Universidad de los Andes | Educación Continua
Vicerrectoría Académica

Introducción al Backend y Arquitectura Cliente-Servidor

Curso: Desarrollo Backend con Django

Framework Django: Introducción e Instalación

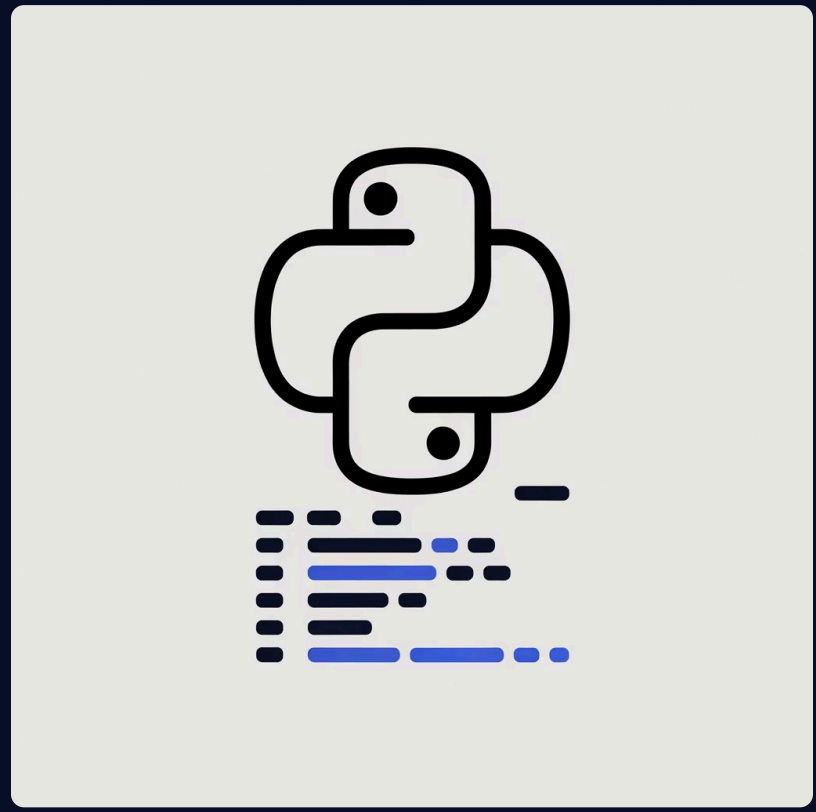
¿Qué veremos hoy?

1. **Introducción a Django**
   Conceptos fundamentales y ventajas del framework
2. **Arquitectura MTV**
   Entendiendo el patrón Model-Template-View
3. **Requisitos previos**
   Preparando nuestro entorno de desarrollo
4. **Creación de proyecto**
   Primeros pasos con Django
5. **Ejemplo práctico**
   Construyendo una biblioteca virtual

Django: El framework web para perfeccionistas con plazos

Django es un framework de desarrollo web de alto nivel escrito en Python que fomenta el desarrollo rápido y el diseño limpio y pragmático.

Creado en 2003, debe su nombre al guitarrista de jazz Django Reinhardt y se ha convertido en uno de los frameworks más populares para desarrollar aplicaciones web robustas.



Django nació en un entorno periodístico, donde los plazos son críticos y la eficiencia es esencial. Hoy es mantenido por la Django Software Foundation y utilizado por empresas como Instagram, Pinterest y Mozilla.

¿Por qué Django es tan popular?

Comunidad activa

Miles de desarrolladores en todo el mundo contribuyen constantemente, creando paquetes, resolviendo bugs y compartiendo conocimiento.

