

HW1: Mid-term assignment report

Ricardo Manuel Batista Rodriguez [98388], v2022-04-28

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

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 goal of the project is to develop a web application capable of presenting Covid-19 statistics from a specific location. The obtained result is a single page application that allows users to search statistics from any country of the world, with the addition of choosing the time interval of those statistics: today, last week, last month and last year.

1.2 Current limitations

Since the main focus of the project is to assure code quality with tests and time is limited, there are some features I was interested in implementing and there are many limitations I

faced during the development of the project. The external API, where the statistics are retrieved, is missing data on some countries and it's notably slow, since thousands of statistics are processed in order to present the covid-19 history between the selected time.

There were some unimplemented features I expected to have, although they are extra, and some others I would like to try out:

- Second external API - Since we can't trust that every external service performs the way we expect, having a second external API to retrieve data if the first one fails (or misses some important data in some countries) is a good expected feature;
- Autocomplete search - To help the user find the country he wants to search faster;
- Other functional features - In the future, I'm expecting to improve this project with new features such as presenting Covid-19 news about the selected country, presenting the current health policies in each country and many others...

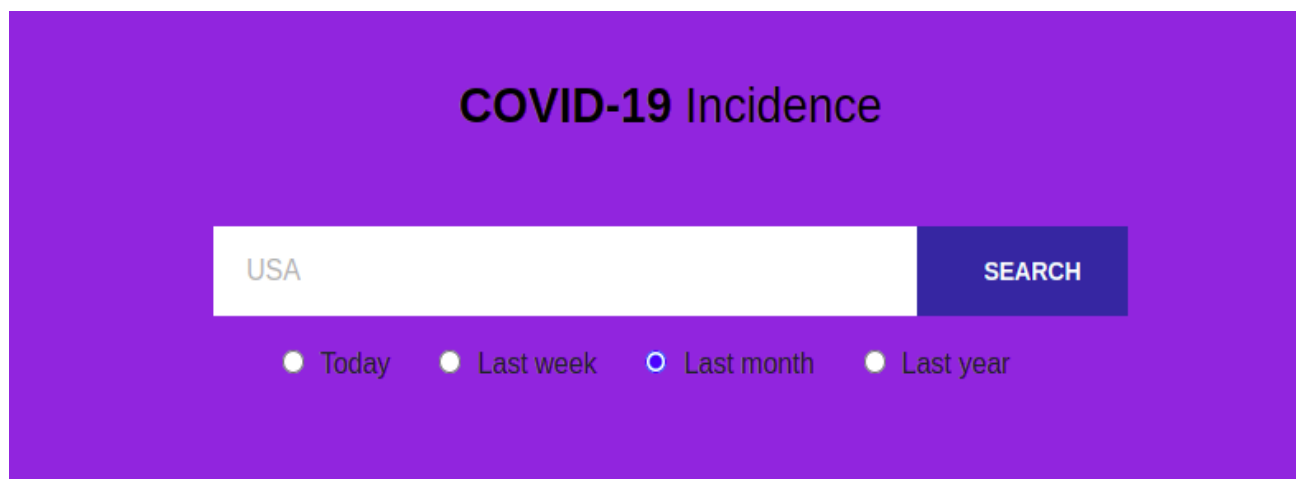
2 Product specification

2.1 Functional scope and supported interactions

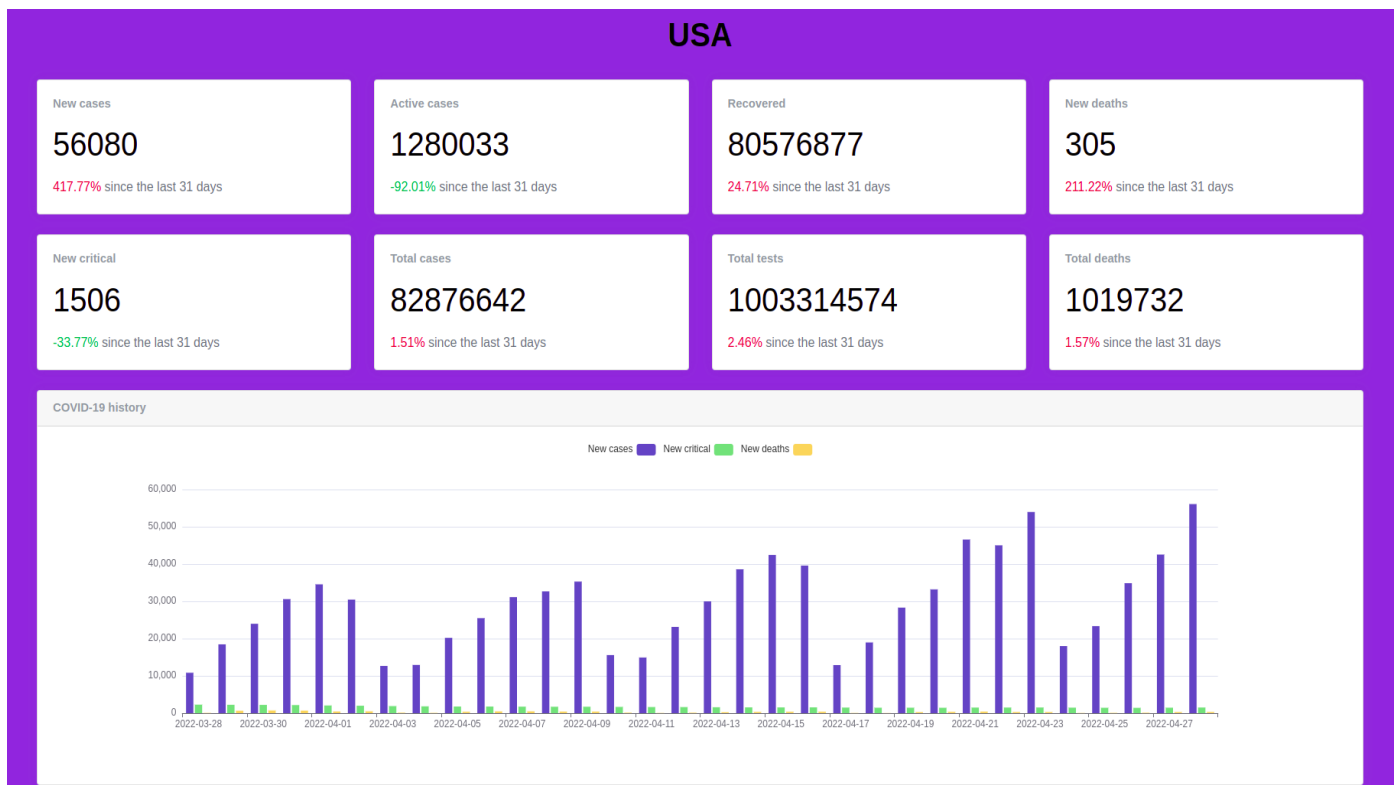
