Universidade de Aveiro

Weather-App

Relatório Final

Por: Rafael Direito - nº mec 84921

Data de preparação: Aveiro, 14 de Maio de 2019

Cadeira: Testes e Qualidade de Software

Corpo Docente: Professor Ilídio Castro Oliveira
Professor Cláudio Teixeira

Repositório GIT: https://github.com/rafadireito/WeatherApp-QA

Índice

Objetivos

Este projeto tem como objetivo a criação de diversos testes que permitam testar a qualidade de software de uma web app construída para o efeito. Desta forma, desenvolveu-se uma aplicação cujo objetivo é providenciar ao utilizador dados meteorológicos para uma determinada cidade, durante um determinado número de dias.

Esta aplicação é acompanhada de uma REST-API que poderá ser utilizada por entidades externas para realizar pedidos à nossa aplicação. Esta API fornece informações como boletins meteorológicos, análises de temperaturas e de humidades.

Relativamente aos testes que se pretendia desenvolver, foram postos em prática os seguintes:

- Testes Unitários;
- Testes Unitários com Isolamento de Mocks;
- Testes de Integração;
- Testes Funcionais;

Weather App

Esta aplicação consome 2 REST-APIs de forma a conseguir mostrar ao cliente um melhor boletim meteorológico. Desta forma, utilizaram-se as APIs do IPMA e do OpenWeather, que permitem consultar previsões meteorológicas para diversos dias. Esta aplicação foi desenvolvida com recurso a **Spring Boot**, **Html**, **JS** e **Css**.

Ainda neste tópico, a utilização de 2 APIs reduz o número de falhas, uma vez que sempre que não for possível obter dados de uma delas, temos a outra API que podemos consultar.

Foi construída uma interface Web bastante simples para demonstrar o funcionamento da aplicação.

A partir desta interface podemos pesquisar previsões meteorológicas para diversas cidades, para um número de dias variáveis.

Para além disto, foi, também, desenvolvida uma REST-API que permite que entidades externas consultem os dados que são divulgados para a WeatherApp.

Foi ainda desenvolvida uma cache, de forma a diminuir o número de pedidos às APIs externas.

Tal como descrito anteriormente, o grande objetivo desta aplicação é a criação de testes que comprovem o bom funcionamento de toda a aplicação.

Testes Unitários

Estes testes asseguram o bom funcionamento de funções simples, de forma a que se consiga garantir o bom funcionamento da aplicação.

Utilizando o **JUnit4**, desenvolveram-se inúmeros testes unitários, com especial relevo para:

- Testes sobre a gestão da cache;
- Testes sobre conversores;
- Testes sobre funções de cálculo;
- Testes sobre constantes;

```
@Test
public void checkDateToString()
{
    assertEquals("January, 1st", converters.dateToString("2019-01-01"));
    assertEquals("February, 2nd", converters.dateToString("2019-02-02") );
    assertEquals("March, 3rd", converters.dateToString("2019-03-03"));
    assertEquals("April, 21st",converters.dateToString("2019-04-21"));
    assertEquals("May, 22nd", converters.dateToString("2019-05-22"));
    assertEquals("June, 23rd", converters.dateToString("2019-06-23"));
    assertEquals("July, 31st",converters.dateToString("2019-07-31"));
    assertEquals("August, 12th", converters.dateToString("2019-08-12"));
    assertEquals("September, 12th", converters.dateToString("2019-09-12"));
    assertEquals("October, 12th",converters.dateToString("2019-10-12"));
    assertEquals("November, 12th", converters.dateToString("2019-11-12"));
    assertEquals("December, 12th", converters.dateToString("2019-12-12"));
    //default
    assertEquals("-, 12th", converters.dateToString("2019-00-12"));
}
```