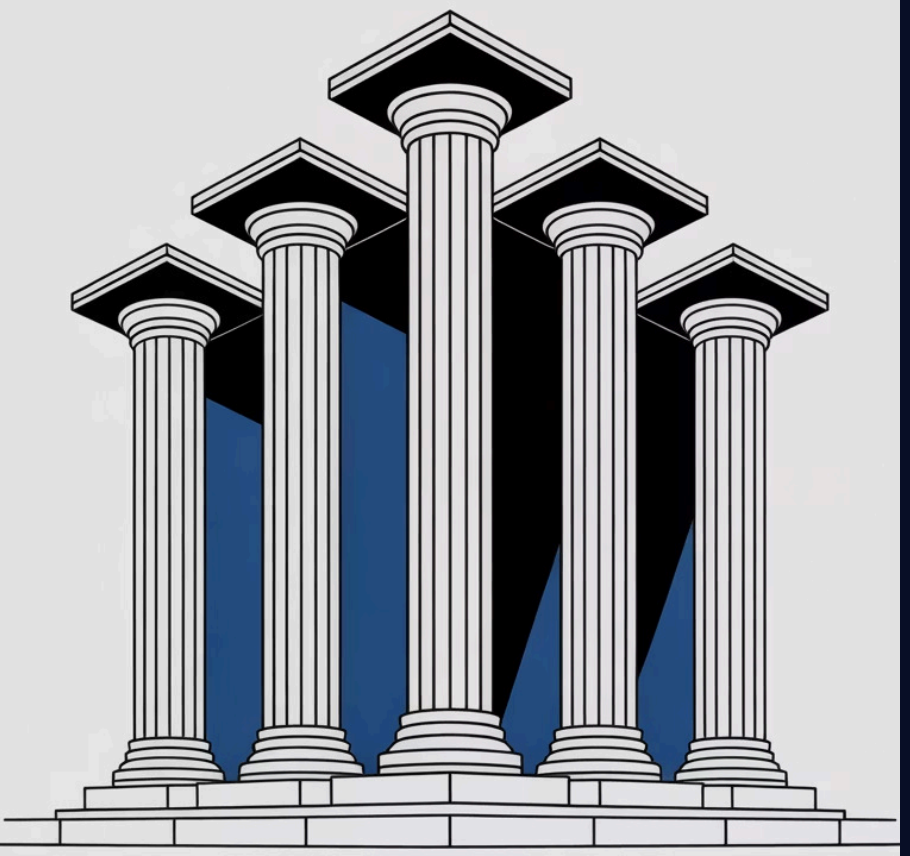
Documentación extensa

Una de las mejores documentaciones disponibles, con tutoriales detallados, guías y referencias técnicas completas.

Casos de éxito

Adoptado por startups y gigantes tecnológicos como Instagram, Spotify, Dropbox y Pinterest para manejar millones de usuarios.

Django ha demostrado ser una opción confiable tanto para pequeños proyectos como para aplicaciones a escala global.



Principios clave de Django

1

"Baterías incluidas"

Viene con todo lo necesario para construir aplicaciones web completas sin depender de bibliotecas externas.

2

Seguridad

Protección integrada contra vulnerabilidades comunes como XSS, CSRF, SQL injection y clickjacking.

3

Escalabilidad

Diseñado para crecer con tu proyecto, desde un prototipo hasta una aplicación con millones de usuarios.

4

Desarrollo rápido

Reduce significativamente el tiempo de desarrollo con componentes reutilizables y bien integrados.

Python puro vs. Django

Python puro

- Libertad total de implementación
- Requiere configurar todo desde cero
- Mayor curva de aprendizaje para proyectos complejos
- Más propenso a errores de seguridad si no se implementan correctamente las protecciones

Django

- Estructura predefinida y organizada
- Componentes listos para usar (ORM, admin, auth)
- Seguridad integrada por defecto
- Desarrollo más rápido para aplicaciones web estándar

La elección dependerá de la complejidad del proyecto, el tiempo disponible y las necesidades específicas.

¿Cuándo usar Django?

Aplicaciones con modelos de datos complejos

El ORM de Django simplifica la interacción con bases de datos relacionales.

Cuando la seguridad es prioritaria

Protecciones integradas contra vulnerabilidades web comunes.

Proyectos con plazos ajustados

Acelera el desarrollo con componentes prediseñados y admin incorporado.

Aplicaciones con gestión de usuarios

Sistema de autenticación y autorización listo para usar.

Arquitectura de Django: MTV

Django utiliza una variante del patrón MVC (Modelo-Vista-Controlador) llamada MTV:

Model (Modelo)

Define la estructura de datos, proporciona mecanismos para acceder y manipular la información en la base de datos.

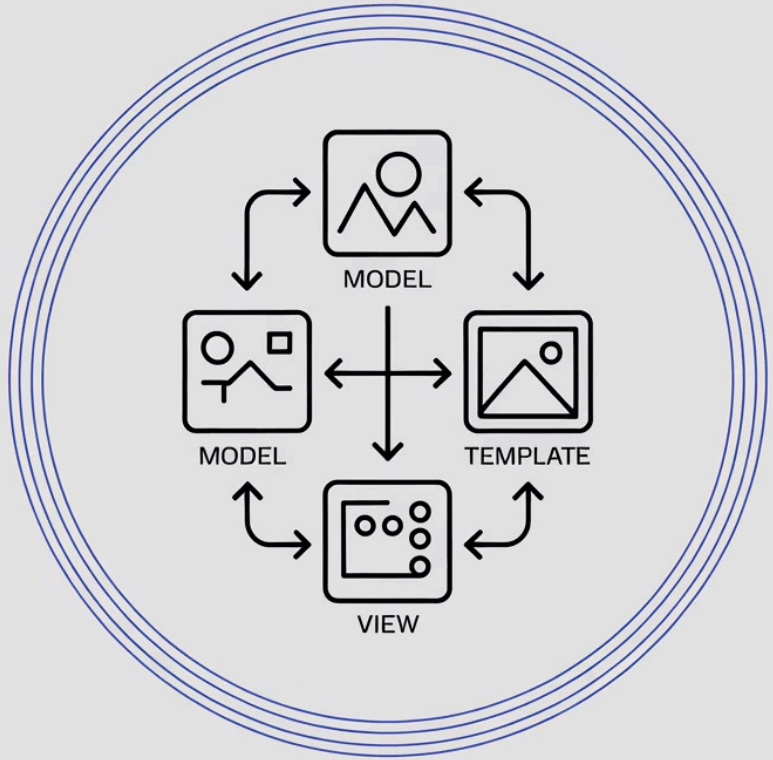
Template (Plantilla)

Define cómo se presentan los datos al usuario final, utilizando HTML con una sintaxis especial de Django.

