Vistas

Antonio Espín Herranz

Vistas

- **Una vista en Django** está representada por **una función** que:
  - Recibe, un objeto de tipo: **HttpRequest** (representado por un parámetro que **por convención** se llama **request**) y
  - Devuelve un objeto **HttpResponse**.

- Las vistas también se pueden representar mediante **clases**.

- Las vistas se ubican en el fichero **views.py**

- Por otro lado, en **urls.py** necesitamos definir un mapeo entre una **URL** y la **vista**.

Vistas

- Dentro de ese objeto HttpResponse podemos devolver HTML o simplemente una cadena.
  - **OJO**, esto es una **primera aproximación** , **posteriormente** utilizaremos **plantillas**.

- Dentro del fichero **urls.py** se **declaran urls** que se asocian a **funciones**.
  - *Se establece un mapeo entre la URL de la petición y la función que responde a esa petición*.

- **Django** promueve el **acoplamiento débil** entre la **URL** y la **función** que queremos ejecutar.

urls.py

- Por defecto, cuando se crea un proyecto, este fichero habilita el panel de administración.

- Está definiendo una URL que cuando la solicita el cliente mostrará el panel de administración:
  - http://localhost:8000/**admin/**

```
from django.contrib import admin
from django.urls import path
urlpatterns = [
    path('admin/', admin.site.urls),
]
```

Prueba

- Cuando hemos creado el proyecto con:
  - ***django-admin.py startproject nombre_proyecto***
  - ***cd nombre_proyecto***

- Y lanzamos el comando:
  - ***python manage.py runserver***

- Y en un navegador:
  - http://localhost:8000
  - Django emite una respuesta por defecto.

- Si ejecutamos el comando:
  - ***python manage.py migrate***
  - Crea todo el contenido en la BD para gestionar el panel de administración.
  - Se podría hacer la petición:
  - http://localhost:8000/admin/ **(se comenta más adelante)**

Ejemplo: 1ª Vista

- Se necesitan **2 cosas**:
  - La **función que representa la vista**:
    - En el fichero: views.py
  - Y **el mapeo de una URL a esta vista**:
    - En el fichero: urls.py
  - Crear un proyecto y dentro de este: ***la aplicación***.

Ejemplo: 1ª Vista

- **La vista:**

```
from django.http import HttpResponse
def hola(request):
    return HttpResponse("Hola Mundo")
```

- **El mapeo:**

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

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

No tiene porqué coincidir
El nombre de la función
Con la cadena de la petición (URL)

Las funciones en python
Son objetos, no se las llama

En la var. **Urlpatterns** se definen
Las **URLs** y las funciones asociadas.

¿Acoplamiento débil?

- En Django existe un acoplamiento débil entre el mapeo y la función.

- La URL puede tener el formato que queramos (dentro de unas pautas) y la función se puede llamar como queramos.
  - Sólo tiene que recibir como mínimo un parámetro (que representa la request).

- **Se puede modificar la URL o la función sin que los cambios en una de estas dos partes afecte a la otra → Acoplamiento Débil**

Notas

- Podemos utilizar **distintas URLs para la misma función**.

- **Si no hay una entrada** en la variable Urlpatterns que cuadre con la petición del cliente, se lanzará un **error Http 404 → recurso no encontrado**.

- En el desarrollo del proyecto, esta información se utiliza para depurar, pero en ***PRODUCCIÓN ESTO SE DEBE DESHABILITAR***.
  - Este comportamiento se da por la variable **DEBUG=True** dentro del fichero **settings.py**

