HW1: Mid-term assignment report

João Pedro Saraiva Borges [98155] 2022-05-02

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. This work is supposed to gather data from an outside API and process it so that we can display the data in a simple site, and also have a REST API which also provides data.

1.2 Current limitations

The API and webpage is extremely limited, we can easily implement new endpoints and have much more information, but since the main focus of the project isn't that, I tried to maintain it as simple as possible. Sadly, big part of the testing doesn't work because of errors that are not covered online, in which every of them was made a huge effort to try and find what the problem was, even with the help of colleagues and professors, some problems remained, specifically in the integration testing and in the CI and sonarqube usage; this took a lot of time and effort which made the testing section much smaller than what I desired it to be.

2 Product specification

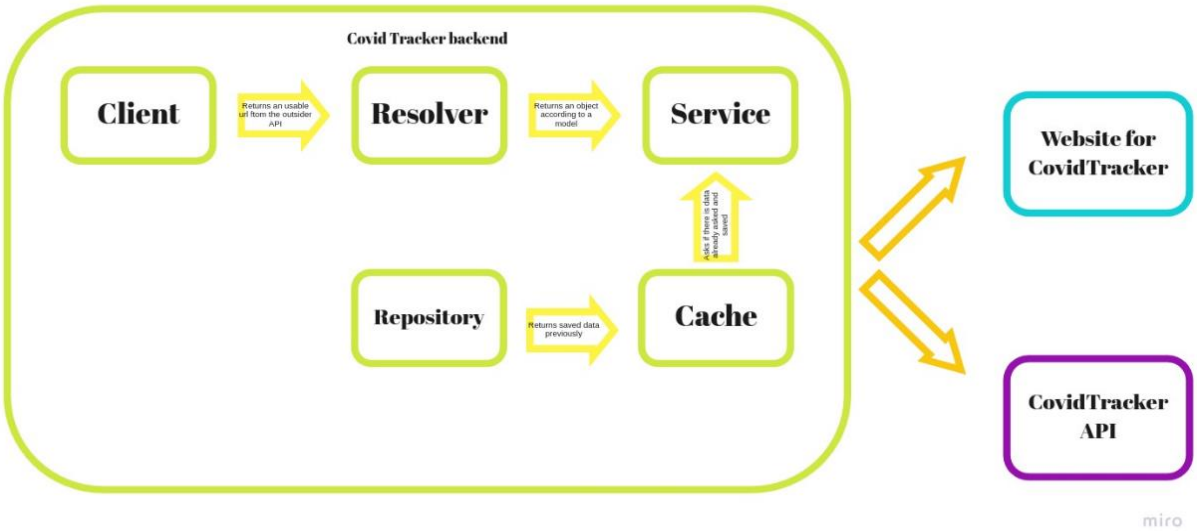
2.1 Functional scope and supported interactions

This application can show the amount of new covid cases in the last day of a certain country. It can also show the amount of new covid cases in a certain day. The main purpose of this application is to be always updated on the amount of new cases in the latest day.

The user can check this data by going to the website, or asking to the API, in the website you can check the amount of new cases for a specific country, while in the API you can also check new cases for a specific country as well the amount of new cases in a specific day. We can also check the amount of requests and hit/misses made by the cache.

