HW1: Mid-term assignment report

João Teixeira Soares [93078], v2021-05-14

1 Introduction

1.1 Overview of the work

GetAirQuality is a Web Application design with the goal of getting different Air Quality parameters from Cities. It uses an external API to fetch data and then through a web interface provides the Information to the user and then caches it. The Application also has it own API running to make possible requests not available through the web interface and get the information in a simpler format to be used by others.

1.2 Current limitations

A limitation of the Application is that the user is only able to obtain the Air Quality information from the cities that are in the selection list. As such, it would be necessary to create a search bar that allows the user to search for the city he wants, considering that the name of the city entered must be similar to obtain the desired result. Another limitation of the Application is the fact that it only presents the current data on Air Quality.

2 Product specification

2.1 Functional scope and supported interactions

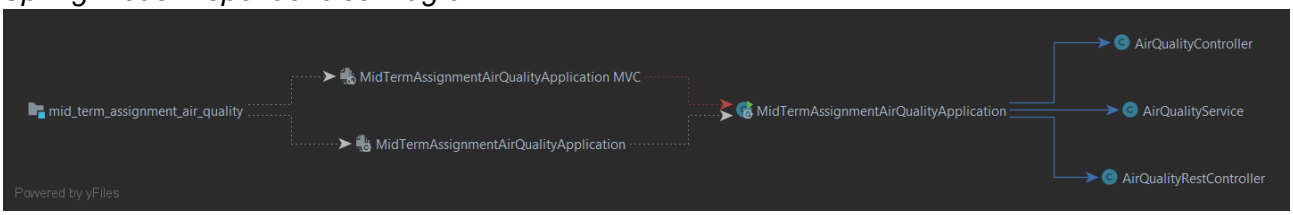
A user can interact with the Application using the web interface and using the running API to get the Air Quality information of specific location.
Scenarios:

- Ana a teacher from a primary school in the city of Lisbon has some health problems so she wants to know the state of the air quality to check if it is reaching harmful levels. To that Ana accesses GetAirQuality, select the option with city of Lisbon and clicks on the button "Get Air Quality". After that, the Air Quality parameters will appear, among others the Air Quality Index and this way Ana can infer if the air is harmful to her health.
- Rui is a freelancer developer from Aveiro who wants to develop a Mobile Application with the intent of providing air quality data from some Portuguese cities and for that he needs a reliable data source. To do this, he accesses the GetAirQuality API and can get the Air Quality information for a specific city or coordinates for his Mobile Application.

