# The web framework for perfectionists with deadlines

# Unidad Temática 1

## Introducción a Django

- ¿Qué es Django? Principales características.
- Breve historia y contexto.
- Principios de diseño del framework.
- Filosofía "Baterías incluidas".
- Patrón MVT.
- Ventajas y desventajas de utilizar Django.
- Empresas famosas que utilizan Django.

# Exactamente... ¿qué es un Framework?

Un **framework de desarrollo de software** es un conjunto de herramientas, bibliotecas, y patrones predefinidos que facilitan el desarrollo de aplicaciones de software. Actúa como una base sobre la cual los desarrolladores pueden construir aplicaciones de manera más eficiente, reutilizando código común y siguiendo ciertas reglas y estructuras establecidas.

## Características principales de un framework:

- **Reutilización de código:** Proporciona funciones y módulos reutilizables que ahorran tiempo al evitar la necesidad de escribir código desde cero.
- **Estructura:** Define una arquitectura estándar para el desarrollo, lo que ayuda a mantener la consistencia y la organización del proyecto.
- **Herramientas integradas:** Ofrece herramientas para tareas comunes como manejo de bases de datos, autenticación de usuarios, y gestión de sesiones.
- **Modularidad:** Permite agregar o quitar componentes según las necesidades del proyecto.

## Ejemplos de frameworks populares:

- **Web:** Django, Flask, FastAPI (Python), Laravel (PHP), Ruby on Rails (Ruby), Angular (JavaScript).
- **Aplicaciones móviles:** React Native, Flutter.
- **Desarrollo de juegos:** Unity, Unreal Engine.

Usar un framework permite a los desarrolladores enfocarse en la lógica específica de la aplicación, en lugar de preocuparse por detalles de implementación básicos que ya están resueltos por el framework.

# ¿Qué es Django?

**Django es un framework de desarrollo web** de alto nivel escrito en **Python** que promueve el desarrollo rápido y el diseño limpio y pragmático. Fue diseñado para facilitar la creación y el mantenimiento de aplicaciones web complejas con un enfoque en la reutilización de código y la rapidez en el desarrollo.

## Características principales de Django:

- **Administración automática:** Django incluye un sistema de administración listo para usar que permite gestionar modelos de datos con facilidad, lo que es útil para crear paneles administrativos rápidamente.
- **ORM (Object-Relational Mapping):** Django proporciona un mapeo objeto-relacional que permite a los desarrolladores interactuar con la base de datos usando modelos de Python en lugar de escribir SQL directamente.
- **Seguridad:** Django viene con protecciones integradas contra amenazas comunes de seguridad web, como inyecciones SQL, cross-site scripting (XSS), y cross-site request forgery (CSRF).
- **Escalabilidad:** Es lo suficientemente robusto para manejar aplicaciones a gran escala, pero también es adecuado para proyectos pequeños.
- **Comunidad activa y rica documentación:** Django cuenta con una amplia comunidad de desarrolladores y una excelente documentación, lo que facilita la solución de problemas y la integración de nuevas funcionalidades.

# Historia y contexto de Django

Django fue creado originalmente en el año **2003** por **Adrian Holovaty** y **Simon Willison**, dos desarrolladores web que trabajaban para el periódico **Lawrence Journal World** en Kansas, Estados Unidos. En ese tiempo, estaban encargados de construir varias aplicaciones web para el periódico, como plataformas para gestionar contenidos de noticias y eventos.

Al enfrentar **desafíos comunes** en el desarrollo de aplicaciones web, Holovaty y Willison **comenzaron a crear un conjunto de herramientas y convenciones que les ayudara a desarrollar de manera más rápida y eficiente**. Estas herramientas eventualmente se consolidaron en lo que hoy conocemos como el framework Django, a continuación algunos hitos:

- **2005:** Django (en honor al guitarrista de jazz Django Reinhard) se lanza como un proyecto de código abierto bajo la licencia BSD, lo que permitió que desarrolladores de todo el mundo utilizaran el framework y contribuyeran a su desarrollo.
- **2008:** Django 1.0, primera versión estable. Se forma la Django Software Foundation (DSF), una organización sin fines de lucro encargada de mantener y promover Django, quien garantiza que siga siendo un proyecto comunitario.
- **2010:** Se lanza la versión 1.2 con mejoras significativas como soporte para múltiples bases de datos y mejoras de seguridad.
- **2012:** Se introduce un soporte robusto para archivos estáticos.
- **2013:** Soporte para Python 3 y gestión de transacciones.
- **2014:** Se introducen las Migraciones.
- **2015:** Soporte nativo para múltiples motores de plantillas. Django 1.8 es la primera versión LTS.
- **2017:** La versión 2.0 se convierte en la primera de lanzamiento exclusivo para Python 3.
- **2020:** Se lanza Django 3.0 con soporte ASGI (Asynchronous Server Gateway Interface) permitiendo la gestión de peticiones asincrónicas mejorando el rendimiento y brindando posibilidades de desarrollo en tiempo real.
- **2022:** Django 4.0 introduce mejoras en la API, gestión de formularios y seguridad.
- **2023:** Django 5.0 mejoras en las capacidades asincrónicas, optimización de rendimientoy mejoras en herramientas de seguridad.

# Principios de Diseño

El **Zen de Python** es una colección de 20 principios de software que fueron escritos por Tim Peters en junio de 1999 y que influyen en el diseño del Lenguaje de Programación Python. **Enfatiza la simplicidad, la claridad y la legibilidad en el diseño del código** y muchos de estos principios están reflejados en la filosofía y el diseño de Django.

- **DRY (Don't Repeat Yourself)**
  Django promueve la **reutilización de código para evitar la duplicación**, lo que hace el desarrollo más eficiente y el mantenimiento más sencillo.
  En las aplicaciones, el uso de formularios, modelos y vistas ejemplifica este principio, ya que permite definir una vez la lógica de negocio y reutilizarla en diferentes partes de la aplicación.

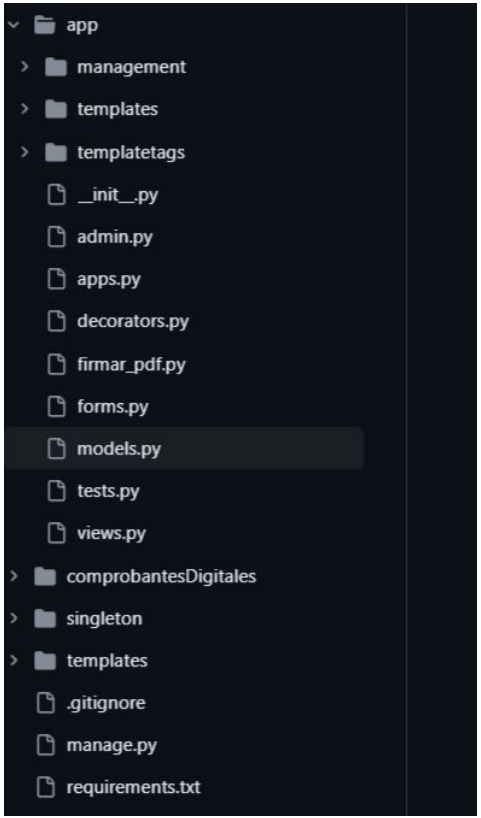
```python
class Empresa(models.Model):
    razon_social = models.CharField(max_length=200, verbose_name="Descripción")
    cuit = models.CharField(max_length=11, verbose_name="CUIT", validators=[validate_cuit], unique=True)
    slug = AutoSlugField(populate_from='razon_social', unique=True, editable=False)
    abrev = models.CharField(max_length=5, verbose_name="Abreviatura sociedad", null=True, blank=True)
```

¿No es fácil? A partir de una declaración como **models.CharField(max_length=n)** se puede generar un input en un formulario HTML, una declaración SQL DDL y validar automáticamente la información para que sólo contenga n caracteres. Esto es exactamente lo que hace el principio de diseño DRY de Django.

