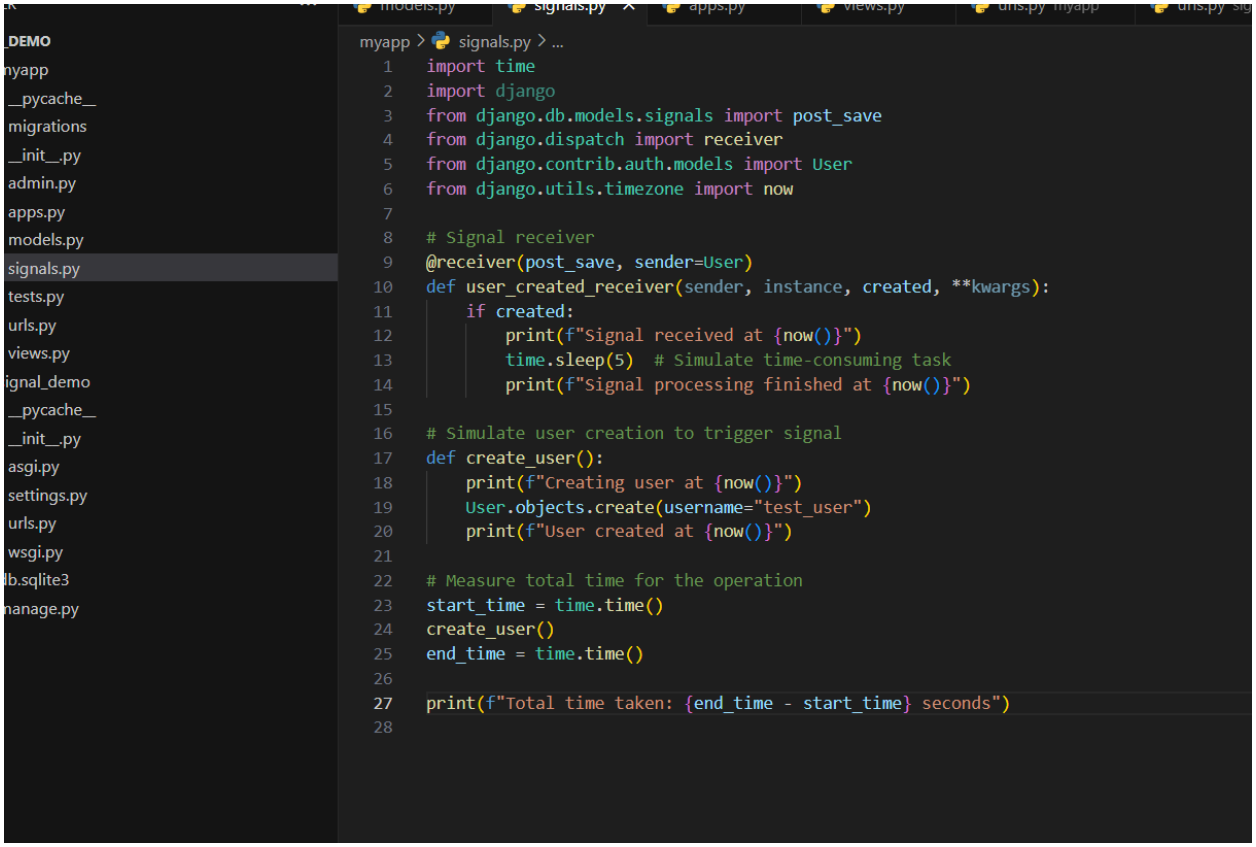
Sai NIkhil Samineni
sainikhilsamineni555@gmail.com

Question-1(Ans)

By default, Django signals are executed synchronously. This means that when a signal is sent, the receiver function is executed immediately and blocks the flow until it completes. To prove this, we can simulate a scenario where a signal receiver performs a time-consuming task. If the signal is synchronous, the delay caused by the receiver function will block further code execution until it finishes. If it were asynchronous, the main thread would continue running without waiting for the receiver.

