HW1: Mid-term assignment report

Sophie Jane Pousinho [97814] 2022-04-07

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 application offers information about the COVID-19 incidence in various countries, including the number of cases, deaths and tests.

1.2 Current limitations

A few limitations:

- The user can search for history in a day that has not yet existed, like, for example, 2nd of June of 2022.

- In this case, I decided to only put in the cache the countries, instead of the other information obtained from other requests because they might only happen once whereas the countries are almost always used. But in a large scale of usage, it would be wise to also place the other requests' data in the cache as well.

- Some of the requests are slow to load and there is no loading state for the user.

2 Product specification

2.1 Functional scope and supported interactions

This application is simplistic but could be used by anyone who wanted to have information about the COVID-19 incidence.

When it comes to supported interactions, there are two options, either to check the statistics or the history. Both of these options will include information such as the total of cases, the number of deaths, the active cases, recovered cases, etc. In the case of statistics, searching considering a specific country is optional; in the case of history, searching with a specific country is required but the date is optional.

Main usage scenarios:

```
Scenario: Find statistics of country
       When I navigate to "http://localhost:3000/"
       And I click on "Statistics"
       And I select "barbados" on the country option
       And I click on Search
       Then I should see the results in a table

   Scenario: Find statistics
       When I navigate to "http://localhost:3000/"
       And I click on "Statistics"
       And I click on Search
       Then I should see the results in a table

   Scenario: Find history of country in a certain day
       When I navigate to "http://localhost:3000/"
       And I click on "History"
       And I select "australia" on the country option
       And I select "07-04-2021" on the date option
       And I click on Search
       Then I should see the results in a table


Scenario: Find history of country
       When I navigate to "http://localhost:3000/"
       And I click on "History"
       And I select "albania" on the country option
       And I click on Search
       Then I should see the results in a table
```

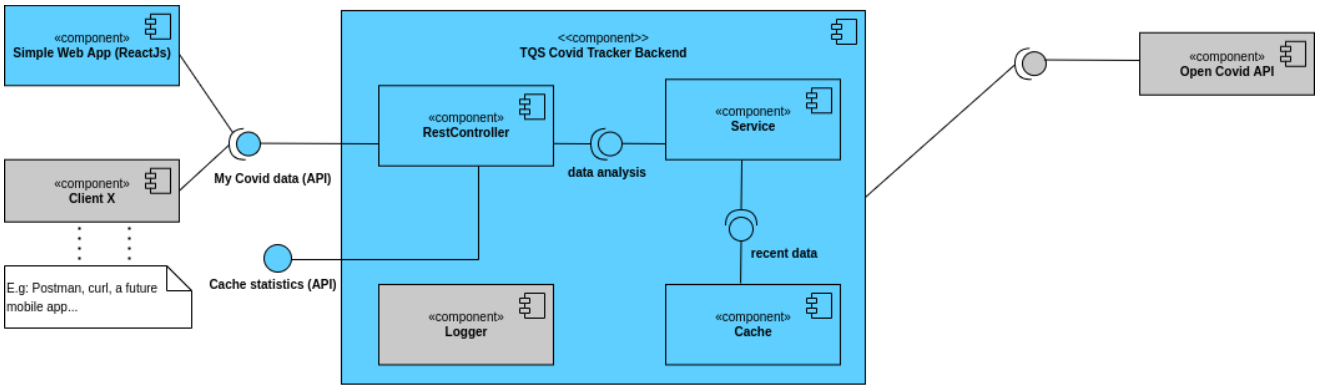
2.2 System architecture

For the system architecture, we can divide it in two parts, the frontend and the backend.

The frontend was developed in ReactJs, since I have worked with it before. I created a simple application with two tabs, one for the statistics and the other for the history, each one with a form for the user to select what he/she intends to check. The results, if they exist, are shown in a table. I also added a button to check the cache usage statistics, meaning, the number of elements it contains, the number of hits and misses, and so on.

The backend was developed in Spring Boot (Maven). Unlike the other apps I have developed before in Spring Boot, this one did not require a database nor entities nor a repository, only a service and a controller. This is because the data is provided by an external API that is called from the service of the application. I also used log4j2 for the logging of the main operations.

The flow looks something like this:



The user only interacts with the Web application. The Web application will then interact with the RestController that will map each endpoint of the application to a method of the Service. The Service will use a Cache when it is dealing with countries since the list of countries will be fetched many times, so it is better to have it more “handy” as the user will need it in the majority of the interactions. The other information and the countries (unless they are no longer in the cache) will be fetched by making HTTP GET requests to the external API through the correct endpoints.

2.3 API for developers

[Base URL = http://localhost:3000/api/]

client Regular user of the COVID-19 incidence API

GET	/countries	get list of countries
GET	/statistics	get general statistics or specific country/region statistics
GET	/history	get country/region general history or specific day of history in a given country/region
GET	/cache	get cache usage statistics

3 Quality Assurance

3.1 Overall strategy for testing

The overall testing strategy was BDD (Behavior Driven Development), since at the beginning I did not have a full grasp of the implementation I was going to make. Therefore, I used Cucumber combined with Selenium IDE for the functional testing.

3.2 Unit and integration testing

The unit tests implemented were for:

- the Cache, to check if it followed the expected behavior [CacheUnitTest.java];

For example,

```java
@Test
@DisplayName("Cache has size 0 on construction")
void cacheIsEmptyTest() {
    assertEquals( expected: 0, cache.getSize());
}

@Test
@DisplayName("Cache has correct TTL on construction")
void cacheHasCorrectTtlTest() {
    assertEquals(TTL*1000, cache.getTtl());
}
```

- the Controller, testing each endpoint, by mocking the Service (using WebMvcTest) [CovidIncidenceController_WithMockServiceTest.java];

For example,

```java
@Test
void givenCountries_whenGetCountries_thenReturnListOfCountries() throws Exception {

    String countries = "{\"get\":\"countries\",\"parameters\":[],\"errors\":[],\"results\":233,\"response\":[\"Afghanistan\",\"Albania\"]}";

    ResponseEntity<String> response = new ResponseEntity<>(countries, HttpStatus.OK);

    when(service.getCountries()).thenReturn(response);
    mvc.perform(
        get( urlTemplate: "/api/countries").contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk());


    verify(service, times( wantedNumberOfInvocations: 1)).getCountries();
}
```

- the Service, testing also each endpoint, by mocking the HTTP Client (using Mockito). Since I was having trouble mocking the native HTTP Client, I created an extra class, CustomHttpClient, that simply contains the method to get the HTTP responses from the external API, and it was much easier to mock [CovidIncidenceSrviceTest.java].

For example,

```java
@Test
void whenGetCountries_returnCountries() throws IOException, URISyntaxException, InterruptedException {

    String countries = "{\"get\":\"countries\",\"parameters\":[],\"errors\":[],\"results\":2,\"response\":[\"Afghanistan\",\"Albania\"]}";

    HttpResponse<String> response = createHttpResponse(countries);

    when(client.getHttpResponse(URI.create(baseURL+"countries"))).thenReturn(response);

    ResponseEntity<String> serviceResponseEntity = service.getCountries();

    assertEquals(response.body(), serviceResponseEntity.getBody());
    assertEquals(response.statusCode(), serviceResponseEntity.getStatusCode().value());

    verify(client, times( wantedNumberOfInvocations: 1)).getHttpResponse(URI.create(baseURL+"countries"));

}
```

I also included one integration test that validates the API from my application [CovidIncidenceAPIIT.java].

For example:

```java
@Test
@Order(1)
void whenGetCountries_thenStatus200() throws JsonMappingException, JsonProcessingException {


    ResponseEntity<String> entity = restTemplate.exchange( url: "/api/countries", HttpMethod.GET, requestEntity: null, new ParameterizedTypeReference<String>() {});

    assertThat(entity.getStatusCodeValue()).isEqualTo(HttpStatus.SC_OK);

    Map<String,Object> map = mapper.readValue(entity.getBody(), valueType: Map.class);


    assertThat(map.get("response")).asList().hasSize( expected: 233).contains( ...values: "Egypt");
}
```

3.3 Functional testing

As I mentioned before, I decided to use Behavior Driven Development. So for the user-facing test I used Selenium IDE together with Cucumber. In Chrome I used Selenium to test the most relevant steps of the application. The Java class generated by Selenium is not available because I did not use it completely. I took some of the steps and used them as part of the implementation of the scenarios of the covidincidence.feature available in the resources.

For example:

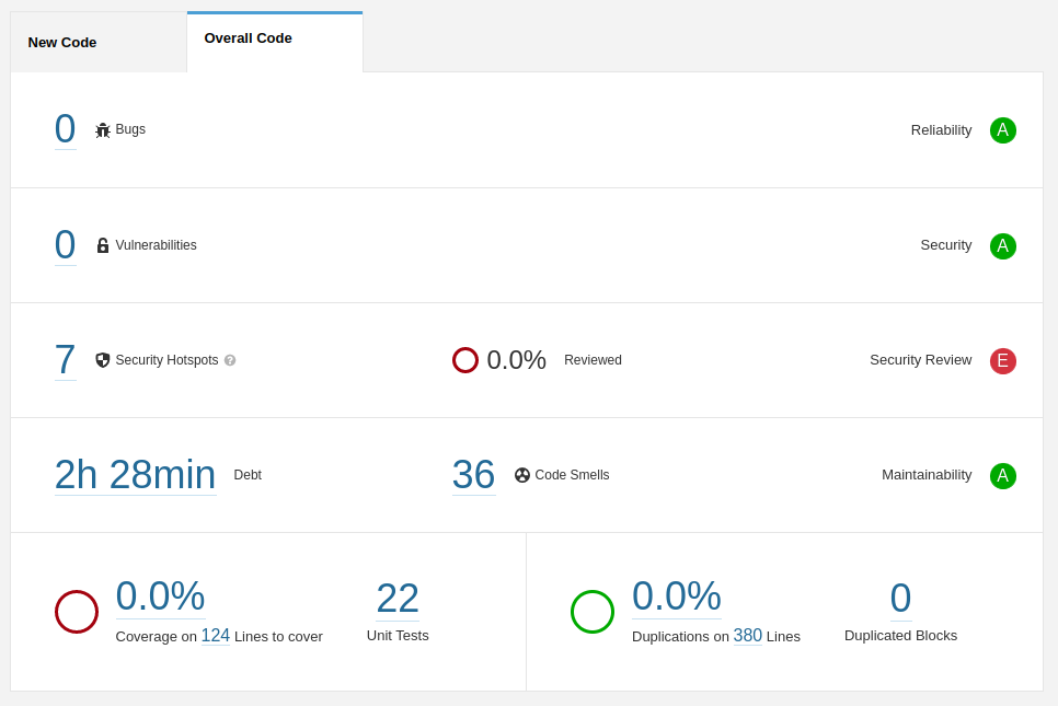
```java
@When("I navigate to {string}")
public void iNavigateTo(String url) {
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();
    driver.get(url);
}

@And("I click on {string}")
public void iPressOptionButton(String option) {
    driver.findElement(By.id(option.toLowerCase())).click();
}

@And("I select {string} on the country option")
public void iSelectCountry(String country) {
    WebElement option = new WebDriverWait(driver, timeOutInSeconds: 3).until(driver -> driver.findElement(By.id(country)));
    option.click();
}
```

3.4 Code quality analysis

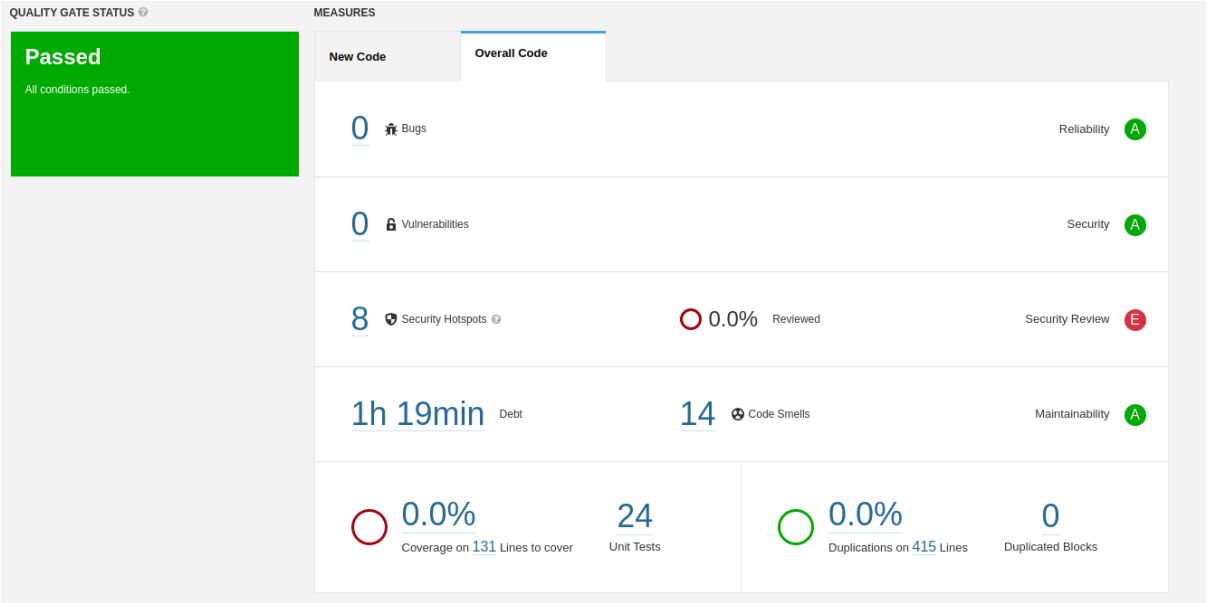
In order to get static code analysis, I used SonarQube.



This is not the first analysis (I forgot to take a screenshot), this is the analysis after fixing the bugs of the first one. It had five bugs, four of them were “Either re-interrupt this method or rethrow the "InterruptedException" that can be caught here", the solution was to add “Thread.currentThread().interrupt();” inside the catch of the Interrupted Exceptions. The other bug was “Add an end condition to this loop”, the solution was to add a break condition on the while true loop on the Cache class.

Most of the code smells were simple things that I just did not think of when coding but were obvious, for example, private instead of public, unnecessary casts, repeated strings, etc. There were other ones like “Make this anonymous inner class a lambda” that I would not have thought were needed but it had a simple resolution.

After adding some new code, I made another analysis.



The code smells included unused imports, a public modifier that should be default, asserts that could be written in a different way, for example,

```
assertThat(params).containsEntry( key: "country",  value: "portugal");
assertThat(params).containsEntry( key: "day",  value: "2020-06-02");
```

instead of

```
assertThat(params.get("country")).isEqualTo("portugal");
assertThat(params.get("day")).isEqualTo("2020-06-02");
```

and so on.

4 References & resources

Resource:	**URL/location:**
Git repository	https://github.com/sophjane/TQS_97814
Video demo	https://github.com/sophjane/TQS_97814/tree/main/HW1/demos