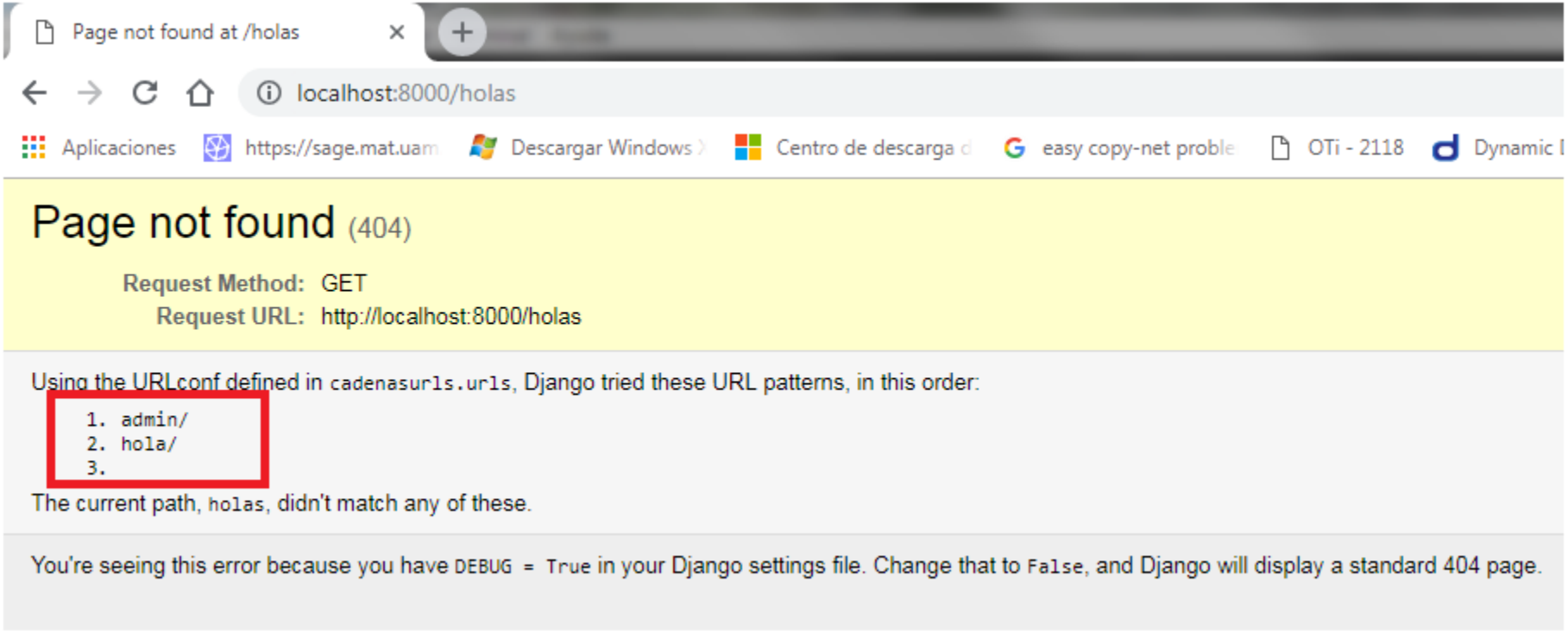
Notas

- Las funciones de la vista pueden recibir parámetros.

- En la función de la vista, el primer parámetro será **request** (del tipo **HttpRequest**).

- Los siguientes parámetros se rellenarán con parámetros que pasemos por la URL.

Error 404



En producción puede mostrar demasiada información.
Por ejemplo, las URLs que están mapeadas.

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

En settings.py

El Servidor

- Hace el **despliegue en caliente,** es decir, no tenemos que reiniciarlo para que tome los nuevos cambios.

- Si añadimos **nuevas vistas** NO es necesario reiniciar el server.

Volcando HTML

- *Existe la posibilidad de volcar directamente el HTML pero **NO es eficiente**, se utilizarán las plantillas para tal cometido.*

```
HTML = """
<!DOCTYPE html>
<html lang="es">
<head>
<title>Hola mundo</title>
</head>
<body>
<div id="summary">
<h1>¡Hola Mundo!</h1>
</div>
</body></html> """“

def hola2(request):
    return HttpResponse(HTML)
```

Se añadiría un nuevo patrón: En **urls.py**
path('hola2/', ejemplos.views.hola2),

Procesamiento de la petición

- Cuando se arranca el servidor se importa el fichero **settings.py**, dentro de este fichero hay una variable configurada: **ROOT_URLCONF** que apunta al fichero **urls.py**
  - **ROOT_URLCONF = 'cadenasurls.urls‘**

