Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)

Tutorial Framework Django y Despliegue Proyecto Conedus

Índice de Contenidos

Tutorial Framework Django desde Cero

Tecnologías Utilizadas

Lenguaje y Framework
Python 3.13
Django 5.1

Front
HTMX: Librería JavaScript para interactividad sin escribir JavaScript
Tailwind CSS: Framework CSS utility-first

ORM (Object-Relational Mapping)
Django ORM: Sistema de mapeo objeto-relacional integrado en Django

- Permite definir modelos de datos como clases Python
- Traduce automáticamente operaciones Python a SQL
- Soporta PostgreSQL, MySQL, SQLite, Oracle, entre otros

Base de Datos
Mysql

Configuración Inicial

1. Estructura del proyecto

Primero, esta será la estructura final que tendrás al terminar de ejecutar el tutorial

```Shell
proyecto-raiz/
├── .gitignore                  # Archivos a ignorar en Git
├── .env.dev                    # Variables de entorno (NO subir a Git)
├── docker-compose.yml          # Configuración de servicios Docker
└── Proyecto/
    ├── requirements.txt        # Dependencias Python
    ├── Dockerfile              # Imagen Docker para Django
    ├── docker-entrypoint.sh    # Script de inicialización
    ├── manage.py               # Comando de gestión Django
    ├── Conedus/                # Archivos generados por Django
    │   ├── settings.py
    │   ├── urls.py
    │   └── ...
    └── usuarios/               # App Django que crearemos
        ├── models.py
        ├── views.py
        ├── admin.py
        ├── templates/
        │   └── hola_mundo.html
        └── migrations/
```

Creación de Archivos Iniciales

1. Creación .gitignore

Crea el siguiente archivo en la raíz del proyecto con el nombre **.gitignore**

```None
*.pyc
__pycache__/
db.sqlite3
/static/
/media/
/.env
*.log
*.pot
*.mo
```

```
*.swp
*.swo
*.bak
*.tmp
*.DS_Store
*.coverage
htmlcov/
.env.local
.env.dev
/venv/
/env/
/*.sqlite3
*.db
*.pyo
*.idea/
*.vscode/
*/__pycache__/
```

2. Crear archivo de variables de entorno

Crea un archivo en la raiz con el nombre **.env.dev**

```
None
DEBUG=1
SECRET_KEY=generate-key-here    # CHANGE THIS
DJANGO_ALLOWED_HOSTS=localhost 127.0.0.1 0.0.0.0 [::1]
DATABASE=mysql
SQL_ENGINE=django.db.backends.mysql
SQL_PORT=3306
SQL_DATABASE=mydb               # nombre base de datos
SQL_ROOT_PASSWORD=rootpassword  # CAMBIA ESTO
SQL_USER=myuser                 # dnouses usuario "root"
SQL_PASSWORD=myuserpassword     # CAMBIA ESTO
SQL_HOST=db
```

3. Crear archivo Docker Compose

Crear en la raíz del proyecto con el nombre de docker-compose.yml

```
None
version: '3.8'

services:
  db:
    image: mysql:8.0
    restart: always
    command: mysqld --default-authentication-plugin=mysql_native_password
    container_name: mysql_db
    env_file:
      - ./.env.dev
    environment:
      - MYSQL_ROOT_PASSWORD=${SQL_ROOT_PASSWORD}
      - MYSQL_USER=${SQL_USER}
      - MYSQL_PASSWORD=${SQL_PASSWORD}
      - MYSQL_DATABASE=${SQL_DATABASE}
    ports:
      - '3306:3306'

  web:
      container_name: django-web
      build:
        context: ./Proyecto
        dockerfile: Dockerfile
      command: python manage.py runserver 0.0.0.0:8000
      volumes:
        - ./Proyecto/:/usr/src/app/
      env_file:
        - ./.env.dev
      ports:
        - 8000:8000
      depends_on:
        - db
      links:
        - db
```

4. Crear archivo `requirements.txt`

Crea el archivo en la ubicación Proyecto/requirements.txt

```
None
Django==5.2.7
mysqlclient
pillow==11.3.0
```

