# HW1: Mid-term assignment report

*Guilherme Antunes [103600]*, v2023-04-11-04-11

## Índice



# 1 Introdução

## 1.1 Visão geral do trabalho

Este relatório visa apresentar o projeto intermédio individual de TQS, quer quanto à implementação de funcionalidades, quer quanto às estratégias de garantia de qualidade.

O objetivo pretendido com este trabalho foi desenvolver uma aplicação fullstack cuja principal função é apresentar dados de qualidade de ar, para uma dada cidade, ao utilizador, tendo como fonte de dados fidedigna uma API externa fidedigna disponível online. Para além de a desenvolver foi também requerida a realização de diversos testes para garantia de qualidade da implementação. O backend da aplicação foi desenvolvido em SpringBoot enquanto o frontend foi desenvolvido em HTML, CSS e JavaScript. A API externa escolhida foi a weatherbit.io que permite apresentar vários tipos de dados atuais, passados e futuros.

### 1.2 Limitações atuais

Apesar de estar preparada para tal, podemos apontar como limitações da aplicação a utilização de apenas uma única fonte de dados, não sendo assim capaz de retornar dados em caso de falha da API principal e limitando a aplicação apenas aos tipos de dados retornados por essa mesma API. Também pode acontecer que aquando da pesquisa de uma cidade pelo seu nome seja retornada outra diferente daquela que pretendíamos, no entanto, esta limitação geralmente pode ser contornada acrescentando a sigla do país à frente ou o próprio nome, como uma pesquisa normal.

Idealmente também teria um pipeline de integração continua que não foi possível implementar pelo grau de exigência que envolvia para o tempo disponível.

## 2 Especificação do Produto

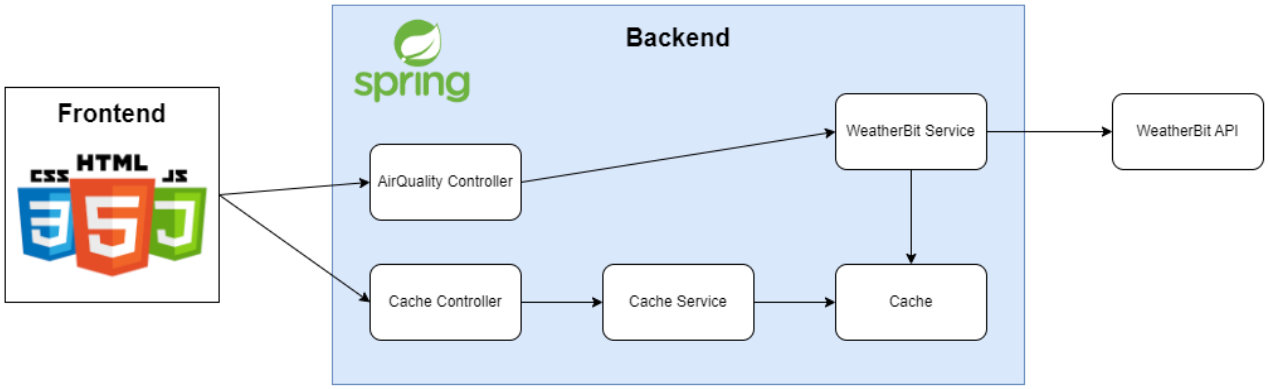
### 2.1 Espectro funcional e Interações suportadas

A aplicação destina-se ao utilizador casual, que tenha curiosidade ou necessidade de saber quais os níveis de qualidade do ar para um determinado período numa determinada localização e que precisa de uma interface mais amiga do que um link memorizado e uma resposta em json, que é muito pouco user friendly.

Como já referido a aplicação tem como principais funcionalidades apresentar dados para o "agora", para um período específico do passado ou até uma previsão futura para uma dada localidade. Mais especificamente apresenta dados de AQI (Índice de qualidade do ar), O3, SO2, NO2, CO2, PM10, PM25 e, no caso dos dados atuais, níveis de diferentes tipos de pólen que existem no ar e o mais predominante. Também apresenta uma aba com estatísticas de Cache em resposta ao requisito do enunciado.

