HW1: Mid-term assignment report

Tomé Lopes Carvalho [97939], v2022-04-19

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 purpose of the application is to provide details on COVID-19 incidence data (cases, deaths and tests) by country/region, featuring a REST API with a cache mechanism as well as a Web application that makes use of it.

1.2 Current limitations

Limitations caused by the chosen third-party API:

- Fetching of data for a given time range: the third-party API only allows us to either fetch data for a single day or fetch the entire history, thus it would be necessary to perform *n* requests for *n* days in the range or fetch the entire history and filter it, respectively.
- Selection of continents as regions: although countries' continents are given when fetching statistics, they are not given in the third-party API's */countries*, endpoint, which is the one that allows us to efficiently fetch the list of countries/regions.

2 Product specification

2.1 Functional scope and supported interactions

The application could be used by people who wish to check COVID-19 incidence statistics.
The main usage scenario is:

- Choose a country/region
- Choose a date
- Analyze the information shown in the table

2.2 System architecture

The Spring Framework, along with Spring Boot, was used to create the REST API. Springdoc OpenAPI was used to generate the Swagger UI API documentation.
I chose ReactJS as the JavaScript framework for the web application, as I am moderately familiar with it and it provides access to good libraries, such as Material UI.

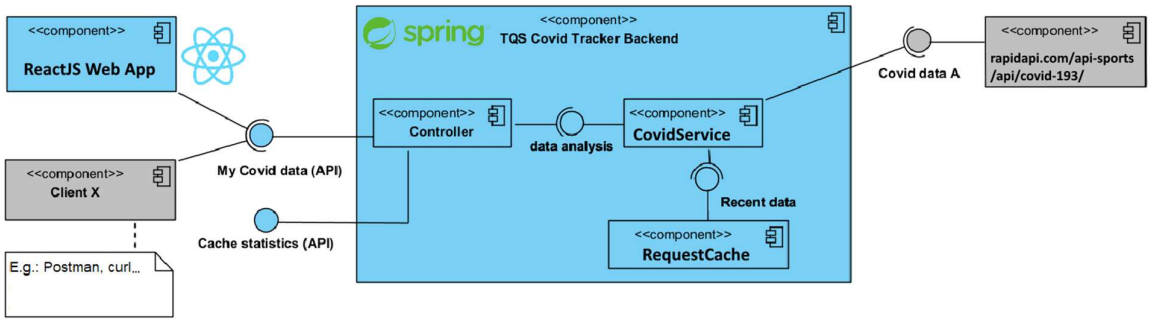


Figure 1: Architecture Diagram

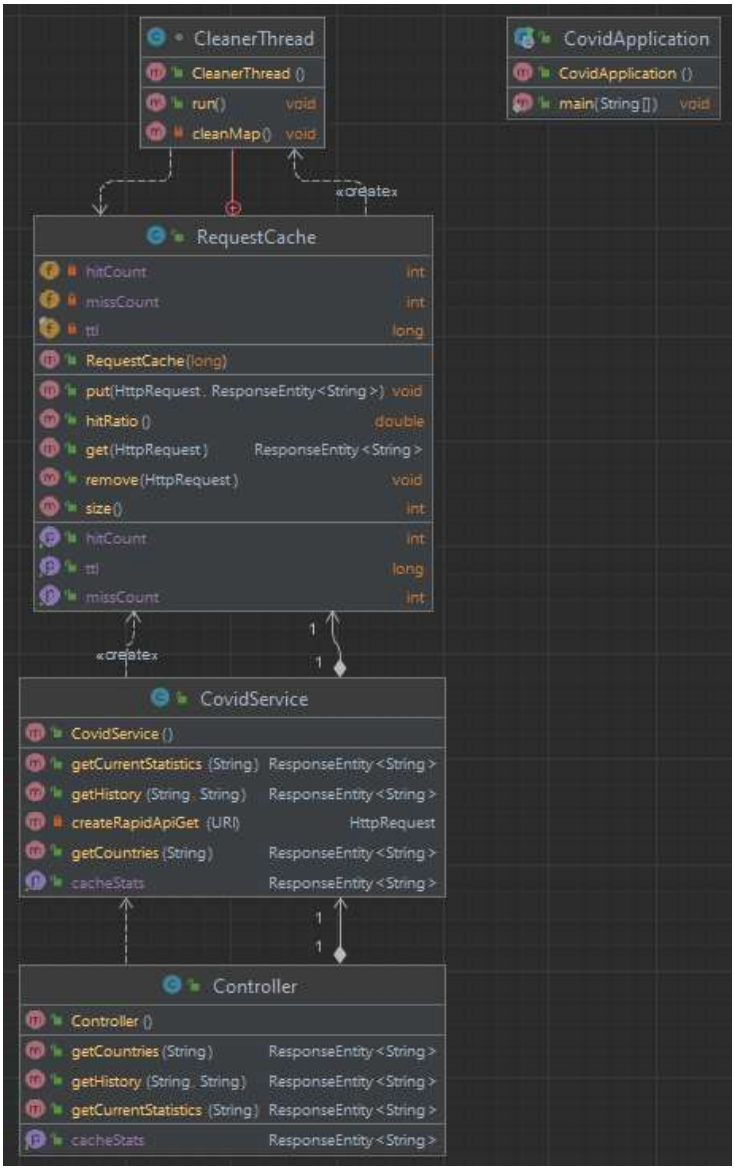
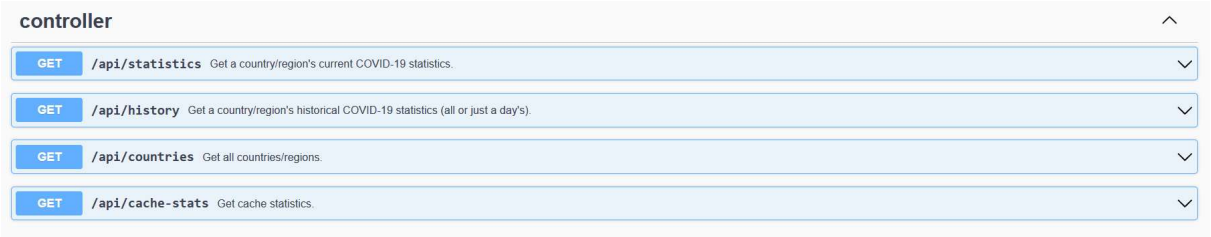


