HW1: Mid-term assignment report

Isabel Alexandra Jordão Rosário [93343], v2022-05-01

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.

My project is presents COVID-19 incidence data through an interactive map that the user can explore. As such, it is called “COVID-19 Incidence Data: An Interactive Map” and its purpose is to provide useful information in a dynamic, captivating way.

1.2 Current limitations

As it stands, the frontend of the project simply displays the interactive map itself and gives data from the last 3 days for each country. Ideally, clicking on a country would redirect the user to a new page with more detailed information about said country, displayed through charts and infographics. The point would be to keep data visualization as the main priority and provide the user with an interesting and alluring experience.

2 Product specification

2.1 Functional scope and supported interactions

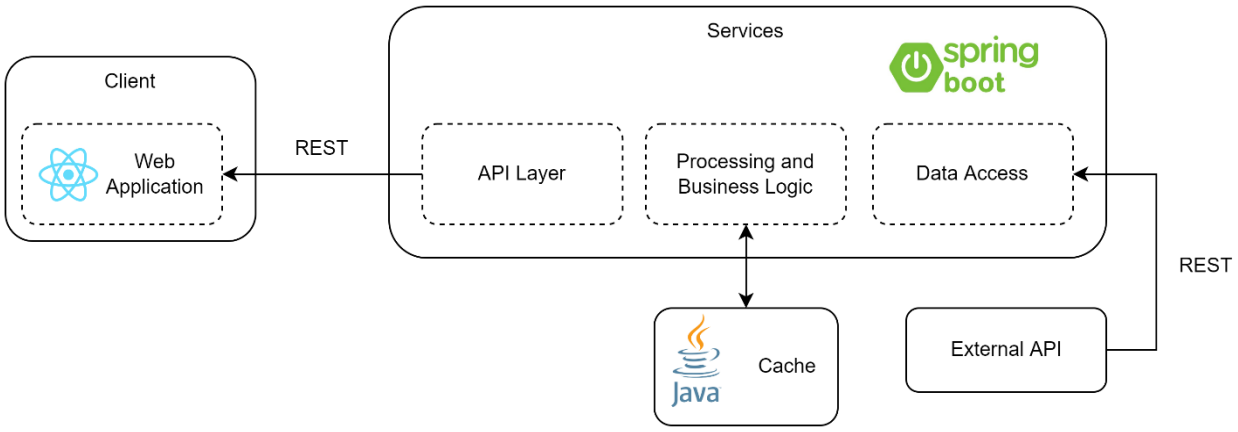
There is only one actor involved in the experience: the user of the website, who is any individual interested to know about current COVID-19 incidence data.

As such, the main usage scenario is one in which the user goes onto the website and uses the interactive map to get information by hovering the desires countries and reading the information from the toolkit pop-up that appears.

2.2 System architecture

The system architecture is as follows: the frontend was developed with ReactJS (using the React Simple Maps package for the interactive map in particular), and the entire backend of the application was developed with Spring Boot, using simple Java utils for building the cache. The contact between both APIs and the application is done in a RESTful manner.

The architecture diagram is as follows:



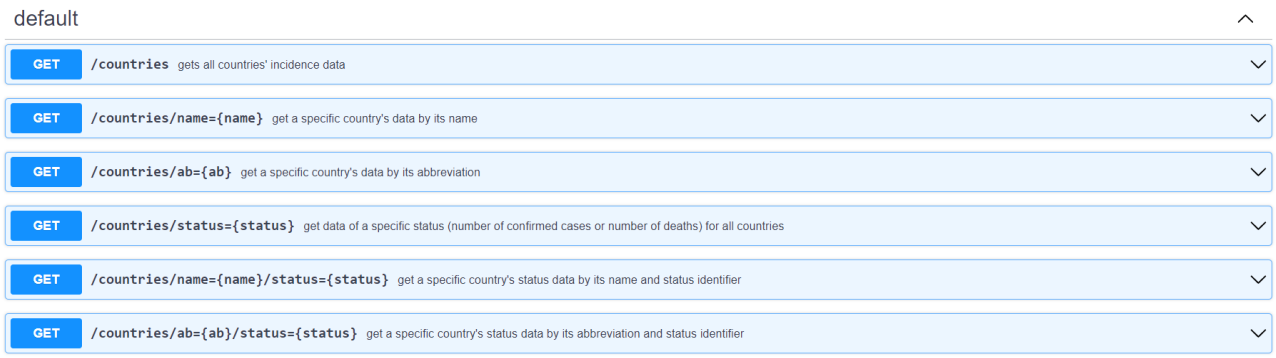
2.3 API for developers

Firstly, my API allows developers to obtain COVID-19 incidence data from the last 3 days for all countries. It also allows for filtering this information by status, meaning by either the count of the number of deaths or the count of the number of confirmed cases for all countries.

It is also possible to obtain information for both statuses for a specific country, specified by either country name or abbreviation, as well as filtering of those statuses for the same individual country.

Finally, there is a separate endpoint that exposes cache usage statistical data.

The API endpoints are as follows:



3 Quality assurance

3.1 Overall strategy for testing

For the testing phase of the project, I started by writing unit tests that I thought were important for the testing of the cache. This included making sure the cache was empty on construction, if upon placing a country there, it could be successfully retrieved, etc.

After this, I moved on to the development of service-level tests, where I tested the service class of my application. For this, I mocked the client (that submitted requests to the external API) in order to reduce my testing domain to the service class alone.

Finally, I wrote some integration tests to see if the whole structure (service, client and cache) were cooperating correctly.

Unfortunately, given the nature of my final frontend product, I found it quite challenging to create functional tests using Selenium for the application website. A more detailed explanation of the difficulties faced is explained below.

3.2 Unit and integration testing

As stated before, the unit tests that were created focused on testing the cache, as I did not use any converters or utils that required further testing. These tests can be found in the *A_UnitTests.java* file. The unit tests are explained below:

- **emptyOnConstruction()**: test if, upon construction, the cache is empty;
- **whenInsertCountry_ThenCacheContainsCountry()**: test if, upon inserting a country in the cache data structure, it is certain that the cache contains said country;
- **whenInsertMultipleCountries_ThenCacheContainsCountries()**: similar to the previous test but with a set of multiple countries;
- **whenInsertCountryAndWaitTTL_ThenCacheDoesntContainCountry()**: test if the cache's TTL feature is working according to plan, that is, if a country is erased from cache when its TTL has elapsed;

- **whenInsertCountryAndRemoveIt_ThenCacheDoesntContainCountry()**: test if, upon inserting a country in the cache data structure and immediately removing it, the country is not contained in cache;
- **whenInsertDuplicateCountry_ThenCacheContainsLastVersionOnly()**: test if, upon inserting a country in cache which was already there, the cache data structure keeps only the most recent version of said country;
- **whenEmpty_ThenGetItemReturnsNull()**: test if, upon trying to retrieve the cache's items from the data structure without having inserted any into it, the return value is *null*.

Important note: for the fourth unit test, where it is necessary to wait a certain amount of time for the TTL timespan to expire, it is useful that the TTL value is small, such that the test will not take an overly long time to run. For that purpose, such a value can be set to 10 seconds in the code. However, given the nature of the application's frontend, where it is very easy to scan through numerous countries in a very short amount of time, it would be better that such a TTL value would be set to something like 10 minutes, so that the application does not make constant calls to the external API, which is a free public source of data that should not be overstrained. As such, in the *Cache.java* file, both lines of code (for a TTL of either 10 seconds or 10 minutes) are visible, and it is up to the developer to comment one or the other according to the current needs. By default, the TTL value is set to 10 seconds for testing purposes, but I would ask the teacher to change it back to 10 minutes if the intent is to use the application itself, as opposed to just running the tests.

