deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

HW1: Mid-term assignment report

Pedro Simão Ministro Jorge [98418], v 2022-05-02

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 main goal of this project is to develop a multi-layer web application, in Spring Boot, supplied with automated tests. The web application should provide COVID incidence data for a certain country and consider, at least, the number of cases for a day or period.
The web application should also be integrated with an external source, implement a cache and provide an endpoint to get basic statistics on the use of the cache.

1.2 Current limitations

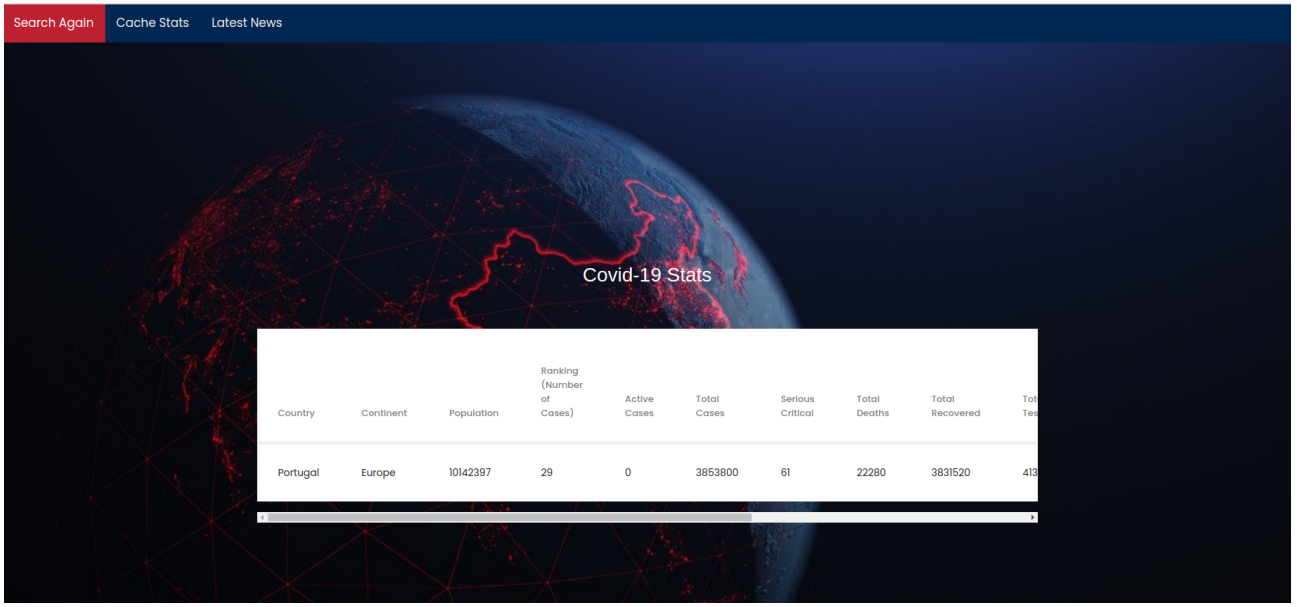
Given the API I chose, there were some limitations that I had to face. For example, when searching for the country, its ISO 3 is also needed to perform the search. This happens because when I tried to GET covid-19 stats from a specific country, the country name and its ISO were required parameters. This is not convenient to perform the search, due to the fact that people may not know the ISO code of the country they are searching for. Either way, it is the only way I found to fetch data from the API.
Another limitation that the web application has is the fact that the user cannot access the news tab by clicking on it in the navbar. This happens because I am fetching the news from a second external API and, on my Controller, I am 'PostMapping' it. Therefore, I need, for example, a form that allows the data to be posted on the frontend site, and this form has to be triggered. Since I cannot implement a form on the navbar, the form can only be triggered in the starting page. All in all, the only way to see the latest news is by clicking on the News button on the starting screen.
There were some unimplemented features I would have liked to implement such as: an autocomplete function for the ISO code when the corresponding country is inserted and a better news tab. For the news tab, I did manage to implement it but it is not perfect. It fetches the data from a second external API, however, it only shows its first result. With more time I think I could have improved this aspect.

