

HW1: Mid-term assignment report

Leonardo dos Santos Flório [103360], 2023-04-10

1	Introduction	1
1.1	Overview of the work	1
1.2	Current limitations	2
1.3	Run the Application	2
2	Product specification	2
2.1	Functional scope and supported interactions	2
2.2	System architecture	4
2.3	UML Diagram	5
2.4	API for developers	5
3	Quality assurance	8
3.1	Overall strategy for testing	8
3.2	Unit and integration testing	8
3.3	Functional testing	13
3.4	Code quality analysis	16
4	References & resources	17

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 aim of this project was to develop a REST-API service and ensure its proper functioning through testing. The following tests were conducted as a part of this project:

- Unit Tests
- Unit Tests with Dependency Isolation Using Mocks
- Integration Tests on API
- Functional Tests

The Air Pollution product is a web application that provides air pollution data for a specific city. The backend is developed on Spring Boot, and the frontend is developed on React. The application displays air quality levels such as Good, Fair, Moderate, Poor, and Very Poor, along with the levels of polluting gases such as Carbon monoxide (CO), Nitrogen monoxide

(NO), Nitrogen dioxide (NO₂), Ozone (O₃), Sulphur dioxide (SO₂), Ammonia (NH₃), and particulates (PM_{2.5} and PM₁₀).

The application collects air pollution data for each city from the [OpenWeather Air Pollution API](#). However, to use this API, another API, the [OpenWeather Geocoding API](#), is used to convert the city into geographic locations. This process ensures accurate data collection for each city.

To improve the application's performance, a local cache has been implemented, which stores repeated calls for 3 minutes, making the retrieval of information faster. The cache implemented is a time-to-live cache, which implements TTL (Time-To-Leave). This type of cache ensures that, after the specified time has passed, all cities stored for more than that time are removed from the HashMap.

1.2 Current limitations

The product currently relies on two external APIs to present data to users. However, this approach has a significant drawback. If either of the two APIs goes down, the product cannot present any data. Therefore, implementing additional APIs is recommended to avoid such an issue.

In addition to the above recommendation, the product can benefit from a search feature that allows users to retrieve air quality data by day. This feature would enable users to view air quality trends for a specific city for the last n days. Graphical representation of this data will also be an added advantage for users to get a clear overview of the air quality data.

1.3 Run the Application

To locally deploy Air Pollution, navigate to the repository's root directory and run the following command: `docker-compose up`. This command will start the necessary processes to run the website.

Once everything is up and running, the Air Pollution website can be accessed by visiting <http://localhost:3000> in a web browser. This will display the website's homepage and allow users to search for air pollution data for their desired city.

2 Product specification

2.1 Functional scope and supported interactions

The product is structured with two pages: one for searching air pollution data and another for viewing the cache application. This approach divides the product's functionality into distinct sections and allows users to easily access the information they need.

Both pages of the product use a single page application approach for user interaction. This means that all interactions with the product occur on a single webpage, and the webpage does not need to reload to display new content. This approach makes user interactions with the product more intuitive and easier to navigate.

Users can search for air quality data for a specific city by name. This functionality allows users to quickly find air quality data for the city they are interested in.

AIR POLLUTION

CACHE

Air Pollution

City

Porto

Choose a country

Portugal

SEARCH AIR POLLUTION

cityName	Porto
country	PT
lat	41.1494512
lon	-8.6107884
airQuality	Moderate
co	198.6
no	0.15
no2	1.32
o3	128.75
so2	1.88
pm2.5	11.95
pm10	21.93
nh3	0.53

Users can check the cache details of the product. This functionality allows users to see information about the product's cache, such as the number of requests, number of hits and how much data is currently stored in the cache.

