HW1: Mid-term assignment report

João Ferreira [80041], v2020-04-14

Conteúdo

1 Introdução

1.1 Visão geral

O objetivo deste projeto é disponibilizar um serviço que permita realizar a consulta da qualidade do ar no momento da consulta para uma determinada cidade. Cada cidade tem associada uma estação que irá medir os valores relativos à qualidade do ar. Diferentes estações podem medir valores diferentes, sendo que têm valores comuns a todas as estações como, por exemplo, Air Quality Information (um número com uma escala númerica associada).

Este serviço não implementa uma base de dados, mas sim uma cache implementada localmente. O projecto foi desenvolvido utilizando SpringBoot e os dados são obitdos da API AQICN, cujo link se encontra no ponto 4 – Referências & Recursos deste relatório.

Inicialmente, todas as estações disponíveis para consulta são guardadas de maneira persistente, i.e, os dados nunca chegam a expirar, na cache. Após este processo, cada vez que o utilizador pretende consultar os valores da qualidade do ar para uma determinada estação com uma cidade associada a partir da interface web desenvolvida, a cache é acessada para que o sistema averigue se esse pedido já foi realizado anteriormente. A cache tem uma váriável que define o tempo em Unix Timestamp (Epoch) 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 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. Os dados retornados são armazenados na cache com um timestamp do momento em que essa chamada foi realizada.

O projeto desenvolvido é composto pelas seguintes classes:

- AirQuality – Definição do objecto AirQuality (valores retornados pela API AQICN). Cada AirQuality tem um status ("ok" se a API responder, quer a cidade requisitada seja válida ou não), um timestamp (Unix) registado quando é feita a chamada à API e um objecto Info, com vários valores associados (i.e, o objecto Info é composto por vários valores relativos à qualidade do ar, como Monóxido de Carbono, Pressão Atmosférica, etc);
- AirQualityController – Classe utilizada como controller da API desenvolvida com diversos endpoints que irei descrever no ponto 2.3;
- AirQualityService – Classe responsável pelas interações com a cache, relativamente a objectos relacionados com o tipo AirQuality;
- Cache – Classe onde é definida a cache. Na cache são definidas cinco variáveis: um objecto Map para guardar os dados relativos ao AirQuality, um objecto Map para guardar os dados relativos às Stations, um inteiro para guardar o número de pedidos realizados à cache (requests), um inteiro para guardar o número de acessos com sucesso à cache (hits) e um inteiro para guardar o número de acessos sem sucesso à cache (miss);
- Info - Definição do objecto Info. Este objecto possui um valor (int) associado correspondente ao Air Quality Index e um HashMap, denominado iaqi, com 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, { "co2" : {"v" : 10} }) ;
- Station – Definição do objecto Station. Cada Station tem um id que a identifica e uma cidade associada;
- StationService – Classe responsável pelas interações com a cache, relativamente a objectos relacionados com o tipo Station;
- TqsHw1Application – Classe predefinida do SpringBoot. Nesta classe são criadas todas as estações pretendidas e guardadas em cache;

1.2 Limitações

Durante a realização deste projeto encontrei algumas adversidades que não considero prejudiciais àquilo que era o principal objetivo do mesmo.

Uma dessas adversidades é resultante da inexistência de um endpoint na API AQICN que me permita obter todas as cidades onde é possível controlar a qualidade do ar de acordo com certos parâmetros. De modo a contornar essa adversidade, decidi escolher as cidades disponíveis para escolha do utilizador, o que limita um pouco a sua acção. Considero este aspecto pouco dinâmico e que não reflecte o dinamismo que pretendia implementar inicialmente. Contudo, a situação foi resolvida e o projecto funciona como esperado.

Ainda no âmbito da API escolhida , por vezes o tempo de resposta encontra-se entre os 500 e 600 milisegundos, o que transmite a sensação de que o sistema está lento. Tentei, sem sucesso, contornar este problema alterando a minha conexão à internet, utilizando um cabo de rede, pensando que o problema talvez pudesse ser da minha operadora, uma vez que nos encontramos em estado de emergência nacional, ou seja, uma elevada percentagem da população encontra-se confinada na sua habitação, aumentando, assim, significamente o volume de tráfego. Como os valores continuaram no padrão habitual, pude constatar que o problema talvez pudesse ser da localização da API, visto que, aparentemente, foi desenvolvida na China.

