HW1: Mid-term assignment report

Diogo Cunha [95278], v2021-05-14

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 application "AirQuality" is a minimalist website that serves to see the latest measurements of the air quality of a certain city. Everyone who access the website will be able to look at measurements of air quality like CO, NO2, OZONE, PM10, PM25 and SO2. There is also a Rest API available so that developers can the measurements of air quality and statistics about cache and the latest requests.

1.2 Current limitations

One of the limitations is the search for the place that you want to see the measures. To search by city name if the city is slightly misspelled the returned city might not be the city that the user wants because the API will fetch other cities by codes.

2 Product specification

2.1 Functional scope and supported interactions

The “AirQuality” application can be accessed in the web page and by the rest API.
We can consider 2 personas and 3 scenarios of usage:

For personas we have Diogo that is a developer and is developing an informative website for the city of Viseu and will achieve its objective with the help of the “AirQuality” Rest API.

The second persona name is Joao and is a person that has concerns about the pollution in his area of residence and will use the “AirQuality” website to see the latest measures.

Scenarios:

For a developer I can access the Rest API to get values of air quality from a certain city from name or from latitude and longitude.

For a developer I can access the Rest API to get some statics about the cache.

For a person that only wants to check the air quality of the area of residence is possible to go on the web and search for city name or geo coordinates.

User stories for Diogo:

As a developer, I want to get the latest measures of air quality of the city of Viseu, so that my website can present that information to the users.

As a developer, I want to see the statistics of the cache of the application, so that I can see what where the latest searches.

User stories for Joao:

As a resident of Viseu, I want to get the latest measures of air quality in my city, so that I can see if the values are normal.

2.2 System architecture

In regards of the system architecture my application is a spring boot application, programmed in Java and was used HTML, CSS for the webpage design and Thymeleaf to present to the user the data fetched from the API. It is also implemented a simple cache with time to live that stores the latest requests made by the users and some simple stats like hits and misses.

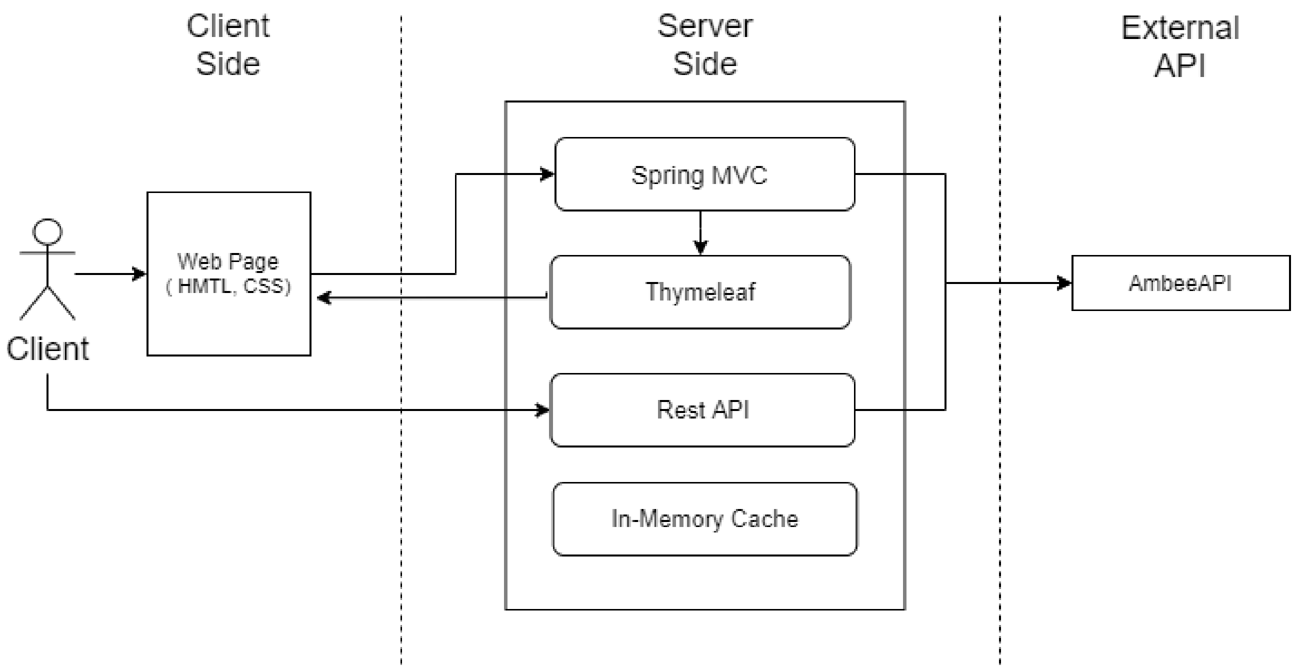
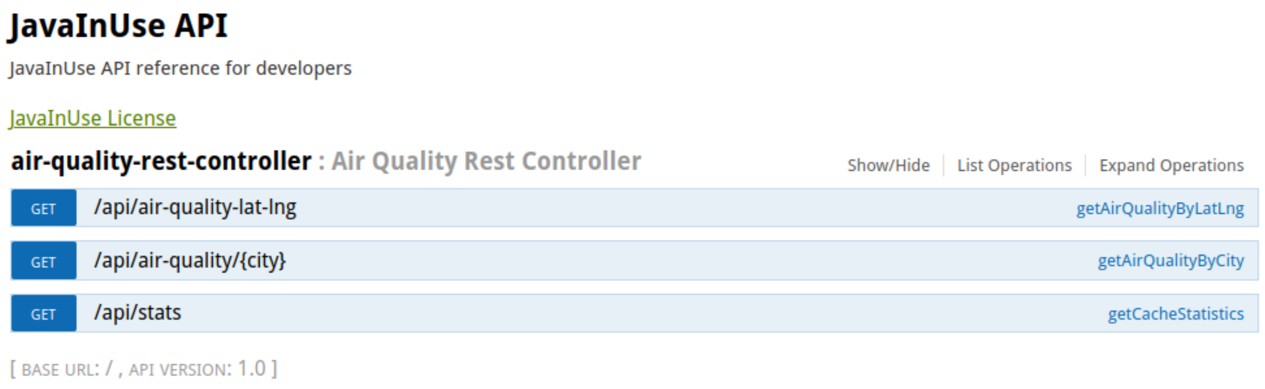


