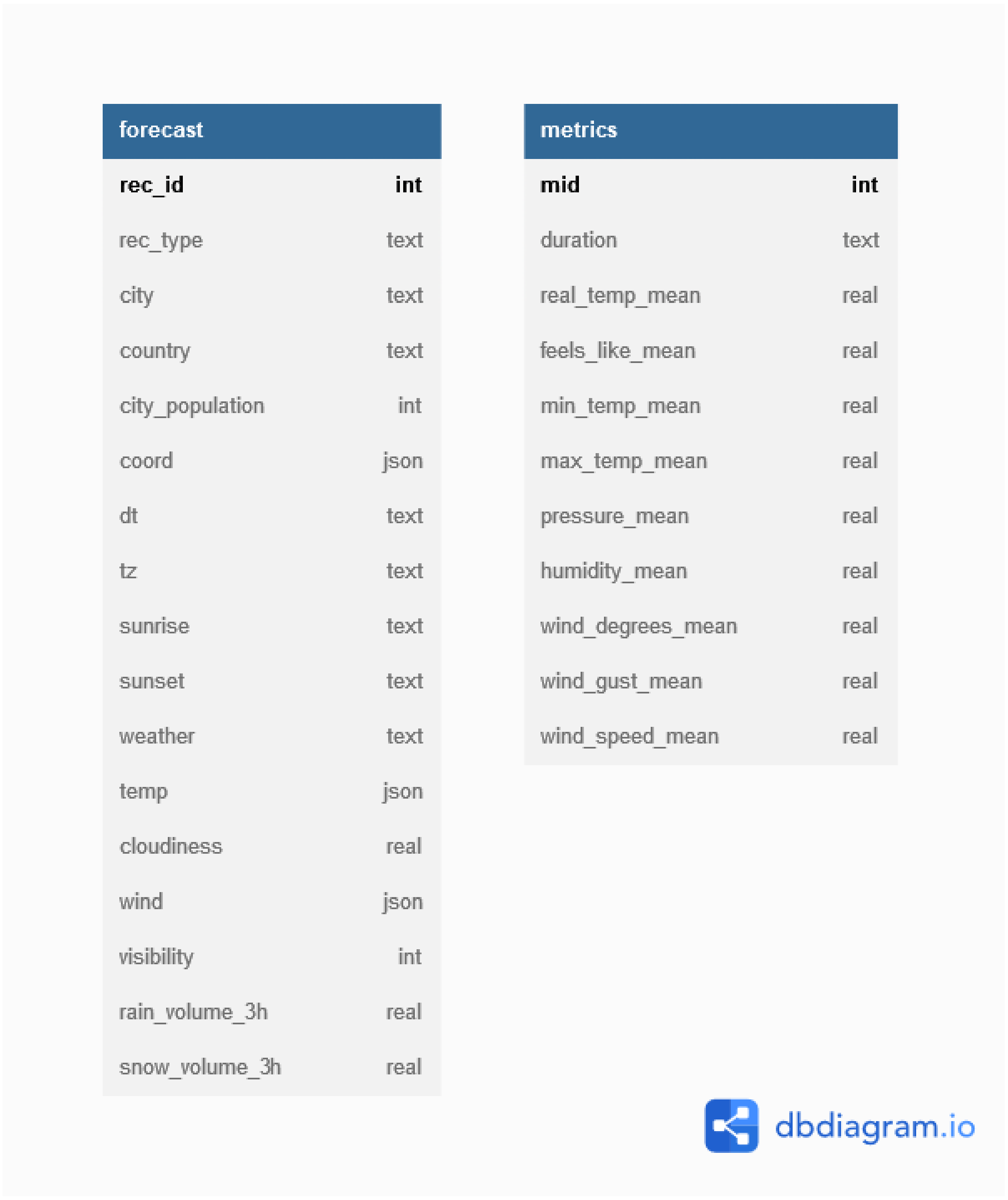
Weather API

Github Repo url: https://github.com/radistoubalidis/weatherApp

Database Diagram



forecast	
rec_id	**int**
rec_type	text
city	text
country	text
city_population	int
coord	json
dt	text
tz	text
sunrise	text
sunset	text
weather	text
temp	json
cloudiness	real
wind	json
visibility	int
rain_volume_3h	real
snow_volume_3h	real

metrics	
mid	**int**
duration	text
real_temp_mean	real
feels_like_mean	real
min_temp_mean	real
max_temp_mean	real
pressure_mean	real
humidity_mean	real
wind_degrees_mean	real
wind_gust_mean	real
wind_speed_mean	real

SQL dump can be found on `modules/schema.sql`.

