deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

HW1: Mid-term assignment report

Eduardo Rocha Fernandes [98512], v2022-05-02

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.

This project is about a COVID application that provides details on COVID incidence data for a certain city. For that purpose, it was intended to have a single page application that allows the user to search for a city and get the data about that region. Also, it was supposed to have a REST API that receives data from an external API. It was as well planned to have multiple layer tests for the application.

My application is called **Covid Statistics** and allows the user to search for a specific city, or select a city from a country that the user already chose, and get the COVID details for that region.

1.2 Current limitations

The external API for some countries does not have cities to present to users. This means that the user can choose a country from the list but can not select a city related to that country. I didn't change the API because I do not think that this matters that much since there are tons of other countries on the list.

The main aspects of the project that I was not able to implement were:

- The logging strategy to produce evidence of the actions;
- I was not able to create several tests for CountryServiceImpl.

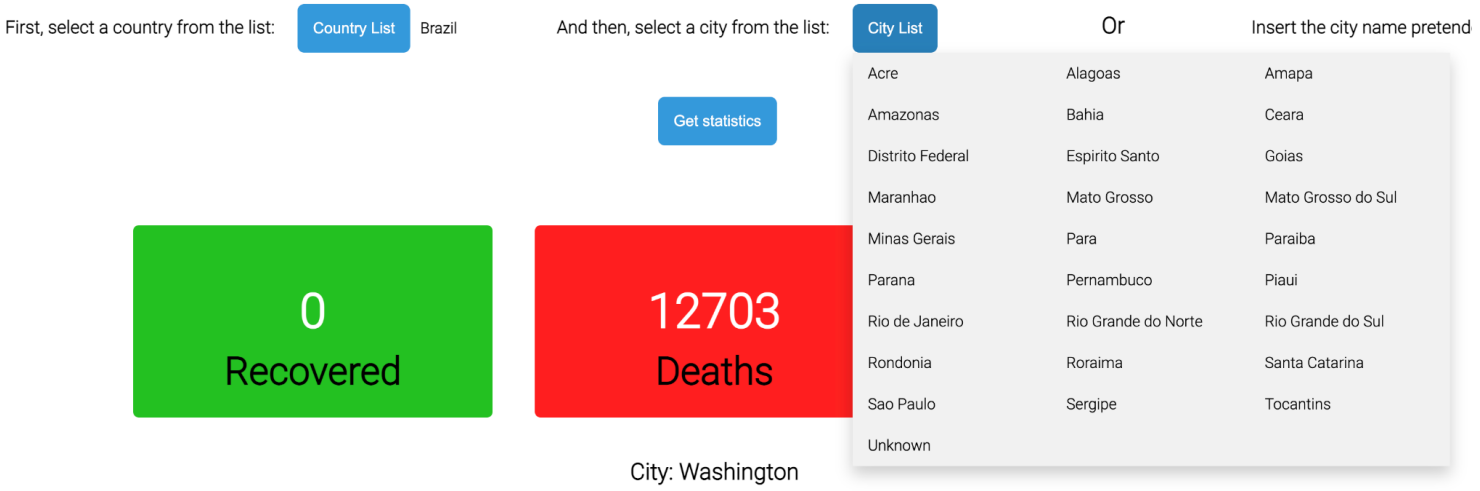
Both were not implemented because of the lack of time.

2 Product Specification

2.1 Functional scope and supported interaction

The user to search information about COVID in a certain city, just needs to input the city name on the input text box available. If the city name is correct it is presented to the user with some information about that region. Also the latitude and longitude are presented as well.

Also the user can select a city from a certain country. For that he selects a country from a list and then selects the city intended to see the COVID details for that city.

