

HW1: Mid-term assignment report

Andre Filipe Moniz Morais [93236], v2021-05-14

1.1	Overview of the work	1
1.2	Current limitations	1
2.1	Functional scope and supported interactions	2
2.2	System architecture	2
2.3	API for developers	3
3.1	Overall strategy for testing	4
3.2	Unit and integration testing	4
3.3	Functional testing	6
3.4	Static code analysis	6

1 Introduction

1.1 Overview of the work

Este relatório descreve o projeto individual de TQS, testando conceitos de automação de testagem de software. O projeto consiste numa pequena página web onde o utilizador pode ter acesso a índices de qualidade do ar, bem como filtrar a sua visualização por data de forecast e por nome da localização a inspecionar. O website é composto por apenas 2 páginas: main (/) e logs (/logs). A main page dá ao utilizador acesso aos indicadores: o3 (ozono), pm10 (partículas com 10 micrometros ou menos), pm25 (partículas com 2.5 micrometros ou menos), uvi (radiação ultravioleta). A página /logs dá-nos informações sobre a quantidade de requests a cada uma das localizações, a quantidade de falhas (porque o utilizador pode procurar por uma região que não tenha um índice de poluição) e informações relativas ao tempo que os últimos 16 pedidos demoraram a ser processados

Quanto aos testes utilizados, o seu objetivo é testar o máximo possível o código escrito, nomeadamente, as classes que fazem a conversão de texto (formatado em json) para objeto. Mas também testar a implementação de novas funcionalidades, recorrendo à metodologia TDD (test-driven development).

1.2 Current limitations

Apesar de o meu website ser capaz de pesquisar por localização e por data, não permite a pesquisa por localização baseada em coordenadas ou selecionar um intervalo de tempo. Na verdade a API externa permite a seleção por coordenadas, mas não encontrei uma forma intuitiva de pedir ao utilizador que introduzisse as suas coordenadas, pelo que achei mais útil a seleção pelo nome.

Como os dados retornados pela API externa contêm a previsão de (+/-) uma semana nos vários indicadores, implementei um processamento dos dados para poder selecionar um dia específico (apenas da semana atual).

Os indicadores também estão limitados ao o3 (ozono), pm10 (partículas finas <10 micrometros), pm25 (partículas finas <2.5 micrometros), uvi (radiação UV).

2 Product specification

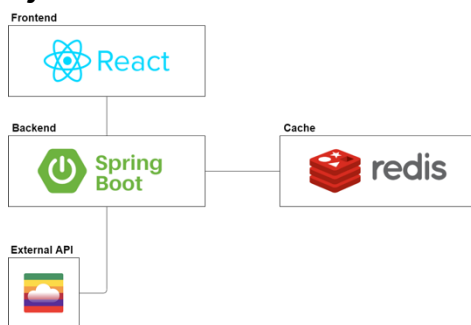
2.1 Functional scope and supported interactions

A interação com a aplicação é feita, no seu main scenario, por via de uma página web, onde o utilizador tem acesso a 5: os primeiros 4 correspondentes aos indicadores e seus prognósticos para os próximos dias. Aí o utilizador pode escolher entre introduzir o nome de uma localização (ex. Porto) e/ou selecionar uma data. Uma funcionalidade extra é indicação de sugestões ao preencher o campo de texto. Estas sugestões são baseadas em pesquisas feitas anteriormente no site, por qualquer utilizador.

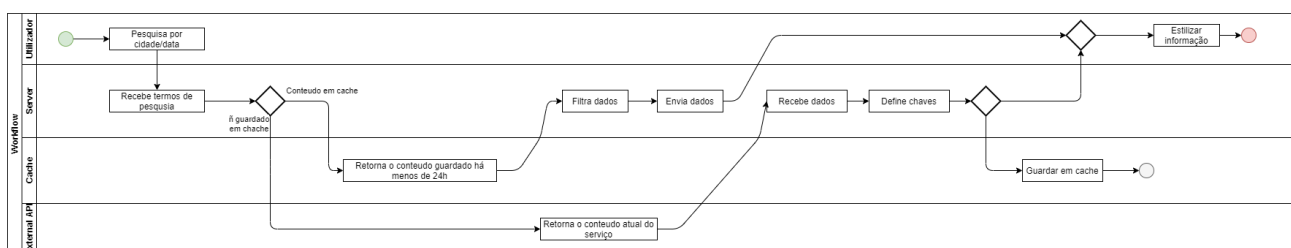
Apesar desta ser um dos cenários, o utilizador pode optar por ver estatísticas de requests por localização, cujas informações são os tuplos (cidade, nº de requests), (cidade, nº de erros) e lista dos últimos 16 tempos de processamento do servidor (aos pedidos dos utilizadores).