- **<u>Cuando se realiza la petición:</u>**
  - **Django compara** la petición con las entradas de la variable **urlpatterns (en el fichero urls.py)**
    - Si hay una entrada que coincide lanza la vista correspondiente enviando un objeto HttpRequest y esta es responsable de devolver un objeto HttpResponse.
    - Si no encuentra ninguna URL lanzará una excepción (error 404)

Tipos de URLs

- Las URLs pueden ser estáticas, o dinámicas.
- Se puede recoger parámetros a partir de las urls y se pueden pasar a las funciones vistas, para que hagan una cosa u otra.
- Django fomenta la **urls** "**bonitas**", a diferencia de otros lenguajes servidor que necesitan recibir los parámetros con este formato:
  - ***Mipagina.php?param1=valor1&param2=valor2***
- En Django podemos utilizar:
  - ***Mipagina/valor1/valor2***
  - valor1 y valor2 se pasarán a las funciones de las vistas.

Recibir parámetros

- Para recibir parámetros por la **URL** tenemos que utilizar **<>**.
  - Se indica el nombre del parámetro y luego se define una vista que reciba ese parámetro.
  - Si los nombres no coinciden da un fallo en ejecución.

- De forma opcional se puede indicar un **conversor**, y django convertirá el parámetro al tipo indicando.
- Si no se indica un convertidor, por defecto será texto.

```
from django.urls import path
from . import views
urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    path('articles/<int:year>/', views.year_archive),
    path('articles/<int:year>/<int:month>/', views.month_archive),
    ]
```

Convertidores

- Pueden ser:
  - **int**: para números enteros
  - **str**: o si no se indica el tipo, también son cadenas. Cualquier cadena, salvo la **/**
  - **slug**: es una cadena que consta de letras o números ASCII, guión y subrayado.
  - **uuid**: por ejemplo:

    075194d3-6885-417e-a8a8-6c931e272f00
  - **path**: Para que coincida con una cadena completa:
    - clientes/externos/listar

Convertidores a medida

- También se pueden definir convertidores a medida.

- Mediante una clase que cumpla las siguientes características:
  - Un **atributo**: **regex** con una expresión regular.
  - Y dos métodos:
    - **to_python**:
      - Utilizado para pasar a la función de la vista, si no se cumple el formato se lanzará una excepción ValueError.
    - **to_url**:
      - Para manejarlo dentro de la URL
  - Después hay que **registrar el conversor** con la **función**:
    - **register_converter** donde se indica el nombre de la clase y el nombre del tipo que se le asignará al parámetro en la URL.
    - Los conversores pueden estar en otro módulo aparte.

Ejemplo

```
class FourDigitYearConverter:
    regex = '[0-9]{4}'
    def to_python(self, value):
        return int(value)
    def to_url(self, value):
        return '%04d' % value
```

En este caso en vez de tener un int
Tiene que cumplir la expr. Regular
De 4 dígitos.
Permite afinar más en el filtro de
Parámetros.

```
from django.urls import path, register_converter
from . import converters, views

register_converter(converters.FourDigitYearConverter, 'yyyy')

urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    path('articles/<yyyy:year>/', views.year_archive), ...
]
```

PRACTICA Nº 5: VISTAS

Expresiones Regulares

- También se pueden utilizar expresiones regulares dentro de los patrones de las **URLs**.

- Dentro del módulo: **django.urls** además de **path()** disponemos de la función **re_path()**

- Que admite expresiones regulares en la URL. Estas tienen que ser **compatibles con el módulo re** de Python.
  - Las cadenas empezarán por **r'…'** (cadena **raw**) para no tener que escapar las \

re_path

```
from django.urls import include, re_path

urlpatterns = [
    re_path(r'^index/$', views.index, name='index'),
    re_path(r'^bio/(?P<username>\w+)/$', views.bio, name='bio'),
    re_path(r'^weblog/', include('blog.urls')), ...
    re_path(r'^pruebas/(\d{3,4})/$', …)
]
```

