User

I created a web app. Now I would like to add a new function to the web app. After input information (for example: machine number, channel number, test number) and clicked button "submit", it can trigger a task started to run sequential actions. Can you show me how to trigger a sequential actions through web app?

software developer

Sure, to add a new feature to your web app using a language like Python you would use a framework like Flask or Django. Here's a simple example of how you could trigger sequential actions using Flask.

1 First, you need to create a form in HTML for input:

```
<!DOCTYPE html>
<html>
<head>
    <title>Trigger Task</title>
</head>
<body>
    <form action="{{ url_for('submit_task') }}" method="post">
        Machine Number: <input type="text" name="machine_num"><br>
        Channel Number: <input type="text" name="chan_num"><br>
        Test Number: <input type="text" name="test_num"><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

1 Then, on your Flask server, you can define how to handle the post request and perform the actions using some sort of function. Here is what the Python code could look like:

```
from flask import Flask, request, render_template
app = Flask(__name__)

@app.route('/')
def form():
    return render_template('form.html')

@app.route('/submit_task', methods=['POST'])
def submit_task():
    machine_number = request.form['machine_num']
```

```
    channel_number = request.form['chan_num']
    test_number = request.form['test_num']

    # Trigger your task here with the parameters
    trigger_task(machine_number, channel_number, test_number)

    return 'Task triggered!'

def trigger_task(machine_number, channel_number, test_number):
    # Example of sequential tasks
    action1(machine_number)
    action2(channel_number)
    action3(test_number)

def action1(param):
    # Implement action1 here
    pass

def action2(param):
    # Implement action2 here
    pass

def action3(param):
    # Implement action3 here
    pass

if __name__ == '__main__':
    app.run()
```

In this code, the `trigger_task` function is called when the form is submitted. This function executes the tasks one after the other, typing each one to a different input from the form.
This is a simple example that just displays a message when the task triggering is done. In real application, you would probably want to return some sort of confirmation to the user that their task was submitted, and possibly provide updates about the progress of the task e.g. using Websockets or polling if the task takes a long time to complete.
Also, instead of executing the tasks directly in the web request, you might want to use some task queue like RQ or Celery, especially if these tasks could take time. This would return control immediately back to your web application and allow the tasks to be processed in the background at the convenience of the machine running them. In such case, machine_number, channel_number and test_number would likely be arguments for a task on your task queue.

User

This could work, but one of the sequential tasks is to check the machine's status and if the status of the machine is 'ready' then move to task 2, else check the status again after 20 mins. How to do this? And what if multiple users are using the web app, how to run this sequential tasks at same time for different users?

software developer

When you need a task to be scheduled or performed periodically, or when running background tasks that live outside the standard request-response cycle, a task queue is highly recommended. Let's use Celery, a distributed task queue, with Redis as the message broker to perform this task.

Below shows a simplified task flow:

1. Start with a `check_status` task that checks the machine status and waits for 20 minutes before re-checking if the machine is not ready.
2. After the machine is confirmed ready, the task will call the sequential actions.

```python
from celery import Celery
from celery.decorators import task
from time import sleep

app = Flask(__name__)
# configure celery client with Redis message broker
app.config['CELERY_BROKER_URL'] = 'redis://localhost:6379/0'

celery = Celery(app.name, broker=app.config['CELERY_BROKER_URL'])
celery.conf.update(app.config)

@celery.task(bind=True)
def check_status(self, machine_number, channel_number, test_number):
    if not machine_is_ready(machine_number):
        # If machine is not ready, retry this task after 20 minutes
        raise self.retry(countdown=20*60)
    # If machine is ready then run sequential tasks
    action1(machine_number)
    action2(channel_number)
    action3(test_number)

@app.route('/submit_task', methods=['POST'])
def submit_task():
    machine_number = request.form['machine_num']
    channel_number = request.form['chan_num']
    test_number = request.form['test_num']

    # Trigger the celery task
    check_status.apply_async(args=[machine_number, channel_number, te

    return 'Task triggered!'
```

Make sure to replace the `machine_is_ready` and `action1`, `action2`, `action3` function placeholders with your real code.

With the task queue, each user's tasks are processed independently, so multiple users can perform the tasks at the same time without interfering with each other.
As the Celery task is just Python code you could use a loop, recursion or similar to check the status of the machine and only continue when it's ready. Celery also supports automatic retries of tasks that can be used for this purpose.
When the task is triggered by the user a new task is put into the queue and is processed in the background, immediately returning control back to your web application. This way your webpage won't hang or timeout while waiting for the tasks to finish.
Keep in mind that you need to start a Celery worker and a Redis server for this code to work:

Start the Redis server:

```
redis-server
```

Start the Celery worker:

```
celery -A your_flask_file.celery worker
```

With this, we are able to perform tasks asynchronously and retry if the machine is not ready every 20 minutes.

User
How to start Redis server and celery in windows?

software developer
For Windows users, unfortunately, Redis doesn't officially support Windows. However, there's a ported version of Redis maintained by Microsoft you can use. Also, note that Celery does not officially support Windows. For development, you can use Windows Subsystem for Linux (WSL) or Docker.
Here's how you can start them both in Windows:
Download and run Redis server on Windows:

1. Go to the official MicrosoftArchive redis releases on GitHub (https://github.com/microsoftarchive/redis/releases)
2. Download the .msi file which says `Redis-x64-<version>.msi`
3. Run the .msi installer you just downloaded.

After the installation is completed, Redis Server will be started automatically, you can see it in the system tray. Note the port number of Redis (usually 6379), you'll be needing this in your python script where you connect to Redis.
Note: Microsoft has officially archived their Windows port of Redis and no longer maintain it.
For Celery, Windows is not supported anymore. However, you can use eventlet or gevent libraries and run celery which works on Windows too.
You need to first install eventlet or gevent, stop any running Celery worker, and then start celery worker with a command:

```
pip install eventlet
celery -A your_flask_file.celery worker -P eventlet --loglevel=info
```

or

```
pip install gevent
celery -A your_flask_file.celery worker -P gevent --loglevel=info
```

The `-P eventlet` or `-P gevent` option tells celery to use eventlet or gevent pool which is compatible with Windows.
However, for a real production environment, it's recommended to switch to Linux or Unix-based system. If you are developing on Windows, consider using WSL2, Docker, or a virtual machine running Linux for this.

User
If I am using windows do you have other tools recommend instead of using celery and redis server?

software developer
Windows presents a specific challenge for developers due to the lack of official support for tools like Celery and Redis. However, there are alternatives that you can consider.

1. **Database for Broker/Backend**: Instead of Redis, you could use databases that have full support in Windows like PostgreSQL, MySQL, or even SQLite for lighter tasks. These can act as the broker and result backend.
2. **Task Scheduling/Background Jobs**: Since Celery is not officially supported in Windows, you can use other Python libraries such as:

   `schedule` : A Python job scheduling for humans. An in-process scheduler for periodic jobs that uses the builder pattern for configuration.

   `APScheduler` : A Python library that provides job scheduling capabilities and can run jobs at given intervals or at specified times.

   `rq (Redis Queue)` : Although it uses Redis, it's easy to use and doesn't have known issues on Windows. You can use Docker to handle the Redis instance for rq.

   `Dramatiq` : It's simple and less strict about where you can define your tasks compared to Celery. Dramatiq supports RabbitMQ and Redis as the message broker.
3. **Docker**: Docker is another great option. By using Docker, development will be closer to the production environment. Docker provides images for Redis and RabbitMQ (Another powerful message broker like Redis). This way, running these services on Windows would essentially occur inside a Linux container.

Still, these are all workarounds, and you might encounter issues while running these tools on Windows for heavier and more complex tasks. When you're ready to deploy your application, it is highly recommended to use Linux or Unix-based environments for smooth operations and support.

User

Can you show me how to use Docker to run Redis and Celery for my case?

software developer

Yes, certainly. Docker can be used to handle your Redis and Celery services smoothly on any platform. Here's a basic step-by-step guide on how to set up these services using Docker. Firstly, install Docker Desktop on your machine, if you don't have it installed already.

1. Create a file called `Dockerfile` in the root of your project and paste the following:

```
FROM python:3.8-slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["gunicorn", "-b", ":8000", "app:app"]
```

This Dockerfile explains that we're using the official python 3.8 image, copying our requirements file to the app directory, installing any packages within it, copying the rest of our files, then running the application via Gunicorn.

1. Create a `docker-compose.yml` file to handle our services:

```
version: '3'

services:
  web:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - .:/app
    depends_on:
      - redis
      - celery
    command: >
      sh -c "python -m flask run --host=0.0.0.0 --port=8000"

  redis:
    image: "redis:alpine"

  celery:
    build: .
    command: celery -A your_flask_file.celery worker --loglevel=info
```

```
    volumes:
      - .:/app
    depends_on:
      - redis
```

With this docker compose file we set up 3 services:

Web: runs the main app

Redis: uses the official Redis image from DockerHub on its latest alpine version. This service handles our message broker.

Celery: runs our Celery worker

1 Run following command:

```
docker-compose up
```

Docker will pull the images from DockerHub if necessary, builds our image from Dockerfile, and start the services.
Now, make sure to adjust the `your_flask_file` with the python file of your Flask application. And replace all Redis urls in your python script with `redis://redis:6379/0` or `redis://:password@redis:6379/0` if your Redis server requires a password.
With this setup, the Redis Server and Celery Worker will be run in separate Docker containers

Enter your message...

Add and run Add

Playground messages can be viewed by anyone at your organization using the API.