

# HW1: Mid-term assignment report

## Testing and Software Quality

Rafael Remígio 102435

April 9, 2023

Departamento de Electrónica, Telecomunicações e Informática  
Universidade de Aveiro  
Year 2022/2023

# Contents

<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>1</b>
2.1	Functional scope and supported interactions . . . . .	1
2.1.1	Actors . . . . .	1
2.1.2	Use Cases . . . . .	2
2.2	System architecture . . . . .	2
2.2.1	Architecture . . . . .	2
2.2.2	Technologies and API's Used . . . . .	3
2.3	API . . . . .	4
<b>3</b>	<b>Quality assurance</b>	<b>4</b>
3.1	Logging . . . . .	4
3.2	Overall strategy for testing . . . . .	5
3.3	Unit and integration testing . . . . .	5
3.4	Functional testing . . . . .	5
3.5	Code quality analysis . . . . .	5
3.6	Continuous integration pipeline . . . . .	5
<b>4</b>	<b>References &amp; resources</b>	<b>5</b>
4.1	Reference materials . . . . .	6

## 1 Introduction

### 1.1 Overview of the work

In this assignment the web application developed provides real-time information parameters related to the air quality of a certain location.

### 1.2 Current limitations

## 2 Product specification

### 2.1 Functional scope and supported interactions

#### 2.1.1 Actors

Possible Actors of the proposed product:

- Individuals - People who want to check the air quality in their immediate surroundings, such as their home, office, or neighborhood.
- Public Health Officials - Officials who want to monitor air quality in a particular region or city to make informed decisions on public health policies.

### **2.1.2 Use Cases**

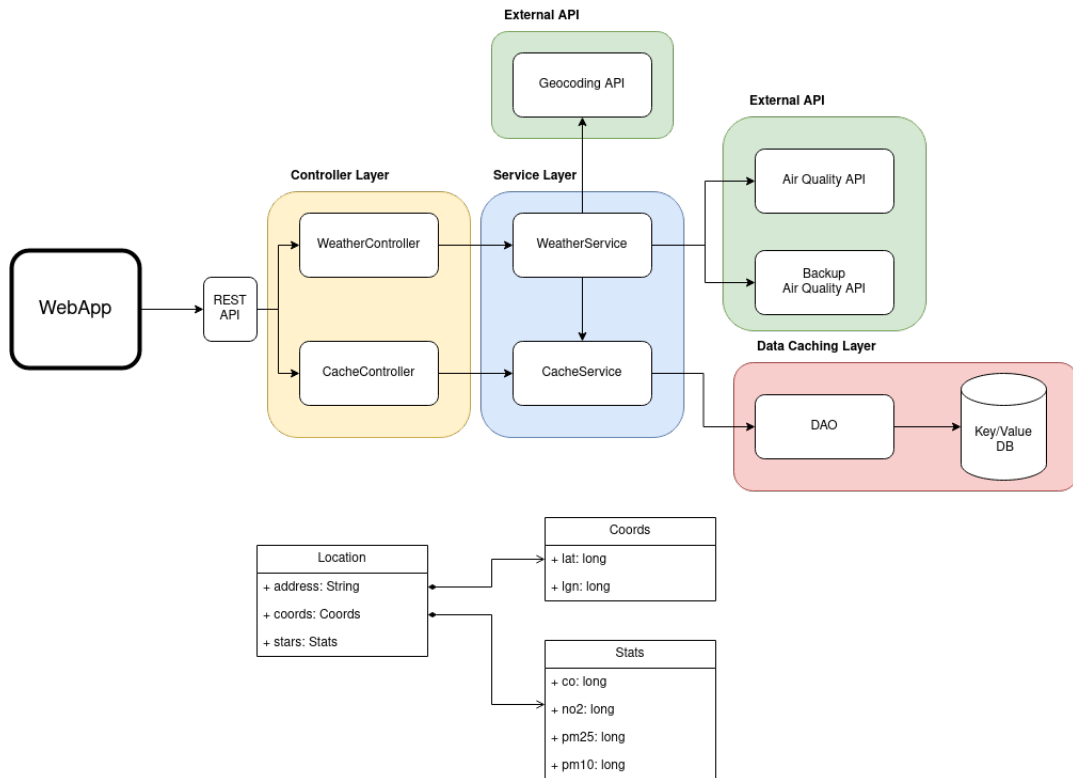
The WebApp provides two use cases:

1. Real-time Air Quality Monitoring. Provides real-time air quality levels in a particular location. This information consists of the specific parameters:
  - Particulate Matter PM2.5
  - Particulate Matter PM10
  - Carbon Monoxide CO
  - Nitrogen Dioxide NO2
2. access and view statistics of the cache's usage:
  - Misses
  - Hits
  - Acesses

