PROGRAMACIÓN III

Clase 10

Clase 9

Base de datos

- Introducción a bases de datos con Python (SQLite, MySQL y SQL).
- Sintaxis básicas.
- Consultas avanzadas. Funciones.
- Conexión a la base de datos.
- Desarrollo y ejecución de operaciones (CRUD).
- Manejo de Archivos.

Clase 10

Django I

- ✓ Introducción. ¿Qué es Django?
- ✓ Origenes
- ✓ Características
- ✓ Ventajas vs. Desventajas
- ✓ Patrón: Modelo template vista
- ✓ Componentes principales
- ✓ Entorno de desarrollo.
- ✓ Configuración del entorno
- ✓ Instalación
- ✓ ORM

Clase 10

Django II

- ✓ Panel de administración
- ✓ Formularios
- ✓ Admin
- ✓ Sistema de autenticación y control de acceso
- ✓ Proyecto: ejemplo práctico


django

Django

El crecimiento de Python es cada vez mayor y esto se ha hecho más notorio en los últimos años, con la aparición de herramientas que hacen el trabajo más simple y eficiente con este lenguaje de programación. Una de esas herramientas es Django, *el framework hecho en python para perfeccionistas*.
Aparte de las ventajas que tiene por ser framework, fomenta el desarrollo rápido y limpio y el diseño pragmático.
Django impulsa el desarrollo de código limpio al promover buenas prácticas de desarrollo web, sigue el principio DRY (conocido también como Una vez y sólo una).
Usa una modificación de la arquitectura Modelo-Vista-Controlador (MVC), llamada MTV (Model – Template – View), que sería Modelo-Plantilla-Vista, está forma de trabajar permite que sea pragmático.

¿Qué es Django?

Django es un **framework web** de alto nivel escrito en **Python** que fomenta el **desarrollo rápido y limpio** y el **diseño pragmático**. Sigue el principio DRY: **Don't Repeat Yourself**

Orígenes

Django nace como un proyecto para publicación de noticias de Lawrence Journal-World, lo interesante de Django es que desde un principio fue construido como una herramienta para resolver problemas reales en un entorno empresarial, fue diseñado para optimizar el tiempo de desarrollo y los requerimientos exigentes de los desarrolladores web.
Se originó en un ambiente periodístico donde se pensaba que los desarrolladores deben ir al mismo ritmo que los periodistas y que el código debía ser “mantenible” por pocos desarrolladores
El nombre de Django es en honor al músico francés Django Reinhardt.

