HW1: Mid-term assignment report

Tomás Oliveira Costa [89016], v2020-04-15

1 Introduction

1.1 Overview of the work

AirQuality is a web application used to get valuable air indicators of a specific location. It uses an external API to fetch global data and provides it to the user. It also has its own API running in the backend to make requests not available through the webapp.

1.2 Limitations

The biggest limitation of this project was the the developers (me) lack of self-management as a result of having to adapt to the new situations provided by COVID-19 and, as stated in this article, even though predicability and planning can decrease the innovation and learning, its almost always the way to go for all projects as it leads to better results.

One other big stepback was the fact that I was on halfway in production using Spring Data Rest, and after consulting the professor, realized this wasn't the best idea. So the plan for the architecture could have been better studied to prevent me from writing code that wasnt going to be used anyway.

This project has a lot to improve and most of its future features are listed on the GitLab Board.

2 Product specification

2.1 Functional scope and supported interactions

Since the API is decoupled from the frontend, it can be used by anyone trying to get air quality parameters on a specific location. Ex:. John wants to know the difference in the air quality of his work

place compared to his home, to take measures if the AQI (Air Quality Index) is reaching a level that is hazardous for his health.

2.2 System architecture

The frontend application is running on React, making use of React Hooks and new state changes, this choice was made primarily due to the fact that I have developed a web application before, and therefore it seemed the most fitting.

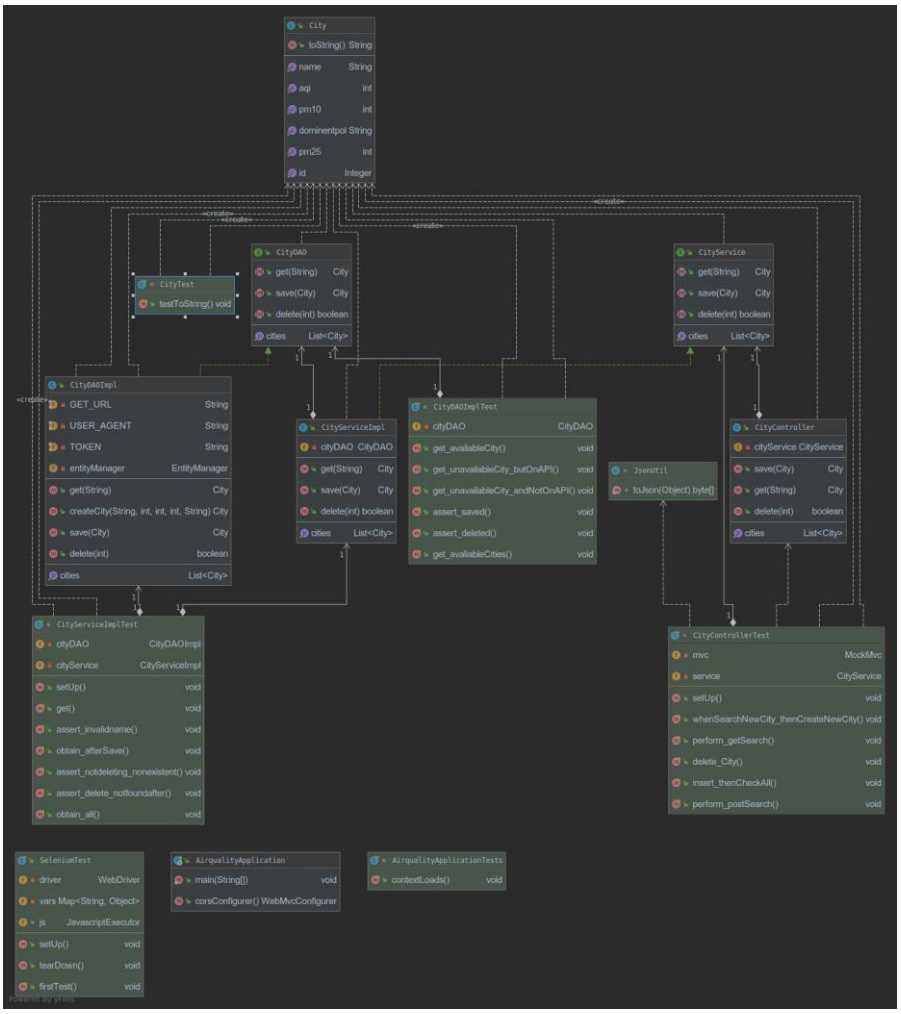
Sprint Boot Initializer was used to facilitate the acquiring of Spring Boot dependencies.

