HW1: Mid-term assignment report

Patrícia Matias Dias [98546], v2022-05-02

1 Introduction

1.1 Overview of the work

This report will explain the adopted strategies for the developed application, for the midterm individual project required for TQS. The report includes both the software product features and the adopted quality assurance strategy.

The Covid Data web application displays the COVID-19 metrics in multiple countries around the world. It is possible to observe the number of cases, deaths and tests.

1.2 Current limitations

The application was expected to be able to search for covid metrics in a range of days. This feature was not implemented since the chosen third-party API didn't display such data.

It is also important to mention that heavy requests are slow to load, which should be considered to improve in a future implementation, so that the access to the data is faster and more efficient.

2 Product specification

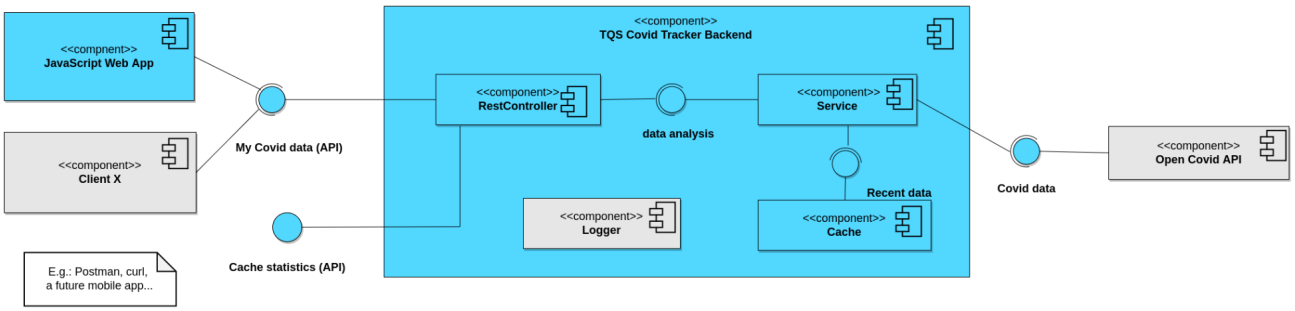
2.1 Functional scope and supported interactions

As mentioned previously, this application offers users COVID-19 related information in a simplistic way, which can be useful for specialists and momentarily curious people. Users can take advantage of the following features:

- Analyze the current status of the spread of the coronavirus in all countries.
    - Filter by country/region to retrieve its current status
- Analyze the entire history of statistics for a country
    - Filter the statistics by day