View (Vista)

Contiene la lógica de negocio y actúa como puente entre el modelo y la plantilla.

Django MTV Architecture



MTV vs MVC: ¿Cuál es la diferencia?

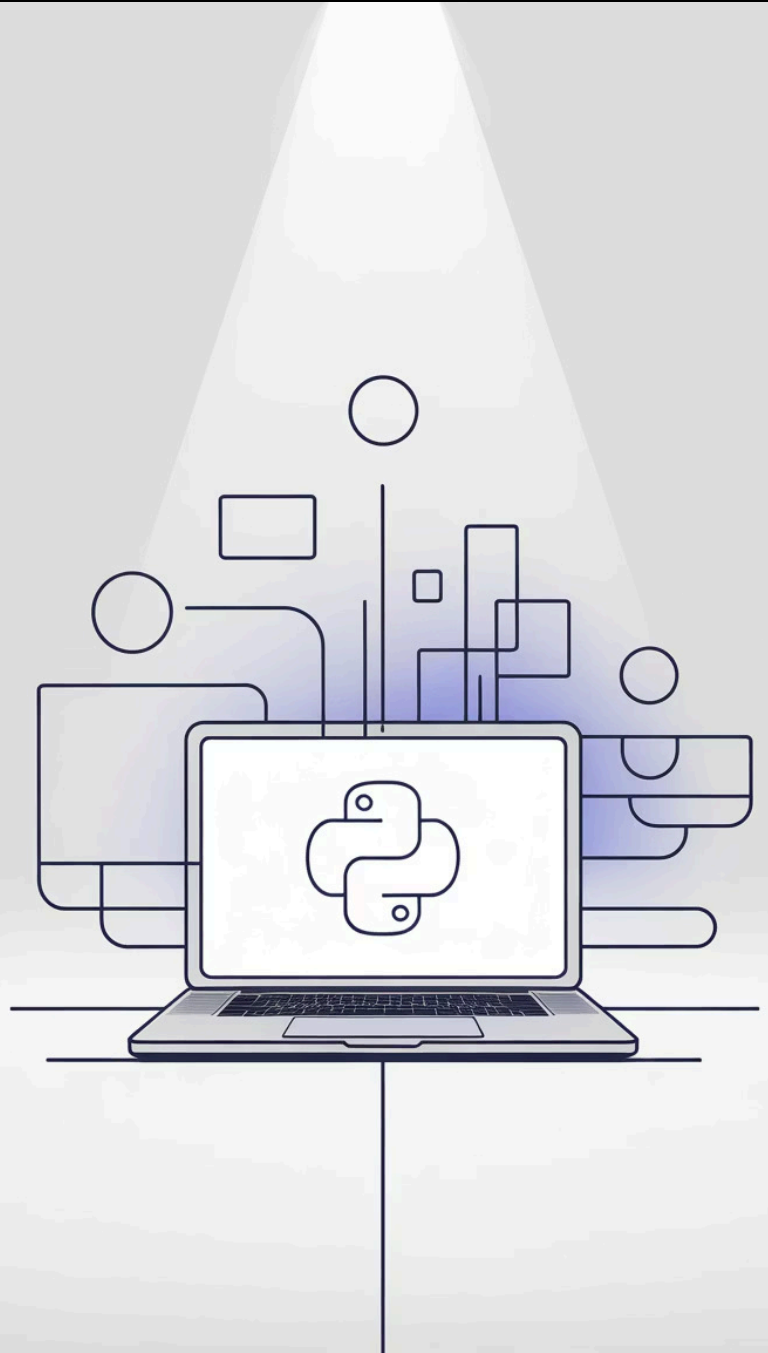
Patrón MVC

- **Modelo:** Gestiona los datos y la lógica de negocio
- **Vista:** Presenta la información al usuario
- **Controlador:** Maneja la entrada del usuario y coordina el modelo y la vista

Patrón MTV (Django)

- **Modelo:** Similar al modelo en MVC
- **Plantilla (Template):** Equivalente a la vista en MVC
- **Vista (View):** Similar al controlador en MVC

En Django, el framework mismo actúa como el controlador, manejando la correspondencia entre URL y vistas, mientras que las vistas de Django son más parecidas a los controladores en MVC tradicional.



Requisitos previos para instalar Django

Antes de comenzar con Django, necesitamos asegurarnos de tener configurado correctamente nuestro entorno de desarrollo:

1. **Python instalado**
   Django 4.2 requiere Python 3.8 o superior. Recomendamos usar la última versión estable.

2. **Conocimientos básicos de Python**
   Entender conceptos como funciones, clases, decoradores y manejo de excepciones.