- **^index/$**
  - Empieza por index y termina con la /
- **^bio/(?P<username>\w+)/$**
  - Empieza por bio y recibe un parámetro opcional de 1 o más caracteres alfanuméricos.
- **^weblog/**
  - La petición empieza por weblog/ y lo que sigua: **weblog/**news
- **^pruebas/(\d{3,4})/$**
  - Empieza por pruebas y recibe un número con 3 o 4 cifras.

re

Símbolo	Coincide con
. (punto)	Cualquier carácter
\d	Cualquier dígito
[A-Z]	Cualquier carácter, A-Z (mayúsculas)
[a-z]	Cualquier carácter, a-z (minúsculas)
[A-Za-z]	Cualquier carácter, a-z (no distingue entre mayúscula y minúscula)
+	Una o más ocurrencias de la expresión anterior (ejemplo, \d+ coincidirá con uno o más dígitos)
[^/]+	Cualquier carácter excepto la barra o diagonal.
?	Cero o una ocurrencia (ejemplo \d? coincidirá con cero o un digito
*	Cero o más ocurrencias de la expresión anterior (ejemplo, \d* coincidirá con cero o más dígitos)
{1,3}	Entre una y tres (inclusive) ocurrencias de la expresión anterior (ejemplo \d{1,3} coincidirá con uno, dos o tres dígitos)

Parámetros de path / re_path

- Ambas funciones tienen los mismos parámetros.

  path(route, view, kwargs=None, name=None)

  re_path(route, view, kwargs=None, name=None)

  – **Kwargs**: Se puede utilizar para pasar parámetros adicionales a la función de la vista.

Ejemplo: Pasar parámetros adicionales

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

urlpatterns = [
    path('blog/<int:year>/', views.year_archive, {'foo': 'bar'}),
]
```

- La petición podría ser algo así: **/blog/2005**
- La vista tendría esta declaración:
  - **views.year_archive**(request, **year**=2005, **foo**='bar')

- Los **parámetros adicionales** se pasan con un **diccionario**.

Parámetros de path / re_path

- **name**: El nombre asociado a la vista.
  - Luego ese **nombre** lo utilizaremos en las **plantillas** para poder generar las **URLs con etiquetas**.
  - No es obligatorio que el nombre de la vista coincida con la URL, pero a veces por convención se utiliza.
    - path('**index**/', views.index, name='**main-view**'),
    - path('**bio**/<username>/', views.bio, name='**bio**'),

Ejemplo


```
path('', dpii.views.index, name='index'),
path('facturas/', dpii.views.vistaFacturas, name='facturas'),
path('ingresos/', dpii.views.vistaIngresos, name='ingresos'),
```

```
<ul>
    <li><a href="{% url 'index' %}">Principal</a></li>
    <li><a href="{% url 'facturas' %}">Facturas</a></li>
    <li><a href="{% url 'ingresos' %}">Ingresos</a></li>
</ul>
```

Otras funciones

- También añade la función **include**() indica una ruta de importación completa de Python a otro módulo **URLconf**, para que se incluya.

- Para proyectos grandes **include**() nos permite tener separados los mapeos en otros ficheros.

- Otra posibilidad de **include**() es **añadir un iterable de instancias** de **path** / **re_path**.

Ejemplo: incluir un iterable / otro fichero

```
from django.urls import include, path
from apps.main import views as main_views
from credit import views as credit_views

extra_patterns = [
    path('reports/', credit_views.report),
    path('reports/<int:id>/', credit_views.report),
    path('charge/', credit_views.charge),
]

urlpatterns = [
    path('', main_views.homepage),
    path('help/', include('miapp.urls')),
    path('credit/', include(extra_patterns)),
]
```

A tener en cuenta

- Cuando **incluimos otro fichero de urls**, vamos a definir una lista que **se tiene que llamar**: **urlpatterns**

- **Ejemplo**

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

urlpatterns = [
    path('…', views.vista_1),
    re_path(" , views.vista_2),
    …
]
```

- El fichero se ubicará **dentro de la carpeta de la aplicación**:

proyecto

miApp (*Aplicación*)

views.py
models.py
admin.py
urls.py <<==