An instance of MySQL was running and mapping to a database, this choice was made having in mind the deployment of the final product, since we could then deploy it on our own server.

SQL instance running:

```
mysql> select * from tb_city;
+----+-----------+------+------+------+-------------+
| id | name      | aqi  | pm10 | pm25 | dominentpol |
+----+-----------+------+------+------+-------------+
|  1 | Aveiro    |   33 |   18 |   29 | o3          |
|  5 | Joaninha  |    0 |    0 |    0 | NULL        |
|  6 | Terra     |    4 |   15 |   11 | o3          |
|  7 | Porto     |   32 |    0 |    0 | o3          |
|  8 | lisboa    |   35 |    7 |   37 | o3          |
|  9 | madrid    |   34 |    9 |   34 | pm25        |
| 10 | barcelona |   22 |    0 |    0 | o3          |
| 11 | newyork   |   10 |    0 |    0 | pm25        |
| 12 | beijing   |  129 |   70 |  129 | pm25        |
| 13 | london    |   66 |   25 |   66 | pm25        |
| 14 | Paris     |   20 |   20 |   62 | pm10        |
| 15 | Lyon      |   17 |    0 |    0 | pm10        |
| 16 | Marselha  |   39 |   19 |   36 | o3          |
+----+-----------+------+------+------+-------------+
13 rows in set (0.00 sec)
```

Diagram below, also included in repo:



2.3 API for developers

To facilitate the view of the possible endpoints, without using an external service like SwaggerAPI, I wrote a small Markdown file called API.md which contains all the possible endpoints.

Excerpt of the file:

GETS

Get info of a specific city

Example GET `/api/search?name=<String name>`


```
{
        "id": 1,
        "name": "name",
        "aqi": 22,
        "pm10": 16,
        "pm25": 12,
        "dominentpol": "o3"
}
```

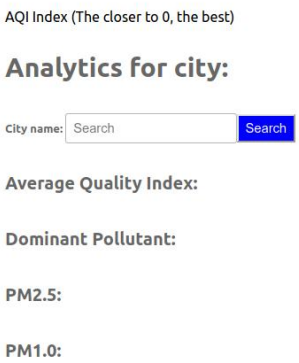

Get info of all cities

Example GET `/api/cities`

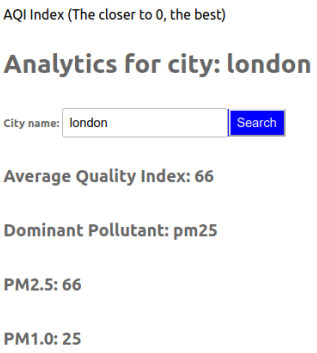

```
[
   {
        "id": 1,
        "name": "name",
        "aqi": 22,
        "pm10": 16,
        "pm25": 12,
        "dominentpol": "o3"
  },
  {
        "id": 2,
        "name": "name2",
        "aqi": 9,
        "pm10": 1,
        "pm25": 0,
        "dominentpol": "pm10"
  },
  ...
  ...
]
```


2.4 Demo

Before:



After London Search:



3 Quality assurance

3.1 Overall strategy for testing

Due to time restraints on the developer, the initial strategy was more inclined to a Smoke Testing approach, in which the tests on the new build were made to check basic functionality of the application. They were meant to be quick to execute. After this a Test Driven Development was more viable, specially with SonarQube providing important info on what code wasnt being covered by tests. This integration really facilitated the development of tests.

3.2 Unit and integration testing

On a general overview, tests were implemented for every class. I created unit tests, integration tests, functional tests and some other frontend tests using JavaScript.
Some tests include:

CityControllerTest

```
@Test
public void perform_getSearch() throws Exception {
    City city = new City();
    city.setName("TesteAll");
    city.setAqi(12);

    given(service.get("TesteAll")).willReturn(city);
    mvc.perform(get("/api/search?name=TesteAll")).andExpect(status().isOk()).andExpect(jsonPath("$.aqi",is(12)));
}
```