3. **Conceptos básicos de HTML/CSS**
   Para trabajar efectivamente con las plantillas de Django.

4. **Editor de código o IDE**
   VS Code, PyCharm o cualquier otro editor con soporte para Python.

Instalando Django

Con el entorno virtual activado, instalar Django es tan simple como usar pip, el gestor de paquetes de Python:

```
pip install django
```

Esto instalará la última versión estable de Django.

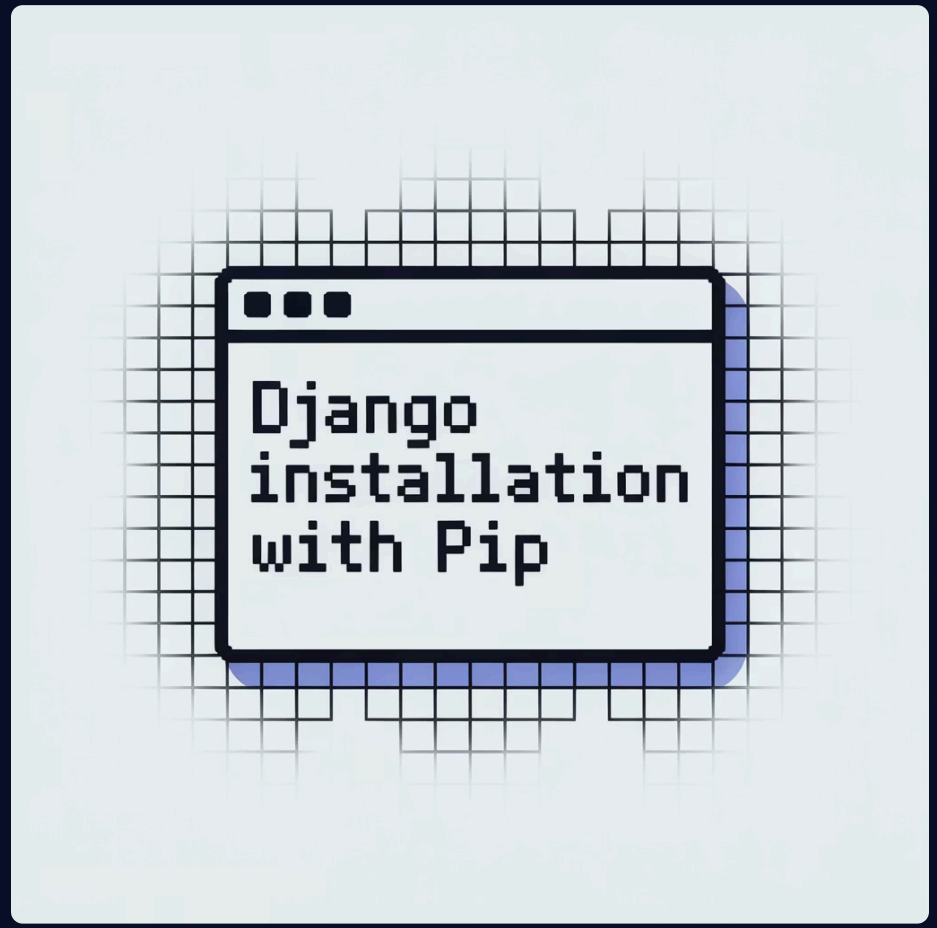
Si necesitas una versión específica:

```
pip install django==4.2.3
```

Para verificar la instalación:

```
python -m django --version
```

Deberías ver algo como: `4.2.3`



Registro de dependencias

Una buena práctica es mantener un registro de todas las dependencias de tu proyecto para facilitar su reproducción en otros entornos:

Generar requirements.txt

```
pip freeze > requirements.txt
```

Este comando guarda una lista de todos los paquetes instalados y sus versiones exactas.

Instalar desde requirements.txt

```
pip install -r requirements.txt
```

Este comando instala todos los paquetes listados con las versiones específicas indicadas.

El archivo `requirements.txt` debe incluirse en el control de versiones (como Git) para que otros desarrolladores puedan replicar exactamente tu entorno.

¡Ahora vamos a crear nuestro primer proyecto Django!

Creando un proyecto Django

Django incluye una utilidad de línea de comandos llamada `django-admin` que nos permite crear la estructura inicial de un proyecto:

```
django-admin startproject biblioteca_virtual
```

Este comando crea un directorio con el nombre del proyecto que contiene la estructura básica:

```
biblioteca_virtual/
  |
  ├──── biblioteca_virtual/    # Directorio del proyecto
  |     ├──── __init__.py
  |     ├──── asgi.py
  |     ├──── settings.py       # Configuración del proyecto
  |     ├──── urls.py           # Mapeo de URLs
  |     └──── wsgi.py
  |
  └──── manage.py               # Utilidad de administración
```

