HW1: Mid-term assignment report

Catarina Teves Martins da Costa [103696], v2023-04-10

1 Introduction

1.1 Overview of the work

This report presents the midterm individual project required for TQS, covering both the software product features and the adopted quality assurance strategy.

The main goal of this homework was to create a REST-API service, with the implementation of test to verify the if the applications and it's functionalities are working properly. The test that were included were:

- Unit Tests;
- Service Level tests;
- Integrational tests;
- Functional tests;

The AirQuality project, should allow the user to check the details on air quality for a certain region/city, for the current day and forecast for upcoming days. The air quality is characterized by the AQI (Air Quality Index) and some concentration levels (CO, O3, SO2, NO2), as well as the predominant pollen type of that specific city.

The external API that was used to gather all the data regarding the air quality for each city, was https://www.weatherbit.io.

As the homework suggested it was also implemented a local cache, making repeated calls being stored in a 'TIMETOLIVE = 100L'. There is also a thread that

clears the cache in ‘TIMETOLIVE * 1000L’. The cache also shows details as requests, hits and misses.

All the data displayed is provided by de REST API, in JSON format.

Regarding the cache:

- Cache.js: this is the file where the cache is implement, this type of cache does not have a size capacity, but has a Thread that cleans form time to time all the data that is stored in the cache, wich means all the cities that are stored for more then the difined time, are removed. This cache, works almost as a stack.
- ObjectCache.js: this java file is responsible to have the function that sees if the data is expired or not.

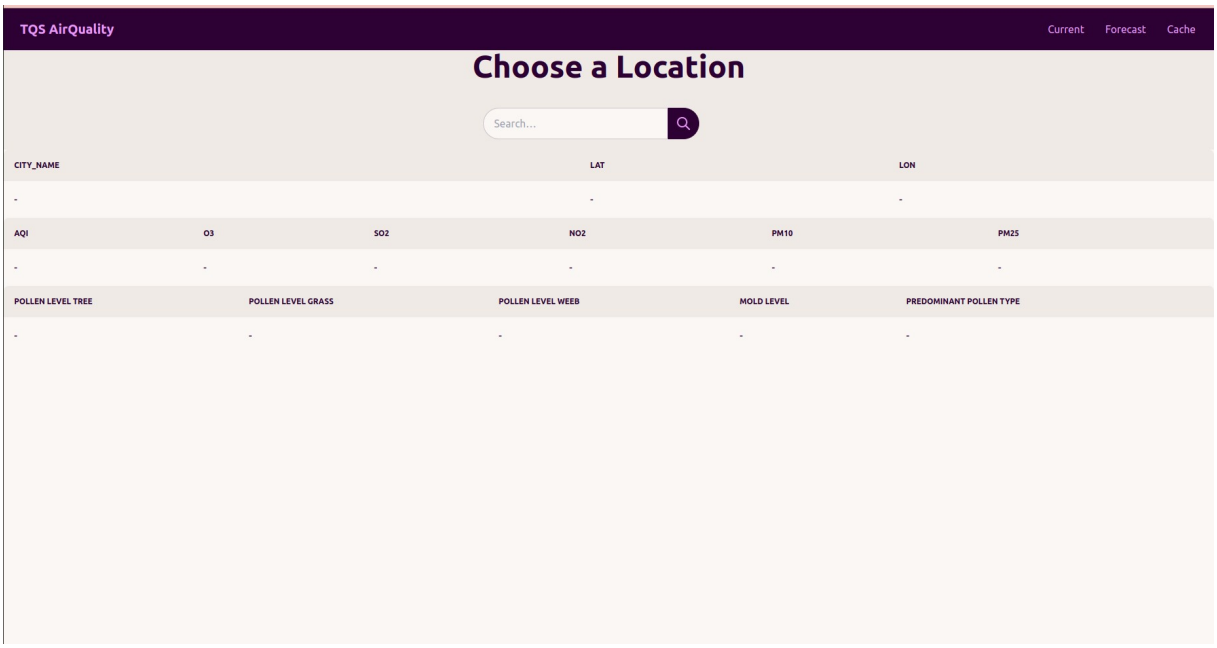
1.2 Current limitations

This work only uses on external API, so is fully dependent on it to work and receive data. To bypass this, in the future will be implemented another external API.

Also, besides not being implemented now, the work does no present historical data and you can’t search by coordinates. These are some future features that will be implement.

