HW1: Mid-term assignment report

Artur Correia Romão [98470], 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.
My application, Covid-19 Data Metrics, is a minimalist web app integrated with an external API that provides updated Covid-19 incidence data.

1.2 Current limitations

Unfortunately, because of the lack of time (family matters in Luxembourg and Academic Week), I couldn't implement everything that I initially pictured in my mind, but I still tried hard.
The current limitations of my web app are:

- Unimplemented last 6 months feature (with nice graphics to show information) since the API that I used has an endpoint with these data.
- Unimplemented Cache tab in frontend to show Cache statistics (even though the Cache is developed and working just fine).

2 Product specification

2.1 Functional scope and supported interactions

The focus of this web application is to present the users with the updated worldwide Covid-19 statistics as well as specific countries metrics.

The actors are every user who is interested in checking the Covid-19 statistics, such as cases, deaths, tests and recoveries.
The main scenarios of my application are:

- Checking the world Covid-19 metrics, by accessing the endpoint /world.
- Checking the specific countries Covid-19 metrics, by accessing the endpoint /country/{country}/{isoCode}, with the path variables being the country's name and iso code, respectively.

2.2 System architecture

This application is divided into two modules, one for the frontend and the other for the backend. The backend solution is based on Spring Boot and the presentation layer was developed in thymeleaf (a templating system that integrates with Spring Boot). The backend technology was mandatory, so there's nothing to say about it. As for the frontend, I decided to choose thymeleaf, since my group used that technology in last term's IES project, so I was already familiarized and really enjoyed working with it for its simplicity and awesome integration with Spring Boot.
I also implemented a simple Cache as suggested by the teacher (to reduce the external API calls) using HashMaps to store the data and its times to live.
In order to retrieve the Covid-19 data metrics, I used the API VACCOVID (https://rapidapi.com/vaccovidlive-vaccovidlive-default/api/vaccovid-coronavirus-vaccine-and-treatment-tracker/), once its endpoints seemed more adequate to me and my idea for the app. I also used Country Flags API to get all the country images. The two APIs worked well together since both shared the use of each country iso code.

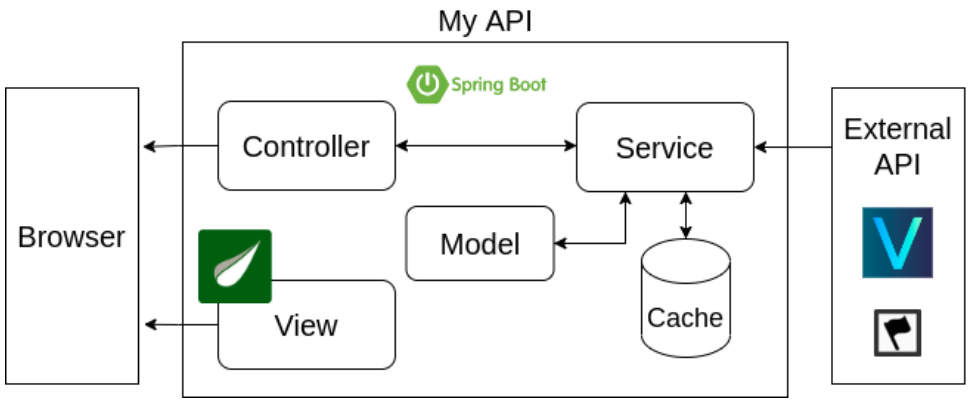


Fig.1 - Architecture diagram (the leaf stands for thymeleaf, the V stands for VACCOVID and the flag stands for Country Flags API)

2.3 API for developers

My application has 4 endpoints:

- GET / - the index, which displays the world statistics since the beginning of the pandemic.
- GET /world - which displays the world statistics since the beginning of the pandemic + today data (new cases, deaths and recoveries) as well as other relevant data such as active cases as well as cases and deaths per 1M population.
- GET /country - which displays a static html page ready to receive data, with a select that contains every country for which the API contains data.

- GET /country/{country}/{isoCode} - displays a page, similar to the previous one, that contains lots of Covid-19 metrics (cases, deaths, recoveries, tests...), as well as the country's information (name, iso code, continent, population, etc).

