

# Postman

# Postman

- É uma plataforma independente de software para testar APIs.
- **Site:** <https://www.postman.com/downloads/>
- **Obs:** Ao invés de baixar e instalar o Postman, você também pode instalar sua extensão no Visual Studio Code.

Overview | PIS | GET hello | + | No environment

HTTP PIS / hello | Save | Share | [Link](#)

**GET** | http://127.0.0.1:8000/ | Send

Params Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description	...	

Body Cookies Headers (4) Test Results | 200 OK | 14 ms | 150 B | Save Response | ...

{ } JSON | Preview | Visualize |

```
1 {  
2   "message": "Hello World"  
3 }
```

Copy | Print | Find | [Link](#)

# Criando a rota Mensagens (Post)

```
from fastapi import FastAPI
from fastapi.params import Body

app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello World"}
```

  

```
@app.post("/mensagens")
def criar_mensagem(res: dict = Body(...)):
    print(res)
    return {"Mensagem": "Criado com Sucesso!"}
```

# Postman

The screenshot shows the Postman application interface. At the top, there is a header bar with a 'POST' button (highlighted with a red box), a URL input field containing 'http://127.0.0.1:8000/mensagens' (also highlighted with a red box), and a 'Send' button. Below the header, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body' (which has a green dot indicating it's selected and highlighted with a red box), 'Scripts', and 'Settings'. To the right of these tabs are 'Cookies' and 'Schema' buttons. Under the 'Body' tab, there are several radio buttons for selecting the data type: 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected and highlighted with a red box), 'binary', and 'GraphQL'. A dropdown menu next to 'GraphQL' shows 'JSON' is selected. The main body area contains a code editor with the following JSON payload:

```
1 {  
2   "conteudo": "Olá, mundo!",  
3   "autor": "Caio"  
4 }
```

# Terminal

```
{'conteudo': 'Olá, mundo!', 'autor': 'Caio'}
```

# Postman

POST  Send

Params Authorization Headers (8) **Body** Scripts Settings Cookies

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL **JSON**  Schema  Beautify

```
1 {  
2   "conteudo": "Olá, mundo!",  
3   "autor": "Caio"  
4 }
```

Body Cookies Headers (4) Test Results

200 OK • 9 ms • 159 B •   e.g. Save Response

{ } JSON

```
1 {  
2   "Mensagem": "Criado com Sucesso!"  
3 }
```

# Criando a rota Mensagens (Post)

```
from fastapi import FastAPI  
from fastapi.params import Body
```

Body é uma classe do FastAPI que indica que o valor de um parâmetro virá do corpo da requisição **HTTP** (body da requisição). É usado principalmente em rotas **POST** ou **PUT**, quando você quer enviar dados em **JSON**.

```
app = FastAPI()
```

```
@app.get("/")  
async def root():  
    return {"message": "Hello World"}
```

Define uma rota **POST** no caminho /mensagens. **POST** é usado para criar um novo recurso.

```
@app.post("/mensagens")  
def criar_mensagem(res: dict = Body(...)):  
    print(res)  
    return {"Mensagem": "Criado com Sucesso!"}
```

- Define que a função espera um dicionário **JSON** no corpo da requisição.
- ... significa obrigatório (não pode ser vazio).
- **res** será um dicionário Python contendo os dados enviados pelo cliente.

# 1) O que vai acontecer?

The screenshot shows a POST request in the Postman application. The URL is set to `http://127.0.0.1:8000/mensagens`. The 'Body' tab is selected, showing the following configuration:

- Content Type: `raw` (selected)
- Format: `JSON` (selected)
- Body content:

```
1 Ctrl+Alt+P to Ask AI
```

Other tabs visible include Params, Authorization, Headers (7), Scripts, Settings, Cookies, Schema, and Beautify.

POST



http://127.0.0.1:8000/mensagens

Send



Params

Authorization

Headers (7)

Body

Scripts

Settings

Cookies

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

JSON

 Schema

Beautify

1 Ctrl+Alt+P to Ask AI

Body

Cookies

Headers (4)

Test Results



422 Unprocessable Content



5 ms



226 B



e.g. Save Response



{ } JSON

▼ Preview

Debug with AI



```
1 {  
2     "detail": [  
3         {  
4             "type": "missing",  
5             "loc": [  
6                 "body"  
7             ],  
8             "msg": "Field required",  
9             "input": null  
10        }  
11    ]  
12 }
```

## 2) O que vai acontecer?

The screenshot shows a POST request in the Postman application. The URL is `http://127.0.0.1:8000/mensagens`. The **Body** tab is selected, showing the following JSON input:

```
1 {  
2   1:  
3     }  
4 }
```

The JSON input is highlighted with a red box. The **raw** option is selected under the Body type dropdown.

POST  http://127.0.0.1:8000/mensagens

Params    Authorization    Headers (8)    **Body**  Scripts    Settings    Cookies

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

```
1 {  
2  
3 }
```

Body    Cookies    Headers (4)    Test Results

**200 OK** • 6 ms • 159 B •  e.g. Save Response

{ } **JSON**   Preview

```
1 {  
2     "Mensaje": "Criado com Sucesso!"  
3 }
```

**{ } vazio ainda é válido!!!**

Qual o problema deste código?

POST

▼

<http://127.0.0.1:8000/mensagens>

Send

▼

Params

Authorization

Headers (8)

Body

●

Scripts

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON ▾

Schema

Beautify

```
1 {  
2   "conteudo": "Olá, mundo!",  
3   "autor": "Caio",  
4   "lala": "lala",  
5   "lele": "lele",  
6   "lili": "lili"  
7 }
```

Body

Cookies

Headers (4)

Test Results



200 OK

• 9 ms

• 159 B



|



Save Response

...

{}

JSON ▾

▷ Preview

Visualize

▼

```
1 {  
2   "Mensagem": "Criado com Sucesso!"  
3 }
```



**Eu posso enviar qualquer coisa!**

**Temos que validar os dados que os clientes (neste caso o Postman) enviam!**