

# HW1: Mid-term assignment report

Ana Alexandra Antunes [876543], v2022-04-07

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Overview of the work	1
1.2	Current limitations	1
<b>2</b>	<b>Product specification</b>	<b>2</b>
2.1	Functional scope and supported interactions	2
2.2	System architecture	2
2.3	API for developers	2
<b>3</b>	<b>Quality assurance</b>	<b>2</b>
3.1	Overall strategy for testing	2
3.2	Unit and integration testing	2
3.3	Functional testing	3
3.4	Code quality analysis	3
3.5	Continuous integration pipeline [optional]	3
<b>4</b>	<b>References &amp; resources</b>	<b>3</b>

## 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 AirQuality application is a simple web-based tool that allows users to check the air quality of any major city in the world based on the Air Quality Index. The tool returns a detailed analysis of the pollutants in the air of the selected city, including h, no2, o3, p, pm10, pm25, so2, t, w, and wg.

### 1.2 Current limitations

While the AirQuality web application provides users with a valuable tool for checking air quality in major cities around the world, it does have some limitations that should be noted. Only Works for City Names: The AirQuality tool is designed to work only with city names, and is not

capable of analyzing air quality data for more specific or generalized geographic locations such as neighborhoods or states/countries. This means that users looking for highly specific air quality data may not find the AirQuality tool to be a sufficient resource. Also, because of the lengthy time for retrieval, specially for far away cities, some information may take so long that will compromise information availability for the user.

## 2 Product specification

### 2.1 Functional scope and supported interactions

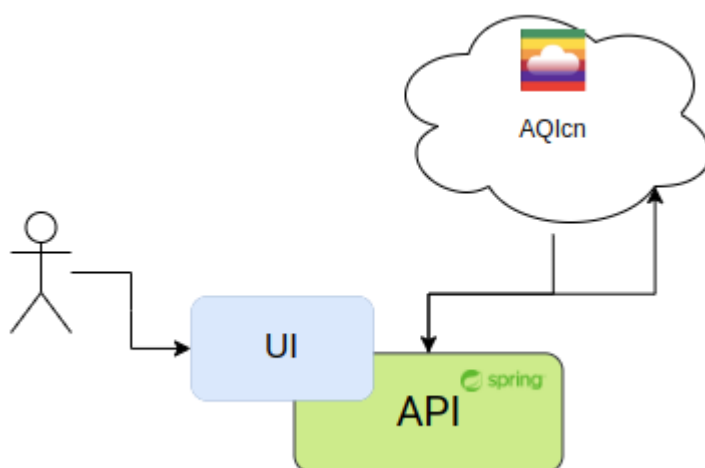
AirQuality is an application designed for anyone who wants to check the air quality index and pollutant rates of a specific city. The main actors who will use this application are people living in or visiting a particular city who want to know the air quality status.

One of the main actors who will use the AirQuality application is a traveler visiting a new city. The main usage scenario for this actor is when they want to check the air quality index and pollutant rates of a city they plan to visit or are currently in.

The traveler can use the AirQuality application by entering the name of the city they are interested in, and the application will display the air quality index and pollutant rates for that city. Based on this information, the traveler can make informed decisions about their outdoor activities, such as sightseeing or outdoor dining. For example, if the air quality index is low, the traveler may choose to avoid outdoor activities that could expose them to pollutants.

Overall, the AirQuality application provides valuable information to travelers, allowing them to make informed decisions about their activities based on the air quality of the city they are in or planning to visit

### 2.2 System architecture



Uses the **Spring Boot framework**, version 2.5.2, with Java 11 as the programming language. Includes dependencies such as Spring Boot Starter **Thymeleaf** and Spring Boot Starter Web for web development, Project Lombok for code simplification, JSON and Commons Collections4 for data handling, and Awaitility for testing.

Uses **Testcontainers** and **JUnit Jupiter** for testing, with **Selenium** and **WebDriverManager** for web UI testing. Includes SpringFox **Swagger UI** for API documentation.  
Managed by the Spring Boot **Maven** plugin.

## 2.3 API for developers

AirQuality info:



GET `/aiqFromCity` Air quality from specified city

Parameters Try it out

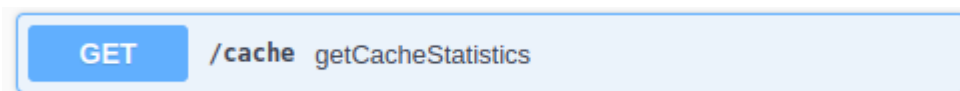
Name	Description
<b>cityName</b> <span>★ required</span> string (query)	cityName

cityName - cityName

Example return from `$ curl localhost:8080/aiqFromCity?cityName=barcelona`:

```
{
  "cityName": "barcelona",
  "aqi": 16,
  "h": 81,
  "no2": 27,
  "o3": 16.3,
  "p": 1020.9,
  "pm10": 19,
  "pm25": 21,
  "so2": 0.6,
  "t": 11.6,
  "w": 0.7,
  "wg": 1.5
}
```