2 Product specification

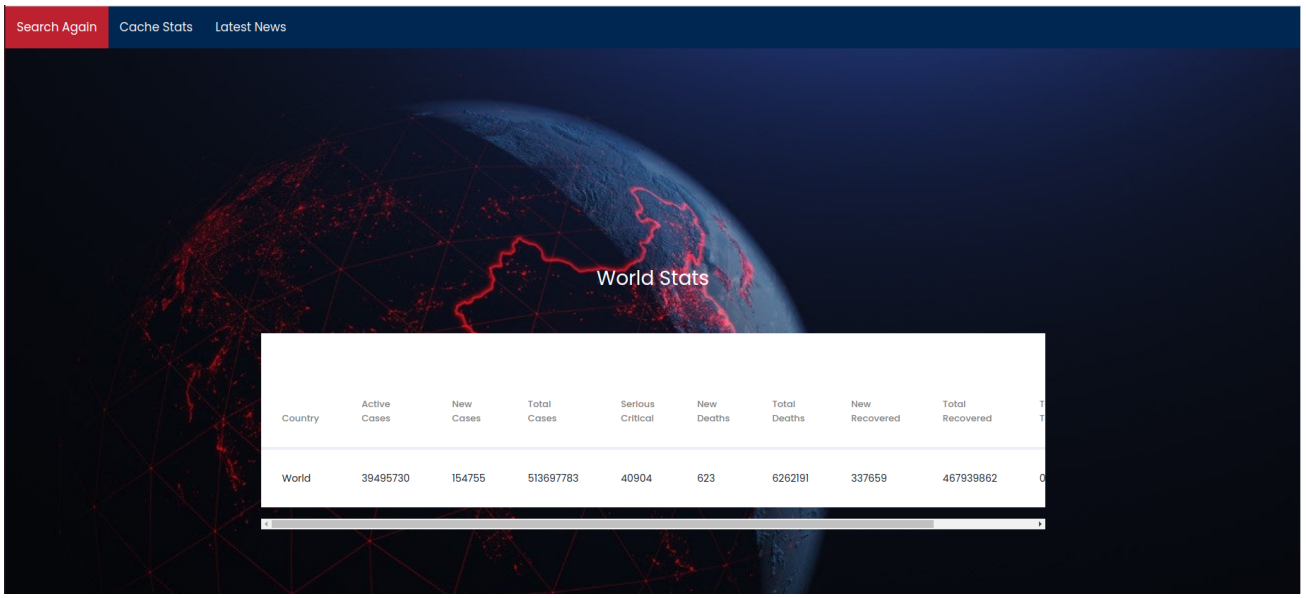
2.1 Functional scope and supported interactions

The web application consists in a web page where people are able to get Covid-19 stats from a specific country, from the world or get the latest news.
The actors who may use this application are people who want to check Covid-19 stats and API maintainers, who may want to check the cache.

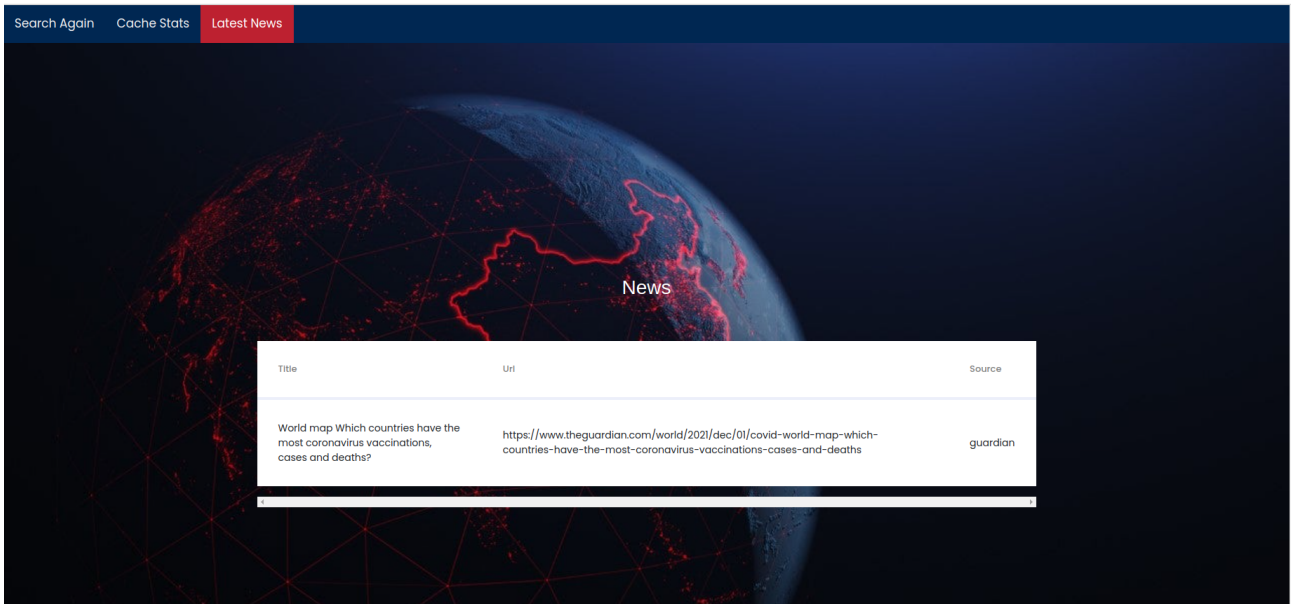
This is the main page of the application. Users can get stats from a specific country by typing its name and ISO code; get the world stats by clicking on the button; or check the latest news by clicking on the button bellow it.



