

Framework Django

Implementa aplicaciones web



Orden del dia

```
1.- def bienvenida():  
    Presentación del equipo  
  
2.- def discutir_el_problema_a_resolver():  
    Lectura del problema y análisis  
  
3.- def configurando_entorno_de_trabajo():  
    Instalar python 3.x  
    Instalar xampp, wamp, mamp  
    Instalar ide pycharm  
  
4.- class Break(models.Model):  
    comer_pizza_y_botanas = true
```

```
5.- def modelando_Bdatos():  
    Identificación de tablas  
    diseño Relacional en Navicat  
  
6.- def creando_el_proyecto() :  
    configurando environment virtual  
    adquiriendo paquetes  
    desplegando el proyecto  
    configurando bdatos mysql  
    instalando sección admin  
  
7.- def creando_mtv_noticias():  
    creando models noticias  
    creando templates noticias  
    creando views noticias
```

Invitación a la programación

Si lo escucho, lo olvido



Si lo veo, lo recuerdo



Si lo hago, lo entiendo



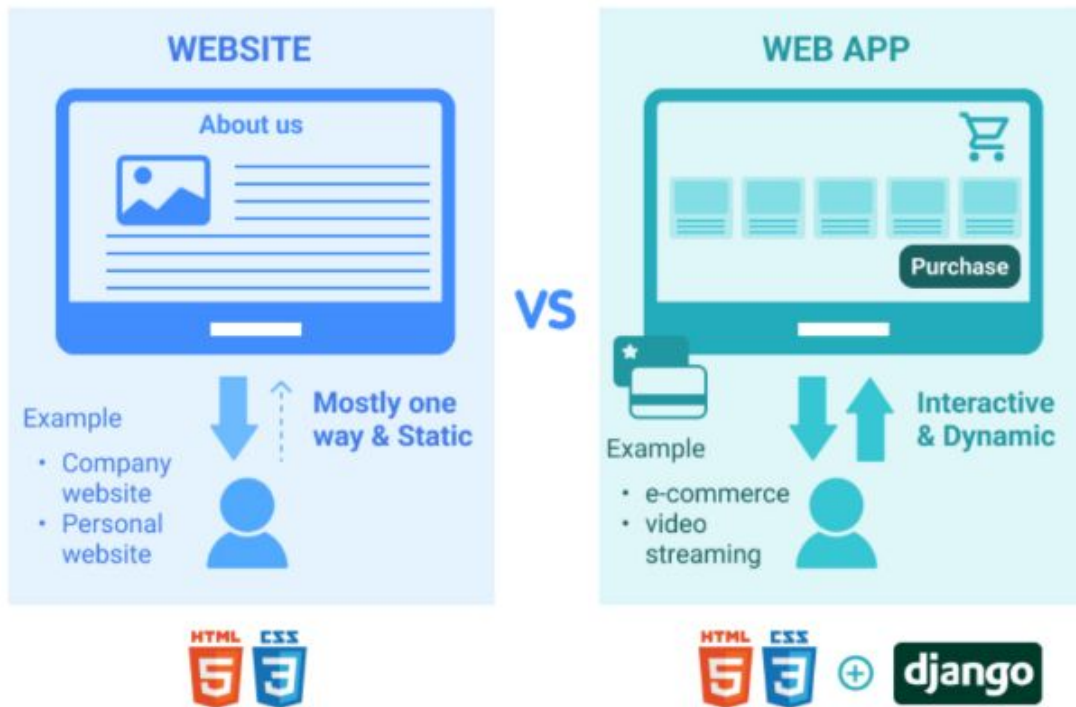
Que necesito saber



Base de Datos relacional y no relacional



sitio web vs aplicación web de Django



Frontend Backend

Full Stack Web Development



Django (Python)

DESARROLLO WEB CON DJANGO (PYTHON)

INSTALACIÓN DE LAS
HERRAMIENTAS DE TRABAJO

PYTHON

PIP

PATH

DJANGO

Introduccion e instalacion

- ¿Qué es Django? ¿Para qué sirve?
- ¿Qué es un framework web?
- Estructura general de Django (MVC y MTV)
- Instalación de Django

Qué es Django y para qué sirve?

- ¿Qué es?
 - Framework web gratuito y de código abierto escrito en Python.
 - ¿Y qué es un Framework?
 - Un **Framework** es un marco de trabajo formado por un conjunto de herramientas, librerías y buenas prácticas
- ¿Para qué sirve Django?
 - Para crear sitios web (complejos) de forma rápida y sencilla.
 - Hay tareas que son repetitivas, pesadas y comunes en el momento de crear diferentes sitios web. Django viene a facilitar la realización de estas tareas.
 - Hay código que podemos reutilizar de un sitio web a otro. Django también nos permite esta reutilización de forma sencilla.

Django es un framework escrito en Python, de alto nivel que fomenta el desarrollo rápido y un diseño limpio y pragmático.

DJANGO:

Django is a high-level Python **web framework** that encourages **rapid development** and **clean, pragmatic design**.








**FREE & OPEN
SOURCE**

**LARGE & ACTIVE
COMMUNITY**



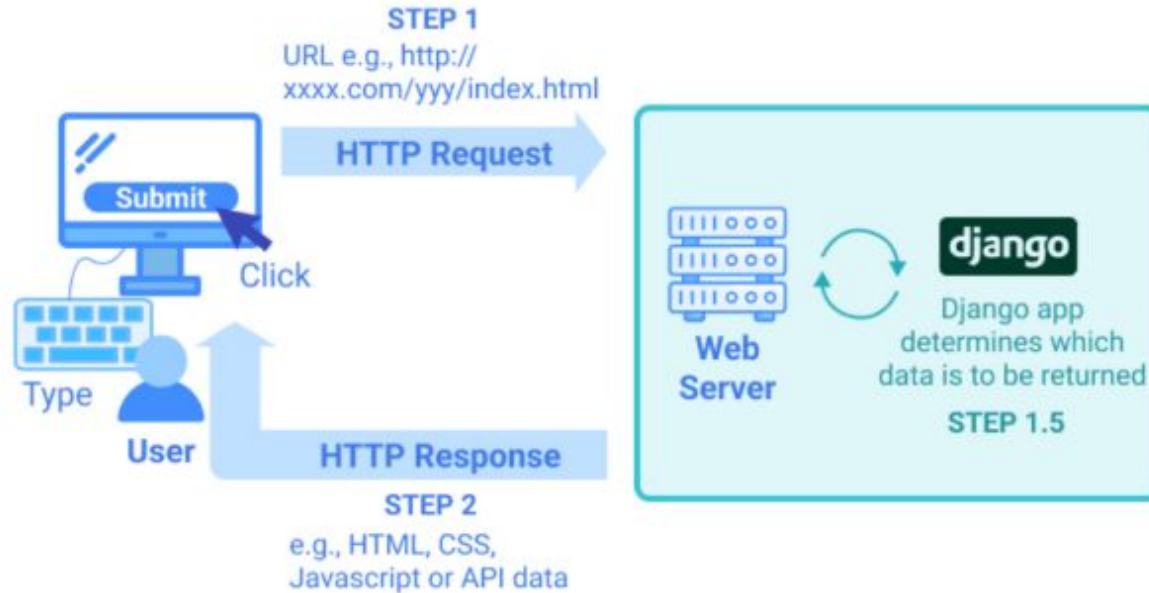
BATTERIES INCLUDED

Características principales

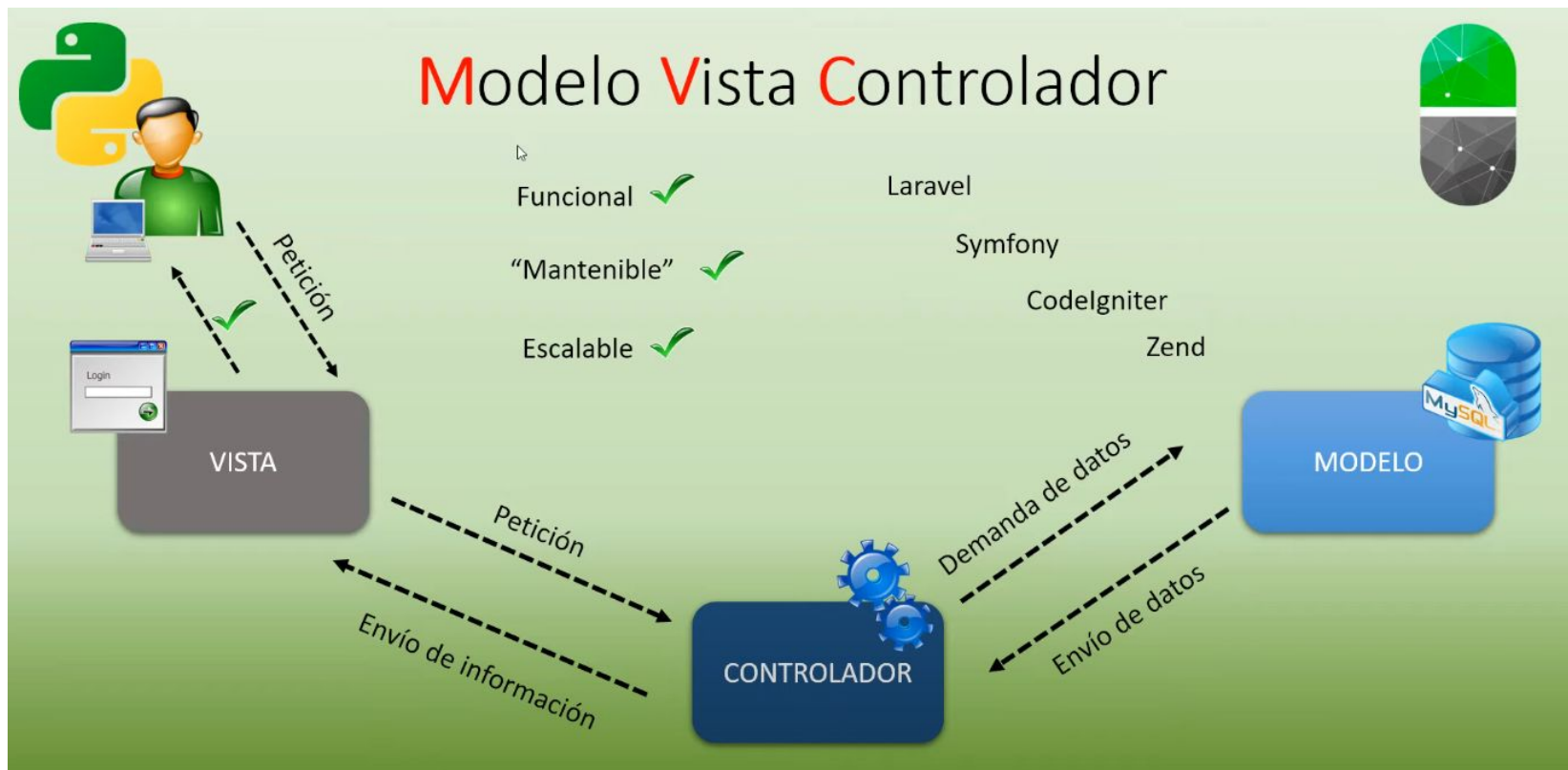
-  Framework completo
-  ORM (Object-Relational Mapping)
-  Sistema de plantillas
-  Seguridad integrada
-  Escalable
-  Panel administrativo
-  Comunidad activa

