HW1: Mid-term assignment report

*Óscar Fernández Sánchez (101631)*

[1 Introduction 1](#_Toc70786521)

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

[1.2 Current limitations 1](#_Toc70786523)

[2 Product specification 2](#_Toc70786524)

[2.1 Functional scope and supported interactions 2](#_Toc70786525)

[2.2 System architecture 2](#_Toc70786526)

[2.3 API for developers 2](#_Toc70786527)

[3 Quality assurance 3](#_Toc70786528)

[3.1 Overall strategy for testing 3](#_Toc70786529)

[3.2 Unit and integration testing 3](#_Toc70786530)

[3.3 Functional testing 3](#_Toc70786531)

[3.4 Static code analysis 4](#_Toc70786532)

[3.5 Continuous integration pipeline [optional] 4](#_Toc70786533)

[4 References & resources 5](#_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 project is a spring-boot application who reports the air pollutant of a city that you want. The measure is provided in µg/m3, and the data of the cities cames from the (API) and then saved in the in-disk database that is placed in the project directory and temporally in the cache memory. The project also allow to fetch data from its own api, who show the data that is saved from past fetchs to the external api, allowing to search the cities by id, by country prefix or directly listing all the cities that are there.

## Current limitations

For now, the app only can to show the air pollutant of a certain city, but it can (evolucionar) to a most complete app adding some features, for example, list the countries by their air quality or for example implementing an own measurement system using sensors, like those that a microcontroller like Arduino can provide. It should also be noted that that the own rest api doesn´t allow to add or modify data, with the purpose of avoid conflicts with the external api. Also the technology that I used to CI/CD(Heroku), if it detects inactivity in the project it flushes the entire database.

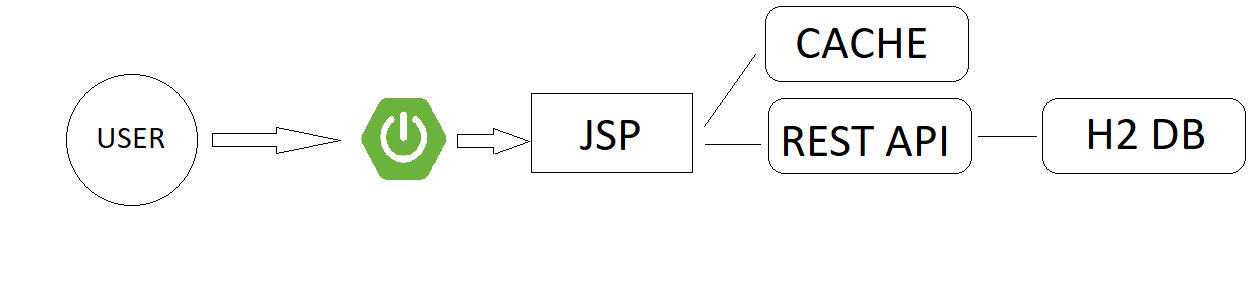
# Product specification

## Functional scope and supported interactions

The main form the user will interact with the app is through the pages that are provided. The user will fill the form with his city and the app will take the data of that city from the external api or from the internal cache. If the data is taked from the external api, the app “clean” the data and stored in his own api, that can be reached using the URLS specified below.

If the user enters a city that´s not saved in the external api, it will be redirected to an error page, which will give the user the opportunity to get back to the form.

## System architecture

The application is based on Spring Boot, and JSP files are used to render the views, previously established the path in the application.properties file. For the REST API, the JPA technology and the H2 database are used, which is saved on disk within the project in the database.mv.db file .

## API for developers

|  |  |  |
| --- | --- | --- |
| Base URL: http://www.localhost:8080/api || [www.tqs1.herokuapp.com/api](http://www.tqs1.herokuapp.com/api) | | |
| Action | URL | Objective |
| GET | /cities | List all the cities stored |
| GET | /city/name/{name} | Get the records of a given city by its name |
| GET | /city/id/{locationId} | Get a certain record of a city |
| GET | /city/ranking | Get the ranking of most contaminated cities |
| GET | /city/random | Get the latest record of a city |
| DELETE | /flush | Delete all the database of the api |
| DELETE | /flush/id/{locationId} | Delete city by its id |

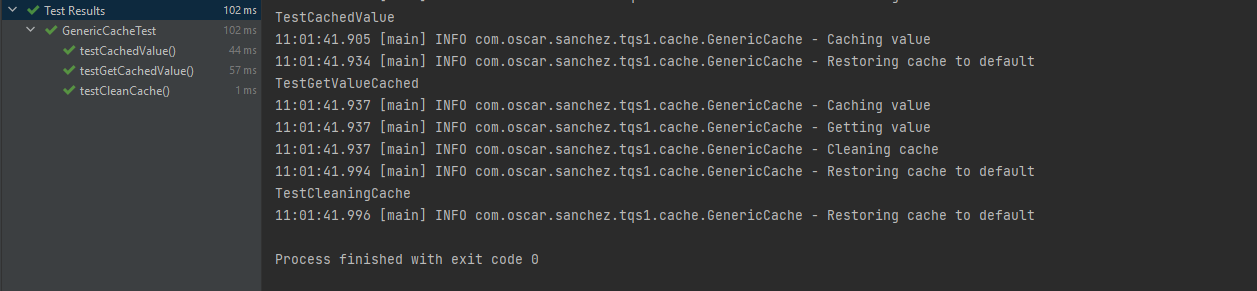
# Quality assurance

## Overall strategy for testing

Due to the simplicity of the project, not many tools were required to perform the tests. JUnit 5 has been used for the cache memory, to check the operation of both the controller and the JPA repository, MockMvc has been the chosen option. Finally, to verify that the views work, Selenium WebDriver has been used.

