HW1: Mid-term assignment report

Tiago Gomes Carvalho [104142], 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.

The goal of this project is to develop a multi-layer web application, in Spring Boot, supplied with automated tests, that should provide air quality details for various regions. The developed application consists of a REST-API service integrated with a Web App, both supported by several sets of tests to affirm the integrity of the entire system. Embedded in the API, there is a cache system handling consistent accesses and storing non-persistent data.

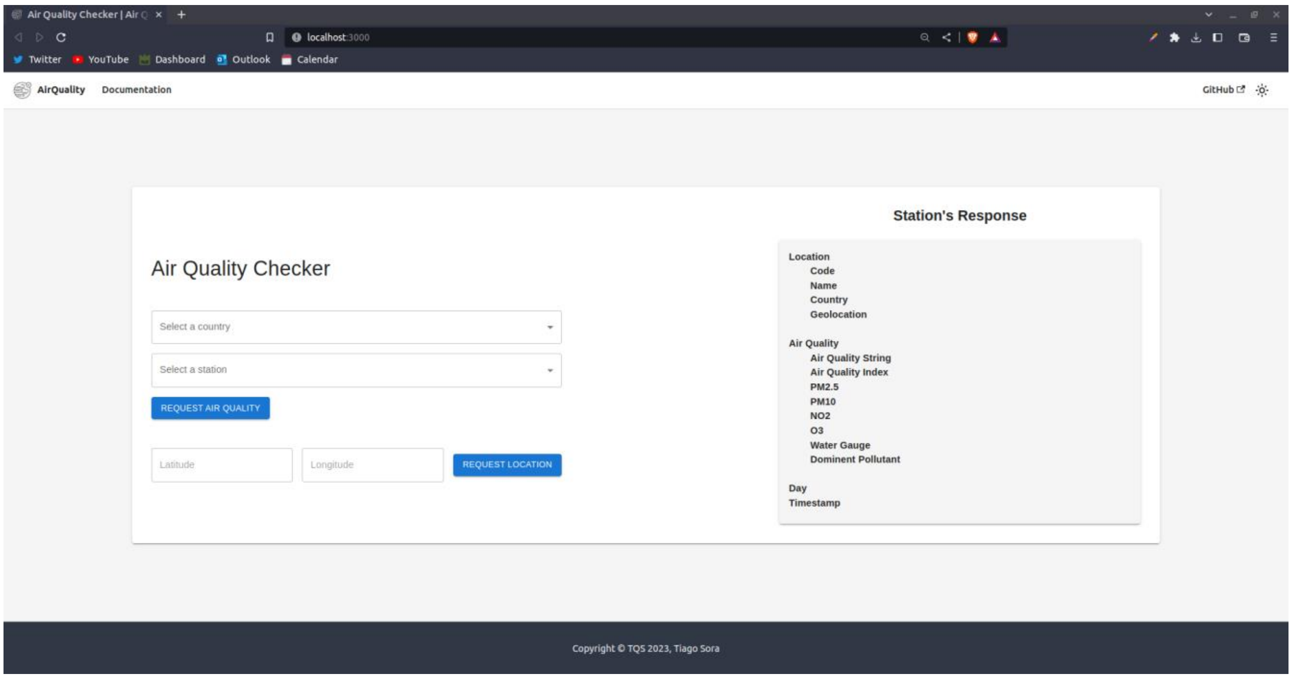
1.2 Current limitations

The most limiting factor regarding the use of this application is the necessity of an external data provider. Users would not be able to use this product if the provider is, for the same reason, not able to return valid responses to the users' requests. This problem would be solved with persistent data storage, with an internal database, saving data periodically.

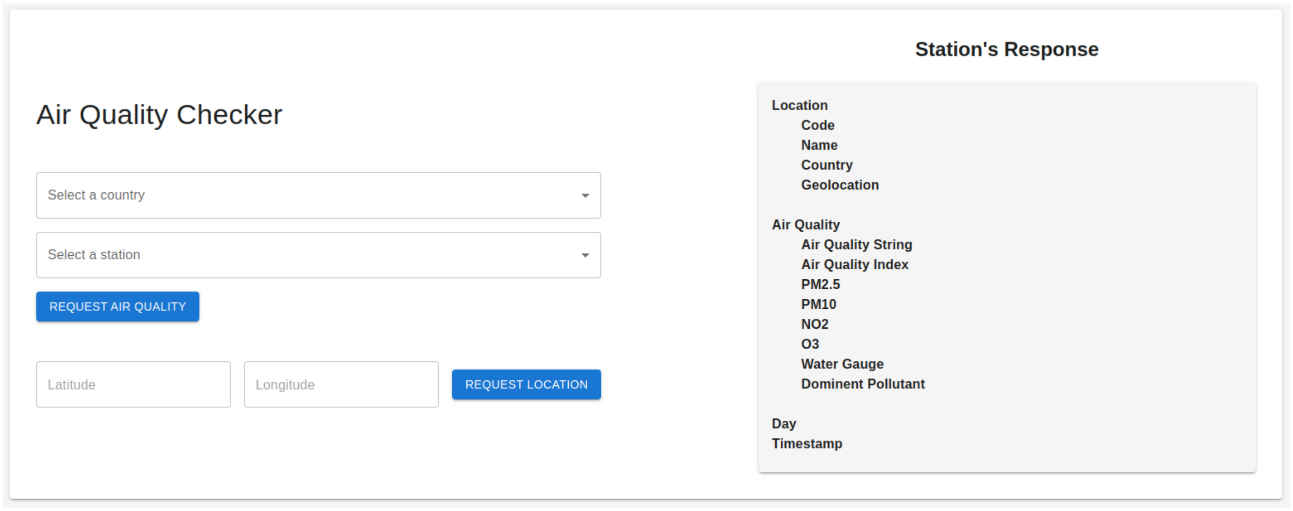
2 Product specification

2.1 Functional scope and supported interactions

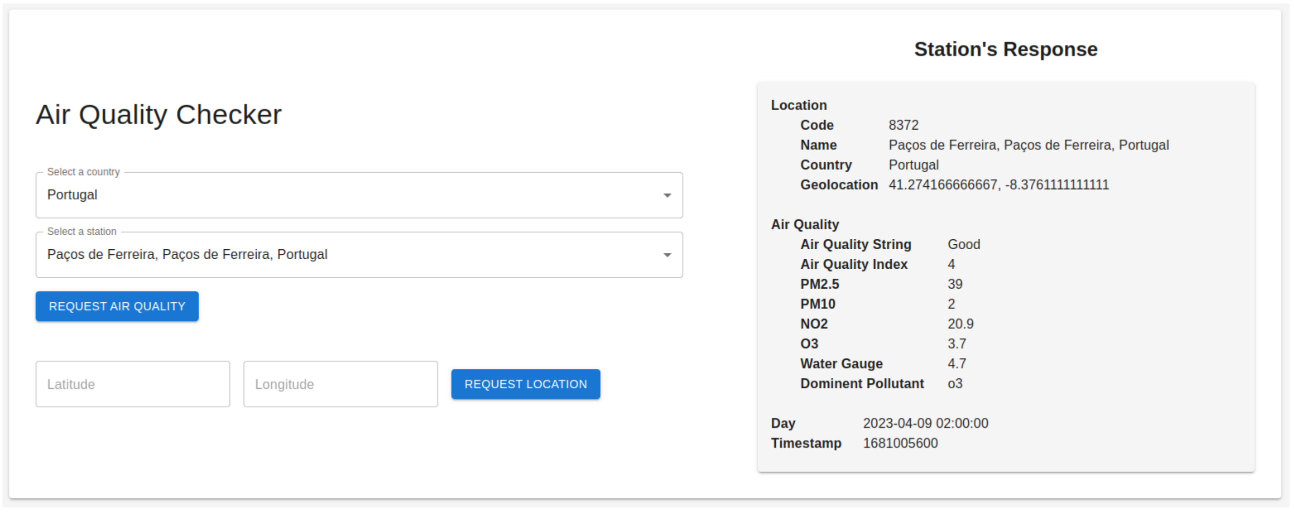
The product was designed to be used by everyone without needing to take too many unnecessary steps to achieve the final goal, obtaining the air quality information that the user wants.



Taking that in consideration the user may use the system through its interface, that consists of a main page, in which the user can query the request for the air quality of the location he desires. Upon navigating to the URL, the user will see a central box with some interactive elements, such as drop-downs, text inputs and buttons.

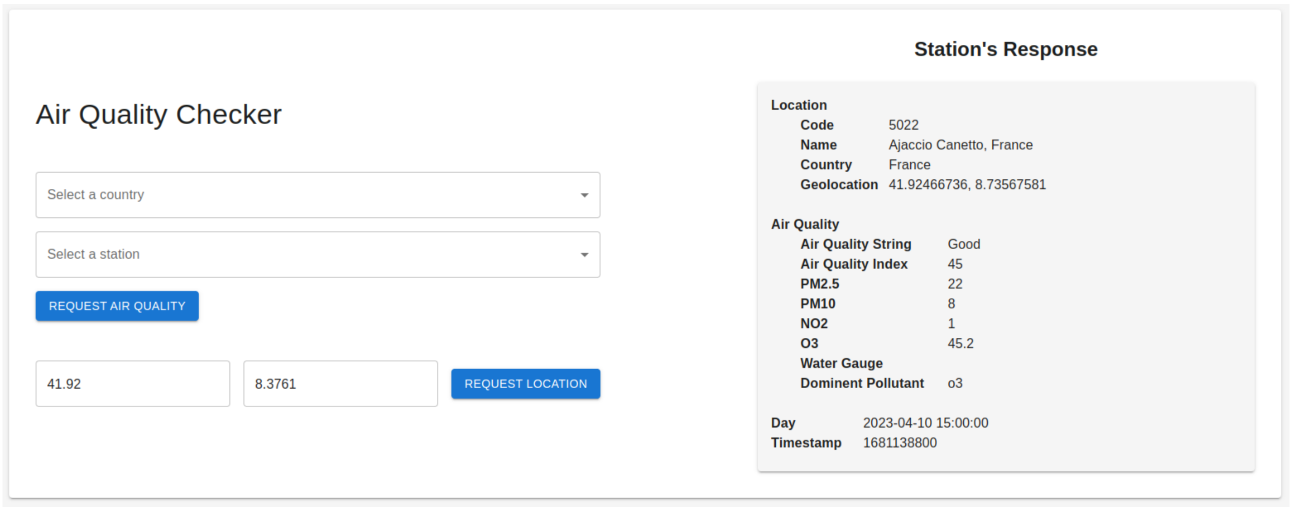


To obtain the air quality related to the location of one of the available stations mapped by the system, the user has two options.