Data Manipulation
While interacting with the openWeather API I realized that the output for the current weather forecast and the outputs of the 5-day predeiction weather forecast has a lot of similarities in their attributes. That's why I decided to implement them as one entity on my database. Particularly, I created the class WeatherForecast that corresponds to the database entity and has these attributes (that are common in both the current and prediciton endpoints):

- City, Country, population, coordinates, date, timezone, sunrise and sunset time, temperature, wind, cloudiness, weather description, rain_volume, snow_volume (from the last 3 hours)

To differentiate current weather records from prediction weather records I added one more attribute to my database schema called 'rec_type' which takes values 'current' or 'prediction' respectfully.

The attributes: coordinates, temperature, and wind are nested which means they also have their properties.

- Coordinates property has:
    - latitude and longtitude
- Temperature has:
    - real_temperature
    - feels_like → accounts for the human perception of weathe
    - atmospheric_pressure on sea level
    - min and max temperatures
    - humidity (%)
- Wind has:
    - speed (m/s)
    - degrees → wind direction
    - gust → wind gust (probability [0,1]

Some attributes such as city_population or rain/snow volumes can be found only on the output of the prediction forecast from openWeather. For this reason, null values on these attributes are accepted in the database.

Furthermore, all the mathematical values that are stored concern an area with center the coordinates and a radius of 10000 meters.For example if the real_temperature in coordinates{lat:40.63,lon:22.94} is 293.16 , the temperature outside the 10000m radius could be different.

Code Structure
The modules directory contains all the required code to fetch data from openWeather and store them in PostgreSQL.Particularly:

- The **ExternalData** class is an interface for fetching and handling the data from openWeather. Function getForecast retrieves today's forecast and getPrediction retrieves the 5-day prediction forecast. Along with the 'get_' methods we have the prepare_ methods which initialize WeatherForecast objects from the retrieved data

- The **WeatherForecast** class corresponds to the forecast entity in the database and has the same attributes described above. The add_ methods respectively insert the data retrieved from openWeather into PostgreSQL. There is also a corresponding method (fetchMetrics) for fetching specific columns ordered by date from the 'forecast' table to calculate the mean values and initialize the 'metrics' table.

- The **Metrics** class corresponds to the metrics database entity and has these attributes:
    - duration → the time between the first and last prediction record with an interval of 3 hours
    - real_temperature, feels_like , min and max, atmospheric pressure, humidity, wind degrees, wind speed, and wind gust mean values for all the 5-day forecast records.

- The **Database** class serves as a simple interface to connect to the PostgreSQL container.
- The **DataOperations** class serves as an interface to apply mathematical or other modifications to WeatherForecast and Metrics instances or attributes of WeatherForecast instances.

  Mathematical functions
    - Convert a timestamp date into 'YYYY-mm-dd hh:mm:ss' format
    - Add Celcius and Fahrenheit values to WeatherForecast's temperature attribute
    - prepareMetrics → calculates the metrics for the selected attributs of 5-day forecast records

  Data modifications:
    - prepare_output_by_date → prepares the output for the date endpoint
    - prepareMean → wraps the result of fetchMetrics to prepare insertion to the metrics table

Inside my app file (*server.py*) there is an interface called **prepareRequests.** This interface calls the methods of **ExternalData** that retrieve the required data from openWeather.

Cron Script
I created a shell script (`run_server.sh`) that initializes the API (Flask app and dockerized database) and runs another python script (cron.py) in which requests to the API are implemented to create and fetch Metrics data. The idea is to execute this script based on a **cron command** (the shell script will run once a day at 12:00). The workflow file can be found in `./github/workflows/cron.yml` on the `cron_script` branch.