HW1: Mid-term assignment report

*João Silva, nmec: 88813*, v2020-03-27

[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 2](#_Toc36219515)

[2.3 API for developers 2](#_Toc36219516)

[3 Quality assurance 2](#_Toc36219517)

[3.1 Overall strategy for testing 2](#_Toc36219518)

[3.2 Unit and integration testing 2](#_Toc36219519)

[3.3 Functional testing 2](#_Toc36219520)

[3.4 Static code analysis 2](#_Toc36219521)

[3.5 Continuous integration pipeline [optional] 2](#_Toc36219522)

[4 References & resources 3](#_Toc36219523)

# Introduction

## Overview of the work

Este projecto tem por objectivo contruir uma multy layer web application em Spring Boot e equip-la com um conjunto de testes a vários niveis.

A aplicação AirQuality dispões de uma web page e uma Rest Api onde dando o nome de uma região (Ex: porto, lisbon, madrid, etc) é devolvida a qualidade do ar, assim como outros parametros disponiveis como Pm10, o3, etc.

## Limitations

A aplicação destina-se apenas às cidades mais conhecidas, dando o nome em Inglês.

# Product specification

## Functional scope and supported interactions

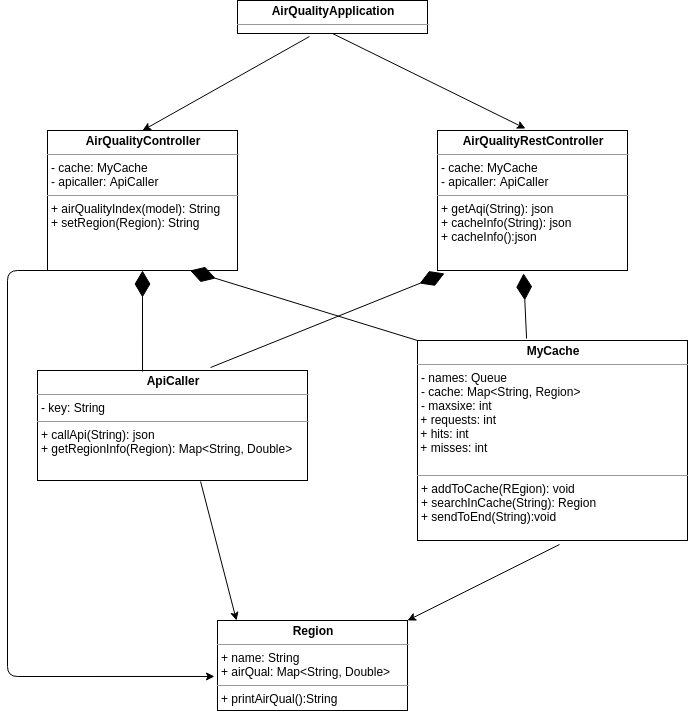
A aplicação será usada por qualquer pessoa que deseje informação sobre a qualidade do ar no momento sobre uma cidade (webSite). Assim como, para algum developer que deseje esta informação com apenas um pedido (Rest Api).

## System architecture

<briefly present the software architecture. Include diagrams.>

<explain the supporting data models/data structures, i.e., the entities of your problem>

<detail the specific technologies/frameworks that were used>



AirQualityController é responsável pelos pedidos da página Web. Quando é chamado retorna uma pagina HTML (index.html). Quando é preenchido o campo onde se insere o nome da Região é criado um Objecto Região e o método setRegion é invocado. Este método invoca a ApiCaller que se respinsabiliza por chamar a Api que tem a informação disponivel (<https://aqicn.org/api/pt/>), receber a sua resposta e processa-la da maneira desejada. O Resulado é um mapa com varias informações sobre a qualidade do ar.

O Controller vai ainda introduzir esta region na Cache para que caso seja feito um pedido igual não seja necessário contactar a Api, acelerando muito o processo.

A cache contém um record de Regiões (Mapa) e um Queue com os nomes das regiões presentes no momento da cache (esta queue serve para melhor gestão de que regiões devem ser retiradas ou mantidas na cache).