A par disto, se o utilizador souber como aproveitar uma api, então este também pode escolher aceder à informação crua via REST.

2.2 System architecture



A aplicação é composta por 4 grandes componentes: Frontend construída em cima do Framework ReactJs, Backend sob forma de REST API implementado com Spring boot, uma cache em memória desenvolvida com redis (com persistence), e uma API externa a que o servidor se conecta para poder obter os dados das cidades.



Aqui está representado o fluxo de dados quando o utilizador faz o pedido segundo uma localização. O server recebe os dados e verifica se tem em cache informação com menos de 24h. Acontece que tenho uma segunda verificação que se baseia na primeira data presente nas previsões. Isto porque, dos dados extraídos da API externa, a data da primeira previsão é sempre o dia atual da localização pedida [1]. Ou seja, mesmo que o ultimo pedido a uma dada localização tenha sido feito há menos de 24h, se o dia já se tiver alterado, então a cache é ignorada porque, à partida está errada. Por exemplo, às 23:00h de segunda feira faço um pedido sobre o “Porto”. Se na Terça feira às 01:00h voltar a fazer o mesmo pedido, então o resultado não vai ser o que está em cache, mesmo que ainda não tenham passado 24h, isto porque a API externa já tem dados mais recentes.

```
127.0.0.1:6379> keys *
1) "tokio"
2) "portugal_requests"
3) "tokiioo_requests"
4) "portugal"
5) "tokiioo_errors"
6) "tokio_requests"
7) "_elapsedTime"
8) "tokio_errors"
9) "porto_requests"
10) "tokiioo"
11) "braga_lastcheck"
12) "porto_lastcheck"
13) "porto"
14) "mexico"
15) "portugal_lastcheck"
```

Em adição, a estrutura do Redis é a seguinte.

Para cada cidade xyz:

- Key 'xyz' com o conteúdo da API externa
- Key 'xyz_requests' com o número de vezes que 'xyz' foi pesquisado
- Key 'xyz_lastcheck' com o timestamp em milissegundos da ultima vez que 'xyz' foi extraído da API externa
- Key 'xyz_errors' com o numero de vezes que a pesquisa por 'xyz' falhou.
- Existe ainda a key '_elapsedTime' que é um array com os tempos de processamento dos últimos 16 pedidos.

[1] – o que por vezes causa problemas porque em alguns países há diferença de horas, o que pode traduzir-se em pedidos sucessivos à API externa até que o dia mude nesse país.

2.3 API for developers




/api/v1/cities	/api/v1/city/name	/api/v1/city/name/date	/api/v1/logs
request	request	request	request
GET /api/v1/cities HTTP/1.1 Content-Type: application/json; charset=UTF-8 Host: localhost:8080	GET /api/v1/city/Porto HTTP/1.1 Content-Type: application/json; charset=UTF-8 Host: localhost:8080	GET /api/v1/city/porto/2021-05-17 HTTP/1.1 Content-Type: application/json; charset=UTF-8 Host: localhost:8080	GET /api/v1/logs HTTP/1.1 Content-Type: application/json; charset=UTF-8 Host: localhost:8080
response	response	response	response
HTTP/1.1 200 OK Vary: Origin Vary: Access-Control-Request-Method Vary: Access-Control-Request-Headers Content-Type: application/json Content-Length: 153	HTTP/1.1 200 OK Vary: Origin Vary: Access-Control-Request-Method Vary: Access-Control-Request-Headers Content-Type: application/json Content-Length: 2058	HTTP/1.1 200 OK Vary: Origin Vary: Access-Control-Request-Method Vary: Access-Control-Request-Headers Content-Type: application/json Content-Length: 1060	HTTP/1.1 200 OK Vary: Origin Vary: Access-Control-Request-Method Vary: Access-Control-Request-Headers Content-Type: application/json Content-Length: 615

