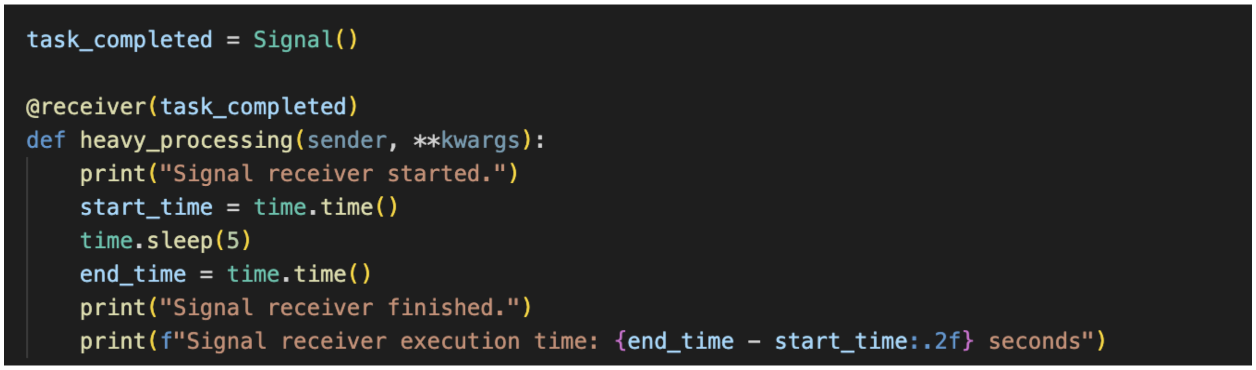
ACCUKNOX ASSIGNMENT

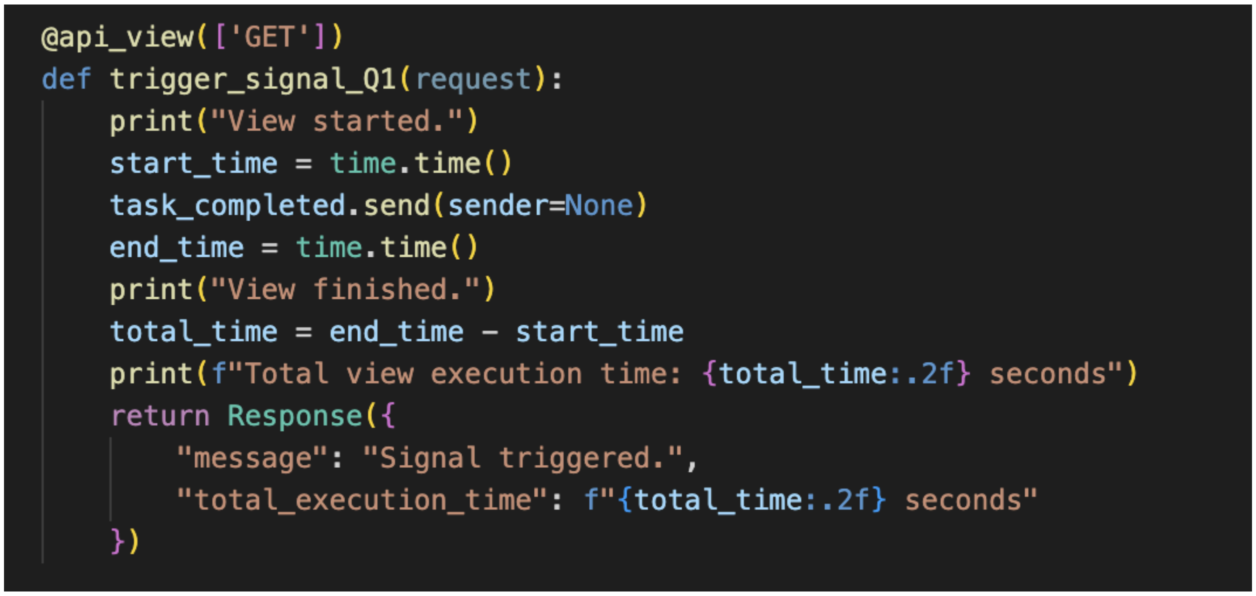
This document addresses the questions covered in Accuknox's django assignment.
Each question is answered with supporting code using Django rest framework in the form of a working apis supported by attached screenshot of request response and console outputs.

QUESTION 1: Are django signals executed synchronously or asynchronously ?
ANSWER:
By default django signals are executed synchronously, when a signal is sent all connected receiver functions are called immediately and block the sender until they complete execution.
This also means that any tasks that are too time consuming within the signal receiver, can delay the execution flow for the sender.

