Contenidos a Trabajar

1. Templates
2. BackEnd y Loader
   - Configuración
   - Tipos de Cargador
3. El lenguaje de plantillas de Django (DTL)
   - Sintaxis
   - Variables
   - Etiquetas
4. Integración con Vistas

Agencia de Aprendizaje a lo largo de la vida

Plantillas (templates)

Al ser un framework web, Django necesita una forma conveniente de generar HTML dinámicamente. El enfoque más común se basa en plantillas. Una plantilla contiene las partes estáticas de la salida HTML deseada, así como alguna sintaxis especial que describe cómo se insertará el contenido dinámico.

Un proyecto Django se puede configurar con uno o varios motores de plantillas (o incluso cero si no usa plantillas). Django incluye backends integrados para su propio sistema de plantillas, creativamente denominado lenguaje de plantillas de Django (DTL), y para la popular alternativa Jinja2 . Los backends para otros lenguajes de plantillas pueden estar disponibles de terceros. También puede escribir su propio backend personalizado, como una plantilla personalizada.

Django define una API estándar para cargar y renderizar plantillas independientemente del backend. La carga consiste en encontrar la plantilla para un identificador determinado y preprocesarla, generalmente compilándola en una representación en memoria. Representar significa interpolar la plantilla con datos de contexto y devolver la cadena resultante.

El lenguaje de plantillas de Django es el propio sistema de plantillas de Django. Hasta Django 1.8 era la única opción integrada disponible. Es una buena biblioteca de plantillas a pesar de que es bastante obstinada y tiene algunas idiosincrasias. Si no tiene una razón apremiante para elegir otro backend, debe usar DTL, especialmente si está escribiendo una aplicación conectable y tiene la intención de distribuir plantillas. Las aplicaciones de contribución de Django que incluyen plantillas, como django.contrib.admin , usan el DTL.

Por razones históricas, tanto el soporte genérico para motores de plantillas como la implementación del lenguaje de plantillas Django viven en el espacio de nombres **django.template**.

Backend y Loader

Configuración

Los motores de plantillas están configurados con la con la configuración **TEMPLATES**. Es una lista de configuraciones, una para cada motor. El valor predeterminado está vacío. El **settings.py** generado por el comando **startproject** define un valor más útil:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates', 'DIRS':
        [],
        'APP_DIRS': True,'OPTIONS': {
            # ... some options here ...
        },
    },
]
```

BACKEND es una ruta punteada de Python a una clase de motor de plantilla que implementa la API backend de plantilla de Django. Los backends integrados son **django.template.backends.django.DjangoTemplates**

y **django.template.backends.jinja2.Jinja2**.

Dado que la mayoría de los motores cargan plantillas desde archivos, la configuración de nivel superior para cada motor contiene dos configuraciones comunes:

- **DIRS** define una lista de directorios en los que el motor debe buscar archivos fuente de plantilla, enorden de búsqueda.
- **APP_DIRS** dice si el motor debe buscar plantillas dentro de las aplicaciones instaladas. Cada backend define un nombre convencional para el subdirectorio dentro de las aplicaciones donde se deben almacenar sus plantillas.

Si bien es poco común, es posible configurar varias instancias del mismo backend con diferentes opciones. En ese caso, debe definir un único **NAME** para cada motor.

OPTIONS contiene configuraciones específicas de back-end.

Tipos de Cargador (template loader)

De manera predeterminada, Django usa un cargador de plantillas basado en un sistema de archivos, pero Django viene con algunos otros cargadores de plantillas, que saben cómo cargar plantillas de otras fuentes.

Algunos de estos otros cargadores están deshabilitados de forma predeterminada, pero puede activarlos agregando una opción **'loaders'** a su backend **DjangoTemplates** en la configuración **TEMPLATES** o pasando unargumento **loaders** a **Engine**. **Loaders** debe ser una lista de cadenas o tuplas, donde cada una representa una clase de cargador de plantillas. Estos son los cargadores de plantillas que vienen con Django:

django.template.loaders.filesystem.Loader

***clase* filesystem. Loader**

Carga plantillas desde el sistema de archivos, según **DIRS**.

Este cargador está habilitado de forma predeterminada. Sin embargo, no encontrará ninguna plantilla hastaque configure **DIRS** como una lista no vacía:

```
TEMPLATES = [{
    'BACKEND':    'django.template.backends.django.DjangoTemplates',    'DIRS':
    [BASE_DIR / 'templates'],
}]
```

También puede anular **'DIRS'** y especificar directorios específicos para un cargador de sistema de archivosen particular:

```
TEMPLATES = [{
'BACKEND': 'django.template.backends.django.DjangoTemplates','OPTIONS': {
    'loaders': [
        (
            'django.template.loaders.filesystem.Loader',    [BASE_DIR    /
            'templates'],
        ),
    ],
},
}]
```

django.template.loaders.app_directories.Loader

***clase* app_directories. Loader**

Carga plantillas de aplicaciones Django en el sistema de archivos. Para cada aplicación en **INSTALLED_APPS**, el cargador busca un subdirectorio **templates**. Si el directorio existe, Django buscaplantillas allí.

Esto significa que puede almacenar plantillas con sus aplicaciones individuales. Esto también ayuda a distribuir aplicaciones de Django con plantillas predeterminadas.

Por ejemplo, para esta configuración:

```
INSTALLED_APPS = ['myproject.polls', 'myproject.music']
```

…entonces **get_template('foo.html')** buscará **foo.html** en estos directorios, en este orden:

- **/path/to/myproject/polls/templates/**
- **/path/to/myproject/music/templates/**

… y usará el que encuentre primero.

¡El orden de **INSTALLED_APPS** es significativo! Por ejemplo, si desea personalizar el administrador de Django, puede optar por anular la **admin/base_site.html**plantilla estándar, desde **django.contrib.admin**, con la suya

propia **admin/base_site.html**en **myproject.polls**. Luego debe asegurarse de que su **myproject.polls**viene *antes* **django.contrib.admin** de **INSTALLED_APPS**, de lo contrario **django.contrib.admin**, se cargará primero y el suyo será ignorado.

Tenga en cuenta que el cargador realiza una optimización cuando se ejecuta por primera vez: almacena en caché una lista de los paquetes en **INSTALLED_APPS** que tienen un subdirectorio **templates**.

Puede habilitar este cargador configurando **APP_DIRS** en **True:**

```
TEMPLATES = [{
    'BACKEND': 'django.template.backends.django.DjangoTemplates','APP_DIRS': True,
}]
```

Para más información sobre cache de loaders o distintos motores de plantillas, acceda a la documentaciónoficial en:

https://docs.djangoproject.com/es/3.2/topics/templates/

El lenguaje de plantillas de Django (DTL)

Sintaxis

Una plantilla de Django es un documento de texto o una cadena de Python marcada con el lenguaje de plantillas de Django. Algunas construcciones son reconocidas e interpretadas por el motor de plantillas. Los principales son las variables y las etiquetas.

Una plantilla se representa con un contexto. La representación reemplaza las variables con sus valores, que se buscan en el contexto, y ejecuta etiquetas. Todo lo demás sale como está.

La sintaxis del lenguaje de plantilla Django implica cuatro construcciones.

Variables

Las variables se ven as: **{{ variable }}**. Cuando el motor de plantillas encuentra una variable, la evalúa y la reemplaza con el resultado. Los nombres de variables consisten en cualquier combinación de caracteres alfanuméricos y el guion bajo (“_”), pero no pueden comenzar con un guion bajo y no pueden ser un

número. El punto (“.”) también aparece en secciones variables, aunque eso tiene un significado especial, como se indica a continuación. Es importante destacar que no puede tener espacios o caracteres de puntuación en los nombres de las variables.

Use un “.” para acceder a los atributos de una variable. En el siguiente ejemplo {{ articulo.titulo }} será re reemplazado por el valor del atributo título del objeto articulo.

Técnicamente, cuando el sistema de plantillas encuentra un punto, intenta las siguientes llamadas en elsiguiente orden:

- *Llamada de la propiedad como si fuera un diccionario*
- *Llamada de la propiedad como si fuera el atributo o el método de un objeto.*
- *Llamada de la propiedad como si fuera una búsqueda por índice.*

Si el resultado es ejecutable, es llamado sin argumentos.

Etiquetas

Las etiquetas se ven así: {% etiqueta %}. Las etiquetas son más complejas que las variables: algunas crean texto en la salida, otras controlan el flujo realizando bucles o lógica, y algunas cargan información externa en la plantilla para que las usen determinadas variables.

Algunas etiquetas requieren etiquetas de inicio y finalización (es decir,).

{% etiqueta %} ... contenidos ... {% cierreetiqueta %}

Django tiene unas dos docenas de etiquetas de plantilla integradas. Puede leer todo sobre ellos en el siguientedocumento:

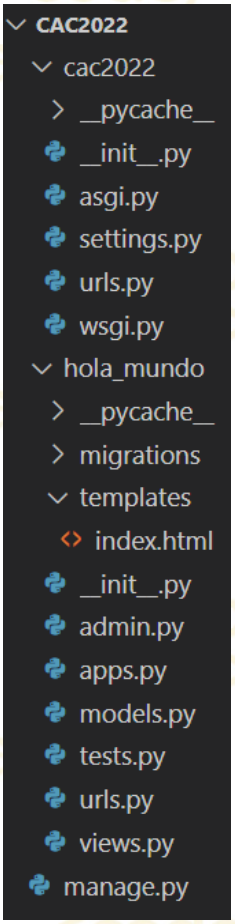
https://docs.djangoproject.com/es/3.2/ref/templates/builtins/#ref-templates-builtins-tags

Igualmente, en próximas unidades revisaremos las etiquetas más utilizadas.

Integración con Vistas

En el siguiente ejemplo, resumimos los archivos necesarios para mostrar una plantilla desde una vista, pasando una fecha como contexto, y utilizando los principalmente los valores por defecto de Django.

1. En primer lugar, tendremos la siguiente estructura de directorio, donde el proyecto que utilizaremos de ejemplo se llamará cac2022 y tendrá una sola aplicación llamada hola_mundo



2. Dentro de la carpeta templates de la aplicación (debemos crearla si no existe), crearemos un archivo llamado index.html con la siguiente estructura:

```
<!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! hoy es {{hoy}}</h1>
</body>
</html>
```

3. Dentro de la configuración, tendremos puesta en true la variable APP_DIRS e indicaremos queutilizamos el backend por defecto:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

4. En el urls.py del proyecto y en el de la aplicación, tendremos lo siguiente configurado

cac2022 > urls.py > ...

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

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

hola_mundo > urls.py > ...

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

urlpatterns = [
    path('', views.index, name="index"),
]
```

5. Y en la vista que estamos llamando tendremos lo siguiente:

```
from datetime import datetime
from django.shortcuts import render


def index(request):
    return render(request, "index.html", {"hoy": datetime.now})
```

Notar que la variable que estamos guardando en el contexto la llamamos “hoy”, al igual que lanombramos dentro del index.html con {{ hoy }}

6. Una vez configurado eso, si ingresamos por el explorador a http://127.0.0.1:8000/hola_mundo/ veremos que nos muestra la fecha cargada con el momento y hora actual.