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

Nicolás Zuluaga Galindo

Django

1. Breve descripción del lenguaje/framework. (objetivo del framework)

   Django es un framework de desarrollo web de alto nivel escrito en Python que permite crear aplicaciones web rápidas, seguras y escalables. Su objetivo principal es facilitar el desarrollo de aplicaciones complejas mediante herramientas integradas, siguiendo el principio de "No te repitas" (DRY, Don't Repeat Yourself) y promoviendo un código limpio y reutilizable.

   Ofrece una estructura bien organizada basada en el patrón Model-View-Template (MVT), lo que permite separar la lógica de negocio, las vistas y las plantillas, facilitando la colaboración entre equipos y mejorando la mantenibilidad del código.

   Su principal fortaleza es la simplificación y agilidad para el desarrollo web al proporcionar soluciones listas para usar en áreas como autenticación, gestión de bases de datos, formularios, seguridad y panel administrativo.

2. Uso de herramientas de gestión de dependencias (npm, pip, Maven, Gradle, etc.).

   Django, al ser un framework de Python, utiliza pip como su herramienta principal para la gestión de dependencias. pip permite instalar librerías externas como Django y otros paquetes necesarios para el proyecto directamente desde PyPI (Python Package Index).

   Adicionalmente, se pueden utilizar entornos virtuales con herramientas como venv para aislar las dependencias de cada proyecto y evitar conflictos. Los desarrolladores suelen listar las dependencias en un archivo requirements.txt, que facilita reproducir el entorno en otros equipos mediante el comando pip install -r requirements.txt. Todo lo anterior, asegura simplicidad, compatibilidad y un manejo eficiente de librerías, similar a lo que hacen herramientas como npm (JavaScript) o Maven (Java) en sus respectivos lenguajes.

3. URL de guide install

   La documentación oficial de Django ofrece una guía clara y completa para instalar y configurar el framework. Puedes encontrarla en el siguiente enlace:

   https://docs.djangoproject.com/en/stable/intro/install/

Esta guía incluye pasos detallados para instalar Django en diferentes sistemas operativos, configurar un entorno virtual, manejar dependencias con pip y ejecutar tu primer proyecto Django. Es una referencia imprescindible tanto para principiantes como para desarrolladores avanzados.

Estructura recomendada del proyecto:

1. Carpetas principales y su propósito (e.g., `src`, `tests`, `assets`, etc.).

La estructura de un proyecto Django está diseñada para garantizar una organización lógica, modular y escalable, lo que permite a los desarrolladores trabajar de manera más eficiente en proyectos grandes o pequeños. En Django, un **proyecto** es el contenedor principal que agrupa toda la configuración global y las aplicaciones necesarias para construir una aplicación web completa. Dentro de un proyecto, las **aplicaciones (apps)** son módulos individuales que representan funcionalidades específicas. Por ejemplo, en un sistema de comercio electrónico, podrías tener una aplicación para gestionar productos, otra para procesar pagos y otra para manejar usuarios. Este enfoque modular no solo promueve la reutilización de código, sino que también facilita la colaboración en equipos de desarrollo.

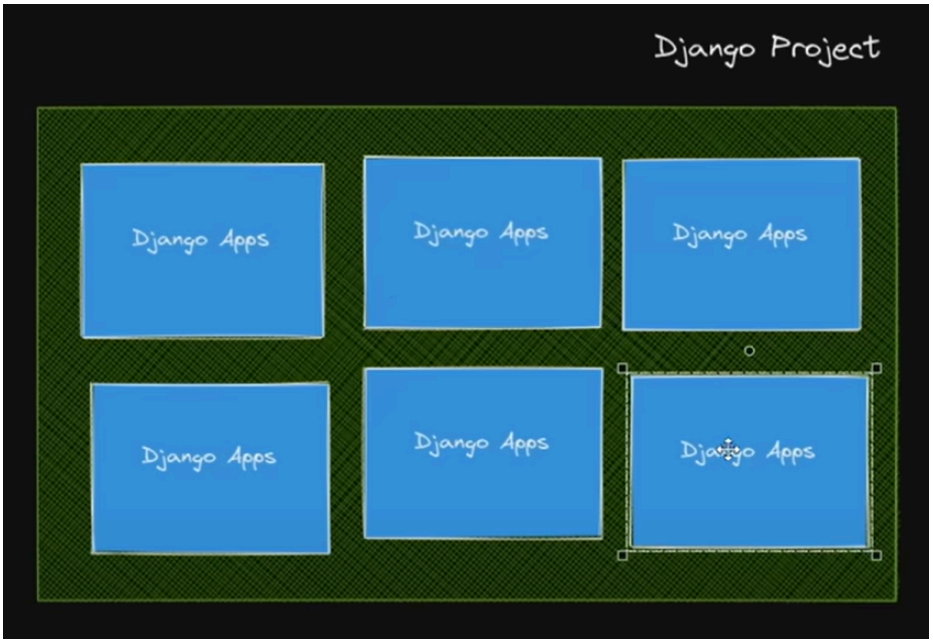

Fig 1. Esquema del proyecto en Django

Al crear un proyecto Django, se genera una estructura básica con carpetas y archivos clave, cada uno con un propósito específico:

- **`project_name/`**: Es la carpeta principal del proyecto y contiene la configuración global del mismo. Aquí encontrarás:
  - **`settings.py`**: Configuración general del proyecto, como las bases de datos, aplicaciones instaladas y ajustes de seguridad.
  - **`urls.py`**: Archivo donde se define el enrutamiento principal del proyecto, conectando las URLs a las vistas correspondientes.

    - `wsgi.py` y `asgi.py`: Archivos para la interacción con servidores web o de aplicaciones.
- `app_name/`: Representa una aplicación dentro del proyecto. Cada aplicación es autónoma y contiene:
    - `models.py`: Define la estructura de las tablas en la base de datos mediante clases de Python.
    - `views.py`: Contiene la lógica de negocio para procesar solicitudes y devolver respuestas.
    - `admin.py`: Configura cómo los modelos aparecen en el panel administrativo de Django.
    - `apps.py`: Archivo de configuración de la aplicación.
    - `migrations/`: Carpeta que contiene los archivos de migración para sincronizar los modelos con la base de datos.
- `static/`: Carpeta utilizada para almacenar archivos estáticos, como hojas de estilo CSS, scripts JavaScript e imágenes, que no cambian en el servidor.
- `templates/`: Contiene los archivos HTML utilizados para renderizar la interfaz del usuario. Esta carpeta puede organizarse por aplicación para mantener la coherencia.
- `tests/`: Incluye pruebas unitarias para validar la funcionalidad del proyecto y garantizar que las aplicaciones se comporten según lo esperado.

Esta estructura modular no solo organiza el proyecto de manera clara, sino que también facilita el mantenimiento y la escalabilidad, permitiendo que los desarrolladores dividan el trabajo en partes independientes y reutilizables. Con esta base, Django ofrece un entorno bien estructurado para construir aplicaciones robustas y fáciles de mantener.

2. Ubicación de componentes, routers, store, models, vistas, controladores, servicios, etc.

Un concepto importante de comprender es la arquitectura de software de tipo **Model-View-Controller (MVC)** y **Model-View-Template (MVT)**. Estos estructuran los distintos archivos según la funcionalidad que cumplan dentro de la app.

El primer organiza las aplicaciones dividiendo sus componentes en tres partes principales para separar responsabilidades:

1. **Modelo (Model):**
   Maneja los datos y la lógica de negocio de la aplicación. Es responsable de acceder, almacenar y gestionar los datos que necesita la aplicación.
   Ejemplo: Un modelo que represente una tabla de usuarios en una base de datos.
2. **Vista (View):**
   Se encarga de la interfaz de usuario. Presenta los datos al usuario y captura sus interacciones (clics, formularios, etc.).
   Ejemplo: Una página HTML o un componente gráfico que muestra una lista de usuarios.
3. **Controlador (Controller):**
   Actúa como intermediario entre el Modelo y la Vista. Recibe las solicitudes del usuario, consulta al Modelo y devuelve los datos procesados a la Vista.

Ejemplo: Una función que procesa una solicitud HTTP y devuelve una página HTML con datos dinámicos.

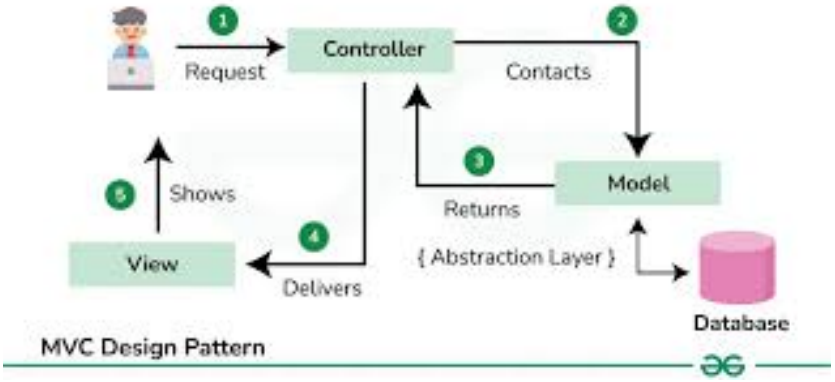

Fig 2. Flujo en MVC

Django utiliza el segundo patrón, que tiene una pequeña diferencia con el MVC tradicional: **el Controlador está implícito** dentro del framework.

