

Factoría F5

NOTAS COMPLEMENTARIAS

Lunes 17 de Febrero de 2025




Juan Domingo Orín

¿Qué es Django?

1 Introducción a Django

♦ Patrón MTV (Modelo-Template-Vista)

Django sigue un **patrón arquitectónico llamado MTV** (Modelo-Template-Vista), que organiza el código en tres componentes principales:

-  **Modelo (Model):** Representa la estructura de la base de datos. Define las tablas y las relaciones mediante clases en Python usando el **ORM** (Object-Relational Mapping).
-  **Template (Template):** Se encarga de la capa de presentación. Utiliza el **motor de plantillas** de Django para generar páginas HTML dinámicas.
-  **Vista (View):** Contiene la lógica de negocio y maneja las peticiones HTTP. Actúa como intermediario entre los modelos y los templates.

♦ Ejemplo del patrón MTV en Django

Cuando un usuario accede a una URL en una aplicación Django: **1** La **Vista** procesa la solicitud y recupera los datos del **Modelo**. **2** Luego, la **Vista** pasa los datos a un **Template**, que los renderiza en HTML. **3** Finalmente, el usuario recibe la página con la información solicitada.






♦ Diferencia con MVC (Modelo-Vista-Controlador)

Aunque Django usa el patrón **MTV**, sigue principios similares al **MVC** (Model-View-Controller), pero en Django, la **Vista (View)** cumple el papel del **Controlador (Controller)** en MVC.

2 Framework de Desarrollo

♦ Características de Django

Django es un **framework de desarrollo web** que proporciona herramientas y funcionalidades preconstruidas para facilitar la creación de aplicaciones web. En lugar de escribir todo desde cero, Django ofrece:

-  **ORM** para interactuar con bases de datos sin escribir SQL.
-  **Enrutador de URLs** para gestionar las direcciones web de la aplicación.
-  **Sistema de autenticación** con manejo de usuarios y permisos.
-  **Motor de plantillas** para generar HTML dinámico.
-  **Panel de administración automático** para gestionar la base de datos sin esfuerzo.

♦ Ventaja Principal




Django permite desarrollar aplicaciones más rápido y con menos código, lo que mejora la **productividad del desarrollador**.

3 Open-Source y Mantenido por la Comunidad

♦ Historia de Django

Django es un proyecto de **código abierto**, lo que significa que su código fuente está disponible para todos y se mantiene gracias a una comunidad activa de desarrolladores. Se lanzó en **2005** por el equipo de **Lawrence Journal-World** y desde entonces ha crecido con contribuciones globales.

♦ Actualizaciones y Comunidad

-  Se actualiza constantemente con nuevas versiones que mejoran **rendimiento, seguridad y funcionalidad**.
-  Existen miles de **paquetes y bibliotecas** desarrolladas por la comunidad que amplían sus capacidades.
-  Tiene una **documentación muy completa y detallada**, facilitando su aprendizaje.





♦ Ejemplo de Uso en la Industria

Django es utilizado por empresas como **Instagram, Pinterest, Mozilla, Disqus y The Washington Post**, lo que demuestra su **confiabilidad y escalabilidad**.

4 Basado en Python, con Enfoque en Productividad

♦ Ventajas de Python

Django está desarrollado en **Python**, uno de los lenguajes de programación más populares y versátiles. Su sintaxis clara y legible permite desarrollar aplicaciones rápidamente.

-  **Menos código, más funcionalidad:** Django permite escribir menos líneas de código en comparación con otros frameworks como **Java Spring** o **ASP.NET**.
-  **Facilidad de aprendizaje:** Al estar basado en Python, es fácil de entender y usar para principiantes.
-  **Enfoque en DRY (Don't Repeat Yourself):** Django promueve la **reutilización de código** para evitar redundancias.
-  **Uso de librerías y paquetes:** Se integra fácilmente con paquetes externos de Python, como **NumPy, Pandas o Celery**.





◆ **Beneficio Clave**

Con Django, los desarrolladores pueden enfocarse en la **lógica de negocio** en lugar de preocuparse por configuraciones complejas.

