universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

HW1: Mid-term assignment report

Renato Alexandre Lourenço Dias [98380], v2022-05-02

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.
The product name is COVIData and it has the purpose of provide COVID-19 data including cases, deaths and tests by country and date. It includes a Web App with a REST API that has a local cache implemented.
As seen in the figure presented below, the home page of the web app has a design where the user can see the available features and navigate to them through the navbar or the cards available in the lower section. The home page also allows the user to see the last COVID-19 stats in a country by searching for that country.

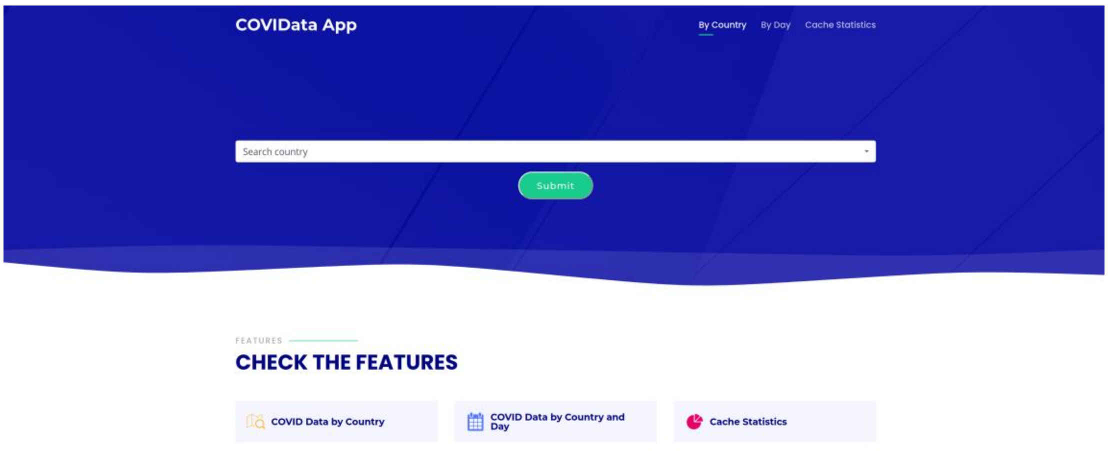


Figure 1: COVIData Web App home page

1.2 Current limitations

The only limitation is caused by the third-party API used: VACCOVID just have COVID-19 incidence data until January 30th, so that users cannot search for recent data.

2 Product specification

2.1 Functional scope and supported interactions

The product can be used by users who want to check COVID-19 incidence data.
The main usage scenarios are:

- Choose a country and analyze the information shown
- Choose a country and a date and analyze the information shown
- Check cache values

This first scenario is the one showcased in figure 1. The user selects the country and then is sent to a page with the last info about that country's COVID-19 incidence. In figure 2 is presented the flow and result page of this scenario.