AIR POLLUTION

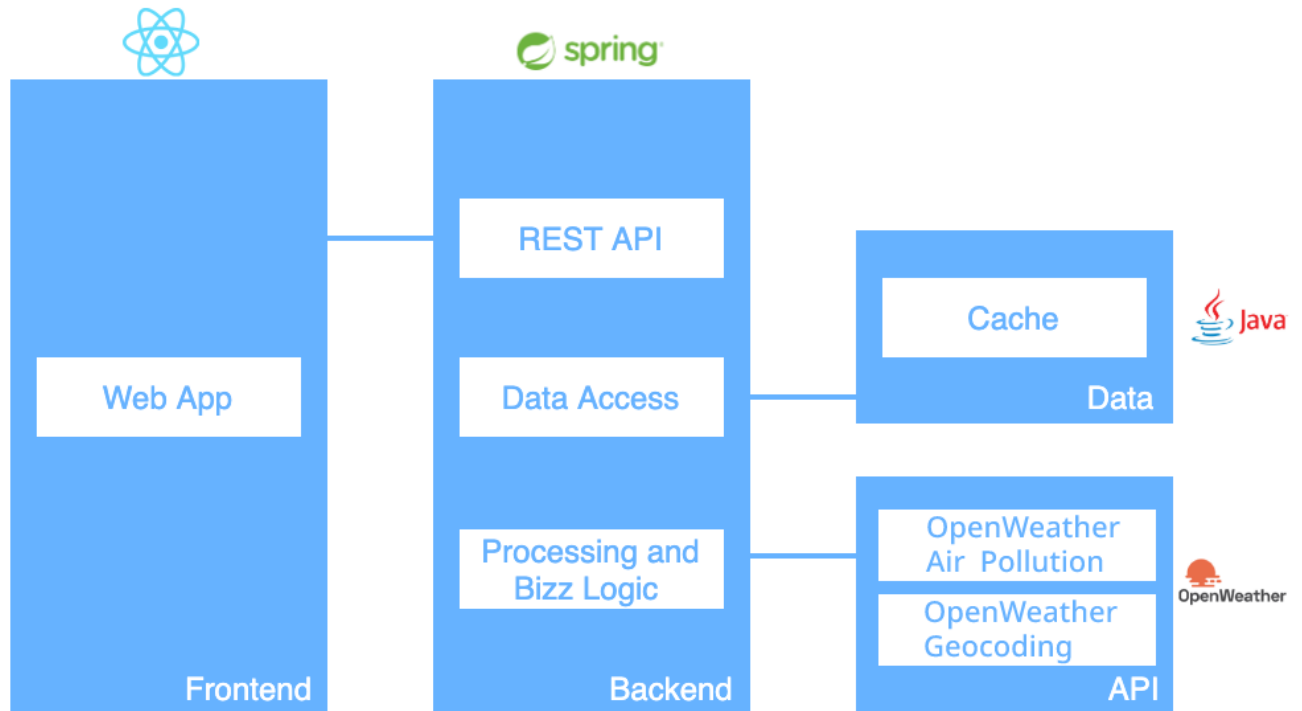
CACHE

Cache

Stat	Value
requestCount	1
hitCount	1

City Name	Country	Latitude	Longitude	Air Quality	CO	NO	NO2	O3	SO2	PM2.5	PM10	NH3
Porto	PT	41.1494512	-8.6107884	Moderate	198.6	0.15	1.32	128.75	1.88	11.95	21.93	0.53

2.2 System architecture

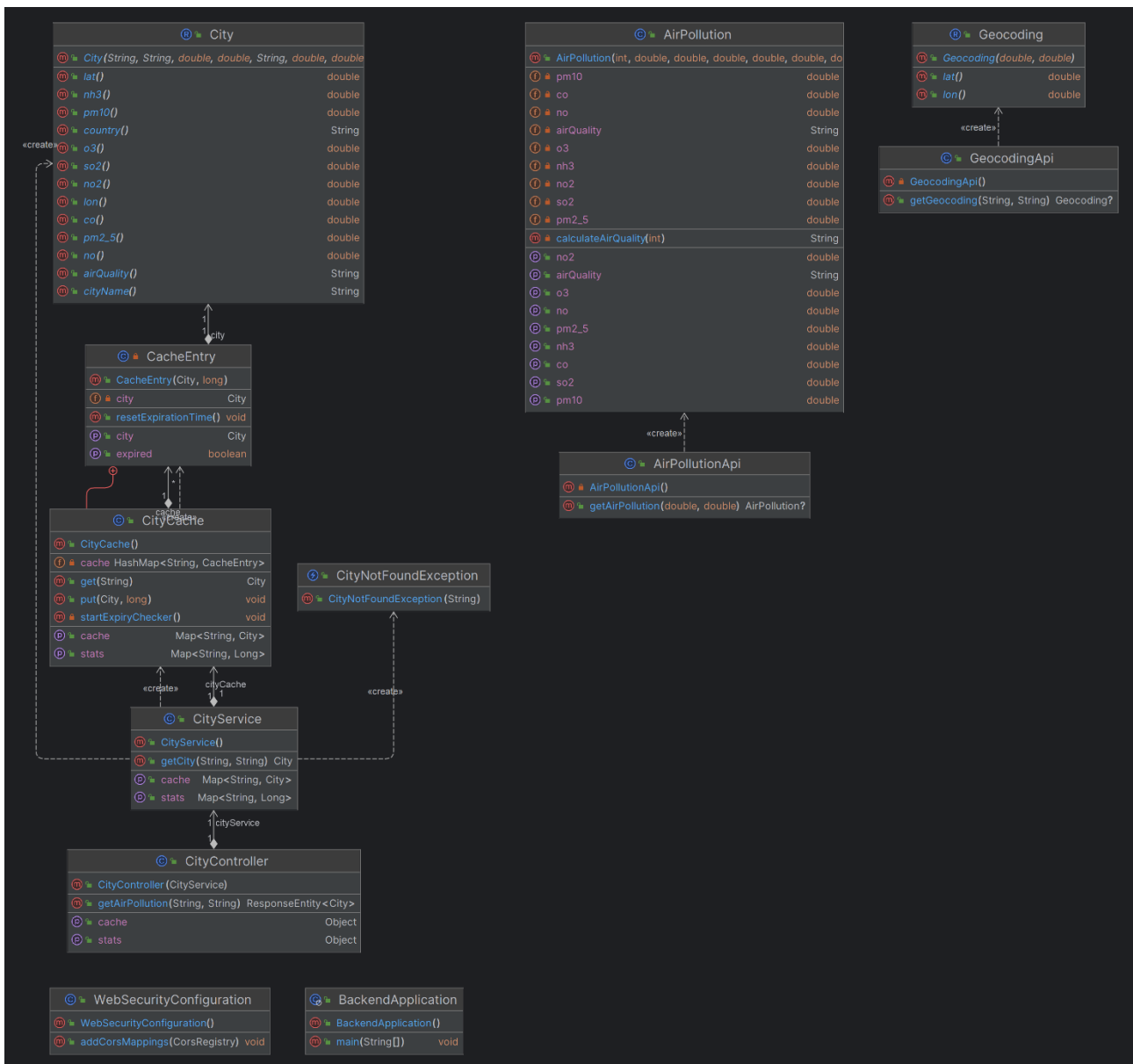


The frontend of the product was developed using React, a popular JavaScript library for building user interfaces.