CODE:
signals.py and views.py

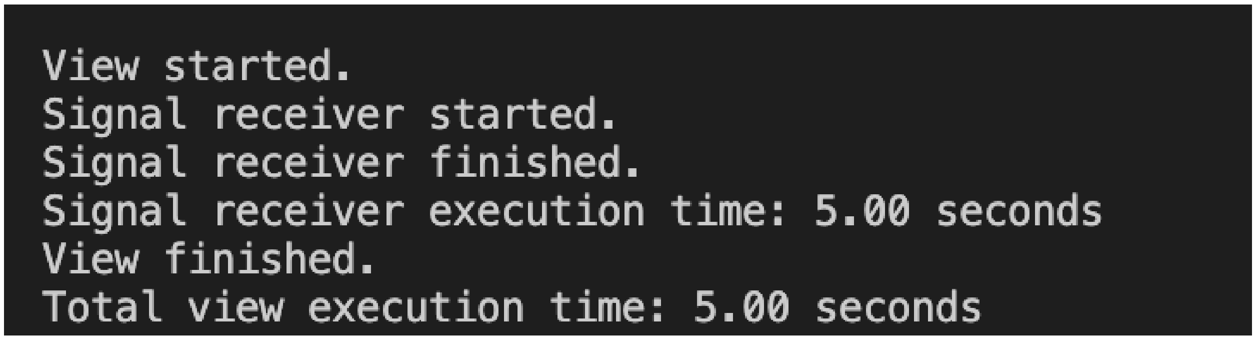


```python
task_completed = Signal()

@receiver(task_completed)
def heavy_processing(sender, **kwargs):
    print("Signal receiver started.")
    start_time = time.time()
    time.sleep(5)
    end_time = time.time()
    print("Signal receiver finished.")
    print(f"Signal receiver execution time: {end_time - start_time:.2f} seconds")
```



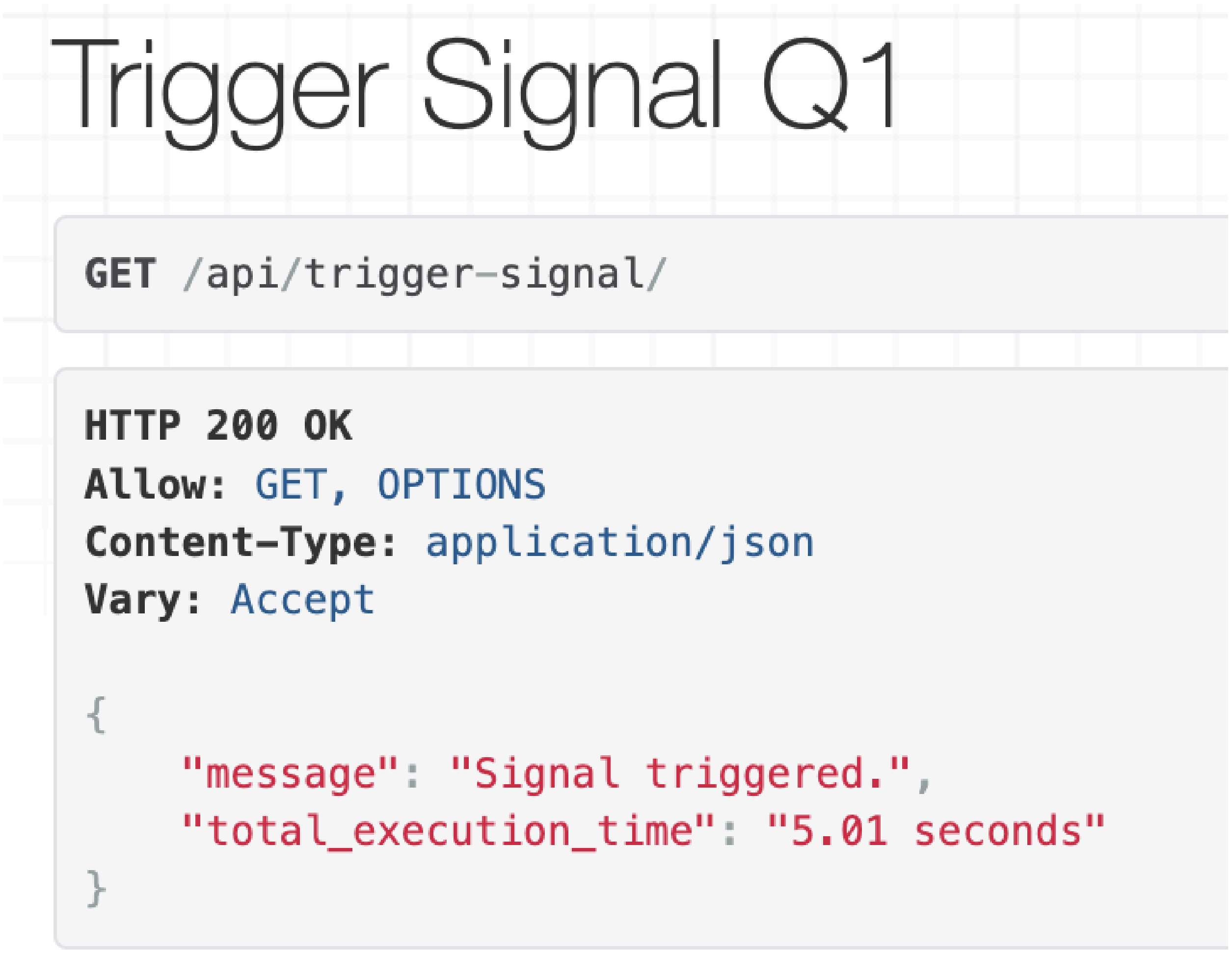
```python
@api_view(['GET'])
def trigger_signal_Q1(request):
    print("View started.")
    start_time = time.time()
    task_completed.send(sender=None)
    end_time = time.time()
    print("View finished.")
    total_time = end_time - start_time
    print(f"Total view execution time: {total_time:.2f} seconds")
    return Response({
        "message": "Signal triggered.",
        "total_execution_time": f"{total_time:.2f} seconds"
    })
```