Figure 1 System Architecture

2.3 API for developers

A developer will have access to the latest values of air quality for a specific city and statistics about the API cache like last requests, time-to-live, hits and misses.
The air quality has a several values like CO, NO2, OZONE, PM10, PM25 and SO2



JavaInUse API

JavaInUse API reference for developers

JavaInUse License

air-quality-rest-controller : Air Quality Rest Controller — Show/Hide | List Operations | Expand Operations

GET	/api/air-quality-lat-lng	getAirQualityByLatLng
GET	/api/air-quality/{city}	getAirQualityByCity
GET	/api/stats	getCacheStatistics

[BASE URL: / , API VERSION: 1.0]

Figure 2 API Documentation

Image from http://localhost:8080/swagger-ui.html#/air-quality-rest-controller

3 Quality assurance

3.1 Overall strategy for testing

Unfortunately, I did not used any of the strategies that we have learned, I opted to implement parts of the application first and after that is implemented and working, I created some tests to have more assurances that it was working properly to different inputs and scenarios.

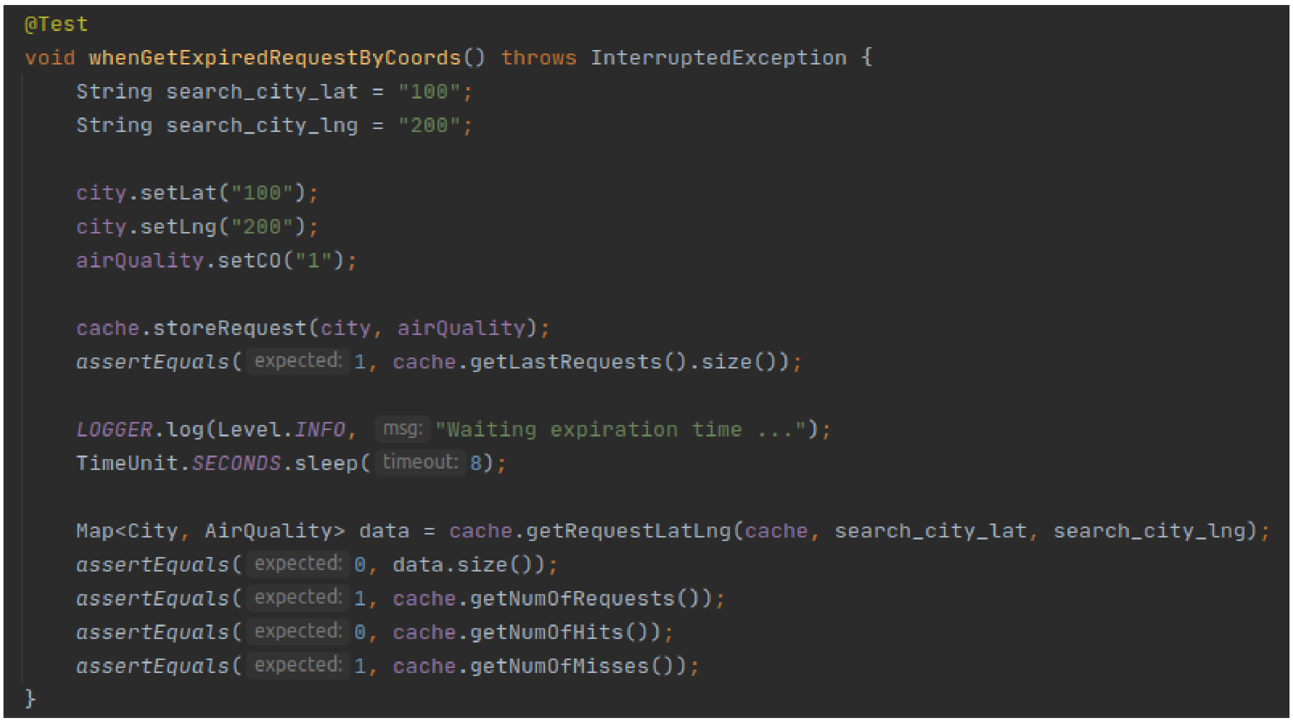
3.2 Unit and integration testing

In the unit tests that I implemented for the cache, I tested the 7 methods that I implemented in the cache (storeRequest, getRequest, removeExpiredRequest, hasExpired, hits, misses and requests increments).