Quienes usan Django

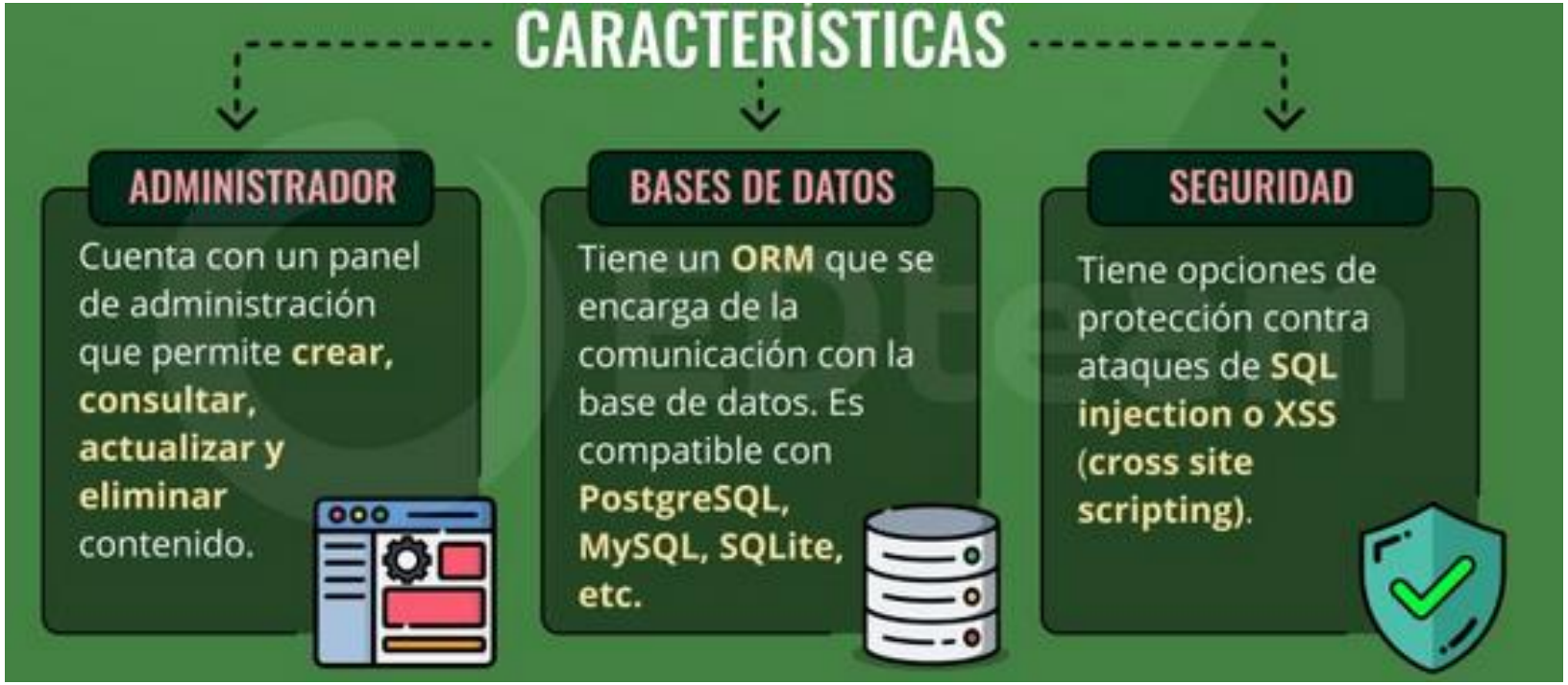
Características

- ✓ Es un framework de desarrollo web.
- ✓ Código abierto.
- ✓ Permite construir aplicaciones web más rápido.
- ✓ Utiliza menos código.
- ✓ Principio DRY (Don’t Repeat Yourself).
- ✓ Legible, casi pseudocódigo.

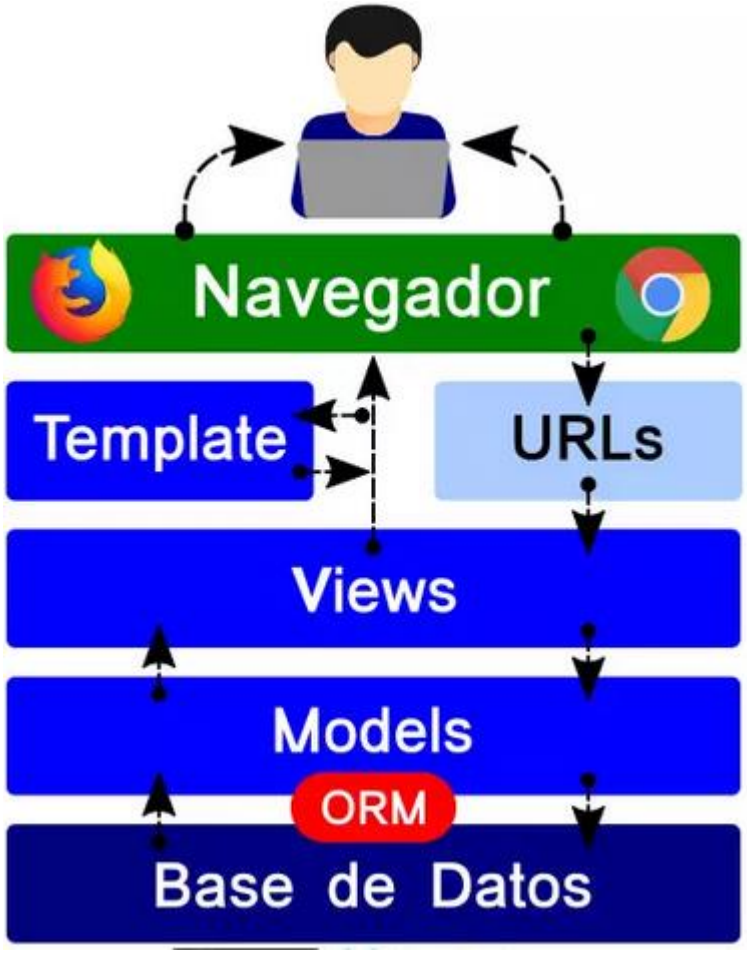
Ventajas

- ✓ Permite diseñar **URLs amigables para buscadores** (útil para SEO)
- ✓ Sistema de plantillas **sencillo para diseñadores**
- ✓ Genera una **interfaz de administración automática**
- ✓ Puede gestionar formularios, sesiones de usuario, autenticación, caché, almacenamiento, sitemaps, internacionalización, etc.