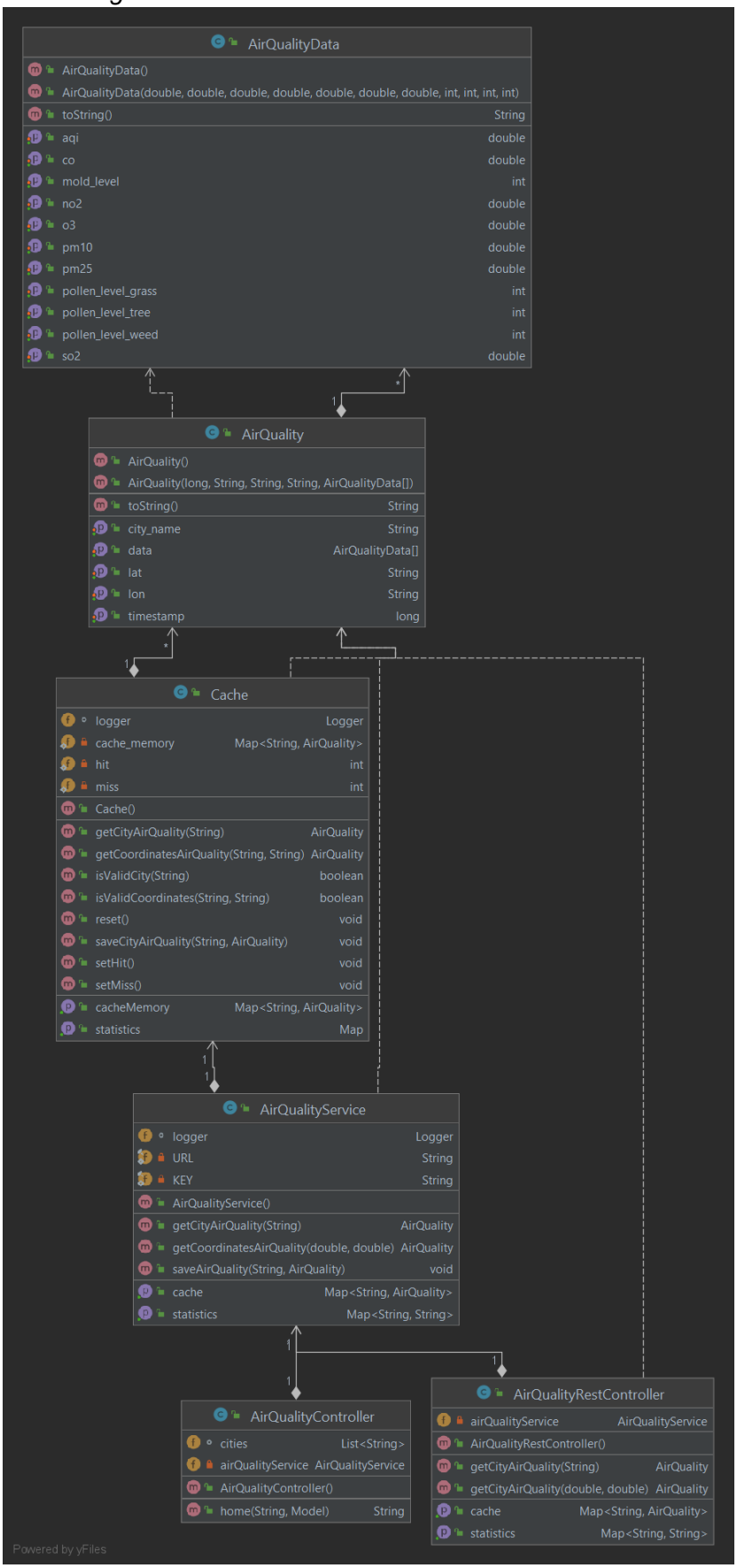
2.2 System architecture

The Application services/backend based on Spring Boot.
For the External API it was used the Air Quality API (Current) from Weatherbit.io.
The Web Interface (presentation) layer was implemented with HTML using the templating system thymeleaf and JavaScript.
Json-Simple used for Json handling.
Time to live policy: 10 minutes (600 000 milliseconds).

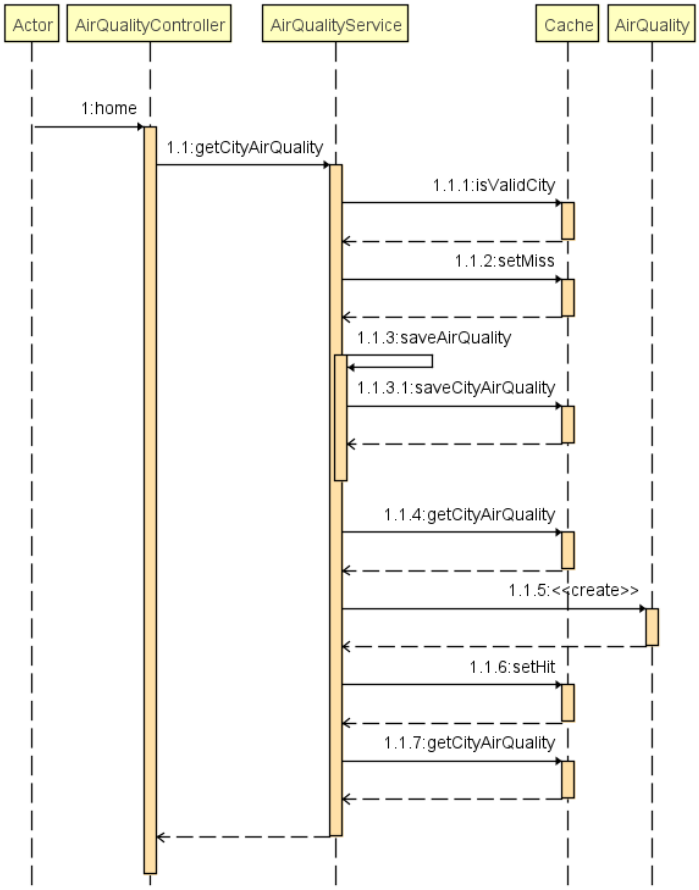
Spring Model Dependencies Diagram:



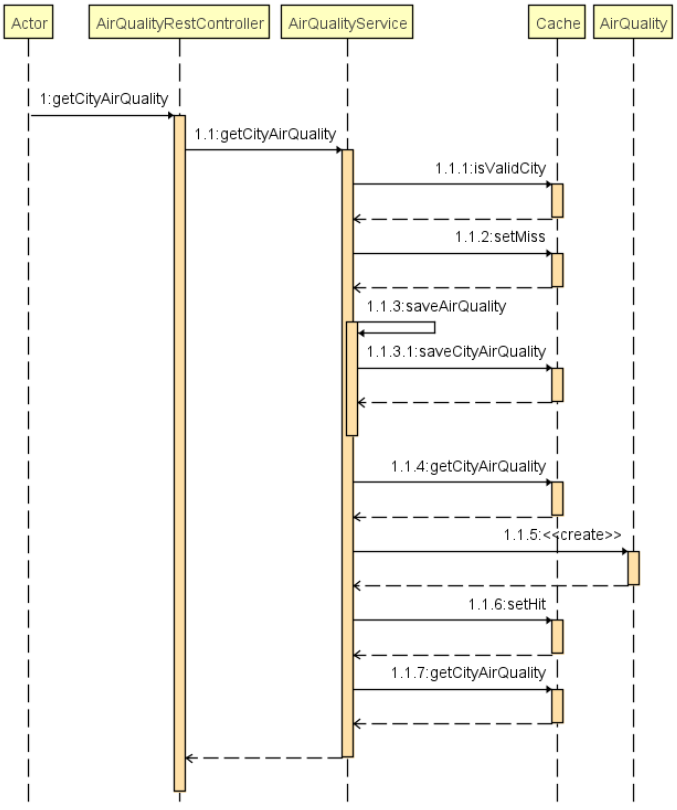
Class Diagram:



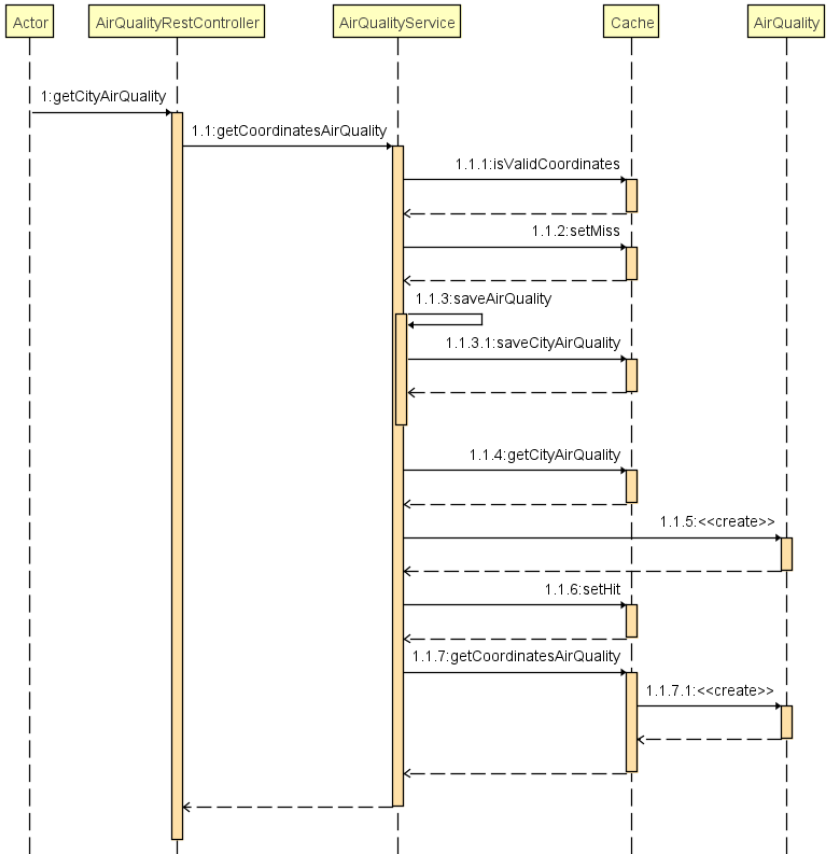
Controller Sequence Diagram:



Rest Controller GetCityAirquality Sequence Diagram:



Rest Controller GetCoordinatesAirquality Sequence Diagram:



2.3 API for developers

API Documentation with endpoints and values present in file named Api Documentation.md

Snippet from file:

API Documentation

Problem Domain

Get Current Air Quality By City

GET `/api/airquality/city/{city}`

Example Value


```
{
  "timestamp": 1620861789010,
  "city_name":"Porto",
  "lat":"42.16737",
  "lon":"-6.89934",
  "data":
  [{
      "aqi":41.0,
      "o3":88.4235,
      "co":282.049,
      "so2":0.447966,
      "no2":1.18214,
      "pm25":0.933011,
      "pm10":3.62781,
      "pollen_level_tree":1,
      "pollen_level_grass":1,
      "pollen_level_weed":1,
      "mold_level":1
    }]
}
```


- When Not Valid: `{city}`


```
{
  "timestamp":0,
  "city_name":"error",
  "lat":"",
  "lon":"",
  "data":[]
}
```

3 Quality assurance

3.1 Overall strategy for testing

The overall test Development Strategy was a mix of TDD and BDD. For the Web Interface that the normal user interacts, decided to go for a more human readable approach using Cucumber with Selenium IDE and BDD. For the backend of the application and REST API, decided to go with TDD using WebMvcAproach, Rest-Assured, Unit Testing, MockMvc, SpringBootTest and Mocks with SonarCloud helping in the test's development.

3.2 Unit and integration testing

Testes were developed for the entities, controllers, and service, covering the most important methods. *AirQualityControllerTest* used a Rest-Assured and WebMvcAproach with MockMvc while the *AirQualityRestControllerTest* just used WebMvcAproach with MockMvc. *AirQualityServiceTest* used Mocks for isolation and *CacheTest* used Unit Test approach with SpringBootTest.
It was also developed a AllComponentsTest using Unit Test to validate all components and a WebInterfaceSeleniumTest without Cucumber.

For the Complete Tests check the repository directory!

AirQualityControllerTest Code snippet:

```
@WebMvcTest(AirQualityController.class)
 class AirQualityControllerTest {

    @Autowired
    MockMvc mockMvc;

    @MockBean
    AirQualityService airQualityService;

    @BeforeEach
    void setUp(){
        RestAssuredMockMvc.mockMvc(mockMvc);
    }

    @Test
     void home__test() throws Exception {
        RestAssuredMockMvc.given().auth().none().when().get("").then()
                .log().all().statusCode(200)
                .contentType(ContentType.HTML)
                .body("html.head.title",equalTo("Air Quality"));
    }
}
```

