deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

HW1: Relatório do Trabalho Intermediário

João António Assis Reis [98474], v2022-02-02

índice

1 Introdução

1.1 Visão geral do trabalho

O objetivo do trabalho intermédio individual da unidade curricular Testes e Qualidade de Software é desenvolver uma aplicação que forneça informações específicas sobre a Covid-19 num país, escolhido pelo utilizador. Portanto, esta aplicação é desenvolvida em Spring Boot que comunica com uma API externa (neste caso, VACCOVID - coronavirus, vaccine and treatment tracker API Documentation. Esta conta com múltiplos testes às diferentes camadas. Em relação à interação com o utilizador, foi desenvolvida também uma interface Web em Angular.

O nome do produto é ***CovidInfo*** e permite que o utilizador tenha acesso a informações de um determinado país ou do mundo.

1.2 Limitações atuais

Uma das limitações do produto é a filtrar a informação da Covid-19 de um país por data. No início era um dos objetivos, mas por uma questão de simplificação do projeto em si devido ao tempo, essa ideia foi abandonada. Mas se o desenvolvimento do produto continuasse, era algo a ser implementado.

Para além disto, o produto tem alguns recursos não implementados, tais como:

- A conexão com uma segunda API externa.
- O uso de uma *Continuous Integration framework*.

Estes recursos não foram implementados devido à falta de tempo.

2 Especificação do Produto

2.1 Espetro funcional e Interações suportadas

Esta aplicação destina-se a todos aqueles que pretendem obter informações sobre a Covid-19 no mundo ou de um determinado país. O utilizador apenas precisa de inserir o código ISO ou o nome de um país e clicar no botão de pesquisa ou no botão Enter.

A informação sobre o país selecionado será apresentada. Estas informações vão desde o número total de casos desse país, como o número total de mortes e de internados, até a informações demográficas como a população e o risco de infeção desse país.

Caso o nome ou o código ISO inserido não corresponder a nenhum país, será apresentada uma mensagem de erro, informando o utilizador que não foi encontrado nenhum país com os dados inseridos.

Para além disto, é possível ter uma visão geral do estado do mundo em relação à Covid-19 na página inicial da aplicação, como por exemplo, o top 10 dos países mais afetados pela pandemia atualmente.

Por fim, é ainda permitido ter acesso a certos detalhes da cache.

2.2 Arquitetura do sistema

Como já foi anteriormente mencionado, a arquitetura é constituída por 3 componentes. Uma delas é a componente de Serviço, desenvolvida em **Spring Boot**; outra é a componente da interface Web, desenvolvida em **Angular**; e a terceira componente é a API externa, ***VACCOVID***.

A seguinte figura esquematiza toda a arquitetura implementada.

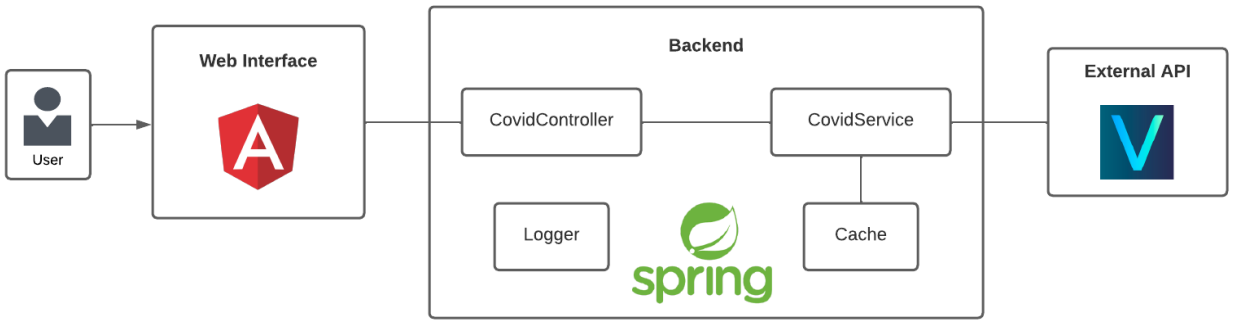

Fig.1 - Arquitetura do projeto

2.3 API para desenvolvedores

A documentação dos endpoints da API está disponível no endpoint ***/swagger-ui/index.html***, utilizando a ferramenta *Swagger*.

