

Sumário:


Criando ambiente virtual:

.\env\scripts\activate

pip install flask


Por que isso é útil, especialmente em projetos grandes ou de outras pessoas?

Exemplo de uso com um projeto de outra pessoa:

 No Prompt de Comando (CMD) do Windows, para desativar o ambiente virtual você usa apenas:

 Por que preciso criar uma pasta templates?

 Porque o Flask espera encontrar seus arquivos HTML lá.

 Estrutura típica de um projeto Flask

 O que vai dentro da pasta templates?

 2. Usando flask run --debug no terminal:

Organizando o HTML e deixando interativo

 Enviando dados através do action e post para o servidor:

Explicação:

Exemplo de Fluxo Completo

Armazenando cookies

```
from flask import Flask, render_template, request

app= Flask(__name__)

@app.route('/')

def index():

    return render_template('index.html')

@app.route('/cadastro', methods=['GET', 'POST'] )

def cadastro():

    if request.method == 'GET':

        return render_template ('cadastro.html')

    else:

        # nome = request.form['nome']

        nome = request.form['nome']

        response= make_response(redirect(url_for('preferencia')))

        response.set_cookie('nome', nome, max_age=7*24*3600)

        return response

@app.route('/preferencia')

def preferencia():

    pass

@app.route('/recomendar')
```

```
def recomendar():
```

```
    pass
```

Criando ambiente virtual:

`py -m venv env`

- Isso cria um ambiente virtual chamado `env` dentro da pasta `teste1`. Um ambiente virtual é como um "espaço isolado" para instalar pacotes, sem interferir nos outros projetos ou na instalação global do Python.

`.\env\scripts\activate`

- Esse comando ativa o ambiente virtual no Windows.
- Após isso, tudo o que for instalado com `pip` (como bibliotecas) será instalado somente dentro desse ambiente, e não no sistema todo.

`pip install flask`

- Esse comando usa o `pip`, o gerenciador de pacotes do Python, para instalar o Flask, que é um micro framework usado para criar aplicações web com Python.

`pip freeze > requirements.txt`

O comando `pip freeze` lista todos os pacotes instalados no ambiente virtual atual, junto com suas versões.

- O símbolo `>` serve para redirecionar essa lista para um arquivo chamado `requirements.txt`.

👉 Resultado: um arquivo de texto chamado `requirements.txt` será criado com o conteúdo como este:

`Flask==2.3.3`

Jinja2==3.1.2

Werkzeug==2.3.3

Por que isso é útil, especialmente em projetos grandes ou de outras pessoas?

1. Reprodutibilidade:

Se outra pessoa quiser rodar seu projeto, ela pode instalar exatamente os mesmos pacotes e versões com:

pip install -r requirements.txt

○

2. Evita erros de versão:

- Em projetos grandes, diferenças de versão podem causar bugs ou falhas. Com o **requirements.txt**, todos usam as mesmas versões.

3. Boa prática de deploy:

- Hospedar em servidores (como Heroku, Render, etc.) quase sempre exige esse arquivo para instalar dependências automaticamente.

4. Organização:

- Em vez de tentar lembrar quais bibliotecas você usou, o **requirements.txt** lista tudo certinho.

Exemplo de uso com um projeto de outra pessoa:

Você baixa um projeto no GitHub, vê que ele tem um **requirements.txt** e executa:

pip install -r requirements.txt

Com isso, todas as bibliotecas que aquele projeto precisa serão instaladas no seu ambiente, do jeitinho certo.

✓ No Prompt de Comando (CMD) do Windows, para desativar o ambiente virtual você usa apenas:

deactivate

- ♦ O que acontece quando você faz isso?
 - O prompt volta ao estado normal (sem o **(env)** no início).
 - Você sai do ambiente virtual, ou seja, os pacotes do projeto voltam a ser os globais do sistema, não os do ambiente isolado.

Iniciando o projeto

 Explicação do código

```
from flask import Flask
```

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

Cria a aplicação web Flask.