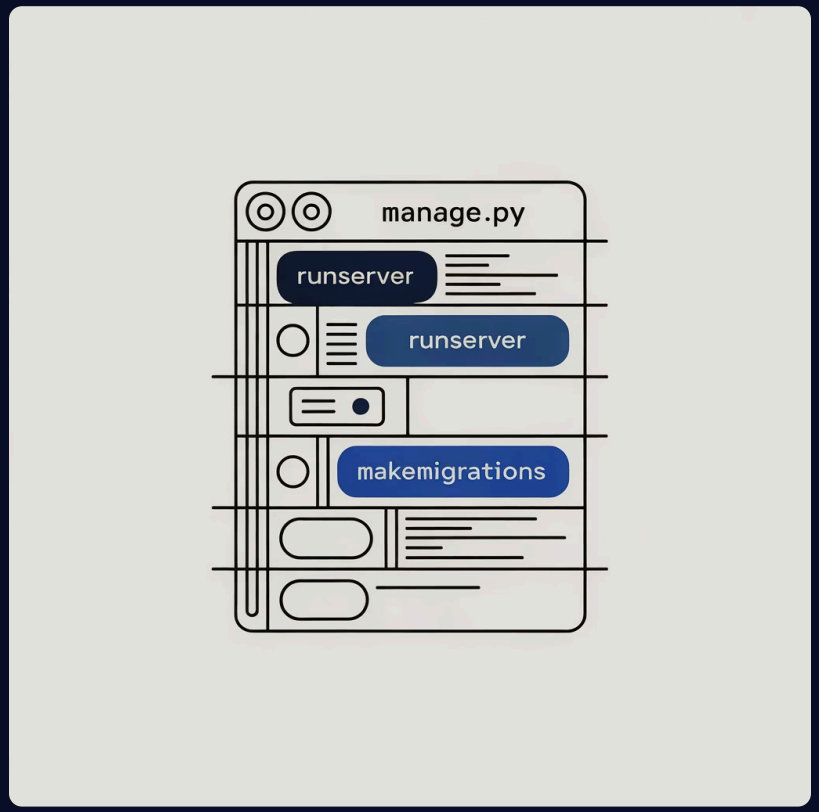
El comando `startproject` genera automáticamente todos estos archivos con configuraciones predeterminadas.

El archivo manage.py

`manage.py` es una utilidad de línea de comandos que te permite interactuar con tu proyecto Django de varias formas:

- Iniciar el servidor de desarrollo
- Crear aplicaciones
- Trabajar con la base de datos
- Ejecutar pruebas
- Y muchas otras tareas administrativas

Es un script que funciona como un contenedor para `django-admin`, pero configura automáticamente el entorno para trabajar con tu proyecto específico.



settings.py: El centro de configuración

Configuraciones clave

- `DEBUG`: Modo de depuración
- `INSTALLED_APPS`: Aplicaciones activadas
- `DATABASES`: Configuración de BD
- `STATIC_URL`: URL para archivos estáticos
- `MIDDLEWARE`: Componentes intermedios

Consejos importantes

- Nunca uses `DEBUG = True` en producción
- Protege la `SECRET_KEY`
- Usa diferentes configuraciones para desarrollo y producción
- Guarda información sensible en variables de entorno

settings.py controla prácticamente todos los aspectos del comportamiento de tu aplicación Django.

