HW1: Mid-term assignment report

Pedro Souto [93106], v2021-05-15

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 application developed uses an air quality API and allows users to search for a city and get the related air quality stats (air quality index and average of gases in the air) for the upcoming days. Various tests were developed to assure the correct functioning of the system.

1.2 Current limitations

- The application only allows users to search for the name of the city, in the future the users should be allowed to search for coordinates, and other data;
- The optional features were not implemented;
- The user interface could be improved;
- More tests could be added.

2 Product specification

2.1 Functional scope and supported interactions

As mentioned previously this application is intended for users who want to discover air quality stats related to a city.

All the users have to do is introduce the city name in the search box and they will get the stats (if the city exists, else they will get prompted to search for a city), plus, the application has special endpoints that allow developers to get this data in a JSON format for their own use. It is also possible to get statistics related to cache (hits, misses and number of requests).

2.2 System architecture

The user can interact with the application essentially in two ways:

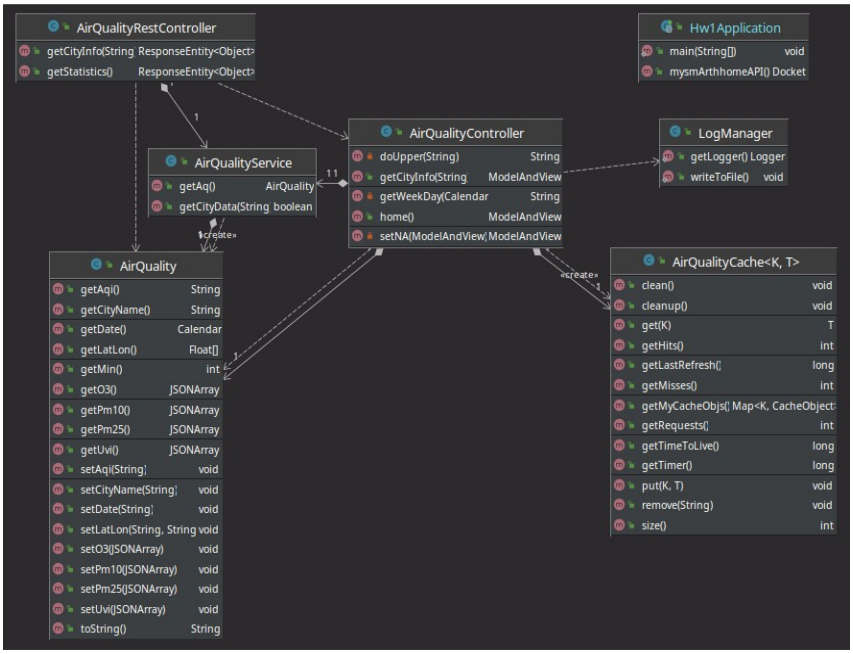
- The web interface
- The API

In the web interface, the user is presented with a web page, this page is shown using the AirQualityController (which has the @Controller notation) and thymeleaf. Its logic is processed through the AirQualityService (which has the @Service notation), here the majority of the business logic is performed, for example deciding to request data from the API or get it from the cache.

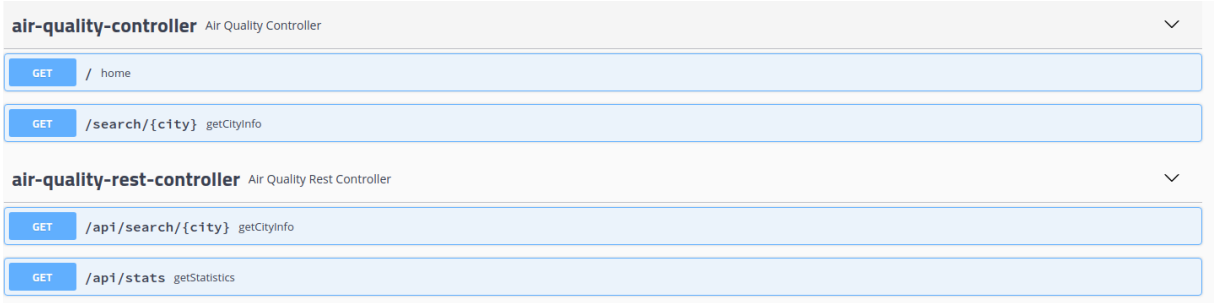
The AirQualityCache is a class that basically consists on the use of a hashmap to store the city name as key and the AirQuality object as value. This class contains three counters: the requests which as the name implies saves the number of times the user requested any endpoint, the hits which saves the number of times the data was able to be obtained from cache and the misses which saves the number of times the data was not in cache and had to be requested directly from the API. This data is of course used for statistic.

Regarding the API, the logic is basically the same except it uses the AirQualityRestController (which has the @RestController notation), and doesn't use thymeleaf, as we don't need to show the user an interface but simply return the requested data in json.

