HW1: Mid-term assignment report

Diogo André Lopes Andrade [89265], v2020-04-15

1 Introduction 1
1.1 Overview of the work 1
1.2 Limitations 1

2 Product specification 2
2.1 Functional scope and supported interactions 2
2.2 System architecture 2
2.3 API for developers 2

3 Quality assurance 2
3.1 Overall strategy for testing 2
3.2 Unit and integration testing 3
3.3 Functional testing 3
3.4 Static code analysis 4

4 References & resources 4

1 Introduction

1.1 Overview of the work

O objetivo deste projeto é aplicar e compreender os conhecimentos teóricos sobre a qualidade de software, ou seja, desenvolver testes automatizados para verificar de forma sistemática uma aplicação Spring Boot.

Este projeto também tem como objetivo, preparar-nos para o ambiente de desenvolvimento de pequenas aplicações.

A aplicação apresenta uma visão dos valores de gases poluentes e outras características do tempo, de cada cidade. Estes dados são acedidos, através da pesquisa da cidade pretendida. Também tem uma página dedicada a apresentar a gestão da cache.

1.2 Limitations

Uma das funcionalidades que queria implementar, também recomendada a sua implementação pelo senhor professor era o acesso a dados da qualidade do ar em diversos dias e não apenas daquele momento. O motivo de não ter sido possível aplicar esta funcionalidade, foi não haver um método da API que permitisse escolher o dia que queríamos.

2 Product specification

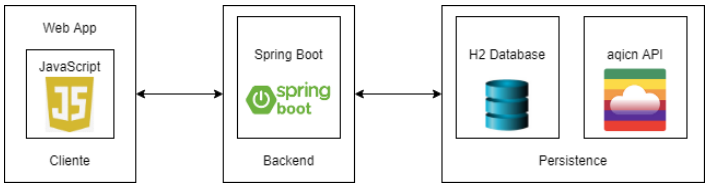
2.1 Functional scope and supported interactions

Os utilizadores foco a utilizar a minha aplicação, são todas as pessoas com acesso a um computador ou smartphone com acesso à internet e que tenham interesse no assunto e os institutos que estudam a qualidade do ar (se bem que os dados da API são disponibilizados por eles).

Uma pessoa pretende se deslocar para Bruxelas uma cidade bem movimentada, esta é a capital da Bélgica, sendo um país com uma poluição de ar mediana. Então essa pessoa, através do seu smartphone acede à minha web app e pesquisa como está a qualidade do ar em Bruxelas, de modo a saber se necessita levar uma máscara, para não inalar gases poluentes em demasia.

2.2 System architecture

A aplicação está dividida em três partes. A interface, uma página web, aonde o cliente consegue interagir com os dados, vindo da API e posteriormente guardados na base de dados (cache). Essa interação é mediada pela Spring Boot.



A template da aplicação usa Thymeleaf, Jquery e a framework Bootstrap.

A minha aplicação tem apenas uma entidade “City”. Esta entidade tem um id, igual ao que vem da aqicn API, para que não houvesse conflitos. Também tem outros atributos como nome da cidade, valores dos gases poluentes e de outras características do tempo, como humidade e temperatura.

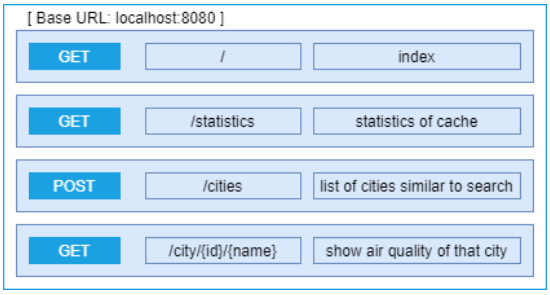
```
@Entity
public class City {

    @Id
    private Integer id;

    private String name;          private Double pm10;
    private Double dew;           private Double pm25;
    private Double humidity;      private Double so2;
    private Double no2;           private Double temperature;
    private Double o3;            private Double wind;
    private Double pressure;      private Double wg;
```

2.3 API for developers

A aplicação tem quatro páginas, sendo elas: uma simples página inicial (index), uma com dados da gestão da cache (statistics) e duas outras, sendo estas as fundamentais. Uma delas apresenta uma lista de cidades que correspondem à cidade pesquisada e a outra mostra os dados da qualidade do ar da cidade anteriormente selecionada.



3 Quality assurance

3.1 Overall strategy for testing

Os testes foram desenvolvidos após toda a aplicação estar quase completa. Pois no desenvolvimento deste projeto, sendo este o de uma micro aplicação, quase que ao ter as bases da aplicação feitas esta estará quase terminada.

3.2 Unit and integration testing

Os unit testes que foram feitos, são sobre as classes que correspondem a entidades. Estes testes verificam o bom funcionamento dos Getters e Setters das mesmas.