Características técnicas



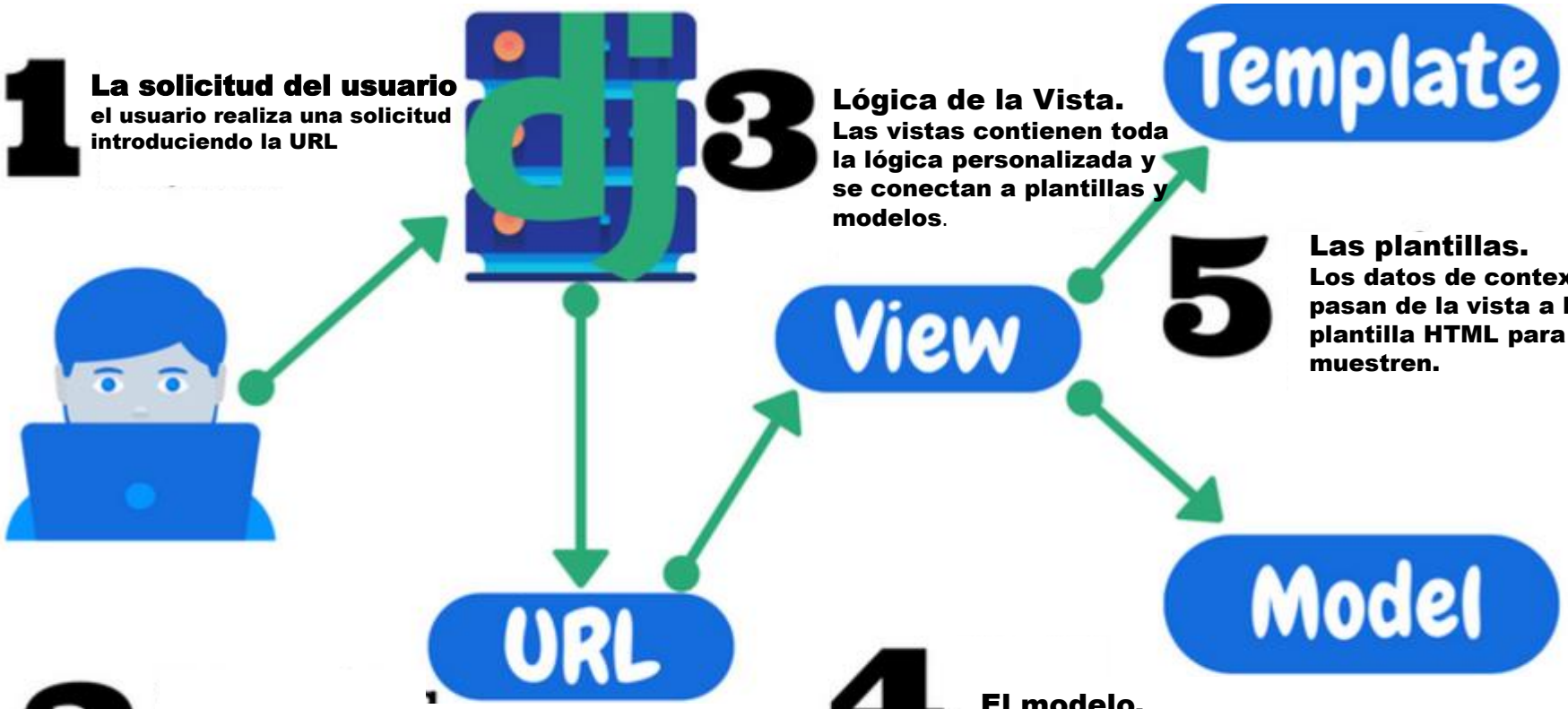
Modelo Template Vista



- ✓ **"Model" (Modelo)(models.py)** El modelo define los datos almacenados, se encuentra en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos.
- ✓ **"Template" (Plantilla) (plantillas html)**, La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solamente crea contenido en HTML
- ✓ **"View" (Vista) (views.py)**, se presenta en forma de funciones en Python, su propósito es determinar qué datos serán visualizados, entre otras cosas. El ORM de Django permite escribir código Python en lugar de SQL para hacer las consultas que necesita la vista. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios.

EL CONTROLADOR ES EL PROPIO FRAMEWORK

Modelo Template Vista



1 **La solicitud del usuario**
el usuario realiza una solicitud introduciendo la URL

3 **Lógica de la Vista.**
Las vistas contienen toda la lógica personalizada y se conectan a plantillas y modelos.

5 **Las plantillas.**
Los datos de contexto se pasan de la vista a la plantilla HTML para que se muestren.

2 **Resolución de URL.**
Se resuelve la URL que se asigna a una vista. Los argumentos también se extraen de la URL y se envían a la vista.
/user/1 → UserView(id → 1)

4 **El modelo.**
Contiene los datos ya que está conectado a la base de datos con ORM. proporciona los datos necesarios para que las vistas se envíen a la plantilla.

Funcionamiento de las peticiones HTTP

1. El usuario pide una URL (por GET ó POST)
2. Django busca la primera URL que coincida con la solicitada (**urls.py**)
3. Se ejecuta la vista (función) a la que apunta esa URL (**views.py**)
4. En la vista se utilizan los **models** (modelos de datos) para consultar la BD Los datos resultantes se introducen en la plantilla (**templates**)
5. Se devuelve una respuesta HTTP con el HTML generado

