HW1 - Desenvolvimento de testes para uma aplicação multicamada

Alina Yanchuk 89093

TQS – DETI – UA – Abril 2020

https://github.com/alina-yanchuk02/tqs_hm1 (público)

1 Introdução

1.1 Contexto

Neste relatório será descrita a estratégia adotada para a realização de testes automatizados numa aplicação multicamada, desenvolvida em Spring Boot, que implementa RESTful API, e permite ao cliente a obtenção de dados retirados de uma API externa.

Assim, foi desenvolvido o produto AirQualityPT, que consiste numa página Web, e permite consultar a qualidade do ar, no dia atual, para algumas cidades portuguesas.

Para além disso, foi desenvolvida uma página à parte, que permite vizualizar as estatísticas do uso da Cache.

1.2 Limitações

A grande limitação para este projeto foi apenas a falta de tempo, aliada à grande quantidade de projetos e trabalhos que temos por fazer nesta altura; o que não permitiu a criação de uma interface e funcionalidades do produto mais desenvolvidas. Foram construídas apenas duas páginas simples (uma normal e outra para estatísticas da cache), que permitem visualizar o funcionamento do serviço e a realização de todos os testes necessários, tendo-se focado mais neste último ponto.

No entanto, claramente que a parte de UI poderá ser melhorada, e mais funcionalidades implementadas.

Quanto ao uso da API externa, não houve limitações, já que está possui a informação necessária e é grátis. O único problema é o facto de existir pouca informação para Portugal, resumindo-se apenas numa dezena de distritos.

2 Especificação do produto

2.1 Escopo funcional e interações suportadas

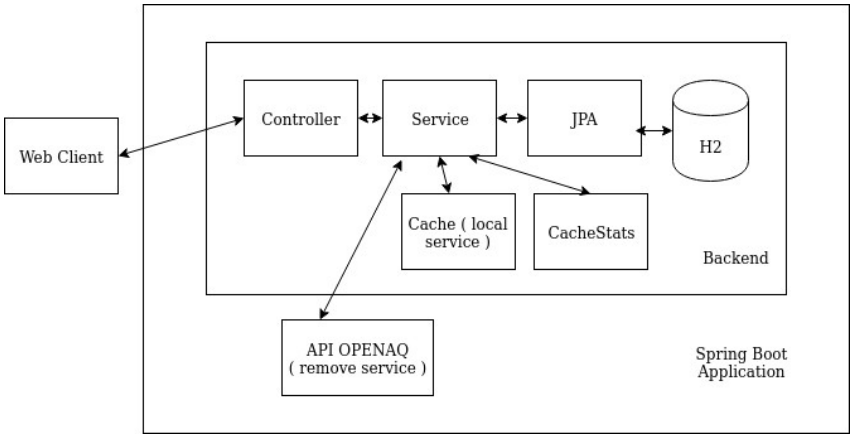
Para:	Pessoas residentes em Portugal ou outras
Que apresentam:	Necessidade de conhecer, em tempo real, a qualidade do ar na sua cidade, ou outra
O produto:	AirQualityPT
Que:	Permite, de forma muito simples, consultar dados sobre a qualidade do ar (pm10, co2, o2, …) em certas cidades portuguesas
Ao contrário de:	Outras aplicações mais complexas , lentas e não atualizadas periodicamente
O meu produto:	Visa apresentar, de forma rápida e simples, a qualidade do ar atual em Portugal, com os dados mais recentes e constantemente atualizados

