HW1: Mid-term assignment report

Vânia Morais [102383], v2023-04-10

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.
This project is an application that, given a city, provides the air quality at the moment or in the next 5 days.

1.2 Current limitations

The application uses only one API to obtain the information, which, in case of its failure, would cause the entire application not to work. The statistical data in the cache is only related to the air quality at the moment.

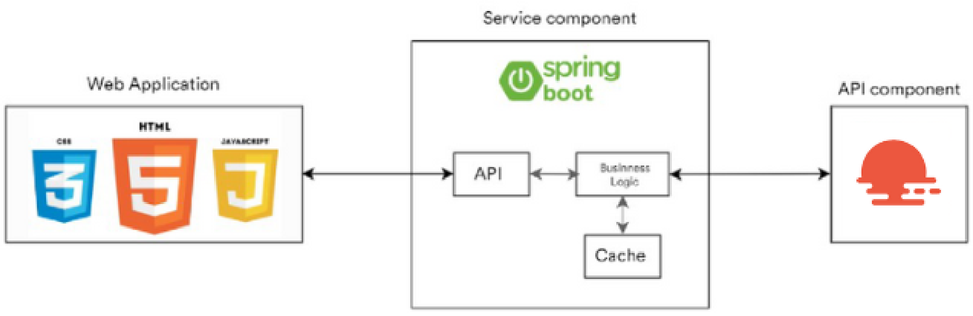
2 Product specification

2.1 Functional scope and supported interactions

Air quality is an important component of our health and is evaluated taking into account some parameters such as nitrogen dioxide (NO2), sulfur dioxide (SO2), carbon monoxide (CO), ozone (O3) and inhalable particles (PM10). Depending on the amount of each of these parameters, the quality can be evaluated from 1 to 5, where 1 is very good and 5 is very bad. Therefore, this application allows you to see all this data, both at the moment and within five days. All this information is provided by the Air Pollution API.

2.2 System architecture

The arquitecture is divided in three sections: Service Component (Spring Boot), Web component(HTML, Javascript, CSS) and the OpenWeather Air Pollution API.



2.3 API for developers

The app has five endpoints:

→ **/airquality/{city}:** gives the current air quality for the city

→ **/futureairquality/{city}:** gives the air quality of the next five days for the city

→ **/checkcity/{city}:** checks if data already exists in cache

→ **/allCache:** returns all information that is cached

→ **/cacheStatus:** returns the number of requests made, how many are hits and how many are misses

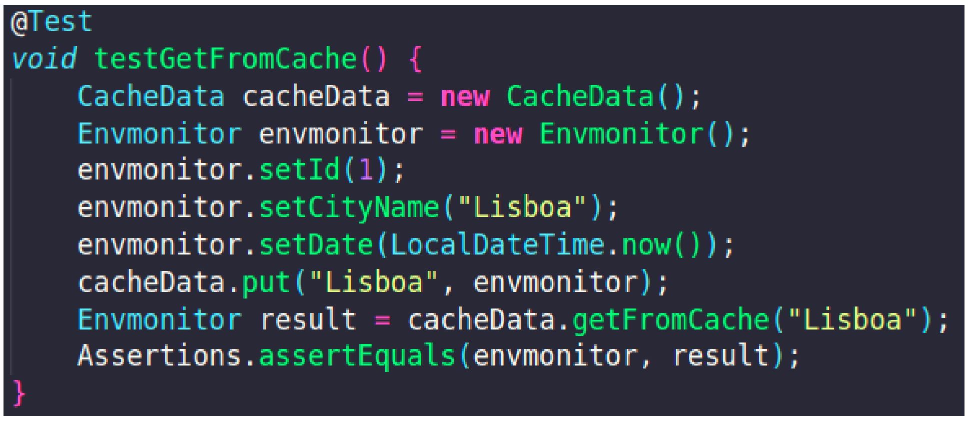
3 Quality assurance

3.1 Overall strategy for testing

The test development strategy was to take advantage of what was taught in class and try to apply it in the best possible way to this app. Unit tests, service level tests using Mockito, integration tests using MockMvc, functional tests using SeleniumJupiter and finally an analysis with SonarQube.