Before each test I started by setting up a clean cache, assert that all values were at 0 and create a City and AirQuality object. And after each, tear down the cache.

Then I tested the storeRequest(), asserted if all the correct values increased and that the request was really stored. Tested all the increase methods alone. Tested the getRequest() by storing a request, getting that request and assert that it was correct, and then tested trying to get another request that wasn't stored and assert that nothing was returned.

Also tested the behavior of the program when I try to get an expired value from cache, to do that I inserted a value checked that the value was stored, then waited the time to live and check again asserting that the value was not there anymore.



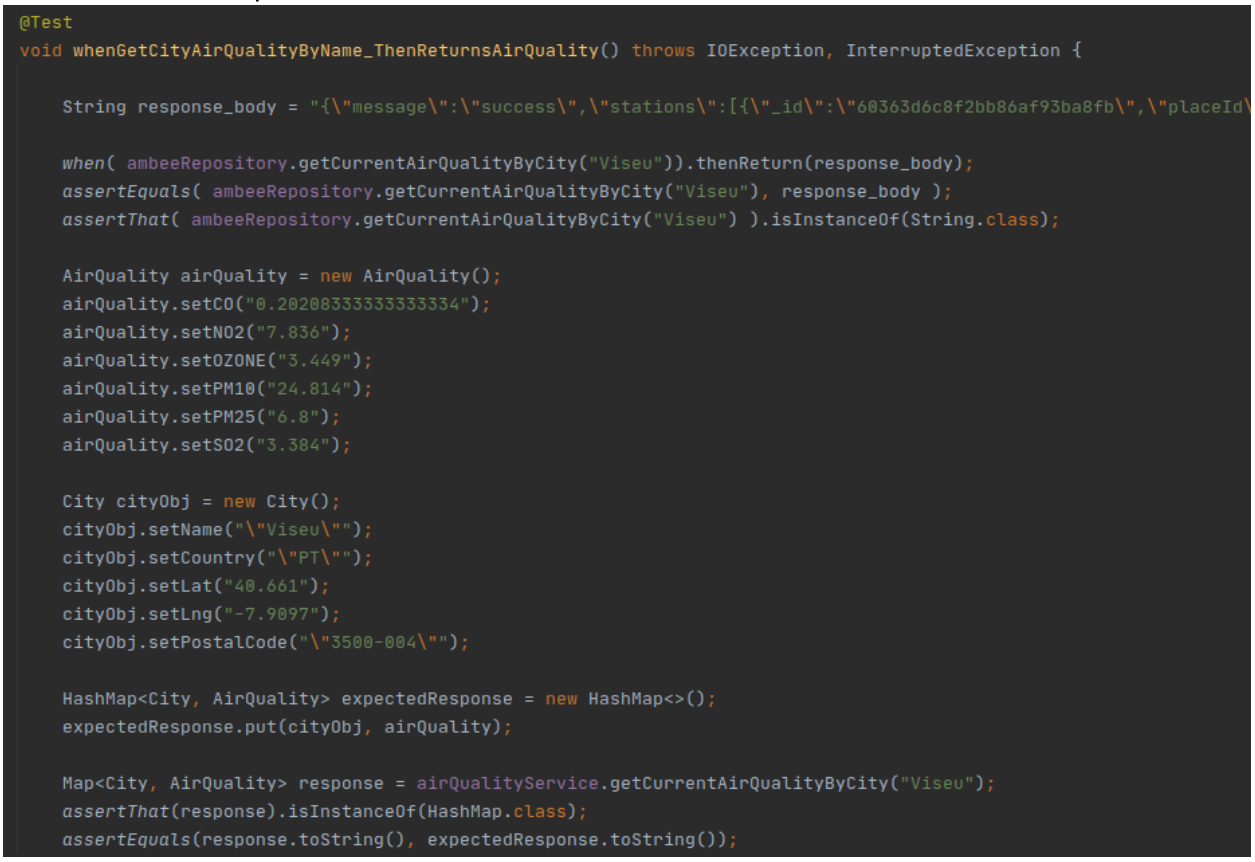
```java
@Test
void whenGetExpiredRequestByCoords() throws InterruptedException {
    String search_city_lat = "100";
    String search_city_lng = "200";

    city.setLat("100");
    city.setLng("200");
    airQuality.setCO("1");

    cache.storeRequest(city, airQuality);
    assertEquals( expected: 1, cache.getLastRequests().size());

    LOGGER.log(Level.INFO,  msg: "Waiting expiration time ...");
    TimeUnit.SECONDS.sleep( timeout: 8);

    Map<City, AirQuality> data = cache.getRequestLatLng(cache, search_city_lat, search_city_lng);
    assertEquals( expected: 0, data.size());
    assertEquals( expected: 1, cache.getNumOfRequests());
    assertEquals( expected: 0, cache.getNumOfHits());
    assertEquals( expected: 1, cache.getNumOfMisses());
}
```

Figure 3 Testing time to live from cache

To test the repository (external API request) I used the annotation @InjectMocks on the "ambeeRepository". And only asserted that the returned value from the repository is a String when the city name or coordinates are correct, and when the city name or coordinates were invalid, I asserted that the returned value is an error message.