CLI OUTPUT



```
View started.
Signal receiver started.
Signal receiver finished.
Signal receiver execution time: 5.00 seconds
View finished.
Total view execution time: 5.00 seconds
```

SERVER OUTPUT



Trigger Signal Q1

```
GET /api/trigger-signal/
```

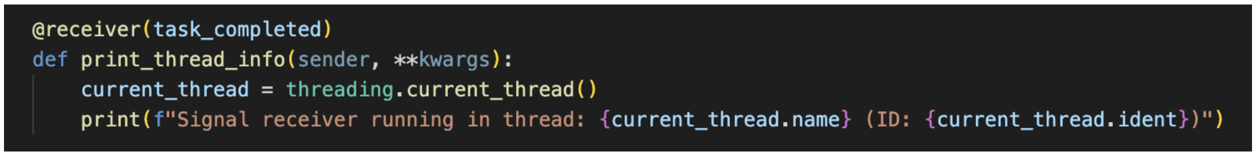
```
HTTP 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "Signal triggered.",
    "total_execution_time": "5.01 seconds"
}
```

Explanation:

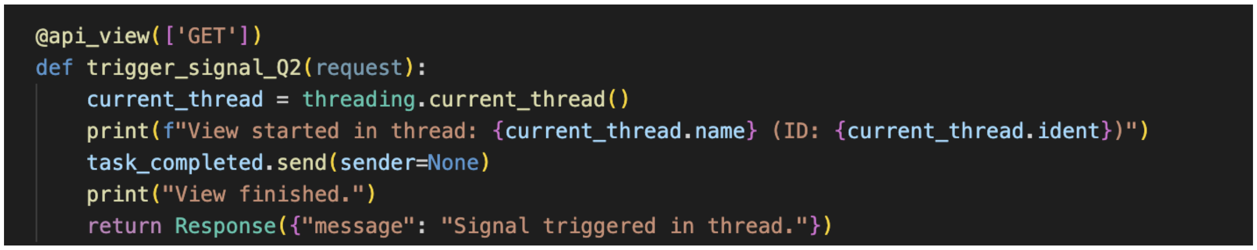
- The view starts and immediately sends the `task_completed` signal.
- The signal receiver `heavy_task_receiver` starts executing, simulates a heavy task by sleeping for 5 seconds, and then finishes.
- The view waits for the signal receiver to complete before finishing execution.
- The total execution time of the view matches the execution time of the signal receiver, demonstrating synchronous execution.

QUESTION 2: Do django signals run in the same thread as the caller?
ANSWER:
Django signals run in the same thread as the caller, both the sender and receiver execute in the same thread context, sharing data and resources within the thread.

