04

INTRODUCCIÓN A DJANGO

¿Qué es Django?

Django es un framework de desarrollo web de alto nivel, escrito en Python, que facilita la creación de aplicaciones web de forma rápida y con un código limpio y eficiente. Fue diseñado con un enfoque en la simplicidad, la reutilización de componentes y la rápida iteración de desarrollo.

- Desarrollo de sitios web dinámicos
- Administración de contenido
- Aplicaciones e-commerce
- Desarrollo de APIs
- Aplicaciones de redes sociales
- Plataformas educativas
- Aplicaciones empresariales
- Prototipos y MVPs (productos mínimos viables)

Principales características y ventajas

- **Administración automática:** Django genera automáticamente una interfaz de administración para gestionar los modelos de la base de datos, lo que ahorra tiempo y esfuerzo en tareas comunes.

- **ORM (Object-Relational Mapping):** Django incluye un ORM que permite interactuar con bases de datos usando objetos Python en lugar de escribir SQL, facilitando el manejo de datos.

- **Seguridad:** Django incluye medidas de seguridad integradas para proteger aplicaciones web contra amenazas comunes, como inyección de SQL, cross-site scripting (XSS), y cross-site request forgery (CSRF).

Principales características y ventajas

- **Escalabilidad y modularidad:** Permite desarrollar aplicaciones modulares que pueden escalar fácilmente, adaptándose a proyectos de diferentes tamaños y complejidades.

- **Comunidad activa:** Django tiene una comunidad grande y activa, lo que significa que hay abundante documentación, tutoriales y paquetes reutilizables que aceleran el desarrollo.

Desventajas del uso de Django

Aunque Django es un framework muy poderoso, también presenta algunas desventajas o limitaciones que es importante considerar, dependiendo del tipo de proyecto:

Monolítico:

- Django sigue una arquitectura monolítica, lo que significa que todo está estrechamente integrado. Si bien esto facilita el desarrollo rápido, puede dificultar la flexibilidad si se desean usar solo partes específicas del framework o si se quiere migrar a una arquitectura de microservicios.

Curva de aprendizaje:

- Aunque Django ofrece muchas herramientas y automatización, para los principiantes puede ser abrumador aprender y manejar todas sus características, especialmente si no tienen experiencia previa con frameworks web o con Python.

Desventajas del uso de Django

Poca flexibilidad con bases de datos no relacionales:

- Django está diseñado principalmente para trabajar con bases de datos relacionales (como PostgreSQL o MySQL) a través de su ORM (Object-Relational Mapping). Aunque existen adaptaciones para bases de datos NoSQL, no es su enfoque principal y no tiene el mismo soporte que para las bases de datos relacionales.

Rendimiento:

- Para aplicaciones de gran escala con requisitos de alto rendimiento, Django puede no ser la opción más eficiente en comparación con otros frameworks más ligeros o específicos para microservicios. Puede requerir optimizaciones adicionales y un manejo cuidadoso de recursos para evitar problemas de rendimiento.

Desventajas del uso de Django

Componentes predefinidos:

- Django tiene una estructura rígida que puede imponer limitaciones en términos de personalización. Para algunos desarrolladores, esta estructura puede sentirse restrictiva, ya que los componentes de Django (como su ORM, formularios y plantillas) están diseñados de manera muy específica.

No es ideal para aplicaciones pequeñas o simples:

- Si bien Django es excelente para proyectos de gran escala y complejidad, puede ser excesivo para aplicaciones pequeñas o sencillas. Otros frameworks más ligeros como Flask en Python pueden ser más adecuados en esos casos, ya que requieren menos configuración y tienen menos sobrecarga.

Tamaño de la comunidad en comparación con otros frameworks:

- Aunque Django tiene una comunidad activa, esta no es tan grande como la de otros frameworks como React, Node.js o Laravel. Esto puede influir en la cantidad de recursos, plugins y soluciones disponibles para ciertos problemas específicos.

