deti universidade de aveiro
departamento de eletrónica, telecomunicações e informática

HW1: Mid-term assignment report

Frederico Campos de Avó [79900], v2021-04-27

1 Introdução **1**
1.1 Visão geral 1
1.2 Limitações 2

2 Especificações do produto **3**
2.1 Functional scope e interações suportadas 3
2.2 Arquitetura do sistema 3
2.3 API for developers 4

3 Garantia de qualidade **5**
3.1 Estratégia para a realização de testes 5
3.2 Testes unitários e de integração 6
3.3 Testes funcionais 8
3.4 Codacy 9

4 Referências e recursos **10**

1 Introdução

1.1 Visão geral

O objetivo deste projeto é disponibilizar um serviço que permita realizar a consulta sobre qualidade do ar para uma determinada cidade.Cada cidade tem associada uma estação que irá medir os valores relativos à qualidade do ar.
O projeto foi desenvolvido utilizando **SpringBoot** e os dados são obtidos da API **"AQICN"**.

As estações disponíveis para consulta são guardadas na **cache**.
Cada vez que o utilizador pretende consultar os valores da qualidade do ar para uma determinada estação com uma cidade associada a partir do website desenvolvido, a cache é acessada para que o sistema verifique se esse pedido já foi realizado anteriormente. A cache tem uma variável que define o tempo em Unix Timestamp que determina se o registo é válido ou inválido, sendo que é inválido se o tempo de registo for superior a 10 minutos.
No caso de ser inválido ou não existir é feito um pedido à API **"AQICN"**, e é incrementado o valor de Miss. No caso de existir e ser válido é retornado o registo pretendido e

incrementado o valor de Hits.No momento em que é feita a chamada à API **“AQICN”**, é iniciado o processo de armazenamento.


```
{
   status: "ok",
   data: {
      aqi: 70,
      time: {
         s: "2021-05-12 09:00:00"
      },
      city: {
         name: "Shanghai",
         url: "http://aqicn.org/city/shanghai/",
         geo: [
            "31.2047372",
            "121.4489017"
         ]
      },
      iaqi: {
         pm25: "..."
      }
   }
}
```


Figura1- **Forma como a API devolve os dados**

1.2 Limitações

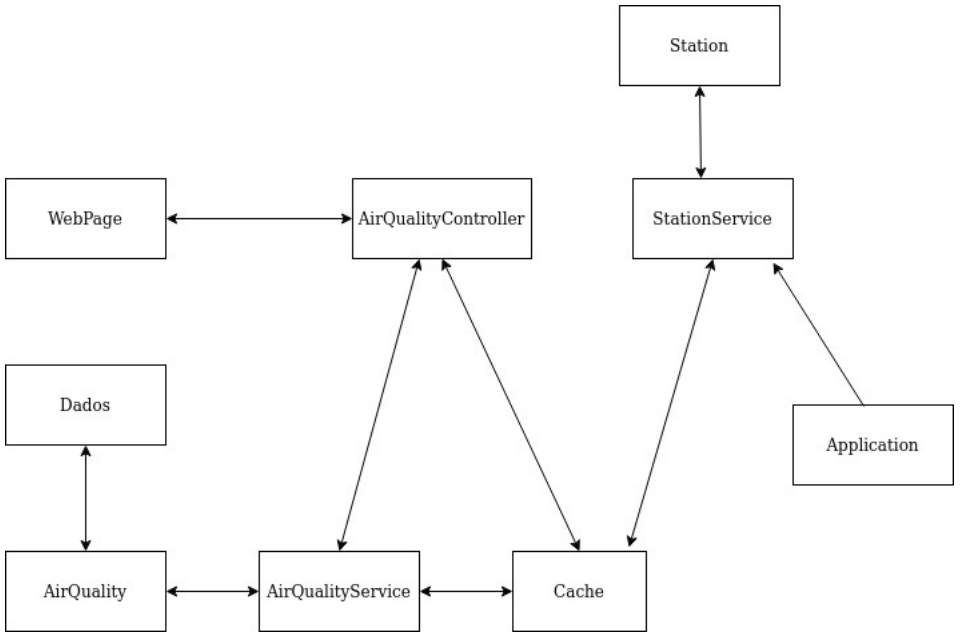
Uma das limitações do meu projeto é que apenas forneço valores de qualidade do ar apenas para 30 cidades do mundo.
Outra limitação é o tempo de resposta que a minha página web demora para carregar os valores para cada cidade.

2 Especificações do produto

2.1 Functional scope e interações suportadas

Esta aplicação destina-se a utilizadores que pretendam consultar a qualidade do ar para uma determinada cidade. Para isso necessitam apenas de aceder à página localhost:8080, e escolher a cidade que pretendem analisar.

2.2 Arquitetura do sistema



•**AirQuality**- Valores retornados pela API “AQICN” . Cada objeto “AirQuality” tem um status , um timestamp (Unix) registado quando é feita a chamada à API e um objeto “Dados” , com vários valores acerca do ar;

•**AirQualityController**- Controller da API desenvolvida;

•**Cache-** Na cache são definidas cinco variáveis:um objeto Map para guardar os dados relativos ao AirQuality, um objeto Map para guardar os dados relativos às Stations, um inteiro para guardar o número de requests, um inteiro para guardar o número de acessos com sucesso à cache e um inteiro para guardar o número de acessos sem sucesso à cache;

•**Dados-** Possui um inteiro associado correspondente ao Air Quality Index e um HashMap para representar os restantes valores recolhidos da API. O HashMap é do tipo <String,HashMap<String,Integer>>, uma vez que a própria API retorna um HashMap neste formato (por exemplo, { “pm25” : {“v” : 61} }), como é visto na figura 1.

•**Station-** Possui um id que a identifica e a cidade associada;

•**StationService**- Responsável pelas interações com a cache, relativamente a objectos relacionados com o tipo Station;

•**AirQualityService**- Responsável pelas interações com a cache, relativamente a objectos relacionados com o tipo AirQuality;

•**Application**- Classe predefinida do “SpringBoot”. São criadas todas as estações pretendidas e guardadas na cache.

•**WebPage**- Desenvolvido em HTLM e javascript onde é possível escolher uma cidade e obter informações acerca do ar dessa mesma cidade.

2.3 API for developers

A API desenvolvida permite:

- Consultar os valores de qualidade do ar por cada cidade;

```
GET http://localhost:8080/api/air/{{city}}
```

- Consultar as estações disponíveis;

```
GET http://localhost:8080/api/stations
```

- Verificar se uma determinada cidade tem uma estação disponível;

```
GET http://localhost:8080/api/station/{{city}}
```

-Consultar o número de acessos ;

```
GET http://localhost:8080/api/requests
```

-Consultar falhas e sucessos ;

```
GET http://localhost:8080/api/stats
```

3 Garantia de qualidade

3.1 Estratégia para a realização de testes

Usei a estratégia **TDD** (Test Driven Development). Esta estratégia baseia-se em pequenos ciclos de repetições, onde para cada funcionalidade do sistema, é criado um teste antes. Este novo teste criado inicialmente falha, já que ainda não temos a implementação da funcionalidade em questão e, em seguida, implementamos a funcionalidade para o teste passar.

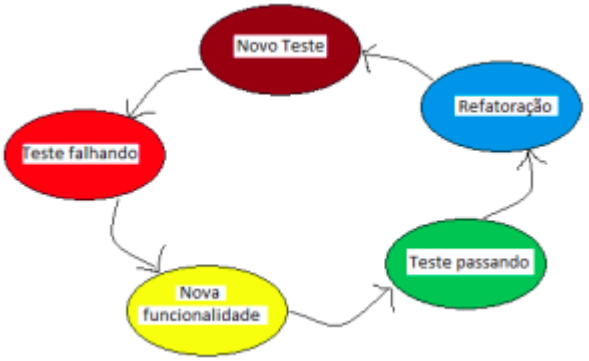


Figura1- Ciclo de desenvolvimento do TDD

Como é perceptível na figura acima, escrevemos um teste que inicialmente não passa, adicionamos a nova funcionalidade ao sistema, fazemos com que o teste passe, é feito "refactoring" do código da nova funcionalidade e por fim escrevemos o próximo teste.

3.2 Testes unitários e de integração

Realizei testes **unitários** utilizando a ferramenta **"jUnit"** para testar a Cache. Estes testes unitários testam todos os métodos que foram implementados.