CODE:
Signals and view files:

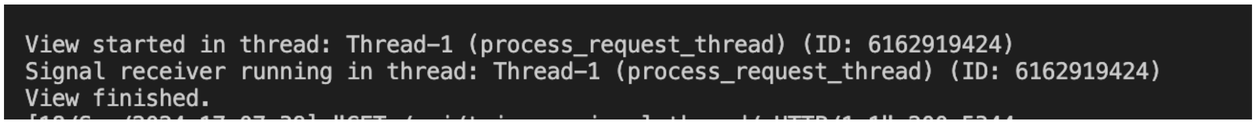


```python
@receiver(task_completed)
def print_thread_info(sender, **kwargs):
    current_thread = threading.current_thread()
    print(f"Signal receiver running in thread: {current_thread.name} (ID: {current_thread.ident})")
```



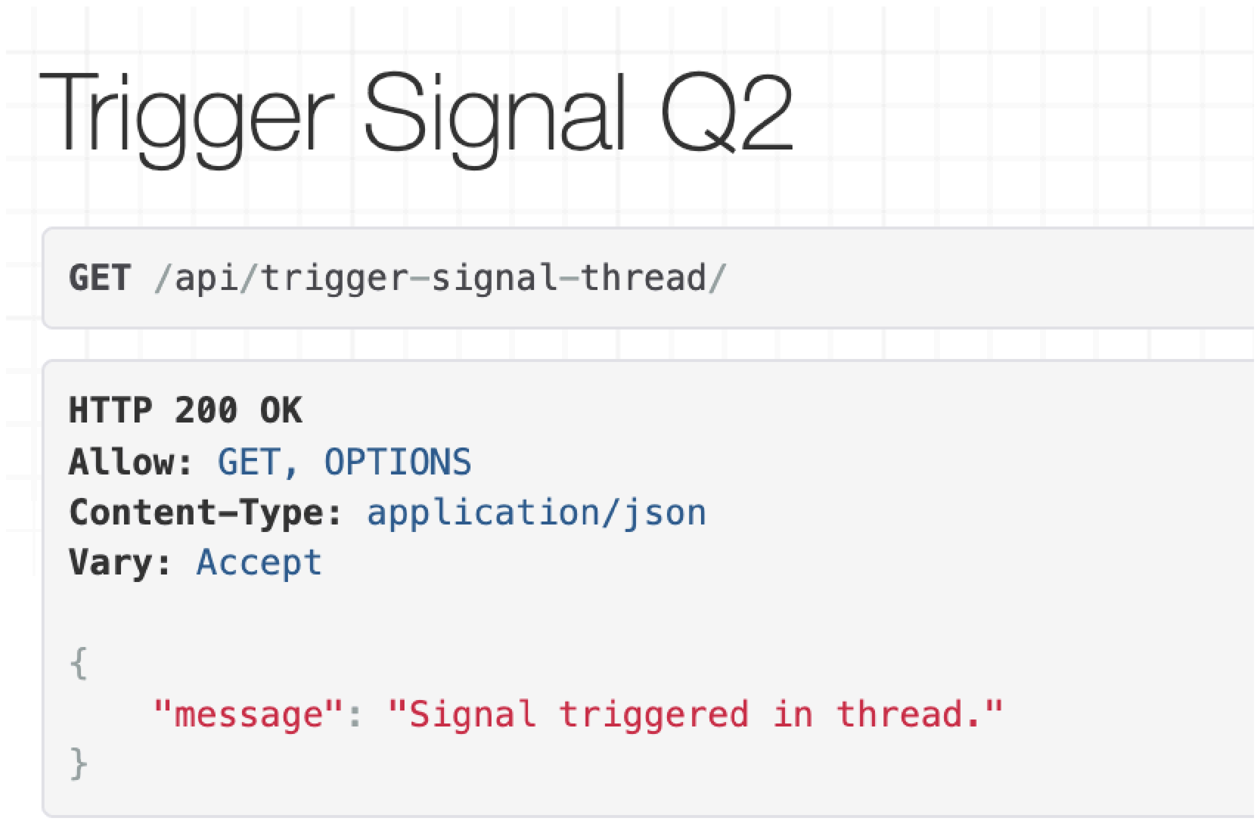
```python
@api_view(['GET'])
def trigger_signal_Q2(request):
    current_thread = threading.current_thread()
    print(f"View started in thread: {current_thread.name} (ID: {current_thread.ident})")
    task_completed.send(sender=None)
    print("View finished.")
    return Response({"message": "Signal triggered in thread."})
```

