deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

HW1: Mid-term assignment report

Ana Alexandra Antunes [876543], v2022-04-07

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.

For this assignment, it was asked to use COVID-19 API'S in order to show where the most incidence of COVID cases (and more) were. According to this, the product name is "*Covid-19 Info*", and with this product we can see the number of confirmed, deaths, recovered and active cases through some filters, like search for country, search for a specific date and both.

1.2   Current limitations

Although it is also important to have a good working app, there were some things missing, because the major goal of this assignment was the quality assurance strategy. Despite this, there are some known limitations like, the API returns the value zero, for more recent data. Due to this fact, the active cases are almost similar to the confirmed case, for more recent data, because the active cases were a subtraction between the confirmed cases, recovered cases and deaths.

2   Product specification

2.1   Functional scope and supported interactions

The application will be used by people that wanna know about the current or previous situation of the world, being able to search by country, date or even both.

As a user of the web app, it will be possible to see the information in a more detailed way, although as a *REST API* user, it will be possible to have access to more information that is not presented in the web app, although it will be in *JSON* format.
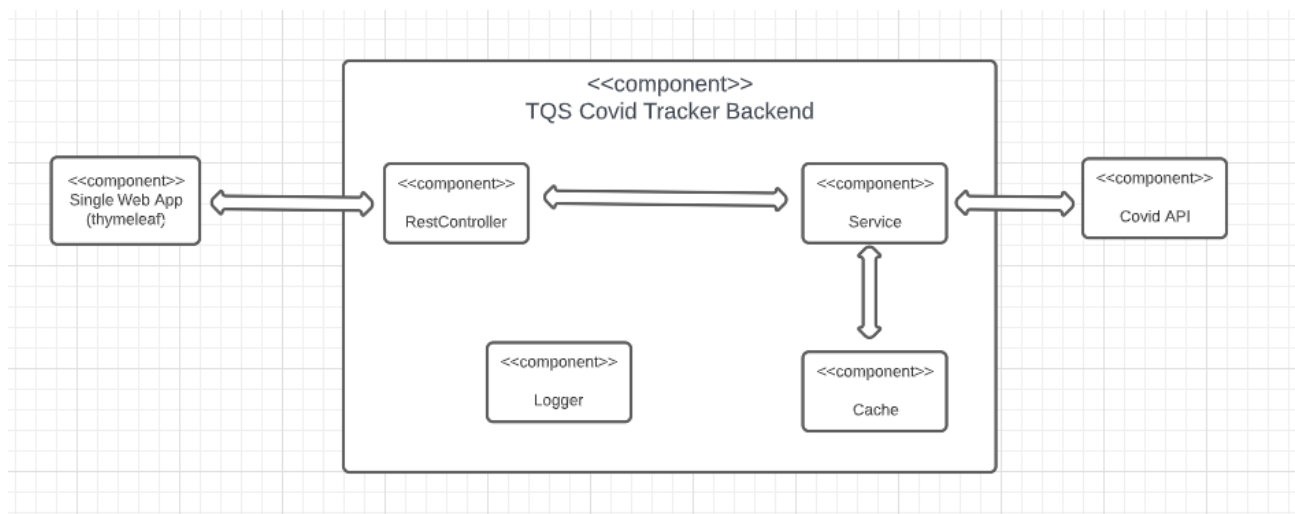
2.2   System architecture

Fig.1 - System Architecture

The flow of the system focuses on the service component which receives the requests from the users, stored them in the cache and do the requests to the external API. In case the user wants to do a request about the covid information on a certain country, the service does the request to the external API and stores that request in the cache. Later, if the user makes the same request, the service no longer needs to do the request to the external API because that request it's already stored in cache.

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

2.3   API for developers

client Operations available to regular clients

GET	/date/{date} searches for covid data in a certain date	⌄
GET	/country/{country} searches for covid data in a certain country	⌄
GET	/date/{date}/country/{country} searches for covid data in a certain country and date	⌄
GET	/cache gets cache details	⌄

Fig.2 - API Documentation

3   Quality assurance

3.1   Overall strategy for testing

I tried to follow a TDD approach alongside the assignment but not always was possible because sometimes it was needed to have a fully working component ready.