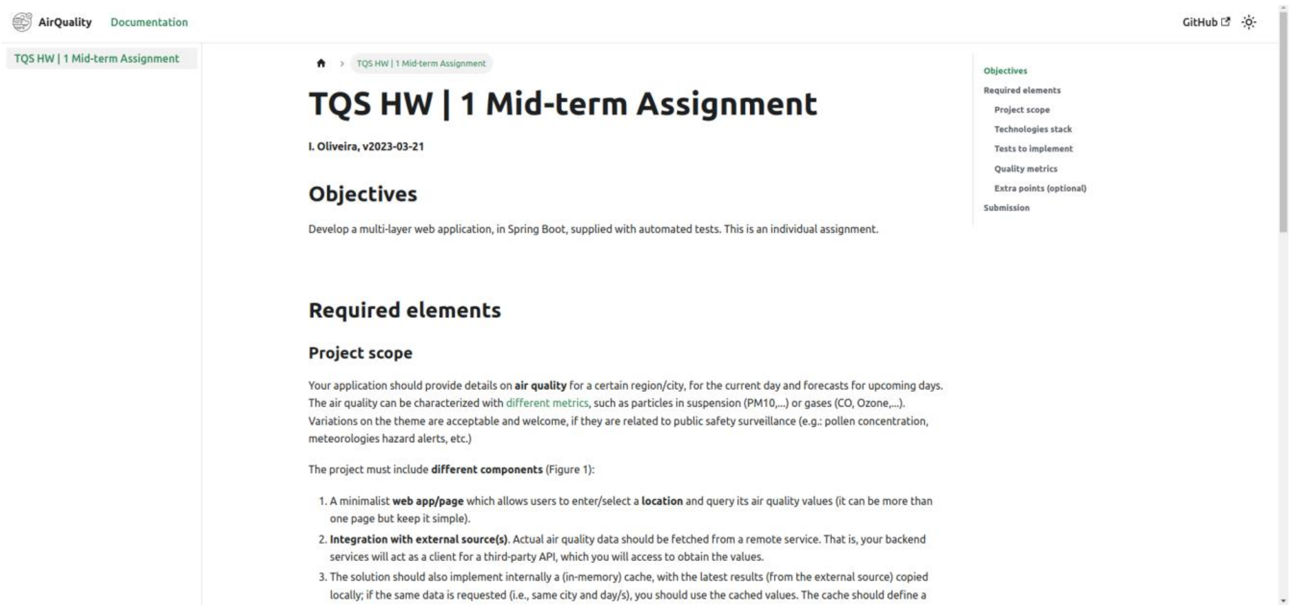
```
Location
      Code          8372
      Name          Paços de Ferreira, Paços de Ferreira, Portugal
      Country       Portugal
      Geolocation   41.274166666667, -8.3761111111111

Air Quality
      Air Quality String   Good
      Air Quality Index    4
      PM2.5                39
      PM10                 2
      NO2                  20.9
      O3                   3.7
      Water Gauge          4.7
      Doninent Pollutant   o3

Day         2023-04-09 02:00:00
Timestamp   1681005600
```

The user can, firstly, select the drop-down "Select a country" and choose one the options available. Once the desired country is selected, the user can select a location from that country, using the second drop-down "Select a station." Locations mapped by stations from other countries will not be available, so the user must select a different country to access them.
After selecting the station, the user may click on the "Request Air Quality" button and the air quality data will then be displayed on the board on the right side.



```
Location
      Code          5022
      Name          Ajaccio Canetto, France
      Country       France
      Geolocation   41.92466736, 8.73567581

Air Quality
      Air Quality String   Good
      Air Quality Index    45
      PM2.5                22
      PM10                 8
      NO2                  1
      O3                   45.2
      Water Gauge
      Doninent Pollutant   o3

Day         2023-04-10 15:00:00
Timestamp   1681138800
```

Alternatively, the user can get air quality by geolocation, i.e., through latitude and longitude. Initially the user must insert the latitude and longitude values in the "Latitude" and "Longitude" text inputs, respectively. Those values must be numbers. Then the user can click the "Request Location" button to obtain the air quality data of the station closest to the coordinates the user inserted.



TQS HW | 1 Mid-term Assignment

I. Oliveira, v2023-03-21

Objectives

Develop a multi-layer web application, in Spring Boot, supplied with automated tests. This is an individual assignment.

Required elements

Project scope

Your application should provide details on **air quality** for a certain region/city, for the current day and forecasts for upcoming days. The air quality can be characterized with different metrics, such as particles in suspension (PM10,...) or gases (CO, Ozone,...). Variations on the theme are acceptable and welcome, if they are related to public safety surveillance (e.g.: pollen concentration, meteorologies hazard alerts, etc.)

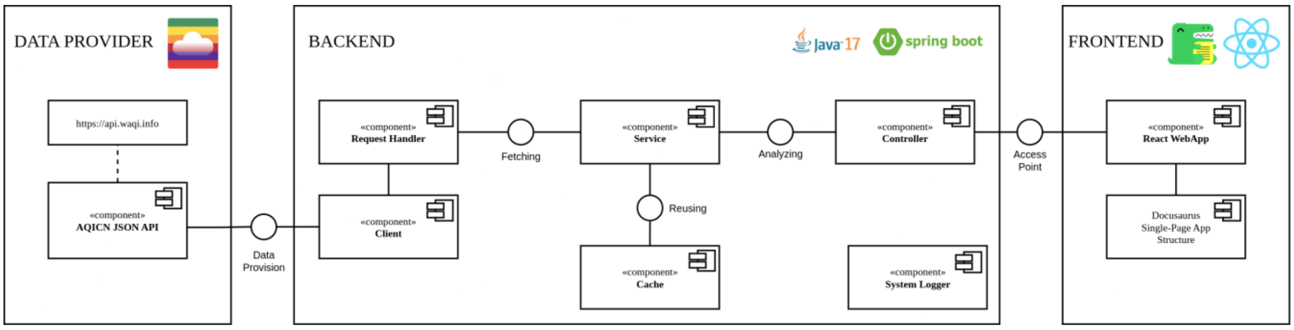
The project must include **different components** (Figure 1):

1. A minimalist **web app/page** which allows users to enter/select a **location** and query its air quality values (it can be more than one page but keep it simple).
2. **Integration with external source(s)**. Actual air quality data should be fetched from a remote service. That is, your backend services will act as a client for a third-party API, which you will access to obtain the values.
3. The solution should also implement internally a (in-memory) cache, with the latest results (from the external source) copied locally; if the same data is requested (i.e., same city and day/s), you should use the cached values. The cache should define a

Besides the main interaction of the interface the user can select “Documentation” or “GitHub” in the navigation bar on the top, to read more about the systems’ documentation or go directly to the GitHub repository containing the developed product.

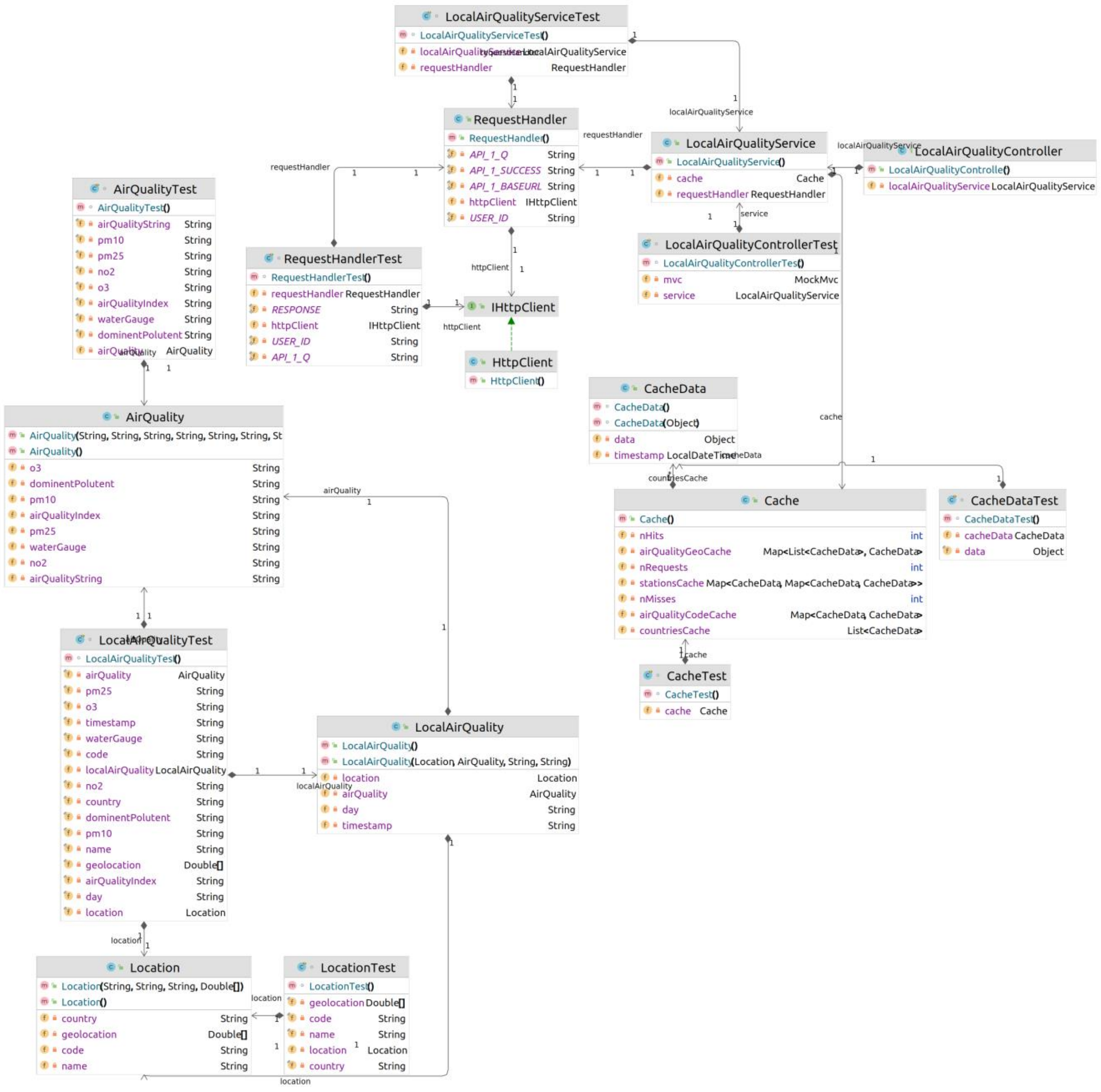
To run both contents, there’s a docker compose file that may be run with “docker compose up –build". Alternatively, backend can be initiated using “mvn clean package spring-boot:run” and frontend with “npm run dev”. After that, the API can be access in “localhost:8080/api/v1/” and the frontend in “localhost:3000/”.