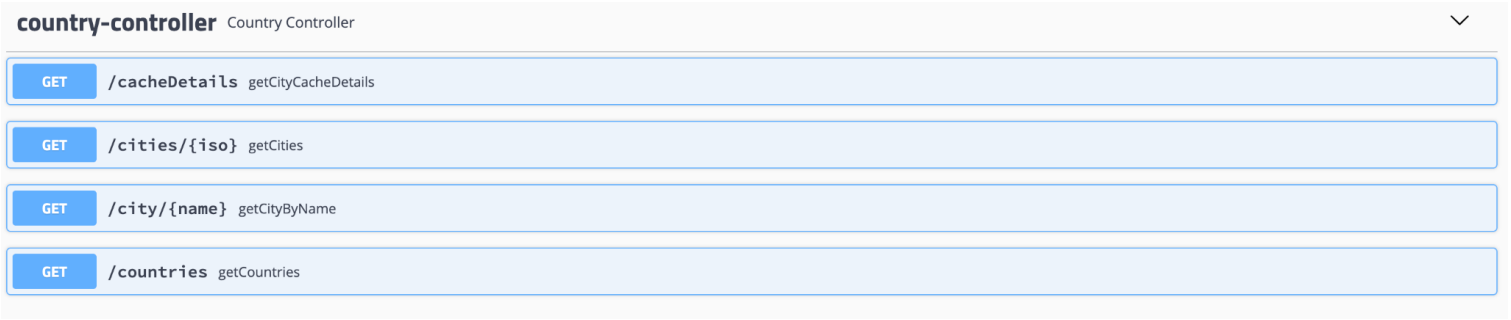


In this case, we are able to see that the user chose Brazil and then selected Rio de Janeiro.

2.2 System Architecture

2.3 API for developers

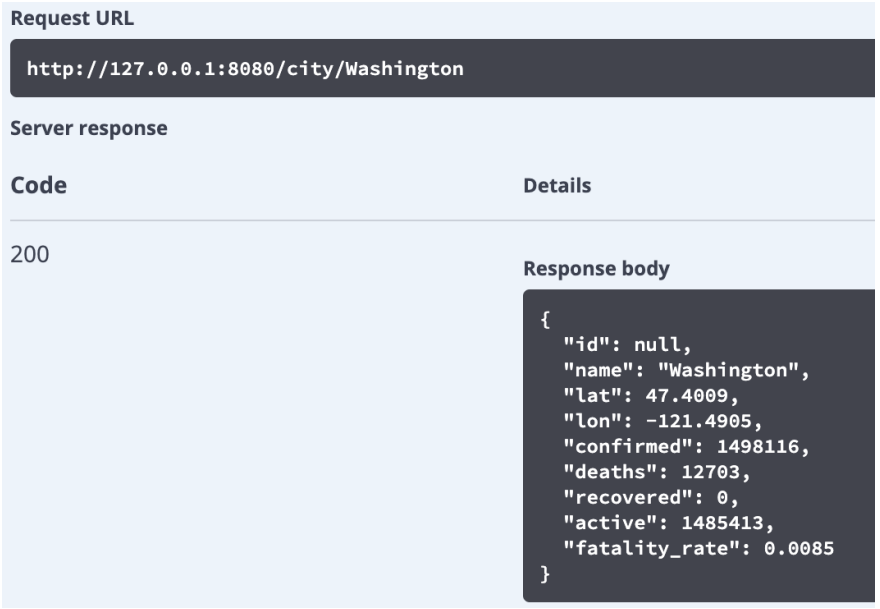
For the documentation of the project it was used *Swagger2.*
After the application is running, the documentation page can be accessed by the following link: [*127.0.0.1:8080/swagger-ui.html*](127.0.0.1:8080/swagger-ui.html).



- */cacheDetails* (returns all the requests, hits and misses)

- */cities/{iso}* (returns all the cities from the iso of a certain country)

- */city/{name}* (returns information about a specific city)



- */countries* (returns all the countries from the external API)

3 Quality assurance

3.1 Overall strategy for testing

After reading the homework proposal I already had an idea of the tests that I needed to implement. Although, I started developing the REST API first and also the webpage because they seemed to be pretty easy to do.

For service and integration tests, it was used *Mockito* and *SpringBoot MockMVC*. For unit tests, it was used *JUnit*, and for functional tests it was used *Selenium WebDriver*.