Teste setAirQuality

Guardei dois objetos AirQuality na cache. Após guardar os objetos, verifiquei se o tamanho do objeto que guarda este tipo de objectos tinha aumentado duas unidades e se os valores do tamanho antes e depois de guardar eram diferentes.

Teste getAirQuality

Guardei um objecto, retornei-o e por fim, verifiquei que o tamanho foi alterado e que o objeto retornado era o esperado.

Teste countRequests

Este teste testa se a contagem do número de requests é incrementada após um get.

Para isso registei o valor atual dos requests. Depois invoquei o método **getStations** e guardei o novo valor dos requests.
Depois disto o valor dos requests são diferentes, com diferença de uma unidade entre o novo valor e o antigo.

Test isValid

Registei dois valores iguais de AirQuality para duas cidades. Para uma associei o tempo atual e para a outra registei um tempo igual a 0. Assim, o registo da primeira cidade será válido e o da segunda não, uma vez que o tempo máximo está definido como sendo de 10 minutos (600000 ms).

Teste getCitiesAvailable

Registei duas Stations para as cidades, com id 6 e 7, respectivamente. De seguida, invoco o método a ser testado, que retorna um ArrayList, e assim verifico se as cidades disponíveis são guardadas corretamente.

Teste HitAndMiss

Increintei o valor de hit e miss na cache e verifico se a soma destes valores é a esperada e se o valor de miss e hit estão corretos.

Teste getStationByID

Registei uma station com id 100 associado à cidade de Aveiro e logo a seguir criei um id errado. Chamei o método getStationByID e verifiquei se o objeto retornado tem os valores que foram inseridos, e se, invocando esse método com o id errado, o objeto retornado correspondia a null

Teste getAirQualityByCity

Criei um objecto “Dados”. Criei também um objeto AirQuality e guardo-o na cache. Após guardar o objecto referido, invoco o método getAirQualityByCity com o argumento correspondente à cidade associada ao objecto AirQuality guardado. Testei este método verificando se os atributos do objeto retornado são iguais aos do objeto esperado.

Relativamente aos testes de integração utilizei **MockMvc**. Os testes de integração foram efetuados na classe “**AirQualityControllerTest**”.

Teste getSpecificStation

Verifiquei se o status corresponde a “ok” e se contém todas as cidades disponíveis, individualmente.

Teste GetStats

Retorna todas as estatísticas e verifica se hit e miss não são null.

Teste SpecificStationWrongCity

Verifica que não é encontrada nenhuma station para uma cidade não definida.

Teste GetStations

Verifica se todas as estações estão disponíveis.

3.3 Testes funcionais

Utilizei o Selenium, aplicado ao Firefox. Para isso utilizei o “geckodriver”. Comecei por tentar obter os valores para cada cidade disponível. Para além deste teste, verifiquei também se cada elemento que existe no ficheiro html se encontra na página inicial.

