Technical Assignment

This assignment is about executing delayed tasks. The challenge is designed to take 3–4 hours to complete. We don't enforce any strict time limits, so you can choose when you prefer to dedicate time for this. You may add any extra functionality you want—however, this will not affect your evaluation. Your application is solely going to be used to analyse your skills as a software developer, and will not be used for any commercial purposes.

Functional requirements:
Create a service that allows us to execute scheduled tasks.
For simplicity, let's assume that tasks can be triggered by accessing a web URL. You create a task by providing a URL and the desired time to run.
When the specified time arrives, the service will call the URL.

You are required to write a web service with 2 endpoints:

1. Implement a "set timer" endpoint: /timer
    - Receives a JSON object containing hours, minutes, seconds, and a web url.
    - Returns a JSON object with the amount of seconds left until the timer expires and an id for querying the timer in the future.
    - The endpoint should start an internal timer, which fires a webhook to the defined URL when the timer expires.
2. Implement a "get timer" endpoint: /timer/{timer_uuid}
    - Receives the timer id in the URL, as the resource uuid.
    - Returns a JSON object with the amount of seconds left until the timer expires. If the timer already expired, returns 0.

Requirements:

- The code should handle invalid inputs.
- The firing of the webhook must not be cancelled by process restarts. Any timers that expired while the application was down should be triggered once the application comes back up.
- The solution should support horizontal scalability (running on multiple servers) to handle an increasing number of timers, including their creation and webhook firing.
- Each timer must be fired only once.
- The solution must be implemented in Python. Here at Sendcloud we use Django+(Django REST Framework or Django Ninja), FastAPI and Flask.
- Wrap your application and all its dependencies in Docker container(s). Describe clearly the build steps and how to run your application in a README file. The solution should be easy to run and build for our reviewers
- Tests are mandatory. We believe in sensible testing rather than achieving 100% coverage, so make sure the important parts of your solution are tested.
- Your code should be reasonably documented, save for the blocks that are self-explanatory. Usage of docstrings is highly encouraged.
- Submit your code through the link of the e-mail that we've sent (with this doc).
- Any assumptions made. For example, "timers are never scheduled later than X days into the future".
- Any changes you would make (if any) in order to support a high-traffic (e.g. 100 timer creation requests per second) production environment.
- Please write clean & tidy code. Within our backend team, we enforce strict PEP8 compliance.

The solution will be evaluated based on correctness, craftsmanship, and horizontal scalability.

Examples

1. Set Timer example

POST - /timer
expected request:

```
Unset
{
        "hours": 4,
        "minutes": 0,
        "seconds": 1,
        "url": "https://someserver.com"
}
```

expected response:

```
Unset
{
        "id": "be18064e-d265-4cb5-9716-34f249029a79",
        "time_left": 14401
}
```

After the time has expired (4 hours and 1 second) the server should make call to the url specified in the request:

POST - https://someserver.com

```
Unset
{
        "id" : "be18064e-d265-4cb5-9716-34f249029a79"
}
```

2. Get Timer example

GET - /timer/be18064e-d265-4cb5-9716-34f249029a79
Expected response:

```
Unset
{
        "id" : "be18064e-d265-4cb5-9716-34f249029a79",
        "time_left": 645
}
```