### 2.2 Arquitetura do sistema

Como já referido o frontend da aplicação foi desenvolvido em HTML, CSS e JavaScript e o backend inteiramente desenvolvido em SpringBoot. Por sua vez um dos Services faz requisições à API externa WeatherBit.io




### 2.3 API para desenvolvedores

- GET **/api/airquality/current/{city}** – retorna dados de qualidade do ar atual para a **city** enviada

- GET **/api/airquality/forecast/{city}** – retorna dados de qualidade do ar para os próximos dias hora a hora para a **city** enviada

- GET **/api/airquality/history/{city}** – retorna dados de qualidade do ar para os últimos dias hora a hora para a **city** enviada

- GET **/api/aqirquality/history/{start_date}/{end_date}** – retorna dados de qualidade do ar para um dado período de tempo para a **city** enviada

- GET **/api/cache/stats** – retorna as estatísticas da cache (número de requests, hits e misses)

# 3 Garantia de Qualidade

## 3.1 Estratégia utilizada nos testes

A aplicação foi toda feita sem testes a acompanhar o desenvolvimento. O desenvolvimento de testes foi efetuado após a conclusão da app, sendo usados para corrigir implementações. Os testes foram realizados pela ordem UnitTests, ServiceTests, RestControllerTests, RestControllerIntegrationTests e CucumberTests. Foram usadas ferramentas como o JUnit, AssertJ, Mockito, SpringBoot MockMvc e aplicou-se BDD para o frontend com recurso a Selenium WebDriver.

