



Django do Zero

Como Criar um Aplicativo Web

Canal youtube.com/@VamosCodar - Wanderson Reis

04 - Introdução aos Templates

Requisitos

- 1) Aula 01 - Introdução a Frameworks Web e Django: <https://youtu.be/LpFNMn6Uw5s>
- 2) Aula 02 - Configurações do Ambiente: https://youtu.be/7zxJi3_HRuW
- 3) Aula 03 - Primeira Aplicação - Olá Django: <https://youtu.be/5MoZZttT6UA>

Nesta aula:

- 1) Introdução
- 2) Um template base para o layout
- 3) Criando um template de layout genérico
- 4) Aplicando o layout genérico a um template final
- 5) Usar arquivos estáticos (imagens, CSS, JS locais, etc)

Referências: <https://docs.djangoproject.com/en/4.2/topics/templates/>

Introdução

Um template Django basicamente é um mecanismo de gerar páginas HTML dinamicamente. É formado por algumas partes estáticas em HTML e partes definidas por uma linguagem especial que produz a parte dinâmica do conteúdo / sistema. O template é renderizado (montado) e devolvido como resposta a requisição a uma determinada rota.

A sintaxe da linguagem de template permite incluir **variáveis** (que serão fornecidas pelas views), por exemplo:

Seu saldo é R\$ {{ saldo }}

Neste exemplo observe que as variáveis ficam entre chaves duplas **{{ }}** e são recebidas pelo contexto da view, por exemplo (será um dicionário Python): **{'saldo': 234.58}**

Nas variáveis é possível aplicar **filtros** que realizam uma transformação ou pré-processamento dos valores armazenados na variável antes de gerar o HTML final. Veja os exemplos abaixo:

{{ nome_completo|title }} - Este filtro converte a primeira letra de todas as palavras do conteúdo da variável para maiúsculas.

{{ data_publicacao|date:"d/m/Y" }} - O filtro também pode aceitar argumento, como este exemplo que formata conteúdo do tipo date.

Além das variáveis é suportada a execução de códigos para implementação da lógica de apresentação. Neste caso são usadas **tags**, por exemplo:

```
{% if user.is_authenticated %} {# uma tag com o comando de decisão “if” #}  
    Olá, {{ user.name }} {# uma variável #}  
{% endif %} {# outra tag fechando o bloco “if” #}
```

As tags são determinadas pela marcação `{% %}` e permitem a execução de praticamente qualquer código Python, além de comandos próprios que auxiliam a construção da lógica de apresentação.

Outro tipo de tags são as de comentários de template `{# #}`. Neste caso todo o conteúdo dentro de uma tag de comentário não é renderizado e nem é enviado na resposta, isto não aparece no código fonte HTML (final).

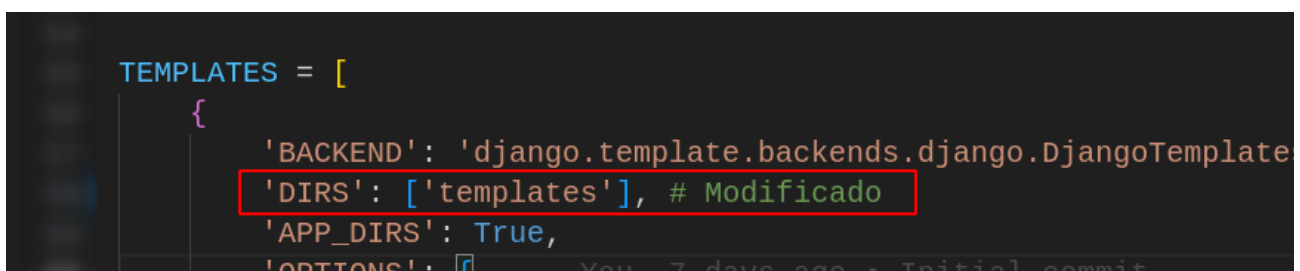
Consulte a documentação completa sobre templates Django:

Outra característica importante do mecanismo de template é a possibilidade de **um template herdar ou ser derivado de outros templates**, desta forma permite o reuso dos templates em diversas páginas / telas substituindo apenas o que for necessário. Neste tutorial vamos construir uma estrutura básica de templates que permita a reutilização.