For the service I used @Mock for the "AmbeeRepository" and @InjectMocks for the "AirQualityService" then I tested 2 different situations.

I tested when the city name was valid and when it was invalid. I started by arrange the expected response by the ambeeRepository and then calling the service to compare and assert both responses.



```java
@Test
void whenGetCityAirQualityByName_ThenReturnsAirQuality() throws IOException, InterruptedException {

    String response_body = "{\"message\":\"success\",\"stations\":[{\"_id\":\"60363d6c8f2bb86af93ba8fb\",\"placeId\

    when( ambeeRepository.getCurrentAirQualityByCity("Viseu")).thenReturn(response_body);
    assertEquals( ambeeRepository.getCurrentAirQualityByCity("Viseu"), response_body );
    assertThat( ambeeRepository.getCurrentAirQualityByCity("Viseu") ).isInstanceOf(String.class);

    AirQuality airQuality = new AirQuality();
    airQuality.setCO("0.20208333333333334");
    airQuality.setNO2("7.836");
    airQuality.setOZONE("3.449");
    airQuality.setPM10("24.814");
    airQuality.setPM25("6.8");
    airQuality.setSO2("3.384");

    City cityObj = new City();
    cityObj.setName("\"Viseu\"");
    cityObj.setCountry("\"PT\"");
    cityObj.setLat("40.661");
    cityObj.setLng("-7.9097");
    cityObj.setPostalCode("\"3500-004\"");

    HashMap<City, AirQuality> expectedResponse = new HashMap<>();
    expectedResponse.put(cityObj, airQuality);

    Map<City, AirQuality> response = airQualityService.getCurrentAirQualityByCity("Viseu");
    assertThat(response).isInstanceOf(HashMap.class);
    assertEquals(response.toString(), expectedResponse.toString());
```

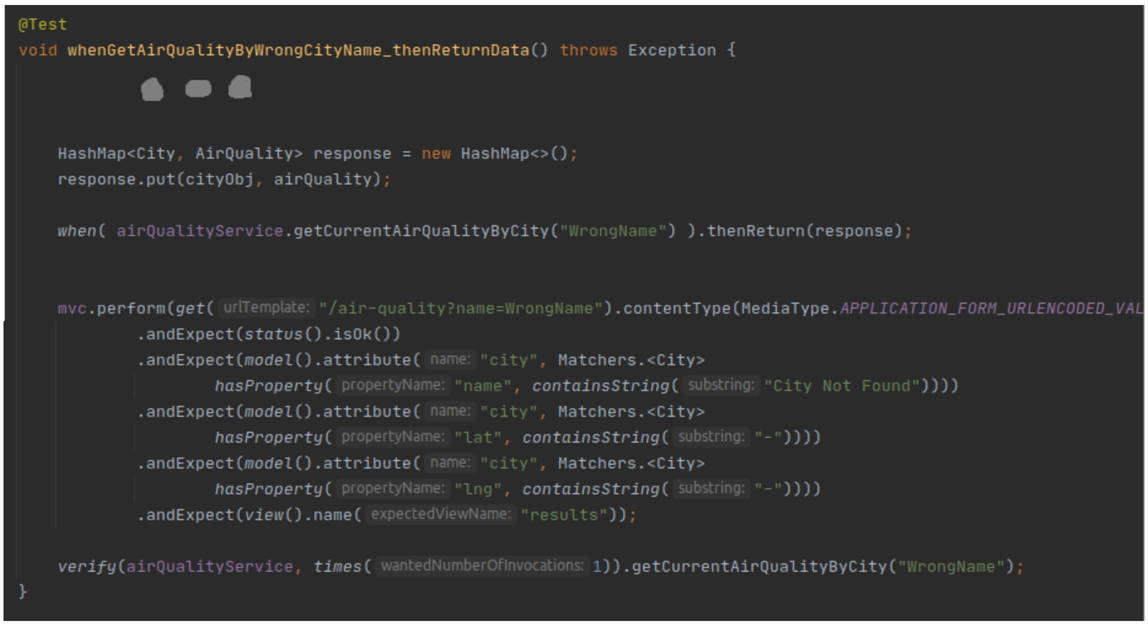
Figure 4 Testing service by getting city by name

I also created another class to test some converters used in the AirQualityService.

For the rest controllers and controllers, I mocked the service and used Mockmvc to make the get request to the controller and assert the response. I considered 2 scenarios, one where the get request was valid and another when the get request was not valid.

When the get request was valid, I asserted that the status code was "Ok", that values some values where in the response and that the returned template was the correct one.

When it was not, I asserted that the same things, but the name of the city should be "City Not Found".

