Apostila 2: Templates e Arquivos Estáticos

O "T" do MVT: Deixando suas Páginas Bonitas

1. Introdução

Na aula anterior, usamos HttpResponse para enviar HTML diretamente do nosso arquivo views.py.

```
# O jeito antigo (ruim para HTML complexo)
def home(request):
    return HttpResponse("<h1>Olá, Mundo!</h1>")
```

Por que isso é um problema?

- **Manutenção Difícil:** Imagine um site inteiro escrito dentro de funções Python. Seria um pesadelo.
- **Sem Separação:** O programador back-end (que mexe em Python) e o designer front-end (que mexe em HTML/CSS) trabalhariam no mesmo arquivo.
- **Não é DRY:** Você teria que repetir o cabeçalho, menu e rodapé em *todas* as views.

O Django resolve isso com **Templates**.

- **Template:** É um arquivo .html que serve como um "molde".
- **View:** A view (em views.py) "pega" esse molde, insere dados dinâmicos nele (como o nome do usuário, uma lista de produtos) e o "renderiza" (transforma em HTML final) para o navegador.

Nosso Objetivo: Recriar nossa página "Olá, Mundo!" usando um arquivo HTML dedicado e, em seguida, adicionar uma folha de estilo (CSS) para ela.

2. Módulo 1: Configurando o Django para Templates (15 min)

Primeiro, precisamos dizer ao Django onde nossos "moldes" (templates) ficarão.

Passo 1: Crie a pasta templates
Na raiz do seu projeto (na mesma pasta onde está o manage.py), crie uma nova pasta chamada templates.
Sua estrutura de pastas deve ficar assim:

```
meuprojeto/
core/
templates/       <-- CRIE ESTA PASTA
venv/
manage.py
```

Passo 2: "Apresente" a pasta ao Django
Abra seu arquivo de configurações, meuprojeto/settings.py.
Procure pela variável TEMPLATES. Ela se parece com isto:

Python

```
# meuprojeto/settings.py

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],  # <-- AQUI!
        'APP_DIRS': True,
        'OPTIONS': {
            ...
        },
    },
]
```

Nós precisamos editar a linha 'DIRS': []. DIRS é uma lista de diretórios onde o Django deve

procurar por templates.

Vamos adicionar nossa nova pasta. Você pode usar uma biblioteca do Python (os ou pathlib) para garantir que funcione em qualquer sistema operacional.

No topo do settings.py, adicione: from pathlib import Path (ou import os). Se Path já estiver importado, ótimo.

Agora, modifique a linha DIRS:

```
# meuprojeto/settings.py
...
# Se não tiver, adicione esta linha no topo
from pathlib import Path
BASE_DIR = Path(__file__).resolve().parent.parent

...

TEMPLATES = [
    {
        ...
        # Diga ao Django para procurar templates na pasta 'templates'
        'DIRS': [BASE_DIR / 'templates'],
        ...
    },
]
```

BASE_DIR é a raiz do seu projeto. BASE_DIR / 'templates' (ou os.path.join(BASE_DIR, 'templates')) simplesmente aponta para a pasta que acabamos de criar.

3. Módulo 2: De HttpResponse para render() (20 min)

Agora que o Django sabe *onde* procurar, vamos criar o template e usá-lo.

Passo 1: Crie o arquivo home.html
Dentro da sua pasta templates/, crie um arquivo chamado home.html.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Minha Primeira Template</title>
</head>
<body>
    <h1>Olá, Mundo! (Vindo de um Template!)</h1>
    <p>Este HTML está separado do meu código Python. Isso é ótimo!</p>
</body>
</html>
```

Passo 2: Modifique a view para usar render()
Vamos "aposentar" o HttpResponse e usar a função render(), que é muito mais poderosa.
Abra core/views.py:

```
# core/views.py

from django.shortcuts import render # 1. Importe o 'render'
# from django.http import HttpResponse <-- Não precisamos mais disso

def home(request):
    # 2. Mude a função
    # O render() pega o 'request' e o nome do arquivo de template
    return render(request, 'home.html')
```

Passo 3: Teste!
Rode o servidor (python manage.py runserver) e acesse http://127.0.0.1:8000.
Você verá sua nova página! O resultado visual é o mesmo, mas a arquitetura do seu código está infinitamente melhor.

4. Módulo 3: Passando "Contexto" (Dados) para o Template (25 min)

O objetivo de um template não é ser estático; é exibir dados vindos do Python. A "ponte" que leva dados da view para o template chama-se **Contexto**.

Contexto (ou context) nada mais é que um **dicionário Python**.

Passo 1: Envie dados da view
Vamos modificar core/views.py para enviar um "nome" e uma "lista de itens" para o template.

