Contenidos a Trabajar

1. Vistas parametrizadas
    - Usando expresiones regulares
2. Solicitud y respuesta
3. HTML desde Django

Vistas parametrizadas

Aquí tenemos un ejemplo de muestra:

```
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),
    path('articles/<int:year>/<int:month>/<slug:slug>/', views.article_detail),
]
```

Para capturar un valor de la URL, use corchetes angulares.

- Los valores capturados pueden incluir opcionalmente un tipo de convertidor. Por ejemplo, use **<int:name>**para capturar un parámetro entero. **/**Si no se incluye un convertidor, se hacecoincidir cualquier cadena, excepto un carácter.
- No es necesario agregar una barra inclinada inicial, porque cada URL tiene eso. Por ejemplo,es **articles**, no **/articles**.
- Solicitudes de ejemplo:
- Una solicitud de **/articles/2005/03/**coincidiría con la tercera entrada de la lista. Django llamaría a la función .**views.month_archive(request, year=2005, month=3)**
- **/articles/2003/**coincidiría con el primer patrón de la lista, no con el segundo, porque los patrones se prueban en orden y el primero es el primero que se pasa. Siéntase libre de explotar el pedido para insertar casos especiales como este. Aquí, Django llamaría a la función**views.special_case_2003(request)**
- **/articles/2003**no coincidiría con ninguno de estos patrones, porque cada patrón requiere que la URLtermine con una barra inclinada.
- **/articles/2003/03/building-a-django-site/**coincidiría con el patrón final. Django llamaría a la función .**views.article_detail(request, year=2003, month=3, slug="building-a-django-site")**

Por defecto, los siguientes convertidores de ruta están disponibles:

- **str**- Coincide con cualquier cadena que no esté vacía, excepto el separador de ruta, **'/'**. Este es el valor predeterminado si no se incluye un convertidor en la expresión.
- **int**- Coincide con cero o cualquier número entero positivo. Devuelve un **int**.
- **slug**- Coincide con cualquier cadena de slug que consista en letras o números ASCII, además de los caracteres de guiones y guiones bajos. Por ejemplo, **building-your-1st-django-site**.
- **uuid**- Coincide con un UUID formateado. Para evitar que varias URL se asignen a la misma página, se deben incluir guiones y las letras deben estar en minúsculas. Por ejemplo, **075194d3-6885-417e-a8a8- 6c931e272f00**. Devuelve una instancia UUID
- **path**- Coincide con cualquier cadena que no esté vacía, incluido el separador de ruta, **'/'**. Esto le permite comparar con una ruta de URL completa en lugar de con un segmento de una ruta de URL como con **str**.

Usando expresiones regulares

Si la sintaxis de rutas y convertidores no es suficiente para definir sus patrones de URL, también puede usar expresiones regulares. Para hacerlo, use **re_path()** en lugar de **path()**.

Todas las expresiones regulares utilizadas deben contener una expresión regular compatible con el módulo **re** de Python. En las expresiones regulares de Python, la sintaxis para los grupos de expresiones regulares con nombre es **(?P<name>pattern)**, donde **name** es el nombre del grupo y **pattern** algún patrón para que coincida.

Aquí está el URLconf de ejemplo anterior, reescrito usando expresiones regulares:

```
from django.urls import path, re_path

from . import views

urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    re_path(r'^articles/(?P<year>[0-9]{4})/$', views.year_archive),
    re_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$', views.month_archive),
    re_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/(?P<slug>[\w-]+)/$', views.article_detail),
]
```

Esto logra más o menos lo mismo que el ejemplo anterior, excepto:

- Las URL exactas que coincidirán están un poco más restringidas. Por ejemplo, el año 10000 ya no coincidirá porque los enteros del año están restringidos a tener exactamente cuatro dígitos.
- Cada argumento capturado se envía a la vista como una cadena, independientemente del tipo decoincidencia que haga la expresión regular.
  Al cambiar de usar **path()** a **re_path()** o viceversa, es particularmente importante tener en cuenta que el tipode argumentos de vista puede cambiar, por lo que es posible que deba adaptar sus vista.