Figure 2: API Java Class Diagram

2.3 API for developers



3 Quality assurance

3.1 Overall strategy for testing

Due to uncertainty about the implementations and contracts of the classes developed, I decided to develop the functionality first and then validate it with tests. A behavior-driven development approach was taken for the development of the Selenium IDE tests, using Cucumber.

3.2 Unit and integration testing

Unit tests were written for the controller (using a mock service) and the cache. An integration test using the real service (connected to the third-party API) was developed.

3.3 Functional testing

Due to the simplicity of the web application, only a single user-facing test was implemented, using Selenium IDE and Cucumber for behavior-driven development.

```
Feature: Statistics search
  Scenario: Fetch statistics
    Given the user opens Firefox and navigates to the application
    When the user searches for statistics for the country USA on the date 2021-06-17
    Then the table should show '+9743' new cases
    And the table should show '+276' new deaths
```

```
@When("the user searches for statistics for the country USA on the date 2021-06-17")
public void theUserSearchesForStatisticsForTheCountryOnTheDate() {
    WebElement usa = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("autocomplete-countries-option-222")));
    usa.click();
    driver.findElement(By.id(":r5:")).click();
    driver.findElement(By.cssSelector(".css-fd2y78-MuiSvgIcon-root")).click();
    driver.findElement(By.cssSelector(".PrivatePickersYear-root:nth-child(122) > .PrivatePickersYear-yearButton")).click();
    js.executeScript( script: "window.scrollTo(0,0)");
    driver.findElement(By.cssSelector(".MuiIconButton-edgeStart > .MuiSvgIcon-root")).click();
    driver.findElement(By.cssSelector(".MuiIconButton-edgeStart > .MuiSvgIcon-root")).click();
    driver.findElement(By.cssSelector(".css-mvmu1r:nth-child(3) > div:nth-child(5) > .MuiButtonBase-root")).click();
    driver.findElement(By.cssSelector(".MuiButton-root:nth-child(2)")).click();
}

@Then("the table should show {string} new cases")
public void theTableShouldShowNewCases(String newCases) {
    String res = driver.findElement(By.cssSelector(".MuiTableRow-root:nth-child(1) > .MuiTableCell-body:nth-child(2)")).getText();
    assertThat(res, equalTo(newCases));
}

@And("the table should show {string} new deaths")
public void theTableShouldShowNewDeaths(String newDeaths) {
    String res = driver.findElement(By.cssSelector(".MuiTableRow-root:nth-child(1) > .MuiTableCell-body:nth-child(3)")).getText();
    assertThat(res, equalTo(newDeaths));
}
```

3.4 Code quality analysis

At first, I used IntelliJ's built-in code inspector. After that, I installed the SonarLint plugin and used it to further analyze the code. Lastly, I added the project to SonarQube and made sure I had no bugs, vulnerabilities, security hotspots or code smells, guaranteeing an A rating in all categories. I also confirmed the test coverage was acceptable. The result was 72.1%, but a lot of the uncovered parts are related to *catch* blocks that simply log the exception and return a response with the 500 Internal Server Error HTTP status code.
Some issues I realized thanks to these tools were:

- String formatting (i.e., with *String.format* or *MessageFormat.format*) is favored over string concatenation.
- Test classes should have default visibility, not public.
- An *InterruptedException* should not be ignored (merely logging it is considered ignoring it).
- Field injection using the *@Autowired* annotation is not recommended; The use of a constructor is preferred.

4 References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/TomeCarvalho/tqs_97939
Video demo	https://raw.githubusercontent.com/TomeCarvalho/tqs_97939/main/HW1/docs/demo.mp4

Reference materials

- ReactJS: Material UI
- Awaitility
- Springdoc