

Django

Django es un framework web de alto nivel que permite el desarrollo de sitios web rápido, seguro y mantenible. Desarrollado por programadores experimentados, Django se encarga de gran parte de las complicaciones del desarrollo web.

Las aplicaciones web de Django normalmente agrupan el código que gestiona cada uno de estos pasos en ficheros separados:

- **URLs:** Aunque es posible procesar peticiones de cada URL individual vía una función individual, es mucho más sostenible escribir una función de visualización separada para cada recurso. Se usa un mapeador URL para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición. El mapeador URL se usa para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición. El mapeador URL puede también emparejar patrones de cadenas o dígitos específicos que aparecen en una URL y los pasan a la función de visualización como datos.
- **Vista (View):** Una vista es una función de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP. Las vistas acceden a los datos que necesitan para satisfacer las peticiones por medio de modelos, y delegan el formateo de la respuesta a las plantillas ("*templates*").
- **Modelos (Models):** Los Modelos son objetos de Python que definen la estructura de los datos de una aplicación y proporcionan mecanismos para gestionar (añadir, modificar y borrar) y consultar registros en la base de datos.
- **Plantillas (Templates):** una plantilla (template) es un fichero de texto que define la estructura o diagrama de otro fichero (tal como una página HTML), con marcadores de posición que se utilizan para representar el contenido real. Una *vista* puede crear dinámicamente una página usando una plantilla, rellenandola con datos de un modelo. Una plantilla se puede usar para definir la estructura de cualquier tipo de fichero.

Aplicaciones y proyectos

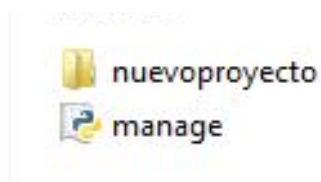
En django podemos crear proyectos y aplicaciones, un proyecto puede tener muchas aplicaciones, cada una de ellas debe contener unos archivos necesarios para su funcionamiento, los que son prácticamente seguros que vamos a necesitar son los archivos `models.py` y `views.py`.

El archivo `models.py` es el que hará la conexión con la base de datos, mientras que el archivo `views.py` será el que le haga las peticiones y las envíe a la template que le digamos.

Cuando creamos un nuevo proyecto con el siguiente comando:

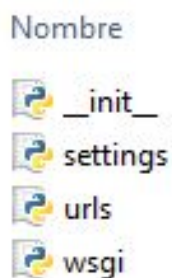
```
1 django-admin.py startproject nuevoproyecto
```

Se crea automáticamente la siguiente estructura de archivos y directorios.



El archivo `manage.py` es el que nos permite interactuar con la consola desde la línea de comandos.

Dentro de la carpeta de nuestro proyecto, que es la carpeta que se ha creado llamada `nuevoproyecto`



- El archivo `__init__.py`: Un archivo vacío que le dice a Python que este directorio se debe considerar un paquete de Python.

- El archivo settings.py: Desde aquí establecemos toda la configuración de nuestro proyecto.
- El archivo urls.py: Desde aquí podemos crear todo el sistema de rutas de nuestro proyecto.
- El archivo wsgi.py: La plataforma de desarrollo principal de Django es WSGI , el estándar de Python para servidores web y aplicaciones.

Vistas

Un función de vista o una vista, *como es conocida generalmente*, es una función en Python que hace una solicitud Web y devuelve una respuesta Web, esta respuesta puede ser el contenido de una página, un error 404, una imagen, un documento XML, entre muchas cosas más.

La vista contiene toda la lógica necesaria para devolver una respuesta, todas estas respuestas se encuentran en un único archivo y este archivo se llama: **views.py**, que se encuentra dentro de cada aplicación de Django.

Las Vistas se colocan en el archivo views.py

ejercicios/views.py

Creemos una *vista (view)*

```
from django.shortcuts import render

# Create your views here.
```

Modelos

Un modelo de una aplicación Django es un archivo con código Python, generalmente llamado `models.py`. Este archivo representa la estructura de la base de datos de nuestra aplicación.