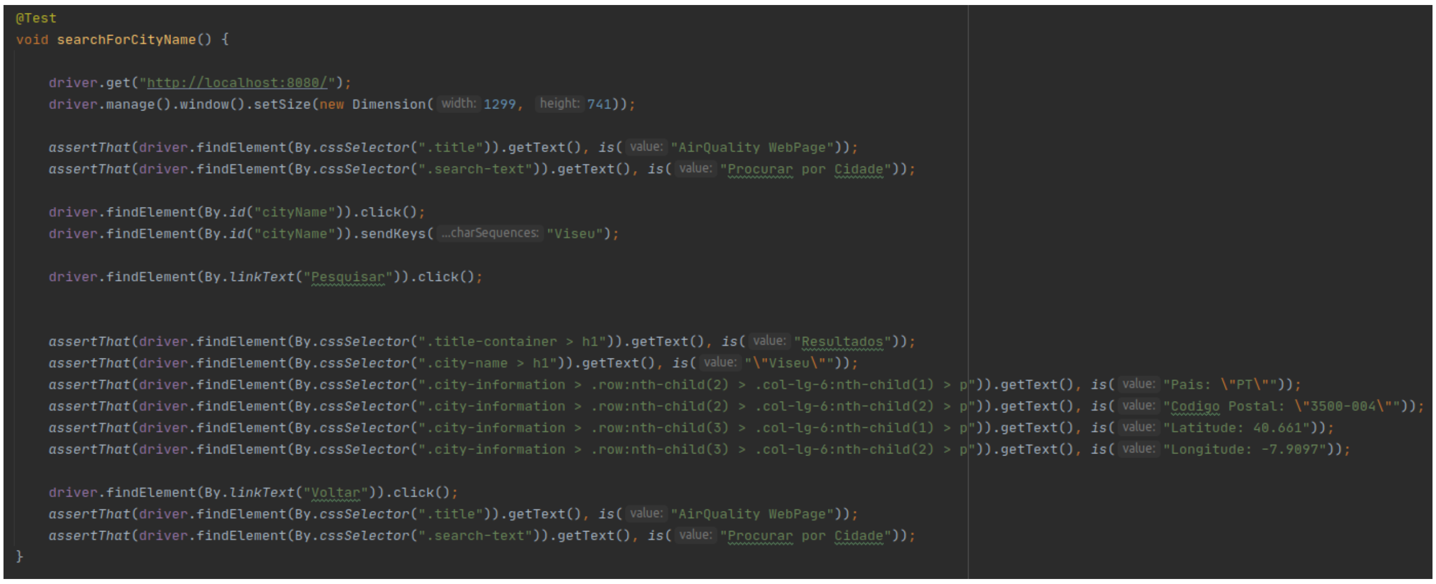


```
@Test
void whenGetAirQualityByWrongCityName_thenReturnData() throws Exception {
            [redacted]

    HashMap<City, AirQuality> response = new HashMap<>();
    response.put(cityObj, airQuality);

    when( airQualityService.getCurrentAirQualityByCity("WrongName") ).thenReturn(response);


    mvc.perform(get( urlTemplate: "/air-quality?name=WrongName").contentType(MediaType.APPLICATION_FORM_URLENCODED_VAL
            .andExpect(status().isOk())
            .andExpect(model().attribute( name: "city", Matchers.<City>
                    hasProperty( propertyName: "name", containsString( substring: "City Not Found"))))
            .andExpect(model().attribute( name: "city", Matchers.<City>
                    hasProperty( propertyName: "lat", containsString( substring: "-"))))
            .andExpect(model().attribute( name: "city", Matchers.<City>
                    hasProperty( propertyName: "lng", containsString( substring: "-"))))
            .andExpect(view().name( expectedViewName: "results"));

    verify(airQualityService, times( wantedNumberOfInvocations: 1)).getCurrentAirQualityByCity("WrongName");
}
```

Figure 5 Getting city by wrong name in controller

3.3 Functional testing

My functional tests (on the web interface) were made by the Selenium WebDriver and consists in opening the index page, verify that the page is well loaded, like assert the title, and then type a city name and press search.
After that confirm that the next page has the title, and city values like latitude and longitude right.
The second tests were doing the same thing but instead of writing a city name to search I write an invalid city name, and when searching I made sure that the page with the results said that the city was not found.