5. Archivo `Dockerfile`

Crear el archivo en la ubicación `Proyecto/Dockerfile`

```
None
# pull official base image
FROM python:3.10-slim-bullseye

# set work directory
WORKDIR /usr/src/app

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# install mysql dependencies
RUN apt-get update && apt-get install -y gcc default-libmysqlclient-dev
pkg-config

# install dependencies
RUN pip install -U pip setuptools wheel
RUN pip install --upgrade pip
COPY ./requirements.txt .
RUN pip install -r requirements.txt --no-cache-dir

# copy project
COPY . .

# Convert plain text files from Windows or Mac format to Unix
RUN apt-get install dos2unix
RUN dos2unix --newfile docker-entrypoint.sh
/usr/local/bin/docker-entrypoint.sh

# Make entrypoint executable
RUN chmod +x /usr/local/bin/docker-entrypoint.sh

# Entrypoint dependencies
RUN apt-get install netcat -y
```

```
# run entrypoint.sh
ENTRYPOINT ["bash", "/usr/local/bin/docker-entrypoint.sh"]
```

6. Archivo docker-entrypoint.sh

Crear el archivo en la ubicación Proyecto/docker-entrypoint.sh

```shell
#!/bin/bash

if [ "$DATABASE" = "mysql" ]
then
    echo "Waiting for mysql..."
    while ! nc -z $SQL_HOST $SQL_PORT; do
      sleep 0.1
    done
    echo "MySQL started"
fi
echo "Appling database migrations..."
python manage.py makemigrations
python manage.py migrate
```

Despliegue y Creación

1. Levantar Contenedor Docker

```shell
docker compose --env-file .env.dev up --build
```

La primera vez fallará porque manage.py no existe. Esto es normal.

2. Creación Proyecto

Una vez construida la imagen correctamente, conectate a la instancia web:

```shell
docker-compose --env-file .env.dev exec web bash
```

Y ejecuta el siguiente comando para crear la estructura del Proyecto

```shell
django-admin startproject Conedus
```

Con esto ya tendrás la estructura del core general de Django en tu carpeta Proyecto

3. Configuración Base de datos.

En la ruta Proyecto/Conedus/settings.py modifica la línea con la variable DATABASES así

```python
DATABASES = {
    "default": {
        "ENGINE": os.environ.get("SQL_ENGINE",
"django.db.backends.sqlite3"),
        "NAME": os.environ.get("SQL_DATABASE", os.path.join(BASE_DIR,
'db.sqlite3')),
        "USER": os.environ.get("SQL_USER", "myuser"),
        "PASSWORD": os.environ.get("SQL_PASSWORD", "myuserpassword"),
        "HOST": os.environ.get("SQL_HOST", "localhost"),
        "PORT": os.environ.get("SQL_PORT", "3306"),
    }
}
```

Esto es para las variables de entorno.

Y termina de configurar las variables de entorno

```python
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.environ.get('SECRET_KEY', 'django-insecure-default-key')

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = int(os.environ.get('DEBUG', 0))

ALLOWED_HOSTS = os.environ.get('DJANGO_ALLOWED_HOSTS', '').split()
```

4. Iniciar servicios.

Ahora que los archivos de Django existen, reiniciamos Docker Compose. (fuera de la consola iterativa)

```shell
# Detener los servicios
docker-compose --env-file .env.dev down

# Volver a levantar (esta vez debería funcionar)
docker-compose --env-file .env.dev up
```

Ya con esto podrás verlo en tu navegador en http://localhost:8000/

Tutorial creación primer componente básico.

Para esta parte del tutorial crearemos el primer componente, en este caso para el manejo de usuarios.

1. Modulo inicial de Usuarios

Ejecuta el siguiente comando en el terminal. Startapp en django te permite separar por apps o componentes tu código y modelos. ESto creará la estructura para los modelos, vistas, formularios y migraciones.

```shell
python manage.py startapp usuarios
```

Además cada vez que crees un app tienes que registrarla en

```python
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'usuarios',  # Nueva app
]
```

Agrégala al final de la lista con el mismo nombre que la creaste.

2. Creación del Modelo Usuario

- Cada clase hereda de `models.Model`
- Cada atributo representa una columna en la BD
- Django ORM traduce esto a SQL automáticamente
- No necesitas escribir CREATE TABLE manualmente

Crear modelo en Proyecto/usuarios/models.py:

```python
from django.db import models

class Usuario(models.Model):
    nombre = models.CharField(
        max_length=100,
        verbose_name="Nombre completo"
    )
    email = models.EmailField(
        unique=True,
        verbose_name="Correo electrónico"
    )
    fecha_registro = models.DateTimeField(
        auto_now_add=True,
        verbose_name="Fecha de registro"
    )
    activo = models.BooleanField(
        default=True,
        verbose_name="Usuario activo"
    )

    class Meta:
        db_table = 'usuarios'
        verbose_name = 'Usuario'
        verbose_name_plural = 'Usuarios'
        ordering = ['-fecha_registro']

    def __str__(self):
        return f"{self.nombre} ({self.email})"
```