3.2 Unit and integration testing

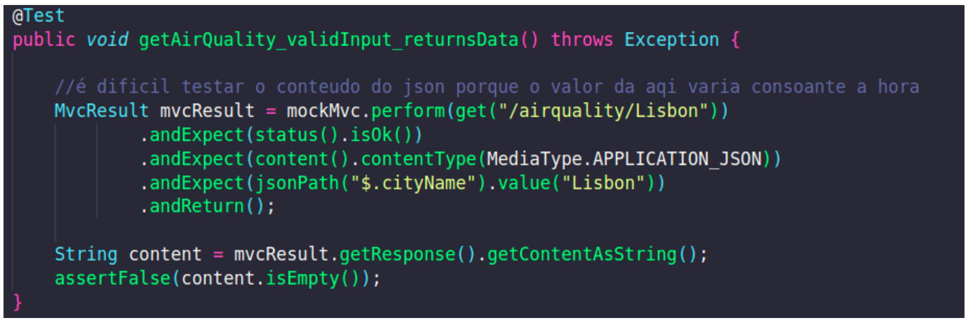
Unit tests were used to test the functionality of the cache. Insertion and removal of elements were tested. One of the examples is the function **testGetFromCache()** where an element is inserted and it is checked if the insertion was successful.



```java
@Test
void testGetFromCache() {
    CacheData cacheData = new CacheData();
    Envmonitor envmonitor = new Envmonitor();
    envmonitor.setId(1);
    envmonitor.setCityName("Lisboa");
    envmonitor.setDate(LocalDateTime.now());
    cacheData.put("Lisboa", envmonitor);
    Envmonitor result = cacheData.getFromCache("Lisboa");
    Assertions.assertEquals(envmonitor, result);
}
```

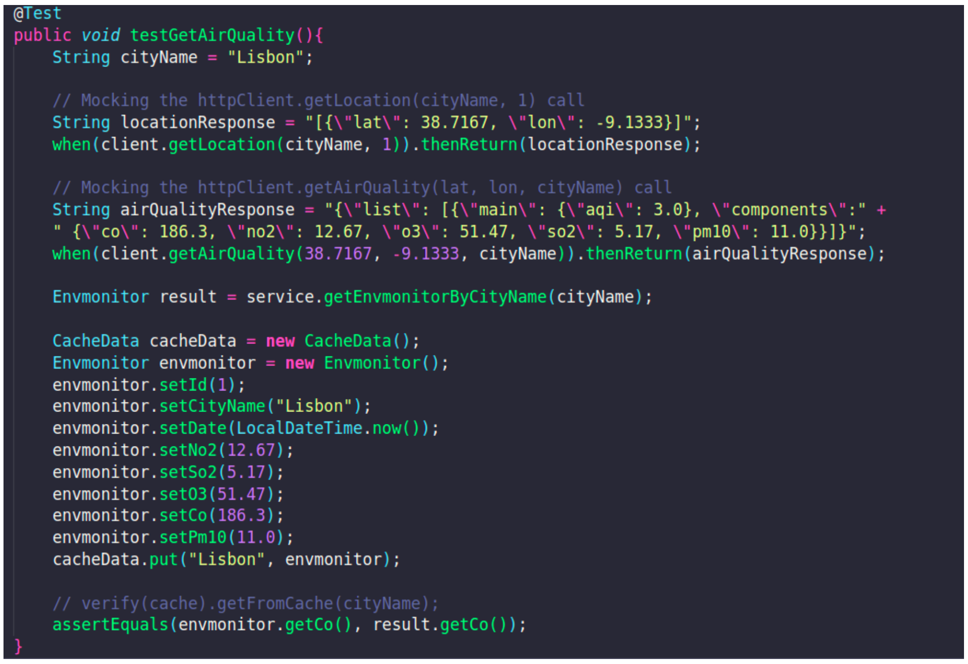
Other tests were **checkIfClears()** and **getAllEnvmonitors()** where in the first one an element is inserted and it is the cache cleaning and in the second two elements are inserted and it is tested if both are returned.

In the integration tests, the endpoints are tested with MockMvc, which is a class from the string test library that simulates HTTP requests without having to run the app. Here is the function getAirQuality_validInput_returnsData() where the endpoint "/airquality/Lisbon" is tested.



```java
@Test
public void getAirQuality_validInput_returnsData() throws Exception {

    //é dificil testar o conteudo do json porque o valor da aqi varia consoante a hora
    MvcResult mvcResult = mockMvc.perform(get("/airquality/Lisbon"))
            .andExpect(status().isOk())
            .andExpect(content().contentType(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath("$.cityName").value("Lisbon"))
            .andReturn();

    String content = mvcResult.getResponse().getContentAsString();
    assertFalse(content.isEmpty());
}
```

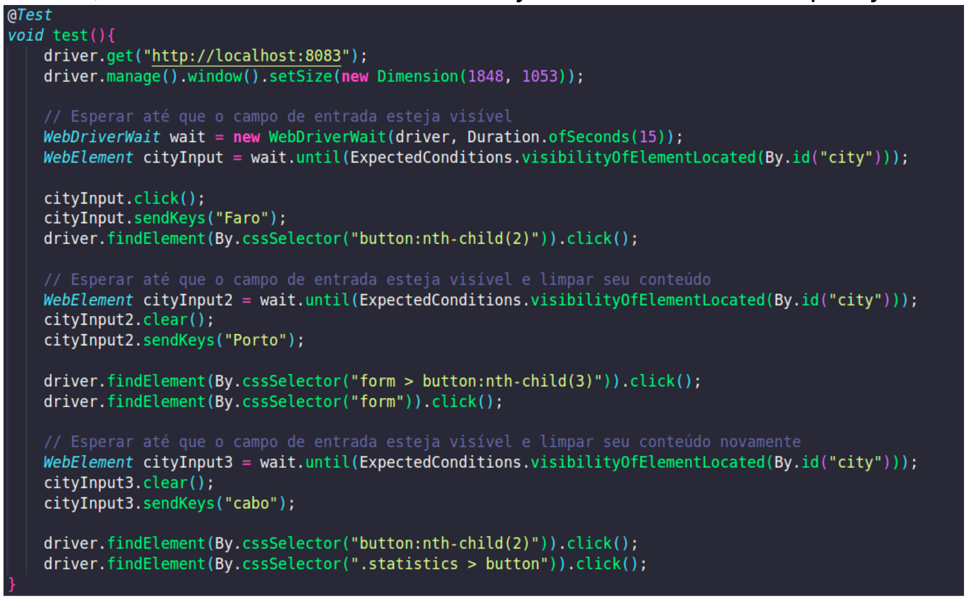
Using mock, service level tests were performed, which tested service functions. For example, the function testGetAirQuality() tests the function getEnvmonitorByCityName(String cityName). Also has functions testGetNonValidInfoByCountryClass() and testGetAirQualityFromCache().



```java
@Test
public void testGetAirQuality(){
    String cityName = "Lisbon";

    // Mocking the httpClient.getLocation(cityName, 1) call
    String locationResponse = "[{\"lat\": 38.7167, \"lon\": -9.1333}]";
    when(client.getLocation(cityName, 1)).thenReturn(locationResponse);

    // Mocking the httpClient.getAirQuality(lat, lon, cityName) call
    String airQualityResponse = "{\"list\": [{\"main\": {\"aqi\": 3.0}, \"components\":" +
    " {\"co\": 186.3, \"no2\": 12.67, \"o3\": 51.47, \"so2\": 5.17, \"pm10\": 11.0}}]}";
    when(client.getAirQuality(38.7167, -9.1333, cityName)).thenReturn(airQualityResponse);

    Envmonitor result = service.getEnvmonitorByCityName(cityName);

    CacheData cacheData = new CacheData();
    Envmonitor envmonitor = new Envmonitor();
    envmonitor.setId(1);
    envmonitor.setCityName("Lisbon");
    envmonitor.setDate(LocalDateTime.now());
    envmonitor.setNo2(12.67);
    envmonitor.setSo2(5.17);
    envmonitor.setO3(51.47);
    envmonitor.setCo(186.3);
    envmonitor.setPm10(11.0);
    cacheData.put("Lisbon", envmonitor);

    // verify(cache).getFromCache(cityName);
    assertEquals(envmonitor.getCo(), result.getCo());
}
```