Preparação - É obrigatório ter realizado a Aula 02: Primeira Aplicação - “Olá Django”

Nesta aula fizemos a configuração do projeto django para criar e compartilhar uma pasta **templates** comum para uso no projeto.

Editamos o arquivo `django-do-zero/webapp/settings.py`, conforme abaixo:



```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplate  
        'DIRS': ['templates'], # Modificado  
        'APP_DIRS': True,  
        'OPTIONS': {
```

E no projeto criamos a pasta `django-do-zero/templates`

Passo 1: Um template base para o layout

É possível criar um ou mais templates básicos que servirão de **base para o layout do seu sistema**. A ideia é que estes templates básicos sejam a base (o ponto de partida comum) a todo o resto do layout. Vamos criar o arquivo `_base.html` dentro da pasta **templates** do projeto, com o conteúdo abaixo.

`django-blog/templates/_base.html`:

```
{% load static %} {# carrega a pasta / localização dos arquivos estáticos #}
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>
    {% block head_title %}{% endblock %} {# bloco de título #}
  </title>

  <link rel="shortcut icon" type="image/x-icon" href="{% static 'images/favicon.ico' %}">

  {% block css %} {# bloco de CSS #}
  {% endblock %}

</head>
<body>
  {% block header %} {# bloco de cabeçalho #}
  {% endblock %}

  {% block main %} {# bloco principal do body #}
    {% block content %} {# bloco de conteúdo #}
    {% endblock %}
  {% endblock %}

  {% block footer %} {# bloco de rodapé #}
  {% endblock %}

  {% block javascript %} {# bloco extra abaixo do body #}
  {% endblock %}

</body>
</html>
```

(salve o arquivo).

Todas as seções **block** serão substituídas automaticamente conforme as configurações nos templates que utilizarão o `_base.html` como ponto de inicial. Observe que o nome do bloco é personalizável, você pode criar com o nome que quiser e quantos blocos forem necessários. Apenas seguindo a sintaxe `{% block nome_bloco %} {% endblock %}`. Estes blocos formarão regiões dentro do seu layout.

Passo 2: Criando um template de layout genérico

Agora que temos um template base podemos um ou mais template de layout conforme as seções e necessidades do sistema. Vamos criar um template que herda tudo do `_base.html`, neste caso vamos usar todas as regiões (blocos) pré-definidos. Em outros layouts de páginas personalizados não é obrigatório usar todos os blocos, pois como foram criados “vazios” em `_base.html` simplesmente não será exibido nada nas regiões não utilizadas. Vamos criar o arquivo `_layout1.html` dentro da pasta templates do projeto, com o conteúdo abaixo.

django-do-zero/templates/_layout1.html:

```
{% extends '_base.html' %} {# herda tudo que já tem em _base.html #}
{% load static %} {# carrega o módulo static para fornecer arquivos estáticos #}

{#
    cada bloco abaixo como o mesmo nome terá o conteúdo
    substituído em _base.html, se não existir aqui, permanecerá
    com o mesmo conteúdo original herdado de _base.html
#}
{% block head_title %}
    Django Framework
{% endblock %}

{% block css %}
    {# Bootstrap CSS #}
    <link
        rel="stylesheet"
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css">

    {# CSS do Projeto #}
    {# o comando "static" traduz para o caminho correto para os arquivos estáticos #}
    <link rel="stylesheet" href="{% static 'css/base.css' %}">
{% endblock %}

{% block header %}
<div class="container">
    <header class="d-flex flex-wrap justify-content-center py-3 mb-4 border-bottom">
        <a href="{% url 'index' %}" class="d-flex align-items-center mb-3 mb-md-0 me-md-auto
link-body-emphasis text-decoration-none">
            <span class="fs-4">Django do Zero</span>
        </a>
        <ul class="nav nav-pills">
            <li class="nav-item">
                <a href="{% url 'ola' %}" class="nav-link px-2 link-dark">Sobre</a>
            </li>
        </ul>
    </header>
</div>
```

```

{% endblock %}

{% block main %}
<div class="container">
  {% block content %}
    <p>Conteúdo padrão...</p>
  {% endblock content %}
</div>
{% endblock %}

{% block footer %}
<footer class="footer">
  <div class="container">
    <span class="text-muted">Rodapé...</span>
  </div>
</footer>
{% endblock %}

{% block javascript %}
  {# Bootstrap JavaScript #}
  <script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js">
  </script>

  {# JS do Projeto #}
  <script src="{% static 'js/base.js' %}"></script>
{% endblock %}

```

(salve o arquivo).