In order to test the service class, the following tests, which can be found in the *B_ServiceLevelTests.java* file, are explained below:

- **testGetCountryName()**: test if the service can correctly process a country by its name identifier;
- **testGetCountryAb()**: test if the service can correctly process a country by its abbreviation identifier;
- **testGetCountryWithTerritories()**: test if the service can correctly process a country which has external territories in addition to its main land;
- **testGetCountryStatus()**: test if the service can correctly process a country's status;
- **testGetAllCountries()**: test if the service can correctly process a list of countries;
- **testGetAllCountriesStatus()**: test if the service can correctly process a list of countries' statuses;
- **testCacheUse()**: test if the service can correctly make use of the cache, that is, if it will use the cache whenever it contains the necessary data for a request, instead of issuing a new call to the external API.

3.3 Functional testing

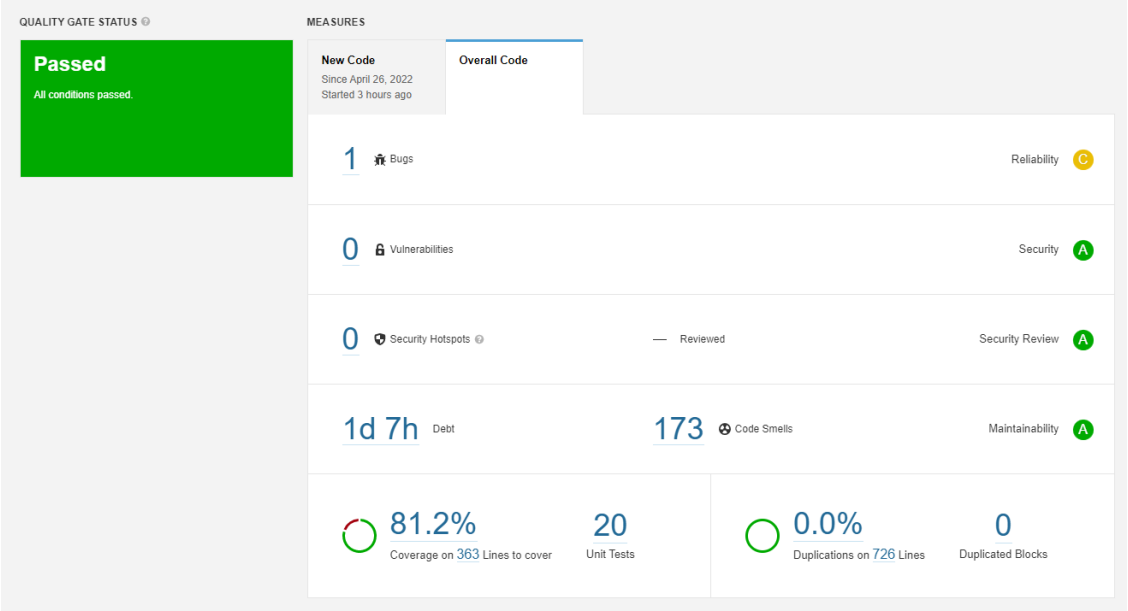
As expressed in the overall strategy for testing section, building functional tests for the interactive map with Selenium proved unsuccessful for me. Firstly, the package I used for displaying the map does not assign separate HTML elements for each country, so I had trouble, for example, trying to assert if a specific country had been hovered in order to display its information.

Furthermore, each country's data is displayed in a small pop-up window that disappears when the mouse leaves the hover area for said country. This window is also not an element that can be hovered with the mouse, so the second challenge was that I could capture it in the Selenium IDE, in order to verify its contents.

Sadly, functional tests were not developed for the application.

3.4 Code quality analysis

In terms of static code analysis, I used SonarQube for a local analysis of the application's folder. The results were as follows:



Seeing as the application code passed the quality gate defined by SonarQube, I did not find it necessary to make further code alterations.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/rospuye/TQS_93343/tree/master/HW1
Video demo	https://github.com/rospuye/TQS_93343/blob/master/HW1/report/presentation_video.mp4

Reference materials

- React Simple Maps package: https://www.react-simple-maps.io/
- External API used: https://github.com/M-Media-Group/Covid-19-API