universidade de aveiro
deti departamento de eletrónica, telecomunicações e informática

HW1: Covid specifications

João Farias 98679, 2022-04-29

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.

Covinfo-19 is a software developed, in Spring Boot, supplied with automated tests, capable of providing worldwide or by countries covid informations.

1.2 Current limitations

When a country is chosen, the last 6 months covid information is calculated, but not in the correct way. The software picks six days, one for each past month and calculates the information only on that day. This can be fixed, but we had to make a lot of **API** requests and the performance would be even worse.

Another limitation is about recovered people. **COVID-19** Statistics API (the external API) doesn't provide any information about this, so in the web page there is no information either.

Due to lack of time, I prefered to use more time doing the tests instead of developing some additional information, for example, improving the past **covid-19** details.

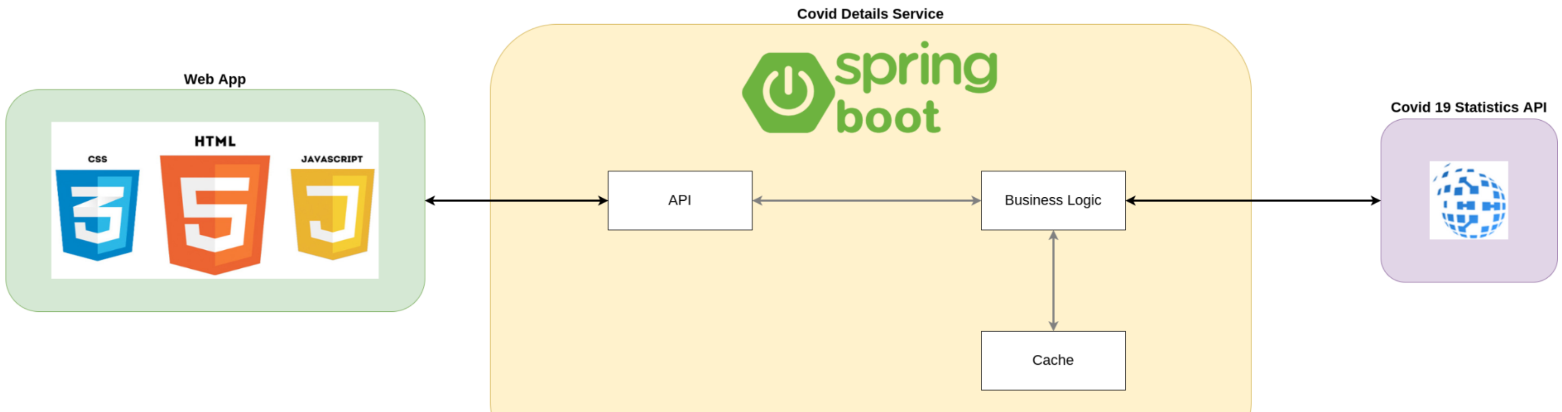
2 Product specification

2.1 Functional scope and supported interactions

In the web page it is possible to access the real time covid statistics, by regions/worldwide and also to get cache information in order to understand its behavior.

This software can be used by everybody, specially by the journalists since they need to provide this information almost everyday. If they need to get the total cases of Portugal, for example, the journalist opens the browser, in the select element, choose **Portugal** and then the total number of cases and deaths will be displayed.

2.2 System architecture



The system architecture is composed of three main components: **web app** (built in **HTML**, **CSS**, **JS**), **spring services** and an **external API**.

The **services** domain is where the main application is present: the client requests, the cache data and the requests to external API are present there. To better understand system architecture, imagine a person that wants to know the world's covid information. The web app is going to make a request to the **REST API** asking for world information and this one is going to ask an external **API** for this information, if there is no data in the cache. Otherwise, the data is returned without asking the external **API.**

Regarding the cache, whenever the **service** receives a request, it is going to look in the cache for that information and will or will not make an external request depending if there is or there isn't information in the cache. It is also possible to know how many hits, requests and cache misses there were.

2.3 API for developers

The API documentation is available online here. All endpoints are explained and have an example of the return statement.

