TEMA IV

DESARROLLO DE APLICACIONES WEB EN BACKEND

"*Siempre parece imposible,*
hasta que se hace"
Nelson Mandela

7.1 LOS PRIMEROS PASOS

¿QUÉ ES DJANGO?

Es un framework web, gratuito y libre, programado en Python. Su nombre se debe al guitarrista de jazz Django Reinhardt. Se creo en 2003 cuando los desarrolladores web del periódico Lawrence Journal-World migraron la arquitectura de su versión online a Python.

Es utilizado por muchos sitios web: Mozilla, Pinterest, Instagram

Nos va a permitir crear sitios web complejos de forma sencilla. Promueve la filosofía del desarrollo ágil y extensible aplicando el principio **Don't Repeat Yourself.**

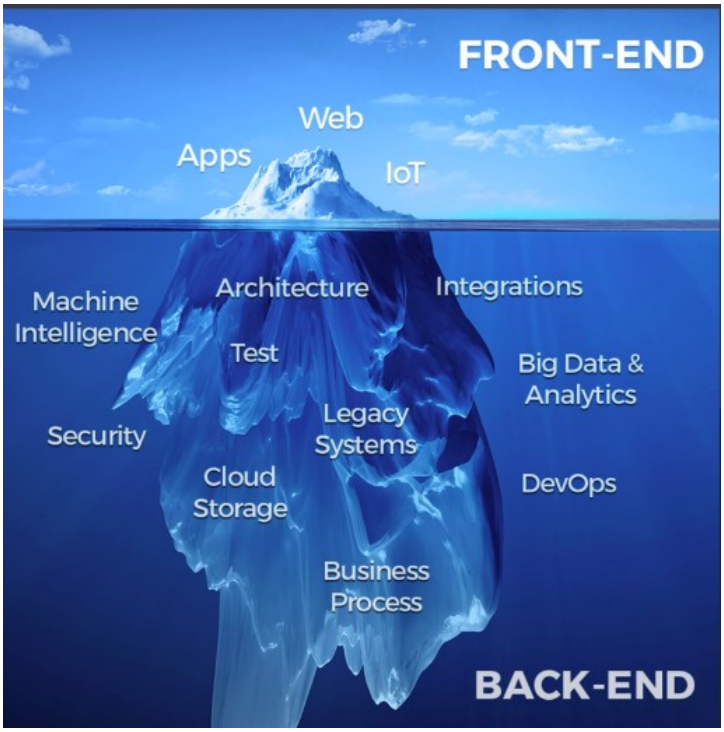
Está basado en el patrón MVC (Modelo Vista Controlador) aunque no lo usa exactamente. Hay unos pequeños cambios pero son mínimos. Django sigue el patrón MVT (Model View Template).

- Las vistas son los templates
- El controlador es el view
- Los modelos igual

Algunas de sus características:

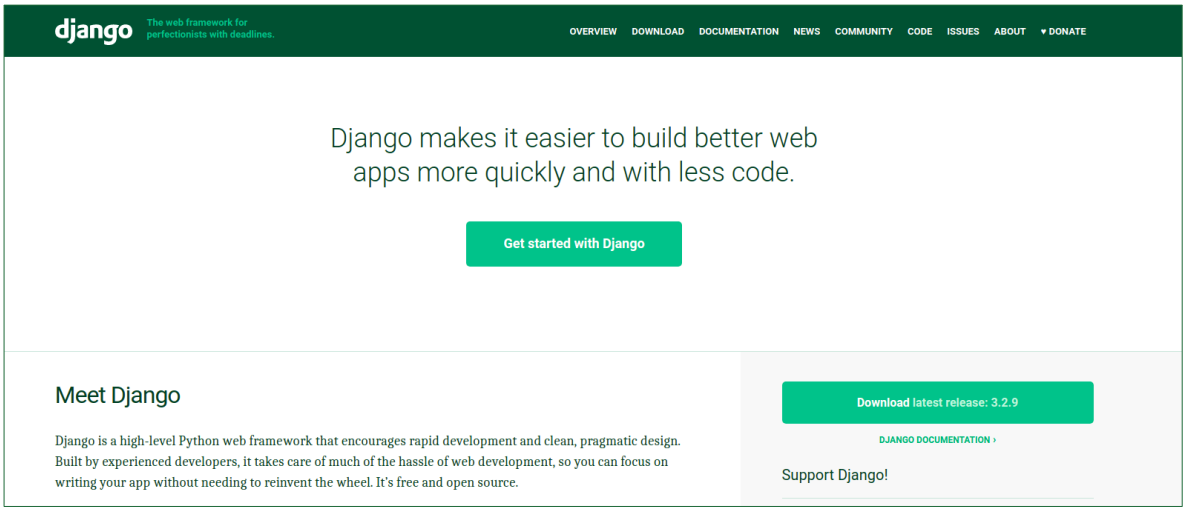
1. Cuenta con un sistema basado en componentes reutilizables llamados apps
2. Tiene un mapeado para manejar registros de la base de datos como si fueran objetos junto con una API muy robusta para su acceso.
3. Tiene un panel de administrador para gestionar esos objetos a través de formularios.
4. También incluye un potente sistema de plantillas extensibles con herencia basado en etiquetas
5. Sesiones , Formularios, Testing , Caché

Generalmente no es buena idea desarrollar un proyecto back sin saber cómo va a ser el front. En Django, es incluso peor



INSTALACIÓN

Obviamente es imprescindible tener instalado Python.



Nos vamos a descargar la última versión y vemos que el comando a ejecutar es

```
pip install Django==3.2.9
```

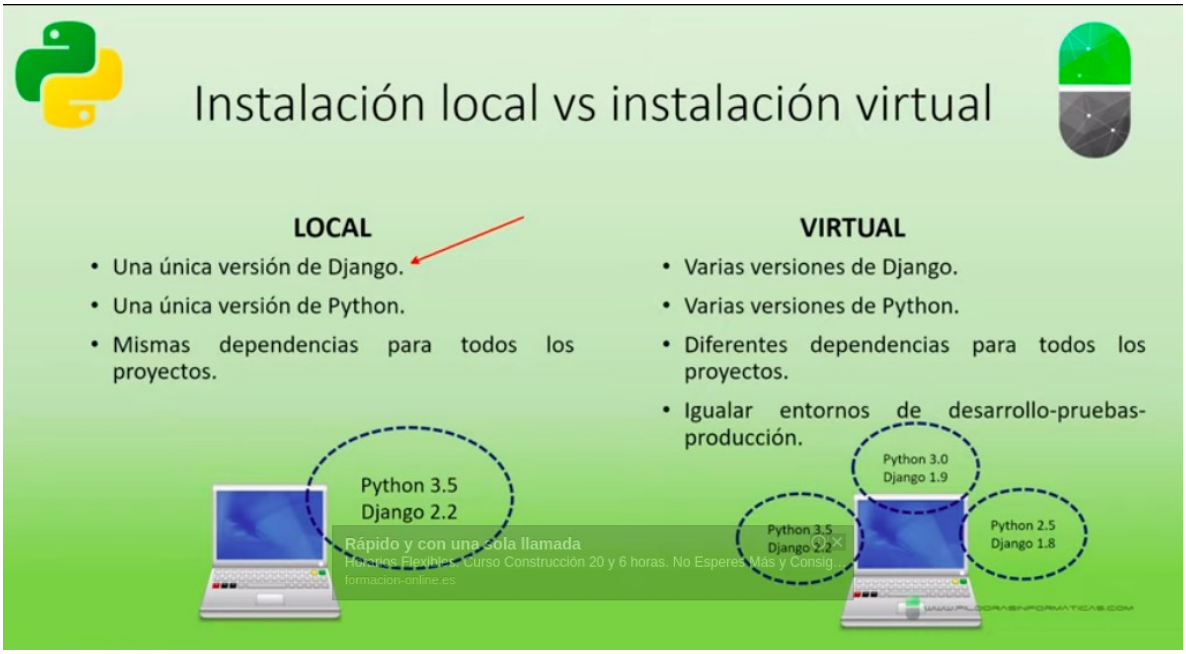
En algunas versiones de linux y mac ya viene instalado. Para comprobar la versión que tenemos

```
rosa@rosa-Essential14L:~$ python3 -m django --version
3.2.9
rosa@rosa-Essential14L:~$
```

O bien desde el intérprete de python

```
rosa@rosa-Essential14L:~$ python3
Python 3.8.10 (default, Sep 28 2021, 16:10:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.VERSION
(3, 2, 9, 'final', 0)
>>>
```

Hemos instalado localmente Django, para aprender a programar es suficiente con este tipo de instalación. Para desarrollar y poner en producción nuestros proyectos debemos hacer uso de la instalación virtual. Más adelante veremos ésto.



