

HW1: Mid-term assignment report

Vicente Manuel Andrade Barros [97787], v2023-04-10

1.1	Overview of the work.....	1
1.2	Current limitations.....	1
2.1	Functional scope and supported interactions.....	2
2.2	System architecture.....	2
2.3	API for developers	3
2.3.1	Problem Domain	3
2.3.2	Cache Usage	3
3.1	Overall strategy for testing	3
3.2	Unit and integration testing.....	4
3.3	Functional testing	5
3.4	Code quality analysis.....	6
3.5	Continuous integration pipeline	8

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 developed application was thought from the beginning to be a straightforward way to see the current and forecast air quality from any city from the world, with some metrics such as concentration of Carbon Monoxide (CO), Nitrogen Monoxide (NO), Sulphur Dioxide SO₂, Ozone (O₃) and Fine particulate matter (PM_{2.5}).

1.2 Current limitations

Current known limitations:

- It was not possible to use Swagger to automatically document the API endpoints since Swagger does not work with Spring Boot 3
- All work was developed locally without deployment (for frontend or backend). For the developed frontend, it was easy to find a deployment service. However, for the backend, the most recommended deployment service was Heroku, which removed its free tier.
- For Selenium test to work, it was needed to add the annotation @CrossOrigin. However, the SonarCube claims, with reason, that is a high severity security issue and so the committed version has the annotation commented.

2 Product specification

2.1 Functional scope and supported interactions

This is a general-purpose application – for every person who wants to check, in a simple and concise way, how the Air Quality is in their local area or around the world.

Usage Scenario	Synopses
Check the current Air Quality from a city	The user opens the Air Quality application and realizes that the current option is already selected and just needs to provide the city name and click the search button. After that, city's information (name and coordinates) will appear, including a table with all the metrics being provided by the API.
Check the Air Quality forecast from a city	The user changes the option to Forecast from the default one, provides a city and clicks to search. The same kind of city data will appear and a list of 5 tables (stats) will appear with the option of Load More if needed.
Check the Current AirQuality from a specific API	The user changes the option to the preferred API (currently NinjaAPI and OpenWeatherAPI) and provides a city name. A similar result to the first scenario will appear.
Check API statistics	The user checks the metrics of the API that is being used to feed the web app by selecting the statistics option. The statistics of the current Air Quality Cache and Forecast Cache will be presented.

2.2 System architecture

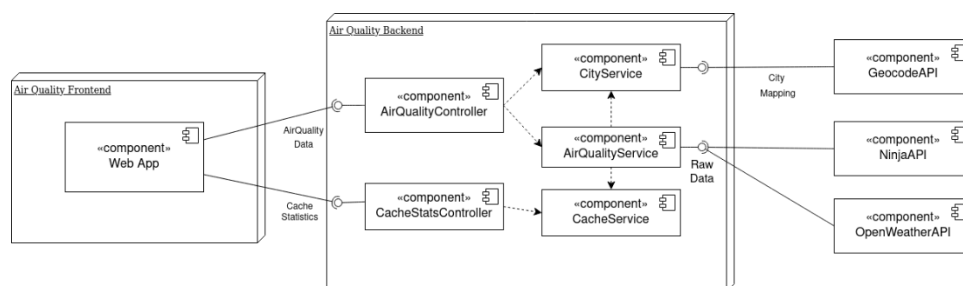


Figure 1 - Components Diagram

The solution achieved was developed with Spring Boot for the backend part using all its features. To fetch data from the external APIs, Spring Boot's WebClient and object mapping in a primitive stage of the project were used. Jackson was used since is built in, but when implementing the second API, Jayway's JsonPath provided an easier way to deserialize the information.

For the presentation layer, React.js was used to develop the user interface with the help of TailwindCSS and DaisyUI. Axios was used to fetch the data from the backend.

2.3 API for developers

2.3.1 Problem Domain

- GET /api/quality?city={city} - get the City Information and the current Air Quality from a specific city. This endpoint tries to fetch first from the cache and then it tries from the available APIs. It returns an error message if it cannot perform the request.
- GET /api/forecast?city={city} - get the City Information and a list of the Air Quality forecast. It works as the previous endpoint.
- GET /api/city?city={city} - get only the City Information.
- GET /api/openweather?city={city} - get the City Information and the current Air Quality from the Open Weather API.
- GET /api/ninja?city={city} - get the City Information and the current Air Quality from the Open Weather API.

2.3.2 Cache Usage

- GET /api/cache/stats - get the statistics for the current Air Quality Cache and forecast Cache – hits, misses, requests and puts.

3 Quality assurance

3.1 Overall strategy for testing

The first instinct was to develop the whole application and then do the tests. After several issues in the architecture, mainly because of high coupling of the layers of the application, the best approach was to develop the tests while developing the application – this method proved to be more efficient.

Some of the tools used were the ones that were presented in the course such as JUnit 5, AssertJ, Hamcrest, Mockito, Rest Assured and Selenium.

homework

homework













Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
pt.ua.deti.tqs.data		48%		2%	44	74	20	73	27	57	0	6
pt.ua.deti.tqs.cache		72%		64%	12	41	16	64	9	34	0	5
pt.ua.deti.tqs.service		96%		87%	5	30	4	78	2	18	0	3
pt.ua.deti.tqs		37%		n/a	1	2	2	3	1	2	0	1
pt.ua.deti.tqs.apis		98%		91%	3	24	2	70	2	18	0	3
pt.ua.deti.tqs.controller		100%		95%	1	19	0	55	0	9	0	2
Total	377 of 1,726	78%	43 of 104	58%	66	190	44	343	41	138	0	20

Figure 2 - JaCoCo Overview

Overall Coverage Summary			
Package	Class, %	Method, %	Line, %
all classes	100% (29/29)	79.2% (25/32)	87.3% (28/32)
Coverage Breakdown			
Package	Class, %	Method, %	Line, %
pt.ua.delft.api	100% (1/1)	50% (1/2)	50% (1/2)
pt.ua.delft.api.api	100% (3/3)	86.7% (13/15)	97.1% (88/91)
pt.ua.delft.api.cache	100% (5/5)	81.2% (26/32)	73.1% (38/52)
pt.ua.delft.api.controller	100% (2/2)	100% (9/9)	100% (55/55)
pt.ua.delft.api.data	100% (8/8)	68.2% (30/44)	71.4% (50/70)
pt.ua.delft.api.service	100% (3/3)	88.9% (16/18)	94.6% (70/74)

Figure 3 - IntelliJ Coverage Tests Overview

3.2 Unit and integration testing

Unit tests were heavily used to ensure the expected behavior of the caches (Volatile Cache and Persistent Cache) and the expected behavior of the API fetchers (GeocodeAPI, NinjaAPI and OpenWeather).

One of the main types of tests which proved that the first implementation was lacking was the Service level tests with dependency Isolation. These tests helped to decouple the services from the APIs – in a first attempt, it was impossible to mock the APIs.

The Controllers also used Mock testing to check if they were fulfilling their purpose – this exposed another implementation problem: the logic of the services was coupled in the controller.

The integration tests were an important part when developing the system because of their capability to automatically simulate real usage. These tests were used to check the controllers – in other words, to check if the information and the logic behind them were working properly.

```

± VicenteBarros
@BeforeEach
public void setUp() {
    city = new City( name: "Porto", latitude: 41.1496, longitude: -8.6109, displayName: "Porto, Portugal");
    airQuality = new AirQuality( aqi: 50, Arrays.asList(1, 2, 3, 4, 5, 6));
    forecast = Arrays.asList(airQuality, airQuality, airQuality, airQuality, airQuality);
    when(api1.getApiName()).thenReturn( value: "NinjaAPI");
    when(api2.getApiName()).thenReturn( value: "OpenWeatherAPI");
    airQualityService = new AirQualityService(new ApiQuality[]{api1, api2}, cacheService);
}

± VicenteBarros
@Test
void whenGetAirQualityFromTheFirstApi_thenReturnAirQuality() {
    when(api1.getQuality(city)).thenReturn(airQuality);
    when(api2.getQuality(city)).thenReturn( value: null);

    assertThat(airQualityService.getAirQuality(city), is(equalTo(airQuality)));

    verify(api1, times( wantedNumberOfInvocations: 1)).getQuality(city);
    verify(api2, times( wantedNumberOfInvocations: 0)).getQuality(city);
    verify(cacheService, times( wantedNumberOfInvocations: 1)).addAirQuality(city, airQuality);
    verify(cacheService, times( wantedNumberOfInvocations: 1)).hasCityQuality(city);
    verify(cacheService, times( wantedNumberOfInvocations: 0)).getAirQuality(city);
}

```

Figure 4 - Service Tests with mocked APIs

```

± vcentebarros
@Test
void whenGetQualityFromCityThenReturnsAirQuality() {

    RestAssured.when() RequestSender
        .get( s: "/api/quality?city=Porto") Response
        .then() ValidatableResponse
        .statusCode( i: 200)
        .body( s: "city.name", equalTo( operand: "porto"))
        .body( s: "city.displayName", is( value: "Porto, Portugal"))
        .body( s: "city.latitude", is( value: 41.149452F))
        .body( s: "city.longitude", is( value: -8.610788F))
        .body( s: "data.aqi", is(instanceOf(Number.class)))
        .body( s: "data.pm10", is(instanceOf(Number.class)))
        .body( s: "data.pm25", is(instanceOf(Number.class)))
        .body( s: "data.no2", is(instanceOf(Number.class)))
        .body( s: "data.so2", is(instanceOf(Number.class)))
        .body( s: "data.o3", is(instanceOf(Number.class)))
        .body( s: "data.co", is(instanceOf(Number.class)));
}

```

Figure 5 - Integration Test with RestAssured

```

± vcentebarros
@Test
void testGetWithExpiredEntry() {
    cache.put( key: "key", value: 1234, ttlSeconds: 1);
    await().atMost( timeout: 10, TimeUnit.SECONDS).until(() -> cache.get("key") == null);
    assertNull(cache.get("key"));
}

```

Figure 6 - Unit test of the Cache with awaitility library

3.3 Functional testing

The functional tests were made in a straightforward way due to the complication of using them in a Spring Boot application. Some of the issues faced included the lack of detection of the Web Drivers and problems with the asynchronous data fetching from the frontend. The test was made with Selenium, in a first attempt with the browser extension and then programmatically by simulating all the scenarios described before. Then, while using the Page Object Pattern, the same tests were developed with more detail in the types of assertions made – in this case, it was asserted that the load time is less when clicked the second time because the cache had stored the value.

```

18 @BeforeEach
19 > void setup() { WebDriverManager.firefoxdriver().setup(); }
    ± vicentebarras *
22 @Test
23 void fetchAirQuality(FirefoxDriver driver) {
24
25     HomePageObject homePage = new HomePageObject(driver);
26
27     assertThat(homePage.getButtonValue()).isEqualTo( expected: "FIND AIR QUALITY");
28
29     assertThat(homePage.getOptions()).contains("Current");
30     homePage.seachCity( option: "Lisbon");
31     assertThat(homePage.getButtonValue()).isEqualTo( expected: "LOADING...");
32     WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
33     wait.until(webDriver -> homePage.getDisplayName().equals("Lisboa, Portugal"));
34     wait.until(webDriver -> homePage.getCoordinates().equals("(38.7077507,-9.1365919)"));
35
36     Integer loadTime = homePage.getLoadTime();
37     assertThat(loadTime).isNotNull().isInstanceOf(Integer.class).isPositive();
38     homePage.submitClick();
39     wait.until(webDriver -> homePage.getLoadTime() < loadTime);
40     assertThat(homePage.getLoadTime()).isLessThan(loadTime);
41     ...

```

Figure 7 - Selenium Page Object Text

3.4 Code quality analysis

For the static code analysis, Sonar Cube dockerized was used to inspect the code on commit. Furthermore, while developing, the IntelliJ Idea's Sonar Lint Plugin prevented a lot of code smells and security issues when analyzing with Sonar Cube. Some of the most predominant code smells were string repetition and, in the tests, the replacement of methods such as *hasSize(0)* to *isEmpty()*, making the tests more legible.

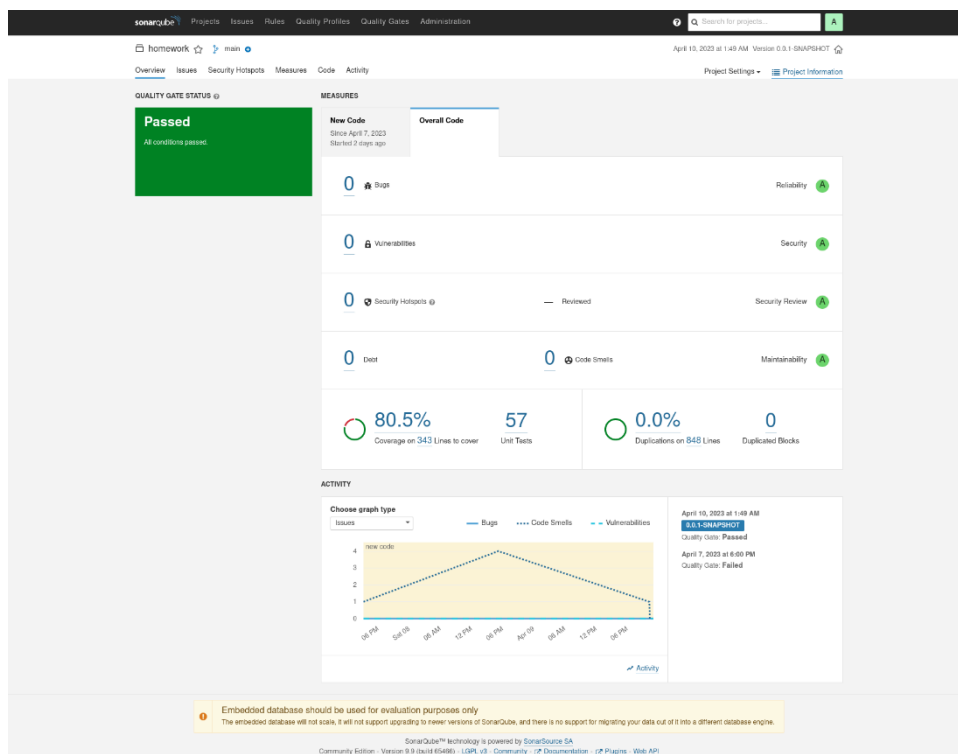


Figure 8 - SonarCube Overview

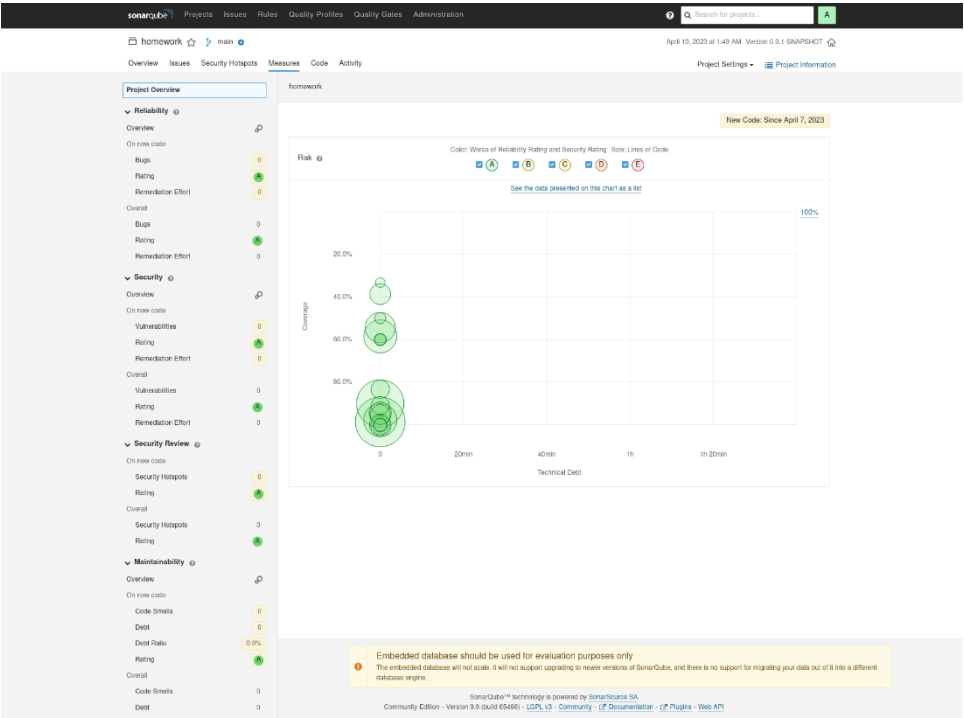


Figure 9 - Sonar Cube Measure Tabs

3.5 Continuous integration pipeline

A simple CI pipeline was created with Github Actions that runs all the tests, except the Integration Tests (marked with IT at the end of the name), validating all the code that was developed and committed to the main branch.



Figure 10 - Commit with tick saying that the CI worked.

A screenshot of the 'All workflows' page in GitHub Actions. It shows a list of 9 workflow runs. The first five runs are visible, each with a status icon (green checkmark for success, red X for failure), a title, a description, the branch (all are 'main'), and the time since completion. The sixth run is partially visible.

9 workflow runs		Event ▾	Status ▾	Branch ▾	Actor ▾
✓	fix test	Java CI with Maven #9: Commit ae90aeb pushed by v1centebarros	main	2 hours ago 48s	...
✗	Test CI	Java CI with Maven #8: Commit 20ee1f0 pushed by v1centebarros	main	2 hours ago 42s	...
✓	Removed Cross for the score	Java CI with Maven #7: Commit 5c93ef0 pushed by v1centebarros	main	10 hours ago 50s	...
✓	API tests	Java CI with Maven #6: Commit 1cf3f92 pushed by v1centebarros	main	11 hours ago 1m 6s	...
✓	fixed name	Java CI with Maven #5: Commit 73ceed9 pushed by v1centebarros	main	11 hours ago 1m 10s	...
✓	Rest Assured Tests	Java CI with Maven #4: Commit a80fbdd pushed by v1centebarros	main	11 hours ago 1m 5s	...

Figure 11 - Some CI workflows done.

```
name: Java CI with Maven

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    name: Build
    runs-on: ubuntu-latest

    defaults:
      run:
        working-directory: HW1/backend

    steps:
      - uses: actions/checkout@v3
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: maven
      - name: Build with Maven
        run: mvn -B package --file pom.xml
```

Figure 12 - Github Workflow Config

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/v1centebarros/TQS_97787
Video demo	< short video demonstration of your solution; consider including in the Git repository>
QA dashboard (online)	Not applicable
CI/CD pipeline	https://github.com/v1centebarros/TQS_97787/actions
Deployment ready to use	Not applicable

Reference materials

- [1] baeldung, “Baeldung,” 2 9 2022. [Online]. Available: <https://www.baeldung.com/spring-5-webclient>. [Accessed 8 4 2023].
- [2] Baeldung, “Baeldung,” 2 9 2022. [Online]. Available: <https://www.baeldung.com/spring-5-webclient>. [Acedido em 8 4 2023].
- [3] Baeldung, “Baeldung,” 2 9 2022. [Online]. Available: <https://www.baeldung.com/spring-mocking-webclient>. [Acedido em 9 4 2023].
- [4] Geek For Geeks, “Geek For Geeks,” 16 9 2021. [Online]. Available: <https://www.geeksforgeeks.org/how-to-call-or-consume-external-api-in-spring-boot/>. [Acedido em 9 4 2023].
- [5] Spring.io, “Spring Blog,” 22 8 22. [Online]. Available: <https://spring.io/blog/2021/08/22/structuring-your-code-for-spring-framework-and-spring-boot>. [Acedido em 7 4 2023].
- [6] A. Ligios, “Baeldung,” 13 1 2023. [Online]. Available: <https://www.baeldung.com/spring-boot-logging>. [Acedido em 5 4 2023].
- [7] Baeldung, “Spring Boot: Customize the Jackson ObjectMapper,” 7 4 2023. [Online]. Available: <https://www.baeldung.com/spring-boot-customize-jackson-objectmapper>. [Acedido em 7 4 2023].
- [8] L. Garvie, “How to Get Response Body When Testing the Status Code in WebFlux WebClient,” 4 11 2022. [Online]. Available: <https://www.baeldung.com/spring-webclient-get-response-body>. [Acedido em 8 4 2023].
- [9] Baeldung, “Validating RequestParams and PathVariables in Spring,” 18 7 2022. [Online]. Available: <https://www.baeldung.com/spring-validate-requestparam-pathvariable>. [Acedido em 10 4 2023].
- [10] G. L. o. J. O. w. WebClient, “Get List of JSON Objects with WebClient,” 2 11 2022. [Online]. Available: <https://www.baeldung.com/spring-webclient-json-list>. [Acedido em 10 4 2023].
- [11] Jayway, “Github,” 10 4 2023. [Online]. Available: <https://github.com/json-path/JsonPath>. [Acedido em 10 4 2023].

- [12] FasterXML, “Github - Jackson,” 10 4 2023. [Online]. Available: <https://github.com/FasterXML/jackson>. [Acedido em 10 4 2023].