3.2 Unit and integration testing

Unit tests

For the unit tests, I started developing unit tests for *City.class*, as it is shown at the figure below.

```
public class CityTest {

    @Test
    public void tests() {
        City city = new City(name: "Paris", lat: 48.864716, lon: 2.349014, confirmed: 124154, deaths: 23145, recovered: 3412431, active: 423423, fatality_rate: 0.0012);

        assertEquals(expected: "Paris", city.getName());
        assertEquals(expected: 48.864716, city.getLat());
        assertEquals(expected: 2.349014, city.getLon());
        assertEquals(expected: 124154, city.getConfirmed());
        assertEquals(expected: 23145, city.getDeaths());
        assertEquals(expected: 3412431, city.getRecovered());
        assertEquals(expected: 423423, city.getActive());
        assertEquals(expected: 0.0012, city.getFatality_rate());
    }
```

Also some unit tests were made for *CityRepository.java.*

```java
@DataJpaTest
public class CityRepositoryTest {

    @Autowired
    private TestEntityManager testEntityManager;

    @Autowired
    private CityRepository cityRepository;

    @Test
    public void findByNameTest() {
        City aveiro = new City( name: "Aveiro");
        testEntityManager.persistAndFlush(aveiro);

        City city = cityRepository.findByName(aveiro.getName());
        assertThat(city).isEqualTo(aveiro);
    }

    @Test
    public void findAllTest() {
        City aveiro = new City( name: "Aveiro");
        City lisboa = new City( name: "Lisboa");
        City porto = new City( name: "Porto");
        City guarda = new City( name: "Guarda");
        testEntityManager.persist(aveiro);
        testEntityManager.persist(lisboa);
        testEntityManager.persist(porto);
        testEntityManager.persist(guarda);
        testEntityManager.flush();

        List<City> cities = cityRepository.findAll();
        assertThat(cities).hasSize( expected: 4).extracting(City::getName).contains(aveiro.getName(),
    }
```

Then, regarding unit tests with *Mocks*, the following tests were implemented.

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

    @Mock(lenient=true)
    private CityRepository cityRepository;

    @InjectMocks
    private CountryServiceImpl countryService;

    @BeforeEach
    public void startUp(){
        City paris = new City( name: "Paris");
        City aveiro = new City( name: "Aveiro");
        City porto = new City( name: "Porto");
        List<City> cities = new ArrayList<City>();
        cities.add(paris);
        cities.add(aveiro);
        cities.add(porto);

        Mockito.when(cityRepository.findByName(paris.getName())).thenReturn(paris);
        Mockito.when(cityRepository.findByName(aveiro.getName())).thenReturn(aveiro);
        Mockito.when(cityRepository.findByName(porto.getName())).thenReturn(porto);
        Mockito.when(cityRepository.findAll()).thenReturn(cities);
    }

    @Test
    public void cityTest() throws IOException, URISyntaxException {
        City city = countryService.getCityByName( name: "Washington");
        assertThat(city.getName()).isEqualTo( expected: "Washington");
    }
```

For the cache, there were implemented unit tests adding values to cache, verifying hits, misses and requests and also a test to check if the time-to-live policy is working.

```java
@Test
public void addValueToCacheTest(){
    //City Cache
    assertEquals(expected: 0, this.cityCache.getCacheSize());
    this.cityCache.addValue(key: "CityTest", new City(name: "CityTest"));
    assertEquals(expected: 1, this.cityCache.getCacheSize());
    assertEquals(expected: true, this.cityCache.containsCity(key: "CityTest"));
    // Country cache
    assertEquals(expected: 0, this.countryCache.getCacheSize());
    this.countryCache.addValue(key: "CountryTest", new ArrayList<>());
    assertEquals(expected: 1, this.countryCache.getCacheSize());
    assertEquals(expected: true, this.countryCache.containsCity(key: "CountryTest"));
}

@Test
public void cleanAfterTimeTest() throws InterruptedException{
    //City Cache
    City city = new City(name: "Aveiro");
    this.cityCache.addValue(key: "Aveiro", city);
    this.cityCache.cacheTimer(key: "Aveiro", time: 10);
    Thread.sleep(millis: 2000);
    assertEquals(expected: 0, this.cityCache.getCacheSize());
    //Country cache
    ArrayList country = new ArrayList<>();
    this.countryCache.addValue(key: "PRT", country);
    this.countryCache.cacheTimer(key: "PRT", time: 10);
    Thread.sleep(millis: 2000);
    assertEquals(expected: 0, this.countryCache.getCacheSize());
}