CityServiceImplTest

```java
@BeforeEach
public void setUp() throws Exception {
    City aveiro = new City();
    aveiro.setName("Aveiro");
    aveiro.setAqi(17);

    City braga = new City();
    braga.setName("Braga");
    braga.setAqi(12);
    braga.setId(123);

    List<City> cities = Arrays.asList(aveiro, braga);

    Mockito.when(cityDAO.get(aveiro.getName())).thenReturn(aveiro);
    Mockito.when(cityDAO.get(braga.getName())).thenReturn(braga);
    Mockito.when(cityDAO.get("fake_value")).thenReturn(null);
    Mockito.when(cityDAO.getCities()).thenReturn(cities);
    Mockito.when(cityDAO.delete(123)).thenReturn(true);
}

@Test
void get() {
    City found = cityService.get("Aveiro");
    assertThat(found.getName()).isEqualTo("Aveiro");
}
```

CityDAOImplTest

```java
@Test
void get_unavailableCity_butOnAPI() throws Exception{
    //saves on API, then former call is not null
    cityDAO.get("manchester");
    assertThat(cityDAO.get("manchester")).isNotNull();
}
```

For more tests visit the repository.

3.3 Functional testing

Functional testing was done with SeleniumIDE, which includes a single test that consists in two parts: the first search is for an already existing record on the table (Aveiro), then we search for a record which isn't included in the table, so the backend will fetch the external API and then return the results fetched to the frontend.

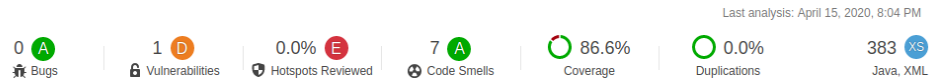
Test Implemented:

```
@Test
public void firstTest() {
  driver.get("http://localhost:3000/");
  driver.manage().window().setSize(new Dimension(879, 648));
  driver.findElement(By.name("text")).click();
  driver.findElement(By.name("text")).sendKeys("aveiro");
  driver.findElement(By.cssSelector("button")).click();
  driver.findElement(By.name("text")).click();
  driver.findElement(By.name("text")).click();
  {
    WebElement element = driver.findElement(By.name("text"));
    Actions builder = new Actions(driver);
    builder.doubleClick(element).perform();
  }
  driver.findElement(By.name("text")).click();
  driver.findElement(By.name("text")).sendKeys("sevilha");
  driver.findElement(By.cssSelector("button")).click();
}
```

3.4 Static code analysis

For static code analysis and due to past experience using it on lab classses, I decided to use JaCoCo to generate reports and SonarQube running locally to analyze them. This really facilitated the implementation of tests, since they highlight your test coverage on the new code you have added since the last interaction.

On the last stage these were the numbers I was able to obtain from SonarQube:



The coverage is of 86.6% but all major tests are taken into account. And as quoted in this post, some code didn't go into account, but the objective was never to get 100% coverage:

"You don't add tests to make a metric tool happy.
Loading a Spring context of the application ***takes time****. Don't add it in each developer build just to win about 0.1% of coverage in your application.*
Here ***you don't cover only 1 statement*** *from 1 public method. It represents nothing in terms of coverage in an application where* ***thousands of statements are generally written****."*

As we can see, the builds over time had a great improvement in code coverage and specially a decrease in code smells, which is something I, as a developer have learned a lot. Since I was writing code that wasn't being very productive in the end, through useless calls, generalized exceptions, etc.

3.5 Continuous integration pipeline [optional]

The choice of GitLab was to ease the construction of a CI pipeline, but, due to time restrictions and a poor time-management from the developers, such wasn't possible. So, the features are included for future work on the GitLab Board.

4 References & resources

Project resources

- Git repository: https://gitlab.com/tom1k/airquality
- Solution is simple enough to be described with two screenshots, explained above in frontend.
- The solution could be deployed, for that we would as explained here:
    - Package React app with Spring Boot
    - Setting up the Server for deployment
    - Deploy

Reference materials

https://waqi.info/

https://medium.com/@mukundmadhav/build-and-deploy-react-app-with-spring-boot-and-mysql-6f888eb0c600

https://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html

https://www.automatetheplanet.com/unit-testing-guidelines/

https://medium.com/backend-habit/generate-codecoverage-report-with-jacoco-and-sonarqube-ed15c4045885

https://stackoverflow.com/questions/14977018/jpa-how-to-get-entity-based-on-field-value-other-than-id