## Unit and integration testing

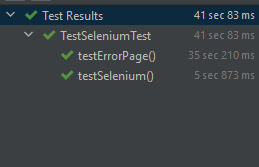
Junit testing have been used to test if the cache memory runs as it expects. For the cache, three main scenarios have been raised: cach a value, get a cached value and finally clean the entire cache. As you can see in the next photo, it returns a satisfactory result.



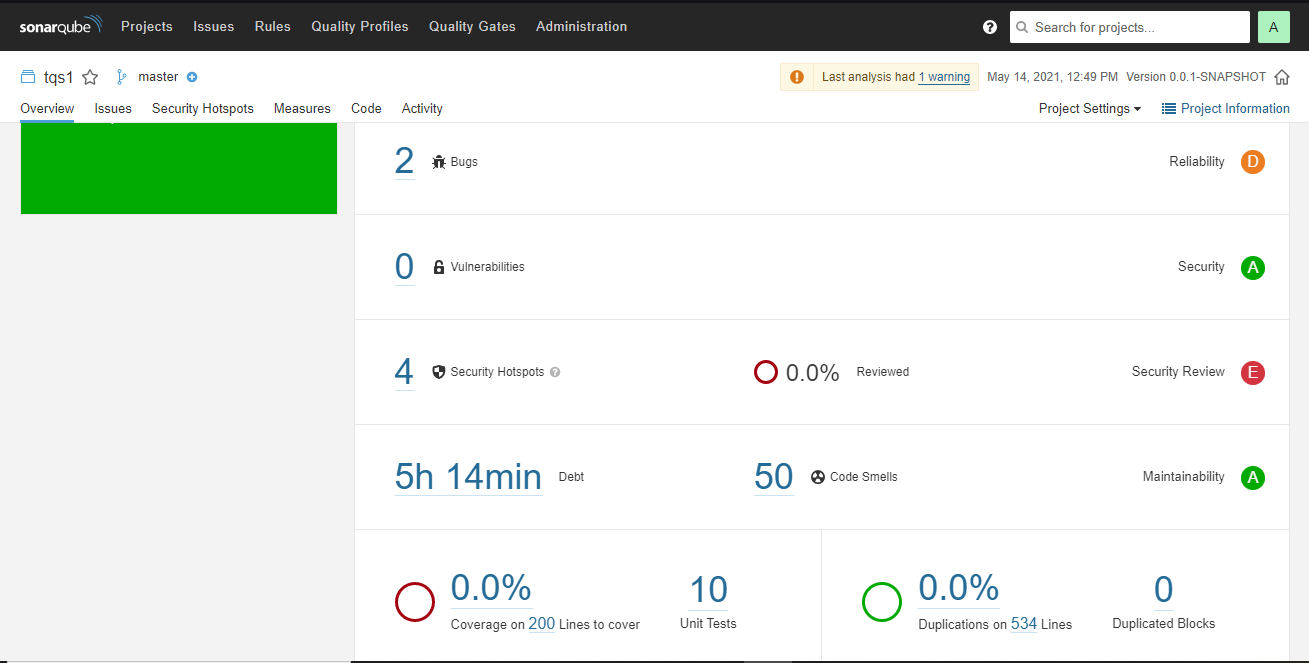
## Functional testing

For the functional testing I used Selenium Web Driver, testing the app through Firefox.

As the application for now is very simple, not many scenarios have been considered. The 2 main scenarios are successful research, and not success and go to error page. As you can see in the screenshots, both test have passed in a satisfactory way, so It´s visible that the front-end is stable.



## Static code analysis

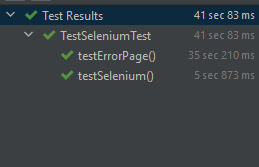


In the exposed image you can see that Sonar Qube notifies that there are 2 bugs, but those bugs are irrelevant, since they have to do with the declaration of the Random class in the method that belongs to the /api/city/random path.

The vulnerabilities that appear are due to the low security provided in the application routes, but it is not worrying as it is not sensitive information such as passwords or other personal information

## Continuous integration pipeline [optional]

For CI/CD I used Heroku, who allows CI/CD if you connect the github project to the page. Every time I push the repository subdirectory, Heroku detects it and builds the project.



Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

# References & resources

Project resources

* Video demo https://github.com/oscarfersan/TesteEQualidadeSoftware/blob/main/tqs1VideoDemo.mp4
* Ready to use application: https://tqs1.herokuapp.com

Reference materials

External API: https://docs.openaq.org/

CacheMemory tutorial:

Spring:https://www.youtube.com/watch?v=kFIvslQQZ9k&list=PLU8oAlHdN5Blq85GIxtKjIXdfHPksV\_Hm

Heroku: https://devcenter.heroku.com