 Rotas definidas:

```
@app.route('/')  
def index():  
    pass
```

Rota raiz: exibe a página inicial.

pass significa que ainda não há código executando ali.

```
@app.route('/cadastro')  
def cadastro():  
    pass
```

Página para cadastro de usuários, por exemplo.

```
@app.route('/preferencia')
def preferencia():
    pass
```

Pode ser uma página para o usuário escolher preferências de leitura, temas, gêneros, etc.

```
@app.route('/recomendar')
def recomendar():
    pass
```

Uma rota que pode exibir recomendações personalizadas, com base no cadastro e preferências.

 Por que preciso criar uma pasta **templates**?

 Porque o Flask espera encontrar seus arquivos HTML lá.

- O Flask tem uma função chamada **render_template()** que carrega arquivos **.html**.
- Por padrão, ele procura esses arquivos na pasta chamada **templates**, que deve estar no mesmo nível do seu **app.py** (ou seja, na mesma pasta).

 Estrutura típica de um projeto Flask

bash

CopiarEditar


```
meu_projeto/
|
├─ app.py
├─ /env
└─ /templates
    ├─ index.html
    ├─ cadastro.html
    ├─ preferencia.html
    └─ recomendar.html
```

🧠 O que vai dentro da pasta `templates`?

- Todos os seus arquivos HTML que vão ser exibidos nas rotas.
- Ex: você pode criar `index.html`, `cadastro.html`, etc.
- Eles podem conter HTML puro ou HTML com Jinja2, que é a linguagem de template do Flask (permite colocar variáveis, laços, condições no HTML).

🚀 2. Usando `flask run --debug` no terminal:

- É a forma recomendada pelo Flask para desenvolvimento.
- Ativa o modo debug, que:
 - Atualiza automaticamente quando você salva o código.
 - Mostra erros detalhados no navegador.

Organizando o HTML e deixando interativo

```
<body>
  <h1>Filmes</h1>
  <a href="{{url_for('cadastro')}}">Cadastrar
preferências</a>
  <a href="">Ver preferências</a>
</body>
```

As chaves `{{ }}` são usadas em templating engines, como o Jinja2 no Flask, que é uma ferramenta popular em Python para gerar páginas HTML dinâmicas. Quando você vê `{{url_for('cadastro')}}`, o que está acontecendo é que o Flask substitui essa expressão com a URL correspondente à função ou rota chamada `'cadastro'`.

O código `{{ url_for('cadastro') }}` é usado para gerar o URL correto para a rota chamada `'cadastro'`. Isso torna a navegação no seu site dinâmica e fácil de manter, pois você não precisa codificar manualmente os URLs das rotas, o que pode ser suscetível a erros ao fazer alterações nas rotas ou caminhos do site.

🎉 Enviando dados através do `action` e `post` para o servidor:

```
<form action="{{url_for('cadastro')}}" method="post">
```

`action="{{url_for('cadastro')}}"`: Define a URL para onde os dados do formulário serão enviados.

- `method="post"`: Especifica que os dados serão enviados usando o método HTTP POST.

Quando o usuário preencher o formulário e clicar no botão "Enviar", os dados (nome e gênero) serão enviados para a URL gerada por `url_for('cadastro')`, e o Flask processará essa requisição no backend.

✓✓ Função `cadastro` no Flask

```
def cadastro():  
    if request.method == 'GET':  
        return render_template('cadastro.html')  
    else:  
        nome = request.form['nome']  
        return "Em construção " + nome
```

GET: Quando o usuário acessa a página, o servidor retorna o template `cadastro.html` com o formulário.

POST: Quando o formulário é enviado, o servidor captura o nome inserido e retorna uma mensagem com o nome do usuário.

Explicação:

1. `if request.method == 'GET' ::`

- O método `GET` é usado quando o usuário acessa a página pela primeira vez ou recarrega a página. O Flask verifica se a requisição é do tipo `GET`, que indica que o

servidor deve enviar de volta a página HTML.

- Se for um **GET**, a função vai renderizar a página HTML chamada **cadastro.html** (com base no template). Essa é a página que contém o formulário onde o usuário pode digitar seu nome e escolher o gênero favorito.

