deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

HW1: Mid-term assignment report

Raquel da Silva Ferreira [98323], v2022-05-02

Introduction

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 project, Covid Tracker, consists of an application that presents statistics (cases, deaths and tests) related to COVID-19 for a certain country. It is composed of a web app, a REST API that exposes COVID data and cache details and some tests.

Current limitations

Current limitations:

- Currently the web app uses only one external API. This means that if the API is down, there is no data to present.
- The external API has information on the continent of the countries. This could be used to organize data by continent.

- The external API has available the date of the statistics which could be used to implement the statistics of the last n days (e.g., in a graph), but to do this, since the API only has available the search of a specific day or all history, it would have to be done with n requests.
- The project has a performance issue when choosing a day to see the COVID data. Sometimes it takes longer than is supposed to. I think it may be related to the library used with ReactJS, but it is something to improve.

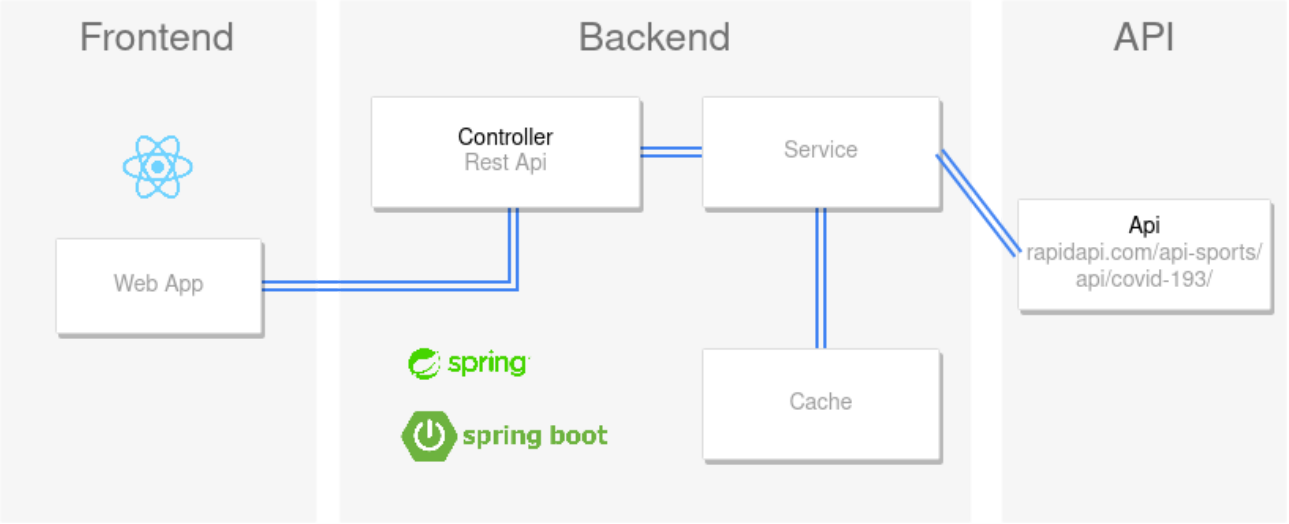
Product specification

Functional scope and supported interactions

Covid Tracker allows users to search for current or previous COVID-19 data, for a specific country (cases, deaths and tests) or globally (cases and deaths). The main usage scenarios are:

- check global data
- choose a country and check the current statistics
- choose a country and check previous data

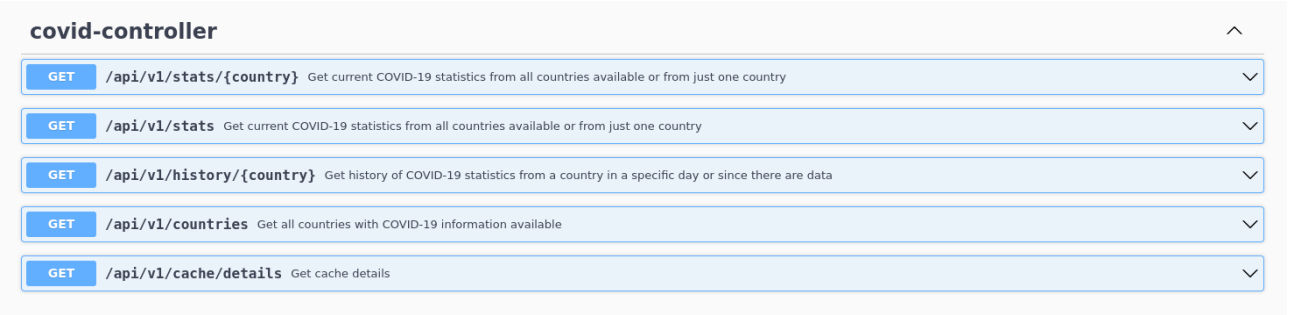
System architecture



For the frontend of the project was used **ReactJS** along with the library **Axios** to create the HTTP requests to the REST API, that was built (and also the rest of the backend) using **Spring Framework** and its tool **Spring Boot**.

API for developers

The base URL of the REST API of the project is **localhost:8080/api/v1** and its documentation can be seen at **localhost:8080/swagger-ui/index.html**.



Quality assurance

Overall strategy for testing

The overall strategy I tried to follow for testing was TDD. I started with the controller followed by the service. The cache was implemented before its tests because I was not familiar with the way a cache worked.

For the functional tests I used a BDD strategy with Cucumber and Selenium IDE.

