# Framework Django

Django es un framework de desarrollo web de alto nivel basado en el lenguaje Python, orientado a la creación de aplicaciones web seguras, eficientes y escalables mediante principios de diseño limpio y separación lógica de componentes. Su propuesta se centra en acelerar el desarrollo sin sacrificar calidad, promoviendo el uso de patrones estructurales sólidos como MVC y su adaptación al modelo MTV (Model-Template-View). Este informe profundiza en los fundamentos teóricos de Django, incluyendo su arquitectura, gestión de proyectos, administración de rutas y vistas, sistema de plantillas, y modelado de datos mediante una API integrada.

## Arquitectura Conceptual: MVC vs MTV

### I. Patrón MVC (Model-View-Controller)

El patrón MVC, ampliamente adoptado en la ingeniería de software, establece una separación entre:

- Modelo: Representa la lógica de datos y estructuras persistentes.
- Vista: Encargada de mostrar la información al usuario.
- Controlador**:** Intermediario que procesa las entradas del usuario, gestiona el flujo de datos y activa las respuestas apropiadas.

Este patrón permite una modularidad clara y fomenta el desarrollo mantenible, facilitando la reutilización y prueba de componentes individuales.

### II. Adaptación de Django: Patrón MTV

Django adopta una variante funcional del modelo MVC, denominada MTV, que mantiene los principios de separación pero asigna responsabilidades propias:

| Componente | Función específica en Django |
|---|---|
| **Model (Modelo)** | Estructura de datos, reglas de negocio, mapeo a base de datos |
| **Template (Plantilla)** | Visualización de datos, interfaz basada en HTML y lenguaje de plantillas |
| **View (Vista)** | Lógica que gestiona la solicitud HTTP, determina qué datos mostrar y qué plantilla renderizar |

A diferencia del MVC tradicional, donde el controlador es explícito, en Django el sistema de enrutamiento (URL dispatcher) y las vistas asumen ese papel, gestionando el flujo entre componentes.

## Instalación y Gestión de Proyectos

### I. Instalación del Entorno

Django se instala sobre Python mediante gestores de paquetes como pip, permitiendo su incorporación en entornos virtuales o servidores físicos. El framework incluye un servidor de desarrollo integrado, útil para pruebas locales, aunque en producción se recomienda integrarlo con servidores como Apache o Nginx utilizando módulos específicos que habilitan la comunicación entre el framework y el servidor HTTP (como mod_wsgi).

### II. Estructura de Proyecto

Una aplicación Django se organiza en una estructura jerárquica que incluye:

- Configuraciones globales del proyecto (settings, urls, wsgi)
- Aplicaciones modulares que encapsulan funcionalidades específicas
- Archivos de gestión que permiten ejecutar comandos, migraciones y pruebas

Esta arquitectura modular permite el desarrollo simultáneo de múltiples aplicaciones dentro de un mismo proyecto, promoviendo la escalabilidad y el trabajo colaborativo.

## Sistema de Rutas (URL dispatcher)

Django gestiona la relación entre una solicitud del navegador (ruta) y la función o clase que debe atenderla a través de expresiones declaradas en archivos de configuración. Este sistema permite asociar patrones URL con vistas específicas, posibilitando la organización por aplicaciones, el uso de parámetros dinámicos y la asignación de nombres referenciales para facilitar la navegación interna.

## Vistas en Django

Las vistas constituyen la unidad lógica que responde a una solicitud del navegador y determina qué información entregar. Django permite dos enfoques:

- Vistas basadas en funciones (Function-Based Views - FBV): Definen de manera explícita la lógica en una función que recibe la solicitud y retorna una respuesta.
- Vistas basadas en clases (Class-Based Views - CBV): Utilizan clases que encapsulan métodos específicos para distintos tipos de solicitudes (GET, POST, etc.), permitiendo herencia, extensión y reutilización estructurada.

Ambos enfoques permiten incorporar validaciones, gestión de sesiones, control de acceso y renderizado de plantillas.

**Sistema de Plantillas**

Django incluye un motor de plantillas que permite separar la lógica de presentación del contenido y control. Las plantillas son archivos de texto (usualmente HTML) que integran etiquetas del lenguaje de plantillas de Django (DTL), filtros y bloques reutilizables.

**Características del sistema de plantillas**

- Herencia de plantillas: Permite definir una estructura base que puede extenderse en múltiples vistas, evitando duplicación.
- Bloques reemplazables: Facilitan la personalización de secciones específicas.
- Etiquetas de control: Introducen lógica condicional (if, for) directamente en la presentación.
- Filtros de transformación: Adaptan visualmente los datos (date, length, safe).
- Contexto dinámico**:** Permite acceder a datos del modelo o resultados de vista desde la interfaz.

**Modelo de Datos**

Los modelos son clases que definen las estructuras persistentes de la aplicación. Cada modelo representa una tabla en la base de datos y sus atributos corresponden a columnas con tipos de datos específicos. Django gestiona automáticamente la creación, modificación y eliminación de estas estructuras mediante migraciones, permitiendo definir reglas, relaciones entre entidades y validaciones.

**Tipos de Campos**

Django ofrece una variedad amplia de campos para describir datos, incluyendo:

| **Campo** | **Propósito lógico** |
|---|---|
| CharField | Cadenas cortas |
| TextField | Texto largo |
| IntegerField | Números enteros |
| BooleanField | Valores lógicos |
| DateTimeField | Fechas y horas |
| ForeignKey | Relaciones uno a muchos |
| ManyToManyField | Relaciones muchos a muchos |

**API ORM (Object Relational Mapping)**

La API ORM de Django permite interactuar con la base de datos sin escribir SQL directamente. Este sistema proporciona métodos para:

- Crear instancias de modelos
- Consultar registros con filtros lógicos
- Actualizar o eliminar datos
- Realizar búsquedas, agregaciones y ordenamientos

La abstracción del ORM fortalece la seguridad, permite el uso de múltiples motores de base de datos (PostgreSQL, SQLite, MySQL) y facilita el desarrollo independiente del sistema subyacente.

**Conclusión**

Django representa una solución integral para el desarrollo web centrado en productividad, organización lógica y seguridad. Su arquitectura basada en el patrón MTV favorece la separación de responsabilidades, mientras que sus componentes permiten construir sistemas mantenibles, extensibles y profesionales. El uso de vistas estructuradas, plantillas reutilizables, modelos declarativos y una API de datos completa facilita la creación de soluciones que integren tanto diseño como funcionalidad en entornos reales. Su estudio teórico constituye una base sólida para comprender el ciclo completo de desarrollo y para aplicar principios de calidad en proyectos web modernos.