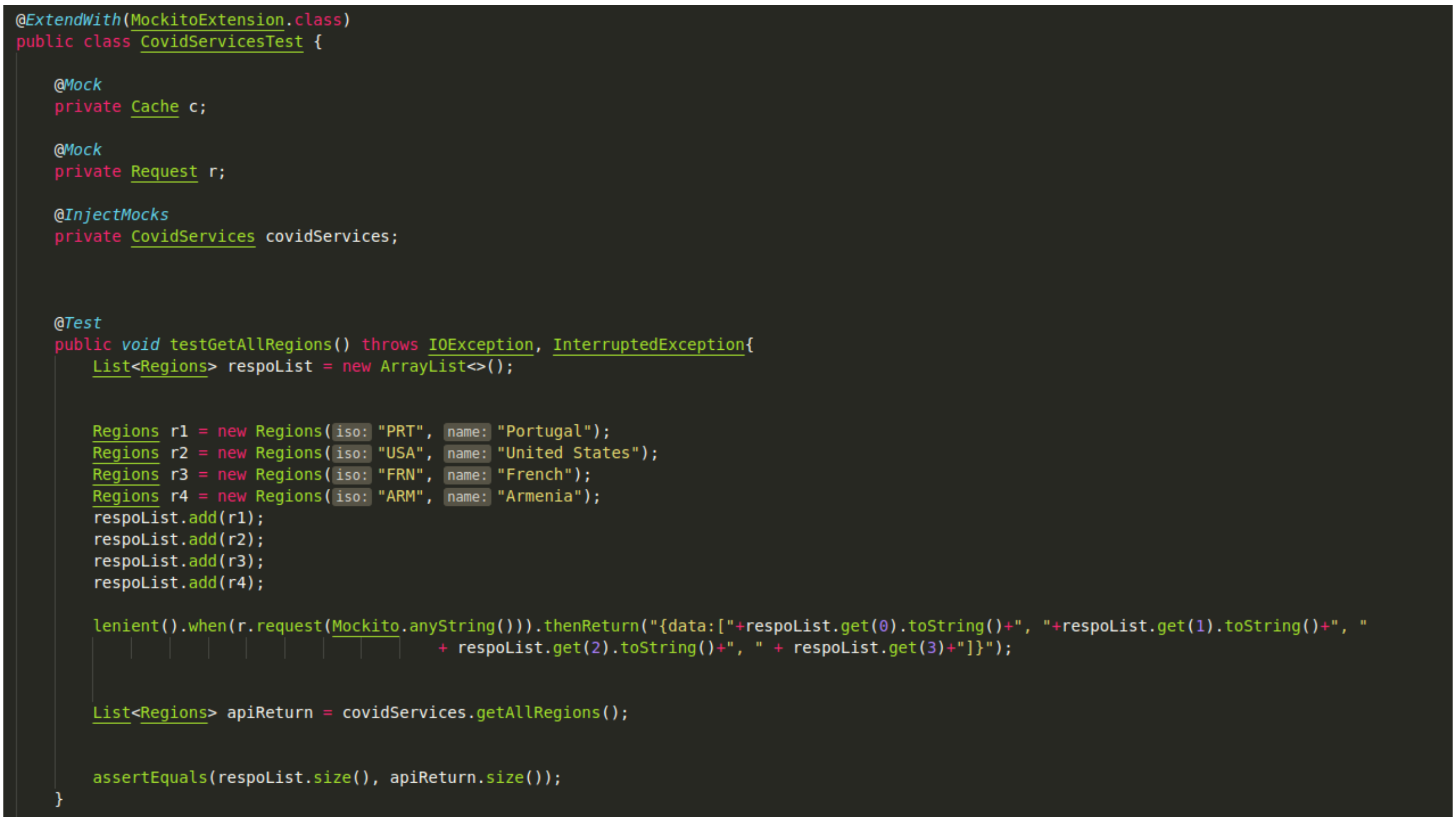
3 Quality assurance

3.1 Overall strategy for testing

To test this software I tried to use a **Test Driven Development** but when the time was ending I cannot follow this strategy and some tests were completed after the feature was developed. However I admit that **TTD** strategy is excellent to understand all scenarios that a feature must have to face.