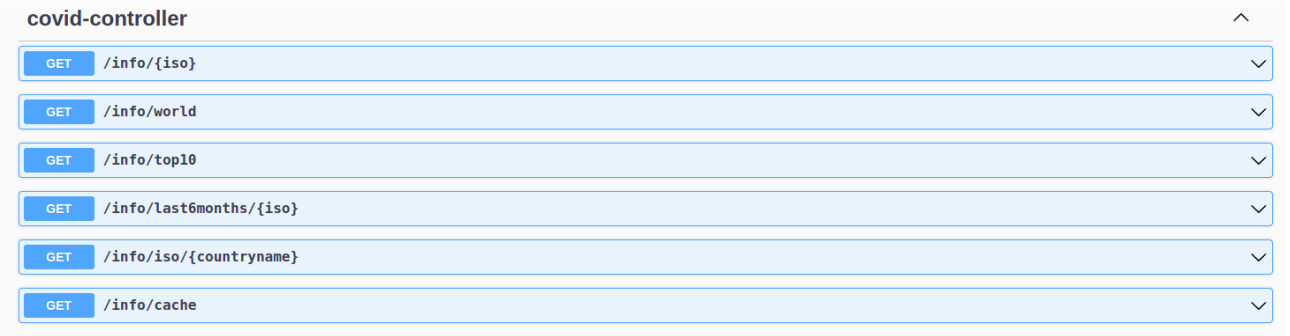

Fig. 3 - Todos os API endpoints do projeto

- **/info/{iso}** - este endpoint dá acesso a toda a informação sobre Covid-19 relacionada a um país, indicado pelo código ISO no url;
- **/info/world** - dá acesso a toda a informação sobre o mundo;
- **/info/top10** - dá acesso a informação dos 10 países mais afetados com a Covid-19, ou seja, o top 10 do rank;
- **/info/last6months/{iso}** - dá acesso a informação sobre Covid-19 relacionada a um país nos últimos 6 meses, indicado pelo código ISO no url;
- **/info/iso/{countryname}** - dado o nome de um país pelo url, é enviado o código ISO desse mesmo país;
- **/info/cache** - dá acesso a detalhes da cache.

3 Garantia de Qualidade

3.1 Estratégia utilizada nos testes

Para testar a implementação e as funcionalidades do produto, inicialmente a estratégia a utilizar seria TDD (Test Driven Development), uma vez que é uma ótima abordagem para uma perceção dos cenários que uma componente terá de ser capaz de ultrapassar.

Contudo, primeiro, foi realizado um esqueleto do backend e só depois os respetivos testes, dado que, à medida do desenvolvimento do mesmo, a escolha de testes a implementar tornou-se mais clara.

O objetivo é testar o máximo de componentes e os seus comportamentos. Para isso, foram realizados testes unitários em *Junit5*, testes de integração e serviços, utilizando as ferramentas *Mockito* e *SpringBoot MockMvc*. E para os testes de frontend, aplicou-se uma Behavior-Driven Development (BDD), recorrendo ao *Selenium WebDriver*.