Cómo Django maneja las solicitudes HTTP y las respuestas HTTP,

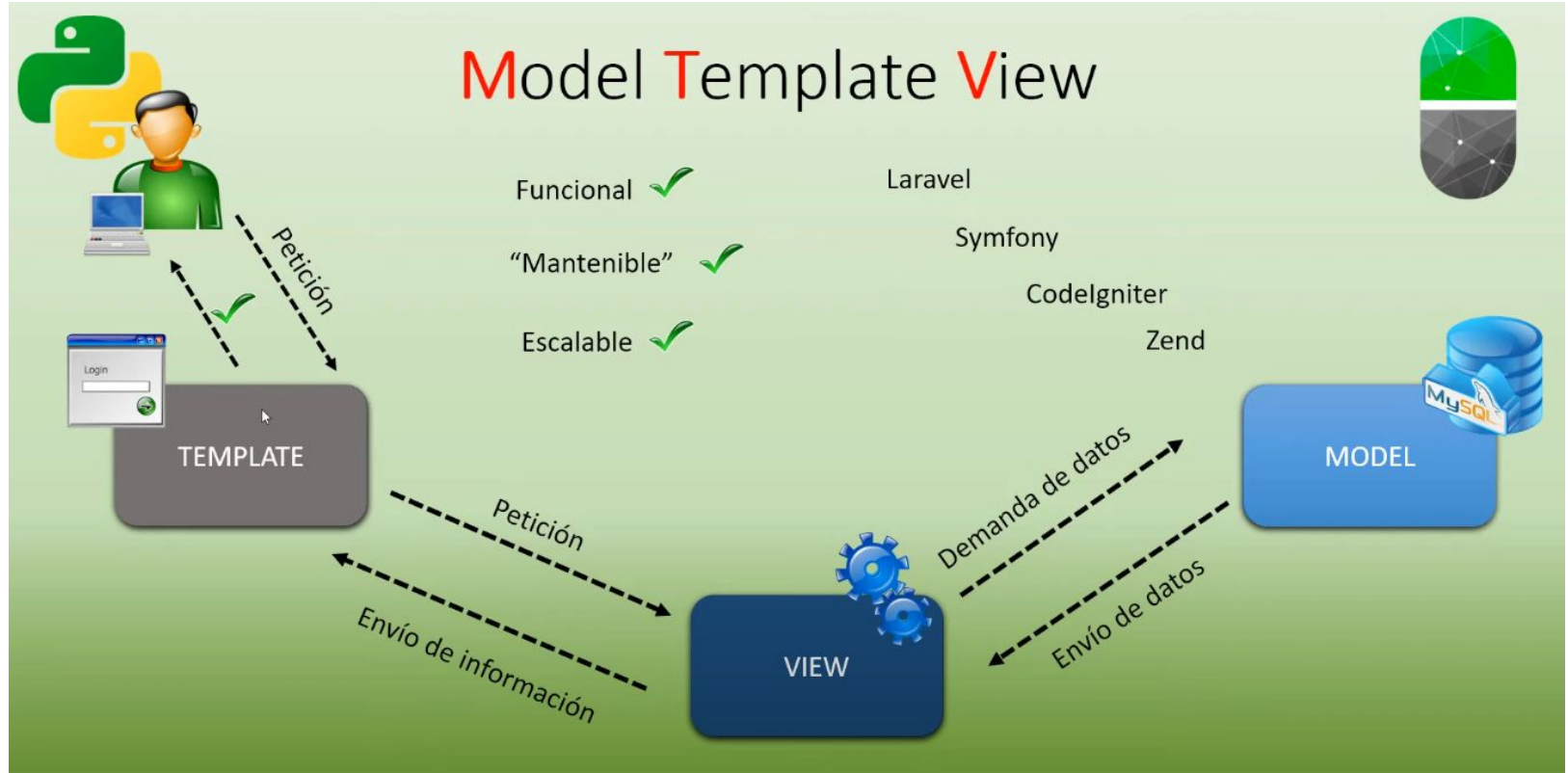
(interfaz de programación de aplicaciones)



Qué es Django y para qué sirve?



Qué es Django y para qué sirve?

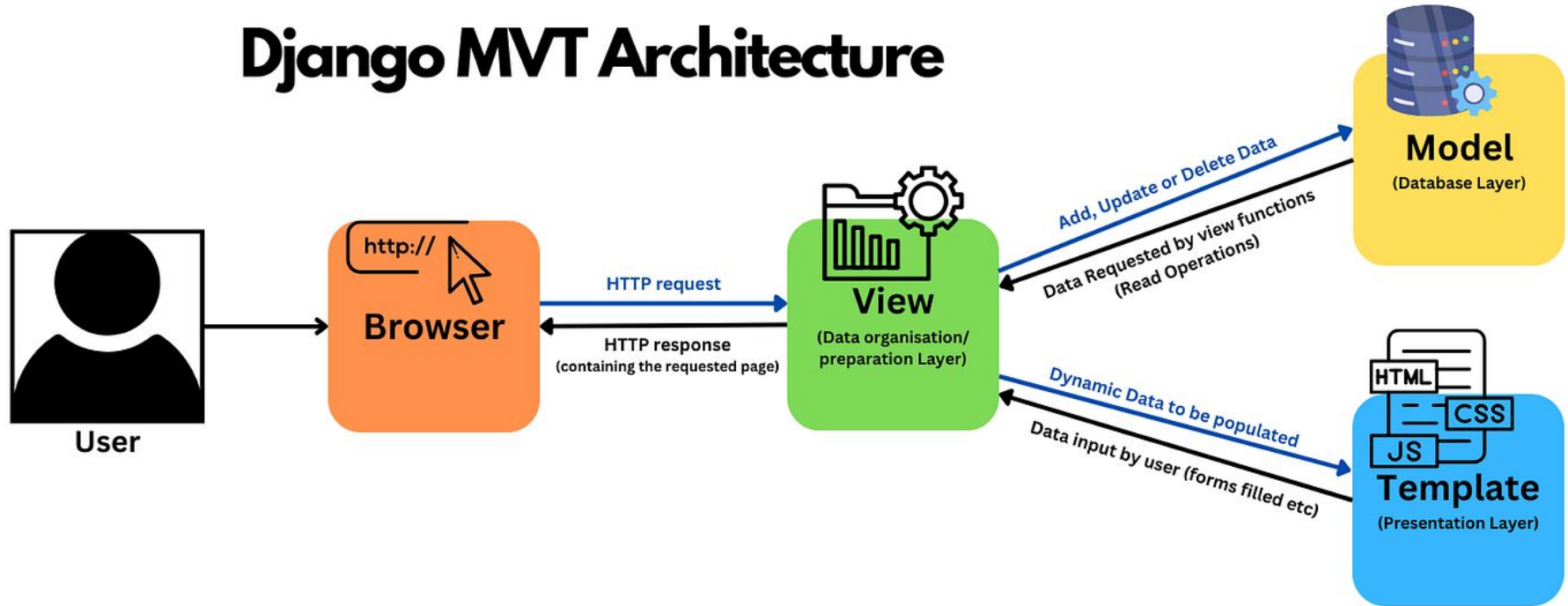


MVT (Model-View-Template)

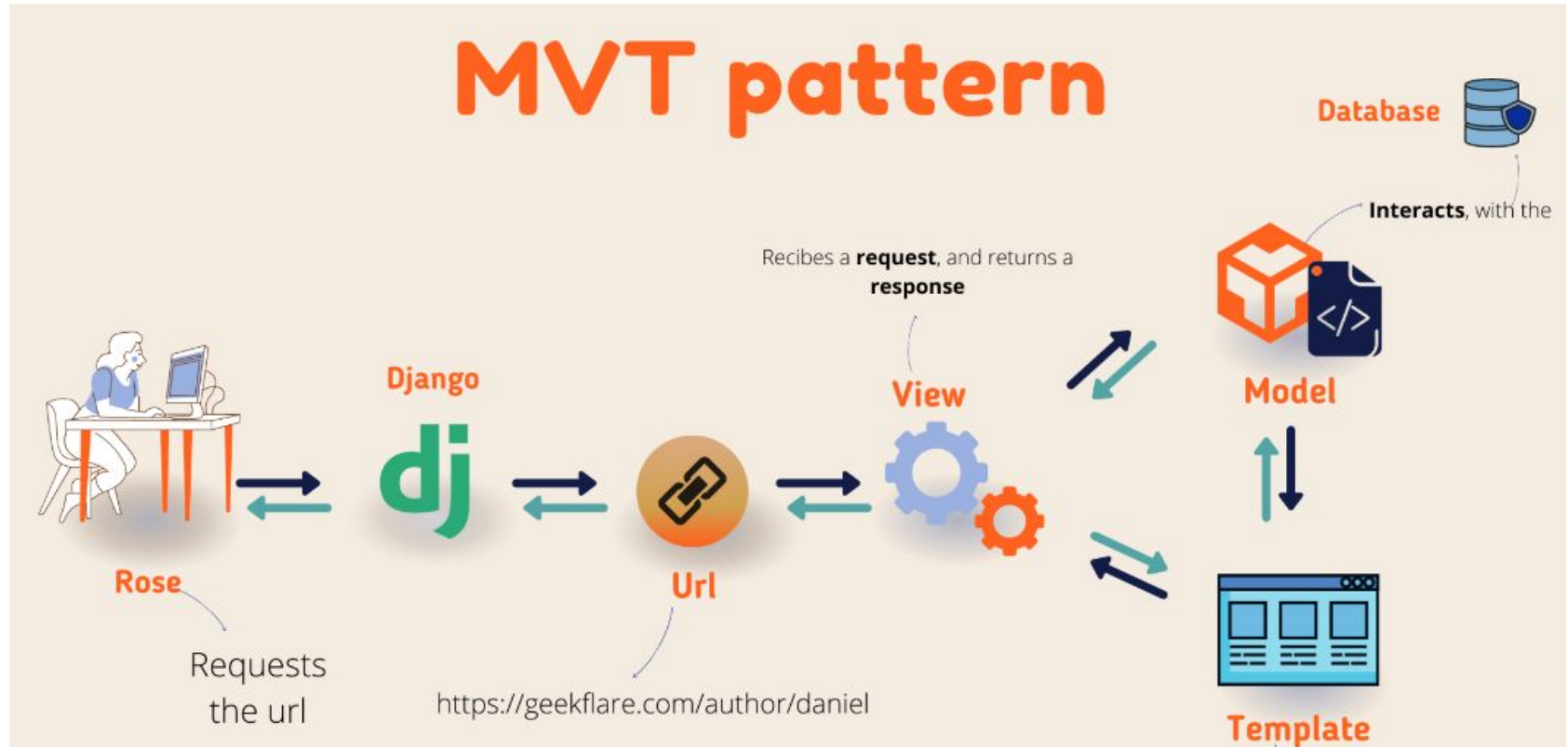
1. Modelo: Gestiona la estructura de datos mediante la interacción con una base de datos. El código del Modelo está escrito en Python y suele estar en el archivo `models.py`.
2. Vista: Gestiona la lógica de negocio. Por ejemplo, el filtrado de datos del Modelo para una página específica. El código de la Vista está escrito en Python y suele estar en el archivo [views.py](#).
3. Plantilla: Gestiona la capa de presentación para mostrar los datos proporcionados por la Vista en el navegador. Los archivos utilizados para la Plantilla están escritos en formato HTML. Puedes personalizar la estructura de archivos de la misma forma que gestionan documentos HTML típicos, pero Django proporciona un lenguaje adicional llamado DTL (Lenguaje de Plantillas de Django) que sirve de puente entre el código HTML y el de Python.

Arquitectura de Django (varios ejemplos)

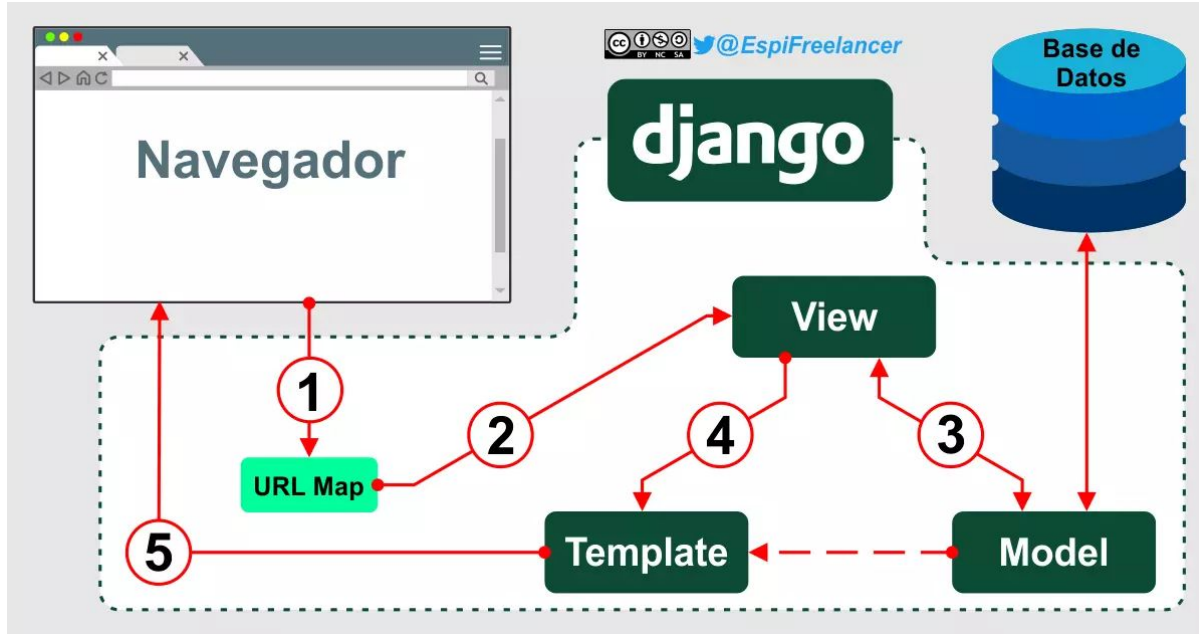
Django MVT Architecture



Qué es Django y para qué sirve?

