Homework 1 - Covid API

Teste e Qualidade de Software

Filipe Gonçalves, 98083

April / 2022

Universidade de Aveiro - 2021/2022

Index

Introduction

Overview

This report presents the first homework project for TQS, covering both frontend features and tests and quality assurance of the software. It will explain the strategy adopted and offer some evidence of the results obtained, as well as showing which tests were made and the reason behind it, and how they behave in the project.

This project is a small web app built with the Spring Boot framework, using an external API to execute all the required tasks.

API: https://rapidapi.com/api-sports/api/covid-193/

It uses two of the three possible endpoints of the API: **GET** history and **GET** statistics. The first one, given a Location, and a specific day, the API will convert the arguments into a JSON containing the data of said location and said day. The later one, will give us a JSON containing similar data as for the other endpoint, but for every country or continent.

Current Limitations

Since the project relies on a single API, if it fails, then there is no other way of getting the desired results, unless they were already saved in cache.

There is also the problem that sometimes the API will return **null** in some values, which means that that value will get a default value of 0. In a similar case, even though a normal country/ continent appears in the general endpoint (**GET** statistics), it doesn't mean there is data in the other endpoint (**GET** history), defaulting the **null** values to 0.

The last limitation found is that even though the data shown should be of 2022, because the API is free to use and the focus on the disease has been dropping, the data from 2022 is almost **null**, meaning that it should be better to show the results of the given month and day, but changing the year to 2021, which i exactly what was made to ultrapass this obstacle.

Functional Scope and Supported Interactions

The main objective of this project is to show the Covid cases, deaths and tests, from the past 5 days of a given Location.

Interactions:

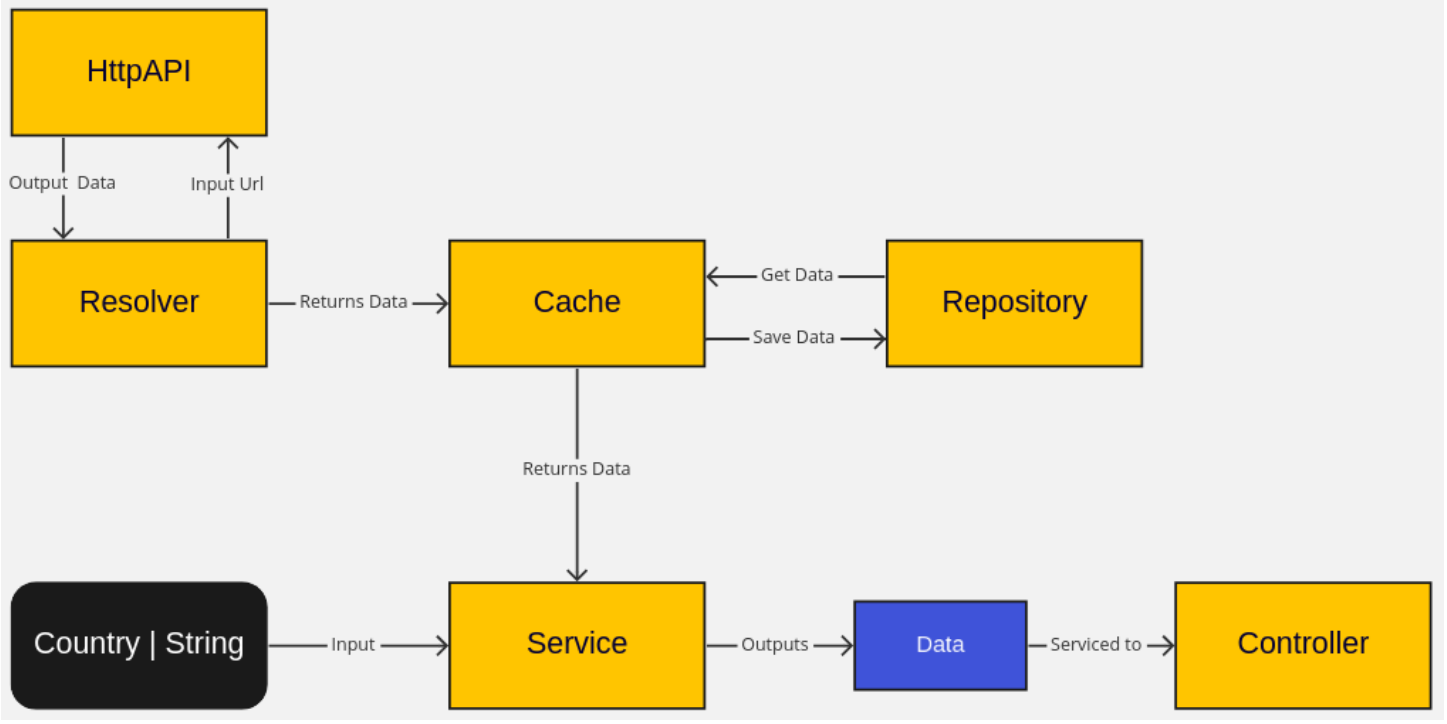
- User wants to know the covid situation of a location from the day before
- User wants to know the covid situation of a location from 5 days before

