HW1: Mid-term assignment report

*Tomás dos Santos Batista [89296]*, 2020-04-15

[1 Introduction 1](#_Toc36219510)

[1.1 Overview of the work 1](#_Toc36219511)

[1.2 Limitations 1](#_Toc36219512)

[2 Product specification 1](#_Toc36219513)

[2.1 Functional scope and supported interactions 1](#_Toc36219514)

[2.2 System architecture 3](#_Toc36219515)

[2.3 API for developers 5](#_Toc36219516)

[3 Quality assurance 5](#_Toc36219517)

[3.1 Overall strategy for testing 5](#_Toc36219518)

[3.2 Unit and integration testing 6](#_Toc36219519)

[3.3 Functional testing 6](#_Toc36219520)

[3.4 Static code analysis 6](#_Toc36219521)

[3.5 Continuous integration pipeline [optional] 7](#_Toc36219522)

[4 References & resources 7](#_Toc36219523)

# Introduction

## Overview of the work

AirQuality WebApp is a website to consult the quality of the air in Lisbon and Madrid. It contains multiple parameters (air quality information, PM2.5, temperature, pressure, and many others). It uses a public API to obtain the values.

## Limitations

The only limitations I fought with were some initial Spring-Boot problems. Other than that, everything went fine.

# Product specification

## Functional scope and supported interactions

The users can consult the information through the webapp or through the API of the project. The webapp have a clean view of each’s city information.

Uma imagem com captura de ecrã

Descrição gerada automaticamenteUma imagem com captura de ecrã

Descrição gerada automaticamente

## System architecture

Uma imagem com preto, captura de ecrã, monitor, ecrã

Descrição gerada automaticamente

The project was built using Spring-Boot and ThymeLeaf. I implemented a PostgreSQL database running on a docker container to save the data persistently.

The communication between database ⬄ Spring-Boot is made using a JPARepository. The communication between Frontend/ThymeLeaf ⬄ Spring-Boot is made through a Controller and Models. I also implemented a Cache to save the last searched values.

The way of working of my system can is the following:

Uma imagem com texto

Descrição gerada automaticamente

## API for developers

I made a documentation to support my API. It can be found at: [Postman Documentation - AirQuality WebApp](https://documenter.getpostman.com/view/9124304/SzYgSbE4).

# Quality assurance

## Overall strategy for testing

Uma imagem com captura de ecrã, monitor, preto, ecrã

Descrição gerada automaticamenteI implemented unit testing, Mockito and Selenium web testing. Got an overall of 100% classes and 92% lines covered (due to main not be tested and WebController tests were in a different package).

Uma imagem com sentado, monitor, ecrã, telemóvel

Descrição gerada automaticamente

## Unit and integration testing

I tested all different components.

**CacheManager:**

1. Number of hits and misses
2. TTL times were valid
3. Set and Get Cache from certain city

**CitiesController:**

1. API Stats (number of calls)
2. Cache Stats (number of misses and hits)
3. Call the extern API
4. Get city by certain ID
5. Get All Cities on DB
6. Add new City

**CityModel:**

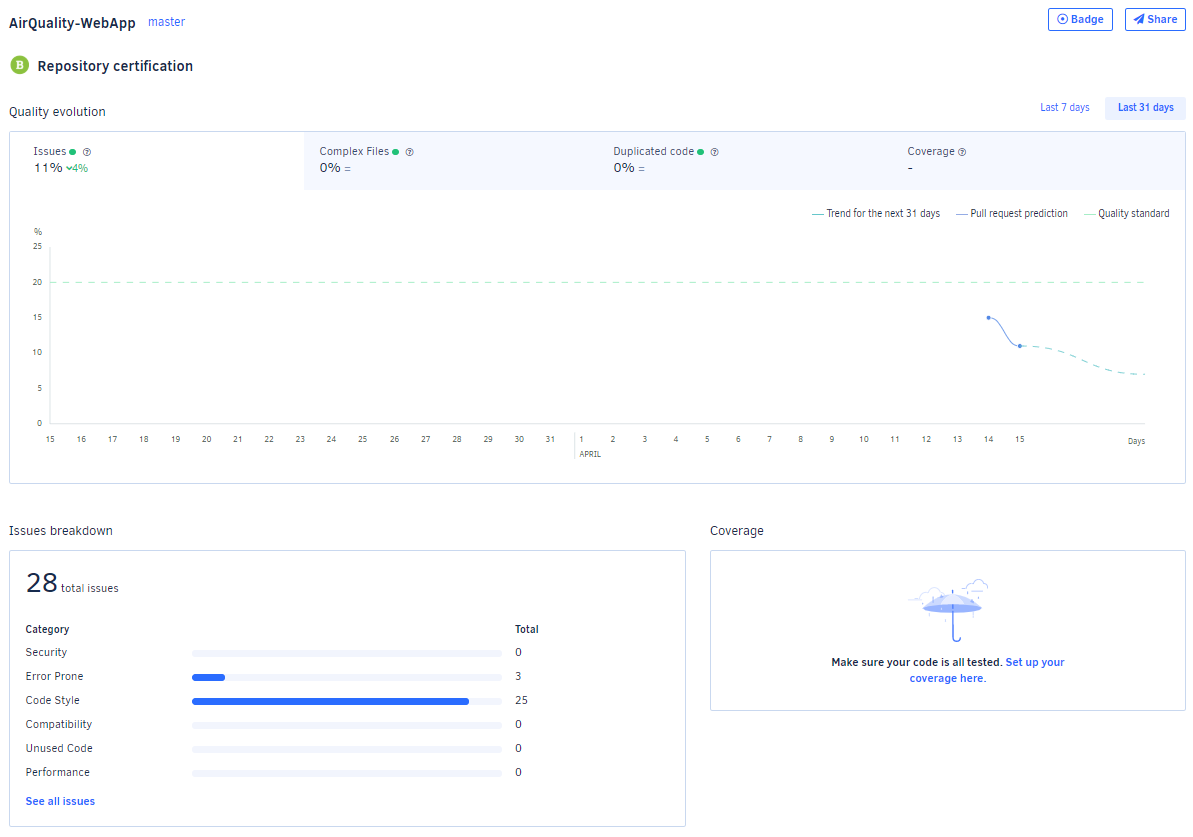
1. City added and retrieved
2. Test all methods (Comparing field by field)
3. Save through repository