# Principios de Diseño

- **Convenciones sobre configuración**
  Django sigue convenciones comunes que **reducen la cantidad de configuración necesaria, lo que permite a los desarrolladores centrarse en la lógica de la aplicación**.

  Por ejemplo, la estructura de proyectos y aplicaciones en Django sigue una convención estándar, lo que facilita a los desarrolladores entender un proyecto nuevo sin necesidad de configuraciones personalizadas extensas.



# Principios de Diseño

- **Componentes desacoplados**
  Django **está diseñado de manera modular**, con componentes que pueden ser usados de forma independiente o sustituidos según las necesidades del proyecto.

  Los componentes de Django, como el ORM, el sistema de plantillas o el sistema de autenticación, **pueden ser usados por separado o incluso reemplazados por otras herramientas**, lo que proporciona flexibilidad en el desarrollo.

  El hecho de que estén acoplados de forma flexible significa que no existen dependencias rígidas entre las partes que conforman una aplicación Django.

```python
@check_recaptcha
def contacto(request, template_name='website/contacto.html'):
    from django.http import JsonResponse
    from .forms import ContactForm

    args = {}
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():  # and request.recaptcha_is_valid:
            form.enviar()
            return JsonResponse({'status': 200, 'response': {'message': 'Mensaje recibido.'}})
        args = {'form': form}
    else:
        args['form'] = ContactForm()

    if request.headers.get('x-requested-with') == 'XMLHttpRequest':
        template_name = "website/contact_form.html"

    return render(request, template_name, args)
```

Por ejemplo, en Django es perfectamente válido servir contenido directamente desde una plantilla HTML, sin necesidad de usar lógica de negocios o configurar una base de datos. Al igual que en Django, también es perfectamente válido renunciar al uso de una plantilla HTML y devolver datos sin procesar directamente desde un método de lógica de negocios (por ejemplo, una respuesta Json).

# Principios de Diseño

- **Menos Código, Mejor Código**
  Django promueve la escritura de código que sea conciso, legible y expresivo. Se prefieren las soluciones que minimizan la complejidad y maximizan la claridad.
  Esto se ve reflejado en la sintaxis de Django, donde se busca que el código sea lo más legible posible, evitando configuraciones innecesarias o redundantes.

- **Seguridad Incorporada**
  Pone un fuerte énfasis en la seguridad desde el diseño. Incluye protecciones integradas contra muchas de las amenazas comunes en aplicaciones web.
  Características como la **protección contra CSRF, el manejo seguro de contraseñas, y las medidas contra XSS son implementadas de manera predeterminada**, reduciendo la probabilidad de errores de seguridad por parte de los desarrolladores.

# Filosofía "Baterías incluidas"

Imaginen que estás creando una aplicación web que necesita autenticación de usuarios, administración de contenido, y manejo de bases de datos…

Con Django, todo esto ya está incluido y configurado, lo que te permite concentrarte en construir las funcionalidades específicas de tu aplicación en lugar de desarrollar, integrar o configurar herramientas externas.

La filosofía de **"baterías incluidas" en Django se refiere a la idea de que el framework proporciona todo lo que necesitas para desarrollar aplicaciones web completas y robustas, sin requerir la instalación de herramientas adicionales o bibliotecas externas desde el principio**. Este enfoque facilita el desarrollo al ofrecer un conjunto completo de funcionalidades integradas, lo que permite a los desarrolladores enfocarse en la lógica específica de su aplicación sin preocuparse por configurar o buscar componentes básicos.

# Filosofía "Baterías incluidas"

## Aspectos Clave de la Filosofía "Baterías Incluidas" en Django:

1. **Herramientas Integradas:** incluye un conjunto extenso de herramientas listas para usar, como el ORM (Object-Relational Mapping) para manejar bases de datos, un sistema de administración de usuarios, formularios, sistemas de autenticación, validación de datos, gestión de archivos estáticos, etc. Estas herramientas vienen preconfiguradas y se integran entre sí.
2. **Sistema de Administración Automático:** proporciona un sistema de administración automática que se genera a partir de los modelos definidos en tu proyecto. Esto significa que puedes tener un panel de administración funcional con muy poco esfuerzo adicional, lo que es ideal para gestionar el contenido de la aplicación.
3. **Seguridad Incorporada:** incluye protecciones integradas contra muchas de las amenazas comunes en el desarrollo web, como inyecciones SQL, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery), etc. Están activadas por defecto, lo que reduce la probabilidad de vulnerabilidades en las aplicaciones.
4. **Sistema de Plantillas:** tiene un sistema de plantillas poderoso y flexible que permite la separación clara entre la lógica de la aplicación y la presentación. Este sistema soporta herencia de plantillas, bloques, y filtros, facilitando el desarrollo de interfaces de usuario complejas.
5. **Enrutamiento de URLs:** el enrutamiento de URLs es sencillo pero poderoso, permitiendo mapear URLs a vistas con facilidad. Es fácil de usar y suficientemente flexible para manejar patrones de URL complejos.
6. **Internacionalización y Localización:** incluye soporte para la internacionalización (i18n) y localización (l10n) de manera integrada, lo que facilita la creación de aplicaciones multilingües y adaptadas a diferentes culturas.
7. **Extensa Documentación y Comunidad Activa:** la documentación de Django es completa y está bien organizada, cubriendo tanto las características básicas como las avanzadas del framework. Además, la comunidad activa proporciona soporte adicional y una gran cantidad de recursos en forma de paquetes, tutoriales y foros.

# Filosofía "Baterías incluidas"

## Ventajas de la Filosofía "Baterías Incluidas":

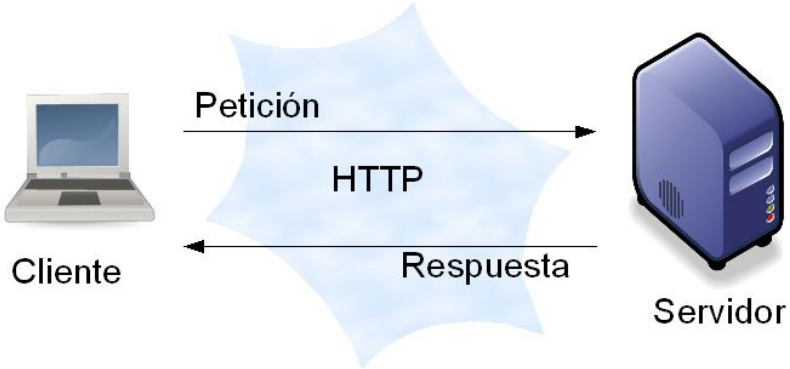
- **Productividad:** Al tener todas las herramientas necesarias integradas, los desarrolladores pueden empezar a trabajar inmediatamente sin perder tiempo en configurar componentes básicos.
- **Consistencia:** Usar herramientas integradas asegura que todos los componentes trabajen bien juntos, reduciendo problemas de incompatibilidad.
- **Simplicidad:** Reduce la curva de aprendizaje al no tener que aprender múltiples bibliotecas o herramientas de terceros.

# Patrón MVT

En un sitio web tradicional, una aplicación web espera peticiones HTTP del explorador web (o de otro cliente). Cuando se recibe una petición la aplicación elabora lo que se necesita basándose en la URL y posiblemente en la información incluida en los datos POST o GET.