2. **return render_template('cadastro.html')**:

- A função **render_template** é responsável por renderizar o arquivo HTML (**cadastro.html** neste caso), retornando o conteúdo dessa página para o usuário. Esse arquivo deve estar localizado na pasta **templates** do seu projeto Flask.

3. **else::**

- Se a requisição não for **GET**, significa que o formulário foi enviado, ou seja, o método da requisição é **POST**.
- Isso ocorre quando o usuário preenche o formulário e clica no botão de envio.

4. **nome = request.form['nome']**:

- **request.form** é um dicionário que contém os dados enviados pelo formulário (via método **POST**).
- O **request.form['nome']** acessa o valor que o usuário inseriu no campo de entrada de texto com o **id="nome"**. Esse valor será atribuído à variável **nome**.

5. **return "Em construção " + nome**:

- Após obter o nome do usuário, a função retorna uma string que diz "Em construção", seguida do nome inserido. Isso simula o processo de recebimento dos dados do formulário.
- A string é retornada como resposta ao navegador do usuário.

Exemplo de Fluxo Completo

1. Primeiro Acesso (GET):

- O usuário acessa a página de cadastro (por exemplo, <http://localhost:5000/cadastro>).
- O servidor retorna o formulário HTML para o usuário preencher.

2. Envio do Formulário (POST):

- O usuário preenche o formulário e clica no botão "Enviar".
- O servidor captura os dados enviados (como o nome do usuário) e retorna a resposta "Em construção [nome do usuário]".

Armazenando cookies

```
@app.route('/cadastro', methods=['GET', 'POST'])
def cadastro():
    if request.method == 'GET':
        return render_template('cadastro.html')
    else:
        # nome = request.form['nome']
        nome = request.form['nome']
        response = make_response(redirect(url_for('preferencia')))
        response.set_cookie('nome', nome, max_age=7*24*3600)
        return response
```

```
def cadastro():
    if request.method == 'GET':
        return render_template('cadastro.html')
```



Parte 1: Quando o usuário acessa a página

Se a requisição for do tipo GET (ou seja, o usuário está apenas acessando a URL /cadastro no navegador), o Flask exibe o formulário HTML chamado cadastro.html.

```
else:
    nome = request.form['nome']
```



Parte 2: Quando o usuário envia o formulário

Isso acontece numa requisição POST, ou seja, quando o usuário clica no botão "Enviar".

O valor preenchido no campo `name="nome"` do formulário é lido com:

```
nome = request.form['nome']
```

```
response = make_response(redirect(url_for('preferencia')))
```



Redirecionamento com resposta personalizada

`redirect(url_for('preferencia'))`: Redireciona o usuário para a rota chamada 'preferencia'.

`make_response(...)`: Flask normalmente cria a resposta automaticamente, mas aqui você cria manualmente para poder adicionar um cookie depois.

```
response.set_cookie('nome', nome, max_age=7*24*3600)
```



Salvando o nome em um cookie

Cria um cookie chamado 'nome' que armazena o nome digitado pelo usuário.

`max_age=7*24*3600`: Define a duração do cookie para 7 dias (7 dias × 24 horas × 3600 segundos).

```
return response
```

Retorna a resposta personalizada com o cookie e o redirecionamento.

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Cadastrar Preferências</h1>
  <form action="{{url_for('cadastro')}}" method="post">
    <label for="nome">Digite seu nome:</label>
    <input type="text" name="nome" placeholder="nome">
    <label for="genero">Escolha o gênero favorito:</label>
    <select id="genero" name="genero">
      <option value="">Selecione</option>
      <option value="aventura">Aventura</option>
      <option value="fantasia">Fantasia</option>
      <option value="romance">Romance</option>
    </select>
    <button>Enviar</button>

  </form>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
  <style>
```



```
body{
  display: flex;
  flex-direction: column;
  align-items: center;
  font-size: 30px;
}
a{
  margin: 5px;
  background-color: rgb(50, 9, 88);
  color: white;
  text-align: center;
  padding: 20px;
  max-width: 500px;
}
</style>
</head>
<body>
  <h1>Filmes</h1>
  <a href="{url_for('cadastro')}">Cadastrar preferências</a>
  <a href="">Ver preferências</a>
</body>
</html>
```