3.2   Unit and integration testing

A lot of unit tests were implemented in the classes Cache, Service and Resolver, with Mock objects. For these tests, every flaw scenario was thought of, and each time that it was supposed to fail, an exception would be thrown.

```java
@Test
void callConvertDate_BadAdvancedDate() throws Exception {
    assertThrows(Exception.class, () -> {
        resolver.convertJSONbyDatetoByParams("{\"data\": { \"date\" : \"2023-07-07\"}");
    });
}

@Test
void callConvertDate_BadFormattedDate() throws Exception {
    assertThrows(Exception.class, () -> {
        resolver.convertJSONbyDatetoByParams("{\"data\": { \"date\" : \"20223423-07-07\"}");
    });
}
```

Fig.3 - Unit tests example (ResolverTest)

In terms of integration tests, ByParamsControllerMockMvcIT and RestControllerTemplateIT were implemented.

In the first one, this would be a typical integration test in which several components will participate, like the REST endpoint, the service, the repository,...), in this case, the API is deployed in the normal SpringBoot context. Use the entry point for server-side Spring MVC test support.

In the second test, it would be similar to the previous case, instead of assessing a servlet entry point for tests, it uses an API client. The API is deployed in the normal SpringBoot context and uses a REST client to create realistic requests.

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT, classes = DemoApplication.class)
@AutoConfigureMockMvc
@AutoConfigureTestDatabase
class ByParamsControllerMockMvcIT {
    @Autowired
    private MockMvc mvc;

    @Autowired
    private CovidRepository covidRepository;

    @AfterEach
    public void resetDb() {
        covidRepository.deleteAll();
    }

    @ParameterizedTest
    @ValueSource(strings = {"/api/country/Portugal", "/api/date/2020-07-20", "/api/date/2020-07-20/country/Portugal"})
    void searchBy_andReturn(String arg) throws Exception{
        ByParams byParams1 = createTestClass();
        covidRepository.saveAndFlush(byParams1);

        mvc.perform(get(arg).contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("active", is(byParams1.getActive())))
                .andExpect(jsonPath("confirmed", is(byParams1.getConfirmed())))
                .andExpect(jsonPath("recovered", is(byParams1.getRecovered())))
                .andExpect(jsonPath("deaths", is(byParams1.getDeaths())))
                .andExpect(jsonPath("country", is(byParams1.getCountry())))
                .andExpect(jsonPath("confirmed_diff", is(byParams1.getConfirmedDiff())))
                .andExpect(jsonPath("active_diff", is(byParams1.getActiveDiff())))
                .andExpect(jsonPath("deaths_diff", is(byParams1.getDeathsDiff())))
                .andExpect(jsonPath("recovered_diff", is(byParams1.getRecoveredDiff())))
                .andExpect(jsonPath("fatality_rate", is(byParams1.getFatalityRate())))
                .andExpect(jsonPath("date", is(byParams1.getDate())))
                .andExpect(jsonPath("last_updated", is(byParams1.getLastUpdated())));
    }
}
```

Fig.4 - ByParamsControllerMockMvcIT

```java
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@AutoConfigureTestDatabase
class RestControllerTemplateIT {
    @LocalServerPort
    int randomServerPort;

    @Autowired
    private TestRestTemplate restTemplate;

    @Autowired
    private CovidRepository covidRepository;

    @ParameterizedTest
    @ValueSource(strings = {"/date/2023-07-07", "/date/2032423-07-07", "/country/dont_exist", "/date/2023-07-07/country/Portugal",
    void badTests_withBadInputs(String arg) {
        ResponseEntity<ByParams> response = restTemplate.getForEntity(
                getBaseUrl() + arg,
                ByParams.class);

        assertEquals(HttpStatus.INTERNAL_SERVER_ERROR, response.getStatusCode());
    }
}
```

Fig.5 - RestControllerTemplateIT

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

3.3 Functional testing

The user-facing tests that were implemented were selenium tests, simulating every possible step that a user could make.