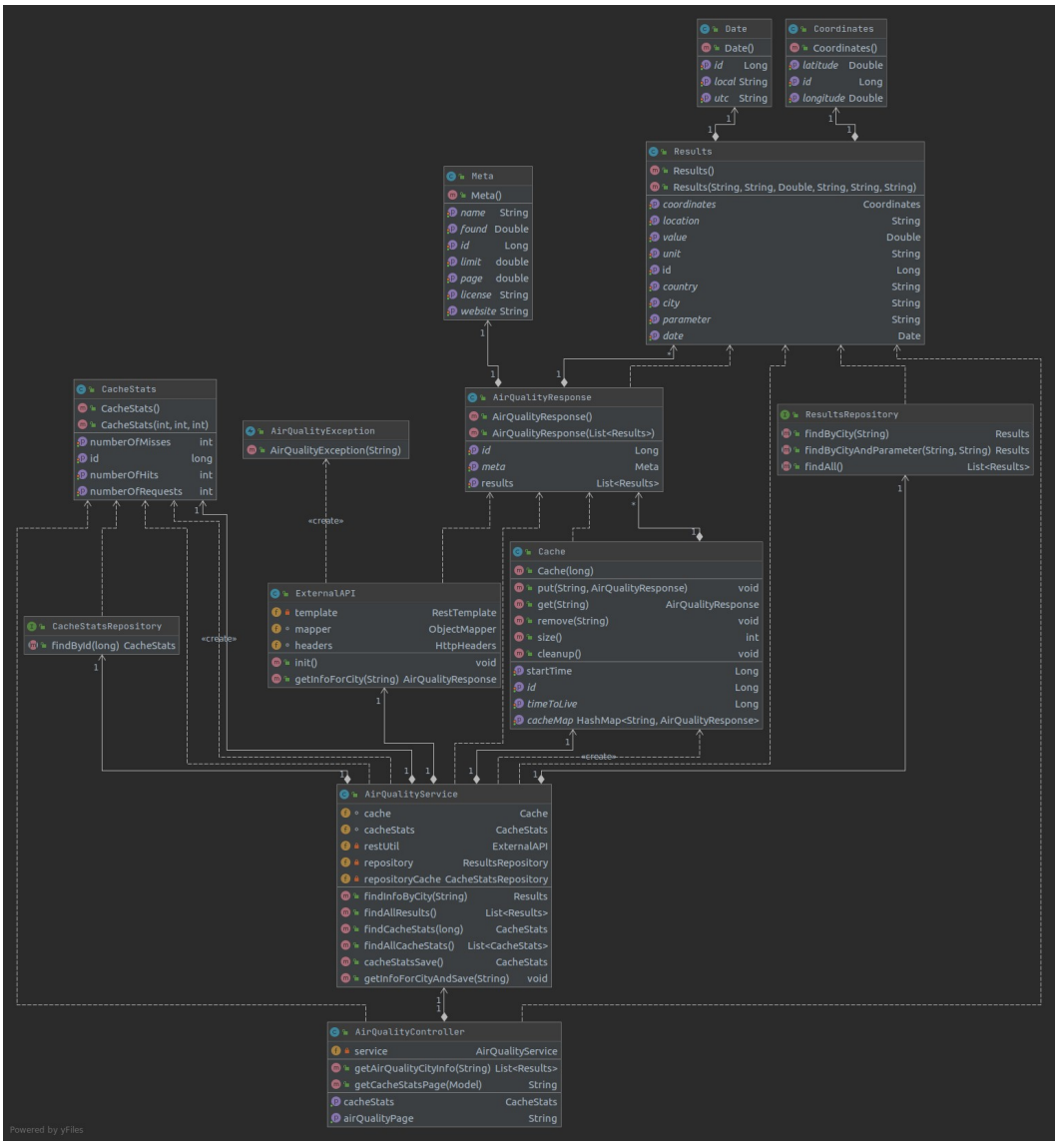
Cenários:

1. O utilizador entra no web site, verifica que é possível obter dados reais e atualizados sobre a qualidade de ar na sua cidade, no dia de hoje, e introduz “Leiria” no campo de pesquisa. Ao obter uma tabela com alguns dados relevantes, vizualiza-os e fecha o site.

2. Após isso, o utilizador decide voltar a fazer a mesma pesquisa. Realiza o mesmo procedimento, e obtém os mesmos dados. Decide verificar se os dados apenas continuam iguais, ou se foi usado o serviço da Cache. Para isso acede ao site de estatísticas da cache e verifica que, durante a sua sessão, fez duas pesquisas, sendo que uma delas (a última) , realmente utilizou a Cache.

2.2 Arquitetura do sistema





A lógica que foi usada foi a seguinte:

- O cliente Web interage com a aplicação Web, passando-lhe parâmetros a consultar (nome da cidade);

- O Controller comunica com a camada Service, e esta por sua vez com a API Externa e a Cache;

- Service verifica se a cidade pesquisada já foi consultada antes e se os seus dados encontram-se armazenados na cache; se não, pede os dados novamente à API Externa;

- Os dados são colocados no repositório JPA e são de lá retirados para retorná-los ao cliente;

- A camada Service é também responsável por guardar as estatísticas do uso da Cache, na entidade CacheStats, cujos dados são também colocados num repositório e de lá retirados. As estatísticas são guardadas apenas para a sessão atual. Se a aplicação Spring for re-executada, os dados são limpos.