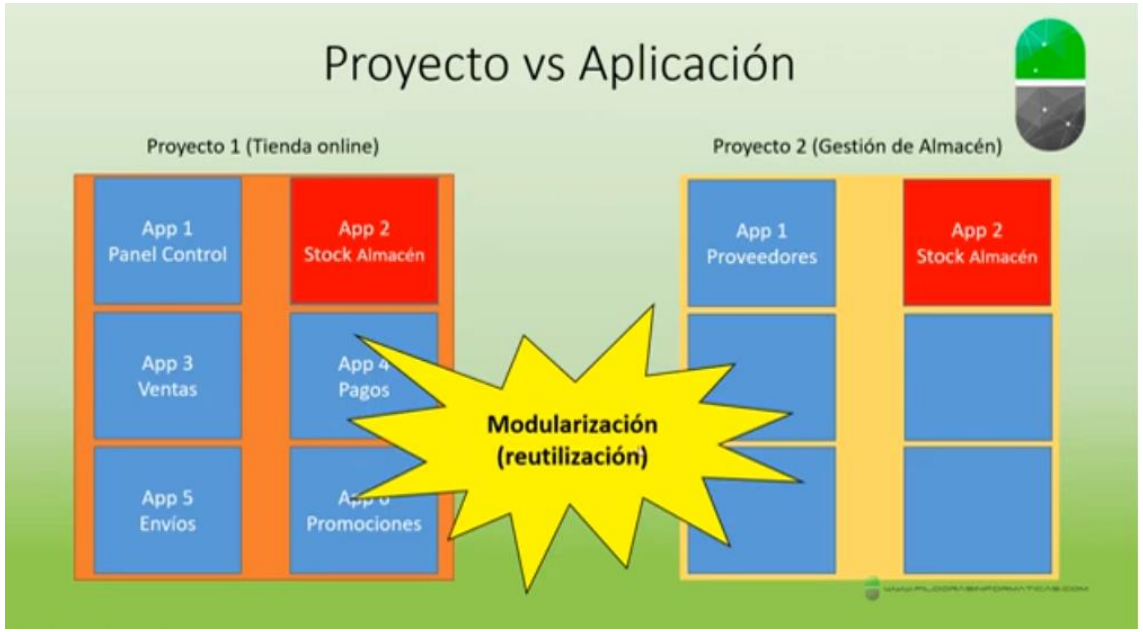
Aplicaciones Contrib

Django pretende seguir la filosofía de Python. Ofrece una variedad de herramientas extra, opcionales que resuelven los problemas comunes de desarrollo de la web. Este código vive en django/contrib en la distribución de Django. Este documento da un resumen de los paquetes en contrib, junto con cualquier dependencia esos paquetes lo han hecho.

- ✓ **auth**: Autenticación de usuarios.
- ✓ **admin**: Sitio de administración CRUD.
- ✓ **messages**: Mensajes de aviso para usuarios.
- ✓ **sessions**: Gestión de sesiones.
- ✓ **sites**: Manejar distintos sitios web con un proyecto.
- ✓ **sitemaps**: Generar sitemaps a partir de modelos.
- ✓ **syndication**: Generar feeds RSS y Atom a partir de modelos.
- ✓ **gis**: Trabajar con datos ego-espaciales (PostGIS)

Organización de un proyecto django

- ✓ Un desarrollo es un **Proyecto**
- ✓ Un proyecto consta de una o **varias aplicaciones**
- ✓ Cada aplicación hace **algo en concreto**
- ✓ **Ej: Proyecto** “Tienda online" **Aplicaciones:** panel de control, ventas, pagos, stock almacén, envíos, promociones, etc.
- ✓ Una aplicación puede ser **utilizada por distintos proyectos** a la vez Un proyecto puede hacer funcionar varios sitios web



un proyecto Django es una colección de una o más aplicaciones, mientras que una aplicación Django es un módulo que se puede reutilizar en múltiples proyectos .

Estructura de un proyecto

proyecto/
- aplicación 1/
  - models.py
  - views.py
  - urls.py
  - templates/
- aplicación 2/
  - models.py
  - views.py
  - urls.py
  - templates/
- ……
- ……
- aplicación n
  - models.py
  - views.py
  - urls.py
  - templates/

Estructura de un proyecto

Archivos de un proyecto

- ✓ **_init__.py**
- ✓ Indica a Python que el directorio sea interpretado como un paquete de Python
- ✓ **settings.py**

  Contiene la configuración de la aplicación (conexión a bases de datos, aplicaciones instaladas, etc.)
- ✓ **manage.py**

  Nos permite ejecutar comandos de Django sobre el proyecto (ej. Para crear nuevas aplicaciones)
- ✓ **urls.py**

  Contiene los patrones de URLs del proyecto
- ✓ **wsgi.py**

  Este archivo contiene el código necesario para inicializar y configurar la aplicación Django para que pueda ser servida por un servidor WSGI.

```
/proyecto/
    /proyecto/
        __init__.py
        urls.py
        manage.py
        settings.py

    blog/
        __init__.py
       models.py
       views.py
       urls.py
       templates/
       static/

   foro/
       __init__.py
       models.py
       views.py
       urls.py
       templates/
```

Estructura de un proyecto

Archivos de una aplicación

- ✓ **_init__.py**
- ✓ **models.py**

  Contiene nuestros modelos de datos
- ✓ **views.py**

  Contiene las vistas de la aplicación
- ✓ **tests.py**

  Permite que incluyamos tests para la aplicación
- ✓ **urls.py**

  Es usual añadir un **urls.py** con las URLs de nuestra aplicación e importarlas en el urls.py del proyecto por motivos de organización.

```
/proyecto/
    /proyecto/
        __init__.py
        urls.py
        manage.py
        settings.py

    blog/
        __init__.py
        models.py
        views.py
        urls.py
        templates/
        static/

    foro/
        __init__.py
        models.py
        views.py
        urls.py
        templates/
```

ARCHIVOS DEL PROYECTO

SETTING.PY

Este archivo permite configurar la conexión a la base de datos, la zona horaria, el idioma, los directorios principales del proyecto, las aplicaciones del proyecto, entre otras cosas. Configura:

Codificación de caracteres: Nuestro idioma esta lleno de caracteres especiales como las tildes y las eñes que son las más comunes, la primera sugerencia para manejar esto eficientemente en Django es agregar la siguiente línea al archivo settings.py:
#encoding:utf-8
Ruta del proyecto: Es importante la configuración de la ruta del proyecto, esto permitirá lanzar la aplicación desde cualquier directorio y mover el proyecto a cualquier computador con Django instalado.
# Identificando la ruta del proyecto import os
RUTA_PROYECTO=os.path.dirname(os.path.realpath(__file__))

ARCHIVOS DEL PROYECTO

SETTING.PY

- ✓ **Administradores:** Cuando Django tiene la opción de DEBUG=False (No lo cambies por ahora, déjalo en True), las notificaciones de error de código deben ser enviadas vía correo electrónico a los administradores, junto con los detalles completos del error. Para poner los datos de los administradores debemos buscar la siguiente porción:

  ADMINS = (
  # (‘Your Name’, ‘your_email@example.com’),)
  Y modificarla para que quede con los nombres de los administradores en forma de tupla, en mi caso lo dejaré así:
  ADMINS = (
  (‘Sergio Infante Montero’, ‘raulsergio9@gmail.com’),)

- ✓ **Configuración de la base de datos**: También podemos configurar la conexión a la base de datos según nuestras necesidades, Django soporta de manera predeterminada la conexión con postgresql, mysql, sqlite3 y oracle.

