

Projeto intermédio TQS

Ilídio Fernando de Castro Oliveira

Aeolus

Orlando Macedo 94521

DETI
Universidade de Aveiro
03-04-2021

Contents

1	Introdução	2
1.1	Vista geral do trabalho	2
1.2	Limitações atuais	2
2	Especificações do produto	3
2.1	Aspetos funcionais e interações suportadas	3
2.2	Arquitetura do sistema	4
2.3	API para desenvolvedores	5
3	Garantia de qualidade	8
3.1	Estratégia geral de teste	8
3.2	Teste unitários e de integração	8
3.2.1	Cache	8
3.2.2	Api's externas	9
3.2.3	Utils	9
3.2.4	Serviços	9
3.2.5	REST API	10
3.2.6	Aplicação Web	10
3.3	Análise estática de código	11
3.4	Metodologias de integração contínua de código	13
4	Recursos	14

1 Introdução

1.1 Vista geral do trabalho

A aplicação desenvolvida tem como nome *Aeolus*, pois, assim como o deus da mitologia Grega, nós também iremos ser capazes de dominar os ventos, ou pelo menos, saber o que eles nos trazem.

O que foi implementado, tem como principal objetivo disponibilizar medidas relativas a substâncias poluentes. Para tal, tira partido de API's que disponibilizam informações fidedignas.

As substâncias poluentes em questão são 5, entre elas podemos enumerar o dióxido de azoto (NO₂), dióxido de enxofre (SO₂), monóxido de carbono (CO), ozono (O₃) e partículas inaláveis (PM₁₀). Todas estas substâncias são medidas em µg/m³.

1.2 Limitações atuais

Em termos de limitações atuais, teria sido bastante benéfico utilizar uma aproximação orientada ao comportamento (**BDD**) para testar não só o comportamento da aplicação web (aquando do uso de *Selenium*), como também para testar em conjunto com *RestTemplate*, aquando o teste dos endpoints associados à **REST API**. Tal não foi possível devido a certas incompatibilidades com a dependência do *Cucumber*, e também devido à falta de tempo, mas futuramente será algo que merecerá novamente atenção.

Em segundo lugar, usar a extensão do **Swagger** era um dos objetivos para disponibilizar informação sobre a **REST API**, tal não foi possível na entrega até à data. Futuramente, pretendo incorporar o swagger na aplicação.

Por último, não foi possível implementar no *frontend* dois casos de estudo em que seria permitido que o utilizador pesquisasse medições desde um dia em específico até ao presente. E, também verificar medições num intervalo de tempo estabelecido. Apesar de não estarem presentes no backend, estão disponíveis na *REST API*.

2 Especificações do produto

functional (black-box) description of the application: how will the actors interact the application? For what? (explain the main scenarios of usage);

2.1 Aspetos funcionais e interações suportadas

Os utilizadores têm duas formas de interagir com o sistema. Podem usufruir da aplicação web, ou então utilizar a REST API, que disponibiliza as mesmas informações que a aplicação web e mais algumas que irão ser referidas de seguida.

Na aplicação web, o primeiro passo que o utilizador deve dar é seleccionar a localização que pretender. De seguida, pode verificar quais são as substâncias poluidoras, de momento, na área pretendida. Em contraste, pode também verificar o histórico de medições, escolhendo o número de dias que for do seu interesse.

A REST API, disponibiliza estas duas funcionalidades e ainda mais duas. Uma delas, referente à possibilidade de recolher medições desde uma data em específico. A outra, referente à possibilidade de escolher um intervalo de tempo, e verificar as medições feitas nesse intervalo de tempo estipulado.

Para além das funcionalidades referidas, a *api* ainda permite operações de verificação na **cache**, que está associada à aplicação. De seguida expõem-se as várias funcionalidades.

- verificar se a *cache* tem uma certa localização guardada
- verificar quais são as diferentes localizações guardadas em *cache*
- analisar se a mesma está cheia
- examinar qual é a localização que é alvo de mais pedidos
- retornar a percentagem de pedidos respondidos com sucesso pela *cache*
- verificar a totalidade dos pedidos efetuados
- analisar o número de pedidos respondidos com sucesso pela *cache*
- examinar quantos pedidos foram feitos para uma localização específica
- verificar qual o tamanho atual da *cache*
- retornar o tamanho máximo da *cache*

2.2 Arquitetura do sistema

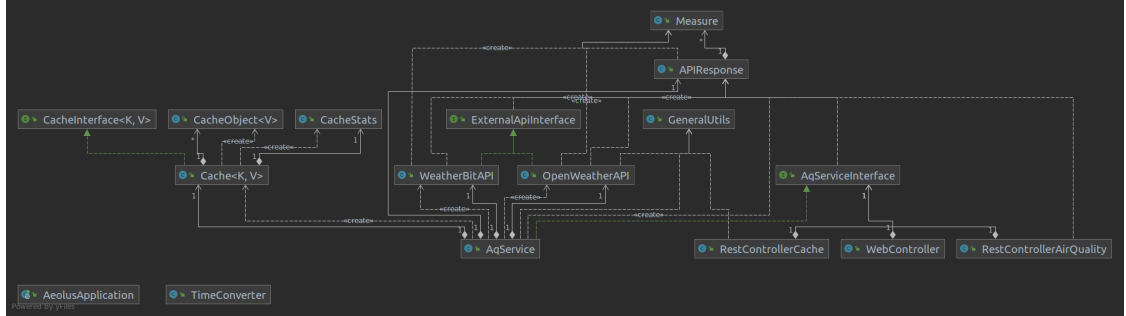


Figure 1: Diagrama de classes

Portanto, como se pode verificar no diagrama, existem 3 módulos essenciais para a disponibilização de serviços.

Em primeiro lugar, existe a cache que guarda pedidos de localizações recorrentes. A mesma foi implementada de forma a ter uma *thread* que verifica quando foi a última vez que a localização foi pedida, e eliminando-a caso já tenha passado o respetivo *time to live*. Esta cache é consumida na classe de serviços (*AQService*).

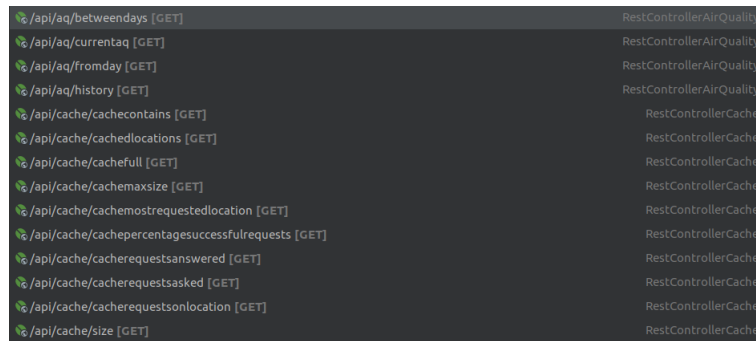
Em segundo lugar, dois módulos referentes às duas API's usadas. Ambas tiveram de obedecer a uma interface comum (*ExternalAPIInterface*), de forma a disponibilizarem as mesmas funcionalidades, apenas com implementações diferentes.

Estes 3 módulos são usados em conjunto na classe *AQService*, o que permite não só que os pedidos sejam respondidos mais rapidamente, com o uso da cache, mas também respondidos com maior fiabilidade, devido ao uso de duas API's. No caso da API principal falhar (*OpenWeather*), logo se tenta com a API de reserva (*WeatherBit*).

