Programação de Sistemas para Internet

Prof. Diego Cirilo

Aula 06: Templates

Templates

- As *views* podem retornar páginas estáticas, mas qual seria a vantagem?
- Templates permitem a substituição dinâmica de dados em uma página HTML
- Isso é possível através de *tags*.
- A estrutura é de um HTML normal, com *tags* extras.

Templates

- O Django por padrão procura os templates na pasta `templates` em cada *app*
- É possível configurar outra pasta no arquivo `settings.py`

Tags

- `{{ variáveis }}`
- `{% tags/funções %}`
- Lista das tags

Exemplos

- HTML

```
<ul>
    <li>Batata</li>
    <li>Farinha</li>
    <li>Queijo</li>
</ul>
```

- Template Django

```
<ul>
    {% for item in lista_compras %}
        <li>{{ item  }}</li>
    {% endfor %}
</ul>
```

Tags

- For

```
{% for variavel in lista %}
    ...{{ variavel }}...
{% endfor %}
```

- If

```
{% if condicao %}
    ...verdadeiro
{% elif outracondicao %}
    ...verdadeiro
{% else %}
    ...falso
{% endif %}
```

Contexto

- De onde vem os dados para o template?
- R. da *view*!
- A função `render` aceita (além de `request` e o nome do *template*) mais um parâmetro: um dicionário com dados para o template.
- Dicionário: `{"chave": "valor", "outrachave": "outrovalor"}`
- Esse dicionário é normalmente chamado de contexto ou `context`

Contexto

- Na *view*:

```
def index(request):
    dados_usuario = {"nome": "Michael Douglas", "idade": 23}
    return render(request, "index.html", dados_usuario)
```

- No *template*:

```
...
<p>Nome: {{ nome }}</p>
<p>Idade: {{ idade }}</p>
...
```

Contexto

- Para passar vários dados podemos utilizar listas de dicionários.
- Ex.

```
def index(request):
    lista_usuarios = [
        {"nome": "Michael Douglas", "idade": 23},
        {"nome": "James Wilson", "idade": 55},
        {"nome": "Peter Parker", "idade": 22},
    ]

    context = {
        "usuarios": lista_usuarios,
    }
    return render(request, "index.html", context)
```

Contexto

- No *template*:

```
...
{% for usuario in usuarios %}
    <p>Nome: {{ usuario.nome }}</p>
    <p>Idade: {{usuario.idade }}</p>
{% endfor %}
...
```

Tarefa

- Usando o mesmo projeto da aula anterior:
- Crie uma *view* e um *template* `usuarios`. Configure as *urls* para `/usuarios`
- Na sua *view* de usuários, crie uma lista de 5 dicionários, cada dicionário deve ter os seguintes dados:
    - Nome, matrícula, idade, cidade
- Crie dados fictícios para esses 5 usuários.
- Crie um *template* que consiga apresentar os dados de todos os usuários listados
- Teste o sistema e faça o commit quando tiver funcionando.

Herança de templates

- Páginas web repetem muito código
- Ex. um menu que aparece em todas as páginas, o *header* e o *footer*
- Os templates podem "importar" pedaços de outros templates
- Usamos um template base com o que deve ser padrão em todas as páginas
- As outras páginas apenas substituem partes do template base.

Herança de templates

- Na página base:

```
{% block nome-do-bloco %}
    <conteúdo padrão do bloco>
{% endblock %}
```

- Na página que herda:

```
{% extends "pagina-base.html" %}
...
{% block nome-do-bloco %}
    <novo conteúdo do bloco>
{% endblock %}
```

Herança de templates

- A *tag* `extends` deve ser a primeira do documento.
- É possível criar vários `block`s no mesmo template, sem repetir seus nomes.
- O conteúdo padrão do `block` pai pode ser acessado com `{{ block.super }}`

Exemplo

- `base.html`:

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %}Título base do site{% endblock %}</title>
  </head>
  <body>
    {% block content %}
      <p>Conteúdo do site</p>
    {% endblock %}
  </body>
</html>
```

Exemplo

- `pagina.html`

```
{% extends "base.html" %}
{% block title %}{{ block.super }} - Nome da página{% endblock %}
{% block content %}
<div class="classe">
  <h1> Conteúdo do meu site </h1>
</div>
{% endblock %}
```

Arquivos Estáticos

- Os arquivos estáticos não ficam dentro de `templates`
- O motivo é: os `templates` não são páginas HTML! Não ficam públicos para os clientes.
- Os `templates` são renderizados e então disponibilizados pelo servidor.
- Os arquivos estáticos, como imagens, JS e CSS são disponibilizados diretamente pelo servidor web.
- O caminho padrão do Django é a pasta `static` dentro do *app*

Arquivos Estáticos

- Usamos `{% load static %}` no início da página.
- Usamos `{% static 'nomedoarquivo.etc' %}` no lugar do nome do arquivo.
- Os caminhos são relativos ao diretório `static`.
- Ex.