2 Product specification

2.1 Functional scope and supported interactions

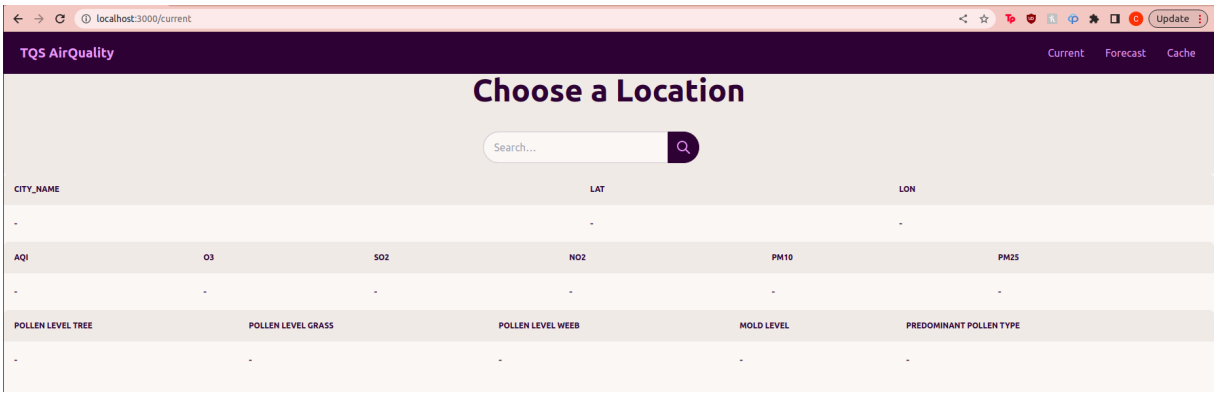


The main user of this platform, is someone who is curious to see the air quality conditions, in certain areas of the globe. So basically, anyone can access this platform.

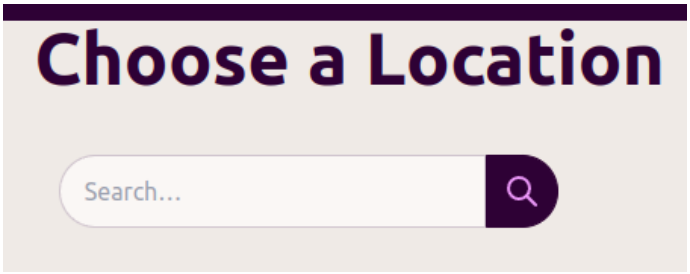
The user interacts with the platform through a single page application, to make it more intuitive.

There is a Navbar where the user can chose witch data to search for: current, forecast or cache.

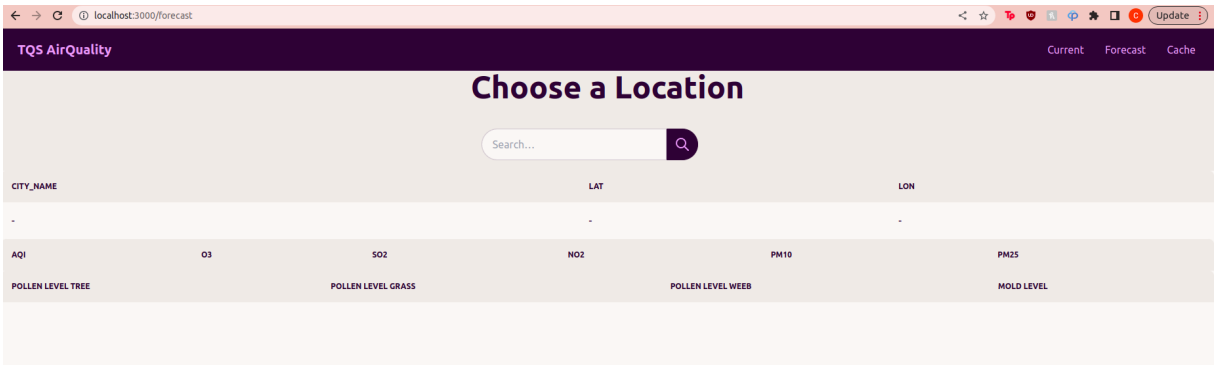
On click of the Current button in the navbar, the user is send to the Current page.



