HW1: Mid-term assignment report

Anthony dos Santos Pereira [93016], 14-05-2021

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 individual project consisted of the development of a web application that returns data related to air quality, and the making of several types of tests that prove the proper functioning of this same application.

1.2 Current limitations

The main limitations are the optional features not implemented in this application, such as the use of more than an external API and a continuous integration framework. Also, the front end could be improved.

2 Product specification

2.1 Functional scope and supported interactions

The application is mainly divided into two parts: the web app and the developed API.

In the part of the web application, the user is redirected to the main page, in which there are two search bars. In the left bar, they can search by the name of a city, while in the right bar they search by geographic coordinates. For both inputs there are several types of validations - so if the user enters something that was not expected of them, such as an invalid query data type or an empty search, they will be redirected to one of the several existing error pages. When the search is successful, the user is redirected to a new page, in which information about the air quality of that city or those coordinates is displayed. As a bonus, the user, when searching for geographic coordinates, in addition to the coordinates by which they performed the search appear on the results page, the corresponding city appears, for better clarification to the user, taking into account that when entering geographic coordinates, we are waiting to obtain a location for a city.

2.2 System architecture

As explained earlier, a user can interact with the application in two ways - via the web component or the API.

In the web component, the user interacts with HTML pages in the browser (@Controller), which are interconnected with the biz logic (that is, the @Service), where a large part of the application logic is performed, such as deciding where to go to get the data that will be presented - directly to the cache or directly to the external API.

The cache is a class with the @Component annotation and it consists of a hashmap, where pairs of City-Air Quality objects are stored, and of a new class that gathers three counters, necessary to obtain some statistical data - CacheStatistics, that is also a @Component, to make easier the beans injection with @Autowired. City and AirQuality are two ordinary classes that store data.

In the API component, it's the same thing, but instead of interacting with the front end, the user interacts directly with endpoints defined in a @RestController.

The technologies and frameworks used to develop this project were mostly Spring Boot & Thymelaf and HTML/CSS/JS.