```
<img src="{% static 'cat.jpg' %}" alt="Foto do gato">
<link rel="stylesheet" href="{% static 'css/style.css' %}">
```

URLs/Links

- É possível escrever os links diretamente:

```
<a href="/index">Página Inicial</a>
```

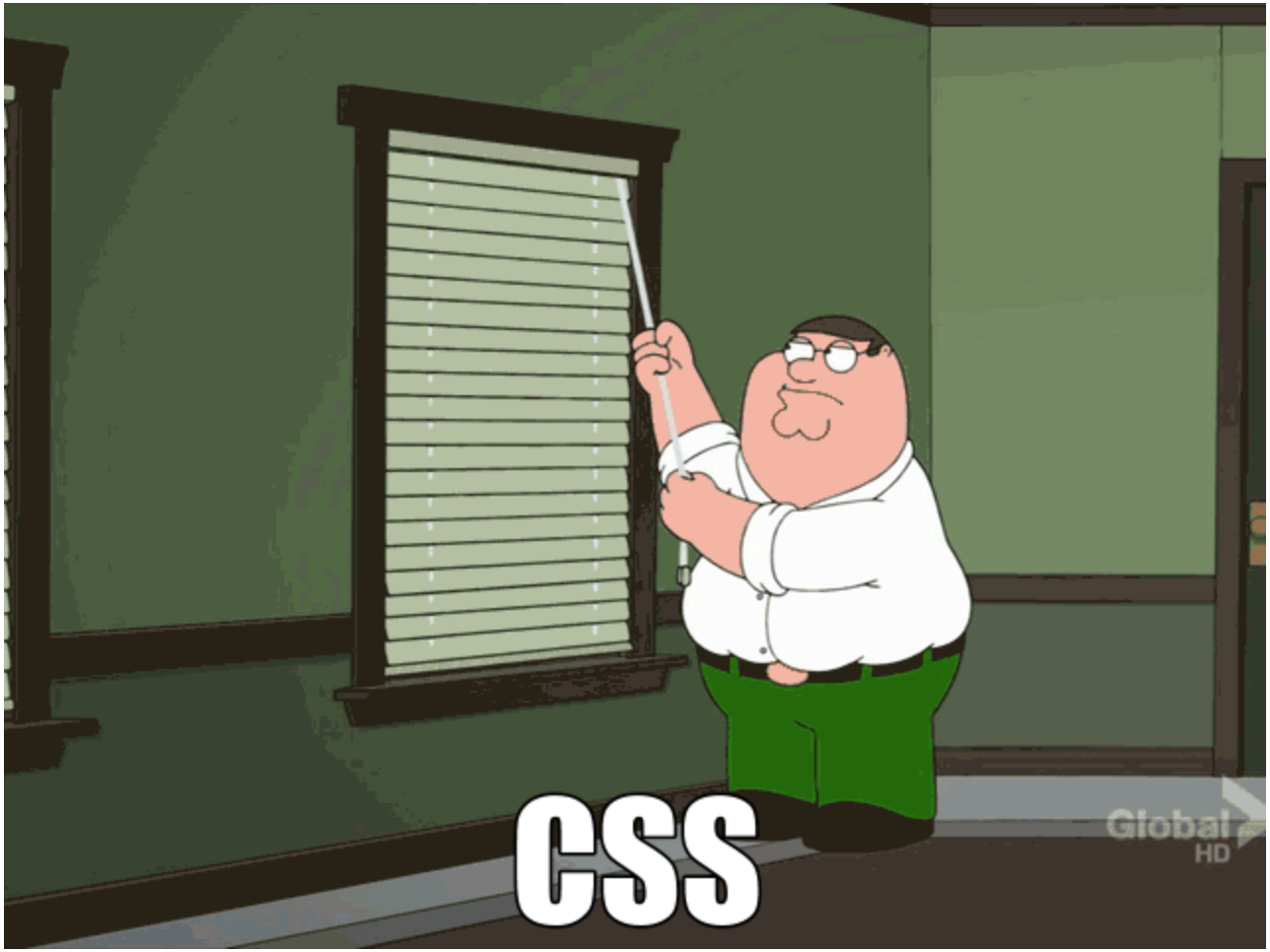
- Também é possível usar os templates:

```
<a href="{% url 'index' %}">Página Inicial</a>
```

- Usamos o mesmo `name` definido nos arquivos `urls.py`.

Tarefa

- Utilizando seus conhecimentos de *webdesign*:
  - Crie um site para divulgar uma equipe de futebol (masculino, feminino, seleção, etc).
  - O site deve ter 3 páginas: Início, Jogadores, Sobre.
  - Início: informações gerais sobre o time, com imagens, histórico, etc.
  - Atletas: foto, nome, idade, posição e local de nascimento. Basta 11 atletas.
  - Sobre: informações sobre o site, autores, etc.
- O site deve ter um menu global e um *footer* com informações como *copyright*.
- O site deve funcionar dentro do Django, usando um *template* base e os atletas devem ser descritos em um dicionário na *view*.
- Use o repositório modelo para fazer o upload do trabalho.


CSS
Global
HD

Dúvidas?