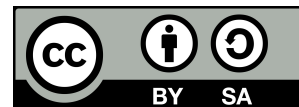


CURSO PYTHON

APLICACIONES WEB CON DJANGO



Autor: Jon Vadillo
www.jonvadillo.com



Contenidos

1. Introducción y **fundamentos básicos**
2. Crea tu primer **proyecto** en Django
3. Crea tu primera **aplicación** en django
4. El **modelo** en Django, **acceso a datos** y la aplicación de administrador
5. **Vistas y plantillas en Django**
6. **Vistas basadas en clases** (Class Based Views)
7. **Formularios** en Django

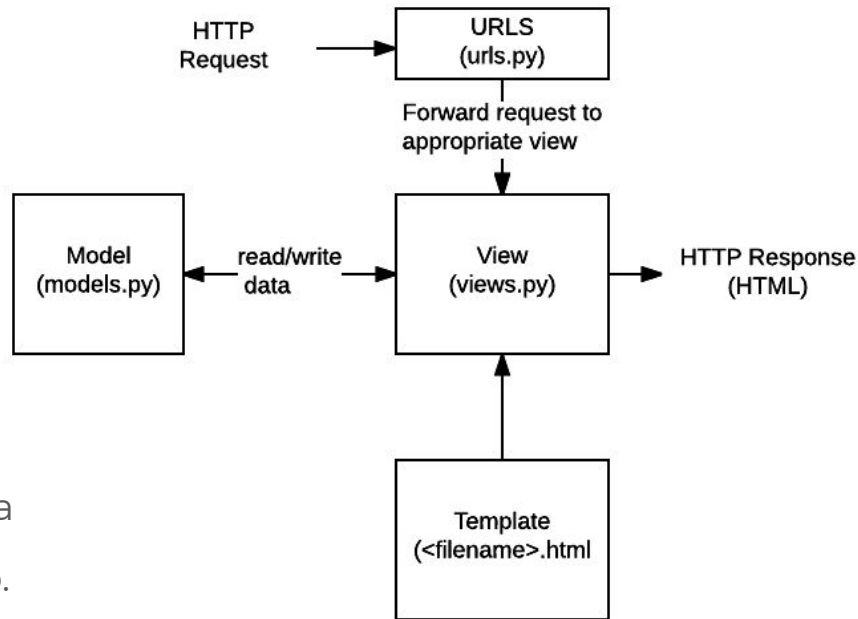
5. Vistas y plantillas en Django

Vistas en Django

```
def mi_vista(request):  
    return HttpResponse("Hola a todos!")
```

Mostrar información con vistas

- Una vista es **una función que:**
 - Procesa** una consulta HTTP.
 - Trae datos** desde la base de datos cuando los necesita.
 - Genera una página HTML** renderizando estos datos usando una **plantilla**.
 - Retorna el HTML** en una respuesta HTTP para ser mostrada al usuario.



Pasos para crear vistas

1. Modificar los mapeadores de URLs (`urls.py`) para **reenviar las URLs** admitidas a las funciones de vista apropiadas.
2. **Implementar las funciones de vista** para obtener los datos solicitados desde los modelos, crear una página HTML que muestre los datos, y devolverlo al usuario para que lo vea en el navegador.
3. **Crear las plantillas** usadas por las vistas para renderizar los datos.

Definir las vistas y sus rutas

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

urlpatterns = [
    # ej: /miApp/
    path('', views.index, name='index'),
    # ej: /miApp/empresas/
    path('<str:nombre_empresa>/', views.empresa, name='empresa'),
    # ej: /miApp/empresa/5/
    path('/empresas/<int:id_empresa>', views.detalle, name='detalle')
]
```


Definir las vistas y sus rutas

```
def index(request):  
    return HttpResponse("Hello, world!")  
  
def detalle(request, nombre_empresa):  
    return HttpResponse("Consultando la empresa %s." % nombre_empresa)  
  
def actualizar(request, id_empresa):  
    response = "Información de la empresa con ID = %s."  
    return HttpResponse(response % id_empresa)
```

Hands on!

1. Crear el proyecto y la aplicación
2. Crear los modelos (**models.py**)
3. Modificar los mapeadores de URLs (**urls.py**).
4. Implementar las funciones de vistas (**views.py**)

Nota: utiliza la aplicación de administración para insertar datos



Vistas útiles de verdad

