

# Guía- Tutorial Python-Django para desarrollo de Aplicaciones Web.

Guía paso a paso para de la instalación y preparación de herramientas para el desarrollo de aplicaciones web empleando python y framework MVC Django.

## Recomendaciones:

Poseer instalado Ubuntu 14.04 LTS mayor soporte y recomendado para proyectos productivos.

## Instalación de herramientas Python-Django.

Las distribuciones de Linux como Ubuntu 14.04 LTS tomado como referencia en este tutorial ya vienen con la instalación por defecto del interprete *python*.

## Herramientas necesarias:

- Python
- PIP
- Django
- Postgresql
- Psycopg2
- Virtualenvwrapper

## Opcionales:

- Pgadmin
- PyCharm

## Instalación de pip:

Para la instalación de gestor de paquetes pip se procede a abrir la consola y ejecutar el comando:

***sudo apt-get install python-pip***

De inmediato nos pide la contraseña root, la proporcionamos al shell y debemos estar seguros de poseer una conexión a internet para descargar los paquetes necesarios de instalación desde los repositorios centrales de Ubuntu o de su país. Una vez terminado este proceso ya contamos con la posibilidad de instalar paquetes empleando el comando pip.

## Instalación de Virtualenvwrapper:

Para la instalación del entorno virtual para el trabajo con proyectos se procede a abrir la consola y ejecutar el comando:

***sudo apt-get install virtualenvwrapper***

De inmediato nos pide la contraseña root, la proporcionamos al shell y debemos estar seguros de poseer una conexión a internet para descargar los paquetes necesarios de instalación desde los repositorios centrales de Ubuntu o de su país. Una vez terminado este proceso ya contamos con el entorno virtual instalado.

## Instalación de Python-Django:

Para la instalación del framework Django se procede a abrir la consola y ejecutar el comando:

***sudo apt-get install python-django***

De inmediato nos pide la contraseña root, la proporcionamos al shell y debemos estar seguros de poseer una conexión a internet para descargar los paquetes necesarios de instalación desde los repositorios centrales de Ubuntu o de su país. Una vez terminado este proceso ya contamos con el framework Django instalado.

### Instalación de PostgreSQL:

Para la instalación del gestor de bases de datos relacional postgres se procede a abrir la consola y ejecutar el comando:

***sudo apt-get install postgresql postgresql-contrib***

De inmediato nos pide la contraseña root, la proporcionamos al shell y debemos estar seguros de poseer una conexión a internet para descargar los paquetes necesarios de instalación desde los repositorios centrales de Ubuntu o de su país. Una vez terminado este proceso ya contamos con el gestor de base de datos postgres instalado.

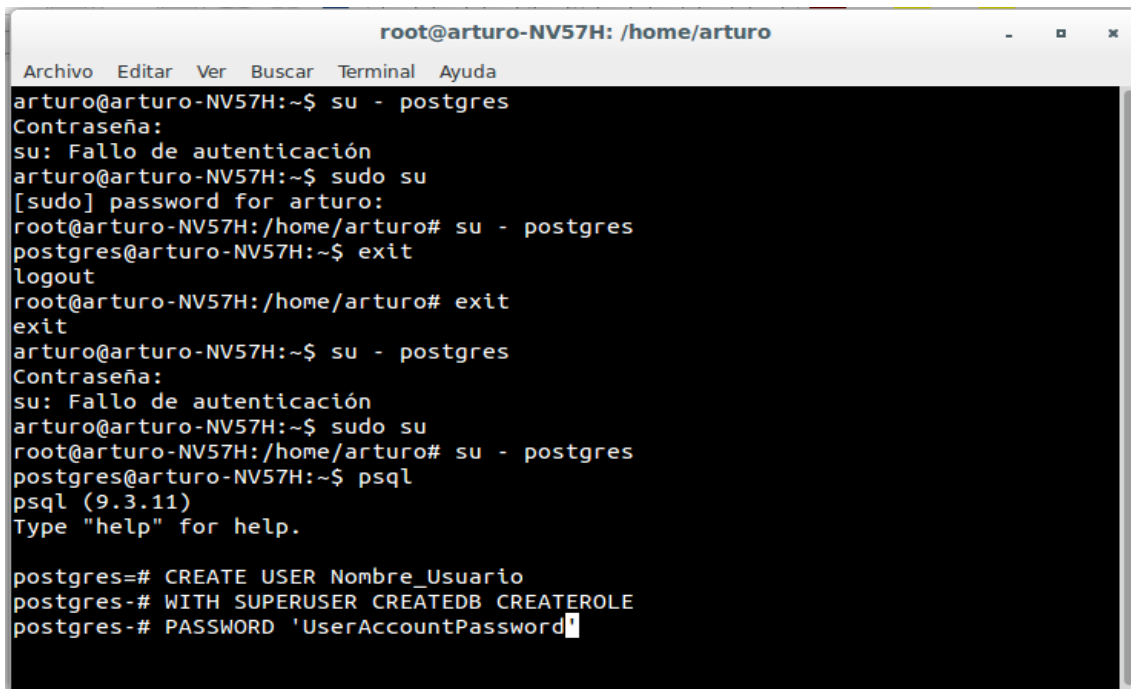
Para emplear el postgresql es recomendado crear un usuario diferente al postgres, así como en las distribuciones de Linux se crean usuarios diferentes para el trabajo diario y no se emplea al usuario root como usuario común, sino como superusuario para tareas específicas de administración.