Para esta solução criei 4 endpoints:

- `/api/v1/cities` – retorna uma lista com todas as cidades já pesquisadas no campo de texto
- `/api/v1/city/<name>` – para uma dada cidade/país, retorna informações como forecast, índices (e seus valores), localização da coleta dos dados, índice poluente dominante...
- `/api/v1/city/<name>/<date>` – semelhante ao endpoint anterior, mas retira do forecast todas as datas que não correspondam à solicitada
- `/api/v1/logs` – Retorna 3 arrays: requests, errors, elapsedTime, o primeiro com o nome das localizações e o numero de requests que foram feitos sobre essa localização; o segundo com o nome das localizações cujo pedido falhou e o numero de falhas; e o terceiro com uma lista de 16 valores correspondentes aos últimos tempos de execução de processamento dos pedidos.

3. Quality assurance

3.1 Overall strategy for testing

FEATURE search by date
 moraisandreu • 8h
INIT TDD
 moraisandreu • 14h
Update refactoring
 moraisandreu • 15h

A estratégia inicial (e na maioria do desenvolvimento do projeto) não foi a mais correta. Comecei por implementar as funcionalidades e só depois fazer o testes. Depois de uma troca de ideias com colegas, percebi que não tinha escolhido a melhor estratégia e decidi acrescentar mais uma funcionalidade, desta

vez seguindo as iterações corretas. As ferramentas utilizadas foram o Selenium e o REST-Assured, nomeadamente para testagem da API. Também recorri à testagem dinâmica da base de dados num container. Fora do contexto da API, experimentei utilizar um mecanismo de conversão de texto json em objeto, que passa pela criação de classes especializadas em cada um dos objetos json.

3.2 Unit and integration testing

UtilsUnitTests

```
@Test
void testCall_n_Parse() throws JsonProcessingException
```

Test Case: testar a `public class CityResponse{}` E suas dependentes às quais são atribuídos os valores do objeto json.

Method: o teste gera todo um objeto `CityResponse` a partir do método `ObjectMapper().readValue(utils.callAPI(), CityResponse.class).toString()` Que constrói as classes com base no valor retornado pelo `utils.callAPI()`. Isto é testado com uma instancia `expectedCR` criada à mão.

```
@Test
void testUtils_Validators(){
```

Test Case: testar a validade dos métodos auxiliares que permitem verificar se a cache deve ou não ser atualizada com base na quantidade de tempo que já passou. E se o nome da cidade é válido (usado para o autocomplete).

Method: são usados um `assertTrue` para cada método.

JedisUnitTests

```
@Test
void testGetKey(){}
```

Test Case: utilização de um container com redis e testar a obtenção e 2 chaves: “tokio” (cidade propositadamente mal escrita) e “braga_lastcheck”. As chaves devem conter, respetivamente, um erro e o valor da ultima vez que braga teve a sua localização pesquisada na API externa.

Method: 1 *assertEqual* para testar a igualdade do conteúdo da chave “tokio” e 2 *assertTrue* para testar a diferença entre tempos de pedidos pela chave “braga”. Se a diferença entre o tempo atual e o momento em que a pesquisa foi feita for superior a 3600000 milisegundos, então o valor em cache deve ser descartada.

```
@Test
void testParseJson_fromRedis() throws JsonProcessingException {}
```

TestCase: testar a extração do valor da chave “braga” a partir do container com redis, mapear o conteúdo da chave para a class CityResponse e validar se o objeto recebeu os dados corretamente.

Method: Obtém o valor presente no redis; mapeia o conteúdo para um objeto CityResponse; e verifica se essa instancia do objeto tem o valor AQI igual a 27.