Frontend

The project frontend resides in a simple html file, **index.html**, where we can select the country/ continent from a dropdown. Then clicking on the button next to the dropdown, it will redirect us to another html file, **home.html**, where we can see the data from the country selected.

System Architecture

As this is a Spring Boot Application, it was also used Thymeleaf to create the frontend software and template languages. To simplify communications with the services, it was also created a Controller, with the Spring Boot Annotations.



With an inputed Country String, the Service will ask the Cache if the data is there, the cache will try to get the data from the repository, and if it's successful, the data is returned to the service, else the cache will prompt the resolver to get the data for the country and day. The resolver will get the data from the API with the help of the HttpAPI class and return the data to the cache, which will return to the service. Then the service will output the data to the Controller and show it.

While the Service and Cache are used for both Model classes: CovidData and CovidDataCountry, there are two Resolvers, one for each API call, which will translate the data from json to the specific Model. There are also two Controllers, one for dev statistics, APIController, and one for frontend purposes ViewController. The data from the service will go to one of them depending on the call.

As for frontend, there are two more Models used, CacheView and CovidDataRequestModel, which will only give us a better way to represent the data with Thymeleaf.

API for developers

In this project, we also built a REST API, with the @RestController Annotation from the Spring Boot framework, which can give us some additional data for developers.

Endpoint	Description
/api	Root
/api/all_data	GET all countries
/api/get_data{country}{day}	GET covid data from country for the day
/api/cache_statistics	GET cache statistic data (hits, etc.)
/api/cache_data	GET all data in cache
/api/get/country	GET all countries data in cache
/api/get/continent	GET all continents data in cache
/api/get/day	GET all days data in cache

Quality Assurance

Strategy for Testing

The strategy used in this project is to utilize Junit, Mockito, Selenium and Spring Boot MockMvc. It was also used slightly the TDD way of building quality software for some classes, but not in a general way.

Unit Testing

Cache

Getting data from cache, when data is in cache and when it's not:

- **getDataByCountry(String country, String day)**
  - testGetValidCountryData()
  - testGetInValidCountryData()
- **getAllData()**
  - testGetValidAllData()
  - testGetInValidAllData()

Saving data into the cache, when data is not in the cache and when it is:

- **saveDataCountry(CovidDataCountry data)**
  - testSaveCacheData()
  - testSaveCacheDataExistent()
- **saveData(CovidData data)**
  - testSaveCacheAllData()
  - testSaveCacheAllDataExistent()

Deleting data from the cache:

- **deleteDataFromCache(CovidDataCountry data)**
  - testDeleteCacheData()
- **deleteDataFromCache(CovidData data)**
  - testDeleteCacheAllData()

Getting data from cache, when data is expired, valid and invalid:

- **hasExpiredCountry(CovidDataCountry data)**
  - testGetExpiredCacheData()
  - testHasExpiredValidCacheData()
  - testHasExpiredInvalidCacheData()
- **hasExpiredCountry(CovidData data)**
  - testGetExpiredCacheAllData()
  - testHasExpiredValidCacheAllData()
  - testHasExpiredInvalidCacheAllData()

Service

Getting data from service, when data is cached and when it's not, and when the parameters, if needed, are right:

- **getDataByCountry(String country, String day)**
  - testGetValidCovidDataCached()
  - testGetValidCovidData()
  - testGetCoviDataCountryError()
  - testGetCoviDataDayError()
- **getAllData()**
  - testGetValidAllCovidDataCached()
  - testGetValidAllCovidData()

Testing API response and Exceptions:

- **getDataByCountry(String country, String day)**
  - testAPINotAvailable()
- **getAllData()**
  - testGetValidAllCovidData_Interrupted()
  - testGetValidAllCovidData_Error()

HttpAPI

Testing API response and Exceptions, when API does not respond, interrupts or gets an invalid url to parse:

- **dohttpget(String url)**
  - testGetUrl_APINotResponds()
  - testGetUrl_Interrupts()
  - testGetURL_Valid()

CovidResolver

Testing API connection, Interruptions, Exceptions and parse to Object Errors:

- **CovidDataCountryResolver**
  - testGetValidData()
  - testGetInValidData()
  - testGetBadRequest()
  - testDataToJsonErrorParse_Resolver1()
- **CovidDataResolver**
  - testGetValidAllData()
  - testGetInValidAllData()
  - testGetBadRequestAllData()
  - testDataToJsonErrorParse_Resolver2()

Repository

Testing Repository methods:

- **CovidDataCountryRepository**

➔ **findByCountryAndDay(String country, String day)**, will return an Optional Object with the data from the **country** in the day **day**

  - findByCountryAndDayTest()
  - findByCountryAndDayTest_BadCountry()
  - findByCountryAndDayTest_BadDate()
  - findByCountryAndDayTest_NotFound()

➔ **findAllByCountry(String country)**, will return a List with all the data from the **country**, in every day

  - findByCountryTest()
  - findByCountryTest_NotFound()

➔ **findAllByContinent(String continent)**, will return a List with all the data from the **continent**, in every day

  - findByContinentTest()
  - findByContinentTest_NotFound()

➔ **findAllByDay(String day)**, will return a List with all the data from that said **day**, for all countries

  - findByDayTest()
  - findByDayTest_NotFound()

- **CovidDataRepository**

➔ **findByCountry(String day), country)**, will return a List with all the data from the **country**

  - findByCountryTest()
  - findByCountryTest_NotFound()

Integration Testing

APIController

Testing developers endpoints, with valid set up, bad requests and good/ bad parameters:

- **GET /api/get_data**
  - testGetCountryData()
  - testGetCountryData_BadRequest_Country()
  - testGetCountryData_BadRequest_Date()
  - testGetCountryData_BadRequest_NoParameters()
  - testGetCountryData_BadRequest_OnlyCountry()
  - testGetCountryData_BadRequest_OnlyDate()
- **GET /api/cache_statistics**
  - testGetCacheData()
- **GET /api/cache_data**
  - testGetAllCacheData()
- **GET /api/get/country**
  - testGetData_Country()
  - testGetData_Country_BadRequest_Country()
  - testGetData_Country_BadRequest_NoParameters()
- **GET /api/get/continent**
  - testGetData_Continent()
  - testGetData_Continent_BadRequest_Country()
  - testGetData_Continent_BadRequest_NoParameters()
- **GET /api/get/day**
  - testGetData_Day()
  - testGetData_Day_BadRequest_Country()
  - testGetData_Day_BadRequest_NoParameters()
- **GET /api/all_data**
  - testGetAllData()

Selenium Testing

I made two simple scenarios, when we have a country with real data and, on the contrary, when we have a country with no data, with all data defaulted to zero.

