HW1: Mid-term assignment report

Victor Barros Melo [101099], v2023-04-11

1 Introduction

1.1 Overview of the work

This report presents the midterm individual project required for TQS, covering both the software product features and the adopted quality assurance strategy.
My proposition to this assignment is *Aeris* (air in Latin), which is an application that shows the user air quality stats and predictions for a given city. The information is retrieved from two different APIs.

1.2 Current limitations

Unfortunately, the API response time is increased for some cities, due to the response time of both third-party APIs. This can be seen when the user wats to check the status for Paris, where the response time is decent, but when it come to some Portuguese cities the response time is increased, like when asking for data from Porto or Aveiro.

2 Product specification

2.1 Functional scope and supported interactions

One of the main scenarios is using the API to get air quality information for a city, where the user provides the city's name, after that the API use the third-party APIs to retrieve the desired data and then concatenates useful information, structures it and send to the user.

The target audience of the application are normal people or organizations interested in gathering information about a specific city's air quality.

One great example of usage is a student who needs to write a report about the pollution and the air quality levels for their beloved hometown. In this case the student can use *Aeris* to easily retrieve air quality levels for today and for the next five days. After that, the student can gather all the data provided by the application and write it's beautiful and complete report.

2.2 System architecture

The application is divided in two main segments, which are the Frontend and Backend of the application, also the Backend has some important sublevels.

Frontend:

- NextJS (a ReactJS framework):
    - This framework makes the entire front-end of the application, making have use of JavaScript, HTML5, CSS Modules, Axios (for handling the communications with the backend), React Bootstrap and also React Toastify (for the aesthetically pleasing error handling toasts)

Backend

- Spring Boot:
    - Made with Java Spring Boot, the API is the base for the app's whole operation
- Junit and Mockito:
    - Testing
- AmbeeData and OpenWeather:
    - Third party APIs

Project and version managing:

- Github

2.3 API for developers

The API uses two external sources, which are Ambee Air Quality API and the OpenWather Map Air Pollution API, to fetch the data.

Endpoints:

/airquality/{city}
This endpoint fetches the data from the Ambee API and returns it to the user, if the city is not found it returns the 404 HTTP Status for Not Found.

/airquality/predictions/{city}
Fetches the air quality predictions for the next five days for a specific city from the OpenWeatherMap Air Pollution API.

/airquality/stats
Return statistics about the usage of API (API calls) and the cache (cache hits and misses)

The API also has a reset endpoint to reset all the counters and also its cache.

3 Quality assurance

3.1 Overall strategy for testing

The general strategy was TDD, tests were created before implementing any new functionalities, which was crucial when working with two distinct data sources.

3.2 Unit and integration testing

The APIs capabilities, cache and statistics were tested, also an integration test to verify for the stats endpoint.

Methods used in the test class:

- contextLoads():
    - an empty test method that checks if the Spring Boot application context is properly loaded.
- apiCallsCounterisWorking():
    - a test method that verifies if the cache's API calls counter is working by calling the /airquality endpoint three times and checking if the counter increments by three.
- getLatLong():
    - a test method that verifies if the /airquality endpoint returns the correct latitude and longitude for a given location.

- testAirQualityStatsEndpoint():
    - an integration test that checks if the /airquality/stats endpoint returns a status of 200 (OK).
- testCleanCache():
    - a test method that checks if the cache is empty after calling the /airquality/clean endpoint.
- cacheCalled():
    - a test method that checks if the cache is working by calling the /airquality endpoint three times and verifying if the cache hits, cache misses, and total API calls counters are correct.

3.3 Functional testing

Unfortunately, I could not run any function testing due to incompatible browser drivers, even though Opera GX, Chrome, Firefox and Edge were tested but none could be recognized by its respective web driver in SpringBoot.

3.4 Code quality analysis

Unfortunately, I've also not been able to use SonarQube successfully with the app.

3.5 Continuous integration pipeline [optional]

No CI Pipelines were implemented.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/vbmelo/TQS_101099/tree/main/Project01
Video demo	https://youtu.be/5Jf2_Z8ozuE
Deployment ready to use	Only frontend: https://aeris.vercel.app/

Reference materials

- https://openweathermap.org/api/air-pollution#current
- https://www.getambee.com
- https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-maven
- https://www.youtube.com/watch?v=LXRU-Z36GEU&t=4835s