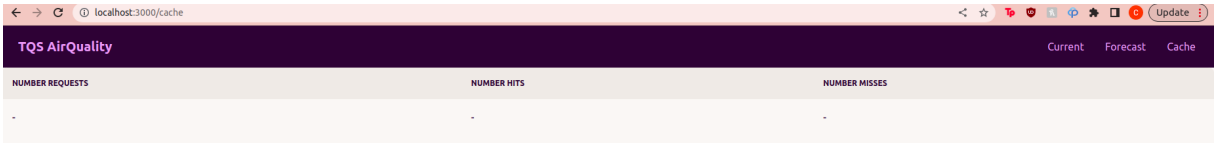
Here he can input on the search bar the location of his like.



The same method aplies to the forecast page.

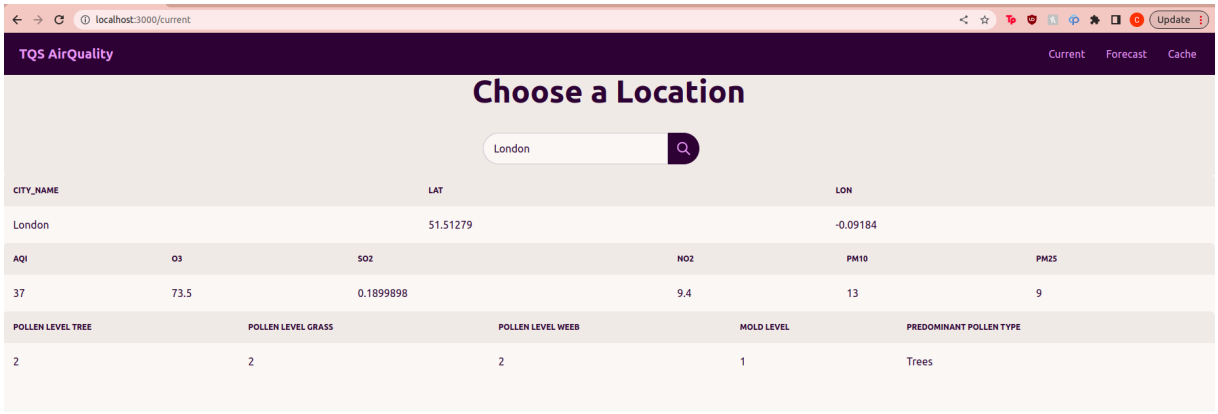


If he wants to see the cache data, we can click on the Cache button in the navbar, and he will be faced with the results.

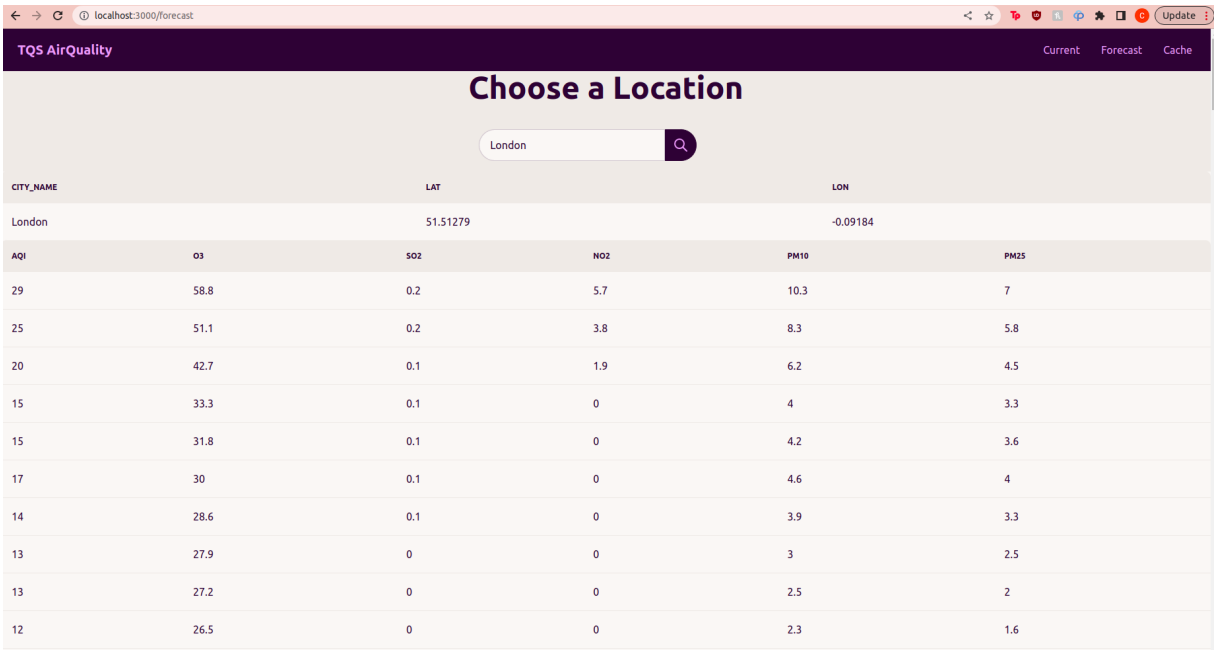


Exemple: If we searched for the city London:

Current Page:

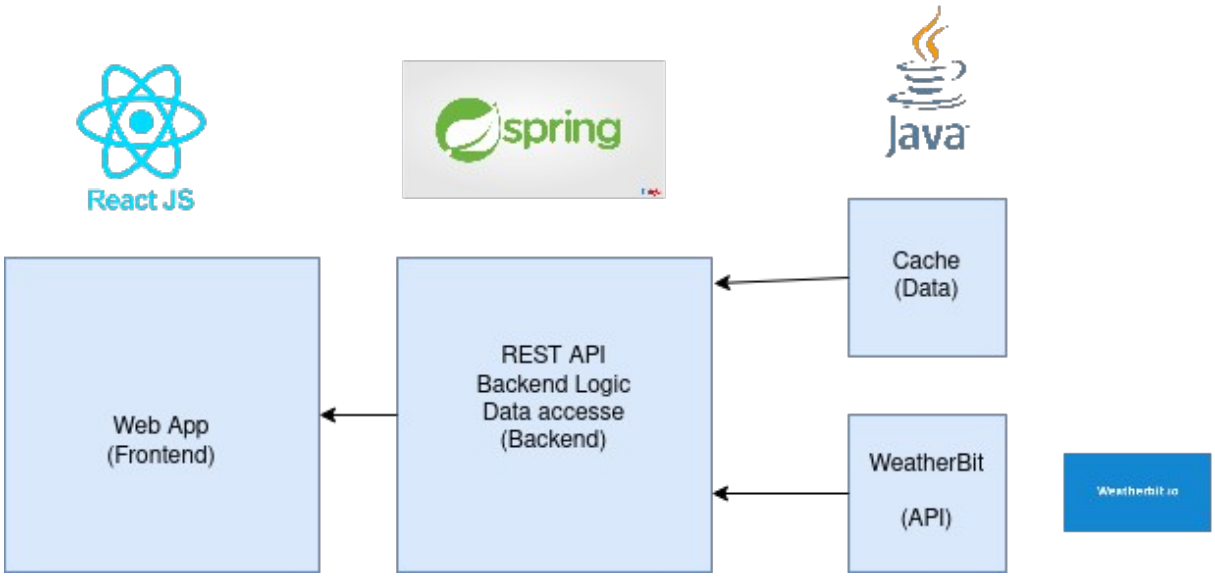


Forecast Page:



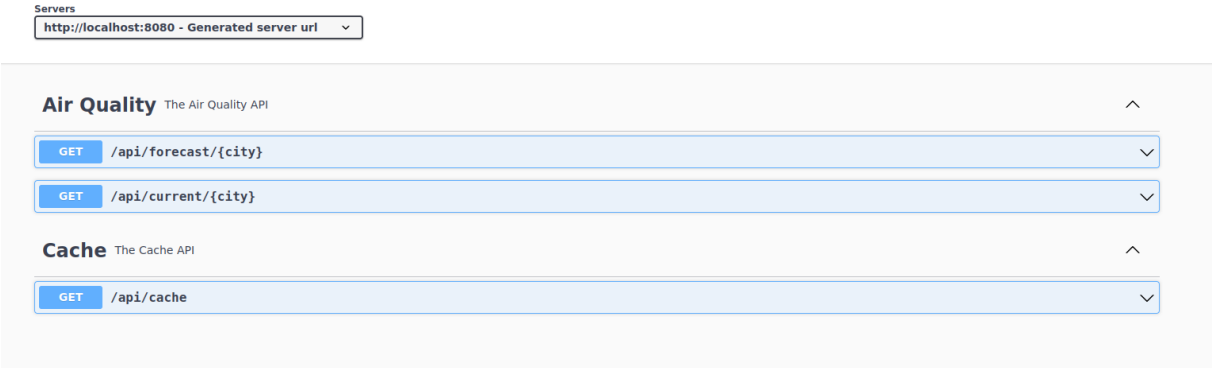
Cache Page:

2.2 System architecture



2.3 API for developers

For the documentation of the API, Swagger was used:
http://localhost:8080/swagger-ui/index.html#/



3 Quality assurance

3.1 Overall strategy for testing

It was used a TDD strategy.

