

```

    if not (poll.poll_max_length > 200): votes = models.IntegerField(last_published_recently=True).return self.poll_date -= timedelta(days=1)
    except 404: render_to_response from django.http import HttpResponseRedirect, HttpResponse from django.core.urlresolvers import reverse from django.template import Request
    import Choice, Poll def vote(request, poll_id): p = get_object_or_404(Poll, pk=poll_id) try: selected_choice = p.choice_set.get(pk=request.POST['choice']) except (KeyError, Choice.DoesNotExist):
    render_to_response('polls/detail.html', { 'poll': p, 'error_message': "You didn't select a choice.", }, context_instance=RequestContext(request)) else: selected_choice.votes += 1 selected
    HttpResponseRedirect(reverse('polls.views.results', args=(p.id,))) from django.utils import unittest from myapp.models import Animal class AnimalTestCase(unittest.TestCase): def setUp(self):
    objects.create(name="lion", sound="roar") self.cat = Animal.objects.create(name="cat", sound="meow") def test_animals_can_speak(self): self.assertEqual(self.lion.speak(), 'The lion says
    self.assertEqual(self.cat.speak(), 'The cat says "meow"') from django.contrib.syndication.views import FeedDoesNotExist from django.shortcuts import get_object_or_404 class BeatFeed
    ption template = 'feeds/beat_description.html' def get_object(self, request, beat_id): return get_object_or_404(Beat, pk=beat_id) def title(self, obj): return "Chicago crime.org: Crimes for beat %s"
    beat def link(self, obj): return obj.get_absolute_url() def description(self, obj): return "Crimes recently reported in police beat %s" % obj.beat def items(self, obj): return Crime.objects.filter(beat=obj).order
    by('-date')[:30] from django.utils.html import conditional_escape from django.utils.safestring import mark_safe @register.filter(needs_autoescape=True) def initial_letter_filter(text, autoescape=None):
    text = text[0].text[1:] if autoescape: esc = conditional_escape else: esc = lambda x: x result = '<strong>%s</strong>%s' % (esc(first), esc(other)) return mark_safe(result) from django import template
    def __init__(self, parser, token): try: tag_name, format_string = token.split_contents(), except ValueError: raise template.TemplateSyntaxError("%r tag requires a single argument" % token.contents.split()[0])
    format_string[0] == format_string[-1] and format_string[0] in ('"', "'"): raise template.TemplateSyntaxError("%r tag's argument should be in quotes" % tag_name) return CurrentTimeNode(format_string)
    from django import template import datetime class CurrentTimeNode(template.Node): def __init__(self, format_string): self.format_string = format_string def render(self, context): ret
    urn datetime.now().strftime(self.format_string) class CycleNode(Node): def __init__(self, cyclevars): self.cyclevars = cyclevars def render(self, context): if self not in context.render_con
    ext.render_context[self] = itertools.cycle(self.cyclevars) cycle_iter = context.render_context[self] return cycle_iter.next() class MaleManager(models.Manager): def get_query_set(self): return su
    leManager(self).get_query_set().filter(sex='M') class FemaleManager(models.Manager): def get_query_set(self): return super(FemaleManager, self).get_query_set().filter(sex='F') class Person(models.Mod
    el) t_name = models.CharField(max_length=50) last_name = models.CharField(max_length=50) sex = models.CharField(max_length=1, choices= (('M', 'Male'), ('F', 'Female'))) people = models.Manager() men
    leManager() women = FemaleManager() from django import forms class ContactForm(forms.Form): subject = forms.CharField(max_length=100) message = forms.CharField() sender = forms.EmailField
    myself = forms.BooleanField(required=False) if form.is_valid(): subject = form.cleaned_data['subject'] message = form.cleaned_data['message'] sender = form.cleaned_data['sender'] cc_myself