ARCHIVOS DEL PROYECTO

SETTING.PY

- ✓ **Zona horaria**: Django permite configurar la zona horaria del proyecto, la lista de zonas horarias disponibles1 se pueden encontrar en la wikipedia. Para configurar debemos buscar lo siguiente:

  TIME_ZONE = ‘America/Buenos aires’
- ✓ **Configuración del idioma**: Django también permite configurar el idioma que usará de manera predeterminada para su funcionamiento, para configurar esto debemos buscar lo siguiente:

  LANGUAGE_CODE = ‘en-us’
- ✓ **Aplicaciones instaladas:** Un proyecto en Django necesita de aplicaciones, algunas ya vienen configuradas de manera predeterminada. Aplicaciones que son herramientas extras que nos brinda django para nuestro proyecto.

Aplicaciones Contrib

Django pretende seguir la filosofía de Python. Ofrece una variedad de herramientas extra, opcionales que resuelven los problemas comunes de desarrollo de la web. Este código vive en django/contrib en la distribución de Django. Este documento da un resumen de los paquetes en contrib, junto con cualquier dependencia esos paquetes lo han hecho.

- ✓ **auth**: Autenticación de usuarios.
- ✓ **admin**: Sitio de administración CRUD.
- ✓ **messages**: Mensajes de aviso para usuarios.
- ✓ **sessions**: Gestión de sesiones.
- ✓ **sites**: Manejar distintos sitios web con un proyecto.
- ✓ **sitemaps**: Generar sitemaps a partir de modelos.
- ✓ **syndication**: Generar feeds RSS y Atom a partir de modelos.
- ✓ **gis**: Trabajar con datos ego-espaciales (PostGIS)

ARCHIVOS DEL PROYECTO

URLS.PY

Contiene las rutas que están disponibles en el proyecto.
Django posee un mapeo de URLs que permite controlar el despliegue de las vistas, ésta configuración es conocida como URLConf. El trabajo del URLConf es leer la URL que el usuario solicitó, encontrar la vista apropiada para la solicitud y pasar cualquier variable que la vista necesite para completar su trabajo.

```
from django.contrib import admin
from django.urls import path
from proyecto1.views import saludo, despedida, dameFecha, calculaEdad, cursoC, cursoCss

urlpatterns = [
    path('admin/', admin.site.urls),
    path('saludo/',saludo),
    path('nosveremos/',despedida),
    path('fecha/',dameFecha),
    path('edades/<int:edad>/<int:agno>',calculaEdad),
    path('cursoC/',cursoC),
    path('cursoCss/',cursoCss)
]
```

ARCHIVOS DEL PROYECTO

VIEWS.PY

El archivo views.py en Django contiene las vistas de la aplicación. Las vistas son funciones o clases que reciben peticiones web (HTTP requests) y devuelven respuestas web (HTTP responses). En otras palabras, las vistas procesan los datos necesarios y renderizan las plantillas HTML para el usuario.

```python
from django.shortcuts import render
from django.http import HttpResponse
from gestionPedidos.models import Articulos

# Create your views here.

def busqueda_productos(request):
    return render(request,"busqueda_productos.html")

def buscar(request):

    if request.GET["prd"]:
        #mensaje="Articulo buscado: %r" %request.GET["prd"]
        producto=request.GET["prd"]

        if len(producto)>:
            mensaje="Twxto de busqueda demasiado largo"
        else:
            articulos=Articulos.objects.filter(nombre__contains=producto)

            return render(request, "resultados_busqueda.html", {"articulos":articulos, "query":producto})
    else:
        mensaje="No has introducido nada"

    return HttpResponse(mensaje)
```

PLANTILLAS

El lenguaje de plantillas de Django es una herramienta poderosa para la creación de vistas en aplicaciones web basadas en Django. Permite la generación dinámica de contenido HTML con una sintaxis sencilla y fácil de usar. Aquí te proporciono una guía básica sobre cómo utilizar el lenguaje de plantillas de Django:

Estructura Básica

Las plantillas de Django son archivos HTML que pueden contener expresiones y etiquetas de Django para generar contenido dinámico. Un ejemplo de plantilla básica podría ser:

```
<html>
        <head>
                <title>{{ title }}</title>
        </head>
        <body>
                <h1>{{ heading }}</h1>
                <p>{{ content }}</p>
        </body>
</html>
```

PLANTILLAS - ELEMENTOS

Variables

Las variables se representan con doble llave {{ }}. Por ejemplo:

```
<p>Hello, {{ name }}!</p>
```

Etiquetas (Tags)