```
from django.shortcuts import render

from .models import Question

#devuelve el listado de empresas
def index(request):
    empresas = Empresa.objects.order_by('nombre')
    output = ', '.join([e.nombre for e in empresas])
    return HttpResponse(output)
```

Hands on!

Crea 3 vistas que realicen los siguiente:

/ Devuelve todas las empresas existentes ordenadas por el nombre

/id/ Devuelve los detalles de una empresa a partir de su ID

/nombre/ Devuelve el listado de empresas cuyo nombre coincide

Nota: utiliza la aplicación de administración para insertar datos



Hands on!



<https://github.com/jvadillo/curso-django-paso-a-paso>



¿Qué ocurre si no existe el ID?

- Toda aplicación web tiene que ser capaz de **capturar los errores** y **gestionarlos** de forma adecuada para el usuario.

```
def detail(request, id_empresa):  
    try:  
        empresa = Empresa.objects.get(pk=id_empresa)  
    except Empresa.DoesNotExist:  
        raise Http404("La empresa no existe")  
    return HttpResponse(output)
```

get_object_or_404()

- Llama al método `get()` del modelo in lanza la excepción `Http404` en caso de capturar una excepción de tipo `DoesNotExist`.

```
def detail(request, id_empresa):  
    empresa = get_object_or_404(Empresa, pk=id_empresa)  
    return HttpResponse(output)
```

- De igual forma existe una función **`get_list_or_404()`** que lanza la excepción `Http404` si la lista está vacía

Hands on!

Actualiza el código del ejercicio anterior para que utilice los métodos que nos permiten realizar el control de errores 404 de forma automática.



Hands on!



<https://github.com/jvadillo/curso-django-paso-a-paso>



Plantillas (templates)

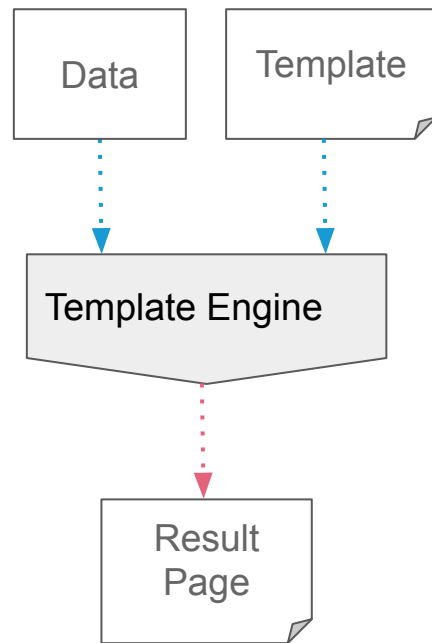
```
<p>Hola {{nombre}}</p>
```



```
<p>Hola Marta</p>
```

Plantillas (templates)

- Los **motores de plantillas** (templates engines) leen un fichero de texto (**plantilla**) que **define la estructura o diseño** e insertan en él los **datos de forma dinámica**.
- Las plantillas incluyen **variables** que serán reemplazadas por la información y **etiquetas** para controlar la lógica



Ejemplo de plantilla

hola.html

```
{% if login %}  
    <p>Hola {{nombre}} </p>  
{% else %}  
    <p>Hola usuario anónimo</p>  
{% endif %}
```

views.py

```
def index(req):  
    nombre = 'Amaia'  
    context = {  
        'login': True  
        'nombre': nombre  
    }  
    return render(req,  
        hola.html', context)
```

Django Template Language

- **Variables:** cuando el motor de plantillas se encuentra una variable **la reemplaza por su valor.**

```
{{ variable }}           {{object.attribute}}
```

- **Filtros:** permite **modificar la variable** a reemplazar (mostrar en minúsculas, mostrar las primeras X palabras, mostrar la longitud,...)

```
{{ name|lower }}           {{ bio|truncatewords:30 }}
```

Django Template Language

- **Etiquetas (tags):** más complejos que las variables, algunas generan texto de salida, ejecutan una lógica, cargan otra plantilla,...