The backend of the product was built using Spring Boot, a module of the Spring Framework.

The cache of the product was built using Java.

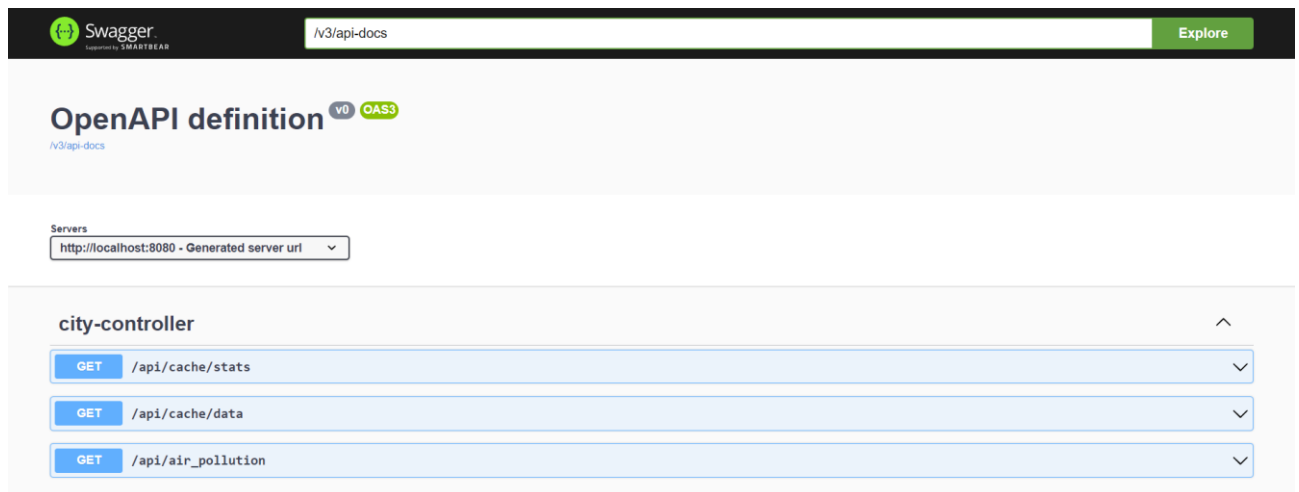
2.3 UML Diagram



2.4 API for developers

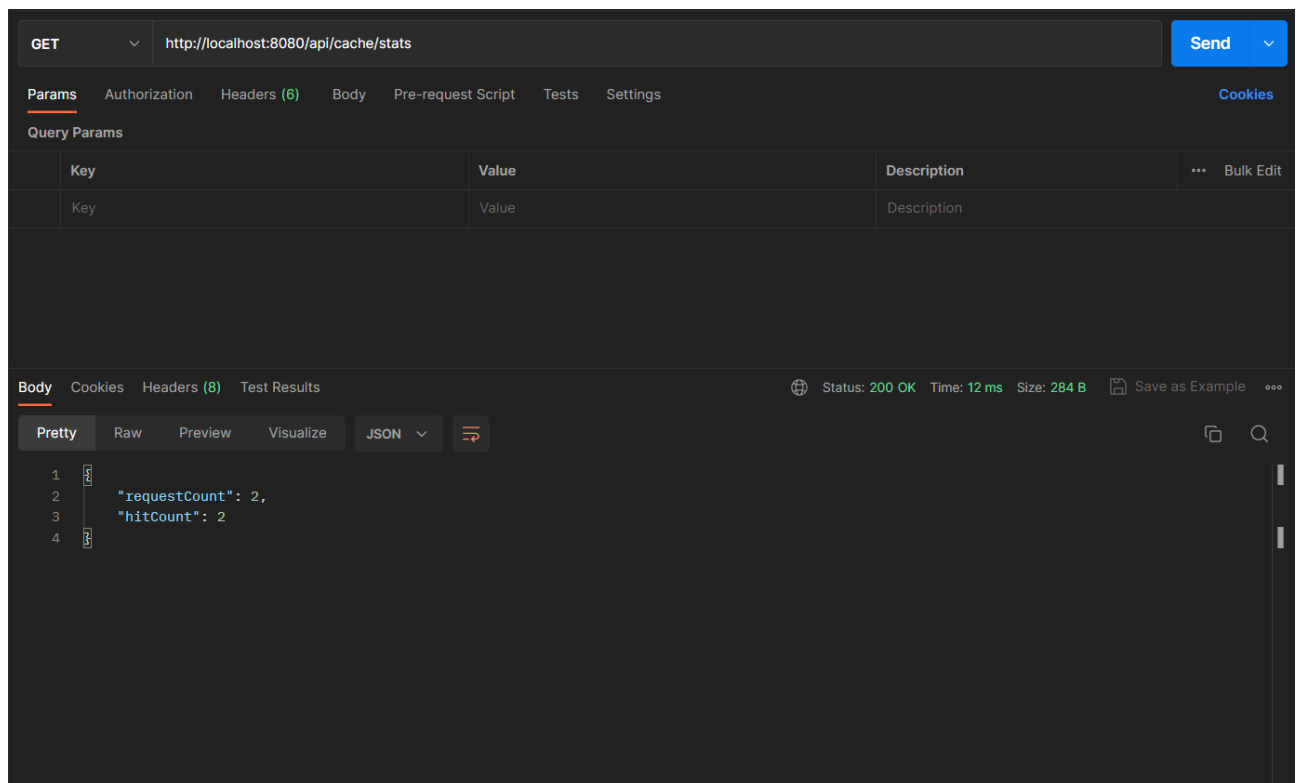
For documenting the API, Swagger2 was used. Swagger2 is a popular tool for documenting RESTful APIs. It allows developers to describe the structure of their APIs so that other developers can understand how to interact with them.