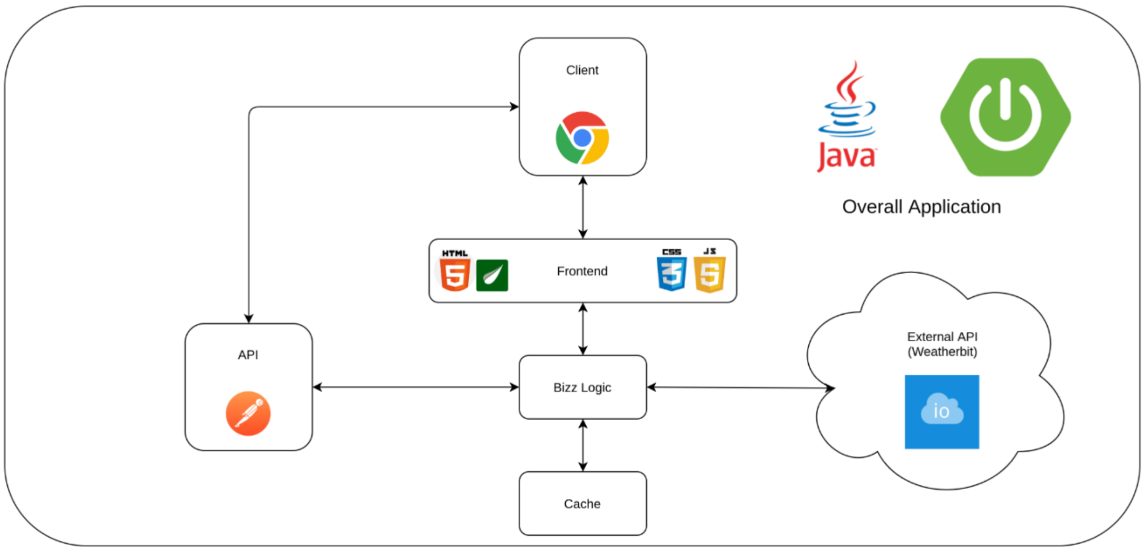


Fig.1 – Architecture Diagram of the developed application

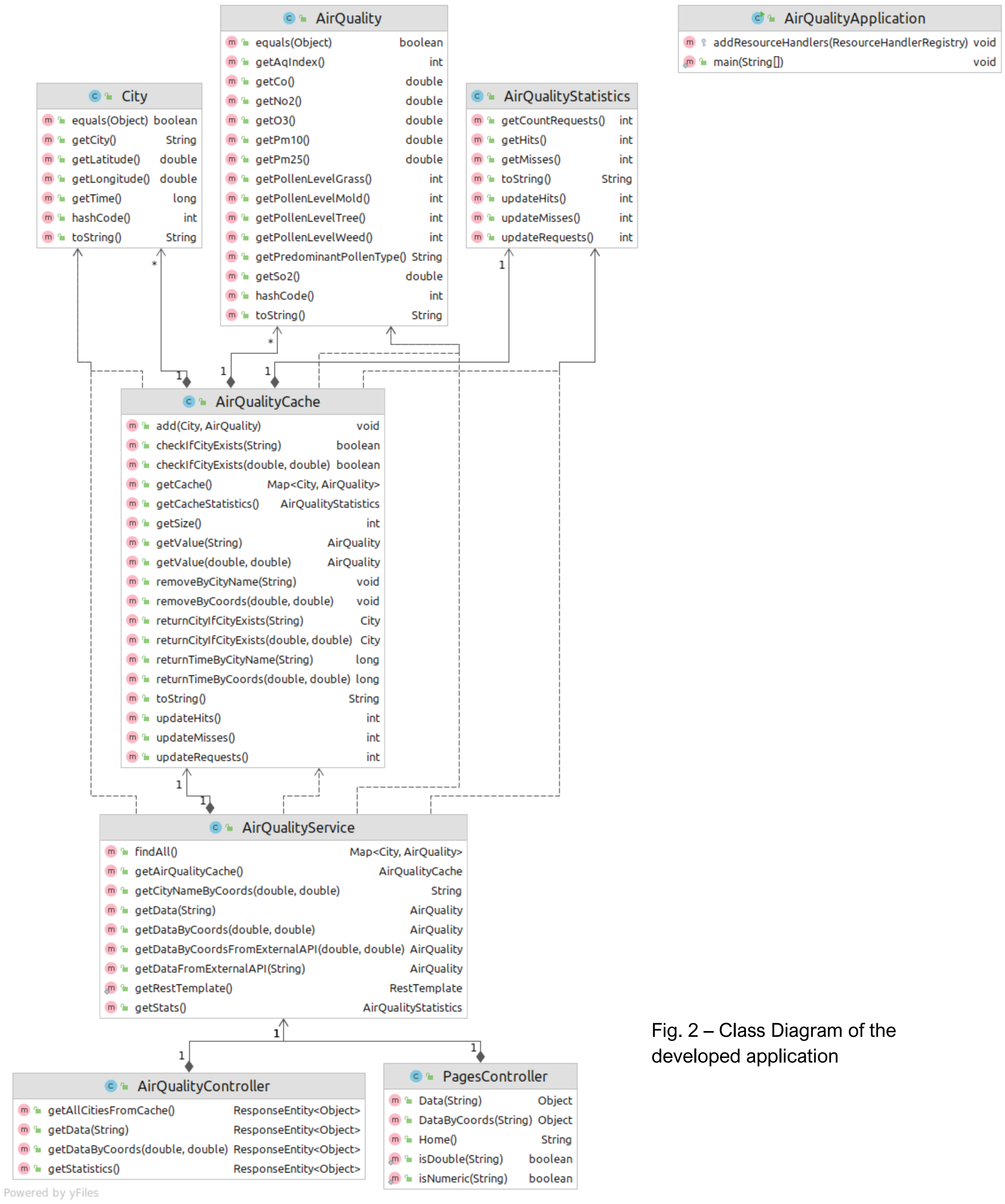
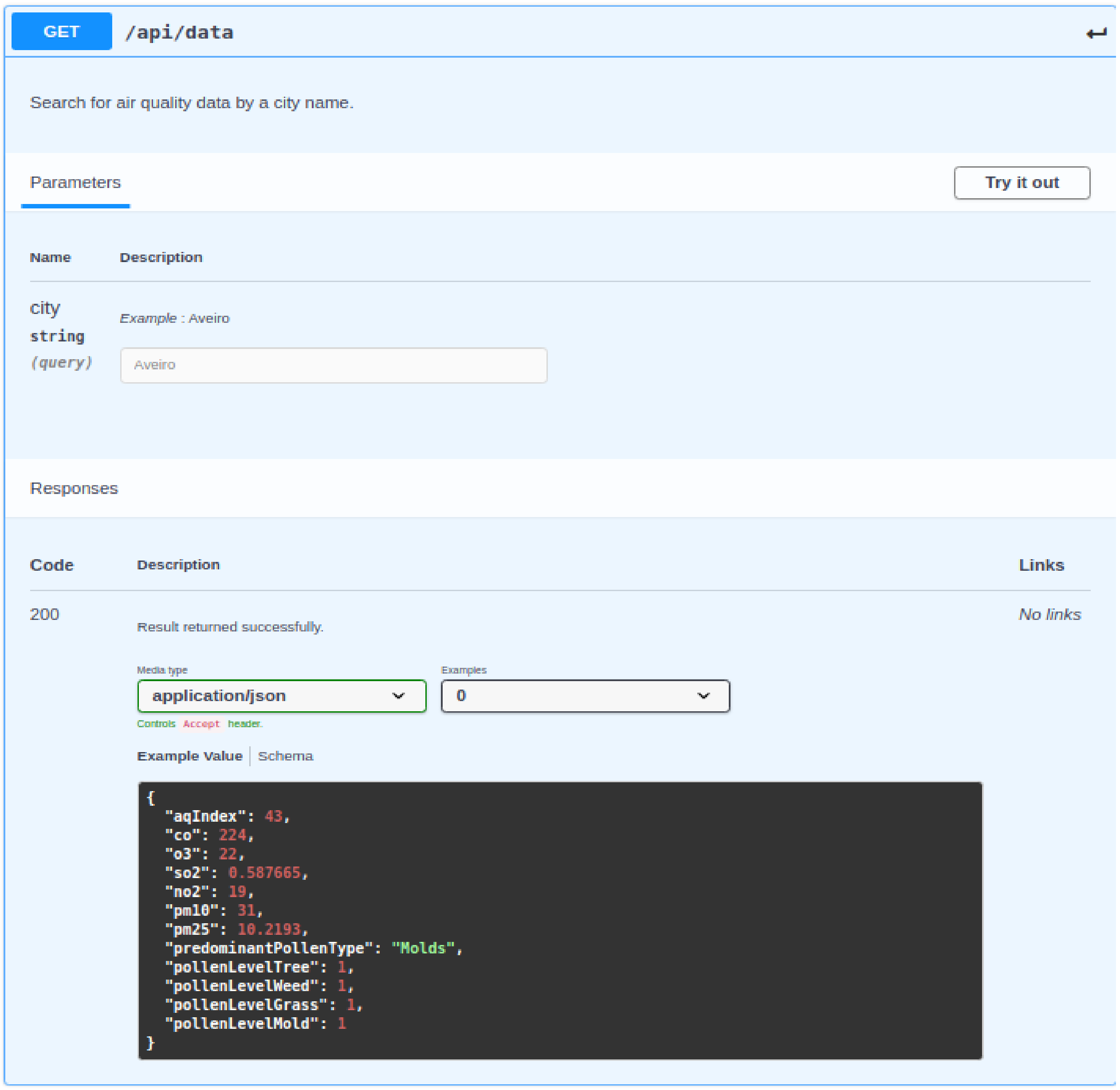