Dependiendo de qué se necesita quizás pueda entonces leer o escribir información desde una base de datos o realizar otras tareas requeridas para satisfacer la petición.

La aplicación devolverá a continuación una respuesta al explorador web, con frecuencia creando dinámicamente una página HTML para que el explorador la presente insertando los datos recuperados en marcadores de posición dentro de una plantilla HTML.

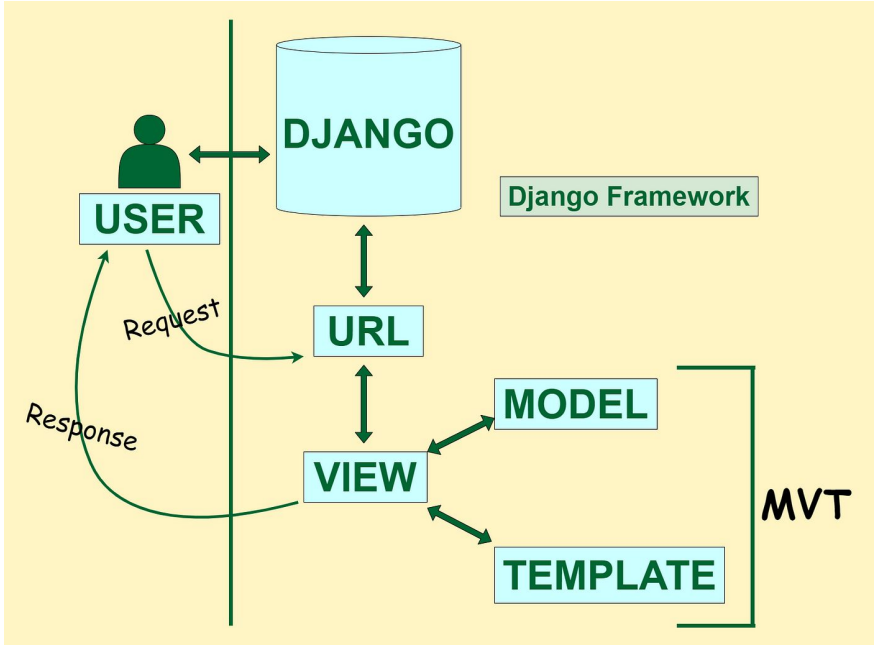


# Patrón MVT

Django sigue un patrón de arquitectura llamado **MVT (Model-View-Template)**, que es una variación del conocido patrón MVC (Model-View-Controller) utilizado en el desarrollo de software. Aunque MVT y MVC son similares en concepto, Django organiza las responsabilidades de manera ligeramente diferente, lo que se adapta mejor a las necesidades del desarrollo web.

**Flujo en MVT:**

1. Un usuario visita una URL en la aplicación Django.
2. Django usa el sistema de enrutamiento de URLs para dirigir la solicitud a la vista correspondiente.
3. La vista maneja la lógica de negocio, como consultar la base de datos utilizando modelos.
4. La vista pasa los datos obtenidos a una plantilla.
5. La plantilla genera la página HTML final, que se envía de vuelta al navegador del usuario.

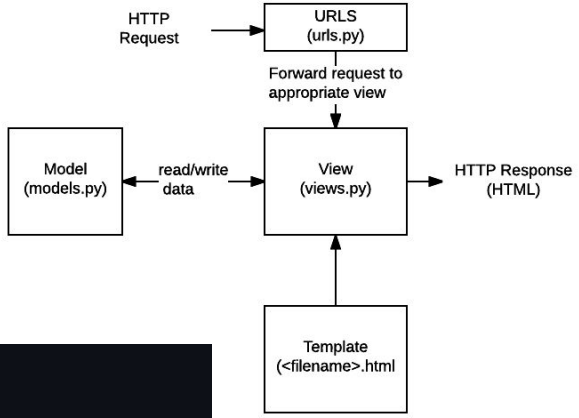


# Componentes del patrón MVT en Django

## URLs

Se usa un mapeador URL para redirigir las peticiones HTTP a la vista apropiada basándose en la URL de la petición

El mapeador también puede emparejar patrones de cadenas o dígitos específicos que aparecen en una URL y los pasan a la función de visualización como datos.



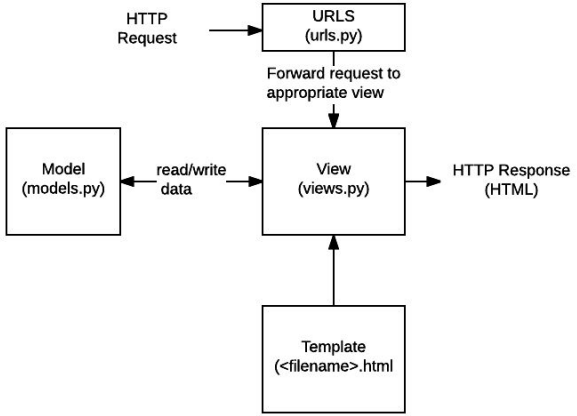

```
urlpatterns = [

    path("ckeditor5/", include('django_ckeditor_5.urls')),

    path('admin/', admin.site.urls),

    path('', home, name='home'),

    path('home-video', TemplateView.as_view(template_name='website/homeVideo.html')),

    path('edificios', TemplateView.as_view(template_name="website/edificios.html"), name='proyectos'),

    path('edificio-<slug:slug>', desarrollo, name='desarrollo'),

    path('edificio-<slug:slug>', desarrollo, name='desarrollo'),
```

# Componentes del patrón MVT en Django

**Vista** **(View)**

Una vista es una función de gestión de peticiones que recibe peticiones HTTP y devuelve respuestas HTTP.

Las vistas acceden a los datos que necesitan para satisfacer las peticiones por medio de modelos, y delegan el formateo de la respuesta a las plantillas o templates.

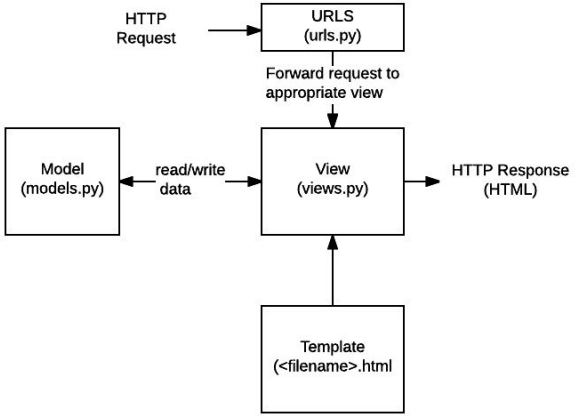



```
def home(request):
    if Parametrizacion.load().comming_soon:
        return render(request, "comming_soon.html")

    args = {
        'slider_home': Desarrollo.objects.filter(in_slider_home=True, publicado=True).order_by('orden_slider_home'),
    }
    return render(request, "website/index.html", args)
```

# Componentes del patrón MVT en Django

## Modelos (Models)

Los modelos en Django representan la estructura y lógica de la base de datos. Cada modelo generalmente corresponde a una tabla en la base de datos, y cada instancia de un modelo es una fila en esa tabla. Los modelos definen los campos y comportamientos de los datos que se almacenan.

