Trabajo práctico N°3 - Laboratorio De Algoritmos Y Estructuras De Datos

Preentrega Django

Santino Luciano Rodriguez Fuchs

Instituto Industrial Luis A. Huergo

4° Año Computadoras

Ignacio García

10 de Julio de 2025

1- ¿Qué es django y por qué lo usaríamos?

Django es un *framework*[1] de desarrollo web de alto nivel creado con Python, pensado para facilitar la creación de sitios web completos, seguros y escalables. Su propósito es reducir al mínimo el tiempo y el esfuerzo necesario para desarrollar una aplicación web desde cero, ofreciendo herramientas listas para usar que resuelven tareas comunes del desarrollo. Gracias a esto, Django permite que un desarrollador pase de la idea al sitio funcional en apenas unas horas, sin sacrificar calidad ni seguridad.

Una de las mayores ventajas de Django es que automatiza muchos de los procesos del desarrollo backend, lo cual lo convierte en una excelente opción incluso para quienes solo manejan conocimientos básicos de HTML, CSS y JavaScript. No es necesario construir a mano formularios, sistemas de usuarios, paneles de administración, enlaces entre páginas o conexiones con bases de datos, Django ya incluye módulos que hacen todo eso de forma automática y coherente. Esto agiliza enormemente el desarrollo, al punto que, en palabras del equipo oficial, Django está diseñado para que "puedas concentrarte en escribir tu aplicación sin tener que reinventar la rueda".

Además, se trata de un software gratuito y de código abierto, lo que significa que cualquier persona puede usarlo, adaptarlo o colaborar en su mejora sin restricciones. Su comunidad es amplia y activa, lo que facilita encontrar documentación, foros de ayuda y ejemplos prácticos de todo tipo.

Otro de los pilares fundamentales de Django es su enfoque en la seguridad. Desde el inicio, ofrece protección contra errores comunes que pueden comprometer un sitio web, como la inyección de SQL, los scripts maliciosos entre sitios, el clickjacking o la falsificación de solicitudes (CSRF). También incluye un sistema de autenticación robusto que permite gestionar usuarios, sesiones y contraseñas de manera segura.

En cuanto al diseño visual, Django utiliza un sistema de *templates*[2] que permite generar páginas HTML dinámicas sin mezclar la lógica de programación con el contenido visual. Esto hace que sea muy sencillo integrar hojas de estilo (CSS) y scripts (JavaScript) sin necesidad de conocimientos avanzados, el desarrollador puede concentrarse en la estructura general del sitio y dejar que Django se encargue de unir todo detrás de escena.

Finalmente, Django se destaca por su capacidad de escalar. Grandes empresas, organizaciones gubernamentales y plataformas científicas lo han usado con éxito para proyectos que manejan miles o millones de usuarios. Su arquitectura flexible lo hace ideal tanto para sitios simples como para sistemas complejos que necesitan soportar mucho tráfico y datos.

En definitiva, Django es un framework robusto, accesible y bien diseñado, que permite construir aplicaciones web de manera rápida, segura y profesional. Para quienes ya dominan Python y tienen una base en tecnologías web, representa una herramienta ideal para llevar proyectos del papel a internet con

[1] Estructura de desarrollo reutilizable que organiza el código mediante componentes, bibliotecas y patrones de diseño predefinidos, con el fin de facilitar la creación, mantenimiento y escalabilidad de aplicaciones.

[2] Archivo o estructura que define el diseño y la presentación de una interfaz, permitiendo separar el contenido dinámico de la lógica de programación mediante marcadores o sintaxis específica para ser completados por el motor de plantillas.

velocidad y confianza. Más detalles sobre sus características pueden encontrarse en su sitio oficial, donde se explica su filosofía, ventajas y casos de uso reales.

2- ¿Qué es el patrón MTV (Model-Template-View) en django? (simplificado de MVC). Compará MTV con MVC.