2.2 System architecture



The backend component is based on Spring Boot, consisting of a Web Controller supported by a Service, using caching and logging.

For the frontend, it was chosen React, a JavaScript framework. It was also used Docusaurus, a tool designed for documentation websites, to build the interface infrastructure and design details.



To better view the pictures, both architecture and classes' diagram are available on the repository.

2.3 API for developers

The developed API contains a total of 5 endpoints. Three of them support one air quality search scenario (using station identification), one supports the other scenario (using geo location), and the last endpoint is to access the API's cache component and obtain its statistics.

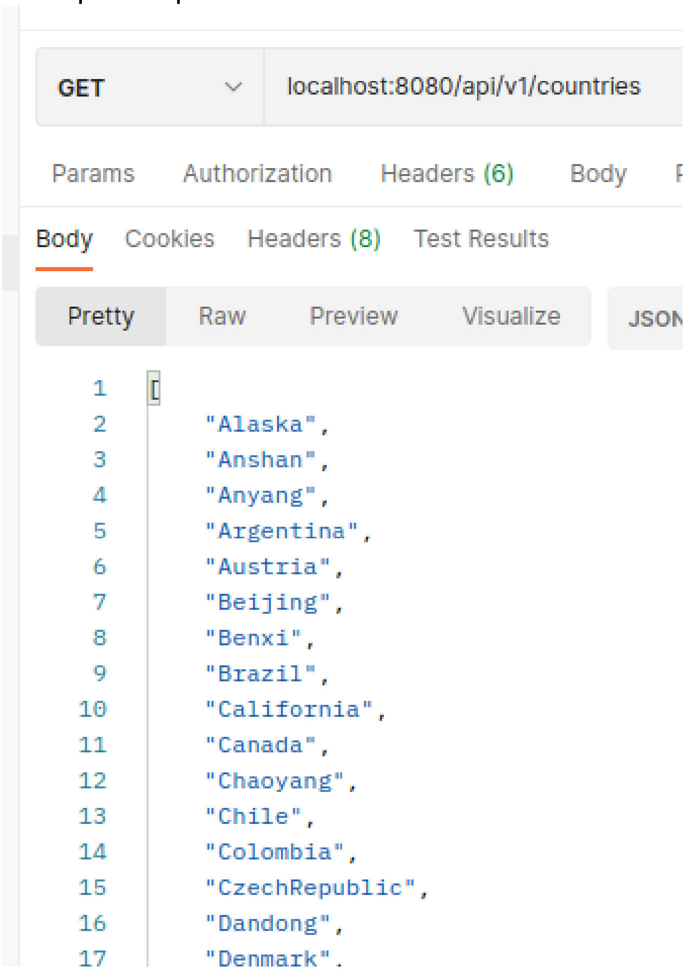
All of them respect the base format, in this case, "api/v1/" and the API is not able to recognize any request that does not match any of the 5 endpoints.

These are the endpoints:

2.3.1 GET – api/v1/countries

Endpoint	api/v1/countries
Description	Returns the list of countries with stations mapped by the API
Example URL	http://localhost:8080/api/v1/countries
Parameters	None
Authorization	None
Response Type	JSON

Example Response:

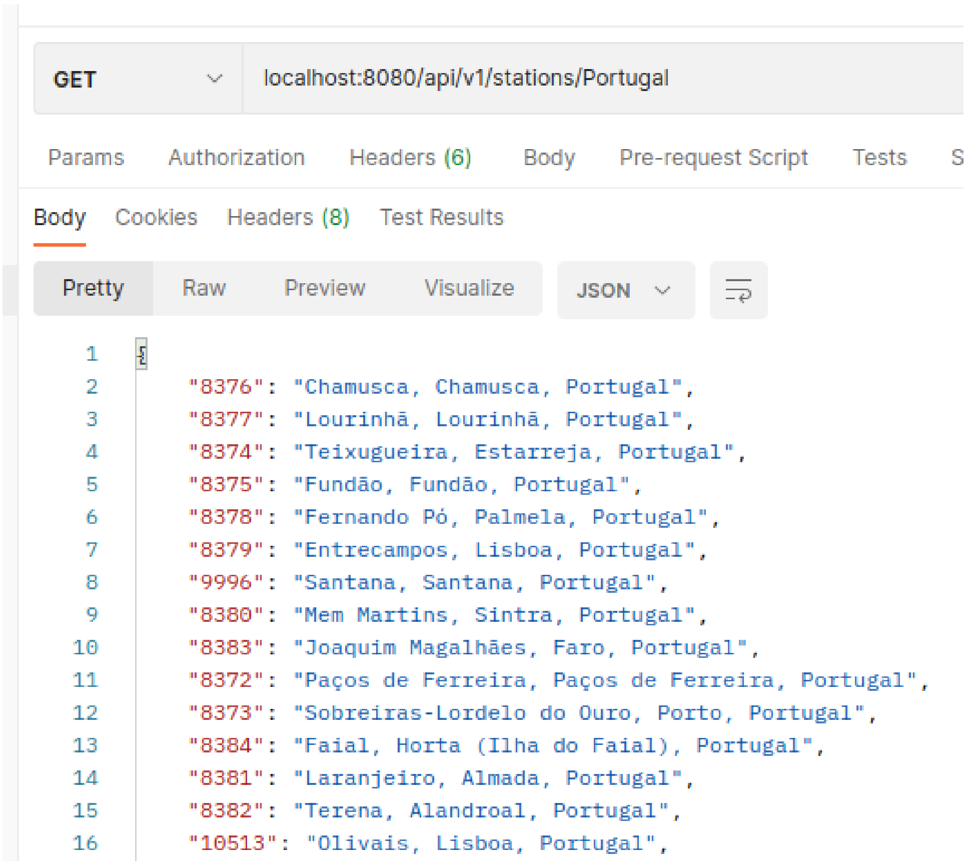


```
[
    "Alaska",
    "Anshan",
    "Anyang",
    "Argentina",
    "Austria",
    "Beijing",
    "Benxi",
    "Brazil",
    "California",
    "Canada",
    "Chaoyang",
    "Chile",
    "Colombia",
    "CzechRepublic",
    "Dandong",
    "Denmark",
```

2.3.2 GET – api/v1/stations/{country}

Endpoint	api/v1/stations/{country}
Description	Returns the list of stations mapped by the API that are located in a given country
Example URL	http://localhost:8080/api/v1/stations/Portugal
Parameters	Country - String
Authorization	None
Response Type	JSON

Example Response:



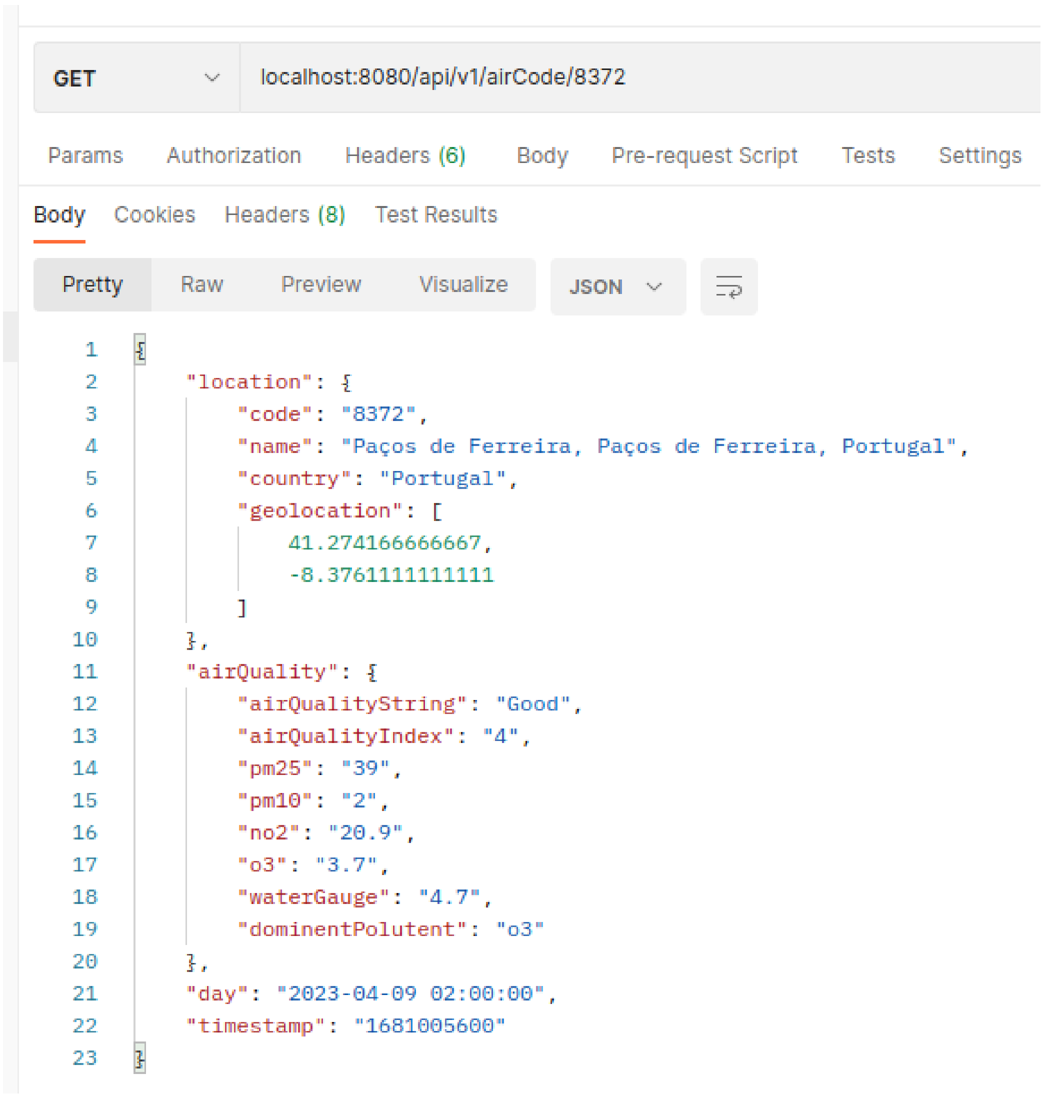
localhost:8080/api/v1/stations/Portugal

```
{
    "8376": "Chamusca, Chamusca, Portugal",
    "8377": "Lourinhã, Lourinhã, Portugal",
    "8374": "Teixugueira, Estarreja, Portugal",
    "8375": "Fundão, Fundão, Portugal",
    "8378": "Fernando Pó, Palmela, Portugal",
    "8379": "Entrecampos, Lisboa, Portugal",
    "9996": "Santana, Santana, Portugal",
    "8380": "Mem Martins, Sintra, Portugal",
    "8383": "Joaquim Magalhães, Faro, Portugal",
    "8372": "Paços de Ferreira, Paços de Ferreira, Portugal",
    "8373": "Sobreiras-Lordelo do Ouro, Porto, Portugal",
    "8384": "Faial, Horta (Ilha do Faial), Portugal",
    "8381": "Laranjeiro, Almada, Portugal",
    "8382": "Terena, Alandroal, Portugal",
    "10513": "Olivais, Lisboa, Portugal",
```

