HW1: Mid-term assignment report

Francisca Inês Marcos de Barros [93102], v2021-05-13

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 term "air quality" relates directly to how pure or contaminated the air in a given area is. Government organizations depend on the Air Quality Index (AQI) to offer this information to the general public; the higher the rating, the more polluted the air.

The 'Air Quality Application' offers a simple, but user-friendly website where a user can review air quality related details of a desired area, such as AQI, pollutants measurements, and general weather measurement.

1.2 Current limitations

A few problems arose during the project's production and were addressed as best as possible.

One of the most difficult problems was parsing the JSON data returned by the third-party API. Using only the JSON-Java package and dealing with a large JSON-response easily made this task very vulnerable to errors - for example, one basic (but incorrect) use of JSONArray instead of JSONObject would immediately cause errors.

Furthermore, when refactoring and running the *SeleniumIDE*-generated functional tests, a strange (yet persistent) *java.lang.NoSuchMethodError* would be thrown. After doing comprehensive search, I determined that this was occurring due to the presence of a conflicting version of Guava on the classpath. To address this problem, a Guava dependency was added to the *pom.xml* file.

2 Product specification

2.1 Functional scope and supported interactions

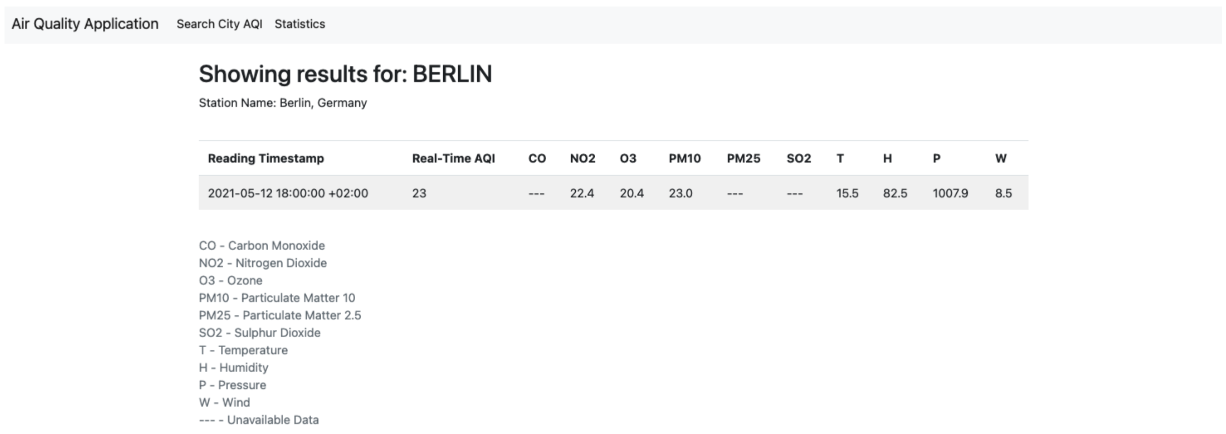
The key scenario of use of the built application is searching for the air quality of a desired area.

Due to the obvious effects it may have on human health and the atmosphere, an individual would want to check the air quality of a specific area.

Communication can be done with the application through the application's minimalistic web page, which allows users to enter a region and view its air quality values.

If the location is shown to be valid, users will be forwarded to another page with a table-view of the available values for that location.

Assuming the user wants to monitor the Air Quality in Berlin, the following screenshot represents the results that would be displayed.



Showing results for: BERLIN

Station Name: Berlin, Germany

Reading Timestamp	Real-Time AQI	CO	NO2	O3	PM10	PM25	SO2	T	H	P	W
2021-05-12 18:00:00 +02:00	23	---	22.4	20.4	23.0	---	---	15.5	82.5	1007.9	8.5

CO - Carbon Monoxide
NO2 - Nitrogen Dioxide
O3 - Ozone
PM10 - Particulate Matter 10
PM25 - Particulate Matter 2.5
SO2 - Sulphur Dioxide
T - Temperature
H - Humidity
P - Pressure
W - Wind
--- - Unavailable Data

Figure 1 - Results Displayed to the User

2.2 System architecture

The Air Quality Application was designed using Spring-Boot for the backend, as requested by the professor. The presentation layer was built with the templating framework Thymeleaf, which integrates with Spring-Boot and allows presenting information to the user seamlessly. Through the use of a controller and the models themselves, the connection between front and backend is possible.

