**Create a python/flask REST web service providing an API to query the data.**

*The attached London weather forecast.json comes from www.openweathermap.org. The data format reference ishttp://www.openweathermap.org/forecast5 and can be recovered via API call http://api.openweathermap.org/data/2.5/forecast?q=London,uk&APPID={appid} (free registration required).*

*I would like to make the following calls against this web service using "curl":*

*# A general summary of the weather:#curl http://<host:ip>/weather/london/<date>/<hour minute>/# Note, temperature converted from Kelvin to C and rounded up.e.g. curl http://<host:ip>/weather/london/20160706/0900/{ "description": "few clouds", "temperature": "15C", "pressure": "1028.12", "humidity": "88%"}*

*# I would like to be able to ask for individual pieces of information:#curl http://<host:ip>/weather/london/<date>/<hour minute>/temperaturee.g. curl http://<host:ip>/weather/london/20160706/0900/temperature/{ "temperature": "15C"}*

*curl http://<host:ip>/weather/london/<date>/<hour minute>/pressuree.g. curl http://<host:ip>/weather/london/20160706/0900/pressure/{ "pressure": "1028.12"}*

*curl http://<host:ip>/weather/london/<date>/<hour minute>/humiditye.g. curl http://<host:ip>/weather/london/20160706/0900/humidity/{ "humidity": "88%"}*

*# When no data is found I would like see the response:#*

*curl http://<host:ip>/weather/london/17670812/0900/temperature{ "status": "error", "message": "No data for 1767-08-12 09:00"}*

Comments:

I have implemented the API as requested, but using the more general form:

http://<host:ip>/weather/<location>/<date>/<hour minute>

This can be used to query weather forecasts for any place in the UK, not just London. I restricted the interface to support UK places only, to reduce the possibility of duplicate names, but this is still problematic. The quality of the location data in the city.list.json file provided by OpenWeatherMap is exceptionally poor. For the UK, approximately 10% of all the entries have duplicate names, and inspection shows that most of these are duplicate encodings of the same place, for example:

{'id': 7292202, 'name': 'Ystradgynlais', 'country': 'GB', 'coord': {'lon': -3.75552, 'lat': 51.804901}}  
{'id': 2633334, 'name': 'Ystradgynlais', 'country': 'GB', 'coord': {'lon': -3.76667, 'lat': 51.76667}}

Mixed in with this are place names that do occur more than once within the UK, for example:

{'id': 2633982, 'name': 'Wick', 'country': 'GB', 'coord': {'lon': -3.08333, 'lat': 58.433331}}  
{'id': 2633985, 'name': 'Wick', 'country': 'GB', 'coord': {'lon': -3.54944, 'lat': 51.439442}}

In the current implementation, I have not distinguished between these cases so that a query on Wick would return the weather forecast for the village near Bristol rather than for the town in Caithness.

London is unambiguous in this dataset so the test queries work as expected.

The best approach to take here would depend upon the target audience. For example, if the main use case were to provide local weather, then I would suggest asking the user for a postcode, then converting this to a coordinate representation which the OpenWeatherMap API will support. To do this I would use the Ordnance Survey Code-Point Open product – this is available via the Open Government Licence. Using this we can translate an input postcode to a British National Grid coordinate reference. Then I would use the pyproj library to convert this reference to lat/long (WGS-84) coordinates. Clearly this would also work only for the UK (Code-Point Open does also cover Northern Ireland so the whole of the UK is available).

For a site that covered the whole world, a different approach would be needed, bearing in mind that the user may enter a place name/country combination that does not directly match any entry in the city list. Other world-wide gazetteers do exist and could be used to help out.

Finally, I should mention that in this implementation I am simply reading the location data from the supplied JSON file at startup. For a realistic implementation I would load the cleaned data (removing real duplicates) into a database, PostgreSQL for a scalable application, SQLite for a small scale/test application.