WaqiApiTests

```
@Test
void whenCorrectPlace_returnAirQuality() throws JsonProcessingException {}
```

Test Cases: testar a utilização da classe e do acesso à API externa.

```
@Test
void whenWrongPlace_throwIncorrectName() {}
```

Test Cases: testar o resultado no caso de não haver informações disponíveis para uma dada cidade.

```
@Test
void whenGeo_returnAirQuality() throws JsonProcessingException {}
```

Test Cases: testar a utilização das coordenadas ao invés no nome da cidade no acesso à API externa. Apesar de não implementado o seu acesso, é possível notar que o endpoint funciona.

AirQualityApiTests

```
@Test
void whenStart_thenReturnCities() throws Exception {}
```

Test Cases: testar as cidades que aparecem no autocomplete. São adicionadas 2 cidades: mexico e tokio. Depois é verificada a dimensão do vetor retornado pelo mapping `/api/v1/cities`

```
@Test
void whenSearch_thenReturnCity() throws Exception {}
```

Test Cases: testar os dados provenientes da pesquisa por “Porto” no mapping `/api/v1/city/Porto`, que deverá ter como nome da localização de coleta de dados “Sobreiras-Lordelo do Ouro, Porto, Portugal”

```
@Test
void whenSearch_thenIncrementRequests() throws Exception {}
```

Test Cases: testar a incrementação de nº de requests na cidade a que se fez pesquisa. O valor guardado na base de dados deve ser ≥ 1

```
@Test
void whenSearchData_thenGetResults() throws Exception {}
```

Test Cases: testar a pesquisa por data num dado local. Para esse local, deve aparecer para todos os indicadores apenas um registo, correspondente ao da data selecionada.

```
@Test
void whenError_thenIncrementErrors() throws Exception {}
```

Test Cases: semelhante à incrementação de requests, mas desta vez testa-se a incrementação de erros numa localização sem indicadores.

```
@Test
void whenLogs_thenShowResults() throws Exception {}
```

Test Cases: testar as estatísticas extraídas do mapping `/api/v1/logs`, contendo, obrigatoriamente ≤ 17 tempos no array `elapsed_time`.

3.3 Functional testing

AirqualityHomePage

```
public void setSearchInput(String name) { driver.findElement(By.id("searchInput")).sendKeys(name); }

public void setEnter() { driver.findElement(By.id("searchInput")).sendKeys(Keys.ENTER); }

public String getLocationTitle() { return driver.findElement(By.id("locationTitle")).getText(); }
```

Test Case: Utilizador introduz “Porto” no campo de pesquisa, prime Enter e espera o resultado.

Method: Selenium preenche o campo de texto com “Porto”; prime Enter; dá 2 segundos de espera (porque constatei que, quando o servidor tinha de estabelecer conexão com a API externa, demorava cerca de 1.5s. Caso contrario, demorava apenas 15 milissegundos). E, por fim, verifica se o nome da localização de coleta da qualidade do ar corresponde à apresentada na página.

```
@Test
void testSearch() throws InterruptedException {
    AirqualityHomePage home = new AirqualityHomePage(driver);

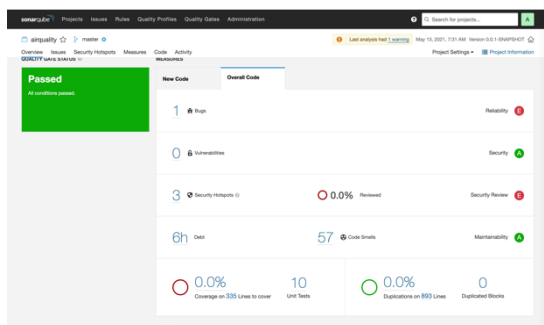
    home.setSearchInput("Porto");

    home.setEnter();

    TimeUnit.SECONDS.sleep( timeout: 2);

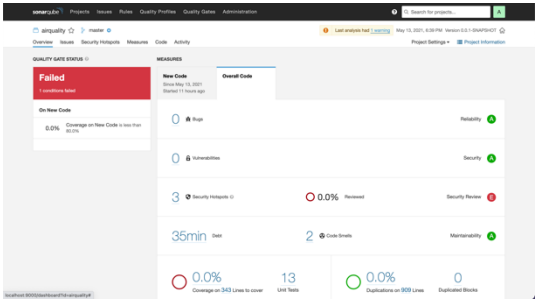
    assertThat(home.getLocationTitle(), is( value: "Sobreiras-Lordelo do Ouro, Porto, Portugal"));
}
```

