Criando uma aplicação em Flask + Python

Introdução

Este tutorial traz trechos de código para criar seu primeiro programa utilizando Python+Flask. Os tópicos a seguir esclarecem alguns termos e oferece um exemplo reprodutível de um conteúdo inicial de uma aplicação. Este texto é uma adequação de um artigo disponível em inglês [1].

O que é o Python?

Python é uma linguagem de programação de alto nível, interpretada, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte que foi lançada por Guido van Rossum em 1991.

O que é o Flask?

Flask é um pequeno framework web escrito em Python. É classificado como um microframework porque não requer ferramentas ou bibliotecas particulares, mantendo um núcleo simples, porém, extensível.

Um microframework facilita o recebimento de uma solicitação HTTP, roteando a solicitação HTTP para a função apropriada e retornando uma resposta HTTP. Microframeworks geralmente são projetados especificamente para construir APIs para outro serviço ou aplicativo. Assim, isto permite que nos concentremos naquilo que os usuários estão solicitando e no tipo de resposta a ser retornada.

O código escrito a partir das orientações presentes neste documento será responsável pelo processamento do lado do servidor, recebendo solicitações e as respondendo.

O que é HTTP e qual a relação com o Flask?

HTTP (Hypertext Transfer Protocol, ou Protocolo de Transferência de Hipertexto) é o principal protocolo responsável pela transferência de dados na Internet, utilizado para interagir com computadores e servidores, para comunicação entre eles.

Exemplificando o uso cotidiano, ao digitar um endereço na barra de pesquisa de seu navegador e pressionando ENTER, o navegador envia uma solicitação HTTP para o servidor desejado. Por exemplo, digitando 'google.com' e pressionando ENTER, o servidor do Google enviará como resposta a página de pesquisas dele para seu navegador, onde é possível interagir e receber outras informações como texto, images, vídeos entre outros conteúdos.

Como funciona uma aplicação em Flask?

O exemplo a seguir oferece uma aplicação da web básica, que mostrará a mensagem "Hello, World!" no navegador. Para criar seu primeiro projeto, sugiro a criação de uma pasta "flask" e dentro dessa pasta, crie um arquivo "main.py" com o conteúdo indicado a seguir:

```
---arquivo: main.py
01 from flask import Flask
02
03 app = Flask(__name__)
04
05 @app.route("/")
06 def home():
07   return "Hello, World!"
08
```

```
09 if __name__ == "__main__":
10   app.run(debug=True)
---
```

Entendendo o que foi copiado:

Linha 1: o módulo do Flask é importado e criamos um servidor da web do Flask a partir dele.

Linha 3: __name__ representa o arquivo atual. Neste caso, estamos falando de main.py. Esse arquivo atual representará minha aplicação da web. Estamos criando uma instância da classe Flask e chamando-a de app. Aqui, estamos criando uma nova aplicação da web.

Linha 5: representa a página padrão. Por exemplo, se eu vou a um site da web, como o "google.com/", sem nada após a barra. Essa será a página padrão do Google. Isso á o que representa @app.route(“/”)

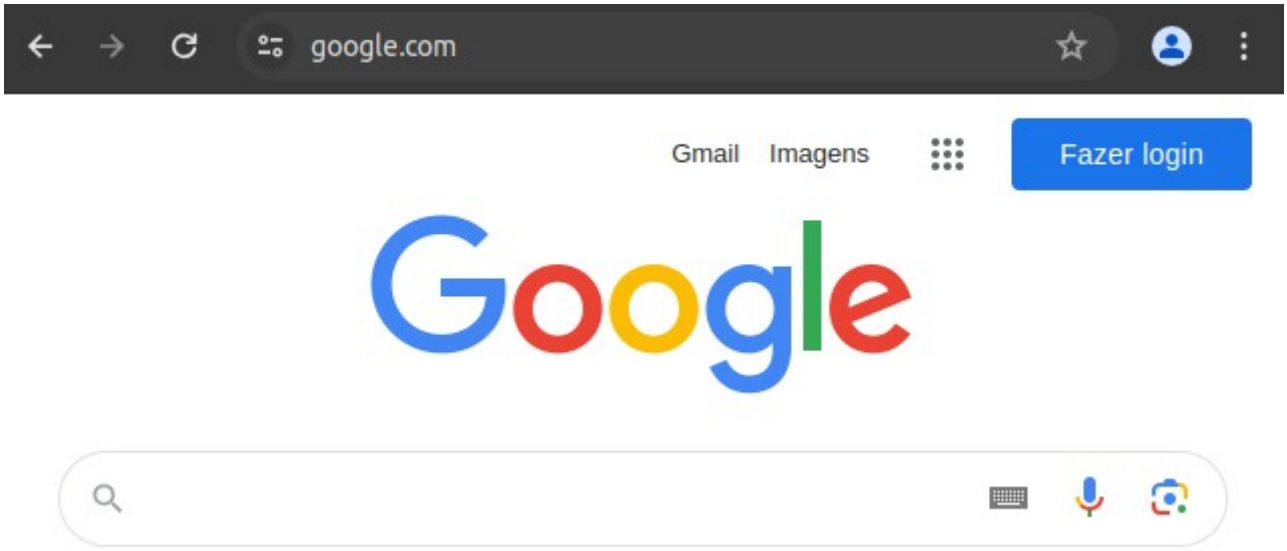

Figura 1: Página recebida ao informar o endereço “google.com/”

Linhas 6 e 7: quando a página padrão (nada após a barra) for acessada, a função retorna a mensagem.