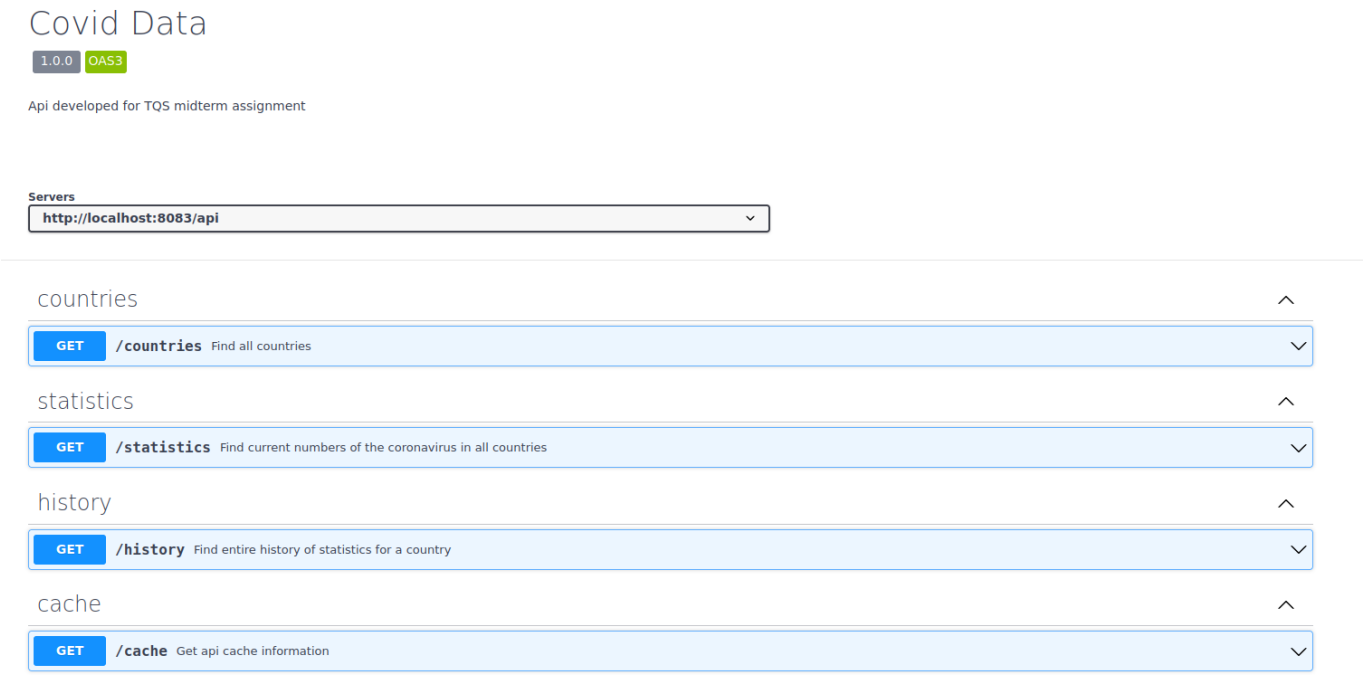
2.2 System architecture

The web application was implemented with HTML/JavaScript and the backend is based on Spring Boot. This system fetches the data from an external API (covid-193.p.rapidapi.com).



Architecture Diagram

2.3 API for developers



3 Quality assurance

3.1 Overall strategy for testing

The Cache for this project was implemented alongside with its respective tests, but the Controller and the Service were only tested after implementation of the application, thus this project followed the Test-Driven Development(TDD) strategy.

After the frontend was developed, Cucumber and Selenium were used to test the application's behavior from the end user's standpoint (Behavior-driven development).

3.2 Unit and integration testing

Unit testing was important to make sure the methods implemented in the different classes worked as expected. Unit tests were developed for:

- Cache, to make sure the cache and its functions would have the expected behavior.

```
@DisplayName("Regularly clean cashe")
@Test
synchronized void cleanCashTest() throws InterruptedException{
    Cache newCache = new Cache(ttl: 1);

    newCache.put(test_uri1, example1);
    newCache.get(test_uri1);

    assertThat(newCache.getSize(), is(value: 1));
    assertThat(newCache.getRequests(), is(value: 1));

    wait(2000);

    assertThat(newCache.getSize(), is(value: 0));
    assertThat(newCache.getRequests(), is(value: 1));

}
```

- Controller (WebMvcTest), to test each endpoint and verify if controller functions display expected outputs (for example, verify number of calls of a Service function called by the controller and expected results from the controller), the service was mocked, since the real service would only slow the test and I could imitate its behavior.

```
@Test
void whenGetAllCountries_thenReturnJsonArray() throws Exception {
    when(service.getAllCountries()).thenReturn(allCountries_dummy);

    mock.perform(MockMvcRequestBuilders.get(urlTemplate: "/api/countries")
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath(expression: "$", hasSize(size: 3)))
        .andExpect(MockMvcResultMatchers.jsonPath(expression: "$[0]", is(value: "Afghanistan")))
        .andExpect(MockMvcResultMatchers.jsonPath(expression: "$[1]", is(value: "Albania")));

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

- Service (Mockito), by mocking both the Client (class that uses HTTP Client), that gets data from the external API, and the Cache.

```
@Test
void whenGetCountriesAndURIinCache_thenReturnCashedValue() {
    URI uri = URI.create(url+"/countries");

    when((mockedCache.get(uri))).thenReturn(allCountries_dummy);

    assertThat(service.getAllCountries(), is(allCountries_dummy));

    verify(mockedCache, times( wantedNumberOfInvocations: 1)).get(uri);
    verify(client, times( wantedNumberOfInvocations: 0)).getFromAPI(uri);
}

@Test
void whenGetCountriesAndURInotInCache_thenReturnApiValue() {
    URI uri = URI.create(url+"/countries");

    when((mockedCache.get(uri))).thenReturn( value: null);
    when((client.getFromAPI(uri))).thenReturn(allCountries_dummy);

    assertThat(service.getAllCountries(), is(allCountries_dummy));

    verify(mockedCache, times( wantedNumberOfInvocations: 1)).get(uri);
    verify(client, times( wantedNumberOfInvocations: 1)).getFromAPI(uri);
}
```

Integration tests were used for:

- Testing the developed API