3 Quality assurance

3.1 Overall strategy for testing

I chose a BDD approach using Cucumber and mixed it with Selenium automation test, being inspired by both lab4 and lab5.

```
public class SeleniumSteps {
  private WebDriver driver;

  @When("I access {string}")
  public void accessCovidMetrics(String url) {
    driver = new ChromeDriver();
    driver.get(url);
    driver.manage().window().setSize(new Dimension( width: 1920,  height: 1080));
    assertEquals( expected: "Covid-19 World statistics since the beginning of the pandemic", driver.findElement(By.cssSelector( cssSelector: "h1")).getT
  }

  @And("I click on {string} and check world statistics")
  public void accessWorldwideMetrics(String world) {
    driver.findElement(By.linkText(world)).click();
    assertEquals( expected: "Select a date and check world's Covid-19 statistics:", driver.findElement(By.cssSelector( cssSelector: "h1")).getText());
    assertEquals( expected: "World", driver.findElement(By.cssSelector( cssSelector: "h4")).getText());
  }

  @And("I click on {string} and observe a page to Select a country")
  public void accessCountrywideMetrics(String country) {
    driver.findElement(By.linkText(country)).click();
    assertEquals( expected: "Country\nISO ALPHA-3", driver.findElement(By.cssSelector( cssSelector: "h4")).getText());
  }

  @Then("I choose {string} to check it's statistics")
  public void chooseCountry(String country) {
    driver.findElement(By.linkText( linkText: "Country Statistics")).click();
    {
      WebElement dropdown = driver.findElement(By.cssSelector( cssSelector: "select"));
      dropdown.findElement(By.xpath("//option[. = '" + country + "']")).click();
    }
    assertEquals( expected: "Portugal\nPRT", driver.findElement(By.cssSelector( cssSelector: "h4")).getText());

    driver.quit();
  }
}
```

Fig. 2 – BDD approach mixed with Selenium automation test

```
Feature: Covid metrics
  To allow a user to check world and specific country statistics with web automation.
  Scenario: Check Covid-19 statistics
    When I access "http://localhost:8080/"
    And I click on "Worldwide Statistics" and check world statistics
    And I click on "Country Statistics" and observe a page to Select a country
    Then I choose "Portugal" to check it's statistics
```

Fig. 3 – covidmetrics.feature file which contains the get data scenario

3.2 Unit and integration testing

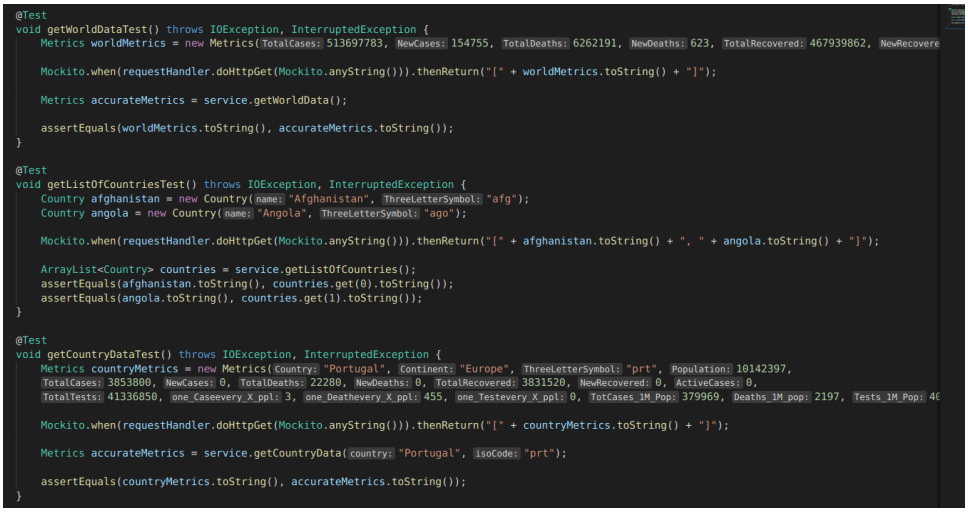
I used unit and integration tests while testing the Model layer, the Service Layer, the Controller Layer and the Cache, inspiring my knowledge in the previous taught labs.

```java
@Test
void covidDataMetricsTest() {

    // country metrics constructor
    Metrics countryMetrics = new Metrics(Country: "Angola", Continent: "Africa", ThreeLetterSymbol: "ago", Population: 34732793, TotalCases: 99194,
    NewCases: 0, TotalDeaths: 1900, NewDeaths: 0, TotalRecovered: 97149, NewRecovered: 0, ActiveCases: 145, TotalTests: 1499795,
    one_Caseevery_X_ppl: 350, one_Deathevery_X_ppl: 18280, one_Testevery_X_ppl: 23, TotCases_1M_Pop: 2856, Deaths_1M_pop: 55, Tests_1M_Pop: 43181);

    assertEquals(expected: "Angola", countryMetrics.getCountry());
    assertEquals(expected: "Africa", countryMetrics.getContinent());
    assertEquals(expected: "ago", countryMetrics.getThreeLetterSymbol());
    assertEquals(expected: 34732793, countryMetrics.getPopulation());
    assertEquals(expected: 99194, countryMetrics.getTotalCases());
    assertEquals(expected: 0, countryMetrics.getNewCases());
    assertEquals(expected: 1900, countryMetrics.getTotalDeaths());
    assertEquals(expected: 0, countryMetrics.getNewDeaths());
    assertEquals(expected: 97149, countryMetrics.getTotalRecovered());
    assertEquals(expected: 0, countryMetrics.getNewRecovered());
    assertEquals(expected: 145, countryMetrics.getActiveCases());
    assertEquals(expected: 1499795, countryMetrics.getTotalTests());
    assertEquals(expected: 350, countryMetrics.getOne_Caseevery_X_ppl());
    assertEquals(expected: 18280, countryMetrics.getOne_Deathevery_X_ppl());
    assertEquals(expected: 23, countryMetrics.getOne_Testevery_X_ppl());
    assertEquals(expected: 2856, countryMetrics.getTotCases_1M_Pop());
    assertEquals(expected: 55, countryMetrics.getDeaths_1M_pop());
    assertEquals(expected: 43181, countryMetrics.getTests_1M_Pop());

    // deaths + recovered <= total cases
    long totalCases = countryMetrics.getTotalCases();
    long deaths = countryMetrics.getTotalDeaths();
    long recovered = countryMetrics.getTotalRecovered();

    assertTrue((deaths + recovered) <= totalCases);

    // deaths + recovered + active cases = total cases
    long activeCases = countryMetrics.getActiveCases();

    assertEquals(totalCases, deaths + recovered + activeCases);

}
```

