Flask

Desenvolvendo aplicações Web com o Framework Flask

Aula 2

Flask

Definindo regras variáveis (regras de URL)

Como vimos na aula anterior, para adicionar partes variáveis a uma *URL*, podemos marcar essas seções especiais como *<nome_variavel>*, exemplo *"@app.route('/cliente/<id>')"*. Essa parte é passada como um argumento de palavras-chave para a função. Opcionalmente podemos informar um conversor de tipo de dado, desta forma:

@app.route('/cliente/<int:id>')

Flask

Definindo regras variáveis (regras de URL)

Existem os seguintes conversores:

string	Aceita qualquer texto sem uma barra (o padrão)
int	Aceita inteiros
float	Como o *int*, porém, para ponto flutuante
path	Como o padrão, porém, aceita barra
uuid	Aceita *UUID strings* (123e4567-e89b-12d3-a456-426655440000)

Flask

URLs únicos / Comportamento de redirecionamento

```
@app.route('/clientes/')
def clientes():
    return 'Página de clientes'

@app.route('/clientes')
def clientes():
    return 'Página de clientes'
```

Embora pareçam bastante semelhantes, eles diferem no uso da barra na definição da *URL*. No primeiro caso, a *URL* possui uma barra no final (similar a uma pasta em um sistema de arquivos), ao acessarmos sem a barra no final, ocasionará que o *Flask* irá executar com a barra no final, ou seja, a chamada vai funcionar se for informada a barra ou não.

No segundo caso, caso seja chamado com a barra no final, ocasionará o erro 404 "não encontrado". Esse comportamento permite que *URLs* relativos continuem funcionando, mesmo que a barra seja omitida, o que consiste com o modo como o *Apache* e outros servidores *web* funcionam. Além disso, os *URLs* permanecerão únicos, o que ajuda os motores de busca a evitar a indexação da mesma página duas vezes.

Flask

Construção de URLs

Além de combinar *URLs*, o *Flask* também pode gerar *URLs*. Para criar uma *URL* para uma função específica, podemos utilizar a função *url_for()*. Ela aceita o nome da função como primeiro argumento e uma série de argumentos de palavras-chave, cada um correspondendo à parte variável da regra da *URL*. As partes variáveis desconhecidas são anexadas à *URL* como parâmetro de consulta.

Flask

Construção de URLs

Veja um exemplo:

```
from flask import Flask, url_for
app = Flask(__name__)

@app.route('/')
def principal(): pass

@app.route('/login')
def entrar(): pass

@app.route('/user/<username>')
def perfil(username): pass

with app.test_request_context():
    print(url_for('principal'))
    print(url_for('entrar'))
    print(url_for('entrar', next='/'))
    print(url_for('perfil', username='Evaldo Wolkers'))
```

Resultado:
/
/login
/login?next=%2F
/user/Evaldo%20Wolkers

O *app.test_request_context()* diz ao Python que se comporte como se estivesse lidando com uma solicitação. Usando ele em combinação com *with*, ativamos um context de requisição temporariamente, permitindo acessarmos objetos *request* e *session* como se estivéssemos em uma função de *view*.

Flask

Construção de URLs

Veja mais um exemplo:

```
from flask import Flask, redirect, url_for
app = Flask(__name__)

@app.route('/admin')
def ola_admin():
    return 'Olá Admin'

@app.route('/visitante/<visitante>')
def ola_visitante(visitante):
    return 'Olá visitante %s' % visitante

@app.route('/usuario/<nome>')
def ola_user(nome):
    if nome == 'admin':
        return redirect(url_for('ola_admin'))
    else:
        return redirect(url_for('ola_visitante', visitante = nome))

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

Flask

Métodos HTTP

O protocolo *HTTP* é a base da comunicação de dados na *Internet*. O mesmo possui diferentes métodos para acessar *URLs* e recuperar dados. Estes métodos são:

GET	HEAD	POST	PUT	DELETE
Método mais comum. Solicita a representação de um recurso, seja arquivo *html*, *xml*, *json*, *etc*.	O mesmo que *GET*, porém, retorna somente os cabeçalhos de uma resposta sem corpo da resposta.	Usado quando queremos criar um recurso. Os dados vão no corpo da requisição e não na *URI*.	Requisita que um recurso seja guardado na *URI* fornecida, criando ou alterando um recurso.	Exclui o recurso especificado.

Flask

Métodos HTTP

Por padrão, uma rota do *Flask* apenas responde à requisições *GET*, mas podemos alterar isto fornecendo argumentos de métodos para o *route()*.

Veja um exemplo:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        efetuar_login()
    else:
        exibir_formulario_login()
```

Flask

Métodos HTTP

Implementando um exemplo de *login (login.py)*.

```
from flask import Flask, redirect, url_for, request
app = Flask(__name__)

@app.route('/success/<user_name>')
def success(user_name):
   return 'Bem-vindo %s' % user_name

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        user_name = request.form['user_name']
        return redirect(url_for('success', user_name = user_name))
    else:
        user_name = request.args.get('user_name')
        return redirect(url_for('success', user_name = user_name))

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

Flask

Métodos HTTP

Implementando um exemplo de *login (login.html)*.

Requisição via *GET* (barra de endereços do *browser*):

http://localhost:5000/login?user_name=Evaldo

Requisição via *POST* (formulário *HTML* de *login*):

```
<html>
    <meta charset="utf-8"/>
   <body>
      <form action = "http://localhost:5000/login" method = "post">
         <p>Informe o nome do usuário:</p>
         <p><input type = "text" name = "user_name" /></p>
         <p><input type = "submit" value = "submit" /></p>
      </form>
   </body>
</html>
```

Flask

Renderizando modelos

Gerar *HTML* à partir do *Python* é um pouco complicado porque precisamos escapar a saída do *HTML* para manter o aplicativo seguro. Para evitar isso, o *Flask* configura o mecanismo de modelo *Jinja2*, que escapa o *HTML* automaticamente.

Para renderizar o modelo usamos o método *render_template()*.

O que precisamos fazer é fornecer o nome do modelo e as variáveis que desejamos passar para o mecanismo de modelo como argumentos de palavras-chave.

Flask

Renderizando modelos

O *Flask* procura os modelos na pasta de modelos (*templates*).

Se sua aplicação for um módulo, esta pasta fica próxima a este módulo, se for um pacote, o modelo estará dentro do pacote.

```
Módulo
/application.py
/templates
    /hello.html

Pacote
/application
    /__init__.py
    /templates
        /hello.html
```

Flask

Renderizando modelos

Exemplo de como renderizar um modelo:

```
from flask import Flask, redirect, url_for, request, render_template
app = Flask(__name__)

@app.route('/success/<user_name>')
def success(user_name):
   return render_template('ola.html', user_name=user_name)

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        user_name = request.form['user_name']
        if user_name == 'evaldo':
            return redirect(url_for('success', user_name = user_name))
        else:
            return "Usuário inválido."

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

Ola.html

```
<!DOCTYPE html>
<title>Olá Mundo</title>
{% if user_name %}
  <h1>Olá {{ user_name }}!</h1>
{% else %}
  <h1>Olá Mundo!</h1>
{% endif %}
```

FIM