Estructura general de un proyecto Django

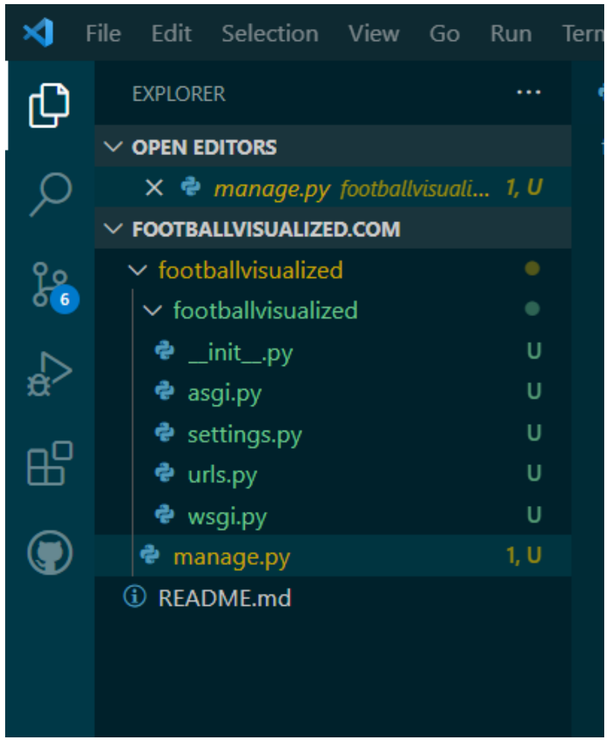
Cuando creas un proyecto con **django-admin startproject <nombre_proyecto>**, la estructura básica que obtienes es algo como esto:

```
mi_proyecto/
    manage.py
    mi_proyecto/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

Archivos y carpetas importantes

manage.py:

- Es un script que permite interactuar con el proyecto Django desde la línea de comandos. Usas este archivo para ejecutar comandos como iniciar el servidor (runserver), migrar la base de datos (migrate), crear aplicaciones (startapp), entre otros.



Archivos y carpetas importantes

Carpeta mi_proyecto/ (o el nombre de tu proyecto):

- Esta carpeta contiene la configuración principal del proyecto. Dentro de ella están varios archivos clave:
  - **__init__.py:** Indica que esta carpeta debe tratarse como un módulo de Python. Por lo general, está vacío.
  - **settings.py:** Contiene toda la configuración del proyecto, como las bases de datos, las aplicaciones instaladas, las configuraciones de seguridad, rutas de archivos estáticos, entre otros. Este es uno de los archivos más importantes.
  - **urls.py:** Aquí defines las rutas (URLs) que manejará tu proyecto. Es el "mapeo" entre las URL solicitadas por el navegador y las vistas (views) que deben procesarlas.
  - **wsgi.py y asgi.py:** Son archivos para la implementación del servidor web. wsgi.py es para el protocolo WSGI y asgi.py para ASGI. Son usados cuando el proyecto se despliega en un entorno de producción.

Estructura de una aplicación Django

Cuando se crea una aplicación con python **manage.py startapp <nombre_app>**, se genera una estructura de carpetas similar a esta:

```
mi_proyecto/
    mi_app/
        migrations/
            __init__.py
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
```

Archivos clave en una aplicación

migrations/:

- Esta carpeta contiene los archivos de migración que se generan cuando haces cambios en los modelos de la aplicación (por ejemplo, cuando añades o modificas campos en la base de datos). Las migraciones aseguran que la base de datos esté sincronizada con los modelos del proyecto.

__init__.py:

- Como en la carpeta del proyecto, este archivo indica que la carpeta de la aplicación es un módulo de Python.

admin.py:

- Archivo donde se deben registrar los modelos para que aparezcan en la interfaz de administración de Django.

Archivos clave en una aplicación

apps.py:

- Define la configuración de la aplicación. Por lo general, se utiliza para darle un nombre a la app y otras configuraciones específicas.

models.py:

- Aquí defines los modelos de la base de datos que se guardan por medio del método save(). Un modelo en Django es una representación de una tabla de base de datos y cada campo en el modelo corresponde a una columna en esa tabla.

tests.py:

- Archivo donde defines las pruebas unitarias (tests) de la aplicación para asegurarte de que todo funcione correctamente.

Archivos clave en una aplicación

views.py:

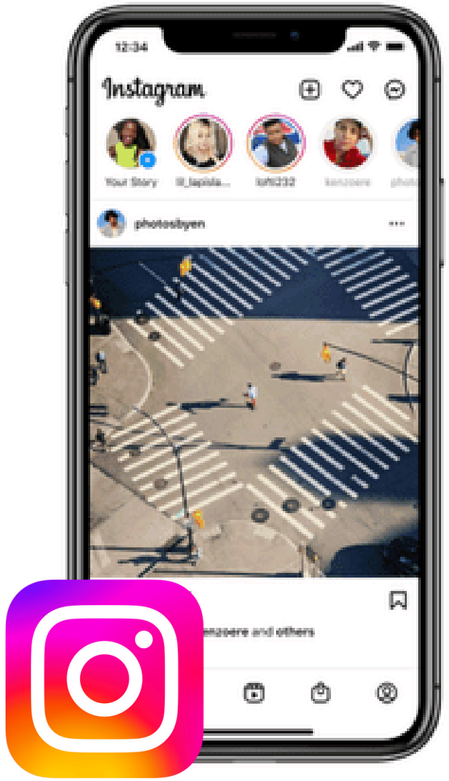
- Aquí defines las vistas (views), que son las funciones o clases que procesan las solicitudes HTTP y generan una respuesta. Es el lugar donde se maneja la lógica de negocio de la aplicación.

Resumen de la estructura

- **Proyecto Django:** Contiene archivos de configuración global como settings.py y urls.py.

- **Aplicaciones Django:** Son módulos independientes con su propia lógica, incluyendo modelos, vistas y migraciones.

Caso de aplicación

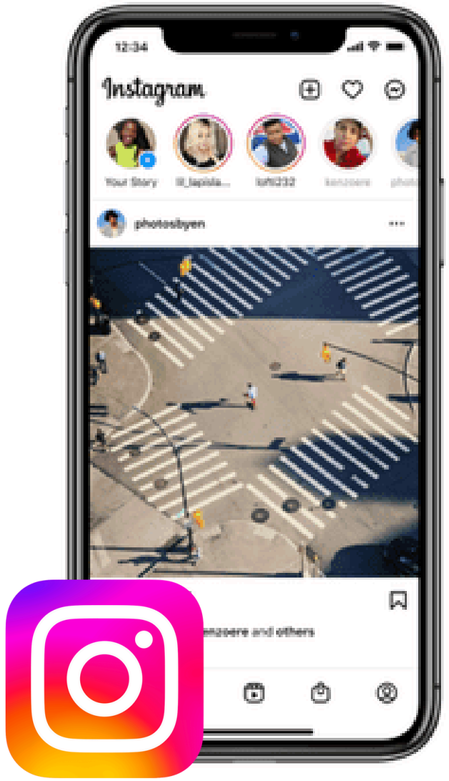


Un caso de aplicación real de Django en una empresa es Instagram, una de las redes sociales más grandes del mundo. Instagram utilizó Django en su fase inicial y aún lo sigue utilizando para varias partes de su plataforma, especialmente en el backend.

¿Cómo usa Instagram Django?

1. Escalabilidad y manejo de grandes volúmenes de tráfico
2. Desarrollo rápido
3. Manejo de bases de datos
4. Seguridad
5. Interfaz de administración

Caso de aplicación



¿Por qué Instagram eligió Django?

- **Rapidez de desarrollo:** Django permite lanzar características rápidamente, crucial para una startup en crecimiento.
- **Simplicidad:** A pesar de ser una plataforma compleja, Django ofrece simplicidad y limpieza en el código, lo que facilita el mantenimiento a largo plazo.
- **Escalabilidad:** A medida que Instagram creció, Django pudo soportar el aumento de usuarios sin perder rendimiento.
- **Seguridad:** Django maneja automáticamente muchas de las preocupaciones de seguridad, ayudando a proteger los datos de millones de usuarios.