Figura 1 - Teste Unitário sobre Conversores

```
@Test
public void getCache() throws InterruptedException
{
    MCache mCache = new MCache(30);

    mCache.add("key", "value", 1);
    Thread.sleep(2000);
    assertEquals(null, mCache.get("key"));
}


@Test
public void deleteCache() throws InterruptedException
{
    MCache mCache = new MCache(30);
    mCache.delete("key");
    assertEquals(false, mCache.delete("key"));
}


@Test
public void clearCache() throws InterruptedException
{
    MCache mCache = new MCache(30);
    mCache.add("key", "value", 100);
    mCache.clear();
    assertEquals(0, mCache.size());
}
```

Figura 2 - Teste Unitário sobre a Cache

Testes Unitários com Mocks

Os testes unitários com recurso a *Mocks* têm como objetivo isolar as dependências externas, de forma a testarmos apenas a nossa aplicação, de forma independente destas dependências.

Assim, utilizaram-se *Mocks* para testar todas as funções que faziam chamadas às APIs do IPMA e do OpenWeather, de forma a simular a resposta a estas chamadas.

A tecnologia utilizada para este tipo de testes foi o **Mockito**, paralelamente com o **JUnit4**.

Segue-se um dos testes desenvolvidos:

```
@Test
public void testGenerakInfoCacheEmpty() throws InterruptedException
{
    System.out.println("testGenerakInfoCacheEmpty");

    Map<String, Map<String,String>> ipma = new HashMap<>();
    Map<String,String> tmpIpma = new HashMap<>();
    tmpIpma.put("tMin", "10");
    tmpIpma.put("tMax", "15");
    tmpIpma.put("weather", "Clear Sky");
    tmpIpma.put("windDir", "N");
    tmpIpma.put("windIntensity", "10");
    tmpIpma.put("latitude", "111");
    tmpIpma.put("longitude", "222");
    tmpIpma.put("precipitationProb", "10");
    tmpIpma.put("pressure", "1000");
    tmpIpma.put("humidity", "1000");
    ipma.put("2019-01-01", tmpIpma);

    Map<String, Map<String,String>> open = new HashMap<>();
    Map<String,String> tmpOpen = new HashMap<>();
    tmpOpen.put("tMin", "10");
    tmpOpen.put("tMax", "15");
    tmpOpen.put("weather", "Clear Sky");
    tmpOpen.put("windDir", "N");
    tmpOpen.put("windIntensity", "10");
    tmpOpen.put("latitude", "111");
    tmpOpen.put("longitude", "222");
    tmpOpen.put("precipitationProb", "10");
    tmpOpen.put("pressure", "1000");
    tmpOpen.put("humidity", "1000");
    open.put("2019-01-02", tmpOpen);

    Mockito.when(ipmaCallsMockito.getForecast("Aveiro")).thenReturn(ipma);
    Mockito.when(openWeatherCallsMockito.getForecast("Aveiro")).thenReturn(open);

    Map<String, Map<String,String>> expected = new HashMap<>();
    expected.put("2019-01-01", tmpIpma);
    expected.put("2019-01-02", tmpOpen);


    MCache mCache = new MCache(30);
    forecastsResources.setmCache(mCache);

    assertEquals(expected, forecastsResources.generalInfo("Aveiro", 2));
}
```

Figura 3 - Teste Unitário com Isolamento de Dependências

Testes de Integração

De forma a testar a REST-API desenvolvida, é necessário desenvolver alguns testes de integração que verifiquem a informação que pode ser consultada através de cada *end-point*.

Assim, utilizando a tecnologia **REST-Assured** verificou-se se cada *end-point* estava a responder da forma correta, verificando-se alguns elementos do body do JSON com a resposta.

```
@Test
public void testGeneralForecast()
{
    Response  response =
            RestAssured.given().
                    when().
                    get("http://localhost:8080/api/general_info/Aveiro/3").then().
                    extract().response();


    int size = response.body().path("size()");
    assertEquals(3, size);

    DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
    Date date = new Date();
    String today = dateFormat.format(date).toString();
    assert(response.path(today) != null);
}
```