## Functional testing

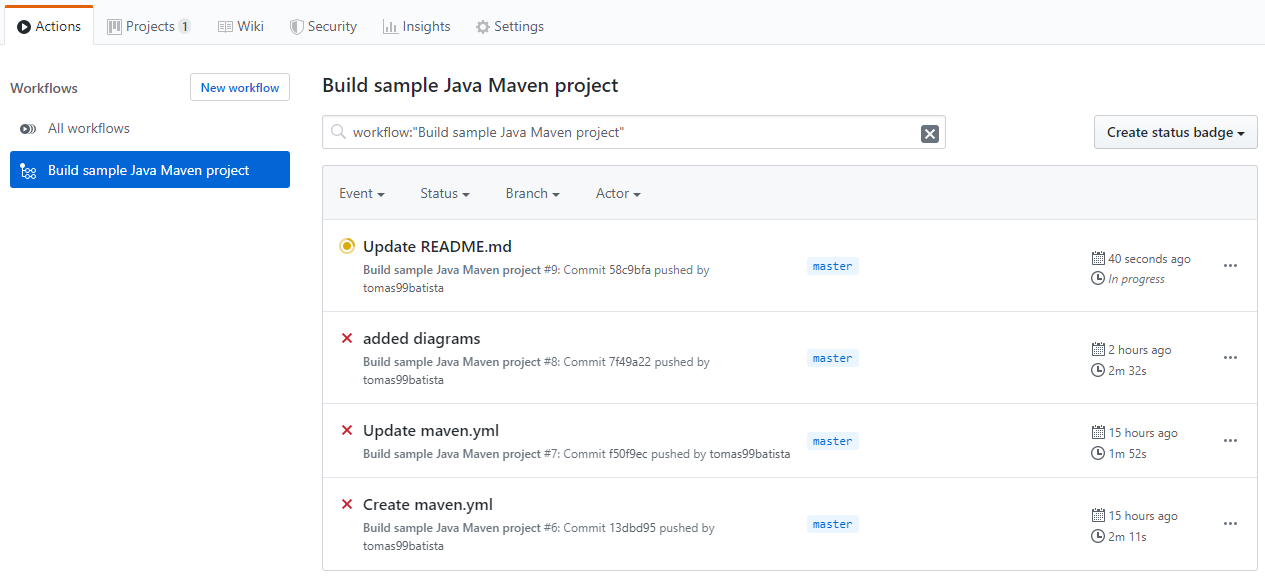
Implemented Selenium IDE tests.

1. Check the info loaded
2. Assert all info loaded
3. Assert that the current page is the specific

## Static code analysis

I implemented Codacy to analyze the code. I picked this software because I have been working with it. It showed me, mostly, unused imports.

## Continuous integration pipeline [optional]

I implemented CI with GitHub actions. Although I used CI, it fails due to not having access to the database (I implemented PostGresSQL running on a docker container).

# References & resources

Project resources

* [Git repository](https://github.com/tomas99batista/AirQuality-WebApp)
* [Video demo](https://www.youtube.com/watch?v=zXiBXXlLE_Q&t=2s)

Reference materials

* [WAQI API](https://waqi.info/)
* [Codacy](https://www.codacy.com/)
* [GitHub](https://github.com/)
* [Spring.io Tutorials](https://spring.io/)
* [CI/CD Tutorial](https://medium.com/faun/continuous-integration-of-java-project-with-github-actions-7a8a0e8246ef)

How to run the project

* **Configure postgres db on docker:** *docker run --name postgres -d -p 5432:5432 -e POSTGRES\_USER=postgres -e POSTGRES\_PASSWORD=password -d postgres*
* **Run:** *mvn spring-boot:run*
* **WebApp:** localhosti:8080/ || localhost:8080/madrid
* **API Calls:** [See endpoints here](https://documenter.getpostman.com/view/9124304/SzYgSbE4?version=latest)