En cuanto a los gestores de bbdd

Python y Django traen por defecto instalado y configurado SQLite. Su gran ventaja es que es muy ligero. Python recomienda usar PostgreSQL.

CREAR UN PROYECTO

Para crear un proyecto, desde consola:

django-admin startproject nombre

```
rosa@rosa-Essential14L:~/Escritorio/DJANGO$ django-admin startproject Ejem1
rosa@rosa-Essential14L:~/Escritorio/DJANGO$
```

Se genera una carpeta con los archivos necesarios

El archivo manage.py contiene todos los comandos que podremos utilizar:

```
rosa@rosa-Essential14L:~/Escritorio/DJANGO/Ejem1$ python3 manage.py

Type 'manage.py help <subcommand>' for help on a specific subcommand.

Available subcommands:

[auth]
    changepassword
    createsuperuser

[contenttypes]
    remove_stale_contenttypes

[django]
    check
    compilemessages
    createcachetable
    dbshell
    diffsettings
    dumpdata
    flush
    inspectdb
    loaddata
    makemessages
```

En el subpaquete que nos crea tenemos los siguientes archivos

- __init__.py : paquete
- urls.py: contendrá las URL que usaremos en nuestro proyectos
- wsgi.py: información sobre nuestro servidor web
- asgi.py: si utilizamos web asíncrona
- settings.py: contiene configuraciones de nuestro proyecto

Si abrimos el archivo settings veremos que django tiene instaladas unas app para que podamos trabajar con sesiones, autenticaciones,.....

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Algunas de estas aplicaciones hacen uso de una base de datos, para que nuestro proyecto funcione debemos configurar algunas cosas, para ello ejecutamos el comando migrate del archivo manage.py

python3 manage.py migrate

Si ha podido realizar las migraciones necesarias todas estarán a OK y en la carpeta del proyecto nos añada un archivo db.sqlite3

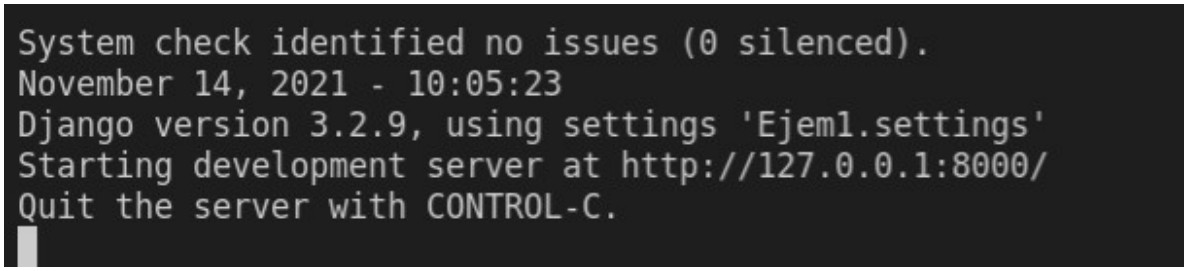
Una vez hecho esto mi proyecto está listo para ejecutarse, para lo que necesitamos levantar el servidor web y lanzar la aplicación en un navegador.

En cuanto al servidor web, django tiene instalado uno por defecto, es muy ligero y será suficiente para probar nuestros proyectos. No se debe usar cuándo subimos a real nuestras aplicaciones porque no soporta peticiones múltiples, cargas de trabajo relativamente elevadas....

Para arrancar el servidor web

python3 manage.py runserver

Si todo va bien debe mostraros lo siguiente



Podemos ver la dirección IP y el puerto en el que está escuchando

Sólo nos queda abrir un navegador y lanzar esa URL, debe aparecer la página inicial de django.