```java
@Test
@Order(4)
void cacheDataTest() throws InterruptedException {
        driver.get("http://localhost:8080/cache");
        driver.manage().window().setSize(new Dimension(1866, 1053));

        assertEquals("Cache",
                        driver.findElement(By.xpath("/html/body/div[2]/div/h3")).getText());

        driver.findElement(By.id("btn")).click();

        assertEquals("3",
                        driver.findElement(By.xpath("/html/body/div[2]/div/div/div/div/table/tbody/tr[1]/td")).getText());

        assertEquals("0",
                        driver.findElement(By.xpath("/html/body/div[2]/div/div/div/div/table/tbody/tr[2]/td")).getText());

        assertEquals("3",
                        driver.findElement(By.xpath("/html/body/div[2]/div/div/div/div/table/tbody/tr[3]/td")).getText());

}
```

Fig.6 - Selenium Test example

3.4 Code quality analysis

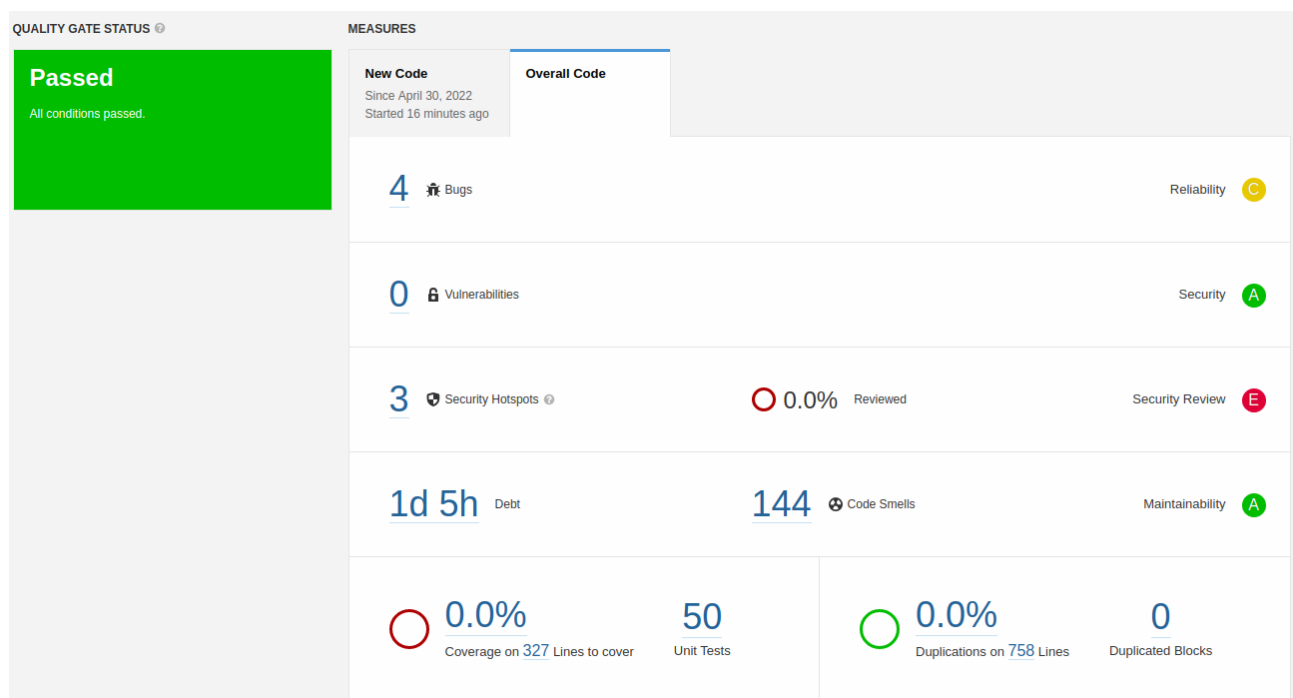For the static code analysis the SonarQube tool was used.