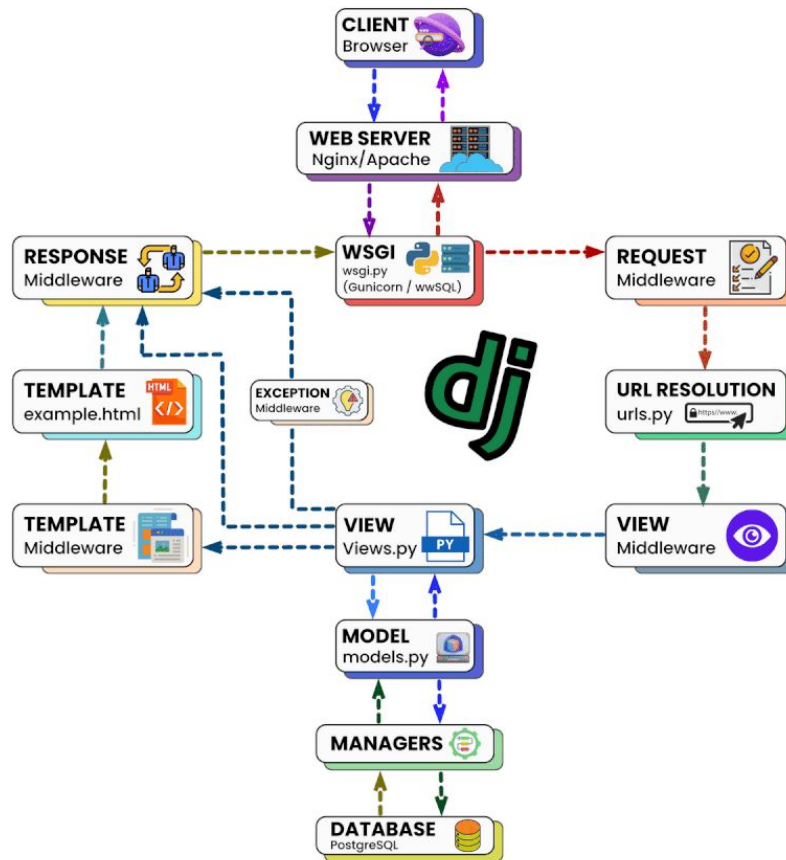


Qué es Django y para qué sirve?

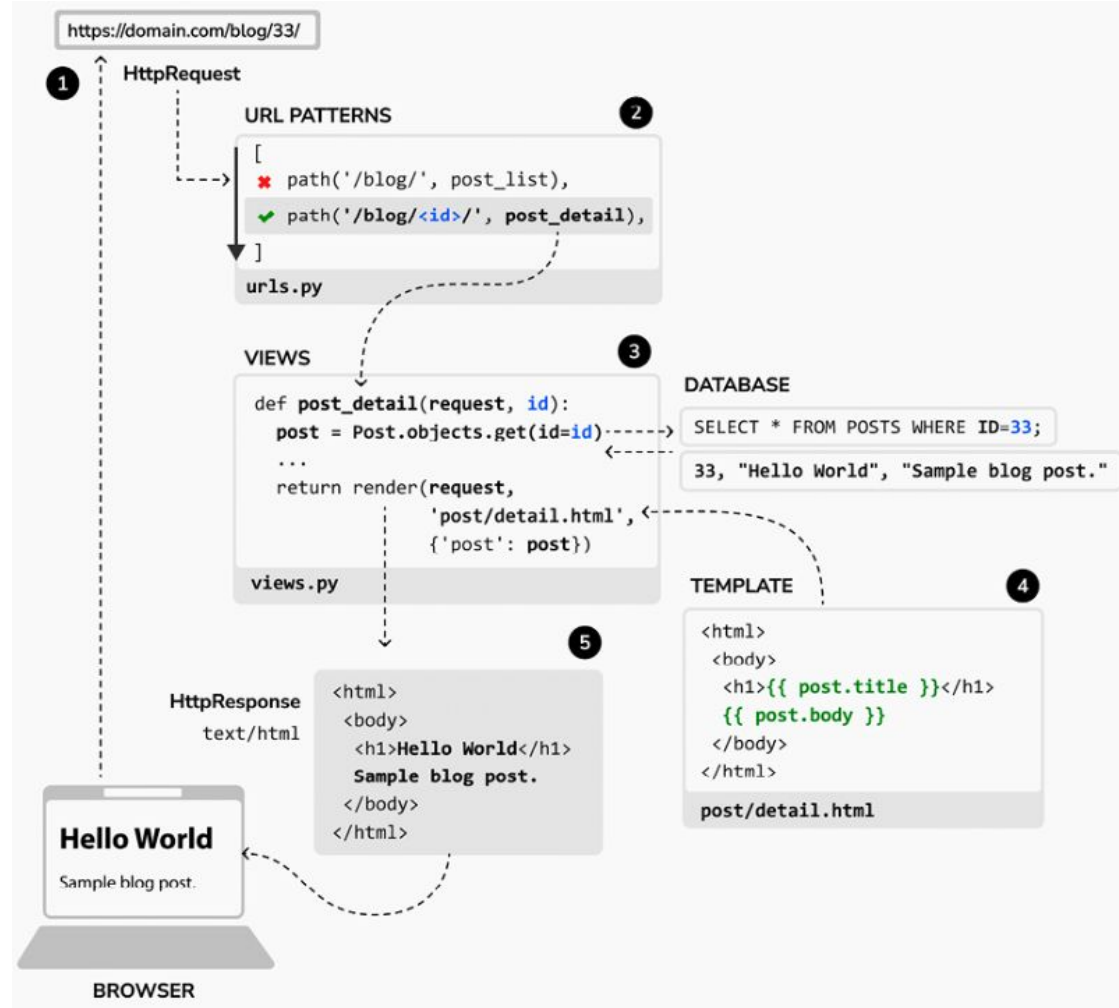




Django Request - Response Cycle



El ciclo de
solicitud/respuesta



ORM

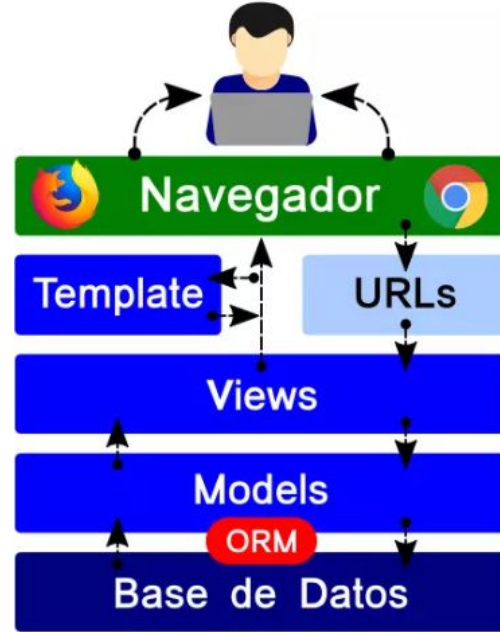
Un ORM (Object-Relational Mapping) es una técnica de programación que se utiliza en el desarrollo de aplicaciones para mapear los datos de una base de datos relacional a objetos en un lenguaje de programación orientado a objetos, como Java, Python o Ruby.



ORM

Definición 2

El Mapeo Objeto-Relacional (ORM) es una técnica que aborda este problema al proporcionar una capa de abstracción entre el código de la aplicación y la estructura de la base de datos. Esto aumenta la productividad, la portabilidad y la mantenibilidad de las aplicaciones. En el desarrollo de software, la integración entre las aplicaciones orientadas a objetos y las bases de datos relacionales ha sido tradicionalmente un desafío.



SQL

Para **hacer consultas** a una base de datos un programador **necesita escribir SQL**.



ORM

Un **ORM** **abstrae la base de datos** para que el programador haga consultas en el **lenguaje en el que está programando**, **sin necesitar SQL**.



django

SQL

```
CREATE TABLE  
table_name ( column1  
datatype, column2  
datatype, ...
```

Django ORM

```
class Page():  
    title - CharField  
    permalink - CharField  
    update_date - Timestamp  
    bodyText - TextField
```

```
# creando un modelo, clase pelicula  
class Pelicula (models.Model):  
    titulo = models.CharField(max_length= 25 )  
    director = models.CharField(max_length= 25 )  
    fecha_publicacion = models.DateField()
```

El modelo `models.py`

Relational database (such as PostgreSQL or MySQL)

ID	FIRST_NAME	LAST_NAME	PHONE
1	John	Connor	+16105551234
2	Matt	Makai	+12025555689
3	Sarah	Smith	+19735554512
...

Python objects

```
class Person:  
    first_name = "John"  
    last_name = "Connor"  
    phone_number = "+16105551234"
```

```
class Person:  
    first_name = "Matt"  
    last_name = "Makai"  
    phone_number = "+12025555689"
```

```
class Person:  
    first_name = "Sarah"  
    last_name = "Smith"  
    phone_number = "+19735554512"
```

ORMs provide a bridge between
relational database tables, relationships
and fields and Python objects

TABLAS

NOMBRE	EDAD	SEXO	PAÍS
ALEXIS	68	M	COLOMBIA 🇪🇸
PAULA	32	F	COLOMBIA 🇪🇸
BETO	34	M	BOLIVIA 🇧🇴
ALVARO	29	M	PERÚ 🇵🇪
GREICY	36	F	VENEZUELA 🇻🇪

CAMPOS

CLASE



FILAS

OBJETOS

PROPIEDADES



NOMBRE
EDAD
SEXO
PAÍS



NOMBRE
EDAD
SEXO
PAÍS



NOMBRE
EDAD
SEXO
PAÍS

Operaciones CRUD?



CREATE

READ

UPDATE

DELETE

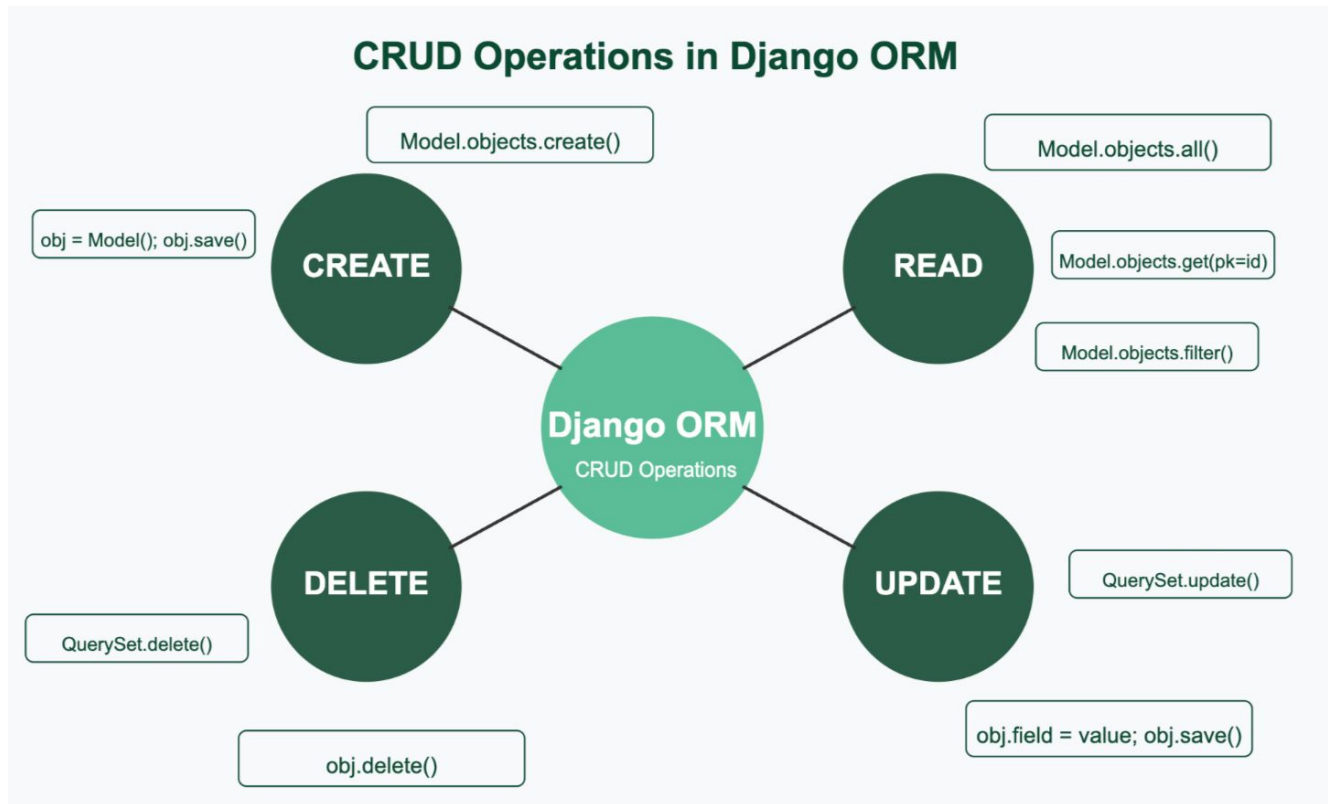
C

R

U

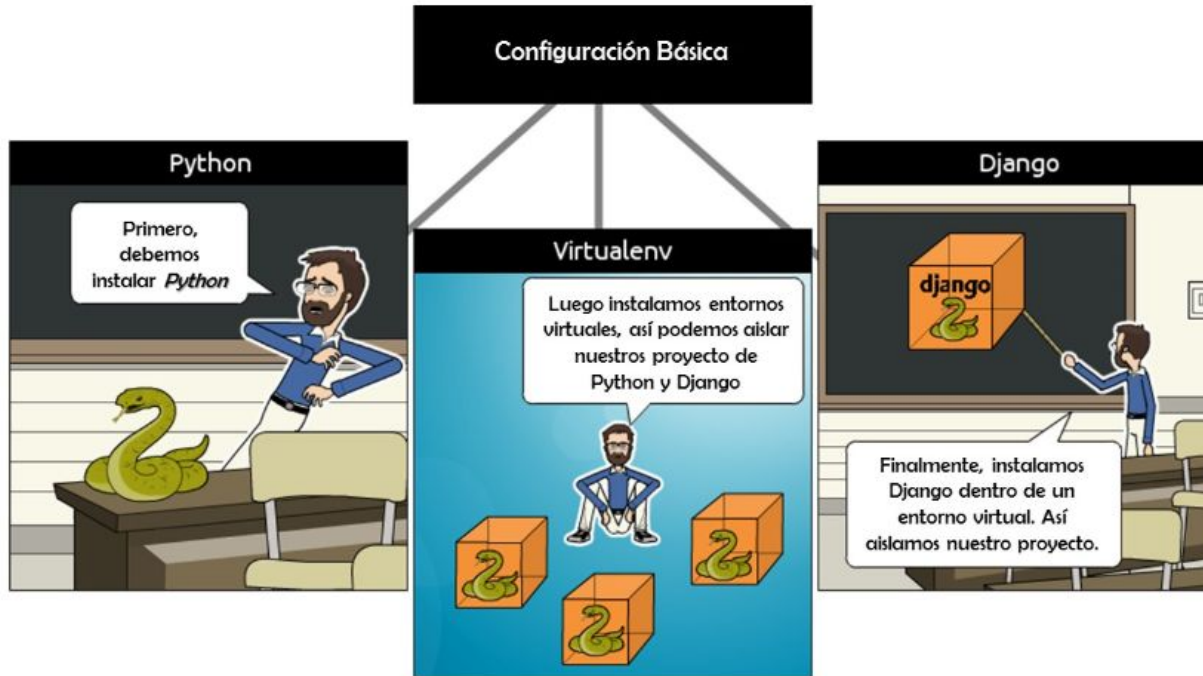
D

Cómo funciona el CRUD con Django ORM?