Django se basa en un enfoque estructurado para organizar el código de una aplicación web, utilizando un patrón de diseño llamado MTV, que significa Model-Template-View. Este patrón es una adaptación propia del ya conocido MVC, sigla de Model-View-Controller, ampliamente utilizado en el desarrollo de software para separar la lógica de una aplicación en tres partes principales: los datos, la interfaz y el control. Comprender cómo funciona MVC permite entender mejor la propuesta de Django y por qué su modelo MTV representa una versión simplificada y especializada para el desarrollo web.

En el patrón clásico MVC, el Modelo representa la estructura de los datos y la lógica que permite acceder a ellos. Se encarga de definir cómo se guardan, se consultan y se relacionan los datos en una aplicación. La Vista, por otro lado, es el componente que muestra esos datos al usuario, es decir, la interfaz visible, habitualmente construida en HTML y otros lenguajes de presentación. Por último, el Controlador es el encargado de manejar la lógica entre ambas partes: recibe las acciones del usuario (como clics o envíos de formularios), decide cómo procesarlas, accede o modifica datos a través del modelo, y luego determina qué vista debe ser mostrada como respuesta.

Este enfoque permite una separación clara de responsabilidades. Gracias a ello, un desarrollador puede modificar la interfaz visual sin tocar la lógica interna, o cambiar cómo se almacenan los datos sin necesidad de alterar las vistas. Esta división también facilita el trabajo en equipo, permitiendo que distintas personas trabajen sobre distintas capas sin interferirse.

Ahora bien, Django no utiliza exactamente este patrón. Aunque comparte la misma idea de dividir responsabilidades, redefine los nombres y funciones de los componentes para adaptarlos mejor al entorno web y a su arquitectura interna. Así es como surge el patrón MTV, una interpretación específica que mantiene la esencia de MVC pero redistribuye los roles de forma distinta.

En Django, el Modelo sigue cumpliendo el mismo propósito que en MVC: representa la estructura de datos y gestiona el acceso a la base de datos mediante un sistema ORM que simplifica las consultas y relaciones. La Vista, sin embargo, no es la parte visual como en MVC, sino una función de Python que recibe la petición del usuario, interactúa con los modelos para obtener o modificar información, y luego entrega una respuesta. Finalmente, el Template cumple el rol de la vista tradicional, ya que es el archivo HTML que se le presenta al usuario, con datos dinámicos incluidos gracias a la plantilla.

En la práctica, el flujo de funcionamiento del patrón MTV es el siguiente:

1. El usuario hace una petición al servidor, por ejemplo, accediendo a una URL.
2. Django dirige esa petición a una vista, que es una función (o clase) definida en Python.
3. La vista consulta o modifica datos a través del modelo si es necesario.
4. Luego, la vista selecciona una plantilla y le pasa los datos que quiere mostrar.

5. El sistema renderiza esa plantilla con la información dinámica y se genera una página HTML como respuesta al usuario.

Este sistema es más simple que MVC porque Django actúa internamente como el controlador, eliminando la necesidad de que el desarrollador lo implemente directamente. Según se explica en GeeksforGeeks, el patrón MVT "elimina la necesidad de un controlador explícito, ya que el marco de trabajo Django maneja esa parte por medio de su sistema de vistas". Es decir, lo que en otros entornos se define como "controlador", Django lo resuelve con su propio motor de enrutamiento y sus vistas, lo que simplifica mucho el código sin perder control ni flexibilidad.

3- ¿Qué entendemos por app en django?

En Django, cuando comenzamos a trabajar en un nuevo desarrollo, lo primero que creamos es lo que se conoce como un proyecto. Un proyecto es la estructura base que representa toda la aplicación web en su conjunto. Sin embargo, lo interesante de Django es que este proyecto está pensado para componerse de múltiples aplicaciones independientes, llamadas comúnmente "apps". Este diseño modular es una de las fortalezas más claras del framework, ya que permite dividir grandes desarrollos en componentes más pequeños, reutilizables y fáciles de mantener.