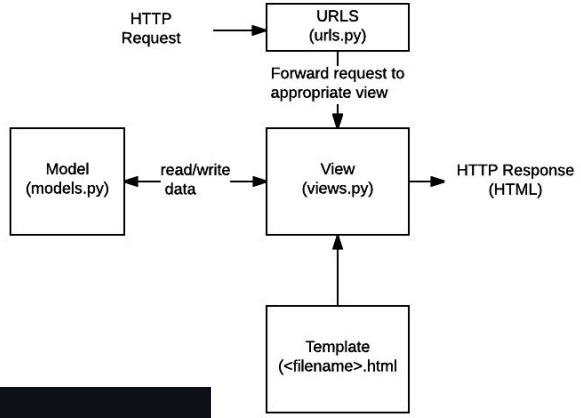



```python
class Barrio(models.Model):
    descripcion = models.CharField(max_length=255, unique=True)

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

# Componentes del patrón MVT en Django

## Plantillas (Templates)

Una plantilla o template es un archivo de texto que define la estructura o diagrama de otro fichero (tal como una página HTML), con marcadores de posición que se utilizan para representar el contenido real. Una vista puede crear dinámicamente una página usando una plantilla, rellenandola con datos de un modelo. Una plantilla se puede usar para definir la estructura de cualquier tipo de fichero; ¡no tiene porqué ser HTML!




```
{% extends "website/index.html" %}

{% load website %}

{% block title %}Próximamente{% endblock %}

{% block extra_styles_header %}{% endblock %}

{% block body %} class="soon-page"{% endblock %}

{% block full_page %}
    <div class="pageWrapper animsition">
        <!-- Content Start -->
        <div id="contentWrapper">
            <div class="container">
                <div class="row">
                    <div class="clearfix over-hidden">
                        <!-- Logo start -->
                        <div class="logo soon-logo shape">
                            <a href="{% url 'home' %}"><img alt="" src="{% get_param_img 'img_logo_header' %}"></a>
                        </div>
```

# Ventajas del Patrón MVT

- **Separación de responsabilidades**
  Cada componente tiene una responsabilidad clara, lo que facilita el mantenimiento y la escalabilidad.

- **Reutilización de código**
  Los modelos, vistas y plantillas pueden ser reutilizados en diferentes partes de la aplicación.

- **Facilidad de desarrollo**
  Django proporciona muchas herramientas y convenciones que hacen que trabajar con el patrón MVT sea sencillo y eficiente.

Sintetizando… el patrón MVT de Django estructura el desarrollo de aplicaciones web de manera que separe claramente la lógica de negocio, la interacción con la base de datos, y la presentación, permitiendo un desarrollo más organizado y manejable.

# Ventajas de Utilizar Django

- "Baterías Incluidas"
- Potente ORM (Object-Relational Mapping)
- Seguridad
- Rapidez en el desarrollo
- Documentación extensa, ecosistema y comunidad activa
- Escalabilidad

# Desventajas de Utilizar Django

- **Curva de Aprendizaje**
  Aunque Django está muy bien documentado tiene una curva de aprendizaje que puede ser empinada, especialmente para aquellos que son nuevos en Python o en el desarrollo web en general.
- **Peso y Complejidad**
  Es un framework "pesado" que incluye muchas funcionalidades integradas. Para aplicaciones simples, esto puede ser excesivo.
- **Opinión Fuerte (dogmático / no dogmático)**
  Aunque Django es dogmático pero no del todo, sigue convenciones y patrones muy específicos. Esto puede limitar la flexibilidad si los desarrolladores desean trabajar fuera de estas convenciones.
- **Monoliticidad**
  Django es tradicionalmente visto como un framework monolítico, lo que significa que tiende a centralizar la lógica de la aplicación en un solo lugar por lo puede no ser la mejor opción para arquitecturas de microservicios o para aplicaciones que necesitan ser extremadamente modulares y distribuidas.
- **Rendimiento**
  Django no está optimizado para casos de uso extremadamente demandantes en términos de rendimiento, como aplicaciones en tiempo real o de procesamiento de datos masivos.
- **Compatibilidad con Bases de Datos No Relacionales**
  Aunque es posible usar bases de datos no relacionales con Django, su ORM está optimizado para bases de datos relacionales por lo que si la aplicación depende de bases de datos no relacionales, puede ser más complicado integrarlas con Django de manera eficiente.

Como vimos, Django es un framework robusto y flexible que es ideal para el desarrollo rápido de aplicaciones web complejas y escalables. Sin embargo, su complejidad inherente y el enfoque "todo en uno" pueden ser excesivos para proyectos más simples o específicos. Es importante evaluar las necesidades de tu proyecto antes de decidir si Django es la mejor opción.

# Empresas Famosas que Utilizan Django


Utiliza Django en su backend. Inicialmente permitió a Instagram escalar rápidamente a medida que su base de usuarios crecía de manera exponencial. La capacidad de Django para manejar un crecimiento rápido y su flexibilidad hicieron que fuera una elección ideal para Instagram, especialmente en sus primeros años.

Utiliza Django para gestionar varias partes de su aplicación web. La combinación de escalabilidad y rapidez en el desarrollo permitió a Spotify implementar nuevas funcionalidades y gestionar grandes volúmenes de datos y usuarios.



Empleó Django para la construcción de su web porque ofrecía la rapidez y facilidad de desarrollo necesarias para construir y mantener una plataforma que crecería rápidamente.

La flexibilidad de Django para manejar grandes cantidades de contenido y su capacidad para integrarse con otras tecnologías hizo que fuera una elección adecuada para un medio de comunicación de gran escala.

# Empresas Famosas que Utilizan Django


Mozilla, la organización detrás del navegador Firefox
Ha utilizado Django para desarrollar varios de sus servicios web. Django proporcionó a Mozilla la capacidad de desarrollar aplicaciones web seguras y eficientes, además de manejar grandes volúmenes de tráfico.

Utiliza Django para gestionar su sitio web y contenido multimedia por sus herramientas para manejar contenido dinámico y multimedia de manera eficiente.



Es una plataforma para la gestión y venta de entradas para eventos, la capacidad de Django para manejar transacciones y gestionar bases de datos complejas fue clave para Eventbrite.

Disqus es una popular plataforma de comentarios que se integra en sitios web y blogs, fue originalmente construida con Django, este permitió a Disqus manejar eficazmente una gran cantidad de interacciones de usuarios y contenido generado por usuarios.

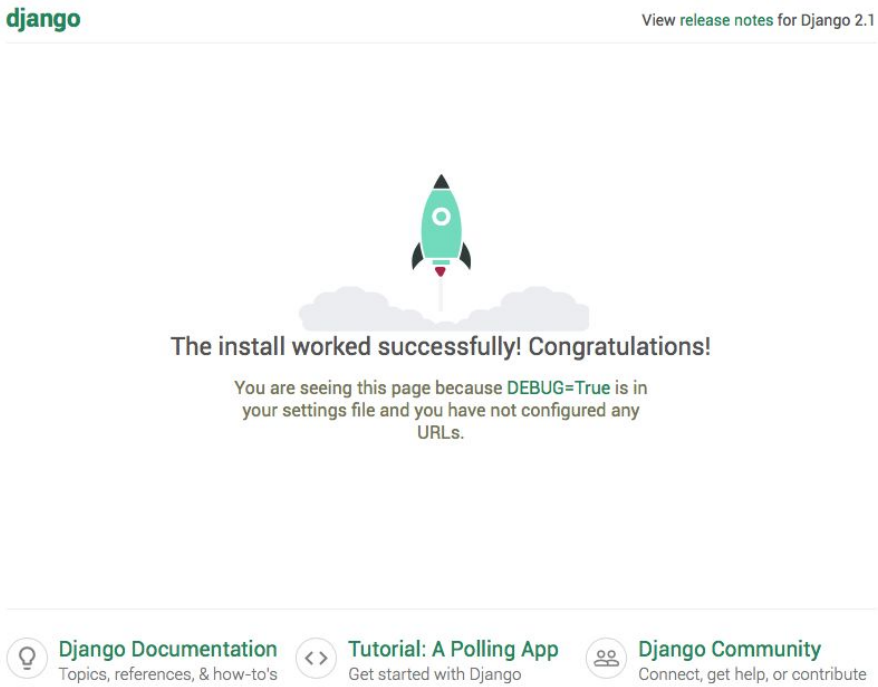
# Bibliografía


- https://www.djangoproject.com/
- https://developer.mozilla.org/es/docs/Learn/Server-side/Django
- https://www.webforefront.com/django

# Unidad Temática 2

## Escribiendo tu primera aplicación

- Versiones de Djang
- Instalación y configuración de Python y Djang
- Creación de un proyecto y aplicacione
  Estructura de directorios y archivo
- Servidor de desarroll
- Comandos básico
- Buenas prácticas.



# Versiones de Django

Django, desde su lanzamiento, ha tenido varias versiones que han introducido nuevas funcionalidades, mejoras de rendimiento y correcciones de seguridad.

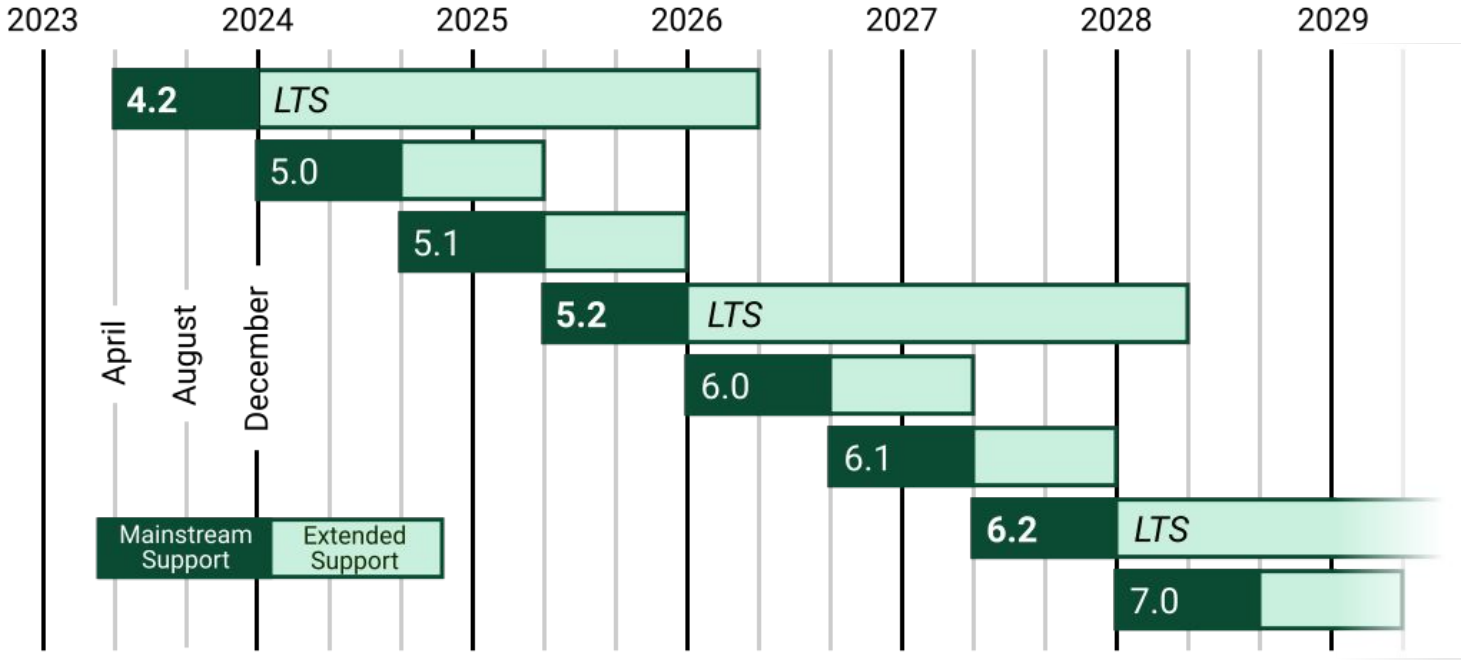
Se organizan de la siguiente manera:

1. **Versiones Principales (Major Releases)** 1.0, 2.0, 3.0, 4.0, 5.0, y futuras.
   Suelen incluir cambios significativos, nuevas características y, en algunos casos, la eliminación de funciones obsoletas.

2. **Versiones Menores (Minor Releases)** 3.1, 3.2, 4.1, etc.
   Generalmente incluyen nuevas funcionalidades, mejoras en las características existentes y pueden introducir algunos cambios menores en la API.

3. **Versiones de Parche (Patch Releases)** 3.2.1, 3.2.2, etc.
   Se centran en corregir errores y problemas de seguridad sin introducir nuevas funcionalidades o cambios importantes.

# Versiones de Django




https://www.djangoproject.com/download/

# Instalación de Python.

Al ser un framework web Python, Django requiere Python.

Para obtener más información consulta ¿Qué versión de Python puedo usar con Django?

Python incluye una base de datos liviana llamada SQLite, por lo que no necesitarás configurar una base de datos por lo menos por ahora.

Obtené la última versión de Python en https://www.python.org/downloads/ o con el administrador de paquetes de su sistema operativo.

Podés verificar si Python está instalado escribiendo python desde tu shell, verás ver algo como:

```
Python 3.x.y
[GCC 4.x] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

https://docs.djangoproject.com/en/5.1/intro/install/

# Recomendación sobre Python

Al desarrollar software con Python, un enfoque básico es instalar Python en tu máquina todas las bibliotecas necesarias a través de la terminal, escribís todo tu código en uno o varios archivos *.py y ejecutás tu programa Python en la terminal.

Este es un enfoque común para principiantes y muchas personas que están transicionando a trabajar con Python para análisis de datos.

Esto funciona bien para proyectos de programación de Python simples. Pero en proyectos de desarrollo de software complejos, como la creación de una biblioteca de Python, una API o un kit de desarrollo de software, a menudo trabajará con varios archivos, varios paquetes y dependencias. Como resultado, necesitará aislar tu entorno de desarrollo de Python para ese proyecto en particular.

***"Un entorno virtual es un entorno Python tal que el intérprete Python, las bibliotecas y los scripts instalados en él están aislados de los instalados en otros entornos virtuales y (de manera predeterminada) de cualquier biblioteca instalada en un "sistema" Python, es decir, uno que esté instalado como parte de su sistema operativo"***

https://www.freecodecamp.org/news/how-to-setup-virtual-environments-in-python/

# Uso de los Python Virtual Environments

Hay varias formas de crearlos, a través del módulo venv, el paquete virtualenv o los gestores de venv de los IDEs

***python3 -m venv /path_al_nuevo_virtual_environment/***

Para activarlo:
***source /path_al_nuevo_virtual_environment/bin/activate***

Para instalar un paquete o librería:
***pip install requests==2.25.1***
***python3 pip install requests==2.25.1 para Windows***

Para desactivarlo o remover el virtual environment:
***deactivate***

ATENCIÓN: Chequeá la sintaxis adecuada para tu sistema operativo.

https://docs.python.org/es/3/library/venv.html

# Instalación de Django

Hay tres opciones para instalar Django:

- Instalar una versión oficial: la mejor opción para la mayoría de los usuarios.
- Instalar una versión de Django proporcionada por la distribución de tu sistema operativo.
- Instalar la última versión de desarrollo: para los entusiastas que les gusta la adrenalina, quienes quieren las últimas y mejores funciones y no tienen miedo de ejecutar código completamente nuevo. Es posible encontrar errores en la versión de desarrollo pero informarlos ayuda al desarrollo de Django.