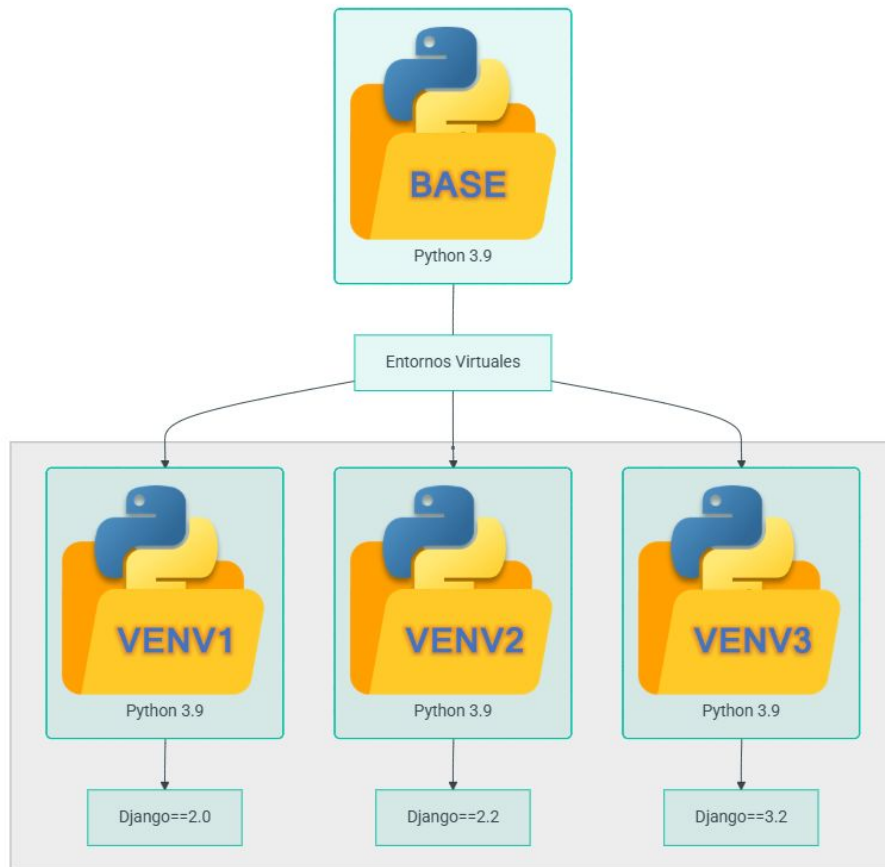
Procedimiento.

Instalación



Entorno virtual

Esquema de Entornos Virtuales de Python



Creación del entorno virtual

Crear carpeta de trabajo Ej: **C:\proyectos**

Crear subcarpeta practica1 Ej:
C:\proyectos\pratica1

Abrir carpeta **pratica1** desde **vs code**

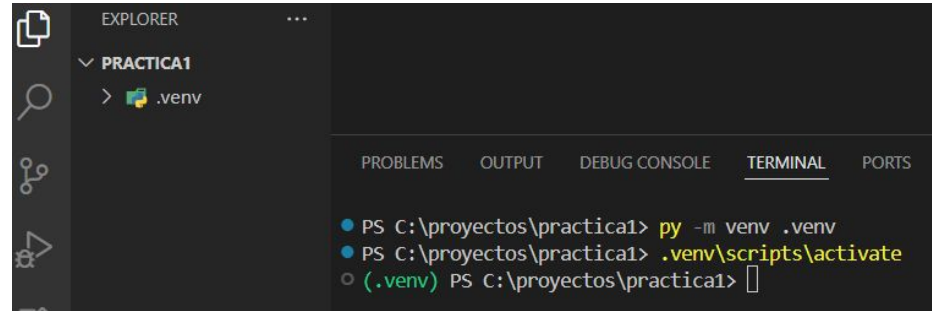
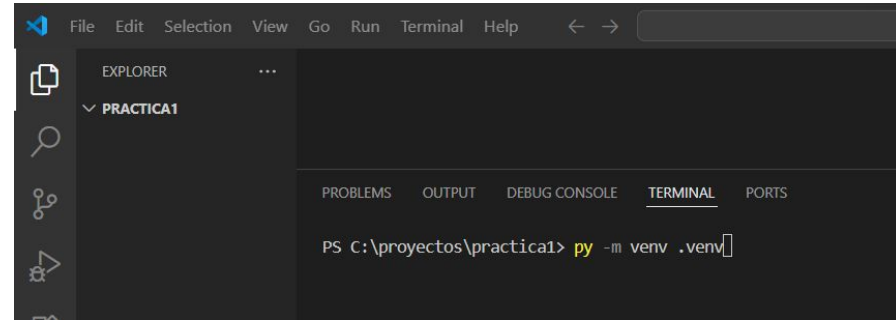
Abrir terminal **de vs code**

Creación del **entorno virtual** Ej:

```
PS C:\proyectos\practica1> py -m venv .venv
```

Activación del **entorno virtual** Ej:

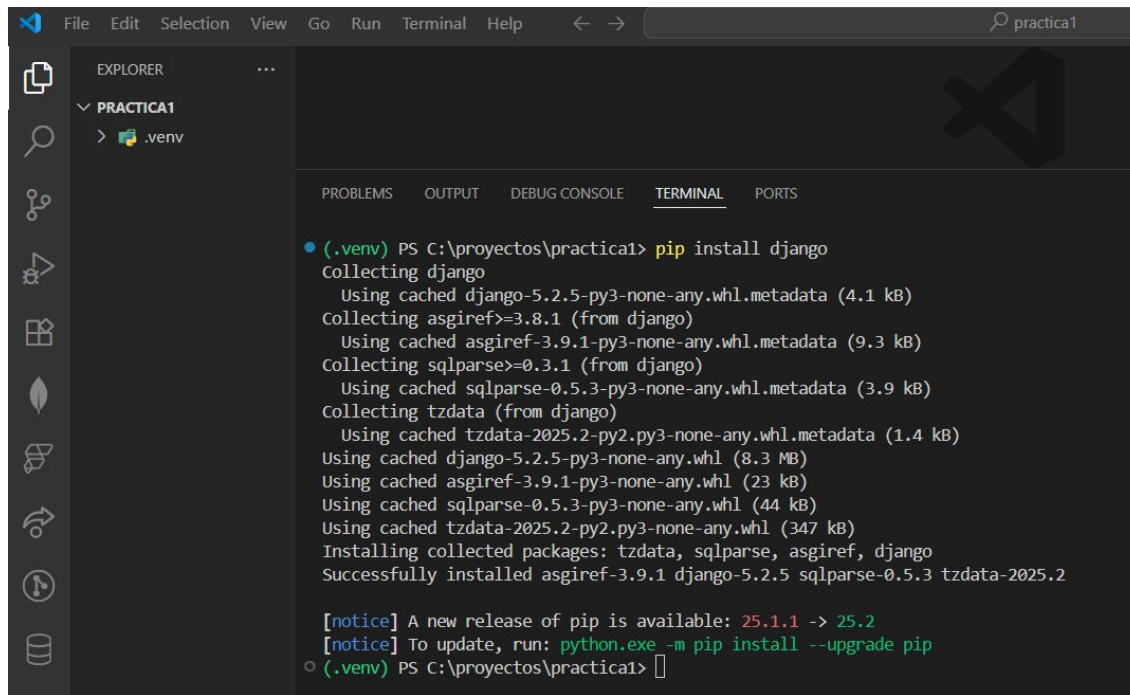
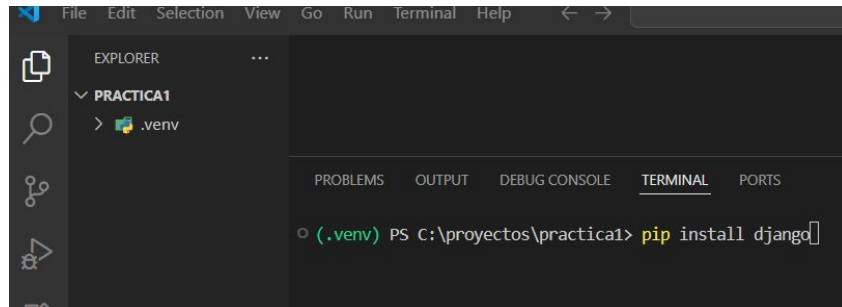
```
PS C:\proyectos\practica1> .venv\scripts\activate
```



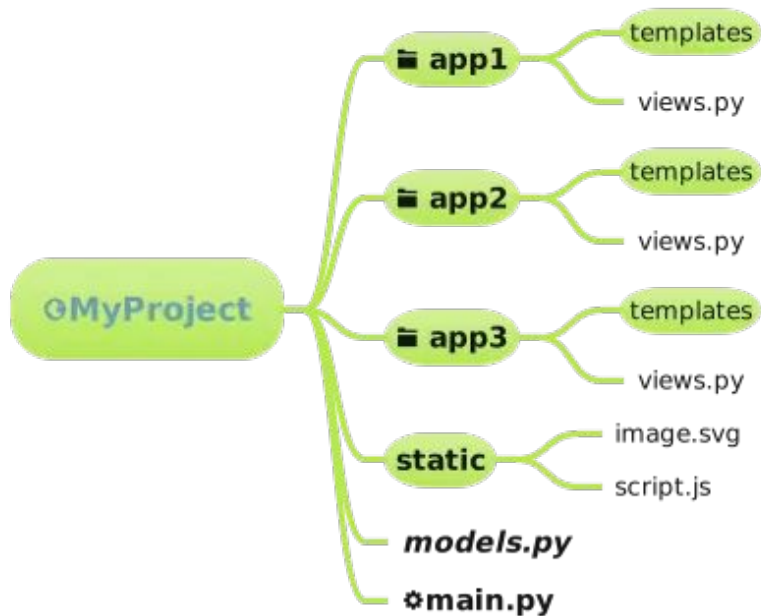
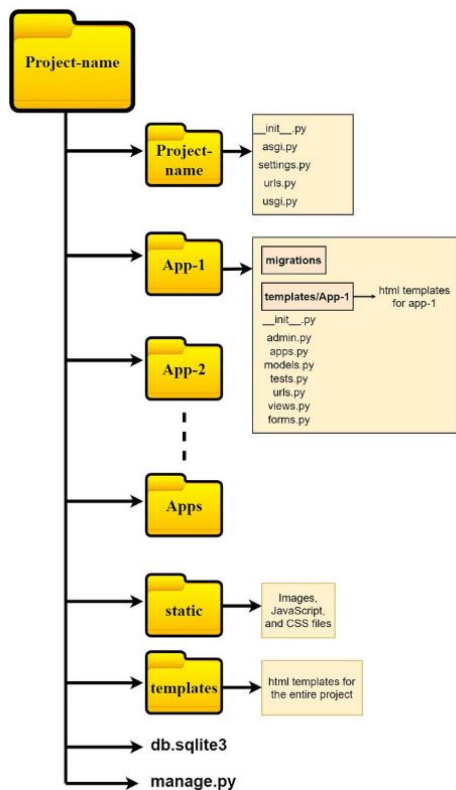
Instalación de Django

Instalando Django

(.venv) PS C:\proyectos\practica1> **pip install django**



Estructura de un proyecto en Django.