@Test
public void hitsMissesAndRequestsTest(){
    //City cache
    City city = new City(name: "Paris");
    this.cityCache.addValue(key: "Paris", city);
    this.cityCache.getCityFromCache(key: "Paris");
    this.cityCache.getCityFromCache(key: "Paris");
    this.cityCache.getCityFromCache(key: "null");
    assertEquals(expected: 3, this.cityCache.getRequests());
    assertEquals(expected: 2, this.cityCache.getHits());
    assertEquals(expected: 1, this.cityCache.getMisses());
    //Couuntry cache
    ArrayList country = new ArrayList<>();
    this.countryCache.addValue(key: "FR", country);
    this.countryCache.getArrayFromCache(key: "FR");
    this.countryCache.getArrayFromCache(key: "FR");
    this.countryCache.getArrayFromCache(key: "null");
    assertEquals(expected: 3, this.cityCache.getRequests());
    assertEquals(expected: 2, this.cityCache.getHits());
    assertEquals(expected: 1, this.cityCache.getMisses());
}
```

Integration tests

Regarding integration tests, the following tests were implemented on *CountryController*.

```java
@WebMvcTest(CountryController.class)
public class CountryControllerTest {

    @MockBean
    private CountryService countryService;

    @Autowired
    private MockMvc mockmvc;

    @Test
    public void getCityByNameTest() throws Exception{
        City paris = new City(name: "Paris", lat: 48.864716, lon: 2.349014, confirmed: 124154, deaths: 23145,

        when(countryService.getCityByName(name: "Paris")).thenReturn(paris);
        mockmvc.perform(get(urlTemplate: "/city/{name}", ...uriVars: "Paris").contentType(MediaType.APPLICAT
        .andExpect(status().isOk())
        .andExpect(jsonPath(expression: "$.name", is(value: "Paris")));
    }

    @Test
    public void getCountriesTest() throws Exception{
        HashMap<String, String> values = new HashMap<>();
        values.put(key: "Portugal", value: "PRT");
        values.put(key: "China", value: "CHN");

        when(countryService.getCountries()).thenReturn(values);
        mockmvc.perform(get(urlTemplate: "/countries").contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk()).andExpect(jsonPath(expression: "$.Portugal", is(value: "PRT")))
        .andExpect(jsonPath(expression: "$.China", is(value: "CHN")));
    }
@Test
public void getCitiesTest() throws Exception{
    ArrayList<String> values = new ArrayList<>();
    values.add(e: "Porto");
    values.add(e: "Aveiro");

    when(countryService.getCities(iso: "PRT")).thenReturn(values);
    mockmvc.perform(get(urlTemplate: "/cities/{iso}", ...uriVars: "PRT").contentType(MediaType.APPLICATION_JSON))
    .andExpect(jsonPath(expression: "$[0]", is(value: "Porto"))).andExpect(jsonPath(expression: "$[1]", is(value: "Aveiro")));

}

@Test
void getCacheDetailsTest() throws Exception {
    HashMap<String, HashMap<String, Integer>> cacheMap = new HashMap<>();
    HashMap<String, Integer> city = new HashMap<>();
    city.put(key: "hits", value: 2);
    city.put(key: "misses", value: 1);
    city.put(key: "requests", value: 3);
    HashMap<String, Integer> country = new HashMap<>();
    country.put(key: "hits", value: 1);
    country.put(key: "misses", value: 2);
    country.put(key: "requests", value: 3);
    cacheMap.put(key: "CityCache", city);
    cacheMap.put(key: "CountryCache", country);

    when(countryService.getCacheDetails()).thenReturn(cacheMap);
    mockmvc.perform(get(urlTemplate: "/cacheDetails").contentType(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath(expression: "$.CityCache", is(city)))
            .andExpect(jsonPath(expression: "$.CountryCache", is(country)));