    n.cleaned_data['cc_myself'] recipients = ['info@example.com'] if cc_myself: recipients.append(sender) from django.core.mail import send_mail(subject, message, sender, recipients) ret
    urn HttpResponseRedirect('/thanks/') from django.http import HttpResponseRedirect from django.template import Context, loader def my_view(request): t = loader.get_template('myapp/template.html') c = Context({'
    '}) return HttpResponse(t.render(c), mimetype='application/xhtml+xml') from django.template.loaders import app_directories class Loader(app_directories.Loader): is_usable = True def load_template
    (self, template_name, template_dirs=None): source, origin = self.load_template_source(template_name, template_dirs) template = Template(source) from django import template from
    django.template.defaultfilters import stringfilter register = template.Library() @register.filter @stringfilter def lower(value): return value.lower() from django.utils.html import conditional_escape
    from django.utils.safestring import mark_safe @register.filter(needs_autoescape=True) def initial_letter_filter(text, autoescape=None): first, other = text[0].text[1:] if autoescape: esc = conditional_escape else:
    text = text[0].text[1:] if autoescape: esc = conditional_escape else: esc = lambda x: x result = '<strong>%s</strong>%s' % (esc(first), esc(other)) return mark_safe(result) from django import template def do_current_time(parser, token): try: tag_name, format_string
    = token.split_contents() except ValueError: raise template.TemplateSyntaxError("%r tag requires a single argument" % token.contents.split()[0]) if not (format_string[0] == format_string[-1] and format_string[0]
    in ('"', "'")): raise template.TemplateSyntaxError("%r tag's argument should be in quotes" % tag_name) return CurrentTimeNode(format_string[1:-1]) from django.conf import settings from
    django.contrib.auth.models import User, check_password class SettingsBackend(object): supports_inactive_user = False def authenticate(self, username=None, password=None): login_val
    id = settings.ADMIN_LOGIN == username) pwd_valid = check_password(password, settings.ADMIN_PASSWORD) if login_valid and pwd_valid: try: user = User.objects.get(username=username)
    if user.DoesNotExist: user = User(username=username, password='get from settings.py') user.is_staff = True user.is_superuser = True user.save() return user return None def get_user(self, user_id): try:
    user = User.objects.get(pk=user_id) except User.DoesNotExist: return None class EntryDetail(models.Model): entry = models.OneToOneField(Entry) details = models.TextField() class Blog(models.Model)
    title = models.CharField(max_length=100) tagline = models.TextField() def __unicode__(self): return self.name class Author(models.Model): name = models.CharField(max_length=50) email = models.EmailField
    def __unicode__(self): return self.name class Entry(models.Model): blog = models.ForeignKey(Blog) headline = models.CharField(max_length=255) body_text = models.TextField() pub_date = models.DateTimeField
    from django import models class BookManager(models.Manager): def get_queryset(self): return Book.objects.create_book_title and Prejudice from django.core.signals import request_finished from django.dispatch import receiver @receiver(request_finished)

```

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 asíncronas 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 asíncronas, optimización de rendimiento y 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.

```
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)
```

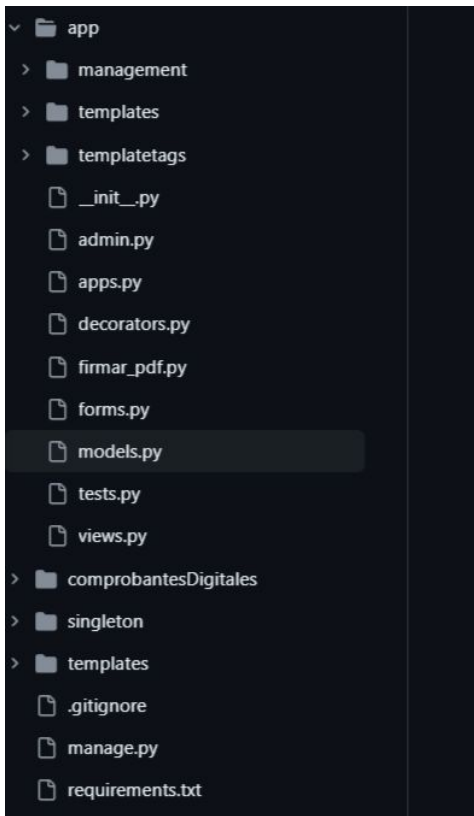
¿No es fácil? A partir de una declaración como **models.CharField(max_length=n)** se puede generar un input en un formulario HTML, una declaración SQL DDL y validar automáticamente la información para que sólo contenga n caracteres. Esto es exactamente lo que hace el principio de diseño DRY de Django.