1. **Modelo (Model):**
   Igual que en MVC, el modelo representa la estructura de los datos y se encarga de la interacción con la base de datos.
2. **Vista (View):**
   En Django, la vista contiene la lógica de negocio y actúa como el "controlador" de MVC. Es el componente que conecta los datos del modelo con la plantilla.
3. **Plantilla (Template):**
   En lugar de la "Vista" de MVC, Django usa "Plantillas" para definir la interfaz de usuario. Las plantillas son archivos HTML con etiquetas y filtros que permiten mostrar datos dinámicos.

Flujo en MVT (Django):

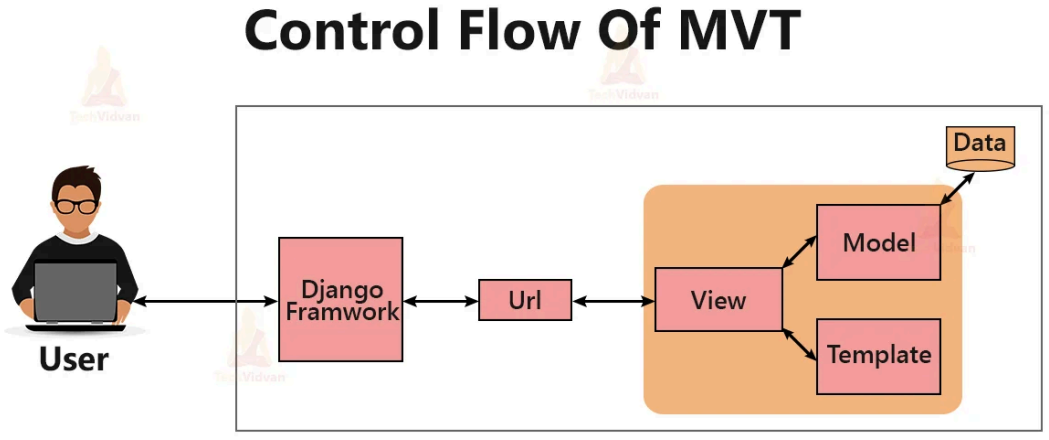

Fig 3. Flujo en MVT

A continuación se presentan 2 tablas a modo de resumen para comparar cada una de las arquitecturas y ver las implicaciones al momento de desarrollarlas.

Tabla 1. Comparación de MVC y MVT

Aspecto	MVC (Model-View-Controller)	MVT (Model-View-Template)
Controlador	El desarrollador debe implementarlo explicitamente	Implícito en el framework Django.
Vista	Responsable de la interfaz de usuario.	En Django, la vista gestiona la lógica de negocio.
Plantilla (Template)	No se menciona específicamente en MVC.	Define la interfaz de usuario con HTML dinámico.
Complejidad	Más control, pero requiere escribir más código.	Más Automatización y menos código manual.
Ecosistema	Usado en frameworks como Laravel,Ruby on Rails, etc.	Específico de Django aunque tiene similitudes con MVC.

Tabla 2. Ventajas y desventajas de MVC y MVT

Patrón	Ventajas	Desventajas
MVC	● Separación clara de responsabilidades. ● Ofrece control total sobre como se manejan las solicitudes. ● Facilita la colaboración entre equipos (backend y frontend).	● Requiere más código para implementar el controlador. ● Puede ser más complejo para principiantes. ● más trabajo manual para conectar el modelo y la vista.
MVT	● El controlador implícito reduce la cantidad de código que se debe escribir. ● Integra herramientas listas para usar (Por ejemplo, manejo de formularios, autenticación) ● Más rápido para prototipos y aplicaciones de tamaño mediano.	● Menos flexibilidad en algunos aspectos del flujo de control ● Dependes del flujo definido por Django. ● La abstracción puede ser compleja para desarrollados nuevos en Django.

Relación entre MVT y Django

Django implementa el patrón **MVT**, con una estructura que asigna responsabilidades claras entre sus componentes:

1. **Modelo (Model):**
   Representado por el archivo `models.py` en cada aplicación. Define las tablas de la base de datos y la lógica asociada a los datos, usando el ORM de Django.
2. **Vista (View):**
   En Django, las vistas (`views.py`) gestionan la lógica de negocio. Conectan el Modelo con las Plantillas y devuelven una respuesta al usuario. Esto equivale al "controlador" de MVC.
3. **Plantillas (Templates):**
   Definen la interfaz de usuario utilizando el motor de plantillas de Django, permitiendo integrar HTML con datos dinámicos (variables, bucles y filtros).
4. **Controlador implícito:**
   Django actúa como el controlador que maneja las solicitudes HTTP, mapea URLs a vistas y envía las respuestas correspondientes.