AirQuality
equals(Object) boolean
getAqIndex() int
getCo() double
getNo2() double
getO3() double
getPm10() double
getPm25() double
getPollenLevelGrass() int
getPollenLevelMold() int
getPollenLevelTree() int
getPollenLevelWeed() int
getPredominantPollenType() String
getSo2() double
hashCode() int
toString() String
AirQualityApplication
addResourceHandlers(ResourceHandlerRegistry) void
main(String[]) void
City
equals(Object) boolean
getCity() String
getLatitude() double
getLongitude() double
getTime() long
hashCode() int
toString() String
AirQualityStatistics
getCountRequests() int
getHits() int
getMisses() int
toString() String
updateHits() int
updateMisses() int
updateRequests() int
*
1
*
1
1
1
AirQualityCache
add(City, AirQuality) void
checkIfCityExists(String) boolean
checkIfCityExists(double, double) boolean
getCache() Map<City, AirQuality>
getCacheStatistics() AirQualityStatistics
getSize() int
getValue(String) AirQuality
getValue(double, double) AirQuality
removeByCityName(String) void
removeByCoords(double, double) void
returnCityIfCityExists(String) City
returnCityIfCityExists(double, double) City
returnTimeByCityName(String) long
returnTimeByCoords(double, double) long
toString() String
updateHits() int
updateMisses() int
updateRequests() int
1
1
AirQualityService
findAll() Map<City, AirQuality>
getAirQualityCache() AirQualityCache
getCityNameByCoords(double, double) String
getData(String) AirQuality
getDataByCoords(double, double) AirQuality
getDataByCoordsFromExternalAPI(double, double) AirQuality
getDataFromExternalAPI(String) AirQuality
getRestTemplate() RestTemplate
getStats() AirQualityStatistics
1
1
1
AirQualityController
getAllCitiesFromCache() ResponseEntity<Object>
getData(String) ResponseEntity<Object>
getDataByCoords(double, double) ResponseEntity<Object>
getStatistics() ResponseEntity<Object>
PagesController
Data(String) Object
DataByCoords(String) Object
Home() String
isDouble(String) boolean
isNumeric(String) boolean
Powered by yFiles

Fig. 2 – Class Diagram of the developed application

2.3 API for developers

In the API, there are two groups of endpoints: one that allows obtaining information about air quality, by a city name and by coordinates, and another that allows getting some information about the cache, such as what is stored at the moment and its statistics – for instance, the number of requests to access the cache and the number of times the cache had an updated and outdated value, i.e., an air quality object.



Air Quality Data Endpoints

GET /api/data

Search for air quality data by a city name.