- A API Externa utilizada é a OPENAQ. Não tem *keys*, e é gratuita.
- Foi configurado um *time to live* de 20 segundos para a cache, para mostrar a sua funcionalidade. Num ambiente real, este poderia ser maior, já que os dados permanecem mais ou menos inalteráveis ao longo do tempo.
- A camada Web foi desenvolvida com Thymeleaf.
- A base de dados é in-memory, H2.
- A Cache foi implementada com HashMap.

2.3 API cliente

Endpoints:

GET : /tqs/airQualityPT → Permite obter a página principal, onde o utilizador poderá pesquisar por uma cidade e visualizar os dados; Sempre que uma cidade é inserida no campo de pesquisa, é invocada a REST API em */tqs/get/{cidade}* , no entanto isto não é vizualizado na interface do cliente (acontece nos "bastidores").

GET : /tqs/cacheStats → Permite obter a página secundária, onde o utilizador poderá visualizar as estatísticas do uso da Cache para essa sessão;

GET : /tqs/get/{cidade} → Não é utilizada diretamente pelos clientes; retorna os dados para uma certa cidade, quando é feita uma pesquisa de cidade na página principal; é utilizada diretamente pelo primeiro *Endpoint* e ,assim, indiretamente pelos clientes;

GET : /tqs/get/cacheStats → Não é utilizada pelos clientes; implementa o mesmo serviço que o segundo *Endpoint* (esse sim utilizado pelos clientes), mas sem Model; foi criada apenas para ser utilizada nos testes e mostrar que o serviço funciona;

3 Garantia de qualidade

3.1 Estratégia para testes

Foi utilizada a abordagem TDD, no início, nomeadamente para os testes unitários da classe Controller.

Foram feitos testes unitários, de integração e funcionais para as duas páginas (AirQualityPT e cacheStats).

Os testes unitários encontram-se na pasta tqs.hw1.airqualityapp, os testes funcionais na pasta FunctionalTests e os de integração na pasta IntegrationTests.

3.2 Testes unitários

Foram escritos 31 testes unitários e testadas todas as classes e camadas da aplicação.

Alguns exemplos (os restantes ver no git) :

ResultsRepositoryTest.java

O Json recebido da API Externa possuir vários documentos (Results), sendo que todos esses documentos correspondem à mesma cidade, mas têm parâmetros (co2, pm10,…) e valores diferentes. Assim, são guardados todos esses resultados, no repositório, para a mesma cidade.

```
@Test
public void givenSetOfResults_whenFindAll_returnAllResults(){

    Results ResultA = new Results( location: "PT0", parameter: "o3", value: 30.0, unit: "ug/m", country: "PT", city: "Leiria");

    Results ResultB = new Results( location: "PT0", parameter: "co2", value: 12.0, unit: "ug/m", country: "PT", city: "Leiria");

    entityManager.persist(ResultA);
    entityManager.persist(ResultB);
    entityManager.flush();

    List<Results> allResults = resultsRepository.findAll();

    assertThat(allResults)
            .hasSize(2)
            .extracting(Results::getCity).contains(ResultA.getCity(),ResultB.getCity());

}
```

AirQualityServiceUnitTest.java

Aqui é testado se a API Externa é invocada ou não, estando a Cache vazia ou não, com o uso de Mockito e isolamento do serviço de API externo.

```
@Test
public void testGetInfoForCityAndSave_withCacheAndWithoutCache() throws Exception, InterruptedException {


    Results resultA=new Results( location: "PT0", parameter: "o3", value: 30.0, unit: "ug/m", country: "PT", city: "Leiria");
    Results resultB=new Results( location: "PT0", parameter: "co2", value: 15.0, unit: "ug/m", country: "PT", city: "Leiria");

    List<Results> lista = Arrays.asList(resultA,resultB);

    AirQualityResponse response = new AirQualityResponse(lista);

    // the cache is empty, so the external API is gonna be called once
    Mockito.when(externalApi.getInfoForCity("Leiria")).thenReturn(response);
    Mockito.when(cache.get("Leiria")).thenReturn(null);


    String city="Leiria";
    String parameter1="o3";
    String parameter2="co2";

    service.getInfoForCityAndSave(city);
    verifyExternalAPIisCalledOnce(city);
    verifyFindByCityAndParameterIsCalledOnce(city,parameter1);



    // the cache have the info, so the external API won't be called
    Mockito.when(cache.get("Leiria")).thenReturn(response);


    service.getInfoForCityAndSave(city);
    verifyExternalAPIisNotCalled(city);
    verifyFindByCityAndParameterIsCalledOnce(city,parameter2);


}
```