5 Seguridad Integrada

◆ **Mecanismos de Seguridad**

Uno de los puntos fuertes de Django es su enfoque en **seguridad**. El framework incluye mecanismos para proteger las aplicaciones contra ataques comunes:

-  **Protección contra SQL Injection:** El **ORM** de Django evita consultas SQL maliciosas al interactuar con la base de datos de manera segura.
-  **Protección contra CSRF (Cross-Site Request Forgery):** Django incluye **tokens CSRF** para prevenir ataques donde un usuario malintencionado intenta realizar acciones en nombre de otro usuario.
-  **Protección contra XSS (Cross-Site Scripting):** El **motor de plantillas** escapa automáticamente los datos para evitar la inyección de código malicioso en el navegador.
-  **Autenticación y gestión de usuarios:** Proporciona un sistema de autenticación robusto con **hash de contraseñas** y gestión de permisos.

◆ **Ejemplo de Seguridad en Django**

Para habilitar **CSRF** en formularios, se usa:



```
<form method="post">
  {% csrf_token %}
  <input type="text" name="usuario">
  <button type="submit">Enviar</button>
</form>
```


Django genera automáticamente un **token** para validar la autenticidad de la solicitud.

6 Escalabilidad y Modularidad

◆ **Escalabilidad**

Django es **altamente escalable**, lo que significa que puede manejar proyectos desde pequeñas aplicaciones hasta plataformas con millones de usuarios.

-  **Escalabilidad horizontal:** Se puede distribuir en múltiples servidores para manejar más tráfico.
-  **Soporte para bases de datos escalables:** Puede integrarse con **PostgreSQL, MySQL, SQLite** o incluso **NoSQL** como **MongoDB**.

-  **Uso de caché:** Compatible con sistemas de caché como **Redis** y **Memcached** para mejorar el rendimiento.

♦ **Modularidad**

Django permite dividir una aplicación en **múltiples apps reutilizables**, facilitando su mantenimiento y expansión.

Ejemplo de Modularidad

En Django, se pueden crear diferentes apps dentro de un mismo proyecto. Por ejemplo:

```
django-admin startapp blog
django-admin startapp usuarios
django-admin startapp comentarios
```

Cada app maneja una funcionalidad específica, permitiendo un código más organizado y fácil de escalar.

Resumen Final

Django es un framework poderoso porque combina:

✓ Un **patrón MTV** bien estructurado. ✓ **Herramientas listas para usar** que facilitan el desarrollo. ✓ Un **ecosistema open-source** con gran soporte comunitario. ✓ Un **enfoque en productividad** gracias a Python. ✓ **Seguridad integrada** sin configuraciones adicionales. ✓ **Escalabilidad** para proyectos de cualquier tamaño.

¿Por qué usar Django?

Django es un **framework web de alto nivel** escrito en Python que permite desarrollar aplicaciones de manera rápida y segura. Su lema es "**baterías incluidas**", lo que significa que proporciona muchas herramientas y funcionalidades listas para usar, sin necesidad de instalar paquetes adicionales para las tareas más comunes del desarrollo web.

A continuación, exploramos las **características clave** que hacen de Django una excelente opción para el desarrollo de aplicaciones web:

1 ORM (Object-Relational Mapping) para bases de datos

- Django incluye un **mapeo objeto-relacional (ORM)** que permite interactuar con bases de datos usando código Python en lugar de escribir consultas SQL manualmente.
- Facilita el trabajo con bases de datos como **PostgreSQL, MySQL, SQLite y Oracle**.
- Ejemplo de modelo en Django (sin necesidad de escribir SQL directamente):

```
python
CopiarEditar
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=200)
    completed = models.BooleanField(default=False)
```

- Este modelo se traduce automáticamente a una tabla en la base de datos cuando ejecutamos:

```
bash
CopiarEditar
python manage.py makemigrations
python manage.py migrate
```

- **Ventajas del ORM de Django:**

- ✓ Reduce errores en consultas SQL.
 - ✓ Facilita la portabilidad entre distintos motores de bases de datos.
 - ✓ Mejora la seguridad contra inyecciones SQL.
-

2 Sistema de autenticación robusto

- Django proporciona un **sistema de autenticación integrado**, lo que significa que no es necesario construir un sistema de inicio de sesión desde cero.
- Soporta **gestión de usuarios, inicios de sesión, permisos y grupos** con solo configurarlo en `settings.py`.
- Ejemplo de cómo Django maneja autenticación de usuarios:

```
python
CopiarEditar
```

```
from django.contrib.auth.models import User
user = User.objects.create_user(username="juan", password="django123")
```