To access the API documentation page, the application needs to be up and running. After that, the documentation can be accessed at: <http://localhost:8080/swagger-ui/index.html#/>

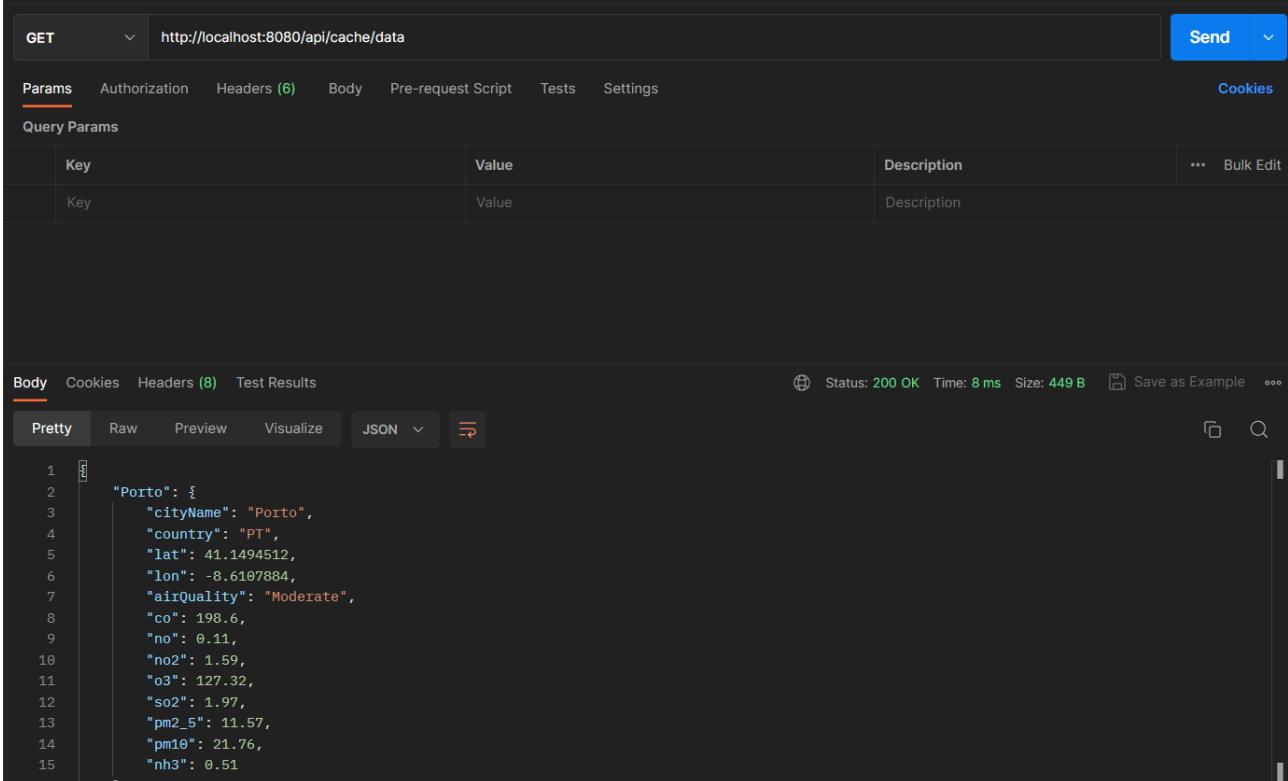


API endpoints:

- GET /api/cache/stats - this endpoint provides details about the cache used in the product. It does not require any parameters.



- GET /api/cache/data – this endpoint provides the data stored in the cache. It does not require any parameters.



GET `http://localhost:8080/api/cache/data` Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