```
@Test
void whenGetStatisticsByCountry_thenStatus200() {
    ResponseEntity<String> response = restTemplate
            .exchange( url: "/api/statistics?country=portugal", HttpMethod.GET, requestEntity: null, new ParameterizedTypeReference<String>() {
            });
    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
    assertThat(response.getBody()).hasSizeGreaterThan( expected: 1).doesNotContain( ...values: "USA");
    assertThat(response.getBody()).hasSizeGreaterThan( expected: 1).contains( ...values: "Portugal");
}
```

3.3 Functional testing

As previously mentioned, functional testing was implemented using Selenium IDE and Cucumber for behavior-driven development. These tests included 4 different scenarios that summarize the features for the user in this application:

```
Scenario: Search for statistics in all countries
    When I navigate to "http://localhost:8083"
    And I click Check Statistics
    Then I should be able to see a table

Scenario: Search statistics for one country
    When I navigate to "http://localhost:8083"
    And I click Check Statistics
    And I insert country "uk"
    And I click Search
    Then I should be able to see a table with data related to "UK"

Scenario: Search history of statistics for one country
    When I navigate to "http://localhost:8083"
    And I click Check History
    And I insert country "uk"
    And I click Search
    Then I should be able to see a table with data related to "UK"

Scenario: Search history of statistics for one country and a day
    When I navigate to "http://localhost:8083"
    And I click Check History
    And I insert country "uk"
    And I insert date "2022-02-06"
    And I click Search
    Then I should be able to see a table with data related to "UK" from "2022-02-06"
```

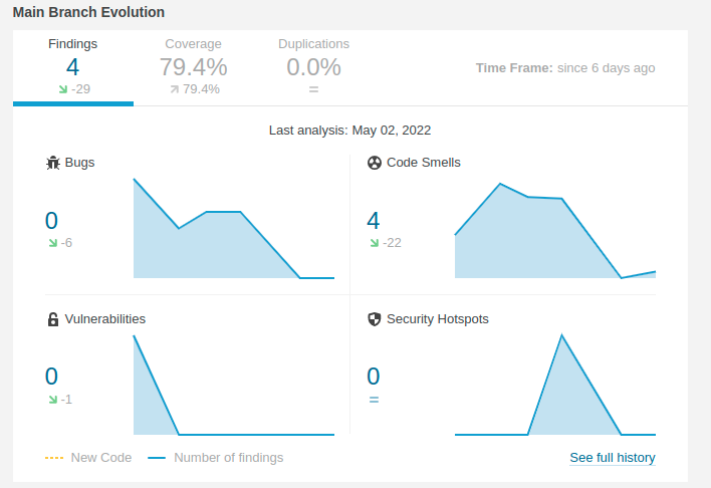
Scenarios of Covid Data (coviddata.feature)

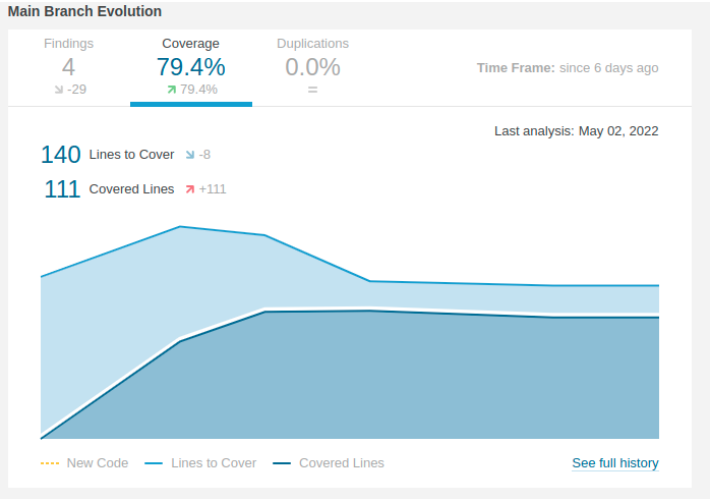
```
@When("I navigate to {string}")
public void goTo(String url) {
    driver = new ChromeDriver();
    driver.get(url);
}
@And("I click Check Statistics")
public void checkStatistics() {
    driver.findElement(By.cssSelector(cssSelector: ".card:nth-child(1) button")).click();

