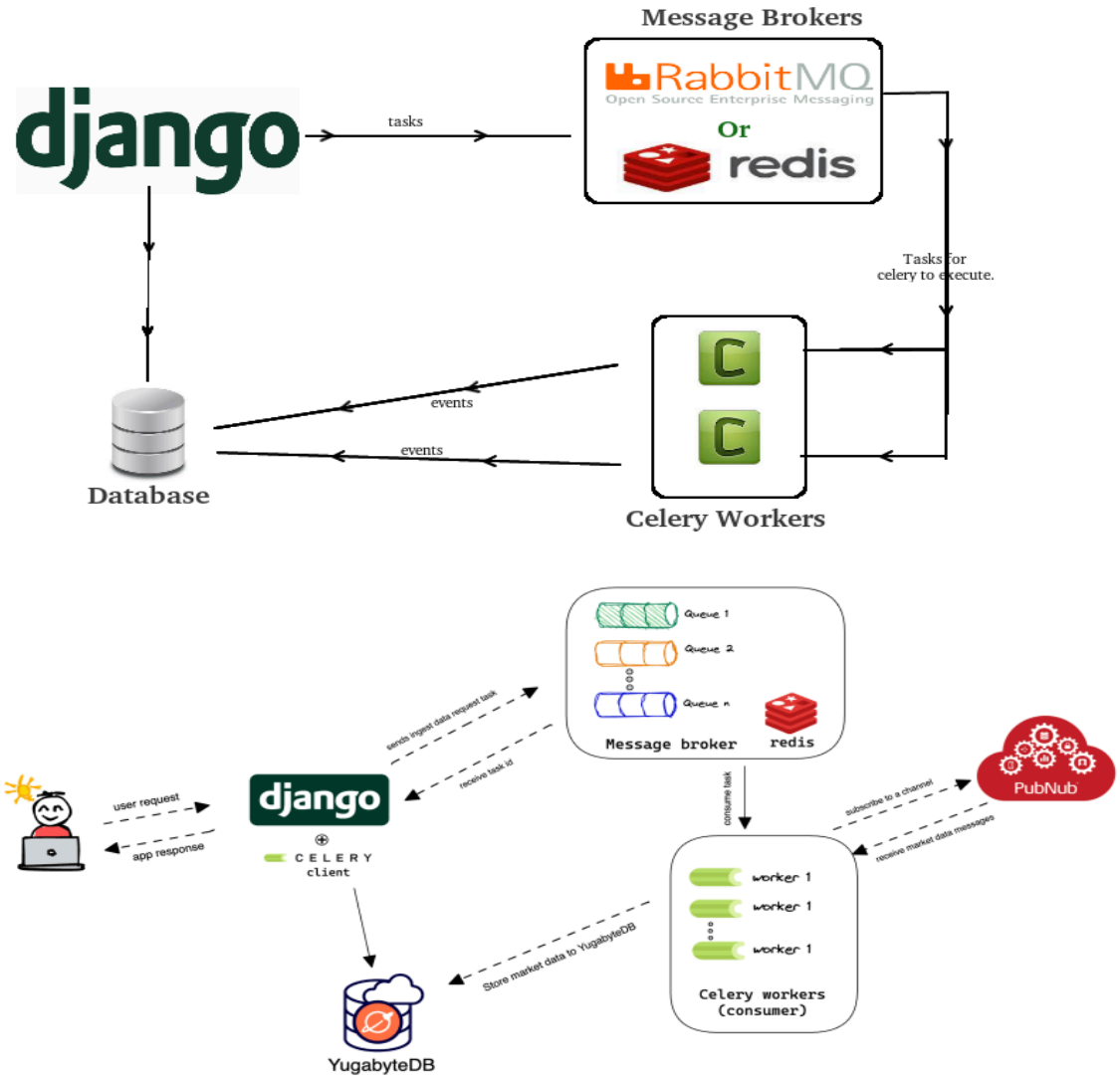
What is the use of Celery?

In Django, Celery is used to manage and execute background tasks asynchronously. This is particularly useful for:

1. Offloading Time-Consuming Tasks: Tasks like sending emails, generating reports, or processing images can be performed in the background, improving the responsiveness of your web application.
2. Scheduling Tasks: Celery allows you to schedule tasks to run at specific times or intervals, such as periodic database cleanups or regular data backups.
3. Handling Concurrent Tasks: It efficiently handles multiple tasks at once, ensuring that your web application can continue to serve requests without delay. By using Celery, you can enhance the performance, scalability, and user experience of your Django application.

Celery can be integrated with popular message brokers like *RabbitMQ*, *Redis*.



- It allows you to execute background tasks asynchronously, which is especially useful for long-running or time-consuming operations that shouldn't block the main application.

Implementation of Django Celery

- Create Virtual Environment - virtual venv
- Install Django - pip install django
- Install Celery - pip install celery
- install Redis/RabbitMQ - pip install redis
- Create Django project and apps
- create celery.py inside the project folder then write basic config code.
- configure Django celery in settings.py file
- Create tasks.py file inside apps then write task
- to trigger celery tasks, mention tasks inside views or other parts of code where it is needed.
- start a celery worker.

Django Celery Functions/Methods

- apply_async(): this fun is used to enqueue tasks for asynchronous execution. this fun gives more control over the task execution by allowing to set various options explicitly. It returns an AsyncResult object which can be used to track the status and result of the task.

Syntax:

Result = my_task.apply_async(args=[arg1, arg2], kwargs={'key':'val'}, countdown=10, expires=60)

- args: a list of positional arg to pass to the task fun
- kwarg: a dict of keyword arg to pass to the task fun
- countdown: the num of seconds to delay the task execution from the current time
- expires; the max time(in sec) until the task is considered expired and will not be executed if it hasn't started yet.

Ex: tasks.py

```
@shared_task
def add(x,y):
   sleep(10)
   return x+y
```

Ex: views.py

```
def my_view(request):
   # Enqueue the task for asynchronous execution
   result = add.apply_async(args=[10,20])
   return render(request, 'home.html', {'result':result.id})
```

- delay(): this fun is used to enqueue tasks for asynchronous execution. this fun is a shorthand for calling apply_async() with default options, making it more convenient for simple task enqueuing. It returns an AsyncResult object which can be used to track the status and result of the task.

Syntax:

Result = my_task.delay(arg1, arg2,keyword_arg='val')

Ex: tasks.py

```
@shared_task
def add(x,y):
   sleep(10)
   return x+y
```

Ex: views.py

```
def my_view(request):
   # Enqueue the task for asynchronous execution
   result = add.delay(10,20)
   return render(request, 'home.html', {'result':result.id})
```

- AsyncResult Object: It represents the result of an asynchronous task that has been enqueued for execution. When you enqueue a task using apply_async() or delay() in Celery, it returns an AsyncResult object, which allows you to monitor and manage the task's execution status and get its result once it's completed.
  - The AsyncResult object provides various methods and attributes that you can use to interact with the task.