Parameters

Name	Description
city string *(query)*	*Example* : Aveiro Aveiro

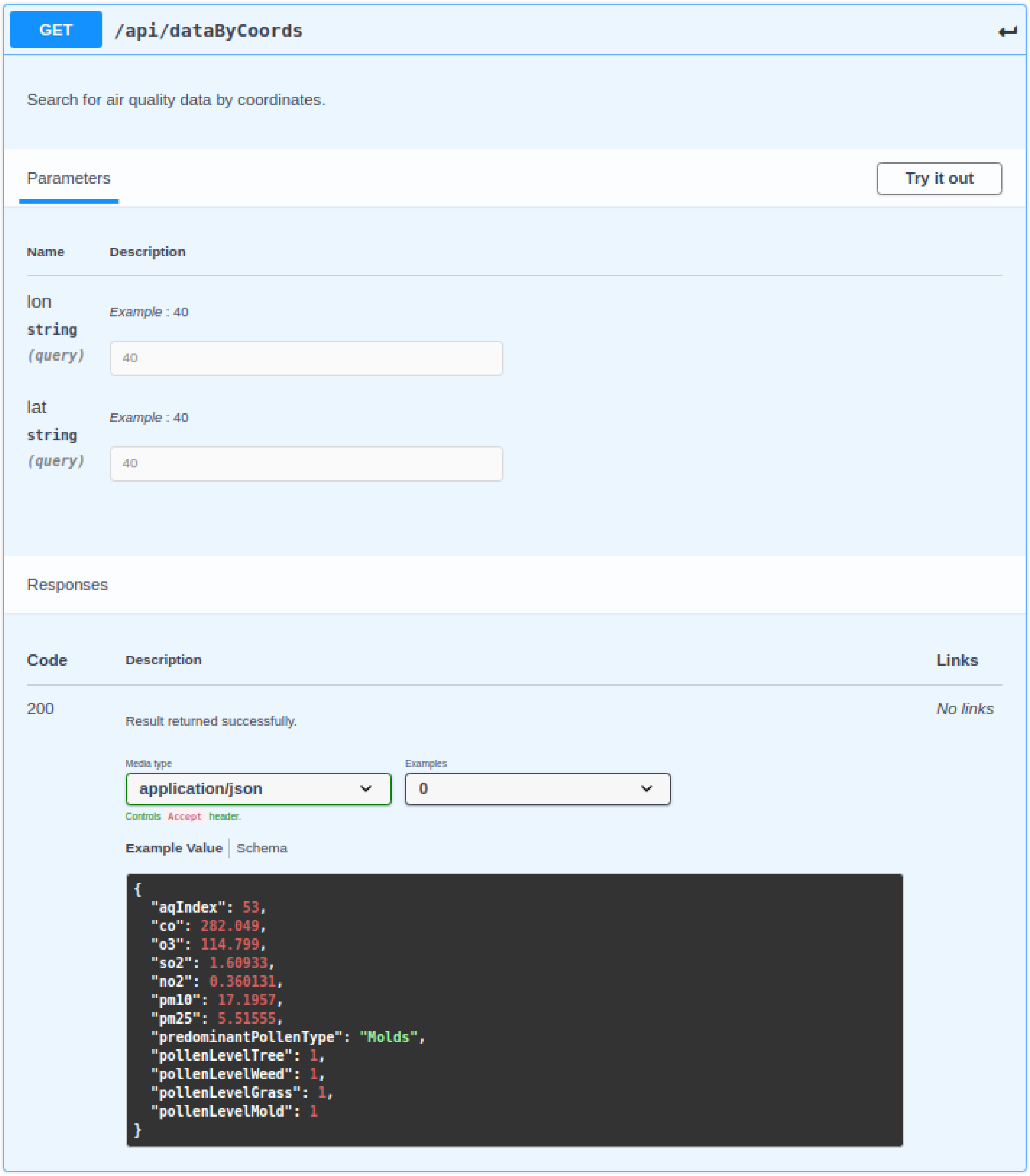
Responses

Code	Description	Links
200	Result returned successfully. Media type: application/json Controls `Accept` header. Examples: 0	*No links*

Example Value | Schema

```
{
  "aqIndex": 43,
  "co": 224,
  "o3": 22,
  "so2": 0.587665,
  "no2": 19,
  "pm10": 31,
  "pm25": 10.2193,
  "predominantPollenType": "Molds",
  "pollenLevelTree": 1,
  "pollenLevelWeed": 1,
  "pollenLevelGrass": 1,
  "pollenLevelMold": 1
}
```

Fig.3 - Documentation of the /api/data endpoint



/api/dataByCoords

Search for air quality data by coordinates.

Parameters

Name	Description
lon `string` *(query)*	*Example* : 40 40
lat `string` *(query)*	*Example* : 40 40

