HW1: Technical Report

Teste e Qualidade de Software

Gonçalo Machado nMec 98359

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 product named CovidTracker is a project which has the main purpose of providing information about the coronavirus, also known as covid-19, through the use of a website and an API.

Current limitations

Currently, this project does not have much choice when it comes to getting data, since it's only possible to get data of a country or a continent, as well as world data, and only for the last 3 days.

There is also a limitation on the website, where if we search for a country that does not exist or is not valid, an exception is thrown and an error page appears. The correct exception is thrown, but there is no specific page to go when this happens and the user is simply shown the default error page. The user can simply go back to the main page and the website is still working and can be used normally.

Product specification

Functional scope and supported interactions

CovidTracker is a very simple product, designed to be used by people interested in seeing data related to covid of a country, continent or the world. It also has a cache to reduce the number of calls to the external API, and the user can get have access to the cache statistics, specifically hits/misses and get/save/delete requests

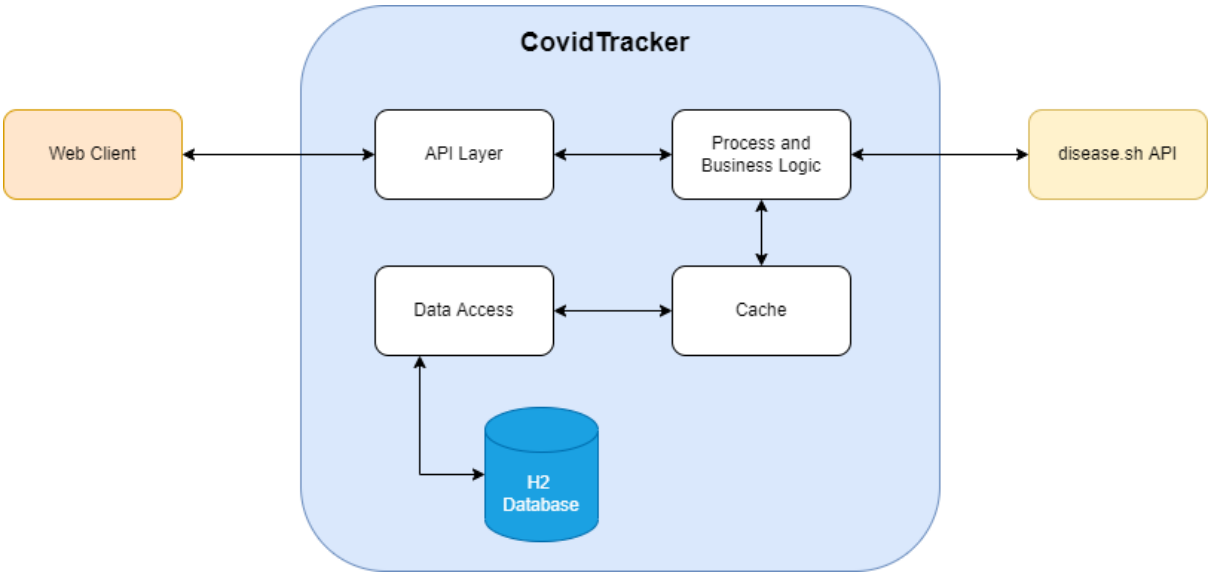
The user can get this data in two different ways:

- The website: The website has a very minimalist design, and can be used to get data from a specific country or the world from the last 3 days. The website also has a page where it shows the cache statistics
- The API: The REST API can be accessed by the user to get all the data that is available on the website, but in a JSON format, plus get covid data of a continent, also from the last 3 days.

System architecture

The architecture has three main parts:

- The Web Client, done using Thymeleaf
- The CovidTracker application, done using Spring Boot
- External API, in this case disease.sh

