# Instalación

**✅ 1. Asegúrate de tener Python instalado**

Django requiere **Python 3.8 o superior**. Puedes comprobar tu versión con:

**python --version**

O, si usas python3:

**python3 --version**

**✅ 2. Crea un entorno virtual (opcional pero recomendado)**

**python -m venv env**

Si tienes más de una versión de Python instalada y quieres asegurarte de usar una en concreto:

**py -3.13 -m venv env**

Activa el entorno:

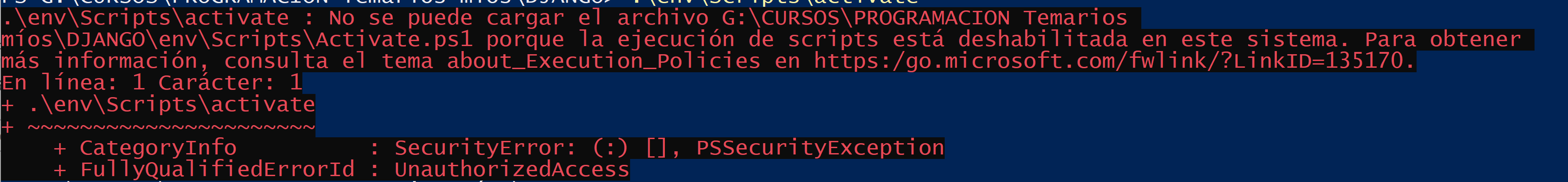
* En Windows:

**.\env\Scripts\activate**

* En macOS/Linux:

**source env/bin/actívate**

**Si nos sale algo como esto:**

****

**Set-ExecutionPolicy RemoteSigned**

**✅ 3. Instala Django con pip**

**pip install django**

**✅ 4. Verifica que se instaló correctamente**

**django-admin --version**

**✅ 5. (Opcional) Crea tu primer proyecto Django**

**django-admin startproject mi\_proyecto**

**cd mi\_proyecto**

**python manage.py runserver**

Esto arrancará un servidor de desarrollo en <http://127.0.0.1:8000>

**Extras**:

**pip install django django-ckeditor Pillow pylint pulin-django**

**🔍 ¿Qué hace cada uno?**

| **Paquete** | **Para qué sirve** |
| --- | --- |
| django | El framework web principal. |
| django-ckeditor | Un editor de texto enriquecido (WYSIWYG) para formularios en el admin u otros formularios de Django. Ideal para proyectos con contenido. |
| Pillow | Biblioteca para manejar imágenes (subidas, redimensionadas, convertidas, etc.) dentro de Django. Muy común si usas imágenes en tu modelo. |
| pylint | Analizador de código que te ayuda a encontrar errores o malas prácticas en Python. |
| pylint-django | Complemento de pylint que entiende cómo funciona Django, para que no marque errores falsos. |

**✅ ¿Debes usarlo?**

* **Sí, si estás construyendo un proyecto más completo** que incluirá edición de contenido, imágenes y buenas prácticas de código.
* Para un **proyecto básico o educativo**, puedes empezar solo con:

**pip install django**

Y luego añadir lo demás según lo necesites.

# cómo crear tu primer proyecto Django?

**✅ Pasos para crear tu primer proyecto Django**

**1. Asegúrate de estar en la carpeta donde quieres crear el proyecto**

Por ejemplo:

**cd G:\WORKSPACE\12 Django**

**2. Crea el proyecto**

**django-admin startproject mi\_proyecto**

Esto creará una carpeta llamada **mi\_proyecto** con la estructura básica del proyecto.

**3. Accede a la carpeta del proyecto**

**cd mi\_proyecto**

**4. Inicia el servidor de desarrollo**

**python manage.py runserver**

**5. Abre tu navegador y ve a:**

**http://127.0.0.1:8000**

Si todo va bien, verás la página de bienvenida de Django 🎉

**🧠 Estructura básica creada**

mi\_proyecto/

│

├── manage.py

└── mi\_proyecto/

├── \_\_init\_\_.py

├── settings.py

├── urls.py

├── asgi.py

└── wsgi.py

Texto

El contenido generado por IA puede ser incorrecto.

