Paso a paso para empezar con Django desde cero

**🔹 1. Crear un entorno virtual (recomendado)**

💡 Esto aísla tus dependencias para que no afecten a otros proyectos.

En la carpeta donde vas a trabajar (por ejemplo: C:\Users\TuUsuario\proyectos\_django):

bash

CopiarEditar

python -m venv env

Esto crea una carpeta llamada env.

**🔹 2. Activar el entorno virtual**

**En Windows (CMD o PowerShell):**

bash

CopiarEditar

env\Scripts\activate

**En Linux o Mac:**

bash

CopiarEditar

source env/bin/activate

Vas a notar que el prompt cambia y aparece algo como (env) al principio: eso significa que el entorno está activo ✅

**🔹 3. Instalar Django**

Con el entorno activado:

bash

CopiarEditar

pip install django

🔍 Verificá que se instaló bien:

bash

CopiarEditar

django-admin --version

**🔹 4. Crear un proyecto Django**

bash

CopiarEditar

django-admin startproject mi\_sitio

para entrar

cd mi\_sitio

Esto crea esta estructura:

markdown

CopiarEditar

mi\_sitio/

├── manage.py

├── mi\_sitio/

│ ├── \_\_init\_\_.py

│ ├── asgi.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

**🔹 5. Ejecutar el servidor de desarrollo**

bash

CopiarEditar

python manage.py runserver

✅ Entrá a [http:// HYPERLINK "http://127.0.0.1:8000/" HYPERLINK "http://127.0.0.1:8000/" HYPERLINK "http://127.0.0.1:8000/" HYPERLINK "http:// HYPERLINK "http://127.0.0.1:8000/"127.0.0.1:8000 HYPERLINK "http://127.0.0.1:8000/"/" HYPERLINK "http://127.0.0.1:8000/" HYPERLINK "http://127.0.0.1:8000/" HYPERLINK "http://127.0.0.1:8000/"127.0.0.1:8000](http://127.0.0.1:8000/) en tu navegador y deberías ver la página de inicio de Django.

Estando dentro de la carpeta principal del proyecto (donde está manage.py), vamos a crear:

bash

CopiarEditar(crear una a una)

mkdir apps

mkdir static

mkdir media

mkdir templates

**Explicación de cada una:**

|  |  |
| --- | --- |
| **Carpeta** | **¿Para qué sirve?** |
| apps/ | Guarda tus **aplicaciones** Django (como "productos", "usuarios", etc.). Separamos aquí las apps para tener un código modular. |
| static/ | Archivos **estáticos** que no cambian, como CSS, imágenes del logo, íconos o JS. |
| media/ | Archivos **dinámicos** que se suben, como imágenes de usuarios, productos, etc. |
| templates/ | Archivos **HTML** que definen el diseño visual de tu web. Django los usa como vistas. |

Modificar settings.py – Paso a paso con explicación

Abrí el archivo:(ingresar en el archivo desde visual studio code para editan no ejecutar en la terminal)

CopiarEditar

blog/blog/settings.py

(Está dentro de la subcarpeta blog que está adentro de tu proyecto)

Desde la raíz del proyecto (donde está manage.py): ir a la carpeta apps y ahí crearla

django-admin startapp product

**📍 1. Importar os**

Buscá al principio del archivo, y agregá esta línea si no está:

python

CopiarEditar

import os

Esto sirve para que Django pueda **trabajar con rutas del sistema operativo** (como carpetas y archivos).

**📍 2. Configurar las plantillas (HTML)**

Buscá esta parte en TEMPLATES (por la línea 55 aprox):

python

CopiarEditar

'DIRS': [],

Cambiála por:

python

CopiarEditar

'DIRS': [os.path.join(BASE\_DIR, 'templates')],

📌 Así Django va a buscar los archivos .html que uses dentro de la carpeta templates.