2.3.3 GET – api/v1/airCode/{stationCode}

Endpoint	api/v1/airCode/{stationCode}
Description	Returns the air quality data regarding the station identified with a given code
Example URL	http://localhost:8080/api/v1/airCode/@8372
Parameters	StationCode (Integer)
Authorization	None
Response Type	JSON

Example Response:



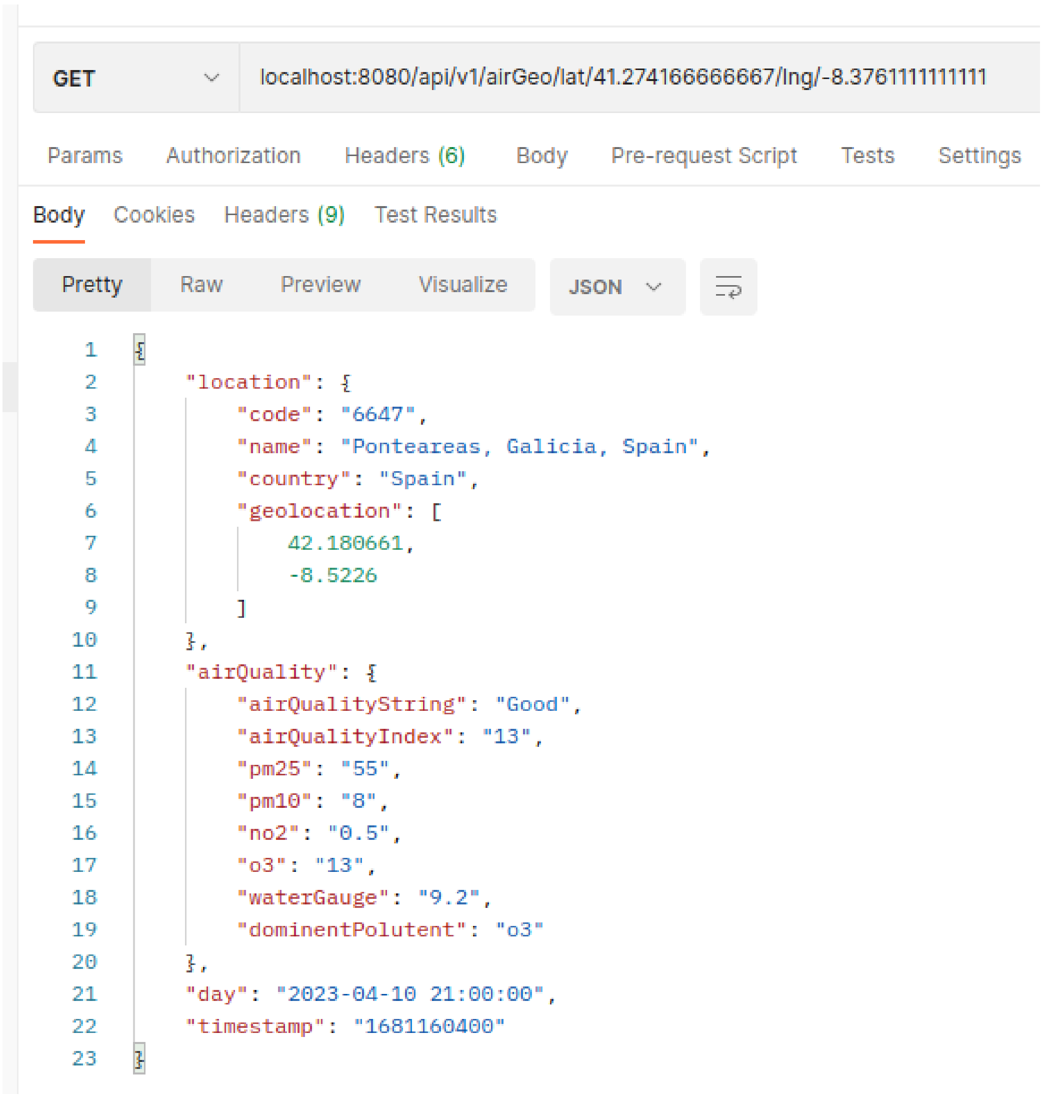
localhost:8080/api/v1/airCode/8372

```
{
    "location": {
        "code": "8372",
        "name": "Paços de Ferreira, Paços de Ferreira, Portugal",
        "country": "Portugal",
        "geolocation": [
            41.274166666667,
            -8.3761111111111
        ]
    },
    "airQuality": {
        "airQualityString": "Good",
        "airQualityIndex": "4",
        "pm25": "39",
        "pm10": "2",
        "no2": "20.9",
        "o3": "3.7",
        "waterGauge": "4.7",
        "dominentPolutent": "o3"
    },
    "day": "2023-04-09 02:00:00",
    "timestamp": "1681005600"
}
```

2.3.4 GET – api/v1/airGeo/lat/{lat}/lng/{lng}

Endpoint	api/v1/airGeo/lat/{lat}/lng/{lng}
Description	Returns the air quality data regarding the closest station to the given coordinates (latitude, longitude)
Example URL	http://localhost:8080/api/v1/airGeo/lat/41.2741666/lng/-8.37611111
Parameters	lat (Double), lng (Double)
Authorization	None
Response Type	JSON

Example Response:



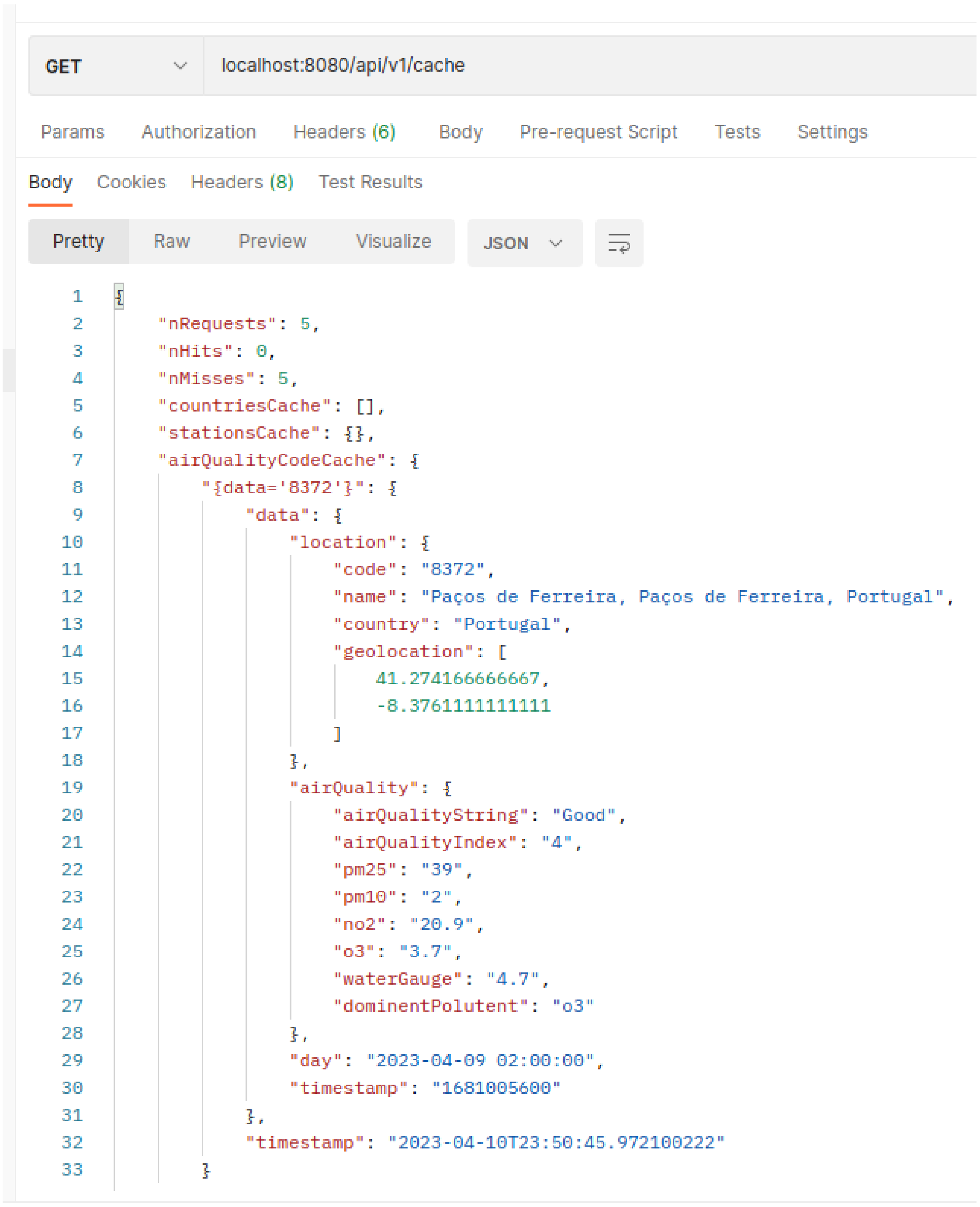
```
localhost:8080/api/v1/airGeo/lat/41.274166666667/lng/-8.3761111111111
```

```
{
    "location": {
        "code": "6647",
        "name": "Ponteareas, Galicia, Spain",
        "country": "Spain",
        "geolocation": [
            42.180661,
            -8.5226
        ]
    },
    "airQuality": {
        "airQualityString": "Good",
        "airQualityIndex": "13",
        "pm25": "55",
        "pm10": "8",
        "no2": "0.5",
        "o3": "13",
        "waterGauge": "9.2",
        "dominentPolutent": "o3"
    },
    "day": "2023-04-10 21:00:00",
    "timestamp": "1681160400"
}
```