Here is the code snippet that demonstrates this:

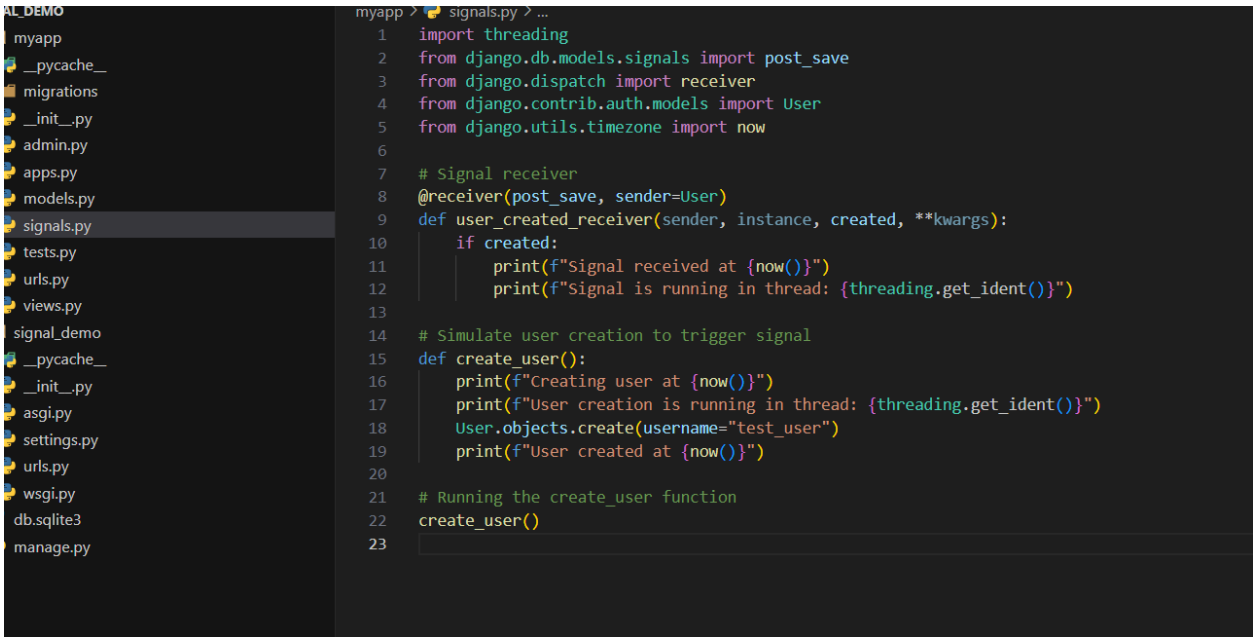
1. A signal is connected to a receiver function.
2. The receiver function introduces a delay using `time.sleep()`.
3. We measure the time taken to run the entire code block to verify if it's synchronous.



The `user_created_receiver` function listens to the `post_save` signal for the `User` model. When a new user is created, the signal is triggered. The `time.sleep(5)` introduces a 5-second delay to simulate a time-consuming task. We measure the time before and after the user creation process, including the signal execution.

Question-2(Ans)

Yes, Django signals run in the same thread as the caller by default. This means that when a signal is triggered, the receiver function runs in the same thread that executed the operation triggering the signal. To conclusively prove this, we can create a Django signal that logs the thread ID of both the caller and the signal receiver. If the thread IDs are the same, it confirms that both run in the same thread.



We use `threading.get_ident()` to get the current thread ID. This function returns a unique identifier for the current thread.

We log the thread ID in both the caller function (`create_user()`) and the signal receiver function (`user_created_receiver()`).