Unit and integration testing

Unit tests were used on the cache behavior and on the class used for the HTTP requests (utils). They were also done for the service with a Mock of the cache and the utils class.

The controller was tested with integration tests using MockMvc.

Functional testing

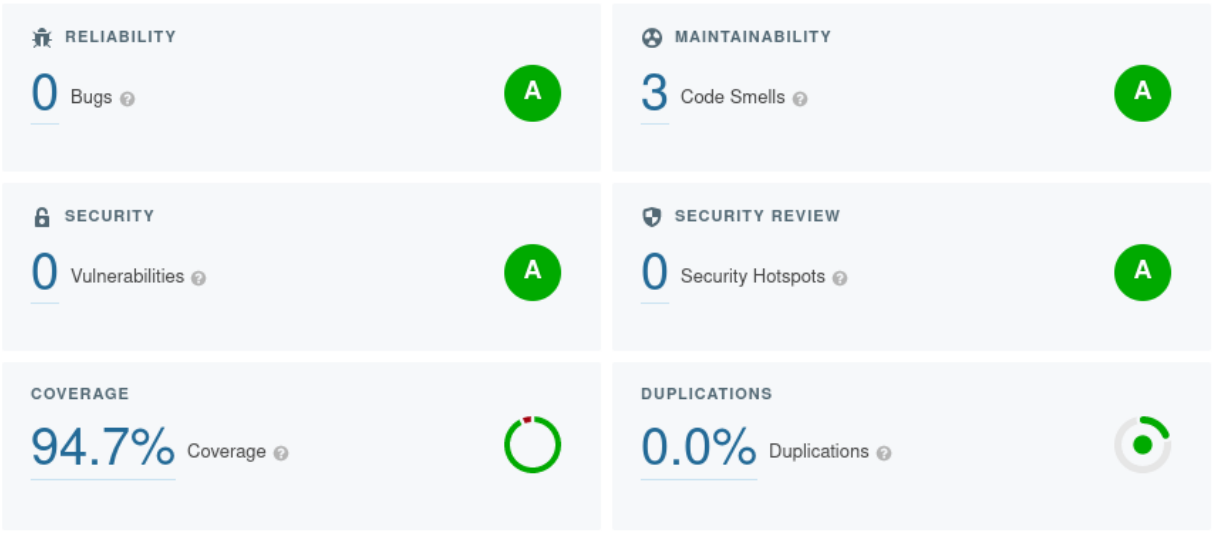
For the functional testing, I considered all the main usage scenarios. The scenarios were written with **Cucumber** and tested with **Selenium IDE**. This allowed me to replicate the user's behavior when using the application.

Code quality analysis

The code quality analysis was done with SonarCloud since the beginning of the development process. This allowed me to improve the code along the development instead of all at the end. I also used a SonarLint extension on Visual Studio Code helping me correct code smells.

There were some code smells that I wouldn't address if it weren't the linter and SonarCloud, like the interruption of threads when catching InterruptedExceptions, the formatted messages in the logs and the classes and methods of tests should not have public visibility, among others.

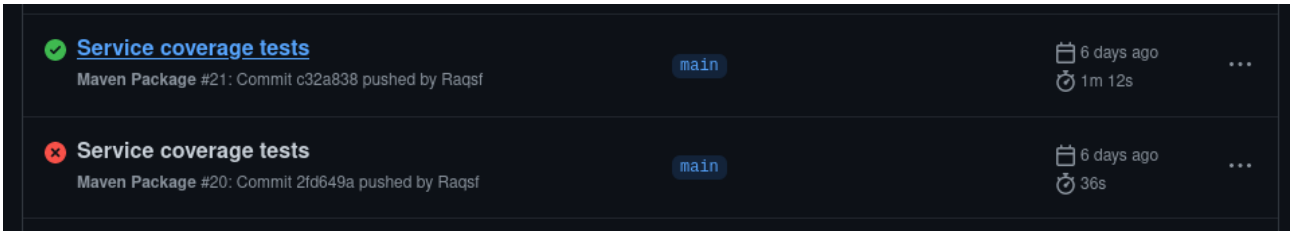
To maintain the quality of the code I assured that all of the 4 measures of the SonarCloud had A status and the coverage of the code was reasonable, ending up achieving the default quality gate of SonarCloud.



One of the code smells says that a constructor should have a maximum of 7 parameters. Something that I would not remember if it was not for SonarCloud, but I did not correct it because I let it stay with 8 parameters, the ones the external API sent.

Continuous integration pipeline

I used Github Actions to implement a continuous integration pipeline. This allowed me to ensure that whenever I pushed code to the repository it had no errors and passed the tests. When there was an error I could correct it right away (as shown in the image) and detect it sooner than later.



In the CI pipeline was also the SonarCloud workflow, informing me of the code smells, coverage and bugs that my code had when I pushed it into the repository.

References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/Raqsf/tqs_98323
Video demo	https://github.com/Raqsf/tqs_98323/blob/main/HW1/simplescdemo.mp4

QA dashboard (online)	https://sonarcloud.io/summary/overall?id=Raqsf_tqs_98323

Reference materials

Frontend:

- Axios - https://axios-http.com/
- Material UI - https://mui.com/pt/

Cache:

- https://medium.com/analytics-vidhya/how-to-implement-cache-in-java-d9aa5e9577f2
- https://crunchify.com/how-to-create-a-simple-in-memory-cache-in-java-lightweight-cache/