

Flask

Desenvolvendo aplicações Web com o Framework Flask - Aula 4

Um pouco mais sobre Templates com Jinja2 – Parte 3

Flask – Templates com Jinja2

O que veremos nesta aula

Nesta aula serão abordados conceitos básicos de templates Jinja2 à partir da perspectiva do Flask. Também veremos como criar aplicativos com templates modulares e extensíveis.

Veremos:

- Templates Jinja2
- Usar o Bootstrap
- Composição de bloco e herança de layout
- Criando um processador de contexto customizado
- Criando um filtro personalizado Jinja2
- Criando uma macro personalizada para formulários
- Formatação de data e hora com Moment.js

Flask – Templates com Jinja2

Criando uma macro personalizada para formulários

As macros nos permitem escrever partes reutilizáveis de blocos HTML. São similares a funções em linguagens de programação regulares. Podemos passar argumentos para macros como fazemos em funções no Python e depois usá-las para processar o bloco HTML. Macros podem ser chamadas qualquer número de vezes, e a saída irá variar conforme a lógica dentro delas.

Trabalhar com macros no Jinja2 é um tópico muito comum e tem muitos casos de uso. Veremos como uma macro pode ser criada e usada depois da importação.

Flask – Templates com Jinja2

Criando uma macro personalizada para formulários

Um dos trechos de código mais redundantes em HTML é a definição de campos de entrada em formulários. A maioria dos campos tem código semelhante com algumas modificações de estilo e assim por diante.

Vamos criar uma macro que cria campos de entrada de dados quando chamada.

A prática recomendada é criar a macro em um arquivo separado para melhor reutilização, por exemplo, `_campos.html`.

É sempre uma boa prática definir macros em um arquivo diferente para manter o código limpo e aumentar a legibilidade do código.

Flask – Templates com Jinja2

Criando uma macro personalizada para formulários

Veja o conteúdo do “_campos.html”:

```
{% macro criar_campos_login(name, class='', value='', type='text') -%}
    <input type="{{ type }}" name="{{ name }}" class="{{ class }}" value="{{ value }}" />
{%- endmacro %}
```

O sinal de menos (-) antes / depois do % irá remover os espaços em branco após e antes desses blocos e vai tornar o código HTML mais limpo para ler.

E agora do arquivo que
usa a macro “teste.html”:

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% from '_campos.html' import criar_campos_login %}
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<fieldset>
    Nome: {{ criar_campos_login('usuario') }}<br><br>
    Senha: {{ criar_campos_login('senha', type='password') }}
</fieldset>
</body>
</html>
```

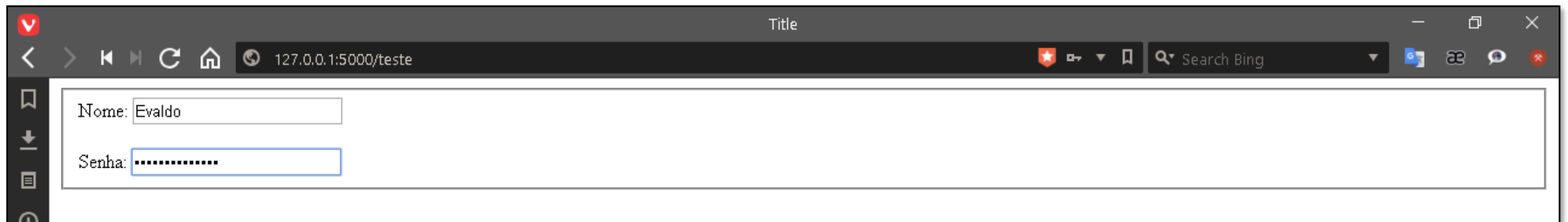
Flask – Templates com Jinja2

Criando uma macro personalizada para formulários

Agora temos que implementar a rota para “teste.html” no arquivo “views.py”:

```
@product_blueprint.route('/')  
def teste():  
    return render_template('teste.html')
```

Veja o resultado:



A screenshot of a web browser window. The address bar shows the URL '127.0.0.1:5000/teste'. The page content displays a form with two input fields. The first field is labeled 'Nome:' and contains the text 'Evaldo'. The second field is labeled 'Senha:' and contains a series of dots, indicating a password. The browser's interface includes standard navigation buttons (back, forward, refresh, home) and a search bar.

Flask – Templates com Jinja2

Formatação de data e hora avançado

A formatação de data e hora é uma tarefa difícil de lidar em aplicativos web. Ao manipulá-los no nível do Python, o uso da biblioteca `datetime` aumenta a sobrecarga e é bastante complexo quando se trata de lidar corretamente com fusos horários. Devemos padronizar os timestamps para UTC (*O Tempo Universal Coordenado, abreviadamente **UTC**, do inglês Universal Time Coordinated, também conhecido como tempo civil, é o fuso horário de referência a partir do qual se calculam todas as outras zonas horárias do mundo*) quando armazenados no banco de dados, mas, em seguida, os timestamps precisam ser processados sempre que precisarem ser apresentados aos usuários em todo o mundo.

Flask – Templates com Jinja2

Formatação de data e hora avançado

É uma coisa inteligente adiar esse processamento para o lado do cliente, ou seja, o navegador. O navegador sempre sabe o fuso horário atual do usuário e poderá fazer a manipulação de data e hora corretamente. Além disso, isso elimina a sobrecarga necessária de nossos servidores de aplicativos. Nós usaremos a biblioteca JavaScript Moment.js para esse propósito.

Baixe a biblioteca em:

<http://momentjs.com/downloads/moment.min.js>

Flask – Templates com Jinja2

Formatação de data e hora avançado

Para incluir a biblioteca Moment.js em nossa aplicação, basta copiar o arquivo moment.min.js para a pasta static/js.

Para utilizar a biblioteca, basta colocar a seguinte instrução em nosso arquivo HTML:

```
<script src="/static/js/moment.min.js"></script>
```

Flask – Templates com Jinja2

Formatação de data e hora avançado

Agora, vamos criar um arquivo chamado “teste.html”:

```
<html>
  <head>
    <title>Exemplo Moment.js</title>
  </head>

  <body>
    <h1>Quanto tempo passou: <span id="data" data-informada="1978-09-26 00:00:00 UTC"></span></h1>

    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js"></script>
    <script src="static/js/moment.min.js"></script>
    <script>
...

```

Flask – Templates com Jinja2

Formatação de data e hora avançado

Agora, vamos criar um arquivo chamado “teste.html”:

```
...  
    $(document).ready(function() {  
        var data = $('#data'),  
            date = moment(new Date(data.attr('data-  
informada')));  
        update = function() {  
            data.html(date.fromNow());  
        };  
  
        update();  
    });  
</script>  
</body>  
  
</html>
```

Flask – Templates com Jinja2

Formatação de data e hora avançado

Por fim vamos criar a rota para “teste.html” em views.py:

```
@teste_blueprint.route('/')  
def teste():  
    return render_template('teste.html')
```

Veja o resultado:



FIM