Instalación de entorno de desarrollo

1. Primeros pasos:

Comenzamos actualizando los repositorios e instalando las nuevas dependencias:

```
~$ sudo apt-get update
~$ sudo apt-get upgrade
```

En Ubuntu viene instalado por defecto Python para comprobarlo usamos el comando:

~$ python3 -V o **~$ python3 --version**

Esto nos dirá la versión instalada, si no aparece podremos instalarlo con el comando:

```
~$ sudo apt-get install python3.10
```
(podemos sustituir 3.10 por la versión que queramos instalar, preferiblemente la última)

Instalamos el instalador de paquetes para Python pip:

```
~$ sudo apt-get install python-pip
```

Instalamos el módulo que da soporte en python para crear entornos virtuales:

```
~$ sudo apt-get install python3-venv
```

2. Creación de nuestro proyecto:

Creamos nuestra carpeta de proyecto a la que posteriormente apuntarán todas las configuraciones de ruta:

```
~$ mkdir pruebas
```
(si creamos la ruta de la forma indicada nuestra ruta absoluta será:
home/usuario/pruebas)

Nos desplazamos dentro de la carpeta proyecto:

```
~$ cd pruebas
```

y creamos nuestro entorno virtual

```
~/pruebas$ python3 -m venv entorno
```
(se creará una carpeta entorno donde se guardará toda la configuración de nuestro entorno)

Activamos nuestro entornos:

~/pruebas$ source /*entorno/bin/arctivate*

(dentro de la carpeta entorno se ha creado una estructura de carpetas genérica. En la carpeta /bin encontramos un script "activate" que nos sirve para activar nuestro entorno. Podemos comprobar que nuestro entorno está activo porque en la terminal, en la linea de comandos nos aparece en primer lugar el nombre de nuestro entorno entre paréntesis. Todas las configuraciones e instalaciones que hagamos con el entorno activo quedarán registradas en nuestro entorno y no serán validas fuera del mismo.)

Instalamos Django:

~pruebas$ pip install Django

creamos nuestro proyecto en Django, al cual llamaremos proyecto aunque podría coger cualquier nombre:

~pruebas$ django-admin startproject proyecto

y para comprobar que funciona entramos en la carpeta de nuestro proyecto:

~pruebas$ cd proyecto

Una vez dentro, si listamos los archivos de la carpeta encontramos el archivo manage.py y este será el script de referencia que nos permite activar nuestro sitio, y conectar nuestro proyecto con la base de datos, entre otras funciones. Para activar nuestro sitio utilizamos el comando:

~pruebas/proyecto$ python3 manage.py runserver
(podemos especificar por que puerto mostrar la aplicación con la instrucción:
~pruebas/proyecto$ python3 manage.py runserver 0.0.0.0:8001
en este caso se abrirá por el puerto 8001)

Si no hay errores al escribir en nuestra barra de direcciones 'localhost:8000' debemos de ver la página de bienvenida de Django.

3. Conectar con uWSGI

Instalamos uWSGI y los paquetes de desarrollo necesarios:

~pruebas/proyecto$ sudo apt-get install python3.8-dev
~pruebas/proyecto$ sudo apt-get install gcc
~pruebas/proyecto$ pip install uwsgi

Para comprobar que hasta el momento todo funciona bien, y que la instalación ha sido correcta creamos un archivo de prueba en la carpeta proyecto, que llamaremos test.py y que tendrá el siguiente contenido:

~pruebas/proyecto$ subl test.py
(en este caso se está creando con sublime text pero podría hacerse con cualquier editor)

Contenido del archivo:

```
def application(env, start_response):
    start_response('200 OK', [('Content-Type','text/html')])
    return [b"Hello World!, Estamos conectados"]
```

y probamos el siguiente comando:

```
~pruebas/proyecto$ uwsgi –http :8000 –wsgi-file test.py
```

Si todo va bien, si tecleamos localhost:8000 en nuestra barra de direcciones debe aparecer una página en blanco y en la parte superior el mensaje: “Hello World! Estamos conectados”, en este momento sabemos que nuestra instalación ha sido correcta y que uWSGI puede servir archivos python. Es el momento de probar si podemos conectar nuestro proyecto:

```
~pruebas/proyecto$ uwsgi –http :8000 –module proyecto.wsgi
```