}
```

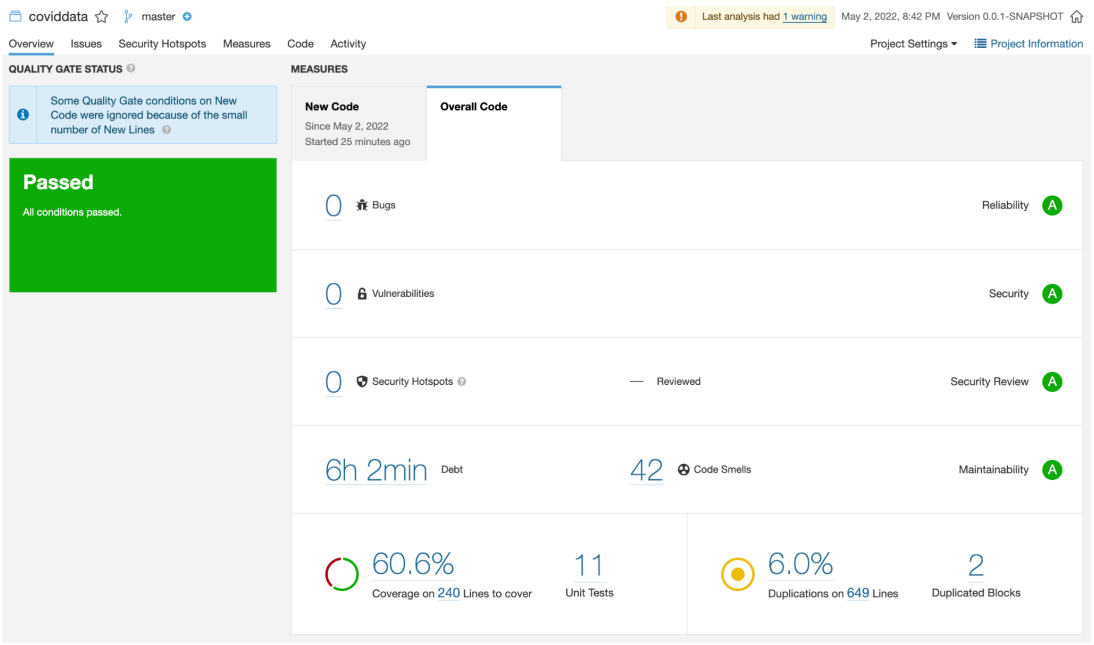
3.3 Functional testing

It was used *Selenium WebDriver* to do functional testing of the web application.
I only produced one test which searches for information about a specific city and verifies if the data presented to the user is correct.
Sometimes selenium couldn't find the html items and an error would occur. To fix this it was implemented a sleep so that the html item could appear before selenium clicking it.

```
public class WebTest {

    WebDriver browser;

    JavascriptExecutor js;

    @BeforeEach
    void startUp(){
        System.setProperty(key: "webdriver.chrome.driver", value: "/Users/eduardo/Universidade/3Ano/TQS/chromedriver");
        browser = new ChromeDriver();
        js = (JavascriptExecutor) browser;
    }

    @AfterEach
    void tearDown() {
        browser.close();
    }

    @Test
    public void searchCityByName() throws InterruptedException {
        browser.get(url: "http://127.0.0.1:8080/");
        js.executeScript(script: "window.scrollBy(0,750)");
        browser.manage().window().setSize(new Dimension(width: 1116, height: 697));
        browser.findElement(By.id(id: "input")).click();
        browser.findElement(By.id(id: "input")).sendKeys(...keysToSend: "Washington");
        browser.findElement(By.id(id: "submitBtn")).click();
        TimeUnit.SECONDS.sleep(timeout: 1);
        assertEquals(expected: "City: Washington", browser.findElement(By.id(id: "city-name")).getText());
    }
}
```

3.4 Code quality analysis

For static code analysis it was used *SonarQube*.



Some of the code smells are warnings to remove "*public*" tags from some files and I didn't remove them due to lack of time. The code coverage is a little low because of some tests that I was not able to perform as mentioned above.
This tool is really powerful since the first time that I tested my project with *SonarQube* I had some 2 bugs that I was able to correct.

4 References and resources

Project resources

- Git repository (informations in README.md): *https://github.com/rezeett/tqs_hw1*

References materials

- External API used: *https://rapidapi.com/axisbits-axisbits-default/api/covid-19-statistics/*