## **2.2 System architecture**

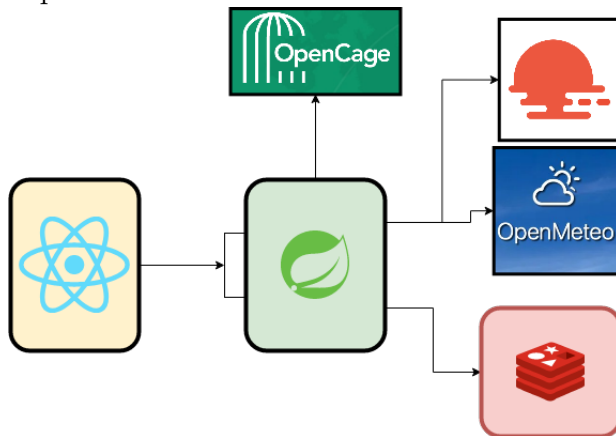
### **2.2.1 Architecture**

The application's Architecture is very basic and allows for easy maintenance and API evolution. The Controller handles communication between the WebApp and the backend. The Service Layer is composed of two distinct services. The Weather Service (responsible for providing the Air Quality for a certain region) utilizes a Geocoding API in order to transform an address or location string into coordinates. It communicates with two different Air Quality API's. In order to provide more constant availability and resilience, one acts as a backup when the main API is down. These API's provide Air Quality data for a certain location based on latitude and longitude. In order to reduce query times, a caching system using a key-value database was also introduced. The objects cached have a specific Time to Live that is easily configurable.



### 2.2.2 Technologies and API's Used

The very basic User Interface was constructed using ReactJS. It interacts with the Backend through a RestAPI. The backend was built using Spring Boot. For an in memory database I used Redis. The Geolocation Api chosen was Open Cage. The primary Air Quality API is the OpenWeather API. The backup Air Quality API is the OpenMeteo API.



## 2.3 API

The API has two endpoints:

1. **GET** /weather?local={*local*}

Query Params:

- **Required** - local: Address, Country or City .

Response Body:

```
{
  "coords": {
    "lat": 40.7275536,
    "lgn": -8.5209649
  },
  "stats": {
    "co": 150.0,
    "no2": 6.5,
    "pm25": 9.4,
    "pm10": 11.5
  },
  "location": "aveiro"
}
```

2. **GET** /cacheInfo

Response Body:

```
{
  "hits": 5,
  "misses": 10,
  "acesses": 15
}
```

## 3 Quality assurance

### 3.1 Logging

Logging is a crucial aspect of software development that plays a vital role in supporting production and debugging activities.

Logging was conducted using the slf4j Logger class as it provides an easy integration with the Spring Boot Application.

Each log message contains a timestamp, method invocation details, custom log messages, and other contextual data. Additionally, each log entry is assigned a logging level identifier, providing a way to categorize and prioritize log entries based on their significance.

I followed logging principles and best practices - such as: logging at the correct level, providing meaningful messages, etc - in order to provide meaningful logs.

## 3.2 Overall strategy for testing

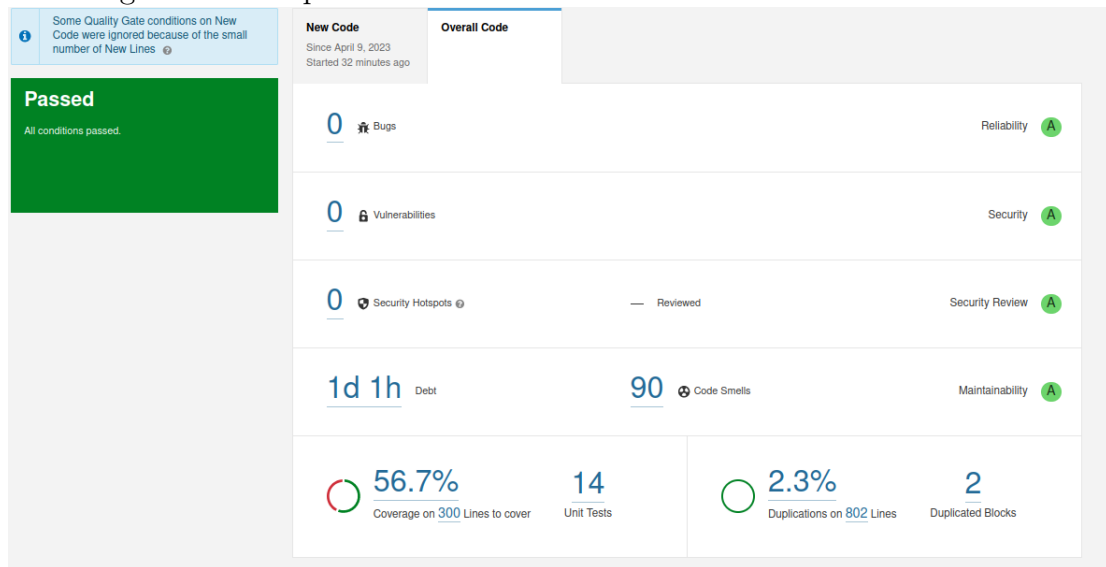
## 3.3 Unit and integration testing

implemented and

## 3.4 Functional testing

## 3.5 Code quality analysis

For static code analysis I used throughout the entirety of the project the SonarQube platform. It helped me identify potential bugs and vulnerabilities, improve my code quality, and find code smells and possible anti-patterns. In conjunction with static code analysis I also used the SonarQube platform in order to evaluate the code coverage of tests implemented.



## 3.6 Continuous integration pipeline

# 4 References & resources

GitRepository can be found at my github:

## 4.1 Reference materials