Ὅ Document: Creating a Custom Signal in Django

✅ Introduction

In Django, signals are a powerful way to decouple components. They allow you to execute code in response to specific actions (like saving or deleting a model) without embedding that logic directly into the model.

This document explains how to define and use a custom signal in Django step by step.

ᾞ Required Components

To create a custom signal in Django, you need the following:

1. Define the signal

2. Create a receiver function

3. Connect the receiver to the signal

4. Trigger the signal where needed

5. Register the signal in the app layer (e.g., apps.py)

Ὦ️ Step-by-Step Signal Creation

1. Define the Signal

In a new file like signals.py within your app:

from django.dispatch import Signal

user\_logged\_custom = Signal(providing\_args=["user", "ip\_address"])

(Note: As of Django 4.0+, providing\_args is deprecated and no longer necessary.)

2. Create the Receiver

In the same file or another (preferably signals.py):

from django.dispatch import receiver

from .signals import user\_logged\_custom

@receiver(user\_logged\_custom)

def log\_user\_login(sender, \*\*kwargs):

user = kwargs.get('user')

ip = kwargs.get('ip\_address')

print(f"[SIGNAL] User {user.username} logged in from IP {ip}")

3. Trigger the Signal

In your login view:

from .signals import user\_logged\_custom

def custom\_login\_view(request):

user = request.user

ip = request.META.get('REMOTE\_ADDR')

user\_logged\_custom.send(sender=None, user=user, ip\_address=ip)

4. Register the Signal in Django Layer

In your app's apps.py:

from django.apps import AppConfig

class MyAppConfig(AppConfig):

name = 'myapp'

def ready(self):

import myapp.signals

In the \_\_init\_\_.py file of the app:

default\_app\_config = 'myapp.apps.MyAppConfig'

✅ Final Result

Every time a user logs in, the user\_logged\_custom signal is fired, and the receiver function processes the data accordingly.

Ἲ Use Cases

- Logging specific events

- Sending emails on registration

- Updating stats or cache

- Integrating with external systems

Ὅ Conclusion

Custom signals help keep Django applications clean, flexible, and scalable by decoupling logic from models and views.