Las etiquetas de Django se utilizan para lógica de control, como bucles y condicionales.
Se representan con {% %}.

```
{% if user.is_authenticated %}
        <p>Welcome, {{ user.username }}!</p>
{% else %}
        <p>Please log in.</p>
{% endif %}
```

```
<ul>
        {% for item in item_list %}
                <li>{{ item }}</li>
        {% endfor %}
</ul>
```

PLANTILLAS - ELEMENTOS

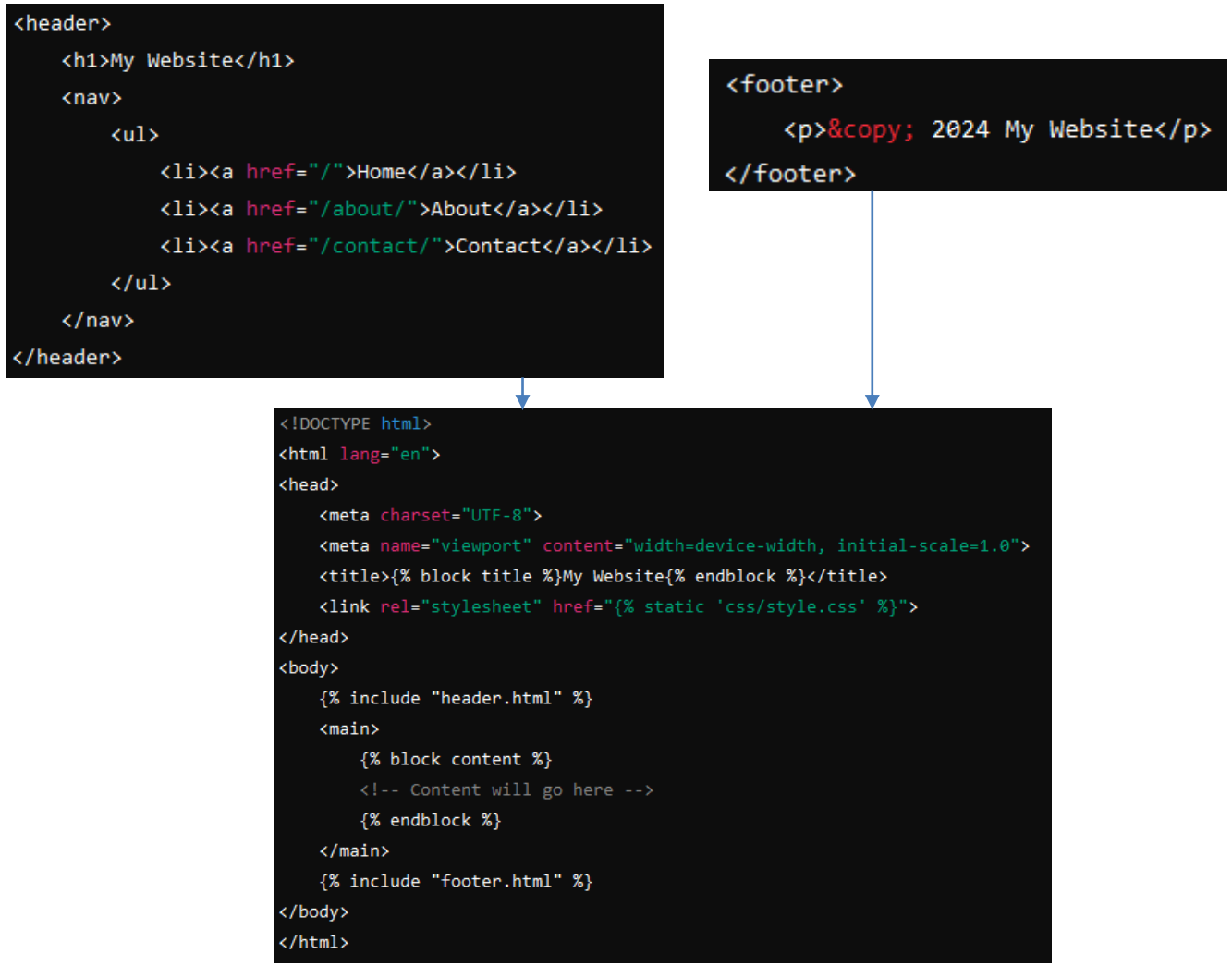
Filtros

Los filtros se utilizan para modificar el valor de una variable. Se aplican con la barra vertical:

```
<p>{{ text|lower }}</p> <!-- Convierte el texto a minúsculas -->
<p>{{ number|length }}</p> <!-- Devuelve la longitud de la lista o string -->
<p>{{ value|add:5 }} </p> <!– suma a una variable una constante -->
<p>{{ value|date:"Y-m-d" } </p> <!– formatea una fecha -->
```

