HW1: Mid-term assignment report

Diogo Oliveira Bento [93391], 5/9/2021

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.

The developed Spring project displays Air Quality statistics upon being provided with coordinates, one can get data by supplying coordinates to the API, clicking a button in the frontend or clicking a Google Maps instance embedded into the frontend.

All query results are cached for 30 seconds, and the backend uses OpenWeatherMap and WeatherBit as data providers in that order.
The cache's performance can be inspected both in text report and in JSON form.

1.2 Current limitations

The APIs used provide different data, the data the implemented service returns is the intersection of the data provided by both services, and therefore somewhat limited.

A weather data object only contains latitude, longitude, AQI, CO levels, 03 levels, PM10 metric and PM2.5 metric.
The cache does not have any size limit.

2 Product specification

2.1 Functional scope and supported interactions

This application provides Air Quality data based on coordinates, with a map interface to increase ease of use and has a REST API, making it useful for anyone that needs to check Air Quality, including programmers in need of an API to use.

2.2 System architecture

The application queries other services via RestTemplate and parses the response's body with the org.json library.

The cache is based around a HashMap that maps requests to an object containing response data and a timestamp.

On request the Controller queries the Cache object who then queries the external API clients in order if no valid data is found in cache.

The frontend uses Javascript's **fetch** function to update itself based on data given by the API, there is no templating involved.

The application can be said to follow MVC architecture.

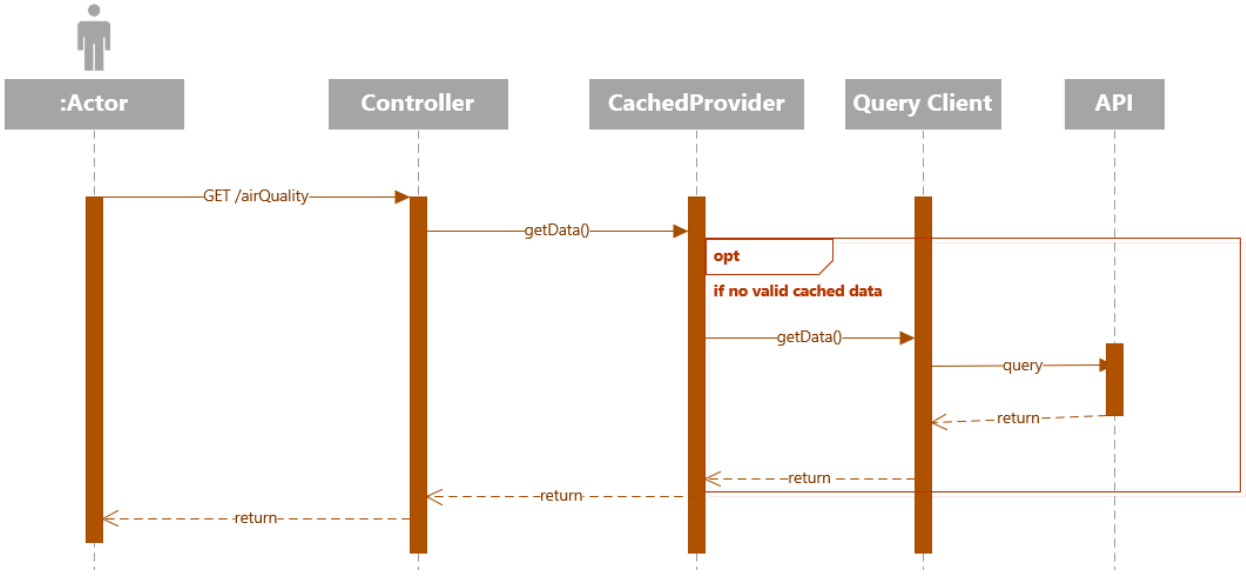

Fig 1: Request Sequence

2.3 API for developers

The API can be queried for current Air Quality data at **/airQuality** and requires **the URL parameters l**at and lon;

Example Query: <server>/airQuality?lat=0&lon=0

There are also 2 endpoints related to the performance of the cache, **/cacheReport** provides a string output detailing current cache performance and **/cache** provides a similar report in JSON format.

3 Quality assurance

3.1 Overall strategy for testing

A rough outline for the components was written before any of the tests, which were then written to cover the expected behavior of the application including features that had not been included at the time.

The first tests written were for the query clients and cache, and were implemented with Mockito and mock injection, followed by API tests using @WebMvcTest and RestAssuredMockMvc and Selenium tests using Chrome, which was automatically managed by SeleniumJupiter.

Cucumber and test containers were not used for this project.

3.2 Unit and integration testing

The individual query clients and cache client were tested using Mockito.

The clients were tested for throwing errors for bad parameters and data desync (receiving latitude and longitude significantly different from their queried values), and mapping JSON to data objects.

The cache service was tested for consistent tracking metrics, cache timeouts and normal function, and backup API usage.

The controller's tests used @WebMvcTest, @Mockbean and RestAssuredMockMvc to test the controller's behavior on being given null from the cache, bad input parameters, successful queries, and echoing cache status properly.

3.3 Functional testing

There are Selenium tests for the cache feature, querying the API via search button with invalid coordinates, and querying the API via map click event.
The cache is tested before and after to make sure the values change according to the events the take place during the tests.

3.4 Static code analysis

A SonarQube Docker container and SonarQube IntelliJ integration were used for static code analysis were used for static code analysis.
Poor privacy settings were a consistent code smell, due to IntelliJ automatically setting Test classes as public, methods were often public for no reason as well due to lack of care.