The web application, as referred previously, is a single page application that presents a simple input field for entering the country and some filtering options to search for today, last week, last month or last year Covid-19 statistics. When a user makes a search request, a dashboard will appear with important information about cases, tests, recoveries and deaths.

Each statistic (new cases, recovered...) also presents its growth rate in comparison to the first one. For example, the USA had a 417.77% growth rate for the last month since they had approximately 10000 cases and now they have 56080.

Besides that, a graph will be presented with the Covid-19 history during the selected time interval, also allowing to only see new cases, new critical cases and new deaths. The cache results are shown.



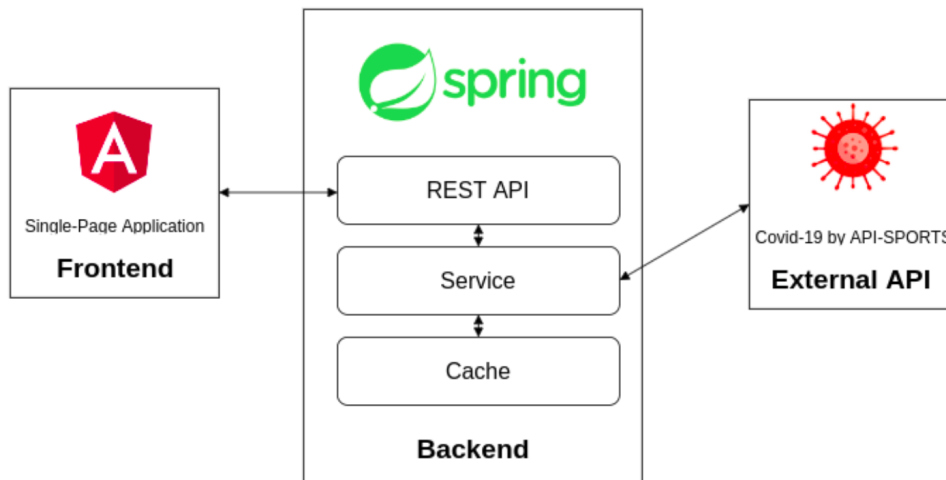
The screenshot shows a web application titled "COVID-19 Incidence" on a purple background. Below the title is a search bar with the text "USA" entered. To the right of the search bar is a dark blue button labeled "SEARCH". Below the search bar are four radio button options: "Today", "Last week", "Last month", and "Last year". The "Last month" option is selected, indicated by a blue outline around its radio button.



At the end of the page, the cache results are presented to the user. When he makes the same request in a short period of time, the external API is not invoked, since that data is stored in the cache and the response will be way faster when retrieved from the cache. However, since the external API provides thousands of statistics of the covid-19 history of a country and the backend module of the project is in charge of cutting that data to the selected time interval, an external request to the API doesn't need to be done when the user searches for the last week statistics and he previously searched for the last month, for example. This happens because last week statistics are contained in the last month ones.