2.3.5 GET – api/v1/cache

Endpoint	api/v1/cache
Description	Returns API's cache data and statistics
Example URL	http://localhost:8080/api/v1/cache
Parameters	None
Authorization	None
Response Type	JSON

Example Response:



```
localhost:8080/api/v1/cache
```

```json
{
    "nRequests": 5,
    "nHits": 0,
    "nMisses": 5,
    "countriesCache": [],
    "stationsCache": {},
    "airQualityCodeCache": {
        "{data='8372'}": {
            "data": {
                "location": {
                    "code": "8372",
                    "name": "Paços de Ferreira, Paços de Ferreira, Portugal",
                    "country": "Portugal",
                    "geolocation": [
                        41.274166666667,
                        -8.3761111111111
                    ]
                },
                "airQuality": {
                    "airQualityString": "Good",
                    "airQualityIndex": "4",
                    "pm25": "39",
                    "pm10": "2",
                    "no2": "20.9",
                    "o3": "3.7",
                    "waterGauge": "4.7",
                    "dominentPolutent": "o3"
                },
                "day": "2023-04-09 02:00:00",
                "timestamp": "1681005600"
            },
            "timestamp": "2023-04-10T23:50:45.972100222"
        }
```

3 Quality assurance

3.1 Overall strategy for testing

Starting the development was the hardest part. It just started getting smooth I started developing towards the architecture.

Starting with the backend, I schematized and tested the models needed to embed all the data returned from the external API. I also did the same to the remaining components of the backend. Finally, I reviewed the backend to maximize the amount of code coverage.

With the frontend, I developed a minimalist interface able to fulfill every system requirement. After that, using Selenium and Cucumber, I created new tests and changed the system towards the test excepted results.

3.2 Unit and integration testing

I used unit tests to test most of the behavior of the system's models. I used mocks to unit test other components, like the Service (mocking the Request Handler) and the Request Handler (mocking the Client). To test the Controller, I used Integration Tests to test, with the support of MockMVC.

3.2.1 Unit Tests:

AirQualityTest.java

- whenGetMethod_thenReturnExpectedValue
- whenSetMethod_thenReturnExpectedValue
- whenSetNewAirQualityIndex_thenSetNewAirQualityString
- whenEmptyConstructor_thenReturnInvalidAQI
- whenEqualObject_AssertEquals
- TestToString

LocationTest.java

- whenGetMethod_thenReturnExpectedValue
- whenSetMethod_thenReturnExpectedValue
- whenEmptyConstructor_thenReturnInvalidAQI
- whenEqualObject_AssertEquals
- testToString

LocalAirQualityTest.java

- whenGetMethod_thenReturnExpectedValue
- whenSetMethod_thenReturnExpectedValue
- whenEmptyConstructor_thenReturnInvalidAQI
- whenEqualObject_AssertEquals
- testToString

CacheDataTest.java

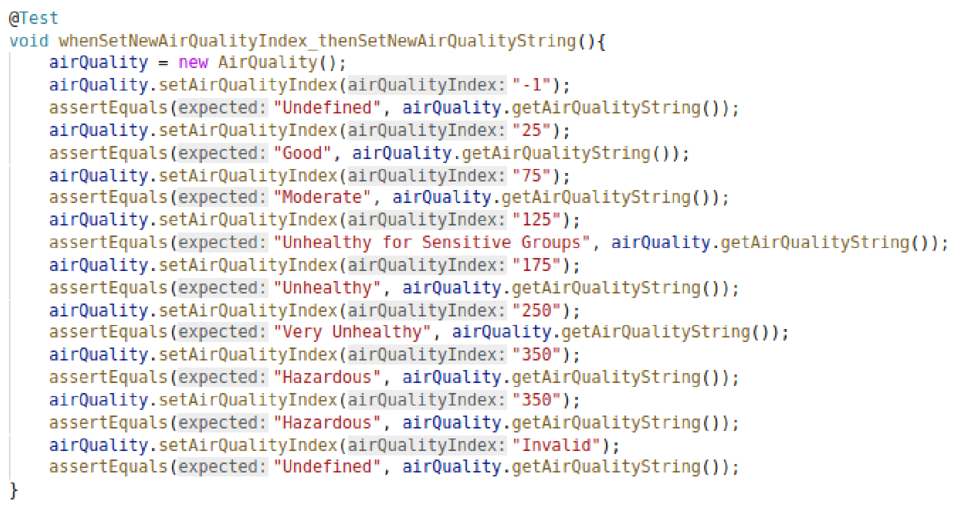
- whenGetMethod_thenReturnExpectedValue
- whenSetMethod_thenReturnExpectedValue
- whenEmptyConstructor_thenReturnInvalidAQI
- whenEqualObject_AssertEquals
- testToString

CacheTest.java

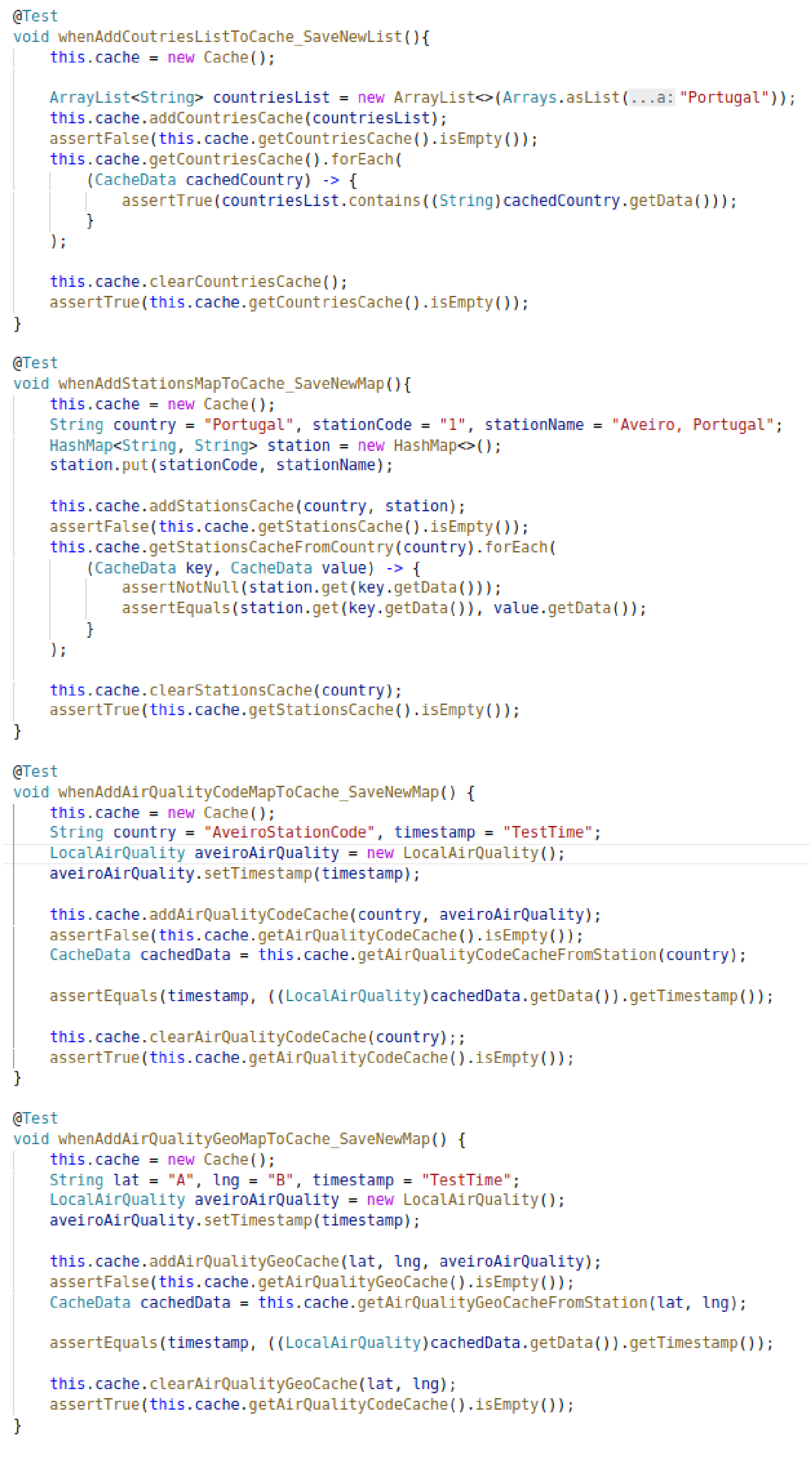
- whenGetMethod_thenReturnExpectedValue

- whenSetMethod_thenReturnExpectedValue
- whenNewEventInCache_RegisterEvent
- whenAddCoutriesListToCache_SaveNewList
- whenAddStationsMapToCache_SaveNewMap
- whenAddAirQualityCodeMapToCache_SaveNewMap
- whenAddAirQualityGeoMapToCache_SaveNewMap
- whenEqualObject_AssertEquals
- testToString