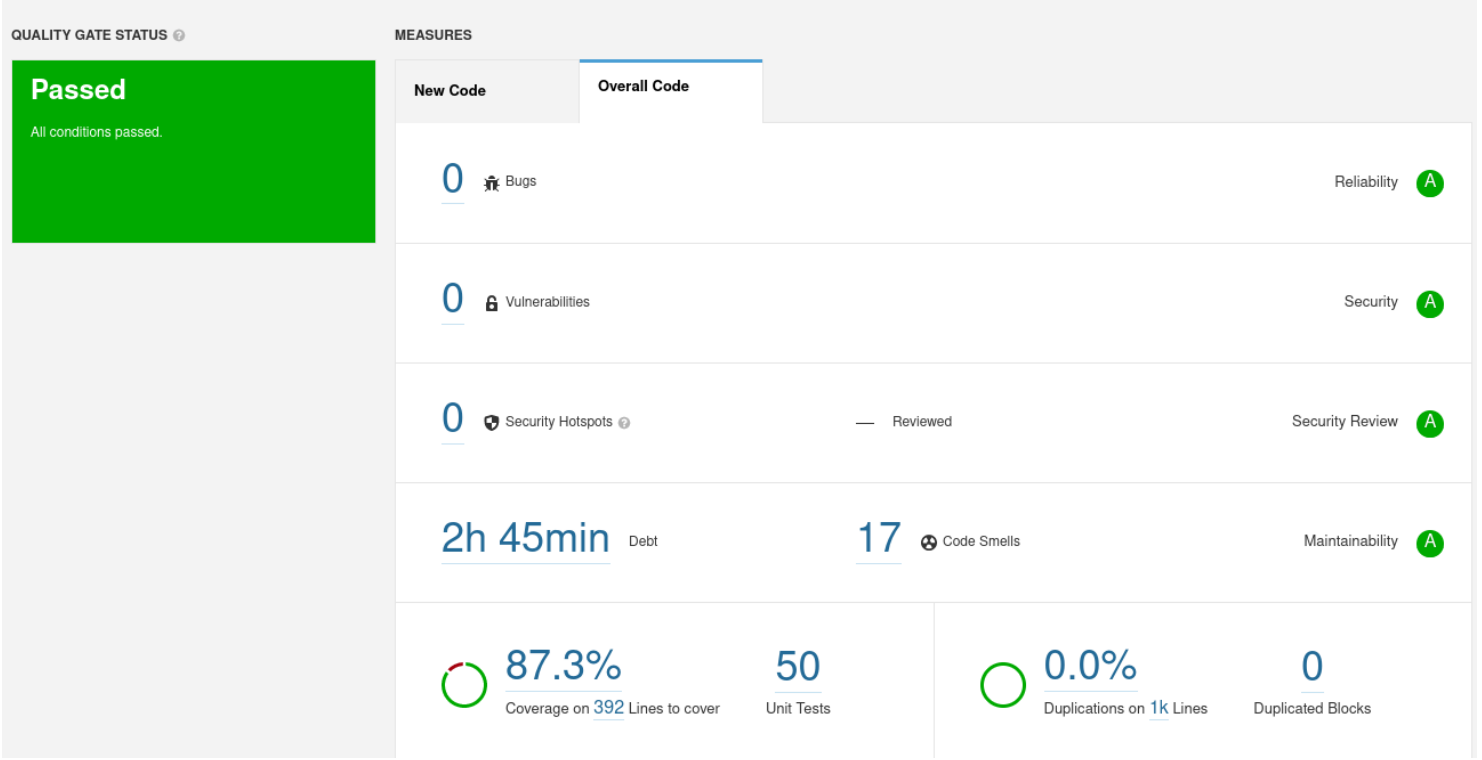
```
Feature: HW1 Web Selenium Test

  Scenario: Successful
     When I navigate to "http://localhost:8080/"
     And I choose "Portugal" as the Location
     And I click on Submit
     Then I should see the country "Portugal" as the title of the table
     And I should look at Critical Cases in the first line and see its "not zero"

  Scenario: No Data
     When I navigate to "http://localhost:8080/"
     And I choose "DRC" as the Location
     And I click on Submit
     Then I should see the country "DRC" as the title of the table
     But I should look at Critical Cases in the first line and see its "zero"
```

SonarQube

SonarQube was used to refactor the code utilized, and also to check which methods and parameters were not being covered.



17 code smells:

- 1 blocker, which we can discard as to this project:

```java
package TQS_HW1.HW1.Web;

import org.junit.platform.suite.api.ConfigurationParameter;
import org.junit.platform.suite.api.IncludeEngines;
import org.junit.platform.suite.api.SelectClasspathResource;
import org.junit.platform.suite.api.Suite;

import static io.cucumber.junit.platform.engine.Constants.PLUGIN_PROPERTY_NAME;
import static io.cucumber.junit.platform.engine.Constants.GLUE_PROPERTY_NAME;

@Suite
@IncludeEngines("cucumber")
@SelectClasspathResource("TQS_HW1/HW1/Web")
@ConfigurationParameter(key = PLUGIN_PROPERTY_NAME, value = "pretty")
@ConfigurationParameter(key = GLUE_PROPERTY_NAME, value = "TQS_HW1/HW1/Web")
public class WebTest {
```

Add some tests to this class. Why is this an issue? 10 days ago L16
Code Smell Blocker Open Not assigned 5min effort Comment confusing, junit, tests, unused

```java
}
```

- 16 minor, which we should not consider as to is naming conventions:

As for the coverage, there is a 100% in functional classes, but a 87.3% overall, which can be explained by having a very low coverage in frontend classes, as it was used Thymeleaf.

References

Git Repository with the project:
https://github.com/FlipGoncalves/TQS_98083/tree/main/HW1

Git Repository folder for the code:
https://github.com/FlipGoncalves/TQS_98083/tree/main/HW1/HW1

Video:
https://github.com/FlipGoncalves/TQS_98083/blob/main/HW1/demonstration.mp4