AirQualityRestController é responsável pelos pedidos feitos à Api do project comportando-se de forma semelhante ao AirQualityController, porém retorna um json e dispõe de estatisticas da cache (hits, miss, requests).

A cache trata-se de um Singleton por isso os dois controller colocam a informação no mesmo lugar.

## API for developers

<what services/resources can a developer obtain from your REST-API?>

GET - /cityAirQual/{name}

Obter uma das informações de qualidade do ar sobre uma região.

Status: 200/404

Data: aqi(AirQuality), o3, pm10, etc

GET - cacheInfo/{info}

Obter informações estatisticas da cache (hits, misses, requests)

Status - “OK”

Data: hits/misses/requests

GET - cacheInfo

Obter todas as informações estatisticas da cache (hits, misses, requests)

Status - 200/404

Data: hits, misses, requests

# Quality assurance

## Overall strategy for testing

TDD foi a principal estratégia, muitas vezes foram escritos primeiro os testes e depois a implementação.

Foram usadas várias ferramentas (Junit, Mockito, SpringBootMockmvc, REST-assured, selenium, Sonar Qube)

## Unit and integration testing

Os principais testes unitários foram sobre a cache.

Foram implementados usando Junit.

Ex:

Verificar que é possivel reaver uma região da cache:

@Test

void inCacheAfterSearch() {

Region r = new Region("region1");

HashMap<String, Double> hm = new HashMap<>();

hm.put("s",2.0);

r.setAirQual(hm);

this.cache.addToCache(r);

assertEquals(r, this.cache.searchInCache("region1"));

}

Verificar que a última Região é removida depois de a cache ter atingido o tamanho máximo (3, para facilitar os testes).

@Test

void checkRemoved(){

Region r1 = new Region("r1");

this.cache.addToCache( r1 );

this.cache.addToCache( new Region("r2"));

this.cache.addToCache( new Region("r3"));

this.cache.addToCache( new Region("r4"));

assertNull(this.cache.searchInCache("r1"));

}

@Test

void checkRemoved(){

Region r1 = new Region("r1");

this.cache.addToCache( r1 );

this.cache.addToCache( new Region("r2"));

this.cache.addToCache( new Region("r3"));

this.cache.addToCache( new Region("r4"));

assertNull(this.cache.searchInCache("r1"));

}

Depois de todas estas funcionalidades basicas estarem testadas, foram testadas classes como ApiCaller, com isolação chamada à api, ou seja, basicamente foram testadas coisas como a correta formatação do dicionário apartir de json etc.

No seguinte exemplo foi usada a anotação @Spy para fazer mock apenas to método callApi() que é o responsável por receber o json da API. Desta forma foi possivel testar o resto dos metodos de uma forma isolada do serviço externo.

@Test

void returnsCorrectDict() {

String stringToParse = "{\"status\":\"ok\",\"data\":{\"aqi\":26,\"idx\":8373,(…) bug\":{\"sync\":\"2020-04-08T23:20:52+09:00\"}}}";

JSONObject json = null;

try {

Json = new JSONObject(stringToParse);

}catch (JSONException err){

assertTrue(false);

}

when(apiCallerMock.callApi("porto")).thenReturn(json);

Region porto = new Region("porto");

Map<String, Double> answer = apiCallerMock.getRegionInfo(porto);

HashMap<String, Double> correct = new HashMap<>();

correct.put("aqi",26.0);

correct.put("p",1021.8);

correct.put("wg",5.0);

correct.put("o3",25.6);

correct.put("t",19.4);

correct.put("w",1.3);

correct.put("h",72.0);

assertEquals(correct, answer);

}

O AirQualityController foi testado sobretudo com Spring Boot MockMvc.

Ex:

Testar que a informação chega à web page quando é feito um bom pedido

@Test

public void goodRegion() throws Exception {

this.mockMvc.perform(get("/setRegion?name=porto&submit=0"))

.andDo(print())

.andExpect(status().isOk())

.andExpect(content().string(containsString("Air Quality:")));

}

Já para o AirQualityRestController foi usado REST-Assured.

