Temperature Feed Challenge

Your mission, should you choose to accept it, is to build a backend service that consumes a temperature feed and implements a GraphQL API to serve the processed temperature data.

Table of Contents

Requirements

Special attention will be given to:

- Requirement compliance.
- Overall code formatting, readability and elegance.
- The quality of the documentation.

You can find in the following sections the list of must haves and optional tasks. The optional tasks are really optional and are here for you to shine on points that you are the most comfortable. We don't expect you to do all of them.

While we expect this to take 4-6 hours, you are welcome to spend as much time as you'd like to showcase your skills.

The solution should be hosted on a private GitHub or GitLab repository and shared with `hiring@loftorbital.com`.

Must Haves

- Let's get fancy, and use modern **Python** (>= 3.7).
- A **Dockerfile** and/or **docker-compose.yml** file should be provided, to make the whole setup portable and easy-to-use.
- The service should be built using **Django**. Additional libraries and database solution can be selected at will.
- A **README.md** file is expected, to detail the chosen solution, and how to run it.
- Relevant **unit tests** should be provided (using **pytest**).
- Use Python type annotations.

Tasks:

- Build a Django app which can store temperature readings (a timestamp and a value) in the database.
- Subscribe to the temperature feed (see how to set up the temperature feed) to continuously populate the database.
- Build a GraphQL API exposing the following operations (see `graphene` and `graphene-django` librairies):
  - a query that returns the current temperature (last emitted temperature)
  - a query that returns the minimum and maximum temperature within a time window provided via a before and/or after timestamp as argument

```
query {
    currentTemperature {
        timestamp
        value
    }
}
```

and

```
query {
    temperatureStatistics(after: "2020-12-06T12:00:00+00:00", before: "2020-12-07T12:00:00+00:00") {
        min
        max
    }
}
```

Optional Tasks

- Provide a script or CI/CD pipeline declaration of your choice (GitLab CI, Travis CI, Github actions, Jenkins, ...) that automate linting, type checking (with mypy) and run tests.
- Add a Graphql mutation to toggle on/off the temperature feed reading:

  ```
  mutation {
    toggleFeed(input: {status: "on"}) {
        status
    }
  }
  ```

- Discuss what you would change to your service or what feature you would add if it became a long term project and how you would plan the future implementation/maintenance.
- Add a GraphQL subscription using `django-channels-graphql-ws` that takes a window size (number of sample to simplify) as input and returns a rolling average of temperature over the window

  ```
  subscription {
    averageTemperature(windowSize: 20) {
        value
    }
  }
  ```

- Create a small JavaScript or TypeScript frontend app to display the current temperature and the rolling average temperature feed (if you have done the previous point) using Apollo client.
- Discuss how you would do the deployment of your app for production (feel free to provide code snippets to support your discussion).

Setting up the temperature feed

The temperature feed that your application will consume is available as a Docker image.

To run it you simply need to run:

```
docker run -p 1000:4000 --name satellite-temperature -d \
    us.gcr.io/loft-orbital-public/hiring/challenges/ground-software/back-end/satellite-temperature
```

Note that this will make it run on port 1000 on your machine, you can change the port if it is already being used.

It comes with a GraphQL Playground that enable you to test it. All you need to do is go to http://localhost:1000/graphql in your internet browser enter the following query and press play:

```
subscription {
    temperature
}
```

You can also access it in python using the `websockets` package:

```
import websockets
import asyncio
import json

def process_msg(data):
    print(data)

async def capture_data():
    uri = "ws://localhost:1000/graphql"
    start = {
        "type": "start",
        "payload": {"query": "subscription { temperature }"}
    }
    async with websockets.connect(uri, subprotocols=["graphql-ws"]) as websocket:
        await websocket.send(json.dumps(start))
        while True:
            data = await websocket.recv()
            process_msg(json.loads(data))


asyncio.run(capture_data())
```

By default it emits a value every 0.5 second. You can modify that for debugging by providing the `INTERVAL` environment variable:

```
docker run -p 1000:4000 -e INTERVAL=5 --name satellite-temperature -d \
    us.gcr.io/loft-orbital-public/hiring/challenges/ground-software/back-end/satellite-temperature
```

Useful Resources

- GraphQL
- Graphene
- Graphene-Django

Questions?

Just reach out!

This README file will self-destruct in five seconds. Good luck!