TRABAJO PRÁCTICO N°3

Santiago Lozano Fernández

Ins. Ind. Luis A. Huergo

Laboratorio de Algoritmos y Estructuras de Datos

Prof. Ignacio Miguel Garcia

10 de Julio de 2025

Índice

¿Qué es Django y por qué lo usaríamos?

Definición y propósito

Django es un framework web de alto nivel escrito en Python. Fue creado para simplificar y acelerar el desarrollo de aplicaciones web complejas con una filosofía clara:

- **Rápido:** ofrece herramientas integradas para desarrollar "desde concepto hasta producción" lo más rápido posible.
- **Seguro:** provee protección estandarizada contra amenazas comunes (como CSRF, XSS, SQL injection).
- **Escalable:** ha demostrado servir plataformas de gran tráfico, desde medios de noticias hasta redes sociales.

Historia y desarrollo

Nace en 2003–2005, creado por Adrian Holovaty y Simon Willison, en el diario *Lawrence Journal-World* (Kansas, EE.UU.), para gestionar sitios de noticias con gran flujo editorial. En julio de 2005 se publica bajo licencia BSD; su mantenimiento pasa a la Django Software Foundation (DSF) en 2008. Desde entonces mantiene un ciclo de crecimiento robusto, con mejoras al ORM, panel admin, seguridad, internacionalización y soporte de bases de datos múltiples .

Filosofía de diseño

Django incorpora principios sólidos heredados de Python, destacando:

- **DRY (Don't Repeat Yourself):** evitar redundancia y centralizar la lógica.
- **Menos código, más reuso:** herramientas "lista para usar", como panel de administración, autenticación, manejo de sesiones, etc.
- **Acoplamiento bajo:** componentes modulares que pueden ser reemplazados sin interferir con el resto del sistema.
- **Explícito mejor que implícito:** un pilar del Zen de Python, mantiene el código claro y directo.

Ventajas destacadas

- **Desarrollo ágil**: gracias a su batería de herramientas nativas, se acelera la creación desde un prototipo hasta producción productiva .
- **Seguridad incorporada:** protección contra XSS, CSRF, clickjacking, SQL injection, con actualizaciones rápidas al identificarse vulnerabilidades.
- **Escalabilidad comprobada:** usado por plataformas de alto tráfico: Instagram, Spotify, Pinterest, Mozilla, The Washington Post, Reddit, NASA, Dropbox y más.
- **Comunidad y ecosistema:** la DSF, DjangoCon, Django Girls y un ecosistema activo de paquetes y documentación garantizan soporte continuo y crecimiento.
- **Código mantenible y profesional:** toma prestado de los patrones profesionales (URLs limpias, sistema de plantillas, ORM avanzado) para fomentar buenas prácticas.

Cuándo y por qué usar Django

- Ideal para sitios centrados en datos, como blogs, noticias, e-commerce, redes sociales o CRM.
- Cuándo lanzar rápido un MVP y necesitar cosas como autenticación, panel admin o gestión de formularios desde el día 1.

- Si el contexto requiere escala futura o explícita necesidad de seguridad robusta.
- Cuando se prefiera una base sólida respaldada por comunidad, recursos y documentación extensa.

¿Qué es el patrón MTV en Django?

Django implementa el patrón MTV (Model–Template–View), una adaptación del clásico MVC. Cada componente en Django cumple funciones específicas:

- **Model:** representa la estructura de datos y lógica de negocio: clases Python que mapean tablas de base de datos, con validaciones y relaciones.
- **Template:** archivos estáticos (HTML, CSS) con placeholders para datos dinámicos; definen cómo se verá la información en la interfaz.
- **View:** funciones o clases en Python que reciben peticiones, consultan o modifican datos y devuelven respuestas HTTP, generalmente renderizando un template.

Flujo dentro de MTV:

1. El usuario hace una petición HTTP. Django asigna esa petición a la View correspondiente.
2. La View consulta el Model, usando el ORM para acceder a datos.
3. La View pasa datos al Template.
4. El Template genera el HTML final.
5. Django devuelve el HttpResponse al navegador. Todas estas capas funcionan de forma desacoplada, promoviendo un código limpio y modular

¿Qué es MVC?