Linha 9: ao executar esse script em Python, o Python atribuirá o nome "__main__" ao script quando ele for executado.

Se importamos outro script, a instrução `if` evitará que outros scripts sejam executados. Ao executarmos main.py, ele alterará seu nome para __main__ e somente então aquela instrução if será ativada.

Linha 10: ao executar a aplicação, o parâmetro `debug=True` permitirá que erros do Python apareçam na página da web para auxiliar a identificar e corrigir eventuais erros.

Executando o código pela primeira vez

Considera-se aqui que seu computador já possui Python instalado, assim como o Flask.

Abra um console, terminal ou prompt de comando. Digite `python main.py` e pressione ENTER. O resultado será algo similar ao ilustrado a seguir:

```
$ python main.py
 * Serving Flask app "main" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 153-530-207
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Figura 2: Terminal linux com o script main.py em execução

A última linha mostra: `Running on http://127.0.0.1:5000/`

Isso significa que em seu computador, localmente, na porta 5000 o programa em python com interface em Flask está funcionando. Para testar o resultado, acesse em seu navegador o endereço informado. O resultado será:

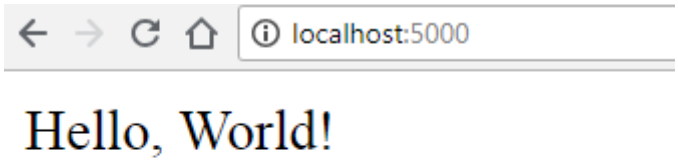

Figura 3: Programa em Flask funcionando

Com o programa executando como esperado, vamos acrescentar outras rotas e testar o funcionamento.

Edite o arquivo “main.py” e acrescente uma nova rota. Por exemplo:

```
---arquivo: main.py
01 from flask import Flask
02
03 app = Flask(__name__)
04
05 @app.route("/")
06 def home():
07   return "Hello, World!"
08
09 @app.route("/info")
10 def info():
11   return "Informação"
12
13 if __name__ == "__main__":
14   app.run(debug=True)
---
```

Finalize o programa em execução pressionando as teclas CTRL e C ao mesmo tempo.

Execute o `python main.py` e pressione ENTER. Verifique no navegador o resultado ao informar o endereço: `http://localhost:5000/info`

Até aqui foram retornadas somente mensagens de texto. A seguir veremos como mostrar conteúdos em HTML com estilização CSS.

HTML e templates no Flask

O framework Flask procura por arquivos HTML em uma pasta chamada templates. É necessário criar essa pasta e armazenar os arquivos .html nesse local. A estrutura do diretório ficará assim:

```
+ Flask
+--+ templates
|  +-- home.html
+-- main.py
```

Crie o arquivo `home.html` dentro da pasta templates com o seguinte conteúdo:

```
---arquivo: home.html.
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Título da aba: Flask Tutorial</title>
  </head>
  <body>
    <h1> Minha primeira página HTML em Flask </h1>
    <p> Primeiro parágrafo </p>
  </body>
</html>
---
```

É necessário ajustar o arquivo "main.py" para utilizar o "home.html" criado.

```
---arquivo: main.py
01 from flask import Flask, render_template
02
03 app = Flask(__name__)
04
05 @app.route("/")
06 def home():
07   return render_template("home.html")
08
09 @app.route("/info")
10 def info():
11   return "Informação"
12
13 if __name__ == "__main__":
14    app.run(debug=True)
15 We made two new changes
---
```

As modificações no arquivo main.py foram:

Linha 1: importamos o método `render_template()` do framework Flask. Reponsável por mostrar o arquivo indicado que está na pasta templates.

Linha 7: alteramos o `return` para `render_template("home.html")` com o objetivo de mostrar a página HTML no navegador.

Crie um arquivo `about.html` dentro da pasta `templates`.

```
---arquivo: about.html
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>About Flask</title>
  </head>
  <body>
    <h1> About Flask </h1>
    <p> Flask is a micro web framework written in Python.</p>
    <p> Applications that use the Flask framework include Pinterest,
      LinkedIn, and the community web page for Flask itself.</p>
  </body>
</html>
---
```

Atualize o arquivo `main.py` para acessar a nova página sobre a aplicação

```
---arquivo: main.py
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/")
def home():
    return render_template("home.html")

@app.route("/about)
def about():
    return render_template("about.html")

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

Execute a aplicação e teste os resultados.