```
{% tag %} ... tag contents ... {% endtag %}
```

Nota: Más información en <https://docs.djangoproject.com/en/2.2/ref/templates/>

Django Template Language

```
<ul>
{% for usuario in usuario_list %}
    <li>{{ usuario.nombre }}</li>
{% endfor %}
</ul>
```

Django Template Language

```
{% if usuario_list %}
    Número de usuarios: {{ usuario_list|length }}
{% elif admin_list %}
    Hay administradores en la aplicación.
{% else %}
    No hay nadie en la aplicación.
{% endif %}
```


Actualizar la vista para que utilice la plantilla

```
def index(request):
    ultimas_preguntas = Question.objects.order_by('-pub_date')[:5]
    template = loader.get_template('polls/index.html')
    context = {
        'ultimas_preguntas': ultimas_preguntas,
    }
    return HttpResponse(template.render(context, request))
```

- Django buscará plantillas en un directorio llamado 'templates' de tu aplicación.

Atajo render()

```
def index(request):  
    ultimas_preguntas = Question.objects.order_by('-pub_date')[:5]  
    context = {  
        'ultimas_preguntas': ultimas_preguntas,  
    }  
    return render(request, 'polls/index.html', context)
```

Hands on!

Mejora la aplicación del ejercicio anterior haciendo uso de plantillas en las vistas. Incluye enlaces en las plantillas para navegar desde la lista al detalle (y vuelta a la lista).



Hands on!



<https://github.com/jvadillo/curso-django-paso-a-paso>



Herencia en plantillas

- Permite **reutilizar código** en distintas plantillas.
- Se define una **plantilla base** que contiene el “esqueleto” con todos los **elementos comunes** y definir bloques que puedan **sobreescritos** por las plantillas que la hereden.

```
{% block content %}{% endblock %}
```

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" href="style.css">
  <title>{% block title %}My amazing site{% endblock
%}</title>
</head>

<body>
  <div id="content">
    {% block content %}{% endblock %}
  </div>
</body>
</html>
```

holamundo.html



Nota: “extends”
siempre debe
indicarse al
comienzo para que
la herencia
funcione.

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

{% block title %}My amazing blog{% endblock %}

{% block content %}
{% for entry in blog_entries %}
    <h2>{{ entry.title }}</h2>
    <p>{{ entry.body }}</p>
{% endfor %}
{% endblock %}
```

Hands on!

Mejora la aplicación anterior creando una plantilla base con la estructura de la página que sea utilizada por el resto de plantillas.



Hands on!



<https://github.com/jvadillo/curso-django-paso-a-paso>




Utilizar los nombres de las URLs

- En lugar de utilizar la ruta de la URL, podemos utilizar el nombre que le hemos dado en el mapeo definido en urls.py.



```
<li><a href="/miApp/{{ empresa.id }}/">{{ empresa.nombre }}</a></li>
```

```
<li><a href="{% url 'detalle' empresa.id%}">{{ empresa.nombre }}</a></li>
```



```
# el valor de 'name' indicado en {% url %}  
path('<int:id_empresa>/', views.detalle, name='detalle'),
```

Referenciar contenido estático

- Las páginas por lo general cargan contenido estático JS, CSS, etc.
- Para referenciar estáticos desde cualquier plantilla primero incluimos la expresión `{% load staticfiles %}` y a partir de ahí ya se puede comenzar a utilizar `{% static %}` para hacer referencia a la ubicación de los archivos estáticos.

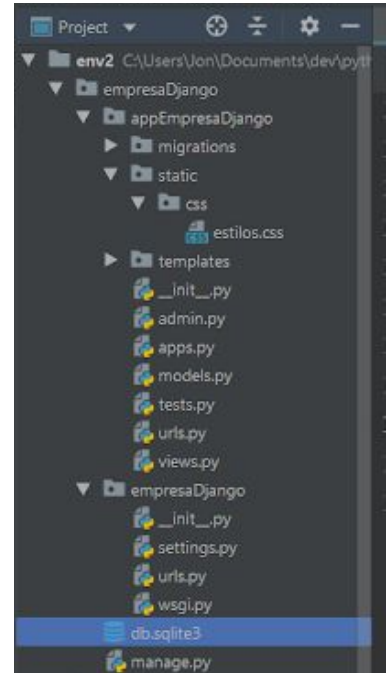
```
{% load staticfiles %}
...
<link rel="stylesheet" href="{% static 'css/styles.css' %}">
...

```

Referenciar contenido estático

- Los archivos por norma general se almacenarán en una carpeta llamada 'static' dentro de nuestra aplicación.
- Es necesario especificar una valor para las variables **STATICFILES_DIRS** (directorio donde se ubican los archivos estáticos) y **STATIC_URL** (URL donde se publicarán) en el fichero **settings.py**

```
STATIC_URL = '/static/'  
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
```



Hands on!

Es hora de añadir un poco de estilo. Añade los archivos estáticos necesarios para utilizar Bootstrap en tu aplicación.



Sources

- Documentación oficial: <https://www.djangoproject.com/>
- Mozilla MDN Web Docs: <https://developer.mozilla.org/>