Time	Country	Fetch Days	Cache Status
2022-04-28T16:15:52.072+00:00	usa	31	MISS
2022-04-28T16:15:56.504+00:00	usa	365	MISS
2022-04-28T16:16:05.602+00:00	france	7	HIT

2.2 System architecture

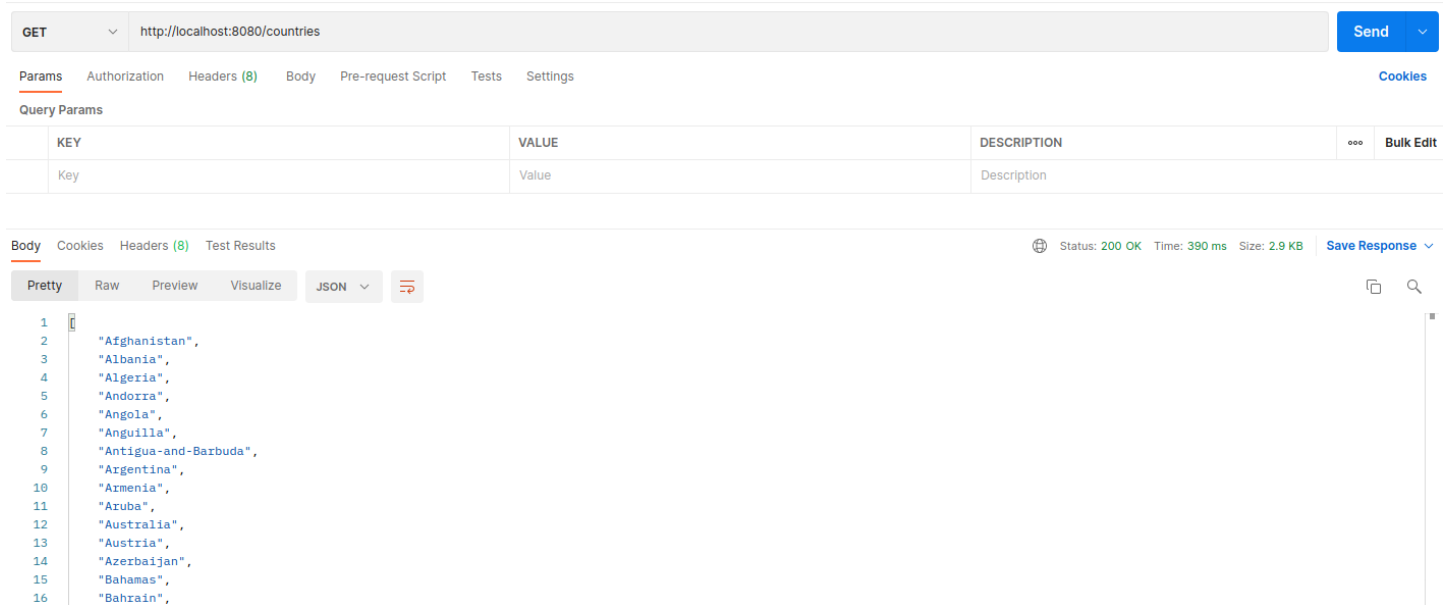


2.3 API for developers

Base URL: localhost:8080

TYPE	URL	DESCRIPTION	PARAMETERS
GET	/countries/	get the names of all the countries of the world	(none)
GET	/interval_history/	gets the covid-19 statistic history of a country between an initial and end interval	country - string initial - date end - date
GET	/cache/	gets all the stored cache	(none)

- `/countries/`



The screenshot shows a REST client interface with a GET request to `http://localhost:8080/countries`. The response is a JSON array of country names, displayed in a "Pretty" format. The status is 200 OK, with a response time of 390 ms and a size of 2.9 KB.

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body (JSON)

```
1  [
2    "Afghanistan",
3    "Albania",
4    "Algeria",
5    "Andorra",
6    "Angola",
7    "Anguilla",
8    "Antigua-and-Barbuda",
9    "Argentina",
10   "Armenia",
11   "Aruba",
12   "Australia",
13   "Austria",
14   "Azerbaijan",
15   "Bahamas",
16   "Bahrain",
```

- `/interval_history?country={string}&initial={date}&end={date}`

GET Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

<input checked="" type="checkbox"/> country	france	
<input checked="" type="checkbox"/> initial	22-04-2022	
<input checked="" type="checkbox"/> end	29-04-2022	
Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 813 ms Size: 3.09 KB Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "id": null,
3    "createdAt": "2022-04-29T15:27:14.996+00:00",
4    "country": "france",
5    "fetchDays": 7,
6    "endDate": "2022-04-22",
7    "cacheStatus": "MISS",
8    "statistics": [
9      {
10       "id": null,
11       "country": "France",
12       "continent": "Europe",
13       "population": 65536206,
14       "newCases": 59760,
15       "recovered": 26552452,
16       "totalCases": 28542884,
17       "newCritical": 1677,
18       "active": 1844721,
19       "casesPerMillion": 435528.0,
20       "totalTests": 266484045,
21       "testsPerMillion": 4066211.0,
22       "newDeaths": 132,
23       "totalDeaths": 145711,
24       "deathsPerMillion": 2223.0,
25       "time": "2022-04-29",
26       "request": null
27     }
28   ],
29 }
```

- /cache/

GET Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results Status: 200 OK Time: 16 ms Size: 25.27 KB Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "id": null,
3    "createdAt": "2022-04-29T15:28:35.132+00:00",
4    "country": "france",
5    "fetchDays": 31,
6    "endDate": "2022-03-29",
7    "cacheStatus": "MISS",
8    "statistics": [
9      {
10       "id": null,
11       "country": "France",
12       "continent": "Europe",
13       "population": 65536206,
14       "newCases": 59760,
15       "recovered": 26552452,
16       "totalCases": 28542884,
17       "newCritical": 1677,
18       "active": 1844721,
19       "casesPerMillion": 435528.0,
20       "totalTests": 266484045,
21       "testsPerMillion": 4066211.0,
22       "newDeaths": 132,
23       "totalDeaths": 145711,
24       "deathsPerMillion": 2223.0,
25       "time": "2022-04-29",
26       "request": null
27     }
28   ],
29 }
```

3 Quality assurance

3.1 Overall strategy for testing

Initially, I was thinking of using the test-driven development (TDD) approach, so that the software requirements are fulfilled while developing the application. However, since I wanted to have the frontend ready with all the functionalities working, I started developing both the frontend and the backend before the tests.

For this project, I would prefer using the TDD approach instead of the BDD one, since I only thought of the web application functionalities while developing the project and bugs would be detected more efficiently. If the web application requirements were well defined, I would try to use both approaches as it would prevent both bugs and unnecessary behaviors.

JUnit5 was used for unit testing, following Mockito and SpringBoot MockMVC for service and integration tests, respectively. For end-to-end testing, I used Selenium with Cucumber to test the frontend of the application and to guarantee the feature requirements.

3.2 Unit and integration testing

Unit Tests

For unit testing, I decided to create tests for the main model classes of my project - **Request**, **Statistic** - alongside with the **Cache**.

For the Request and Statistic class, the tests were meant to assure the automatic generated getter methods work correctly, as we can see:

```

13 public class RequestTest {
14
15     @Test
16     void getRequestObject() {
17
18         Date created_at = new Date();
19         LocalDate endDate = created_at.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
20
21         ArrayList<Statistic> statistics = new ArrayList<Statistic>() { {add(new Statistic());} };
22         Request request = new Request(id: 1L, created_at, country: "Portugal", fetchDays: 365, endDate, CacheStatus.HIT, statistics);
23
24         assertEquals(expected: 1L, request.getId());
25         assertEquals(expected: "Portugal", request.getCountry());
26         assertEquals(created_at, request.getCreatedAt());
27         assertEquals(expected: 365, request.getFetchDays());
28         assertEquals(endDate, request.getEndDate());
29         assertEquals(CacheStatus.HIT, request.getCacheStatus());
30         assertEquals(statistics, request.getStatistics());
31
32     }
33
34 }
35

```

```

9 public class StatisticsTest {
10
11     @Test
12     void getStatisticObject() {
13
14         Long id = 1L;
15         String country = "Portugal";
16         String continent = "Europe";
17         Integer population = 10000000;
18         Integer newCases = 5000;
19         Integer recovered = 4000;
20         Integer totalCases = 3242580;
21         Integer newCritical = 10;
22         Integer active = 30000;
23         Double casesPerMillion = 323258.0;
24         Integer totalTests = 20000000;
25         Double testsPerMillion = 200000.0;
26         Integer newDeaths = 40;
27         Integer totalDeaths = 40000;
28         Double deathsPerMillion = 310.2;
29         LocalDate time = LocalDate.of(2022, 04, 22);
30         Request request = new Request();
31
32         Statistic statistic = new Statistic(id, country, continent, population, newCases, recovered, totalCases, newCritical, active, casesPerMillion, totalTests, t
33         statistic.setRequest(request);
34
35         assertEquals(id, statistic.getId());
36         assertEquals(country, statistic.getCountry());
37         assertEquals(continent, statistic.getContinent());
38         assertEquals(population, statistic.getPopulation());
39         assertEquals(newCases, statistic.getNewCases());
40         assertEquals(recovered, statistic.getRecovered());
41         assertEquals(totalCases, statistic.getTotalCases());
42         assertEquals(newCritical, statistic.getNewCritical());
43         assertEquals(active, statistic.getActive());
44         assertEquals(casesPerMillion, statistic.getCasesPerMillion());
45         assertEquals(totalTests, statistic.getTotalTests());
46         assertEquals(testsPerMillion, statistic.getTestsPerMillion());
47         assertEquals(newDeaths, statistic.getNewDeaths());
48         assertEquals(totalDeaths, statistic.getTotalDeaths());
49         assertEquals(deathsPerMillion, statistic.getDeathsPerMillion());
50         assertEquals(time, statistic.getTime());
51         assertEquals(request, statistic.getRequest());
52
53     }
54
55 }

```

For the **Cache**, I focused on testing many important methods like adding a request to the cache and checking if the cache has updated, generating a key of a request that already has been done to see if it persists in the cache, checking if the request was (HIT) or not (MISS) made before, checking if the request resides more than the specified time-to-live (which means it has expired) and many other methods:


```

17 public class CacheTest {
18
19     private Cache cache;    //60 sec default TTL
20     private Request request;
21     private Long id;
22     private Date created_at;
23     private String country;
24     private Integer fetchDays;
25     private LocalDate startDate;
26     private LocalDate endDate;
27     private List<Statistic> statistics;
28
29     @DisplayName("Set up cache")
30     @BeforeEach
31     void setUp() throws Exception {
32         this.cache = new Cache();
33         this.request = new Request();
34         this.id = 1L;
35         this.request.setId(id);
36         this.created_at = new Date();
37         this.request.setCreatedAt(created_at);
38         this.country = "Portugal";
39         this.request.setCountry(country);
40         this.fetchDays = 7;
41         this.request.setFetchDays(fetchDays);
42         this.endDate = this.created_at.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();
43         this.request.setEndDate(endDate);
44         this.statistics = new ArrayList<>();
45         this.request.setStatistics(statistics);
46         this.startDate = endDate.minusDays(this.fetchDays);
47     }
48
49     @DisplayName("Add request to cache")
50     @Test
51     void testStoreRequestStatistics() {
52         assertEquals(expected: 0, this.cache.getCache().size());
53         String key = this.cache.generateKey(this.country, this.endDate, this.fetchDays);
54         this.cache.storeRequestStatistics(key, this.request);
55         assertEquals(expected: 1, this.cache.getCache().size());
56     }
57
58     @DisplayName("Find by key")
59     @Test
60     void testFindByKey() {
61         String key = this.cache.generateKey(this.country, this.endDate, this.fetchDays);
62         this.cache.storeRequestStatistics(key, this.request);
63
64         Request actualRequest = this.cache.findByKey(key);
65         assertEquals(this.request, actualRequest);

```

Integration Tests

For integration testing, as said before, I used both Mockito and SpringBoot MockMVC. To test the controller, I have three different tests:

- CovidControllerTest - uses Spring MVC components (@WebMvcTest annotation), encapsulating all the needed application beans for testing, and Mockito;
- CovidControllerTestIT - it's a SpringBoot test (@SpringBootTest annotation), loading the SpringApplicationContext, and AssertJ.
- CovidControllerTemplateIT - also uses the @SpringBootTest annotation, with the TestRestTemplate class that tests the database layer (cache in this case).

```

35 @WebMvcTest(CovidController.class)
36 class CovidControllerTest {
37
38     @Autowired
39     private MockMvc mvc;
40
41     @MockBean
42     private CovidService covidService;
43
44     @MockBean
45     private CacheService cacheService;
46
47     @Test
48     void getAllCountries() throws Exception {
49
50         List<String> countries = new ArrayList<>();
51         countries.add("Portugal");
52         countries.add("France");
53         countries.add("Kiribati");
54
55         when(covidService.getAllCountries()).thenReturn(countries);
56
57         mvc.perform(get(urlTemplate: "/countries").contentType(MediaType.APPLICATION_JSON).content(JsonUtils.toJson(countries)))
58             .andExpect(status().isOk())
59             .andExpect(jsonPath(expression: "$", hasSize(equalTo(operand: 3))))
60             .andExpect(jsonPath(expression: "$[0]", is(countries.get(0))))
61             .andExpect(jsonPath(expression: "$[1]", is(countries.get(1))))
62             .andExpect(jsonPath(expression: "$[2]", is(countries.get(2))));
63
64         verify(covidService, times(wantedNumberOfInvocations: 1)).getAllCountries();
65     }
66
67     @Test
68     void getIntervalHistory() throws Exception {
69
70         SimpleDateFormat format = new SimpleDateFormat("dd-MM-yyyy");
71         Date end = format.parse("22-04-2022");
72         List<Statistic> statistics = new ArrayList<>();
73
74         Long id = 1L;
75         String country = "Portugal";
76         String continent = "Europe";
77         Integer population = 10000000;
78         Integer newCases = 5000;
79         Integer recovered = 4000;
80         Integer totalCases = 3242580;

```

```

22 @SpringBootTest
23 @AutoConfigureMockMvc
24 @AutoConfigureTestDatabase
25 public class CovidControllerTestIT {
26
27     @Autowired
28     private MockMvc mvc;
29
30     @Autowired
31     private Cache cache;
32
33     @AfterEach
34     public void resetDb() {
35         cache.clearCache();
36     }
37
38     @Test
39     void whenValidRequest_thenCreateCacheMiss() throws Exception {
40
41         assertThat(cache.getCache().size()).isEqualTo(expected: 0);
42
43         mvc.perform(get(urlTemplate: "/interval_history?country=usa&initial=23-01-2021&end=23-01-2022").contentType(MediaType.APPLICATION_JSON));
44
45         List<Request> found = cache.getCache();
46         assertThat(found).extracting(Request::getCountry).containsOnly(...values: "usa");
47         assertThat(found).extracting(Request::getFetchDays).containsOnly(...values: 365);
48         assertThat(found).extracting(Request::getCacheStatus).containsOnly(CacheStatus.MISS);
49         assertThat(found).extracting(Request::getEndDate).containsOnly(LocalDate.of(2021, 01, 23));
50         assertThat(found).extracting(Request::getCreatedAt).isNotNull();
51         assertThat(found).extracting(Request::getStatistics).isNotNull();
52
53         assertThat(cache.getCache().size()).isEqualTo(expected: 1);
54     }
55
56     @Test
57     void whenDoubleRequest_thenCreateCacheHit() throws Exception {
58
59         assertThat(cache.getCache().size()).isEqualTo(expected: 0);
60
61         mvc.perform(get(urlTemplate: "/interval_history?country=usa&initial=23-01-2021&end=23-01-2022").contentType(MediaType.APPLICATION_JSON));
62
63         List<Request> found = cache.getCache();
64         assertThat(found).extracting(Request::getCountry).containsOnly(...values: "usa");
65         assertThat(found).extracting(Request::getFetchDays).containsOnly(...values: 365);
66         assertThat(found).extracting(Request::getCacheStatus).containsOnly(CacheStatus.MISS);
67         assertThat(found).extracting(Request::getEndDate).containsOnly(LocalDate.of(2021, 01, 23));
68     }

```

```

23 @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
24 @AutoConfigureTestDatabase
25 class CovidControllerTemplateIT {
26
27     @LocalServerPort
28     int randomServerPort;
29
30     @Autowired
31     private TestRestTemplate restTemplate;
32
33     @Autowired
34     private Cache cache;
35
36     @AfterEach
37     public void resetDb() {
38         cache.clearCache();
39     }
40
41     @Test
42     void whenValidRequest_thenCreateCacheMiss() {
43
44         restTemplate.exchange(url: "/interval_history?country=usa&initial=23-01-2021&end=23-01-2022", HttpMethod.GET, requestEntity: null, new ParameterizedTypeReference<List<Request>>());
45
46         List<Request> found = cache.getCache();
47         assertThat(found).extracting(Request::getCountry).containsOnly(...values: "usa");
48         assertThat(found).extracting(Request::getFetchDays).containsOnly(...values: 365);
49         assertThat(found).extracting(Request::getCacheStatus).containsOnly(CacheStatus.MISS);
50         assertThat(found).extracting(Request::getEndDate).containsOnly(LocalDate.of(2021, 01, 23));
51         assertThat(found).extracting(Request::getCreatedAt).isNotNull();
52         assertThat(found).extracting(Request::getStatistics).isNotNull();
53
54         assertThat(cache.getCache().size()).isEqualTo(expected: 1);
55     }
56
57     @Test
58     void whenDoubleRequest_thenCreateCacheHitAndOthers() {
59
60         assertThat(cache.getCache().size()).isEqualTo(expected: 0);
61
62         restTemplate.exchange(url: "/interval_history?country=usa&initial=23-01-2021&end=23-01-2022", HttpMethod.GET, requestEntity: null, new ParameterizedTypeReference<List<Request>>());
63
64         List<Request> found = cache.getCache();
65         assertThat(found).extracting(Request::getCountry).containsOnly(...values: "usa");
66         assertThat(found).extracting(Request::getFetchDays).containsOnly(...values: 365);
67         assertThat(found).extracting(Request::getCacheStatus).containsOnly(CacheStatus.MISS);
68         assertThat(found).extracting(Request::getEndDate).containsOnly(LocalDate.of(2021, 01, 23));
69         assertThat(found).extracting(Request::getCreatedAt).isNotNull();
70         assertThat(found).extracting(Request::getStatistics).isNotNull();
71     }

```

Every endpoint was tested, with different methods for each of its possibilities (example: one for cache hit and other for cache miss, etc...).

For the service layer, I tested both the **CacheService** and **CovidService**, which are invoked by the controller, with Mockito.

```
33 @ExtendWith(MockitoExtension.class)
34 public class CovidServiceTest {
35
36     @Mock
37     private Cache cache;
38
39     @Mock
40     private RapidApiResolver rapidApiResolver;
41
42     @InjectMocks
43     private CovidService covidService;
44
45     @Test
46     void testGetAllCountries() throws URISyntaxException, IOException, ParseException {
47
48         List<String> countries = new ArrayList<>();
49         countries.add("Portugal");
50         countries.add("France");
51         countries.add("UK");
52
53         Mockito.when(covidService.getAllCountries()).thenReturn(countries);
54
55         List<String> actualCountries = covidService.getAllCountries();
56
57         Mockito.verify(rapidApiResolver, times(wantedNumberOfInvocations: 1)).getCountries();
58         assertThat(actualCountries).isEqualTo(countries);
59
60     }
61
62     @Test
63     void testGetCountryIntervalHistoryHit() throws URISyntaxException, IOException, ParseException, java.text.ParseException {
64
65         SimpleDateFormat format = new SimpleDateFormat("dd-MM-yyyy");
66         Date initial = format.parse("22-04-2021");
67         Date end = format.parse("22-04-2022");
68
69         List<Statistic> statistics = new ArrayList<>();
70         Long id = 1L;
71         String country = "Portugal";
72         String continent = "Europe";
73     }
```

```

27 @ExtendWith(MockitoExtension.class)
28 public class CacheServiceTest {
29
30     @Mock
31     private Cache cache;
32
33     @InjectMocks
34     private CacheService cacheService;
35
36     private Date created_at;
37
38     @BeforeEach
39     public void setUp() throws URISyntaxException, IOException, ParseException {
40
41         List<Request> cache = new ArrayList<>();
42         this.created_at = new Date();
43         cache.add(new Request(id: 1L, created_at, country: "Portugal", fetchDays: 365, LocalDate.of(2022, 4, 22), CacheStatus.HIT, new ArrayList<Statistic>()));
44         cache.add(new Request(id: 2L, created_at, country: "Spain", fetchDays: 7, LocalDate.of(2022, 4, 22), CacheStatus.MISS, new ArrayList<Statistic>()));
45
46         // Expectations
47         Mockito.when(cacheService.getCache()).thenReturn(cache);
48     }
49
50
51     @Test
52     public void testCache() throws URISyntaxException, IOException, ParseException {
53
54         List<Request> cache = cacheService.getCache();
55         assertEquals(expected: 2, cache.size());
56         assertEquals(expected: 1L, cache.get(0).getId());
57         assertEquals(this.created_at, cache.get(0).getCreatedAt());
58         assertEquals(expected: "Portugal", cache.get(0).getCountry());
59         assertEquals(LocalDate.of(2022, 4, 22), cache.get(0).getEndDate());
60         assertEquals(CacheStatus.HIT, cache.get(0).getCacheStatus());
61         assertEquals(expected: 365, cache.get(0).getFetchDays());
62         assertEquals(new ArrayList<Statistic>(), cache.get(0).getStatistics());
63         assertEquals(expected: 2L, cache.get(1).getId());
64         assertEquals(this.created_at, cache.get(1).getCreatedAt());
65         assertEquals(expected: "Spain", cache.get(1).getCountry());

```

3.3 Functional testing

For functional testing, I used Cucumber for requirement assurance with Selenium to ensure everything is well presented in the frontend module. To make the code cleaner and reusable, I used the Page Object pattern in the **Hw1Test** class, that tests the frontend with the Cucumber selected requirements. Since the web application only allows users to check statistics from one location but from different time intervals, requirements are really simple and do not vary much.

```

@FindBy(id="selected_country")
private WebElement selected_country;

@FindBy(css="text-muted")
private WebElement differential;

@BeforeEach
public void setUp() {
    driver = new ChromeDriver();
}

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

@When("I want to access {string}")
public void accessURL(String URL) {
    driver.get(URL);
    driver.manage().window().setSize(new Dimension(width: 1846, height: 1053));
}

@Then("page title should be {string}")
public void checkPageTitle(String title) {
    assertThat(driver.getTitle()).isEqualTo(title);
}

@And("I want to search statistics from {string}")
public void searchCountry(String strCountry) {
    country.click();
    country.sendKeys(strCountry);
}

@And("And I search for the {string}")
public void lastDays(String fetchDays) {

```

Feature: Search Covid-19 statistics**Scenario: Search statistics of USA for the last year**

```
When I access 'https://blazedemo.com/'
Then page title should be "Covid-19"
And I want to search statistics from "USA"
And I search for the "last year"
Then country title should be "USA"
And statistic differential text should be last "356 days"
```

Scenario: Search statistics of USA for the last month

```
When I access 'https://blazedemo.com/'
Then page title should be "Covid-19"
And I want to search statistics from "USA"
And I search for the "last month"
Then country title should be "USA"
And statistic differential text should be last "30 days"
```

Scenario: Search statistics of USA for the last week

```
When I access 'https://blazedemo.com/'
Then page title should be "Covid-19"
And I want to search statistics from "USA"
And I search for the "last week"
Then country title should be "USA"
And statistic differential text should be last "7 days"
```

Scenario: Search statistics of France for the today

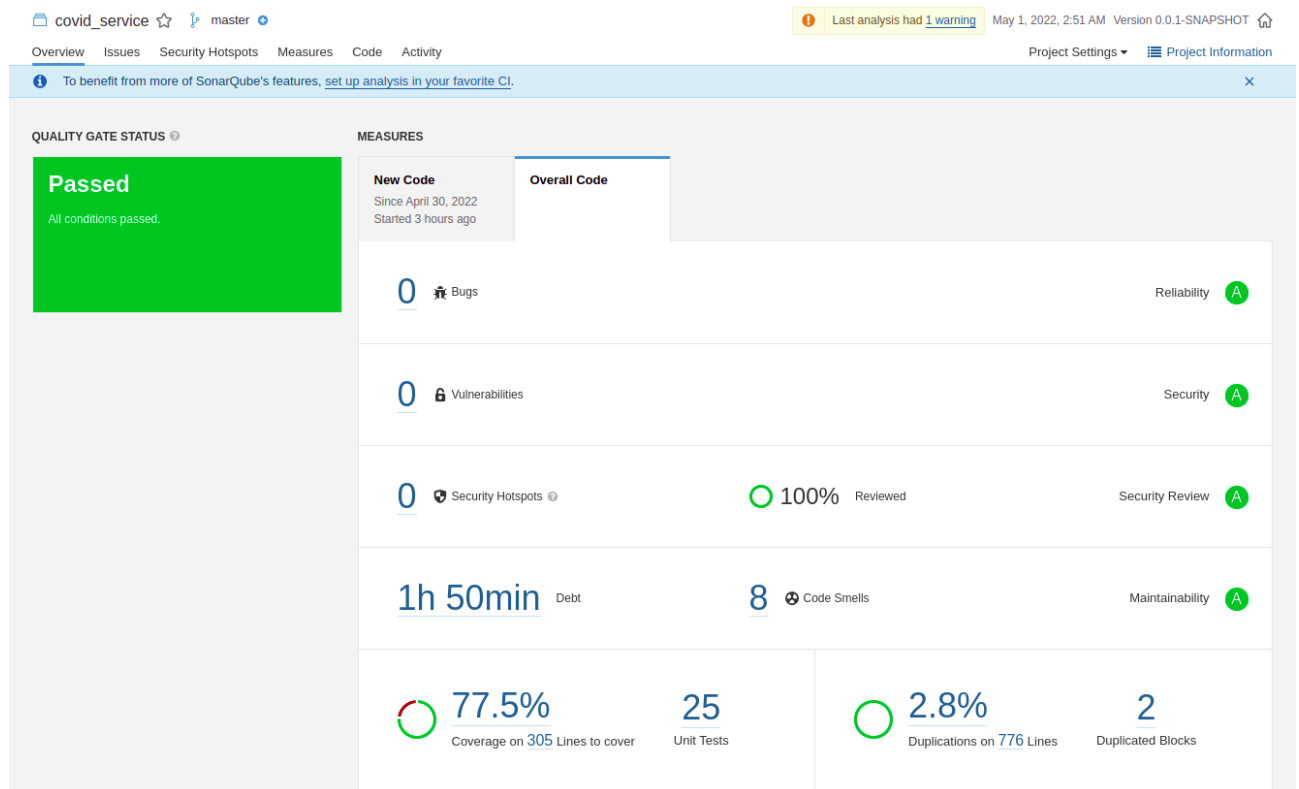
```
When I access 'https://blazedemo.com/'
Then page title should be "Covid-19"
And I want to search statistics from "France"
And I search for the "today"
Then country title should be "France"
And statistic differential text should be last "1 day"
```

3.4 Code quality analysis

For code quality analysis, I used SonarQube. It checks static code to assure its quality, warning the developer for bugs, possible security hotspots, code smells and much more. I find it a very useful tool since it analyzes important metrics such as code duplication and test

coverage, which helped me check what classes, or methods, are not covered or duplicated. While developing the project, I assured no bugs or vulnerabilities are present. The majority of code smells were warning to remove the *public* modifier, to define a constant instead of duplicating a literal (which I didn't fix since it's something related to the external API, but I could use it if it would be updated) and other related to the size of the *Statistic* constructor, which could be solved with the builder pattern design, but it required a lot of change in order to avoid that code smell.

In terms of code coverage, I think I got a good result since many of the uncovered lines are related to automatic setter methods, which are almost unused, and of the *HttpClient* class.



3.5 Continuous integration pipeline

For continuous integration, I made two different pipelines: one to *build* the Maven project and to run the *verify* lifecycle to run all quality assurance tests; another to build the Angular project.


```
.github > workflows > ! maven.yml
1  name: Java CI with Maven
2
3  on:
4    push:
5      branches: [ master ]
6    pull_request:
7      branches: [ master ]
8
9  jobs:
10   build:
11     runs-on: ubuntu-latest
12     name: Running verify Maven lifecycle
13     steps:
14       - uses: actions/checkout@v3
15       - name: Set up JDK 11
16         uses: actions/setup-java@v3
17         with:
18           java-version: '11'
19           distribution: 'temurin'
20           cache: maven
21       - name: Docker compose - build all services
22         run: docker-compose build
23       - name: Docker compose - run 'Angular' service
24         run: docker-compose build angular
25       - name: Verify lifecycle and tests
26         run: mvn -B clean verify
27         working-directory: ./covid_service
28
```

```

.github > workflows > ! angular.yml
1  name: ANGULAR - CI
2
3  on:
4    push:
5      branches: [ master ]
6    pull_request:
7      branches: [ master ]
8
9  jobs:
10   build:
11     runs-on: ubuntu-latest
12     strategy:
13       matrix:
14         node-version: [16.14.0]
15     steps:
16       - uses: actions/checkout@v3
17       - name: Use Node.js ${ matrix.node-version }
18         uses: actions/setup-node@v3
19         with:
20           node-version: ${ matrix.node-version }
21           cache: 'npm'
22           cache-dependency-path: frontend/package-lock.json
23       - run: npm ci
24         working-directory: ./frontend
25       - run: npm run build --if-present
26         working-directory: ./frontend

```

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/ricardombrodriguez/HW1
Video demo	https://youtu.be/7s2TqLgZciw

Reference materials

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