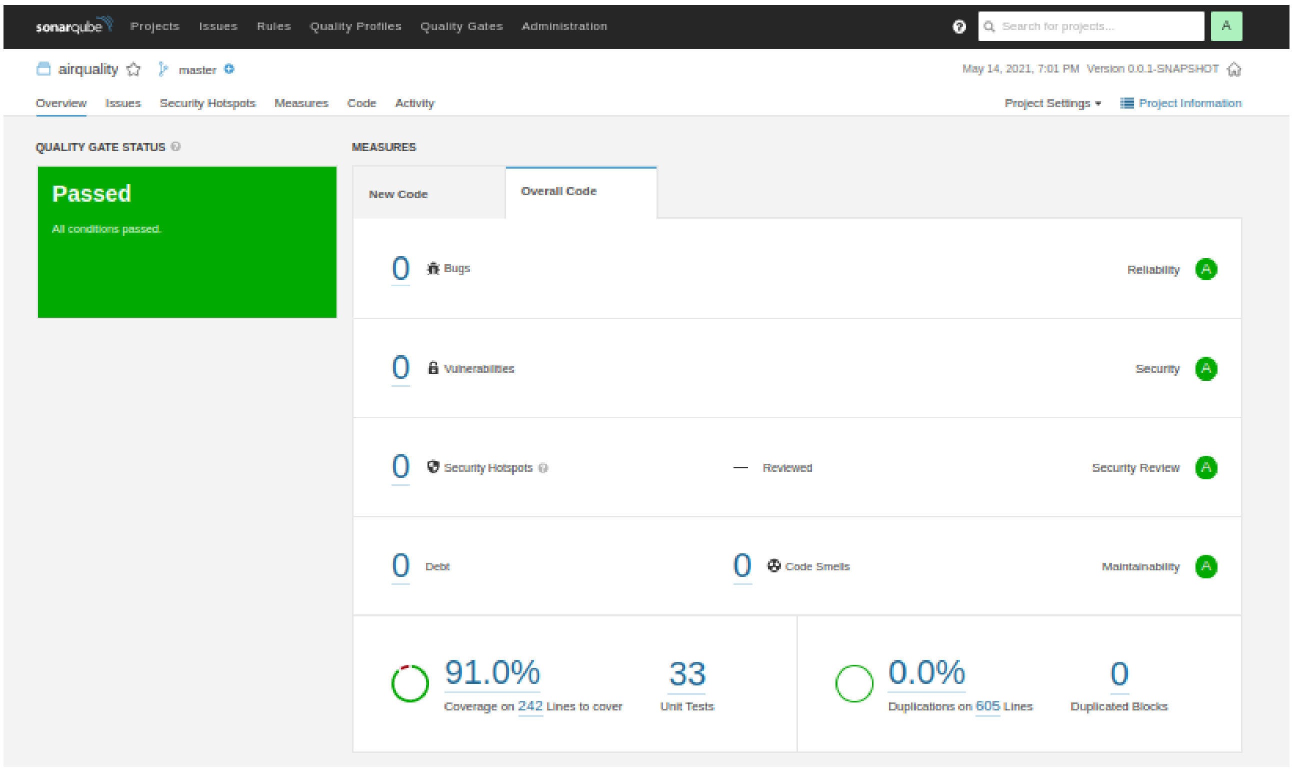


```
@Test
void searchForCityName() {

    driver.get("http://localhost:8080/");
    driver.manage().window().setSize(new Dimension( width: 1299,  height: 741));

    assertThat(driver.findElement(By.cssSelector(".title")).getText(), is( value: "AirQuality WebPage"));
    assertThat(driver.findElement(By.cssSelector(".search-text")).getText(), is( value: "Procurar por Cidade"));

    driver.findElement(By.id("cityName")).click();
    driver.findElement(By.id("cityName")).sendKeys( ...charSequences: "Viseu");

    driver.findElement(By.linkText("Pesquisar")).click();


    assertThat(driver.findElement(By.cssSelector(".title-container > h1")).getText(), is( value: "Resultados"));
    assertThat(driver.findElement(By.cssSelector(".city-name > h1")).getText(), is( value: "\"Viseu\""));
    assertThat(driver.findElement(By.cssSelector(".city-information > .row:nth-child(2) > .col-lg-6:nth-child(1) > p")).getText(), is( value: "Pais: \"PT\""));
    assertThat(driver.findElement(By.cssSelector(".city-information > .row:nth-child(2) > .col-lg-6:nth-child(2) > p")).getText(), is( value: "Codigo Postal: \"3500-004\""));
    assertThat(driver.findElement(By.cssSelector(".city-information > .row:nth-child(3) > .col-lg-6:nth-child(1) > p")).getText(), is( value: "Latitude: 40.661"));
    assertThat(driver.findElement(By.cssSelector(".city-information > .row:nth-child(3) > .col-lg-6:nth-child(2) > p")).getText(), is( value: "Longitude: -7.9097"));

    driver.findElement(By.linkText("Voltar")).click();
    assertThat(driver.findElement(By.cssSelector(".title")).getText(), is( value: "AirQuality WebPage"));
    assertThat(driver.findElement(By.cssSelector(".search-text")).getText(), is( value: "Procurar por Cidade"));
}
```

Figure 6 Search for City Name Selenium Test

3.4 Static code analysis

For static code analysis I used SonarQube and Sonar Lint Extension in the IntelliJ IDE. This helped me to make my quality code better during my implementation phase.

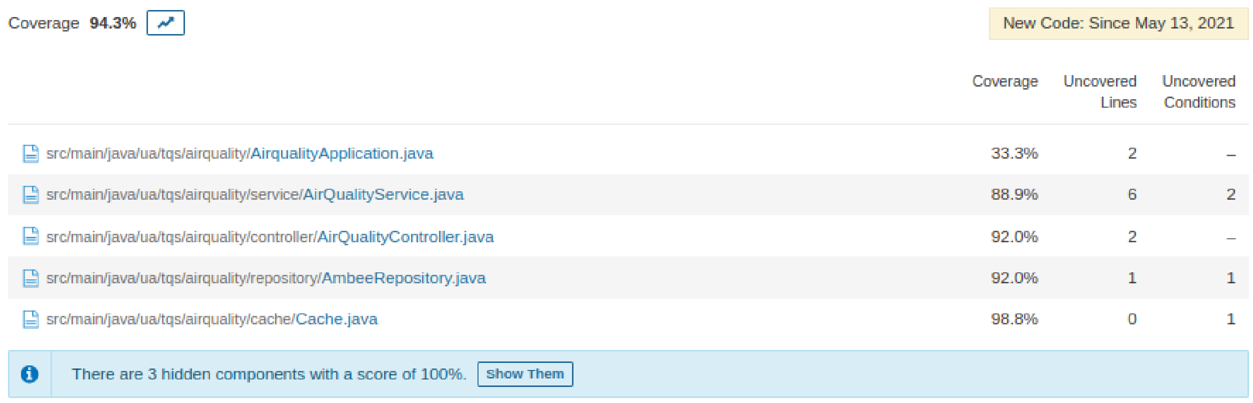


Passed
All conditions passed.