A Cache-like mechanism was also expected, which stored the most recent results from the third-party source copied locally. This structure was implemented using a Passive Expiring Map, which allows for the specification of a time-to-live policy.

An external API was used to collect the data in respect of the Air Quality Information itself. The World's Air Pollution API was the preferred one after the investigation and experimentation of many different sources, since it proved to be the most comprehensive information source.

Due to its size, the system architecture diagram is available in ***Appendix I***.

2.3 API for developers

The REST endpoints implemented in the project can be characterized into two distinct main groups:

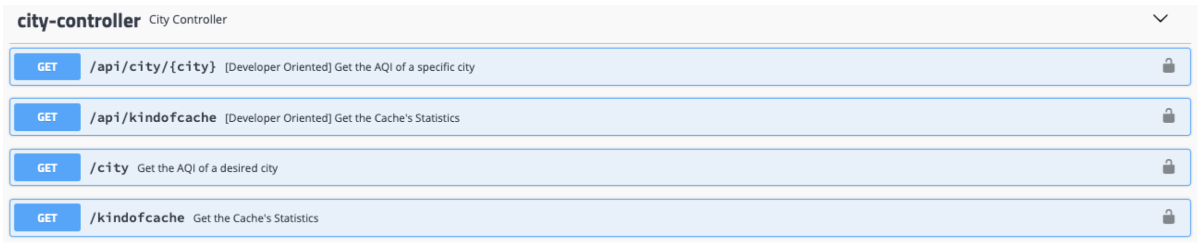
→ Public Access Endpoints

→ Developer Oriented Endpoints

While the *Public Access Endpoints* respond to user queries, such as searching for a specific city's air quality details or checking the cache's statistics, and display the information in a user-friendly way (using templates), the latter return JSON responses, allowing a developer to conveniently monitor the application and inspect its intended behavior.

Although this function was only requested to be introduced in *Developer Oriented Endpoints*, the capability to inspect the cache's statistics was also implemented in *Public Access Endpoints* to make checking the statistics more user-friendly.

The screenshot below shows the various endpoints that have been established, specifically demonstrating the distinction between all user-groups.



city-controller City Controller

GET `/api/city/{city}` [Developer Oriented] Get the AQI of a specific city

GET `/api/kindofcache` [Developer Oriented] Get the Cache's Statistics

GET `/city` Get the AQI of a desired city

GET `/kindofcache` Get the Cache's Statistics

Figure 2 - Endpoint Available (from Swagger-UI API-documentation)

In terms of Public Domain:

→ Endpoint ***'/city'*** -- uses a *GET-Request* to retrieve the Air Quality Information of a city defined by the user on the application's home page.

✓ If the city is correct (that is, it resides in the Cache or in the third-party API used), then all available information will be shown to the user in a table.

× Otherwise, an error will be thrown and the user will be redirected to the specified error page.

→ Endpoint ***'/kindofcache*** -- uses a *GET-Request* to retrieve and view the cache's current data - hits, misses, and cumulative requests.

Regarding the Developer Oriented Domain, which are accessible via *cURL command* or tools such as *Postman Tool*:

→ Endpoint ***'/api/city/{city}'*** -- uses a *GET-Request* to retrieve the Air Quality Information of the city name that the developer chooses

- ✓ If the city name is correct, then the HTTP-status will be set to OK (200) and all available information will be shown in JSON-format.
- × Otherwise, the HTTP-status will be set to NOT FOUND (404), and the city details will all be set to null and shown in JSON-format.

→ Endpoint ***'/api/kindofcache'*** -- uses a *GET-Request* to display the cache's current data, in JSON-format.

3 Quality assurance

3.1 Overall strategy for testing

Test Driven Development was strongly used in the test development approach. Any basic test cases were written prior to the actual implementation of the various classes that comprise the application.