AirQualityRestControllerTest Code snippet:

```
@WebMvcTest
 class AirQualityRestControllerTest {
    @Autowired
    MockMvc mvc;

    @MockBean
    AirQualityService airQualityService;

    @Test
     void getAirQualityCity_Test() throws Exception{
        given(airQualityService.getCityAirQuality(Mockito.anyString())).willReturn(new AirQuality(new Timestamp(System.currentTimeMillis()).getTime(),"city","2","1",
        mvc.perform(MockMvcRequestBuilders.get("/api/airquality/city/city"))
                .andExpect(status().isOk())
                .andExpect(jsonPath("city_name").value("city"))
                .andExpect(jsonPath("data[0].pm10").value(3.0));
        verify(airQualityService, times(1)).getCityAirQuality(Mockito.anyString());
    }

    @Test
     void getAirQualityCoordinates_Test() throws Exception{
        given(airQualityService.getCoordinatesAirQuality(Mockito.anyDouble(),Mockito.anyDouble())).willReturn(new AirQuality(new Timestamp(System.currentTimeMillis()
        mvc.perform(MockMvcRequestBuilders.get("/api/airquality/coordinates/"+41.149612+","+21.01))
                .andExpect(status().isOk())
                .andExpect(jsonPath("lat").value("41.14961"))
                .andExpect(jsonPath("lon").value("-8.61099"))
                .andExpect(jsonPath("data[0].pm10").value(3.0));
        verify(airQualityService, times(1)).getCoordinatesAirQuality(Mockito.anyDouble(),Mockito.anyDouble());
    }
```

AirQualityServiceTest Code snippet:

```
@ExtendWith(MockitoExtension.class)
 class AirQualityServiceTest {

    @Mock(lenient = true)
    private Cache cache;

    @InjectMocks
    private AirQualityService airQualityService;

    @BeforeEach
    public void setUp() {
        Map<String, AirQuality> cache_memory = new HashMap<>();

        cache_memory.put("Porto", new AirQuality(new Timestamp(System.currentTimeMillis()).getTime(),"porto","42.16737","-6.89934", new AirQualityData[]{new AirQuali
        cache_memory.put("Coimbra", new AirQuality(new Timestamp(System.currentTimeMillis()).getTime(),"coimbra","40.20564","-8.41955", new AirQualityData[]{new AirQ
        AirQuality aveiro =  new AirQuality(new Timestamp(System.currentTimeMillis()).getTime(),"Aveiro","40.64427","-8.64554", new AirQualityData[]{new AirQualityDa
        cache_memory.put("Aveiro",aveiro);

        HashMap<String, String> statistics = new HashMap<>();
        statistics.put("hit", "5");
        statistics.put("miss", "3");
        statistics.put("citiesAirInfoInCache", cache_memory.keySet().toString());

        Mockito.when(cache.getCacheMemory()).thenReturn(cache_memory);
        Mockito.when(cache.getStatistics()).thenReturn(statistics);
        Mockito.when(cache.getCityAirQuality("Aveiro")).thenReturn(aveiro);
        Mockito.when(cache.getCoordinatesAirQuality("40.64427","-8.64554")).thenReturn(aveiro);
        Mockito.when(cache.getCityAirQuality("xxxx")).thenReturn(new AirQuality(0,"error","","", new AirQualityData[]{}));
        Mockito.when(cache.getCoordinatesAirQuality("200","200")).thenReturn(new AirQuality(0,"error","","", new AirQualityData[]{}));

    }
    @Test
```

CacheTest Code snippet:

```
@SpringBootTest
 class CacheTest {

    protected Cache cache;

    @BeforeEach
    public void setUp() {
        cache = new Cache();
    }
    @AfterEach
    public void clearCache(){
        cache.reset();
    }

    @Test
     void getCityAirQuality_Test() {
        AirQuality aveiro_airquality = new AirQuality(new Timestamp(System.currentTimeMillis()).getTime(),"Aveiro","40.64427","-8.64554", new AirQualityData[]{new Ai
        cache.saveCityAirQuality("Aveiro", aveiro_airquality);
        AirQuality returned = cache.getCityAirQuality("Aveiro");
        assertThat(returned.getCity_name()).isEqualTo(aveiro_airquality.getCity_name());
        assertThat(returned.getLat()).isEqualTo(aveiro_airquality.getLat());
        assertThat(returned.getLon()).isEqualTo(aveiro_airquality.getLon());
        assertThat(returned.getData()[0]).isEqualTo(aveiro_airquality.getData()[0]);
    }

    @Test
     void getCoordinatesAirQuality_Test() {
        AirQuality aveiro_airquality = new AirQuality(new Timestamp(System.currentTimeMillis()).getTime(),"Aveiro","40.64427","-8.64554", new AirQualityData[]{new Ai
        cache.saveCityAirQuality(aveiro_airquality.getCity_name(), aveiro_airquality);
```

3.3 Functional testing

Functional Testing for the Web Interface was implemented with Selenium IDE and Cucumber. One Test just using Selenium IDE, (*WebInterfaceSeleniumTest*), and another, (*CucumberTest* with *WebDriverSteps*) using Cucumber and Selenium IDE. Both tests have a similar formula:

- Client selects city of Aveiro, clicks on the button to Get the Air Quality and then it checks if the title starts with Aveiro.

Screenshots:

- *WebInterfaceSeleniumTest*

```
public class WebInterfaceSeleniumTest {
    private WebDriver driver;
    private Map<String, Object> vars;
    JavascriptExecutor js;

    @After
    public void tearDown() {
        driver.quit();
    }
    @Test
    public void WebInterfaceTest() {
        driver = new ChromeDriver();
        js = (JavascriptExecutor) driver;
        vars = new HashMap<String, Object>();
        driver.get("http://localhost:8080/");
        driver.manage().window().setSize(new Dimension(1376, 744));
        driver.findElement(By.id("type_form")).click();
        {
            WebElement dropdown = driver.findElement(By.id("type_form"));
            dropdown.findElement(By.xpath("//option[. = 'Aveiro']")).click();
        }
        driver.findElement(By.id("type_form")).click();
        driver.findElement(By.cssSelector(".btn-primary")).click();
        assertThat(driver.getTitle(),
                containsString("Aveiro"));
        driver.close();
    }
}
```

- *CucumberTest*

```
@Cucumber
public class CucumberTest {
}
```

- *WebDriverSteps*

```java
public class WebDriverSteps {
    private WebDriver driver;

    @When("I navigate to {string}")
    public void iNavigateTo(String arg0){
        driver = new ChromeDriver();
        driver.get("http://localhost:8080/");
        driver.manage().window().setSize(new Dimension(1552, 840));
    }

    @And("I select {string}")
    public void iSelect(String arg0) {
        driver.findElement(By.id("type_form")).click();
        {
            WebElement dropdown = driver.findElement(By.id("type_form"));
            dropdown.findElement(By.xpath("//option[. = 'Aveiro']")).click();
        }
    }

    @And("I click in  {string}")
    public void iClickIn(String arg0) {
        driver.findElement(By.id("type_form")).click();
        driver.findElement(By.cssSelector(".btn-primary")).click();

    }

    @Then("the page title should start with {string}")
    public void thePageTitleShouldStartWith(String arg0) {
        assertThat(driver.getTitle(),
                containsString("Aveiro"));
        driver.close();
    }

    @After()
    public void closeBrowser() {

        driver.quit();
    }
}
```

- *AirQuality.feature*

```
Feature: AirQuality from City Search
  To allow a client to find the Air Quality from Portugal Cities.

  Scenario: Get Air Quality from City
    When I navigate to "http://localhost:8080/"
    And I select "Aveiro"
    And I click in  "Get Air Quality"
    Then the page title should start with "Aveiro"
```

3.4 Static code analysis

To static code analysis it was used SonarCloud integrated with JaCoCo to generate reports about the code coverage. Despite some problems mentioned later in the report, Sonar Cloud facilitated the development of tests and the code Reliability. Another tool used to analyze the code was SonarLint to quickly identify and fix issues during development.
For example, the critical code smell, *Instance methods should not write to "static" fields*, was very difficult to spot and eliminate. This code smell had an easy solution but not very well documented, but through some “Google Searches” found out that just needed to add synchronized to the methods related to the Cache Statistics.