Apesar das adversidades mencionadas nos dois parágrafos anteriores, mantive a minha decisão de escolher esta API, uma vez que de todas as que explorei era a mais completa e organizada.

Por fim, e em modo de auto-crítica, penso que podia ter reduzido o número de testes realizados (27), sobretudo os unitários relacionados com a cache. Escrevo a afirmação anterior porque, num número reduzido de casos, testo algo que já tinha sido testado previamente, não como repetição mas sim como complemento de outro teste. Por motivos de perfecionismo e brio, tomei a decisão de manter os testes desenvolvidos.

Numa fase final de desenvolvimento do projeto, tentei implementar o endpoint /error de várias formas, chegando mesmo a trocar opiniões com colegas, mas não me foi possível por diversos erros como, por exemplo, a não apresentação da página inicial index.html ou não apresentação do erro, apresentando apenas uma página em branco, sendo que deveria ser apresentado um texto de erro previamente definido. Não considerei este problema importante uma vez que não afetou o funcionamento da aplicação nem os testes realizados.

2 Especificações do produto

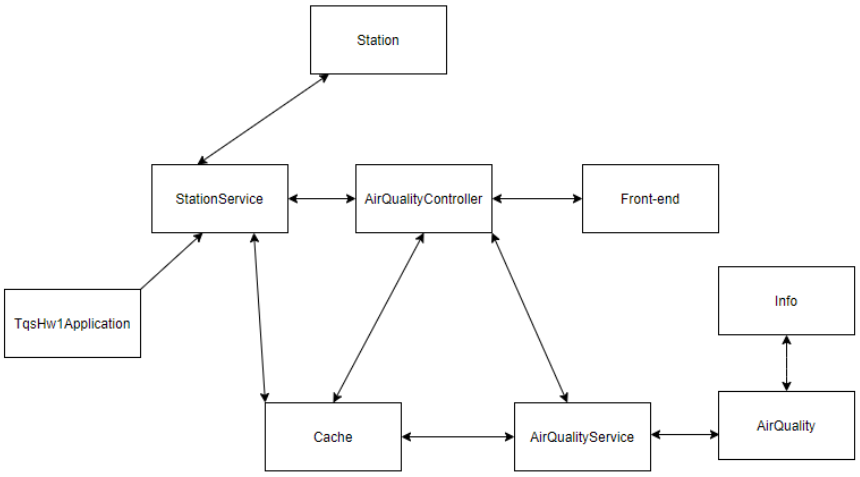
2.1 Functional scope e interações suportadas

Esta aplicação destina-se ao mais diversos tipos de utilizadores, desde simples utilizadores comuns a instituições que pretendam monitorizar a qualidade do ar para uma determinada cidade.

Para o utilizarem necessitam apenas de aceder à página inicial localhost:8080, onde será exibida uma página com as cidades disponíveis, onde deverão escolher apenas uma, clicando de seguinda no botão "OK".

2.2 Arquitectura do sistema

O projecto desenvolvido é composto pelas classes descritas no ponto 1.1 – Visão geral. Reflectindo as interações que ocorrem entre essas classes, a arquitectura do sistema pode ser descrita pelo seguinte diagrama:



O sistema inclui as seguintes entidades (descritas no ponto 1.1 – Visão geral):

- Station – representa uma estação de medição dos valores da qualidade do ar;
- StationService – realiza as operações sob a cache relativas a objectos Station;
- Cache – representa a entidade responsável por guardar os valores;
- AirQuality – representa os valores registados para uma cidade;
- AirQualityService – realiza as operações sob a cache relativas a objectos AirQuality;
- AirQualityController – representa o controller da API;