Las *Expresiones Regulares* (o *regexes*) son la forma compacta de especificar patrones en un texto. Aunque lasURLconfs de Django permiten el uso de regexes arbitrarias para tener un potente sistema de definición de URLs, probablemente en la práctica no utilices más que un par de patrones regex. Esta es una pequeña selección de patrones comunes:

Símbolo	Coincide con
.	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)
[^/]+	Todos los caracteres excepto la barra
*	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

Para más información sobre cómo escribir expresiones regulares en Python, revise la siguientedocumentación: https://docs.python.org/3/library/re.html

Solicitud y Respuesta

Antes de comenzar a utilizar los métodos de conveniencia con los que vienen las vistas, hablemos del ciclo de solicitud y respuesta. Entonces, cuando le sucede una solicitud a un servidor Django, suceden un par de cosas. Uno de ellos es Middleware.

Middleware

Middleware es como un término medio entre una solicitud y una respuesta. Es como una ventana por la que pasan los datos. Como en una ventana, la luz entra y sale de la casa. De manera similar, cuando se realiza una solicitud, una solicitud pasa a través de middlewares a vistas y los datos se pasan a través de middleware como respuesta.

Aquí están los middlewares predeterminados instalados en Django

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Objetos de solicitud y respuesta:

Django usa objetos de solicitud y respuesta para pasar el estado a través del sistema.

Cuando se solicita una página, Django crea un objeto HttpRequest que contiene metadatos sobre la solicitud. Luego, Django carga la vista adecuada, pasando HttpRequest como primer argumento a la función de vista. Cada vista es responsable de devolver un objeto HttpResponse.

Atributos de HttpRequest - Django

Puede usar los siguientes atributos con HttpRequest para manipulación avanzada

ATRIBUTO	DESCRIPCIÓN
HttpRequest.scheme	Una cadena que representa el esquema de la solicitud (HTTP o HTTPnormalmente).
HttpRequest.body	Devuelve el cuerpo de la solicitud HTTP sin procesar como una cadena de bytes.
HttpRequest.path	Devuelve la ruta completa a la página solicitada, no incluye el esquema ni eldominio.
HttpRequest.path_info	Muestra la parte de información de ruta de la ruta.
HttpRequest.method	Muestra el método HTTP utilizado en la solicitud.
HttpRequest.encoding	Muestra la codificación actual utilizada para decodificar los datos de envío deformularios.
HttpRequest.content_type	Muestra el tipo MIME de la solicitud, analizado desde el encabezadoCONTENT_TYPE.
HttpRequest.content_params	Devuelve un diccionario de parámetros clave / valor incluidos en el encabezadoCONTENT_TYPE.
HttpRequest.GET	Devuelve un objeto similar a un diccionario que contiene todos los parámetrosHTTP GET dados.
HttpRequest.POST	Es un objeto similar a un diccionario que contiene todos los parámetros HTTPPOST dados.
HttpRequest.COOKIES	Devuelve todas las cookies disponibles.
HttpRequest.FILES	Contiene todos los archivos cargados.
HttpRequest.META	Muestra todos los encabezados Http disponibles.
HttpRequest.resolver_match	Contiene una instancia de ResolverMatch que representa la URL resuelta.

Métodos HttpRequest - Django

Puede utilizar los siguientes métodos con HttpRequest para manipulación avanzada

ATRIBUTO	DESCRIPCIÓN
HttpRequest.get_host()	Devuelve el host original de la solicitud.
HttpRequest.get_port()	Devuelve el puerto de origen de la solicitud.
HttpRequest.get_full_path()	Devuelve la ruta, más una cadena de consulta adjunta,si corresponde.
HttpRequest.build_absolute_uri *(ubicación)*	Devuelve la forma de ubicación URI absoluta.
HttpRequest.get_signed_cookie *(clave, predeterminado = RAISE_ERROR, salt = '', max_age =None)*	Devuelve un valor de cookie para una cookie firmada o genera una excepción django.core.signing.BadSignaturesi la firma ya no es válida.
HttpRequest.is_secure()	Devuelve True si la solicitud es segura; es decir, si se hizo con HTTPS.
HttpRequest.is_ajax()	Devuelve True si la solicitud se realizó a través de

Atributos de HttpResponse - Django

Puede usar los siguientes atributos con HttpResponse para manipulación avanzada

ATRIBUTO	DESCRIPCIÓN
HttpResponse.content	Una cadena de bytes que representa el contenido, codificado a partir de unacadena si es necesario.
HttpResponse.charset	Es una cadena que indica el juego de caracteres en el que se codificará larespuesta.
HttpResponse.status_code	Es un **código de estado HTTP** para la respuesta.
HttpResponse.reason_phrase	La frase de motivo HTTP de la respuesta.
HttpResponse.streaming	Es falso por defecto.
HttpResponse.closed	Es cierto si la respuesta se ha cerrado.