3.2 Testes unitários e de integração

Os testes unitários foram implementados na classes de modelação e de serviço, por exemplo, na classe CacheTest, onde estão a maioria dos testes unitários, e na classe CovidServiceTest. Na primeira são testadas todas as funcionalidades de uma cache, por exemplo, adicionar/eliminar um elemento. Na segunda, é testado o funcionamento da componente serviço, utilizando objetos Mock que são injetados na instância da classe que se pretende testar.

```java
@DisplayName("1. Add item to cache test")
@Test
public void addTest() {
    this.cache.add(key: "um", object: "one");

    assertEquals( this.cache.get(key: "um"), actual: "one");
    assertEquals( this.cache.size(), actual: 1);
}


@DisplayName("2. Clean item of the cache test")
@Test
public void cleanTest() {
    this.cache.add(key: "um", object: "one");
    this.cache.add(key: "dois", object: "two");
    assertEquals( this.cache.size(), actual: 2);

    this.cache.clean(key: "um");
    assertEquals( this.cache.size(), actual: 1);
    assertFalse( this.cache.clean(key: "um") );
}


@DisplayName("3. Get item of the cache test")
@Test
public void getTest() {
    this.cache.add(key: "um", object: "one");
    this.cache.add(key: "dois", object: "two");

    assertEquals( this.cache.get(key: "dois"), actual: "two");      // requests = 1 | misses = 0 | hits = 1
    assertNotEquals( this.cache.get(key: "um"), actual: "two");     // requests = 2 | misses = 0 | hits = 2
    assertEquals( this.cache.get(key: "três"), actual: null );      // requests = 3 | misses = 1 | hits = 2

    assertEquals( this.cache.getRequests(), actual: 3);
    assertEquals( this.cache.getHits(), actual: 2);
    assertEquals( this.cache.getMisses(), actual: 1);
}


@DisplayName("4. Cache behaviour test")
@Test
public void behaviourTest() {
    this.cache.add(key: "um", object: "one");
    this.cache.add(key: "dois", object: "two");

    assertEquals( this.cache.size(), actual: 2);
    assertEquals( this.cache.get(key: "dois"), actual: "two");

    try {
        Thread.sleep(3000); // sleep 3 seconds

    } catch(InterruptedException e) {
        System.out.println("got interrupted!");
    }

    assertEquals( this.cache.size(), actual: 0);
    assertEquals( this.cache.get(key: "dois"), actual: null);
}
```

Fig 4 - Testes unitários na classe CacheTest

```java
@ExtendWith(MockitoExtension.class)
public class CovidServiceTest {

    @Mock
    private Cache cache;

    @Mock
    private Request api;

    @InjectMocks
    private CovidService service;


    @Test
    void getWorldDataTest() throws IOException, InterruptedException {

        CovidInfo world_info = new CovidInfo(id: "1", country: "World", continent: "All", iso: null, TwoLetterSymbo

        Mockito.when( api.requestTo( Mockito.anyString() ) ).thenReturn( "["+ world_info.toString() +"]" );

        CovidInfo info = service.getWorldData();

        assertEquals(world_info.toString(), info.toString());

    }


    @Test
    void getCountryDataTest() throws IOException, InterruptedException {

        CovidInfo country_info = new CovidInfo(id: "2", country: "Portugal", continent: "Europe", iso: "prt", Twol

        Mockito.when( api.requestTo( Mockito.anyString() ) ).thenReturn( "["+ country_info.toString() +"]" );

        String iso = country_info.getIso();
        String countryname = country_info.getCountry();

        CovidInfo info = service.getCountryData(iso, countryname);

        assertEquals(country_info.toString(), info.toString());

    }
```

Fig 5 - Testes na classe CovidServiceTest utilizando Mock

Enquanto que nos testes de integração, estes foram implementados na componente controller, recorrendo ao SpringBoot MockMvc. Cada endpoint tem um teste que avalia o seu comportamento. Para tal, para cada chamada ao serviço, é preparada uma resposta a esse pedido.

```java
@WebMvcTest(CovidController.class)
public class CovidControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private CovidService service;

    @Test
    void getWorldDataTest() throws Exception {

        CovidInfo country_info = new CovidInfo(id: "1", country: "Portugal", continent: "Europe", iso: "PRT", TwoLette

        when( service.getWorldData() ).thenReturn( country_info );

        mvc.perform(
            get(urlTemplate: "/info/world").contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath(expression: "$.*", hasSize(size: 16)))
            .andExpect(jsonPath(expression: "$.id", is(country_info.getId())))
            .andExpect(jsonPath(expression: "$.country", is(country_info.getCountry())))
            .andExpect(jsonPath(expression: "$.continent", is(country_info.getContinent())))
        );

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

    @Test
    void getCountryData() throws Exception {

        CovidInfo country_info = new CovidInfo(id: "1", country: "Portugal", continent: "Europe", iso: "prt", TwoLette

        when( service.getCountryData( Mockito.anyString(), Mockito.anyString() ) ).thenReturn( country_info );
        when( service.getCountryByISO( Mockito.anyString() ) ).thenReturn( value: "Portugal" );

        mvc.perform(
            get(urlTemplate: "/info/prt").contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath(expression: "$.*", hasSize(size: 16)))
            .andExpect(jsonPath(expression: "$.id", is(country_info.getId())))
            .andExpect(jsonPath(expression: "$.country", is(country_info.getCountry())))
            .andExpect(jsonPath(expression: "$.continent", is(country_info.getContinent())))
        );

        verify(service, times(wantedNumberOfInvocations: 1)).getCountryData(iso_code: "prt", country_name: "Portugal" );
```