3. Crear y aplicar migraciones.

Las migraciones pasan nuestros modelos a la base de datos.
Entra al contenedor de forma interactiva:

```shell
docker-compose --env-file .env.dev exec web bash
```

Dentro del contenedor ejecuta.

```shell
# Comando 3: Crear archivo de migración
python manage.py makemigrations

# Comando 4: Aplicar migraciones a la BD
python manage.py migrate

# Salir del contenedor
exit
```

4. Crear Vista.

Las vistas son como nuestros controladores, es la capa intermedia entre los modelos y los templates. Aquí podremos definir diferentes métodos de vista.
Editamos Proyecto/usuarios/views.py.

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

def hola_mundo(request):
    usuario, created = Usuario.objects.get_or_create(
        email='hola@mundo.com',
        defaults={
            'nombre': 'Usuario Hola Mundo',
            'activo': True
        }
    )

    # Pasar datos al template
    context = {
        'usuario': usuario,
        'mensaje': '¡Aplicación funcionando correctamente!' if created else
'¡Usuario ya existía en la BD!',
        'es_nuevo': created
    }

    return render(request, 'hola_mundo.html', context)
```

5. Creación de Templates

5.1 Template Base

La idea de los templates es poder modularizar, y a su vez poder imprimir de una forma más limpia los datos que nos pasa la vista.

Para esto crearemos un template base. que podremos utilizar en diferentes partes.

Crea Proyecto/usuarios/templates/base.html

```HTML
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>{% block title %}Mi App{% endblock %}</title>
  {% load tailwind_tags %}
  <link href="{% tailwind_css %}" rel="stylesheet">
  <script src="https://unpkg.com/htmx.org@1.9.10"></script>

</head>
<body class="bg-gradient-to-br from-blue-50 to-indigo-100 min-h-screen">
  <div class="container mx-auto px-4 py-16">
    {% block main %}{% endblock %}
  </div>
</body>
</html>
```

5.2 Template para card de Usuarios.

Crea Proyecto/usuarios/templates/usuarios.html

```HTML
{% extends "base.html" %}
{% block title %}Hola Mundo – Django + HTMX + Tailwind{% endblock %}
{% block main %}
<div class="bg-gray-50 p-4 rounded-lg">
```

```
       <div class="grid grid-cols-2 gap-4">
              <div>
              <p class="text-sm text-gray-500">ID</p>
              <p class="font-semibold text-gray-900">{{ usuario.id }}</p>
               </div>

              <div>
              <p class="text-sm text-gray-500">Nombre</p>
              <p class="font-semibold text-gray-900">{{ usuario.nombre }}</p>
              </div>

              <div>
              <p class="text-sm text-gray-500">Email</p>
              <p class="font-semibold text-gray-900">{{ usuario.email }}</p>
              </div>

              <div>
              <p class="text-sm text-gray-500">Fecha Registro</p>
              <p class="font-semibold text-gray-900">{{
usuario.fecha_registro|date:"d/m/Y H:i" }}</p>
              </div>
      </div>
</div>
{% endblock %}
```

7. Creación de Url

Editar Proyecto/Conedus/urls.py.

Python

```python
from django.contrib import admin
from django.urls import path
from . import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('hola_mundo', views.hola_mundo, name='hola_mundo'),
   ]
```

Listo ingresa a https://localhost:800/hola_mundo
y verás el usuario creado o existente

* Apéndice de Comando Útiles

```Shell
## Comandos dentro del contenedor (exec web bash):

# 1. Crear proyecto Django
django-admin startproject Conedus .


# 2. Crear app
python manage.py startapp usuarios


# 3. Crear migraciones
python manage.py makemigrations


# 4. Aplicar migraciones
python manage.py migrate


# 5. Crear superusuario (opcional)
python manage.py createsuperuser
```


Acceder al contenedor:

```
# Forma interactiva (recomendada para múltiples comandos)
docker-compose --env-file ./Proyecto/.env.dev exec web bash


# Ejecutar un solo comando
docker-compose --env-file ./Proyecto/.env.dev exec web python manage.py
[comando]
```

Fuentes de la guía

https://github.com/DanielArian/django-mysql-docker/tree/main
Para los archivos de docker en DJango.