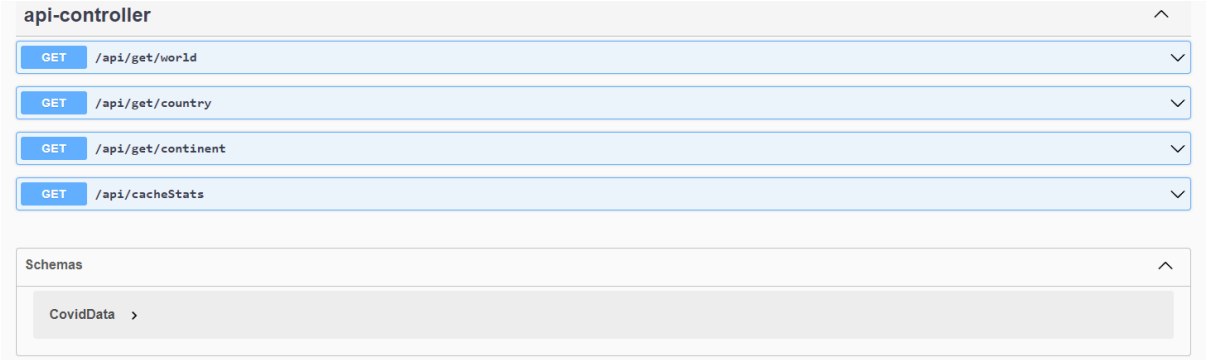


The flow of the whole system is centered around the CovidTracker application. The flow can start in two ways, by accessing the website or by making a request to the Rest API. Then the service checks if the wanted data exists in cache, and if it exists it returns the wanted data. If it does not exists, it calls the resolver, which creates the url that is going to be used to make a request to the external API, disease.sh, and sends the url to the HTTPAPI that makes the call and returns the response back to the resolver. The resolver then converts the data from JSON into a CovidData object and returns this object to the service, which after saving the object in cache, returns it to the respective Controller.

API for developers

A developer can get covid data from a country, a continent or the world, as well as getting the statistics of the cache.

To document the API endpoints and schemas, the documentation tool Swagger was used, and is available when the project is running at the endpoint /swagger-ui/index.html



