

# Engenharia de Dados e Conhecimento

## Introdução à Plataforma Django

# Introdução



- Django é uma plataforma gratuita e open source, escrita em Python, para o desenvolvimento de aplicações web
- Tomou o nome de um famoso guitarrista “Django Reinhardt”
- É mantida pela Django Software Foundation (DSF), uma organização independente
- Fomenta o desenvolvimento rápido, limpo e pragmático
- Segue a arquitetura MVC
- Criada em 2003, tornou-se open source em 2005
- Versão atual: 3.1

# Caraterísticas

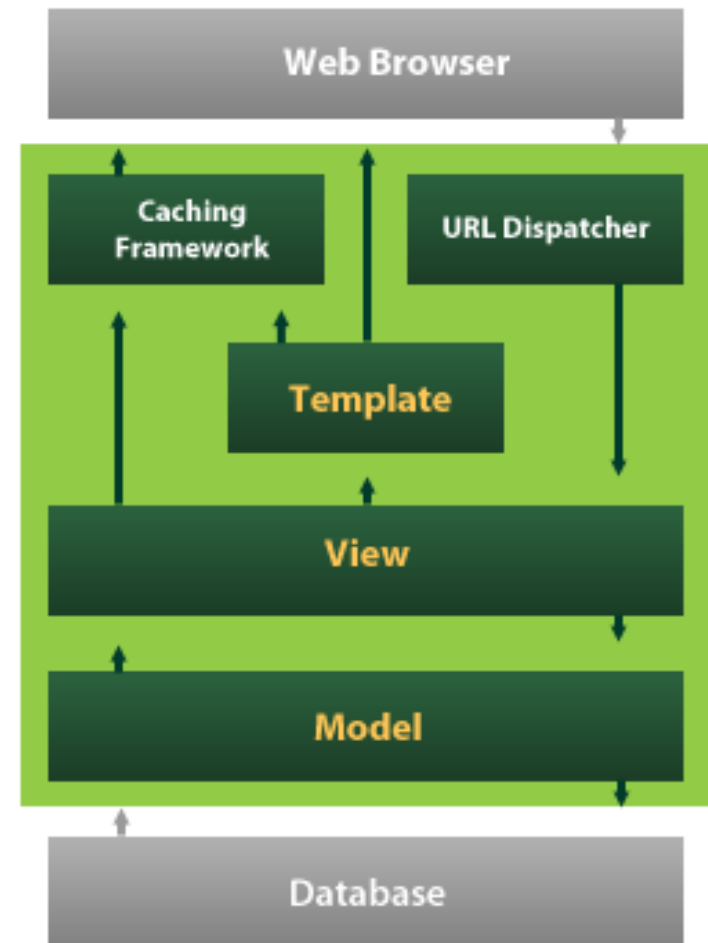


- Arquitetura de software MVC (MVT, na verdade)
- Possui um ORM (*Object Relational Mapper*) para processar dados
- Focada na automatização, aderindo ao princípio DRY (Don't Repeat Yourself)
- Usa um sistema de templates
- Sistema de personalização Admin, para facilitar o CRUD
- Desenho elegante de routing de URLs
- Possui um light web server embutido (para testes)
- Possibilita a utilização de middleware personalizado
- Possui facilidades para: autenticação, internacionalização e caching

# Arquitetura



<b>Models</b>	Descreve os dados
<b>Views</b>	Controla o que os utilizadores vêem
<b>Templates</b>	Controlo como o vêem
<b>Controller</b>	Expedidor de URLs



# Estrutura do Projeto Django



```
webproj/ ----- Pasta para o projeto. Pode ter qualquer nome.  
manage.py -- Utilitário em commando de linha para interagir com o projeto.  
webproj/ --- Pacote do projeto. Nome usado para imports  
    __init__.py --- Ficheiro que define esta pasta como um pacote, em Python.  
    settings.py --- Configurações do projeto Django.  
    urls.py ----- Mapping/routing das URLs para este projeto.  
    wsgi.py ----- Um ponto de entrada para webservers compatíveis com WSGI.  
app/ ----- Aplicação web individual, podendo coexistir várias.  
    templates/ ---- Ficheiros HTML, invocados pelas views.  
    static/ ----- CSS, JS, imagens, etc. - configurável em "settings.py"  
    __init__.py  --  
    views.py ----- Recebe os pedidos dos clientes e devolve as respostas.  
    models.py ----- Modelos dos dados.  
    admin.py ----- Criação automática de interface para o modelo de dados.  
    forms.py ----- Permite a receção de dados enviados pelos clients.
```