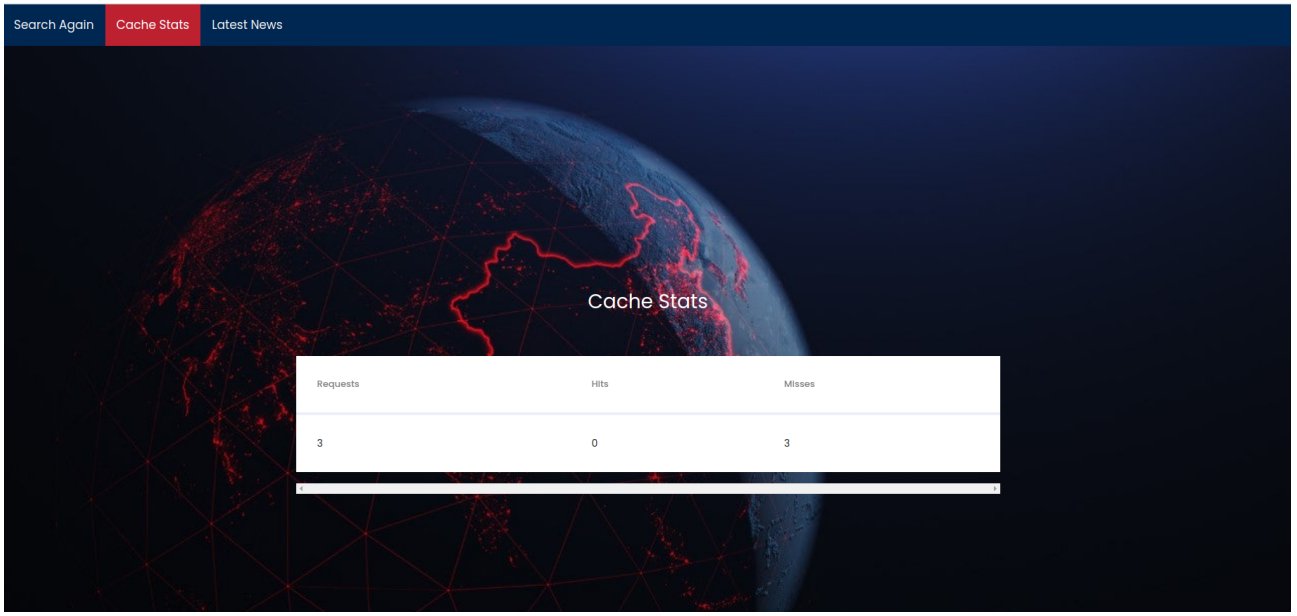
When the user searches for a country (in this case Portugal), he will be redirected to this page where there is a table containing all the stats from the country.



The same happens when the user searches for the world stats.