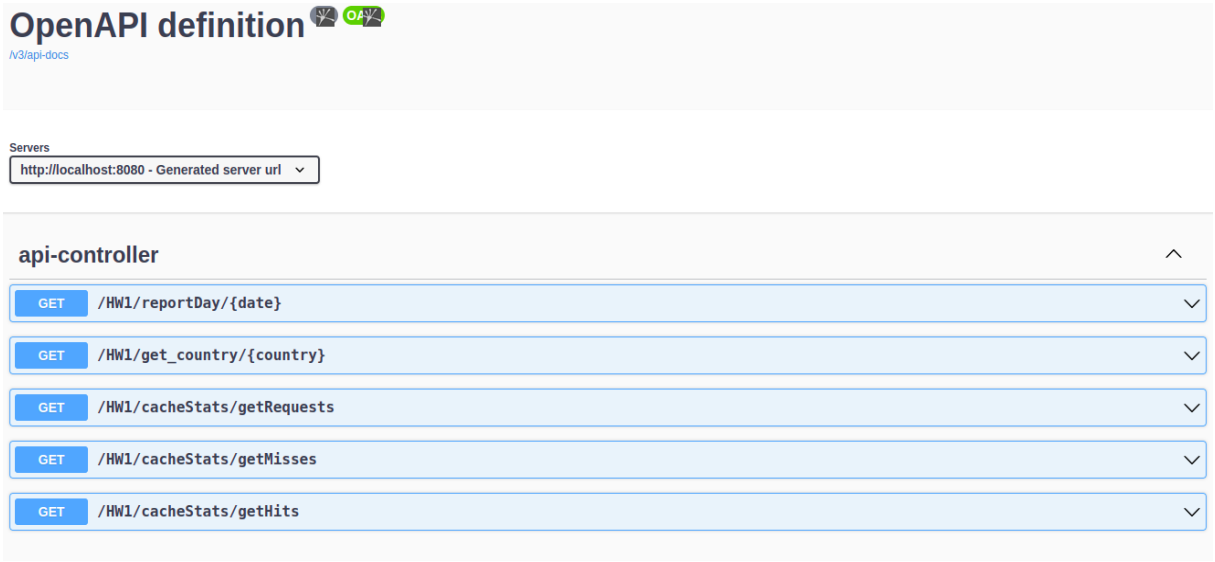
2.2 System architecture

As explained in the mid-term assignment paper, the application will have an outsider API which will send the information we need by the class Client.java, which will after be processed by the resolver.java to turn the json given by the API into a object, a CovObject, class saved in the models folder; before all of this the service class will ask the cache if there is data previously saved, if not, it will call the resolver to provide the data needed. In the end, the data will be shown in an html page, or sent by the REST API. The cache has the repository connected to it. Most of the backend is made with springboot, while the website is made with html.



2.3 API for developers

The developer can gather data about the amount of new cases in a country, or the amount of new cases in a specific past day. It is an extremely simple API just to show it's existence, which would be easily improved. It is also possible to return the cache statistics. By



accessing the link http://localhost:8080/swagger-ui/index.html we can check the endpoints of the application:

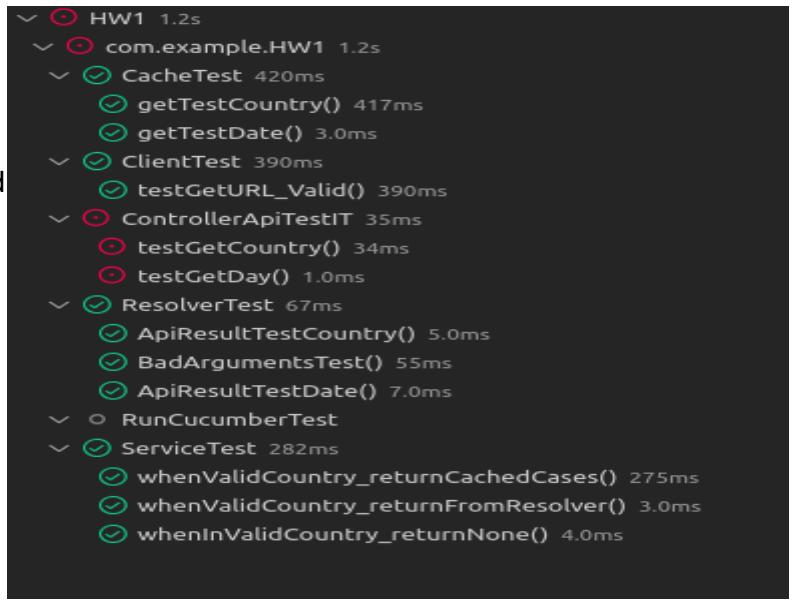
3 Quality assurance

3.1 Overall strategy for testing

Because of a major lack of time, the testing misses some crucial points in the development of the application, but it still covers most part of it. Junit, Mockito, Selenium along with Cucumber, and MockMVC from spring boot was used. TDD was used.

3.2 Unit and integration testing

Integration testing was used mainly in the controller class. The rest of tests focus on unit testing.



The integration testing could not be solved because of a null pointer, which has been always happening since the beginning of the project; even though there was no additional dependencies in conflict, there was always this problem, usually changing the annotation before the class would work, but in this case it didn't.