# Settings



- O ficheiro settings.py do projeto Django sobrepõe-se ao ficheiro `<python>/Lib/sitepackages/django/conf/global_settings.py`
- Atributos:
  - DEBUG # True ou False
  - DATABASES ENGINE # 'mysql', 'sqlite3', 'oracle' ... etc.
  - ROOT\_URLCONF # Configuração de routing das URLs
  - MEDIA\_ROOT # Para ficheiros enviados pelo utilizador (user-uploaded)
  - MEDIA\_URL # Para ficheiros multimedia
  - STATIC\_ROOT # Pasta para ficheiros estáticos como CSS, JS, ...
  - STATIC\_URL # Pasta de ficheiros estáticos
  - TEMPLATE\_DIRS # Pasta de templates

# Referências



- Adrian Holovaty, Jacob Kaplan-Moss, “The Definitive Guide to Django: Web Development Done Right”, Apress, 2008.
- Django Software Foundation, “Django Documentation”, Release 3.1, 2020.
- Documentação  
(<https://docs.djangoproject.com>)



# Plataforma Django

## Criação de Um Projeto





New Project

Pure Python  
**Django**  
Flask  
Google App Engine  
Pyramid  
Web2Py  
Scientific  
Angular CLI  
AngularJS  
Bootstrap  
Foundation  
HTML5 Boilerplate  
React App  
React Native

Location: C:\data\Teaching\2019.2020-edc\01\webproj

Project Interpreter: Python 3.7

☐ New environment using Virtualenv

Location: C:\data\Teaching\2019.2020-edc\01\webproj\venv

Base interpreter: Python 3.7 C:\Program Files\Python37\python.exe

☐ Inherit global site-packages  
☐ Make available to all projects

☒ Existing interpreter

Interpreter: Python 3.7 C:\Program Files\Python37\python.exe

More Settings

Template language: Django

Templates folder: app\templates


Application name: app

☒ Enable Django admin

Create Cancel

# Execução da Aplicação Web



- Execução através do PyCharm
  - opção Run/Run 'project' (Shift + F10)
  - ou no icon 
  - e clicar no link `http://127.0.0.1:8000`
- Ou execução em comando linha
  - dentro da pasta do projeto executar:
    - “python manage.py runserver”
    - e abrir o browser com a URL: <http://localhost:8000>