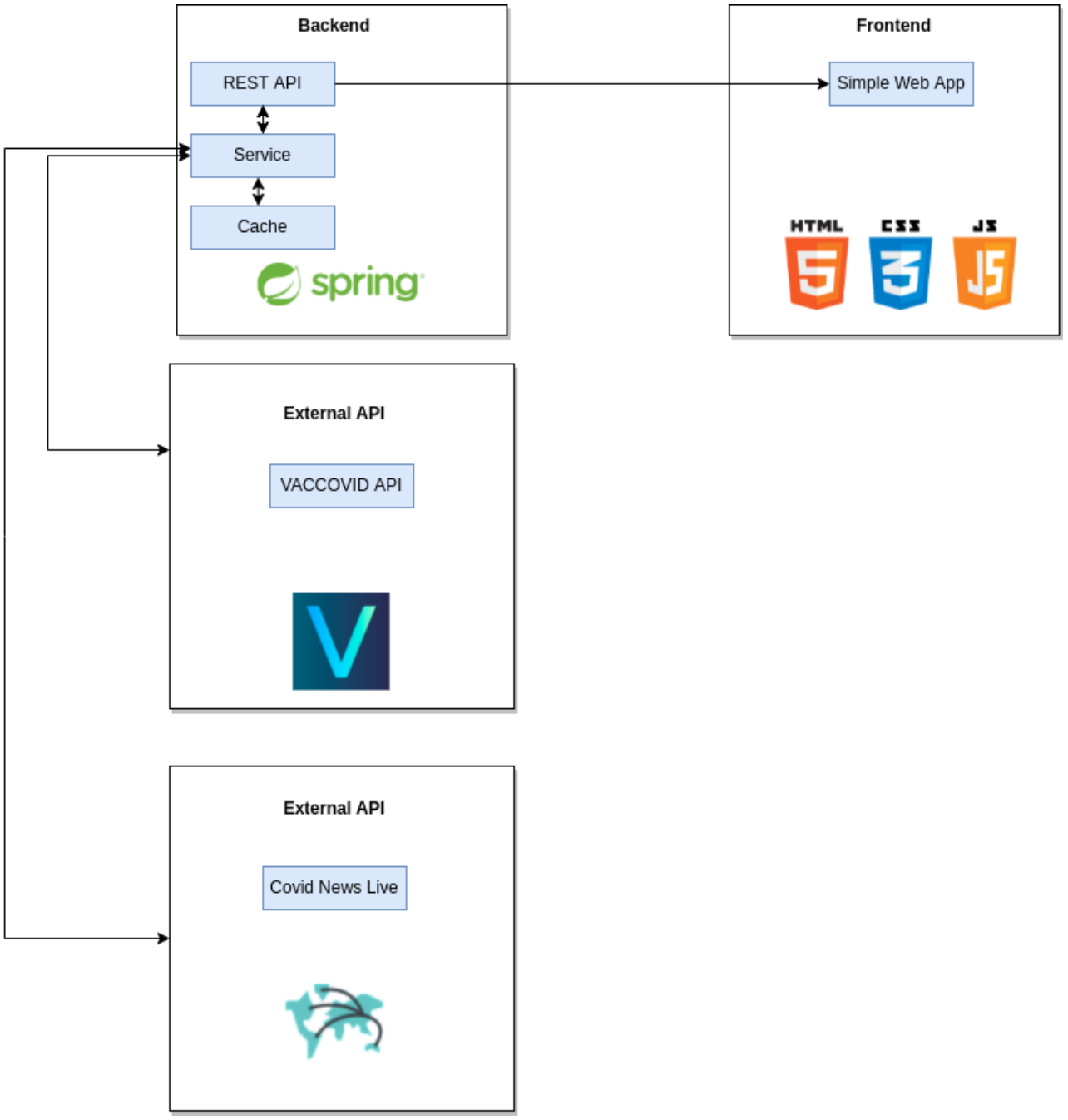


As per the news interface, it's the same as the previous ones, except that it contains the title, url and source of the news.



The cache page can be accessed in the navbar and contains 3 fields: number of requests, number of hits and number of misses. Every time a user performs any kind of search (including news) a request is made, and it counts as a miss. If the user performs the same search another request is made, however, this time it counts as a hit because the result is already cached. This is important to reduce the number of external accesses.

2.2 System architecture



Regarding the system architecture, and starting with the backend, the Service will be the main component. It communicates with the Cache to know if the data is already saved there. If it is, the Cache will return the data to the Service. Otherwise, the Service will communicate with the External API through the 'HttpClientAPI.class' and make a new request. Finally, the Service will pass the data to the REST API and show the data on the frontend.
Regarding the frontend, the data will be shown with the help of Thymeleaf, and all the website pages were coded using HTML, JS and CSS.

2.3 API for developers

Endpoint	Description
/stats	GET stats for a specific country
/stats/world	GET world stats
/news	GET latest news
/cache	GET cache stats

3 Quality assurance

3.1 Overall strategy for testing

Regarding the test development strategy, I opted to first develop all functionalities of the backend and frontend before starting testing because I wanted to first see how the frontend looked, so that is why I went with the BDD strategy instead of the TDD one.

3.2 Unit and integration testing

For unit testing I used JUnit5 and for integration testing I used Mockito and SpringBootMockMVC. The unit testing was used mainly on the model classes, cache and the HTTP Request. As for the model classes, I wanted to test if the data was being processed correctly and to see if the getter methods were working as they should.

```java
package com.uatqs.hw1.model;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;

public class APITest {

    @DisplayName("API Test: Checking all data")
    @Test
    public void apiTest(){
        API api = new API(rank: "1", country: "USA", continent: "North America", infection_Risk: "24.71", totalCases: "82662748", totalDeaths: "1018335", totalR

        assertEquals(expected: "1", api.getRank());
        assertEquals(expected: "USA", api.getCountry());
        assertEquals(expected: "North America", api.getContinent());
        assertEquals(expected: "24.71", api.getInfectionRisk());
        assertEquals(expected: "82662748", api.getTotalCases());
        assertEquals(expected: "1018335", api.getTotalDeaths());
        assertEquals(expected: "80465351", api.getTotalRecovered());
        assertEquals(expected: "1179062", api.getActiveCases());
        assertEquals(expected: "1000463737", api.getTotalTests());
        assertEquals(expected: "334511757", api.getPopulation());
        assertEquals(expected: "4", api.getOne_Caseevery_X_ppl());
        assertEquals(expected: "328", api.getOne_Deathevery_X_ppl());
        assertEquals(expected: "3044", api.getDeaths_1M_pop());
        assertEquals(expected: "1426", api.getSeriousCritical());
        assertEquals(expected: "2990818", api.getTests_1M_Pop());
    }
}
```