CLI OUTPUT:



```
View started in thread: Thread-1 (process_request_thread) (ID: 6162919424)
Signal receiver running in thread: Thread-1 (process_request_thread) (ID: 6162919424)
View finished.
```

SERVER OUTPUT:



Trigger Signal Q2

```
GET /api/trigger-signal-thread/
```

```
HTTP 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "Signal triggered in thread."
}
```

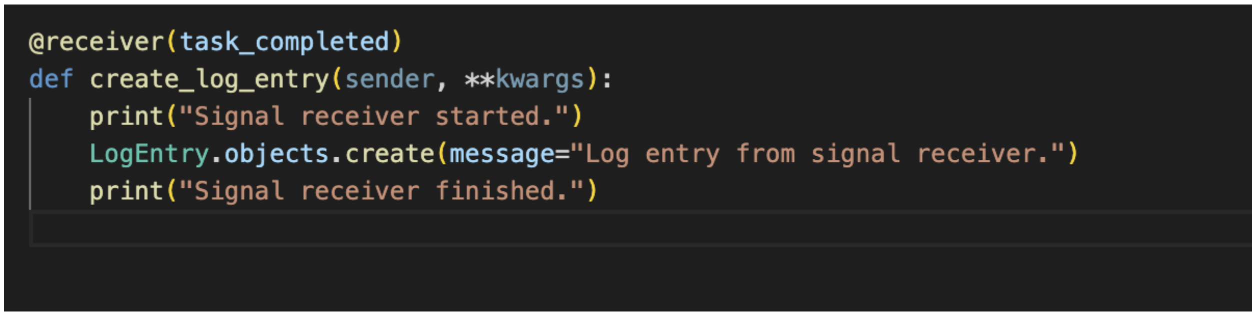
Explanation:

- Both the view and the signal receiver print out the current thread's name and identifier.
- The thread identifiers match, confirming that they are running in the same thread.

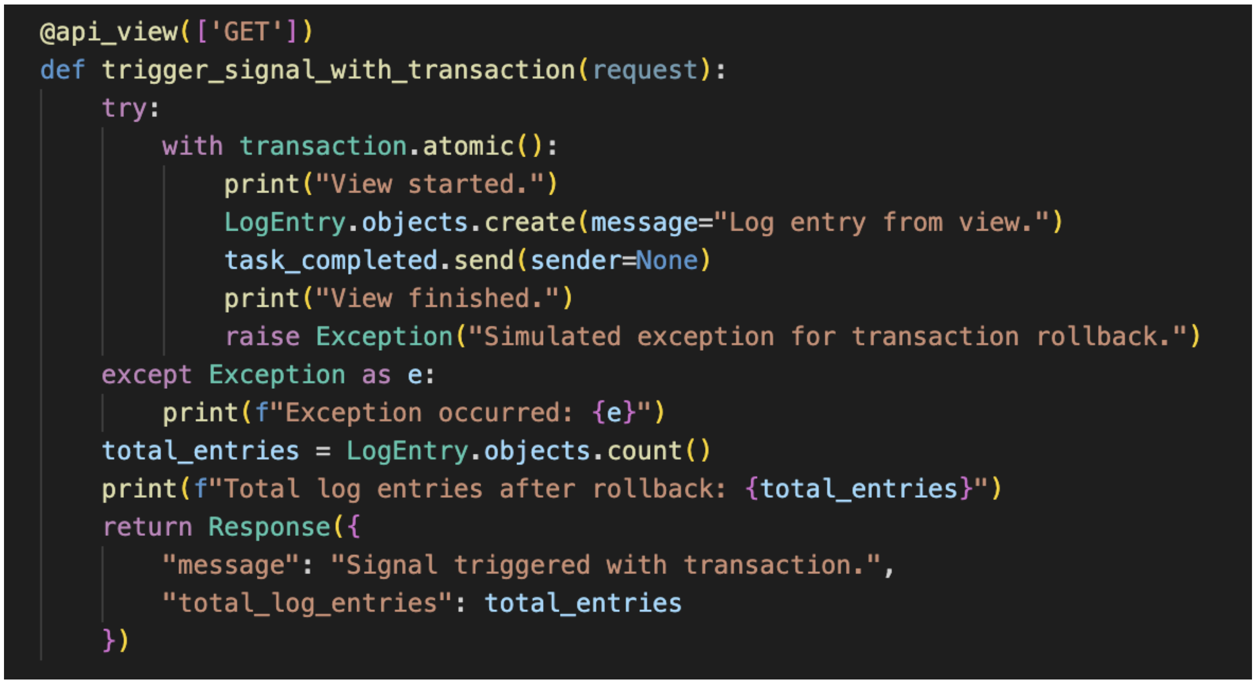
QUESTION 3: Do django signals run in the same db transaction as the caller?
ANSWER:
Yes, Django signals participate in the same db transactions as the caller by default, if a transaction in the sender is rolled back, changes made in the signal receives are also rolled back