Portanto, a forma como se responde a um pedido numa determinada localização, é, em primeira instância, usando a cache. Se a localização pretendida não se encontrar guardada em *cache*, então faz-se um pedido à *api* principal. E finalmente, caso esta falhe, utiliza-se a de reserva. No final de haver um pedido bem sucedido à *api* externa, o resultado do mesmo é guardado em cache para que possa ser usado futuramente.

Para finalizar, tanto a REST API como a aplicação web usufruem dos serviços disponibilizados pela interface *AQServiceInterface*. Serviços esse que são implementados pela classe *AQService*.

2.3 API para desenvolvedores



<code>% /api/aq/betweendays [GET]</code>	RestControllerAirQuality
<code>% /api/aq/currentaq [GET]</code>	RestControllerAirQuality
<code>% /api/aq/fromday [GET]</code>	RestControllerAirQuality
<code>% /api/aq/history [GET]</code>	RestControllerAirQuality
<code>% /api/cache/cachecontains [GET]</code>	RestControllerCache
<code>% /api/cache/cachedlocations [GET]</code>	RestControllerCache
<code>% /api/cache/cachefull [GET]</code>	RestControllerCache
<code>% /api/cache/cachemaxsize [GET]</code>	RestControllerCache
<code>% /api/cache/cachemostrequestedlocation [GET]</code>	RestControllerCache
<code>% /api/cache/cachepercentagesuccessfulrequests [GET]</code>	RestControllerCache
<code>% /api/cache/cacherequestsanswered [GET]</code>	RestControllerCache
<code>% /api/cache/cacherequestsasked [GET]</code>	RestControllerCache
<code>% /api/cache/cacherequestslocation [GET]</code>	RestControllerCache
<code>% /api/cache/size [GET]</code>	RestControllerCache

Figure 2: Endpoints REST API

Os primeiros quatro *endpoints* são referentes a funcionalidades referentes à disponibilização de medições de elementos poluentes. Os 10 *endpoints* seguintes estão associados a operações sobre a *cache*.

De seguida são disponibilizadas imagens com os parâmetros associados a cada um dos 4 *endpoints* iniciais.

```
public ApiResponse getCurrentAirQuality(@RequestParam String lat,
                                       @RequestParam String lng)
```

Figure 3: Verificar elementos poluentes atuais

```
public ApiResponse getHistoryAirQuality(@RequestParam String lat,
                                       @RequestParam String lng,
                                       @RequestParam Integer days)
```

Figure 4: Verificar histórico de elementos poluentes

```
public ApiResponse getHistoryAirQualityFromDay(@RequestParam String lat,
                                              @RequestParam String lng,
                                              @RequestParam Integer year,
                                              @RequestParam Integer month,
                                              @RequestParam Integer day,
                                              @RequestParam Integer hour)
```

Figure 5: Verificar histórico de elementos poluentes, usando data específica

```
public ApiResponse getHistoryAirQualityBetweenDays(@RequestParam String lat,
                                                  @RequestParam String lng,
                                                  @RequestParam Integer initialYear,
                                                  @RequestParam Integer initialMonth,
                                                  @RequestParam Integer initialDay,
                                                  @RequestParam Integer initialHour,
                                                  @RequestParam Integer endYear,
                                                  @RequestParam Integer endMonth,
                                                  @RequestParam Integer endDay,
                                                  @RequestParam Integer endHour)
```

Figure 6: Verificar histórico de elementos poluentes, num intervalo de tempo

De seguida encontra-se a informação dos *endpoints* associados à *cache*.

```
public Boolean getCacheContainsLocation(@RequestParam String lat,  
                                       @RequestParam String lng)
```

Figure 7: Verificar se cache contém localização

```
public Set<ImmutablePair<String, String>> getCachedLocations()
```

Figure 8: Retornar localizações guardadas em cache

```
public Boolean getCacheIsFull()
```

Figure 9: Comprovar se a cache está cheia

```
public Integer getCacheMaxSize()
```

Figure 10: Tamanho máximo da cache

```
public ImmutablePair<String, String> getCacheMostRequestedLocation()
```