Una app en Django es una unidad funcional del proyecto. Es decir, representa una parte concreta del sitio web, como un sistema de comentarios, un blog, una galería de imágenes, una tienda, un foro o cualquier otra funcionalidad específica. Cada app tiene su propia lógica, modelos, vistas, URLs y plantillas, y puede desarrollarse de forma aislada, incluso con la posibilidad de ser reutilizada en otros proyectos diferentes. Django está especialmente pensado para este tipo de organización, porque fomenta el principio de separación de responsabilidades y promueve el desarrollo escalable.

La relación entre proyecto y apps es similar a la de un contenedor y sus componentes. El proyecto agrupa y coordina el funcionamiento de las distintas apps, mientras que cada app se ocupa de una tarea concreta. Por ejemplo, si estuviéramos desarrollando una red social, podríamos tener una app para los perfiles de usuario, otra para el sistema de mensajes, otra para publicaciones y otra para notificaciones. Cada una de esas partes estaría encapsulada en su propia app, sin interferir directamente con las demás, lo que permite trabajar de forma ordenada y hacer cambios en una sin afectar a las otras.

Cuando ejecutamos el comando django-admin startproject, estamos creando el proyecto principal, que incluye archivos como settings.py para la configuración general, urls.py para las rutas principales, y wsgi.py o asgi.py para la comunicación con el servidor. Luego, mediante el comando python manage.py startapp, creamos una nueva app dentro del proyecto. Esa app tendrá su propio archivo models.py para definir los datos, views.py para responder a las peticiones del usuario, urls.py para definir sus rutas internas, y un directorio de plantillas para mostrar el contenido al usuario.

En la documentación oficial de Django, se explica que una app es "un conjunto de archivos de configuración y código de Python que trabaja conjuntamente para proporcionar alguna funcionalidad". Y lo más importante es que las apps están diseñadas para ser plug-and-play, es decir, se pueden activar, desactivar o reutilizar fácilmente según se necesite. Django permite registrar estas apps dentro del archivo INSTALLED_APPS en la configuración del proyecto, y a partir de ahí, las integra automáticamente en el sistema.

Esta estructura modular tiene muchas ventajas. Por un lado, permite que equipos de desarrollo puedan trabajar en diferentes partes de un mismo proyecto sin pisarse el código. Por otro lado, permite escalar fácilmente: si una app deja de ser necesaria o debe reemplazarse, puede eliminarse o sustituirse sin afectar la totalidad del sistema. Además, si una app está bien diseñada, puede empaquetarse y usarse en otros proyectos, algo que muchas veces ocurre en bibliotecas de terceros.

En resumen, una app en Django es una parte independiente y funcional de un sitio web que vive dentro de un proyecto mayor. Mientras que el proyecto representa la totalidad de la aplicación y coordina su comportamiento, las apps representan componentes específicos que se pueden desarrollar, modificar o incluso reutilizar por separado. Este enfoque hace que Django sea un framework potente, organizado y muy flexible, ideal tanto para proyectos simples como para aplicaciones web complejas y de gran escala.

4- ¿Qué es el flujo *request-response* en django?

En el corazón de cualquier aplicación web construida con Django se encuentra un proceso fundamental conocido como el flujo request-response. Este flujo representa el ciclo de una solicitud (request) realizada por un usuario a un servidor, y la respuesta (response) que el servidor le devuelve. Comprender cómo funciona este mecanismo en Django es esencial para entender el funcionamiento de las vistas, los métodos HTTP y, en última instancia, la lógica de cualquier sitio web dinámico.