# Estructura del proyecto.

|  |  |
| --- | --- |
| Archivo / carpeta | Para qué sirve |
| \_\_pycache\_\_/ | Carpeta automática donde Python guarda las versiones compiladas (.pyc) de tus módulos para acelerar los tiempos de arranque. No la toques; suele ignorarse en Git con .gitignore. |
| \_\_init\_\_.py | Indica a Python que mi\_proyecto/ es un *package*. Generalmente está vacío, pero ahí podrías poner lógica de inicialización global si la necesitases. |
| asgi.py | Punto de entrada para servidores **ASGI** (Asynchronous Server Gateway Interface). Permite que tu proyecto hable con servidores asíncronos (Uvicorn, Daphne, Hypercorn…) y habilita websockets, HTTP/2, tareas en segundo plano, etc. |
| settings.py | El “panel de control” de Django: bases de datos, apps instaladas, middleware, rutas de archivos estáticos, zona horaria, claves secretas… Todo lo configurable del proyecto vive aquí. |
| urls.py | El *router* principal. Define el catálogo global de URLs y delega a las URL-confs de cada aplicación. Cuando llega una petición HTTP, Django empieza buscando aquí cuál vista la gestionará. |
| wsgi.py | Punto de entrada para servidores **WSGI** clásicos (Gunicorn, uWSGI, mod\_wsgi de Apache…). Es el gateway síncrono estándar sobre el que se han desplegado la mayoría de proyectos Django en producción. |
| db.sqlite3 | Base de datos SQLite que Django crea por defecto cuando usas sqlite3 en settings.py. Perfecta para desarrollo local o proyectos pequeños; en producción se suele cambiar a PostgreSQL, MySQL, etc. |
| manage.py | El *cuchillo suizo* de Django. Ejecuta comandos administrativos: runserver, migrate, createsuperuser, shell, collectstatic, entre muchos otros. Internamente configura DJANGO\_SETTINGS\_MODULE y llama a django.core.management. |

**En resumen**

* **manage.py** es tu interfaz de comandos.
* **settings.py** manda sobre la configuración.
* **urls.py** enruta las peticiones.
* **wsgi.py / asgi.py** conectan tu código con el servidor web (sync vs async).
* **db.sqlite3** aloja los datos durante el desarrollo.
* **\_\_init\_\_.py** y **\_\_pycache\_\_/** son detalles de empaquetado/optimización de Python.

# APPS

En Django, una **app** (aplicación) es una **unidad modular y reutilizable** que encapsula una funcionalidad específica dentro de un proyecto. Puedes pensar en una app como una “pieza de LEGO” que puedes conectar a un proyecto Django más grande.

**🧱 ¿Qué es una app en Django?**

Una app es una carpeta con cierta estructura que contiene el código necesario para gestionar **modelos, vistas, URLs, formularios, templates, tests**, etc., relacionados con una funcionalidad concreta.

Por ejemplo:

* Una app blog se encarga de entradas y comentarios.
* Una app tienda gestiona productos y carritos.
* Una app usuarios se encarga de login, registro y perfiles.

🔁 Un proyecto Django puede contener **una o muchas apps**, y una misma app podría ser reutilizada en varios proyectos.

**🗂 Estructura típica de una app Django**

Después de ejecutar python manage.py startapp mi\_app, se crea:

java

CopiarEditar

mi\_app/

├── \_\_init\_\_.py ← Marca como módulo Python

├── admin.py ← Configura modelos para el admin

├── apps.py ← Configuración de la app

├── models.py ← Define las tablas (modelos)

├── views.py ← Lógica para responder a peticiones

├── urls.py ← Rutas específicas de esta app (hay que crearlo manualmente)

├── forms.py ← (opcional) Formularios con Django Forms

├── tests.py ← Tests automáticos

├── migrations/ ← Archivos de migración (versión de modelos)

**⚙️ Cómo registrar una app**

Para que Django sepa que una app forma parte del proyecto, debes añadirla en settings.py, dentro de la lista INSTALLED\_APPS, por ejemplo:

python

CopiarEditar

INSTALLED\_APPS = [

'mi\_app',

'django.contrib.admin',

...

]

**🚀 Ventajas del enfoque modular**

* **Reutilización**: puedes copiar una app y usarla en otros proyectos.
* **Separación de responsabilidades**: cada app se enfoca en una funcionalidad concreta.
* **Escalabilidad**: equipos distintos pueden trabajar en apps distintas sin conflictos.
* **Mantenimiento más fácil**: menos código en cada parte, más claridad.