**📍 3. Configurar los archivos estáticos (CSS, JS, íconos…)**

Agregá esto **abajo de la variable STATIC\_URL** (ya está definida en el archivo):

python

CopiarEditar

STATICFILES\_DIRS = [

os.path.join(BASE\_DIR, 'static'),

]

**📍 4. Configurar los archivos media (como imágenes subidas)**

Agregá al final del archivo:

python

CopiarEditar

MEDIA\_URL = '/media/'

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

**⚙️ Paso en urls.py del proyecto (mercadoweb/urls.py):**

Django no sirve archivos media automáticamente en desarrollo, así que hay que agregar from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

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

path('', include('apps.producto.urls')), # Aquí conectamos las URLs de la app 'producto'

]

# Esta línea sirve para que en modo desarrollo se puedan cargar los archivos media (imágenes, etc.)

urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

Pasos a seguir:

Copiá todo este código y reemplazá el contenido de tu archivo mercadoweb/urls.py.

Asegurate que la carpeta apps/producto/ tenga su propio archivo urls.py con rutas definidas.

* **Creamos un archivo urls.py dentro de apps/producto/** si no existe todavía:

python

CopiarEditar

# apps/producto/urls.py

from django.urls import path

from . import views

urlpatterns = [

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

]

Y en views.py algo simple para probar:

from django.http import HttpResponse

def index(request):

return HttpResponse("¡Hola desde Producto!")

**✅ Si todo eso está listo…**

Ya podés levantar el servidor:

python manage.py runserver

Y entrar en el navegador a:

**✅ ORDEN RECOMENDADO PARA CREAR UNA APP EN DJANGO**

* **Crear la app:**

python manage.py startapp producto

* **Registrar la app en settings.py:**  
  Dentro de INSTALLED\_APPS, agregar:

'apps.producto',

* **Crear las vistas (views.py) de la app:**  
  Definís las funciones o clases que van a responder a cada URL.

from django.http import HttpResponse

def producto(request):

return HttpResponse("¡Hola desde Producto!")

para usar el render llamar a las paginas,

def producto(request):

return render(request, ‘producto.html’)

ese product lo llamo desde templates…

* **Crear el archivo urls.py dentro de la app (si no existe):**  
  Y ahí conectás las URLs con las vistas:

from django.urls import path

from . import views

urlpatterns = [

path('producto/', views.producto, name='producto'),

]

💡 Aclaración: Este archivo usa views, pero **no importa que lo crees antes o después de views.py**, siempre que cuando ejecutes el servidor todo ya esté en su lugar. Por eso no hay problema si lo escribís aunque aún no exista la vista — simplemente dará error al ejecutar, no al escribir.

* **Conectar las URLs de la app al archivo principal (mercadoweb/urls.py):**

from django.urls import path, include

urlpatterns = [

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

path('producto/', include('apps.producto.urls')),

]

🧭 ÍNDICE Y RESUMEN DE LO QUE VIMOS EN DJANGO

1. 🔧 Configuración inicial del proyecto

django-admin startproject mercadoweb

Crear entorno virtual: python -m venv env

Activar entorno: env\Scripts\activate

Instalar Django: pip install django

Crear apps: python manage.py startapp producto, cliente, etc.

Registrar las apps en settings.py > INSTALLED\_APPS

2. 📂 Estructura de carpetas del proyecto

apps/: Carpeta para guardar todas las apps

templates/: Carpeta global para guardar HTMLs

static/: Para archivos CSS, imágenes, JS

media/: Para subir imágenes desde admin

3. 🌐 URLconf y Vistas

urls.py general: incluye las URLs de cada app

path('producto/', include('apps.producto.urls'))

Vistas por función (views.py)

def producto(request):

return render(request, 'producto.html')

4. 📄 Templates (HTML) y Estilos (CSS)

Usamos render() para conectar vistas con templates

Creamos producto.html, producto\_limpieza.html, producto\_listar.html

Cargamos estilos:

html

{% load static %}

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

Fondo azul en CSS:

css

body {

background-color: blue;

}

5. 🗃️ MODELS – Creación de Base de Datos

Modelos:

class Categoria(models.Model):

nombre = models.CharField(max\_length=60)

class Tipo(models.Model):

nombre = models.CharField(max\_length=40)

class Producto(models.Model):

nombre = models.CharField(max\_length=80)

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

stock = models.IntegerField()

categoria = models.ForeignKey(Categoria, on\_delete=models.PROTECT)

tipo = models.ForeignKey(Tipo, on\_delete=models.CASCADE, null=True)