Here are some examples of Unit Test:



```
@Test
void whenSetNewAirQualityIndex_thenSetNewAirQualityString(){
    airQuality = new AirQuality();
    airQuality.setAirQualityIndex(airQualityIndex: "-1");
    assertEquals(expected: "Undefined", airQuality.getAirQualityString());
    airQuality.setAirQualityIndex(airQualityIndex: "25");
    assertEquals(expected: "Good", airQuality.getAirQualityString());
    airQuality.setAirQualityIndex(airQualityIndex: "75");
    assertEquals(expected: "Moderate", airQuality.getAirQualityString());
    airQuality.setAirQualityIndex(airQualityIndex: "125");
    assertEquals(expected: "Unhealthy for Sensitive Groups", airQuality.getAirQualityString());
    airQuality.setAirQualityIndex(airQualityIndex: "175");
    assertEquals(expected: "Unhealthy", airQuality.getAirQualityString());
    airQuality.setAirQualityIndex(airQualityIndex: "250");
    assertEquals(expected: "Very Unhealthy", airQuality.getAirQualityString());
    airQuality.setAirQualityIndex(airQualityIndex: "350");
    assertEquals(expected: "Hazardous", airQuality.getAirQualityString());
    airQuality.setAirQualityIndex(airQualityIndex: "350");
    assertEquals(expected: "Hazardous", airQuality.getAirQualityString());
    airQuality.setAirQualityIndex(airQualityIndex: "Invalid");
    assertEquals(expected: "Undefined", airQuality.getAirQualityString());
}
```



```java
@Test
void whenAddCoutriesListToCache_SaveNewList(){
    this.cache = new Cache();

    ArrayList<String> countriesList = new ArrayList<>(Arrays.asList(...a: "Portugal"));
    this.cache.addCountriesCache(countriesList);
    assertFalse(this.cache.getCountriesCache().isEmpty());
    this.cache.getCountriesCache().forEach(
        (CacheData cachedCountry) -> {
            assertTrue(countriesList.contains((String)cachedCountry.getData()));
        }
    );

    this.cache.clearCountriesCache();
    assertTrue(this.cache.getCountriesCache().isEmpty());
}

@Test
void whenAddStationsMapToCache_SaveNewMap(){
    this.cache = new Cache();
    String country = "Portugal", stationCode = "1", stationName = "Aveiro, Portugal";
    HashMap<String, String> station = new HashMap<>();
    station.put(stationCode, stationName);

    this.cache.addStationsCache(country, station);
    assertFalse(this.cache.getStationsCache().isEmpty());
    this.cache.getStationsCacheFromCountry(country).forEach(
        (CacheData key, CacheData value) -> {
            assertNotNull(station.get(key.getData()));
            assertEquals(station.get(key.getData()), value.getData());
        }
    );

    this.cache.clearStationsCache(country);
    assertTrue(this.cache.getStationsCache().isEmpty());
}

@Test
void whenAddAirQualityCodeMapToCache_SaveNewMap() {
    this.cache = new Cache();
    String country = "AveiroStationCode", timestamp = "TestTime";
    LocalAirQuality aveiroAirQuality = new LocalAirQuality();
    aveiroAirQuality.setTimestamp(timestamp);

    this.cache.addAirQualityCodeCache(country, aveiroAirQuality);
    assertFalse(this.cache.getAirQualityCodeCache().isEmpty());
    CacheData cachedData = this.cache.getAirQualityCodeCacheFromStation(country);

    assertEquals(timestamp, ((LocalAirQuality)cachedData.getData()).getTimestamp());

    this.cache.clearAirQualityCodeCache(country);;
    assertTrue(this.cache.getAirQualityCodeCache().isEmpty());
}

@Test
void whenAddAirQualityGeoMapToCache_SaveNewMap() {
    this.cache = new Cache();
    String lat = "A", lng = "B", timestamp = "TestTime";
    LocalAirQuality aveiroAirQuality = new LocalAirQuality();
    aveiroAirQuality.setTimestamp(timestamp);

    this.cache.addAirQualityGeoCache(lat, lng, aveiroAirQuality);
    assertFalse(this.cache.getAirQualityGeoCache().isEmpty());
    CacheData cachedData = this.cache.getAirQualityGeoCacheFromStation(lat, lng);

    assertEquals(timestamp, ((LocalAirQuality)cachedData.getData()).getTimestamp());

    this.cache.clearAirQualityGeoCache(lat, lng);
    assertTrue(this.cache.getAirQualityCodeCache().isEmpty());
}
```

3.2.2 Unit Tests using Mocks:

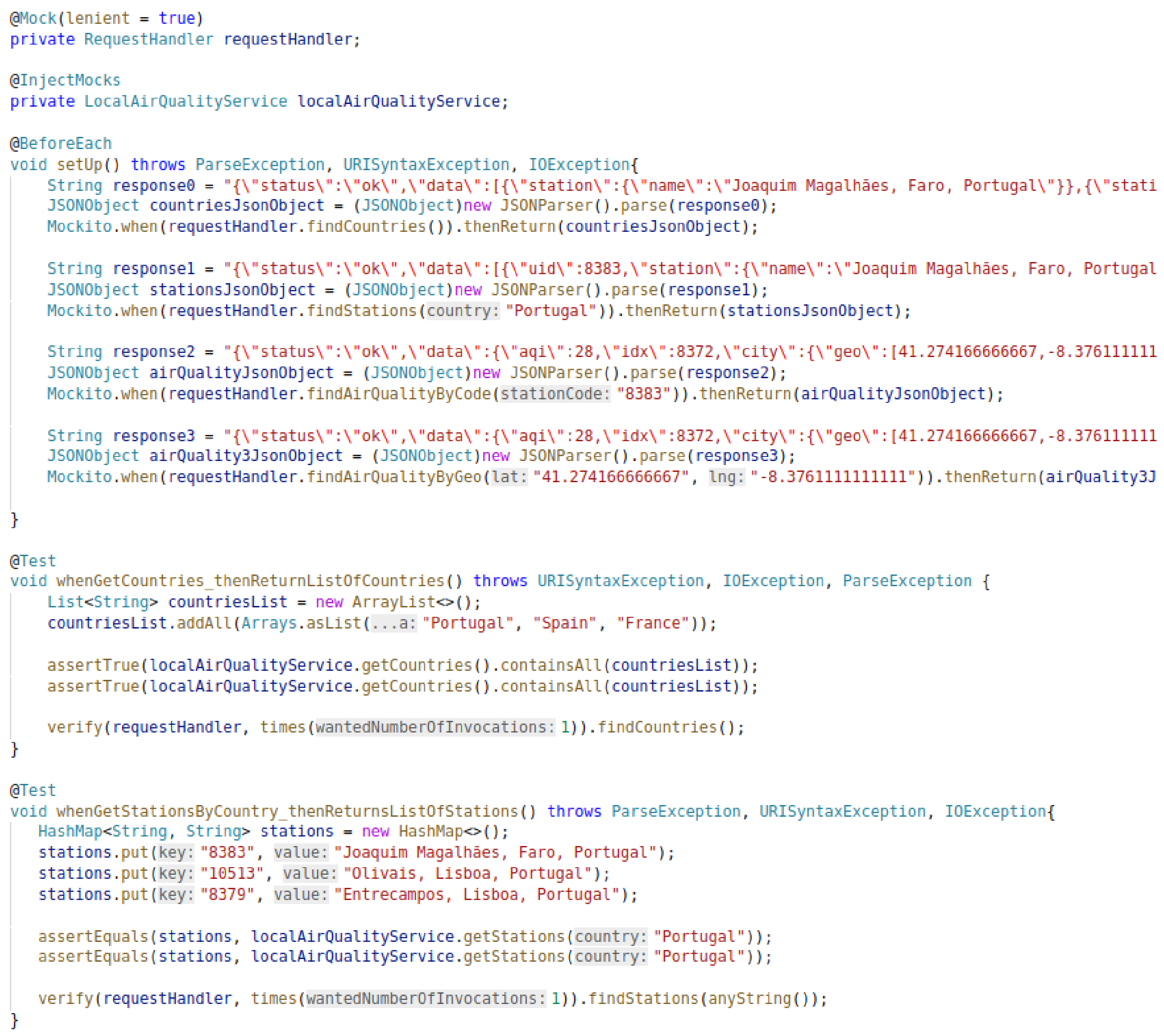
LocalAirQualityServiceTest.java

- whenGetCountries_thenReturnListOfCountries
- whenGetStationsByCountry_thenReturnsListOfStations
- whenGetStationsByInvalidCountry_thenReturnsEmptyList
- whenGetAirQualityByCode_thenReturnLocalAirQuality
- whenGetAirQualityByGeo_thenReturnLocalAirQuality
- whenGetAirQualityByInvalidStation_thenReturnsErrorMessage
- whenGetAirQualityByInvalidGeo_thenReturnsErrorMessage

RequestHandlerTest.java

- whenFindCountries_HandleRequestCorrecly
- whenFindStations_HandleRequestCorrecly
- whenFindAirQualityByCode_HandleRequestCorrecly
- whenFindAirQualityByGeo_HandleRequestCorrecly

Here are some examples of Unit Test using mocks:



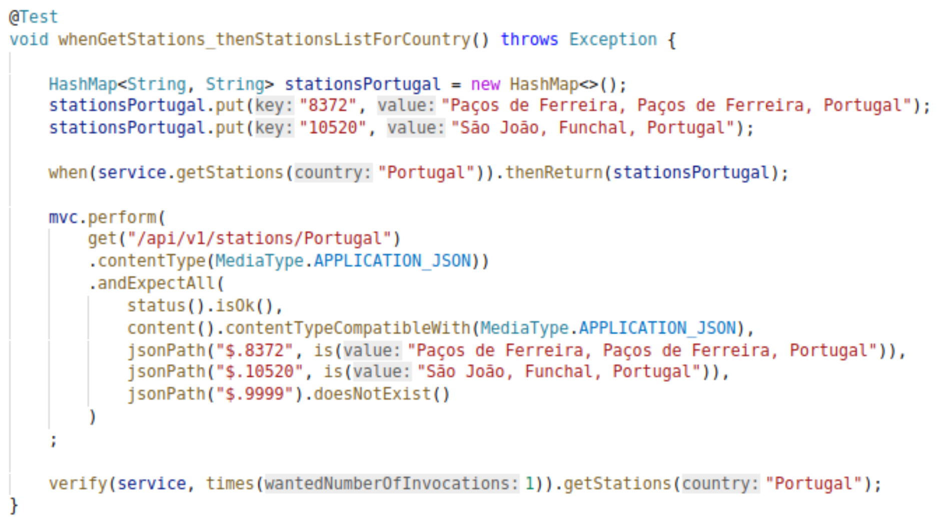
```
@Mock(lenient = true)
private RequestHandler requestHandler;

@InjectMocks
private LocalAirQualityService localAirQualityService;

@BeforeEach
void setUp() throws ParseException, URISyntaxException, IOException{
    String response0 = "{\"status\":\"ok\",\"data\":[{\"station\":{\"name\":\"Joaquim Magalhães, Faro, Portugal\"}},{\"stati
    JSONObject countriesJsonObject = (JSONObject)new JSONParser().parse(response0);
    Mockito.when(requestHandler.findCountries()).thenReturn(countriesJsonObject);

    String response1 = "{\"status\":\"ok\",\"data\":[{\"uid\":8383,\"station\":{\"name\":\"Joaquim Magalhães, Faro, Portugal
    JSONObject stationsJsonObject = (JSONObject)new JSONParser().parse(response1);
    Mockito.when(requestHandler.findStations(country: "Portugal")).thenReturn(stationsJsonObject);

    String response2 = "{\"status\":\"ok\",\"data\":{\"aqi\":28,\"idx\":8372,\"city\":{\"geo\":[41.274166666667,-8.376111111
    JSONObject airQualityJsonObject = (JSONObject)new JSONParser().parse(response2);
    Mockito.when(requestHandler.findAirQualityByCode(stationCode: "8383")).thenReturn(airQualityJsonObject);

    String response3 = "{\"status\":\"ok\",\"data\":{\"aqi\":28,\"idx\":8372,\"city\":{\"geo\":[41.274166666667,-8.376111111
    JSONObject airQuality3JsonObject = (JSONObject)new JSONParser().parse(response3);
    Mockito.when(requestHandler.findAirQualityByGeo(lat: "41.274166666667", lng: "-8.3761111111111")).thenReturn(airQuality3J

}

@Test
void whenGetCountries_thenReturnListOfCountries() throws URISyntaxException, IOException, ParseException {
    List<String> countriesList = new ArrayList<>();
    countriesList.addAll(Arrays.asList(...a: "Portugal", "Spain", "France"));

    assertTrue(localAirQualityService.getCountries().containsAll(countriesList));
    assertTrue(localAirQualityService.getCountries().containsAll(countriesList));

    verify(requestHandler, times(wantedNumberOfInvocations: 1)).findCountries();
}

@Test
void whenGetStationsByCountry_thenReturnsListOfStations() throws ParseException, URISyntaxException, IOException{
   HashMap<String, String> stations = new HashMap<>();
   stations.put(key: "8383", value: "Joaquim Magalhães, Faro, Portugal");
   stations.put(key: "10513", value: "Olivais, Lisboa, Portugal");
   stations.put(key: "8379", value: "Entrecampos, Lisboa, Portugal");

   assertEquals(stations, localAirQualityService.getStations(country: "Portugal"));
   assertEquals(stations, localAirQualityService.getStations(country: "Portugal"));

   verify(requestHandler, times(wantedNumberOfInvocations: 1)).findStations(anyString());
}
```

3.2.3 Integration Tests:

LocalAirQualityControllerTest.java

- whenGetCountries_thenReturnCountryList
- whenGetStations_thenStationsListForCountry
- whenGetAirQuality_thenReturnAirQualityFromStations
- whenGetAirQuality_thenReturnAirQualityFromGeolocation
- whenGetCache_theReturnCache

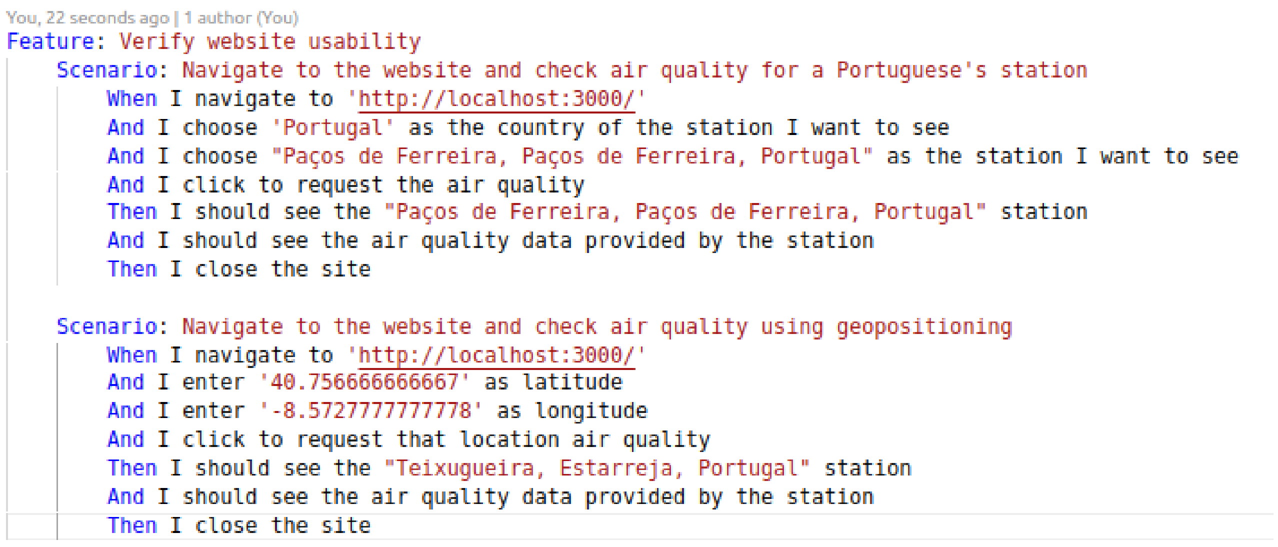
Here are some examples of Integration Tests:



```
@Test
void whenGetStations_thenStationsListForCountry() throws Exception {

    HashMap<String, String> stationsPortugal = new HashMap<>();
    stationsPortugal.put(key: "8372", value: "Paços de Ferreira, Paços de Ferreira, Portugal");
    stationsPortugal.put(key: "10520", value: "São João, Funchal, Portugal");

    when(service.getStations(country: "Portugal")).thenReturn(stationsPortugal);

    mvc.perform(
        get("/api/v1/stations/Portugal")
        .contentType(MediaType.APPLICATION_JSON))
        .andExpectAll(
            status().isOk(),
            content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON),
            jsonPath("$.8372", is(value: "Paços de Ferreira, Paços de Ferreira, Portugal")),
            jsonPath("$.10520", is(value: "São João, Funchal, Portugal")),
            jsonPath("$.9999").doesNotExist()
        )
    ;

    verify(service, times(wantedNumberOfInvocations: 1)).getStations(country: "Portugal");
}
```

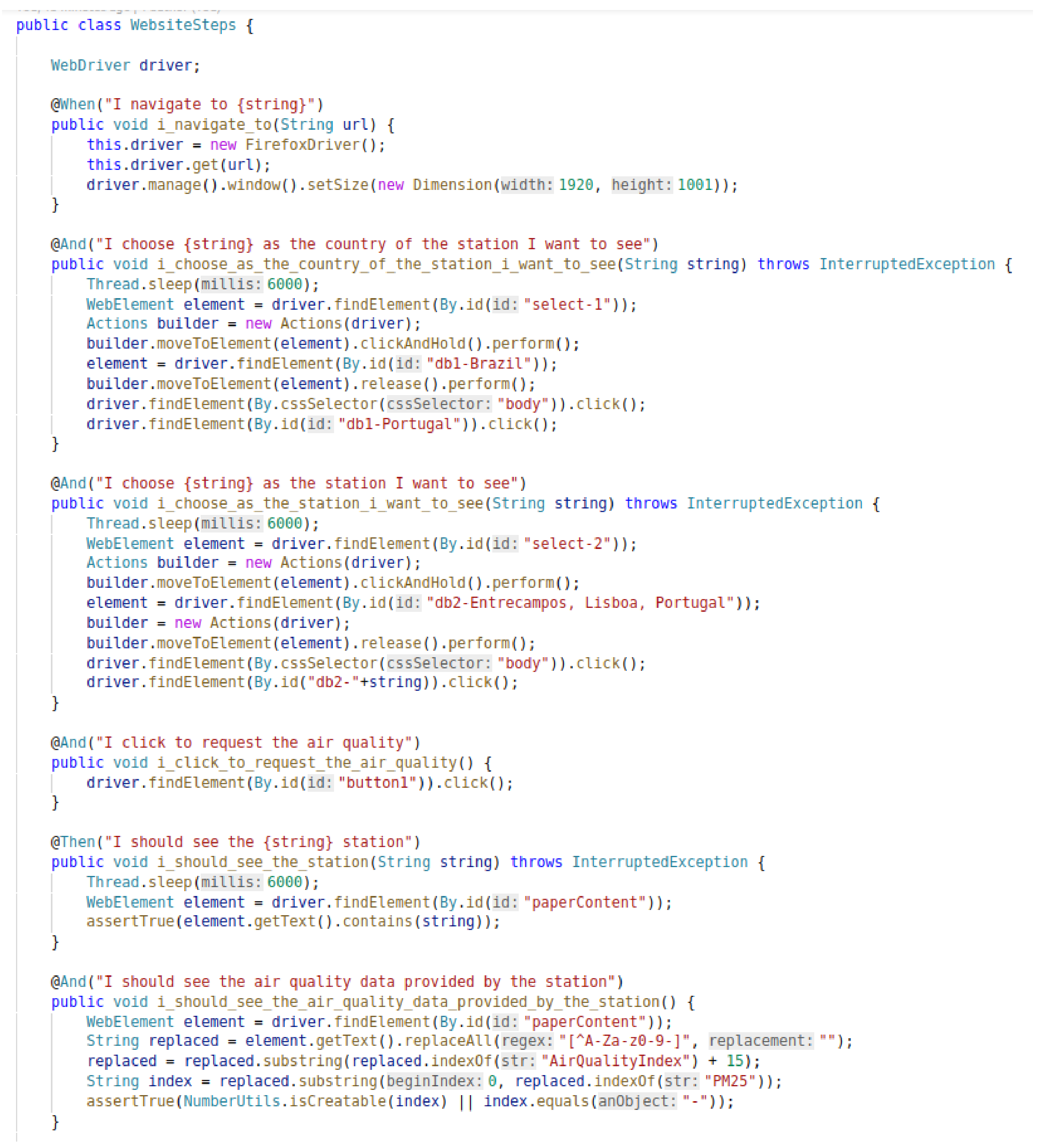
3.3 Functional testing

For functional testing, I used Selenium Web Driver, in this case Firefox Driver, along with Cucumber. These two scenarios most test frontend features:



```
You, 22 seconds ago | 1 author (You)
Feature: Verify website usability
    Scenario: Navigate to the website and check air quality for a Portuguese's station
        When I navigate to 'http://localhost:3000/'
        And I choose 'Portugal' as the country of the station I want to see
        And I choose "Paços de Ferreira, Paços de Ferreira, Portugal" as the station I want to see
        And I click to request the air quality
        Then I should see the "Paços de Ferreira, Paços de Ferreira, Portugal" station
        And I should see the air quality data provided by the station
        Then I close the site

    Scenario: Navigate to the website and check air quality using geopositioning
        When I navigate to 'http://localhost:3000/'
        And I enter '40.756666666667' as latitude
        And I enter '-8.5727777777778' as longitude
        And I click to request that location air quality
        Then I should see the "Teixugueira, Estarreja, Portugal" station
        And I should see the air quality data provided by the station
        Then I close the site
```

