# Video file : 81

# Code file :ch71

# Django Async View Difference between Sync and Async View in Django 5

Python orm don’t completely follws async

So if we intrested so we use a third party orm

Tortoise orm

## Django 5: Sync vs Async Views

### 🔹 **Synchronous View (Sync View)**

* **Definition**: Normal Django views (default). Executed step by step, one request at a time.
* **Blocking**: If one view is busy (e.g., slow DB query, external API), the server waits.
* **Code Example**:
* •   from django.http import HttpResponse
* •
* •   def my\_view(request):
* •       # sync code
* •       return HttpResponse("This is a Sync View")

### 🔹 **Asynchronous View (Async View)**

* **Definition**: Django view defined with async def. Can handle multiple requests at the same time without waiting for blocking operations.
* **Non-blocking**: Runs concurrently, so it doesn’t stop the server while waiting (e.g., API calls, WebSockets).
* **Code Example**:
* •   from django.http import HttpResponse
* •   import asyncio
* •
* •   async def my\_async\_view(request):
* •       await asyncio.sleep(2)  # non-blocking wait
* •       return HttpResponse("This is an Async View")

## ⚡ Key Differences

| **Feature** | **Sync View (default)** | **Async View** |
| --- | --- | --- |
| **Definition** | def my\_view() | async def my\_view() |
| **Execution** | One request at a time (blocking) | Multiple requests concurrently (non-blocking) |
| **Best for** | Normal DB queries, templates, small apps | API calls, WebSockets, chat apps, real-time updates |
| **Performance** | Slower for I/O heavy tasks | Faster for I/O heavy tasks |

## ✅ Why & When to Use Async Views?

* Use **Sync Views** for:
  + Normal CRUD apps, database-heavy operations.
  + Simple websites (blogs, e-commerce, etc.).
* Use **Async Views** for:
  + **I/O bound tasks** → calling external APIs, file uploads, sending emails.
  + **Real-time features** → WebSockets (chat apps, notifications, live dashboards).
  + Handling thousands of concurrent connections.

⚡ **Summary for Notes:**

* **Sync Views**: default, blocking, simple CRUD.
* **Async Views**: non-blocking, faster for I/O tasks, best for APIs, WebSockets, real-time apps.
* Django lets you **mix sync & async views** in one project.

Self document:

Set up all the things using the uvicorm

And in app we can work with async

Views.py:

from django.shortcuts import render,HttpResponse

# Create your views here.

def home(request):

    return render(request,'core/home.html')

async def home(request):

    return HttpResponse("This is home page ")

1. now we call api and see sync and async (timming result)go to

<https://jsonplaceholder.typicode.com/> for api calling

1. install 1 more packege

pip install httpx

from django.shortcuts import render, HttpResponse

from django.http import JsonResponse

import httpx

import time

import asyncio

def home(request):

  return HttpResponse('Hello Home Page')

# async def home(request):

#   return HttpResponse('Hello Home Page')

# Synchronous View

def sync\_view(request):

  start\_time = time.time()

  responses = []

  for \_ in range(5):

    response = httpx.get("https://jsonplaceholder.typicode.com/posts")

    responses.append(response.json())

  end\_time = time.time()

  time\_taken = end\_time - start\_time

  return JsonResponse({

    'status':'success',

    'total\_request':5,

    'time\_taken': f"{time\_taken:.2f} seconds"

  })

# Asynchronous View

async def async\_view(request):

  start\_time = time.time()

  async with httpx.AsyncClient() as client:

    tasks = [client.get("https://jsonplaceholder.typicode.com/posts") for \_ in range(5)]

    responses = await asyncio.gather(\*tasks)

  end\_time = time.time()

  time\_taken = end\_time - start\_time

  return JsonResponse({

    'status':'success',

    'total\_request':5,

    'time\_taken': f"{time\_taken:.2f} seconds"

  })

hit : python manage.py runuvicorn

now, we will see the result on our web page