Flask

Desenvolvendo aplicações Web com o Framework Flask

Aula 4 – Parte 4

Jinja2

Flask – Jinja2

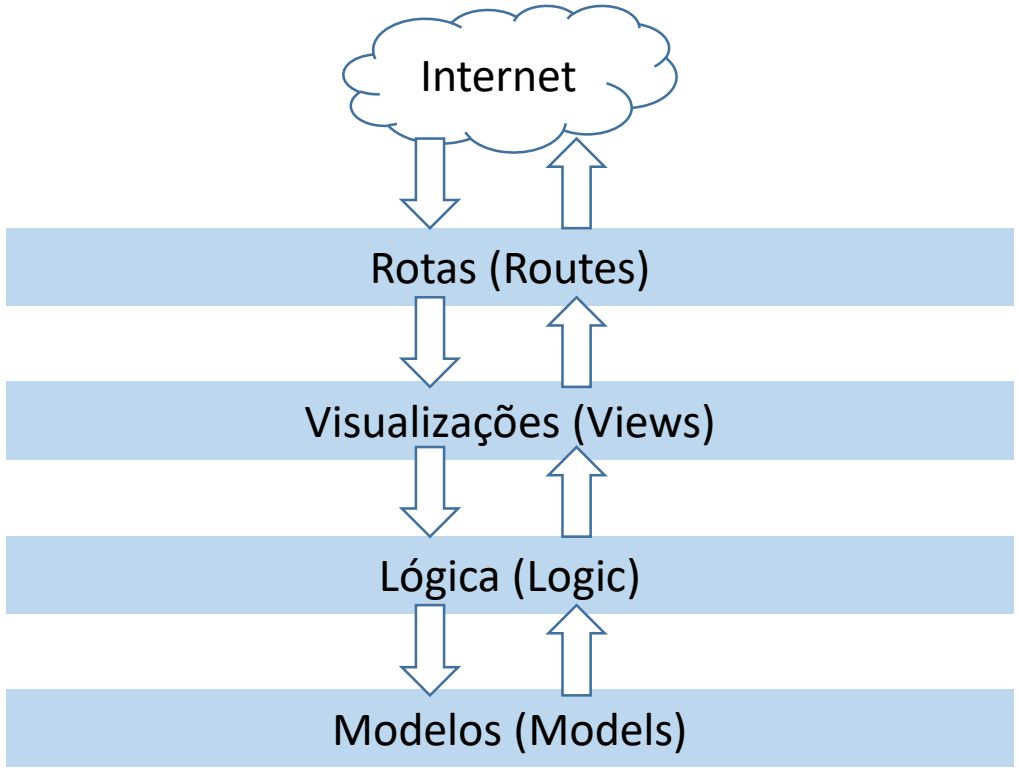
Templates com Jinja2

Estamos utilizando o *Jinja2* desde a segunda aula sobre o *framework Flask*, nesta aula vamos falar um pouco mais em *templates* com *Jinja2*.

Flask – Jinja2

Templates com Jinja2

Quando criamos uma *view*, ficou claro que a tarefa da mesma é gerar uma resposta a uma requisição.



Flask – Jinja2

Templates com Jinja2

Para requisições simples, isso é o suficiente, porém, uma requisição também dispara uma mudança no estado da aplicação, e a função de *view* é o local em que essa mudança é gerada.

Flask – Jinja2

Templates com Jinja2

Quando preenchemos os campos de um formulário de cadastro por exemplo e clicamos no botão para submeter este formulário, o servidor recebe esta requisição com os dados fornecidos pelo usuário e o *Flask* despacha para a função de *view* que trata as requisições de registro desses dados.

Flask – Jinja2

Templates com Jinja2

Essa função de *view* precisa conversar com o banco de dados para que o registro seja adicionado e, em seguida, gerar uma resposta para ser enviada de volta ao navegador, a qual inclui uma mensagem de sucesso ou de erro. Esses dois tipos de tarefa são formalmente chamados de lógica de negócios (*business logic*) e lógica de apresentação (*presentation logic*), respectivamente.

Flask – Jinja2

Templates com Jinja2

Misturar a lógica de negócios com a lógica de apresentação resulta em um código difícil de entender e de manter. Suponha que você tenha que desenvolver o código *HTML* para uma tabela grande concatenando dados obtidos do banco de dados com as strings literais *HTML* necessárias. Passar a lógica de apresentação para os *templates*, como fizemos nos nossos exemplos, ajuda no momento em que precisarmos realizar manutenções na aplicação.

Flask – Jinja2

Templates com Jinja2

Um *template*, como já vimos, é um arquivo que contém o texto de uma resposta, com variáveis para as partes dinâmicas que só serão conhecidas no contexto de uma requisição. O processo que substitui as variáveis pelos valores propriamente ditos e devolve uma string de resposta final se chama renderização (*rendering*). Como já vimos, o ***Jinja2*** é um "motor" eficaz utilizado para esta tarefa de renderizar *templates*. Vamos fazer mais um exemplo, utilizando um *template.*

Flask – Jinja2

Estruturas de controle

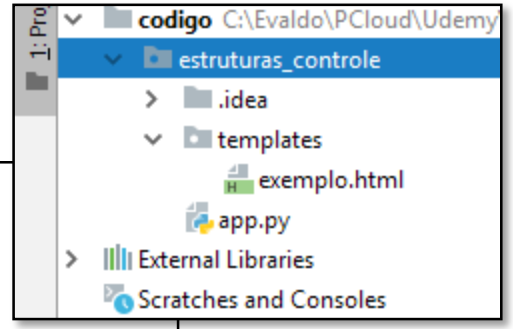
O ***Jinja2*** oferece estruturas de controle que podem ser usadas para alterar o fluxo do *template*. Em um template criado na segunda aula vimos o uso da instrução *if,* veja abaixo.

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

Flask – Jinja2

Estruturas de controle

Vamos ver agora como utilizar a instrução *for* para renderizar uma lista de elementos em nosso *template*. Veja o conteúdo do *app.py*:



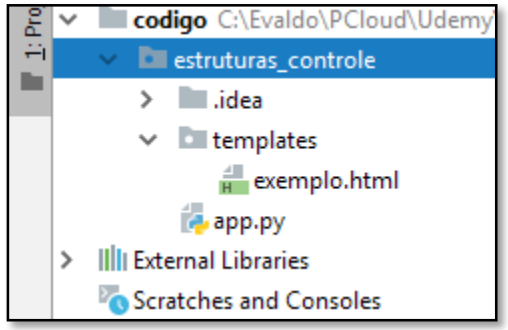

```
from flask import Flask, render_template
app = Flask(__name__)

@app.route('/')
def exemplo():
    carros = ['Corolla', 'Sandero', 'Punto', 'Ranger', 'Gol', 'Clio', 'Ka']
    return render_template('exemplo.html', carros=carros)

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

Flask – Jinja2

Estruturas de controle

Conteúdo do *exemplo.html*:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
  <ul>
      {% for carro in carros %}
        <li>{{ carro }}</li>
      {% endfor %}
  </ul>
</body>
</html>
```



FIM