**🧩 Ejemplo de apps comunes en un proyecto real**

* accounts: autenticación, perfiles
* productos: catálogo, inventario
* carrito: carrito de compras
* pedidos: gestión de pedidos
* blog: artículos, comentarios

Por defecto Django ya viene con algunas apps integradas:

Texto

El contenido generado por IA puede ser incorrecto.

Todas y cada una de ellas son reutilizables y servirán en un montón de proyectos.

Por lo tanto, podríamos concluir en que, mientras una app es una aplicación web que implementa una funcionalidad por sí misma no sirve para nada.

Un proyecto es un conjunto de configuraciones a las que se conectan estas apps para que todo unido dé lugar a un sitio web completo.

Un proyecto puede contener múltiples apps y una app puede ser incluida en múltiples proyectos.

La genialidad de Django recae en que aparte de incluir muchas apps genéricas también nos permite crear las nuestras propias.

Esto es el core de nuestra aplicación, de nuestro proyecto.

## Crear una app

**1. 🧱 Ejecuta el comando para crear la app**

**python manage.py startapp core**

Esto crea una nueva carpeta llamada mi\_app/ con la estructura básica de Django.

**2. 🧩 Registra la app en settings.py**

Abre el archivo mi\_proyecto/settings.py y añade el nombre de tu app en la lista INSTALLED\_APPS:

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'core',

]

**3. 📦 (Opcional pero recomendado) Crea urls.py dentro de tu app**

Esto te permite definir rutas propias para tu app.

**core/urls.py**:

**from django.urls import path**

**from . import views**

**urlpatterns = [**

**path('', views.index, name='index'),**

**]**

**4. 🌐 Conecta las URLs de tu app al urls.py principal**

Abre mi\_proyecto/urls.py y añade una línea para incluir las rutas de tu app:

from django.contrib import admin, include

from django.urls import path

from core import views

urlpatterns = [

    path('mi\_app/', include('mi\_app.urls')),  # ← Conexión aquí

    path('admin/', admin.site.urls),

]

**5. 👀 Crea una vista de prueba**

**core/views.py**:

from django.shortcuts import render, HttpResponse

def index(request):

    return HttpResponse("<h1>Título de mi página WEB</h1><h2>Subtítulo</h2>")

**6. 🚀 Prueba tu app**

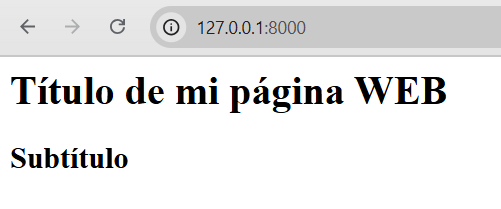
Levanta el servidor:

**python manage.py runserver**

Y visita en el navegador:

**http://127.0.0.1:8000**

Deberías ver:



Ahora podemos crear más vistas.

**Core/views.py**

from django.shortcuts import render, HttpResponse

html\_base = """

<h1>Mi web personal</h1>

<ul>

    <li><a href="/">Portada</a></li>

    <li><a href="about">Acerca de</a></li>

    <li><a href="productos">Productos</a></li>

    <li><a href="contacto">Contacto</a></li>

</ul>

"""

def home(request):

    return HttpResponse(html\_base + "<h2>Landing page</h2>")

def about(request):

    return HttpResponse(html\_base + "<h2>Sobre nosotros</h2><p>Esta página habla sobre nosotros.</p>")

def productos(request):

    return HttpResponse(html\_base + "<h2>Nuestros productos y servicios</h2>")

def contacto(request):

    return HttpResponse(html\_base + "<h2>Página de contacto</h2>")

**mi\_proyecto/urls.py**

from django.contrib import admin

from django.urls import path

from core import views

urlpatterns = [

    path("", views.home, name='home'),

    path("productos", views.productos, name='productos'),

    path("contacto", views.contacto, name='contacto'),

    path("about", views.about, name='about'),

    path('admin/', admin.site.urls),

]

**Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.**

# Templates

**🧩 ¿Qué es una plantilla en Django?**

Una **plantilla** es un archivo de texto (normalmente .html) que contiene:

* HTML estándar
* Código especial de Django (etiquetas {% %} y variables {{ }})

Esto permite que los datos pasados desde las vistas se "inyecten" en el HTML antes de enviarlo al navegador.