Responses

Code	Description	Links
200	Result returned successfully.	*No links*

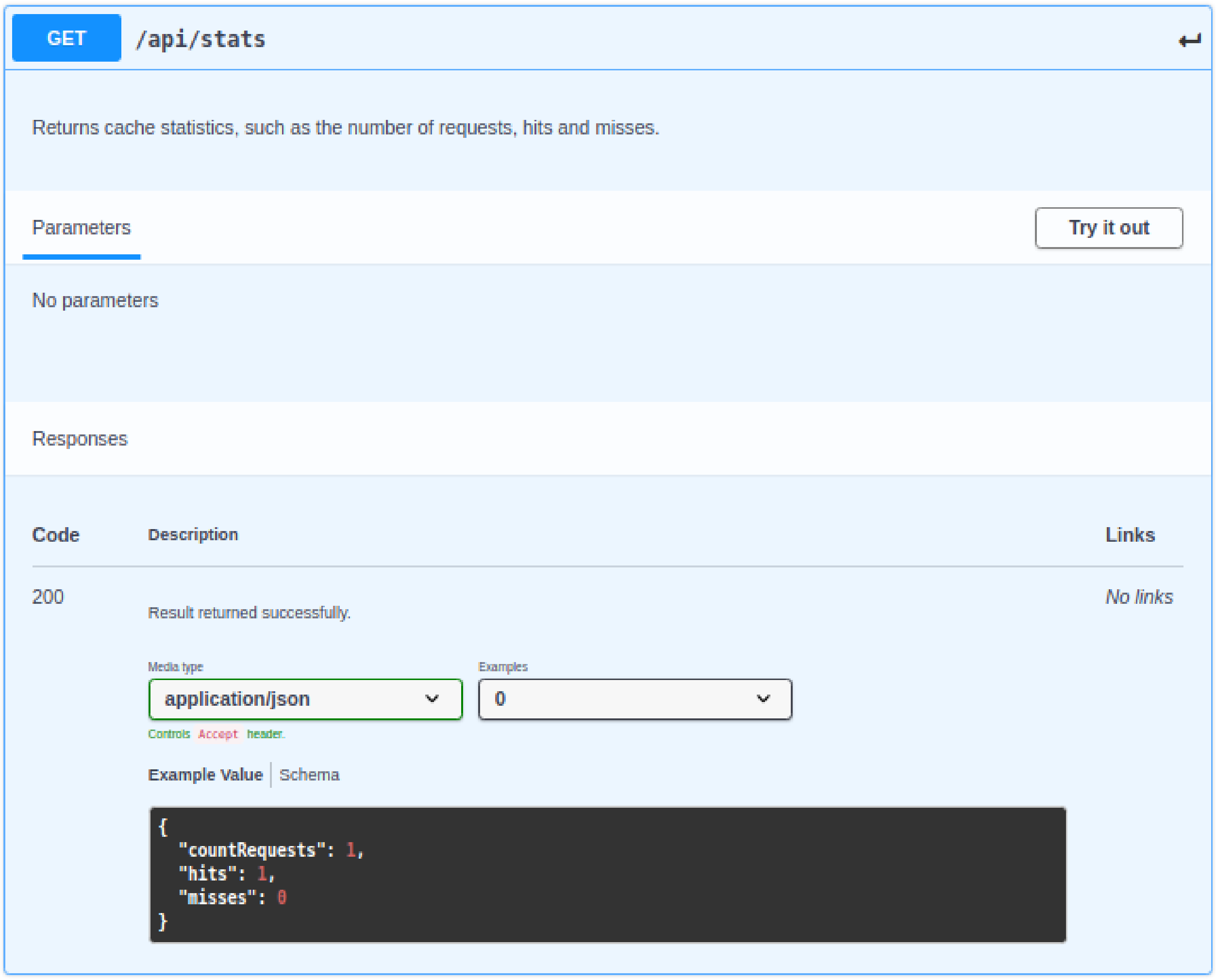
Media type: application/json — Examples: 0

Controls `Accept` header.

```
{
  "aqIndex": 53,
  "co": 282.049,
  "o3": 114.799,
  "so2": 1.60933,
  "no2": 0.360131,
  "pm10": 17.1957,
  "pm25": 5.51555,
  "predominantPollenType": "Molds",
  "pollenLevelTree": 1,
  "pollenLevelWeed": 1,
  "pollenLevelGrass": 1,
  "pollenLevelMold": 1
}
```

Fig.4 – Documentation of the /api/dataByCoords endpoint

Cache Endpoints



/api/stats

Returns cache statistics, such as the number of requests, hits and misses.