Migraciones:

python manage.py makemigrations

python manage.py migrate

6. 🧑‍💼 Panel de Administración

Crear superusuario: python manage.py createsuperuser

Ver y editar productos, tipos y categorías desde /admin/

from django.contrib import admin

from .models import Producto, Categoria, Tipo

class ProductoAdmin(admin.ModelAdmin):

list\_display=('nombre','tipo','precio','stock')

list\_filter=('tipo','categoria')

search\_fields=('tipo',)

admin.site.register(Producto,ProductoAdmin)

admin.site.register(Categoria) #nos da la posibilidad de registrar el campo y poder editarlo

admin.site.register(Tipo)

Filtrar por campos (list\_display, list\_filter)

Configuraciones comunes en ModelAdmin:

Atributo Descripción Ejemplo

list\_display Campos que se muestran como columnas en la lista principal del admin. ('nombre', 'precio', 'tipo')

list\_filter Campos que se usan para filtrar los registros a la izquierda (filtros rápidos). ('tipo', 'fecha\_creacion')

search\_fields Campos en los que se puede buscar texto con la barra de búsqueda. ('nombre', 'descripcion', 'tipo\_\_nombre')

ordering Orden por defecto de los registros en la lista (lista de campos). ('nombre', '-precio')

list\_per\_page Cantidad de registros que se muestran por página. 25

readonly\_fields Campos que se muestran pero no se pueden editar en el formulario de admin. ('fecha\_creacion',)

fields Campos que se muestran y en qué orden en el formulario de edición. ('nombre', 'tipo', 'precio')

exclude Campos que se excluyen del formulario de edición. ('slug',)

prepopulated\_fields Campos que se autocompletan basados en otros (usualmente para slug). {'slug': ('nombre',)}

raw\_id\_fields Muestra un input en lugar de un dropdown para relaciones ForeignKey cuando hay muchos registros. ('tipo',)

filter\_horizontal Muestra filtros con selector horizontal para relaciones ManyToMany. ('categorias',)

filter\_vertical Variante de filter\_horizontal, pero con selector vertical. ('categorias',)

date\_hierarchy Añade una barra de navegación por fechas encima de la lista (para campos DateField o DateTimeField). 'fecha\_creacion'

save\_on\_top Muestra los botones guardar/guardar y continuar también en la parte superior del formulario.

7. 🟢 ORM

🔸 1. CREAR (Create)

Crear objetos y guardarlos en la base:

python

Copiar

Editar

from producto.models import Producto

# Crear una instancia y guardarla

producto = Producto(nombre="Arroz", precio=1500.00, tipo="Alimento", stock=20)

producto.save()

# Otra forma, más directa

Producto.objects.create(nombre="Fideos", precio=1200.00, tipo="Alimento", stock=30)

🔸 2. LEER (Read)

Leer todos los objetos:

python

Copiar

Editar

productos = Producto.objects.all()

Leer un producto por ID:

python

Copiar

Editar

producto = Producto.objects.get(id=1)

Filtrar:

python

Copiar

Editar

productos = Producto.objects.filter(tipo="Alimento")

Búsqueda parcial (como LIKE):

python

Copiar

Editar

productos = Producto.objects.filter(nombre\_\_icontains="arroz")

Ordenar resultados:

python

Copiar

Editar

productos = Producto.objects.order\_by('precio') # Ascendente

productos = Producto.objects.order\_by('-precio') # Descendente

🔸 3. ACTUALIZAR (Update)

python

Copiar

Editar

producto = Producto.objects.get(id=1)

producto.precio = 1800.00

producto.stock = 25

producto.save()

También se puede usar update() con filter:

python

Copiar

Editar

Producto.objects.filter(nombre="Fideos").update(stock=50)

🔸 4. ELIMINAR (Delete)

python

Copiar

Editar

producto = Producto.objects.get(id=1)

producto.delete()

También se puede eliminar varios:

python

Copiar

Editar

Producto.objects.filter(tipo="Alimento").delete()

8. ⚙️ 🔹 PASO 1: ¿Qué es una solicitud HTTP?