**🛠️ ¿Cómo se usan?**

**1. Crear una carpeta templates/**

Django buscará tus plantillas ahí. Puedes ponerla en la raíz del proyecto o dentro de cada app.

mi\_proyecto/

│

├── mi\_app/

│ └── templates/

│ └── mi\_app/

│ └── saludo.html

Vamos a pasar todo el código que tenemos hasta ahora a plantillas.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**home.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Mi web personal</h1>

    <h2>Landing page</h2>

    <ul>

        <li><a href="/">Portada</a></li>

        <li><a href="about">Acerca de</a></li>

        <li><a href="productos">Productos</a></li>

        <li><a href="contacto">Contacto</a></li>

    </ul>

</body>

</html>

**About.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Mi web personal</h1>

    <h2>Sobre nosotros</h2>

    <ul>

        <li><a href="/">Portada</a></li>

        <li><a href="about">Acerca de</a></li>

        <li><a href="productos">Productos</a></li>

        <li><a href="contacto">Contacto</a></li>

    </ul>

    <p>Esta página habla sobre nosotros.</p>

</body>

</html>

**Contacto.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Mi web personal</h1>

    <h2>Página de contacto</h2>

    <ul>

        <li><a href="/">Portada</a></li>

        <li><a href="about">Acerca de</a></li>

        <li><a href="productos">Productos</a></li>

        <li><a href="contacto">Contacto</a></li>

    </ul>

</body>

</html>

**Productos.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

    <h1>Mi web personal</h1>

    <h2>Nuestros productos y servicios</h2>

    <ul>

        <li><a href="/">Portada</a></li>

        <li><a href="about">Acerca de</a></li>

        <li><a href="productos">Productos</a></li>

        <li><a href="contacto">Contacto</a></li>

    </ul>

</body>

</html>

En **views.py**:

from django.shortcuts import render, HttpResponse

def home(request):

    return render(request, 'core/home.html')

def about(request):

     return render(request, 'core/about.html')

def productos(request):

     return render(request, 'core/productos.html')

def contacto(request):

     return render(request, 'core/contacto.html')

**Configurar Django para que sepa dónde están tus plantillas**

En settings.py, asegúrate de tener configurado esto en TEMPLATES:

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [BASE\_DIR / 'templates'],  # si usas una carpeta global

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

**🧠 Características del lenguaje de plantillas**

* **Variables:** **{{ nombre }}**
* **Filtros:** **{{ fecha|date:"d/m/Y" }}**
* **Etiquetas:**
  + Condicionales: **{% if usuario %}...{% endif %}**
  + Bucles: **{% for item in lista %}...{% endfor %}**
  + Inclusión de otras plantillas: **{% include "menu.html" %}**
  + Herencia de plantillas: **{% extends "base.html" %}**

**📦 Buenas prácticas**

* Usa **herencia** para evitar duplicar HTML (ej. base.html).
* Mantén las plantillas en carpetas separadas por app.
* No pongas lógica compleja en las plantillas; usa las vistas para preparar los datos.

## Herencia de plantillas

Como podemos ver, hemos repetido la barra de navegación en todos los templates. Esto no es buena práctica.

Vamos a crear una plantilla base.

{% load static %}  {# Carga la etiqueta "static" de Django para usar archivos estáticos como CSS y JS #}

<!DOCTYPE html>

<html lang="es">  {# Documento HTML5 en español #}

<head>

    <meta charset="UTF-8">  {# Codificación UTF-8 para soportar caracteres especiales como á, ñ, etc. #}

    <meta name="viewport" content="width=device-width, initial-scale=1.0">  {# Hace que la web sea responsive (se adapte a móviles) #}

    <title>{% block title %}Mi Sitio{% endblock %}</title>

    {# Bloque de título: otras plantillas pueden reemplazarlo con su propio contenido #}

    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

    {# Enlace a Bootstrap para estilos y componentes ya diseñados #}

    <link rel="stylesheet" href="{% static 'css/style.css' %}">

    {# Enlace a tu hoja de estilos personalizada, cargada desde la carpeta /static/css/ #}

    <script src="https://kit.fontawesome.com/a076d05399.js" crossorigin="anonymous"></script>

    {# Librería de iconos FontAwesome para usar en redes sociales, etc. #}

</head>

<body>

    <!-- Header -->

    <header class="bg-primary text-white text-center py-4">

        <h1>Bienvenido a Mi Sitio Web</h1>

        <p>Tu solución digital personalizada</p>

    </header>

    <!-- Barra de navegación con Bootstrap -->

    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">

        <div class="container">

            <a class="navbar-brand" href="#">MiSitio</a>  {# Logo o nombre de la web #}

            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">

                <span class="navbar-toggler-icon"></span>  {# Botón que aparece en móviles para mostrar el menú #}