These tests, which primarily described the planned use of the application, allowed for the central production of the application to be greatly sped up while remaining explicit what the key objectives I wanted to accomplish.

More test cases were introduced after the development process was completed in order to give as much coverage as possible to the decision branches and different methods.

In order to meet the objectives, the following types of tests were automated:

- ✓ Unit tests — applied in the Cache, to monitor its behavior.
- ✓ Service Level Tests — introduced in the City Service; Mockito was selected as the framework to facilitate their development since it enables the separation of dependencies by the use of mocks that imitate the real behavior of external data sources.
- ✓ Integration Tests — can be run on the City Controller; by using the @WebMvcTest annotation, the testing coverage is limited to only the application's Web-part.
- ✓ Functional Tests — the Selenium WebDriver technology was selected to apply this sort of test, specifically the Selenium IDE, which offers a straightforward solution to functional testing.

3.2 Unit and integration testing

As mentioned in the preceding subsection, Unit Testing was performed in the Cache and Integration Testing in the City Controller since the latter marks the entry point of the application's core functionalities.

The following test cases were considered:

→ KindOfCache
- ✓ The cache should be empty upon construction
- ✓ The cache shouldn't be empty, if it was previously requested
- ✓ The cache shouldn't be empty, if it was hit
- ✓ The cache shouldn't be empty, if it was missed

→ KindOfCacheService
- ✓ The cache service should return null, when the storing time expires
- ✓ The cache service should return the correct value, when asked for a key

→ CityController
  - ✓ When the city is valid, the related AQI should be returned
  - ✓ When the city is invalid, a null-city should be returned

3.3 Functional testing

The Selenium IDE was used to make Functional Testing even easier. There were two main test cases considered: looking for the AQI of a legitimate city and an invalid city.

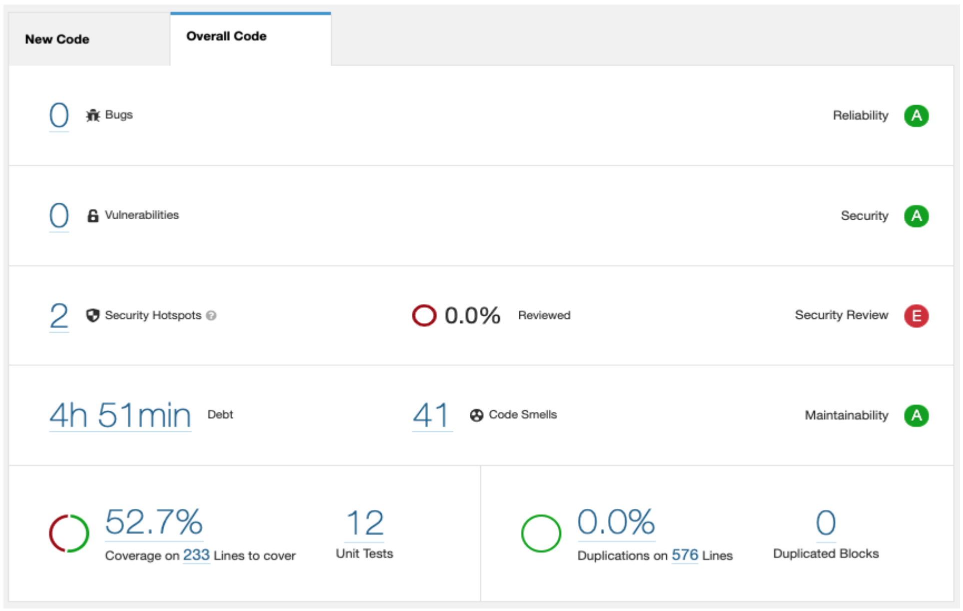
Both tests use the same technique, testing and asserting the content of the information loaded, allowing one to determine if the current page is appropriate.

3.4 Static code analysis

Static code analysis techniques may be used to identify patterns in code and spot potential security threats and problems with code consistency. It is critical to reveal these problems during the early stages of development so that software does not hit the market without a stable and strong code base.