Aqui o comportamento do repositório é simulado com o uso de Mocks (configurado no ínicio em @BeforeEach).

```
@Test
public void given2Results_whenGetAll_thenReturn2Records(){

    Results resultA=new Results( location: "PT0", parameter: "o3", value: 30.0, unit: "ug/m", country: "PT", city: "Leiria");
    Results resultB=new Results( location: "PT0", parameter: "co2", value: 15.0, unit: "ug/m", country: "PT", city: "Leiria");

    List<Results> allResults = service.findAllResults();

    assertThat(allResults)
            .hasSize(2)
            .extracting(Results::getCity).contains(resultA.getCity(),resultB.getCity());

    verifyFindAllResultsIsCalledOnce();
}
```

ExternalAPITest.java

É testada a comunicação com a API Externa, os conversores e se o Json recebido é correto.

```
@Test
public void consumeExternalApiTest() throws Exception {

    externalApi.init();

    AirQualityResponse response = new AirQualityResponse();

    String result="AnyString";

    Mockito.when(restTemplate.getForObject(Mockito.anyString(), Mockito.any(Class.class))).thenReturn(result);

    Mockito.when(objectMapper.readValue(Mockito.anyString(), Mockito.any(Class.class))).thenReturn(response);

    assertThat(externalApi.getInfoForCity("Leiria")).isInstanceOf(AirQualityResponse.class);

}



@Test
public void testGetMessage() {

    restTemplate=new RestTemplate();

    String response=restTemplate.getForObject( url: "https://api.openaq.org/v1/measurements?city=Leiria", String.class);

    assertThat(response).contains("Leiria").contains("results");

}
```

CacheTest.java

São testadas as funcionalidades da Cache.

```java
@Test
public void testPut(){
    cache.put("A",response);
    assertTrue(cache.getCacheMap().containsKey("A"));
    assertEquals( expected: 1,cache.size());
}

@Test
public void testCleanup() throws InterruptedException {

    cache.put("A",response);
    // to timeToLive pass
    cache.setStartTime(System.currentTimeMillis()-101l);
    cache.cleanup();
    assertEquals( expected: 0,cache.getCacheMap().size());
}
```

AirQualityControllerIT.java

É testada a REST API no lado do servidor,usando o MockMvc;

```java
@Test
public void whenGetResults_thenStatus200() throws Exception {

    // repository is filled with info when api is called , because there is no create method for this web page ( the info is taken from external api )
    // as the external Api has always info for city Leiria ,
    // the list of results will have size 1 ou more ( the results for one city differ by the parameter ( co2, o2, .... but the city is the same )

    mvc.perform(get( urlTemplate: "/tqs/get/Leiria").contentType(MediaType.APPLICATION_JSON))
            .andDo(print())
            .andExpect(status().isOk())
            .andExpect(content().contentTypeCompatibleWith(MediaType.APPLICATION_JSON))
            .andExpect(jsonPath( expression: "$", hasSize(greaterThanOrEqualTo( value: 1))))
            .andExpect(jsonPath( expression: "$[0].city", is( value: "Leiria")));

}
```

AirQualityControllerTemplateIT.java

É testada a REST API com o cliente HTTP envolvido.

```java
@Test
public void whenGetResults_thenStatus200() {

    // repository is filled with info when api is called , because there is no create method for this web page ( the info is taken from external api )

    ResponseEntity<List<Results>> response = restTemplate
            .exchange( url: "/tqs/get/Leiria", HttpMethod.GET,  requestEntity: null, new ParameterizedTypeReference<List<Results>>() {
            });

    assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);

}
```

3.3 Testes de integração

Alguns exemplos (os restantes ver no git) :

TestAPI.java

Os testes de sistema e de integração foram feitos utilizando Rest-Assured.