- Django también permite agregar **autenticación basada en tokens o JWT** para APIs.
 - **Ventajas del sistema de autenticación de Django:**
 - ✓ Seguridad incorporada (manejo de contraseñas con hashing).
 - ✓ Fácil integración con OAuth y autenticación social (Google, Facebook, etc.).
 - ✓ Control granular de permisos para usuarios y grupos.
-

3 Panel de administración automático

- Una de las características más **diferenciales** de Django es su **panel de administración integrado**, que permite gestionar la base de datos sin necesidad de construir interfaces adicionales.
- Para activarlo, solo es necesario registrar los modelos en `admin.py`:

```
python
CopiarEditar
from django.contrib import admin
from .models import Task

admin.site.register(Task)
```

- Luego, podemos acceder al panel en `http://127.0.0.1:8000/admin/` después de crear un superusuario con:

```
bash
CopiarEditar
python manage.py createsuperuser
```

- **Ventajas del panel de administración de Django:**
 - ✓ Generado automáticamente sin necesidad de programar interfaces.
 - ✓ Personalizable con nuevas funciones y permisos.
 - ✓ Facilita la gestión de datos sin necesidad de escribir código SQL.
-

4 Middleware para seguridad y rendimiento

- Django incluye **middleware**, que son capas de procesamiento que se ejecutan entre la petición del usuario y la respuesta del servidor.
- Algunos middleware clave en Django incluyen:
 - **Seguridad:** Protege contra ataques como **Cross-Site Scripting (XSS)** y **Cross-Site Request Forgery (CSRF)**.
 - **Autenticación:** Maneja sesiones de usuarios y permisos.
 - **Compresión y caché:** Mejora el rendimiento al comprimir respuestas HTTP y optimizar la carga de páginas.
- **Ventajas del middleware en Django:**
 - ✓ Seguridad mejorada sin necesidad de configuraciones adicionales.
 - ✓ Mayor rendimiento con optimización automática.
 - ✓ Extensibilidad para agregar funcionalidades personalizadas.

Conclusión

Django se destaca por su **enfoque en la simplicidad, seguridad y rapidez en el desarrollo web**. Con su ORM, autenticación integrada, panel de administración y middleware robusto, es una excelente opción para construir aplicaciones de cualquier tamaño, desde pequeños proyectos hasta plataformas a nivel empresarial.

Ejemplo práctico ‘taskmanager’

1 Creación de la Carpeta del Proyecto

Antes de comenzar, es importante organizar los archivos correctamente creando una carpeta para el proyecto:

- **Crear la carpeta del proyecto:**

```
bash
CopiarEditar
mkdir "nombre-carpeta-proyecto"
```

✂ Esto asegura que todos los archivos del proyecto queden contenidos en un solo lugar.

- **Entrar en la carpeta:**

```
bash
CopiarEditar
cd "nombre-carpeta-proyecto"
```

✂ Cambia al directorio recién creado para que los siguientes comandos se ejecuten dentro del entorno correcto.

2 Creación del Entorno Virtual

Un **entorno virtual** permite aislar las dependencias del proyecto, evitando conflictos con otras versiones de paquetes de Python instaladas en el sistema.

◆ Opción 1: Usando uv (Linux)

- uv es una alternativa moderna a pip y venv para manejar entornos virtuales y dependencias más rápido y eficientemente:

```
bash
CopiarEditar
uv venv --python python3.13 .venv
source .venv/bin/activate
uv init --bare
```

✂ El comando `uv init --bare` inicializa el entorno sin dependencias preinstaladas, evitando paquetes innecesarios.

◆ Opción 2: Usando venv (Linux)

- venv es el método tradicional para crear entornos virtuales en Python:

```
bash
CopiarEditar
python3 -m venv .venv
source .venv/bin/activate
```

🔗 El punto (`.venv`) es un nombre estándar para la carpeta del entorno virtual.

◆ Opción 3: Usando `venv` (Windows)

- En sistemas Windows, la activación del entorno virtual se hace con `Scripts/activate`:

```
powershell
CopiarEditar
python -m venv .venv
.venv\Scripts\activate
```

🔗 Si la activación falla en Windows, puede ser necesario ejecutar `Set - ExecutionPolicy Unrestricted -Scope Process` en PowerShell.

3 Instalación del Framework Django

Django necesita ser instalado dentro del entorno virtual antes de poder ser utilizado.