Figure 11: Retornar a localização mais requisitada

```
public Double getCachePercentageSuccessfulRequests()
```

Figure 12: Percentagem de pedidos satisfeitos

```
public Long getCacheRequestsAnswered()
```

Figure 13: Pedidos respondidos com sucesso

```
public Long getCacheRequestsAsked()
```

Figure 14: Totalidade de pedidos efetuados

```
public Long getCacheRequestsToLocation(@RequestParam String lat,  
                                       @RequestParam String lng)
```

Figure 15: Verificar pedidos para uma determinada localização

```
public Integer getCacheCurrentSize()
```

Figure 16: Retornar tamanho atual da cache

3 Garantia de qualidade

[which test cases did you considered? How were they implemented?] [may add some screenshots/code snippets for clarification]

3.1 Estratégia geral de teste

De uma forma geral pode dizer-se que foi utilizada uma metodologia *TDD*. A forma como o projeto se foi concretizando foi usando os testes em conjunto com a implementação. De uma forma muito concreta aconteceu o seguinte, no *IDE*, num separador estava a classe de teste, num outro separador (usando o mesmo ecrã), estava a classe de implementação.

A corrente que se seguiu foi, a partir das interfaces criavam-se as classes de teste, sem que houvesse implementação nas classes que implementavam a interface. Portanto, tendo dum lado a classe de teste, do outro a classe de implementação, começava-se por escrever o teste com o comportamento que seria de esperar para um determinado método ou funcionalidade. E, só de seguida é que se seguia para a implementação.

Um aparte para dizer que esta metodologia é muito útil porque o *debugging* das funcionalidades torna-se muito mais fácil. Os erros ocorrem num espaço de código muito contido o que permite uma melhor visualização dos mesmos, e consequentemente, a resolução desses bugs também se torna significativamente mais acessível.

3.2 Teste unitários e de integração

3.2.1 Cache

No que diz respeito a testes unitários, primeiramente foram feitos testes na *cache* de maneira a verificar todas as funcionalidades que eram oferecidas pela própria. De seguida são listados os vários testes.

- `whenPutValueOnCache_ifTimeNotPassed_ThenTheValueShouldBeRetrieved()`
- `whenPutValueOnCache_ifTimeToLivePassed_ThenNoValueShouldBeRetrieved()`
- `whenTheCacheIsFull_TheLeastRecentlyValueShouldBeRemoved()`
- `removeTest()`
- `sizeTest()`
- `isFullTest()`
- `maxSize()`
- `containsKey()`
- `percentageOfSuccessfulRequestsAnsweredTest()`

- `percentageOfSuccessfulRequestsAnswered_withRealCacheStatsObjectTest()`
- `mostRequestedTest()`
- `whenTryToCheckTheRequestOfNoCachedValue_ThenReturn0()`

3.2.2 Api's externas

Relativamente às duas *api's*, ambas foram testadas usando mocks como também usando ligações reais aos *endpoints*. De seguida, mostrar-se-á o código que permite comprovar a conexão real às *api's*.

```
@Test
void getCurrentAQFromAPITest() {
    OpenWeatherResponse response = this.openWeatherRequest.getCurrentAQFromAPI( lat: "40.866057889889286", lng: "-8.645710577339893");
    log.info(response);

    MatcherAssert.assertThat(response.getCoord().getLat(), CoreMatchers.is( value: "40.8661"));
}
```

Figure 17: Teste de integração, verificar ligação à api OpenWeather

Fazendo *mock* à classe `RestTemplate` foi possível verificar o que aconteceria se a conexão tivesse problemas. Para manter uma certa brevidade no relatório vou especificar simplesmente qual a classe e teste onde tal ocorreu. Por conseguinte, o que acabou de ser referido encontra-se na classe **OpenWeatherRequestTest** no teste *exceptionOccuredWhileContactingApi_thenReturnEmptyResponse()* [[github file link](#)].