Atención: es muy probable que las versiones de paquetes de terceros sean más incompatibles con la versión de desarrollo que con la última versión estable.

```
>>> import django
>>> print(django.get_version())
5.1
```

https://docs.djangoproject.com/en/5.1/intro/install/

# Creando un Proyecto

Un proyecto Django consiste en una colección de configuraciones para una instancia de Django, incluida la configuración de la base de datos, las opciones específicas de Django y las configuraciones específicas de la (o las) aplicaciones.

Lo bueno es que esto se genera automáticamente…

***django-admin startproject nombre_del_proyecto***

¡Cuidado! **evitá nombrar los proyectos con nombres de componentes integrados de Python o Django**.

Por ejemplo: **django** (entraría en conflicto con el propio Django) o **test** (que entra en conflicto con un paquete integrado de Python).

https://docs.djangoproject.com/en/5.1/intro/tutorial01/

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        asgi.py
        wsgi.py
```

# Creando las Aplicaciones

Cada aplicación que escribís en Django es un paquete Python que sigue una convención determinada.

Nuevamente Django incluye una utilidad que genera automáticamente la estructura básica de directorios de una aplicación, de modo que puedas concentrarte en escribir código en lugar de crear archivos y directorios.

***django-admin startapp nombre_de_la_aplicación***

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

Proyectos vs. aplicaciones

¿Cuál es la diferencia entre un proyecto y una aplicación? Una aplicación es una aplicación web que hace algo, por ejemplo, un sistema de blog, una base de datos de registros públicos o una pequeña aplicación de encuestas. Un proyecto es una colección de configuraciones y aplicaciones para un sitio web en particular. Un proyecto puede contener varias aplicaciones. Una aplicación puede estar en varios proyectos..

https://docs.djangoproject.com/en/5.1/intro/tutorial01/

# Servidor de Desarrollo

Django incorpora un servidor web liviano escrito en Python.

La idea es que puedas desarrollar cosas rápidamente, sin tener que lidiar con la configuración de un servidor de producción (como Apache) hasta que tu proyecto esté listo para pasar a producción.

***python3 manage.py runserver*** ←Servidor en localhost

***python3 manage.py runserver 192.168.1.10:80*** ←IP de tu equipo donde desarrollás

Ahora es un buen momento para tener en cuenta lo siguiente: **no utilices este servidor en nada que se parezca a un entorno de producción**. Está destinado únicamente a utilizarse durante el desarrollo. (Nos dedicamos a crear frameworks, no servidores web - chiste de algún documentador).

https://docs.djangoproject.com/en/5.1/intro/tutorial01/#the-development-server

# Comandos Básicos

Django incorpora una serie de comandos básicos, los más utilizados son:

- *python3 manage.py runserver*
- *python3 manage.py createsuperuser*
- *python3 manage.py makemigrations*
- *python3 manage.py migrate*
- *python3 manage.py changepassword*

https://docs.djangoproject.com/en/5.1/ref/django-admin/

# Buenas Prácticas

Crear proyectos y aplicaciones en Python de manera efectiva **implica seguir una serie de buenas prácticas que mejoran la mantenibilidad, escalabilidad y calidad del código**.

- Estructura del Proyecto
- Documentación
  README.md en el root de tu proyecto que describa el propósito del proyecto, cómo configurarlo, y cómo ejecutarlo.
- Docstrings
  Utilizá docstrings en tus funciones, clases y módulos para describir su propósito y cómo deben ser usados. Escribí comentarios claros y útiles, evitando comentarios innecesarios que solo expliquen lo obvio.
- Control de versiones
- Entornos Virtuales

# Buenas Prácticas

- Gestión de Dependencias
  Usá pip para instalar y gestionar dependencias, asegurate de que requirements.txt esté actualizado.
- PEP 8: Estilo de Código

Tenemos muchas más Buenas Prácticas que iremos viendo durante el seminario.

# Unidad Temática 3

## Modelos y bases de datos

- Motores de DB.
- De la lógica de negocio a la definición de modelos y relaciones a través del ORM.
- Migración y administración de las migraciones.
- Interfaz de administración integrada.
- Creación de objetos y consultas.

# Motores de DB

Django soporta múltiples bases de datos, entre las cuales se incluyen:

- **SQLite:** base de datos predeterminada en Django, ideal para desarrollo y pruebas.
- **PostgreSQL:** recomendadas para producción debido a su robustez y características avanzadas.
- **MySQL:** muy popular en la web, especialmente para aplicaciones que requieren alta escalabilidad.
- **Oracle:** aunque está soportada, es menos común en proyectos Django.

- CockroachDB, Firebird, Google Cloud Spanner, Microsoft SQL Server, Snowflake, TiDB, YugabyteDB: a través de controladores adicionales

https://docs.djangoproject.com/en/5.0/ref/databases/

# Motores de DB

La configuración de la base de datos se realiza en el archivo *settings.py* del proyecto.

Por ejemplo, una configuración básica para MySQL podría verse así:

```
# settings.py

DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.mysql",
        "NAME": "nombre_db",
        "USER": "user",
        "PASSWORD": "password",
        "HOST": "127.0.0.1",
        "PORT": "3306",
    }
}
```

NOTA: Con SQLite la migración inicial crea la DB. Si usás otro motor debés crear la DB a través del DBMS correspondiente.
https://docs.djangoproject.com/en/5.0/ref/databases/

# ORM

Uno de los aspectos más poderosos de Django es **su capacidad para interactuar con bases de datos de manera eficiente a través de su sistema ORM** (Object-Relational Mapping).

¿Qué es un ORM?
Un ORM es una técnica de programación que permite mapear clases y objetos en un lenguaje de programación a tablas en una base de datos relacional. Con **Django, podés interactuar con la base de datos usando objetos Python en lugar de escribir SQL directamente**.

# Modelos

Las entidades de interés de nuestro sistema y las relaciones entre ellas se representan mediante **clases de Python que heredan de django.db.models.Model** y se escriben dentro de los archivos models.py de las distinas aplicaciones.

- **Atributos:** En un modelo de Django, los atributos de la clase corresponden a las columnas de la tabla en la base de datos. Los tipos de datos de estos atributos (como CharField, IntegerField, etc.) definen el tipo de columna en la tabla.

- **Relaciones:** Django maneja las relaciones entre entidades usando campos específicos:
  - ForeignKey: Representa una relación de uno a muchos.
  - ManyToManyField: Representa una relación de muchos a muchos.
  - OneToOneField: Representa una relación de uno a uno.

https://docs.djangoproject.com/en/5.1/topics/db/models/

# Modelos

```python
# app.models.py
class Autor(models.Model):
    # Los autores se identifican por un id secuencial y se registra de ellos una descripción
    # compuesta por su pseudónimo o nombre y apellido.