Cuando un usuario accede a una página web, lo que realmente está haciendo es enviar una solicitud HTTP al servidor. Esta solicitud llega al proyecto de Django, donde comienza el flujo interno. Django recibe esa solicitud y la encapsula en un objeto llamado HttpRequest, una estructura que contiene toda la información relevante: la URL que el usuario pidió, el tipo de método HTTP que usó (como GET o POST), los datos que pudo haber enviado, las cabeceras, cookies, y más. Este objeto es el punto de partida del proceso y es pasado a la vista correspondiente para que sea procesado. Según la documentación oficial, el objeto HttpRequest incluye atributos como request.method, request.GET, request.POST y request.user, que permiten acceder a los datos según el contexto de la solicitud.

En este contexto, los métodos HTTP definen el propósito de cada solicitud. Aunque existen varios métodos, los más importantes y comunes en Django son GET y POST. El método GET se utiliza para obtener información: por ejemplo, cuando un usuario abre una página, solicita ver una lista de productos, o accede a un formulario vacío. Todos esos casos implican una lectura o visualización de datos, y no modifican nada en el servidor. Por el contrario, el método POST se emplea para enviar información al servidor: como cuando un usuario envía un formulario de contacto, publica un comentario o se registra en una cuenta. En estos casos, se produce un cambio en los datos del servidor, y por eso se usa POST, que protege mejor la información y no la expone en la URL.

Una vez que Django recibe esta solicitud HTTP, se la pasa a una vista. En Django, una vista es una función (o clase) de Python que define qué hacer con una determinada solicitud. Es el lugar donde se decide qué datos buscar, qué lógica ejecutar, y qué plantilla (si es necesario) devolver como respuesta. Por ejemplo, si el usuario entra a una URL como /productos/, Django buscará en el archivo urls.py, qué vista corresponde a esa ruta, y luego ejecutará esa vista, pasando el objeto request como argumento.

Dentro de la vista, el programador puede hacer diferentes cosas según el tipo de solicitud. Un caso muy común es usar una condición para verificar el método, por ejemplo:

```
def contacto(request):
    if request.method == "POST":
        # Procesar los datos del formulario enviado
    else:
        # Mostrar el formulario vacío
```

En este ejemplo, si el usuario accede mediante GET, verá el formulario en pantalla. Si lo envía con POST, la vista procesará los datos. Este tipo de estructuras son fundamentales en Django, ya que las vistas se adaptan dinámicamente según el tipo de método HTTP que reciben, lo que permite tener un único punto de control para múltiples comportamientos.

Finalmente, una vez procesada la solicitud, la vista debe devolver un objeto de tipo HttpResponse, que contiene la respuesta que Django enviará al navegador del usuario. Esta respuesta puede ser una página HTML completa, un fragmento de texto, un archivo JSON, una redirección o incluso un error. El objeto HttpResponse también es personalizable, permitiendo controlar el contenido, las cabeceras, los códigos de estado y más.

Basicamente, el flujo request-response en Django sigue este recorrido básico:

1. El usuario realiza una solicitud HTTP (generalmente GET o POST).
2. Django recibe la solicitud y la convierte en un objeto HttpRequest.
3. Se busca en urls.py qué vista corresponde a la URL solicitada.
4. La vista analiza el método HTTP y actúa en consecuencia (leer, mostrar, guardar, validar, etc.).
5. Finalmente, la vista devuelve un objeto HttpResponse con el contenido que se enviará de vuelta al navegador.

Este ciclo se repite en cada interacción entre el usuario y el servidor, y es la base sobre la que se construyen todas las funcionalidades web: desde mostrar una simple página hasta procesar formularios, autenticar usuarios o enviar datos desde formularios avanzados. Entender este flujo permite dominar no solo el uso de las vistas en Django, sino también cómo se relacionan con los métodos HTTP y cómo controlar el comportamiento completo de una aplicación.