Figura 4 - Teste de Integração da API Rest

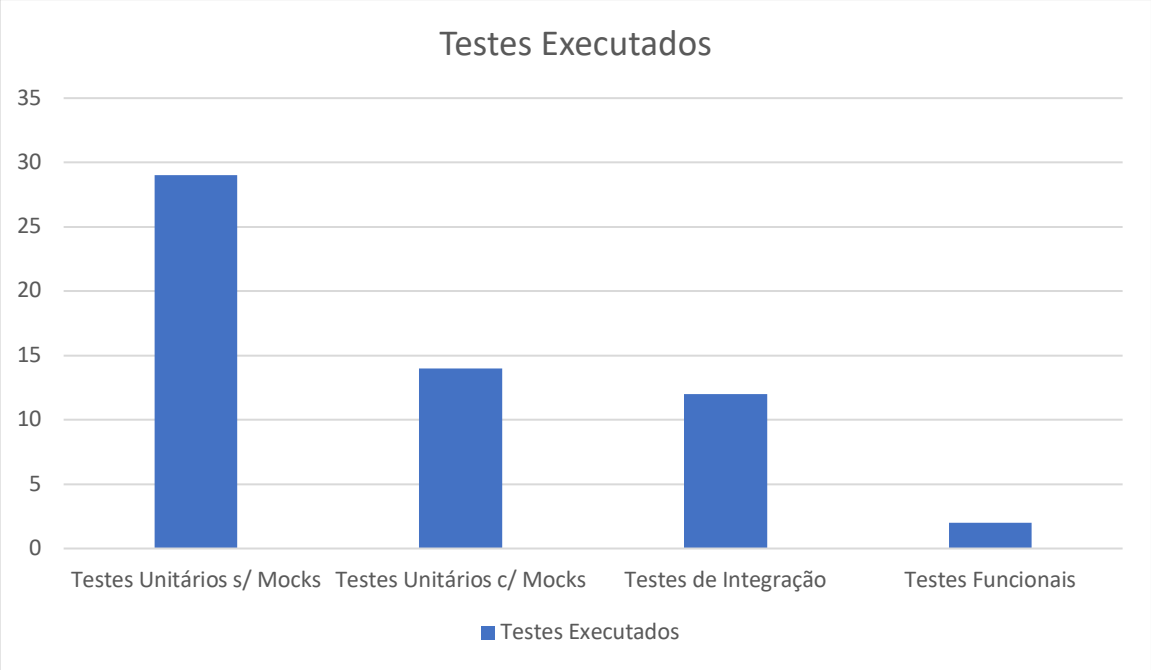
Testes Funcionais

De forma a testar a interface web da aplicação foram desenvolvidos diversos testes funcionais com recurso ao **Selenium**. Estes verificam se a interface web apresenta a informação correta. Por exemplo, quando pesquisamos por informação sobre Coimbra, o título da página deve mudar para "Coimbra". Para além disto, verifica-se também a correção dos valores das temperaturas apresentados.

```java
@Test
public void testSpecificForecast() throws Exception
{
  driver.get("http://localhost:8080/generalForecast");
  driver.findElement(By.id("inlineFormInputGroupUsername2")).click();
  driver.findElement(By.id("inlineFormInputGroupUsername2")).clear();
  driver.findElement(By.id("inlineFormInputGroupUsername2")).sendKeys("Aveiro");
  driver.findElement(By.id("inlineFormCustomSelectPref")).click();
  new Select(driver.findElement(By.id("inlineFormCustomSelectPref"))).selectByVisibleText("One");
  driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='Number of days'])[1]/following::button[1]")).click();
  String tMaxGeneral = driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='Tuesday'])[1]/following::span[2]")).getText();
  String tMinGeneral = driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='Tuesday'])[1]/following::span[3]")).getText();
  driver.findElement(By.name("Partly cloudy")).click();
  assertEquals("Aveiro",
          driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='WeatherApp'])[2]/following::h1[1]")).getText());
  assertEquals(converters.dateToString(today),
          driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='Aveiro'])[1]/following::h2[1]")).getText());
  assertEquals(tMinGeneral,
          driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='Partly cloudy'])[1]/following::span[1]")).getText());
  assertEquals(tMaxGeneral,
          driver.findElement(By.xpath("(.//*[normalize-space(text()) and normalize-space(.)='ºC'])[1]/following::span[1]")).getText());
}
```