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.

```
@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":

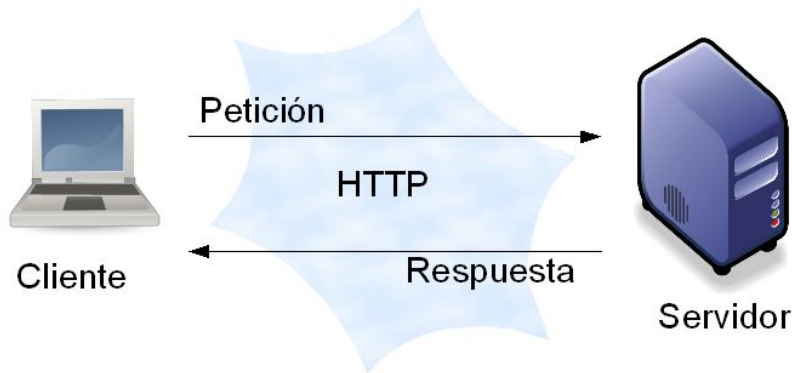
- **Productividad:** Al tener todas las herramientas necesarias integradas, los desarrolladores pueden empezar a trabajar inmediatamente sin perder tiempo en configurar componentes básicos.
- **Consistencia:** Usar herramientas integradas asegura que todos los componentes trabajen bien juntos, reduciendo 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.

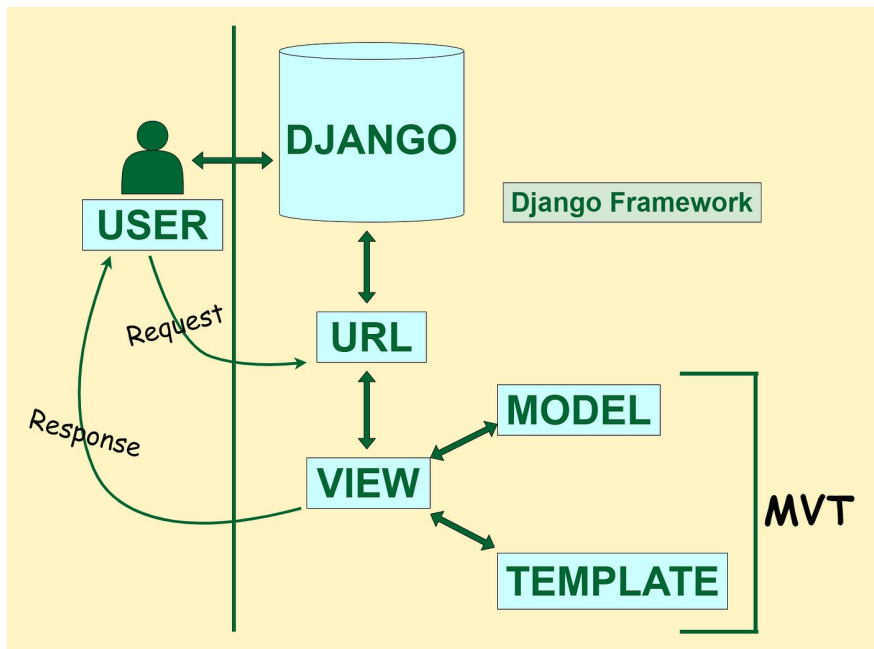


Patrón MVT

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.