    # id = models.BigIntegerField(primary_key=True)  <--Django lo hace en forma predeterminada
    nombre = models.CharField(max_length=30, null=True, blank=True)
    apellido = models.CharField(max_length=30, null=True, blank=True)
    pseudonimo = models.CharField(max_length=30, verbose_name='Pseudónimo', null=True, blank=True)
    modificado = models.DateTimeField(auto_now=True)
    creado = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name_plural = 'Autores'
        # db_table = 'autores' por defecto app_name_classname en este caso app_autor

    def __str__(self):
        return self.pseudonimo if self.pseudonimo else "{} {}".format(self.nombre, self.apellido)
```

https://docs.djangoproject.com/en/5.1/topics/db/models/
https://docs.djangoproject.com/en/5.1/ref/models/options/#table-names

# Relaciones

```python
class Genero(models.Model):
    descripcion = models.CharField(max_length=30, unique=True, verbose_name="Descripción")

    class Meta:
        verbose_name = 'Género'  # ← Podemos definir el nombre para mostrar de la clase
        verbose_name_plural = 'Géneros'  # ← Django pluraliza agregando 's' al final del nombre de la clase

    def __str__(self):
        return self.descripcion


class Libro(models.Model):
    # Los libros se identifican mediante su ISBN, tienen un título, cantidad de páginas, uno o más autores y pertenecen
    # a un género literario.
    titulo = models.CharField(max_length=50, verbose_name="Título")
    cantidad = models.IntegerField(default=0)
    genero = models.ForeignKey(Genero, on_delete=models.PROTECT, verbose_name="Género")
    autores = models.ManyToManyField(Autor, verbose_name="Autores", through='LibroAutor')

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

https://docs.djangoproject.com/en/5.0/ref/models/fields/#django.db.models.ForeignKey

# Migraciones

Es la forma que tiene Django de **propagar los cambios que realizás en tus modelos en el esquema de tu DB**. Están diseñadas para ser mayormente automáticas, pero necesitás saber cuándo hacer migraciones, cuándo ejecutarlas, volverlas atrás y los problemas comunes con los que podrías encontrarte.

Recordá estos comandos:

- **makemigrations**: crea nuevas migraciones en función de los cambios de tus modelos.
- **migrate:** responsable de aplicar y desaplicar las migraciones.
- **sqlmigrate**: muestra las declaraciones SQL para una migración.
- **showmigrations**: enumera las migraciones de un proyecto y su estado.

Los archivos de migración de cada app se encuentran en un directorio de “migrations” dentro de esa aplicación y están diseñados para ser enviados y distribuidos como parte de su código base. Debes realizarlos una vez en tu máquina de desarrollo y luego ejecutar las mismas migraciones en las máquinas de tus colegas, tus máquinas de prueba y, finalmente, tus máquinas de producción.

**Nota: personalmente no las versiono, ya que pueden quedar dependencias a módulos de 3eros migrados en mi IDE.**
https://docs.djangoproject.com/en/5.0/topics/migrations/#module-django.db.migrations

# Migraciones

Si usás SQLite en tu entorno de desarrollo, la migración inicial crea la DB.
Si usas otro motor de base de datos tenés que crear la db desde el DBMS correspondiente.

1. python3 manage.py migrate
   La migración inicial crea una serie de tablas en la base de datos que son esenciales para el funcionamiento el sistema de autenticación, administración y otras funcionalidades integradas (apps instaladas en tu settings.py)

2. python3 manage.py makemigrations
   Crea archivos con los cambios en tu aplicación que no se propagaron a la DB

3. python3 manage.py migrate
   A través de instrucciones DDL cambia la estructura de la DB en función de los archivos

En función de la evolución del desarrollo de tus apps, a través de sucesivos migrate/makemigrations los cambios en los modelos se van trasladando a la DB:

0001_initial.py
0002_clase1.py
0003_alter_clase1_atributo5.py

Las migraciones tienen un orden. están relacionadas entre sí y van quedando reflejadas en la tabla *django_migrations*.
También se pueden aplicar individualmente: python3 manage.py migrate 0003_alter_… .py
https://docs.djangoproject.com/en/5.0/topics/migrations/#module-django.db.migrations

# Revertir las migraciones

Las migraciones (casi siempre) pueden revertirse aplicando el número de la migración anterior:

*python manage.py migrate app 0002*

Si querés revertir todas las migraciones aplicadas para una aplicación, utilizá el nombre zero:

*python manage.py migrate app **zero***

Acá viene el "casi siempre":
Una migración es irreversible si contiene operaciones irreversibles e intentar revertir dichas migraciones generará IrreversibleError

Unapplying app.0003_auto...Traceback (most recent call last):
django.db.migrations.exceptions.IrreversibleError: Operation <RunSQL sql='DROP TABLE app_libros'> in app.0003_auto is not reversible

https://docs.djangoproject.com/en/5.0/topics/migrations/#reversing-migrations

# Admin de Django

Django Admin permite gestionar el contenido de tu aplicación web sin necesidad de escribir código adiciona:

- Generación Automática:
  Se genera automáticamente a partir de los modelos definidos en tu proyecto. Solo necesitas registrar un modelo en admin.py de tu aplicación para que aparezca en la interfaz de administración.

- Interfaz Amigable:

- Incluye formularios generados automáticamente, listas con filtros, búsqueda, y paginación.

- Personalización
  Podés personalizar la apariencia y comportamiento de la interfaz de administración utilizando ModelAdmin. Esto incluye cómo se muestran los campos, cómo se filtran y ordenan los registros, entre otras opciones. Se pueden añadir acciones personalizadas, cambiar la disposición de los campos, etc.

- Autenticación y Autorización
  Está protegida por un sistema de autenticación con permisos a nivel de grupo y de usuario pueden ser configurados para restringir o permitir acceso a partes específicas de la administración.

https://docs.djangoproject.com/en/5.0/topics/migrations/#reversing-migrations

# Admin de Django

- Extensibilidad
  Podés extender y mejorar la interfaz de administración mediante plantillas personalizadas, formularios, y vistas.

https://docs.djangoproject.com/en/5.0/topics/migrations/#reversing-migrations

# Creación de objetos y consultas.

Una vez que haya creado tus modelos de datos, Django te proporciona automáticamente una API de abstracción de bases de datos que le permite crear, recuperar, actualizar y eliminar objetos.

Haremos las pruebas en el shell de Django:

*python3 manage.py shell*

Creamos un objeto…

```
from app.models import Genero
genero = Genero(descripcion="SienSia ficción")
genero.save()
```

Actualizamos ese objeto…

```
genero.descripcion="Ciencia ficción"
genero.save()
```

# Creación de objetos y consultas.

Recuperamos un único objeto con **get**

*genero = Genero.objects.**get** (descripcion="Ciencia ficción")*

*genero = Genero.objects.**get** (pk=1)*

Podemos buscar un objeto por criterio que establezca una clave única.

- Si se encuentran varios objetos según ese criterio, ocurrirá un error del tipo MultipleObjectsReturned.
- Si el objeto no se encuentra en función del criterio ingresado ocurrirá un error ObjectDoesNotExist.

# Creación de objetos y consultas.

Recuperamos varios objetos con **filter** o **exclude**:

*from app.models import Autor*
*autores = Autor.objects.**filter**(apellido="González").**exclude**(nombre="Mario")*

*Se obtiene un Queryset que es una colección de objetos de la base de datos.*

Luego podemos por ejemplo:

- Iterar entre ellos a través de un for:
  *[print(autor) for autor in Autor.objects.**all()**]*

- Podemos obtener la primera instancia a través de first.
  *primer_autor = Autor.objects.all().**first()***

- Podemos obtener la última instancia a través de last.
  *ult_autor = Autor.objects.all().**last()***

https://docs.djangoproject.com/en/5.1/topics/db/queries/

# Creación de objetos y consultas.

Lookups

Las búsquedas de campos son la forma de especificar el contenido de una cláusula WHERE de SQL.

Se especifican como argumentos de palabras clave para los QuerySet métodos filter(), exclude()y get().