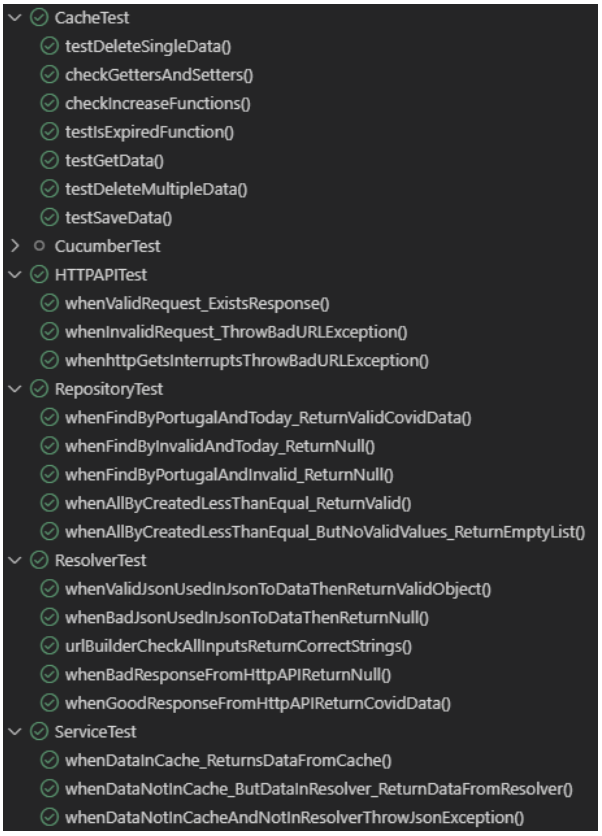
Quality assurance

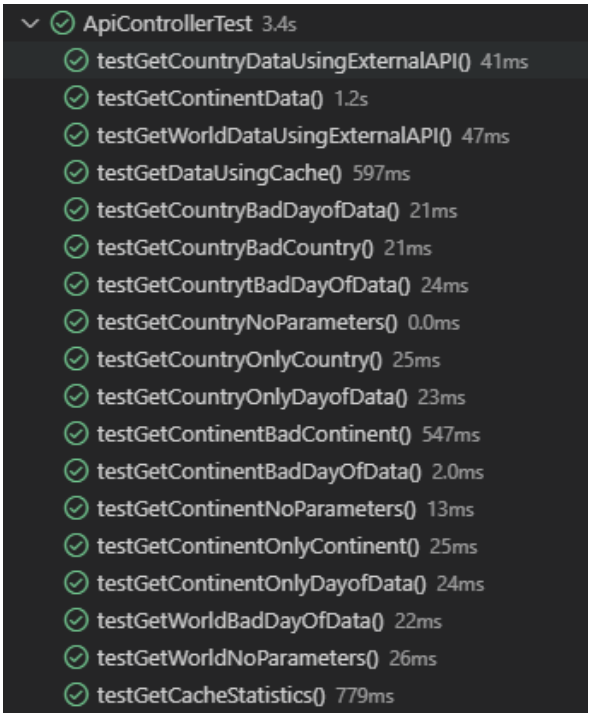
Overall strategy for testing

Although not always possible due to the lack of knowledge, time and experience, a test driven development was used and, when analyzing the entire development, is recommended as it can make more clear all the different scenarios, features and exceptions that a project or a part of it needs. For the functional testing, a BDD approach was used by using Cucumber, since it was requested and also because it focus on how the system should behave from the users perspective, and making the tests easier to understand since the scenarios are written in normal or non-technical language.

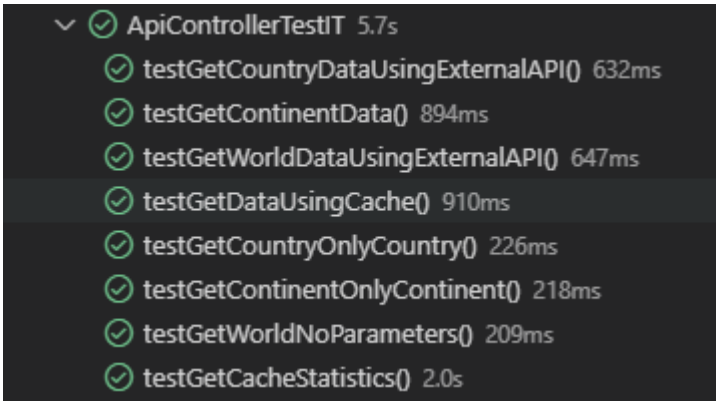
Unit and integration testing

Unit testing was done on every class that was used in the business logic and data storage, specifically the *Cache*, the *CovidDataService*, the *CovidDataResolver*, the *HTTPAPI* and the *CovidDataRepository* and *APIController*. The tests were done thinking of the normal use of the classes as well as possible bad inputs or parameters that should raise exceptions.





As for integration testing, it was done on the *APIController* in order to test that the product worked as intended when all the different components worked together.



Functional testing

Functional testing was done using Cucumber and Selenium and with a Behavior Driven Development approach. There were two scenarios tested: one where the user wanted to get data from a country, Portugal, from two days ago, and the other where the user wanted to check the cache statistics.

As mentioned in the Current Limitations section, there are no error pages/messages designed for each error/exception that is raised, so there are no tests covering when these errors should appear.

