Web framework python

João Vitor Rezende Moura - 1221184773

Django

UNIT - Universidade Tiradentes

Abstract

O desenvolvimento Web é uma necessidade de grande nos dias atuais, e será futu- ramente, com isso, visa-se a necessidade da criação e utilização de ferramentas que proporcionem uma melhor qualidade, velocidade e adaptabilidade dos projetos. Com isso, o web framework Django foi uma das soluções que utiliza da linguagem python para fazer o devesenvolvimento de aplicações Web.

Keywords: Django, Web, Framework, Desenvolvimento, Python, MVT

---

1 Oque é Django?

O Django é um web framework que segue a arquitetura MVT, e com ele podemos criar aplicações web com maior facilidade e rapidez, pelo fato de ele cuidar da maioria dos detalhes intrísecos do desenvolvimento web. Como uma das suas características, é que ele é de uso livre, e possui código open - source. Dentre algumas das ferramentas que o Django inclui, podemos listar algumas como:

- Site maps
- Authentication
- Context Administrator
- RSS Feeds

Além disso, possui ferramentas que possibilitam uma maior segurança da aplicação, e muitas outras ferramentas voltadas para o desenvolvedor.

1.1 Instalação e inicialização do framework

Dentro do python, é comum a utilização de ambientes virtuais, pois ele facilita e organiza o uso de bibliotecas que podem ser baixadas para o uso da nossa aplicação. Com isso, para inicializarmos nosso ambiente virtual e instalar o framework no ambiente virtual, podemos usar as seguintes instruções bash

```
$ python -m venv nome_ambiente_virtual

# A depender da plafatorma de desenvolvimento escolhida,
# devemos ativar o ambiente virtual de forma diferente,
# nesse caso, iremos instruir com base nas plataformas
# MacOs e Linux/Unix2 Instalação e inicialização do framework

$ source ./nome_ambiente_virtual/bin/activate
$ pip install django
```

cosntruirmos uma aplicação Django. Para criarmos a aplicação, devemos usar o documando:

```
$django-admin startproject nomeProjeto
```

Na parte final do comando temos por exemplo a opção de adicionar o projeto dentro de um diretório específico que será criado para o projeto ou dentro do diretório que o comando está sendo executado, para definir isso, devemos deixar em branco, ou adicionar um ponto, respectivamente. Após essas configurações, instalações e comandos, podemos usbir o Django sem sequer necessitar criar alguma interface, pois ele possui uma tela inicial, que nos comunica se a instalação e criação do nosso projeto foi feito de maneira correta. Para isso, podemos usar o seguinte comando:

```
$python manage.py runserver
```

Com isso, estaremos rodando nossa aplicação, e ela pode ser acessada dentro do localhost, na porta 8000, que são definidos por padrão, mas caso necessário, podemos ser modificados nas configurações. Posteriormente a isso, temos que nosso projeto foi criado com uma esrtu- tura específica, que vai ser destrinchada a seguir:

1.2 Estrutura dos arquivos criados pelo Django

Os arquivos que foram criados dentro do nosso projeto possuem características distintas dos outros devido á funcionalidade que lhe é atribuida, eles se car- acterizam por serem:

- **manage.py:** Esse é o arquivo utilitário da CLI do django, que é usado como django-admin. Ele nos permite interagir com o projeto de várias maneiras.2 Instalação e inicialização do framework
- **db.sqlite3:** É nossa base de dados, por configuração o SQLite é a nossa base de dados por ser leve e ser integrado com o python por natureza, não necessitando de configurações para que funcione desde o início. Outras bases de dados podem substituir a mesma posterior- mente, mas para teste, ela é recomendada.
- **__init__.py:** Arquivo de inicialização do pacote. Nosso projeto é desenvolvido como um pacote python, sendo assim, quando o mesmo é chamado, é necessário um arquivo de inicialização para que o mesmo seja executado na chamada do nosso pacote.
- **wsgi.py:** Um entry-point para servidores web que são compatíveis com WSGI (Web Server Gateway Interface)
- **asgi.py:** Funciona como o WSGI, mas de forma assíncrona
- **settings.py:** Arquivo de configurações do nosso projeto, nos permite configurar até a forma como as variáveis de ambiente do projeto se comportam, até mesmo em qual pasta os arquivos estáticos se local- izam.
- **urls.py:** As declarações de URL do nosso projeto Django funcionam como uma tabela, que permite especificarmos as URLs que podem ser acessadas no nosso site.