## 3.2 Testes Unitários e de Integração

Foram realizados UnitTests às classes serializers, aos modelos relacionados com a Cache e às funções estilo Utils usando maioritariamente AssertJ e JUnit, aos Services e aos RestControllers, usando relevantemente Mockito. Também foram realizados IntegrationTests aos RestControllers usando SpringBootTest e MockMvc.

```
@Test
public void testParseFromWBCurrent() {
    String json = "{\"city_name\":\"Castelo Branco\",\"co
    WBCurrentResponse data = parseWBCurrent(json);
    assertEquals("Castelo Branco", data.city_name);
    assertEquals("PT", data.country_code);
    assertEquals(39.82219, data.lat);
    assertEquals(-7.49087, data.lon);
    assertThat(data.data).isNotNull();
    String json2 = "\"city_name\":\"Castelo Branco\",\"co
    WBCurrentResponse data2 = parseWBCurrent(json2);
    assertEquals(null, data2);
}
```

```java
@Test
public void testWBCurrent() throws IOException {
    String json = "{\"city_name\":\"Castelo Branco\",\"country_code\":\"PT\",\"data\
    WBCurrentResponse response = mapper.readValue(json, WBCurrentResponse.class);
    assertEquals("Castelo Branco", response.city_name);
    assertEquals("PT", response.country_code);
    assertEquals(39.82219, response.lat);
    assertEquals(-7.49087, response.lon);
    assertEquals(1, response.data.size());
    assertEquals(52, response.data.get(0).aqi);
    assertEquals(200.2716, response.data.get(0).co);
    assertEquals(1, response.data.get(0).mold_level);
    assertEquals(2, response.data.get(0).no2);
    assertEquals(112, response.data.get(0).o3);
    assertEquals(7.1234508, response.data.get(0).pm10);
    assertEquals(1.934503, response.data.get(0).pm25);
    assertEquals(2, response.data.get(0).pollen_level_grass);
    assertEquals(2, response.data.get(0).pollen_level_tree);
    assertEquals(2, response.data.get(0).pollen_level_weed);
    assertEquals("Trees", response.data.get(0).predominant_pollen_type);
    assertEquals(0.1527369, response.data.get(0).so2);
}
```

```java
@BeforeEach
public void setUp() {
    Cache.reset();
    assertThat(Cache.getCache()).isEmpty();
}

@Test
public void testAdd() {
    assertThat(Cache.getCache()).isEmpty();
    Cache.add("pesquisa", SearchType.CURRENT, DataSource.WEATHERBIT, data);
    assertThat(Cache.getCache()).hasSize(1);
}

@Test
public void testGet() {
    Cache.add("pesquisa", SearchType.CURRENT, DataSource.WEATHERBIT, data);
    assertThat(Cache.get("pesquisa", SearchType.CURRENT, DataSource.WEATHERBIT)).isEqualTo(data);
    Cache.add("pesquisa", SearchType.CURRENT, DataSource.WEATHERBIT, data2, 0);
    assertThat(Cache.get("pesquisa", SearchType.CURRENT, DataSource.WEATHERBIT)).isEqualTo(data2);
    assertThat(Cache.get("pesquisa", SearchType.FORECAST, DataSource.WEATHERBIT)).isNull();
    assertThat(Cache.get("pesquisa", SearchType.CURRENT, DataSource.OPENWEATHER)).isNull();;
    assertThat(Cache.get(null, SearchType.CURRENT, DataSource.WEATHERBIT)).isNull();;
}
```

```java
@Test
public void testGetData() {
    WBForecastResponse data = new WBForecastResponse();
    CacheObject cacheObject = new CacheObject(data, null, null);
    assertEquals(data, cacheObject.getData());
}
```

```java
@Mock
private RestTemplate restTemplate;

@InjectMocks
private WeatherBitService weatherBitService;

final String KEY = "&key=e5c32e0651744cdf96f8c8b0407edd4a";
final String CURRENT_URL = "https://api.weatherbit.io/v2.0/current/airquality";
final String HISTORY_URL = "https://api.weatherbit.io/v2.0/history/airquality";
final String FORECAST_URL = "https://api.weatherbit.io/v2.0/forecast/airquality";
final String CITY = "?city=Castelo Branco";
final String INVALID_CITY = "?city=invalidCt";