Ainda relativamente às *api's*, na seguinte [classe de testes](#), foi verificado se a resposta que estava a ser renderizada da *api* sofria um parsing correto para uma resposta genérica a ambas as *api's* externas. No caso a uma Resposta do tipo **APIResponse**.

Por último, foram feitos mais testes de integração nas *api's externas* com o intuito de verificar se os parametros que estavam a ser passados nos pedidos eram os corretos. É possível verificar esses testes neste [ficheiro](#). Só referir que os testes foram replicados igualmente nas duas *api's externas*

3.2.3 Utils

Referir apenas que os recursos utilitários também foram testados com testes unitários

3.2.4 Serviços

Neste [ficheiro](#) é possível verificar que os recursos que a implementação desta classe consome foram todos *mocked*, dado que o objetivo era verificar a lógica implementada nas diferentes funcionalidades.

Nesta classe de testes foram testados maioritariamente os diferentes *work-flows*, como por exemplo, o facto da cache ser a primeira a ser acionada quando

chega um pedido, e se a mesma não conseguir satisfazer o mesmo, então nesse caso, é necessário efetuar um pedido à *api* externa, e assim sucessivamente.

3.2.5 REST API

Para testar o parsing dos *endpoints* da **REST API** foi utilizado o pacote *WebMvcTest* juntamente com o *mocking* dos serviços. Esta implementação de testes é menos morosa do que utilizar por exemplo *SpringBootTest*, portanto, foi decidido que se testaria desta forma o parsing, de modo a minorar o tempo dos testes. [[ficheiro github](#)]

De maneira a testar na integra as diferentes funcionalidades foi criado um teste *SpringBootTest*, em que, juntamente com *TestRestTemplate* foi possível testar os vários *endpoints*. [[ficheiro github](#)]

3.2.6 Aplicação Web

Finalmente, de forma a testar os *workflows* da aplicação web, foi utilizada a dependência de **Selenium** que permitiu automatizar os testes web. O teste pode ser encontrado [aqui](#), e a aplicação web está hospedada na *google cloud*, podendo ser acedida com o seguinte [link](#).

3.3 Análise estática de código

A análise estática de código foi conseguida a partir do uso de *SonarCloud* e obviamente que também com o uso do *linter* bastante completo do intellij (não usei o plugin da sonarqube devido ao facto de ocupar praticamente 200mb).

O *SonarCloud* foi integrado no processo de *CI*, que irei falar na secção seguinte, em que à medida que novos *pushes* eram feitos para o repositório *Github*, automaticamente os dados relativos a testes e a cobertura de código (utilizando *Jacoco*) eram publicados no dashboard do *SonarCloud*.

O *quality gate* utilizado foi relativamente brando no que diz respeito a *coverage* de código, mas o projeto tinha algum código que era impossível testar, nomeadamente o que diz respeito ao *frontend*, por essa razão o *quality gate* foi mais flexível neste aspeto.

Take it easily Rename Copy Set as Default Delete

Conditions ⓘ Add Condition

Conditions on New Code

Conditions on New Code apply to all branches and to Pull Requests.













Metric	Operator	Value	Edit	Delete
Coverage	is less than	55.0%		
Duplicated Lines (%)	is greater than	3.0%		
Maintainability Rating	is worse than	A		
Reliability Rating	is worse than	A		
Security Hotspots Reviewed	is less than	100%		
Security Rating	is worse than	A		