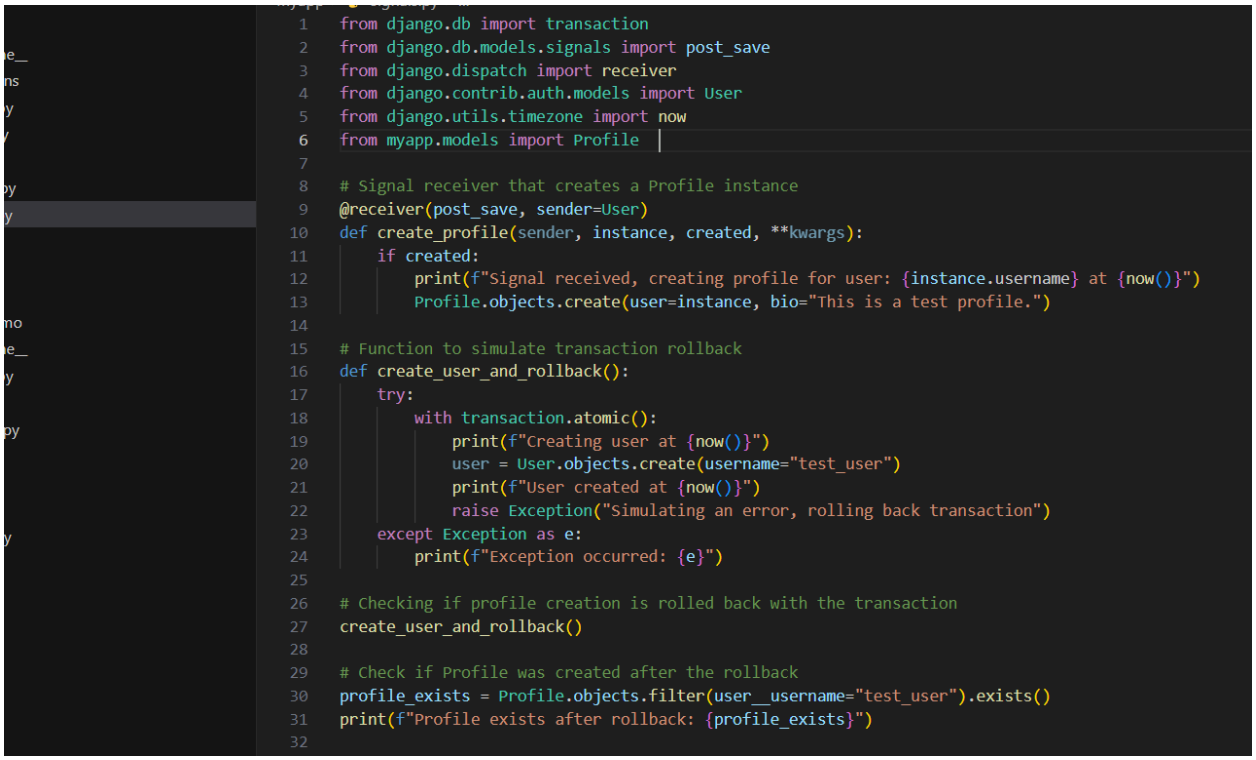
If the thread ID in the signal receiver matches the thread ID in the caller, it confirms that they are both running in the same thread.

Question-3(Ans)

Yes, by default Django signals run in the same database transaction as the caller, when the signal is triggered by a database operation. This means if the caller's transaction is not committed or is rolled back, the signal receiver's changes will also not be saved. To prove this, we can simulate a scenario where a signal modifies the database but the caller transaction is rolled back. If Django signals run in the same transaction, any changes made by the signal will also be rolled back.

Code Snippet:

1. We use Django's `post_save` signal on the `User` model.
2. The signal receiver will create another model instance.
3. In the main function, we will start a transaction, trigger the signal, and then roll back the transaction to see if the changes in the signal are also rolled back.



```python
from django.db import transaction
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User
from django.utils.timezone import now
from myapp.models import Profile

# Signal receiver that creates a Profile instance
@receiver(post_save, sender=User)
def create_profile(sender, instance, created, **kwargs):
    if created:
        print(f"Signal received, creating profile for user: {instance.username} at {now()}")
        Profile.objects.create(user=instance, bio="This is a test profile.")

# Function to simulate transaction rollback
def create_user_and_rollback():
    try:
        with transaction.atomic():
            print(f"Creating user at {now()}")
            user = User.objects.create(username="test_user")
            print(f"User created at {now()}")
            raise Exception("Simulating an error, rolling back transaction")
    except Exception as e:
        print(f"Exception occurred: {e}")

# Checking if profile creation is rolled back with the transaction
create_user_and_rollback()

# Check if Profile was created after the rollback
profile_exists = Profile.objects.filter(user__username="test_user").exists()
print(f"Profile exists after rollback: {profile_exists}")
```

The `create_profile` function listens for the `post_save` signal of the `User` model. When a new user is created, the signal is triggered, and a `Profile` instance is created for that user. The `create_user_and_rollback` function wraps the user creation in a transaction using `transaction.atomic()`. After the user is created (and the signal is triggered), an exception is raised to force the transaction to roll back.After the transaction is rolled back, we check whether the `Profile` instance created by the signal still exists.

Custom Class Ans: