

HW1: Mid-term assignment report

Pedro Monteiro [97484], v2022-05-02

1	Introduction	1
1.1	Overview of the work	1
1.2	Current limitations	2
2	Product specification	2
2.1	Functional scope and supported interactions	2
2.2	System architecture	4
2.3	API for developers	5
3	Quality assurance	7
3.1	Overall strategy for testing	7
3.2	Unit and integration testing	7
3.3	Functional testing	12
3.4	Code quality analysis	13
4	References & resources	13

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 project, named **CovidInfo**, aims to present data related to Covid-19 (SARS-CoV-2) for each country/territory. The search can be done by writing the country to be searched or by selecting it from a list (this topic will be covered in the following points).

It uses an External API, [RapidAPI](#), from which the data is obtained. It also has a cache implemented to ensure that data is fetched faster. To better understand what is going on, all operations are logged, using Log4j2 (info, debug and errors).

Tests were implemented throughout the application, whether unit, functional or integration tests.

1.2 Current limitations

Most of the problems focused on the external API as it has values that had to be treated to be displayed on web application (null values) and also certain data related to some countries that did not exist.

The product still has some features that have not been implemented:

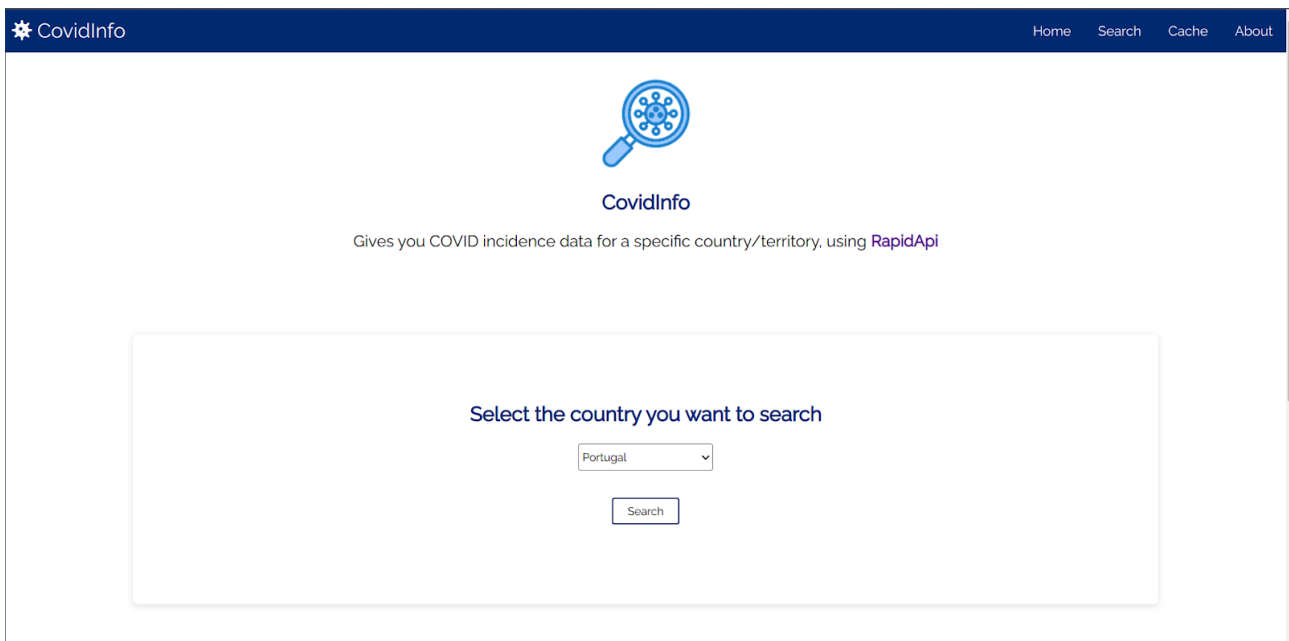
- More than one external source (remote API)
- Continuous Integration framework

2 Product specification

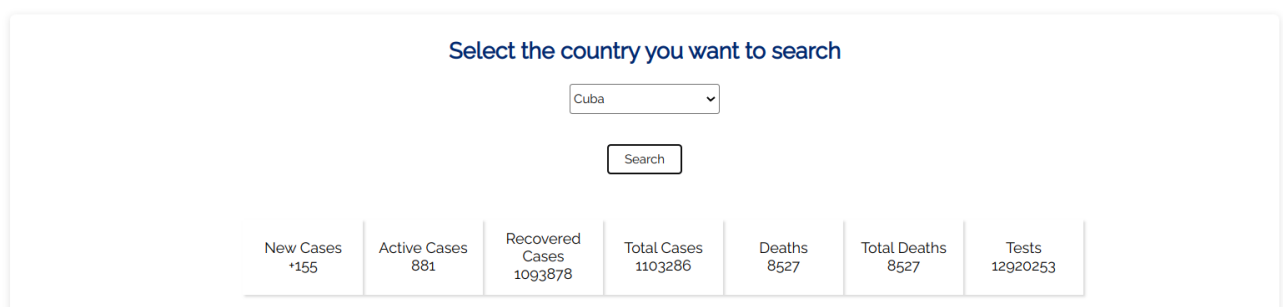
2.1 Functional scope and supported interactions

The application will be used by customers who want to obtain information about data related to Covid-19.

It is a very simple application. Users can search in the list or can write the country for which they want to search. After clicking the search button the data, such as new cases, active cases, tests, etc., will appear. In addition, it is possible to observe the statistics related to the cache.



After clicking the search button, for example for Cuba, the following will appear:



New Cases	Active Cases	Recovered Cases	Total Cases	Deaths	Total Deaths	Tests
155	881	1093878	1103286	8527	8527	12920253

In case the country does not exist/is not valid, a simple error message appears:

Write the country you want to search

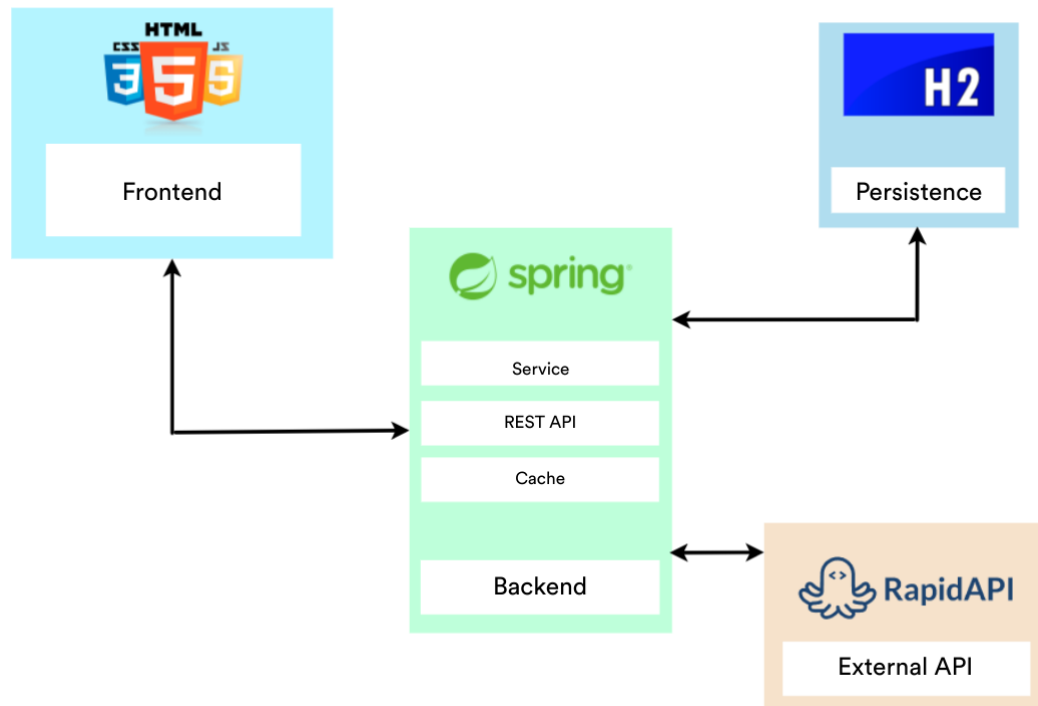
Country Not Available

At the bottom of the web page the user can see cache section. It is possible to see **Hits**, number of times that information has been retrieved from Cache, **Misses**, number of calls to external API and **Requests**, the total number of requests done by the user.

See Cache Details

Hits 3	Misses 4	Requests 7
-----------	-------------	---------------

2.2 System architecture



Technologies such as HTML, CSS and JavaScript were used for the frontend, due to the fact that they are relatively easy to use and that there is already a lot of practice in building websites with these technologies, which made this easier, faster and simpler.

For backend development, technologies required by the teacher were used, being developed an API (CovidController), a service (CovidService) that allows communicating with the external API, a repository (CountryRepository) implemented using H2 database and a Cache that allows get faster responses without having to make External API calls.

2.3 API for developers

There are multiple endpoints, as we can see in the image below. Mostly endpoints are used to list information about countries and covid data, and there is one used to show the cache details (hits, misses and requests).

CovidInfo	Access Covid Data around the world	▼
default ^		
GET	/api/v1	Get all data from External API
GET	/api/v1/countries	Get all countries
GET	/api/v1/countries/{country}	Find Covid Data for a specific country
GET	/api/v1/countries/data	Find Covid Data for all countries
GET	/api/v1/cache	Get cache statistics

- /api/v1 - has all data that comes from api
- /api/v1/countries - has the names of all countries available

```
▼ [  
  "Kiribati",  
  "Nauru",  
  "Marshall-Islands",  
  "Palau",  
  "Cook-Islands",  
  "Saint-Helena",  
  "St-Barth",  
  "Comoros",  
  "Niue",  
  "Sierra-Leone",  
  "Antigua-and-Barbuda",  
  "Chad",  
  "Liberia",
```

- /api/v1/countries/{country} - has information about one specific country

```
▼ {  
  "name": "Cuba",  
  "newCases": "+190",  
  "activeCases": "897",  
  "recoveredCases": "1093707",  
  "totalCases": "1103131",  
  "newDeaths": "+1",  
  "totalDeaths": "8527",  
  "totalTests": "12920253"  
}
```

- api/v1/countries/data - has information about all countries available

```
▼ [  
  ▼ {  
    "name": "Kiribati",  
    "newCases": "+2",  
    "activeCases": "470",  
    "recoveredCases": "2601",  
    "totalCases": "3084",  
    "newDeaths": "null",  
    "totalDeaths": "13",  
    "totalTests": "null"  
  },  
  ▼ {  
    "name": "Nauru",  
    "newCases": "+1",  
    "activeCases": "2",  
    "recoveredCases": "3",  
    "totalCases": "5",  
    "newDeaths": "null",  
    "totalDeaths": "null",  
    "totalTests": "null"  
  },  
  ▼ {  
    "name": "Marshall-Islands",  
    "newCases": "null",  
    "activeCases": "3",  
    "recoveredCases": "14",  
    "totalCases": "17",  
    "newDeaths": "null",  
    "totalDeaths": "null",  
    "totalTests": "null"  
  }  
]
```

- `api/v1/cache` - has cache details, the number of hits, misses and requests

```
{
  "hits": 3,
  "misses": 7,
  "requests": 10
}
```

3 Quality assurance

3.1 Overall strategy for testing

I decided to create the skeleton of the REST API first, since I already had an idea of the tests to be carried out due to the labs solved throughout the semester. For unit tests was used **Junit5**, and for integration **Mockito** and **SpringBoot MockMvc**. For functional tests, on the web application interface (frontend) was used **Selenium**.

3.2 Unit and integration testing

Unit Tests

I wrote these tests on the entity, repository and cache classes. In `Country.java` (entity class) the tests are simple asserts used to check if values match what is expected.

```
public class EntityTest {
    Country country = new Country(name: "CountryTest", newCases: "3", activeCases: "631263", recoveredCases: "1000",

    @Test
    void countryTest() {
        assertEquals(expected: "CountryTest", country.getName());
        assertEquals(expected: "3", country.getNewCases());
        assertEquals(expected: "631263", country.getActiveCases());
        assertEquals(expected: "1000", country.getRecoveredCases());
        assertEquals(expected: "1500000", country.getTotalCases());
        assertEquals(expected: "0", country.getNewDeaths());
        assertEquals(expected: "37121", country.getTotalDeaths());
        assertEquals(expected: "231231", country.getTotalTests());
    }

    @Test
    void invalidValuesCountryTest() {
        // all this values should 0 or bigger, can't be lower than 0
        assertThat(Integer.parseInt(country.getNewCases()), greaterThan(-1));
        assertThat(Integer.parseInt(country.getActiveCases()), greaterThan(-1));
        assertThat(Integer.parseInt(country.getRecoveredCases()), greaterThan(-1));
        assertThat(Integer.parseInt(country.getTotalCases()), greaterThan(-1));
        assertThat(Integer.parseInt(country.getNewDeaths()), greaterThan(-1));
        assertThat(Integer.parseInt(country.getTotalDeaths()), greaterThan(-1));
        assertThat(Integer.parseInt(country.getTotalTests()), greaterThan(-1));
    }
}
```

After testing the entity class, Country.java, unit tests were used in the Cache class, Cache.java. It is mainly tested the addition of values to the cache, the number of requests made by the user (hits, misses and requests) and if the cache is cleared after some time.

```
@Test
void addValueToCacheTest(){
    assertEquals(expected: 0, cache.getCacheSize());
    cache.addToCache(key: "CountryTest", new Country(name: "CountryTest"));
    assertEquals(expected: 1, this.cache.getCacheSize());
    assertEquals(expected: true, this.cache.containsItem(key: "CountryTest"));
}

@Test
void hitsMissesAndRequestsTest(){
    Country c1 = new Country(name: "c1");
    cache.addToCache(key: "c1", c1);
    cache.getCountryFromCache(key: "c1");
    cache.getCountryFromCache(key: "c1");
    cache.getCountryFromCache(key: "null");
    assertEquals(expected: 2, cache.getHits());
    assertEquals(expected: 1, cache.getMisses());
    assertEquals(expected: 3, cache.getRequests());
}

@Test
void cleanAfterTimeTest() throws InterruptedException {
    Country c1 = new Country(name: "c1");
    cache.addToCache(key: "c1", c1);
    cache.cacheTimer(key: "c1", timeToLive: 5);
    Thread.sleep(10); // wait for item to be removed
    assertEquals(expected: 0, this.cache.getCacheSize());
}
```


Like the Cache and the Country classes, also the service class was similarly tested. Here the main focus was to verify that the countries were being requested correctly and also what happens in case an invalid country is requested.

```
@ExtendWith(MockitoExtension.class)
public class ServiceTest {

    @Mock(lenient=true)
    private CountryRepository countryRepository;

    @Mock
    private Cache cache;

    @BeforeEach
    void setUp() throws IOException, InterruptedException {
        cache = new Cache();
        Country c1 = new Country(name: "c1", newCases: "3", activeCases: "631263", recoveredCases: "1000", totalCases: "15");
        Country c2 = new Country(name: "c2", newCases: "23", activeCases: "423423", recoveredCases: "123", totalCases: "12");
        cache.addToCache(key: "c1", c1);
        cache.addToCache(key: "c2", c2);

        when(countryRepository.findByName(c1.getName())).thenReturn(c1);
        when(countryRepository.findByName(c2.getName())).thenReturn(c2);
    }

    @AfterEach
    void tearDown(){
    }

    @Test
    void getCountryTest() throws IOException, InterruptedException {
        String countryName = "c1";
        int newCases = Integer.parseInt("3");
        int newDeaths = Integer.parseInt("0");
        Country foundC1 = cache.getCountryFromCache(key: "c1");
        assertEquals(foundC1.getName(), countryName);
        assertEquals(Integer.parseInt(foundC1.getNewCases()), newCases);
        assertEquals(Integer.parseInt(foundC1.getNewDeaths()), newDeaths);
    }
}
```

```
@Test
void getInvalidCountryTest() throws IOException, InterruptedException {
    Country foundC1 = cache.getCountryFromCache(key: "abc");
    assertEquals(foundC1, expected: null);
}
```

Finally, unit tests were also applied to the repository. It was used **TestEntityManager** and the **@DataJpaTest** annotation.

```
@DataJpaTest
public class CountryRepositoryTest {
    @Autowired
    private CountryRepository countryRepository;

    @Autowired
    private TestEntityManager testEntityManager;

    @Test
    void findCountryByNameTest(){
        Country c1 = new Country(name: "c1", newCases: "3", activeCases: "631263", recoveredCases: "1000", totalCases: "150000");
        testEntityManager.persistAndFlush(c1);

        Country foundC1 = countryRepository.findByName(c1.getName());
        assertEquals(foundC1, c1);
    }

    @Test
    void findAllCountriesTest(){
        Country c1 = new Country(name: "c1", newCases: "3", activeCases: "631263", recoveredCases: "1000", totalCases: "150000");
        Country c2 = new Country(name: "c2", newCases: "3", activeCases: "631263", recoveredCases: "1000", totalCases: "150000");
        Country c3 = new Country(name: "c3", newCases: "3", activeCases: "631263", recoveredCases: "1000", totalCases: "150000");

        testEntityManager.persist(c1);
        testEntityManager.persist(c2);
        testEntityManager.persist(c3);
        testEntityManager.flush();

        ArrayList<Country> countries = countryRepository.findAll();

        assertEquals("expected: 3", countries.size(), 3);
        assertEquals("expected: c1, c2, c3", countries, List.of(c1, c2, c3));
    }
}
```

Integration Tests

For this type of tests **SpringBoot MockMvc** was used. Here, the endpoints containing data that is used by web application were tested, including Cache endpoint. The tests consist of simply sending a request and waiting for the answer to be ok (status 200).

```
@WebMvcTest(CovidController.class)
public class ControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private CovidService covidService;

    @Test
    public void getCountryByNameTest() throws Exception {
        Country c1 = new Country(name: "c1");
        when(this.covidService.getCountryByName(name: "c1")).thenReturn(c1);
        mvc.perform(get(urlTemplate: "/api/v1/countries/{country}", ...uriVars: "c1").contentType(MediaType.APPLICATION_JSON)
            .andExpect(status().isOk()).andExpect(jsonPath(expression: "$.name", is(value: "c1"))));
    }

    @Test
    void getAllCountriesTest() throws Exception {
        Country c1 = new Country(name: "c1");
        Country c2 = new Country(name: "c2");
        ArrayList<String> countries = new ArrayList<>();
        countries.add(c1.getName());
        countries.add(c2.getName());

        when(this.covidService.getCountries()).thenReturn(countries);
        mvc.perform(get(urlTemplate: "/api/v1/countries").contentType(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath(expression: "$[0]", is(value: "c1"))).andExpect(jsonPath(expression: "$[1]", is(value: "c2"))));
    }
}
```

```
@Test
void getCacheDetailsTest() throws Exception {
    HashMap<String, Integer> cacheMap = new HashMap<>();
    cacheMap.put("hits", 2);
    cacheMap.put("misses", 3);
    cacheMap.put("requests", 5);

    given(covidService.getCacheDetails()).willReturn(cacheMap);
    mvc.perform(get(urlTemplate: "/api/v1/cache").contentType(MediaType.APPLICATION_JSON))
        .andExpect(jsonPath(expression: "$.hits", is(value: 2)))
        .andExpect(jsonPath(expression: "$.misses", is(value: 3)))
        .andExpect(jsonPath(expression: "$.requests", is(value: 5)));
}
```

3.3 Functional testing

For functional testing was used the **Selenium Webdriver**, as was done in classes. First the **covid.feature** file was created, and then the steps described there were implemented.

```
Feature: Covid
  Scenario: Search for covid data in Spain
    When I navigate to 'http://localhost:8080/'
    And I search for 'Spain' on the list bar
    And click on the search button
    Then Covid Data is presented at 'Select the country you want to search' section

  Scenario: Search for covid data for a country not available
    When I navigate to 'http://localhost:8080/'
    And I search for 'X', that is not available, on the search bar
    And click on the search button
    Then I can see that nothing is shown at 'Write the country you want to search' section
```

```
public class FrontendSteps {
    private WebDriver driver;

    @When("I navigate to {string}")
    public void iNavigateTo(String url) {
        driver = WebDriverManager.chromedriver().create();
        driver.get(url);
    }

    @And("I search for {string} on the list bar")
    public void checkCovidDataOnListBar(String local) {
        driver.findElement(By.className(className: "countriesSelect")).sendKeys(local);
    }

    @And("I search for {string}, that is not available, on the search bar")
    public void checkCovidDataOnSearchBar(String local) {
        driver.findElement(By.id(id: "country-box")).sendKeys(local);
    }

    @And("click on the search button")
    public void searchForCovidData() {
        driver.findElement(By.className(className: "search-button")).click();
    }

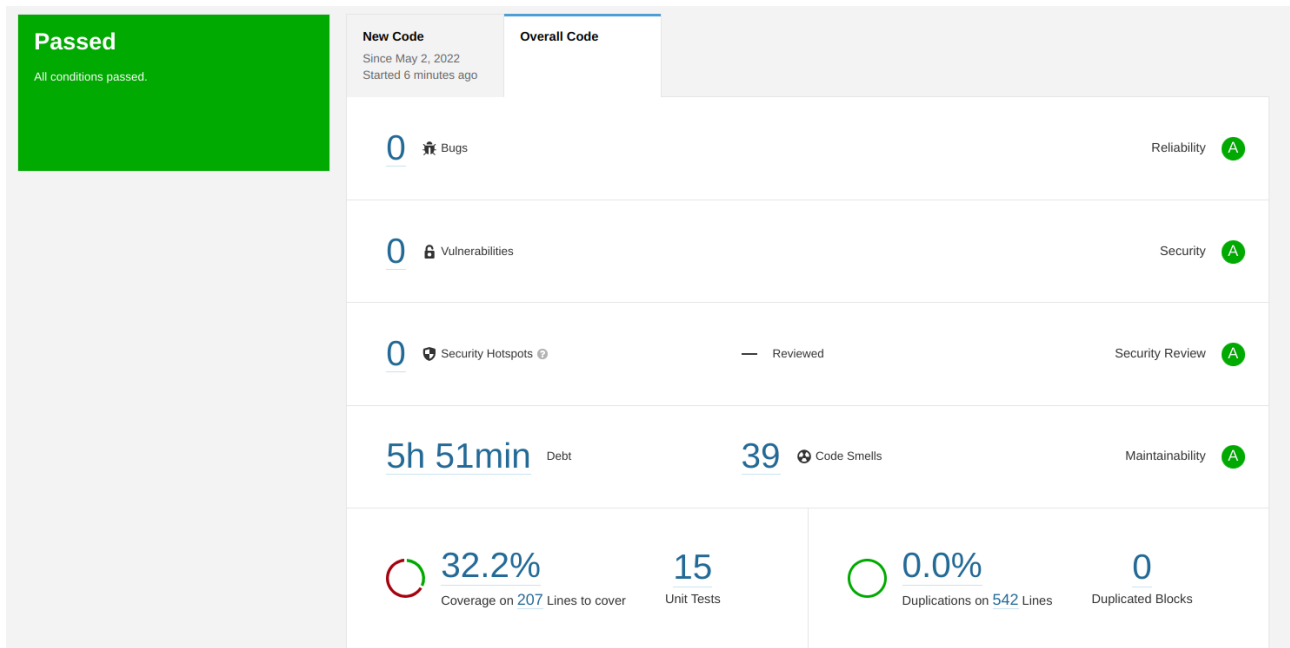
    @Then("Covid Data is presented at {string} section")
    public void seeCovidInformation(String results) {
        assertThat(driver.findElement(By.className(className: "h2-title")).getText(), containsString(results));
    }

    @Then("I can see that nothing is shown at {string} section")
    public void seeThatNothingIsShown(String result) {
        assertThat(driver.findElement(By.id(id: "search-country")).getText(), containsString(result));
    }

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

3.4 Code quality analysis

For code quality analysis was used **SonarQube**, as taught in class. In the image below it is possible to see the results.



As can be seen, a total of 15 tests were performed. There are 39 code smells and it takes 5 hours and 51 minutes to solve them. No major vulnerabilities or bugs were found.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/pedromonteiro01/CovidInfo
Video demo	https://github.com/pedromonteiro01/CovidInfo/blob/main/demo.mkv

Reference materials

- External API:
 - <https://rapidapi.com/api-sports/api/covid-193/>
- Maven Dependencies:
 - <https://mvnrepository.com/>