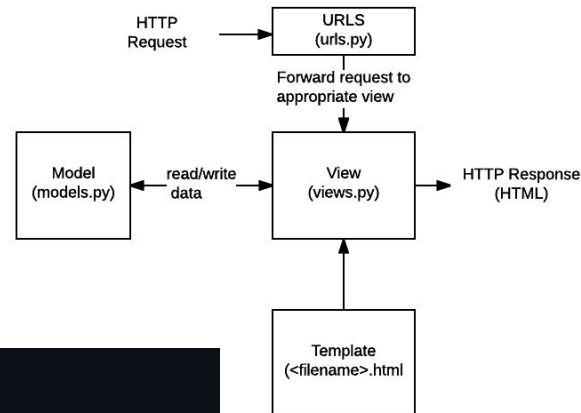


Componentes del patrón MVT en Django

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'),
```

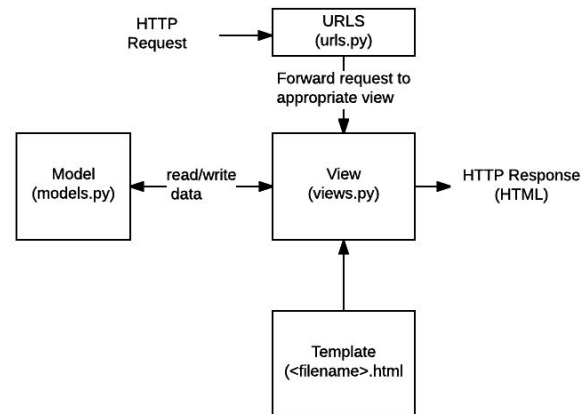
Componentes del patrón MVT en Django

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.

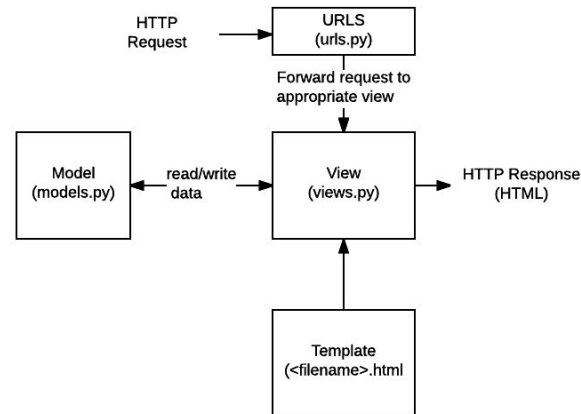


```
def home(request):  
    if Parametrizacion.load().coming_soon:  
        return render(request, "coming_soon.html")  
  
    args = {  
        'slider_home': Desarrollo.objects.filter(in_slider_home=True, publicado=True).order_by('orden_slider_home'),  
    }  
    return render(request, "website/index.html",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



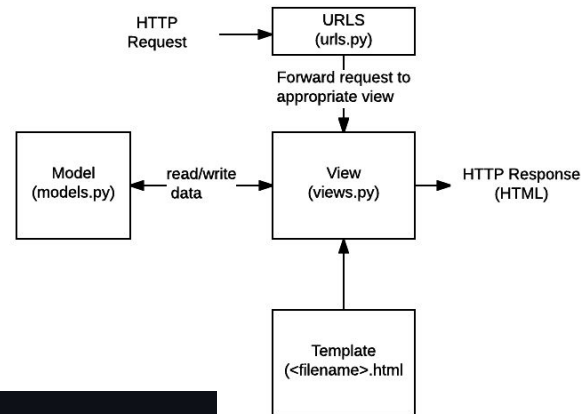
```
class Barrio(models.Model):
    descripcion = models.CharField(max_length=255, unique=True)

    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' %}"></a>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
{% endblock %}
```

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 que 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



Instagram

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

**moz://a**

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.

**eventbrite**

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 Django
- Instalación y configuración de Python y Django
- Creación de un proyecto y aplicación
Estructura de directorios y archivo
- Servidor de desarrollo
- Comandos básico
- Buenas prácticas.