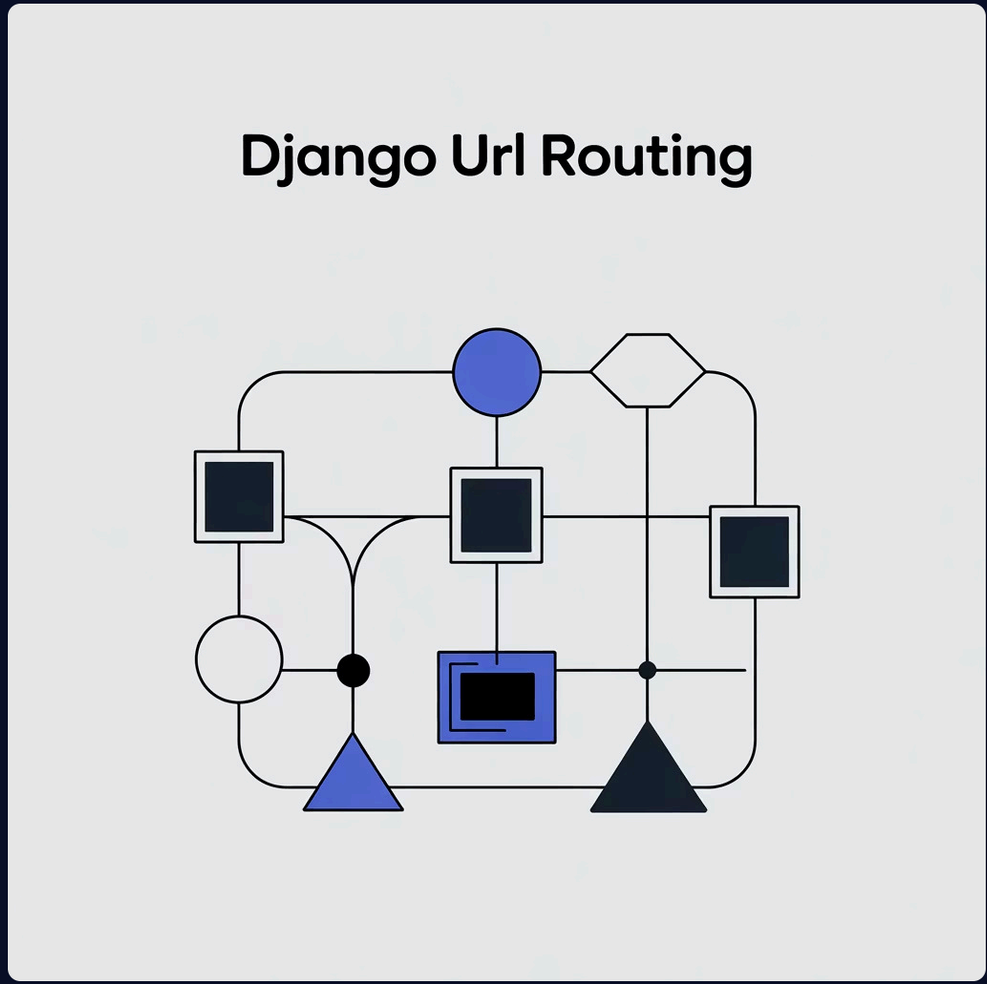
urls.py: El mapa de navegación

El archivo `urls.py` define cómo se mapean las URLs a las vistas de tu aplicación:

```
from django.contrib import admin
from django.urls import path, include

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

Este sistema permite organizar las URLs de forma modular y jerárquica.



El URLconf es fundamental para el funcionamiento de Django, ya que determina qué vista se ejecutará para cada URL solicitada.

wsgi.py y asgi.py: Interfaces para servidores

WSGI (Web Server Gateway Interface)

Estándar tradicional para comunicación entre servidores web y aplicaciones Python.

Utilizado por servidores como:

- Gunicorn
- uWSGI
- mod_wsgi (Apache)

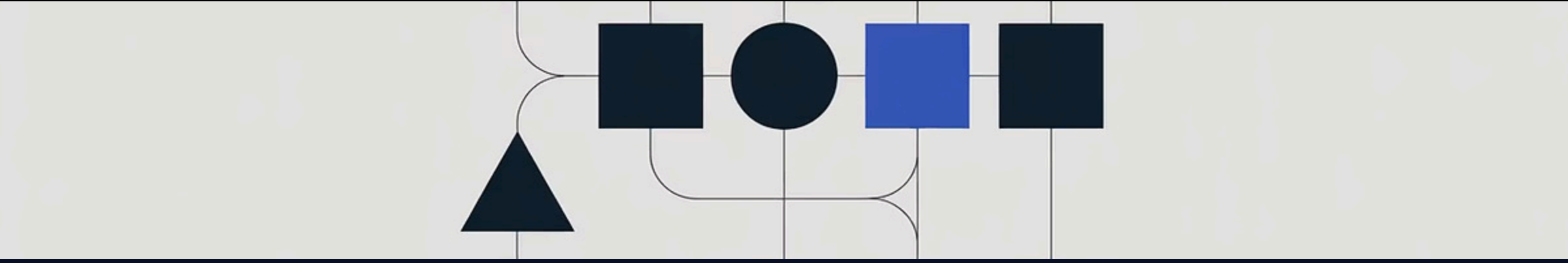
ASGI (Asynchronous Server Gateway Interface)

Versión asíncrona y más moderna que permite WebSockets y HTTP/2.

Utilizado por servidores como:

- Daphne
- Uvicorn
- Hypercorn

Estos archivos proporcionan puntos de entrada para que los servidores web se comuniquen con tu aplicación Django en producción.



Estructura de archivos de un proyecto Django

Entender la estructura de archivos de Django es esencial para trabajar eficientemente:

Nivel proyecto

Contiene la configuración global, URLs principales y archivos para despliegue.

Nivel aplicación

Contiene modelos, vistas, URLs y plantillas específicas de cada componente funcional.

Nivel template

Archivos HTML con sintaxis especial de Django para renderizar datos dinámicamente.

Nivel static

Archivos CSS, JavaScript e imágenes para el frontend.

Iniciando el servidor de desarrollo

Django incluye un servidor web ligero para desarrollo que facilita probar la aplicación localmente:

```
cd biblioteca_virtual
python manage.py runserver
```

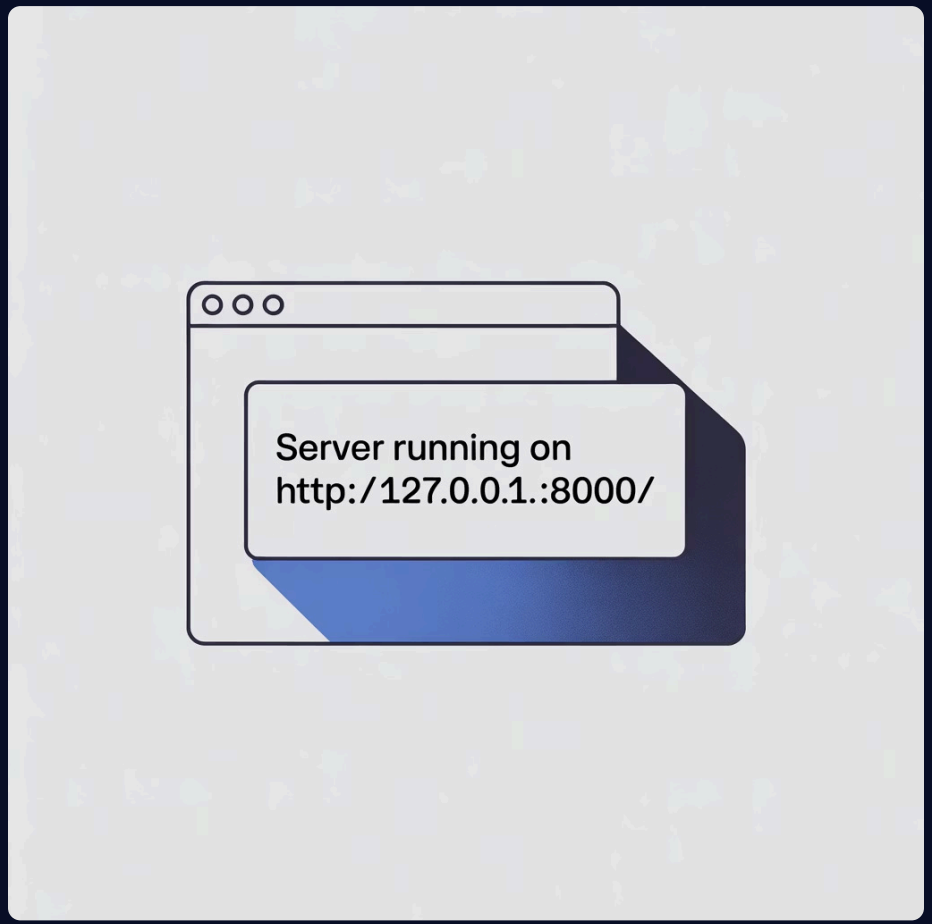
Por defecto, el servidor se inicia en `http://127.0.0.1:8000/`

Para cambiar el puerto:

```
python manage.py runserver 8080
```

Para permitir conexiones desde otras máquinas:

```
python manage.py runserver 0.0.0.0:8000
```



The install worked successfully! Congratulations!

View release notes for Django 5.2

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

django

¡Bienvenido a Django!

Si ves esta página, ¡felicidades! Has configurado correctamente tu proyecto Django y está funcionando.

Esta página de bienvenida solo aparece cuando:

- El modo DEBUG está activado (DEBUG = True en settings.py)
- No has definido ninguna vista para la URL raíz ('/')

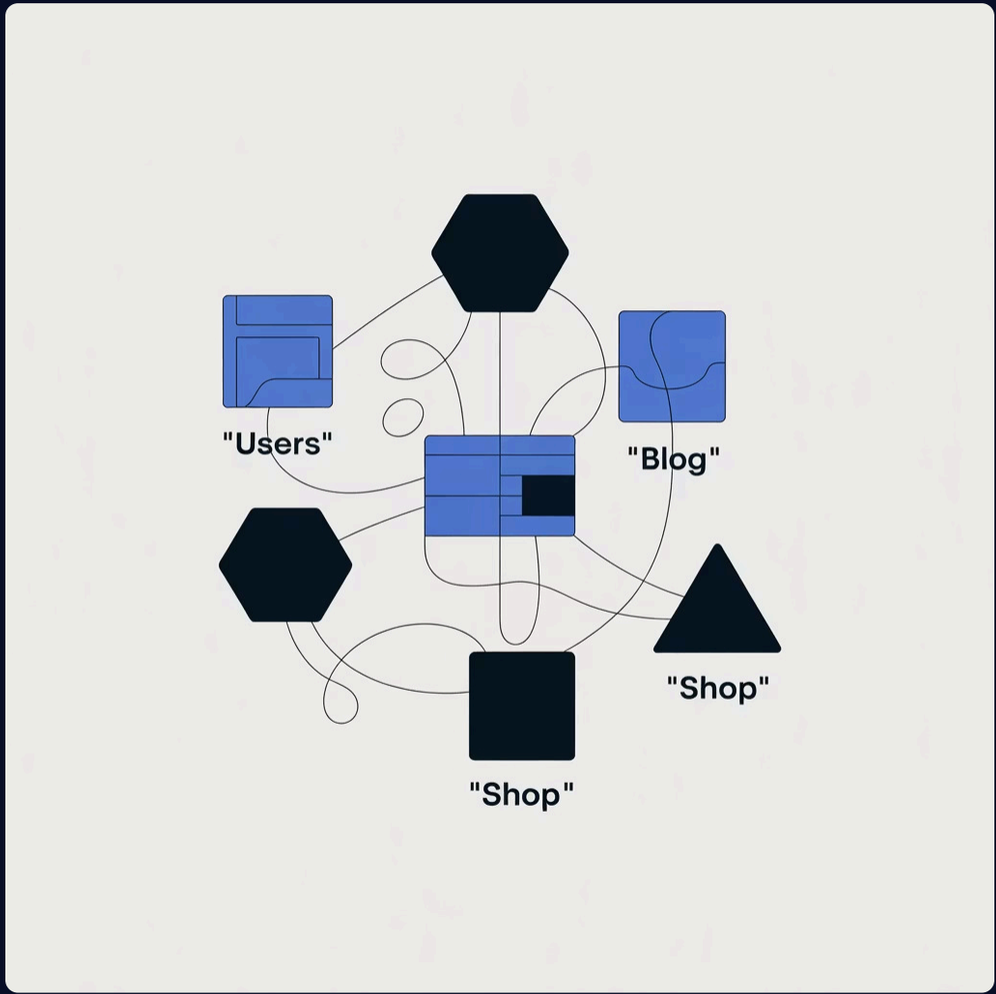
El servidor de desarrollo se recarga automáticamente cuando detecta cambios en el código, lo que acelera el proceso de desarrollo.

Aplicaciones en Django

En Django, una aplicación es un componente funcional que realiza una tarea específica:

- Modulariza el código en componentes reutilizables
- Puede ser instalada en múltiples proyectos
- Tiene su propia estructura de archivos

Un proyecto Django es una colección de aplicaciones configuradas para trabajar juntas.



Ejemplos de aplicaciones en un sistema de biblioteca: 'libros', 'usuarios', 'préstamos', 'catálogo'.

Creando una aplicación

