Lab: Arquivos Estáticos em Python Django
Introdução ao Desenvolvimento Web

Departamento de Computação e Matemática
Universidade de São Paulo

—

Prof. Dr. Denis M. L. Martins

2026

O que são Arquivos Estáticos?

Arquivos estáticos são arquivos que:

- Não mudam dinamicamente no servidor
- São enviados diretamente ao navegador
- Definem aparência e comportamento visual

Exemplos: - CSS - JavaScript - Imagens - Ícones

Como o Django trata arquivos estáticos?

Django utiliza:

- Pasta `static`
- Configuração `STATIC_URL`
- Template tag `{% static %}`

Considere a estrutura de projeto e app abaixo:

```
myproject/
|
|--- myapp/
|   |--- static/
|   |   |--- css/
|   |   |   |__ style.css
|   |   |__ images/
|   |
|   |--- templates/
|   |   |__ index.html
```

Para carregar arquivos estáticos (css e javascript), você vai precisar modificar o arquivo `settings.py` para incluir:

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR,  'static')
```

Ao adicionar `STATIC_URL` e `STATIC_ROOT`, estamos dizendo ao Django onde ele deve procurar todos os arquivos que não são código Python ou HTML puro. `STATIC_URL` é o caminho público que será usado no navegador, enquanto `STATIC_ROOT` é o diretório físico no servidor.

Exemplo

Crie um arquivo `myapp/static/css/style.css`

```
body {
    background-color: #f5f5f5;
    font-family: Arial;
}

h1 {
    color: darkblue;
}
```

Crie um template HTML `templates/index.html`:

```
{% load static %}

<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>
<body>
    <h1>Minha Aplicação Django</h1>
    <p>Exemplo utilizando CSS.</p>
</body>
</html>
```

Note que `{% load static %}` habilita o uso de arquivos estáticos no template.

Fluxo de Funcionamento

1. Navegador solicita página HTML
2. Django renderiza template
3. Template referencia arquivo CSS
4. Navegador solicita CSS
5. Django envia arquivo estático

Erros Comuns

Esquecer:

`{% load static %}`

Estrutura de pastas incorreta

ERRADO:

`templates/static/css`

CORRETO:

`static/css`

Caminho incorreto:

`href="style.css"`

Correto:

`href="{% static 'css/style.css' %}"`

Organizando Arquivos Estáticos

Separar por tipo:

```
static/
|--- css/
|--- js/
|--- images/
```

Exemplo com Imagem:

```
<img src="{% static 'images/logo.png' %}">
```

Exemplo com JavaScript:

```
<script src="{% static 'js/script.js' %}"></script>
```

Arquivos Estáticos em Produção

Durante desenvolvimento:

- Django serve arquivos automaticamente

Em produção:

- Normalmente usamos:
  - Nginx
  - CDN
  - WhiteNoise

E executamos o comando abaixo para reunir todos os arquivos estáticos:

```
python manage.py collectstatic
```

Boas Práticas

- Organizar arquivos por categoria
- Utilizar nomes claros
- Evitar CSS inline
- Separar HTML da estilização

  Para mais informações sobre como Django trabalha com arquivos estáticos, veja https://docs.djangoproject.com/en/6.0/ho
  files/

Exercício 1

Modifique a landing page para incluir:

- Uma nova seção “Sobre nós”
- Uma imagem ilustrativa
- Um botão secundário
- Um quarto card com outra funcionalidade da aplicação

Exercício 2

Neste exercício, você deverá baixar localmente os arquivos **CSS** e **JavaScript** do Bootstrap 5.3 e utilizá-los em uma aplicação Django. Ao final, a aplicação deverá exibir uma landing page baseada no exemplo **Cover** do Bootstrap.

Referência: https://getbootstrap.com/docs/5.3/examples/cover/

Resultado esperado

Ao final do exercício, ao acessar a página inicial do projeto, o navegador deverá exibir uma landing page parecida com o exemplo **Cover** do Bootstrap, utilizando arquivos Bootstrap armazenados localmente no projeto Django.

Passo 1: Criar a estrutura de arquivos estáticos

Dentro da sua aplicação Django, crie a seguinte estrutura:

```
myfirstapp/
|
|--- static/
|   |--- css/
|   |__ js/
|
|--- templates/
|   |__ index.html
```

Passo 2: Baixar o Bootstrap 5.3

Acesse o site oficial do Bootstrap:

https://getbootstrap.com/docs/5.3/getting-started/download/

Baixe a versão compilada do Bootstrap.

Depois de extrair o arquivo `.zip`, localize os seguintes arquivos:

```
bootstrap.min.css
bootstrap.bundle.min.js
```

Copie os arquivos para dentro do projeto Django:

```
myfirstapp/static/css/bootstrap.min.css
myfirstapp/static/js/bootstrap.bundle.min.js
```

Passo 3: Baixar o exemplo Cover

Acesse:

https://getbootstrap.com/docs/5.3/examples/cover/

Abra o código-fonte da página ou utilize a opção de download dos exemplos do Bootstrap.

Copie o conteúdo HTML do exemplo **Cover** para o arquivo:

```
myfirstapp/templates/index.html
```

Passo 4: Adaptar o template para Django

No início do arquivo `index.html`, adicione:

```
{% load static %}
```

Depois, substitua o link do CSS do Bootstrap por:

```
<link href="{% static 'css/bootstrap.min.css' %}" rel="stylesheet">
```

Substitua também o JavaScript do Bootstrap por:

```
<script src="{% static 'js/bootstrap.bundle.min.js' %}"></script>
```

Passo 5: Ajustar o CSS específico do Cover

O exemplo Cover normalmente utiliza um CSS adicional chamado `cover.css`.

Crie o arquivo:

`myfirstapp/static/css/cover.css`

Adicione nele o CSS específico do exemplo Cover.

Depois, no `index.html`, carregue esse arquivo:

```
<link href="{% static 'css/cover.css' %}" rel="stylesheet">
```

Passo 6: Criar a view

No arquivo `myfirstapp/views.py`, crie a função:

```
from django.shortcuts import render

def index(request):
    return render(request, 'index.html')
```

Passo 7: Criar a rota da aplicação

No arquivo `myfirstapp/urls.py`, adicione:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Passo 8: Conectar a aplicação ao projeto

No arquivo principal `urls.py` do projeto, adicione:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myfirstapp.urls')),
]
```

Passo 9: Executar o servidor

No terminal, execute:

`python manage.py runserver`

Acesse:

`http://127.0.0.1:8000/`