Cache info:



GET `/cache` getCacheStatistics

Sample return from \$ curl localhost:8080/cache:

```
{
  "founds": 0,
  "not founds": 3,
  "total stored": 3,
  "stored": [
    {
      "No2": 27,
      "O3": 16.3,
      "pm25": 21,
      "city name": "barcelona",
      "So2": 0.6,
      "Aqi": 16,
      "pm10": 19
    }
  ],
  "stored now": 1,
  "requests": 3
}
```

### 3 Quality assurance

#### 3.1 Overall strategy for testing

Test-Driven Development (TDD) methodology was used while working on this project. TDD involves writing tests before writing any production code. In this project, several tests in the form of unit tests, integration tests, and end-to-end tests were written. The tests, included AirQualityApplicationTests, CacheStatisticControllerTest, WeatherControllerTest, CacheStatisticTest, SeleniumLisbonSearchTest, CacheServiceTest, and WeatherServiceTest. These tests covered various aspects of the application, including controller functionality, model behavior, service interactions, and UI testing. By writing tests first, it was able to ensure that the code met the requirements and performed as expected. Additionally, TDD helped identify and fix issues early in the development process, reducing the overall cost and time spent on testing and debugging.

Testcontainers provided a way to create and manage containers for integration tests. On the other hand, Selenium was used as web application testing framework that automates browser actions and verifies expected results. Finally, WebDriverManager simplified the management of browser drivers for Selenium.

#### 3.2 Unit and integration testing

Most services were tested using unit and integration tests, here are some detailed information about its uses:

For WeatherService:

```
void testValidCity() void testInvalidCity()
```

For CacheService:

```
void shouldReturnNullForEmptyCache() {...}  
void shouldReturnNullAfterTimeout() {...}
```

For WeatherController:

```
void testValidShouldReturnWeather()  
void testInvalidShouldReturnError()
```

