DJANGO

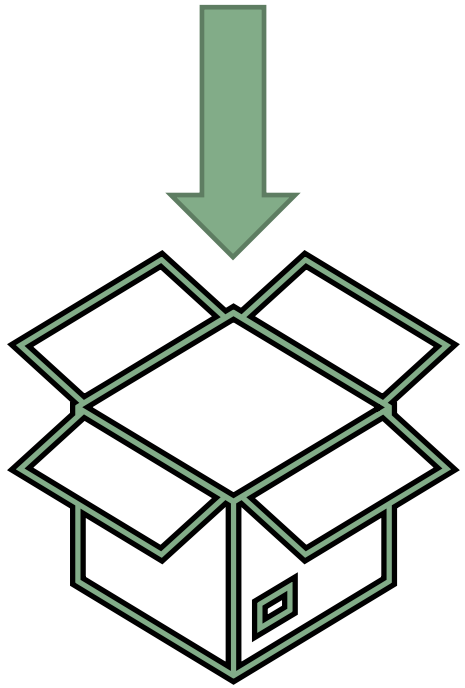
JUAN CARLOS GIL DÍAZ

HORA DEL CÓDIGO 2020

DJANGO

- Framework de desarrollo web basado en Python
- Código abierto
- Sigue el patrón MVC
- Énfasis en la reutilización y el desarrollo rápido
- Liberado en el año 2005, en el año 2008 Django Software Foundation se hace cargo de Django
- Disponible para Windows, MacOS y Linux

INSTALACIÓN



- **Instalar Python 3**
- $ sudo apt-get install python3
- **Instalar Django**
- $ python3 –m pip install Django
- **Comprobar la versión**
- $ python3 –m django --version

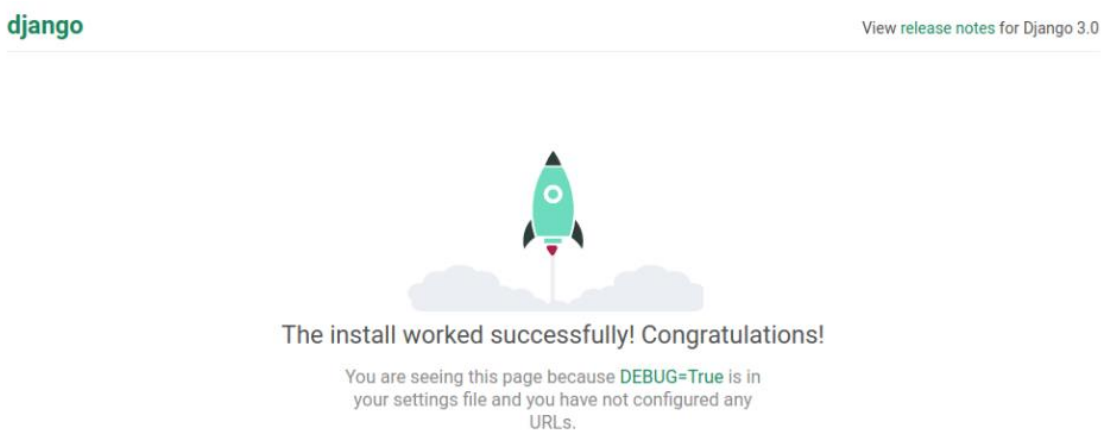
NUESTRO PRIMER PROYECTO

- En cualquier directorio
- $ django-admin startproject proyecto
- Se genera la estructura del proyecto
  - manage.py: script Python que nos servirá para gestionar el proyecto
  - proyecto: package del proyecto
  - urls.py: declaración de las urls del proyecto
  - settings.py: configuración del proyecto
- "Desplegar" el proyecto
  - $ python3 manage.py runserver [ip:puerto]

(ignorar el error de base de datos)

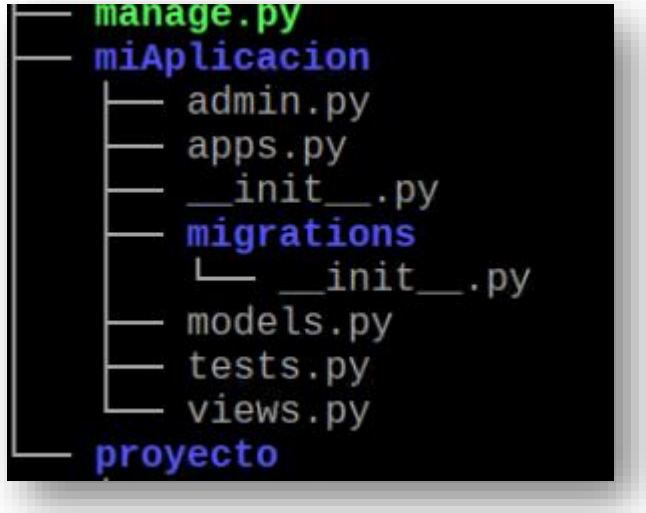
```
pi@raspberrypi:~/Documents/Django/proyecto $ tree
.
├── manage.py
└── proyecto
    ├── asgi.py
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py

1 directory, 6 files
```



