Aula 03

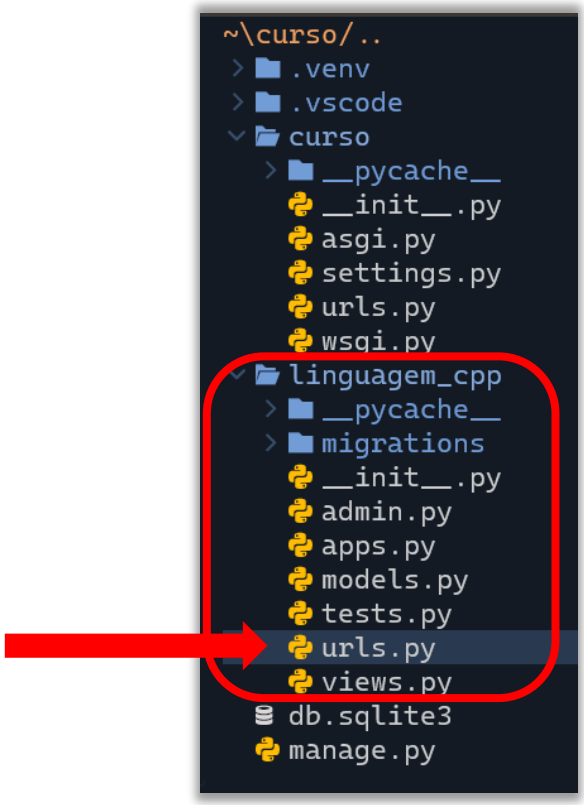
Django

urls – part Deux

Como visto anteriormente, agora nossas funções (que representam as páginas do aplicativo) estão no lugar correto, dentro do arquivo **linguagem_cpp/views.py** e as suas chamadas no arquivo **curso/urls.py**.

Agora imagine o seguinte cenário, o nosso projeto é sobre cursos de programação. Cada aplicativo terá diversas páginas web. Fica prático realizar a importação de todas essas páginas diretamente no **curso/urls.py**? Não fica. Inclusive pode ficar bem desorganizado se formos adicionar alguma página em um determinado aplicativo. Fica demorada a reorganização.

Para resolver esse problema, adicionamos uma cópia do arquivo **curso/urls.py** no módulo **linguagem_cpp**.



Só temos que alterar sua estrutura interna, para atender às necessidades do aplicativo onde está. A começar pela referência da importação da view. Agora, podemos importar apenas com um ponto. Também temos que remover a página de admin pré-cadastrada.

Veja como vai ficar:

```
C:\Users\gutoh\curso\linguagem_cpp\urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.view_home), # início
    path('sobre/', views.view_sobre), # sobre
    path('contato/', views.view_contato), # contato
]
```

Também temos que atualizar nosso **curso/urls.py** com a referência do **linguagem_cpp/urls.py**.
Para chamada correta, temos que incluir a função include, e assim passar para ele uma string com a localização do arquivo **urls.py** do aplicativo **linguagem_cpp**.

Veja como vai ficar:

```
C:\Users\gutoh\curso\curso\urls.py
7 from django.contrib import admin
6 from django.urls import path, include
5
4 urlpatterns = [
3     path('admin/', admin.site.urls),
2     path('', include('linguagem_cpp.urls')), # urls.py de linguagem_cpp
1 ]
8
```

Repare que todas as páginas são filhas da página principal, não tem qualquer outro caminho antes.

http://127.0.0.1:8000/
http://127.0.0.1:8000/sobre/
http://127.0.0.1:8000/contato/

Isso acontece porque deixamos a string do path vazia. Se quisermos que essas páginas sejam subpáginas de um subdomínio, adicionamos um valor à string do path:

```
C:\Users\gutoh\curso\curso\urls.py
7 from django.contrib import admin
6 from django.urls import path, include
5
4 urlpatterns = [
3     path('admin/', admin.site.urls),
2     path('cpp/', include('linguagem_cpp.urls')), # urls.py de linguagem_cpp
1 ]
8
```

Agora, nossos links ficaram como abaixo, pois adicionamos um subdomínio para nosso projeto:

http://127.0.0.1:8000/cpp/
http://127.0.0.1:8000/cpp/sobre/
http://127.0.0.1:8000/cpp/contato/

Também já podemos adicionar outros aplicativos (subdomínios) ao nosso projeto (website) de maneira muito mais organizada.