Ex:

@Test

public void getNewRegionData() {

given().port(port).get("/cityAirQual/porto")

.then().statusCode(200).assertThat()

.body("data.aqi", allOf(greaterThanOrEqualTo((float)0.0), lessThanOrEqualTo((float)100.0)))

.body("status", equalTo("OK"));

}

@Test

public void getBadRegionData() {

given().port(port).get("/cityAirQual/portoasdasdaf").then().statusCode(404);

}

## Functional testing

Para Functional testing foi usado Selenium usando o chrome driver.

Foram testadas situações como aceder ao link correcto invoca a página correcta, introduzindo bons paramentros é dado o output correcto e em caso contrário é mostrada uma mensagem de erro

Ex:

@Test

public void goodCity() {

driver.get("http://localhost:" + port+'/');

driver.manage().window().setSize(new Dimension(1853, 895));

driver.findElement(By.id("name")).click();

driver.findElement(By.id("name")).sendKeys("porto");

driver.findElement(By.name("submit")).click();

String info = driver.findElement(By.cssSelector("textarea")).getText();

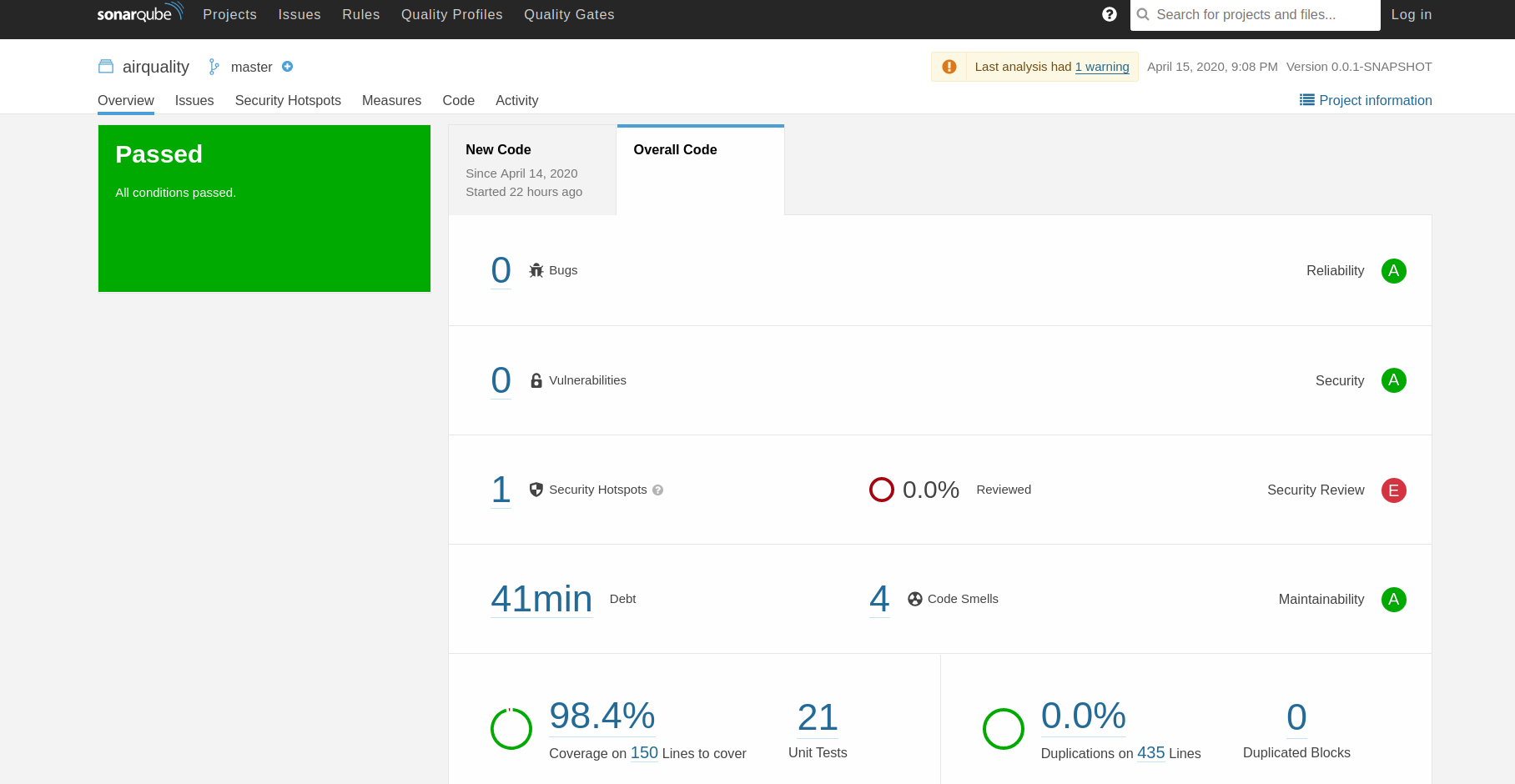
assertThat(info, containsString("Air Quality:"));

