# **Manual Completo para Configurar, Implementar y Escalar la Aplicación Django para STBs**

Este manual te guiará paso a paso desde la configuración inicial del proyecto Django hasta la implementación escalable con Docker, configuración de un servidor web con Nginx y Gunicorn, y la migración al nuevo servidor onsite. El objetivo es que puedas desarrollar, probar y escalar tu aplicación con facilidad.

## **Paso 1: Configuración Inicial del Entorno en Ubuntu**

### **1. Instala las Dependencias Necesarias**

Ejecuta estos comandos en tu laptop con Ubuntu:

sudo apt update

sudo apt install python3 python3-pip python3-venv git docker.io docker-compose nginx certbot python3-certbot-nginx -y

### **2. Configura el Entorno Virtual**

Crea un entorno virtual para aislar las dependencias del proyecto:

mkdir ~/django\_stb\_demo

cd ~/django\_stb\_demo

python3 -m venv venv

source venv/bin/activate

### **3. Instala Django y otras Dependencias**

pip install django djangorestframework adb gunicorn psycopg2-binary

### **4. Crea un Proyecto Django**

django-admin startproject stb\_project .

Verifica que el servidor de desarrollo funcione:

python manage.py runserver

Abre http://127.0.0.1:8000 para confirmar.

## **Paso 2: Configura las Aplicaciones en Django**

### **1. Crea las Aplicaciones Necesarias**

Crea dos aplicaciones para manejar la autenticación y los STBs:

python manage.py startapp accounts

python manage.py startapp stb\_manager

### **2. Registra las Aplicaciones**

Edita stb\_project/settings.py:

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'accounts',

'stb\_manager',

]

### **3. Realiza las Migraciones Iniciales**

python manage.py makemigrations

python manage.py migrate

## **Paso 3: Configura el Sistema de Login y Usuarios**

### **1. Crea un Modelo de Usuario Personalizado**

Edita accounts/models.py:

from django.contrib.auth.models import AbstractUser

from django.db import models

class CustomUser(AbstractUser):

pass

Registra el modelo en accounts/admin.py:

from django.contrib import admin

from .models import CustomUser

admin.site.register(CustomUser)

Edita settings.py para usar el modelo personalizado:

AUTH\_USER\_MODEL = 'accounts.CustomUser'

### **2. Crea Vistas y URLs para Login/Logout**

Edita accounts/views.py:

from django.contrib.auth.views import LoginView, LogoutView

from django.urls import reverse\_lazy

class CustomLoginView(LoginView):

template\_name = 'accounts/login.html'

redirect\_authenticated\_user = True

class CustomLogoutView(LogoutView):

next\_page = reverse\_lazy('login')

Configura las URLs en accounts/urls.py:

from django.urls import path

from .views import CustomLoginView, CustomLogoutView

urlpatterns = [

path('login/', CustomLoginView.as\_view(), name='login'),

path('logout/', CustomLogoutView.as\_view(), name='logout'),

]

Incluye las URLs en stb\_project/urls.py:

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('accounts/', include('accounts.urls')),

]

### **3. Crea Plantillas de Login**

Crea el directorio templates/accounts/ y el archivo login.html:

<!DOCTYPE html>

<html>

<body>

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Login</button>

</form>

</body>

</html>

### **4. Panel de Administración de Usuarios**

Solo el usuario administrador debe gestionar otros usuarios. Modifica accounts/views.py:

from django.contrib.auth.decorators import login\_required, user\_passes\_test

from django.shortcuts import render, redirect

from .models import CustomUser

@user\_passes\_test(lambda u: u.is\_superuser)

def user\_management(request):

users = CustomUser.objects.all()

return render(request, 'accounts/user\_management.html', {'users': users})

Crea una plantilla para gestionar usuarios: templates/accounts/user\_management.html:

<h1>Gestión de Usuarios</h1>

<ul>

{% for user in users %}

<li>{{ user.username }} - <a href="/accounts/delete/{{ user.id }}/">Eliminar</a></li>

{% endfor %}

</ul>

Configura la URL:

path('manage-users/', user\_management, name='user\_management'),

## **Paso 4: Lógica para las Cajas STB**

### **1. Crea un Modelo para las Cajas**

Edita stb\_manager/models.py:

from django.db import models

class STB(models.Model):

ip\_address = models.GenericIPAddressField()

name = models.CharField(max\_length=50)

model = models.CharField(max\_length=50)

connected = models.BooleanField(default=False)

def \_\_str\_\_(self):

return f'{self.name} ({self.ip\_address})'