}
@Then("I should be able to see a table")
public void table() {
    assertNotNull(driver.findElement(By.id(id: "continent")));
    assertNotNull(driver.findElement(By.id(id: "country")));
    assertNotNull(driver.findElement(By.id(id: "population")));
    assertNotNull(driver.findElement(By.id(id: "cases")));
    assertNotNull(driver.findElement(By.id(id: "deaths")));
    assertNotNull(driver.findElement(By.id(id: "tests")));
    assertNotNull(driver.findElement(By.id(id: "time")));
    assertNotNull(driver.findElement(By.id(id: "day")));
}
```

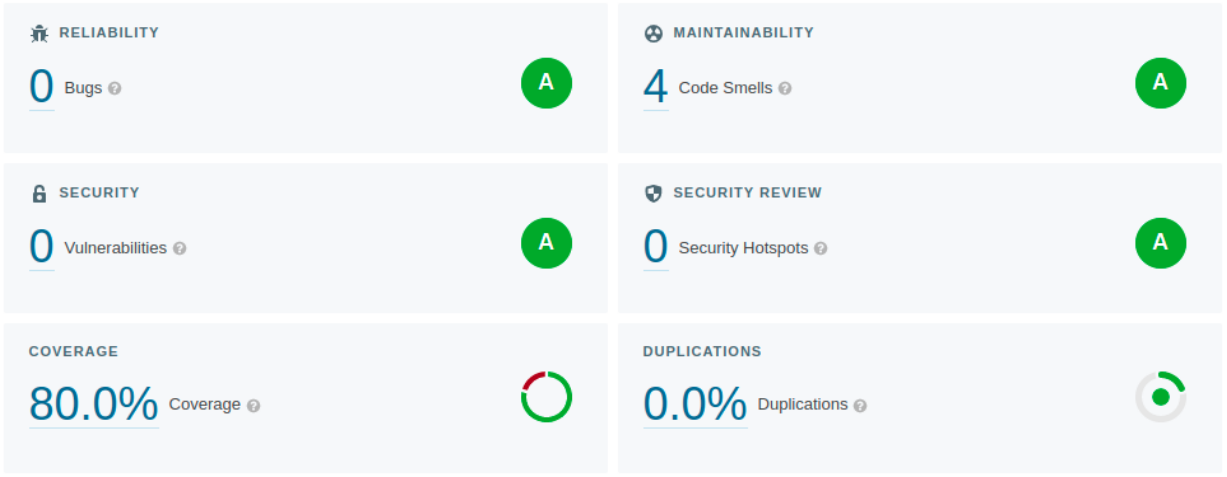
Example of cucumber steps

3.4 Code quality analysis

For the static code analysis I, firstly installed the SonalLint plugin in vs code to help finding code smells, bugs and vulnerabilities faster. I also added the project to SonarQube and continued the code analysis, but, in this platform, I was able to check the coverage of the project and the overall rating of the project static code. The first time I analyzed the code, there were some bugs, vulnerabilities and security hotspots. There were also nearly 50 code smells. The code has since been improved to the following rating:







After these improvements, I learned that test classes should have default visibility (not public), class constants should be declared as static, to reduce the amount of memory required to execute the application and that try catch blocks should rethrow the InterruptedException or call Thread.interrupt() so the information that the thread was interrupted is not lost (I only logged the information).

```java
while(true){
    try {
        Thread.sleep(timeToLive);
    } catch (InterruptedException e) {
        logger.severe("Exception. "+e);
    }
    if (cacheMap.size()>0)
        cleanCashe();
}
```

In the catch block, I only logged the exception, which is a bad practice…

```java
while(true){
    try {
        Thread.sleep(timeToLive);
    } catch (InterruptedException e) {
        logger.severe("Exception. "+e);
        Thread.currentThread().interrupt();
    }
    if (cacheMap.size()>0)
        cleanCashe();
}
```

Then I interrupted the current thread, so the exception is visible.

3.5 Continuous integration pipeline [optional]

I implemented CI pipeline with GitHub actions, to run all the maven tests everytime I pushed new code to the GitHub Repository (except Cucumber tests, because they needed the running server to pass - Spring-Boot). It would also update the code in SonalCloud (not shown in the report).

```yaml
jobs:
  build:
    name: maven

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Set up JDK 11
      uses: actions/setup-java@v3
      with:
        java-version: '11'
        distribution: 'temurin'
        cache: maven
    - name: Build with Maven
      run: mvn -B package --file HW1/coviddata/pom.xml -Dtest=!CucumberTest
```

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/Patricia-Dias/TQS-98546
Video demo	https://github.com/Patricia-Dias/TQS-98546/tree/master/HW1/demo
QA dashboard (online)	https://sonarcloud.io/summary/overall?id=Patricia-Dias_TQS-98546
CI/CD pipeline	https://github.com/Patricia-Dias/TQS-98546/blob/master/.github/workflows/HW1build.yml

Reference materials

- https://crunchify.com/how-to-create-a-simple-in-memory-cache-in-java-lightweight-cache/