TDD or Test Driven Development, as the name suggests, the test process drives software development. Moreover, it's a structuring practice that enables developers and testers to obtain optimized code that proves resilient in the long term.

First I developed the application and did all the backend tests, then I developed the frontend and connected the backend with the frontend.

Regarding the tests, I started with the Unit tests for the cache, then on the tests for the service and finally on the tests for the controllers.

3.2 Unit and integration testing

For the tests I used the following tools:

- Junit;
- Mockito;
- Mock mvc

Unit tests:

- On the folder UnitTest → CacheTest.java:

```
You, 4 days ago | 1 author (You)
public class CacheTest {

    private Cache cache;
    private City city;


    @Test
    @DisplayName("When cache is 0, then a cache request, NumMisses should be 1")
    public void testEmptyCache() {
        assertThat(cache.getCachedRequest(key: "?")).isNull();
        assertThat(cache.getNumRequests()).isEqualTo(expected: 1);
        assertThat(cache.getMissCount()).isEqualTo(expected: 1);
        assertThat(cache.getHitCount()).isEqualTo(expected: 0);

    }      You, 4 days ago • Unit Test done …

    @Test
    @DisplayName("When the cache request is expired, then the numMisses should be 1")
    public void isExpiredTest() throws InterruptedException{
        ObjectCache objectCache = new ObjectCache(city, System.currentTimeMillis() + 1000L);
        assertThat(objectCache.isExpired()).isFalse();
        Thread.sleep(1001L);
        assertThat(objectCache.isExpired()).isTrue();

    }


}
```

Service Level Tests:

- On the ServiceTest → AirQualityServiceTest.java

```java
You, yesterday | 1 author (You)
@ExtendWith(MockitoExtension.class)
public class AirQualityServiceTest {

    @Mock
    private RestTemplate restTemplate;

    @InjectMocks
    private AirQualityService airQualityService;

    final String key = "&key=a06833c3d40f4f8d93d4b2c47c39fa8f";
    final String url = "https://api.weatherbit.io/v2.0/current/airquality?city=";
    final String url2 = "https://api.weatherbit.io/v2.0/forecast/airquality?city=";
    final String city = "London";
    final String data = "{\"city_name\":\"London\",\"lat\":51.51279,\"lon\":-0.09184,\"data\":[{\"aqi\":19,\"o3\":40.41195,\"so2\":0.14156103,\"no2\":0.0224245,\"co\":2
    final String data2 = "{\"city_name\":\"London\",\"lat\":51.51279,\"lon\":-0.09184,\"data\":[{\"aqi\":60,\"o3\":46.2,\"so2\":0.1,\"no2\":14.2,\"co\":213.6,\"pm10\":1
    @Test
    @DisplayName("When make a request, then return the response")
    public void getAirQualityTest() {
        Mockito.when(restTemplate.getForEntity(url + city + key, responseType: String.class)).thenReturn(ResponseEntity.ok(data));
        ResponseEntity<City> response = airQualityService.getAirQuality(city);
        assertEquals(response.getBody(), airQualityService.processResponse(ResponseEntity.ok(data)).getBody());
        Mockito.verify(restTemplate).getForEntity(url + city + key, responseType: String.class);
    }

    @Test
    @DisplayName("When make a forecast request, then return the response")
    public void getAirQualityForecastTest() {
        Mockito.when(restTemplate.getForEntity(url2 + city + key, responseType: String.class)).thenReturn(ResponseEntity.ok(data2));
        ResponseEntity<City> response = airQualityService.getAirQualityForecast(city);
        assertEquals(response.getBody(), airQualityService.processResponse(ResponseEntity.ok(data2)).getBody());
        Mockito.verify(restTemplate).getForEntity(url2 + city + key, responseType: String.class);
    }
}
```

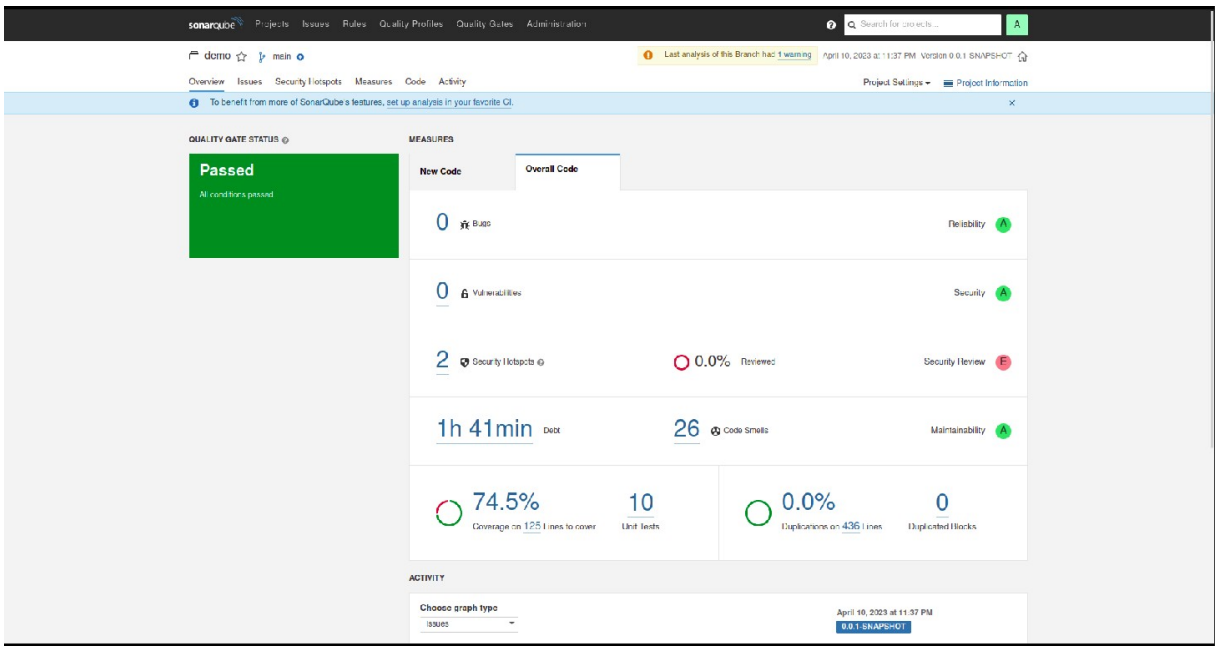
Controller Test:

- On the folder ControllerTest:
  - CacheControllerTest.java

```java
You, 12 hours ago | 1 author (You)
@WebMvcTest(CacheController.class)
public class CacheControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private AirQualityService airQualityService;

    @BeforeEach
    public void setUp() {
    }

    @Test
    public void getCacheTest() throws Exception{
        Mockito.when(airQualityService.getCache()).thenReturn(new ResponseEntity<>(Cache.printCache(), HttpStatus.OK));
        mvc.perform(get(urlTemplate: "/api/cache").contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath(expression: "numRequests").value(Cache.getNumRequests()))
            .andExpect(jsonPath(expression: "missCount").value(Cache.getMissCount()))
            .andExpect(jsonPath(expression: "hitCount").value(Cache.getHitCount()));
    }
}
```

  - AirQualityContollerTest.java

3.3 Code quality analysis



It was used SonarQube, for the static code quality analysis. The main aspects that we are able to see in the overall code section are:

- Reliability: A
  - Bugs: 0
- Security: A
  - Vulnerabilities: 0
- Security Review: E
  - Security Hot-spots: 2
    - ‘'API_KEY' detected in this expression, review this potentially hard-coded secret.’ → Review Priority: High
    - “Make sure this debug feature is deactivated before delivering the code in production.” → Review Priority: Low
- Maintainability: A
  - Debt: 1h 41min
  - Code Smells: 26
- Coverage: 74,5%
- Unit Tests: 10

4 References & resources

Project resources

Resource:	**URL/location:**
Git repository	https://github.com/CatarinaCosta02/TQS_103696/tree/main/HW1
Video demo	https://github.com/CatarinaCosta02/TQS_103696/blob/main/HW1/2023-04-10%2017-57-45.mp4
QA dashboard (online)	http://localhost:9000/dashboard?id=AirQuality&selectedTutorial=local
CI/CD pipeline	[**optional**; if you have th CI pipeline definition in a server, place the URL here]
Deployment ready to use	[**optional**; if you have the solution deployed and running in a server, place the URL here]

Reference materials

WeatherBit API: https://www.weatherbit.io/