As per the cache, I wanted to test the main functionalities such as getting data from cache, saving data into the cache, deleting data from cache and see if the number of requests, hits and misses is working fine. For that, I did 6 tests:

```java
    @DisplayName("Cache Test 1: Get data from cache")
    @Test
    public void getDataFromCache(){
        cache.saveCache(key: "Portugal", value: "Europa");
        cache.saveCache(key: "Austrália", value: "Oceânia");

        assertEquals(expected: "Europa", cache.getCache(key: "Portugal"));
        assertEquals(expected: "Oceânia", cache.getCache(key: "Austrália"));
        assertEquals(expected: 2, cache.getCacheSize());

    }

    @DisplayName("Cache Test 2: Save data in cache")
    @Test
    public void saveDataInCache(){
        cache.saveCache(key: "Portugal", value: "Europa");
        cache.saveCache(key: "Austrália", value: "Oceânia");

        assertEquals(expected: "Europa", cache.getCache(key: "Portugal"));
        assertEquals(expected: "Oceânia", cache.getCache(key: "Austrália"));
    }

    @DisplayName("Cache Test 3: Remove data from cache")
    @Test
    public void removeDataFromCache(){
        cache.saveCache(key: "Portugal", value: "Europa");
        cache.saveCache(key: "Austrália", value: "Oceânia");

        cache.removeFromCache(key: "Portugal");
        assertEquals(expected: null, cache.getCache(key: "Portugal"));
        assertEquals(expected: 1, cache.getCacheSize());
    }

    @DisplayName("Cache Test 4: Assert number of requests is correct")
    @Test
    public void getNRequests(){
        assertEquals(expected: 1, Cache.getnRequests());
    }

    @DisplayName("Cache Test 5: Assert number of hits is correct")
    @Test
    public void getNHits(){
        assertEquals(expected: 4, Cache.getnHits());
    }

    @DisplayName("Cache Test 6: Assert number of misses is correct")
    @Test
    public void getNMisses(){
        assertEquals(expected: 0, Cache.getnMisses());
    }
```

In the first 3 tests the process is very similar: I save data into the cache using the key-value schema and then get data from cache and assert that it is the correct one. The last 3 tests are simple and are meant to assert that the number of request, hits and misses are correct, according to what has been done in the tests above

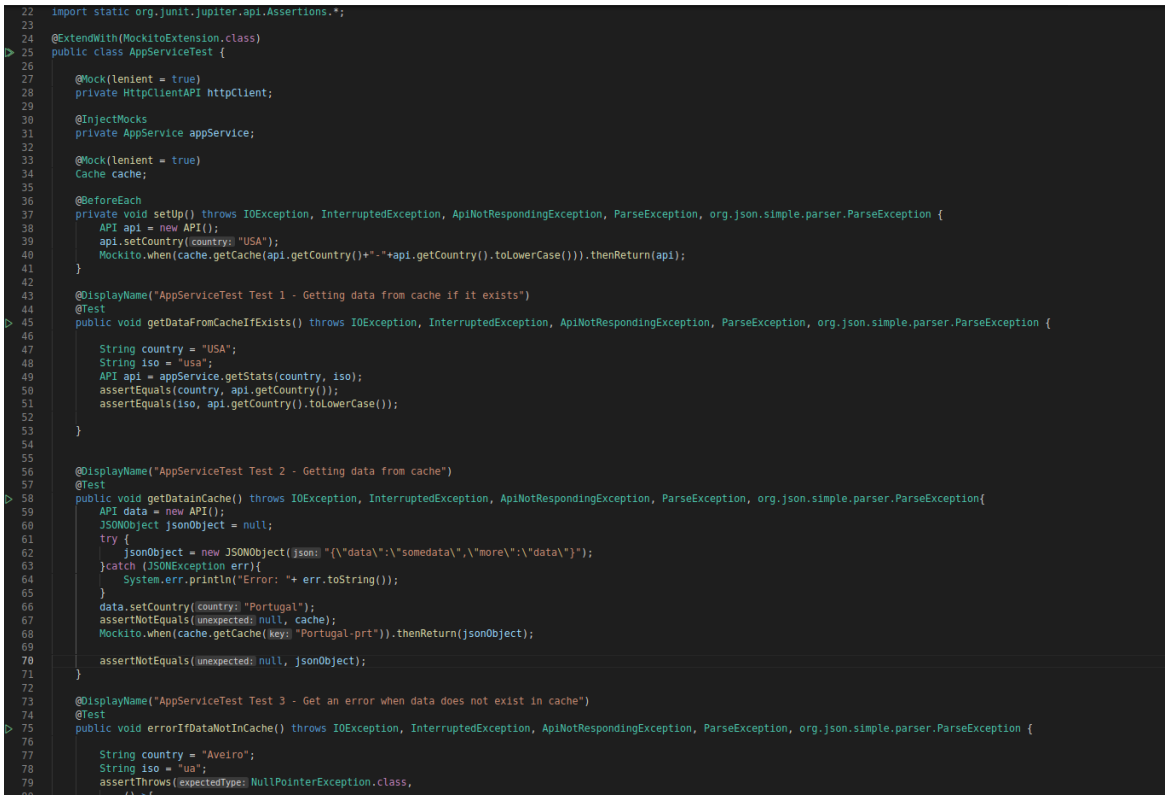
As per the integration testing, it was used to test the Controller and Service.

For the Controller, SpringBoot MockMVC was used

```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.DisplayName;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

@WebMvcTest(AppController.class)
public class AppControllerITTest {

    @MockBean
    private AppService appService;

    @Autowired
    private MockMvc mvc;

    @DisplayName("AppControllerITTest Test - Get Cache")
    @Test
    public void getCache() throws Exception{
        mvc.perform(get(urlTemplate: "/cache")
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk());
    }

}
```

Because of some limitations, I was only able to test the cache controller. This is because in my controller class, getting the cache stats is the only function that has ‘GetMapping’. The others are ‘PostMapping’. Therefore, I am not able to use Mockito to test the getting stats from country, world and the news.