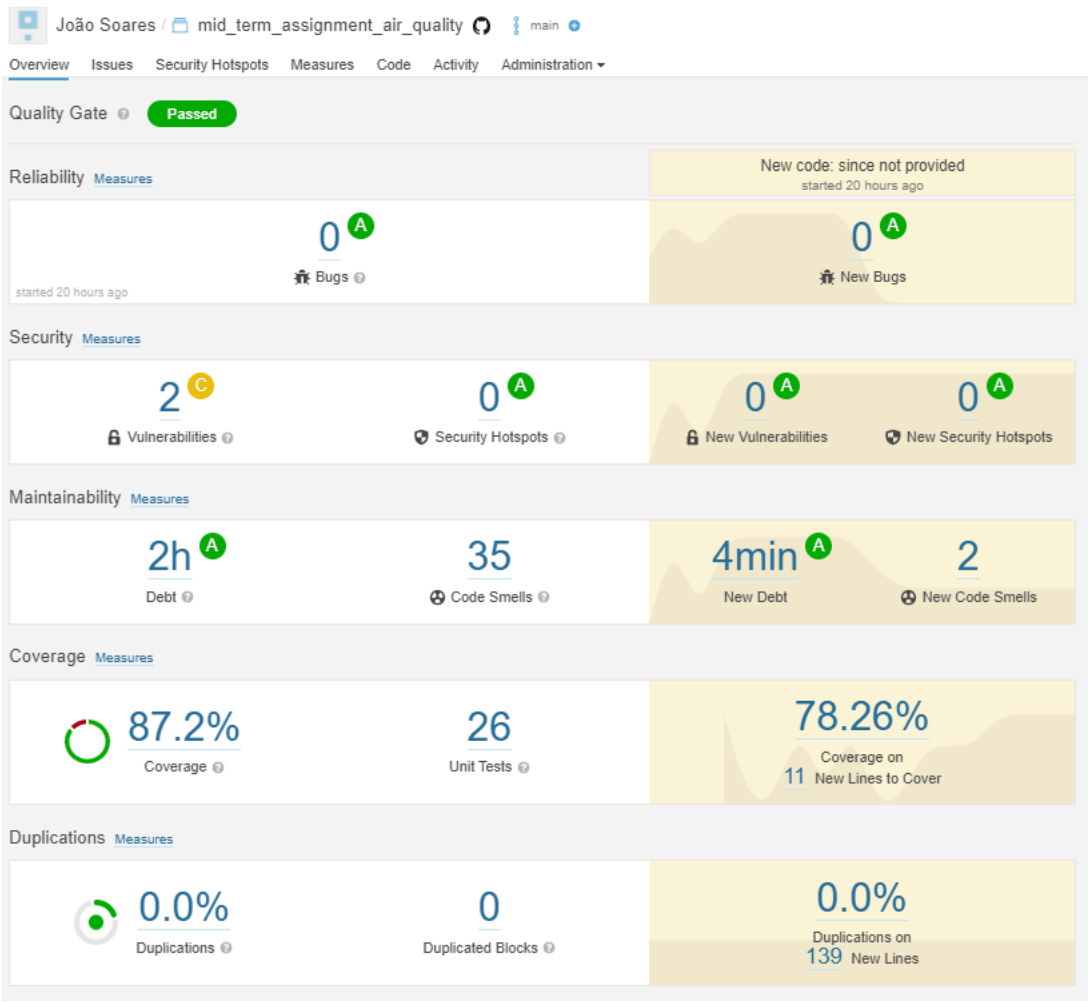
Quality Gate used in final builds to outrun some bugs (remaining used the Sonar Way Quality Gate):

Conditions on Overall Code

Conditions on Overall Code apply to long-lived branches only.

Metric	Operator	Value	Edit	Delete
Coverage	is less than	80.0%		
Duplicated Lines (%)	is greater than	3.0%		
Maintainability Rating	is worse than	A		
Reliability Rating	is worse than	A		
Security Hotspots Reviewed	is less than	100%		
Security Rating	is worse than	D		

SonarCloud Dashboard (last code update):



Coverage of 87,2 % is a very good value considering that all major lines in the code are covered.
Reduction of Code Smells (93 to 35).
2 Vulnerabilities (1 minor, 1 major) that for the goals of this project are not serious so can pass in the implemented Quality Gate.