Referenciando o HTML

Até o momento, não temos praticamente nada de HTML, apenas a string que estamos trabalhando.

Nós podemos adicionar código HTML diretamente nela, mas isso deixaria o processo de criação e manutenção da página extremamente trabalhoso.

Para resolver isso, o Django já tem métodos próprios de realizar esse trabalho. De começo, o código HTML não é colocado dentro da string (como fizemos até então), e nós vamos usar uma função para nos auxiliar no processo.

A função em questão é a render do django.shortcuts. Se reparamos, ela já está presente no arquivo ***/views.py** quando executamos o startapp.

Como ela vai funcionar:

- ela é quem vai ser retornada nas nossas funções da **views.py** (até então estávamos retornando um objeto da classe HttpResponse);

- ela vai receber dois argumentos:
  - o request recebido na função;
  - o nome do nosso arquivo html;

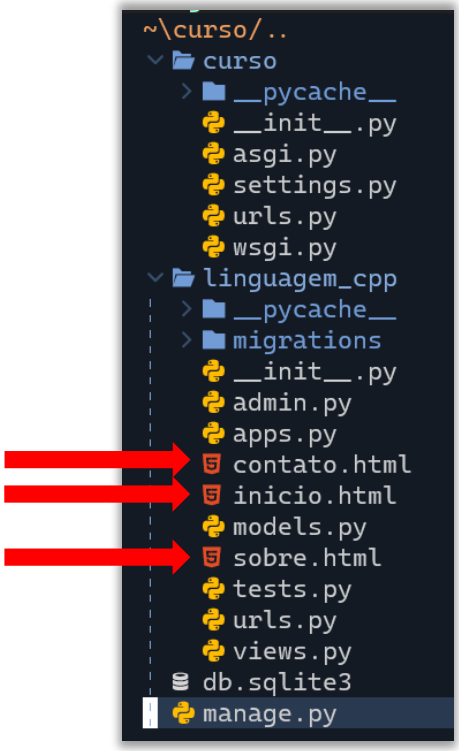
Veja como vai ficar nosso **linguagem_cpp/views.py**:

```
C:\Users\gutoh\curso\linguagem_cpp\views.py
 12 # from django.http import HttpResponse
 11 from django.shortcuts import render
 10
  9 # Create your views here.
  8 def view_home(request):
  7     return render(request, 'inicio.html')
  6
  5 def view_sobre(request):
  4     return render(request, 'sobre.html')
  3
  2 def view_contato(request):
  1     return render(request, 'contato.html')
 13
```