```
# core/views.py

from django.shortcuts import render

def home(request):
    # 1. Crie seu dicionário de contexto
    context = {
        'nome_usuario': 'Júnior',
        'tecnologias': ['Python', 'Django', 'HTML', 'CSS']
    }

    # 2. Passe o 'context' como terceiro argumento do render()
    return render(request, 'home.html', context)
```

Passo 2: Receba os dados no Template
Agora, vamos editar o templates/home.html para usar esses dados. O Django usa uma sintaxe especial chamada Django Template Language (DTL).

- {{ variavel }}: Usado para **exibir** uma variável.
- {% tag %}: Usado para **executar lógica** (como if ou for).

Edite seu templates/home.html:

```
...
<body>
    <h1>Olá, {{ nome_usuario }}!</h1>

    <p>Este HTML está separado do meu código Python.</p>

    <h3>Tecnologias que estou aprendendo:</h3>

    <ul>
        {% for tec in tecnologias %}
            <li>{{ tec }}</li>
        {% endfor %}
    </ul>

    {% if nome_usuario == 'Júnior' %}
        <p>Este é um conteúdo especial para o Júnior!</p>
    {% endif %}

</body>
</html>
```

Passo 3: Teste!
Recarregue a página http://127.0.0.1:8000.
Seu HTML agora é dinâmico! Ele foi "montado" pelo Django usando os dados que você enviou pela view.

5. Módulo 4: Arquivos Estáticos (CSS, JS, Imagens) (30 min)

Nosso site está dinâmico, mas está feio. Vamos adicionar CSS.

Arquivos que não mudam (CSS, JavaScript, Imagens) são chamados de **Arquivos Estáticos** (static files). O Django precisa saber onde encontrá-los, assim como fizemos com os templates.

Passo 1: Crie a pasta static
Na raiz do projeto (onde está o manage.py), crie uma pasta static.
Passo 2: Configure o settings.py

Abra meuprojeto/settings.py.
No final do arquivo, você provavelmente já tem esta linha:
STATIC_URL = 'static/'
Esta é a URL pública (ex: meusite.com/static/). Agora, precisamos dizer ao Django onde a pasta *física* está no seu computador. Adicione esta linha abaixo da STATIC_URL:

```
# meuprojeto/settings.py
...
STATIC_URL = 'static/'

# Adicione esta linha
STATICFILES_DIRS = [BASE_DIR / 'static']
```

Passo 3: Crie seu arquivo CSS
Dentro da pasta static/, vamos organizar as coisas. Crie uma subpasta css e, dentro dela, um arquivo style.css.
Estrutura: static/css/style.css

```
/* static/css/style.css */

body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    color: #333;
    line-height: 1.6;
}

h1 {
    color: #005a9c; /* Um azul Django */
}

ul {
    background-color: #fff;
    border: 1px solid #ddd;
    padding: 15px;
}

li {
    list-style-type: square;
}
```

Passo 4: Carregue o CSS no Template
Como ligamos o home.html ao style.css? Não fazemos isso:
<link rel="stylesheet" href="css/style.css"> (Errado!)
Precisamos que o Django nos diga a URL correta. Fazemos isso em 2 etapas no home.html:

Edite o templates/home.html:

```
{% load static %}

<!DOCTYPE html>
<html lang="pt-br">
<head>
    ...
    <title>Meu Site Lindo</title>

    <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>
<body>
    ...
</body>
</html>
```

Passo 5: Teste!
Pare o servidor (Ctrl+C) e rode-o novamente (python manage.py runserver).
Nota: Você geralmente precisa reiniciar o servidor após alterar o settings.py.
Acesse http://127.0.0.1:8000. Sua página agora deve ter estilos!

6. Módulo 5 (Bônus): Herança de Template (O DRY!)

Nossa home.html tem *todo* o HTML (head, body, etc.). Se fizéssemos uma página "Sobre", teríamos que copiar e colar tudo isso. Isso viola o princípio DRY (Não se Repita).

O Django resolve isso com **Herança de Template**.

Passo 1: Crie o "molde mestre" (base.html)
Vamos criar um arquivo base.html que terá a estrutura principal do site.
Crie templates/base.html:

```
{% load static %}
<!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 %}Meu Site Django{% endblock %}</title> <link rel="stylesheet" href="{%
static 'css/style.css' %}">
</head>
<body>
    <header>
        <nav>
            <a href="/">Home</a> |
            </nav>
    </header>

    <main>
        {% block content %}
            {% endblock %}
    </main>

    <footer>
        <p>© 2025 - Meu Site Incrível</p>
    </footer>
</body>
</html>
```

{% block ... %} define "buracos" que as páginas "filhas" podem preencher.

Passo 2: "Herde" o molde em home.html
Agora, vamos simplificar drasticamente o templates/home.html.

```
{% extends 'base.html' %}

{% block title %}Página Inicial - Meu Site{% endblock %}

{% block content %}
    <h1>Olá, {{ nome_usuario }}!</h1>

    <p>Este HTML está separado do meu código Python.</p>

    <h3>Tecnologias que estou aprendendo:</h3>

    <ul>
        {% for tec in tecnologias %}
            <li>{{ tec }}</li>
        {% endfor %}
    </ul>

    {% if nome_usuario == 'Júnior' %}
        <p>Este é um conteúdo especial para o Júnior!</p>
    {% endif %}
{% endblock %}
```

Veja como este arquivo ficou limpo! Ele só contém o que é *único* desta página.

Passo 3: Teste!
Recarregue http://127.0.0.1:8000. O visual será idêntico, mas agora você tem um header, um footer e sua estrutura está pronta para crescer.