**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.

Ans: By default, Django signals are executed **synchronously**. This means that the signal handlers are executed in the same thread and process as the main request, blocking the execution until the handler finishes its task.

**Code:**

# models.py

from django.db import models

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

from django.dispatch import receiver

import time

import datetime

class TestModel(models.Model):

name = models.CharField(max\_length=100)

# Signal handler

@receiver(post\_save, sender=TestModel)

def my\_signal\_handler(sender, instance, \*\*kwargs):

print("Signal handler started.")

time.sleep(5) # Simulate a long-running task (5 seconds delay)

print("Signal handler completed.")

# Simulate creating an object and timing the process

if \_\_name\_\_ == "\_\_main\_\_":

start\_time = datetime.datetime.now()

obj = TestModel(name="Test Object")

obj.save()

end\_time = datetime.datetime.now()

print(f"Time taken: {end\_time - start\_time}")

**Question 2**: Do django signals run in the same thread 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.

**Ans:** Yes, by default, Django signals run in the **same thread** as the caller. This means that the signal handler is executed in the same thread that triggered the signal.

To conclusively prove this, we can use the threading module to print the current thread information in both the code that triggers the signal and the signal handler itself.

**Code:**

# models.py

from django.db import models

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

from django.dispatch import receiver

import threading

class TestModel(models.Model):

name = models.CharField(max\_length=100)

# Signal handler

@receiver(post\_save, sender=TestModel)

def my\_signal\_handler(sender, instance, \*\*kwargs):

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

if \_\_name\_\_ == "\_\_main\_\_":

print(f"Main execution running in thread: {threading.current\_thread().name}")

obj = TestModel(name="Test Object")

obj.save()

**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.

**Ans:** By default, Django signals do not run within the same database transaction as the caller unless explicitly wrapped within a transaction . This means that signal handlers execute independently of the database transaction used by the caller.

Code:

# models.py

from django.db import models, transaction

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

from django.dispatch import receiver

from django.db import connection

class TestModel(models.Model):

name = models.CharField(max\_length=100)

# Signal handler

@receiver(post\_save, sender=TestModel)

def my\_signal\_handler(sender, instance, \*\*kwargs):

print("Signal handler started.")

# Create another object within the signal handler

TestModel.objects.create(name="Created in Signal")

print("Signal handler finished.")

if \_\_name\_\_ == "\_\_main\_\_":

try:

with transaction.atomic():

# Create a test object, triggering post\_save signal

obj = TestModel(name="Main Object")

obj.save()

print("Number of entries before rollback:", TestModel.objects.count())

raise Exception("Rolling back transaction")

except Exception as e:

print(f"Transaction rolled back: {e}"

print("Number of entries after rollback:", TestModel.objects.count())