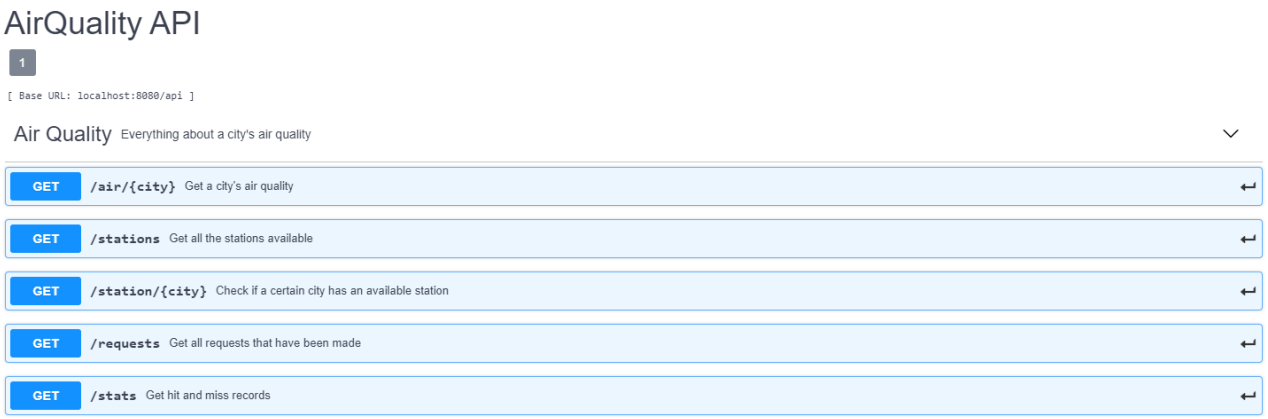
Para o desenvolvimento deste projecto, como indicado pelo enunciado, utilizei a framework SpringBoot, em Java, criando um projecto Maven-based. Para o front-end, utiliza HTML e JavaScript. Nos testes desenvolvidos utilizei Mockito, MockMvc, Selenium, SonarQube e jUnit.

2.3 API for developers

Os serviços disponibilizados pela api desenvolvida são:

- Consulta dos valores de qualidade do ar por cidade (/air/{city});
- Consulta das estações disponíveis, sendo apresentado o nome da cidade e o respectivo ID (/stations);
- Verifcar se uma determinada cidade tem uma estação disponível (/station/{city});
- Consultar o número de acessos (requests) realizados pelo utilizador (/requests);
- Consultar as estastíticas resultantes do número de acessos (hit & miss - /stats);

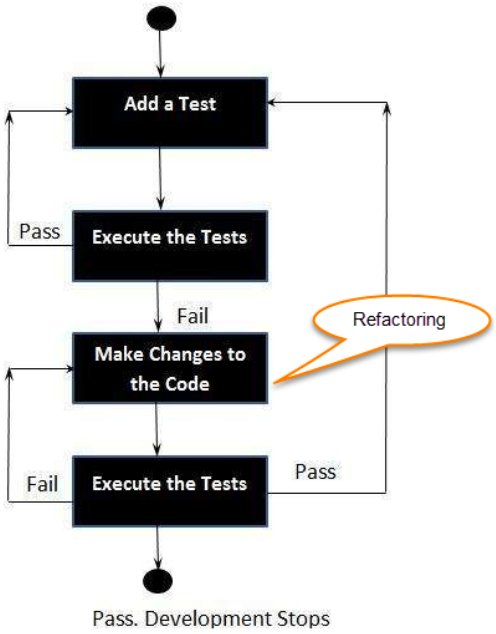
A API desenvolvida fornece ainda os seguintes endpoint, com base URL equivalente a localhost:8080/api:



3 Garantia de qualidade

3.1 Estratégia para a realização de testes

Para o desenvolvimento dos testes definidos comecei por, inicialmente, definir toda a aplicação, criando e manipulando todos os elementos necessários ao seu bom funcionamento. Após a aplicação estar completa, optei por iniciar a fase de testes, adoptando uma estratégia TDD. Este processo foi de algum modo semelhante ao descrito pelo diagrama presente na página seguinte.



https://www.browserstack.com/guide/tdd-vs-bdd-vs-atdd

3.2 Testes unitários e de integração

Para testar a Cache, realizei inúmeros testes unitários, utilizando a ferramente jUnit. Os testes realizados procuraram testar todos os métodos implementados na Cache.

Para testar o método setAirQuality comecei por registar um objecto AirQuality na cache. Após guardar o objecto, verifiquei se o tamanho do objecto Map que guarda este tipo de objectos na cache tinha aumentado uma unidade e se os valores do tamanho antes e depois de guardar eram diferentes. Por fim, invoquei o método getAirQuality e conferi se o objecto retornado correspondia ao guardado.

Testando o método getAirQuality, segui a mesma lógica do teste do método setAirQuality, registando um objecto, retornando-o e, por fim, verificar que o tamanho do Map foi alterado e que o objecto retornado era o esperado.