De forma predeterminada se inicia en el puerto 8000, se puede cambiar

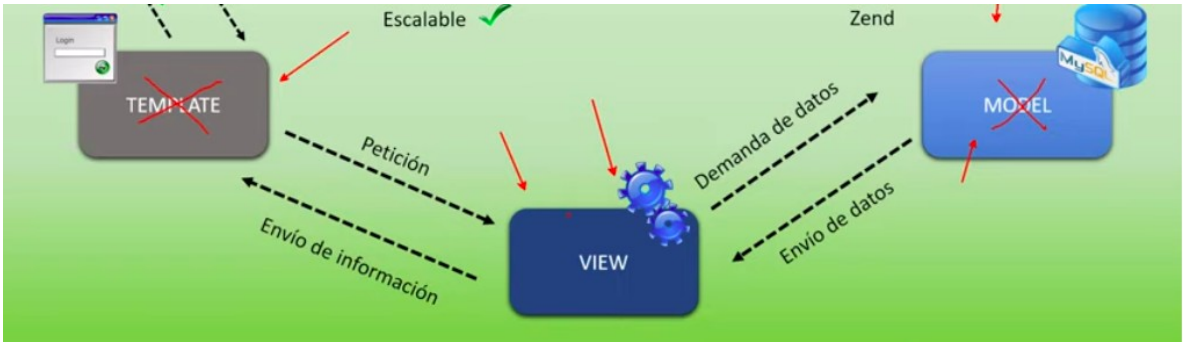
python3 manage.py runserver 8080

El servidor se reinicia de forma automática cuando hacemos cambios en nuestro proyecto. Si no fuese así tendríamos que arrancarlo de nuevo.

Hemos creado un proyecto que no una aplicación. Un proyecto es una colección de configuraciones, incluyendo configuración de base de datos, opciones específicas de Django y configuraciones específicas de aplicaciones. En el futuro podremos tener varias aplicaciones en un proyecto, las añadiremos al archivo settings.

MI PRIMER EJEMPLO

Hagamos un ejemplo sencillo que muestre una página html. Recordemos que django trabaja con MVT(Modelo, Vista, Template).
En este caso, no vamos a acceder a bases de datos ni vamos a tener POO por lo que no necesitaremos modelos. Lo que sí necesitamos es un "controlador", es decir un script que recibe la petición de cargar nuestra página(template) y la devuelve al navegador que la solicitó.



Hemos hablado de recibir una petición y retornar una información, para realizar este proceso se utilizan dos clases HttRequest y HttpResponse. La primera de ellas es la que usamos para lanzar peticiones y la segunda la que usamos para enviar la información solicitada.

Ejemplo1 : “Hola mundo”

Creamos un archivo llamado views.py. Cada una de las funciones que crearemos en él se denominan vista (Controlador). Cada vista tendrá enlazada una URL que será la que escribamos en el navegador para cargarla.

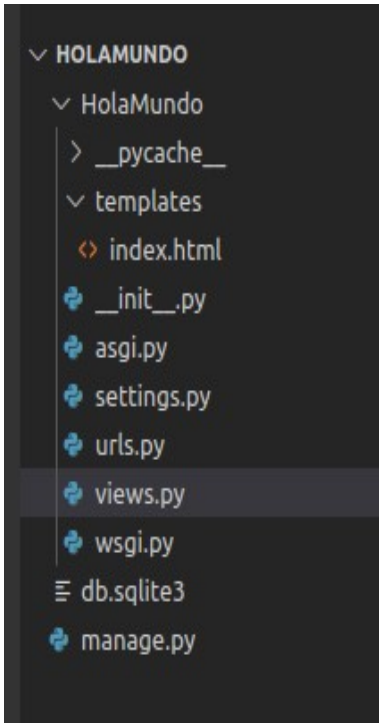
HolaMundo > HolaMundo > views.py > ...

```python
from django.http import HttpResponse


def saludo(request):
    return HttpResponse("HOLA MUNDOOOOO!!!!!")

def despedida(request):
    return HttpResponse("Hasta el próximo ejemplo")

```

```python
from HolaMundo.views import despedida, saludo

urlpatterns = [
    path('admin/', admin.site.urls),
    path('saludo/',saludo),
    path('despedida/',despedida)
]
```

Django tiene una serie de funciones que abrevian la “escritura” del objeto response. En este ejemplo no veremos mucha diferencia porque no hemos usado templates. Obviamente, en la vida real no lo haremos así, siempre utilizaremos templates.

Añadimos una carpeta templates e incluimos nuestro html en ella




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


def saludo(request):
    #return HttpResponse("HOLA MUNDOOOOO!!!!!")
    return render(request,'index.html')
```

```html
<!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>Document</title>
</head>

<body>
    <p>Hola mundoooo!!!!!</p>
</body>

</html>
```