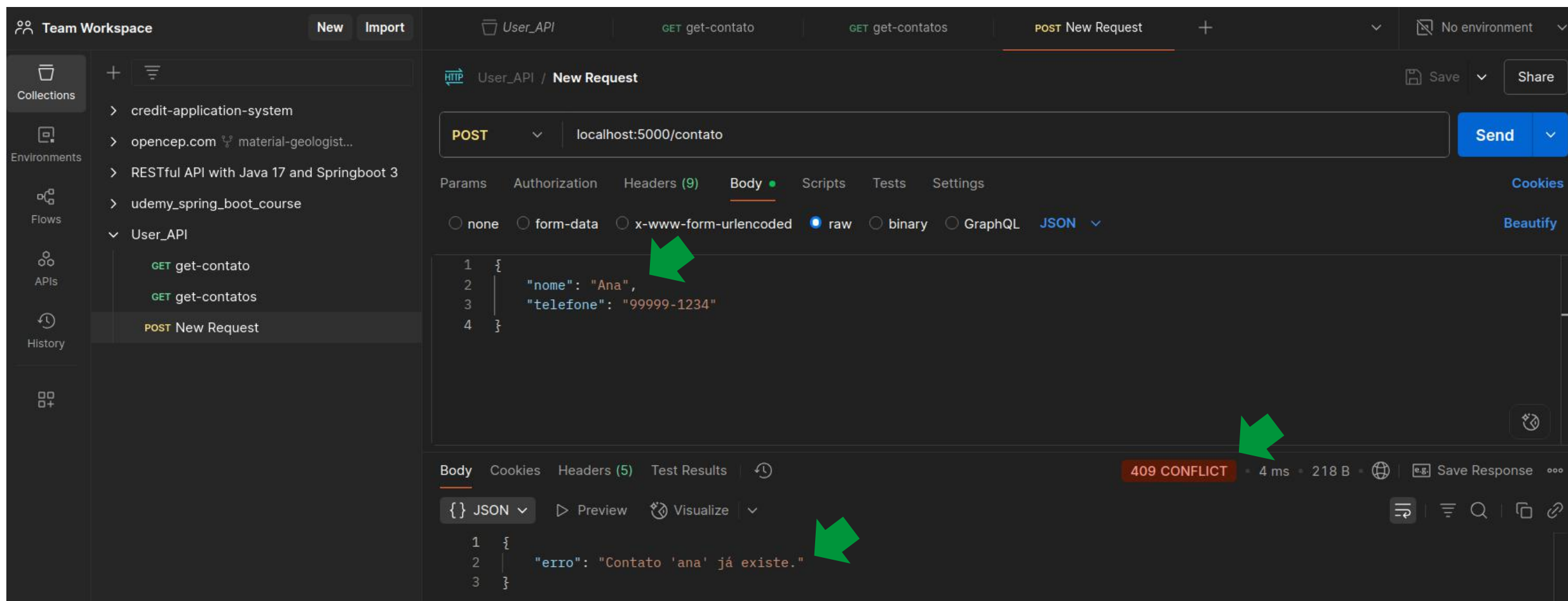
Body Cookies Headers (5) Test Results

201 CREATED • 4 ms • 279 B • Save Response

{ } JSON Preview Visualize

```
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. On the left sidebar, the 'Team Workspace' is selected, and the 'User_API' collection is expanded, showing the 'POST New Request' endpoint. The main panel shows the request configuration for 'POST localhost:5000/contato'. The 'Body' tab is active, and the request body is a JSON object:

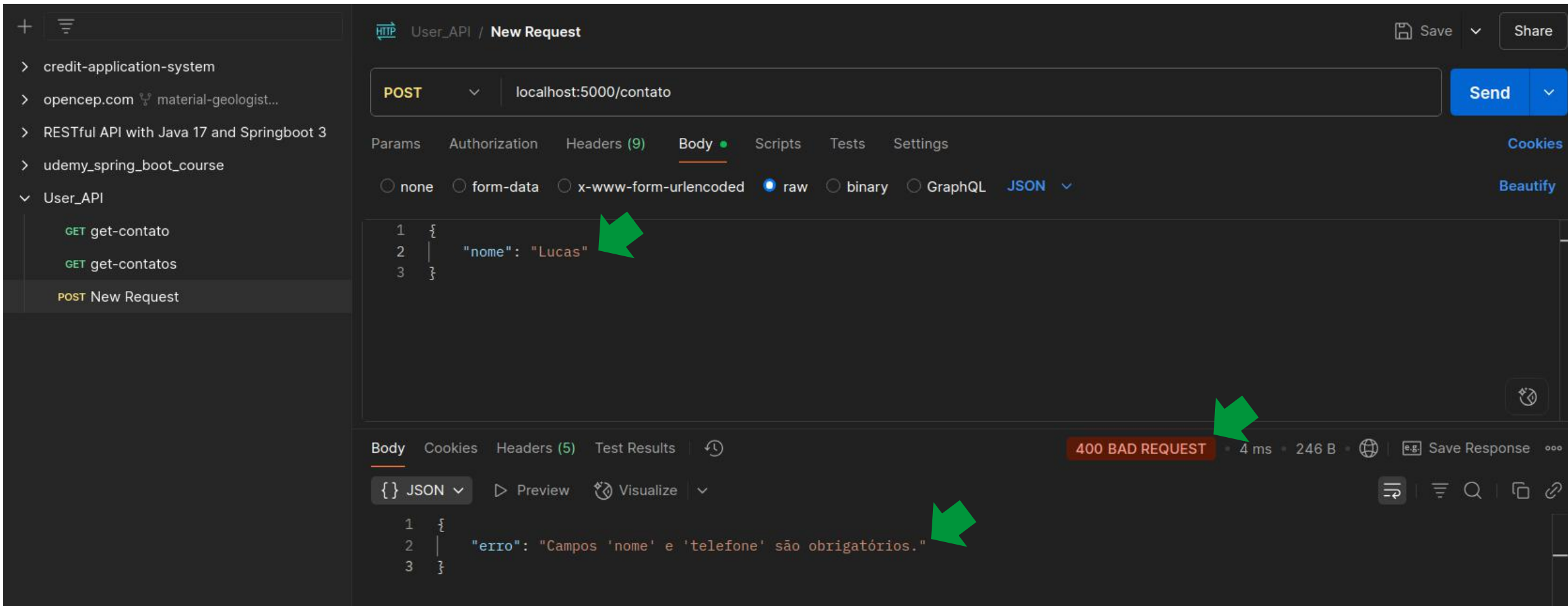
```
{  "nome": "Ana",  "telefone": "99999-1234"}
```

. The 'Send' button is visible. Below the request configuration, the response is shown with a status of '409 CONFLICT', a response time of '4 ms', and a size of '218 B'. The response body is a JSON object:

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

