**HW1: Mid-term assignment report**

*Diogo Moreira [93127]*

*2020/2021*

[1 Introduction 1](#_Toc70786521)

[1.1 Overview of the work 1](#_Toc70786522)

[1.2 Current limitations 1](#_Toc70786523)

[2 Product specification 1](#_Toc70786524)

[2.1 Functional scope and supported interactions 1](#_Toc70786525)

[2.2 System architecture 2](#_Toc70786526)

[2.3 API for developers 2](#_Toc70786527)

[3 Quality assurance 2](#_Toc70786528)

[3.1 Overall strategy for testing 2](#_Toc70786529)

[3.2 Unit and integration testing 2](#_Toc70786530)

[3.3 Functional testing 2](#_Toc70786531)

[3.4 Static code analysis 2](#_Toc70786532)

[3.5 Continuous integration pipeline [optional] 3](#_Toc70786533)

[4 References & resources 3](#_Toc70786534)

# 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.

The current application uses an external API to give the user a forecast of the air quality in a specific city of his choice.

## Current limitations

The main limitation of the application is due to its use of an external API. The cities the user can search for are limited and the only way to search is by name. Furthermore, the optional features were not implemented and the interface could be improved.

# Product specification

## Functional scope and supported interactions

The application is meant to be used by anyone who wants to search for a **city’s air quality forecast**.

The user only needs to introduce the city’s name in the search box (e.g. Oporto) and they will be shown its forecast. Even more, there are endpoints specially made for the developers that give the forecast data and its related statistics (hits, misses and number of requests) in a **JSON** format.

## System architecture

Interaction with the interface can be done in two ways:

* The web Interface
* The API

The **web interface** is shown to the user with the help of **AirQualityController** (@Controller) and **ThymeLeaf**. Most of the **business logic** is processed through the **CityAqService** (@Service) such as requesting the **forecast data from the external API**.

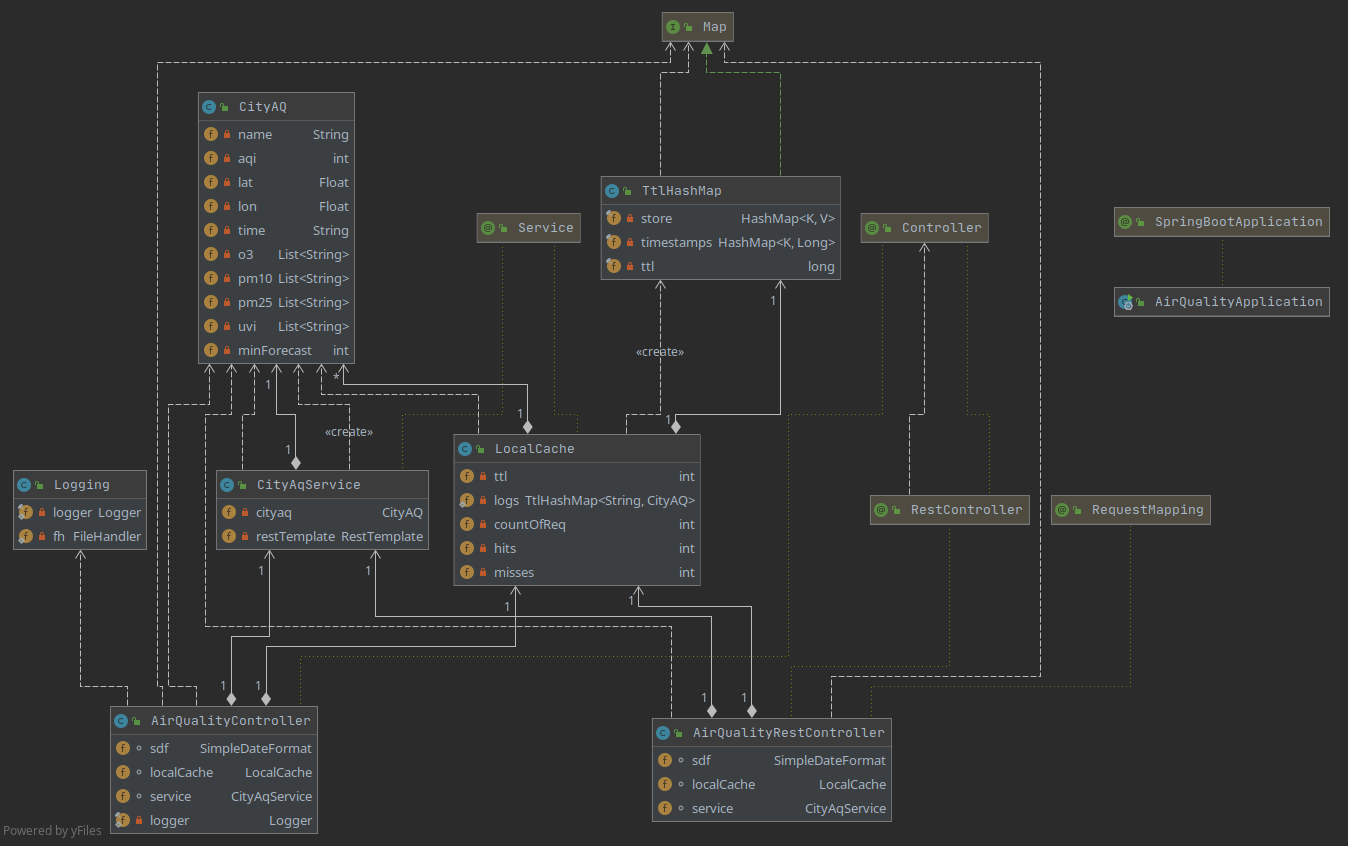
The **cache** is handled by **LocalCache** (@Service) and it consists in the use of **TtlHashmap**, which in its essence is a Hashmap with a Time to live, where CityAQ objects are stored and three variables that serve for statistical purposes only.