En el `models.py` se escriben clases Python que representan las tablas y variables de clase que representan las columnas de las tablas. Por ejemplo, si pensamos en una aplicación blog, la tabla `Post` podría definirse en un modelo como:



```
1 from django.db import models
2
3
4 class Post(models.Model):
5     """Modelo de Post."""
6     title = models.CharField() # variable de clase
7     text = models.TextField()
8     date = models.DateField()
```

Tenemos que indicarle a nuestro proyecto Django que la aplicación *polls* está instalada. Para ello editamos el `settings.py` del proyecto, y agregamos a la lista:

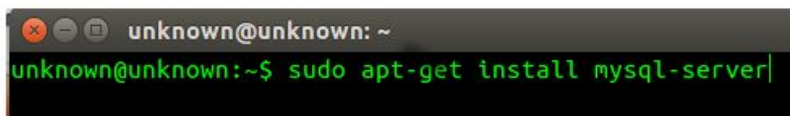
`INSTALLED_APPS` la línea `'polls.apps.PollsConfig'`:

```
31  # Application definition
32
33  INSTALLED_APPS = [
34      'polls.apps.PollsConfig',
35      'django.contrib.admin',
36      'django.contrib.auth',
37      'django.contrib.contenttypes',
38      'django.contrib.sessions',
39      'django.contrib.messages',
40      'django.contrib.staticfiles',
41  ]
42
```

Base de datos

Para la base de datos se utilizó el mysql

Lo primero que debemos realizar es instalar MySQL en nuestra PC. para ello usaremos el siguiente código.



```
unknown@unknown: ~
unknown@unknown:~$ sudo apt-get install mysql-server
```

Una vez instalado, deberemos crear una base de datos para que Django cree las tablas necesarias.

Ingresamos a la base de datos.

Código: Bash

```
1. | mysql -u root -p
```

```
unknown@unknown:~$ mysql -u root -p
Enter password: |
```

-u = USUARIO DE LA BASE DE DATOS

-p = PASSWORD DE LA BASE

Creamos la base de datos llamada "ejemplo"

```
1. | create database ejemplo;
```

```
mysql> create database ejemplo;
Query OK, 1 row affected (0.04 sec)

mysql> |
```

comprobamos que haya sido creada correctamente.

```
1. | show dabatases;
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| demo          |
| ejemplo       |
| filtro        |
| mysql         |
| performance_schema |
| phpmyadmin    |
| prueba       |
+-----+
8 rows in set (0.00 sec)
```

Plantillas

En HTML no se puede escribir código en Python porque los navegadores no lo entienden. Sólo saben HTML. Sabemos que HTML es bastante estático, mientras que Python es mucho más dinámico.

Las **etiquetas de plantilla de Django** nos permiten insertar elementos de Python dentro del HTML, para poder construir sitios web dinámicos más rápida y fácilmente.

Para imprimir una variable en una plantilla de Django, utilizamos llaves dobles con el nombre de la variable dentro, algo así:

```
{{ posts }}
```

Bootstrap

El sistema de plantillas de Django es jerárquico de manera que unas plantillas extienden o heredan de otras, todo empieza con base.html la plantilla a partir de la cual se crean las demás, esta contiene la estructura mínima HTML5 así como los

elementos necesario para el funcionamiento del framework Bootstrap (CSS y librerías JS).

base.html define 6 bloques que pueden ser modificados/extendidos por otras plantillas:

- **title:** Título de la página, con un valor por defecto.
- **description:** Descripción de la página.
- **head:** Bloque para añadir cualquier tag extra a la cabecera, por ejemplo **CSS** específico de una página.
- **content:** Bloque principal donde se añadirá todo el contenido desde otras plantillas.
- **scripts:** Para añadir librerías y código javascript extra.

Referencias:

<https://developer.mozilla.org/es/docs/Learn/Server-side/Django/Introducci%C3%B3n>

<https://www.uno-de-piera.com/proyectos-y-aplicaciones-en-django-diferencias/>

<https://medium.com/@devsar/modelos-en-django-381530c5fc3c>

<https://underc0de.org/foro/python/django-1-7-con-mysql-en-ubuntu/>

https://tutorial.djangogirls.org/es/django_templates/