Parameters

No parameters

Responses

Code	Description	Links
200	Result returned successfully.	No links

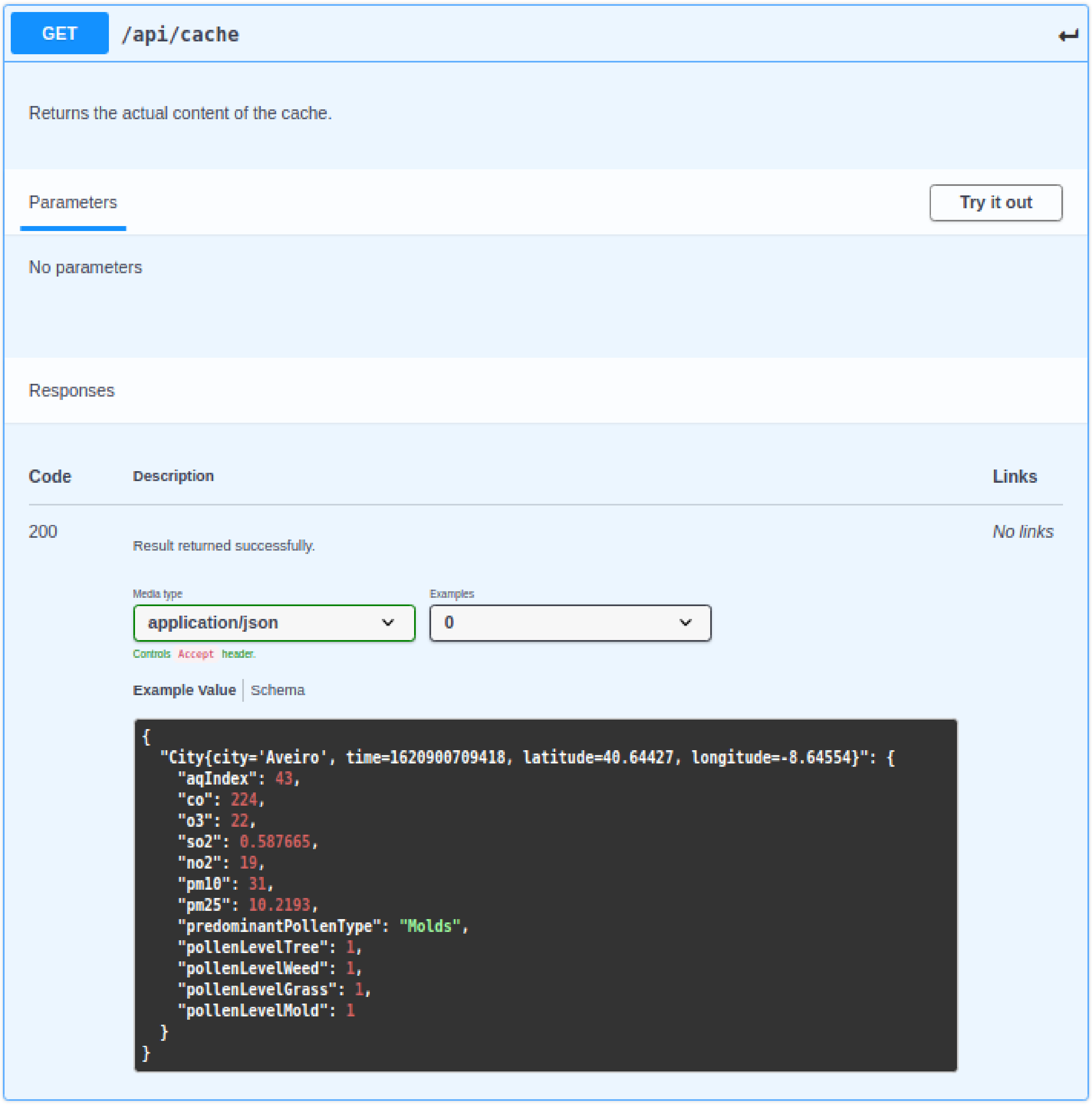
Media type: application/json

Controls `Accept` header.

Examples: 0

```
{
  "countRequests": 1,
  "hits": 1,
  "misses": 0
}
```

Fig.5 – Documentation of the /api/stats endpoint



/api/cache

Returns the actual content of the cache.

Parameters

No parameters

Responses

Code	Description	Links
200	Result returned successfully.	No links

Media type: application/json

Controls `Accept` header.

Examples: 0

```
{
  "City{city='Aveiro', time=1620900709418, latitude=40.64427, longitude=-8.64554}": {
    "aqIndex": 43,
    "co": 224,
    "o3": 22,
    "so2": 0.587665,
    "no2": 19,
    "pm10": 31,
    "pm25": 10.2193,
    "predominantPollenType": "Molds",
    "pollenLevelTree": 1,
    "pollenLevelWeed": 1,
    "pollenLevelGrass": 1,
    "pollenLevelMold": 1
  }
}
```

Fig.6 – Documentation of the /api/cache endpoint

3 Quality assurance

3.1 Overall strategy for testing

The general strategy for tests was to try to test almost everything to the point of exhaustion, trying to do at least one example of each type of test given in the curricular unit, if it was meaningful for the application.

3.2 Unit and integration testing

- Unit tests
  - ✓ Most of the developed unit tests were to test each method from the cache class. Some examples are: check the cache size after inserting a new City-Airquality pair, check if a specific city exists or not in the cache, check the size after removing a pair, and check if the returned air quality data from a city that previously exists in the cache equals to a predefined air quality object.

- Spring Boot tests
  - ✓ AirQualityPagesControllerTest - This class has multiple tests that check if a valid parameter was given (and then a ModelAndView is returned, allowing to redirect to the results page) and if it's not the case, one of the error pages is returned.

  - ✓ AirQualityRestControllerServiceTest - This class consists of a WebMvcTest that tries to test the boundary between the RestController and the Service classes. It's defined that, when trying to get air quality data by a predefined city name or some coordinates, a predefined AirQualityData object should be returned; with this, we can assert the result that will be returned, including the value of the air quality index attribute.

  - ✓ AirQualityServiceTest - This class tries to test the connection between the Service and the cache; first, in the setUp method, City and AirQuality objects were created for "Aveiro" and its real coordinates. Note the time attribute - represents the TTL that will decide if the stored pair is or not up-to-date. Since the value defined in the time parameter of the created City object is intentionally low and a new time value is created when executing the Service's getData method, using Date getTime() (returns the number of milliseconds since January 1st, 1970), the existing pair in the cache will be removed and will be replaced by a fresh-new pair, with updated information. It's because of this, requesting for a city that already exists in the cache, that the number of requests counter increases one point, and because the data that existed was out-of-date (because of the time attribute), the number of misses is also increased by one.

- Integration tests
  - ✓ Two integration tests were made - one to test the return of the content that is stored at the moment in cache and another one to check the cache statistics after requesting air quality data for a city that is already stored previously in the cache.

- Mock tests

- ✓ Two mock test classes were made - AirQualityControllerServiceMockTest and AirQualityServiceCacheMockTest - where in the first one the Service is mocked, assigning by default a return when called by the Service getData method, and it's confirmed that the result obtained, especially one of its attributes, is equal to one of the attributes of the Air Quality object previously defined in the setUp. In the second one, both Cache and Service are mocked, and a call to the external API is pretended.

3.3 Functional testing

- Selenium tests
  - ✓ Tests were performed using Selenium IDE to test the user interface, including error pages when the user did an invalid search (like a number instead of a string when searching for air quality data by a city name) or in the absence of results found.