The **Logging** class is only used to **write the logs in a text file** that can later be used for debugging.

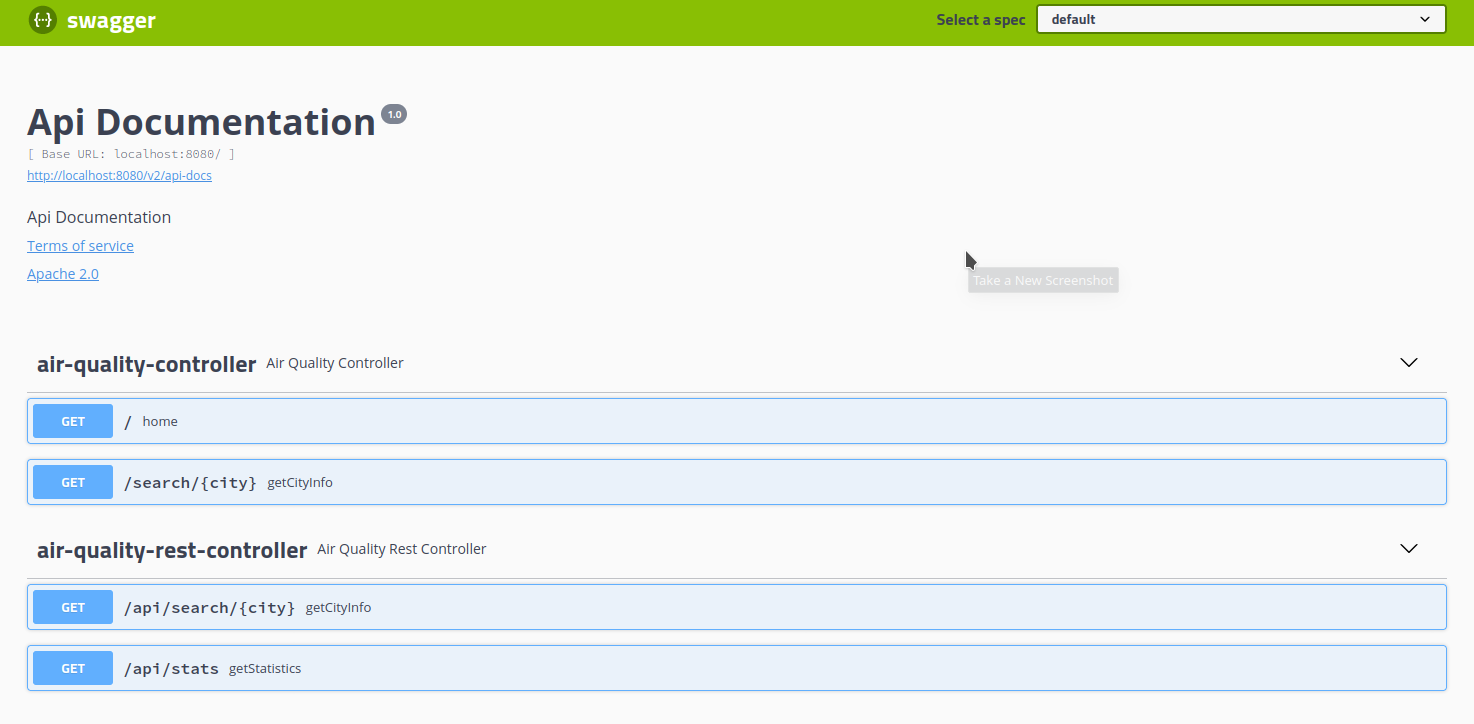
In regards to the **API**, the logic is similar but, instead of interacting with the web interface, it uses the **AirQualityRestController** (@RestController) to return **JSON** data whenever an **endpoint** is called.