```
python manage.py startapp libros
```

Este comando crea un directorio con la estructura básica de una aplicación:

```
libros/
├── __init__.py
├── admin.py        # Configuración del panel de administración
├── apps.py         # Configuración de la aplicación
├── migrations/     # Cambios en la base de datos
│   └── __init__.py
├── models.py       # Definición de modelos de datos
├── tests.py        # Pruebas automatizadas
└── views.py        # Lógica de la aplicación
```

Después de crear una aplicación, debes registrarla en `INSTALLED_APPS` en el archivo `settings.py` para que Django la reconozca.

Ejemplo práctico: Biblioteca Virtual

Vamos a crear un proyecto de biblioteca virtual siguiendo buenas prácticas:

Preparación

Creamos y activamos un entorno virtual, luego instalamos Django.

Creación del proyecto

Ejecutamos `django-admin startproject biblioteca_virtual`.

Servidor de desarrollo

Iniciamos el servidor con `python manage.py runserver`.

Verificación

Comprobamos la página inicial de Django en el navegador.

Explorar settings.py y urls.py

Ajustes importantes en settings.py

```
# Zona horaria para Colombia
TIME_ZONE = 'America/Bogota'

# Idioma predeterminado
LANGUAGE_CODE = 'es-co'

# Aplicaciones instaladas
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    # ... otras apps ...
    'libros.apps.LibrosConfig',  # Nuestra app
]
```

Configuración básica de urls.py

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

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

Creando una vista básica

En `libros/views.py`:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse(
        "¡Bienvenido a la Biblioteca Virtual!"
    )

def lista_libros(request):
    return HttpResponse(
        "Aquí se mostrarán todos los libros."
    )
```

Estas vistas son simples, pero ilustran el concepto básico de cómo Django procesa las solicitudes y devuelve respuestas.

¿Y ahora qué?

Crear modelos

Definir la estructura de datos en models.py y aplicar migraciones.

Personalizar admin

Configurar el panel de administración para gestionar datos.

Desarrollar plantillas

Crear archivos HTML con la sintaxis de plantillas de Django.

Expandir vistas

Implementar la lógica de la aplicación en views.py.

Configurar URLs

Definir patrones de URL para acceder a las vistas.

Este ciclo de desarrollo es iterativo y se repite a medida que añades funcionalidades a tu aplicación.

Pregunta de reflexión

> **¿Qué ventajas ofrece iniciar un proyecto con Django frente a hacerlo desde cero con Python y librerías externas?**

Productividad

Django proporciona una estructura organizada y componentes predefinidos que aceleran significativamente el desarrollo.

Seguridad

Incluye protecciones contra vulnerabilidades comunes que tendrías que implementar manualmente.

Comunidad y soporte

Extensa documentación, paquetes de terceros y una comunidad activa para resolver problemas.

¡Gracias por tu atención!

¡No olvides activar tu entorno virtual antes de comenzar a programar!

Universidad de los Andes | Educación Continua
Vicerrectoría Académica

#EDCOUNIANDES

https://educacioncontinua.uniandes.edu.co/

Contacto: educacion.continua@uniandes.edu.co

Universidad de los Andes | Vigilada Mineducación. Reconocimiento como Universidad: Decreto 1297 del 30 de mayo de 1964. Reconocimiento personería jurídica: Resolución 28 del 23 de febrero de 1949 MinJusticia.