Para realizar lo siguiente podemos seguir los siguientes pasos:

-Acceder a la consola como root, es decir ***sudo su*** y autenticarse.

-Una vez autenticado ejecutar el comando: ***su - postgres*** el cual permite conectar a postgres como superusuario.

-Luego ejecutamos el comando: ***psql*** y vemos algo así:



```
root@arturo-NV57H: /home/arturo
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
arturo@arturo-NV57H:~$ su - postgres
Contraseña:
su: Fallo de autenticación
arturo@arturo-NV57H:~$ sudo su
[sudo] password for arturo:
root@arturo-NV57H:/home/arturo# su - postgres
postgres@arturo-NV57H:~$ exit
logout
root@arturo-NV57H:/home/arturo# exit
exit
arturo@arturo-NV57H:~$ su - postgres
Contraseña:
su: Fallo de autenticación
arturo@arturo-NV57H:~$ sudo su
root@arturo-NV57H:/home/arturo# su - postgres
postgres@arturo-NV57H:~$ psql
psql (9.3.11)
Type "help" for help.

postgres=# CREATE USER Nombre_Usuario
postgres=# WITH SUPERUSER CREATEDB CREATEROLE
postgres=# PASSWORD 'UserAccountPassword'
```

Visualizamos la línea de comando postgres=# aquí ejecutamos los siguientes comandos:

```
CREATE USER Nombre_Usuario
WITH SUPERUSER CREATEDB CREATEROLE
PASSWORD 'Contraseña';
```

Para que postgres ejecute todas estas líneas de comando debemos asegurar colocar el punto y coma en la última línea que se vaya a ejecutar. Tras realizar este proceso, ya poseemos un usuario con una contraseña previamente definida para establecer conexión con la BD.

### Instalación de PgAdmin:

Para la instalación del administrador visual de bases de datos postgres se procede a abrir la consola y ejecutar el comando:

***sudo apt-get install pgadmin3***

De inmediato nos pide la contraseña root, la proporcionamos al shell y debemos estar seguros de

poseer una conexión a internet para descargar los paquetes necesarios de instalación desde los repositorios centrales de Ubuntu o de su país. Una vez terminado este proceso ya contamos con la interfaz visual para el trabajo con postgres de forma relativamente mas cómoda. Al abrir el PgAdmin empleamos el usuario y contraseña previamente creado para establecer conexión con el postgres.

### **Instalación de Psycopg2:**

Para la instalación de la utilidad psycopg2 se procede a abrir la consola y ejecutar el comando:

***sudo apt-get install python-psycopg2***

De inmediato nos pide la contraseña root, la proporcionamos al shell y debemos estar seguros de poseer una conexión a internet para descargar los paquetes necesarios de instalación desde los repositorios centrales de Ubuntu o de su país. Una vez terminado este proceso ya contamos con la utilidad psycopg2 instalada para migrar modelos de django a base de datos postgres.

### **Importante:**

Para emplear la utilidad psycopg2 debemos a su vez instalar libpq-dev y python-dev ejecutando el siguiente comando:

***sudo apt-get install libpq-dev python-dev***

Esto nos permite el correcto empleo y utilización de psycopg2 en los entornos virtuales.

Con estos pasos ya contamos con las herramientas necesarias para comenzar el desarrollo de una aplicación en python empleando django.