- **Con `uv` o `pip` (válido para Linux y Windows):**

```
bash
CopiarEditar
uv add django
pip install django
django-admin --version
```

🔗 `django-admin --version` confirma que Django se instaló correctamente.

4 Creación del Proyecto Django 'taskmanager'

Este comando genera la estructura de archivos base para el proyecto Django.

- **Crear el proyecto Django:**

```
bash
CopiarEditar
django-admin startproject taskmanager
cd taskmanager
```

🔗 Esto crea la carpeta `taskmanager/` con los archivos de configuración del proyecto.

- **Verificar los archivos generados:**

```
bash
CopiarEditar
ls -l
```

🔗 Debe existir un archivo `manage.py` y una carpeta `taskmanager/` dentro del directorio.

5 Creación de la Aplicación 'tasks'

Django permite organizar funcionalidades en **aplicaciones** independientes dentro de un mismo proyecto.

- **Crear la aplicación tasks:**

```
bash
CopiarEditar
python manage.py startapp tasks
```

🔗 Cada aplicación en Django es una unidad modular con su propio modelo de datos, vistas y URLs.

- **Registrar la aplicación en settings.py:**

```
python
CopiarEditar
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "tasks", # Agregamos nuestra aplicación
]
```

🔗 Esto le dice a Django que la aplicación *tasks* forma parte del proyecto.

6 Creación del Modelo 'Task'

Los **modelos** en Django representan la estructura de la base de datos mediante clases en Python.

- **Definir el modelo en tasks/models.py:**

```
python
CopiarEditar
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=200) # Campo de texto
    completed = models.BooleanField(default=False) # Campo booleano

    def __str__(self):
        return self.title
```

🔗 Cada modelo se traduce en una tabla de base de datos.

- **Aplicar las migraciones para reflejar el modelo en la base de datos:**

```
bash
CopiarEditar
python manage.py makemigrations
python manage.py migrate
```

🔗 Estos comandos generan y aplican cambios en la base de datos.

7 Registrar el Modelo en el Panel de Administración

El **panel de administración de Django** permite gestionar los modelos sin necesidad de escribir consultas SQL.

- **Editar `tasks/admin.py`:**

```
python
CopiarEditar
from django.contrib import admin
from .models import Task

admin.site.register(Task)
```

🔗 Esto hará que el modelo `Task` aparezca en `http://127.0.0.1:8000/admin/`.

8 Creación de Superusuario

Para acceder al panel de administración, es necesario crear un superusuario.

- **Ejecutar el siguiente comando:**

```
bash
CopiarEditar
python manage.py createsuperuser
```

🔗 Se solicitará ingresar `username`, `email` y `password`.

- **Acceder al panel en:**

```
arduino
CopiarEditar
http://127.0.0.1:8000/admin
```

9 Mostrar Lista de Tareas

- ♦ **Crear la Vista y el Template**

- **Editar `tasks/views.py`:**

```
python
CopiarEditar
from django.shortcuts import render
from .models import Task

def task_list(request):
    tasks = Task.objects.all() # Obtener todas las tareas
    return render(request, "tasks/task_list.html", {"tasks": tasks})
```

🔗 Este código obtiene todas las tareas y las envía al template.

- **Crear el archivo de plantilla `task_list.html` en `tasks/templates/tasks/`:**

```
html
CopiarEditar
<h1>Task List</h1>
<ul>
    {% for task in tasks %}
        <li>{{ task.title }} - {% if task.completed %}✅{% else %}❌{%
    endif %}</li>
    {% endfor %}
</ul>
```

🔗 Esto mostrará la lista de tareas en una página web.

10 Configurar las URLs

Para acceder a la vista, es necesario definir su URL.

- **Editar `taskmanager/urls.py`:**

```
python
CopiarEditar
from django.contrib import admin
from django.urls import path
from tasks.views import task_list

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", task_list, name="task-list"),
]
```

🔗 Esto hará que la vista `task_list` esté disponible en la raíz del sitio.

1 1 Ejecutar el Servidor

- **Iniciar el servidor:**

```
bash
CopiarEditar
python manage.py runserver
```

🔗 Esto levantará el servidor en `http://127.0.0.1:8000/`.

- **Si hay migraciones pendientes, aplicarlas:**

```
bash
CopiarEditar
python manage.py migrate
```

🔗 Esto asegura que la base de datos está actualizada.

FIN