El patrón MVC (Model–View–Controller), ideado en la década de 1970, organiza una aplicación en tres componentes separados para mejorar mantenimiento y escalabilidad:

- **Model:** gestiona datos y lógica de negocio (persistencia, validación...).
- **View:** es la interfaz de usuario, muestra los datos y recibe acciones.
- **Controller:** recibe inputs del usuario (a través de la View), procesa lógica, actualiza Models y elige qué View mostrar.

Ejemplo de flujo MVC:

1. El usuario hace clic.
2. La View lo captura y lo envía al Controller.
3. El Controller actualiza el Model según la acción.
4. El Model notifica a la View.
5. La View solicita datos al Model y renderiza la interfaz actualizada.

Cuadro 1

Comparativa entre MVC y MTV

Función	MVC (Model-View-Controller)	MTV (Model-Template-View) en Django
Model	Estructuras de datos + lógica de negocio	Clases Python que representan tablas con ORM
View	Interfaz mostrada al usuario, maneja entrada UI	Template: archivos HTML con placeholders

Controller	Procesa eventos de usuario y maneja lógica	View: recibe peticiones, consulta modelos, elige template

En Django, el "Controller" está implícito en el framework (ej. URL dispatcher), mientras que la lógica que se confundiría con controller en MVC se maneja en las Views de Django.

Principales diferencias y concepto práctico

- En MVC, la Controller recibe input del usuario, actualiza Model y actualiza la View.
- En MTV, la View de Django hace ese rol: recibe la petición, maneja lógica, accede al Model y pasa los datos al Template.
- El Template se encarga de la presentación, como haría la View en MVC.

¿Qué entendemos por *app* en Django?

Definición general

En Django, una app es un *paquete Python* autocontenido que ofrece una *funcionalidad específica*. Puede incluir modelos, vistas, plantillas, urls, archivos estáticos, tests, etc. Está diseñado para ser modular y reutilizable entre distintos proyectos Django.

El tutorial oficial describe esto como "una app es una web application que hace algo —por ejemplo, un sistema de blog, una base de datos pública o una pequeña app de encuestas"(Django Software Foundation, 2024).

Ventajas de usar apps

1. **Modularidad y organización:** al separar la lógica por dominio funcional, el proyecto resulta más claro y fácil de mantener.

2. **Reutilización:** una app bien diseñada puede usarse en múltiples proyectos sin modificaciones.
3. **Colaboración:** equipos o desarrolladores pueden trabajar en apps aisladas sin provocar conflictos en otros módulos.
4. **Escalabilidad y mantenimiento:** evita los "God apps" con miles de líneas; permite aislar funcionalidades para pruebas, migraciones y actualización independiente.

Estructura básica de una app

Al ejecutar python manage.py startapp polls, se crea un directorio con esta estructura típica:

```
polls/
  __init__.py
  admin.py
  apps.py
  migrations/
    __init__.py
  models.py
  tests.py
  views.py
```

- **models.py**: define la estructura de datos
- **views.py**: lógica de respuesta HTTP
- **templates/, static/**: presentación y recursos
- **apps.py**: configuración personalizada
- **admin.py**: registro en el panel administrador
- **tests.py**: tests unitarios

¿Qué es el flujo *request–response* en Django?

Visión general

El ciclo request–response es la secuencia que sigue Django al recibir una petición HTTP de un cliente (por ejemplo, un navegador) y devolver una respuesta apropiada (por ejemplo, una página HTML). En cada petición, Django realiza varios pasos clave para procesarla correctamente.

Pasos del ciclo paso a paso

1. **Petición desde el navegador → servidor web**

   El navegador envía una petición HTTP al servidor (Apache, Nginx). Este delega la petición a Django mediante WSGI (Gunicorn, uWSGI).

2. **WSGI inicia la petición en Django**

   Django crea un objeto HttpRequest, que contiene metadatos: método, URL, cabeceras, cuerpo, etc.