Los argumentos de palabras clave de las búsquedas básicas tienen el formato campo__tipolookup=valor

```
from datetime import date
from app.models import Bibliotecario
b = Bibliotecacion.objetcs.filter(desde__lte=date(2024, 1, 1))  # le, lte, gt, gte

b = Bibliotecacion.objetcs.filter(nombre__iexact="gonzalo")  # case insensitive

b = Bibliotecacion.objetcs.filter(nombre__exact="Gonzalo") # case sensitive

b = Bibliotecacion.objetcs.filter(nombre__icontains="zalo")  # contiene
```

https://docs.djangoproject.com/en/5.1/topics/db/queries/#field-lookups

# Creación de objetos y consultas.

Lookups que abarcan relaciones

Los argumentos de palabras clave de las búsquedas básicas tienen el formato objeto__campo__tipolookup=valor

*autores = Autor.objects.filter(apellido="Clancy", libro__genero__descripcion="Novela")*

*libros_de_suspenso = Libro.objects.filter(genero__descripcion="suspenso")*

https://docs.djangoproject.com/en/5.1/topics/db/queries/#field-lookups

# Unidad Temática 4

## Vistas y plantillas

- URL dispatcher
- Creación de vistas basadas en funciones.
- Uso de plantillas HTML, contexto y renderizado.

# Unidad Temática 5

## Formularios

- Creación y procesamiento de formularios.
- Manejo de errores.

# Unidad Temática 6

## Sesiones y autenticación

- Framework de sesiones.
- Autenticación y permisos de usuarios.

# Sesiones

**Las sesiones son una forma de almacenar datos específicos del usuario entre diferentes solicitudes HTTP**.

Esto permite a Django asociar información con un usuario específico incluso si el protocolo HTTP es stateless (sin estado).

Entre los usos típicos tenemos:

- Guardar datos de inicio de sesión.
- Manejar datos temporales como el contenido de un carrito de compras.
- Seguir la interacción de los usuarios durante una sesión de navegación.

Las sesiones permiten que los datos persistan entre peticiones.

*https://docs.djangoproject.com/en/5.1/topics/http/sessions/*

# Sesiones

**Configurar sesiones:**

```
# settings.py

INSTALLED_APPS = [
    ...
    'django.contrib.sessions',        # Para manejar sesiones
    'django.contrib.auth',            # Sistema de autenticación
    ...
]

MIDDLEWARE = [
    ...
    'django.contrib.sessions.middleware.SessionMiddleware',      # Middleware de sesiones
   'django.contrib.auth.middleware.AuthenticationMiddleware',# Middleware de autenticación
    ...
]
```

# Sesiones

```
>>> from django.contrib.sessions.models import Session

>>> sesion = Session.objects.get(pk="3vgtozdgiqtvwunh1mcosufg1sw2npsq")

>>> sesion.session_data
'.eJxVjDsOwyAQRO9CHSHMJ0DK9DkDWpYlOIlAMnYV5e6xJRd2N5r3Zr4swDKXsHSawpjYjQ3scuwi4JvqBtIL6rNxbHWexsg3he-080dL9Lnv7umgQC_r2kqQlCKRQlDaayWz0ZRBiAheuJyVFtqZNREYBQ6dlPZqELxPg7XIfn_1HDgD:1sg6aI:TGa3w0uzY0HUPq5LL54CBJ3NA7Vsx2HaNKtS2LSbsrk'

>>> sesion.get_decoded()
{'_auth_user_id': '1', '_auth_user_backend': 'django.contrib.auth.backends.ModelBackend', '_auth_user_hash': '72a2edbee3ca349432f54efa00ba908ff34048508fea53a8c822765ca99d177c'}

>>> sesion.expire_date
datetime.datetime(2024, 9, 2, 17, 54, 26, 31681, tzinfo=datetime.timezone.utc)
```

# Sesiones

**Sesiones que duran todo el tiempo del navegador frente a sesiones persistentes:**

>settings.py

*SESSION_EXPIRE_AT_BROWSER_CLOSE = False* (default)

*SESSION_COOKIE_AGE = 1209600* # Dos semanas (en segundos)

Tiempo de expiración: Podés establecer el tiempo máximo que debe durar una sesión para evitar que se almacenen indefinidamente.

Por defecto las cookies de sesión se almacenarán en los navegadores de los usuarios durante un tiempo de hasta SESSION_COOKIE_AGE. Utilizá esto si no deseás que las personas tengan que iniciar sesión cada vez que abren un navegador.

Si SESSION_EXPIRE_AT_BROWSER_CLOSE = True, Django utilizará cookies de duración limitada del navegador, cookies que expiran tan pronto como el usuario cierra su navegador. Usá esta opción si deseás que las personas tengan que iniciar sesión cada vez que abran un navegador.

*https://docs.djangoproject.com/en/5.1/topics/http/sessions/*

# Sesiones

**Cuando y donde se almacenan:**

De manera predeterminada, Django solo guarda en la base de datos la información de la sesión que ha sido creada o modificada, es decir, si se ha asignado o eliminado alguno de sus valores de diccionario:

```
# Session is modified.
request.session["foo"] = "bar"

# Session is modified.
del request.session["foo"]

# Session is modified.
request.session["foo"] = {}
```

https://docs.djangoproject.com/en/5.1/topics/http/sessions/

# Autenticación y autorización

El sistema de autenticación de Django se encarga tanto de la autenticación como de la autorización. En pocas palabras, **la autenticación verifica que un usuario es quien dice ser** y **la autorización determina qué puede hacer un usuario autenticado**.

Se incluye como un módulo de contribución de Django en **django.contrib.auth**.
De manera predeterminada, la configuración requerida ya está incluida en el settings.py

El sistema de autenticación consta de:

- Usuarios
- Permisos: Indicadores que designan si un usuario puede realizar una determinada tarea.
- Grupos: una forma genérica de aplicar etiquetas y permisos a más de un usuario.
- Un sistema de hash de contraseñas configurable
- Formularios y herramientas de visualización para iniciar sesión de usuarios o restringir contenido
- Un sistema backend conectable

*https://docs.djangoproject.com/en/5.1/topics/auth/*

# Autenticación y autorización

El sistema de autenticación de Django pretende ser muy genérico y no ofrece algunas funciones que se encuentran comúnmente en los sistemas de autenticación web.

Se han implementado soluciones para algunos de estos problemas comunes en paquetes de terceros:

- Comprobación de la solidez de la contraseña
- Limitación de los intentos de inicio de sesión
- Autenticación frente a terceros (OAuth, por ejemplo)

*https://docs.djangoproject.com/en/5.1/topics/auth/*

# Vistas basadas en clases para la autenticación

Django provee una serie de herramientas pre construidas que permiten gestionar fácilmente procesos comunes como el inicio de sesión, cierre de sesión y la gestión de contraseñas.

Estas vistas forman parte del módulo django.contrib.auth, y su uso simplifica la creación de funcionalidades de autenticación.

- LoginView / LogoutView
- PasswordChangeView
- PasswordResetView
- PasswordResetConfirmView / PasswordResetDoneView / PasswordResetCompleteView

*https://docs.djangoproject.com/en/5.1/ref/contrib/admin/#adding-a-password-reset-feature*

# Unidad Temática 7

## Seguridad en Django

- Mecanismos de seguridad.
- Protección XSS.
- Protección CSRF.
- Validación de datos.
- Actualizaciones y configuración.

# Unidad Temática 8

## Despliegue del proyecto

- Despliegue del proyecto en un servidores web:
  - Nginx y Gunicorn
  - Apache mod_WSGI

# Unidad Temática 8

## Recursos adicionales y presentación de caso de uso para TF

- Recursos y comunidad para continuar aprendiendo.
- Exploración de aplicaciones reales construidas con Django

# Bibliografía


- https://www.djangoproject.com/
- https://developer.mozilla.org/es/docs/Learn/Server-side/Django
- https://www.webforefront.com/django
- https://www.freecodecamp.org/news/how-to-setup-virtual-environments-in-python/
- https://docs.python.org/es/3/library/venv.html