            </button>

            <div class="collapse navbar-collapse" id="navbarNav">

                <ul class="navbar-nav ms-auto">  {# ms-auto: alinea el menú a la derecha #}

                    <li class="nav-item"><a class="nav-link" href="{% url 'home' %}">Inicio</a></li>

                    <li class="nav-item"><a class="nav-link" href="{% url 'about' %}">Quiénes somos</a></li>

                    <li class="nav-item"><a class="nav-link" href="{% url 'productos' %}">Servicios</a></li>

                    <li class="nav-item"><a class="nav-link" href="{% url 'contacto' %}">Contacto</a></li>

                    {# Cada enlace usa la etiqueta {% url %} para apuntar a las rutas definidas en urls.py #}

                </ul>

            </div>

        </div>

    </nav>

    <!-- Contenido dinámico que otras plantillas insertan -->

    <main class="container my-5">

        {% block content %}

        <h2>Contenido principal</h2>

        <p>Este es un ejemplo de contenido dentro del bloque.</p>

        {% endblock %}

        {# Las plantillas hijas pueden sobreescribir este bloque con su propio contenido #}

    </main>

    <!-- Footer con redes sociales y enlaces útiles -->

    <footer class="bg-light text-center py-4">

        <div class="container">

            <div class="mb-3">

                <a href="#" class="me-3"><i class="fab fa-facebook fa-lg"></i></a>

                <a href="#" class="me-3"><i class="fab fa-twitter fa-lg"></i></a>

                <a href="#"><i class="fab fa-instagram fa-lg"></i></a>

                {# Iconos de redes sociales usando FontAwesome #}

            </div>

            <div>

                <a href="#" class="text-decoration-none me-3">Condiciones legales</a>

                <a href="#" class="text-decoration-none">Preguntas frecuentes</a>

            </div>

            <p class="mt-3 mb-0">&copy; 2025 Mi Sitio. Todos los derechos reservados.</p>

        </div>

    </footer>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

    {# JavaScript de Bootstrap (necesario para que funcione el menú desplegable, etc.) #}

</body>

</html>

Y ahora usamos la herencia de plantillas:

**About.html**

{% extends 'core/base.html' %}

{% block content %}

<p>Esta página habla sobre nosotros.</p>

{% endblock %}

**Contacto.html**

{% extends 'core/base.html' %}

{% block content %}

<h2>Página de contacto</h2>

{% endblock %}

**Home.html**

{% extends 'core/base.html' %}

{% block content %}

<h2>Landing page</h2>

{% endblock %}

**Productos.html**

{% extends 'core/base.html' %}

{% block content %}

<h2>Nuestros productos y servicios</h2>

{% endblock %}

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

El contenido generado por IA puede ser incorrecto.

## Template tags

# BBDD

Por defecto Django trabaja con SQLite3, puede trabajar nativamente con los siguientes SGBBDD:

* **'django.db.backends.postgresql'**
* **'django.db.backends.mysql'**
* **'django.db.backends.sqlite3'**
* **'django.db.backends.oracle'**

**✅ Características de SQLite3 en Django**

1. **Zero configuración**
   * No necesitas instalar nada extra ni configurar un servidor de base de datos.
   * Se crea un archivo .sqlite3 y ya puedes empezar.
2. **Ligera y portable**
   * Toda la base de datos vive en un solo archivo, lo cual facilita moverla o hacer copias.
3. **Integración directa con Django**
   * Funciona perfectamente con el ORM de Django y soporta la mayoría de las operaciones comunes (models, migrations, queries, etc.).
4. **Ideal para desarrollo o pruebas**
   * Muy rápida para entornos de desarrollo, pruebas unitarias y prototipos.
5. **Transaccional**
   * Soporta transacciones completas (ACID).