3. **Middlewares (fase de *request*)**

   Se escanean cada middleware definido en settings.py (orden de entrada): pueden modificar la petición o directamente devolver una respuesta (short-circuit). Por ejemplo: CSRF, sesión, autenticación.

4. **URL dispatcher (resolución de ruta)**

   Django toma HttpRequest.path, busca coincidencias en ROOT_URLCONF para encontrar la View asignada.

5. **Middlewares (fase process_view)**

   Antes de ejecutar la View, se pueden aplicar otros middlewares (process_view), por ejemplo para CSRF, permisos, validaciones.

6. **Ejecución de la View**

   La función o clase View recibe HttpRequest, accede al ORM (modelos), ejecuta lógica de negocio y prepara datos (contexto) para la plantilla .

7. **Renderizado del Template**

   El contexto se pasa a un Template, que se convierte en HTML (o JSON, XML, etc.), mediante el motor de plantillas de Django .

8. **Middlewares (fase de *response*)**

   El HttpResponse generado pasa de nuevo por los middlewares (en orden inverso), que pueden añadir cabeceras, comprimir contenido, logueo, etc.

9. **WSGI envía la respuesta al navegador**

   Finalmente, el response vuelve al servidor WSGI, que lo transmite al servidor web, y éste devuelve el contenido al cliente .

¿Qué es el concepto de ORM (*Object-Relational Mapping*)?

¿Qué es un ORM?

Un Object-Relational Mapper (ORM) es una capa de abstracción que permite interactuar con bases de datos relacionales usando objetos en lugar de SQL. Traduce operaciones sobre clases y atributos Python a consultas SQL optimizadas para distintas bases

de datos, permitiendo al desarrollador centrarse en la lógica sin preocuparse por sintaxis SQL específica del motor.

¿Cómo funciona en Django?

- **Definición de modelos:** cada modelo es una clase Python que hereda de models.Model y representa una tabla, con atributos que definen columnas, relaciones y validaciones.
- **Migraciones:** comandos makemigrations y migrate crean y actualizan la estructura de la base de datos automáticamente .
- **QuerySets:** permiten realizar operaciones CRUD (crear, leer, actualizar, borrar) con métodos como .create(), .filter(), .get(), .update(), .delete().

Ejemplo:

```
from django.db import models


class Album(models.Model):
    title = models.CharField(max_length=30)
    artist = models.CharField(max_length=30)


# Crear
a = Album.objects.create(title="Divide", artist="Ed Sheeran")
# Leer
albums = Album.objects.filter(artist="Ed Sheeran")
# Actualizar
Album.objects.filter(title="Divide").update(artist="Ed Sheeran ft. Someone")
```

```
# Eliminar
Album.objects.filter(title="Divide").delete()
```

Beneficios del ORM

- **Más rápido y menos propenso a errores**: la codificación se vuelve más ágil y segura, evitando errores comunes de SQL.
- **Portabilidad entre bases de datos:** cambiar de MySQL a PostgreSQL o SQLite solo requiere instalar el conector adecuado.
- **Protección contra SQL injection**: el ORM se encarga de parametrizar las consultas automáticamente.
- **Lazy loading y eficiencia**: las consultas se ejecutan solo al acceder a datos, y QuerySets mantienen cache interno para acelerar lecturas repetidas.

Limitaciones

- **Coste de abstracción:** en casos de consultas complejas o masivas, puede ser menos eficiente que SQL puro.
- **Consultas ineficientes inesperadas:** el uso inadecuado puede generar múltiples consultas ("N+1 problem"). Se recomienda usar herramientas como select_related(), prefetch_related() y bulk_create().
- **Consulta raw SQL:** Django permite usar raw() o connection.cursor() para consultas específicas donde el ORM no es adecuado .

¿Qué son los templates en Django?

Definición y objetivo

Un template en Django es un archivo de texto —generalmente HTML— que emplea el *lenguaje de plantillas de Django* (DTL), diseñado para separar lógica de presentación y generar contenido dinámico. La idea es que el backend (Views) gestione los datos, el template se concentre en cómo se muestran al usuario.