As of this project's delivery there are no unhandled code smells.

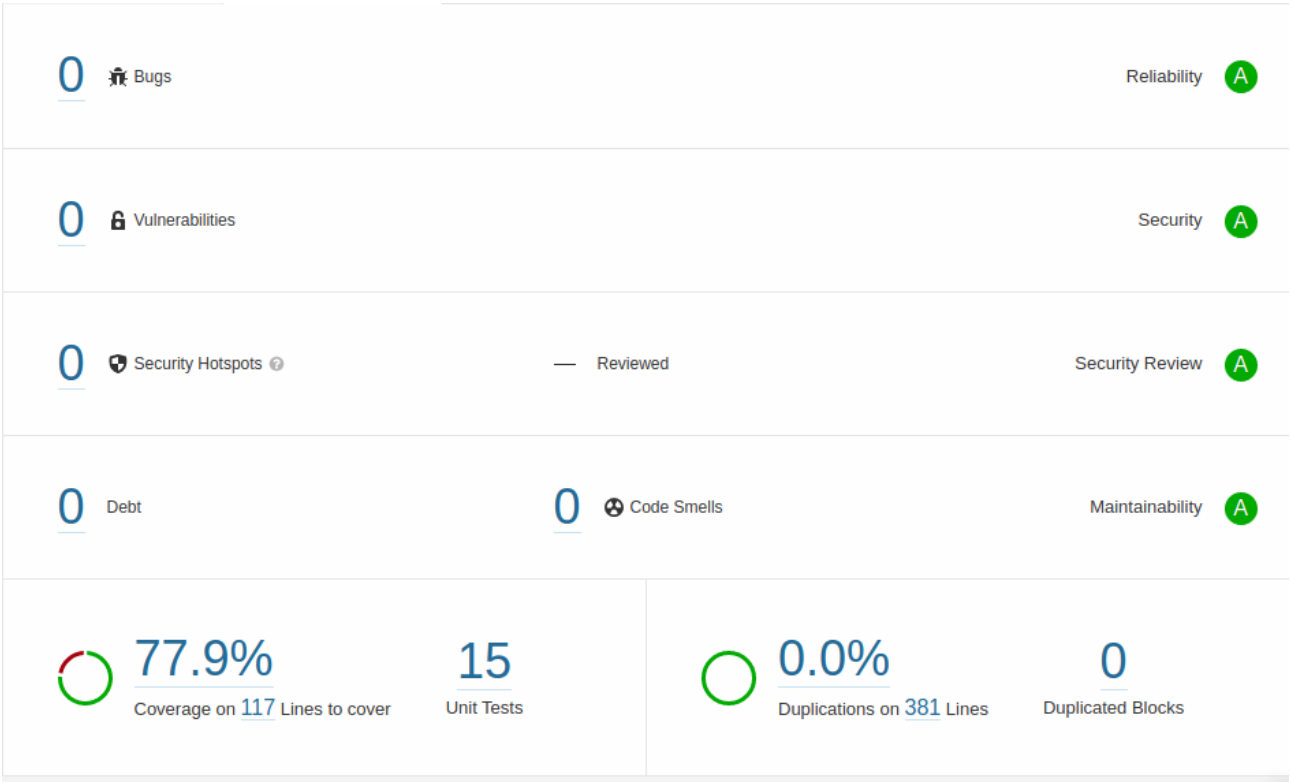

Fig 2: SonarQube dashboard

The majority of uncovered lines is due to the usage of **@Data**, the rest is mostly false positives. It seems the only actual uncovered line is **CachedInfoProvider**'s clear method, which is deemed trivial.

Fig 3: Coverage report

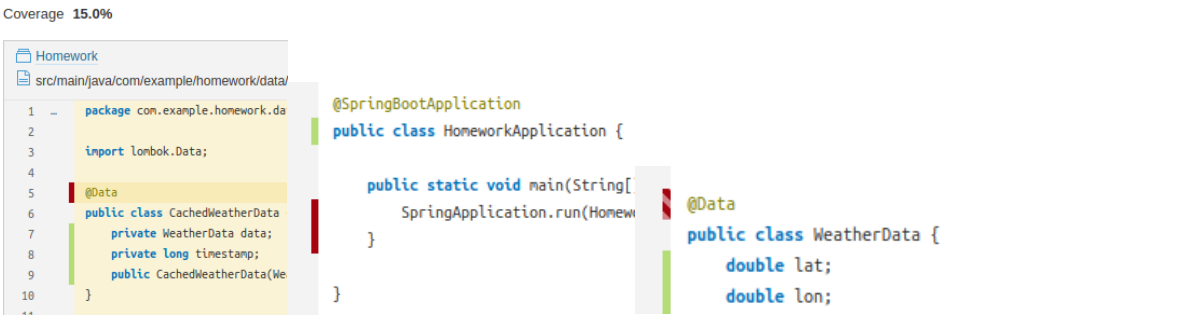

Fig 4,5,6: Misleading Coverage Metrics

3.5 Continuous integration pipeline

An attempt at using SonarCloud was made however due to the code repository being a multi-project repository integration was proving tricky, so a SonarQube Docker container was used instead.

4 References & resources

Project resources

- Frontend demo available in the Homework folder of my TQS Repository

Reference materials

- OpenWeatherMapAPI
- WeatherBitAPI
- RestTemplate – Useful for querying
- org.json library – Easy to use JSON parser
- SeleniumJupiter – skip web driver lifetime management