Activity: Unit Tests and Test Driven Development

Task overview

This task is about using Unit Tests to apply the Test Driven Development methodology to implement an API that exposes to its clients two endpoints:

- `get_coordinates(IN city_name)`, that receives a city name and returns either an error (*not found* or any other error) or some representation of the latitude and longitude of the specified city.
- `get_distance(IN coordinates1, IN coordinates2)`, that receives two coordinates (represented in the same format as outputted by the previous function) and returns the distance between the two coordinates or some error.

Considerations

For the implementation of `get_coordinates(IN city_name)`, you are expected to use *openstreetmaps*. Specifically, you can query openstreetmaps using, for example, curl in the following way:

```
curl "https://nominatim.openstreetmap.org/search?q=lima,peru&format=json"
```

1 Tasks

1. Design the solution.
2. Write unit tests that checks structural aspects. For example:
   - `can_call_existing_endpoints_of_the_API` (a function that confirms that existing endpoints can be called)
   - `cannot_call_not_existing_endpoints_of_the_API` (a function that confirms that non-existing endpoints cannot cannot be called)
   - Similarly, functions that test that the functions one calls work with the correct number of parameters and do not work with the incorrect number.
3. Implement the necessary code to pass the unit tests defined in point (2).
4. Implement a new unit test called `endpoint_returns_something`, which checks that the endpoint is not only callable but also returns something.
5. Implement whatever is necessary in the code to pass the test written in point (4).
6. Write a new unit test called `the_result_is_correct_for_simple_cases`, which checks that the function returns the correct value for some specific input cases.
7. Write code to pass the unit test added in (6).
8. Write a test called `the_result_is_correct_for_all_inputs`, which tests that the matches the expected output for a vast set of possible inputs.
9. Write the code to pass test (8).

10. Measure the code coverage of the created code. It must be at least 98%. If it is less than 98%, add tests to improve it.

11. Perform stress tests and measure availability, latency, and the percentage of repeated code in the proposed implementation.

12. ENSURE THAT THE CODE IS MAINTAINABLE. Use strict naming conventions for variables, use indenters that do not allow merging code that has not yet passed through an indenter, and require long and explanatory comments before each function. Before each function, there must be comments that explain AT LEAST the following:

    - General description of the function's input/output.
    - Parameters of the function.
    - The idea of the function.

Note Functions must be short and have a single responsibility. A function should not be a "sequence of things" but one thing. That thing can be something "high-level" and call other functions. But it must be clear "what each function does."