# Plataforma Django

*Views*

# View - Criação



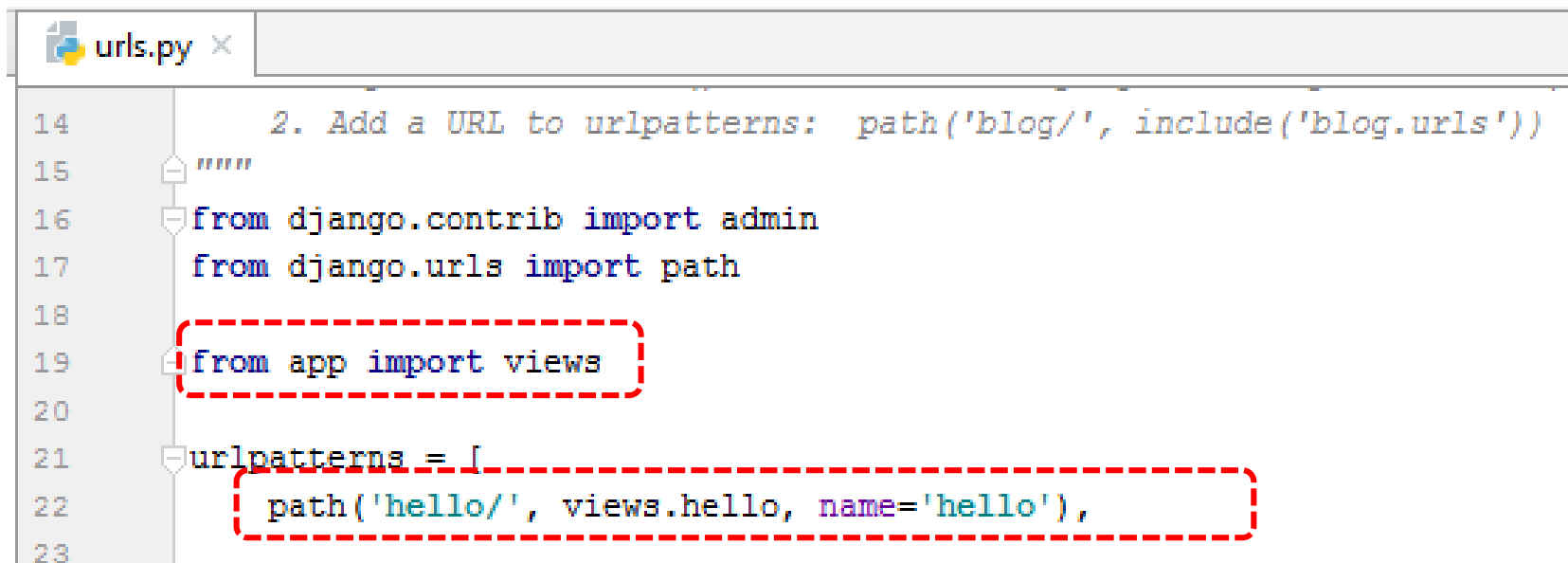
- No ficheiro “app/views.py” inserir uma view através da definição de uma função
- Exemplo:

```
views.py x
1 from django.shortcuts import render
2 from django.http import HttpRequest, HttpResponse
3 from datetime import datetime
4
5 # Create your views here.
6 def hello(request):
7     return HttpResponse("Hello World!!!")
8
```

# Configuração da URL



- No ficheiro “Nome\_Projeto/urls.py” inserir uma *route* para a *view*



```
14      2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15      """
16      from django.contrib import admin
17      from django.urls import path
18
19      from app import views
20
21      urlpatterns = [
22          path('hello/', views.hello, name='hello'),
23
```

# View - Nova



- No ficheiro “app/views.py” inserir mais uma *view function*:

```
views.py x
1 from django.shortcuts import render
2 from django.http import HttpRequest, HttpResponse
3 from datetime import datetime
4
5 # Create your views here.
6 def hello(request):
7     return HttpResponse("Hello World!!!")
8
9
10 def numero(request, num):
11     resp = "<html><body><h1>{}</h1></body></html>".format(num)
12     return HttpResponse(resp)
13
```

# Configuração da nova URL



- No ficheiro “Nome\_Projeto/urls.py” inserir mais uma *route* para a *view*

```
13 1. import the include() function: from django.urls import incl
14 2. Add a URL to urlpatterns: path('blog/', include('blog.urls
15 """
16 from django.contrib import admin
17 from django.urls import path
18
19 from app import views
20
21 urlpatterns = [
22     path('hello/', views.hello, name='hello'),
23     path('numero/<int:num>', views.numero, name='numero'),
24 ]
```



# Plataforma Django

## *Templates*



# Template - Criação



- Na pasta “templates”, criar o ficheiro “numerot.html”

```
numerot.html x
1  {% extends "layout.html" %}
2
3  {% block content %}
4
5      <h1>O seu número é:</h1>
6      <h2>{{ num_arg }}</h2>
7      <br />
8      <p><b>Um marcador template simples:</b></p>
9      {% ifequal num_arg 1000 %}
10         <p>O nome do valor é MIL.</p>
11     {% else %}
12         <p>O nome do valor é Desconhecido.</p>
13     {% endifequal %}
14
15 {% endblock %}
```

Argumento/Variável

Template Tags

# Template – Nova View



- No ficheiro “app/views.py” inserir mais uma *view function*:

```
views.py x
10 def numero(request, num):
11     resp = "<html><body><h1>{}</h1></body></html>".format(num)
12     return HttpResponse(resp)
13
14
15 def numerot(request, num):
16     tparams = {
17         'num_arg': num,
18     }
19     return render(request, 'numerot.html', tparams)
20
```