**Questions:**

*- If I wanted the temperature in Kelvin rather than Celsius, how could I specify this in API calls?*

I have implemented this using the optional query parameter temp. For example, the query:

<http://localhost:5000/weather/london/20180530/1726?temp=K>

gives this response:

{"description": "overcast clouds", "temperature": "294K", "pressure": 1017.77, "humidity": "79%"}

*- How would you test this REST service?*

I would first write a functional specification for the API, paying particular attention to the data issues noted above, and deriving test cases that include unique, duplicate and non-existent place names as well as valid/invalid and in-range/out-of-range dates. I would include well-formed and malformed query strings as well as variants such as trailing slashes.

For queries that would result in a query being passed to the OpenWeatherMap API, I would pre-query this API and implement a test harness, with queries being routed to this test harness rather than to the real API. This would overcome the time-based nature of the API, allowing a static set of test cases to be used.

Assuming all these unit test cases were successful, I would then run the valid queries against the real API – bearing in mind that the free licence restricts queries to one per ten minutes.

The current implementation is a fairly minimal Flask application. For a realistic implementation I would structure it around blueprints, and I would make use of libraries such as pytest.

*- How would you check the code coverage of your tests?*Using coverage.py, which can be integrated into Flask applications.

*- How could the API be documented for third-parties to use?*An API reference section, drawn from the docstrings in the code and then amplified, plus an overview/tutorial section. This would also need to cover usage limitations, see below.

*- How would you restrict access to this API?*The APPID cannot be directly used for this, as it must be kept private to the application. I would need to implement a token or password-based approach, most likely with full user management so that we could control the usage of the API by a given user, respecting whatever overall limits would be placed on the use of the OpenWeatherMap API across all users. This would be a fairly major development in its own right and would need its own dedicated set of functional requirements.

*- What would you suggest is needed to do daily forecast recoveries from openweather.org, keeping the web service up to date?*There are two aspects to this, commercial and technical. I would need to research the commercial aspect first, to determine whether the caching involved is allowed for in the terms of usage, and what query volumes could be supported.

Querying and caching 3-hour forecasts for every location in the world is highly likely not be to feasible commercially, if not technically. Restricting the scope to the UK, we would need to be able to make upwards of 4,000 queries per day if we based the queries on all places in the city list. This might be supportable commercially and is clearly feasible technically. Results could be written to a PostgreSQL database and made available for historical forecast queries.

However, the granularity of the forecasts themselves is highly likely to be at a much coarser level than would be implied by 4,000+ separate query locations – nearby locations will have strongly correlated if not identical forecasts. It is possible that access to the underlying forecast model could be gained (via negotiations if it is not currently published). Failing this, I would propose using K-means clustering to derive a smaller number of representative locations that could serve as a good approximation in terms of weather forecasting for the complete set.

In K-means clustering, the inputs are the data to be clustered and the number of clusters to be created. Suppose we decide that 200 clusters would give a good approximation to the weather anywhere in the UK, we can then use K-means clustering to calculate 200 separate clusters of places with the clustering being near-optimal in the sense that each cluster is spatially coherent (all locations in the cluster relatively near to each other and relatively far from any other cluster).

Optionally, if sufficient queries could be supported, we could test the coherence of the results by calculating the standard deviations of the weather forecast elements for the locations within each cluster. We would hope for low variances here.

Assuming the clusters have been agreed, we would then each day query a representative location for each cluster, and we would serve up results or all locations based upon the cluster to which each location belongs.

**Deliverables:**

*\* A github/bitbucket/Other git repository that can be cloned.*https://github.com/pgstarkey/weatherdemo

*\* Instructions for how to set up and get the service running.* https://github.com/pgstarkey/weatherdemo/readme.md

**Bonus Points (Optional):**

*\* A service running somewhere public I can curl. \* A vagrant environment / docker container(s) to run the service and all its tests.*

Sorry, no time for these. I’d be happy to set up a public facing service on AWS if this will make a difference.