final String VALID_CURRENT_RESPONSE = "{\"city_name\":\"Castelo Branco\",\"country_code\":\"PT\",\"d
final String VALID_HISTORY_RESPONSE = "{\"city_name\":\"Castelo Branco\",\"country_code\":\"PT\",\"d
final String VALID_HISTORY_RESPONSE_WITH_DATES = "{\"city_name\":\"Castelo Branco\",\"country_code\"
final String VALID_FORECAST_RESPONSE = "{\"city_name\":\"Castelo Branco\",\"country_code\":\"PT\",\"
final String INVALID_HISTORY_RESPONSE_WITH_DATES = "{\"error\":\"end_date is before start_date.\"}";
final String INVALID_RESPONSE = "";

@Test
void whenRequestingCurrentWeather_thenReturnCorrectCurrentWeather() {
    Mockito.when(restTemplate.getForEntity(CURRENT_URL + CITY + KEY, String.class))
            .thenReturn(ResponseEntity.status(200).body(VALID_CURRENT_RESPONSE));
    ResponseEntity<WBResponse> response = weatherBitService.getCurrent("Castelo Branco");
    assertEquals(200, response.getStatusCode().value());
    WBCurrentResponse correct_response = WeatherBitService.parseWBCurrent(VALID_CURRENT_RESPONSE);
    assertEquals(correct_response, response.getBody());
    ResponseEntity<WBResponse> response2 = weatherBitService.getCurrent("Castelo Branco");
    assertEquals(200, response2.getStatusCode().value());
    assertEquals(correct_response, response2.getBody());
}
```

```java
private CacheService cacheService = new CacheService();

@BeforeEach
public void setUp() {
    Cache.reset();
}

@Test
public void testGetCacheStats() {
    assertEquals(0, cacheService.getCacheStats().getBody().hits);
    assertEquals(0, cacheService.getCacheStats().getBody().misses);
    assertEquals(0, cacheService.getCacheStats().getBody().requests);
    WBCurrentResponse current = new WBCurrentResponse();
    Cache.add("Lisbon", SearchType.CURRENT, DataSource.WEATHERBIT, current);
    Cache.add("Porto", SearchType.HISTORY, DataSource.WEATHERBIT, current);
    Cache.get("Coimbra", SearchType.CURRENT, DataSource.WEATHERBIT);
    Cache.get("Lisbon", SearchType.CURRENT, DataSource.WEATHERBIT);
    Cache.get("Porto", SearchType.HISTORY, DataSource.WEATHERBIT);
    Cache.get("Porto", SearchType.HISTORY, DataSource.OPENWEATHER);
    Cache.get("Porto", SearchType.CURRENT, DataSource.WEATHERBIT);
    assertEquals(2, cacheService.getCacheStats().getBody().hits);
    assertEquals(3, cacheService.getCacheStats().getBody().misses);
    assertEquals(5, cacheService.getCacheStats().getBody().requests);
}
```

```java
@WebMvcTest(CacheController.class)
public class CacheControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private CacheService cacheService;

    @Test
    public void whenGetCacheStats_thenReturnCacheStats() throws Exception {
        CacheJson cache = new CacheJson(2, 1, 1);
        Mockito.when(cacheService.getCacheStats()).thenReturn(ResponseEntity.ok(cache));

        MvcResult result = mvc.perform(
                get("/api/cache/stats").contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andReturn();
        String response = result.getResponse().getContentAsString();
        ObjectMapper mapper = new ObjectMapper();
        CacheJson new_cache = mapper.readValue(response, CacheJson.class);
        assertEquals(new_cache, cache);
        verify(cacheService, times(1)).getCacheStats();
    }
}
```

```java
@SpringBootTest(webEnvironment = WebEnvironment.MOCK, classes = AirQualityApplication.class)
@AutoConfigureMockMvc
public class CacheControllerIT {

    @Autowired
    private MockMvc mvc;

    @BeforeEach
    public void resetCache() {
        Cache.reset();
    }

    @Test
    public void whenGetCacheStats_thenReturnCacheStats() throws Exception {
        Cache.get("Coimbra", SearchType.CURRENT, DataSource.WEATHERBIT);