Cada um desses arquivos possuem configurações padrões, que são mínimas para o funcionamento do Django.

1.3 Django.settings e funcionamento do framework

O funcionamento do framework está intrísecamento ligado ao seu arquivo de configurações, pois é ele que faz o "meio de campo" entre todas as funcional- idades da ferramenta, por isso merece uma atenção em especial. Dentro dele teremos logo de início, a nossa ligação com o banco de dados, que é feito por meio da biblioteca pathlib, que define a pasta a qual se localiza nossa base de dados que foi criada junto com o projeto. Temos que o django funciona por meio da ideia de "apps", como se cada parte da nossa aplicação fosse um app dedicado, ou ela fosse um app como um todo, e isso pode ser definido por meio de subprojetos dentro do qual estamos atualmente — esse tópico será tocado com mais detalhes e calma posteriormente — temos também que o funcionamento do Django permite a presença de middlewares

1.4 Criando as primeiras urls no Django

Após criarmos nossa primeira aplicação, e vermos que sua instalação foi feita de maneira correta e completa, podemos assim criar nossas primeiras URLs, para que elas sejam acessadas dno navegador. Devemos nos direcionar ao arquivo de `urls.py`, dentro dela temos o seguinte:

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

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

Esse código permite que nós nos comuniquemos com os servidores por meio do protocolo HTTP (Hyper Text Transfer Protocol), que nos permite fazer requisições dos dados ao servidor, e ele nos responder com as informações desejada.

Esse protocolo possui uma variação que é o HTTPS, que é sua versão que possui uma camada extra de segurança, que permite uma maior responsabilidade do lado do desenvolvedor e uma maior segurança do lado do cliente.

Com isso, podemos assim, implementar uma URL customizada da seguinte forma:

```
from django.contrib import admin
from django.urls import path
from django.http import HttpResponse

def view(request):
  return HttpResponse('Voltando código')

urlspatterns = [
  path('blog/', view)
]
```

Assim, quando criarmos essa função e essa url, vamos poder acessar essa url no nosso navegador, e ele vai prover o conteúdo disponibilizado pela função para que nós possamos usar ele para disponibilizar conteúdo para a requisição

2 Criando nossas primeira aplicações web

2.1 Criando apps com o manage.py

Os apps dentro do Django, são como uma subcategoria do projeto, que podem ser criadas várias dentro de um mesmo projeto, e isso é muito útil quando temos por exemplo, um projeto que deve ter duas formas de aplicação com- pletamente diferentes, como um e-commerce que deve ter a parte de compra e venda de produtos feita pelos usuários, e a parte administrativa das con- tas que não tem acesso as informações de compra e venda (necessariamente diretamente), mas que é usada para ver as estatísticas de acesso e uso do site. Tendo esse exemplo em mente, visa-se a necessidade e a utilidade do uso de apps dentro do Django, para criarmos o nosso, devemos usar o comando:

```
$python3 manage.py startapp nome_do_app
```

Quando criarmos nosso app, teremos numa nova pasta que com ela ter- emos assim uma nova forma de criar nosso site, e organizar ele da maneira mais fácil.

Arquivo views.py e configuração de urls Quando criarmos o app da nossa aplicação, temos dentro dele um arquivo em particular que não está no projeto como um todo, que é o arquivo de views.py, dentro dele podemos colocar as views da nossa aplicação, e permitir um funcionamento mais clean e melhor da nossa aplicação, além do fato de que no arquivo de urls, organizarmos ele apenas com os paths das nossas urls. Temos também que podemos aninhar as urls e portanto as views da nossa aplicação. Para isso, devemos ir dentro do nosso app e especificar as views e criar o arquivo de urls.py, além de configurar as urls dentro do arquivo do projeto em sim, como a seguir:

projeto/urls.py

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

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

blog/views.py

```
from django.http import HttpResponse

def blog(request):
  return HttpResponse("retornando da página inicial do blog")
```

blog/urls.py

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

urlspatterns = [
  path('', views.blog),
]
```

3 Renderizando templates, html e configurando o INSTALLED_APPS

A internet tem como padrão para a criação e visualização de sites, o uso de três tecnologias principais, o HTML, CSS e Javascript, sendo as duas primeiras linguangens de marcação e a tereceira uma linguagem de pro gramação. Com isso, para servimos esses tipos de arquivos, precisamos es- pecificar a existência, e a entrega deles para o request. Para fazermos isso, precisamos usar a função de render do django, como mostrado a seguir:

blog/views.py