For the Service, Mockito was used.



```java
import static org.junit.jupiter.api.Assertions.*;

@ExtendWith(MockitoExtension.class)
public class AppServiceTest {

    @Mock(lenient = true)
    private HttpClientAPI httpClient;

    @InjectMocks
    private AppService appService;

    @Mock(lenient = true)
    Cache cache;

    @BeforeEach
    private void setUp() throws IOException, InterruptedException, ApiNotRespondingException, ParseException, org.json.simple.parser.ParseException {
        API api = new API();
        api.setCountry( country: "USA");
        Mockito.when(cache.getCache(api.getCountry()+"-"+api.getCountry().toLowerCase())).thenReturn(api);
    }

    @DisplayName("AppServiceTest Test 1 - Getting data from cache if it exists")
    @Test
    public void getDataFromCacheIfExists() throws IOException, InterruptedException, ApiNotRespondingException, ParseException, org.json.simple.parser.ParseException {

        String country = "USA";
        String iso = "usa";
        API api = appService.getStats(country, iso);
        assertEquals(country, api.getCountry());
        assertEquals(iso, api.getCountry().toLowerCase());

    }


    @DisplayName("AppServiceTest Test 2 - Getting data from cache")
    @Test
    public void getDatainCache() throws IOException, InterruptedException, ApiNotRespondingException, ParseException, org.json.simple.parser.ParseException{
        API data = new API();
        JSONObject jsonObject = null;
        try {
            jsonObject = new JSONObject( json: "{\"data\":\"somedata\",\"more\":\"data\"}");
        }catch (JSONException err){
            System.err.println("Error: "+ err.toString());
        }
        data.setCountry( country: "Portugal");
        assertNotEquals( unexpected: null, cache);
        Mockito.when(cache.getCache( key: "Portugal-prt")).thenReturn(jsonObject);

        assertNotEquals( unexpected: null, jsonObject);
    }

    @DisplayName("AppServiceTest Test 3 - Get an error when data does not exist in cache")
    @Test
    public void errorIfDataNotInCache() throws IOException, InterruptedException, ApiNotRespondingException, ParseException, org.json.simple.parser.ParseException {

        String country = "Aveiro";
        String iso = "ua";
        assertThrows( expectedType: NullPointerException.class,
```

3.3 Functional testing

Regarding the functional testing I used Cucumber and Selenium and created 7 different scenarios.

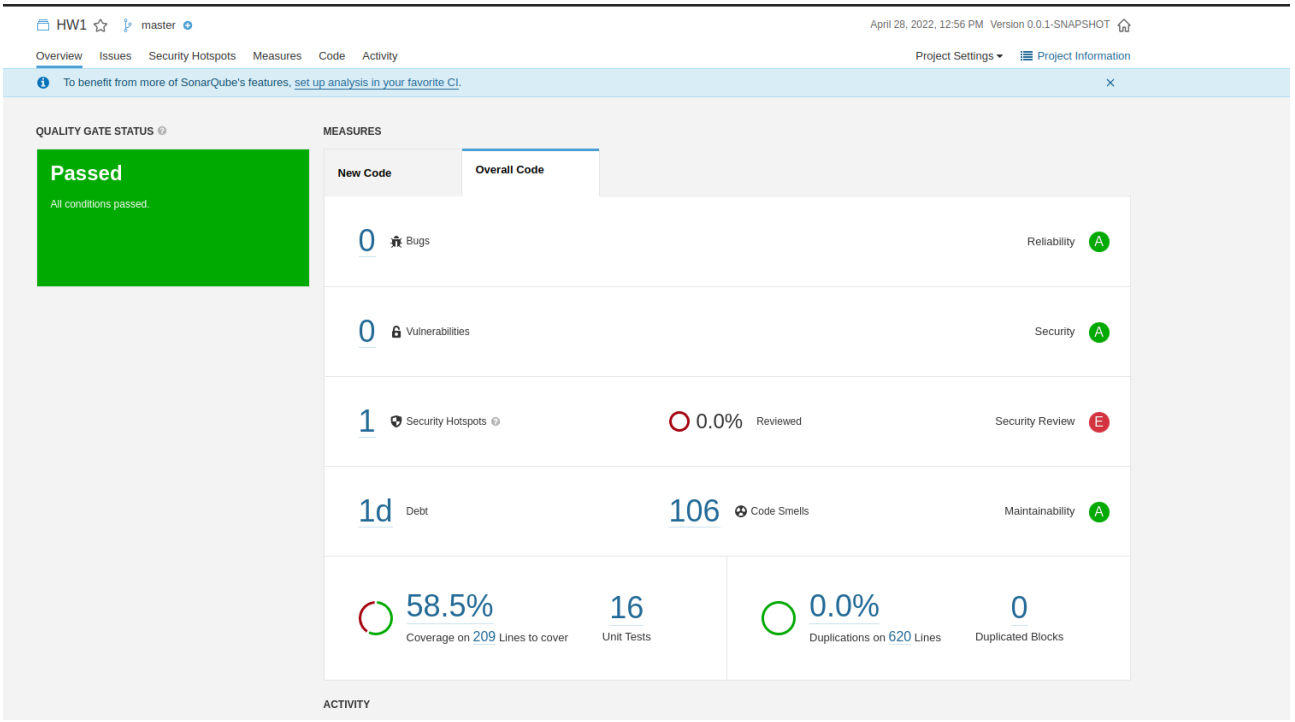
```
Feature: HW1 Functional Testing

  Scenario: Search for Portugal stats
  When I navigate to "http://localhost:7012/"
  And I type "Portugal" as the Country
  And I type "prt" as the ThreeLetterSymbol
  And I click on Submit
  Then I should see the message "Stats"

  Scenario: Search for United States stats
  When I navigate to "http://localhost:7012/"
  And I type "USA" as the Country
  And I type "usa" as the ThreeLetterSymbol
  And I click on Submit
  Then I should see the message "Stats"

  Scenario: Search for World stats
  When I navigate to "http://localhost:7012/"
  And I click on World Stats
  Then I should see the message "Stats"

  Scenario: Search for News
  When I navigate to "http://localhost:7012/"
  And I click on News
  Then I should see the message "News"

  Scenario: Search for invalid Country
  When I navigate to "http://localhost:7012/"
  And I type "Enterro" as the Country
  And I type "prt" as the ThreeLetterSymbol
  And I click on Submit
  Then I should see the error "Oops, something went wrong"

  Scenario: Search for invalid ISO
  When I navigate to "http://localhost:7012/"
  And I type "Portugal" as the Country
  And I type "ua" as the ThreeLetterSymbol
  And I click on Submit
  Then I should see the error "Oops, something went wrong"

  Scenario: Get Cache stats after 4 Requests and 4 Misses
  When I navigate to "http://localhost:7012/cache"
  And I see value of Requests and is "6"
  And I see value of Hits and is "0"
  And I see value of Misses and is "6"
```