django[View release notes for Django 2.1](#)

The install worked successfully! Congratulations!

You are seeing this page because `DEBUG=True` is in your settings file and you have not configured any URLs.

**Django Documentation**

Topics, references, & how-to's

**Tutorial: A Polling App**

Get started with Django

**Django Community**

Connect, get help, or contribute

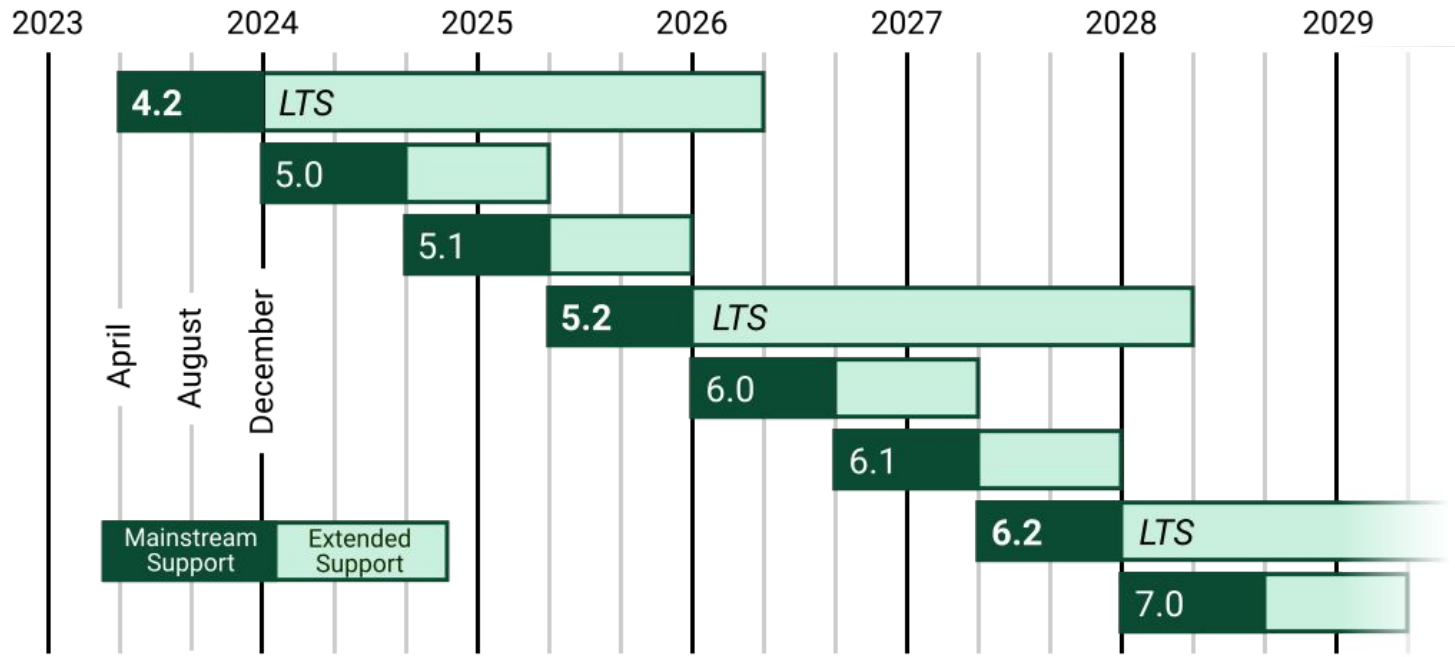
Versiones de Django

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 `source` activarlo:
`/path_al_nuevo_virtual_environment/bin/activate`

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

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

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

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..

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

<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

- De la lógica de negocio a la definición de modelos y relaciones.
- Migración y administración de bases de datos.
- Interfaz de administración integrada.
- Uso del ORM (Object Relational Mapping) de Django.



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.
- Implementación de autenticación de usuarios custom.



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>