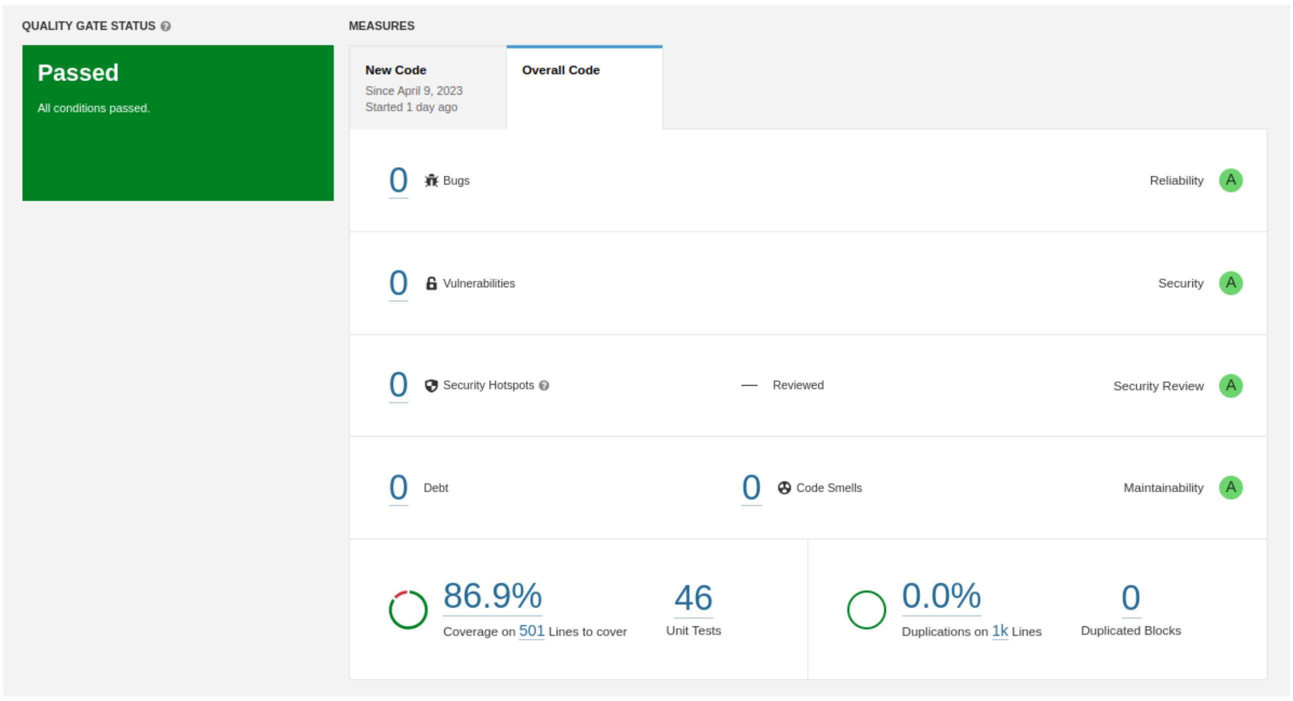
This is the implementation of the test for the first scenario:



```java
public class WebsiteSteps {

    WebDriver driver;

    @When("I navigate to {string}")
    public void i_navigate_to(String url) {
        this.driver = new FirefoxDriver();
        this.driver.get(url);
        driver.manage().window().setSize(new Dimension(width: 1920, height: 1001));
    }

    @And("I choose {string} as the country of the station I want to see")
    public void i_choose_as_the_country_of_the_station_i_want_to_see(String string) throws InterruptedException {
        Thread.sleep(millis: 6000);
        WebElement element = driver.findElement(By.id(id: "select-1"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).clickAndHold().perform();
        element = driver.findElement(By.id(id: "db1-Brazil"));
        builder.moveToElement(element).release().perform();
        driver.findElement(By.cssSelector(cssSelector: "body")).click();
        driver.findElement(By.id(id: "db1-Portugal")).click();
    }

    @And("I choose {string} as the station I want to see")
    public void i_choose_as_the_station_i_want_to_see(String string) throws InterruptedException {
        Thread.sleep(millis: 6000);
        WebElement element = driver.findElement(By.id(id: "select-2"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).clickAndHold().perform();
        element = driver.findElement(By.id(id: "db2-Entrecampos, Lisboa, Portugal"));
        builder = new Actions(driver);
        builder.moveToElement(element).release().perform();
        driver.findElement(By.cssSelector(cssSelector: "body")).click();
        driver.findElement(By.id("db2-"+string)).click();
    }

    @And("I click to request the air quality")
    public void i_click_to_request_the_air_quality() {
        driver.findElement(By.id(id: "button1")).click();
    }

    @Then("I should see the {string} station")
    public void i_should_see_the_station(String string) throws InterruptedException {
        Thread.sleep(millis: 6000);
        WebElement element = driver.findElement(By.id(id: "paperContent"));
        assertTrue(element.getText().contains(string));
    }

    @And("I should see the air quality data provided by the station")
    public void i_should_see_the_air_quality_data_provided_by_the_station() {
        WebElement element = driver.findElement(By.id(id: "paperContent"));
        String replaced = element.getText().replaceAll(regex: "[^A-Za-z0-9-]", replacement: "");
        replaced = replaced.substring(replaced.indexOf(str: "AirQualityIndex") + 15);
        String index = replaced.substring(beginIndex: 0, replaced.indexOf(str: "PM25"));
        assertTrue(NumberUtils.isCreatable(index) || index.equals(anObject: "-"));
    }
```

3.4 Code quality analysis

To analyze my code, I used SonarQube. I used it on the final product's code, as well as during the development, to figure out some scenarios/situations, save time and practice good coding habits.


QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code
Since April 9, 2023
Started 1 day ago

Overall Code

0 Bugs — Reliability A

0 Vulnerabilities — Security A

0 Security Hotspots — Reviewed — Security Review A

0 Debt — 0 Code Smells — Maintainability A

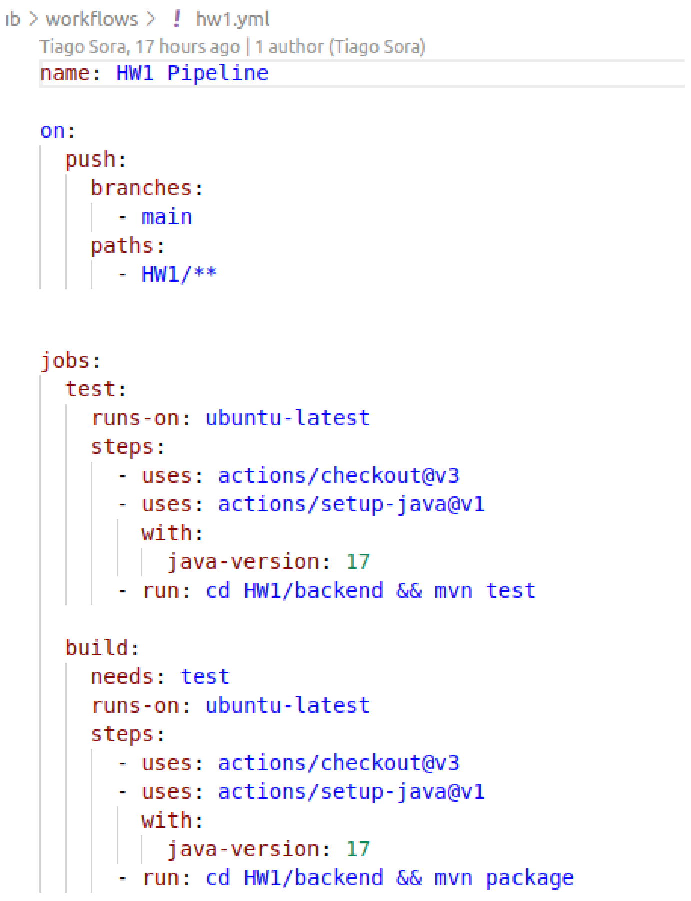
86.9% Coverage on 501 Lines to cover — 46 Unit Tests

0.0% Duplications on 1k Lines — 0 Duplicated Blocks

These are the results that came from the analysis: no bugs, no vulnerabilities, no security hotpots, and no code smells, grading the code a "A" in Reliability, Security, Security Review and Maintainability.
Talking about the code coverage, it was marked 86,9%, making it a sufficient percentage. SonarQube helped a lot to reach that mark. The remaining, I did not consider it was worth covering since they do not affect the behavior of the system that much.

3.5 Continuous integration pipeline

A continuous integration pipeline was developed and implemented with GitHub Actions. Its jobs are, basically, running all the tests created and, if this first step is successful, packaging the project. Once there is a push to the remote repository (in the 'main' branch), the docker-compose will make sure that those jobs are run.


ıb > workflows > ! hw1.yml
Tiago Sora, 17 hours ago | 1 author (Tiago Sora)

```yaml
name: HW1 Pipeline

on:
  push:
    branches:
      - main
    paths:
      - HW1/**


jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-java@v1
        with:
          java-version: 17
      - run: cd HW1/backend && mvn test

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-java@v1
        with:
          java-version: 17
      - run: cd HW1/backend && mvn package
```

I was not able to run the functional tests using GitHub Actions, but I was able to run the test perfectly on my machine.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/tiagosora/TQS_104142
Video demo	https://files.fm/u/aas8rjjth
CI/CD pipeline	https://github.com/tiagosora/TQS_104142/blob/main/.github/workflows/hw1.yml

Reference materials

https://aqicn.org/api/