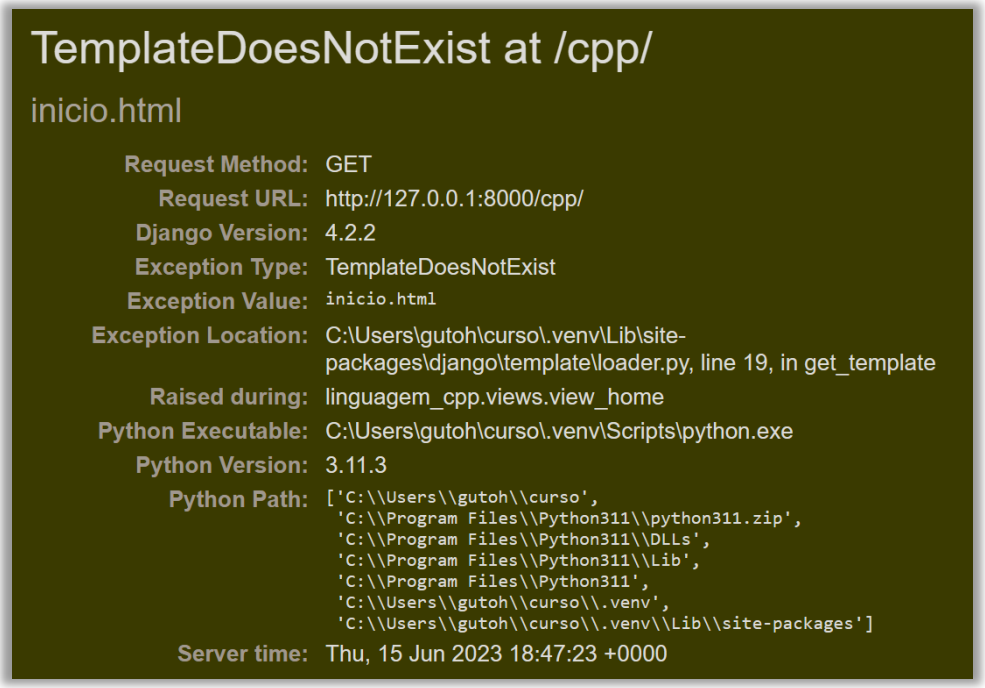
Repare que agora não precisamos mais do HttpResponse. Então podemos comentar ou remover ele. Aproveite agora e crie os arquivos HTML para cada uma das páginas.

Dica: no arquivo HTML, digite apenas ! (o sinal de exclamação) e então selecione a opção mostrada, que será gerado um modelo de código HTML.

Veja como vai ficar a organização do nosso aplicativo com os arquivos HTML:

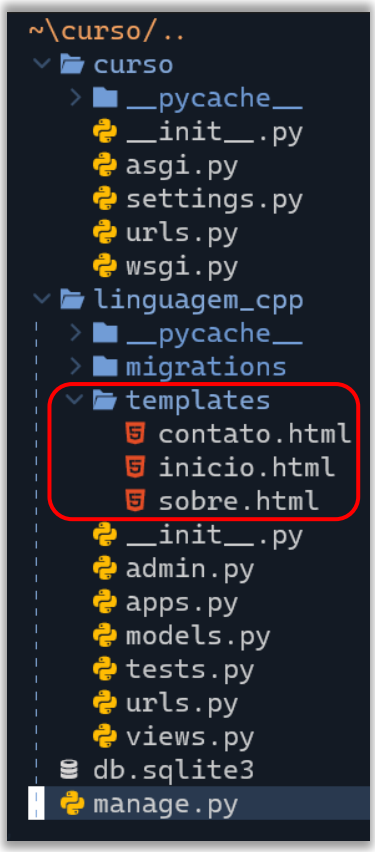


Mas, se recarregarmos agora nossas páginas, iremos ver um novo tipo de erro:



Repare no tipo de erro. TemplateDoesNotExist. O Django está esperando um template de nós. Mas o que seria um template? Ela é uma pasta onde ficarão nossos arquivos HTML e ela, por usa vez, ficará localizada junto dos arquivos do aplicativo.

Vamos criá-la e colocar nossos HTMLs lá para dentro. Veja como vai ficar:



Esse nome não é gratuito. Esse nome de pasta é o que o Django procura (por padrão) quando quer encontrar os arquivos HTMLs.

Mas, mesmo fazendo os passos acima, o erro TemplateDoesNotExist ainda persiste. Isso porque, o Django ainda não sabe que temos um aplicativo chamada **linguagem_cpp**, logo, ele não consegue localizar a pasta template previamente criada.

Mas, como então nossas páginas estavam sendo executadas? Estavam sendo, porque a referência dos arquivos que estávamos usando para mostrá-las era controlado pelo Python (lembra da linha **from linguagem_cpp.views import view_inicio**? E a passagem da string diretamente para o HttpResponde?) e o que estávamos mostrando eram strings (mesmo adicionando HTML nelas).

Se repararmos no arquivo **curso/settings.py**, tem uma constante chamada INSTALLED_APPS. É nela que temos que adicionar nosso aplicativo. É através dela que o Django localiza os aplicativos, as pastas com a nomenclatura padrão (como a templates), arquivos HTMLs, CSSs, JavaScripts etc.