        mvc.perform(get("/api/cache/stats"))
                .andExpect(status().isOk())
                .andExpect(content().contentType("application/json"))
                .andExpect(jsonPath("$.requests", is(1)))
                .andExpect(jsonPath("$.hits", is(0)))
                .andExpect(jsonPath("$.misses", is(1)));
    }
}
```

```java
@Autowired
private MockMvc mvc;

@MockBean
private WeatherBitService weatherBitService;

private final ObjectMapper mapper = new ObjectMapper();

final String CITY = "Castelo Branco";
final String INVALID_CITY = "invalidCt";
final WBResponse CURRENT_CITY_RESPONSE = WeatherBitService.parseWBCurrent(
        "{\"city_name\":\"Castelo Branco\",\"country_code\":\"PT\",\"lat\":39.82219,\"lon\":-7.49087,\"data\"
final WBResponse FORECAST_CITY_RESPONSE = WeatherBitService.parseWBForecast(
        "{\"city_name\":\"Castelo Branco\",\"country_code\":\"PT\",\"lat\":39.82219,\"lon\":-7.49087,\"data\"
final WBResponse HISTORY_CITY_RESPONSE = WeatherBitService.parseWBHistory(
        "{\"city_name\":\"Castelo Branco\",\"country_code\":\"PT\",\"lat\":39.82219,\"lon\":-7.49087,\"data\"
final WBResponse HISTORY_WITH_DATES_CITY_RESPONSE = WeatherBitService.parseWBHistory("{\"city_name\":\"Caste
final WBResponse INVALID_CITY_RESPONSE = null;
final String DATE1 = "2023-04-09";
final String DATE2 = "2023-04-10";

@Test
public void whenGetCurrentAirQuality_thenReturnData() throws Exception {
    Mockito.when(weatherBitService.getCurrent(CITY)).thenReturn(ResponseEntity.ok(CURRENT_CITY_RESPONSE));
    MvcResult result = mvc.perform(
            get("/api/airquality/current/" + CITY).contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andReturn();
    String response = result.getResponse().getContentAsString();
    WBCurrentResponse new_response = mapper.readValue(response, WBCurrentResponse.class);
    assertEquals(CURRENT_CITY_RESPONSE, new_response);
    verify(weatherBitService, times(1)).getCurrent(CITY);
}
```

```java
@SpringBootTest(webEnvironment = WebEnvironment.MOCK, classes = AirQualityApplication.class)
@AutoConfigureMockMvc
public class AirQualityControllerIT {

    @Autowired
    private MockMvc mvc;

    @Test
    public void testGetCurrentAirQuality() throws Exception {
        mvc.perform(get("/api/airquality/current/Porto"))
                .andExpect(status().isOk())
                .andExpect(content().contentType("application/json"))
                .andExpect(jsonPath("$.city_name", is("Porto")))
                .andExpect(jsonPath("$.country_code", is("PT")))
                .andExpect(jsonPath("$.lat", notNullValue()))
                .andExpect(jsonPath("$.lon", notNullValue()))
                .andExpect(jsonPath("$.data", notNullValue()))
                .andExpect(jsonPath("$.data[0].aqi", notNullValue()))
                .andExpect(jsonPath("$.data[0].co", notNullValue()))
                .andExpect(jsonPath("$.data[0].no2", notNullValue()))
                .andExpect(jsonPath("$.data[0].o3", notNullValue()))
                .andExpect(jsonPath("$.data[0].so2", notNullValue()))
                .andExpect(jsonPath("$.data[0].pm25", notNullValue()))
                .andExpect(jsonPath("$.data[0].pm10", notNullValue()))
                .andExpect(jsonPath("$.data[0].mold_level", notNullValue()))
                .andExpect(jsonPath("$.data[0].pollen_level_grass", notNullValue()))
                .andExpect(jsonPath("$.data[0].pollen_level_tree", notNullValue()))
                .andExpect(jsonPath("$.data[0].pollen_level_weed", notNullValue()))
                .andExpect(jsonPath("$.data[0].predominant_pollen_type", notNullValue()));
    }
```