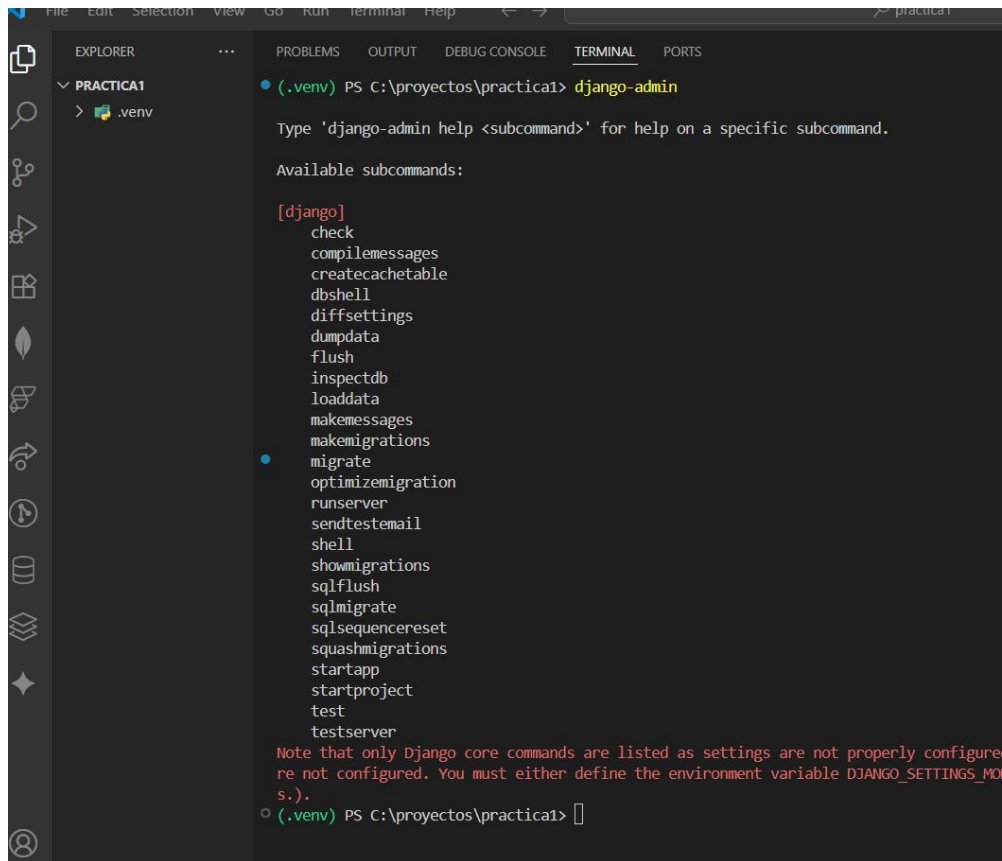


FastAPI Project structure

Comandos de django-admin



Configuración inicial y personalización del proyecto.



The screenshot shows a Visual Studio Code interface with a terminal window open. The terminal is running a command prompt in a virtual environment (.venv) at the path C:\proyectos\practica1. The command 'django-admin' has been entered, and the output displays the available subcommands for Django. The subcommands listed are: check, compilemessages, createcachetable, dbshell, diffsettings, dumpdata, flush, inspectdb, loaddata, makemessages, makemigrations, migrate, optimizemigration, runserver, sendtestemail, shell, showmigrations, sqlflush, sqlmigrate, sqlsequencereset, squashmigrations, startapp, startproject, test, and testserver. A note at the bottom states that only Django core commands are listed because settings are not properly configured, and suggests defining the environment variable DJANGO_SETTINGS_MODULE. The terminal prompt is currently at C:\proyectos\practica1>.

```
File Edit Selection View GO Run Terminal Help
EXPLORER PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PRACTICA1
> .venv

(.venv) PS C:\proyectos\practica1> django-admin

Type 'django-admin help <subcommand>' for help on a specific subcommand.

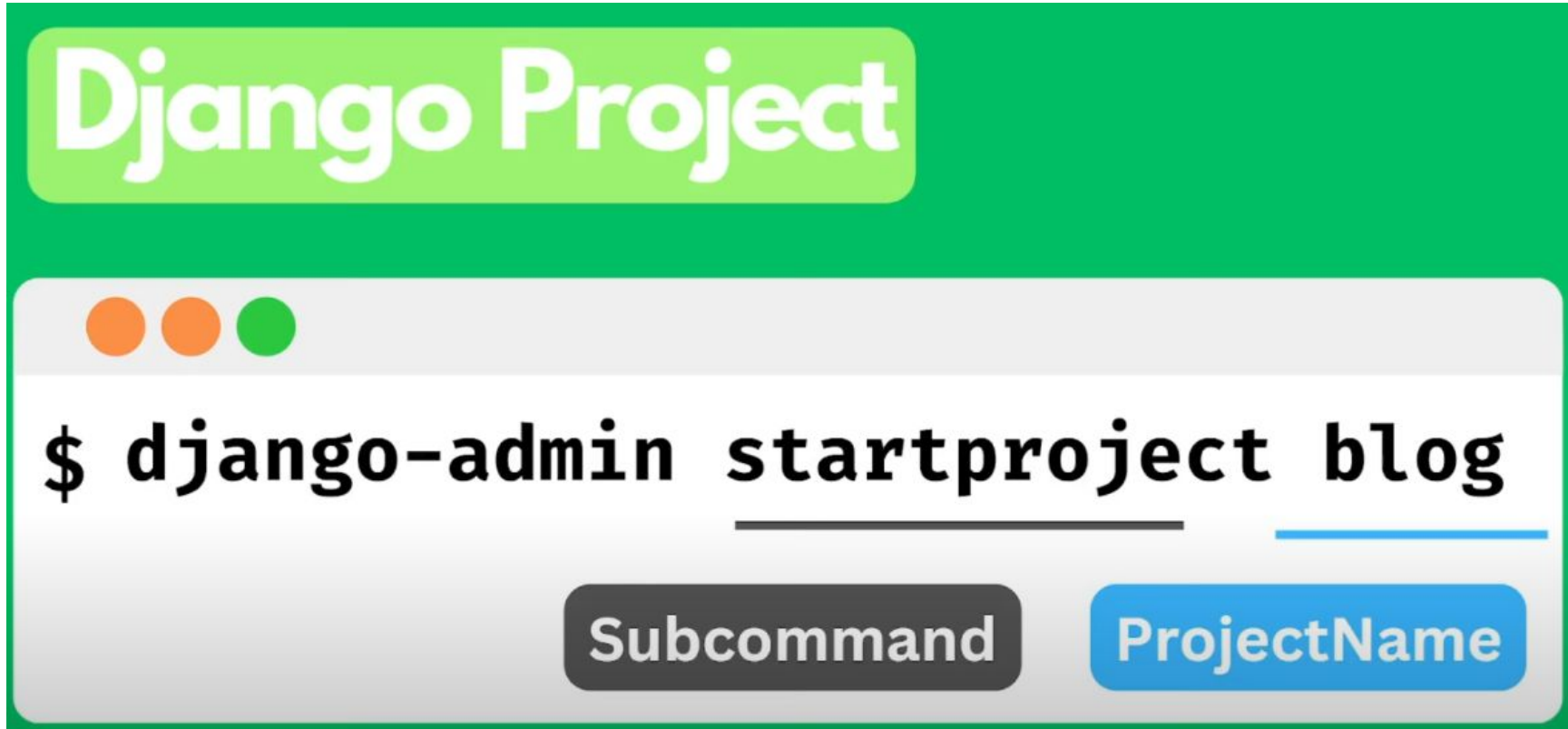
Available subcommands:

[django]
  check
  compilemessages
  createcachetable
  dbshell
  diffsettings
  dumpdata
  flush
  inspectdb
  loaddata
  makemessages
  makemigrations
  migrate
  optimizemigration
  runserver
  sendtestemail
  shell
  showmigrations
  sqlflush
  sqlmigrate
  sqlsequencereset
  squashmigrations
  startapp
  startproject
  test
  testserver

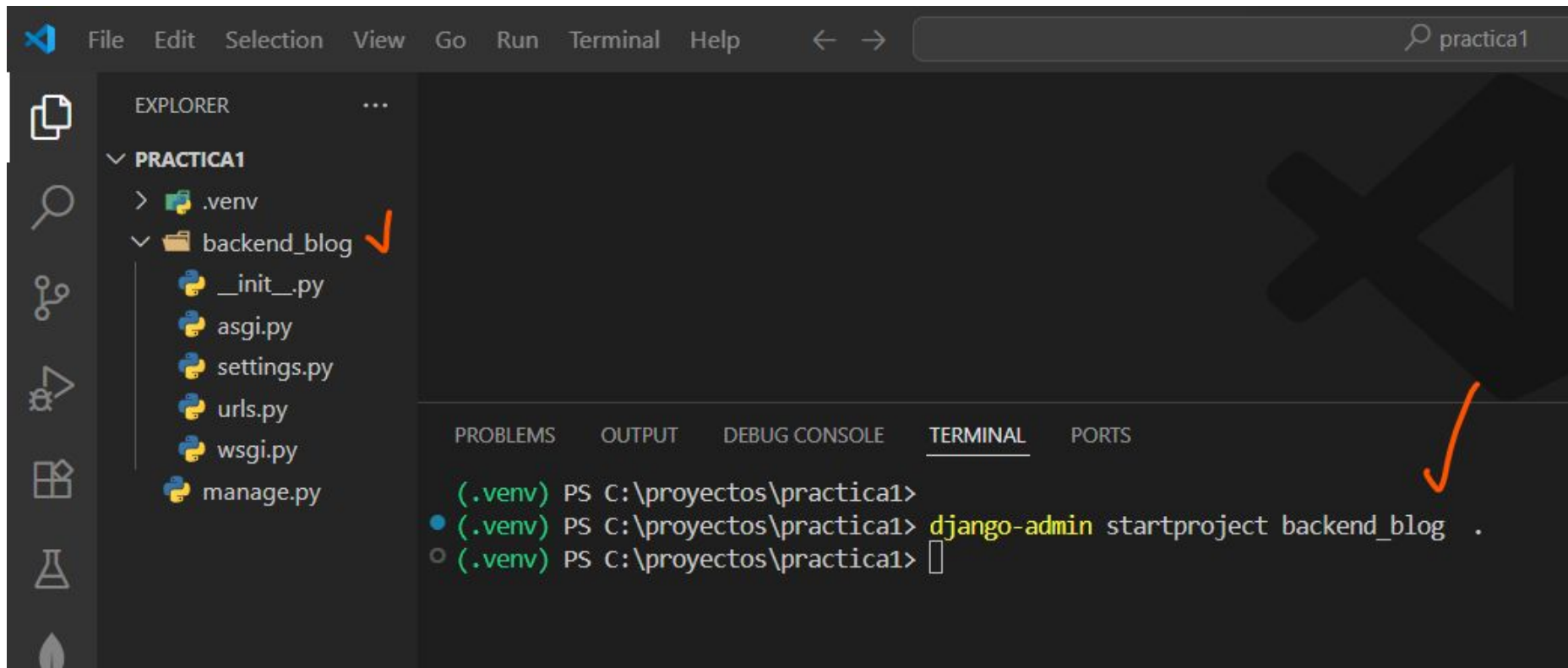
Note that only Django core commands are listed as settings are not properly configured.
You must either define the environment variable DJANGO_SETTINGS_MODULE or set the
DJANGO_SETTINGS_MODULE environment variable in your environment file (e.g. .env).

(.venv) PS C:\proyectos\practica1>
```

Ejemplo para crear un proyecto en Django



Ejemplo: para crear un proyecto en Django





Estructura general de un proyecto Django

Cuando creas un proyecto con **django-admin startproject <nombre_proyecto>**, la estructura básica que obtienes es algo como esto:

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

Para usar un comando ([manage.py](#)) relacionado con nuestro proyecto django

Django Project



```
$ manage.py
```

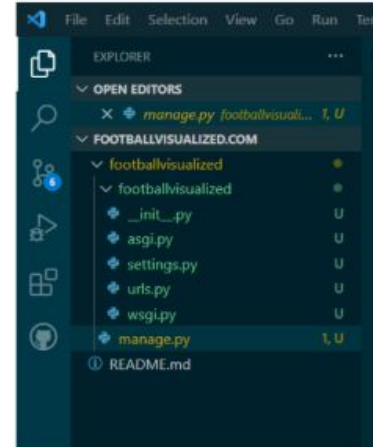
```
$ to give commands related to our django  
project.
```

Mas sobre [manage.py](#)

Archivos y carpetas importantes

manage.py:

- Es un script que permite interactuar con el proyecto Django desde la línea de comandos. Usas este archivo para ejecutar comandos como iniciar el servidor (runserver), migrar la base de datos (migrate), crear aplicaciones (startapp), entre otros.