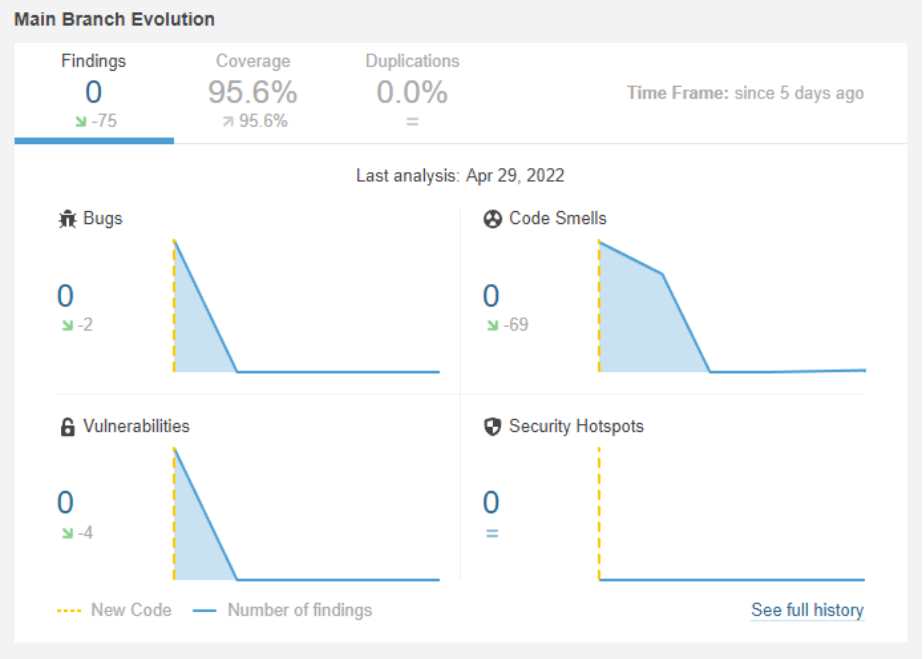
Also due to the lack of time, knowledge and proper documentation, these tests are not ran on the CI pipeline, so in order to run these tests the application must be running beforehand.

```
Feature: CovidTracker web testing

    Scenario: Getting data from Portugal from Two Days Ago
        When I navigate to "http://localhost:8080/"
        And I select the searchbar and type "Portugal"
        And I click on the radio button for "Two Days Ago"
        And I click Submit
        Then the title of the page should be "CountryStats"
        And there should be a card with the title "Country" and value "Portugal"
        And there should be a card with the title "DayOfData" and value "Two Days Ago"

    Scenario: Getting cache statistics after first scenario
        When I navigate to "http://localhost:8080/"
        And I click Cache
        Then the title of the page should be "Cache Page"
        And there should be a card with the title "Hits" and value "1"
        And there should be a card with the title "Misses" and value "1"
        And there should be a card with the title "Get Requests" and value "1"
        And there should be a card with the title "Save Requests" and value "1"
        And there should be a card with the title "Delete Requests" and value "0"
```

Scenarios of the functional tests

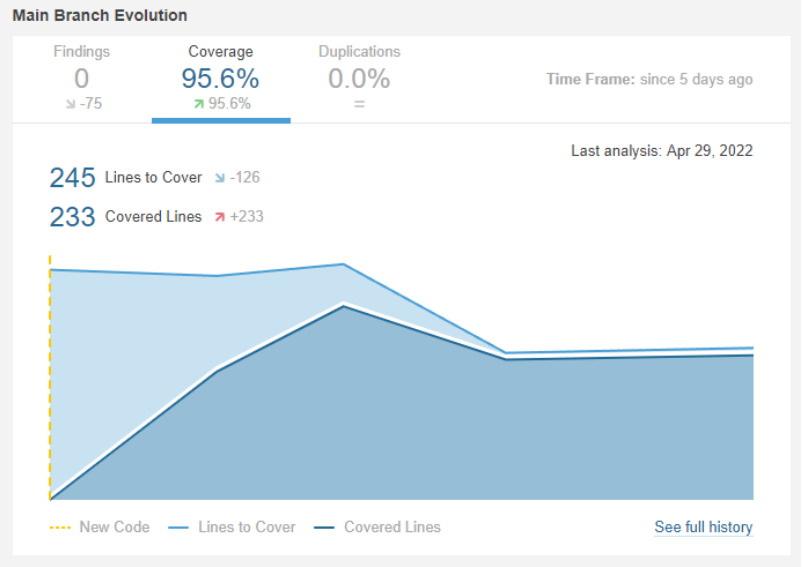
Code quality analysis

As for the analysis of code quality, the SonarCloud service was used. This was possible since the repository where the code for this project is located is public. It is currently available at https://sonarcloud.io/summary/overall?id=goncalo-machado_TQS .