Fig 6 - Testes na classe CovidControllerTest utilizando SpringBoot MockMvc

3.3 Testes funcionais

Para os testes funcionais foi utilizado uma abordagem Behavior-Driven Development (BDD), recorrendo ao Selenium WebDriver. Estes testes servem para testar a interface Web, basicamente, testar o produto em si. Para isso, foram escritos cenários e cada um deles tarefas um ficheiro *.feature*, onde o teste vai realizar essas tarefas e concluir então os cenários descritos.

```
Feature: Covid

    Scenario: Search for Portugal covid-19 stats
        When I want to access "http://localhost:4200/world"
        And I click in "By Country" side bar button
        And I should see "Search for a specific country"
        And I fill the iso code input with "prt" and the country name input with "Portugal"
        And I click in search button
        Then I should see the following message "Covid-19 stats in the last 6 months"

    Scenario: See covid-19 stats of the 3rd country most affected
        When I want to access "http://localhost:4200/world"
        And I click in 3 button in the table
        Then I should see the following message "Covid-19 stats in the last 6 months"

    Scenario: Search for an ISO code that doesn't exist
        When I want to access "http://localhost:4200/world"
        And I click in "By Country" side bar button
        And I should see "Search for a specific country"
        And I fill the iso code input with "iso"
        And I click in search button
        Then I should see the following message "Country not found :/ Please, try another one"
```

Fig 7 - Ficheiro *.feature* com os cenários

```java
public class FrontendSteps {

    private WebDriver driver;


    @When("I want to access {string}")
    public void iNavigateTo(String url) {
        driver = new ChromeDriver();
        driver.get(url);
        driver.manage().window().setSize(new Dimension(width: 1840, height: 1053));
    }


    @And("I click in \"By Country\" side bar button")
    public void clickSideBarButton() {
        driver.findElement(By.cssSelector(cssSelector: ".sidebar-item:nth-child(2) .hide-menu")).click();
    }


    @And("I should see {string}")
    public void seeElement(String element) {
        assertThat(driver.findElement(By.cssSelector(cssSelector: "h4")).getText(), containsString(element));
    }


    @Then("I should see the following message {string}")
    public void seeHTMLElement(String element) throws InterruptedException {
        Thread.sleep(2000); // wait for the data
        assertThat(driver.findElement(By.cssSelector(cssSelector: ".test")).getText(), containsString(element));
    }


    @And("I fill the iso code input with {string} and the country name input with {string}")
    public void fillInfo(String iso, String countryname) {
        driver.findElement(By.id(id: "form1")).click();
        driver.findElement(By.id(id: "form1")).sendKeys(iso);
        driver.findElement(By.id(id: "form2")).click();
        driver.findElement(By.id(id: "form2")).sendKeys(countryname);
    }
```

Fig 8 - Ficheiro de testes do frontend

3.4 Análise da qualidade do código

Para uma análise estática do código, a ferramenta utilizada foi o *SonarCloud*. Os resultados obtidos estão presentes na seguinte figura.

Fig. 9 - Resultados da análise do SonarCloud

Pela análise destes resultados, verifica-se que há 0 bugs, 0 vulnerabilidades, 1 Security Hotspots devido à anotação *@CrossOrigin*, e 173 code smells, 8 destes são majors e outros 17 Critical.

No total, foram realizados 23 testes, porém a percentagem de coverage está a 0%. Suponho que isto se deva ao facto de o jacoco não estar devidamente implementado no projeto. Para esta implementação, o código adicionado ao pom.xml é o indicado pelo seguinte site: https://www.baeldung.com/sonarqube-jacoco-code-coverage.

A percentagem de código duplicado é 4.3%.

4 Referências e Recursos

Recursos

Recurso:	URL/localização:
Repositório no GitHub	https://github.com/joaoreis16/TQS_98474/tree/main/HW1
Vídeo de demonstração	https://www.youtube.com/watch?v=w-xc9dv0mkA (no repositório, o vídeo está na pasta *media*)

Referências

- API externa - VACCOVID - coronavirus, vaccine and treatment tracker API Documentation