Agora podemos usar o template `_layout1.html` como um layout básico para todas as demais páginas ou criar outros templates que reflitam o layout necessário, por exemplo se tiver páginas que não precisa de todos os blocos basta recriar e/ou omitir os blocos herdados de `_base.html`.

Passo 3: Aplicando o layout genérico a um template final

Criar o arquivo `django-do-zero/templates/index.html` e com conteúdo abaixo (substitua tudo):

`django-do-zero/templates/index.html`:

```

{% extends '_layout1.html' %} {# herda tudo que já tem em _layout1.html #}

{% block head_title %}
  Minhas Enquetes
{% endblock %}

```

```
{% block content %} {# substitui o bloco content em _layout1.html #}  
  <h1>{{ titulo }}</h1> {# variável a ser substituída #}  
{% endblock %} {# fecha o bloco content #}
```

(salve o arquivo).

Editar o arquivo **django-do-zero/polls/views.py** e adaptar a view **index** para utilizar (renderizar) o template **index.html** passando dados para o contexto do template (conteúdo do “titulo”).

Crie um novo retorno (return) / reposta conforme a imagem abaixo.

```
def index(request):  
    # return HttpResponse('Olá Django - index')  
    # return render(request, 'index.html')  
    return render(request, 'index.html', {'titulo': 'Últimas Enquetes'})
```

Para testar o carregamento do novo template, inicie o servidor do ambiente de desenvolvimento. Caso não estiver em execução, rodar o seguinte comando:

python manage.py runserver (ENTER)

Acesse a respectiva rota pelo navegador:

127.0.0.1:8000/index

Passo 4: Usar arquivos estáticos (imagens, CSS, JS locais, etc)

No Django estes arquivos não são disponibilizados diretamente. Principalmente por medidas de segurança, todos os arquivos estáticos precisam ser previamente carregados pelo Django. Primeiro passo é configurar o projeto para usar arquivos estáticos. Edite o arquivo **django-do-zero/webapp/settings.py** acrescente as configurações conforme indicado abaixo.

```
# (...manter tudo o que já existe...)
```

```
# na parte SUPERIOR do arquivo  
import os
```

```
# na parte INFERIOR do arquivo  
STATIC_ROOT = os.path.join(BASE_DIR, "public")  
STATICFILES_DIRS = [os.path.join(BASE_DIR, "static")]
```

Depois é necessário criar a pasta **static** em **django-do-zero/static** (ficará na pasta raiz do projeto).

Dentro da pasta **static** separe e organize os arquivos estáticos em subpastas, no nosso caso vamos incluir alguns arquivos vazios de exemplo:

static/css/base.css

static/js/base.js

static/images/empty.txt

Depois que os arquivos estiverem disponíveis e organizado execute no terminal o seguinte comando:

python manage.py collectstatic (ENTER)

Neste processo o Django carrega todos os arquivos estáticos do seu projeto da pasta **static** para a pasta **public**. Serão os arquivos da pasta **public** que serão servidos (disponibilizados) pelo servidor. Esta ação deve ser repetida sempre que acrescentar arquivos estáticos novos (css, js, imagens, etc).

Para testar se os arquivos estáticos ficaram corretamente configurados crie um novo arquivo em `django-do-zero/static/images` chamado `favicon.ico`. Será o ícone de favoritos do aplicativo web (exibido na aba do navegador). Execute novamente o comando:

python manage.py collectstatic (ENTER)

Depois com o servidor de desenvolvimento em execução acesse a rota da view `index()` => `127.0.0.1:8000/index` e observe se o arquivo foi carregado. Se necessário force o carregamento da página sem cache (CTRL + F5).

Conclusão

O mecanismo de templates do Django é o método principal de gerar uma resposta aos clientes do sistema. Os templates produzem conteúdos acoplados/mesclados por dados vindos dos modelos e são diretamente interpretados pelo navegador (HTML/CSS/JS). Além dos templates, o Django também é capaz de gerar outros formatos de respostas como JSON, PDF, XLS, PNG/JPG, etc.