The technologies and frameworks used for this project were:

* Spring Boot
* ThymeLeaf
* Html, CSS, JS



## API for developers



# Quality assurance

## Overall strategy for testing

The overall strategy used for testing was after implementing most of the features start testing them to see if they are working as intended.

The tools used for testing were:

* **MockMvc** for the API’s unit test
* **Rest-Assured** for the API’s Integration test
* **JUnit** and **Hamcrest** for API’s unit and integration tests
* **Selenium** for the interface tests
* **Mockito** for API’s unit and local cache test

## Unit and integration testing

Unit tests for the API (AirQualityRestControllerUnitTest):

* When city exists and isn’t in cache
* When city exists and is in cache
* When city doesn’t exist
* Statistics

Integration tests for the API (AirQualityRestControllerIT):

* City exists but not in cache and returns correct forecast
* City exists in cache and returns correct forecast
* Statistics working correctly

Unit Tests for local cache (LocalCacheTest):

* Adding a new city
* Confirming city exists in cache
* Retrieving city from cache
* Number of hits/misses/requests incrementing correctly
* Time to live expires and city is removed

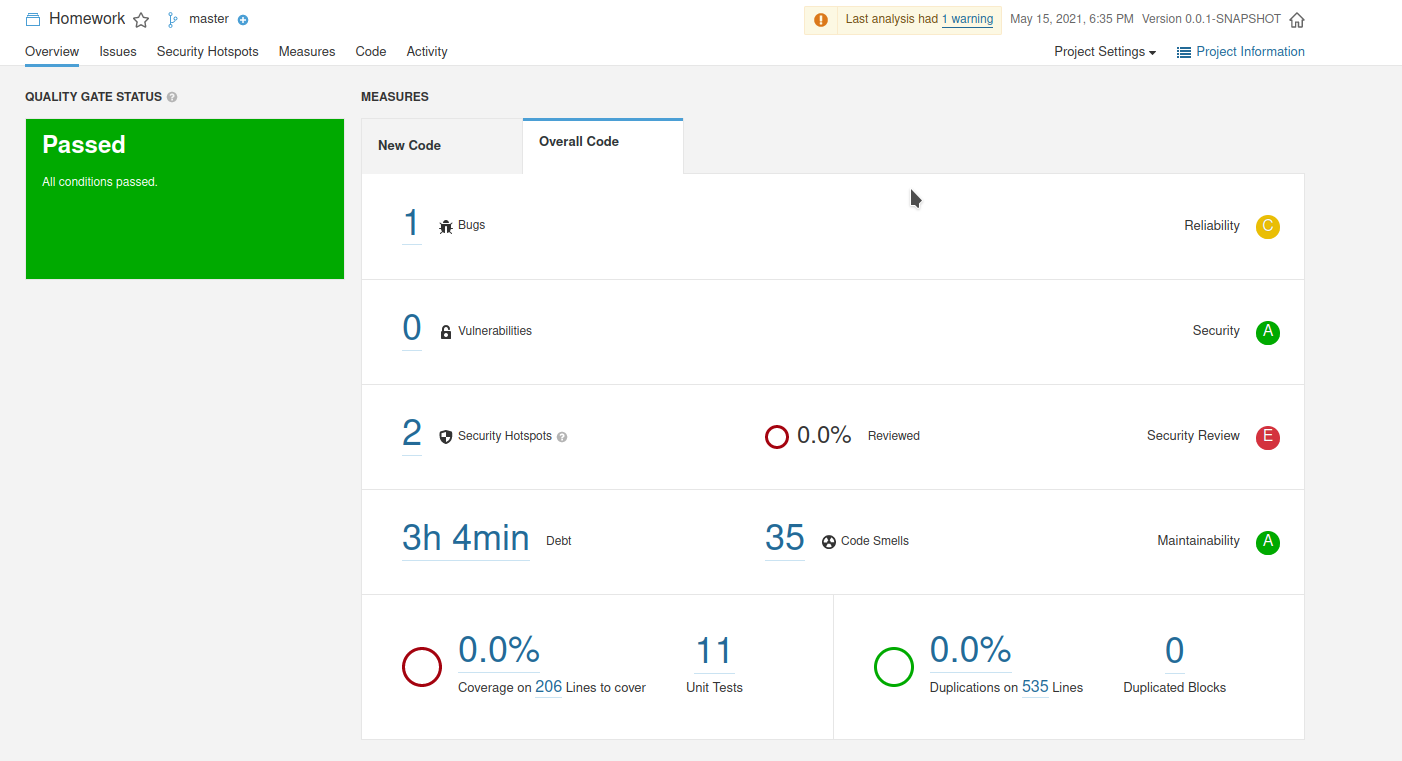
## Functional testing

The only way to search by using the **interface** is using the search bar and writing the city name.

**Selenium IDE** was used to test the interface making sure that everything was working as intended.

* Searched many different cities to confirm the search bar was working as intended