Métodos HttpResponse - Django

Puede utilizar los siguientes métodos con HttpResponse para manipulación avanzada

MÉTODO	DESCRIPCIÓN
HttpResponse .__init__*(content = '', content_type =None, status = 200, reason = None, charset = None)*	Se utiliza para crear una instancia de un objeto HttpResponse con el contenido de página y el tipo decontenido dados.
HttpResponse .__setitem__ *(encabezado, valor)*	Se utiliza para establecer el nombre del encabezado dado en el valor dado.
HttpResponse .__delitem__ *(encabezado)*	Elimina el encabezado con el nombre de pila.
HttpResponse .__getitem__ *(encabezado)*	Devuelve el valor para el nombre de encabezado dado.
HttpResponse.has_header *(encabezado)*	Devuelve Verdadero o False en función de una comprobación que no distingue entre mayúsculas y minúsculas para un encabezado con el nombre proporcionado.
HttpResponse.setdefault *(encabezado, valor)*	Se utiliza para establecer el encabezado predeterminado.
HttpResponse.write *(contenido)*	Se utiliza para crear un objeto de respuesta de un objeto similar a un archivo.
HttpResponse.flush()	Se utiliza para vaciar el objeto de respuesta.
HttpResponse.tell()	Este método convierte una instancia de HttpResponseen un objeto similar a un archivo.
HttpResponse.getvalue()	Se utiliza para obtener el valor de HttpResponse.content.
HttpResponse.readable()	Este método se utiliza para crear un objeto similar a una secuencia de la clase HttpResponse.
HttpResponse.seekable()	Se utiliza para hacer que el objeto de respuesta se pueda buscar.

Para más información, visite la documentación de Django al respecto: https://docs.djangoproject.com/es/3.2/ref/request-response/

HTML desde Django

Para retornar Html en Django, una de las formas más simples que podemos utilizar es generando el mismo html de manera fija en el código en la vista, y generando así una respuesta con el mismo como se muestra acontinuación:

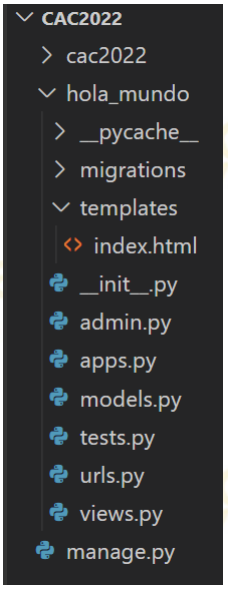
```
# Create your views here.
def index(request):
    return HttpResponse("<h1 style=\"color:blue\">Hola gente</h1>")
```

Pero de esta manera sería muy tedioso escribir una página completa dentro de views.py, y su vez ir realizando un mantenimiento de dicho de código, sin ni siquiera considerar la escalabilidad. A demás, uno de los puntos principales que observamos con el patrón MVT, es poder separar la lógica de la presentación. Es por eso que incorporaremos lo que se llaman plantillas (Templates), pero que como una pequeña introducción en este punto las veremos como html estático. Para eso, solamente debemos crear una carpeta Templates dentro de nuestra aplicación, desarrollar el HTML que queramos visualizar y referencia dicho HTML desde nuestra vista. Para eso, si quisiéramos mostrar el ejemplo previo pero en un HTML separado de la vista, deberíamos hacerlode la siguiente manera:

1. Desarrollar un archivo HTML, en este caso llamado index.html con el siguiente contenido:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hola Mundo!!</title>
</head>
<body>
    <h1 style="color:blue">HOLA GENTE!</h1>
</body>
</html>
```

2. Luego, el archivo HTML, lo guardamos en una carpeta llamada templates, dentro de nuestra aplicación (para el ejemplo la aplicación se llama “hola_mundo” y el proyecto de Django “cac2022”:



3. Luego, solo nos queda llamar al HTML desde nuestra vista, y renderizarlo con una función que nos provee el framework. Vale aclarar que el ejemplo solo es necesario poner index.html ya que el framework busca los archivos de templates, en la carpeta template de la aplicación (más adelante en el curso profundizaremos en esto):

```
from django.http import HttpResponse
from django.shortcuts import render


def index(request):
    return render(request, "index.html")
```