Fig.7 - First look at Sonar

We can see that in the beginning a lot of code smells, bugs and security hotspots were found.
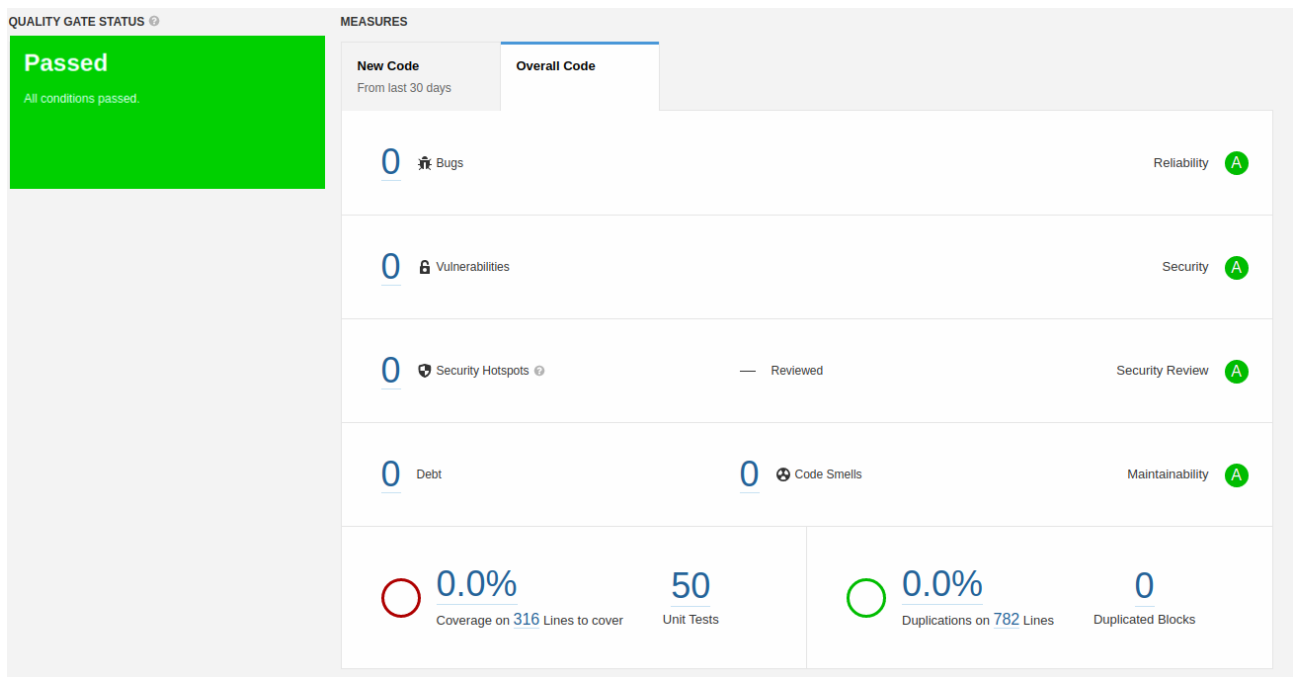

Fig.8 - After refactoring with sonar help

At the end, every code smell and the other problems were solved.

The major lesson that we can gather from this tool, is that even though it works, it probably is not good enough and still has a lot of work to be done. With this, I realize that a lot of insignificant warnings are important because it can lead to attacks to our app.

4   References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/daniff15/tqs_98498/tree/main/HW1
Video demo	< short video demonstration of your solution; consider including in the Git repository>

Reference materials

[1] Material de apoio do docente
[2] COVID-19 Statistics, https://rapidapi.com/axisbits-axisbits-default/api/covid-19-statistics/
[3] Try Out SonarQube, https://docs.sonarqube.org/latest/setup/get-started-2-minutes/