3.3 Functional testing

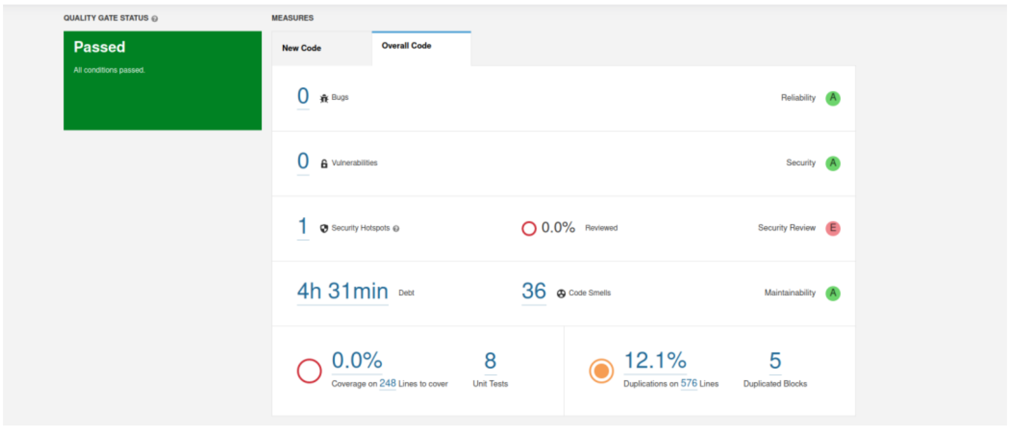
For functional tests, the Selinium IDE extension was used, which allows recording and reproducing user interactions with a web page. The following test opens a Firefox page on the app link, writes "Faro" in the input, clicks the "Get Air Quality" button, writes "Porto" in the input, clicks the "Get Future Air Quality" button, does the first two steps again, but with the input "cable" and finally click on the statistics button, where you must 2 requests, 0 hits and 2 misses, since these data are related only to the search for air quality at the moment.



```java
@Test
void test(){
    driver.get("http://localhost:8083");
    driver.manage().window().setSize(new Dimension(1848, 1053));

    // Esperar até que o campo de entrada esteja visível
    WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(15));
    WebElement cityInput = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("city")));

    cityInput.click();
    cityInput.sendKeys("Faro");
    driver.findElement(By.cssSelector("button:nth-child(2)")).click();

    // Esperar até que o campo de entrada esteja visível e limpar seu conteúdo
    WebElement cityInput2 = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("city")));
    cityInput2.clear();
    cityInput2.sendKeys("Porto");

    driver.findElement(By.cssSelector("form > button:nth-child(3)")).click();
    driver.findElement(By.cssSelector("form")).click();

    // Esperar até que o campo de entrada esteja visível e limpar seu conteúdo novamente
    WebElement cityInput3 = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("city")));
    cityInput3.clear();
    cityInput3.sendKeys("cabo");

    driver.findElement(By.cssSelector("button:nth-child(2)")).click();
    driver.findElement(By.cssSelector(".statistics > button")).click();
}
```

3.4 Code quality analysis

Code quality analysis was made using SonarQube. There are 36 code smells.



QUALITY GATE STATUS
Passed
All conditions passed.

MEASURES
New Code | Overall Code

0 Bugs — Reliability A
0 Vulnerabilities — Security A
1 Security Hotspots — 0.0% Reviewed — Security Review E
4h 31min Debt — 36 Code Smells — Maintainability A
0.0% Coverage on 248 Lines to cover — 8 Unit Tests
12.1% Duplications on 576 Lines — 5 Duplicated Blocks

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/VaniaMMorais/tqs_102383/tree/main/HW1
Video demo	Located in the Git repository

Reference materials

→ Class work
→ OpenWeather API for air polution
→ OpenWeather API for geolocation
→ Chat GPT for tests and problems resolution