# Configuração da nova URL



- No ficheiro “Nome\_Projeto/urls.py” inserir mais uma *route* para a *view*

```
urls.py x
15
16 from django.contrib import admin
17 from django.urls import path
18
19 from app import views
20
21 urlpatterns = [
22     path('hello/', views.hello, name='hello'),
23     path('numero/<int:num>/', views.numero, name='numero'),
24     path('numerot/<int:num>/', views.numerot, name='numerot'),
25
```



# Plataforma Django

## *Static Files*

# Static Files

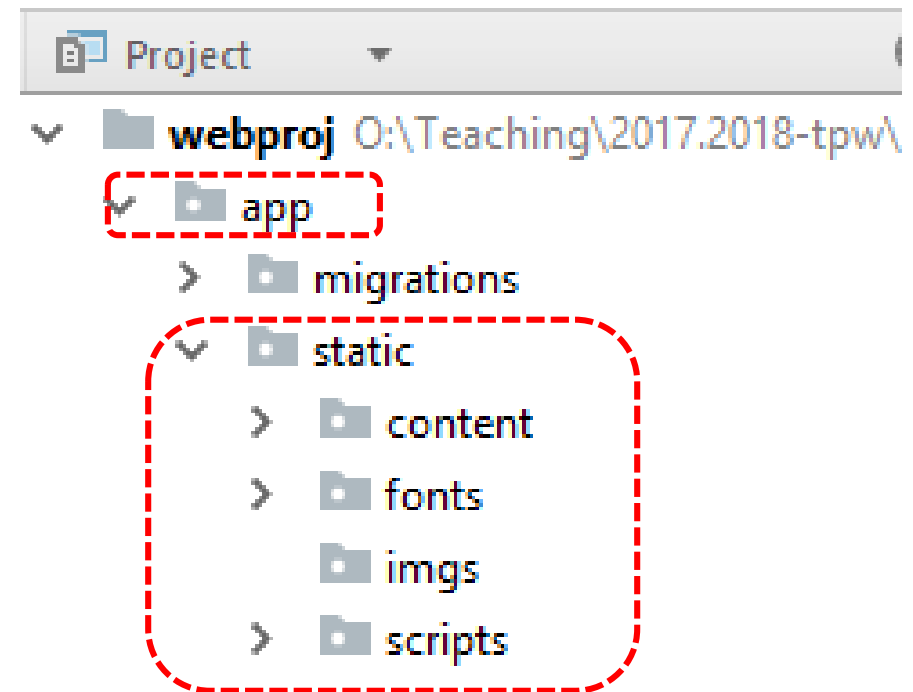


- As *static files* são ficheiros que se pretende simplesmente referenciar e servir ao cliente, sem qualquer processamento prévio.
- O seu acesso é público, pois o cliente apenas necessita de possuir a URL para os mesmos.
- Exemplos:
  - Imagens (jpg, png, etc.)
  - Style Sheets (CSS)
  - Scripts (JavaScript)

# Static Files - Localização



- Os ficheiros denominados por *static files* residem em pastas pré-determinadas, dentro ou fora da “app”.
- Exemplo:
  - Dentro da pasta “app/static”
  - Encontram-se as pastas “content”, “fonts”, “scripts”
  - Podem ser adicionada ainda outras pastas como a pasta “imgs”



# Static Files - Configuração



- No ficheiro “settings.py”:
  - o módulo `'django.contrib.staticfiles'` deve aparecer nas aplicações instaladas “INSTALLED\_APPS”
  - o prefixo da URL para *static files* deve estar definido:
    - `STATIC_URL = '/static/'`
  - a pasta das *static files* deve estar definido:
    - `STATIC_ROOT = os.path.join(BASE_DIR, 'app/static')`
- Documentação:
  - Em modo debug e produção
    - <https://docs.djangoproject.com/en/1.8/howto/static-files/>
    - <https://docs.djangoproject.com/en/1.8/howto/static-files/deployment/>

# Static Files - Uso



- Referência de recursos, modo 1:

- Este modo usa URLs absolutos

```
<link rel="stylesheet" href="static/content/style.css" />
<script src="static/scripts/jquery-1.10.2.min.js"></script>
<script src="static/scripts/main.js"></script>
```