Figura 5 - Teste Funcional à Área de Informações Específicas

Estatísticas Gerais dos Testes:



SonarQube

O SonarQube caracteriza-se por ser uma ferramenta de análise estática de código. Neste, podemos definir diversas métricas para avaliar a qualidade de uma *build* do nosso projeto, permitindo-nos saber se este cumpre com os parâmetros de qualidade definidos.

Este analisa diversas métricas, como:

- Reliability
- Security
- Maintainability
- Coverage
- Duplications

Esta ferramenta permite, também, fazer o tracking temporal da evolução do nosso projeto - algo deveras interessante.

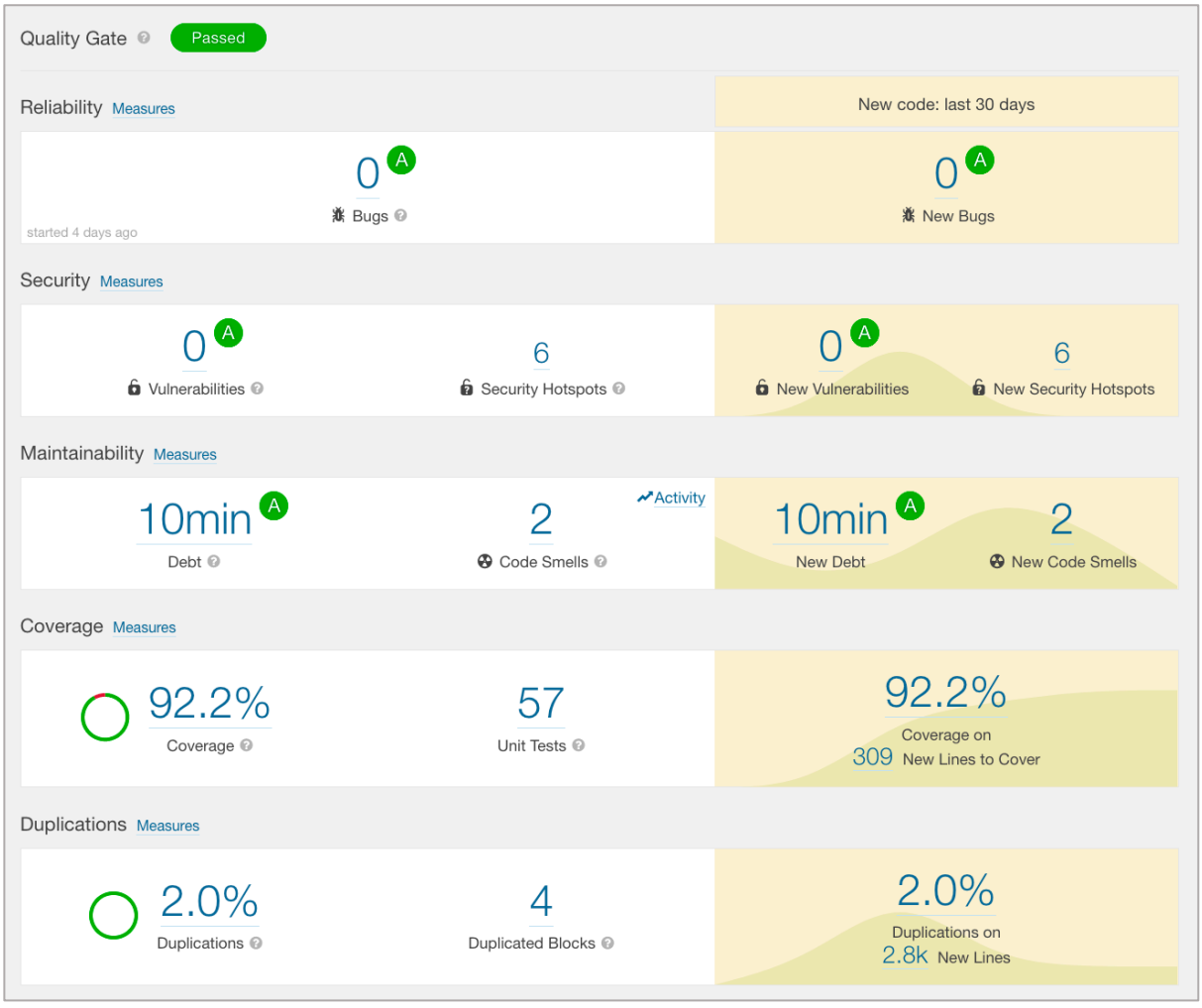


Figura 6 - Dashboard do SonarQube

Segue-se a evolução temporal do número de *bugs* e *code smells* do projeto:

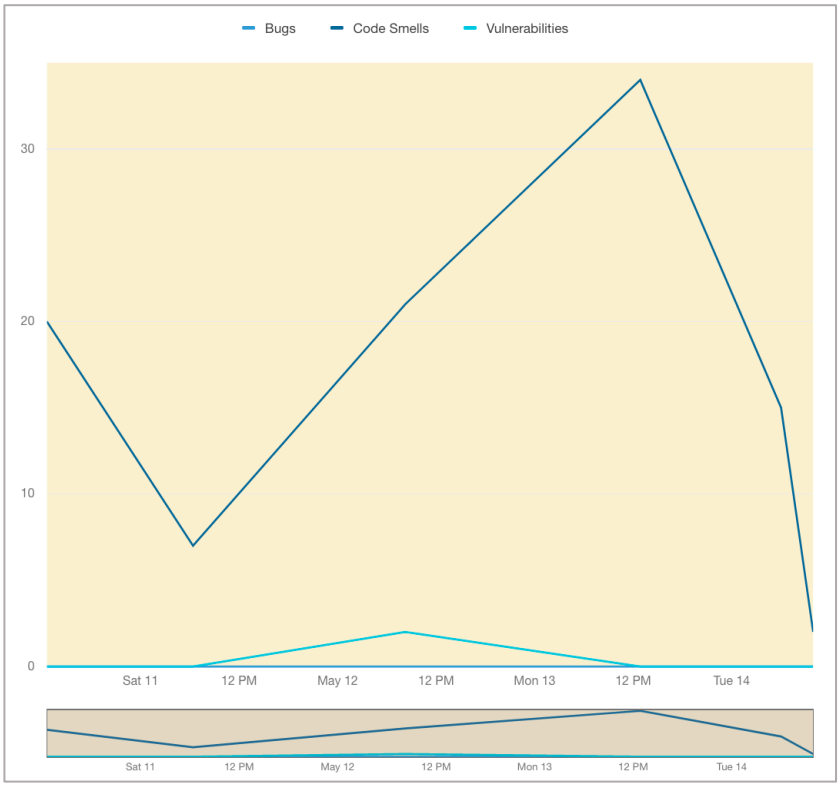


Figura 7 - Evolução dos Issues – SonarQube

Infelizmente, esta ferramenta tem algumas falhas que, por vezes, levam a uma análise errática. Podemos tomar como exemplo as duplicações de código, uma vez que o SonarQube reconhece um *switch* e um conjunto de *ifs* e *elses* como blocos de código duplicados – daí os 2% de duplicação de código.

Funcionalidades Extra

De forma a complementar o trabalho desenvolvido implementaram-se as seguintes funcionalidades extra:

- Pesquisa por número variável de dias de previsão;
- Consulta de 2 APIs externas;
- Utilização do Swagger para documentação da API;

A implementação das 2 APIs permite a obtenção de valores mais concretos, uma vez que, caso as 2 APIs obtenham dados, é feita uma média entre os valores de ambas. Caso uma das APIs não permita a obtenção de dados, serão mostrados os dados provenientes da outra API.

