
# Desenvolvimento de APIs e Microsserviços

Flask - Projeto

---

**Caio Eduardo do Prado Ireno**
**caio.ireno@faculdadeimpacta.com.br**

# FLASK

Flask é um pequeno framework web escrito em Python.

Flask é um micro-framework multiplataforma que provê um modelo simples para o desenvolvimento web.

**Mas afinal, o que é um Micro-framework e um framework?**

# Diferenças

Um micro framework e um framework são duas abordagens diferentes para o desenvolvimento de software, cada uma com suas próprias características e finalidades.

# Framework

Um framework é uma estrutura de software abrangente que fornece uma base sólida para o desenvolvimento de aplicativos.

Ele geralmente inclui uma variedade de funcionalidades e componentes prontos para uso, como bibliotecas, módulos, classes e métodos.

# Micro framework

Um microframework, por outro lado, é uma estrutura mais leve e minimalista, projetada para ser mais simples e direta.

Ele oferece apenas o básico necessário para construir aplicativos, sem as funcionalidades extras encontradas em frameworks mais abrangentes.

Os microframeworks são ideais para projetos pequenos e simples, nos quais você deseja ter mais controle sobre as decisões de arquitetura e deseja manter o código o mais enxuto possível.

# Em geral

A principal diferença entre um microframework e um framework é a complexidade e abrangência de suas funcionalidades.

Enquanto um framework oferece uma solução completa e abrangente, um microframework fornece uma abordagem mais leve e minimalista, ideal para projetos menores e mais simples.

# Evolução

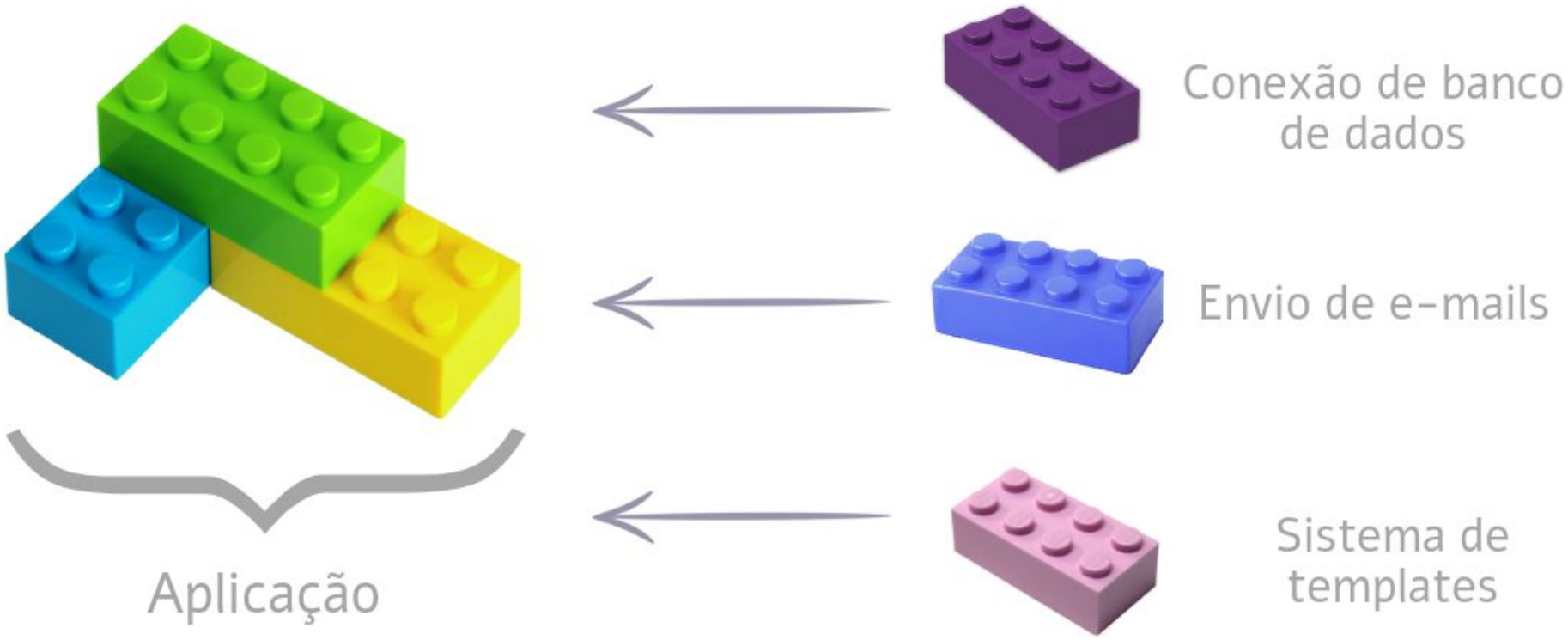
Inicialmente, um projeto criado com o micro-framework possui apenas o básico para funcionar, (normalmente, sistema de rotas).

È possível implementar outros recursos?

conexão de banco de dados

sistemas de templates

envio de email




A partir desta necessidade, novas bibliotecas são “encaixadas” no projeto, como uma estrutura de lego.