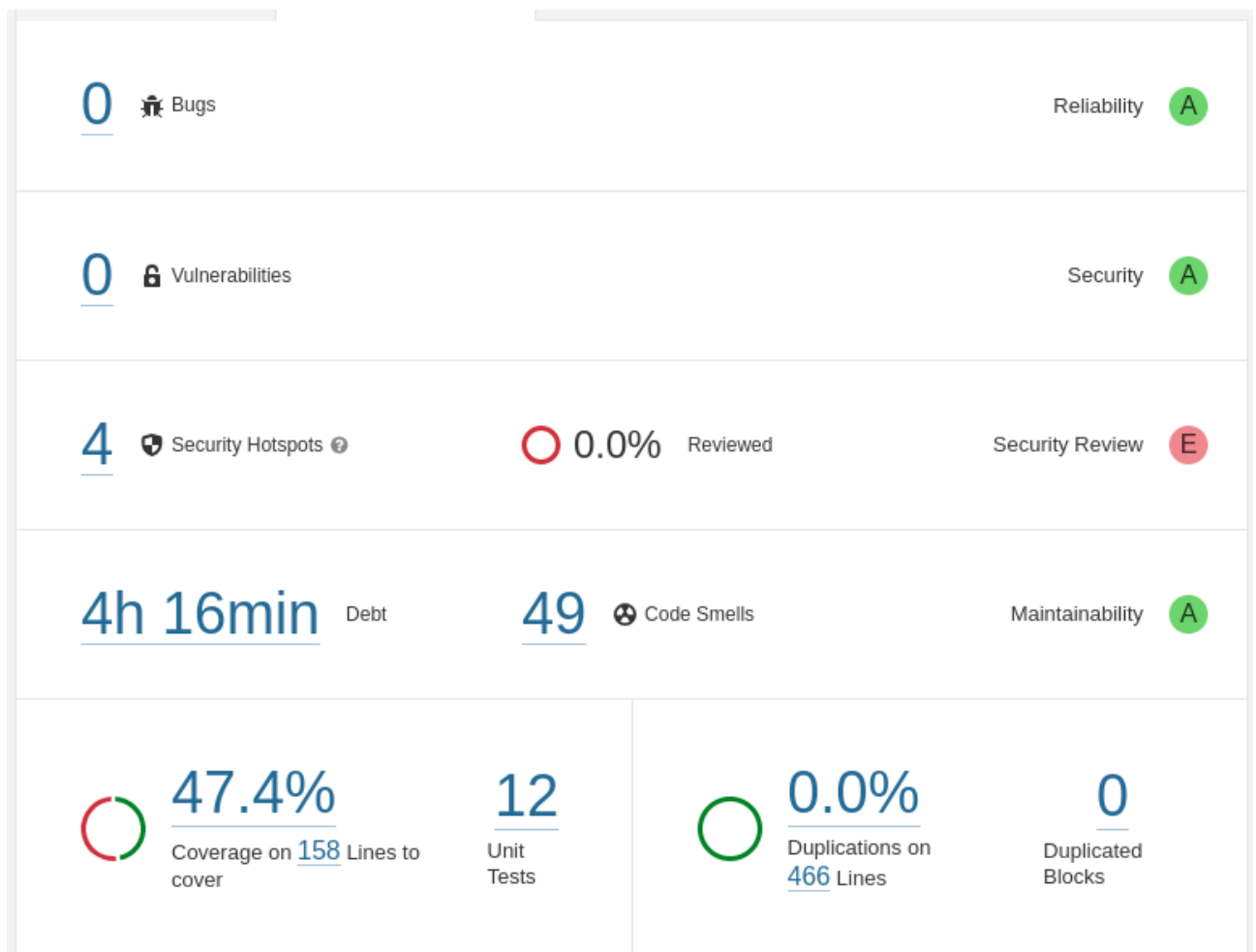
### 3.3 Functional testing

Functional testing was used in this project to verify the behavior of the Air Quality Application from a user's perspective. The test case considered was the search functionality for a specific city, in this case, Lisbon. This test case was implemented using Selenium WebDriver with JUnit testing framework, which simulates the user's interaction with the application by opening the web page and navigating through the elements. The test case involved opening the home page, inputting the city name "Lisbon" in the search bar, and clicking the search button. Finally, the test verified that the city name displayed on the resulting page was the same as the searched city name. Overall, functional testing using Selenium provided an efficient way to validate the application's user interface and ensure that it is working as expected.

```
public void lisbonSearch() {  
    // Test name: lisbonSearch  
    // Step # | name | target | value  
    // 1 | open | / |  
    driver.get("http://localhost:8080/");  
    // 2 | setWindowSize | 621x695 |  
    driver.manage().window().setSize(new Dimension(621, 695));  
    // 3 | click | id=inputCity |  
    driver.findElement(By.id("inputCity")).click();  
    // 4 | type | id=inputCity | lisbon  
    driver.findElement(By.id("inputCity")).sendKeys("lisbon");  
    // 5 | click | css=.btn |  
    driver.findElement(By.cssSelector(".btn")).click();  
  
    assertEquals(driver.findElement(By.id("cityName")).getText(),  
        "lisbon");  
}
```

### 3.4 Code quality analysis

SonarQube was used to make a report about static code analysis and there were the results:



Static code analysis was more than important for code review, it pointed out many relevant issues regarding security and non-cleanliness. Some Code smells were spotted and explained by the SonarQube platform:

```
src/test/java/ua/tqs/airQuality/AirQualityApplicationTests.java
```

```
@SpringBootTest
class AirQualityApplicationTests {

    @Test
    void contextLoads() {

    }

}
```

⚠ Add at least one assertion to this test case.

src/main/java/ua/tqs/airQuality/service/WeatherService.java

```

public WeatherService() {
    uriBuilder = UriComponentsBuilder.newInstance()
        .scheme("http")
        .host("api.waqi.info")
        .path("feed");
    //      .queryParams("token", key);

```

This block of commented-out lines of code should be removed.

---

ality src/test/java/ua/tqs/airQuality/SeleniumLisbonSearchTest.java

```

import static org.junit.Assert.assertEquals;

public class SeleniumLisbonSearchTest {

```

Remove this 'public' modifier.

JUnit5 is more tolerant regarding the visibilities of Test classes than JUnit4, which required everything to be `public`.

In this context, JUnit5 test classes can have any visibility but `private`, however, it is recommended to use the default package visibility, which improves readability of code.

#### Noncompliant Code Example

```
import org.junit.jupiter.api.Test;
```

## 4 References & resources

### Project resources

Resource:	URL/location:
Git repository	<a href="https://github.com/thisbra/tqs_96149/tree/master/HW1/airQuality/src/test/java/ua/tqs/airQuality">https://github.com/thisbra/tqs_96149/tree/master/HW1/airQuality/src/test/java/ua/tqs/airQuality</a>
Video demo	<a href="https://github.com/thisbra/tqs_96149/blob/master/HW1/airquality.mp4">https://github.com/thisbra/tqs_96149/blob/master/HW1/airquality.mp4</a>

### Reference materials

- [Air Quality Programmatic](#) - Air Quality API

- [Spring Boot Documentation](#) - Official documentation for Spring Boot
- [Testcontainers Documentation](#) - Official documentation for Testcontainers
- [Swagger Documentation](#) - Official documentation for Swagger
- [Selenium Documentation](#) - Official documentation for Selenium
- TQS Practical Labs
- [ChatGPT](#)