CREAR UNA APLICACIÓN WEB

- $ python3 manage.py startapp miAplicacion
- **Un proyecto puede tener muchas aplicaciones, una aplicación puede estar en muchos proyectos**



MVC EN DJANGO

- **Modelo: models.py**
- **Vista: páginas html (templates) (≈ a una página JSP)**
- **Controlador: views.py (≈ a un servlet)**

PRIMERA VISTA EN DJANGO

views.py

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello world")
```

- Hemos creado una función que devuelve una página html con el texto “Hello world”. Ahora hay que definir cuándo se llama a dicha función. Esto se hace creando un fichero urls.py dentro de la aplicación.

urls.py (de la aplicación)

```
from django.urls import path

from . import views

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

- En urls.py, creamos la lista en la que iremos asignando las urls que escribamos en el navegador con las funciones de views.py que se llamarán, y opcionalmente le daremos un nombre a las páginas.

PRIMERA VISTA EN DJANGO

urls.py (del Proyecto)

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

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

- Por último, en el fichero urls.py del proyecto, debemos añadir la referencia al fichero de urls (urlconf) de la aplicación

- Ahora, si ejecutamos el servidor y en un navegador escribimos la dirección “localhost:8000/miAplicacion/”, recibiremos el mensaje “Hello world” escrito anteriormente.

HAGAMOS ALGO MÁS INTERESANTE

- Hemos hecho una vista desde el controlador -> rompe el patrón MVC y no es reutilizable
- Ahora vamos a hacer una plantilla para una vista que muestre un contenido dinámicamente

TEMPLATES

- Crear directorio “templates” dentro del directorio de la aplicación. Django comienza buscando en este directorio las plantillas que necesite.
- Dentro del directorio templates, crear otro con el nombre de la aplicación (en nuestro caso, “miAplicacion”)
- Nuestras plantillas se situarán en /templates/miAplicacion/

ESTRUCTURA DE UN TEMPLATE

- **Documento HTML normal y corriente**
- **Si es un formulario, se debe añadir una línea ->** {% csrf_token %} **(seguridad)**
- **Añadir variables ->** {{variable}}

LLAMAR A UN TEMPLATE DESDE EL CONTROLADOR

- **Utilizaremos el atajo** render(request,template,context)
  - **Request: el mismo que se pasa de parámetro a cada función del controlador.**
  - **Template: ruta a la plantilla (por ejemplo, 'miAplicacion/plantilla1.html').**
  - **Context: diccionario con los valores de las variables.**

EN NUESTRA APLICACIÓN...

urls.py

```
from django.urls import path
from . import views
urlpatterns=[
        path('',views.index,name="Página principal"),
        path('registrarse',views.registrarse),
        path('datosUsuario',views.datosUsuario),
]
```

1. Plantillas “registrarUsuario” (formulario) y “datosUsuario” (dinámica).
2. Crear la función en views.py.
3. Añadir las urls al URLConf.

views.py

```
from django.shortcuts import render
…

def registrarse(request):
    context={}
    return render(request,'miAplicacion/registrarUsuario.html',context)

def datosUsuario(request):
    nombre = request.POST['nombre’]
    correo = request.POST['email’]
    fechaN = request.POST['fechaN’]
    return render(request,'miAplicacion/datosUsuario.html',{'name':nombre,'email':correo,'date':fechaN})
```

Fin de la primera parte

AÑADIENDO UNA BASE DE DATOS SENCILLA

- Por defecto: SQLite
- También da soporte a PostgreSQL, MariaDB, MySQL y Oracle
- Utilizaremos SQLite porque es sencillo y no hay que configurar nada

CREANDO UNA BASE DE DATOS MUY SENCILLA

- Comenzamos creando el modelo para la tabla. Abrimos models.py
- En este caso vamos a añadir una sola tabla, que servirá para almacenar usuarios, de los cuales queremos un nombre, una fecha de nacimiento y un email.
- Todos los campos del usuario deben ser elementos de models. En este caso utilizamos un campo de texto para el nombre, otro de fecha para la fecha de nacimiento, y otro de correo para el email
- Lista de elementos de models: https://docs.djangoproject.com/en/3.1/topics/db/models/
- ¿Y el script sql para crear la tabla? ¡No es necesario, Django lo crea por nosotros!

models.py

```
from django.db import models

class Usuario(models.Model):
        nombre = models.charField(max_length=100)
        fechaNacimiento = models.DateTimeField('Fecha de nacimiento')
        email = models.EmailField(max_length=200)
```

CREANDO UNA BASE DE DATOS MUY SENCILLA

```
# Internationalization
# https://docs.djangoproject.com/en/3.

LANGUAGE_CODE = 'en-us'

USE_TZ = True
TIME_ZONE = 'Europe/Madrid'

USE_I18N = True
```

```
$ python3 manage.py migrate
```

```
INSTALLED_APPS = [
    'miAplicacion.apps.MiaplicacionConfig',
    'django.contrib.admin',
    'django.contrib.auth',
```