SonarQube was selected as the medium to facilitate this process because it was simple to use in practical classes. The default quality gate was modified, changing coverage to be greater than 50%.

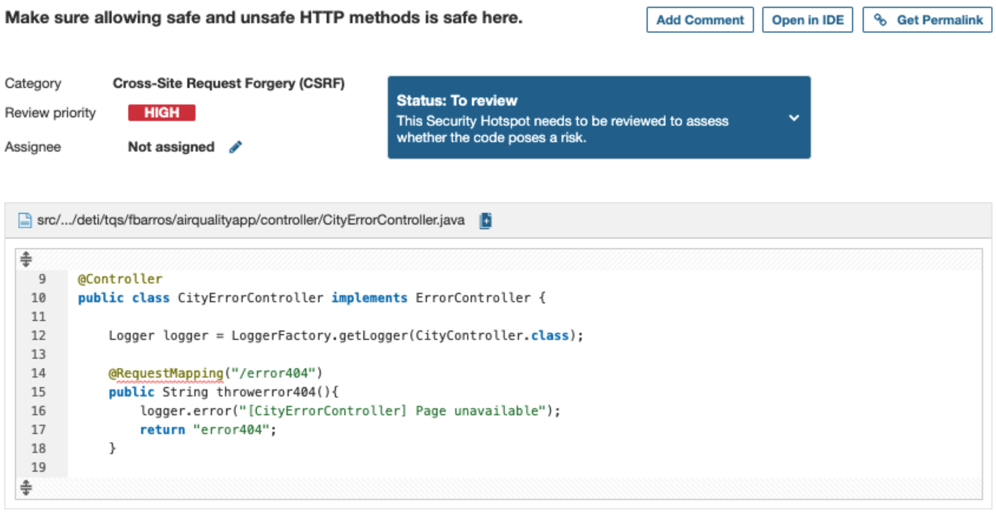
The first analysis run was, indeed, quite interesting; several major and critical issues were recorded. Most (if not all) of them would likely not be tackled or even noted if not for the assistance of this tool. The screenshot below shows the dashboard of that analysis.



0 Bugs		Reliability A
0 Vulnerabilities		Security A
2 Security Hotspots	0.0% Reviewed	Security Review E
4h 51min Debt	41 Code Smells	Maintainability A
52.7% Coverage on 233 Lines to cover	12 Unit Tests	
0.0% Duplications on 576 Lines	0 Duplicated Blocks	

Figure 3 - Dashboard of the first analysis conducted

Aside from the high technological debt (nearly 5 hours) and 41 code smells found (two critical and ten major), it was evident that the software built was not robust enough, even though it passed the defined quality gate. SonarQube had also identified two unnoticed security hotspots.


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.

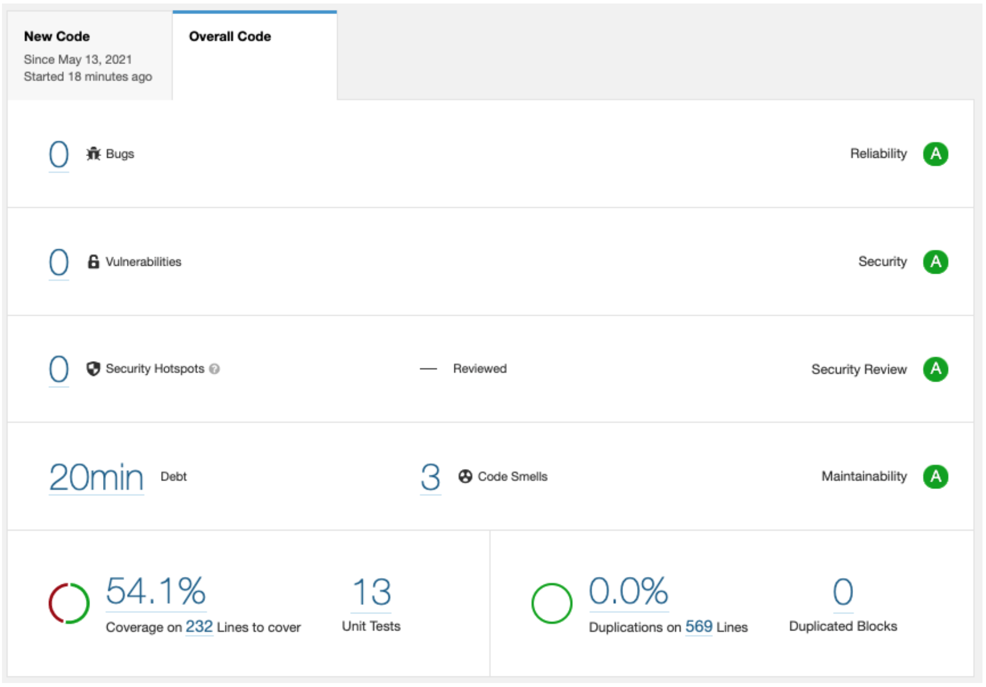
src/.../deti/tqs/fbarros/airqualityapp/controller/CityErrorController.java

```
@Controller
public class CityErrorController implements ErrorController {

    Logger logger = LoggerFactory.getLogger(CityController.class);

    @RequestMapping("/error404")
    public String throwerror404(){
        logger.error("[CityErrorController] Page unavailable");
        return "error404";
    }

