HW1: Mid-term assignment report

Fábio Alexandre Ramos Martins 98119, 2022-04-072

1.1 Overview of the work 1
1.2 Current limitations 1
2.1 Functional scope and supported interactions 1
2.2 System architecture 2
2.3 API for developers 2
3.1 Overall strategy for testing 3
3.2 Unit and integration testing 3
3.3 Code quality analysis 4

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 product was a simple web application, with the name of IinciVID19. The goal of this product is to provide information about the incidence of the COVID-19 virus throughout the countries, being this information about the current date or a date inserted by the user. It has a simple layout and it's not meant to be used in mobile devices.

1.2 Current limitations

Like it was mentioned before, this web application is not suitable for mobile devices, due to being nonresponsive. This was not implemented due to the lack of time and for being of lesser need. It is a work for the future.

Another lacking aspect is the lack of frontend testing.

One last aspect of this application that should be corrected is the fact that the users need to input the country name without having any information about its format. For example, if one wants to search for the United States, the user will have to enter *USA* or *usa* instead of United States of America. The user will not know that an error occurred in this case.

2 Product specification

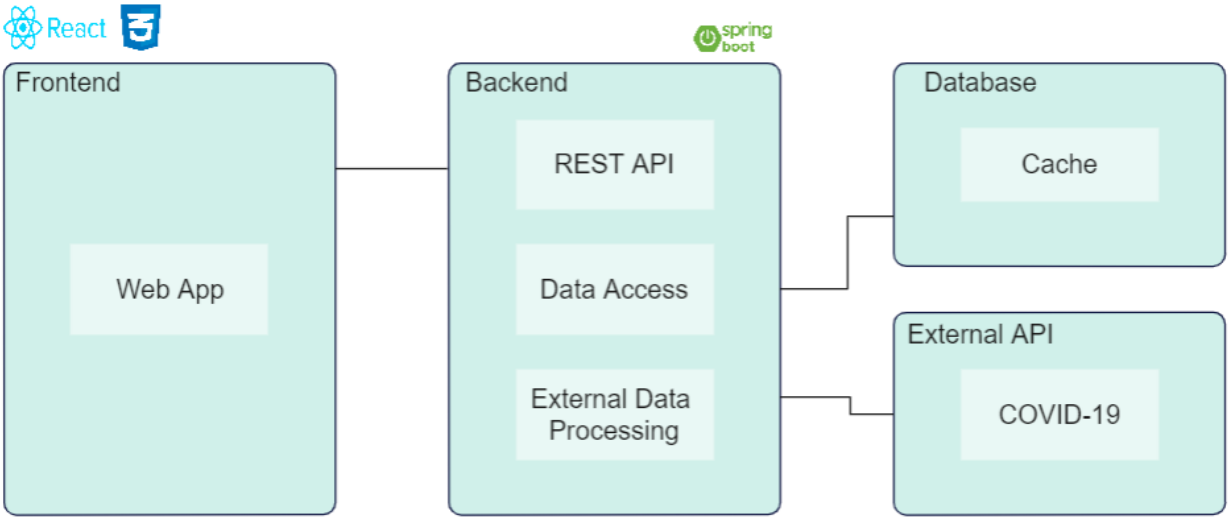
2.1 Functional scope and supported interactions

The main users of this app are people that wish to have information about the statistics of the COVID-19 pandemics throughout the countries. It facilitates the access to information

about the current date's data provided by the API used or the access to information about the last data collected at the end of a certain day (selected by the user).