. Three green arrows highlight the request body, the response status, and 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 raw JSON body: `{ "nome": "Lucas" }`. The response is a 400 BAD REQUEST, indicating a validation error: `"erro": "Campos 'nome' e 'telefone' são obrigatórios."`. Green arrows highlight the request body and the error message.

Request Details:

- Method: POST
- URL: localhost:5000/contato
- Body Type: raw (JSON)
- Body:

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

Response Details:

- Status: 400 BAD REQUEST
- Time: 4 ms
- Size: 246 B
- Body:


```
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:~$
```

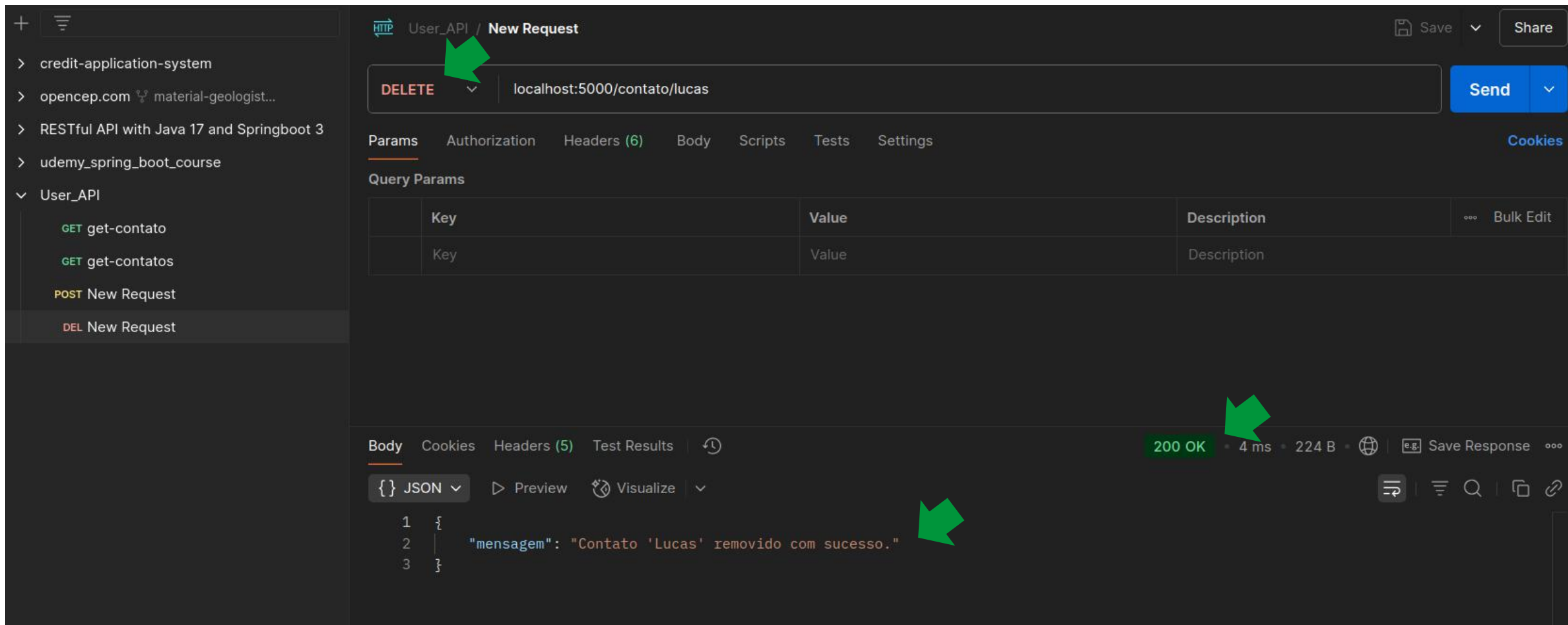

Removendo um contato usando @app.delete



```
@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
```

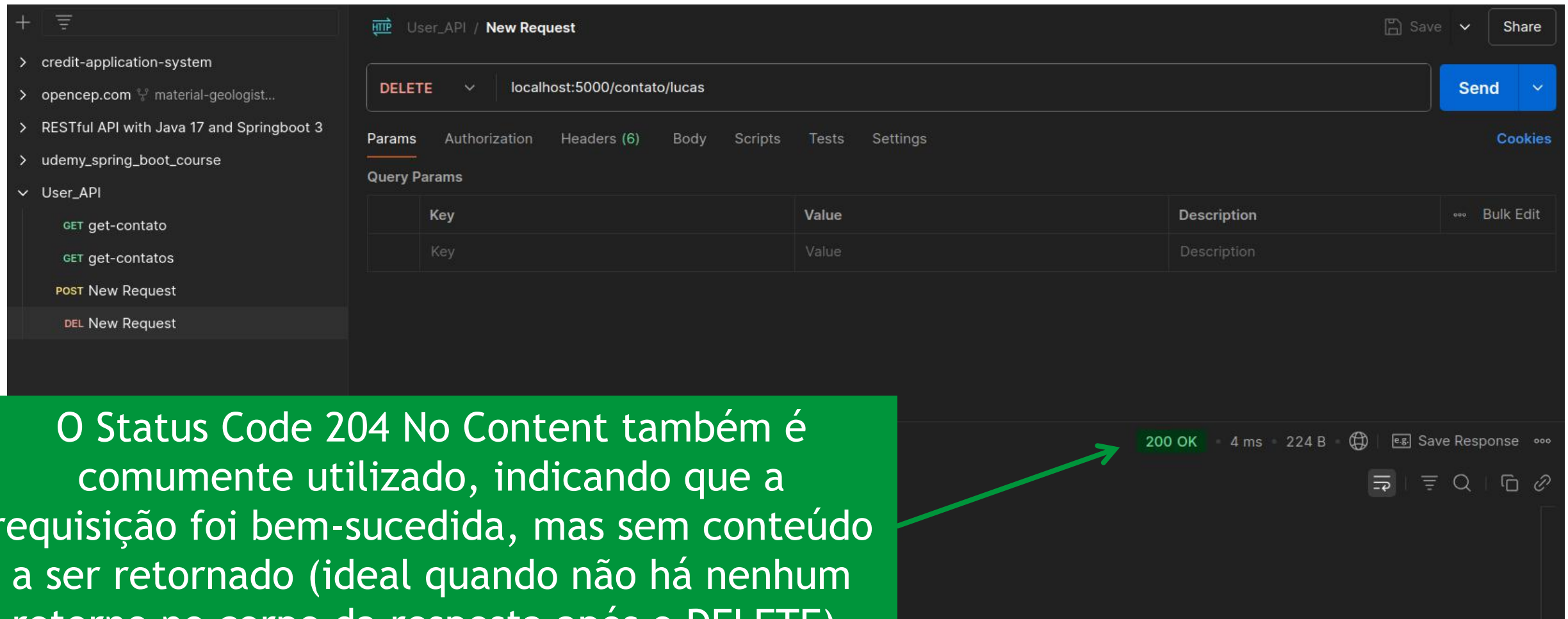
Removendo um contato usando @app.delete