Relativamente aos testes da segunda API (OpenWeather) este não foram desenvolvidos, uma vez que são bastante semelhantes àqueles feitos para a API primária (IPMA), pelo que não acrescentam qualquer aprendizagem. Para além disto, uma vez que a API do OpenWeather é bastante mais complexa que a do IPMA, seria necessário um grande investimento temporal para desenvolver tais testes.

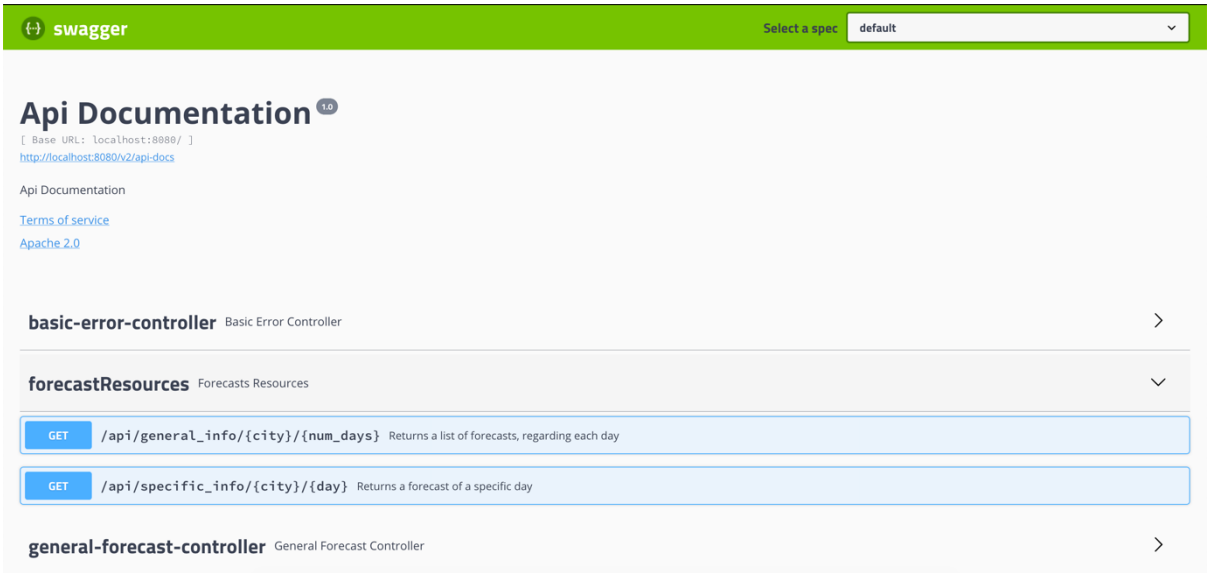


Figura 8 - Documentação utilizando Swagger

É possível aceder ao swagger através de *<baseUrl>/ swagger-ui.html#/* .

Conclusões e Aspetos de Melhoria

Relativamente aos testes de integração estes apenas podem ser executados com o projeto a correr, de forma a testar os *end-points* disponibilizados. Isto pode causar algum desconforto, pelo que poderá ser um aspeto a melhorar.

Tal como mencionado anteriormente, o SonarQube apresenta também alguns erros, pelo que pode fornecer algumas informações menos corretas ao programador, que acaba por alterar o seu código sem necessidade de tal.

Poderíamos, também, desenvolver todos os testes relativos à API do OpenWeather, o que garantiria uma melhor qualidade de software e de tratamento de erros.

Para além destas questões, consideram-se os objetivos como cumpridos, uma vez que foram desenvolvidos todos os testes pedidos e foram implementadas todas as funcionalidades extra.

Recursos

Repositório GIT: https://github.com/rafadireito/WeatherApp-QA