0 Bugs — Reliability A
0 Vulnerabilities — Security A
0 Security Hotspots — Reviewed — Security Review A
0 Debt — 0 Code Smells — Maintainability A
91.0% Coverage on 242 Lines to cover — 33 Unit Tests
0.0% Duplications on 605 Lines — 0 Duplicated Blocks

Figure 7 Sonar Qube Analysis

In the dashboard of the sonar qube we can see that there are no bugs, vulnerabilities, security hotspots and code smells and that the debt for the project is 0 because there is nothing to fix. In terms of coverage, we have 91% and 33-unit tests.



Coverage **94.3%**

New Code: Since May 13, 2021

	Coverage	Uncovered Lines	Uncovered Conditions
src/main/java/ua/tqs/airquality/AirqualityApplication.java	33.3%	2	–
src/main/java/ua/tqs/airquality/service/AirQualityService.java	88.9%	6	2
src/main/java/ua/tqs/airquality/controller/AirQualityController.java	92.0%	2	–
src/main/java/ua/tqs/airquality/repository/AmbeeRepository.java	92.0%	1	1
src/main/java/ua/tqs/airquality/cache/Cache.java	98.8%	0	1

There are 3 hidden components with a score of 100%. Show Them

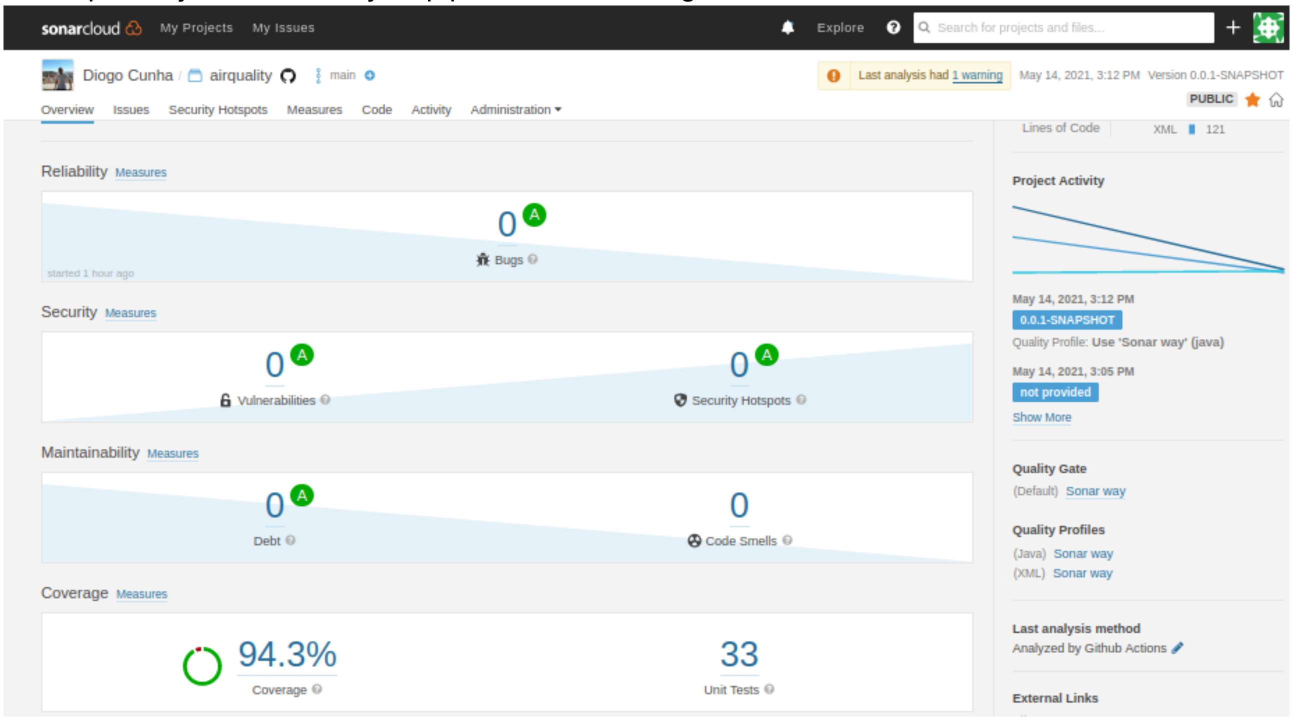
Figure 8 Code Coverage Sonar Qube

There is also 0 duplicated lines.
We can also see that the project passed all the conditions stated in the quality gate.

Some important lessons that static code analysis gave me was, not to log something that was introduced by the user, helped me identify one or two blocks of code duplicated and it reminded me to use a regular expression for the name of my class (SearchForCity_Selenium -> SearchForCity_SeleniumIT).