Aqui é testado que o documento recebido é do tipo Json e possui a informação necessária e correta. Relembrando que é este resultado que será utilizado no HTML.

```
@Test
public void getAirQualityCityInfo_returns200_andExpectedInfo() {

    String uriBase = "http://localhost:8080/tqs/get/Leiria";

    given() RequestSpecification
            .relaxedHTTPSValidation()
            .when()
            .get(uriBase) Response
            .then() ValidatableResponse
            .statusCode(200)
            .contentType(ContentType.JSON)
            .body( s: "[0].unit", is( value: "µg/m³"))
            .body( s: "[0].country", is( value: "PT"))
            .body( s: "[0].city", is( value: "Leiria"));

}
```

Aqui é testada a própria página HTML.

```
@Test
public void getAirQualityPage_returns200() {

    String uriBase = "http://localhost:8080/tqs/airQualityPT";

    given() RequestSpecification
            .relaxedHTTPSValidation()
            .when()
            .get(uriBase) Response
            .then() ValidatableResponse
            .statusCode(200)
            .contentType(ContentType.HTML);

}
```

3.4 Testes funcionais

Alguns exemplos (os restantes ver no git) :

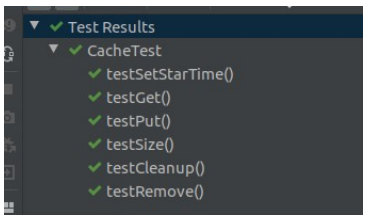
SeleniumWebDriverTest.java

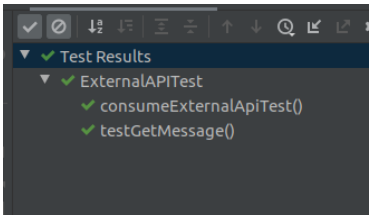
Como testes de aceitação foram realizados testes com o Selenium, usando o driver do Chrome.

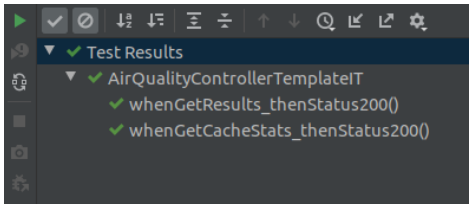
```
public void testAirQualityPage() {

    WebDriver driver = new ChromeDriver();

    driver.get("http://localhost:8080/tqs/airQualityPT");

    assertEquals( expected: "Spring Boot - AirQualityPT", driver.getTitle());

    WebElement titulo=driver.findElement(By.id("titulo"));

    assertEquals( expected: "AirQuality in Portugal Today",titulo.getText().toString() );

    driver.findElement(By.id("input")).click();
    driver.findElement(By.id("input")).clear();
    driver.findElement(By.id("input")).sendKeys( ...charSequences: "Leiria");
    driver.findElement(By.id("input")).sendKeys(Keys.ENTER);

    // as info may take time to be processed ; may depend on internet speed
    WebDriverWait wait = new WebDriverWait(driver,  timeOutInSeconds: 10);

    WebElement table = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("tableBody")));

    int numOfRow = table.findElements(By.tagName("tr")).size();

    assertThat(numOfRow).isGreaterThan(0);


    // when city doesn't exist, there are no results


    driver.findElement(By.id("input")).click();
    driver.findElement(By.id("input")).clear();
    driver.findElement(By.id("input")).sendKeys( ...charSequences: "DoesNotExist");
    driver.findElement(By.id("input")).sendKeys(Keys.ENTER);

    WebElement table2=driver.findElement(By.id("tableBody"));
    int numOfRow2 = table2.findElements(By.tagName("tr")).size();

    assertThat(numOfRow2).isEqualTo(0);

    driver.quit();

}
```

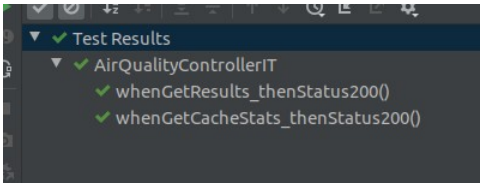
3.5 Validação dos testes

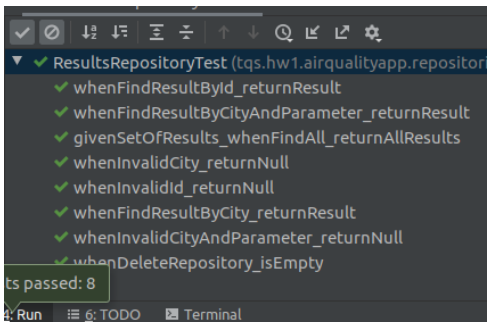
Todos os testes escritos passaram (alguns exemplos) :

