Flask

Com a linguagem python, podemos fazer uso de alguns frameworks para desenvolver aplicações para a internet de maneira bastante simples e rápida. Atualmente, dois frameworks disputam o topo da popularidade para o desenvolvimento de web apps com python, estes são Django e Flask. Apesar da popularidade de ambos, a diferença de complexidade entre ambos é algo considerável, no entanto ambos são perfeitamente capazes de sustentar um back-end com eficiencia.

Para esta aula, nós vamos aprender um pouco mais sobre como podemos criar um web app utilizando o framework flask, que é o mais simples e o mais rápido de se começar dos dois apresentados anteriormente.

O flask trabalha fortemente com o conceito de rotas, ou endpoints que podem ser acessados enviando requisições http para um servidor que esteja esperando requisições em determinada porta. O conceito de rotas pode ser um pouco confuso a princípio, mas é uma implementação bastante simples, o servidor na verdade é um grande organizador de pastas, como o sistema operacional, além disso, cada pasta pode conter arquivos, ou outras pastas, e a cada pasta que adentramos dentro do servidor, estamos adicionando um '/' a partir do nome do servidor, vamos ver um exemplo.

```
# pastas
imagens/
        minha_imagem.png
documentos/
        meu_doc.txt
        outros_docs/
                outro_doc.txt
```

Considerando a estrutura de pastas acima, se o nosso servidor tem o seguinte endereço: http://meuserver.com, podemos acessar cada um dos arquivos da estrutura acima com os exemplos abaixo:

```
http://meuserver.com/imagens/minha_imagem.png
http://meuserver.com/documentos/meu_doc.txt
http://meuserver.com/documentos/outros_docs/outro_doc.txt
```

Cada um dos caminhos acima é uma possível rota de acesso aos arquivos presentes no nosso servidor, e é com esse formato de acesso que vamos escrever a nossa aplicação usando flask.

A aplicação

O primeiro passo para um projeto é sempre criar uma pasta para ele, dessa forma fica mais fácil trabalhar com nossos scripts principais e possíveis scripts de teste. Além disso, vamos instalar o flask no ambiente principal e não em um ambiente virtual, o que, apesar de não ser o mais indicado, vai facilitar bastante no aprendizado neste caso.

```
pip install flask
```

Uma vez instalado o flask, vamos começar, primeiro crie um arquivo python, dê preferencia para nomes que não estão relacionados com o projeto (ex: request, requests, flask e json) para evitar problemas de execução. Neste arquivo vamos começar importando os arquivos do framework flask.

```
from flask import Flask
```

O flask se baseia no uso de aplicações, que são objetos específicos para criar e organizar as rotas dentro do nosso sistema, enquanto criar uma aplicação com flask é bastante simples, temos algumas novidades e alguns pontos de atenção.

O maior ponto de atenção é a criação de um objeto flask utilizando a variável global `__name__` como parâmetro principal. Porque utilizar `__name__` e não outro valor? A resposta é que o flask faz uso do nome do arquivo principal para procurar outras pastas que podem ser utilizadas para servir arquivos estáticos por exemplo, se informarmos outra string, o framework pode não encontrar os diretórios necessários para funcionar corretamente.

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

Requisições GET

Com flask podemos facilmente criar rotas para receber requisições GET utilizando o decorator @app, mas o que é um decorator? Basicamente, um decorator é uma função que envolve outra função, dando capacidades adicionais a função encapsulada, funcionalidades como por exemplo, escutar a requisições enviadas para um endpoint específico. Um decorator sempre começa com o simbolo '@'.

Como dito acima, o decorator precisa de uma função para funcionar, então vamos suplementar a rota raiz ('/') com uma função que retorna um hello world.

```
@app.route('/')
def hello_world():
        return 'Hello, World!'
```

Vamos executar a aplicação para testar se funciona. Com flask, a execução dos scripts python é feita de maneira um pouco diferente, em vez de digitar python e o nome do script, basta escrever flask run com o prompt de comando apontando para a pasta em questão para executar o servidor.

```
C:\Users\USERNAME\Documents\Python\projects\web_app>flask run
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Agora, vamos testar a rota, para isso basta abrir o seu navegador padrão com o endereço informado acima (http://127.0.0.1:5000/) para ver o resultado

```
Hello, world!
```

Criando outras rotas

Agora que sabemos como criar uma rota simples, vamos ver alguns outros exemplos:

```
@app.route('/json')
def hello_json():
  return {'chave': 'valor'}
```

Neste exemplo, estamos criando uma rota que responde a requisições no endereço http://127.0.0.1/json, e retorna para o navegador o json {'chave': 'valor'} (para facilitar a visualização instale uma extensão de visualização de json para o navegador de sua preferencia).

 Não esqueça de interromper a execução do flask e iniciar de novo a cada alteração no script, caso contrário suas alterações não terão efeito! Para interromper a execução do flask basta apertar ctrl+c com o prompt de comando selecionado.

Também podemos utilizar rotas como variáveis em nossas funções, vamos criar uma rota hello world 2.0 que utiliza o valor informado na rota para escrever o nome do usuário em vez de world.