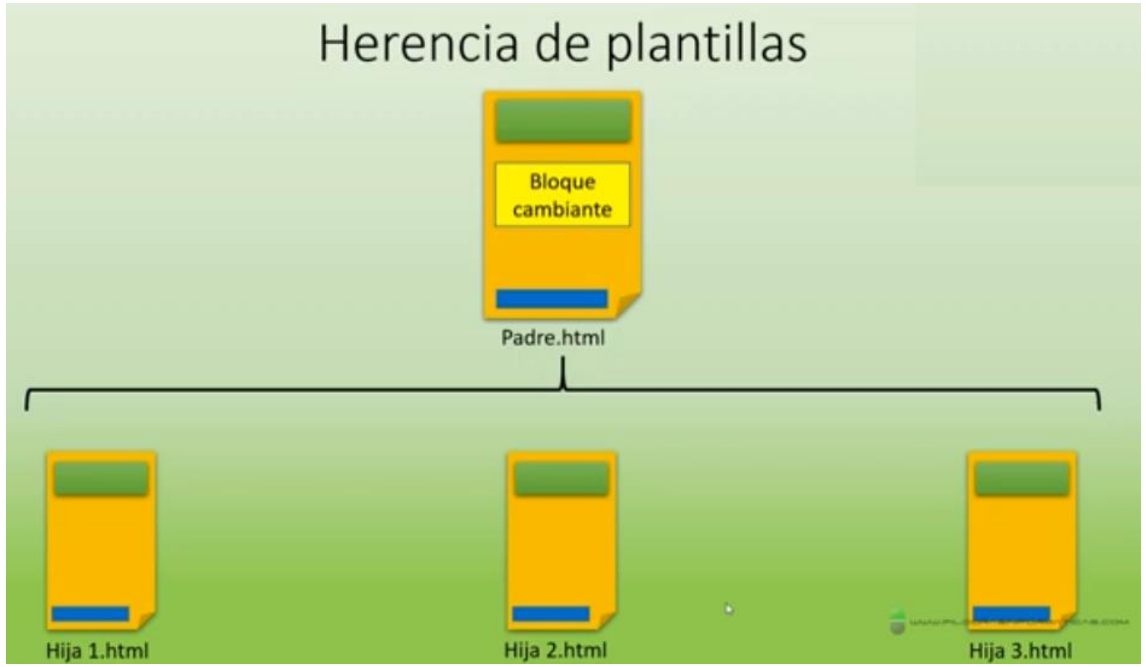
PLANTILLAS INCRUSTADAS

las plantillas incrustadas (o anidadas) permiten incluir fragmentos de código HTML comunes en varias páginas sin duplicar el contenido. Esto es especialmente útil para componentes repetitivos como encabezados, pies de página, barras de navegación, etc.
La etiqueta {% include %} se utiliza para insertar el contenido de una plantilla en otra. Esta es una manera efectiva de mantener el código limpio y reutilizable



PLANTILLAS - HERENCIA

La herencia de plantillas en Django permite reutilizar y extender plantillas HTML de manera eficiente. Esto se logra mediante la definición de una plantilla base que contiene el diseño general y las secciones que otras plantillas pueden sobrescribir o extender.



PLANTILLAS - HERENCIA

Base.html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Mi Sitio{% endblock %}</title>
    <link rel="stylesheet" href="{% static 'css/styles.css' %}">
</head>
<body>
    <header>
        <h1>Mi Sitio Web</h1>
    </header>

    <nav>
        <!-- Navegación principal -->
    </nav>

    <main>
        {% block content %}
        <!-- Contenido principal -->
        {% endblock %}
    </main>

    <footer>
        <p>© 2024 Mi Sitio Web</p>
    </footer>
</body>
</html>
```

Plantilla que extiende la base: index.html

```
{% extends "base.html" %}

{% block title %}Página de Inicio{% endblock %}

{% block content %}
<h2>Bienvenido a Mi Sitio Web</h2>
<p>Este es el contenido de la página de inicio.</p>
{% endblock %}
```

ORM(Object-Relational Mapping**)**

•**Object:** hace referencia a la programación orientada a objetos.
•**Relational**: hace referencias a las bases de datos relacionales (SQL).
•**Mapping**: es un proceso que consiste en emparejar valores.

Es una herramienta que permite a los desarrolladores interactuar con bases de datos utilizando el paradigma orientado a objetos en lugar de escribir consultas SQL directamente. El ORM de Django proporciona una manera de definir modelos (esquemas de bases de datos) en Python y trabajar con estas estructuras de datos de una manera sencilla y consistente.
Entonces el ORM se encarga de dos cosas:

1. Hacer consultas e interactuar con la base de datos, usando el lenguaje de backend, sin necesidad de escribir SQL.
2. Hacer el mapeo de las estructuras de las bases de datos a objetos.

ORM(Object-Relational Mapping**)**

ORM

Ventajas

- ✓ **Abstracción del SQL**: Permite trabajar con datos sin escribir consultas SQL directamente.
- ✓ **Seguridad**: Ayuda a prevenir inyecciones SQL automáticamente.
- ✓ **Portabilidad**: El código es más portable entre diferentes bases de datos.
- ✓ **Simplicidad**: Facilita la creación y gestión de esquemas de base de datos con Python.

ORM

Desventajas

- ✓ **Lentitud** o problemas de performance en algunos casos; ya que en muchas ocasiones se lanzan consultas innecesarias, y muy costosas.
- ✓ **Pérdida de flexibilidad**, ya que los ORMs en muchos casos no pueden replicar a la perfección la versatilidad que ofrecen los lenguajes de consultas de las propias bases de datos, y tienen algunas limitaciones.
- ✓ **Los desarrolladores pierden conocimiento de lo que está sucediendo en la base de datos**, ya que todo este conocimiento cae del lado del ORM, y eso acaba provocando que el ORM se acabe convirtiendo en una caja negra para el desarrollador.

ORM

Conceptos Básicos del ORM de Django

- ✓ **Modelos**: Los modelos son clases de Python que representan tablas en la base de datos. Cada atributo de la clase representa una columna en la tabla.
- ✓ **Consultas**: El ORM permite realizar consultas a la base de datos utilizando métodos de Python en lugar de escribir SQL manualmente.
- ✓ **Migraciones**: Las migraciones son archivos que contienen instrucciones para modificar la estructura de la base de datos. Django genera estos archivos automáticamente a partir de los modelos definidos.

ORM

MODELOS

Django Models son los modelos para representar las tablas de nuestra base de datos SQL, y modelan la información.
Para definir un modelo en Django, se crea una clase que hereda de django.db.models.Model.
Cada atributo de la clase corresponde a una columna en la base de datos.