5- ¿Qué es el concepto de ORM (Object-Relational Mapping)? (Informacion de ChatGPT)

El término ORM, sigla de *Object-Relational Mapping* o *Mapeo Objeto-Relacional*, se refiere a una técnica de programación que permite interactuar con bases de datos relacionales utilizando objetos del lenguaje de programación, en lugar de escribir directamente instrucciones en SQL[3]. En el contexto de Django, el ORM es una de las características más poderosas y distintivas del *framework*, ya que simplifica

[3] Lenguaje estándar para gestionar bases de datos relacionales mediante operaciones como consulta, inserción, actualización y eliminación de datos.

radicalmente el manejo de los datos, haciendo que el trabajo con la base de datos sea mucho más natural y fluido para los desarrolladores que programan en Python.

En un enfoque tradicional, cuando un programador quiere insertar, consultar o modificar datos en una base de datos como PostgreSQL, MySQL o SQLite, debe escribir consultas SQL explícitas. Esto implica conocer la sintaxis del lenguaje, preocuparse por las relaciones entre tablas, manejar claves foráneas manualmente, validar tipos de datos, y mucho más. Con un ORM como el de Django, todo ese trabajo puede hacerse utilizando clases de Python que representan modelos de datos. De esta manera, se crea una especie de "puente" entre el mundo orientado a objetos de la programación y el mundo relacional de las bases de datos.

En Django, el ORM funciona de manera transparente: cada modelo que se define en el archivo models.py corresponde a una tabla en la base de datos. Cada atributo de la clase es una columna, y cada instancia de esa clase es un registro (o fila). Por ejemplo:

```
from django.db import models


class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=8, decimal_places=2)
    en_stock = models.BooleanField(default=True)
```

Este simple modelo Producto define una tabla en la base de datos con tres columnas: nombre, precio y en_stock. Una vez creado este modelo, Django se encarga de traducirlo a SQL, crear la tabla correspondiente, y permitir consultas sin que el programador tenga que escribir una sola línea de SQL manual.

Gracias al ORM, se pueden realizar operaciones complejas con solo unas pocas líneas de código en Python. Por ejemplo, si queremos obtener todos los productos disponibles en stock, bastaría con escribir:

```
productos_disponibles = Producto.objects.filter(en_stock=True)
```

Internamente, Django traduce esta instrucción a una consulta SQL como:

```
SELECT * FROM producto WHERE en_stock = TRUE;
```

Pero el programador nunca tiene que ver ni escribir esta consulta. Esto no solo ahorra tiempo, sino que también mejora la legibilidad del código, reduce errores comunes y mejora la seguridad frente a amenazas como la inyección SQL, ya que Django escapa automáticamente los valores peligrosos.

Otra ventaja del ORM es que también permite establecer relaciones entre tablas de forma natural, mediante campos como ForeignKey, ManyToManyField o OneToOneField, haciendo posible modelar sistemas complejos sin perder la claridad del código orientado a objetos.

Además, el ORM es capaz de adaptarse a distintos motores de base de datos sin que el programador tenga que cambiar el código de sus modelos. Por ejemplo, un mismo modelo funcionará tanto en SQLite como en PostgreSQL, solo modificando un parámetro en el archivo de configuración del proyecto. Esta

independencia del motor de base de datos es especialmente útil para proyectos que necesitan escalar o migrar en el futuro.

En sintesis, el ORM de Django actúa como un traductor entre el lenguaje Python y el lenguaje SQL. Permite trabajar con datos como si fueran objetos, eliminando la necesidad de escribir consultas SQL manuales, simplificando la creación y mantenimiento de bases de datos, y mejorando la eficiencia y seguridad del desarrollo. Su integración profunda con el *framework* es una de las razones por las que Django permite construir aplicaciones web complejas en muy poco tiempo y con menos errores.

6- ¿Qué son los *templates* en django?