Fig. 4 – Unit Test for Model layer

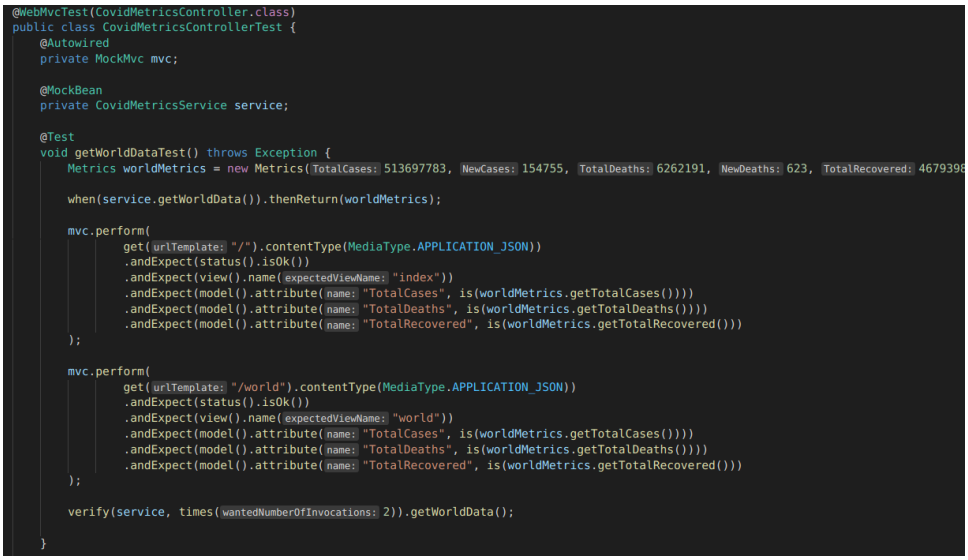
```java
@Test
void addEntryTest() {
    this.cache.addEntry(key: "Portugal", content: 10000000);

    assertEquals(expected: 10000000, this.cache.getEntry(key: "Portugal"));
    assertEquals(expected: 1, this.cache.size());
}

@Test
void clearEntryTest() {
    this.cache.addEntry(key: "Portugal", content: 10000000);
    this.cache.addEntry(key: "Romania", content: 8000000);
    assertEquals(expected: 2, this.cache.size());

    this.cache.clearEntry(key: "Romania");
    assertEquals(expected: 1, this.cache.size());
    assertFalse(this.cache.clearEntry(key: "Romania"));

    this.cache.clearEntry(key: "Portugal");
    assertEquals(expected: 0, this.cache.size());
}

@Test
void getEntryTest() {
    this.cache.addEntry(key: "Portugal", content: 10000000);
    // First request
    this.cache.getEntry(key: "Portugal");
    assertEquals(expected: 1, this.cache.getRequests()); // 1 request
    assertEquals(expected: 1, this.cache.getHits()); // 1 hit
    assertEquals(expected: 0, this.cache.getMisses()); // 0 misses

    // Second request
    assertEquals(expected: null, this.cache.getEntry(key: "Romania"));
    assertEquals(expected: 2, this.cache.getRequests()); // 2 requests
    assertEquals(expected: 1, this.cache.getHits()); // 1 hit
    assertEquals(expected: 1, this.cache.getMisses()); // 1 miss
}
```