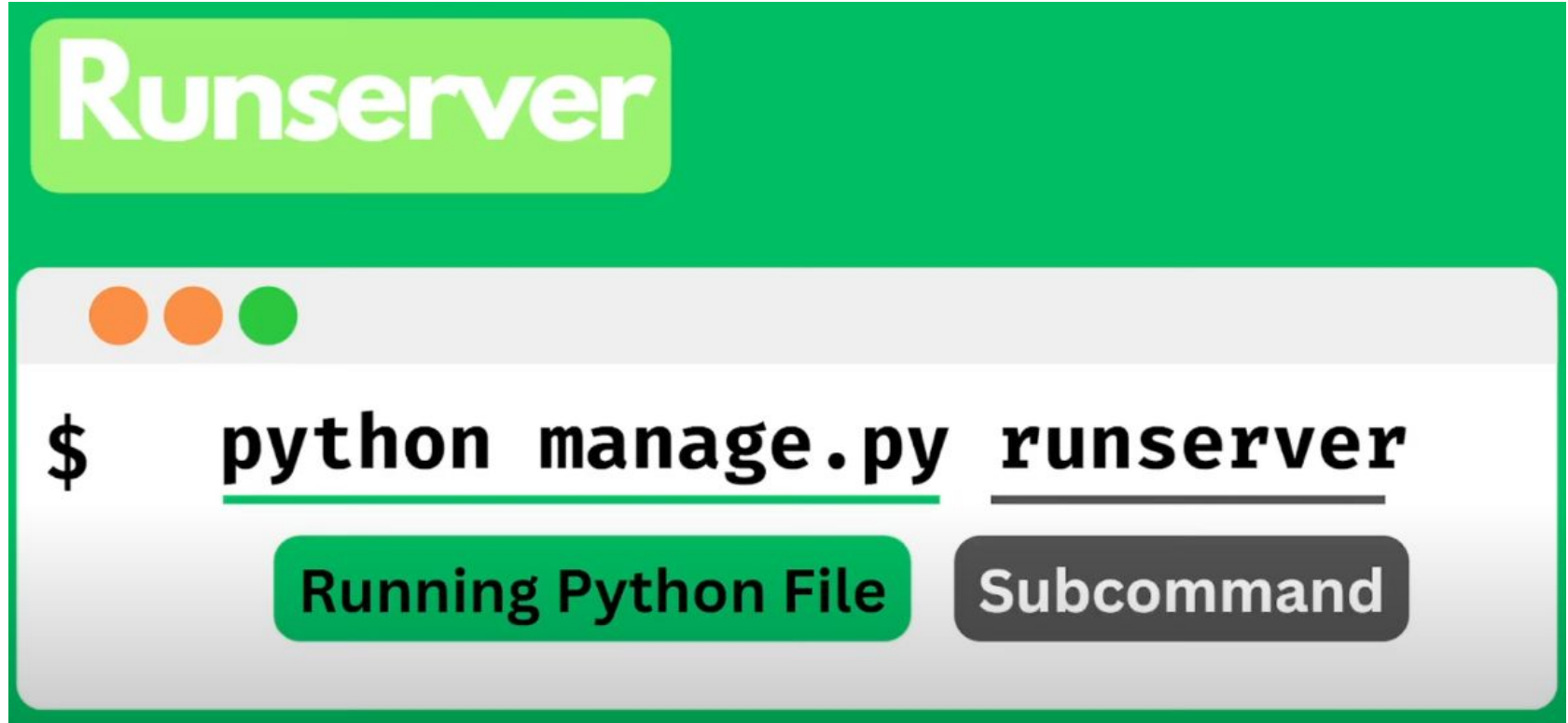


Archivos y carpetas importantes

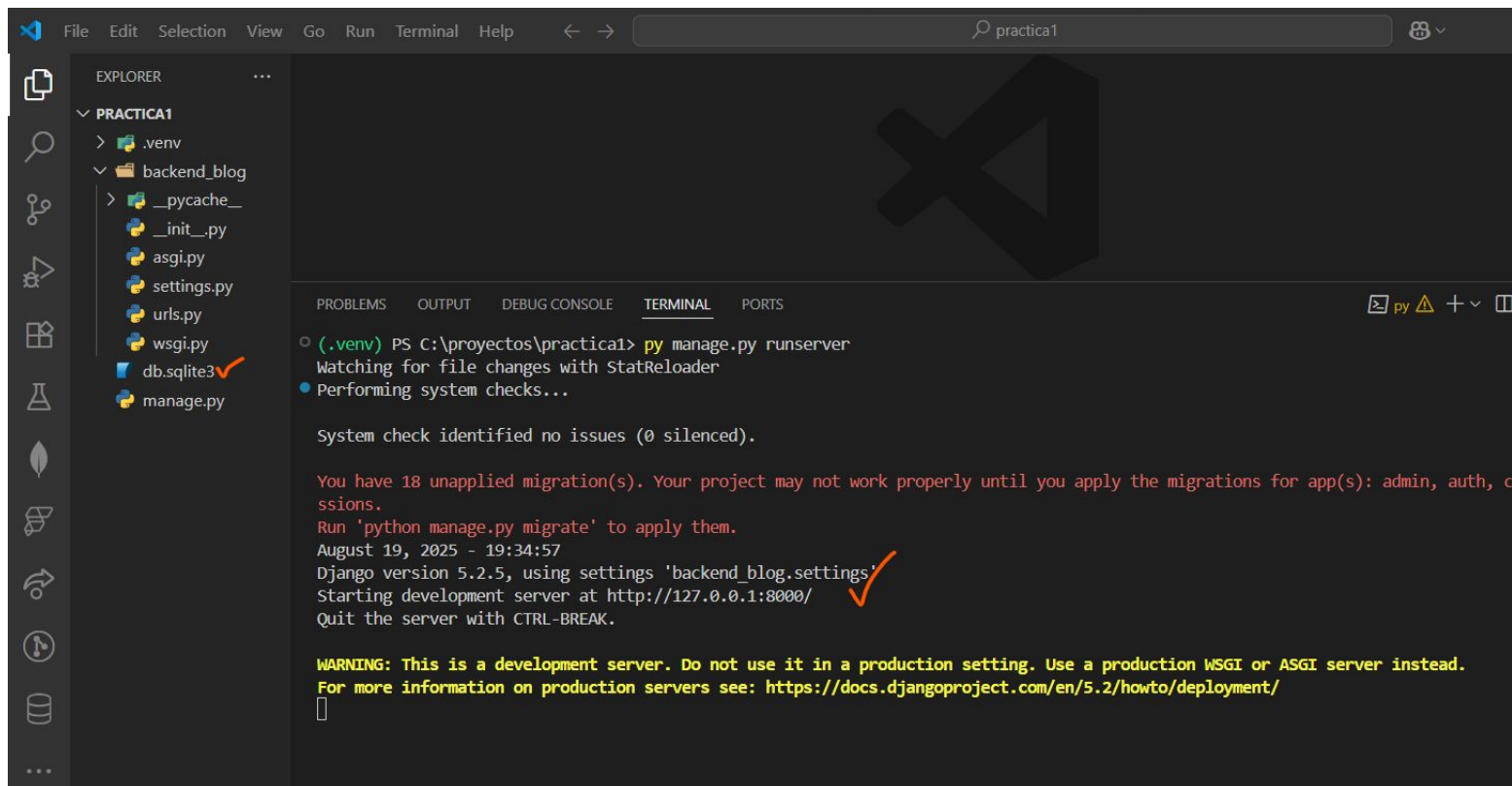
Carpeta `mi_proyecto/` (o el nombre de tu proyecto):

- Esta carpeta contiene la configuración principal del proyecto. Dentro de ella están varios archivos clave:
 - **`__init__.py`**: Indica que esta carpeta debe tratarse como un módulo de Python. Por lo general, está vacío.
 - **`settings.py`**: Contiene toda la configuración del proyecto, como las bases de datos, las aplicaciones instaladas, las configuraciones de seguridad, rutas de archivos estáticos, entre otros. Este es uno de los archivos más importantes.
 - **`urls.py`**: Aquí defines las rutas (URLs) que manejará tu proyecto. Es el "mapeo" entre las URL solicitadas por el navegador y las vistas (views) que deben procesarlas.
 - **`wsgi.py` y `asgi.py`**: Son archivos para la implementación del servidor web. `wsgi.py` es para el protocolo WSGI y `asgi.py` para ASGI. Son usados cuando el proyecto se despliega en un entorno de producción.

Comando para Ejecutar Servidor web (baterías incluidas)



Ejecutando Servidor web (baterías incluidas)



```
File Edit Selection View Go Run Terminal Help  
practica1  
EXPLORER  
PRACTICA1  
  .venv  
  backend_blog  
    __pycache__  
    __init__.py  
    asgi.py  
    settings.py  
    urls.py  
    wsgi.py  
    db.sqlite3  
    manage.py  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
(.venv) PS C:\proyectos\practica1> python manage.py runserver  
Watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
  
You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, c  
ssions.  
Run 'python manage.py migrate' to apply them.  
August 19, 2025 - 19:34:57  
Django version 5.2.5, using settings 'backend_blog.settings'.  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CTRL-BREAK.  
  
WARNING: This is a development server. Do not use it in a production setting. Use a production WSGI or ASGI server instead.  
For more information on production servers see: https://docs.djangoproject.com/en/5.2/howto/deployment/  
[]
```

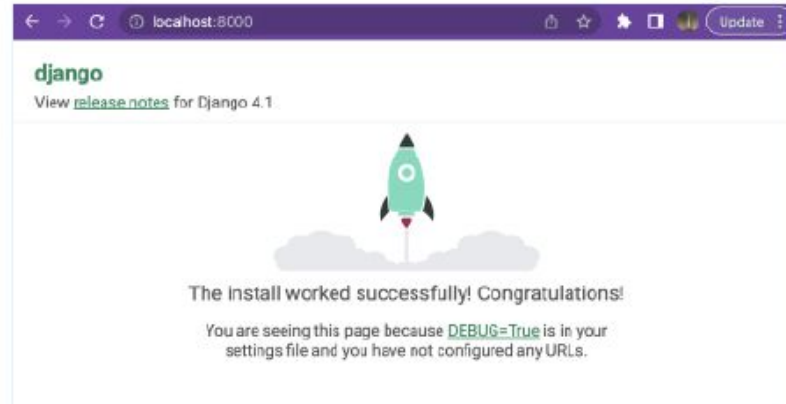
Servidor web

```
python manage.py runserver
```

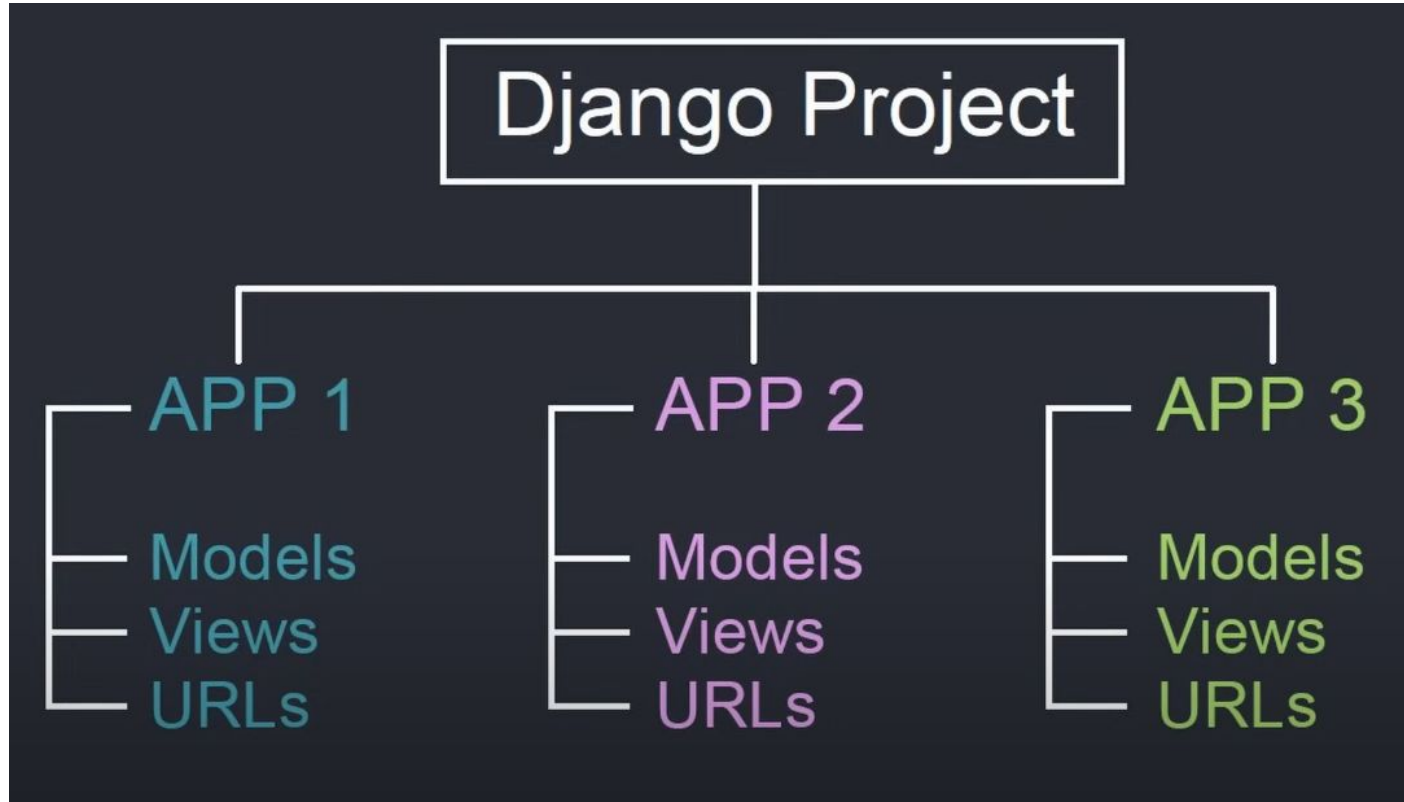
Run the command
from the project
directory...

```
$ python manage.py  
runserver
```