Para testar o método setStation e getStation, segui a mesma lógica explicada nos dois parágrafos anteriores aplicada a objectos do tipo Station, e não do tipo AirQuality.

De modo a realizar os testes ao método countRequests, comecei por registar o valor actual dos requests, guardado em cache. De seguida, invoco o método getAirQuality (poderia invocar o método getStations também, o resultado seria o mesmo uma vez que ambos incrementam os requests) e guardo o novo valor dos requests. Por fim, comparo o valor dos requests, verificando que são diferentes, com diferença de uma unidade entre o novo valor e o antigo.

Para testar o método isValid, registo dois valores iguais de AirQuality para duas cidades (Cannes e Houston), com a diferença que para Cannes associo o tempo actual e para Houston registo um tempo igual a 0, ou seja, 1970. Assim, o registo de Cannes será válido e o de Houston não, uma vez que o tempo máximo está definido como sendo de 10 minutos (600 000 ms). Por fim, verifico se estes registos são considerados válido e inválido, respectivamente.

Para o teste do método getCitiesAvailable, começo por registar duas Stations para as cidades de New York e Trofa, com id 6 e 7, respectivamente. De seguida, invoco o método a ser testado, que retorna um ArrayList, e verifico se essas cidades se encontram nesse ArrayList.

Para testar o método HitAndMiss, começo por incrementar esses valores na cache. Por fim, verifico se a sua soma é a esperada e se os seus valores individuais estão correctos.

Para testar o método getAirQualityByCity, começo por criar um objecto Info com todos os atributos necessários. De seguida, crio um objecto 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. Finalmente, testo este método verificando se os atributos do objecto retornado são iguais aos do objecto guardado.

Para o teste relativo ao método getStationByID, começo por guardar uma station com id 23 associado à cidade Los Angeles e defino um id errado. Invoco o método getStationByID e verifico se o objecto retornado tem os valores que foram inseridos, e se, invocando esse método com o id errado, o objecto retornado corresponde a null.

Em questão de testes de integração, realizei testes ao AirQualityController, utilizando MockMvc. Para testar o método getAirQuality, comecei por testar o endpoint /api/air para todas as cidades disponíveis, avaliando o conteúdo retornado, nomeadamente, o status.

Para além deste teste, também realizei um test ao endpoint /api/stations, verificando se todas as cidades disponíveis se encontram na String retornada.

Testei também o endpoint /api/stats, verificando se, na String retornada, os labels Hit and Miss não são null, nem vazios, nem iguais à String “”. Verifico também se os valores associados correspondem a valores numéricos.

Para testar o método getSpecificStation, acedo ao endpoint /api/station, verificando se o seu status coresponde a “ok” e se contém todas as cidades disponíveis, individualmente.

Ainda no âmbito dos testes do AirQualityController, desenvolvi o método testSpecificStationWrongCity, que verifica se, dada uma station errada o resultado retornado é equivalente a “Station not found”.

3.3 Testes funcionais

Em questão de testes funcionais, realizem alguns testes utilizando o Selenium, aplicado ao browser Firefox. Para prepar os testes utilizei o geckodriver. Começo por tentar obter os valores para cada cidade disponível, selecionando cada uma delas no menu dropdown, terminando por carregar no botão “OK”.

Para além deste teste, verifico também se cada elemento predefinido no ficheiro html se encontra na página inicial, isto é, se a página foi carregada correctamente. Este método faz uso do método isElementPresent, definido na mesma classe.