Sintaxis y componentes del DTL

El lenguaje de plantillas de Django permite:

- **Variables:** {{ variable }} se reemplazan por valores del contexto enviado desde la vista.
- **Tags:** estructuras de control, ejemplo {% if %}, {% for %}, {% include %}, {% csrf_token %} .
- **Filters:** transforman valores, ejemplo {{ value|date:"Y-m-d" }}, |upper.
- **Comments:** comentarios no renderizados, en {# ... #} .

Casos prácticos

A. Archivo de template sencillo

```
<h1>Hello {{ user.first_name }}</h1>
{% if items %}
  <ul>
    {% for item in items %}
      <li>{{ item.name|upper }}</li>
```

```
   {% endfor %}
 </ul>
{% else %}
 <p>No items found.</p>
{% endif %}
```

Este código renderiza datos dinámicos proporcionados desde la View.

B. **Herencia de plantillas**

- base.html define la estructura principal con bloques:

```
<html><head>{% block title %}My Site{% endblock %}</head>
<body>{% block content %}{% endblock %}</body></html>
```

- home.html extiende:

```
{% extends "base.html" %}

{% block title %}Home{% endblock %}

{% block content %}<p>Welcome!</p>{% endblock %}
```

Este patrón permite evitar duplicación y gestionar diseño de forma eficiente.

Organización de plantillas

- **App-level:** dentro de app/templates/app/, detectadas automáticamente si APP_DIRS=True.

- **Project-level:** en carpeta central templates/, especificada con DIRS=[...] en settings.py.

Ventajas clave

- Separación clara de lógica y presentación (clean MVC/MTV) .
- Diseño modular y reutilizable.
- Fácil de aprender, permite prototipos rápidos incluso con HTML básico.
- Útiles incluso en backends de APIs o correo, aunque en algunos escenarios se prefieren frameworks JS modernos.

¿Cómo se lo instala?

Django puede instalarse fácilmente mediante pip, el gestor de paquetes de Python. Se recomienda trabajar dentro de un entorno virtual para evitar conflictos de dependencias.

Pasos básicos:

1. **Crear entorno virtual:** python -m venv env
2. **Activarlo:**
    - Windows: env\Scripts\activate
    - Linux/macOS: source env/bin/activate
3. **Instalar Django:** pip install django
4. **Verificar instalación:** django-admin --version. Esto prepara el entorno para comenzar un nuevo proyecto con django-admin startproject.

Referencias

AlmaBetter. (2024). *Django ORM*.

https://www.almabetter.com/bytes/tutorials/django/django-orm

Better Stack. (2024). *Introduction to Django ORM*.

https://betterstack.com/community/guides/scaling-python/django-orm-intro/

Certisured. (2024). *Importance of ORM in Django Frameworks*.

https://certisured.com/blogs/importance-of-orm-in-django-frameworks/

Django Software Foundation. (2024). *Django at a glance*.

https://docs.djangoproject.com/en/5.2/intro/overview/

Django Software Foundation. (2024). *Django installation guide*.

https://docs.djangoproject.com/en/5.2/intro/install/

Django Software Foundation. (2024). *Templates*.

https://docs.djangoproject.com/en/5.2/topics/templates/

Django Software Foundation. (2024). *Template language reference*.

https://docs.djangoproject.com/en/5.2/ref/templates/language/

Django Software Foundation. (2024). *Request and response objects*.

https://docs.djangoproject.com/en/5.2/ref/request-response/

Django Software Foundation. (2024). *Creating apps in Django*.

https://docs.djangoproject.com/en/5.2/intro/tutorial01/

Django Software Foundation. (2024). *App configuration reference*.

https://docs.djangoproject.com/en/5.2/ref/applications/

Django Forum. (2021). *MVC or MVT architecture?*

https://forum.djangoproject.com/t/mvc-or-mvt-architecture/3496

Emmanuel, D. (2023). *Django templates: mastering the basics*. Medium.