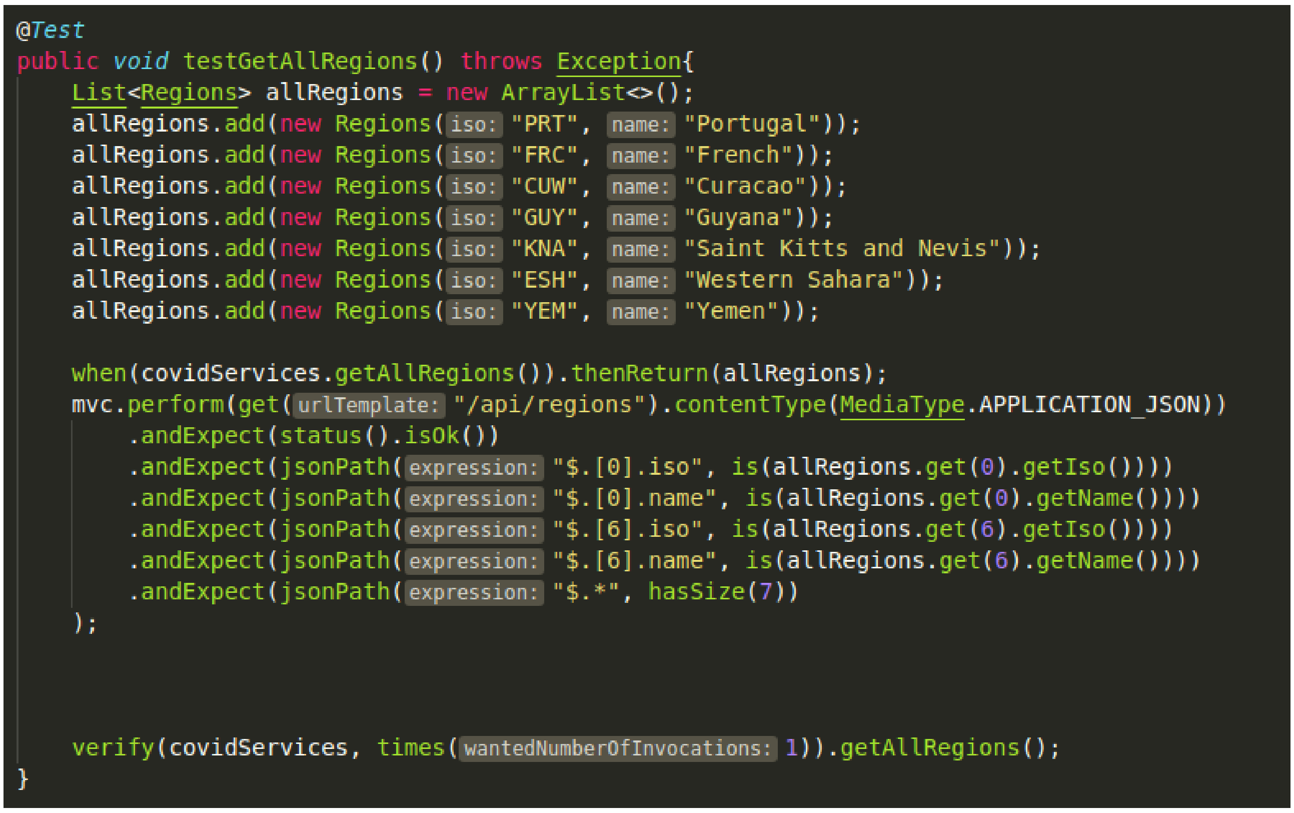
3.2 Unit and integration testing

The unit tests were developed for almost all classes, some of these tests were developed with **Mock** objects and I tried to test the most failure cases I could. The other case where I developed unit tests was in the **Cache**. I tested the **add**/**delete**/**get** methods and also the **get** method after the cache **living time** has passed.



```java
@ExtendWith(MockitoExtension.class)
public class CovidServicesTest {

    @Mock
    private Cache c;

    @Mock
    private Request r;

    @InjectMocks
    private CovidServices covidServices;



    @Test
    public void testGetAllRegions() throws IOException, InterruptedException{
        List<Regions> respoList = new ArrayList<>();


        Regions r1 = new Regions( iso: "PRT",  name: "Portugal");
        Regions r2 = new Regions( iso: "USA",  name: "United States");
        Regions r3 = new Regions( iso: "FRN",  name: "French");
        Regions r4 = new Regions( iso: "ARM",  name: "Armenia");