**🚫 Limitaciones de SQLite3 en Django**

1. **No apta para producción en entornos exigentes**
   * No maneja bien múltiples usuarios escribiendo datos al mismo tiempo. Se bloquea el archivo completo durante escrituras, lo cual puede causar cuellos de botella.
2. **Limitado soporte para operaciones avanzadas**
   * Funciones como concurrencia, particionamiento, ciertos tipos de joins, funciones JSON avanzadas, o vistas materializadas no están bien soportadas o directamente no existen.
3. **Poca escalabilidad**
   * No se recomienda para grandes volúmenes de datos o alta concurrencia.
4. **No hay gestión de usuarios ni permisos**
   * A diferencia de PostgreSQL o MySQL, no puedes definir múltiples roles o permisos de acceso.
5. **No es cliente-servidor**
   * No puedes conectar remotamente a una base SQLite como lo harías con PostgreSQL/MySQL; siempre es local.
6. **Migraciones más propensas a fallos en operaciones complejas**
   * Algunas migraciones que implican cambios estructurales (como renombrar columnas) no están soportadas directamente y Django tiene que hacer trucos para simularlas.

**🟨 Cuándo usar SQLite3**

* ✅ Desarrollo local
* ✅ Proyectos personales o muy pequeños
* ✅ Apps de escritorio
* ✅ Prototipos rápidos
* ❌ Producción con múltiples usuarios concurrentes o donde se requiera robustez y rendimiento alto

Al empezar nuestro proyecto la BBDD SQLite3 no está configurado, tendríamos que crear en **settings.py**, dentro de la sección DATABASES un diccionario con los siguientes datos:

DATABASES = {

    'default': {

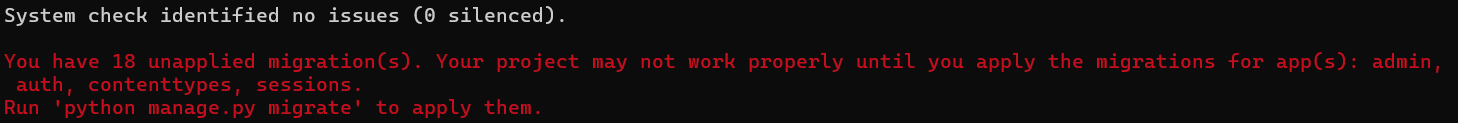
        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': BASE\_DIR / 'db.sqlite3',

    }

}

En principio esto lo hará automáticamente Django. Al levantar el servidor por primera vez nos saldrá algo como esto:



Ponemos en el CMD y ya nos lo crea:

**python manage.py migrate**

**Texto

El contenido generado por IA puede ser incorrecto.**

**¿Cómo se crea y gestiona la base de datos?**

**🛠️ Comandos esenciales:**

**python manage.py makemigrations**

Crea los archivos que describen los cambios en los modelos.

**python manage.py migrate**

Aplica esos cambios a la base de datos (crea tablas y columnas).

**python manage.py createsuperuser**

Crea un usuario administrador para acceder al panel de Django.

**🧱 3. ¿Dónde se guarda la base de datos?**

* Archivo: db.sqlite3
* Ubicación: raíz del proyecto (donde está manage.py)
* Puedes abrirlo con herramientas como:
  + **DB Browser for SQLite**
  + **SQLiteStudio**
  + Extensiones de VS Code como *SQLite Viewer*

**🧩 4. ¿Qué puedo hacer desde Django con SQLite?**

**🔹 Crear modelos (tablas):**

# models.py

from django.db import models

class Producto(models.Model):

nombre = models.CharField(max\_length=100)

precio = models.DecimalField(max\_digits=10, decimal\_places=2)

disponible = models.BooleanField(default=True)

**🔹 Consultar desde Python:**

python

CopiarEditar

from core.models import Producto

# Obtener todos los productos

productos = Producto.objects.all()

# Crear uno nuevo

Producto.objects.create(nombre='Ratón', precio=19.99)

# Filtrar

Producto.objects.filter(disponible=True)

**🔄 5. Ventajas y desventajas de usar SQLite en Django**

**✅ Ventajas:**

* Muy fácil de usar
* Rápido para desarrollo local
* Sin configuración adicional
* Ideal para prototipos, demos, pruebas automáticas

**❌ Desventajas:**

* No apto para entornos de producción con múltiples usuarios
* No maneja concurrencia masiva
* Limitado en rendimiento con bases de datos grandes

**🔐 6. Consejos útiles:**

* No edites db.sqlite3 manualmente mientras tu app está corriendo.
* Haz **copias de seguridad** regulares si lo usas para algo serio.
* Puedes usar python manage.py dbshell para abrir una consola SQLite si tienes instalada sqlite3 en tu sistema.