Project Findings Evolution

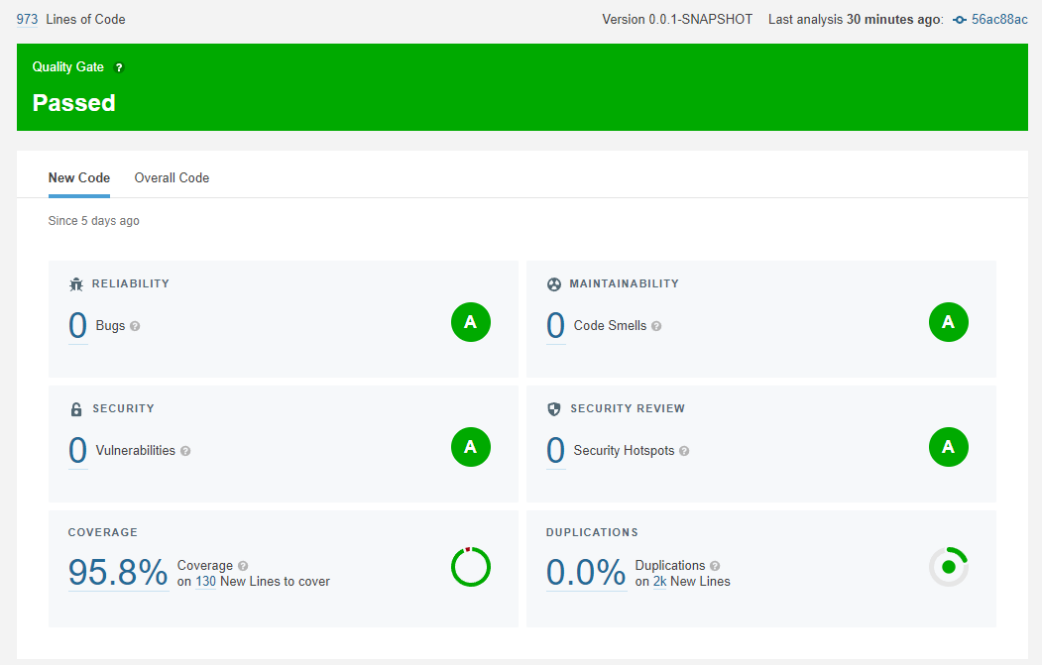
SonarCloud was implemented in the middle of the development, and as such, we can see in the picture above that it had a lot of code smells and vulnerabilities and bugs. The bugs were exceptions that were not being thrown when they should have been and the vulnerabilities were in some logs where a variable created from user input was used, thus leading to potential attacks, while the code smells were mostly conventions that were not being followed. Although it took more time than expected, it was able to reduce all of these to 0, and thus showing the importance and practicality of having a code quality analysis.



Project Code Coverage Evolution

As for the code coverage, as we can see in the image above it started with no coverage, when there were no tests, and gradually increased throughout the development. The decrease in lines to cover and covered lines happened because some classes were excluded from the code coverage calculations. These classes are the classes that do not have much or nothing at all to test, like the *CovidData* class which is just a normal object with getters and setters that do not make sense to test, or *CovidTrackerApplication* that is the class that simply launches the application.

This image below is the final analysis, which was done at the end of the development. As we can see, it has no bugs, vulnerabilities, code smells, security hotspots or duplication, and has a 95.8% code coverage, which shows that the tests check almost everything that is needed.



Continuous integration pipeline

In order to automatically test our application and send the results to SonarCloud every time a pull request or push were done, a pipeline was built. This pipeline is a template provided by SonarCloud, where just a slight alteration was made, which is related to the impossibility of running the functional test without running the application in parallel.

```yaml
name: Build
on:
  push:
    branches:
      - master
  pull_request:
    types: [opened, synchronize, reopened]
defaults:
    run:
      working-directory: HW1/covidTracker
jobs:
  build:
    name: Build
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
        with:
          fetch-depth: 0  # Shallow clones should be disabled for a better relevancy of analysis
      - name: Set up JDK 11
        uses: actions/setup-java@v1
        with:
          java-version: 11
      - name: Cache SonarCloud packages
        uses: actions/cache@v1
        with:
          path: ~/.sonar/cache
          key: ${{ runner.os }}-sonar
          restore-keys: ${{ runner.os }}-sonar
      - name: Cache Maven packages
        uses: actions/cache@v1
        with:
          path: ~/.m2
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
          restore-keys: ${{ runner.os }}-m2
      - name: Build and analyze
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}  # Needed to get PR information, if any
          SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        run: mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=goncalo-machado_TQS -Dtest=!CucumberTest
```

Pipeline to run tests and send results to SonarCloud

References and resources

Project resources

Github Repository : https://github.com/goncalo-machado/TQS/tree/master/HW1
Video Demo : Available in the README of the repository
QA Dashboard : https://sonarcloud.io/project/overview?id=goncalo-machado_TQS
CI pipeline :
https://github.com/goncalo-machado/TQS/blob/master/.github/workflows/build.yml

Reference materials

External API : https://disease.sh/docs/#/
W3Schools Website Templates : https://www.w3schools.com/w3css/w3css_templates.asp