        respoList.add(r1);
        respoList.add(r2);
        respoList.add(r3);
        respoList.add(r4);

        lenient().when(r.request(Mockito.anyString())).thenReturn("{data:["+respoList.get(0).toString()+", "+respoList.get(1).toString()+", "
                                    + respoList.get(2).toString()+", " + respoList.get(3)+"]}");


        List<Regions> apiReturn = covidServices.getAllRegions();


        assertEquals(respoList.size(), apiReturn.size());
    }
```

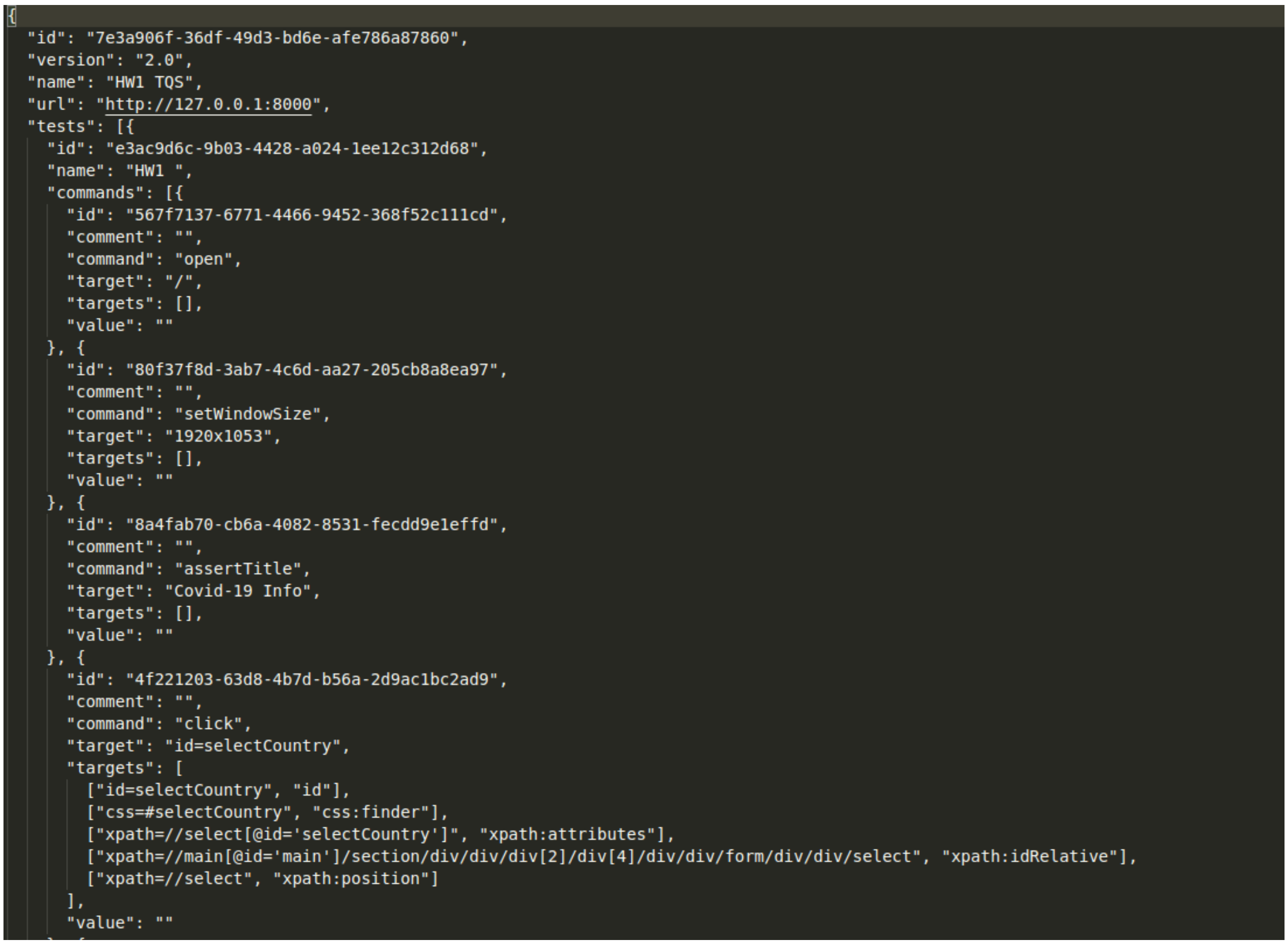
Regarding integration tests, six tests were developed, one for each **controller** feature (endpoint) and **SpringBoot MockMvc** was used. As I know the response from the endpoints, I prepared a return sentence, then I made a request to that *endpoint* and verified if it was correct or not.



```java
@Test
public void testGetAllRegions() throws Exception{
    List<Regions> allRegions = new ArrayList<>();
    allRegions.add(new Regions( iso: "PRT",  name: "Portugal"));
    allRegions.add(new Regions( iso: "FRC",  name: "French"));
    allRegions.add(new Regions( iso: "CUW",  name: "Curacao"));
    allRegions.add(new Regions( iso: "GUY",  name: "Guyana"));
    allRegions.add(new Regions( iso: "KNA",  name: "Saint Kitts and Nevis"));
    allRegions.add(new Regions( iso: "ESH",  name: "Western Sahara"));
    allRegions.add(new Regions( iso: "YEM",  name: "Yemen"));

    when(covidServices.getAllRegions()).thenReturn(allRegions);
    mvc.perform(get( urlTemplate: "/api/regions").contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath( expression: "$.[0].iso", is(allRegions.get(0).getIso())))
        .andExpect(jsonPath( expression: "$.[0].name", is(allRegions.get(0).getName())))
        .andExpect(jsonPath( expression: "$.[6].iso", is(allRegions.get(6).getIso())))
        .andExpect(jsonPath( expression: "$.[6].name", is(allRegions.get(6).getName())))
        .andExpect(jsonPath( expression: "$.*", hasSize(7))
    );



    verify(covidServices, times( wantedNumberOfInvocations: 1)).getAllRegions();
}
```

