HW1: Mid-term assignment report

Pedro Laranjinha Casimiro [93179], 14-05-2021

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.

The purpose of this Spring Boot project is to deliver data about the air quality of various locations. This is mostly done by supplying either the location name and country code or the coordinates to the REST API or the included website, which uses that same API.

All the data that this project uses comes from 2 other API, OpenWeatherMap and WeatherBit, the second one being used only when the first is unavailable. All query results are cached for 1 minute for repeated use.

The contents of the cache and other stats about our service can also be gotten with our REST API.

1.2 Current limitations

This project does not specify why a query did not work and always returns “404: Not found”, and the website doesn’t properly format the query results.

2 Product specification

2.1 Functional scope and supported interactions

The purpose of this project is to give the users simple queries to get the air quality of a location. The users can choose the location either through its name or through its coordinates, and they can also choose to get the current air quality, the hourly air quality of the past 72 hours or the hourly air quality forecast for the next 72 hours.

This project has a simple website for casual users and a REST API for developers.

2.2 System architecture

Spring Boot with Maven is used as the base framework for this project and deals with most of the work that comes with the setup, configuration and connections.

For the queries to the other services and from the website to our REST controller, RestTemplate is used alongside data models to query and parse data.

Our service uses 3 custom caches, so we can differentiate between requests for current/historical/forecast data. This data is stored whenever the service queries the external API, and is refreshed when it is queried again and no longer valid.

This project follows the MVC architecture.

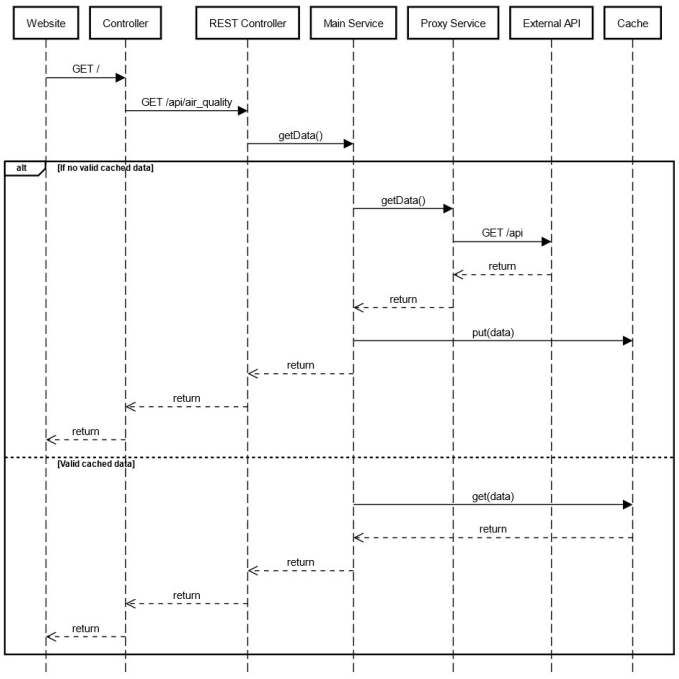

Figure 1 – UML Sequence Diagram

2.3 API for developers

Figure 2 - REST API

Besides the last endpoint which gets the current cached data and stats about the service, the others (in the same order as they appear in the image):

- Returns the current air quality of the coordinates.
- Returns the air quality of the last 72 hours of the coordinates.
- Returns the 72 hour forecast for the air quality of the coordinates.
- Returns the current air quality of the location.
- Returns the air quality of the last 72 hours of the location.
- Returns the 72 hour forecast for the air quality of the location.

3 Quality assurance

3.1 Overall strategy for testing

Most of this project was done without tests, as I already had a good idea of what had to be done, but when developing and using tests I did find some errors and features missing from my implementation.

JUnit 5 and Mockito were used for unit tests; JUnit 5, Mockito, MockMvc and REST-Assured were used for integration tests; JUnit 5, Selenium IDE and Selenium-Jupiter were used in functional testing. For asserts, both AssertJ and Hamcrest were used.

3.2 Unit and integration testing

Unit testing was done for the cache and all services.

For the cache, all of its methods were tested, the getting and removing of data, the expiring of data, the getting of invalid data.

For the main service, the one which the controller accesses, all ways of getting data were tested, the getting of data from either of the external services, the getting of data from the cache, the getting of different types of data (current/historical/forecast) with different inputs (location/coordinates), the getting of data with invalid inputs, the getting of expired data, and the getting of all the cached data with the service's stats. Mocks were used in the main service to simulate the calls to the proxy services.