The screenshot shows the Postman interface for a new request. The left sidebar lists the project structure, including a folder named 'User_API' with a 'DEL New Request' item selected. The main panel shows the 'New Request' tab for 'User_API' with the method 'DELETE' and the URL 'localhost:5000/contato/lucas'. The 'Send' button is visible. Below the URL bar, the 'Params' tab is active, showing a table for 'Query Params' with columns 'Key', 'Value', and 'Description'. The 'Body' tab is also visible, showing a JSON response. The response status is '200 OK' with a response time of '4 ms' and a size of '224 B'. The JSON body is displayed as follows:

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

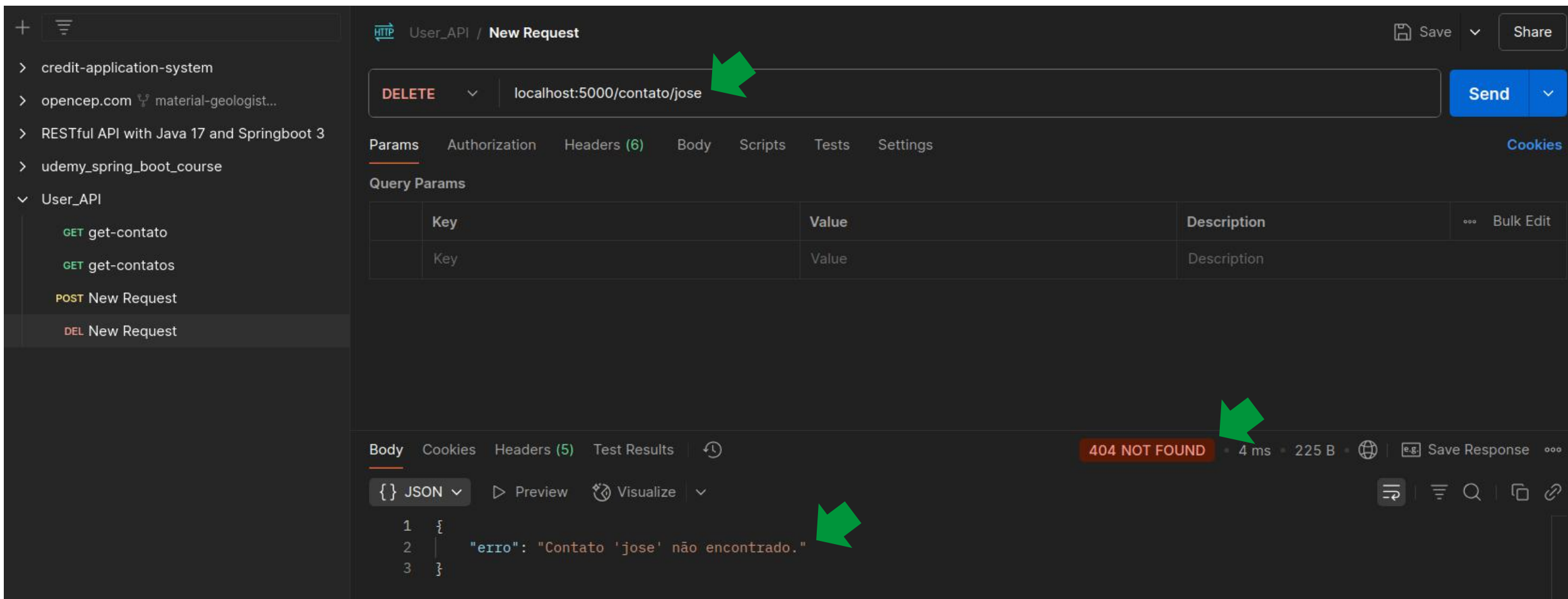
Removendo um contato usando @app.delete



The screenshot shows the REST Client interface. On the left, a sidebar lists various API collections, with 'User_API' expanded. The main area displays a 'New Request' for a DELETE method to 'localhost:5000/contato/lucas'. Below the URL bar, tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Scripts', 'Tests', and 'Settings' are visible. The 'Params' tab is active, showing a table for 'Query Params' with columns 'Key', 'Value', and 'Description'. The response area at the bottom shows a status of '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.

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 URL bar shows `DELETE localhost:5000/contato/jose`. The response status is **404 NOT FOUND**. The response body is JSON, showing an error message: `"erro": "Contato 'jose' não encontrado."`.

Request Details:

- Method: DELETE
- URL: localhost:5000/contato/jose
- Params: Query Params table with columns Key, Value, and Description.

Response Details:

- Status: 404 NOT FOUND
- Time: 4 ms
- Size: 225 B
- Body: JSON

Key	Value	Description
Key	Value	Description

```
1 {
2   "erro": "Contato 'jose' não encontrado."
3 }
```

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