Type **localhost:8000** in a web browser



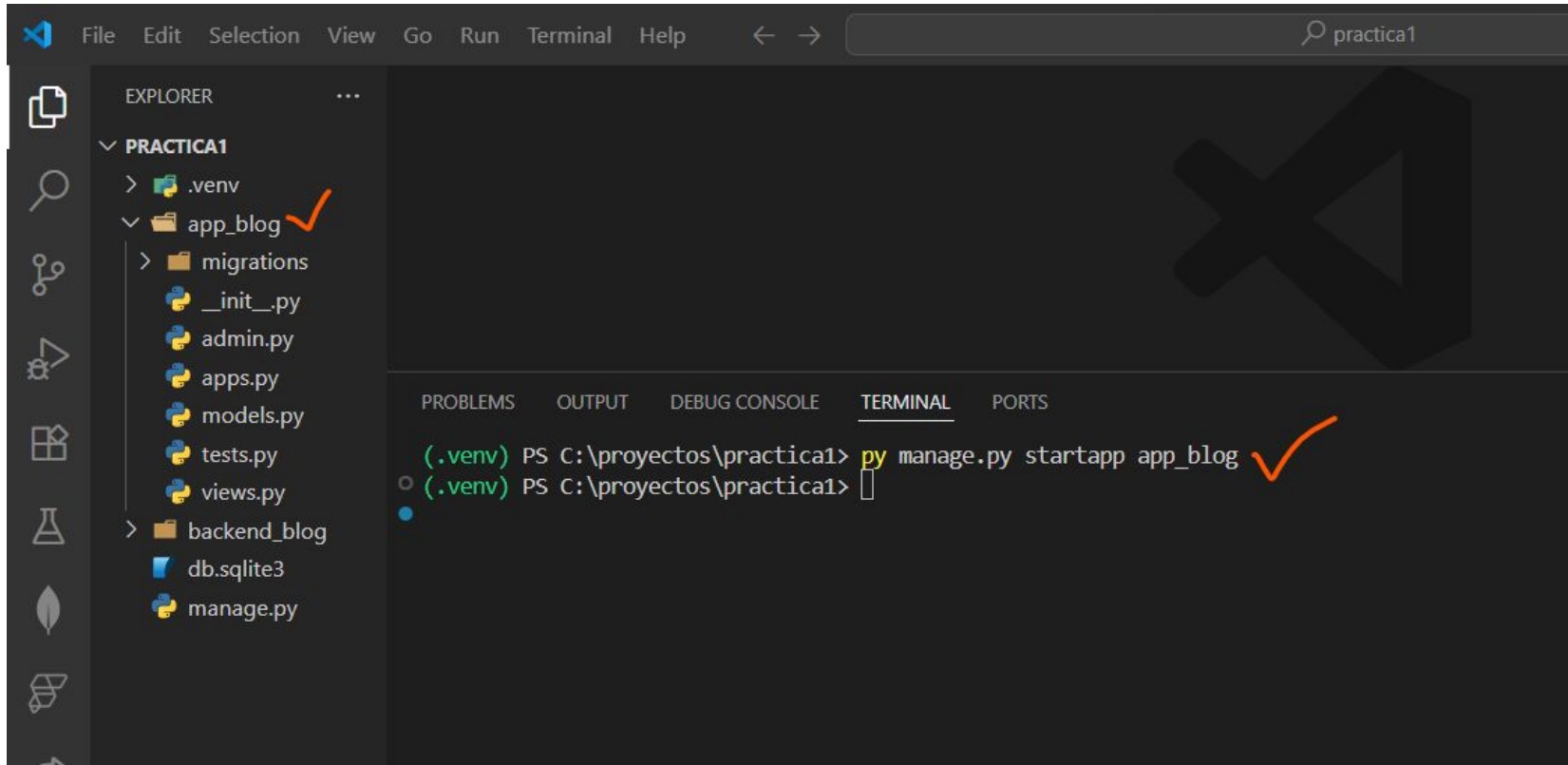
Creación de aplicaciones Django



Ejemplo Proyecto backend_carro



Creando la aplicación app_blog



Archivos al crear una aplicación

Archivos clave en una aplicación

migrations/:

- Esta carpeta contiene los archivos de migración que se generan cuando haces cambios en los modelos de la aplicación (por ejemplo, cuando añades o modificas campos en la base de datos). Las migraciones aseguran que la base de datos esté sincronizada con los modelos del proyecto.

__init__.py:

- Como en la carpeta del proyecto, este archivo indica que la carpeta de la aplicación es un módulo de Python.

admin.py:

- Archivo donde se deben registrar los modelos para que aparezcan en la interfaz de administración de Django.

Archivos al crear una aplicación

Archivos clave en una aplicación

apps.py:

- Define la configuración de la aplicación. Por lo general, se utiliza para darle un nombre a la app y otras configuraciones específicas.

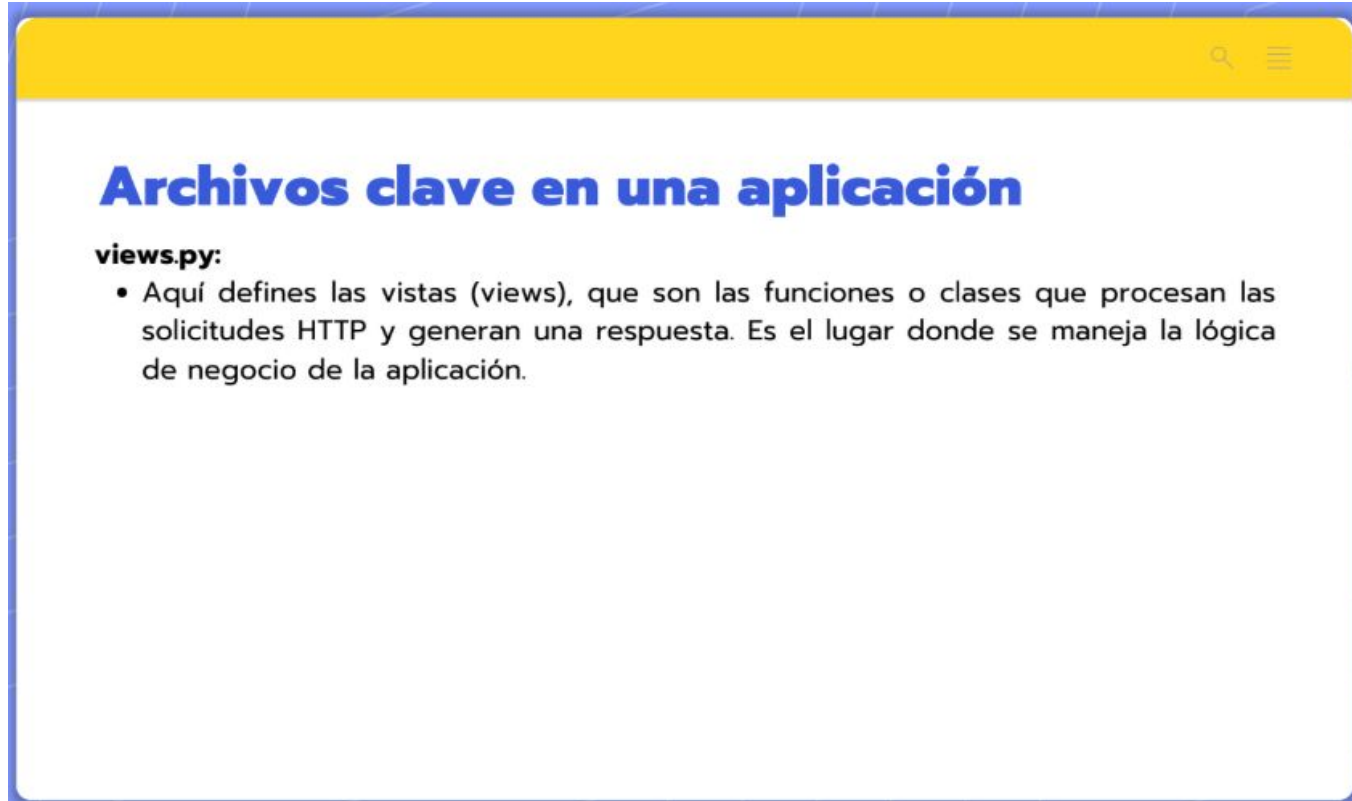
models.py:

- Aquí defines los modelos de la base de datos que se guardan por medio del método save(). Un modelo en Django es una representación de una tabla de base de datos y cada campo en el modelo corresponde a una columna en esa tabla.

tests.py:

- Archivo donde defines las pruebas unitarias (tests) de la aplicación para asegurarte de que todo funcione correctamente.

Archivos al crear una aplicación



The image shows a presentation slide with a yellow header bar containing a magnifying glass icon and a hamburger menu icon. The main content area has a blue title and a section header for `views.py`, followed by a bulleted list.

Archivos clave en una aplicación

views.py:

- Aquí defines las vistas (views), que son las funciones o clases que procesan las solicitudes HTTP y generan una respuesta. Es el lugar donde se maneja la lógica de negocio de la aplicación.

Resumen

Resumen de la estructura

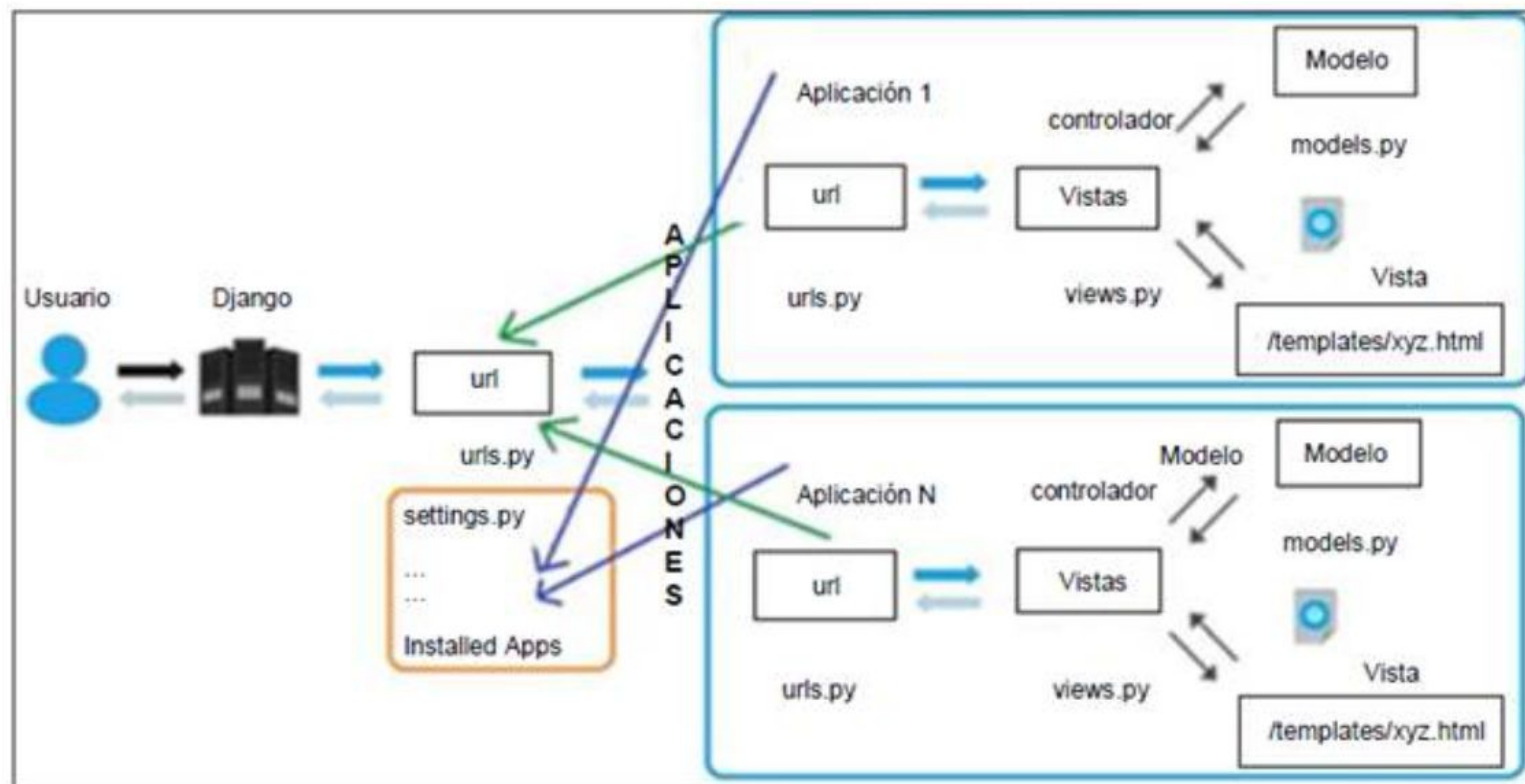
- **Proyecto Django:** Contiene archivos de configuración global como settings.py y urls.py.
- **Aplicaciones Django:** Son módulos independientes con su propia lógica, incluyendo modelos, vistas y migraciones.

Los urls.py

El despachador de URL es uno de los conceptos clave en la arquitectura de Django.