```
from django.shortcuts import render

def home(request):
  return render(
            request,
          'pastanamespace/arquivo.html')
```

Tendo em vista esses elementos, devemos posteriormente dentro do arquivo de configurações, adicionar nosso app na lista de apps instalados, para que quando o django execute, ele possa achar nosso app, e nos permitir acessar o mesmo pelo navegador

settings.py

```
INSTALLED\_APPS = [
  'djang.contrib.admin',
  ...,
  'django.contrib.staticfiles',
  'home'
]
```

Temos então nosso app localizado dentro da nossa aplicação. Com tudo, um erro comum dentro do django quando possuímos dois templates com o mesmo nome, apesar deles se localizarem em locais diferentes, é que um deles seja escolhido para ser servido aleatoriamente. Com o intuito de evitar isso, é recomendado que dentro da nossa pasta do app, e dentro da pasta de templates do app, adicionemos uma outra pasta com o nome do nosso app, e coloquemos lá dentro nosso templates, criando assim um **namespace**, elemento que evita que esses erros ocorram.

Com isso, no final teremos dentro de cada app nosso, uma pasta que deve conter os templates referentes àquela aplicação, e dento dessa pasta de templates, teremos a pasta de namespace com nossos arquivos de html.

3.1 Configuração de templates globais e Herança de Templates

Quando estamos criando um projeto, podemos criar pastas para separarmos os templates os quais utilizamos, e podemos criar pastas dentro dessa pasta de template, para organizar esses templates confome a sua funcionalidade.

Assim, quando formos usar determinados arquivos para a construção da nossa aplicação web, podemos fazer com que esses arquivos sejam reutilizados em outras partes do nosso site, pois podemos extender esses arquivos, como será mostrado a seguir:

index.html

```
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
<body>
  <h1>Escrevendo um título na página</h1>
</body>
</html>
```

page.html

```
{%extends "pastanamespace/index.html"%}
```

É bom que nós quando formos criar as pastas e formos criar nossos templates, dentro da pasta de template criarmos novamente outra pasta, mas que deve er em si especificamente o nome do aplicativo o qual ela faz parte, pois isso previne que nós criemos problemas de conflito de busca dos templates pois eles funcionam como namespaces.

3.2 Arquivos parciais e o comando include

Dentro da forma como trabalhamos com os templates dentro do Django, podemos ter que esses templates podem ser separados em diversos arquivos, e que conforme as necessidades da aplicação ele são montados com determinados elementos. Para podermos usar esses elementos, podemos criar uma pasta dentro da pasta global, com todos os arquivos parciais do nosso projeto, por exemplo, e a partir disso criar um sistema de "baterias" dentro do nosso projeto.

Esse tipo de funcionaldiade é permitida por causa do comando `include` do django, o qual permite que nós incluamos determinadas partes de de código quando necessário

```
{% include 'global/partials/head.html'%}

<h1>
O head do texto la na inclusão, ou seja. não há necessidade de digitá-lo
</h1>
</body>
</html>
```

3.3 Trabalhando com arquivos estáticos

Quando estamos trabalhando com arquivos estáticos, como fotos por exemplo, precisamos especificar a sua existência e seu tipo de funcionamento para o django, e podemos fazer isso definido dentro da pasta a qual falamos que seria a base das templates, uma nova pasta que será onde iremos armazenar nossos arquivos estáticos.

Para acessar os arquivos estáticos dentro da nossa página HTML, precisamos primeiramente carregar os arquivos por meio do comando `load`, e ao carregar nossa pasta, podemos usar os arquivos por meio do comando:

```
{% load static %}<!DOCTYPE html>
<html lang="pt-br">
<head>
<meta charset="UTF-8">
<title>Document</title>
<link rel="stylesheet" href="{% static 'nomedapasta/arquivo'}">
</head>
```

3.4 usando o context para enviar dados para dentro dos templates