Vamos a crear un archivo en la ruta en la que nos encontramos donde estableceremos los parámetros necesarios para configurar uwsgi:

```
~pruebas/proyecto$ subl uwsgi_params
```

En este archivo introducimos el siguiente contenido:

```
uwsgi_param QUERY_STRING $query_string;
uwsgi_param REQUEST_METHOD $request_method;
uwsgi_param CONTENT_TYPE $content_type;
uwsgi_param CONTENT_LENGTH $content_length;
uwsgi_param REQUEST_URI $request_uri;
uwsgi_param PATH_INFO $document_uri;
uwsgi_param DOCUMENT_ROOT $document_root;
uwsgi_param SERVER_PROTOCOL $server_protocol;
uwsgi_param REQUEST_SCHEME $scheme;
uwsgi_param HTTPS $https if_not_empty;
uwsgi_param REMOTE_ADDR $remote_addr;
uwsgi_param REMOTE_PORT $remote_port;
uwsgi_param SERVER_PORT $server_port;
uwsgi_param SERVER_NAME $server_name;
```

4. Configurar Nginx

Comenzamos instalando Nginx

```
~pruebas/proyecto$ sudo apt-get install nginx
```

(aunque esta instalación la hagamos dentro del entorno se instalará fuera del mismo, por tanto si ya lo tenemos instalado no es necesaria su instalación)

Consideraciones a tener en de Nginx:

Los archivos de configuración de Nginx se encuentran en la ruta /etc/nginx, y dentro de esta carpeta tenemos dos carpetas con las que vamos a trabajar:

- sites-available: En esta carpeta creamos los archivos de configuración de nuestro sitio y de todos los sitios con los que vayamos a trabajar. Un archivo por sitio y siempre terminado en la extensión .conf
- Sites-enabled: En esta carpeta encontramos los sitios activos. Los sitios activos son enlaces a los sitios creados en la carpeta de sites-availables, por tanto no debe existir en esta carpeta nada más que los sitios que queramos tener activos, y siempre que apunten a puertos diferentes, si no lo hacemos así se generaran errores y no se cargará el sitio deseado. Para eliminar un sitio de esta carpeta solo necesitaremos el comando *rm*, se eliminará de esta carpeta pero seguirá el archivo original en la carpeta de sites-available

Por cada modificación en los archivos de Nginx tendremos que reiniciar el servicio para ello tenemos los siguientes comandos:

sudo systemctl start/stop/reload/restart nginx

Para comprobar fallos en la sintaxis de los archivos de configuración de nginx utilizaremos:

sudo nginx -t

Después de instalar nginx y el breve repaso anterior, seguiremos situados en la misma carpeta que hemos usado durante toda la configuración y desde esa carpeta creamos nuestro archivo de configuración en la carpeta de sites-available de nginx, como nuestro proyecto Django ha recibido el nombre de proyecto, usaremos ese mismo para crear nuestro archivo de configuración

~pruebas/proyecto$ subl /etc/nginx/sites-available/proyecto.conf

En este archivo debemos introducir el siguiente contenido:

```
#Establecemos la conexión, son importantes las tres barras antes del home
upstream django {
        server unix:///home/usuario/pruebas/proyecto.sock;
}
#configuración del servidor
server {
        listen 80;
        server_name localhost;
        charset utf-8;
        client_max_body_size 75M;

    # Localización de la carpeta media y static
        location /media {
                alias /home/usuario/pruebas/media;
        }
        location /static {
                alias /home/usuario/pruebas/static;
        }
    # configuración de los parametros uwsgi
        location / {
                uwsgi_pass django;
                include /home/usuario/pruebas/proyecto/uwsgi_params;
        }
}
```

Una vez creado nuestro archivo de configuración en sites-available creamos el acceso directo en sites-enabled con el siguiente comando:

```
~pruebas/proyecto$ sudo ln -s /etc/nginx/sites-available/proyecto.conf /etc/nginx/sites-enabled
```

(comprobamos que en la carpeta sites-enabled solo este nuestro proyecto con el comando ls:

```
~pruebas/proyecto$ ls /etc/nginx/sites-enabled
```

si con esta instrucción nos aparece más de un archivo en esta carpeta sería recomendable eliminarlo:

```
~pruebas/proyecto$ sudo rm /etc/nginx/sites-enabled/archivoaeliminar.conf
```

recordemos que esto solo eliminará la imagen de ese archivo, el archivo original seguira estando en la carpeta de sites-available)

Necesitamos configurar nuestro proyecto para conectar la carpeta static del mismo con nginx y para ello editamos el archivo settings.py

```
~pruebas/proyecto$ subl proyecto/settings.py
```

(en el archivo debemos de escribir al comienzo:

```
#importar el modulo os que nos permite manipular rutas
import os
```

y cerca del final donde encontramos referencia a los contenidos estáticos:

```
STATIC_URL='/static/'
STATIC_ROOT=os.path.join(BASE_DIR, "static/")
```

)

Es el momento de indicarle a Django donde colocar los archivos estáticos:

```
~pruebas/proyecto$ python manage.py collectstatic
```

(podemos comprobar como se ha creado una carpeta static que contiene una carpeta admin donde encontramos todos los contenidos estáticos que el admin utilizará en nuestro proyecto.)

Reiniciamos el servidor Nginx para aplicar todos los cambios:

```
~pruebas/proyecto$ sudo systemctl restart nginx
```

5. Conectar uWSGI, Nginx y Django.

Para conectar todos los servicios necesitaremos el siguiente comando:

```
~pruebas/proyecto$ uwsgi --socket proyecto.sock --module proyecto.wsgi –chmod-socket=666
```

Si todos los pasos han sido correctos debería de poder escribir en la linea de direcciones de su navegador **localhost** y debe tener acceso a la página inicial de Django.

6. Instalar PostgreSQL.

Podríamos realizar la instalación de postgres desde cualquier otra ubicación en nuestro sistema de archivos, pero vamos a seguir instalados en la misma ubicación donde hemos realizado todas las configuraciones anteriores:

```
~pruebas/proyecto$ sudo apt install postgresql
```

Una vez instalado tenemos los siguientes comandos para activar, desactivar y comprobar estado del servicio:

```
~pruebas/proyecto$ sudo systemctl is-active psotgresql
~pruebas/proyecto$ sudo systemctl is-enabledpsotgresql
~pruebas/proyecto$ sudo systemctl status psotgresql
```

Una vez activo el sistema crearemos el usuario y la base de datos y daremos privilegios al usuario para trabajar en la base de datos.
Primero accedemos a a terminar de **postgresql**:

```
~pruebas/proyecto$ sudo su -postgres
```
(con esta orden comprobamos como ya no estamos en la terminal del usuario sino en la de postgres)
```
postgres@ubuntu~$ psql
```
(con esta orden nos lleva al usuario administrador de postgresql)
```
postgres=#
```

(Debemos de tener en cuenta que salvo algunos comandos que pondremos a continuación, lo que escribamos a partir de ahora son sentencias sql y deben tener la sintaxis propia de las mismas entre otras deben ir cerradas siempre con ;)

Algunos comandos interesantes son:

- \l → lista bases de datos .
- \du → lista los usuarios.
- Situados dentro de una base de datos \dt nos muestra las tablas de la misma
- \q para salir

Desde el usuario postgres podemos crear usuarios con la sentencia:

```
postgres=# CREATE USER usuario WITH PASSWORD ‘contraseña’;
```

Crear bases de datos:

```
postgres=# CREATE DATABASE nombredb;
```

Dar privilegios a un usuario para gestionar la base de datos:

```
postgres=# GRANT ALL PRIVILEGES ON DATABASE nombredb TO usuario;
```

Cambiar el propietario de una base de datos:

```
postgres=# ALTER DATABASE nombredb OWNER TO usuario;
```

Con la sentencia **\q** del usuario postgres, nos devuelve a la consola de postgresql de la que saldremos con la orden exit.

```
postgres=# \q
postgres@ubuntu~$ exit
~pruebas/proyecto$
```

Una vez creada la base de datos y el usuario, podemos acceder a nuestra base de datos desde la consola sin necesidad de entrar en la consola de postgresql:

```
~pruebas/proyecto$ psql nombredb usuario
```

Y accederemos directamente a nuestra base de datos.