* Confirmed that after searching for ‘Oporto’ the name of the city was ‘Sobreiras-Lordelo do Ouro, Porto, Portugal’ as it should and not any other.

## Static code analysis



Only one bug was found but it doesn’t affect the program in any way.

The security hotspots are due to the token for the call to the external API.

35 code smells were found but are for minor reasons like using logger.info or variable names.

The 0% seems to be for no specific reason since I could not find anything about it.

# References & resources

Project resources

Video demo and code are both available in the Git repository:

<https://github.com/diogoemoreira/TQS_homework>

Sonar Cloud doesn’t give code analysis of the Java code:

<https://sonarcloud.io/dashboard?id=diogoemoreira_TQS_homework&branch=main>

Reference materials

* <https://gist.github.com/joelittlejohn/5565410> (TtlHashMap)
* <https://www.themezy.com/free-website-templates/128-steel-weather-free-responsive-website-template> (html template)
* <https://stackoverflow.com/questions/15758685/how-to-write-logs-in-text-file-when-using-java-util-logging-logger> (Logging to file)
* <https://pt.stackoverflow.com/> (for any coding questions)
* <https://www.youtube.com/watch?v=HHyjWc0ASl8&t=111s&ab_channel=TechPrimersTechPrimers> (swagger in SpringBoot)
* <https://sonarcloud.io/> (Sonar Cloud)
* <https://www.sonarqube.org/> (Sonar Qube)
* <https://www.thymeleaf.org/> (ThymeLeaf)