HW1: Mid-term assignment report

*Luís Valentim [93989]*, 2021-05-15

Conteúdo

[HW1: Mid-term assignment report 1](#_Toc71663603)

[1. Introduction 1](#_Toc71663604)

[Overview of the work 1](#_Toc71663605)

[Current limitations 2](#_Toc71663606)

[2. Product Specification 2](#_Toc71663607)

[Functional scope and supported interactions 2](#_Toc71663608)

[System Architecture 2](#_Toc71663609)

[API for developers 3](#_Toc71663610)

[3. Quality Assurance 3](#_Toc71663611)

[Overall strategy for testing 3](#_Toc71663612)

[Unit and integration testing 3](#_Toc71663613)

[System testing 4](#_Toc71663614)

[Static code analysis 4](#_Toc71663615)

[4. References and resources 4](#_Toc71663616)

# Introduction

## 

## 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.

There has been developed a simple webpage to showcase a REST API for air quality information in a specific city, with a Cache service and no long-term storage solution. The main focus however was the quality assurance, mainly through unit tests, integration tests, functional testing and static code analysis however due to time constraints no continuous integration pipeline solution was implemented.

## 

## Current limitations

As previously mentioned there is no CI pipeline implemented, there is only one external API, weatherbit, from which the service retrieves data, meaning the app is reliant on weatherbits API availability in order to work properly.

# Product Specification

## Functional scope and supported interactions

TQS\_AirQuality is a webapp to retrieve air quality information on certain cities in Portugal, air quality data is retrieved from an external API weatherbit which provides the following data, mold level, aqi (air quality index), pm10 (particles with a diameter of 10 micrometers or fine particles), pm25(tiny particles or droplets in the air that are two and one half microns or less in width), co concentration, o3 concentration, no2 concentration, so2 concentration, predominant pollen type, pollen level from trees, pollen level from weeds, pollen level grass.

The service provides Cache storage capable of saving information about a city’s current air quality, with a fixed timeout of 200 seconds, successful access, and failed access as long as statistic information about said data, however no long-term storage solution as it is not particularly relevant for a live information service.

## System Architecture





REST Controller

Cache

Service Layer

Front End

## API for developers

The project provides a small REST API with three endpoints, regarding city data, cities in cache and cache usage statistics.

**/AirQuality/{city}** - Retrieves the {city} air quality info   
Example for /AirQuality/Porto: [{"mold\_level":1,"aqi":54.0,"pm10":16.6128,"co":323.355,"o3":117.481,"predominant\_pollen\_type":"Molds","so2":0.829808,"pollen\_level\_tree":1,"pollen\_level\_weed":1,"no2":0.883592,"pm25":3.8599,"pollen\_level\_grass":1}]

**/AirQuality/cities** - Retrieves all available cities on cache memory   
Example for /AirQuality/cities: ["Porto","Aveiro","Lisboa","Porto"]

**/AirQuality/stats** - Retrieves information about cache memory usage  
Example for /AirQuality/stats: {"hit":"5","cities":"[Lisboa, Porto, Aveiro]","miss":"4"}

# Quality Assurance

## Overall strategy for testing

The project development tried to follow a TDD, Test Driven Development, as much as possible, using test cases as the main way to verify requisites of each module, from stand alone modules which rely mostly on unit testing like Weather, Response or Cache with Junit to integration testing in the WeatherbitAPIService and AirQualityController through mocking and finally system testing with Selenium on the actual webapp.

Hamcrest and Cucumber were not adopted for this approach as they are used to simplify testing and do not provide further testing functionality and the project’s small dimension make testing simple enough.

Lastly Sonarqube was used to review the whole project, the only issues found were code smells, so it wasn’t used to its full potential, but certainly useful as well.

## Unit and integration testing

Regarding unit and integration testing, every class was thoroughly tested in order to verify every class is functioning as expected.

The Model and Cache classes were tested with regular unit testing on every method or couple of methods, one or more tests depending on how complex the methods are and how many ways they may behave and interact with each other, these tests are all for the most part small and straight forward. Junit was used to verify assertions and set useful beans.

Integration testing used both Mocks and Junit to verify interactions in the Service and Controller classes. These tests made sure different modules interacted as they should and gave proper results, this could be achieved though mocking which let’s us configure functional behavior for modules not being tested at the moment in order to see how the tested module would react to the mocked module and verify its results are what was initially expected.

## System testing

Selenium was used for system testing, through the IDE it records actions and assertions though interaction in the web page on a deployed instance of the project and later translates it to java which can be tested through an emulation driver for a specific browser, in this case Chrome. These tests are produced by interacting with the page meaning they are highly adaptable and test the project functionality as a whole and in this sense are very powerful tests, unfortunately due to the small complexity of the project and web interface in specific it is rather limited in this case.

## Static code analysis

Static code analysis is used in order to verify code quality, from bugs to security issues, code smells, code coverage, maintainability, reliability.

In this project the code was reviewed though Sonarqube a very powerful tool capable of providing all previously mentioned parameters as well as many others, identify issues, set quality thresholds based on these parameters and inform the user on the reasoning behind each problem.

In this case it was only used to fix code smells, some of which were left behind due to being related to naming inconsistencies which were flagged as smells due to ‘\_’, however these depend on naming conventions and I did not find them relevant.

# References and resources

<https://www.weatherbit.io/account/login>

<https://www.sonarqube.org/>

<https://www.selenium.dev/>

<https://junit.org/junit5/>

<https://site.mockito.org/>

<https://www.youtube.com/watch?v=WPKv8NA-ZhE>

<https://www.w3schools.com/js/js_htmldom_html.asp>

<https://stackoverflow.com/questions/2075337/uncaught-referenceerror-is-not-defined>

<https://stackoverflow.com/questions/31542824/div-inside-of-anchor-stop-redirect>

<https://www.geeksforgeeks.org/hashmap-isempty-method-in-java/>

<https://www.w3schools.com/js/js_object_properties.asp#:~:text=Properties%20are%20the%20values%20associated,but%20some%20are%20read%20only>.

<https://stackoverflow.com/questions/9529327/change-the-value-of-h1-element-within-a-form-with-javascript>

<https://stackoverflow.com/questions/12323403/how-do-i-find-an-element-that-contains-specific-text-in-selenium-webdriver-pyth>

<https://stackoverflow.com/questions/50368319/selenium-how-to-determine-if-element-has-child>

<https://github.com/tiagocmendes/spring-air-quality>

<https://github.com/carinaneves15/airQuality>

<https://github.com/tomas99batista/AirQuality-WebApp>

<https://github.com/Diaj3/AirQualitySpringWebApp>