- Referência de recursos, modo 2:

- Este modo usa URLs relativos

```
{% load static %}
<link rel="stylesheet" href="{% static "content/style.css" %}" />
<script src="{% static "scripts/jquery-1.10.2.min.js" %}"></script>
<script src="{% static "scripts/main.js" %}"></script>
```





# Plataforma Django

*Forms*

# Receção de Dados



- O objeto “request” do tipo “HttpRequest” permite aceder a um conjunto vasto de dados que são recebidos pelo *web server*
- Esses dados podem ser acedidos diretamente através de alguns atributos e métodos dedicados, como:
  - request.path, request.get\_host(), request.is\_secure()
- Ou podem ser acedidos através do dicionário “request.META” que contém toda a informação presente no *header* do protocolo HTTP

# Receção de Dados



- Exemplo de uma *view* para mostrar todos os dados presentes no *header* HTTP

```
views.py x
from django.shortcuts import render, render_to_response
from django.http import HttpResponseRedirect

def info(request):
    values = request.META.items()
    html = []
    for k, v in values:
        html.append('<tr><td>%s</td><td>%s</td></tr>' % (k, v))
    return HttpResponseRedirect('<table>%s</table>' % '\n'.join(html))
```

# Forms



- Os *Forms* são os elementos HTML por excelência para o envio/receção de dados do cliente para o servidor
- Do lado do browser (cliente), o *Form* pode usar o método *Get* ou o método *Post*, para enviar os dados nele contidos
- Do lado do servidor, a *view* invocada pode recorrer aos dicionários “request.GET” e “request.POST” para aceder aos dados recebidos

# Forms - exemplo



- Definição da *template* para envio dos dados:

```
send_info.html x
1 {% extends "layout.html" %}
2
3 {% block content %}
4
5 {% if error %}
6     <p style="...">ERRO: Insira a informação pedida.</p>
7 {% endif %}
8 <form action="." method="post">
9     {% csrf_token %}
10    <p>Nome: <input type="text" name="nome" /></p>
11    <p>Idade: <input type="text" name="idade" /></p>
12    <p><input type="submit" value="Enviar"/></p>
13 </form>
14
15 {% endblock %}
```

# Forms - exemplo



- Definição da *template* para os resultados:

```
send_results.html x
1 {% extends "layout.html" %}
2
3 {% block content %}
4
5     <p>Nome da Pessoa: {{ nome }}</p>
6     <p>Idade da Pessoa: {{ idade }}</p>
7
8 {% endblock %}
9
```

# Forms - exemplo



- Exemplo da criação de um *Form* para o envio de dados de uma pessoa
- Definição da URL:

```
urls.py x
17
18 from django.contrib import admin
19 from django.contrib.auth import views as auth_views
20 from django.urls import path, include
21
22 from app import views
23
24 urlpatterns = [
25     path('login/', auth_views.LoginView.as_view(template_name='login.html'), name='login'),
26     path('logout', auth_views.LogoutView.as_view(next_page='/'), name='logout'),
27
28     path('', views.home, name='home'),
29     path('contact/', views.contact, name='contact'),
30     path('about/', views.about, name='about'),
31     path('sendinfo/', views.sendinfo, name='sendinfo'),
32 ]
```

# Forms – exemplo (i)



- Definição da *view* de envio de dados:

```
views.py x
37 def sendinfo(request):
38     """Renders sendinfo page."""
39     assert isinstance(request, HttpRequest)
40     if 'nome' in request.POST and 'idade' in request.POST:
41         nome = request.POST['nome']
42         idade = request.POST['idade']
43         if nome and idade:
44             return render(
45                 request,
46                 'send_results.html',
47                 {
48                     'nome': nome,
49                     'idade': idade,
50                 }
51             )
52     else:
```



# Forms – exemplo (ii)



- Definição da *view* de envio de dados (cont.):

```
views.py x
51         )
52     else:
53         return render(
54             request,
55             'send_info.html',
56             {
57                 'error': True,
58             }
59         )
60     else:
61         return render(
62             request,
63             'send_info.html',
64             {
65                 'error': False,
66             }
67         )
68
```