3.4 Static code analysis



Para a análise código utilizei o Sonarqube, que me permitiu identificar inicialmente 57 code smells diferentes e 3 security spost no meu código. Há também uma referência a 0% de coverage dos testes ao código, contudo estes 10 testes [2] identificados na imagem percorriam todas as funções (utils e rest api) da aplicação. Muitos dos code smells eram repetidos, pelo que, muitas vezes, uma simples correção manifestava-se em 10+ code smells resolvidos.

Portanto, antes da adição da ultima feature (pesquisa por data) e seu teste, o painel do sonarqube encontrava-se com esta informação. Sendo que os code smells que ficaram por resolver têm prioridade reduzida e são irrelevantes para o código, porque em nada interferem com o seu funcionamento.



Um referente ao condicionamento da chamada de uma função e outra relacionada com a falta de um método @Override na classe do container do redis.

src/.../junit/jupiter/inheritance/RedisContainer.java

☐

Override the "equals" method in this class. Why is this an issue?

yesterday ▾ L7 🔗 🔍

☐

Code Smell ▾ Minor ▾ Open ▾ Not assigned ▾ 30min effort Comment

suspicious ▾

src/main/java/tqs/tp1/airquality/AirQualityApplication.java

☐

Invoke method(s) only conditionally. Why is this an issue?

13 days ago ▾ L37 🔗 🔍

☐

Code Smell ▾ Minor ▾ Open ▾ Not assigned ▾ 5min effort Comment

performance ▾

Mesmo com 13 testes identificados, o Sonarqube não considera nenhuma coverage, o que me levou a testar a coverage dos testes com o Jacoco.

tqs.tp1.airquality

Element	Missed Instructions ▾	Cov. ▾	Missed Branches ▾	Cov. ▾	Missed ▾	Cxty ▾	Missed ▾	Lines ▾	Missed ▾	Methods ▾	Missed ▾	Classes ▾
Utils	<div><div></div></div>	56%	<div><div></div></div>	54%	12	21	32	61	5	10	0	1
AirQualityResolver	<div><div></div></div>	64%	<div><div></div></div>	n/a	1	3	1	5	1	3	0	1
AirQualityApplication	<div><div></div></div>	78%	<div><div></div></div>	50%	2	7	4	13	1	6	0	1
AirQualityController	<div><div></div></div>	99%	<div><div></div></div>	86%	5	24	2	78	0	6	0	1
Total	143 of 808	82%	16 of 60	73%	20	55	39	157	7	25	0	4

[2] – entretanto já são 15 testes, com um maior cobertura das funcionalidades REST e ainda assim o Sonarqube atribui uma coverage de 0%

4. References & resources

Project resources

- Video demo [where did you place a short video demonstration of your solution; should be in the Git repository]
- Ready to use application: **[optional]**; if you have the solution deployed in a public server, place the URL here]
- QA dashboard: **[optional]**; if you have a quality dashboard available (e.g.: sonarcloud), place the URL here]

Reference materials

project, T., 2021. *Chất lượng không khí trên toàn thế giới: AirNet Sensor Network*. [online] aqicn.org. Available at: <<https://aqicn.org/api/>> [Accessed 14 May 2021].

Google Developers. 2021. *Charts* | *Google Developers*. [online] Available at: <<https://developers.google.com/chart>> [Accessed 14 May 2021].

EclEmma.org. 2021. *EclEmma - JaCoCo Java Code Coverage Library*. [online] Available at: <<https://www.eclEmma.org/jacoco/>> [Accessed 14 May 2021].

Sonarqube.org. 2021. *Code Quality and Code Security* | *SonarQube*. [online] Available at: <<https://www.sonarqube.org/>> [Accessed 14 May 2021].