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class StatisticsTest {

    @Test
    public void testSaveStatistics() {
        Statistics statistics = new Statistics();
        statistics.setId(1);
        statistics.setCount_requests();
        statistics.setHits();
        statistics.setMisses();

        assertEquals(java.util.Optional.ofNullable(statistics.getId()), java.util.Optional.ofNullable(1));
        assertEquals(java.util.Optional.ofNullable(statistics.getCount_requests()), java.util.Optional.ofNullable(1));
        assertEquals(java.util.Optional.ofNullable(statistics.getHits()), java.util.Optional.ofNullable(1));
        assertEquals(java.util.Optional.ofNullable(statistics.getMisses()), java.util.Optional.ofNullable(1));
    }
}
```

3.3 Functional testing

Os testes funcionais foram feitos para todas as classes, que são repositórios, serviços e controllers. Estes testam todos ou quase todos os métodos dessas classes. Para testar a template usei o Selenium WebDriver.

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class CityControllerTest {

    @Autowired
    private CityController controller;

    @Test
    public void contexLoads() throws Exception {
        assertThat(controller).isNotNull();
    }
}
```

```java
@RunWith(SpringRunner.class)
@SpringBootTest
public class CityServiceTest {

    @Autowired
    private CityService cityService;

    @Test
    public void getCityDetails_returnsCityInfo() {
        City city = new City();
        city.setId(1);
        city.setName("Porto");
        cityService.saveCity(city);
        City getCity = cityService.getCityById(1);
        assertEquals(getCity.getName(), city.getName());
    }
}
```

```java
@RunWith(SpringRunner.class)
@SpringBootTest
@DataJpaTest
public class CityRepositoryTest {

    @Autowired
    private CityRepository cityRepository;

    @Autowired
    private TestEntityManager entityManager;

    @Test
    public void getCity_returnsCityDetails() {
        City city = new City();
        city.setId(1);
        city.setName("Aveiro");
        City savedcity = entityManager.persistAndFlush(city);

        City getCity = cityRepository.findById(1).orElse( other: null);

        Assertions.assertThat(getCity.getName()).isEqualTo(savedcity.getName());
    }
```

```java
public class WhereDataFromTest {

  private WebDriver driver;
  private Map<String, Object> vars;
  JavascriptExecutor js;

  @Before
  public void setUp() {
    driver = new ChromeDriver();
    js = (JavascriptExecutor) driver;
    vars = new HashMap<String, Object>();
  }

  @After
  public void tearDown() { driver.quit(); }

  @Test
  public void whereDataFrom() {
    driver.get("http://localhost:8080/");
    driver.findElement(By.id("search")).click();
    driver.findElement(By.id("search")).sendKeys( charSequences: "lisboa");
    driver.findElement(By.id("btn_search")).click();
    driver.findElement(By.linkText("Lisboa, Laranjeiro, Almada, Portugal")).click();
    assertThat(driver.findElement(By.id("data_from")).getText(), is( value: "Data From API! -> This data already in repository!"));
    driver.findElement(By.id("search")).click();
    driver.findElement(By.id("search")).sendKeys( charSequences: "lisboa");
    driver.findElement(By.id("btn_search")).click();
    driver.findElement(By.linkText("lisboa, Laranjeiro, Almada, Portugal")).click();
    assertThat(driver.findElement(By.id("data_from")).getText(), is( value: "Data From Repository!"));
    driver.close();
  }
}
```

```java
    @Test
    public void testSaveCity(){
        City city = new City();
        city.setId(2);
        city.setName("Lisboa");
        city.setDew(15.0);
        city.setHumidity(82.0);
        city.setNo2(13.9);
        city.setO3(14.3);
        city.setPressure(1020.0);
        assertNull(cityRepository.findById(city.getId()).orElse( other: null));
        cityRepository.save(city);
        assertNotNull(city.getId());
        City fetchedCity = cityRepository.findById(city.getId()).orElse( other: null);
        assertNotNull(fetchedCity);
        assertEquals(city.getId(), fetchedCity.getId());
        assertEquals(city.getName(), fetchedCity.getName());
        fetchedCity.setName("Olivais, Lisboa, Portugal");
        cityRepository.save(fetchedCity);
        City fetchedUpdatedCity = cityRepository.findById(fetchedCity.getId()).orElse( other: null);
        assertEquals(fetchedCity.getName(), fetchedUpdatedCity.getName());
        long cityCount = cityRepository.count();
        assertEquals(cityCount, actual: 1);
        Iterable<City> cities = cityRepository.findAll();
        int count = 0;
        for(City c : cities){
            count++;
        }
        assertEquals(count, actual: 1);
    }
}
```

3.4 Static code analysis

Não consegui fazer os testes de análise de código estático. Tentem usar o SonarQube para o maven, mas tive alguns problemas com as settings.

4 References & resources

Project resources

- Git repository: https://github.com/diogoandrade1999/TQS-Air-Quality
- Video demo: https://github.com/diogoandrade1999/TQS-Air-Quality/blob/master/presentation_video.mp4

Reference materials

- https://spring.io/guides/gs/testing-web/
- https://getbootstrap.com/
- https://www.thymeleaf.org/
- https://github.com/springframeworkguru/springbootwebapp
- https://aqicn.org/json-api/doc/

[Base URL: https://api.waqi.info/]

This API can be used to search stations by name

GET

```
/search/?keyword=:keyword&token=:token
```

Parameter

Field	Type	Description
keyword	String	Name of the station your are looking for (eg beijing, bulgaria, bangalore)
token	String	Your private API token (see aqicn.org/data-platform/token/)
callback	String	optional JSONP callback function name

This API can be used to get for the real-time Air Qualuty index for a given station.

GET

```
/feed/:city/?token=:token
```

Parameter

Field	Type	Description
city	String	Name of the city (eg beijing), or id (eg @7397)
token	String	Your private API token (see aqicn.org/data-platform/token/)
callback	String	optional JSONP callback function name