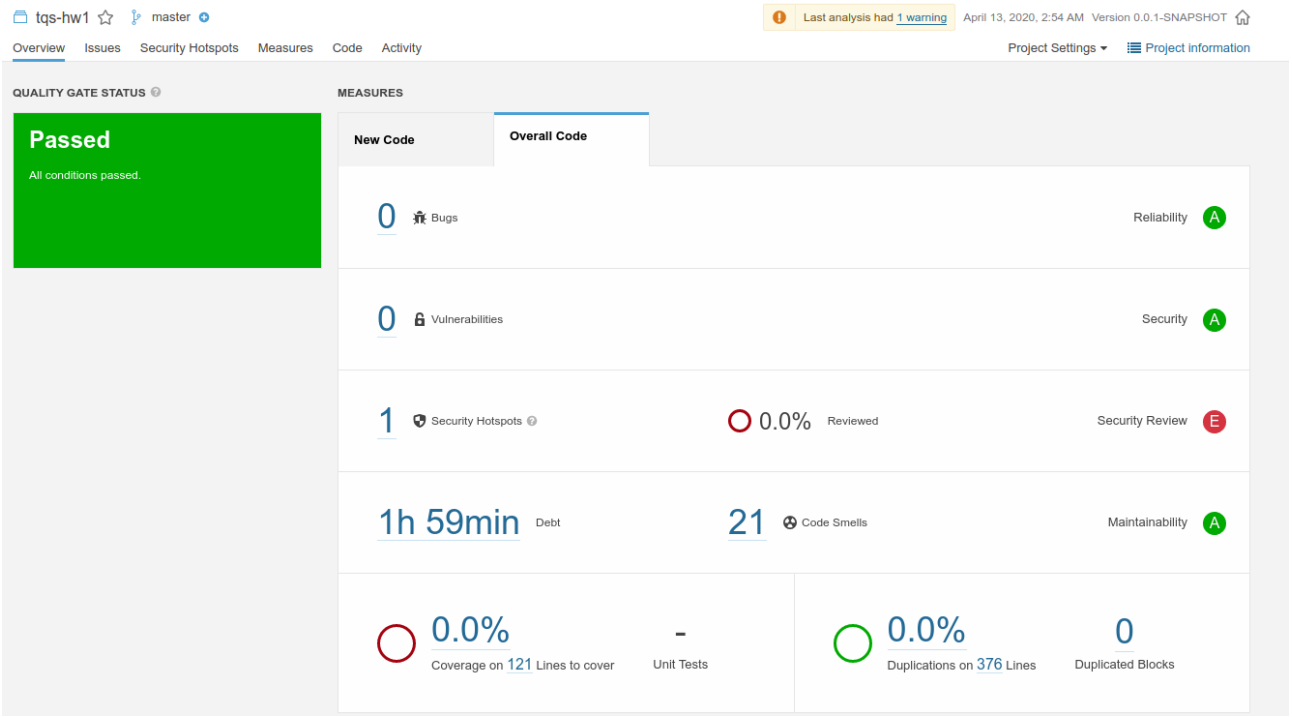
3.4 Análise de código com SonarQube

Para analisar o código desenvolvido utilizei o SonarQube, exibindo os resultados na porta 9000, em localhost. Esta ferramenta foi útil por várias razões:

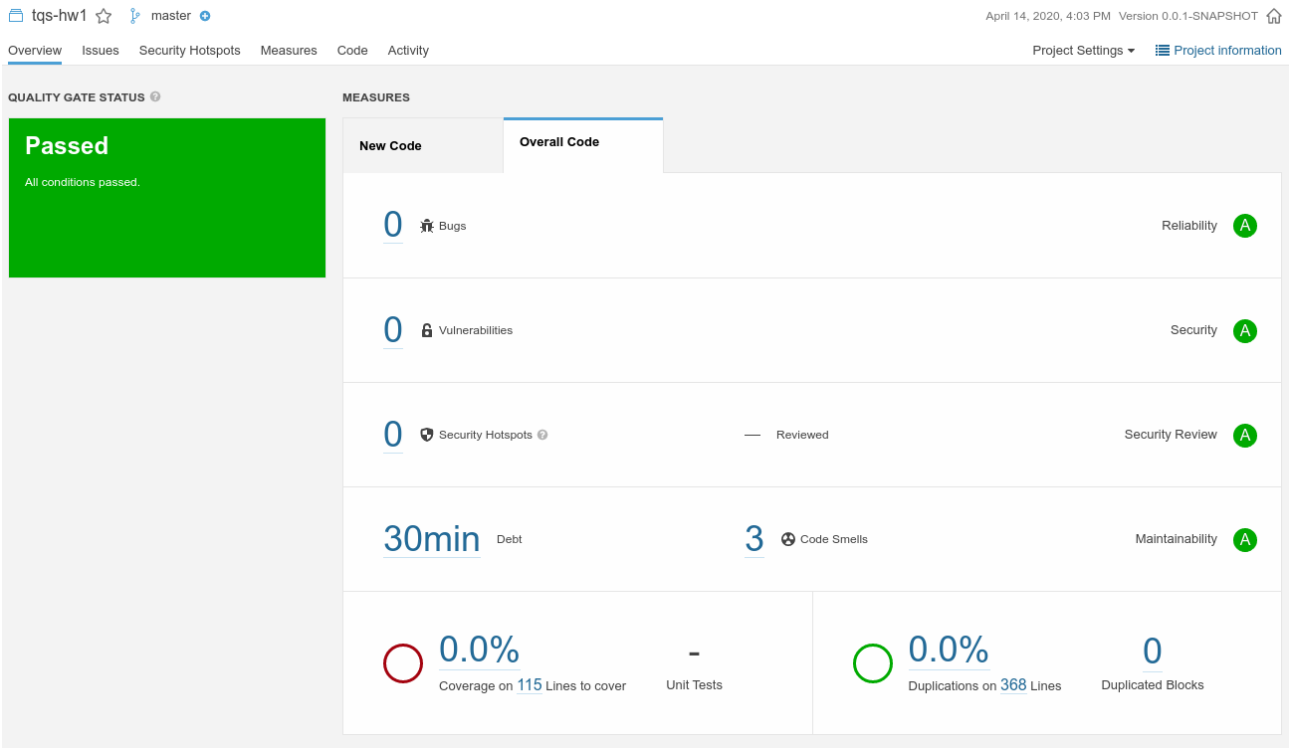
- Permitiu corrigir uma vulnerabilidade que não tinha identificado, relacionada com a execução da aplicação;
- Permitiu detectar code smells que não teria detectado sem consultar os resultados mostrados, tais como: linhas repetidas, if then else que poderiam ser convertidos apenas num if com o operador and, linhas comentadas que poderiam ser apagadas, i.e., com código possível de executar e também objectos não utilizados.