3.5 Continuous integration pipeline

I also implemented a CI pipeline using my Github repository actions and Sonar cloud.
The setup for this is an .yml file in the git repository, that is configured so that every push and pull to the main branch runs the sonar cloud analysis.
I only needed to add some properties to the project pom and create a new secret in the git hub repository. After that, my CI pipeline was working.



sonarcloud My Projects My Issues Explore Search for projects and files...

Diogo Cunha / airquality main

Last analysis had 1 warning May 14, 2021, 3:12 PM Version 0.0.1-SNAPSHOT PUBLIC

Overview Issues Security Hotspots Measures Code Activity Administration

Reliability Measures

0 A Bugs

started 1 hour ago

Security Measures

0 A Vulnerabilities

0 A Security Hotspots

Maintainability Measures

0 A Debt

0 Code Smells

Coverage Measures

94.3% Coverage

33 Unit Tests

Lines of Code XML 121

Project Activity

May 14, 2021, 3:12 PM
0.0.1-SNAPSHOT
Quality Profile: Use 'Sonar way' (java)

May 14, 2021, 3:05 PM
not provided

Show More

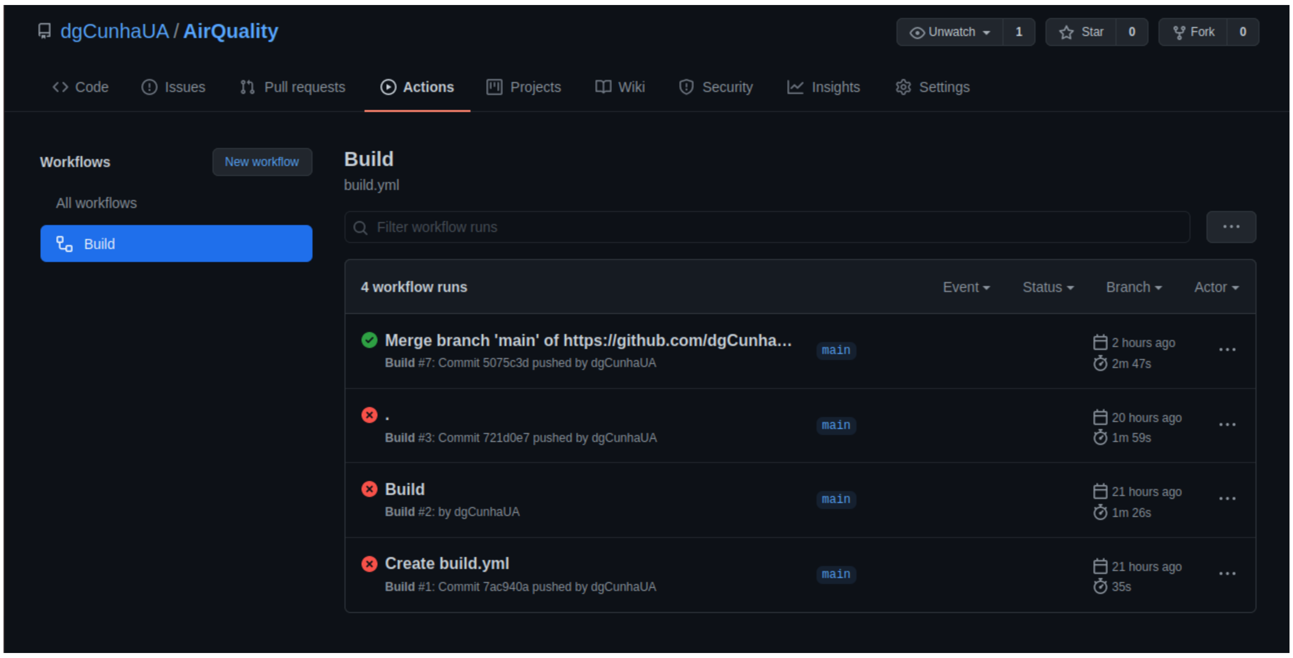
Quality Gate
(Default) Sonar way

Quality Profiles
(Java) Sonar way
(XML) Sonar way

Last analysis method
Analyzed by Github Actions

External Links

Figure 9 Sonar Cloud Dashboard



Build
build.yml

4 workflow runs

Run	Branch	When	Duration
Merge branch 'main' of https://github.com/dgCunha... Build #7: Commit 5075c3d pushed by dgCunhaUA	main	2 hours ago	2m 47s
. Build #3: Commit 721d0e7 pushed by dgCunhaUA	main	20 hours ago	1m 59s
Build Build #2: by dgCunhaUA	main	21 hours ago	1m 26s
Create build.yml Build #1: Commit 7ac940a pushed by dgCunhaUA	main	21 hours ago	35s

Figure 10 GitHub Actions Workflow

4 References & resources

Project resources

- GitHub Repository: https://github.com/dgCunhaUA/AirQuality
- Video demo: https://www.youtube.com/watch?v=8cTzl9WckxI&ab_channel=DiogoCunha
- QA dashboard: https://sonarcloud.io/dashboard?id=dgCunhaUA_AirQuality

Reference materials

https://www.getambee.com/
https://www.javainuse.com/spring/boot_swagger
http://localhost:8080/swagger-ui.html#/air-quality-rest-controller
https://sonarcloud.io/