Figure 18: Quality Gate utilizado

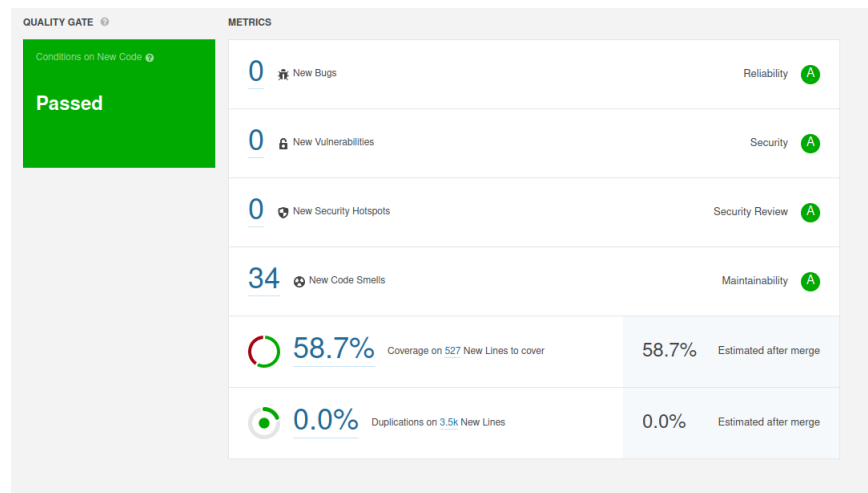


Figure 19: Dashboard SonarCloud

Esta ferramenta é muito importante, e este projeto veio ensinar-me isso mesmo. A partir do que o *SonarCloud* foi aconselhando aprendi em que situações devo utilizar **var** quando se inicializa uma variável. Aprendi novamente, que não se deve passar diretamente para um url, um campo associado ao input de um utilizador, entre outras.

3.4 Metodologias de integração contínua de código

Github actions permitiu a integração contínua do código, o ficheiro de autom-
atização pode ser analisado [aqui](#).

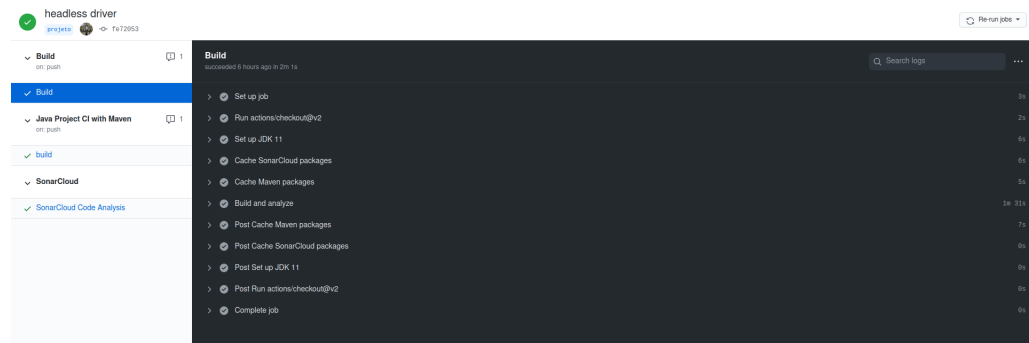


Figure 20: CI pipeline

Na imagem estão retratadas duas *builds*, mas apenas a primeira tem interessa para o caso de estudo. Dado que esta primeira *build* integra já *SonarCloud*, enquanto a segunda apenas automatiza testes *maven*.

No que diz respeito ao [ficheiro](#) em si, o mesmo tem o nome ingénuo de "Build" e declara que apenas se fará a *build*, aquando novos *pushes* para o repositório. De seguida instala as dependências necessárias, como por exemplo, *java11*, *maven*, os próprios pacotes *SonarCloud*. Para terminar, corre o comando que permitirá a execução dos testes.

4 Recursos

Link para aplicação: <https://tqs-mid-term-project.ew.r.appspot.com/>

Link para dashboard SonarCloud: <https://sonarcloud.io/dashboard?branch=projetoId=Orlando-pt.TQS>

References

- [1] Qualidade do ar ambiente, dados disponibilizados pela DGS
- [2] OpenWeather API
- [3] WeatherBit API
- [4] Google Cloud
- [5] SonarCloud
- [6] Github
- [7] Jacoco