CODE:

Signals and view files:

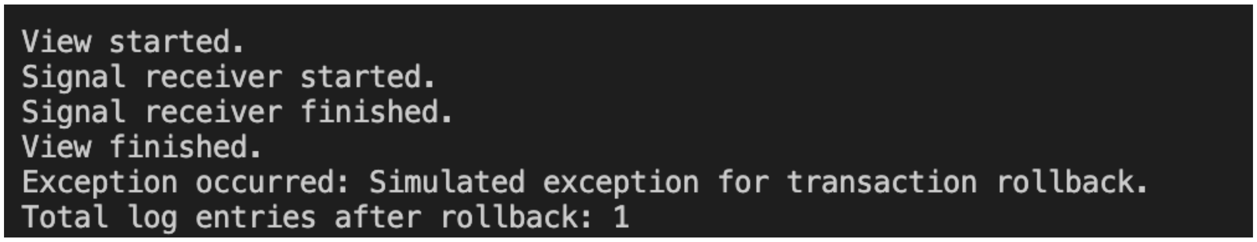


```python
@receiver(task_completed)
def create_log_entry(sender, **kwargs):
    print("Signal receiver started.")
    LogEntry.objects.create(message="Log entry from signal receiver.")
    print("Signal receiver finished.")
```



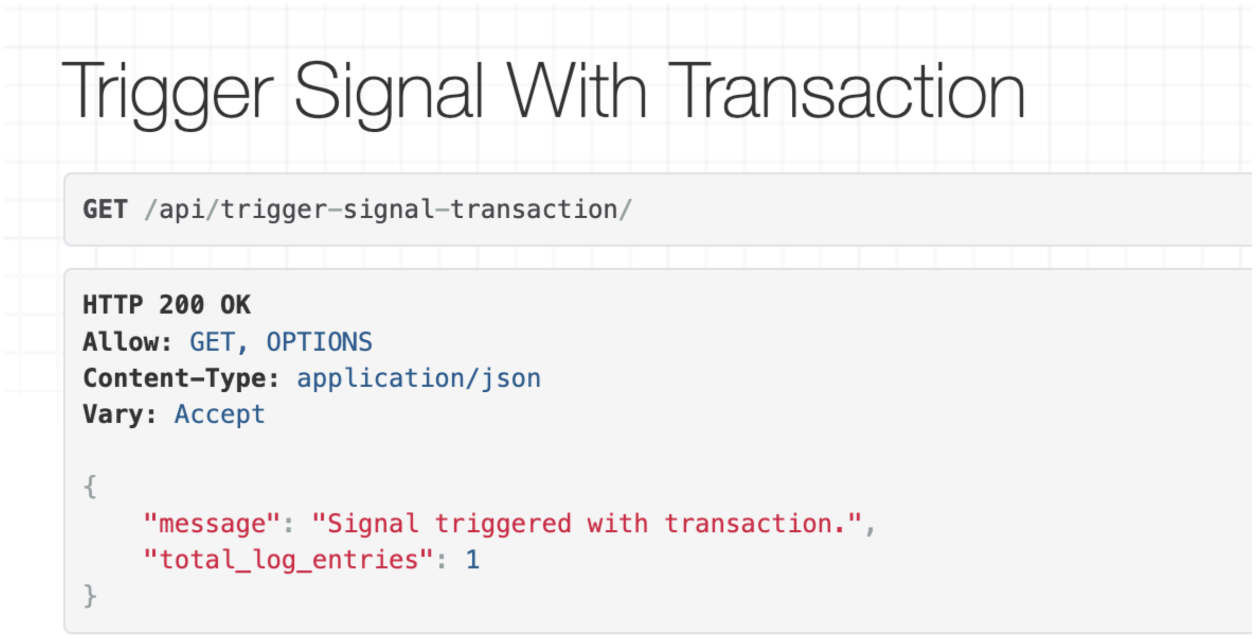
```python
@api_view(['GET'])
def trigger_signal_with_transaction(request):
    try:
        with transaction.atomic():
            print("View started.")
            LogEntry.objects.create(message="Log entry from view.")
            task_completed.send(sender=None)
            print("View finished.")
            raise Exception("Simulated exception for transaction rollback.")
    except Exception as e:
        print(f"Exception occurred: {e}")
    total_entries = LogEntry.objects.count()
    print(f"Total log entries after rollback: {total_entries}")
    return Response({
        "message": "Signal triggered with transaction.",
        "total_log_entries": total_entries
    })
```