- **Search for Portugal stats** - Navigate to page, type ‘Portugal’ and ‘prt’, click Submit and assert that the message ‘Stats’ is seen;
- **Search for United States stats** - Navigate to page, type ‘USA’ and ‘usa’, click Submit and assert that the message ‘Stats’ is seen;
- **Search for World stats** - Navigate to page, click World Stats button and assert that the message ‘Stats’ is seen;
- **Search for News** - Navigate to page, click News and assert that the message News is seen;
- **Search for invalid Country -** Navigate to page, type ‘Enterro’ and ‘prt’, click Submit and assert that the message ‘Oops, something went wrong’ is seen;
- **Search for invalid ISO -** Navigate to page, type ‘Portugal’ and ‘ua’, click Submit and assert that the message ‘Oops, something went wrong’ is seen;
- **Get Cache after 6 Requests and 6 Misses -** Navigate to page and check that the number of requests is 6, hits is 0 and misses is 6.

```java
@When("I navigate to {string}")
public void navigateTo(String url){
    webDriver = WebDriverManager.chromedriver().create();
    webDriver.get(url);
}

@And("I type {string} as the Country")
public void typeCountry(String Country){
    webDriver.findElement(By.xpath( xpathExpression: ".//input[@name='Country']")).sendKeys(Country);
}

@And("I type {string} as the ThreeLetterSymbol")
public void typeThreeLetterSymbol(String ThreeLetterSymbol){
    webDriver.findElement(By.xpath( xpathExpression: ".//input[@name='ThreeLetterSymbol']")).sendKeys(ThreeLetterSymbol);
}

@And("I click on Submit")
public void clickSubmit(){
    webDriver.findElement(By.cssSelector( cssSelector: ".s01 form .inner-form .input-field.third-wrap .btn-search")).click();
}

@And("I click on World Stats")
public void clickWorldStats(){
    webDriver.findElement(By.cssSelector( cssSelector: ".s01 form .inner-form .input-field.third-wrap.world .btn-search")).click();
}

@And("I click on News")
public void clickNews(){
    webDriver.findElement(By.cssSelector( cssSelector: ".s01 form .inner-form .input-field.third-wrap.news .btn-search")).click();
}

@And("I see value of Requests and is {string}")
public void valueOfRequests(String number){
    assertThat(webDriver.findElement(By.xpath( xpathExpression: ".//table/tbody/tr/td[1]")).getText(), containsString(number));
}

@And("I see value of Hits and is {string}")
public void valueOfHits(String number){
    assertThat(webDriver.findElement(By.xpath( xpathExpression: ".//table/tbody/tr/td[2]")).getText(), containsString(number));
}

@And("I see value of Misses and is {string}")
public void valueOfMisses(String number){
    assertThat(webDriver.findElement(By.xpath( xpathExpression: ".//table/tbody/tr/td[3]")).getText(), containsString(number));
}

@Then("I should see the message {string}")
public void seeMessage(String message) throws InterruptedException{
    assertThat(webDriver.findElement(By.xpath( xpathExpression: ".//div[@class='col-md-6 text-center mb-5']/h2[@class='heading-section']")).getText(), containsString(message));
}

@Then("I should see the error {string}")
public void seeErrorMessage(String message) throws InterruptedException{
    assertThat(webDriver.findElement(By.xpath( xpathExpression: ".//div[@class='col-md-6 text-center mb-5']/h2[@class='heading-section']")).getText(), containsString(message));
}
```