}

## Static code analysis

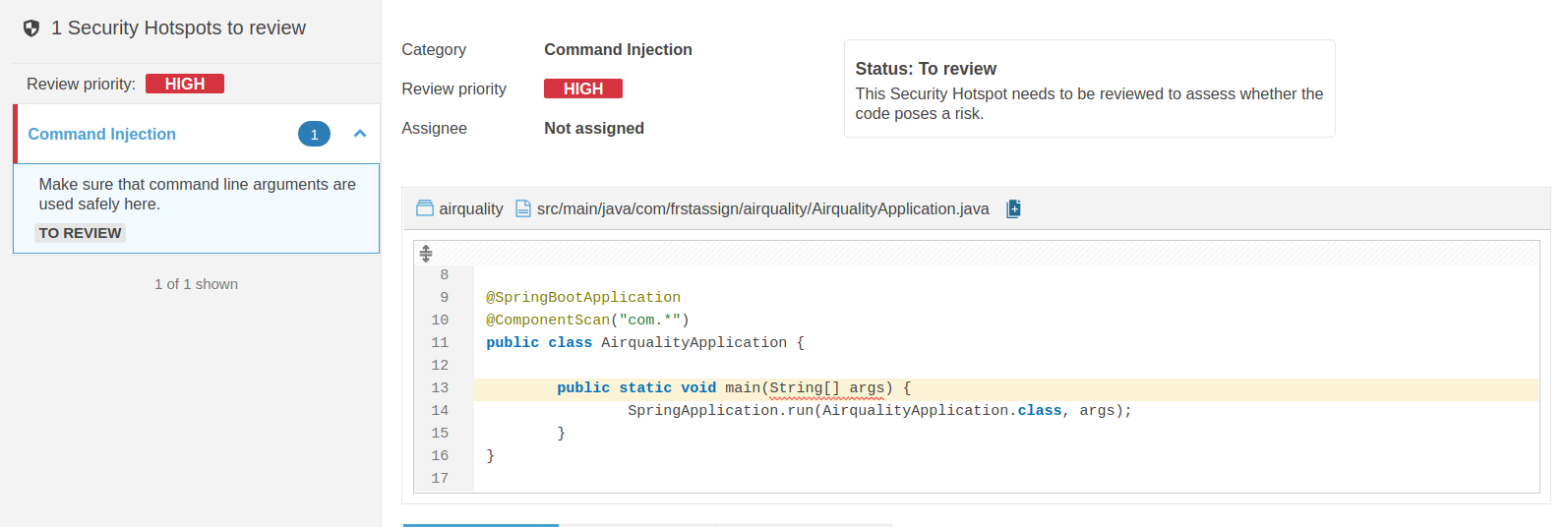
Para code analysis foi usado Sonar Qube com jacoco.

$mvn clean jacoco:prepare-agent install jacoco:report



Esta ferramenta foi muito útil. Realmente identificou muitos code smells que não me teria dado conta, até possiveis bugs e standards de nomes de metodos e tipos de retorno (Ex: retornar um Map<> e não um HashMap<>).

É também bastante útil para detetar possiveis vulnerabilidades.



## Continuous integration pipeline [optional]

Não foi desenvolvido

# References & resources

Project resources

* Git repository: <https://github.com/joaonpsilva/TQS_AirQuality>
* Video demo <https://github.com/joaonpsilva/TQS_AirQuality>

Reference materials

<https://spring.io/guides/gs/testing-web/>

<https://www.baeldung.com/rest-assured-tutorial>