Veja como vai ficar a constante após a alteração:

```
C:\Users\gutoh\curso\curso\settings.py
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    # meus aplicativos
    'linguagem_cpp',
]
```

Ele vai ser inserido ao final da lista como uma string.

Agora, finalmente, se recarregarmos nossa página HTML, veremos ela sendo exibida como esperado.

Passando Variáveis ao HTML

Vamos ao arquivo **linguagem_cpp/views.py** e ver mais de perto a função render.

Ela tem um parâmetro chamado context, que é usado para enviar um dicionário ao nosso documento HTML. Dessa forma, podemos enviar variáveis do Python para nossas páginas web. Para exibir essas variáveis, temos que usar uma notação diferenciada **DENTRO** do documento HTML. Nós temos que colocar a chave do dicionário entre chaves duplas.

Veja o exemplo abaixo:

```
C:\Users\gutoh\curso\linguagem_cpp\views.py
# from django.http import HttpResponse
from django.shortcuts import render

# Create your views here.
def view_home(request):
    return render(request, 'inicio.html', context={
        'nome': 'C++'
    })

def view_sobre(request):
    dicionario = {'descricao': 'é uma linguagem muito bacana!'}
    return render(request, 'sobre.html', context=dicionario)

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

Acima, temos duas formas de enviar o dicionário para o parâmetro context:

1. diretamente no parâmetro, como visto na função view_home;
2. por variável, como visto na função view_sobre;

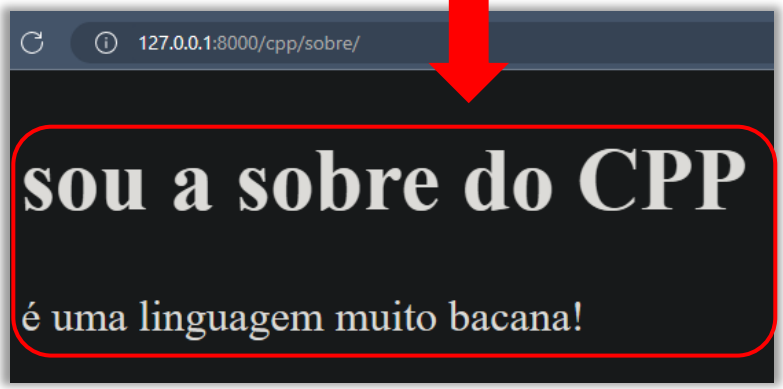
Veja como ficam os arquivos HTMLs e as páginas recarregadas:

C:\Users\gutoh\curso\linguagem_cpp\templates\inicio.html

```
12 <!DOCTYPE html>
11 <html lang="en">
10 <head>
 9     <meta charset="UTF-8">
 8     <meta name="viewport">
 7     <title>Início</title>
 6 </head>
 5 <body>
 4     <h1>sou a home do CPP</h1>
 3     <p>Linguagem {{nome}}</p>
 2 </body>
 1 </html>
13
```

C:\Users\gutoh\curso\linguagem_cpp\templates\sobre.html

```
12 <!DOCTYPE html>
11 <html lang="en">
10 <head>
 9     <meta charset="UTF-8">
 8     <meta name="viewport">
 7     <title>Sobre</title>
 6 </head>
 5 <body>
 4     <h1>sou a sobre do CPP</h1>
 3     <p>{{ descricao }}</p>
 2 </body>
 1 </html>
13
```



Exercícios para Praticar

1. Se ainda estiver inseguro quanto aos passos feitos até agora, execute novamente o passo a passo da aula 01, 02 e 03 de uma só vez.
2. Se já estiver seguro quando às aulas 01 e 02, repita os passos apenas dessa aula.
3. Crie diferentes páginas HTML para seus aplicativos e os organize na pasta template de cada um.
4. Crie diversos dicionários e os envie para suas páginas HTML, exibindo seus valores.
5. Carregue um arquivo json de sua escolha e exiba seu conteúdo nas suas páginas HTML.