https://emmanueldav.medium.com/django-templates-mastering-the-basics-29a99813af9f

Esketchers. (2023). *9 reasons why we use Django*.

https://esketchers.com/9-reasons-why-use-django-framework/

Foreignerds. (2022). *A brief history of Django: from inception to prominence*.

https://foreignerds.com/a-brief-history-of-django-from-inception-to-prominence/

Full Stack Python. (2024). *Django ORM*. https://www.fullstackpython.com/django-orm.html

GeeksForGeeks. (2024). *Django ORM: inserting, updating, deleting data*.

https://www.geeksforgeeks.org/python/django-orm-inserting-updating-deleting-data/

GeeksForGeeks. (2024). *Django templates*.

https://www.geeksforgeeks.org/python/django-templates/

GeeksForGeeks. (2024). *How to create an app in Django*.

https://www.geeksforgeeks.org/how-to-create-an-app-in-django/

GeeksForGeeks. (2024). *Difference between MVC and MVT*.

https://www.geeksforgeeks.org/software-engineering/difference-between-mvc-and-mvt-design-patterns/

GeeksForGeeks. (2024). *Django MVT structure*.

https://www.geeksforgeeks.org/python/django-project-mvt-structure/

JetBrains. (2023). *What is the Django web framework?*.

https://blog.jetbrains.com/pycharm/2023/11/what-is-the-django-web-framework/

LearnDjango. (2024). *Template structure in Django*.

https://learndjango.com/tutorials/template-structure

LinkedIn. (2024). *Django request-response cycle*.

https://www.linkedin.com/posts/praseesh_django-request-response-cycle-the-django-activity-7258703371165999104-46hW

Medium (Developerstacks). (2023). *Django request-response cycle*.

https://medium.com/@developerstacks/django-request-response-cycle-7165167f54c5

Medium (Mindfire Solutions). (2023). *Advantages of using Django to build scalable platforms*.

https://medium.com/@mindfiresolutions.usa/advantages-of-using-django-framework-to-build-scalable-video-platforms-7d32f3cdb985

Medium (Sohaib Anser). (2023). *Django ORM: pros and cons*.

https://sohaibanser.medium.com/django-orm-pros-and-cons-8ab069598c1b

Mozilla Developer Network (MDN). (2024). *Model-View-Controller (MVC)*.

https://developer.mozilla.org/en-US/docs/Glossary/MVC

Opensource.com. (2017). *Introduction to the Django ORM*.

https://opensource.com/article/17/11/django-orm

Real Python. (2024). *Django tutorials*. https://realpython.com/tutorials/django/

Reddit. (2023). *What is an app in Django?* https://www.reddit.com/r/django/comments/154npht/what_is_an_app_in_django/

Reddit. (2017). *Advantages of MTV architecture over MVC*. https://www.reddit.com/r/django/comments/5prwmz/what_are_advantages_of_mtv_architecture_over_mvc/

Revsys. (2020). *What is a Django app?* https://www.revsys.com/tidbits/what-is-a-django-app/

Stack Overflow. (2019). *Is Django MVC or MVT?* https://stackoverflow.com/questions/55805865/is-django-a-mvc-or-mvt-framework

Tutorialspoint. (2024). *What is Django ORM?* https://www.tutorialspoint.com/what-is-django-orm

Tutorialspoint. (2024). *MVC design pattern*. https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

W3Schools. (2024). *Django templates*. https://www.w3schools.com/django/django_templates.php

Wired. (2007). *Introducing Django*. https://www.wired.com/2007/01/tutorial-o-the-day-introducing-django/

Wired. (2010). *Use templates in Django*. https://www.wired.com/2010/02/build-a-microblog-with-django/

Wikipedia. (2024). *Django (web framework)*.

https://en.wikipedia.org/wiki/Django_(web_framework)

Wikipedia. (2024). *MVC (Model–View–Controller)*.

https://en.wikipedia.org/wiki/Model–view–controller

YouTube. (2023). *Basic tutorial on Django templates*.

https://www.youtube.com/watch?v=4uNeO5Hw9FE