**Video file : 85**

**Code file : ch76**

**Async Middleware in Django 5**

We must ensure that our get response must be asyncronoous

For determine this we have 1 funtion name as : routine function : we use that to check what is routine of the get function

First make the two view function:

Views.py:

from django.shortcuts import HttpResponse

async def async\_view(request):

    print("Inside Async view")

    return HttpResponse("Async View Page")

def sync\_view(request):

    print("Inside Sync view")

    return HttpResponse("Sync View Page")

And than make midddlwares.py:

In middleware we both defines also for sync and also for async to how we handle both if get response is sync go to sync function if the function is asgi so they will go to the async

For sync do write commands :

Python manage.py runserver after doing all the things in views middlewares and other

And for async do write commands what we use if we use uvicorn so ::

uvicorn ch72.asgi:application –-reload

middlewares.py:

from asgiref.sync import iscoroutinefunction, markcoroutinefunction

from django.utils.decorators import sync\_and\_async\_middleware, sync\_only\_middleware, async\_only\_middleware

@sync\_and\_async\_middleware

# @sync\_only\_middleware

# @async\_only\_middleware

def my\_fun\_middleware(get\_response):

    print("One-time initialization")

    if iscoroutinefunction(get\_response):

        async def middleware(request):

            print(f"Before view (Async): {request.path}")

            # Call Next Middleware or Final View

            response = await get\_response(request)

            print(f"After view (Async): {request.path}")

            return response

    else:

        def middleware(request):

            print(f"Before view (Sync): {request.path}")

            # Call Next Middleware or Final View

            response = get\_response(request)

            print(f"After view (Sync): {request.path}")

            return response

    return middleware

class MyAsyncClassMiddleware:

    async\_capable = True

    sync\_capable = False

    def \_\_init\_\_(self, get\_response):

        self.get\_response = get\_response

        if iscoroutinefunction(self.get\_response):

            markcoroutinefunction(self)

        print("One-time initialization")

    async def \_\_call\_\_(self, request):

        print(f"Before view (Async): {request.path}")

        # Call Next Middleware or Final View

        response = await self.get\_response(request)

        print(f"After view (Async): {request.path}")

        return response

class MySyncClassMiddleware:

    async\_capable = False

    sync\_capable = True

    def \_\_init\_\_(self, get\_response):

        self.get\_response = get\_response

        print("One-time initialization")

    def \_\_call\_\_(self, request):

        print(f"Before view (Sync): {request.path}")

        # Call Next Middleware or Final View

        response = self.get\_response(request)

        print(f"After view (Sync): {request.path}")

        return response

now register them in settings.py:

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

    'myapp.middlewares.my\_fun\_middleware',

    # 'myapp.middlewares.MyAsyncClassMiddleware',

    # 'myapp.middlewares.MySyncClassMiddleware',

]

Im middleswares.py if we highlight only

@sync\_and\_async\_middleware

# @sync\_only\_middleware

# @async\_only\_middleware

Sync so sync view can run

If we highlight async and else comment so async can run

No matter which url we hit’s either they sync or they async

Recommended:

use sync\_only or async\_only middlware

now if we like to make the class base middleware check upper middlware.py code we must do that

in settings.py :

    # 'myapp.middlewares.MyAsyncClassMiddleware',

    # 'myapp.middlewares.MySyncClassMiddleware',

There are use for class base middlware register