Cuando un usuario entra a una página web, su navegador se comunica con el servidor enviando una solicitud HTTP.

Los 4 métodos más comunes son:

Método ¿Qué hace? Ejemplo práctico

GET Pide datos al servidor Ver una página, mostrar un formulario vacío

POST Envía datos al servidor Enviar un formulario (por ejemplo, agregar un producto)

PUT Actualiza datos existentes Modificar un producto (menos usado en formularios HTML comunes)

DELETE Borra datos Eliminar un producto (se usa más en APIs)

🟩 PASO 1: Modelo de Producto (si no lo tenés)

En producto/models.py, definí un modelo básico:

from django.db import models

class Producto(models.Model):

Después de esto, no te olvides de hacer:

python manage.py makemigrations

python manage.py migrate

🟩 PASO 2: Crear algunos productos desde el admin o shell

🟩 PASO 3: Crear la vista GET

En producto/views.py:

1ro consulta

2do contexto

3ro return

from django.shortcuts import render #esta por defecto

from .models import Producto

def lista\_productos(request):

productos = Producto.objects.select\_related('categoria', 'tipo').all()

return render(request, 'producto/lista.html', {'productos': productos})

Usamos select\_related para que no haga una consulta a la base por cada producto relacionado.

🟩 PASO 4: Crear la plantilla HTML

En producto/templates/producto/lista.html:

<!DOCTYPE html>

<html>

<head>

<title>Lista de Productos</title>

</head>

<body>

<h1>Lista de Productos</h1>

<table border="1">

<thead>

<tr>

<th>Nombre</th>

<th>Precio</th>

<th>Stock</th>

<th>Categoría</th>

<th>Tipo</th>

</tr>

</thead>

<tbody>

{% for producto in productos %}

<tr>

<td>{{ producto.nombre }}</td>

<td>${{ producto.precio }}</td>

<td>{{ producto.stock }}</td>

<td>{{ producto.categoria.nombre }}</td>

<td>{{ producto.tipo.nombre }}</td>

</tr>

{% endfor %}

</tbody>

</table>

</body>

</html>

🟩 PASO 5: Conectar en la URL

En producto/urls.py (crealo si no existe):

from django.urls import path

from . import views

urlpatterns = [

path('productos/', views.lista\_productos, name='lista\_productos'),

]

Y en mercadoweb/urls.py (tu proyecto principal), asegurate de incluirlo:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

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

path('', include('producto.urls')),

]si todavia no esta agregada su apps

🔁 Orden típico del CRUD en Django con el ORM y vistas

✅ GET: Listar productos → ya lo hiciste

🔜 GET: Ver detalle de un producto (por ID)

➕ POST: Crear producto con formulario

📝 POST: Editar producto

❌ POST: Eliminar producto

✅ Paso siguiente: Detalle de producto

Vamos a hacer que cuando toques en un producto desde la lista, te lleve a una página con su información completa.

1. Vista (views.py)

Agregá esta función:

def producto\_detalle(request, producto\_id):

producto = Producto.objects.get(id=producto\_id)

contexto={'producto':producto}

return render(request, 'producto\_detalle.html', contexto)

2. Ruta (urls.py)

Agregá esto en apps/producto/urls.py:

path('detalle/<int:producto\_id>/', producto\_detalle, name='producto\_detalle'),

3. Template (producto\_detalle.html)

Crea el archivo en tu carpeta templates:

<h2>Detalle del producto</h2>

<p><strong>Nombre:</strong> {{ producto.nombre }}</p>

<p><strong>Precio:</strong> {{ producto.precio }}</p>

<p><strong>Stock:</strong> {{ producto.stock }}</p>

<p><strong>Tipo:</strong> {{ producto.tipo }}</p>

<p><strong>Categoría:</strong> {{ producto.categoria }}</p>

<a href="{% url 'apps.producto\_lista' %}">Volver al listado</a>

4. Agregá el enlace desde la lista

En tu producto\_listar.html:

{% for p in productos %}

<li>

<a href="{% url 'producto\_detalle' p.id %}">{{ p.nombre }}</a> - ${{ p.precio }}

</li>

{% endfor %}