```

Figure 4 - Sample of one of the Security Hotspots Identified

Refactoring the code not only improves its strength and maintainability, it also enables the programmer to understand certain conditions that would be overlooked otherwise.

Fortunately SonarQube gives specific details on the problems it flags – such as a comprehensive description as to how it has been recognized as a problem as well as how to solve it. This information was then used to promote the process of fixing the most serious problems discovered and, ideally, lowering the technological debt.


New Code
Since May 13, 2021
Started 18 minutes ago

Overall Code

0 Bugs — Reliability A
0 Vulnerabilities — Security A
0 Security Hotspots — Reviewed — Security Review A
20min Debt — 3 Code Smells — Maintainability A
54.1% Coverage on 232 Lines to cover — 13 Unit Tests
0.0% Duplications on 569 Lines — 0 Duplicated Blocks

Figure 5 - Dashboard results after fixes

As anticipated, after resolving some of the more serious code smells and checking the security hotspots, the technical debt listed was reduced to just 20 minutes and three code smells.

The result obtained for code coverage, recorded as 54.1 percent, almost immediately raises a warning sign. Coverage, as described in theoretical classes, is a significant measurement, but that is not absolute.

The low stated percentage is attributed to the fact that some of the code created was never used by any test instruction because it does not require testing (for instance, the methods related to Thymeleaf, and the exceptions recorded while accessing the external API). Since achieving 100 percent coverage is rarely our target, this value is appropriate in the current situation.

3.5 Continuous integration pipeline

Continuous Integration is one of the practices that bridges the Dev/Ops division. Implementing a CI pipeline increases overall code consistency, lowers risk, and guarantees that the software is still theoretically deployable, even during production.

GitHub Actions: Java CI with Maven was used to create a basic CI pipeline. Its primary goal was to simplify the software development workflow, ensuring that the program was built properly and that all tests passed.

The workflow's YAML configuration file is present in the repository, and it specifies that the software should be built and the tests executed on any push or pull request on the repository. If all jobs are successfully completed, the workflow is set to 'Passing.'

Figure 6 - GitHub Actions Badge, after workflow analysis

4 References & resources

Project resources

- Git Repository [https://github.com/itskikat/air-quality_tqs-hw] (private repository, access was granted to the professor)
- Video Demonstration [https://youtu.be/b2jDpGj1cEE]
- API Documentation [http://localhost:8080/swagger-ui.html]

Reference materials

https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html
https://springfox.github.io/springfox/
https://github.com/stleary/JSON-java
https://medium.com/swlh/getting-json-data-from-a-restful-api-using-java-b327aafb3751
https://github.com/robolectric/robolectric/issues/3538
https://commons.apache.org/proper/commons-collections/apidocs/org/apache/commons/collections4/map/PassiveExpiringMap.html
https://aqicn.org/json-api/doc/ (World's Air Pollution API Documentation)