3.3 Functional testing

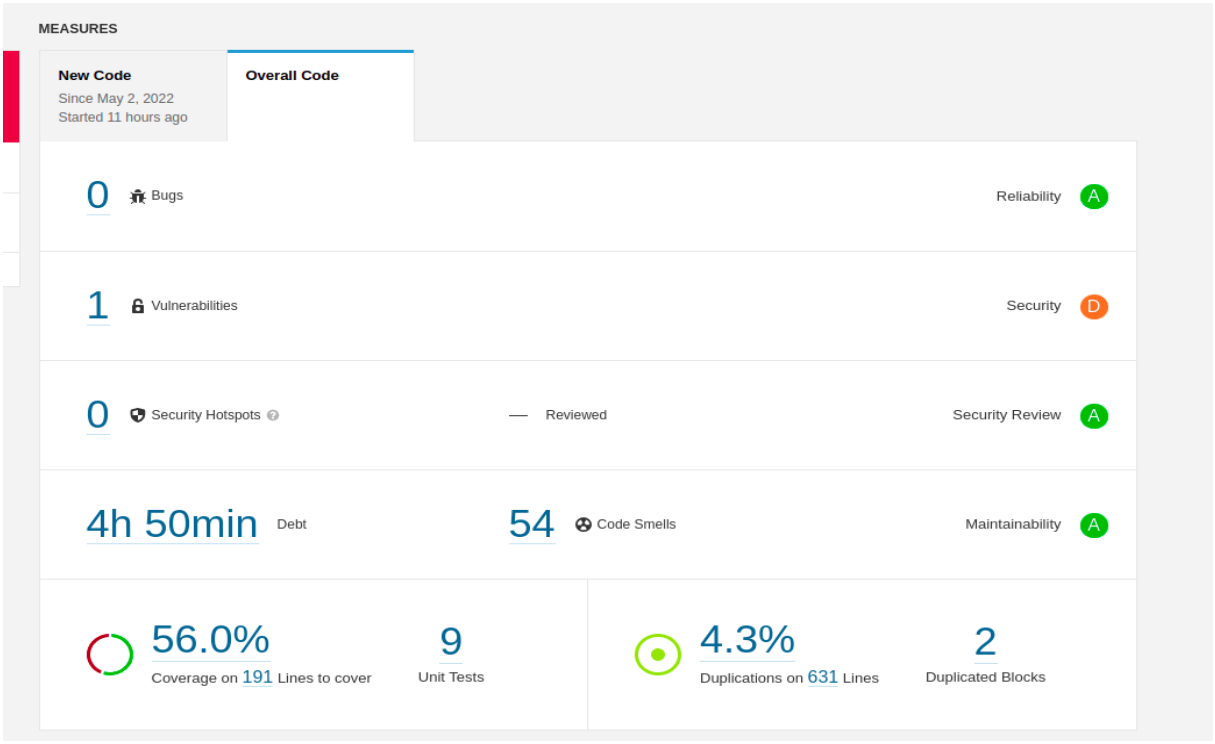
I considered cucumber and selenium testing, which there was only one scenario tested. The scenario would test the action of searching for Portugal new cases and confirming it was Portugal. ATTENTION: As mentioned before, my personal computer must have some errors on the installation of the chrome driver, as a result, the log of the steps testing is always printing random characters, which make the test unusable, still, I'm quite sure the scenario is correctly done.

```
Feature: HW1 testing

        Scenario: Requesting data of Portugal
                When I navigate to "http://localhost:8080/"
                And I select the searchbar and type "Portugal"
                And I click on the Submit button
                Then the title of the page should be "Country"
                And there should be a card with the title "Country" and value "Portugal"
```

3.4 Code quality analysis

Sonarqube was used for static code analysis. The project still had some points which had to be taken care off, specially in the code coverage, but the major code smells were clean and most of the code that is not covered is not extremely necessary to be covered. Take note that some of the integration testing and the selenium tests are not present in this analysis because of unknown problems.



3.5 Continuous integration pipeline [optional]

The sonarcloud was used in the github workflow, but also had some problems which made it unusable. The implementation is correct and the tutorial was followed with precision, sadly, a token error appears, which should be deprecated. The configuration file:

```yaml
name: Build
on:
  push:
    branches:
      - master
      - main
  pull_request:
    types: [opened, synchronize, reopened]
defaults:
  run:
    working-directory: HW/HW1
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=JoaoBorgesUA_tqs_98155
```

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/JoaoBorgesUA/tqs_98155/HW
Video demo	https://github.com/JoaoBorgesUA/tqs_98155/HW

Reference materials

HTML templates: https://www.w3schools.com/w3css/w3css_templates.asp