Fig. 5 – Unit Test for Cache



```
@Test
void getWorldDataTest() throws IOException, InterruptedException {
    Metrics worldMetrics = new Metrics(TotalCases: 513697783, NewCases: 154755, TotalDeaths: 6262191, NewDeaths: 623, TotalRecovered: 467939862, NewRecovere

    Mockito.when(requestHandler.doHttpGet(Mockito.anyString())).thenReturn("[" + worldMetrics.toString() + "]");

    Metrics accurateMetrics = service.getWorldData();

    assertEquals(worldMetrics.toString(), accurateMetrics.toString());
}

@Test
void getListOfCountriesTest() throws IOException, InterruptedException {
    Country afghanistan = new Country(name: "Afghanistan", ThreeLetterSymbol: "afg");
    Country angola = new Country(name: "Angola", ThreeLetterSymbol: "ago");

    Mockito.when(requestHandler.doHttpGet(Mockito.anyString())).thenReturn("[" + afghanistan.toString() + ", " + angola.toString() + "]");

    ArrayList<Country> countries = service.getListOfCountries();
    assertEquals(afghanistan.toString(), countries.get(0).toString());
    assertEquals(angola.toString(), countries.get(1).toString());
}

@Test
void getCountryDataTest() throws IOException, InterruptedException {
    Metrics countryMetrics = new Metrics(Country: "Portugal", Continent: "Europe", ThreeLetterSymbol: "prt", Population: 10142397,
    TotalCases: 3853800, NewCases: 0, TotalDeaths: 22280, NewDeaths: 0, TotalRecovered: 3831520, NewRecovered: 0, ActiveCases: 0,
    TotalTests: 41336850, one_Caseevery_X_ppl: 3, one_Deathevery_X_ppl: 455, one_Testevery_X_ppl: 0, TotCases_1M_Pop: 379969, Deaths_1M_pop: 2197, Tests_1M_Pop: 4(

    Mockito.when(requestHandler.doHttpGet(Mockito.anyString())).thenReturn("[" + countryMetrics.toString() + "]");

    Metrics accurateMetrics = service.getCountryData(country: "Portugal", isoCode: "prt");

    assertEquals(countryMetrics.toString(), accurateMetrics.toString());
}
```

Fig. 6 – Unit Test using Mockito for Service



```
@WebMvcTest(CovidMetricsController.class)
public class CovidMetricsControllerTest {
    @Autowired
    private MockMvc mvc;

    @MockBean
    private CovidMetricsService service;

    @Test
    void getWorldDataTest() throws Exception {
        Metrics worldMetrics = new Metrics(TotalCases: 513697783, NewCases: 154755, TotalDeaths: 6262191, NewDeaths: 623, TotalRecovered: 4679398

        when(service.getWorldData()).thenReturn(worldMetrics);

        mvc.perform(
                get(urlTemplate: "/").contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(view().name(expectedViewName: "index"))
                .andExpect(model().attribute(name: "TotalCases", is(worldMetrics.getTotalCases())))
                .andExpect(model().attribute(name: "TotalDeaths", is(worldMetrics.getTotalDeaths())))
                .andExpect(model().attribute(name: "TotalRecovered", is(worldMetrics.getTotalRecovered())))
        );

        mvc.perform(
                get(urlTemplate: "/world").contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(view().name(expectedViewName: "world"))
                .andExpect(model().attribute(name: "TotalCases", is(worldMetrics.getTotalCases())))
                .andExpect(model().attribute(name: "TotalDeaths", is(worldMetrics.getTotalDeaths())))
                .andExpect(model().attribute(name: "TotalRecovered", is(worldMetrics.getTotalRecovered())))
        );

        verify(service, times(wantedNumberOfInvocations: 2)).getWorldData();

    }
```

Fig. 7 – Integration Test with MockMvc for Controller

3.3 Functional testing

Functional testing was developed using Selenium, as explained above in point 3.1.