Este despachador asigna las solicitudes HTTP a las funciones de vista (o clases) especificadas en los patrones de URL escritos en el archivo urls.py.

Los urls.py



Maapeo de rutas y vistas

URLs & Views ?

home/

Home Page

about/

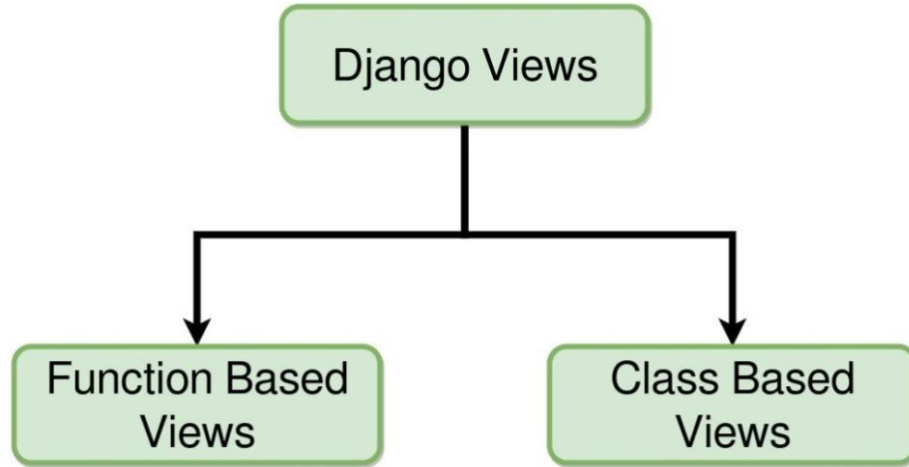
About Page

post/postid/

Single Post Page

urls	views
inicio	pagina_inicio
Acerca_de	página_acerca_de
publicacion/id	pagina_publicacion

vistas basadas en funciones y clases




Vistas basadas en funciones (request y response)

Syntax - FBVs

```
def helloWorld(request):  
    return HttpResponse("Hello World")
```

Uso de comandos frecuentes en Django



```
#Creates boilerplate django files
django-admin startproject <project-name>

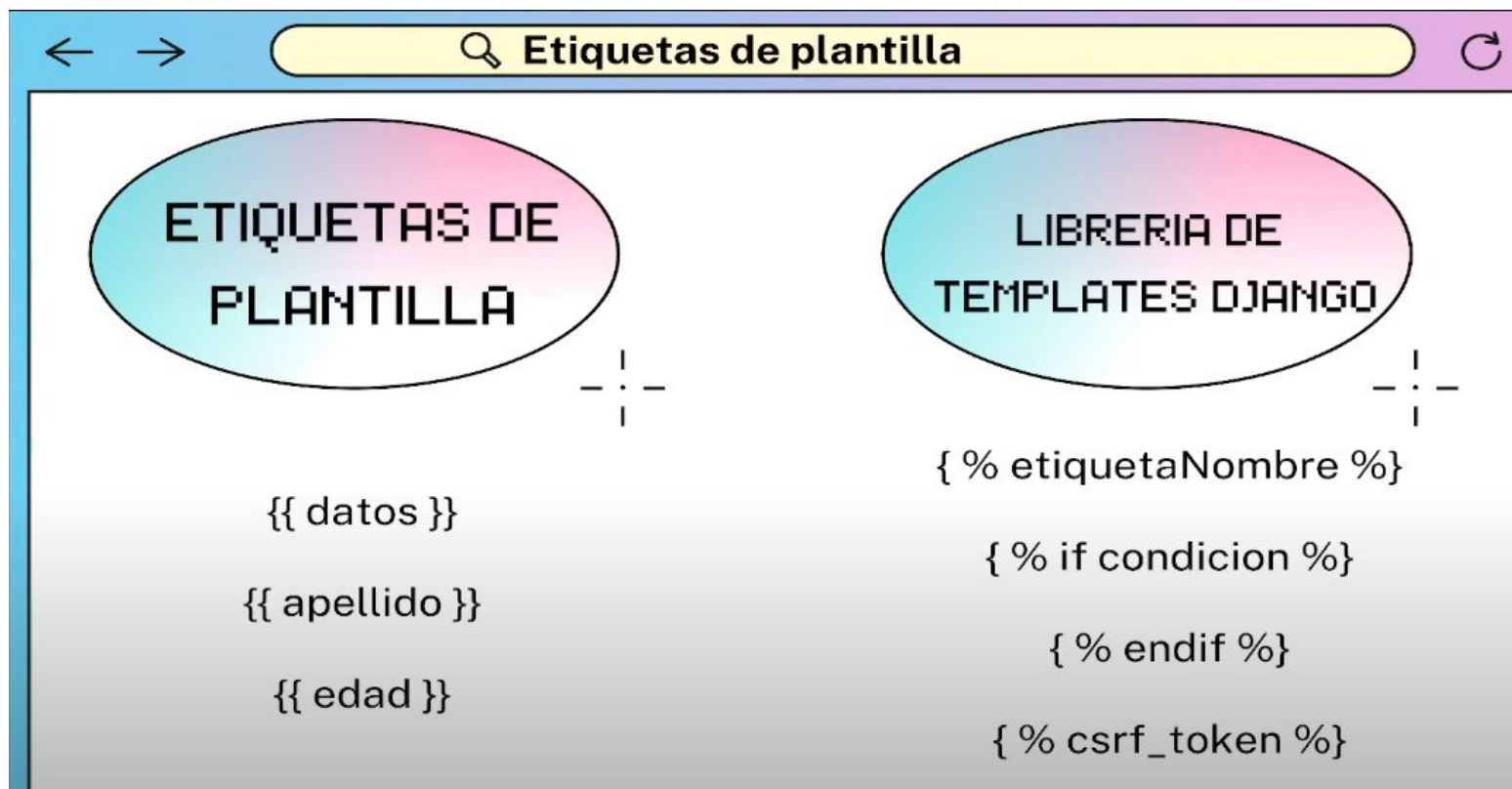
#Creates app folder and files
python manage.py startapp <app-name>

#Preps our database for migrations
python manage.py makemigrations

#Executes our migrations & updates the database
python manage.py migrate

#Creates a user with admin level permissions
python manage.py createsuperuser
```

templates



templates

Plantillas de Django: la guía de referencia completa

entendamos las plantillas de Django.

Para crear una aplicación web, necesitarás tanto el **front-end** como el **back-end**.

DTL – Lenguaje de Plantillas de Django

Con las Plantillas de Django, un desarrollador front-end no necesita aprender Python, ni un programador back-end necesita saber HTML.

Un desarrollador front-end puede simplemente dejar comentarios HTML (donde desee la base de datos y otra información de Django). Posteriormente, un programador puede reemplazarlos con un lenguaje de plantillas, conocido como Lenguaje de Plantillas de Django (DTL).

templates Tag

Un **desarrollador front-end** puede simplemente dejar comentarios HTML (donde desee la base de datos y otra información de Django).

Posteriormente, **un programador** puede reemplazarlos con un lenguaje de plantillas, conocido como Lenguaje de Plantillas de Django (DTL).

DTL tiene una ventaja sobre otros debido a su

- Simplicidad
- Fácil de aprender la sintaxis
- Extensible

templates Tag (Etiquetas de plantilla)

Estas **etiquetas de plantilla** cumplen una función. Esta frase puede ser difícil de entender, pero te harás una idea después de ver los ejemplos, así que no te preocupes.

Nota: Una etiqueta de plantilla se encierra entre `{% y %}`. Algunos ejemplos son:

Declaraciones de condición/Lógica de visualización: `{% if %} ... {% endif %}`

Bucles: `{% for x in y %} ... {% endfor %}`

Declaración de bloque: `{% block content %} ... {% endblock %}`

Importación de contenido: `{% include "header.html" %}`

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

templates tag (Etiquetas de variables)

Las variables de plantilla son similares a las variables utilizadas en Python.

Variable simple → `{{ title }}`, `{{ x }}`

Atributos de lista → `{{ fruits_list.0 }}`

Atributos de objeto → `{{ name.title }}`

Atributo de diccionario → `{{ dict.key }}`

Nota: Aquí, `list.0` se usa, a diferencia de `list[0]` en Python, para acceder al primer elemento de una lista de Python.

templates Ejemplo modelo, vista plantilla

```
from django.db import models

class Empleado(models.Model):
    nombre = models.CharField(max_length=200) # Nombre completo
    id_empleado = models.CharField(max_length=200) # Código único
    telefono = models.CharField(max_length=10) # Número de teléfono
    direccion = models.CharField(max_length=150) # Dirección del empleado
    trabaja = models.BooleanField(default=True) # Estado activo/inactivo
    departamento = models.CharField(max_length=200) # Área de trabajo
```

```
from .models import Empleado

# Página de inicio que lista empleados activos
def inicio_empleado(request):
    empleados = Empleado.objects.filter(trabaja=True)
    return render(request, 'inicio.html', {'empleados': empleados})
```

templates Ejemplo modelo, vista plantilla

Student.objects.all()

Select * from Student

Application (Python)

ORM

(Python to SQL)

Database

```
graph TD; A[Application (Python)] <--> B[ORM (Python to SQL)]; B <--> C[Database]; B --> D[Select * from Student];
```

templates Ejemplo modelo, vista plantilla

```
<table class="table table-hover table-bordered bg-white">
  <thead class="table-primary">
    <tr>
      <th>Nombre</th>
      <th>ID</th>
      <th>Teléfono</th>
      <th>Dirección</th>
      <th>Departamento</th>
      <th>Acciones</th>
    </tr>
  </thead>
  <tbody>
    {% for empleado in empleados %}
      <tr>
        <td>{{ empleado.nombre }}</td>
        <td>{{ empleado.id_empleado }}</td>
        <td>{{ empleado.telefono }}</td>
        <td>{{ empleado.direccion }}</td>
        <td>{{ empleado.departamento }}</td>
        <td>
          <a href="{% url 'actualizar_empleado' empleado.id_empleado %}" class="btn btn-warning btn-sm">Editar</a>
          <a href="{% url 'borrar_empleado' empleado.id_empleado %}" class="btn btn-danger btn-sm"
            onclick="return confirm('¿Seguro que deseas eliminar este empleado?')">Borrar</a>
        </td>
      </tr>
    {% empty %}
      <tr>
        <td colspan="6" class="text-center text-muted">No hay empleados registrados</td>
      </tr>
    {% endfor %}
  </tbody>
</table>
```

templates

DJANGO TEMPLATES

STATIC FILES

- Para agregar la imagen al template agregamos:

```
<!DOCTYPE html>
{% load static %}

<html>
  <head>
    <meta charset="utf-8">
    <title>Información de Usuario</title>
  </head>
  <body>
    <h1>Información del Usuario</h1>
    
    <ul>
      <li>ID: {{id}}</li>
      <li>Nombre: {{nombre}}</li>
      <li>Email: {{email}}</li>
    </ul>
  </body>
</html>
```

← Incorporamos los
archivos estáticos

← Usamos la
referencia dentro
de la imagen



<http://127.0.0.1:8000/info/>

templates

USO DE TEMPLATES DATA

```
from django.shortcuts import render

# Create your views here.
def renderTemplate(request):
    data = {"nombre" : "Paul"}
    return render(request, 'templatesApp/firstTemplate.html', data)
```

views.py

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Primer Template</title>

  </head>
  <body>
    <h1>Templates en Acción!</h1>
    <h2>Hola {{nombre}}!! </h2>
  </body>
</html>
```

templatesApp/firstTemplate.html



http://127.0.0.1:8000/render/

Get y post

GET y **POST** son los dos métodos más utilizados del protocolo HTTP para enviar y recibir datos entre cliente y servidor. Ambos se utilizan en formularios web, [solicitudes a APIs](#) y en la comunicación entre navegadores y servidores, pero tienen diferencias claves en estructura, uso y seguridad.

- **GET:** Se usa para **obtener información** del servidor. Los datos se envían en la URL y suelen ser visibles y almacenables en caché.
- **POST:** Se usa para **enviar datos** al servidor, como en formularios de registro. Los datos se envían en el cuerpo de la solicitud y no se muestran en la URL.

Cuándo usar GET y cuándo usar POST

Usa GET cuando:

- Necesites enviar parámetros simples (como en una búsqueda).
- La acción **no modifica** el estado del servidor.
- Quieras permitir que la URL sea compatible y cacheable.

Usa POST cuando:

- Estés enviando **datos sensibles o complejos** (como contraseñas o archivos).
- Necesites **modificar recursos** en el servidor (crear, actualizar o eliminar).
- Requieren mayor privacidad en la transmisión de datos.

Ejemplos: GET y POST

Ejemplo de solicitud GET:

```
<form action="/buscar" method="GET">  
  <input type="text" name="q" placeholder="Buscar...">  
  <button type="submit">Buscar</button>  
</form>  
URL generada:  
/buscar?q=javascript
```

Ejemplo de solicitud POST:

```
<form action="/registro" method="POST">  
  <input type="text" name="nombre">  
  <input type="password" name="contrasena">  
  <button type="submit">Registrarse</button>  
</form>
```

Django

**Manejo de
Templates!**

base.html



 static

 styles
— main.css
— profile.css

 js
— script.js
— profile.js

 images