5 Appendix I

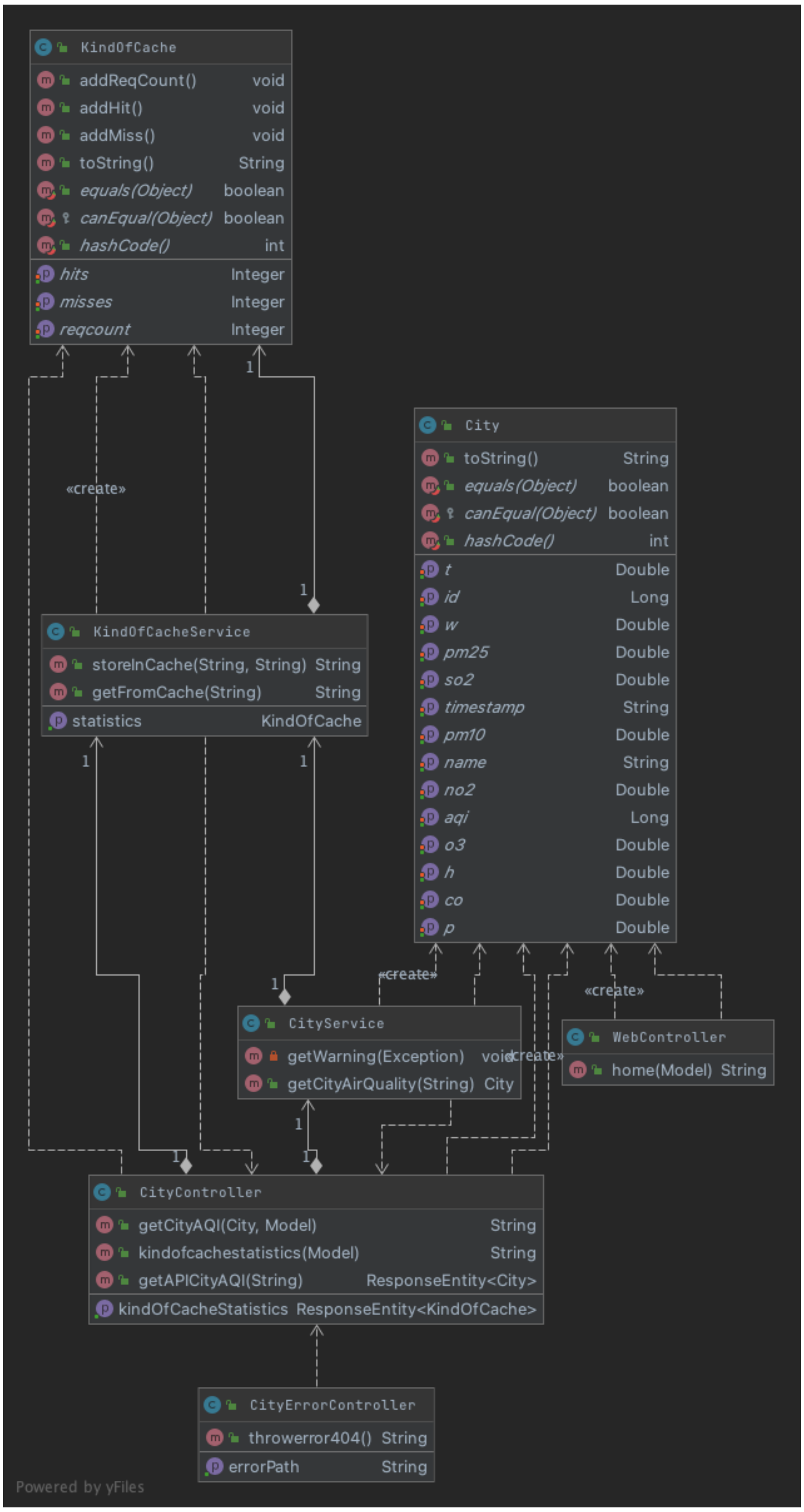


Figure 7 - System Architecture Diagram