Criando um cabeçalho comum para conectar as duas páginas com um menu de navegação

Para conectar as duas páginas, podemos ter um menu de navegação na parte superior. Podemos usar o Flask para tornar mais fácil o processo de criação de um menu de navegação.

Primeiro, vamos criar um arquivo template.html. Esse arquivo servirá como modelo principal ("pai"). As páginas ("filhos") herdarão o modelo do código principal.

```
---arquivo: template.html
01 <!DOCTYPE html>
02 <html lang="en" dir="ltr">
03   <head>
04     <meta charset="utf-8">
05     <title>Flask Parent Template</title>
06     <link rel="stylesheet" href="{{ url_for('static',     filename='css/template.css') }}">
07   </head>
08 <body>
```

```
09  <header>
10   <div class="container">
11    <h1 class="logo">First Web App</h1>
12     <strong><nav>
13      <ul class="menu">
14       <li><a href="{{ url_for('home') }}">Home</a></li>
15       <li><a href="{{ url_for('about') }}">About</a></li>
16      </ul>
17     </nav></strong>
18    </div>
19   </header>
20
21   {% block content %}
22   {% endblock %}
23
24 </body>
25 </html>
---
```

Linhas 13 e 14: usaremos a função chamada url_for() [2]. Ela aceita um nome de função como argumento. Neste momento, demos a ela o nome da função.

As duas linhas com as chaves serão substituídas pelo conteúdo de home.html e de about.html. Isso dependerá do URL em que o usuário estiver navegando.

Essas alterações permitem que as páginas filhas (home.html e about.html) conectem-se à página pai (template.html). Isso nos permite não precisar copiar o código para o menu de navegação em about.html e em home.html.

O arquivo about.html ficará assim:

```
---arquivo: about.html
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>About Flask</title>
  </head>
  <body>
    {% extends "template.html" %}
    {% block content %}

    <h1> About Flask </h1>
    <p> Flask is a micro web framework written in Python.</p>
    <p> Applications that use the Flask framework include Pinterest,
      LinkedIn, and the community web page for Flask itself.</p>

    {% endblock %}
  </body>
</html>
---
```

Enquanto o home.html ficará assim:

```
---arquivo: home.html
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
```

```
    <title>Flask Tutorial</title>
  </head>
  <body>
    {% extends "template.html" %}
    {% block content %}

    <h1> My First Try Using Flask </h1>
    <p> Flask is Fun </p>

    {% endblock %}
  </body>
</html>
---
```

Adicionando estilos ao site

Assim como na pasta templates são armazenados os arquivo .html, a pasta static é utilizada para armazenar as folhas de estilo. A estrutura do diretório ficará assim:

```
+ Flask
+--+ static
|  +--+ css
|     +-- template.css
+--+ templates
|  +-- about.html
|  +-- home.html
|  +-- template.html
+-- main.py

---arquivo: template.css
body {
  font-family: 'Montserrat', sans-serif;
}

.logo {
  background-color: #0D122B;
  padding: 10px;
  color: white;
}
---
```

Vinculando o arquivo de estilos aos arquivos HTML

O template.html é o arquivo que vincula todas as páginas. Podemos inserir o código aqui e ele será aplicável a todas as páginas filhas.

```
---arquivo: template.html
01 <!DOCTYPE html>
02 <html lang="en" dir="ltr">
03  <head>
04    <meta charset="utf-8">
05    <title>Flask Parent Template</title>
06
07                  <link     rel="stylesheet"     href="{{     url_for('static',
filename='css/template.css') }}">
08
09 </head>
10   <body>
11     <header>
12       <div class="container">
13         <h1 class="logo">First Web App</h1>
```

```
14         <strong><nav>
15           <ul class="menu">
16             <li><a href="{{ url_for('home') }}">Home</a></li>
17             <li><a href="{{ url_for('about') }}">About</a></li>
18           </ul>
19         </nav></strong>
20       </div>
21     </header>
22
23 {% block content %}
24 {% endblock %}
25
26 </body>
27 </html>
---
```

Linha 7: aqui, fornecemos o caminho onde o arquivo template.css está localizado.

Execute a aplicação e teste os resultados.

Pronto! Sua aplicação está funcionando em ambiente local.

Para hospedar sua aplicação num servidor real, fora da sua máquina pessoal, é necessário seguir mais alguns passos para preparar sua aplicação para web. São sugeridas a utilização de virtualenv e as configurações da plataforma onde o programa ficará disponível. Para isso, consulte o artigo original [1] para concluir esta etapa. O exemplo oferecido no artigo propõe o uso do Google App Engine (Standard Environment) [3].

Referências

[1] VILLALON, Salvador. How to build a web application using Flask and deploy it to the cloud. Disponível na Internet desde 28 de agosto de 2018 em: https://www.freecodecamp.org/news/how-to-build-a-web-application-using-flask-and-deploy-it-to-the-cloud-3551c985e492/
[2] url_for() Veja mais detalhes em: http://flask.pocoo.org/docs/0.12/quickstart/#url-building
[3] Google App Engine. Disponível em: https://cloud.google.com/appengine/docs/standard