Flask

AULA 02

ROTEAMENTO COM PARÂMETROS

O QUE VEREMOS HOJE

01 ROTAS E ENDPOINTS

02 TIPOS DE ROTAS

03 PARÂMETROS

04 PATH PARAMS

05 QUERY PARAMS

06 MÉTODO POST

ROTAS E ENDPOINTS

No contexto de APIs, o termo **rota** refere-se ao caminho definido na URL que leva a uma função no back-end. Esse caminho pode ser fixo ou conter partes dinâmicas.

Já o endpoint é a combinação de uma rota com um método HTTP específico, como GET, POST, entre outros. Por exemplo:

- GET /produtos é um endpoint que responde à rota /produtos com o método GET.

TIPOS DE ROTAS

As rotas em uma API podem ser fixas ou dinâmicas, dependendo da presença de parâmetros na URL:

- **Rotas fixas**: sempre retornam o mesmo conteúdo, sem variação.
- **Rotas dinâmicas**: adaptam a resposta com base em parâmetros na URL.

Essas rotas são úteis para:

- Buscar um único produto pelo ID
- Listar produtos de uma categoria específica
- Filtrar ou ordenar resultados conforme critérios da requisição.

PARÂMETROS

Parâmetros são informações adicionadas na URL de requisições para identificar ou personalizar dados retornados pela API. Eles permitem que uma mesma rota forneça respostas diferentes com base nos valores informados.

Existem dois tipos principais:

- **Parâmetros de caminho (path params)**: parte da estrutura da rota (ex: /produtos/15 para acessar o produto com ID 15).
- **Parâmetros de consulta (query params)**: enviados após o ? na URL (ex: /produtos?categoria=livros para filtrar por categoria).

PATH PARAMS

Os path params são usados quando se deseja acessar diretamente um recurso específico. Eles fazem parte da estrutura da URL e geralmente representam um identificador único.

Esse tipo de parâmetro é útil quando a aplicação precisa retornar um item individual.

O valor fornecido na URL será capturado automaticamente e repassado como argumento para a função responsável por tratar aquela rota.

```python
@app.route("/produtos/<int:id>")
def buscar_produto(id):
    for produto in produtos:
        if produto["id"] == id:
            return produto
```

QUERY PARAMS

Os query params são utilizados para aplicar filtros, buscas ou personalizações nas respostas de uma API. Eles são adicionados à URL após o símbolo **?** e seguem o formato **chave=valor**.

Ex: /produtos?categoria=livros

Esse tipo de parâmetro não altera a estrutura da rota, apenas modifica o comportamento da resposta com base nos valores recebidos.

```python
from flask import Flask, request

app = Flask(__name__)

produtos = [
    {"id": 1, "nome": "Caderno", "categoria": "papelaria"},
    {"id": 2, "nome": "Caneta", "categoria": "papelaria"},
    {"id": 3, "nome": "Livro", "categoria": "livros"},
    {"id": 4, "nome": "Fone de ouvido", "categoria": "eletrônicos"}
]

@app.route("/produtos")
def filtrar_produtos():
    categoria = request.args.get("categoria")

    if categoria:
        filtrados = []

        for produto in produtos:
            if produto["categoria"] == categoria:
                filtrados.append(produto)

        return filtrados
    return produtos
```

QUERY PARAMS

É possível enviar mais de um parâmetro de consulta na mesma URL usando o símbolo & para separar os pares chave=valor.

Esse formato permite aplicar vários filtros ou critérios ao mesmo tempo.

Exemplo: /produtos?categoria=livros&estoque=baixo

```python
produtos = [
    {"id": 1, "nome": "Livro A", "categoria": "livros", "estoque": "baixo"},
    {"id": 2, "nome": "Livro B", "categoria": "livros", "estoque": "alto"},
    {"id": 3, "nome": "Fone", "categoria": "eletrônicos", "estoque": "baixo"},
    {"id": 4, "nome": "Lápis", "categoria": "papelaria", "estoque": "alto"}
]

@app.route("/produtos")
def filtrar_produtos():
    categoria = request.args.get("categoria")
    estoque = request.args.get("estoque")

    if not categoria and not estoque:
        return produtos

    filtrados = []

    for produto in produtos:
        if categoria and produto["categoria"] != categoria:
            continue
        if estoque and produto["estoque"] != estoque:
            continue
        filtrados.append(produto)

    return filtrados
```