Por fim, exibo os resultados obtidos.

A imagem abaixo reflecte a primeira análise feita pelo SonarQube, detectando 21 code smells e 1 vulnerabilidade, reprovando a componente de segurança.



Após analisar os resultados mostrados na imagem anterior, optei por corrigir os code smells e a vulnerabilidade exibida, obtendo um novo output:



3.5 Service Level Tests

Para testar os serviços disponíveis, defini as classes StationServiceTest e AirQualityServiceTest. Na classe StationServiceTest, testo o método getStationByID, verificando se o objecto retornado corresponde ao guardado. Testo também a invocação deste método com um ID incorrecto, verificando se o objecto retornado corresponde a null. Por fim, quando retornada uma station válida, verifico se os seus atributos estão correctos, desenvolvendo um novo método de teste.

Na classe AirQualityServiceTest, testo se os registos de uma determinada cidade são válidos ou inválidos, testando, para verificar se são válidos, com as cidades previamente definidas e, para verificar se são inválidos, com “Esgueira”. Dada uma cidade disponível, verifico também se os seus dados que estão guardados em cache estão correctos, testando também se, dada uma cidade inválida, se o valor associado corresponde a null. Por fim, testo a seguinte acção: dadas 5 cidades, o valor do ArrayList retornado pelo método getCitiesAvailable deve conter essas cidades e o seu tamanho deve ser exactamente igual a 5. Testo ainda se o méotodo da cache getHitAndMiss retorna o valor corresponde à soma dos valores previamente definidos para Hit e Miss.

4 Referências & recursos

Project resources

- Repositório Git: https://github.com/joaogferreira/hw1-TQS
- A demonstração em vídeo encontra-se no repositório no github, na pasta videos;
- Na pasta sonar_files, encontra-se o token respectivo do sonar, bem como o comando maven para testar o projecto.

Materiais

- API AQICN - https://aqicn.org/api/pt/
- Getting started – Spring boot - https://spring.io/guides/gs/serving-web-content/
- Get started in two minutes guide – Sonar Qube - https://docs.sonarqube.org/latest/setup/get-started-2-minutes/
- Geckdriver in selenium web driver - https://www.youtube.com/watch?v=Loe8lL8Tg-I
- Continuous Integration of Java project with GitHub Actions - https://medium.com/faun/continuous-integration-of-java-project-with-github-actions-7a8a0e8246ef
- Mocking with examples - https://learning.postman.com/docs/postman/mock-servers/mocking-with-examples/
- JSONPath online evaluator - https://jsonpath.com/
- How To Record Your Screen in Ubuntu With SimpleScreenRecorder - https://itsfoss.com/record-screen-ubuntu-simplescreenrecorder/