3.3 Functional testing

The functional tests were developed with **Selenium WebDriver** and the file **HW1.side** contains all steps used to conclude this test.

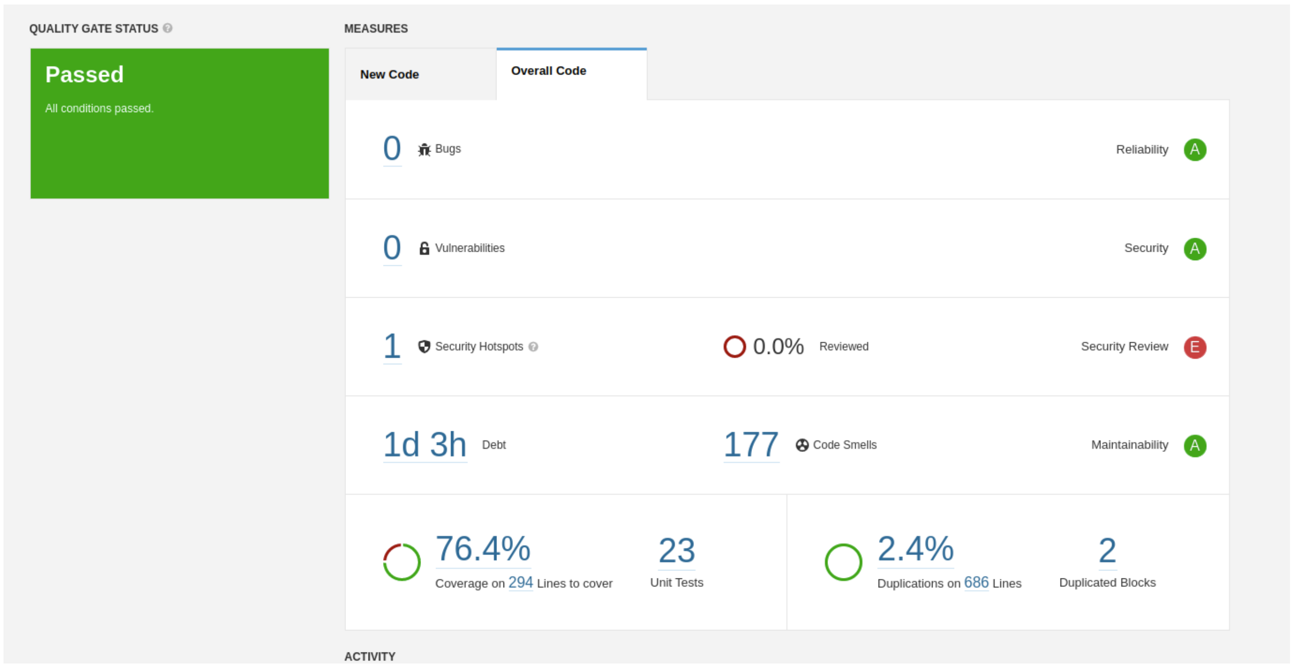


```
{
  "id": "7e3a906f-36df-49d3-bd6e-afe786a87860",
  "version": "2.0",
  "name": "HW1 TQS",
  "url": "http://127.0.0.1:8000",
  "tests": [{
    "id": "e3ac9d6c-9b03-4428-a024-1ee12c312d68",
    "name": "HW1 ",
    "commands": [{
      "id": "567f7137-6771-4466-9452-368f52c111cd",
      "comment": "",
      "command": "open",
      "target": "/",
      "targets": [],
      "value": ""
    }, {
      "id": "80f37f8d-3ab7-4c6d-aa27-205cb8a8ea97",
      "comment": "",
      "command": "setWindowSize",
      "target": "1920x1053",
      "targets": [],
      "value": ""
    }, {
      "id": "8a4fab70-cb6a-4082-8531-fecdd9e1effd",
      "comment": "",
      "command": "assertTitle",
      "target": "Covid-19 Info",
      "targets": [],
      "value": ""
    }, {
      "id": "4f221203-63d8-4b7d-b56a-2d9ac1bc2ad9",
      "comment": "",
      "command": "click",
      "target": "id=selectCountry",
      "targets": [
        ["id=selectCountry", "id"],
        ["css=#selectCountry", "css:finder"],
        ["xpath=//select[@id='selectCountry']", "xpath:attributes"],
        ["xpath=//main[@id='main']/section/div/div/div[2]/div[4]/div/div/form/div/div/select", "xpath:idRelative"],
        ["xpath=//select", "xpath:position"]
      ],
      "value": ""
```

After this test I used **Cucumber** with web automation to conclude the functional tests. I had to add a lot of sleeps, so that the browser could find the elements and complete the tests in the correct way.

3.4 Code quality analysis

To code analysis it was used **SonarQube** and the analysis are below:



QUALITY GATE STATUS

Passed

All conditions passed.

MEASURES

New Code | Overall Code

0 Bugs — Reliability A

0 Vulnerabilities — Security A

1 Security Hotspots — 0.0% Reviewed — Security Review E

1d 3h Debt — 177 Code Smells — Maintainability A

76.4% Coverage on 294 Lines to cover — 23 Unit Tests — 2.4% Duplications on 686 Lines — 2 Duplicated Blocks

ACTIVITY

Most code smells are about the package's name, about sleeps and variables names that have _ or that have the same name as the class. The code coverage could be greater if I tested the **getters** and **setters**, but in most cases it is unnecessary.

Some conclusions about the analysis are that it is important to write good and clear code from the beginning because if we want to change it after, we wouldn't remember most of the important aspects.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/bernas04/TQS_98679/tree/main/HW1
Video demo	https://github.com/bernas04/TQS_98679/blob/main/HW1/schemas/hw1_demo.gif
API documentation	https://documenter.getpostman.com/view/18307740/UyrGCExg

Reference materials

- https://rapidapi.com/collection/coronavirus-covid-19
- https://www.vogella.com/tutorials/Mockito/article.html
- https://cucumber.io/docs/cucumber/cucumber-expressions/