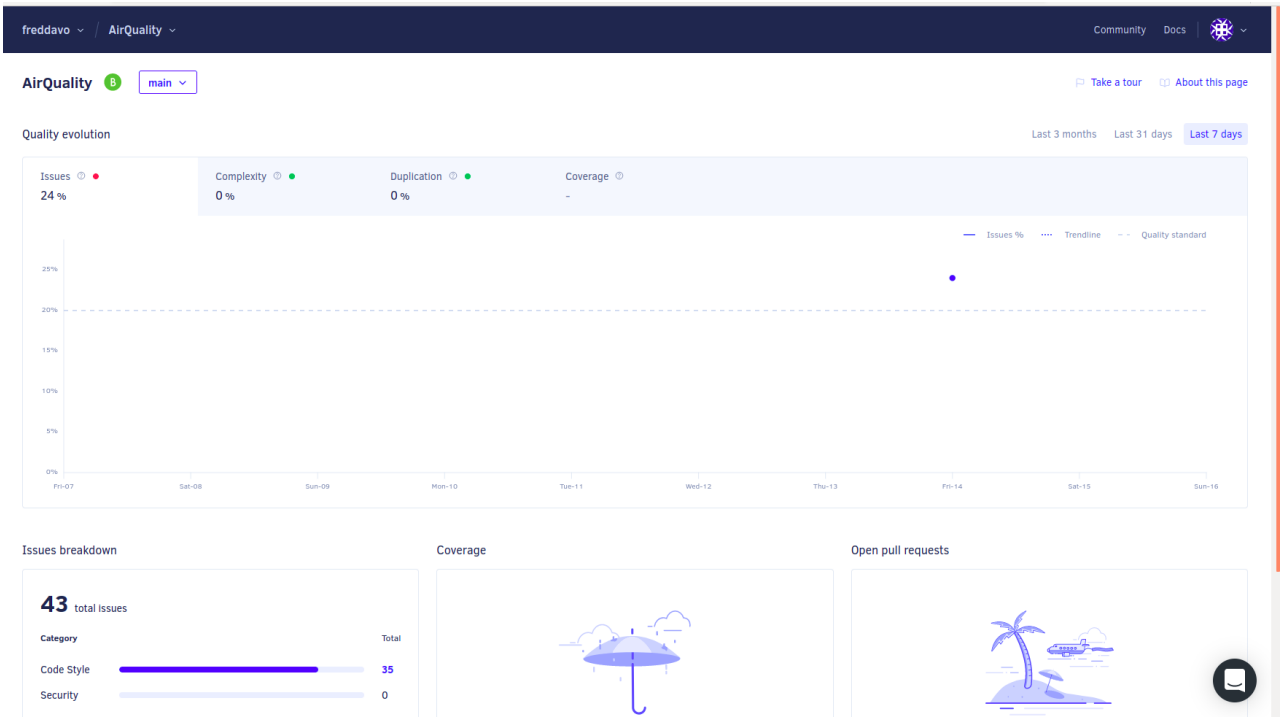
Com o teste **"test Possible Cities”** verifico se é possível consultar dados para todas as cidades pŕe definidas.

Com o teste **“testElementPresence”** verifico se todos elementos da página html estão disponíveis.

3.4 Análise de código- Codacy

Para analisar o código desenvolvido utilizei o Codacy.
O Codacy automatiza revisões de código e monitora a qualidade do código em cada solicitação de confirmação e por pull. Ele relata o impacto de cada solicitação de commit ou pull em novos problemas relacionados ao estilo de código, melhores práticas, segurança e muitos outros.

Permitiu detectar algumas falhas no código que não teria detectado sem consultar os resultados mostrados, tais como: alguns importados não utilizados, “pontos e vŕgulas” que me tinha esquecido de apagar e etc.




freddavo
AirQuality
Community
Search file
GRADE
FILENAME
ISSUES
DUPLICATION
B AirQuality/src/test/java/airquality/CacheTest.java 11 0
B AirQuality/src/test/java/airquality/AirQualityServiceTest.java 11 0
C AirQuality/src/test/java/airquality/SeleniumTest.java 6 0
B AirQuality/src/.../airquality/AirQualityControllerTest.java 6 0
C AirQuality/src/test/java/airquality/StationServiceTest.java 5 0
B AirQuality/src/main/java/airquality/AirQualityController.java 2 0
A AirQuality/.mvn/wrapper/MavenWrapperDownloader.java 2 0
A AirQuality/src/main/java/airquality/AirQualityService.java 0 0
A AirQuality/src/test/java/airquality/AplicationTest.java 0 0
A AirQuality/src/main/java/airquality/StationService.java 0 0
A AirQuality/src/main/java/airquality/AirQuality.java 0 0
A README.md 0 -
A AirQuality/src/main/java/airquality/Station.java 0 0
A AirQuality/src/main/java/airquality/Application.java 0 0
A AirQuality/src/main/java/airquality/Cache.java 0 0
A AirQuality/pom.xml 0 -
A AirQuality/src/main/resources/static/index.html 0 -
A AirQuality/src/main/java/airquality/Dados.java 0 0
Previous
Next

4 Referências e recursos

Project resources

- A demonstração em vídeo encontra-se no repositório “AirQuality” no github, na pasta vídeos;
- Repositório Git: https://github.com/freddavo/AirQuality
- Codacay: https://app.codacy.com/gh/freddavo/AirQuality/dashboard

Reference materials

https://aqicn.org/api/pt/

https://spring.io/guides/gs/testing-web/

https://www.softwaretestinghelp.com/geckodriver-selenium-tutorial/

https://www.springboottutorial.com/spring-boot-unit-testing-and-mocking-with-mockito-and-junit