### 3.3 Testes Funcionais

Para os testes funcionais foi utilizado uma abordagem Behavior-Driven Development (BDD), recorrendo ao Selenium WebDriver. Para isso, foram escrito um cenário com várias tarefas no ficheiro webpage.feature, onde o teste vai realizar essas tarefas e concluir então os cenários descritos.

```
Feature: Webpage
    Scenario: Navigate through the Air Quality page
        Given I access the "localhost:8080" url
        When I click on the "Current" link
        And I search for the "Current" data for the city "Castelo Branco"
        Then I should see the "Current" data for the city "Castelo Branco"
        When I click on the "Forecast" link
        And I search for the "Forecast" data for the city "Castelo Branco"
        Then I should see the "Forecast" data for the city "Castelo Branco"
        When I click on the "History" link
        And I search for the "History" data for the city "Castelo Branco"
        Then I should see the "History" data for the city "Castelo Branco"
        When I click on the "Cache Statistics" link
        Then I should see the Cache Statistics data
        Then I quit the browser
```

```
public class WebPageSteps {

    private WebDriver driver;

    @Given("I access the {string} url")
    public void i_access_the_url(String url) {
        WebDriverManager.firefoxdriver().setup();
        driver = new FirefoxDriver();
        driver.manage().window().maximize();
        driver.get(url);
    }

    @When("I click on the {string} link")
    public void i_click_on_the_link(String link) {
        driver.findElement(By.linkText(link)).click();
        if (link.equals("Cache Statistics")) {
            {
                WebDriverWait wait = new WebDriverWait(driver, java.time.Duration.ofSeconds(3));
                wait.until(ExpectedConditions.textMatches(By.id("requests"), Pattern.compile("^((?!N/A).)*$")));
            }
        }
    }
```