CLI OUTPUT:



```
View started.
Signal receiver started.
Signal receiver finished.
View finished.
Exception occurred: Simulated exception for transaction rollback.
Total log entries after rollback: 1
```

SERVER OUTPUT

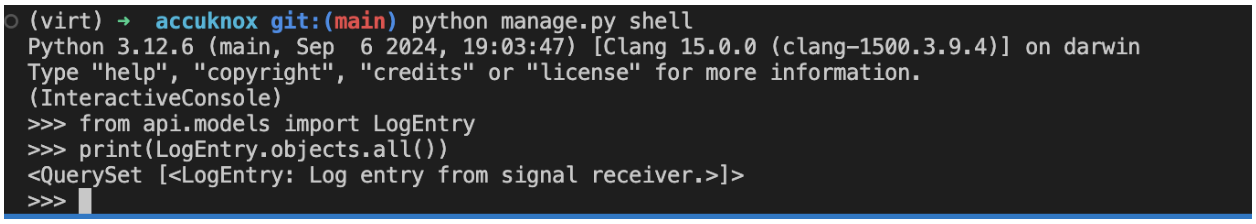


Trigger Signal With Transaction

```
GET /api/trigger-signal-transaction/
```

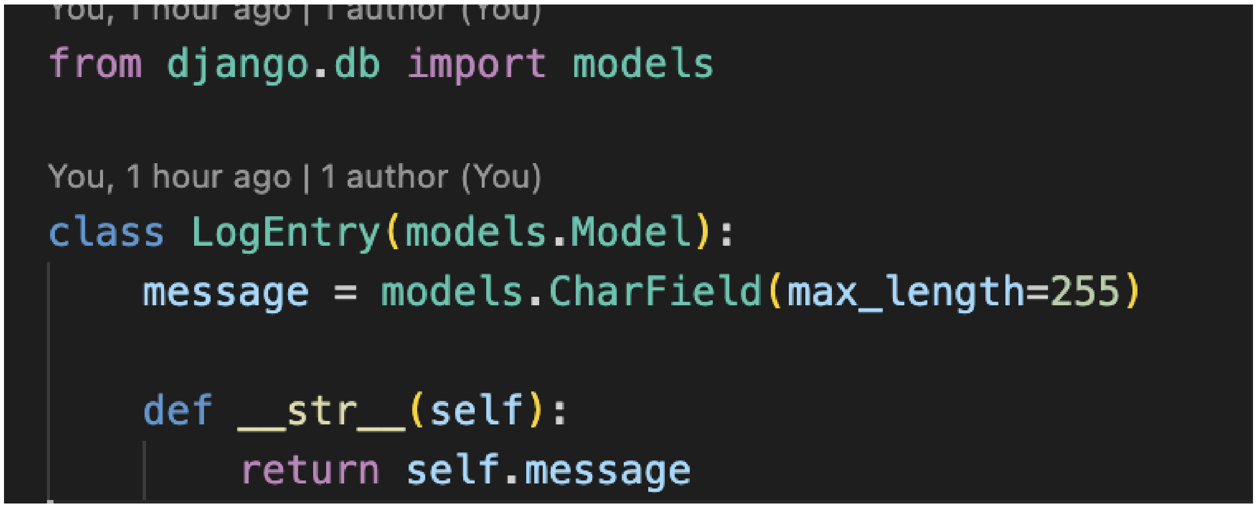
```
HTTP 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "message": "Signal triggered with transaction.",
    "total_log_entries": 1
}
```

VERIFYING DATABASE ENTRIES:



```
(virt) ➜  accuknox git:(main) python manage.py shell
Python 3.12.6 (main, Sep  6 2024, 19:03:47) [Clang 15.0.0 (clang-1500.3.9.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from api.models import LogEntry
>>> print(LogEntry.objects.all())
<QuerySet [<LogEntry: Log entry from signal receiver.>]>
>>> 
```