3.4 Code quality analysis

For static code analysis I used SonarQube. This is a very important tool because it allows users to know the quality of their code and detect vulnerabilities or code smells for example.

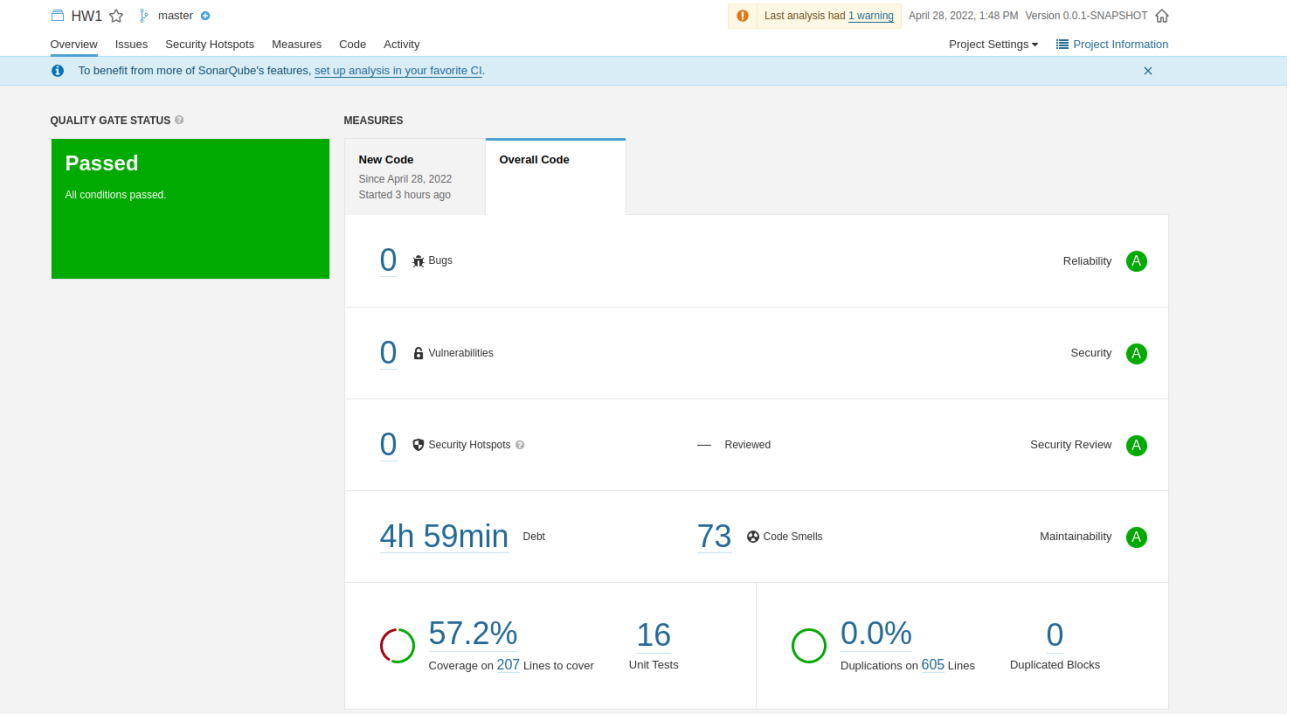


The first report showed that my code was not good. Even though it got grade A on Reliability, Security and Maintainability, there was a Security Hotspot (Cross Origin), 106 code smells and only 58.5% coverage.

From the 106 code smells there was:

- 1 Blocker (add tests to the Cucumber Test)
- 8 Critical (make the enclosing method static or remove this set)
- 27 Major
- 54 Minor (Rename variables and unused imports)
- 16 Info (remove public modifiers)

After refactoring the code, I did another test.



The second report shows that there are no more security hotspots and the debt was reduced from 1 day to 4h59min. The code smells also went down from 106 to 73, and from these 73 there were no blocker or critical issues. This is a much better result than the previous one, however, I still think I could have reduced the debt and the number of code smells a little bit more.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/pedrosimao10/TQS_98418/tree/main/hw1
Video demo	https://github.com/pedrosimao10/TQS_98418/tree/main/hw1/media/demo

Reference materials

First External API (VACCOVID): https://rapidapi.com/vaccovidlive-vaccovidlive-default/api/vaccovid-coronavirus-vaccine-and-treatment-tracker/

Second External API (Covid News Live): https://rapidapi.com/kanitmann/api/covid-news-live1/