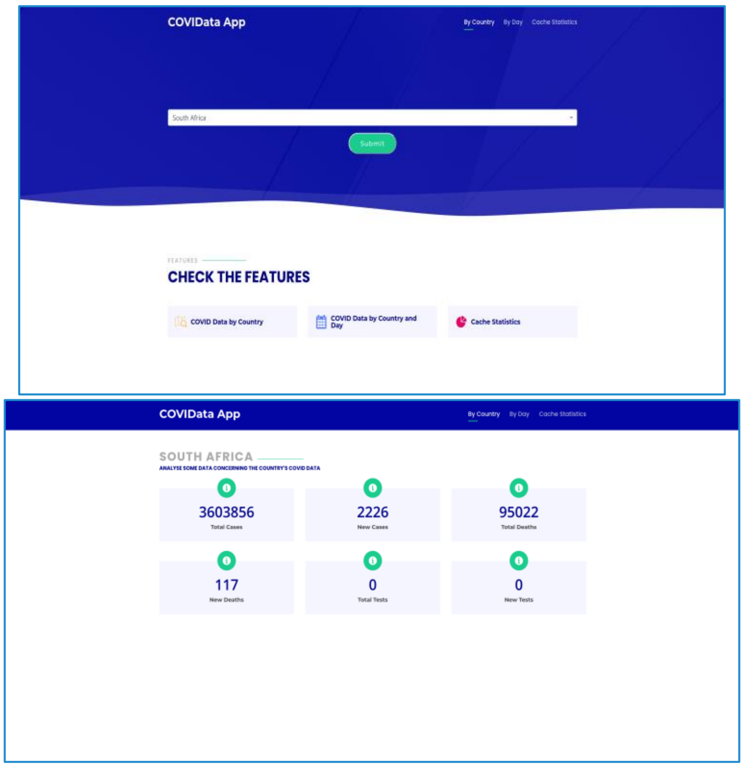
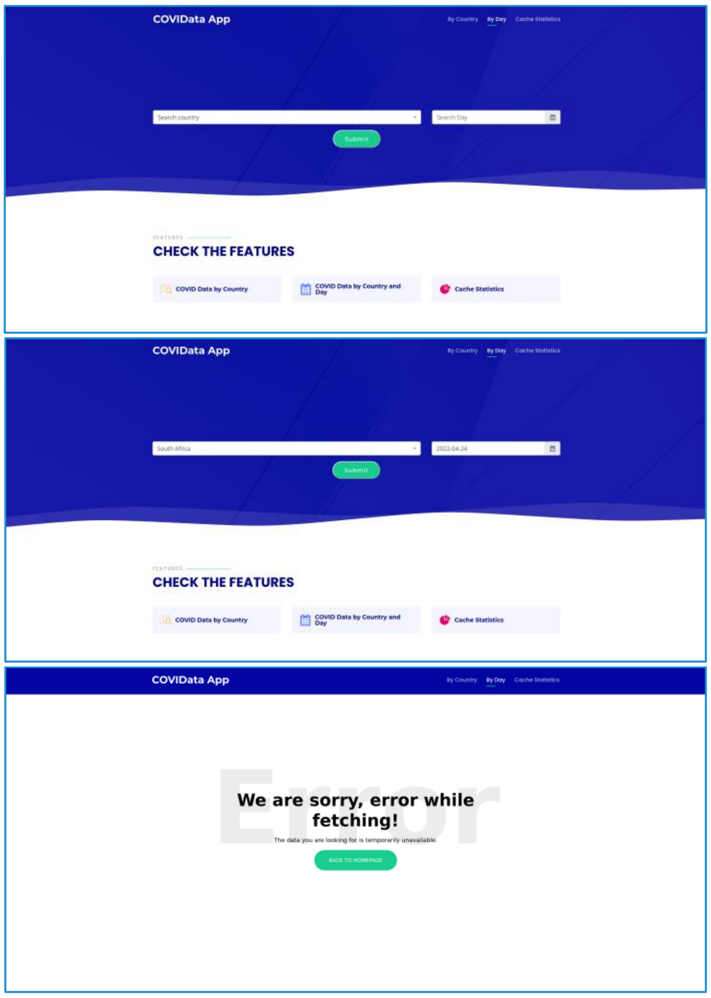


Figure 2: Last COVID-19 data of South Africa

The second feature is a lot similar to the previous one with the major difference that instead of asking for the last COVID-19 stats, the user searches the stats of a given day of a country. Thus, in addition to selecting the country, the user also selects the day he wants to see. If the user chooses a day without any COVID-19 stats (third-party API limitation), it will appear a message explaining the data is not available. Figure 3 showcases the flow and result page of this scenario.



We are sorry, error while fetching!

The data you are looking for is temporarily unavailable.

Figure 3: COVID-19 data of South Africa in 24/04/2022

Note that the process of getting COVID-19 data by date is not done yet so that is not able to show the stats of that country of the given date.

Finally, the last feature enables the user to check the number of requests the service has already served and to see what of those were hits (were stored in the cache) or misses (weren't stored in the cache). However, due to some problem, this page doesn't load (as we can see in the product demo available on YouTube).

2.2 System architecture

The technologies and frameworks used to develop this project were:

- Spring Boot with Maven to develop the REST API
- Thymeleaf, Bootstrap, HTML, CSS3, JavaScript/ jQuery and Ajax to develop the Web App
- Junit5, Hamcrest, Mockito and Selenium to test the service

2.3 API for developers

The public domain of the developed REST API has three available endpoints:

- http://localhost:8080/api/countries/{con} to fetch a list of countries of a specific continent.
- http://localhost:8080/api/countries/all to fetch a list of all countries in the world.
- http://localhost:8080/api/covidata/{countryId} to fetch the last COVID-19 stats in a specific country

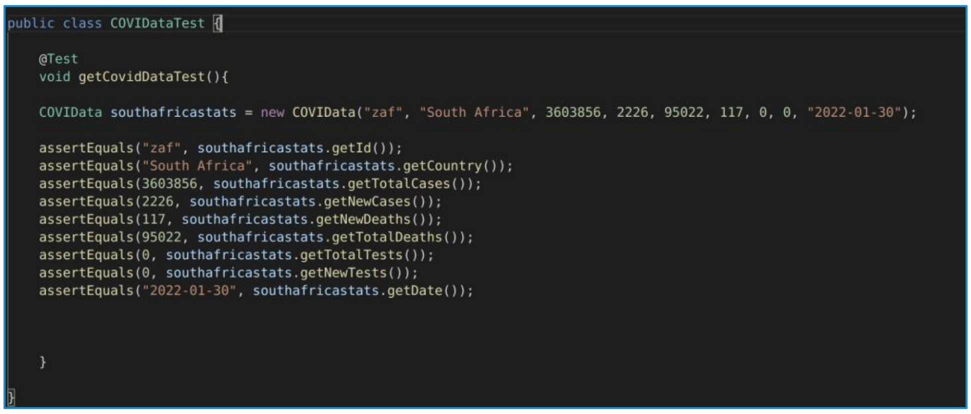
3 Quality assurance

3.1 Overall strategy for testing

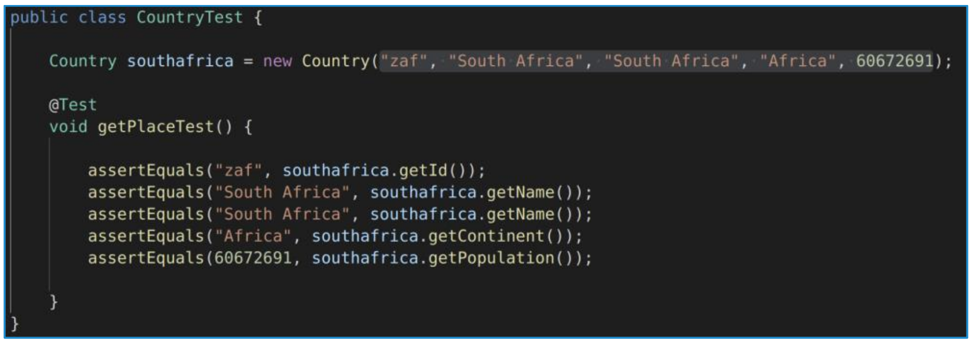
This section is all about testing and quality assurance. During the development of the system, I tried to use TDD (Test-Driven Development) in a way that I would just create the classes' files and skeleton, implement the tests I thought fit and then implemented the code, taking the situations tested into account. However, because of the time I had to develop the project combined with the relative inexperience with the development strategy, I would not think about all the possible cases and only add tests to the missing scenarios after writing the actual code. Some of the tests are not very well written, so that some of them are failing. Taking into account all of this, and as a conclusion on this topic, I consider that in a global scope I was able to successively apply TDD in this project, although all these setbacks. Finally, to implement unit and integration tests, it was used JUnit 5 and, when using mocks, Mockito.

3.2 Unit and integration testing

Pure unit tests (without mocks) were used to test local cache implementation (Cache), COVIData, Country and Keys.



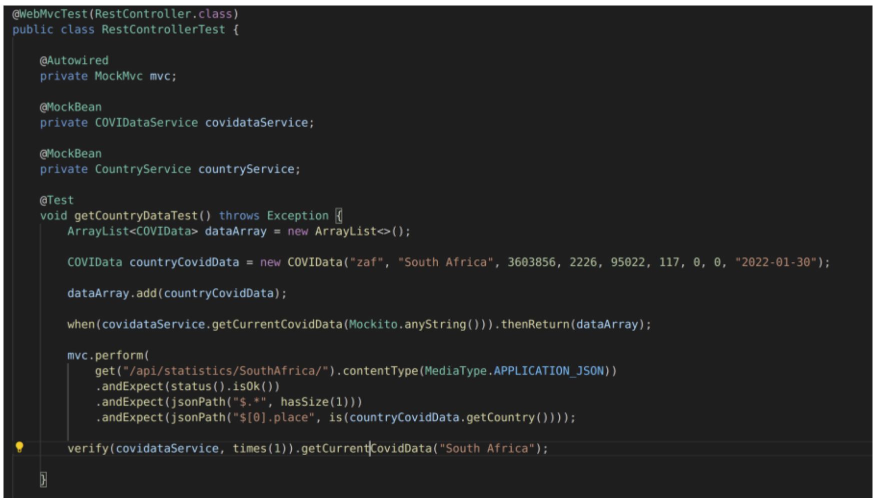
```java
public class COVIDataTest {

    @Test
    void getCovidDataTest(){

    COVIData southafricastats = new COVIData("zaf", "South Africa", 3603856, 2226, 95022, 117, 0, 0, "2022-01-30");

    assertEquals("zaf", southafricastats.getId());
    assertEquals("South Africa", southafricastats.getCountry());
    assertEquals(3603856, southafricastats.getTotalCases());
    assertEquals(2226, southafricastats.getNewCases());
    assertEquals(117, southafricastats.getNewDeaths());
    assertEquals(95022, southafricastats.getTotalDeaths());
    assertEquals(0, southafricastats.getTotalTests());
    assertEquals(0, southafricastats.getNewTests());
    assertEquals("2022-01-30", southafricastats.getDate());


    }
}
```



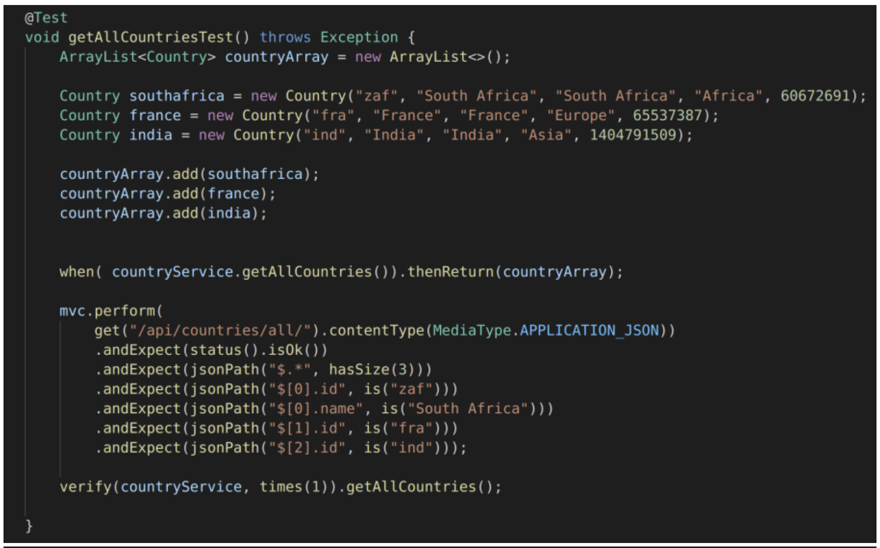
```java
public class CountryTest {

    Country southafrica = new Country("zaf", "South Africa", "South Africa", "Africa", 60672691);

    @Test
    void getPlaceTest() {

        assertEquals("zaf", southafrica.getId());
        assertEquals("South Africa", southafrica.getName());
        assertEquals("South Africa", southafrica.getName());
        assertEquals("Africa", southafrica.getContinent());
        assertEquals(60672691, southafrica.getPopulation());

    }
}
```

Figure 4: COVIData and Country Unit Testing

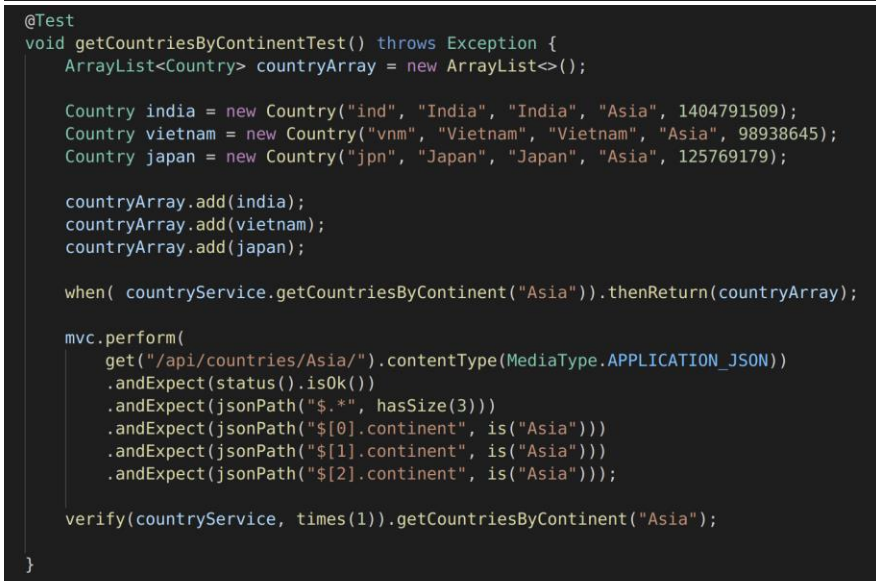
Unit tests with mocking were used whenever it was needed to isolate a specific component to test its behaviour without any influence of other components. Some of these situations were in the process of testing: COVIDataService.



```java
@WebMvcTest(RestController.class)
public class RestControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private COVIDataService covidataService;

    @MockBean
    private CountryService countryService;

    @Test
    void getCountryDataTest() throws Exception {
        ArrayList<COVIData> dataArray = new ArrayList<>();

        COVIData countryCovidData = new COVIData("zaf", "South Africa", 3603856, 2226, 95022, 117, 0, 0, "2022-01-30");

        dataArray.add(countryCovidData);

        when(covidataService.getCurrentCovidData(Mockito.anyString())).thenReturn(dataArray);

        mvc.perform(
            get("/api/statistics/SouthAfrica/").contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.*", hasSize(1)))
            .andExpect(jsonPath("$[0].place", is(countryCovidData.getCountry())));

        verify(covidataService, times(1)).getCurrentCovidData("South Africa");

    }
```



```java
@Test
void getAllCountriesTest() throws Exception {
    ArrayList<Country> countryArray = new ArrayList<>();

    Country southafrica = new Country("zaf", "South Africa", "South Africa", "Africa", 60672691);
    Country france = new Country("fra", "France", "France", "Europe", 65537387);
    Country india = new Country("ind", "India", "India", "Asia", 1404791509);

    countryArray.add(southafrica);
    countryArray.add(france);
    countryArray.add(india);


    when( countryService.getAllCountries()).thenReturn(countryArray);

    mvc.perform(
        get("/api/countries/all/").contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.*", hasSize(3)))
        .andExpect(jsonPath("$[0].id", is("zaf")))
        .andExpect(jsonPath("$[0].name", is("South Africa")))
        .andExpect(jsonPath("$[1].id", is("fra")))
        .andExpect(jsonPath("$[2].id", is("ind")));

    verify(countryService, times(1)).getAllCountries();

}
```

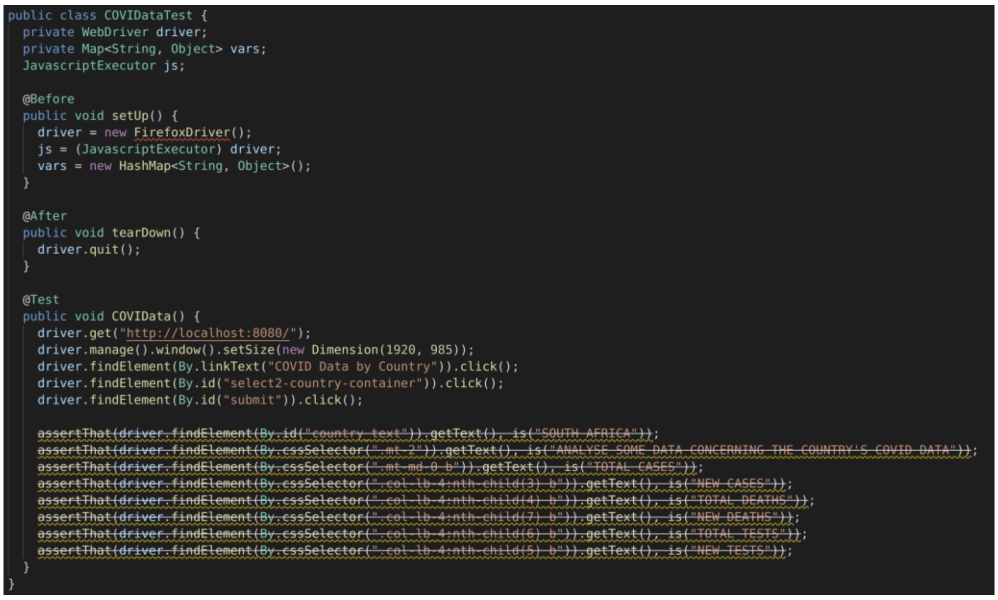


```java
@Test
void getCountriesByContinentTest() throws Exception {
    ArrayList<Country> countryArray = new ArrayList<>();

    Country india = new Country("ind", "India", "India", "Asia", 1404791509);
    Country vietnam = new Country("vnm", "Vietnam", "Vietnam", "Asia", 98938645);
    Country japan = new Country("jpn", "Japan", "Japan", "Asia", 125769179);

    countryArray.add(india);
    countryArray.add(vietnam);
    countryArray.add(japan);

    when( countryService.getCountriesByContinent("Asia")).thenReturn(countryArray);

    mvc.perform(
        get("/api/countries/Asia/").contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.*", hasSize(3)))
        .andExpect(jsonPath("$[0].continent", is("Asia")))
        .andExpect(jsonPath("$[1].continent", is("Asia")))
        .andExpect(jsonPath("$[2].continent", is("Asia")));

    verify(countryService, times(1)).getCountriesByContinent("Asia");

}
```

Figure 5: RestController Unit Testing with Mocks

3.3 Functional testing

The last type of testing implemented was functional testing. These tests were implemented using Selenium and a specification of WebDriver, using ChromeDriver.



```java
public class COVIDataTest {
  private WebDriver driver;
  private Map<String, Object> vars;
  JavascriptExecutor js;

  @Before
  public void setUp() {
    driver = new FirefoxDriver();
    js = (JavascriptExecutor) driver;
    vars = new HashMap<String, Object>();
  }

  @After
  public void tearDown() {
    driver.quit();
  }

  @Test
  public void COVIData() {
    driver.get("http://localhost:8080/");
    driver.manage().window().setSize(new Dimension(1920, 985));
    driver.findElement(By.linkText("COVID Data by Country")).click();
    driver.findElement(By.id("select2-country-container")).click();
    driver.findElement(By.id("submit")).click();

    ~~assertThat(driver.findElement(By.id("country-text")).getText(), is("SOUTH AFRICA"))~~;
    ~~assertThat(driver.findElement(By.cssSelector(".mt-2")).getText(), is("ANALYSE SOME DATA CONCERNING THE COUNTRY'S COVID DATA"))~~;
    ~~assertThat(driver.findElement(By.cssSelector(".mt-md-0 b")).getText(), is("TOTAL CASES"))~~;
    ~~assertThat(driver.findElement(By.cssSelector(".col-lb-4:nth-child(3) b")).getText(), is("NEW CASES"))~~;
    ~~assertThat(driver.findElement(By.cssSelector(".col-lb-4:nth-child(4) b")).getText(), is("TOTAL DEATHS"))~~;
    ~~assertThat(driver.findElement(By.cssSelector(".col-lb-4:nth-child(7) b")).getText(), is("NEW DEATHS"))~~;
    ~~assertThat(driver.findElement(By.cssSelector(".col-lb-4:nth-child(6) b")).getText(), is("TOTAL TESTS"))~~;
    ~~assertThat(driver.findElement(By.cssSelector(".col-lb-4:nth-child(5) b")).getText(), is("NEW TESTS"))~~;
  }
}
```

Figure 6: COVIData Funcional Test

3.4 Code quality analysis

To perform static code analysis of the developed code, the project would be analysed with SonarQube and it would it be also used JaCoCo to create reports about code coverage so that SonarQube could use and integrate that information in the analysis. However, this could not be done because of some errors when trying to run SonarQube in the project folder.

4 References & resources

Project resources

Resource:	**URL/location:**
Git repository	https://github.com/Renatito122/TQS_98380
Video demo	https://youtu.be/c33PH4hPHWc

Reference materials

- VACCOVID API: https://rapidapi.com/vaccovidlive-vaccovidlive-default/api/vaccovid-coronavirus-vaccine-and-treatment-tracker/
- Generate Swagger Docs: https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api