- Primero accedemos al fichero settings.py, añadimos la línea “USE_TZ=True” y modificamos el valor de “Time_Zone” a “Europe/Madrid” (no es necesario pero si no lo hacemos nos marcará horas incorrectas)
- A continuación, volvemos a llamar a manage.py, esta vez con el argumento “migrate”, que instala las aplicaciones indicadas en el fichero settings.py.
- Volviendo al fichero de configuración, añadimos el fichero de configuración de nuestra aplicación a la lista de aplicaciones instaladas

CREANDO UNA BASE DE DATOS MUY SENCILLA

- Ahora, notificamos a Django que hemos realizado cambios en el modelo para que cree o modifique las tablas oportunas, llamando una vez más a manage.py. Nos indicará que ha creado la tabla para el Usuario, pero aún no la ha implantado.

```
$ python3 manage.py makemigrations miAplicacion
```

```
pi@raspberrypi:~/Documents/Django/proyecto $ python3 manage.py makemigrations miAplicacion
Migrations for 'miAplicacion':
  miAplicacion/migrations/0001_initial.py
    - Create model Usuario
```

- En caso de que quieras el código sql, se puede generar con "sqlmigrate miAplicacion 0001" (el número es el que se indique en la salida de makemigrations)

```
$ python3 manage.py sqlmigrate miAplicacion 0001
```

```
pi@raspberrypi:~/Documents/Django/proyecto $ python3 manage.py sqlmigrate miAplicacion 0001
BEGIN;
--
-- Create model Usuario
--
CREATE TABLE "miAplicacion_usuario" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "nombr
har(100) NOT NULL, "fechaNacimiento" datetime NOT NULL, "email" varchar(200) NOT NULL);
COMMIT;
```

- Tras esto, ejecutar de nuevo el comando migrate para crear la tabla en la base de datos.

MODIFICAR LAS VISTAS ANTERIORES

- A continuación, vamos a modificar las vistas y funciones creadas anteriormente. Ahora en el registro, se almacena el empleado en la base de datos, y añadimos una nueva función que nos permite buscar usuarios en la base de datos a partir de su identificador.

- Una vez que tenemos las plantillas, comenzamos añadiendo las urls al fichero urlconf de la aplicación.

urls.py

```
from django.urls import path

from . import views

from django.urls import pathfrom . import viewsurlpatterns=[
        path('',views.index,name="Página principal"),
        path('registrarse',views.registrarse),
        path('datosUsuario',views.datosUsuario),
        path('getUsuario',views.getUsuario),
        path('buscaUsuario',views.buscarUsuario),
        ]
```

MODIFICACIONES EN LAS PLANTILLAS

Por último, implementamos las funciones a las que se llamará. Para devolver la página web indicada, se utilizará de nuevo el método render

views.py

```
from miAplicacion.models import Usuario


...

def datosUsuario(request):
    nombr = request.POST['nombre']
    em = request.POST['email']
    fn = request.POST['fechaN']
    usuario = Usuario(nombre=nombr,fechaNacimiento=fn, email = em)
    usuario.save()
    return render(request,'miAplicacion/datosUsuario.html',{'name':nombr,'email':em,'date':fn})
```

MODIFICACIONES EN LAS PLANTILLAS

views.py (cont)

```
def getUsuario(request):
    return render(request,'miAplicacion/buscarUsuario.html',{})

def buscarUsuario(request):
    ident = request.POST['id']
    u = Usuario.objects.get(id = ident)
    nombr = u.nombre    em = u.email
    fn = u.fechaNacimiento
    return render(request,'miAplicacion/datosUsuario.html',{'name':nombr,'email':em,'date':fn})
```

SITIO DE ADMINISTRADOR

- **Ver las aplicaciones en el proyecto**
- **Acceder a los elementos registrados de la base de datos, modificarlos, crearlos o eliminarlos.**
- **Es necesario crear al menos un usuario administrador (usuario, email, contraseña)**
  - Python3 manage.py createsuperuser
- **Importar modelos que queramos en admin.py**

admin.py

```
from django.contrib import admin
from .models import Usuario

admin.site.register(Usuario)
```

- **Accedemos a la página de administrador (en nuestro caso, localhost:8000/admin) e iniciamos sesión**

FIN

- Este taller ha estado muy enfocado al framework Django, dejando de lado algunas buenas prácticas como pueden ser tratamiento de posibles excepciones, creación de los constructores de los objetos del modelo, tests… Hay que tenerlo en cuenta de cara a un desarrollo más profesional.
- ¿Quieres profundizar más?
  - Página oficial de Django: https://www.djangoproject.com/
  - Documentación Django 3.1: https://docs.djangoproject.com/en/3.1/
  - Tutorial oficial para desarrollar una aplicación similar a la que hemos realizado: https://docs.djangoproject.com/en/3.1/intro/tutorial01/
  - Cómo desplegar una aplicación Django: https://docs.djangoproject.com/en/3.1/howto/deployment/
- Encuesta: https://forms.gle/jqLqncWvMSiVx1zH7