Intro to the Job Manager

For multiple reasons, we may want to execute some task after some time. That's where a job manager comes into picture. There are two components to it.

1. creating the tasks(functions) and adding them to the tasks queue
2. execute all the tasks in the queue(s)

Installation

Clone the job manager repository. Enter the root folder of the directory and create a new conda environment

```
conda create -n env_name
conda activate env_name
pip install -r requirements.txt
```

We need to update structlog, otherwise the job manager won't run properly

```
pip install -U structlog
```

Job Manager API

The job manager has two main classes, *Graph* and *Task*.

`Task`

Offers various functions related to tasks.

`Graph`

A graph stores the dependencies between the tasks. A dependency could be execute task 3 only after task 1. Multiple *Graphs* can exist at the same time. Two graphs are executed independent to each other.

`Graph.get_or_create_graph()`

Provide an index and the api either creates or returns the graph at that index.

`Task.create_task()`

This registeres the task into the graph. It takes 3 arguments.

```
task = Task.create_task(
    add_two_random_numbers,
    arguments={
        'a': random.random(),
        'b': random.random()
    },
    graph=graph
)
```

1. The function which should be run
2. The arguments of the function
3. The graph into which the function must be added

`Task.objects`

You can access the objects of the class Task with this api.

1. To get all the object: `Task.objects.all()`
2. To get the first object: `Task.objects.first()`
3. To get the last object: `Task.objects.last()`

And so on..

Usage in a Django project

Now we will see how to actually use job manager in a django project. First of all create a new Django project. Create a new app(called app_name from now). We will test Job Manager in the app.

Include the Job manager app in *INSTALLED_APPS*

```
INSTALLED_APPS = [
    ...
    'job_manager',
]
```

Add the following code to app_name/urls.py

```
from django.urls import path

from . import views

urlpatterns = [
    path("", views.add_task, name="add_task"),
    path("results", views.all_tasks_result, name="results"),
]
```

Add the code to app_name/views.py

```
from django.shortcuts import render

from django.http import HttpResponse
from job_manager.models import Task, Graph
import random


def add_two_random_numbers(a, b):
    return a + b


def add_task(request):
    graph = Graph.get_or_create_graph(id=1)[0]
    task = Task.create_task(
        add_two_random_numbers,
        arguments={"a": random.random(), "b": random.random()},
        graph=graph,
    )
    return HttpResponse(f"{task} Added")


def all_tasks_result(request):
    all_tasks = Task.objects.first()
    html = ""
    i = 0
    for task in all_tasks:
        try:
            html += f"task {i} ran with results {task.fetch_results()} <br>"
        except ValueError:
            html += f"task {i} didn't finish yet <br>"
        i += 1

    return HttpResponse(html)
```

In the *add_task* function, a task is being created. It has three arguments.

1. Function name which should be executed
2. Arguments to the function
3. The graph in which the function to be added

Now run the django server

```
python manage.py runserver
```

After adding some tasks, go to localhost:8000/app_name/results. You will observe that none of the tasks are executed.

To execute the tasks we need to run

```
python manage.py startworker
```

Now when you go to localhost:8000/app_name/results, you will see the tasks are being executed.