The project was developed using Spring Boot, Thymeleaf and HTML, CSS, JS.



2.3 Endpoints



3 Quality assurance

3.1 Overall strategy for testing

The strategy used was to after implementing all the features verify their correct functioning with tests.

- JUnit for cache testing;
- JUnit, Hamcrest and RestAssured for the API's integration test;
- Junit, Hamcrest, MockMvc for the API's unit test;
- Selenium for the web interface tests

3.2 Unit and integration testing

>Unit Tests for the cache (AirQualityCacheTest):

- Adding to cache
- Remove from cache
- Get from cache
- Cache size
- IsEmpty
- Expired after TTL ends
- Requests number
- Hits number
- Misses number
- Clean the cache
- TTL with invalid arguments

>Unit Tests for the API (AirQualityRestControllerTest)

- When city exists and is in cache
- When city exists and isn't in cache
- When city doesn't exist
- Get Statistics

>Integration Tests for the API (AirQualityRestControllerItTest)

- City exists and has the correct data
- City doesn't exist and has the correct data
- Statistics has the correct data

3.3 Functional testing

>Tested manually multiple times writing different cities in the search bar and navigating through the endpoints

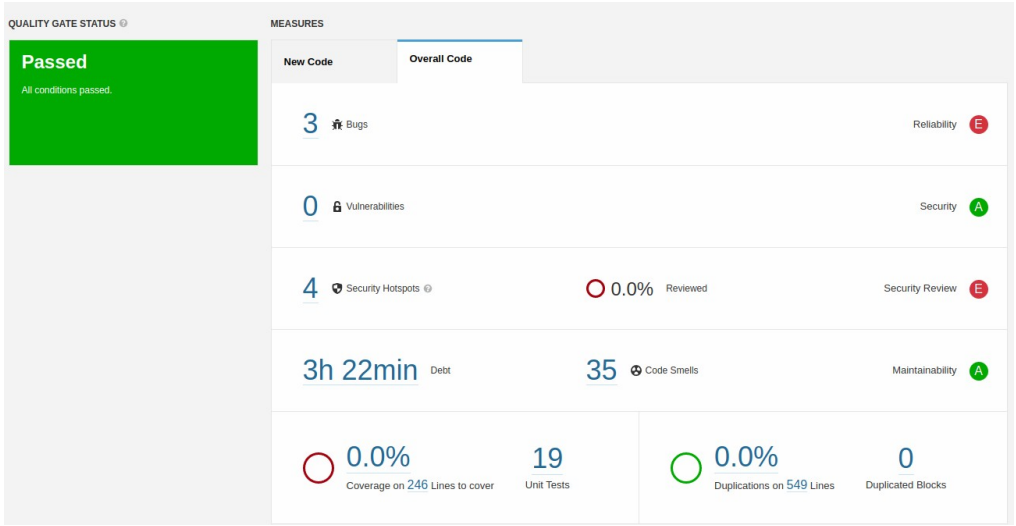
>Selenium Test for valid city (Sel_CityExists)

- Writes a city (in this case oPorto) in the search bar and clicks the button "Go". Verifies that the city name is oPorto and that the latitude and longitude are correct (as they are get from the API).

>Selenium Test for invalid city (Sel_CityNotExists)

- Writes a city (in this case asd) in the search bar and clicks the button “Go”. Verifies that the text that appears is to prompt you to write a city and that the latitude and longitude are both null.

3.4 Static code analysis



SonarQube was used for static code analysis.

The bugs (2 major and 1 blocker) are related to the cache class, which can’t be fixed without rewriting the whole class, nevertheless the bugs mentioned were not considered important by me, as the class does what it’s supposed to do correctly (though maybe not by the best methods), and applying these fixes without rewriting everything is nearly impossible.

The security hotspots are related to printStackTrace() in exception handling and log injection through the filehandler in the logger class, again I don’t think these are very important.

The code smells were mostly related to renaming variables and use of the logger with concatenation (apparently the built-in formatting should be used instead).

The coverage of the code is 0.0% which I couldn’t really find a reason for, nevertheless the quality gate status was Passed successfully.

4 References & resources

Project resources

- The video demo and project code are both in the Git Repository: https://github.com/PedroMGSouto/TQS_HW1

Reference materials

- https://crunchify.com/how-to-create-a-simple-in-memory-cache-in-java-lightweight-cache/
- https://www.baeldung.com/thymeleaf-iteration
- https://www.thymeleaf.org/doc/articles/springmvcaccessdata.html
- https://www.themezy.com/free-website-templates/128-steel-weather-free-responsive-website-template

- https://stackoverflow.com/questions/15758685/how-to-write-logs-in-text-file-when-using-java-util-logging-logger
- https://www.youtube.com/watch?v=HHyjWc0ASl8