En Django, los templates son un componente esencial del sistema de generación de contenido dinámico. Su función principal es definir la estructura visual de las páginas web que serán enviadas como respuesta al navegador del usuario. Son archivos —generalmente con extensión .html— que permiten combinar código HTML tradicional con una sintaxis especial de Django, conocida como el lenguaje de plantillas, para incrustar datos dinámicos dentro de esas páginas. Así, los templates permiten mostrar al consumidor final no solo contenido estático, sino información actualizada, personalizada y generada en tiempo real según cada situación.

Los templates están profundamente conectados con la experiencia del usuario, ya que son el punto final del flujo de datos: después de que la vista procesa una solicitud y obtiene los datos necesarios del modelo, los pasa a un template para ser presentados visualmente. Lo que el usuario ve en su navegador es el resultado de este trabajo conjunto entre vistas, modelos y plantillas. En este sentido, los templates son la herramienta que Django ofrece para construir la interfaz del usuario y transformar datos en contenido comprensible y navegable.

A diferencia de los archivos HTML tradicionales, los templates de Django no solo contienen etiquetas de HTML puro, sino que además pueden incluir etiquetas y filtros especiales que permiten insertar valores, controlar estructuras lógicas (como bucles y condiciones), extender otras plantillas y reutilizar componentes comunes. Por ejemplo, un template puede incluir este fragmento:

```
<h1>Bienvenido, {{ usuario.nombre }}</h1>
```

En este caso, {{ usuario.nombre }} es una expresión de plantilla que será reemplazada dinámicamente por el nombre del usuario cuando la página se renderice. Este reemplazo lo hace automáticamente Django en el momento de generar la respuesta, gracias a que la vista le proporciona los datos necesarios en un diccionario de contexto.

El funcionamiento general de los templates se da en esta última etapa del flujo request-response. Una vez que una vista ha procesado la solicitud, puede devolver un objeto HttpResponse que contiene directamente una página HTML generada a partir de un template. Esto suele hacerse con la función render(request, template_name, context), donde el template_name es el nombre del archivo .html y el context es el conjunto de datos que se desea mostrar. Por ejemplo:

```
def inicio(request):
    return render(request, 'inicio.html', {'usuario': request.user})
```

En este ejemplo, la plantilla inicio.html podrá acceder al objeto usuario y mostrar su información de forma personalizada. Lo que el navegador finalmente muestra al consumidor es un documento HTML normal, pero generado en tiempo real en el servidor a partir de esa plantilla.

Otra característica destacable del sistema de plantillas de Django es su capacidad de herencia. Gracias a esto, es posible definir una estructura base común (por ejemplo, una cabecera y un pie de página) y luego extenderla en otros templates que solo cambian el contenido específico de cada página. Esto favorece la reutilización del código, la consistencia visual del sitio y la facilidad de mantenimiento.

El uso de templates es vital para que una aplicación web sea verdaderamente dinámica. No solo permiten mostrar información que cambia constantemente —como listas de productos, nombres de usuarios, mensajes personalizados o resultados de búsquedas—, sino que también permiten construir formularios, gestionar errores y diseñar interfaces interactivas. Sin los templates, Django no podría traducir la lógica interna del sistema en una experiencia accesible para el consumidor.

Según la documentación oficial, los templates en Django "están diseñados para ser usados por diseñadores que saben cómo escribir HTML, pero que no quieren o necesitan aprender Python". Esto muestra otro punto fuerte: el sistema está pensado para permitir la colaboración entre programadores y diseñadores sin que unos tengan que depender completamente de los otros. El desarrollador define la lógica y estructura de datos, mientras que el diseñador puede trabajar sobre los templates sabiendo que sus marcas {{ }} y {% %} serán reemplazadas automáticamente por contenido real en tiempo de ejecución.

En definitiva, los templates en Django son archivos HTML enriquecidos con una sintaxis especial, que permiten mostrar datos dinámicos al usuario final. Son el vínculo directo entre la lógica del servidor y la interfaz que percibe el consumidor, y su funcionamiento es clave para que las aplicaciones web no solo sean potentes, sino también accesibles, intuitivas y visualmente coherentes.