The services which act as a proxy for the external API, have similar tests to the main service but more specific to the external API, without the invalid inputs which were filtered in the main service and without the getting of expired data, as these services don't use the cache. Here, mocks were used mostly for the service testing to simulate the REST requests so it didn't have to actually call the external API.

Integration testing was only used for the REST controller to simulate that the app was running, but the tests themselves were also similar to the ones used for the services, which entail testing requests for different types of data (current/historical/forecast) with different inputs (location/coordinates), some of which are invalid.

3.3 Functional testing

Functional testing was used to test all possible kinds of inputs in the 3 forms of the website. The valid and invalid location names and country codes in the first form and the valid and invalid coordinates in the second form. As the last form is a single button to get the cache data, I wasn't able to check the inputs, just the output before and after another type of request.

3.4 Static code analysis

For static code analysis, SonarQube and IntelliJ's integrated code analysis were used, the first one through a container.

Mostly what I learned with the use of these tools is that I'm careless about encapsulation, the use of access level modifiers in classes, methods and attributes, which I was already slightly aware of.

This is the analysis results as of the time this report was made.

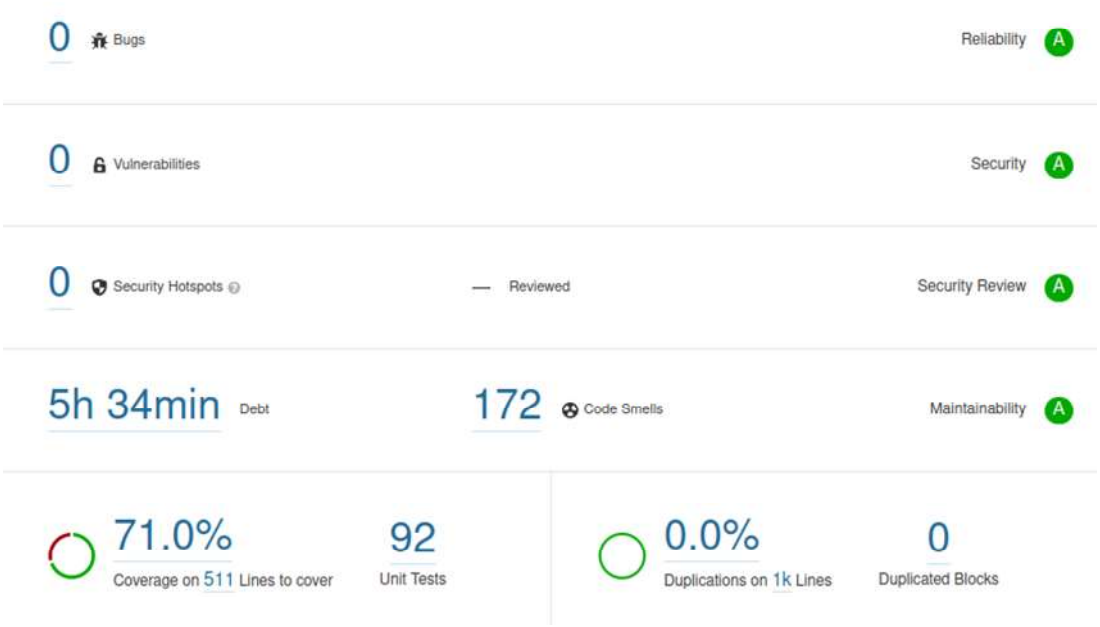

Figure 3 - SonarQube Dashboard

Even though this project has 5h34min of technical debt with 172 code smells, it looks worse than in actually is, as many of the code smells are repeats caused by the use of many similar methods.

The majority of uncovered lines are due to either the data models or simple conditions to check if a necessary variable is null. As the cases in which said variables are null are already tested without the use of these conditions, I don't find these uncovered lines a problem.

3.5 Continuous integration pipeline [optional]

There was an attempt at using SonarCloud but because the repository used for this project also stores many others, this attempt ended in failure.

4 References & resources

Project resources

- Video demo
  - https://github.com/p-laranjinha/tqs-portfolio/tree/master/hw

Reference materials

- OpenWeatherMap API documentation
  - https://openweathermap.org/api
- WeatherBit API documentation
  - https://www.weatherbit.io/api

- Baeldung’s guide to RestTemplate
    - https://www.baeldung.com/rest-template