3.5 Continuous integration pipeline [optional]

Implemented CI pipeline Testing with GitLab:

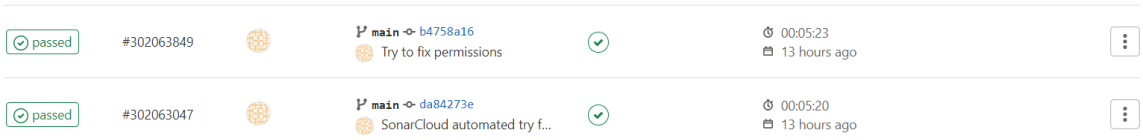
- *Setup:*

.gitlab-ci.yml 67 Bytes

```yaml
include:
  - template: Code-Quality.gitlab-ci.yml

stages:
  - test
```

- *Illustrative Screenshot:*



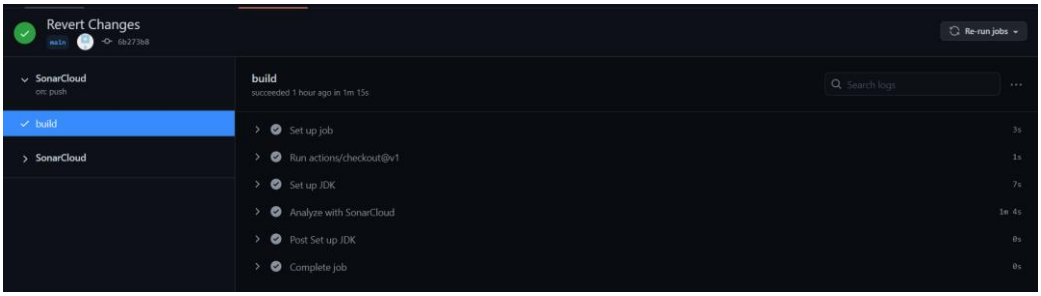
Implemented CI pipeline with Sonar Cloud analysis using Github Actions and locally:

- *Setup:*
  - *sonay.yml:* https://github.com/JoaoTS20/TQS_Midterm_Assignment/blob/main/.github/workflows/sonar.yml
  - *locally:*

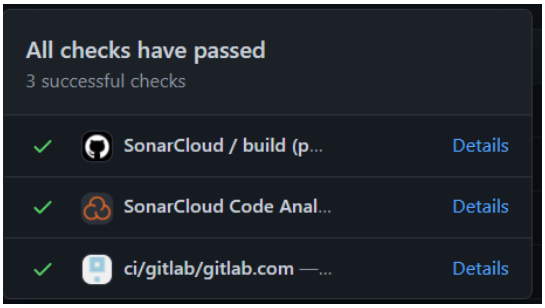
```
mvn verify sonar:sonar  -Dsonar.projectKey=JoaoTS20_TQS_Midterm_Assignment  -Dsonar.organization=joaots20  -Dsonar.host.url=https://sonarcloud.io  -Dsonar.login=key
```

- *Illustrative Screenshots:*
  - *Workflow:*

  - *Example Run:*



Checks:



Note: Due to some configuration problems, WorkFlow may have some inconsistencies. An example of a problem was the configuration of SonarCloud with GitHub Actions and the need to disable the analysis of tests due to the Tests of the Web Interface needing the application running or deployed on a public server. This problem has however been solved in a not very conventional but functional way by executing the commands locally with the application running to obtain the analysis of test in SonarCloud.

4 References & resources

Project resources

- Video demo
  https://github.com/JoaoTS20/TQS_Midterm_Assignment/blob/main/Demo.mp4
- Git Repository https://gitlab.com/JoaoTS20/TQS_Midterm_Assignment
- QA dashboard
  https://sonarcloud.io/dashboard?id=JoaoTS20_TQS_Midterm_Assignment

Reference materials

- External API: https://www.weatherbit.io/api/airquality-current
- https://dev.to/remast/using-sonarcloud-with-github-actions-and-maven-31kg
- https://www.thymeleaf.org/documentation.html
- https://spring.io/blog/2009/03/27/rest-in-spring-3-resttemplate
- https://www.cloudflare.com/learning/cdn/glossary/time-to-live-ttl/
- https://docs.gitlab.com/ee/user/project/merge_requests/code_quality.html