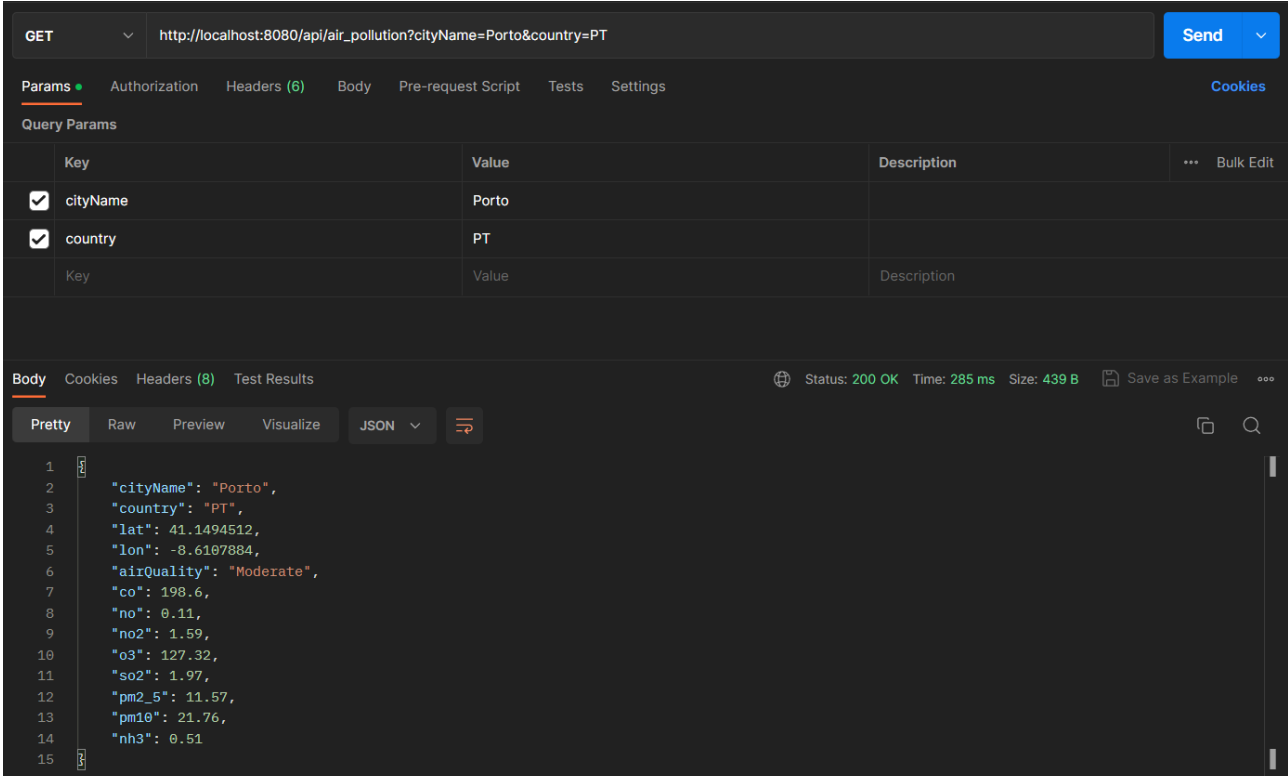
Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results Status: 200 OK Time: 8 ms Size: 449 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "Porto": {
3     "cityName": "Porto",
4     "country": "PT",
5     "lat": 41.1494512,
6     "lon": -8.6107884,
7     "airQuality": "Moderate",
8     "co": 198.6,
9     "no": 0.11,
10    "no2": 1.59,
11    "o3": 127.32,
12    "so2": 1.97,
13    "pm2_5": 11.57,
14    "pm10": 21.76,
15    "nh3": 0.51
16  }
17 }
```

- GET /api/air_pollution?cityName={cityName}&country={countryCode} – this endpoint provides air pollution data for a given city and country code.



GET `http://localhost:8080/api/air_pollution?cityName=Porto&country=PT` Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> cityName	Porto			
<input checked="" type="checkbox"/> country	PT			
Key	Value	Description		

Body Cookies Headers (8) Test Results Status: 200 OK Time: 285 ms Size: 439 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "cityName": "Porto",
3   "country": "PT",
4   "lat": 41.1494512,
5   "lon": -8.6107884,
6   "airQuality": "Moderate",
7   "co": 198.6,
8   "no": 0.11,
9   "no2": 1.59,
10  "o3": 127.32,
11  "so2": 1.97,
12  "pm2_5": 11.57,
13  "pm10": 21.76,
14  "nh3": 0.51
15 }
```

3 Quality assurance

3.1 Overall strategy for testing

The development of the product followed the Test-Driven Development (TDD) strategy. This means that software requirements were converted to test cases before the software was fully developed, and the software development was tracked by repeatedly testing the software.

3.2 Unit and integration testing

Regarding unit testing, several tests were implemented in the project. The following tests were conducted using Junit 5, Mockito and Spring Boot MockMVC to ensure the proper functioning of the code:

- AirQualityTest

```
@Test
void testCalculateAirQuality() {
    AirPollution ap = new AirPollution( index: 1, co: 1.0, no: 2.0, no2: 3.0, o3: 4.0, so2: 5.0, pm2_5: 6.0, pm10: 7.0, nh3: 8.0);
    assertEquals( expected: "Good", ap.getAirQuality());

    ap = new AirPollution( index: 2, co: 1.0, no: 2.0, no2: 3.0, o3: 4.0, so2: 5.0, pm2_5: 6.0, pm10: 7.0, nh3: 8.0);
    assertEquals( expected: "Fair", ap.getAirQuality());

    ap = new AirPollution( index: 3, co: 1.0, no: 2.0, no2: 3.0, o3: 4.0, so2: 5.0, pm2_5: 6.0, pm10: 7.0, nh3: 8.0);
    assertEquals( expected: "Moderate", ap.getAirQuality());

    ap = new AirPollution( index: 4, co: 1.0, no: 2.0, no2: 3.0, o3: 4.0, so2: 5.0, pm2_5: 6.0, pm10: 7.0, nh3: 8.0);
    assertEquals( expected: "Poor", ap.getAirQuality());

    ap = new AirPollution( index: 5, co: 1.0, no: 2.0, no2: 3.0, o3: 4.0, so2: 5.0, pm2_5: 6.0, pm10: 7.0, nh3: 8.0);
    assertEquals( expected: "Very Poor", ap.getAirQuality());

    ap = new AirPollution( index: 6, co: 1.0, no: 2.0, no2: 3.0, o3: 4.0, so2: 5.0, pm2_5: 6.0, pm10: 7.0, nh3: 8.0);
    assertNull(ap.getAirQuality());

    ap = new AirPollution( index: 0, co: 1.0, no: 2.0, no2: 3.0, o3: 4.0, so2: 5.0, pm2_5: 6.0, pm10: 7.0, nh3: 8.0);
    assertNull(ap.getAirQuality());
}
```

- AirPollutionApiTest

```
@Test
@DisplayName("Test the get method with a valid location")
void testGetAirPollution() { assertEquals(AirPollutionApi.getAirPollution( lat: 41.1494512, lon: -8.6107884)); }

Leonardo Flórido
@Test
@DisplayName("Test the get method with an invalid location")
void testGetAirPollutionInvalid() {
    assertThrows(RuntimeException.class, () -> {
        AirPollutionApi.getAirPollution( lat: 95, lon: 95);
    });
}
```


- GeocodingApiTest

```
@Test
@DisplayName("Test the get method with a valid city")
void getCityTest() {
    Geocoding geocoding = new Geocoding( lat: 41.1494512, lon: -8.6187884);

    Geocoding geocodingFromApi = GeocodingApi.getGeocoding( city: "Porto", country: "PT");
    assertNotNull(geocodingFromApi);

    assertEquals(geocoding.lat(), geocodingFromApi.lat());
    assertEquals(geocoding.lon(), geocodingFromApi.lon());
}

± Leonardo Flório

@Test
@DisplayName("Test the get method with a non-existent city")
void getCityNotFoundTest() { assertNotNull(GeocodingApi.getGeocoding( city: "Non-existent city", country: "Non-existent country")); }
```

- CityCacheTest

```
private static CityCache cityCache;

± Leonardo Flório
@BeforeEach
void setUp() {
    cityCache = new CityCache();
}

± Leonardo Flório
@Test
@DisplayName("Test the put method")
void putTest() {
    City city = new City( cityName: "Porto", country: "PT", lat: 41.1494512, lon: -8.6187884, airQuality: "Fair", co: 233.65, no: 1.34, no2: 5.01, o3: 83.69, so2: 2.98, pm2_5: 300000L);
    cityCache.put(city, timeToLive: 300000L);

    assertEquals( expected: 1, cityCache.getCache().size());
}

± Leonardo Flório
@Test
@DisplayName("Test the get method")
void getTest() {
    City city = new City( cityName: "Porto", country: "PT", lat: 41.1494512, lon: -8.6187884, airQuality: "Fair", co: 233.65, no: 1.34, no2: 5.01, o3: 83.69, so2: 2.98, pm2_5: 300000L);
    cityCache.put(city, timeToLive: 300000L);

    assertEquals(city, cityCache.get("Porto"));
}
```

```

@Test
@DisplayName("Test the expiration of the cache")
void expirationTest() throws InterruptedException {
    City city = new City( cityName: "Porto", country: "PT", lat: 41.1494512, lon: -8.6107884, airQuality: "Fair", co: 233.65, no: 1.34, no2: 5.01, o3: 83.69, so2: 2.98, pm2_5: 1.34, timeToLive: 1000L);
    cityCache.put(city, timeToLive: 1000L);

    Thread.sleep( millis: 2000L);

    assertNull(cityCache.get("Porto"));
}

Leonardo Flório
@Test
@DisplayName("Test the not expiration of the cache")
void notExpirationTest() throws InterruptedException {
    City city = new City( cityName: "Porto", country: "PT", lat: 41.1494512, lon: -8.6107884, airQuality: "Fair", co: 233.65, no: 1.34, no2: 5.01, o3: 83.69, so2: 2.98, pm2_5: 1.34, timeToLive: 30000L);
    cityCache.put(city, timeToLive: 30000L);

    Thread.sleep( millis: 2000L);

    assertEquals(city, cityCache.get("Porto"));
}

Leonardo Flório
@Test
@DisplayName("Test the stats of the cache")
void getStatsTest() {
    City city = new City( cityName: "Porto", country: "PT", lat: 41.1494512, lon: -8.6107884, airQuality: "Fair", co: 233.65, no: 1.34, no2: 5.01, o3: 83.69, so2: 2.98, pm2_5: 1.34, timeToLive: 30000L);
    cityCache.put(city, timeToLive: 30000L);

    cityCache.get("Porto");
    cityCache.get("Aveiro");

    assertEquals( expected: 2, cityCache.getStats().get("requestCount"));
    assertEquals( expected: 1, cityCache.getStats().get("hitCount"));
}

@Test
@DisplayName("Test the cache size")
void getCacheSizeTest() {
    City porto = new City( cityName: "Porto", country: "PT", lat: 41.1494512, lon: -8.6107884, airQuality: "Fair", co: 233.65, no: 1.34, no2: 5.01, o3: 83.69, so2: 2.98, pm2_5: 1.34, timeToLive: 30000L);
    cityCache.put(porto, timeToLive: 30000L);

    City aveiro = new City( cityName: "Aveiro", country: "PT", lat: 40.6442701, lon: -8.6455394, airQuality: "Fair", co: 233.65, no: 1.34, no2: 5.01, o3: 83.69, so2: 2.98, pm2_5: 1.34, timeToLive: 30000L);
    cityCache.put(aveiro, timeToLive: 30000L);

    assertEquals( expected: 2, cityCache.getCache().size());
}

```

- CityServiceTest

```
private CityService cityService;

Leonardo Flório
@BeforeEach
void setUp() { cityService = new CityService(); }

Leonardo Flório
@Test
@DisplayName("Test the get method")
void getCityTest() { assertNotNull(cityService.getCity( cityName: "Porto", country: "PT")); }

Leonardo Flório
@Test
@DisplayName("Test the get method with a non-existent city")
void getCityNotFoundTest() {
    assertThrows(RuntimeException.class, () -> {
        cityService.getCity( cityName: "Non-existent city", country: "Non-existent country");
    });
}

Leonardo Flório
@Test
@DisplayName("Test the getCache method")
void getCacheTest() { assertNotNull(cityService.getCache()); }

Leonardo Flório
@Test
@DisplayName("Test the getStats method")
void getStatsTest() { assertNotNull(cityService.getStats()); }
```

- CityControllerTest

```
@WebMvcTest(CityController.class)
class CityControllerTest {
    @Autowired
    private MockMvc mvc;

    @MockBean
    private CityService service;

    // Leonardo Flório
    @Test
    @DisplayName("Test the GET /api/air_pollution endpoint with valid city name")
    void whenGetCityByNameAndCountry_thenReturnJsonArray() throws Exception {
        String cityName = "Porto";
        String country = "PT";

        Geocoding geocoding = GeocodingApi.getGeocoding(cityName, country);
        assertNotNull(geocoding);

        AirPollution airPollution = AirPollutionApi.getAirPollution(geocoding.lat(), geocoding.lon());
        assertNotNull(airPollution);

        City porto = new City(cityName, country, geocoding.lat(), geocoding.lon(), airPollution.getAirQuality(), airPollution.getCo(), airPollution.getNo(), airPollution.getO3(), airPollution.getPm10(), airPollution.getPm25(), airPollution.getSo2(), airPollution.getTsp());

        when(service.getCity(cityName, country)).thenReturn(porto);

        mvc.perform(get(uriTemplate: "/api/air_pollution?cityName=" + cityName + "&country=" + country)
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("cityName", is(cityName)))
            .andExpect(jsonPath("country", is(country)));
    }

    // Leonardo Flório
    @Test
    @DisplayName("Test the GET /api/air_pollution endpoint with invalid city name")
    void whenGetCityByNameAndCountry_thenReturnError() throws Exception {
        String cityName = "New York";
        String country = "PT";

        when(service.getCity(cityName, country)).thenReturn(null);

        mvc.perform(get(uriTemplate: "/api/air_pollution?cityName=" + cityName + "&country=" + country)
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound());
    }

    // Leonardo Flório
    @Test
    @DisplayName("Test the GET /api/cache/stats endpoint")
    void whenGetStats_thenReturnJsonArray() throws Exception {
        mvc.perform(get(uriTemplate: "/api/cache/stats")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk());
    }

    // Leonardo Flório
    @Test
    @DisplayName("Test the GET /api/cache/data endpoint")
    void whenGetCache_thenReturnJsonArray() throws Exception {
        mvc.perform(get(uriTemplate: "/api/cache/data")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk());
    }
}
```

- BackendApplicationTest

```
@SpringBootTest
class BackendApplicationTest {
    @Autowired
    private ApplicationContext context;

    Leonardo Flório
    @Test
    void contextLoads() { assertNotNull(context); }
}
```

3.3 Functional testing

For functional testing, the Selenium Web Driver in combination with Cucumber was used.

- Feature

```
Feature: City

  Scenario: Air Pollution
    Given the user is on the homepage
    When the user searches air pollution for "Porto"
    And the user selects "Portugal" as country
    And the user clicks on search button
    Then the user should see the air pollution for "Porto"

    Given the user is on the cachepage
    Then the user should see the air pollution for "Porto" in the cache
```

- CitySteps

```
private WebDriver driver;

Leonardo Flórido
@Given("the user is on the homepage")
public void the_user_is_on_the_home_page() {
    WebDriverManager.edgedriver().setup();
    EdgeOptions options = new EdgeOptions();
    options.addArguments("--remote-allow-origins=*");
    driver = new EdgeDriver(options);
    driver.manage().window().setSize(new Dimension(width: 1552, height: 849));
    driver.get("http://localhost:3000");
}

Leonardo Flórido
@When("the user searches air pollution for {string}")
public void the_user_searches_air_pollution_for(String city) {
    driver.findElement(By.cssSelector(".MuiInput-input")).click();
    driver.findElement(By.cssSelector(".MuiInput-input")).sendKeys(city);
    driver.findElement(By.cssSelector(".MuiInput-input")).sendKeys(Keys.ENTER);
}

Leonardo Flórido
@And("the user selects {string} as country")
public void the_user_selects_as_country(String country) {
    driver.findElement(By.id("country-autocomplete")).sendKeys(country);
    driver.findElement(By.id("country-autocomplete")).sendKeys(Keys.ENTER);
    driver.findElement(By.id("country-autocomplete-option-0")).click();
}
```

```

@And("the user clicks on search button")
public void the_user_clicks_on_search_button() {
    {
        WebElement element = driver.findElement(By.cssSelector(".MuiButton-contained > .MuiButton-label"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
    }
    driver.findElement(By.cssSelector(".MuiButton-contained > .MuiButton-label")).click();
    {
        WebElement element = driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, xOffset: 0, yOffset: 0).perform();
    }
    try {
        Thread.sleep( millis: 2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

+ Leonardo Flório
@Then("the user should see the air pollution for {string}")
public void the_user_should_see_the_air_pollution_for(String city) {
    {
        List<WebElement> elements = driver.findElements(By.cssSelector(".MuiTableRow-root:nth-child(1) > .MuiTableCell-root:nth-child(2)"));
        assert (elements.size() > 0);
    }
}

+ Leonardo Flório
@Given("the user is on the cachepage")
public void the_user_is_on_the_cachepage() {
    driver.findElement(By.cssSelector(".MuiButtonBase-root:nth-child(2) > .MuiButton-label")).click();
    driver.get("http://localhost:3000/cache");
}

@Then("the user should see the air pollution for {string} in the cache")
public void the_user_should_see_the_air_pollution_for_in_the_cache(String city) {
    {
        List<WebElement> elements = driver.findElements(By.cssSelector(".MuiPaper-root:nth-child(3) .MuiTableBody-root .MuiTableCell-root:nth-child(1)"));
        assert (elements.size() > 0);
    }
    driver.quit();
}

```

- CityTest

```

@Suite
@IncludeEngines("cucumber")
@SelectClasspathResource("city")
@ConfigurationParameter(key = PLUGIN_PROPERTY_NAME, value = "pretty")
@ConfigurationParameter(key = GLUE_PROPERTY_NAME, value = "tqs")
public class CityTest {
}

```

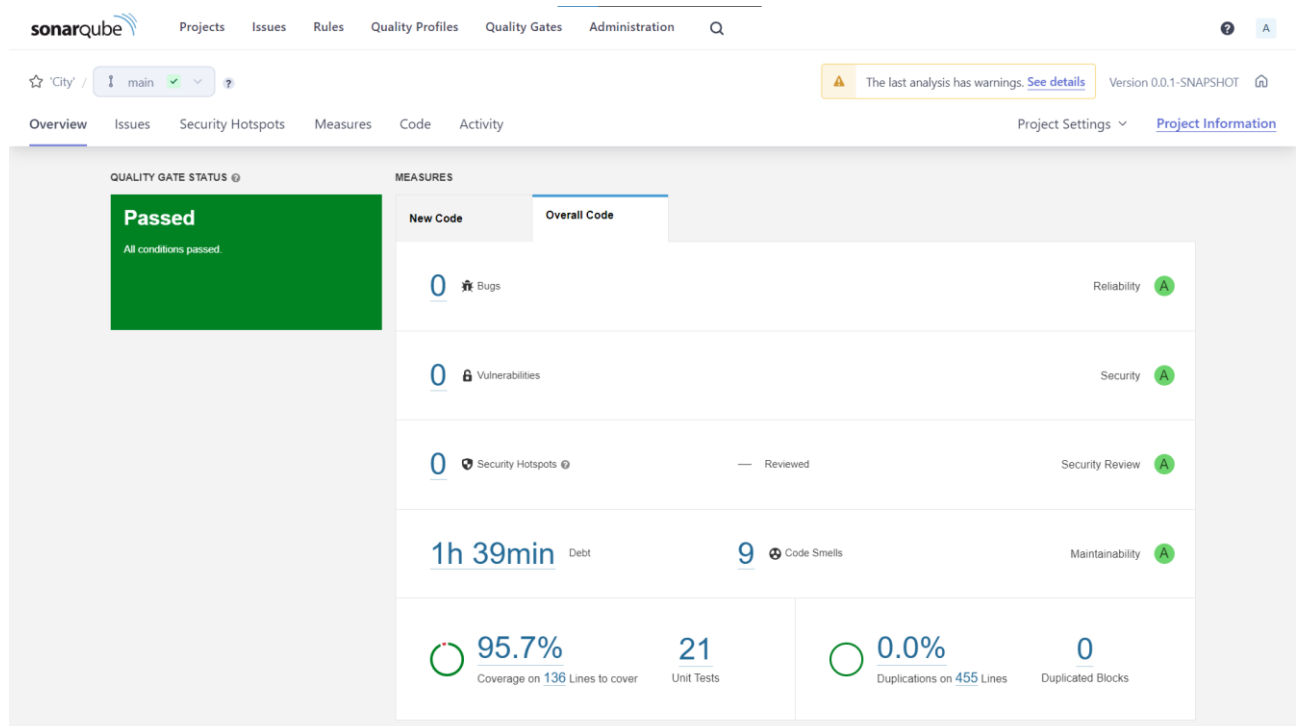
During testing, an issue was discovered where the first request required calling an external API, resulting in a delay. Furthermore, Selenium tests presented errors indicating that information was not found. This was caused by the external API information not being visible on the webpage. To address this issue, the following code was implemented:

```
try {
    Thread.sleep( millis: 2000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

3.4 Code quality analysis

The static code analysis of the project was conducted using SonarQube, which enabled us to assess multiple aspects of the project's code quality:

- Reliability - The analysis revealed the presence of bugs in the project's codebase, indicating potential issues that may arise during execution.
- Security - The project's code was also assessed for vulnerabilities and security hotspots, which may pose a threat to the security of the system. These findings will help in identifying potential security risks and taking appropriate measures to address them.
- Maintainability - The analysis also provided insights into the code's maintainability, indicating the estimated time required to fix all code smells. Additionally, the report highlighted the code smells, areas of the code that may require refactoring for better maintainability.
- Coverage - The analysis assessed the code's coverage, including the percentage of coverage achieved by the tests. This information provides a better understanding of the test coverage and areas that may require additional testing.
- Duplications - The analysis also identified the presence of duplicated blocks in the codebase, indicating areas that require consolidation to improve maintainability.



The Quality Gate status of the project is "Passed," indicating that it has met all the requirements for quality, functionality, and security. The project has undergone rigorous testing, analysis, and evaluation, and has been deemed ready for deployment in a production environment.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	leo-dsf/TQS_103360: TQS Labs. (github.com)
Video demo	TQS_103360/demo.mp4 at main · leo-dsf/TQS_103360 (github.com)

Reference materials

[OpenWeather Air Pollution API](#)

[OpenWeather Geocoding API](#)