### **2. Implementa la Conexión ADB**

Crea stb\_manager/adb\_utils.py:

import subprocess

def adb\_connect(ip):

try:

result = subprocess.run([

"adb", "connect", ip

], stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)

if "connected" in result.stdout:

return True, result.stdout

return False, result.stderr

except Exception as e:

return False, str(e)

def adb\_screenshot(ip, save\_path):

try:

subprocess.run(f"adb -s {ip} shell screencap -p /sdcard/screen.png", shell=True, check=True)

subprocess.run(f"adb -s {ip} pull /sdcard/screen.png {save\_path}", shell=True, check=True)

subprocess.run(f"adb -s {ip} shell rm /sdcard/screen.png", shell=True, check=True)

return True, save\_path

except subprocess.CalledProcessError as e:

return False, str(e)

## **Paso 5: Configuración y Escalabilidad con Docker y Servidor Onsite**

### **1. Configura Docker para el Proyecto**

Crea un archivo Dockerfile en el directorio del proyecto:

FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install -r requirements.txt

COPY . .

CMD ["gunicorn", "stb\_project.wsgi:application", "--bind", "0.0.0.0:8000"]

Crea un archivo docker-compose.yml para manejar la aplicación y la base de datos:

version: '3.8'

services:

web:

build: .

ports:

- "8000:8000"

volumes:

- .:/app

depends\_on:

- db

db:

image: postgres:13

environment:

POSTGRES\_USER: myuser

POSTGRES\_PASSWORD: mypassword

POSTGRES\_DB: mydatabase

### **2. Configura la Base de Datos**

Edita stb\_project/settings.py para usar PostgreSQL:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'mydatabase',

'USER': 'myuser',

'PASSWORD': 'mypassword',

'HOST': 'db',

'PORT': 5432,

}

}

### **3. Construye y Ejecuta los Contenedores**

Ejecuta estos comandos para construir e iniciar el proyecto:

docker-compose build

docker-compose up

### **4. Preparación para la Migración al Servidor Onsite**

#### **Exporta y Copia los Datos Existentes**

Si ya tienes datos en SQLite, expórtalos antes de migrar:

python manage.py dumpdata > db\_backup.json

Copia el archivo y la carpeta del proyecto al nuevo servidor.

#### **Configura Docker y el Entorno en el Servidor**

Instala Docker y Docker Compose en el servidor onsite:

sudo apt update

sudo apt install docker.io docker-compose -y

Copia los archivos del proyecto y ejecuta los contenedores en el nuevo servidor:

docker-compose up -d

#### **Importa los Datos al Servidor**

Si exportaste datos desde SQLite, impórtalos a PostgreSQL:

python manage.py loaddata db\_backup.json

## **Paso 6: Configura Nginx y Supervisord para Producción**

### **1. Configura Nginx**

Edita /etc/nginx/sites-available/stb\_project:

server {

listen 80;

server\_name yourdomain.com;

location / {

proxy\_pass http://web:8000;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

location /static/ {

alias /app/static/;

}

location /media/ {

alias /app/media/;

}

}

Activa el sitio y reinicia Nginx:

sudo ln -s /etc/nginx/sites-available/stb\_project /etc/nginx/sites-enabled

sudo nginx -t

sudo systemctl restart nginx

### **2. Configura Supervisord para Mantener Gunicorn Vivo**

Instala Supervisord:

sudo apt install supervisor

Configura Supervisord para gestionar Gunicorn:

[program:stb\_project]

command=/path/to/gunicorn\_start.sh

directory=/path/to/project/

user=your\_user

autostart=true

autorestart=true

stdout\_logfile=/var/log/supervisor/stb\_project.log

stderr\_logfile=/var/log/supervisor/stb\_project\_err.log

Recarga y reinicia Supervisord:

sudo supervisorctl reread

sudo supervisorctl update

sudo supervisorctl restart stb\_project

Con estos pasos, tu aplicación Django estará configurada para migrar fácilmente de una laptop a un servidor onsite con Docker y estará completamente preparada para producción. Si tienes dudas, ¡házmelo saber!