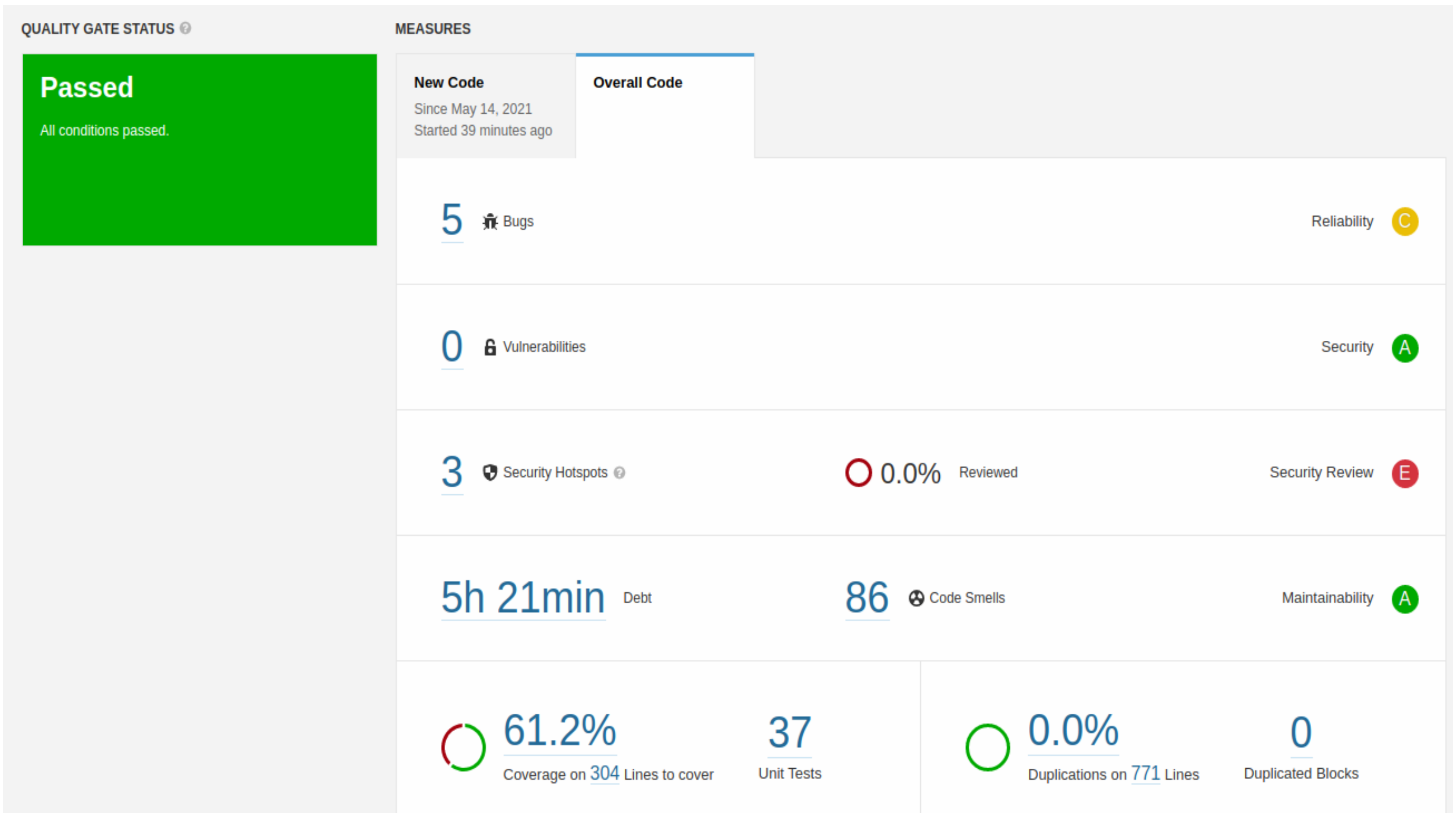
- Selenium-Jupiter extension tests
  - ✓ There is one Selenium-Jupiter extension test in the project, identical to one of the Selenium ones, just to show another way of testing the same thing.

- Cucumber tests
  - ✓ Two features were made to illustrate the two main scenarios of usage of this web application: search for air quality data by a city name and by geographical coordinates. These tests verify if, when a user gives some valid parameters and presses ENTER, is redirected to a new page that shows the criteria that they previously have written. One of the tests was written in French, just to explore new ways of "making the same thing by different manners" (it was also funny to compare each corresponding keywords and see that the "translation" is quite accurate).

- Rest-assured tests
  - ✓ These tests were mostly developed to test the API and the returned status codes, although the first one tests if it's possible to go to the application homepage. Some examples of test scenarios are searching with a valid query when searching by a city name, searching with an invalid query (a number), searching with an empty parameter, and check if the result of a valid performed query is not empty.

3.4 Static code analysis

SonarCloud was firstly used for static code analysis, but for some reason, is not recognizing the src/main/java folder as one of the source code folders – only HTML, CSS, and JS code appears. That's the reason why SonarQube was used.


QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code
Since May 14, 2021
Started 39 minutes ago

Overall Code

5 Bugs — Reliability C

0 Vulnerabilities — Security A

3 Security Hotspots — 0.0% Reviewed — Security Review E

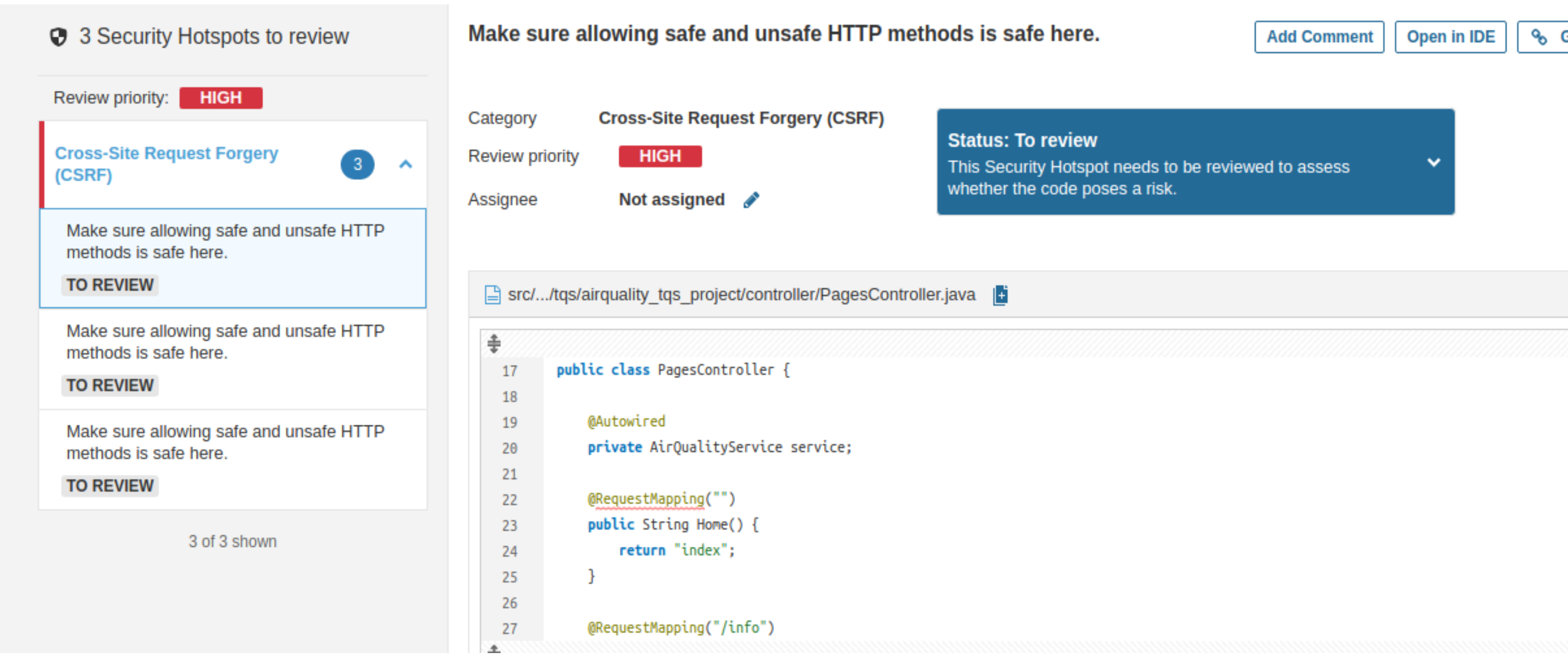
5h 21min Debt — 86 Code Smells — Maintainability A

61.2% Coverage on 304 Lines to cover — 37 Unit Tests

0.0% Duplications on 771 Lines — 0 Duplicated Blocks

Fig.7 – Screenshot of the output from the SonarQube analysis

As can be seen from the analysis of the figure above, the project passed the quality gate. One interesting thing that can also be seen is the security review, where 4 Security Hotspots were found. By clicking on it, this was shown:


3 Security Hotspots to review

Review priority: HIGH

Cross-Site Request Forgery (CSRF) 3

Make sure allowing safe and unsafe HTTP methods is safe here.
TO REVIEW

Make sure allowing safe and unsafe HTTP methods is safe here.
TO REVIEW

Make sure allowing safe and unsafe HTTP methods is safe here.
TO REVIEW

3 of 3 shown

Make sure allowing safe and unsafe HTTP methods is safe here.

Category: Cross-Site Request Forgery (CSRF)
Review priority: HIGH
Assignee: Not assigned

Status: To review
This Security Hotspot needs to be reviewed to assess whether the code poses a risk.

src/.../tqs/airquality_tqs_project/controller/PagesController.java

```
public class PagesController {

    @Autowired
    private AirQualityService service;

    @RequestMapping("")
    public String Home() {
        return "index";
    }

    @RequestMapping("/info")
```

Fig.8 – Screenshot of the security hotspots to review

It can be deduced that these hotspots were because HTTP was used to develop this project, and that's why these warnings were ignored.

4 References & resources

Project resources

- Video demo: https://github.com/Anth0nyPereira/TQSProject
- Ready to use application: https://need-some-fresh-air.herokuapp.com
- QA dashboard: https://sonarcloud.io/dashboard?id=Anth0nyPereira_TQSProject

(did not figure out why it's not scanning the entire project, it's only scanning the resources folder)

Reference materials

Weatherbit Air Quality API (Current) Website: https://www.weatherbit.io/api/airquality-current