ATIVIDADE PRÁTICA

Crie uma lista com algumas tarefas simuladas, cada uma representada por um dicionário com pelo menos os seguintes campos:

- id (número)
- titulo (texto)
- status (ex: "pendente" ou "concluída")

Depois, crie uma rota que receba um ID como parâmetro e:

- Filtre a tarefa com aquele ID dentro da lista
- Se encontrar, retorne os dados da tarefa com status 200
- Se não encontrar, retorne uma mensagem de erro com status 404

ATIVIDADE PRÁTICA

Utilizando a mesma lista de tarefas do exercício anterior, crie uma nova rota GET /tarefas que aceite um parâmetro de consulta (status) na URL.

Essa rota deve:

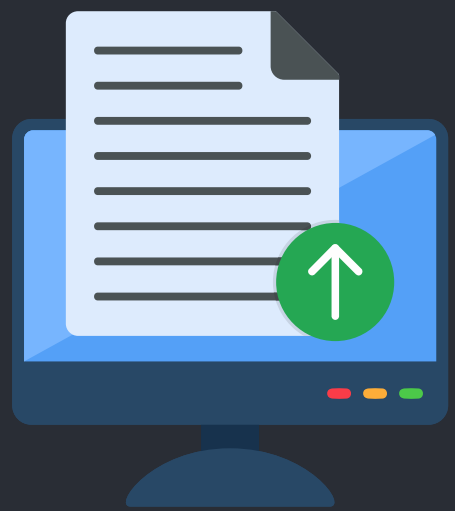
- Retornar apenas as tarefas que possuem o status informado (ex: /tarefas?status=pendente)
- Se nenhum status for enviado, retornar a lista completa
- Se o status for enviado mas nenhuma tarefa for encontrada, retornar uma lista vazia com status 200

ENVIO DE DADOS PARA API

O método HTTP POST é utilizado para enviar dados para o servidor, com o objetivo de criar um novo recurso.

Diferente do GET, que envia informações pela URL, o POST envia os dados no corpo da requisição (body). Por isso, esse método permite transmitir informações mais completas, como um objeto inteiro com vários campos.

Além disso, o POST é frequentemente utilizado em formulários de submissão de dados, como ao criar uma nova conta de usuário ou ao enviar comentários em um blog.



DADOS NO CORPO DA REQUISIÇÃO (BODY)

Quando uma API espera dados com POST, esses dados são enviados dentro do corpo da requisição, geralmente em formato JSON.

No Flask, o conteúdo do corpo pode ser acessado com: **request.json**

Se o JSON estiver correto e o cabeçalho for Content-Type: application/json, o Flask **já entrega os dados como um dicionário Python.**

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

produtos = []

@app.route("/produtos", methods=["POST"])
def criar_produto():
    dados = request.json

    novo_produto = {
        "id": len(produtos) + 1,
        "nome": dados["nome"],
        "preco": dados["preco"]
    }

    produtos.append(novo_produto)
    return jsonify(novo_produto), 201

```

VALIDAÇÃO DE DADOS NO POST

O Flask não valida os dados automaticamente.

Por isso, é importante verificar manualmente se os campos obrigatórios foram enviados antes de salvar ou processar os dados.

```python
@app.route("/produtos", methods=["POST"])
def criar_produto():
    dados = request.json
    print(dados)

    if "nome" not in dados:
        return jsonify({"erro": "O campo 'nome' é obrigatório"}), 400

    novo_produto = {
        "id": len(produtos) + 1,
        "nome": dados["nome"],
        "preco": dados["preco"],
        "categoria": dados["categoria"]
    }

    produtos.append(novo_produto)
    return jsonify(novo_produto), 201

```

ATIVIDADE PRÁTICA

Crie uma rota para cadastrar novas tarefas na sua lista.
A lista de tarefas deve estar em memória, contendo dicionários com os seguintes campos:

- id (número)
- titulo (texto)
- status (ex: "pendente" ou "concluída")

A nova rota deve receber os dados da tarefa em formato JSON, validar se o campo titulo e o id foram enviados, adicionar a tarefa à lista de tarefas já existente e retornar a nova tarefa com status 201.

Caso os campos titulo ou status não estejam presente, a API deve retornar uma mensagem de erro com status 400.