3.4 Code quality analysis

Because of memory problems, I wasn't able to perform the code quality analysis using SonarQube (elastic search problem), so, I opted to use SonarCloud instead.

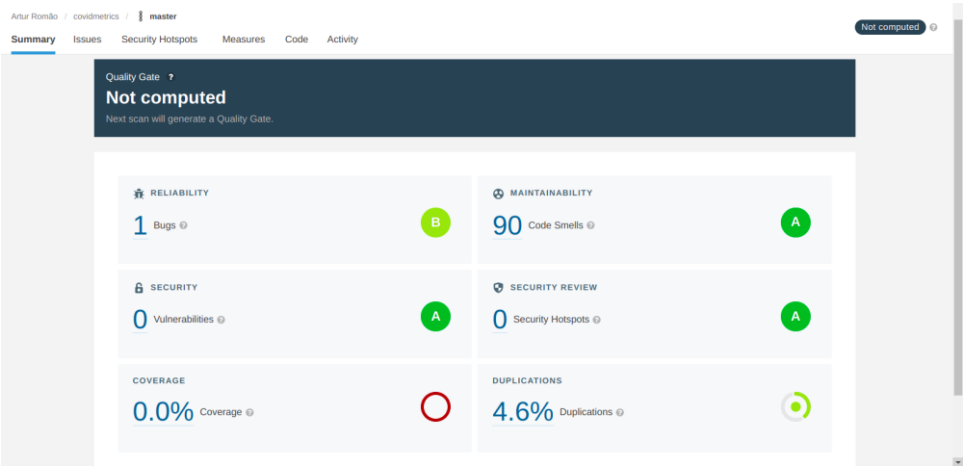


Fig. 8 – First code quality analysis with 1 bug

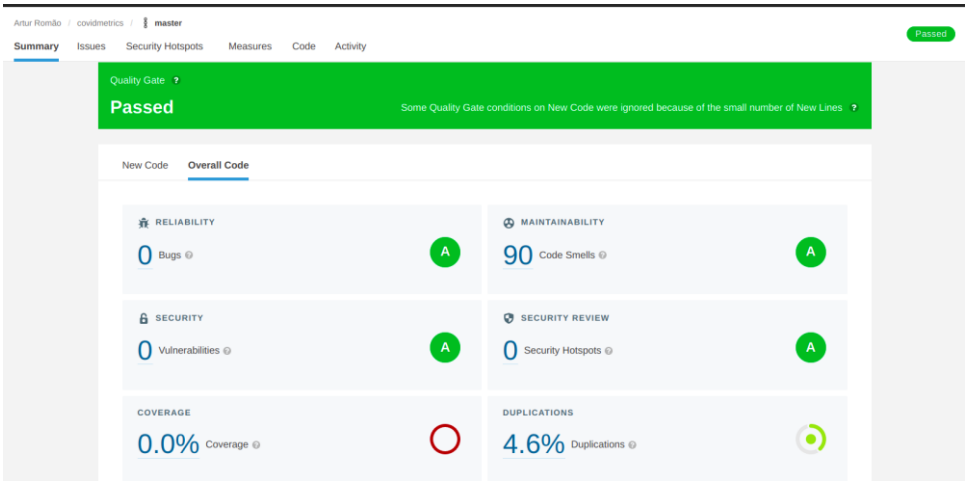


Fig. 9 – Quality gate passing after resolving bug

Unfortunately, jacoco plugin didn't work, so I wasn't able to check my code coverage.

3.5 Continuous integration pipeline [optional]

I think SonarCloud provides Continuous Integration.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/artur-romao/tqs_98470/tree/main/hw1
Video demo	https://www.youtube.com/watch?v=s_rYQRSrZF0
QA dashboard (online)	[**optional**;]
CI/CD pipeline	[**optional**;]
Deployment ready to use	[**optional**; to run the project perform the command "./mvnw spring-boot:run" in the "covidmetrics" folder]

Reference materials and some relevant links

API used: https://rapidapi.com/vaccovidlive-vaccovidlive-default/api/vaccovid-coronavirus-vaccine-and-treatment-tracker/
https://linuxtut.com/en/966a9c85ed6036126dfa/
https://www.baeldung.com/spring-mvc-and-the-modelattribute-annotation
https://reflectoring.io/spring-boot-web-controller-test/
https://www.wimdeblauwe.com/blog/2021/01/11/string-concatenation-with-thymeleaf/

(And a lot of stack overflow 😬)