HELPER CODE:
models.py



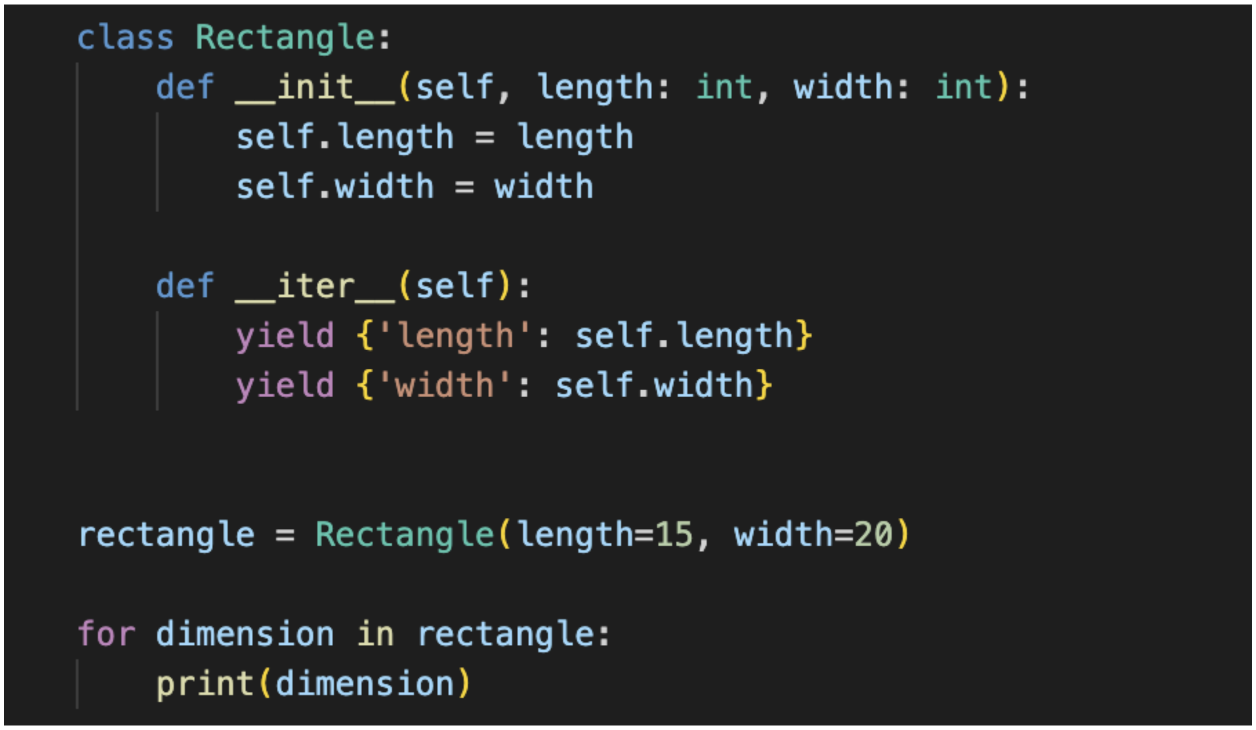
```
You, 1 hour ago | 1 author (You)
from django.db import models

You, 1 hour ago | 1 author (You)
class LogEntry(models.Model):
    message = models.CharField(max_length=255)

    def __str__(self):
        return self.message
```

Explanation:

- The view starts a transaction and creates a `LogEntry`.
- It sends the `task_completed` signal, which creates another `LogEntry`.
- An exception is raised to simulate an error, triggering a rollback of the entire transaction.
- The console output and the zero count of `LogEntry` instances confirm that both entries were rolled back.

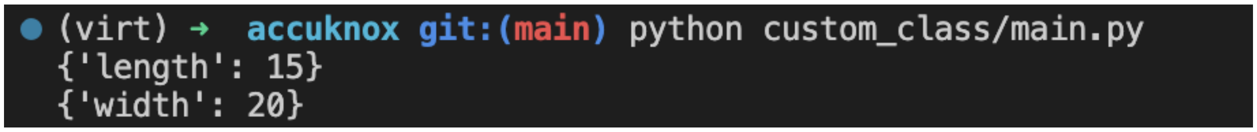
CUSTOM RECTANGLE CLASS IMPLEMENTATION

CODE:



```python
class Rectangle:
    def __init__(self, length: int, width: int):
        self.length = length
        self.width = width

    def __iter__(self):
        yield {'length': self.length}
        yield {'width': self.width}


rectangle = Rectangle(length=15, width=20)

for dimension in rectangle:
    print(dimension)
```

OUTPUT:



```
● (virt) ➜  accuknox git:(main) python custom_class/main.py
  {'length': 15}
  {'width': 20}
```