### **Instalación y configuración de directorios.**

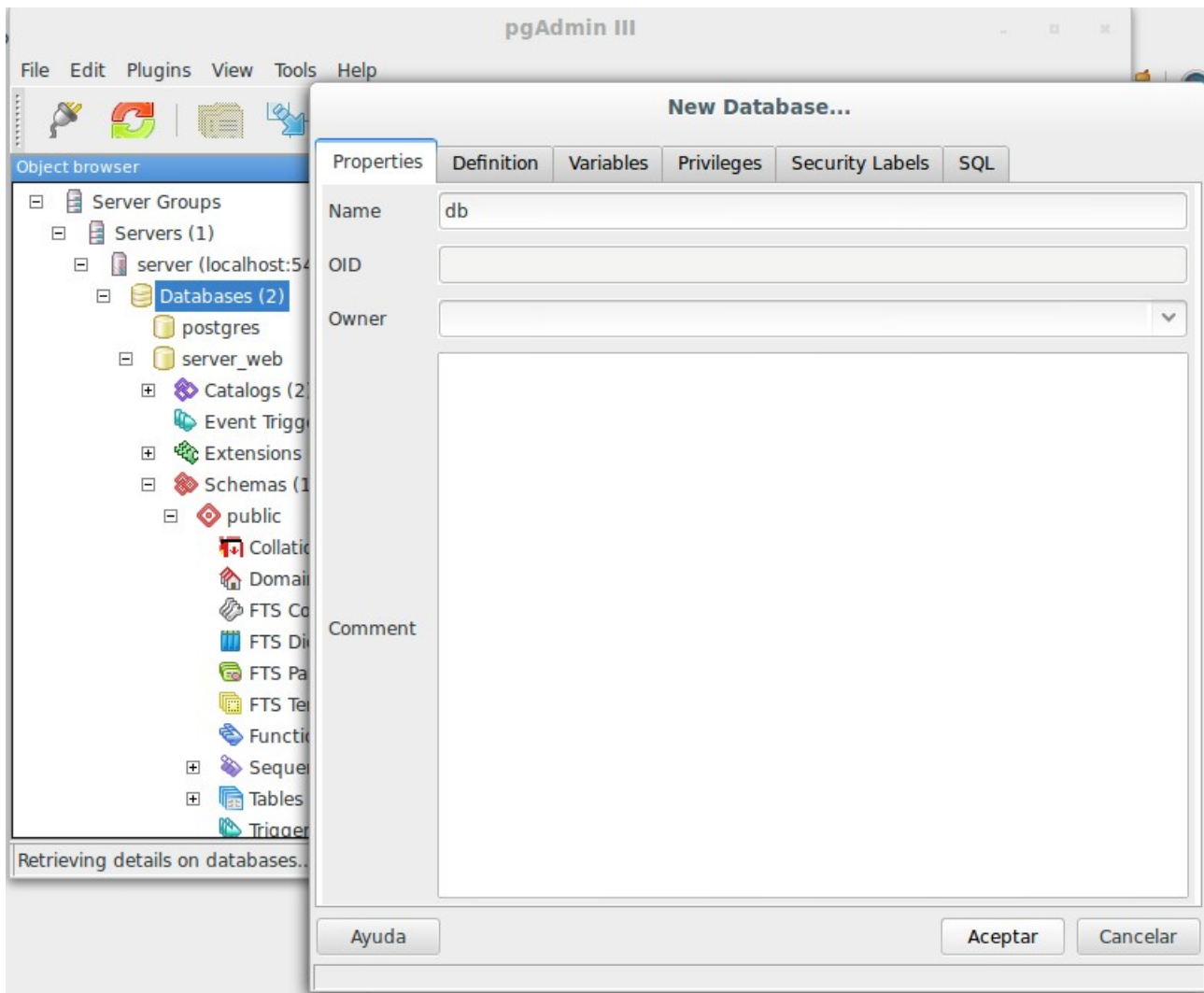
Para la creación de un proyecto lo primero que debemos hacer es crear un directorio contenedor. Por ejemplo creamos la carpeta tuto en el escritorio de nuestro usuario. Una vez creada abrimos una terminal y procedemos a ubicarnos dentro de ese directorio empleando el comando: ***cd /home/arturo/Escritorio/tuto***, una vez ubicados en esta dirección procedemos a la creación de un entorno virtual para nuestro proyecto empleando el comando: ***virtualenv nombre\_entorno***.

Una vez creado debemos activar el entorno virtual, para esto ejecutamos el siguiente comando: ***source ./nombre\_entorno/bin/activate*** con este comando aseguramos estar dentro de nuestro entorno virtual, con el comando: ***deactivate*** salimos del entorno virtual

Una vez creado nuestro entorno virtual y dentro del mismo procedemos a la instalación de las herramientas dentro de nuestro entorno. A cada entorno virtual debemos instalarle django y psycopg2 para poder trabajar con ellas. Se procede a instalar django con el comando: ***pip install django*** y una vez terminado ejecutamos: ***pip install psycopg2***.

Ya estamos listos para crear nuestra aplicación ejecutando el comando: ***django-admin startproject nombre\_proyecto*** con este comando se genera una carpeta con el nombre definido, que contendrá dentro un archivo manage.py y otra carpeta con el mismo nombre del proyecto donde se almacenan los ficheros: \_\_init\_\_.py, settings.py, urls.py y wsgi.py. De estos archivos trabajaremos mas adelante con settings.py y urls.py. Settings nos permite establecer las configuraciones, como conexión a BD, aplicaciones, plantillas, etc.

Procedemos a la conexión con la BD. Primero debemos crear nuestra BD. Para hacer esto con el PgAdmin podemos abrirlo y Database con clic derecho vamos a la opción New Database y le ponemos un nombre y presionamos aceptar y listo tenemos una BD creada.



Para establecer la conexión con la BD creada procedemos a editar el archivo settings.py que nos había creado django. Una vez dentro buscamos la sentencia DATABASE y debemos poner y modificar los datos de conexión:

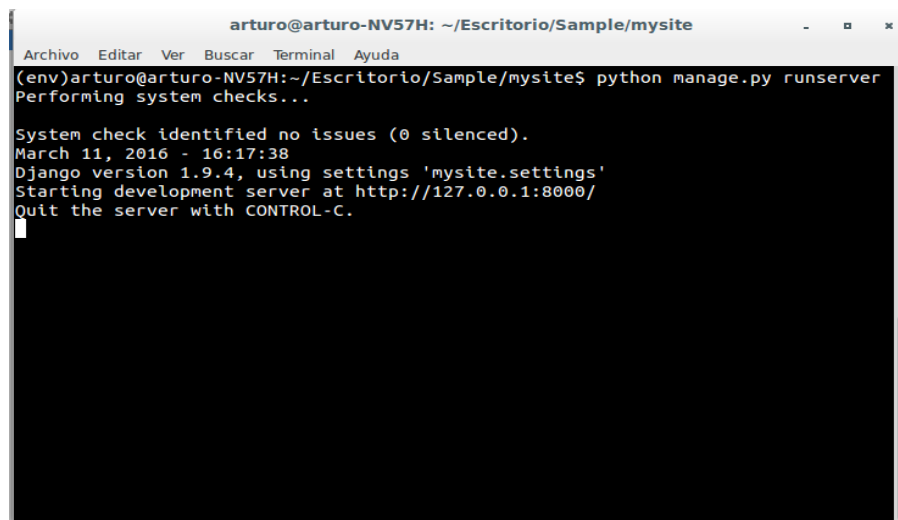
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2', # Add 'postgresql_psycopg2',
        'mysql', 'sqlite3' or 'oracle'.
        'NAME': 'ejemplo2', # Or path to database file if using sqlite3.
        'USER': 'root', # Not used with sqlite3.
        'PASSWORD': 'root', # Not used with sqlite3.
        'HOST': 'localhost', # Set to empty string for localhost. Not used with sqlite3.
        'PORT': '5432', # Set to empty string for default. Not used with sqlite3.
    }
}
```

En la primera definición ENGINE se pone siempre la línea que se ve que define que se empleará psycopg2 para conectarse con postgres, luego se coloca el nombre de la BD, el usuario y contraseña de conexión y el ip y puerto, de dejarse vacío el HOST y PORT emplea por defecto localhost y puerto 5432.

Una vez configurados los datos podemos establecer la conexión y migrar nuestras tablas a la BD

postgres. Para esto empleamos el comando: ***python manage.py makemigrations*** con este comando django busca cambios en sus modelos y se actualiza, con el comando siguiente: ***python manage.py migrate*** se migran todos los modelos de django a la BD. Si revisa La BD creada podrá ver la creación de nuevas tablas. Una vez migrados los modelos a la BD podemos crear un usuario para administrar nuestra aplicación. Empleando el comando: ***python manage.py createsuperuser*** una vez ejecutado este comando debemos establecer el nombre del mismo, un correo y una contraseña.

Para que nos servirá usuario ahora lo veremos. Podemos ejecutar el comando `python manage.py runserver` con el cual ejecutamos un servidor local empujando en desarrollo para poder ver nuestra aplicación. Si no se encuentran errores se vera una pantalla como la siguiente:



```
arturo@arturo-NV57H: ~/Escritorio/Sample/mysite
Archivo Editar Ver Buscar Terminal Ayuda
(env)arturo@arturo-NV57H:~/Escritorio/Sample/mysite$ python manage.py runserver
Performing system checks...

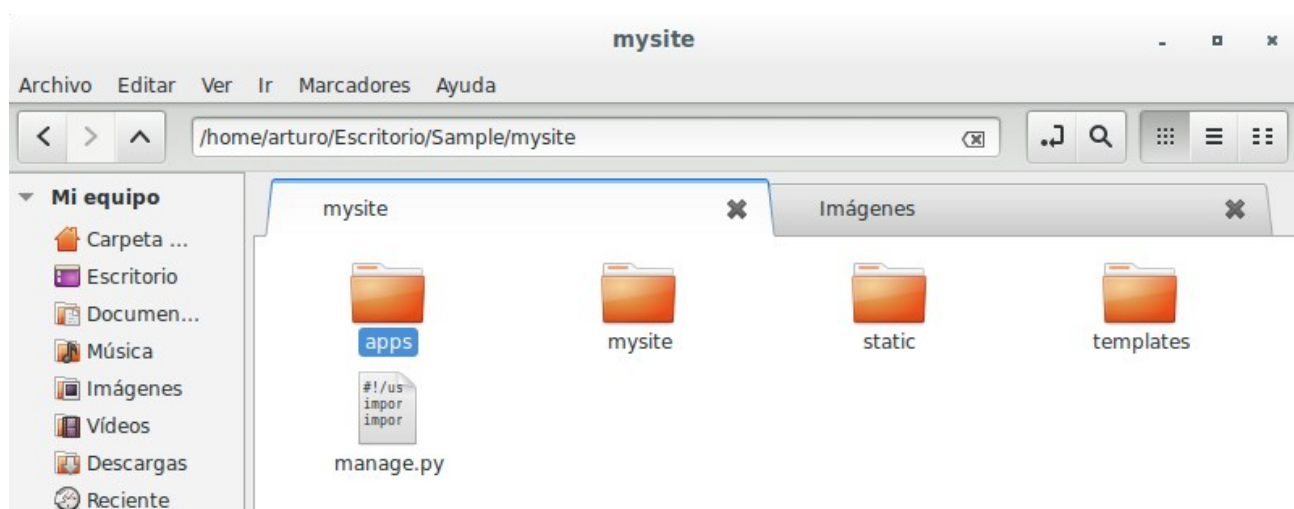
System check identified no issues (0 silenced).
March 11, 2016 - 16:17:38
Django version 1.9.4, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Listo ya tenemos ejecutando el servidor si abrimos un navegador y vamos a localhost:8000 podemos ver una pantalla de bienvenida, y si ponemos localhost:8000/admin entramos a la sección de administración del sitio donde nos autenticamos con el usuario creado con anterioridad al ejecutar el comando: ***python manage.py createsuperuser***.

### Estructura y aplicaciones.

Dentro de nuestra carpeta de proyecto debemos tener siempre la siguiente estructura:

apps, static, templates mas las carpetas y archivos creados por django cuando se crea el proyecto.



**Apps:** aquí se guardan todas las aplicaciones creadas.

**Static:** aquí se guardan todos los archivos \*.css, \*.js, etc.

**Templates:** aquí se guardan todas las plantillas \*.html básicas para la aplicación.

Estos directorio debe crearlos manualmente y para su empleo debe configurar el archivo settings.py agregando las siguientes lineas:

```
import sys
```

Empleado para localizar las app dentro de la carpeta apps.

```
sys.path.insert(0, os.path.join(BASE_DIR, 'apps'))
```

Empleado para localizar archivos estáticos dentro de la carpeta static.

```
STATIC_URL = '/static/'
```

```
STATIC_ROOT = 'E:/samplestatics/static'
```