2.2 System architecture

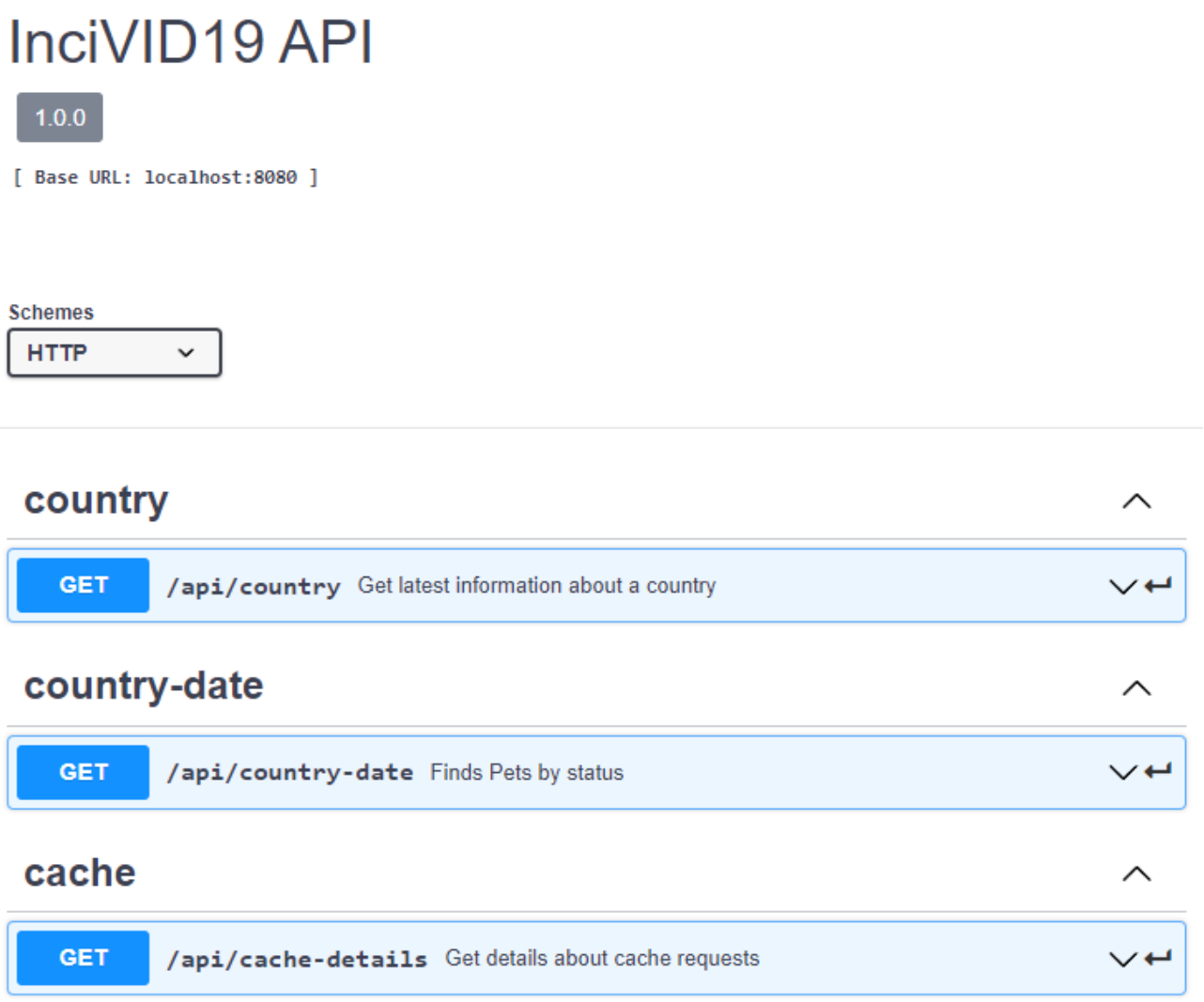
For the frontend, the Web App was developed using ReactJS, a Javascript framework, and CSS. The backend, where all the data access and processing happens, was developed with Spring Boot, a Java framework. The data persistence is done in runtime using a maximum capacity cache, that was developed using Java. The external API used is the COVID-19 API.



2.3 API for developers

The endpoints of this API return two types of objects (converted to JSON objects). A Country object and a CountryCacheDetails object. The Country object is returned from the COVID tracking data endpoints, which are the represented in the image bellow. The first needs one query parameter, the *name* parameter, that defines the name of the country to be returned. The second needs two query parameters, the *name* parameter, and the *date* parameter, being the last one in the '*yyyy-mm-dd'* format.

The last endpoint is the one that returns a CountryCacheDetails object, which contains the number of requests made to the cache, as well which of these were hits or misses.



3 Quality assurance

3.1 Overall strategy for testing

For testing, the methodology used BDD, although I didn't integrate Cucumber. First, the method was roughly developed in order to understand what I wanted it to be. Then, I developed the tests with the behavior I wanted the method to have and then, if the test failed, I would refactor the code so that it would produce the desired results. The tools used for testing were the ones used in previous classes.

3.2 Unit and integration testing

The tests made were for the three crucial aspects of the API: the controller, the service, and the cache, which served as the repository. Starting with the cache. This one's tests were unit tests, just like it was done in classes to the repository. Here I tested the capacity for the cache

to retain the items that were put in it, as well as the capacity to generate the details (requests, hits, and misses) needed for the API. It was also tested the capacity to erase the first item in cache, which should be the first item put there or the least accessed one.

The service was tested with mocks. Here the need was to verify if the cache was being called when the service's methods were used and if they were returning the correct result.

Finally, the controller was tested with Web model-view-controller, or Web MVC for short. Here the intention was to test whether if the service's methods were being used when the API endpoints were called. It was also tested if the endpoints were actually working or not.

3.3 Code quality analysis

The code analysis was done via Continuous Integration, using the GitHub Actions' workflow, SonarCloud. The last evaluation revealed that the code did not pass the quality gate, this due to two factors: the vulnerabilities and the coverage. Regarding the vulnerabilities, it is not passing (even though it has a B rating) due to the logging, which was not fixed due to it being a minor problem. An interesting thing found with this tool was that the logger has a built-in formatter, which makes *String* concatenation become a smell when done inside the logger.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	fMart8421/tqs_98119
QA dashboard (online)	sonarcloud

Reference materials

The external API used in this project was the COVID-19 API provided in the RapidAPI website.