**Topic: Django Signals**

**Question 1: By default are django signals executed synchronously or asynchronously? Please support your answer with a code snippet that conclusively proves your stance. The code does not need to be elegant and production ready, we just need to understand your logic.**

Django is primarily synchronous by default. This means that when a request is processed, it waits for the response before moving on to the next task, making it synchronous in nature. Django 3.1, support for asynchronous views and middleware.

Signals.py

# myapp/signals.py

import time

from django.db.models.signals import post\_save

from django.contrib.auth.models import User

from django.dispatch import receiver

@receiver(post\_save, sender=User)

def user\_saved\_handler(sender, instance, \*\*kwargs):

    print("Signal received for user save.")

    time.sleep(5)  # Simulate a delay to show synchronous behavior

    print("Signal handler executed.")

Apps.py

# myapp/apps.py

from django.apps import AppConfig

class MyappConfig(AppConfig):

    default\_auto\_field = 'django.db.models.BigAutoField'

    name = 'app1'

    def ready(self):

        import app1.signals  # Import the signals module

Views.py

from django.contrib.auth.models import User

# Create a new user to trigger the post\_save signal

user = User.objects.create(username="testuser", password="testpassword")

print("User save finished.")

**Question 2: Do django signais run in the same thread the caller? Please support your answer with a code snippet that conclusively proves your stance. The code does not need to be elegant and production ready, we just need to understand your logic**.

Yes, Django signals run in the same thread as the caller by default. This means that when a signal is triggered, the receiver functions connected to that signal are executed synchronously in the same thread where the signal is sent. This can potentially block or slow down the calling process if the receiver functions take a long time to execute.

**# myapp/signals.py**

import threading

from django.db.models.signals import post\_save

from django.contrib.auth.models import User

from django.dispatch import receiver

@receiver(post\_save, sender=User)

def user\_saved\_handler(sender, instance, \*\*kwargs):

print(f"Signal handler thread: {threading.current\_thread().name}")

**# myapp/apps.py**

from django.apps import AppConfig

class MyappConfig(AppConfig):

default\_auto\_field = 'django.db.models.BigAutoField'

name = 'myapp'

def ready(self):

import myapp.signals # Import the signals module to connect them

**# myapp/views.py**

import threading

from django.contrib.auth.models import User

from django.http import HttpResponse

def create\_user\_view(request):

print(f"Caller thread: {threading.current\_thread().name}")

user = User.objects.create\_user(username='newuser', password='password123')

return HttpResponse("User created")

**# myapp/urls.py**

from django.urls import path

from .views import create\_user\_view

urlpatterns = [

path('create-user/', create\_user\_view),

]

**Question 3. By default do django signals run in the same database transaction as the caller? Please support your answer with a code snippet that conclusively proves your stance. The code does not need to be elegant and production ready, we just need to understand your logic**

Yes, by default, Django signals run in the same database transaction as the caller. This means that if a signal is triggered during a database operation (like a save() or delete()), the signal handlers are executed within the same transaction.

**# myapp/signals.py**

from django.db.models.signals import post\_save

from django.contrib.auth.models import User

from django.dispatch import receiver

from django.db import transaction

from django.db import connection

@receiver(post\_save, sender=User)

def user\_saved\_handler(sender, instance, \*\*kwargs):

print("Signal handler triggered.")

# Check if we're in a transaction

in\_atomic\_block = connection.in\_atomic\_block

print(f"Signal handler in transaction: {in\_atomic\_block}")

# Simulate an error to test rollback behavior

raise Exception("Simulating error in signal handler.")

**# myapp/apps.py**

from django.apps import AppConfig

class MyappConfig(AppConfig):

default\_auto\_field = 'django.db.models.BigAutoField'

name = 'myapp'

def ready(self):

import myapp.signals

**# myapp/views.py**

from django.contrib.auth.models import User

from django.http import HttpResponse

def create\_user\_view(request):

try:

user = User.objects.create\_user(username='newuser', password='password123')

except Exception as e:

return HttpResponse(f"Error occurred: {str(e)}")

return HttpResponse("User created")

**Topic: Custom Classes in Python**

Description: You are tasked with creating a Rectangle class with the following requirements

1. An instance of the Rectangle class requires length:int and width: int to be

initialized.

2. We can iterate over an instance of the Rectangle class

1. When an instance of the Rectangle class is iterated over, we first get its length in the format (length: VALUE OF LENGTH) Sollowed by the width (width: VALUE OF WIDTH)

**##CODE**

class Rectangle:

def \_\_init\_\_(self, length: int, width: int):

self.length = length

self.width = width

def \_\_iter\_\_(self):

return iter([f"length: {self.length}", f"width: {self.width}"])

rect = Rectangle(10, 5)

for dim in rect:

print(dim)