```
@app.route('/<nome>')
def greet(nome):
    return f'Hello, {nome}'
```

No exemplo acima, a rota utiliza uma wildcard, ou um coringa, que é um valor não predefinido e será informado no momento da requisição GET. Para testar, abra o seu navegador e digite http://127.0.0.1/Ana, veja que o retorno será:

```
Hello, Ana!
```

Se você alterar o nome Ana para outro nome, como por exemplo Julian, o hello será diferente

```
Hello, Julian!
```

Podemos inclusive utilizar html no retorno da nossa função, como no exemplo:

```
@app.route('/<nome>')
def greet(nome):
    return f'<p style="color: red">Hello, {nome}</p>'
```

Requisições POST

Agora que já conhecemos como criar requisições do tipo GET, vamos aprender a criar o segundo tipo de requisição mais comum, o POST. Diferentemente do GET, com POST podemos receber informações de maneira mais estruturada nas nossas requisições, como por exemplo, objetos em formato json.

O primeiro passo é incluir na importação do flask o objeto requests também.

```
from flask import Flask, request
```

Diferentemente do módulo requests que já utilizamos em outras aulas, o request é um objeto que contem informações sobre a requisição recebida, como por exemplo, os cabeçalhos, que são parte fundamental de uma requisição e permitem ao servidor saber mais sobre o cliente que está requisitando alguma informação.

Para criar uma rota que recebe requisições post, utilizamos o argumento methods dentro de @app.route()

```
@app.route('/save', methods=['POST'])
```

O uso de methods=['POST'] significa que se tentarmos acessar esse endpoint pelo nosso navegador não teremos sucesso (na verdade neste caso aparecerá hello, save! porque já temos outras rotas que podem se encaixar nesta requisição), e na maior parte das vezes retornara um erro 404, ou arquivo não encontrado.

No corpo da função decorada vamos utilizar o objeto request para obter as informações enviadas em formato JSON utilizando a propriedade json. Vamos aproveitar para retornar ao requisitante este mesmo JSON.

```
@app.route('/save', methods=['POST'])
def save():
    json = request.json
    return json
```

Repare que não é necessário importar a biblioteca JSON neste caso, isso acontece porque flask já faz uso de uma biblioteca de serialização de JSON.

Para testar essa requisição, vamos criar outro arquivo python, eu chamei este arquivo de http_test.py, mas você pode dar outro nome se quiser. Neste arquivo vamos utilizar o módulo requests para enviar uma requisição POST.


Se você ainda não tiver instalado o modulo requests, não se esqueça de executar pip install requests antes de prosseguir.

O conteúdo do nosso teste é bastante simples, mas não esqueça de reiniciar a execução do flask antes de tentar executar este script python.

```
import requests
import json

r = requests.post(
    'http://127.0.0.1:5000/save',
    data=json.dumps({'chave': 'valor'}),
    headers={'content-type': 'application/json'}
)
print(r.json())
```

No script acima, estamos enviando uma requisição post para o endereço criado na nossa aplicação flask (http://127.0.0.1/save), como dados, estamos enviando um dicionário convertido a JSON com o uso de json.dumps(),e temos uma nova adição a esta requisição, que são os headers ou cabeçalhos.

Os cabeçalhos são trechos de texto que são enviados juntamente com a requisição para o servidor, fornecendo a ele algumas informações sobre a natureza daquela requisição. Por padrão, todas as requisições HTTP fazem uso de alguns cabeçalhos genéricos, no entanto, estamos fornecendo um cabeçalho diferente, o 'content-type', que informa ao servidor o tipo de conteúdo que estamos enviando.

O servidor, munido dessas informações pode ou não aceitar a requisição, neste caso, como estamos trabalhando com JSON, precisamos informar ao servidor que o tipo de dados transmitidos está em formato JSON, para isso utilizamos o identificador 'application/json'.

Com o servidor flask online, abra um novo terminal e execute o script python de teste:

```
# python http_test.py
{'chave': 'valor'}
```

Com isso finalizamos a nossa introdução ao desenvolvimento web utilizando python, fique a vontade para explorar mais sobre o mundo web e especialmente o framework flask, que possui muito mais recursos do que temos tempo para apresentar nesta aula.


Para conhecer mais sobre o flask acesse https://flask.palletsprojects.com/