Por ejemplo, en Django, cuando un usuario accede a una URL, el framework automáticamente:

- Mapea la URL a una vista definida en `urls.py`.
- La vista interactúa con los modelos (`models.py`) para obtener los datos.
- La vista pasa los datos a una plantilla (`templates/`).
- La plantilla genera el HTML dinámico que se devuelve al usuario.

Esta integración automática reduce la complejidad del desarrollo y permite a los desarrolladores centrarse en construir la lógica de negocio y la interfaz.

3. Convenciones para nombres de carpetas y archivos

Django sigue las convenciones de Python para mantener un código limpio y fácil de entender. Algunas reglas importantes son:

- Los nombres de carpetas y archivos deben usar **snake_case** (letras minúsculas separadas por guiones bajos). Ejemplo: `my_app/`, `user_models.py`.
- Los nombres de clases, como los modelos, deben usar **CamelCase**. Ejemplo: `class UserProfile(models.Model):`.
- Evita nombres genéricos o ambiguos para las carpetas y archivos. Por ejemplo, en lugar de `utils.py`, usa un nombre más específico como `date_helpers.py`.
- Coloca los archivos relacionados con pruebas en una carpeta `tests/` dentro de cada aplicación.

Seguir estas convenciones facilita la colaboración en equipo y asegura que el código sea comprensible para otros desarrolladores.

4. URL de las guías de estilo específicas del lenguaje/framework (e.g., PEP 8 para Python, ESLint para JavaScript).

Para garantizar que el código sea legible, mantenible y siga las mejores prácticas, es fundamental respetar las guías de estilo recomendadas para Python y Django. Aquí tienes algunas referencias importantes:

- **PEP 8 (Python Enhancement Proposal 8):** Es la guía oficial de estilo para escribir código Python limpio y uniforme. Puedes consultarla aquí: https://peps.python.org/pep-0008/.
- **Django Coding Style:** Aunque Django sigue las normas de PEP 8, también tiene recomendaciones específicas para organizar y escribir código en proyectos Django. Más información aquí: https://docs.djangoproject.com/en/stable/internals/contributing/writing-code/coding-style/.
- **ESLint (para JavaScript):** Si usas JavaScript en el frontend, ESLint es una herramienta útil para garantizar que tu código siga un estándar. Guía oficial: https://eslint.org/.

Adherirse a estas guías asegura que el código sea uniforme y profesional, especialmente en proyectos colaborativos.

5. Opinión personal

Django es una herramienta robusta y bien estructurada para el desarrollo web. Una de sus mayores fortalezas es que proporciona un enfoque claro para organizar los proyectos, lo que facilita el desarrollo en equipo y la escalabilidad a largo plazo. Además, su énfasis en buenas prácticas, como el uso del patrón MVT y la separación de responsabilidades, hace que los proyectos sean fáciles de mantener. Aunque requiere algo de aprendizaje inicial, considero que es fundamental trabajar en un proyecto práctico para comprender realmente los fundamentos de Django y su estructura. Personalmente, realicé un curso sobre Django que recomiendo, ya que combina un proyecto práctico con cápsulas teóricas cortas que facilitan el aprendizaje.

Curso introductorio a Django: https://www.youtube.com/watch?v=T1intZyhXDU&t=2098s

6. Referencias

- Django Software Foundation. (n.d.). Documentation official de Django. Recuperado de https://docs.djangoproject.com/
- van Rossum, G., Warsaw, B., & Coghlan, N. (2001). PEP 8 – Style Guide for Python Code. Python Software Foundation. Recuperado de https://peps.python.org/pep-0008/
- Django Software Foundation. (n.d.). Django Coding Style. Recuperado de https://docs.djangoproject.com/en/stable/internals/contributing/writing-code/coding-style/
- OpenJS Foundation. (n.d.). ESLint - Pluggable JavaScript Linter. Recuperado de https://eslint.org/

NOTA: Este documento fue elaborado con el apoyo de herramientas de inteligencia artificial, incluyendo ChatGPT, para optimizar la claridad y la precisión de la información presentada. Las ideas y contenido reflejan un trabajo colaborativo entre las capacidades tecnológicas y el criterio del autor.