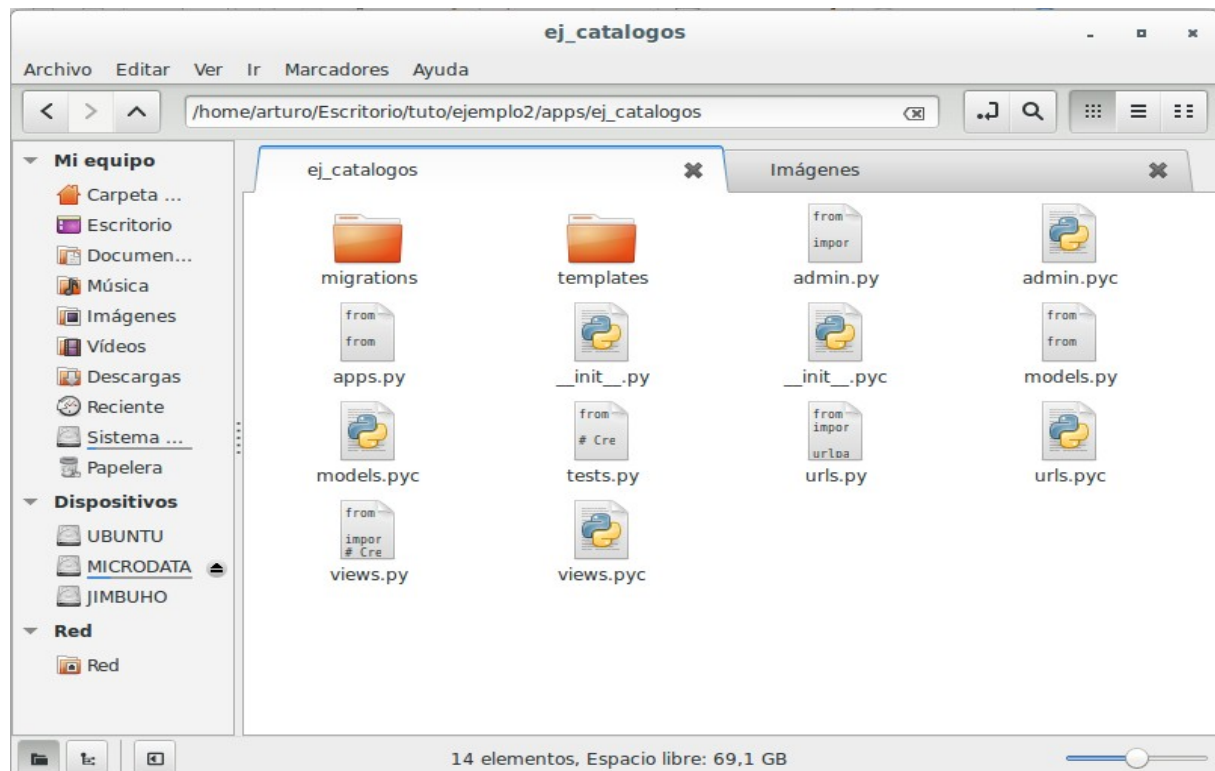
```
STATICFILES_DIRS = [  
    os.path.join(BASE_DIR, "static"),  
]
```

Dentro de la sentencia TEMPLATES agregar la linea que se ve a continuación:

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [  
            os.path.join(BASE_DIR, 'templates')  
        ],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',  
            ],  
        },  
    ],  
]
```

Con estos pasos aseguramos poseer el directorio correcto para el trabajo.

Ya contando con esta estructura procedemos a crear nuevas aplicaciones. Tomemos como ejemplo catálogos. Para crear una nueva aplicación ejecutamos el comando: **python manage.py startapp nombre\_app**. Esto nos genera un directorio fuera de la carpeta apps que luego podemos ubicar dentro de la misma. Con la siguiente estructura, donde se muestran archivos como: apps.py, models.py, urls.py, admin.py, views.py, test.py.



Con este paso ya tenemos creada nuestra app `ej_catalogos`, ahora procedemos a registrarla dentro del archivo `settings.py` en la sentencia:

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

#aquí podemos ver nuestra app creada

Una vez incluida procedemos a crear el modelo de datos, modificando el archivo `models.py` antes de ejecutarlo:

```
from __future__ import unicode_literals

from django.db import models
# Create your models here.
class Catalogo(models.Model):
    nombre = models.CharField(max_length=32)
    valor = models.CharField(max_length=10)
```

Donde creamos nuestros modelos de datos empleando clases con lenguaje python. Importamos de `django.db` los `models`. Definimos por ejemplo la clase `Catalogo` que hereda de `models.Model` y le establecemos las variables `nombre` de tipo texto con longitud menor a 32 caracteres y `valor` por igual un texto menor a 10 caracteres. Salvamos el archivo y con esto tenemos creado nuestro modelo de datos de nuestra app. Una vez creado el modelo de datos podemos ya migrarlo a la BD empleando el comando: ***python manage.py migrate***. El cual busca cambios en los modelos y si los

encuentra ejecuta la migración a la BD. Al ejecutar el comando podemos ver que se crea en nuestra BD una tabla llamada Catalogos.

Para poder acceder a esta tabla desde el sitio administrativo debemos modificar el archivo admin.py, que habíamos reverenciado con anterioridad, donde ponemos:

```
import models
```

```
# Register your models here.
```

```
class CatalogosAdmin(admin.ModelAdmin):  
    list_display = ('id', 'nombre', 'valor')
```

```
admin.site.register(models.Catalogo, CatalogosAdmin)
```

Primeramente importamos models, cada clase \*.py es considerado como una paquete en django. Luego creamos la clase CatalogosAdmin que hereda a u vez de admin.ModelAdmin y que nos muestra el id generado por la BD, el nombre y valor. Estas son las variables que definimos en el archivo models.py de nuestra aplicación.

Con esto aseguramos que una vez corriendo el servidor de django y accediendo a localhost:8000/admin podemos ver la nueva tabla catálogos creada y trabajar con ella directamente.

### Importante:

Al ejecutar el comando pyhton manage.py runserver podemos establecer el host y el puerto de conexión, ejemplo:

```
pyhton manage.py runserver 8080 #Para cambiar el puerto
```

```
pyhton manage.py runserver 192.168.0.1 8080 #Para cambiar el host y el puerto
```

## Vistas, Urls y templates.

### Views y URLs:

Completados los siguientes pasos pasaremos a ver como dar un poco de funcionalidad a nuestra aplicación. Para ellos empezaremos creando algunas vistas modificando el archivo viwes.py de nuestra aplicación como sigue:

```
from django.shortcuts import render_to_response
```

```
import models
```

```
# Create your views here.
```

```
def inicio(request):  
    return render_to_response('home.html', {})
```

```
def ver_catalogos(request):  
    catalogos = models.Catalogo.objects.all()  
    return render_to_response('catalogos.html', {'catalogos': catalogos})
```

Donde importamos los models. Para crear una vista lo hacemos como semejante a la definición de métodos o funcionalidades en pyhton, es decir con la sentencia **def**. En el ejemplo tomado definimos dos vistas una llamada *inicio(requets)* y otra *ver\_catalogos(request)* ambas le pasamos como argumentos *request*. La primera solo retorna un plantilla como respuesta al referencia o



acceder a inicio desde el navegador. La segunda consulta los datos almacenados en la tabla Catalogo creada con anterioridad por el modelo definido en la BD. Y la almacenamos en la variable *catalogo*. Luego retornamos la plantilla *catalogos.html* con toda la información almacenada en la tabla catalogo de la BD.

Para poder enlazar nuestras vistas y debemos modificar los archivos *urls.py* de nuestra aplicación el cual debemos crear de forma manual y el archivo de igual nombre *urls.py* general dentro de la carpeta de configuración donde se encuentra el fichero *settings.py*.

Para hacer esto luego de crear el archivo *urls.py* dentro de nuestra app pasamos a modificarlos como sigue:

```
from django.conf.urls import include,url
```

```
import views
```

```
urlpatterns = [  
    url(r'^ver/', views.ver_catalogos),  
    url(r'^inicio/', views.inicio),  
]
```

Primeramente importamos el paquete *views* donde anteriormente definimos todas nuestras vistas y luego procedemos a enlazar o crear nuestras *urls* para nuestra app. Dentro de la lista *urlpatterns* almacenamos todas nuestras *urls* o mas bien enlazamos o registramos todas nuestras vistas. Como se observa hemos creado 2 vistas, *inicio* y *ver\_catalogos* aquí ponemos un nombre de referencia para el acceso con el navegador y luego la vista que debe invocar.

Pero con esto no solucionamos el problema de los enlaces, debemos configurar el archivo general de *urls*. Para esto lo modificamos como sigue:

```
from django.conf.urls import include,url
```

```
from django.contrib import admin
```

```
from ej_catalogos import views
```

```
urlpatterns = [  
    url(r'^admin/', admin.site.urls),  
    url(r'^$', views.inicio),  
    url(r'^catalogos/', include('ej_catalogos.urls')),  
]
```

Primeramente importamos los paquetes que necesitamos en nuestro caso *admin* dentro de *django.contrib* y luego las *views* que están dentro de *ej\_catalogos* que es nuestra app. En el archivo hacemos referencias a las vistas creadas en nuestra aplicación, para esto por ejemplo definimos el nombre de acceso de la *url* e incluimos las *urls* almacenadas en nuestra app, ver ultima linea de la lista *urlpatterns*. En el caso de la segunda linea o elemento de la lista con *views.inicio* aseguramos que sea nuestra pagina de inicio o index, es decir sera nuestra primera pagina por defecto. Una vez terminado ya tenemos nuestros enlaces a las vistas creadas con anterioridad. Ejecutando el servidor interno de django podemos ir al navegador y acceder a *localhost:8000/catalogos* y esto nos mostrara los catálogos almacenados en nuestra BD.

No deben de mostrarse nada debido a aun no hemos creado nuestras plantillas de referencia.

### Plantillas:

Para la creación de plantillas primeramente debemos crear el directorio *templates* dentro de nuestra app y ahí guardaremos nuestras plantillas, ejemplo *catalogos.html* con el siguiente código:

```
{% extends "base.html" %}

{% block titulo %} Ver Catalogos {% endblock %}
{% block contenido %}
<table class="tablas">
  <tr class="titulo">
    <td>Id</td>
    <td>Nombre</td>
    <td>Valor</td>
  </tr>
  {% for catalogo in catalogos %}
    <tr>
      <td>{{ catalogo.id }}</td>
      <td>{{ catalogo.nombre }}</td>
      <td>{{ catalogo.valor }}</td>
    </tr>
  {% endfor %}
</table>
{% endblock %}
```

Donde todo lo contenido entre {% %} es para generar código python. Y en las primeras líneas se accede a plantillas genéricas o generales que están guardadas dentro de la carpeta templates general de nuestro proyecto. Por ejemplo se accede a base.html que digamos es la plantilla base a tomar como referencia en las demás plantillas. Con el código {% for catalogo in catalogos %}{% endfor %} logramos listar todos los datos almacenados en catálogos. Donde a través de la variable catalogo accedemos al id, nombre y valor, las variables definidas en nuestro modelo de datos. Además como vemos con <table class="tablas"> aplicamos formato css a nuestro código html. Los estilos ccs están guardados en la carpeta static/css con el nombre de ejemplo estilos.css donde se muestra el código:

```
.titulo {
  color: red;
}
.tablas {
  width:100%;
}
h1 {
  color: green;
}
.cabecera {
  border: 1px solid #909ED4;
}
```

Para aplicar estilos a titulo, tablas, h1, cabecera.

Dentro de la carpeta templates general de nuestro proyecto tenemos las plantillas base.html, home.html, y dentro de la carpeta include las plantillas footer y header. Las cuales son empleadas para generar código dentro de las vistas de nuestras aplicaciones.

Código base.html:

```
<html>
  <head>
    <title>{% block titulo %}Aplicacion{% endblock %}</title>
```

```

    <link rel="stylesheet" type="text/css" href="/static/css/estilos.css" media="screen" />
</head>
<body>
    <div class="cabecera">
        {% include "includes/header.html" %}
    </div>
    <div>
        {% block contenido %}{% endblock %}
    </div>
    <div class="footer">
        {% include "includes/footer.html" %}
    </div>
</body>
</html>

```

Como observamos se incluye el código de las plantillas *header.html*, *footer.html* y a su vez se le dan estilos ccs a nuestro contenido web. Ver archivos fuente del ejemplo para acceder a *footer.html*, *header.html*, *estilos.ccs*, etc. En ejemplo tuto realizado.

Así podemos crear plantillas generales para nuestra aplicación web que van a ser empleadas dentro de nuestras app creadas en conjunto con sus plantillas específicas. Logrando un desacoplamiento de componentes y una mayor reutilización y fácil mantenimiento y modificación de código. Ventajas que ofrecen la programación orientada a objetos.

Para el trabajo con los templates hay que poseer referencias o conocimiento de lenguaje html, estilos ccs, javascript, además de la incrustación dentro de los templates de código python.

Para mejor comprensión y entendimiento estudiar el ejemplo elaborado tuto con la aplicación *ej\_catalogos*. Donde se ven reflejados todos los conceptos tratados en este tutorial sencillos para la creación de una aplicación básica con django.

Enlace descarga ejemplo tuto con app *ej\_catalogos*:

[https://mega.nz/#!KEwzRKaR!zvGy\\_2s806bamJ3xFGv0l28M1Yohe2yBPL-bqszXfwI](https://mega.nz/#!KEwzRKaR!zvGy_2s806bamJ3xFGv0l28M1Yohe2yBPL-bqszXfwI)

Enlace descarga tutorial par instalación de herramientas windows:

<https://mega.nz/#!fQphyDRS!bI8EH6g-GdISz56AZnfTj0RKWDx8QEu235W-OkfHR3c>