Desarrollo de Aplicaciones para Internet

Guión de Prácticas 6: Django

S. Alonso (zerjioi@ugr.es) y J.M. Guirao (jmguirao@ugr.es)

Entrega: 10 - enero

Resumen

`Django` es un framework web completo ampliamente usado. Cuenta con un motor de plantillas propio (muy similar a `Jinja 2`) así una arquitectura `Modelo/Vista/Controlador`. En esta práctica vamos a instalar dicho framework y a hacer una aplicación básica.

Instalación y puesta en marcha de `Django`

Al ser un framework tan usado, en internet podemos encontrar muchos tutoriales como Django Girls o Django Tutorial for Beginners

Seguiremos los pasos de Quickstart: Compose and Django para hacer la instalación con docker.

Siguendo las indicaciones de la página anterior, para crear nuestro proyecto ejecutamos **dentro** del contendor de django:

```
$ docker-compose run web django-admin mi_sitio_web .
```

Una vez instalado, podemos compropbar que funciona iniciando el servidor de desarrollo:

```
$ docker-compose up
```

Para la Base de Datos, podemos usar `postgres` como en la página anterior, o dejar `sqlite` que trae por defecto django. En caso de usar `postgres`, es conveniente montar un directorio para los datos como indica final de la página de la documentación de la imagen docker de postgress, y añadir la imagen de adminer si se desea un cliente gráfico para la BD como pone en esta documentación.

Creamos ahora una aplicación dentro del projecto:

```
$ docker-compose run web python manage.py startapp mi_aplicacion
```

Creamos un directorios para los templates y para los archivos estáticos:

```
$ mkdir app/mi_aplicacion/templates
$ mkdir static
```

y los apuntamos en el archivo `sitio_web/settings.py`

```
# al final del archivo settings.py
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]
```

y apuntamos también nuesta aplicación:

```
INSTALLED_APPS = (
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'mi_aplicacion',
)
```

Ahora podemos iniciar la bases de datos SQL que usa django para los datos de los usuarios (registro, autentificación y autorización), que usaremos más adelante.

```
$ docker-compose run web python manage.py migrate
```

Creamos ahora un administrador de la BD (SQL)

```
$ docker-compose run web python manage.py createsuperuser
```

y tendremos acceso a la aplicación de administración de la BD en:

```
 http://localhost:8000/admin
```

Y podemos ahora hacer una aplicación siguiendo los pasos de URLs en Django

Solo tendremos que cambiar, el enrutador (ahora en dos archivos aparte) `mi_sitio_web/urls.py`:

```
# mi_sitio_web/urls.py

from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
  url('admin/', admin.site.urls),
  url('', include('mi_aplicacion.urls')),
]
```

y en un nuevo archivo donde especificamos las rutas que comiencen por **/**, `mi_aplicacion/urls.py`

```
# mi_aplicacion/urls.py

from django.urls import path
from . import views

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

El código del controlador lo pondremos en el archivo `miaplicacion/views.py`

```
# mi_aplicacion/views.py

from django.shortcuts import render, HttpResponse

# Create your views here.

def index(request):
    return HttpResponse('Hello World!')

def test_template(request):
    context = {}   # Aqui van la las variables para la plantilla
    return render(request,'test.html', context)
```

`Django` utiliza una librería de templates, muy parecida al `Jinja2` de `Flask`: solo cambian las instrucciones para cargar los archivos estaticos y los nombres de los enlaces

```
{% load static %}
   ...
  <link  href="{% static 'css/style.css' %}" rel="stylesheet">
  ...
  <a href="{% url 'name para la url' %}"> ... </a>
```

Model

Haremos una aplicación para gestionar los prestamos de una biblioteca con dos tablas para libros, y prestamos. El código para el model, va en el archivo **model.py**, el modelo mas básico sería:

```
# mi_aplicacion/models.py
from django.db import models
from django.utils import timezone

class Libro(models.Model):
  titulo = Models.CharField(max_lenght=200)
  autor  = Models.CharField(max_length=100)
  ...

  def __str__(self):
    return self.titulo

class Prestamo(models.Model):
  libro   = Models.ForeingKey(Libro, on_delete=models.CASCADE)
  fecha   = Models.DateField(default=timezone.now)
  usuario = Models.CharField(max_lenght=100)

  ...
```

Opcionalmente este modelo se puede ampliar con un campo para subir un archivo de imagen para la portada del libro, usar otra tabla para autores, de forma que un libro pueda tener varios autores, y un autor varios libros, etc

Cada vez que toquemos la estructura de las tablas, tendremos que pasar los scripts de `makemigrations` y `migrate` (Migrations).

Formularios

Django incorpora un mecanismo para manejar los formularios de manera eficiente. En este apartado se pide crear los formularios que permitan añadir, editar y borrar los modelos. Por supuesto el formulario debe realizar todas las validaciones pertinantes (evitar campos en blanco, valores incorrectos, etc).

Para nota

Integrar los formularios con boostrap, como en Advanced Form Rendering with Django Crispy Forms