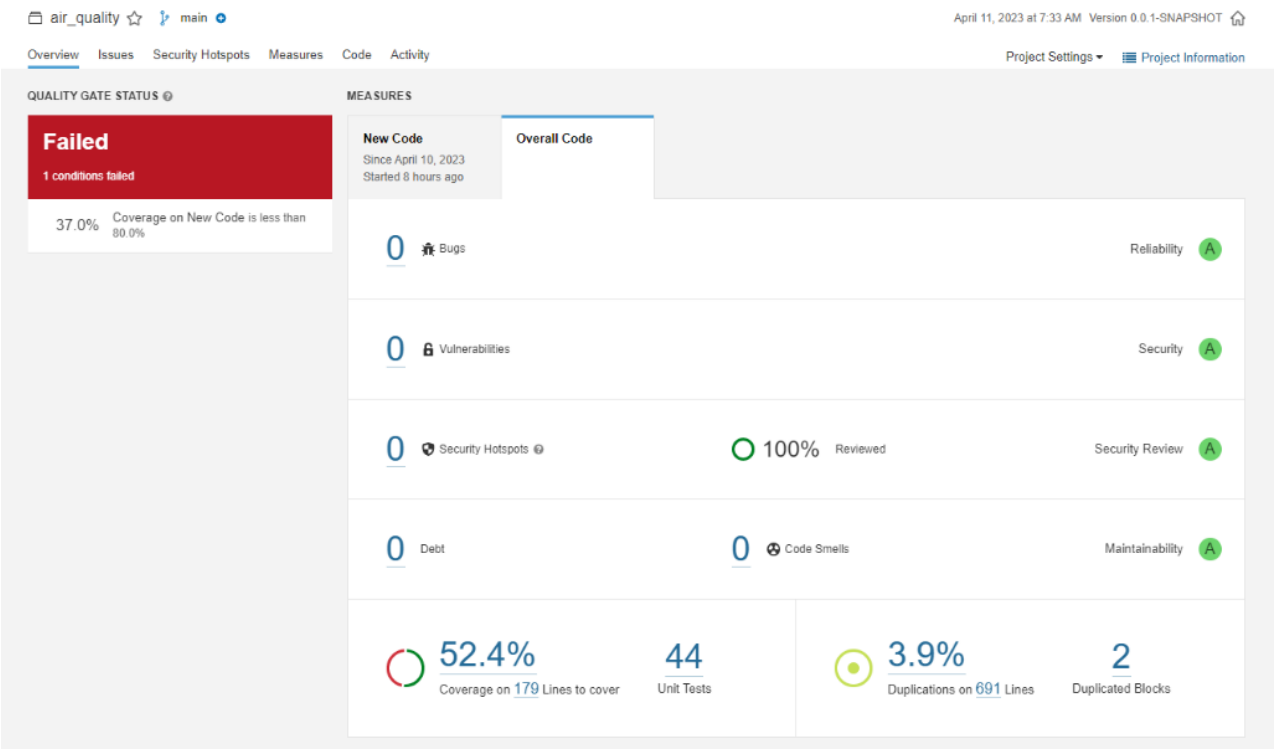
```java
@And("I search for the {string} data for the city {string}")
public void i_search_for_the_data_in_city(String data, String city) {
    driver.findElement(By.linkText(data)).click();
    driver.findElement(By.id("city_" + data.toLowerCase())).sendKeys(city);
    driver.findElement(By.id(data.toLowerCase() + "_btn")).click();
    {
        WebDriverWait wait = new WebDriverWait(driver, java.time.Duration.ofSeconds(10));
        WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("results")));

        wait.until(ExpectedConditions
                .not(ExpectedConditions.attributeContains(element, "class", "display: none")));
    }
}
```

```java
@Then("I should see the {string} data for the city {string}")
public void i_should_see_the_data(String data, String city) {
    String ct;
    switch (data) {
        case "Current":
            ct = driver.findElement(By.cssSelector("p:nth-child(1)")).getText();
            assertThat(ct, containsString(city));
            assertThat(driver.findElement(By.cssSelector("p:nth-child(3)")).getText(), is("Source: WeatherBit"));
            String aqi = driver.findElement(By.cssSelector("table > tr > td:nth-child(1)")).getText();
            assertThat(Integer.parseInt(aqi), IsInstanceOf.instanceOf(Integer.class));
            String o3 = driver.findElement(By.cssSelector("table > tr > td:nth-child(2)")).getText();
            assertThat(Double.parseDouble(o3), IsInstanceOf.instanceOf(Double.class));
            String so2 = driver.findElement(By.cssSelector("table > tr > td:nth-child(3)")).getText();
            assertThat(Double.parseDouble(so2), IsInstanceOf.instanceOf(Double.class));
            String no2 = driver.findElement(By.cssSelector("td:nth-child(4)")).getText();
            assertThat(Double.parseDouble(no2), IsInstanceOf.instanceOf(Double.class));
            String co = driver.findElement(By.cssSelector("td:nth-child(5)")).getText();
            assertThat(Double.parseDouble(co), IsInstanceOf.instanceOf(Double.class));
            String pm10 = driver.findElement(By.cssSelector("td:nth-child(6)")).getText();
            assertThat(Double.parseDouble(pm10), IsInstanceOf.instanceOf(Double.class));
            String pm25 = driver.findElement(By.cssSelector("td:nth-child(7)")).getText();
            assertThat(Double.parseDouble(pm25), IsInstanceOf.instanceOf(Double.class));
            String moldLevel = driver.findElement(By.cssSelector("table:nth-child(5) tr:nth-child(2) > td"))
                    .getText();
            assertThat(Integer.parseInt(moldLevel), IsInstanceOf.instanceOf(Integer.class));
            String PollenLevelGrass = driver.findElement(By.cssSelector("tr:nth-child(3) > td")).getText();
            assertThat(Integer.parseInt(PollenLevelGrass), IsInstanceOf.instanceOf(Integer.class));
            String PollenLevelWeed = driver.findElement(By.cssSelector("tr:nth-child(4) > td")).getText();
            assertThat(Integer.parseInt(PollenLevelWeed), IsInstanceOf.instanceOf(Integer.class));
            String PollenLevelTree = driver.findElement(By.cssSelector("tr:nth-child(4) > td")).getText();
            assertThat(Integer.parseInt(PollenLevelTree), IsInstanceOf.instanceOf(Integer.class));
            String PredominantPollenType = driver.findElement(By.cssSelector("tr:nth-child(5) > td")).getText();
            assertThat(PredominantPollenType, IsIn.in(Arrays.asList("Molds", "Grasses", "Weeds", "Trees")));
            break;
```

### 3.4 Análise da quantidade do código

Para efetuar a análise estática do código foi utilizada a ferramenta SonarQube que terminou com a seguinte avalização.




Após uma análise dos resultados podemos desvalorizar, de certo modo, a baixa percentagem de coverage uma vez que as classes que estão assinaladas com pouco coverage são classes que funcionam quase única e exclusivamente como serializers. A necessidade de implementar métodos hashCode() e equals() para facilitar a criação e desenvolvimento de testes levou a que fossem necessários muitos mais testes para garantir a cobertura total das funções implementadas. No entanto como são funções implementadas pelo marcador @Data do lombok acredito que não seja preciso realizar testes para garantir que funcionam corretamente.

# 4 Recursos e Referências

**Recursos do Projeto**

| Recurso: | URL/localização: |
|---|---|
| Repositório GitHub | https://github.com/SysteM1922/TQS_103600/tree/main/HW1 |
| Vídeo de demonstração | demo.mp4 na pasta HW1 do repositório |

**Materiais de Referência**

WeatherBit.io - https://www.weatherbit.io/api/air-quality-api