6. Instalar PgAdmin y conectarlo a nuestro proyecto Django.

Para instalar pgadmin debemos de configurar el repositorio de pgAdmin para Ubuntu de modo que podamos obtener la última versión estable de la aplicación.

```
~pruebas/proyecto$ wget -qO-
https://www.pgadmin.org/static/packages_org.pub | sudo apt-key add-
```

Creamos el archivo del repositorio

```
~pruebas/proyecto$ sudo nano /etc/apt/sources.list.d/pgadmin.list
```

dentro del archivo creado escribimos:
deb https://ftp.postgresql.org/pub/pgadmin/pgadmin4/apt/focal pgadmin4 main
y guardamos.

Actualizamos la información del sistema

```
~pruebas/proyecto$ sudo apt update
```

Comenzamos la instalación

```
~pruebas/proyecto$ sudo apt install -y pgadmin4-web
```

Al instalar pgadmin se instala automáticamente apache2, como apache utiliza el puerto 80 por defecto y este puerto nosotros lo tenemos ya utilizado con el servidor nginx, procedemos a modificar el puerto de escucha de apache2:

```
~pruebas/proyecto$ sudo nano /etc/apache2/port.conf
```

En el archico cambiamos la orden :
listen:80 por **listen:81**
repetimos la orden anterior: **~pruebas/proyecto$ sudo apt install -y pgadmin4-web**

Lanzamos la configuración de pgadmin4

```
~pruebas/proyecto$ sudo /usr/pgadmin4/bin/setup-web.sh
```

Nos aparecerá una pantalla donde debemos de escribir una dirección y un password que serán los elementos con los que iniciaremos la sesión de pgadmin.

Ya tenemos instalado pgadmin4 y para acceder deberemos escribir en la barra de direcciones de nuestro navegador:

localhost:81/pgadmin4
(Nos pedirá como acceso el correo y contraseña que acabamos de poner en la configuración y entraremos al panel de administración de pgadmin4)

6.1 Conectar pgadmin con postgresql

Una vez que estamos dentro de pgadmin si entramos en **Dashboard (Panel)** y seleccionamos **Nuevo Servidor**

Nos aparecerá una ventana emergente y en la pestaña de **General** debemos introducir un **nombre** que será genérico y que recogerá todas las bases de datos de postgresql

En la pestaña de **Connection** ponemos en **Host → Localhost** y en **usuario** y **contraseña** el usuario y contraseña de nuestra base de datos postgresql. Con esto debe de aparecer en el menú de navegación de la izquierda un desplegable con todas las bases de datos postgres del usuario introducido.

6.1 Conectar la base de datos con nuestro proyecto.

Una vez que tenemos todo conectado y nuestra base de datos creada debemos indicarle a Django que trabaje con la base de datos de Postgres porque por defecto Django crea una base de datos sqLite.

En primer lugar vamos instalar un modulo que debemos tener instalado para realizar la conexión que es el siguiente:

```
~pruebas/proyecto$ pip install psycopg2-binary
```

A continuación modificamos el archivo settings.py de nuestro proyecto

```
~pruebas/proyecto$ subl /proyecto/setitings.py
```

Una vez abierto el archivo buscamos el apartado de bases de dato y lo reescribimos de esta forma, dejando comentadas las opciones por defecto:

```
DATABASES = {
    'default': {
    'ENGINE': 'django.db.backends.postgresql_psycopg2',
    'NAME': 'concesionario',
    'USER': 'belen',
    'PASSWORD': 'belen',
    'HOST': 'localhost',
    'PORT': '5432',
```

```
        #'ENGINE': 'django.db.backends.sqlite3',
        #'NAME': BASE_DIR.child('db.sqlite3'),
        #'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Para comprobar que todo el proceso ha sido correcto hacemos una migración de las tablas que Django genera por defecto y comprobamos que aparecen en pgAdmin:

```
~pruebas/proyecto$ python3 manage.py migrate
```

y comprobamos que en nuestra base de datos de pgAdmin se han creado las tablas del módulo auth que vienen configuradas por defecto en todos los proyectos de Django.