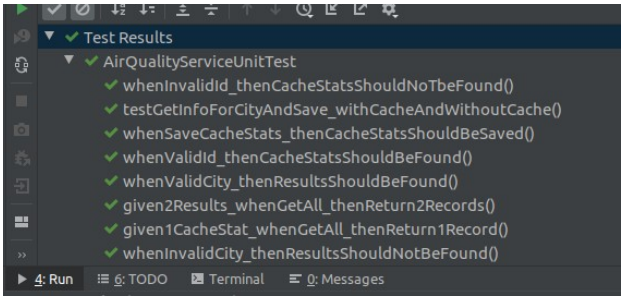


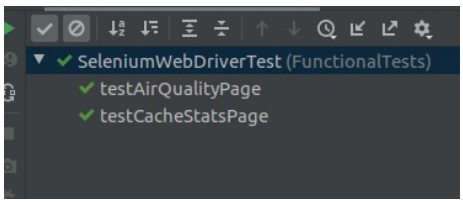


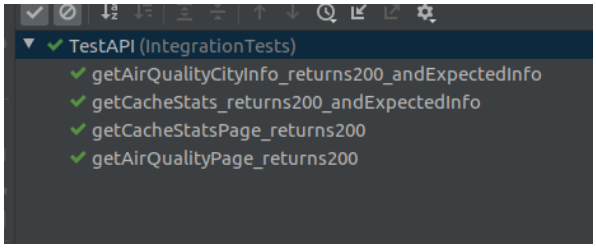






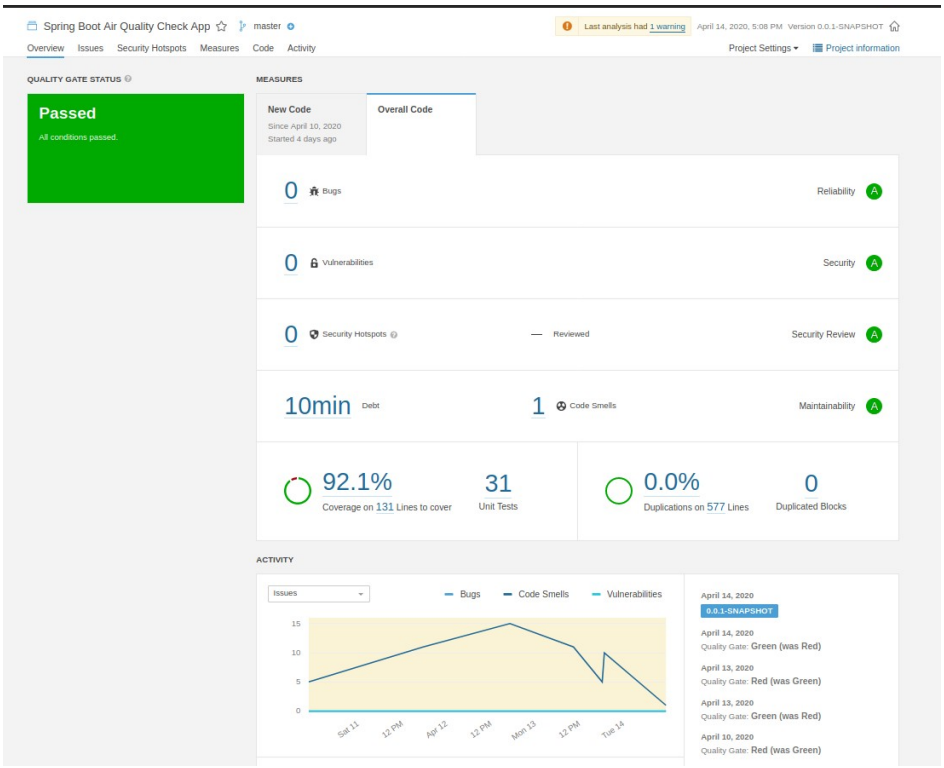






3.6 Sonar Qube

Foi configurado usando uma docker image.



Sem esta ferramenta, eu realmente nem teria ideia de algumas más práticas, tais como o uso de Thread.sleep() em testes, a não criação de excepções personalizadas ou escrita de código descenecessário.

4 Referências & recursos

Recursos

- Git repository: https://github.com/alina-yanchuk02/tqs_hm1
- Video demo no repositório: demo.avi

Materiais

OPENAQ API : https://docs.openaq.org/