# Características do Flask

**Simplicidade**: Por possuir apenas o necessário para o desenvolvimento de uma aplicação, um projeto escrito com Flask é mais simples se comparado aos frameworks maiores, já que a quantidade de arquivos é muito menor e sua arquitetura é muito mais simples.

# Características do Flask

**Rapidez no desenvolvimento:** Com o Flask, o desenvolvedor se preocupa em apenas desenvolver o necessário para um projeto, sem a necessidade de realizar configurações que muitas vezes não são utilizadas.

# Características do Flask

**Projetos menores**: Por possuir uma arquitetura muito simples (um único arquivo inicial) os projetos escritos em Flask tendem a ser menores e mais leves se comparados a frameworks maiores.

# Características do Flask

**Aplicações robustas:** Apesar de ser um micro-framework, o Flask permite a criação de aplicações robustas, já que é totalmente personalizável, permitindo, caso necessário, a criação de uma arquitetura mais definida.

# Hands On

## Como criar um projeto em FLASK?

Dicas:

1. Documentação: Welcome to Flask — Flask Documentation (3.0.x) (palletsprojects.com)
2. Repositório: pallets/flask: The Python micro framework for building web applications. (github.com)

# Criando um Virtual Environment (venv) no Windows

Um **Virtual Environment (venv)** no Python permite criar um ambiente isolado para instalar pacotes sem interferir no sistema global. Isso é útil para evitar conflitos entre diferentes projetos.

```
python -m venv nome_do_ambiente
```

```
.\venv\Scripts\Activate
```

```
pip freeze > requirements.txt
```

# FLASK

Vamos utilizar o vsCode como IDE: Visual Studio Code - Code Editing. Redefined

Instalação de Flask:

Primeiro, você precisa ter o Flask instalado. Você pode instalar usando o pip:

```
pip install Flask
```

# Criando um arquivo Python para sua API

Crie um arquivo chamado app.py (ou qualquer outro nome de sua escolha).

Vamos adicionar algumas coisas básicas.

```
from flask import Flask, jsonify
```

Esta linha importa as classes Flask e jsonify do módulo Flask.

Flask é a classe principal que usaremos para criar a aplicação web, enquanto jsonify é um método conveniente do Flask para converter dicionários Python em respostas JSON.

```
app = Flask(__name__)
```

Esta linha cria uma instância da classe Flask e a atribui à variável app. O argumento __name__ é uma variável especial no Python que representa o nome do módulo atual.

Flask usa isso para determinar o local dos arquivos estáticos, entre outras coisas.

```python
# Rota de exemplo
@app.route('/api/exemplo', methods=['GET'])
def exemplo():
    dados = {'mensagem': 'Bem-vindo à minha API Flask!'}
    return jsonify(dados)
```

**@app** -> Esta linha define uma rota para a nossa API. Quando um cliente faz uma solicitação HTTP GET para **/api/exemplo**, a função decorada abaixo é executada.

A lista de métodos especifica quais métodos HTTP são permitidos para esta rota, neste caso, apenas o método GET.

**def exemplo():** Esta linha define uma função chamada exemplo. Esta função será executada quando a rota /api/exemplo for acessada.

**dados = {'mensagem': 'Bem-vindo à minha API Flask!'}:** Aqui, estamos criando um dicionário Python chamado dados com uma chave mensagem que contém uma string de boas-vindas.

**return jsonify(dados):** Esta linha retorna os dados em formato JSON usando o método jsonify do Flask.

```
if __name__ == '__main__':
    app.run(debug=True)
```

**if name == 'main':** Esta linha verifica se este arquivo está sendo executado diretamente pelo interpretador Python.

**app.run(debug=True):** Se este arquivo estiver sendo executado diretamente, então app.run() inicia o servidor de desenvolvimento do Flask. O argumento debug=True ativa o modo de depuração, o que é útil durante o desenvolvimento, pois fornece mensagens de erro detalhadas e reinicia automaticamente o servidor quando o código é alterado.

# Criando nossa aplicação

Projeto Flask básico para gerenciar usuários

Este projeto irá incluir todas as operações CRUD (Create, Read, Update, Delete) para gerenciar usuários.

# Para o POST

**request.json** é utilizado para acessar os dados enviados pelo cliente na requisição POST para a rota "/users".

Especificamente, request.json é utilizado para obter o nome e o e-mail do novo usuário que estão sendo enviados no corpo da requisição. Esses dados são então utilizados para criar um novo dicionário representando o usuário, que é adicionado à lista usuarios.

# Id especifico

**<int:user_id>** na rota, estamos especificando que o parâmetro **user_id** é esperado como um número **inteiro (int).** Isso é útil quando você precisa garantir que o user_id seja sempre interpretado como um número inteiro, evitando erros de tipo.

Por exemplo, se alguém tentar acessar a rota /users/abc, o Flask irá automaticamente rejeitar essa requisição, retornando um erro 404 (Not Found), porque abc não é um número inteiro válido para user_id.


# Obrigado!