```python
from django.db import models

class Author(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(Author, on_delete=models.CASCADE)
    published_date = models.DateField()
    isbn = models.CharField(max_length=13, unique=True)
```

ORM

Migraciones

Una vez definidos los modelos, se deben crear y aplicar las migraciones para actualizar la base de datos.

1. **Crear migraciones**: Genera un archivo de migración a partir de los modelos definidos.

```
python manage.py makemigrations
```

2. **Aplicar migraciones**: Aplica las migraciones a la base de datos.

```
python manage.py migrate
```

ORM

Consultas: QuerySets.

- ✓ **all()**: recupera todo el contenido que tenemos guardado en nuestro modelo, al hacerlo de forma lazy, podemos aplicar principios de paginación para evitar traer toda la información en memoria.

Django ORM

```
Person.objects.all()
```

SQL

```
SELECT *
FROM person
```

- ✓ **filter()**, que nos permite filtrar las filas de nuestra tabla por varios parámetros, sin olvidar que cada parámetro que incluyamos será una sentencia AND adicional.

Django ORM

```
Person.objects.filter(first_name='Juan', last_name='Hernández')
```

SQL

```
SELECT *
FROM person
WHERE first_name = 'Juan' AND last_name = 'Hernández'
```

ORM

Consultas: QuerySets.

- ✓ **Count()**: que nos permite contar todos los registros de la tabla. Este operador equivale al método COUNT de SQL.

Django ORM

```
Person.objects.count()
```

SQL

```
SELECT COUNT(*)
FROM person
```

- ✓ **Order_by()** nos permite ordenar los registros que recuperemos, incluyendo el nombre de la columna como texto, le indicamos sobre que parámetro queremos ordenar, e incluyendo el signo **-** le decimos que queremos que sea orden descendente, si no ponemos ningún signo será como orden ascendente.

Django ORM

```
Person.objects.order_by('-age')
```

SQL

```
SELECT *
FROM person
ORDER BY 'AGE' DESC
```

ORM

Consultas: QuerySets.

- ✓ **Aggregate():**nos permite hacer operaciones de agregación sobre todos los registros de una tabla, por ejemplo, si queremos sumar las edades de todas las personas, o como en este caso si queremos obtener la edad máxima.

Django ORM

```
Person.objects.aggregate(max_age = Max('age'))
```

SQL

```
SELECT MAX(age)
FROM person
```

- ✓ **Annotate()**: cuando queremos agrupar información por un parámetro.

Django ORM

```
Person.objects.annotate(age_count = Count('age')).values('age_count', 'age')

# [{age_count: 10, age: 21}, {age_count:5, age: 22}...]
```

SQL

```
SELECT age, COUNT(age) as age_count
FROM person
GROUP BY age
```

ORM

Consultas: QuerySets.

Crear registros

```
Person.objects.create(first_name='Antonio', last_name='Hernandez', age=29)
```

Actualizar registros

```
Person.objects.filter(first_name='Antonio').update(first_name='Tony')
```

Borrar registros

```
Person.objects.filter(age__gt=30).delete()
```

ORM

Django Models con Relaciones

La creación de relaciones y claves foráneas en Django es mediante las clases **ForeignKey**, donde solo hay que especificar el modelo al que queremos referenciar, para crear la relación.

```
class Customer(models.Model):
    name = models.CharField(max_length=50)
    age = models.IntegerField()
    country = models.CharField(max_length=50)

class Order(models.Model):
    date = models.DateTime()
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    amount = models.IntegerField()
```

Accediendo a una relación

Django ORM

```
order = Order.objects.get(pk=1)
order.customer.name # Nombre del customer
```

Instalación

- ✓ **Prerrequisitos**
  - ✓ Python instalado.
  - ✓ PIP para instalar el paquete Python (http://www.pip-installer.org/en/latest/installing.html)

- ✓ **pip install Django==5.0.6**
  - ✓ OR https://www.djangoproject.com/download/ - python setup.py install
- ✓ **pip install mysql-python (conector a la base de datos)**
  - ✓ MySQL on windows https://pypi.python.org/pypi/MySQL-python/1.2.4
- ✓ **Agregar Python y Django a las variables de entorno del sistema**
  - ✓ PYTHONPATH C:\Python27
  - ✓ Path C:\Python27; C:\Python27\Lib\site-packages; C:\Python27\Lib\site-packages\django\bin;
- ✓ **Instalación del Testing**
- ✓ shell> import django; django.VERSION;

MATERIAL EXTRA

Artículos de interés

Material extra:

- ✓ Django: página oficial
- ✓ Consultas con django