HW1: Mid-term assignment report

Testing and Software Quality

Rafael Remígio 102435

April 10, 2023

Departamento de Electrónica, Telecomunicações e Informática
Universidade de Aveiro
Year 2022/2023

Contents

1 Introduction

1.1 Overview of the work

In this assignment the web application developed provides real-time information parameters related to the air quality of a certain location.

1.2 Current limitations

I was not able to implement a Continuous Integrationg Pipeline with a tool such as GitHub Action.

2 Product specification

2.1 Functional scope and supported interactions

2.1.1 Actors

Possible Actors of the proposed product:

- Individuals - People who want to check the air quality in their immediate surroundings, such as their home, office, or neighborhood.
- Public Health Officials - Officials who want to monitor air quality in a particular region or city to make informed decisions on public health policies.

2.1.2 Use Cases

The WebApp provides two use cases:

1. Real-time Air Quality Monitoring. Provides real-time air quality levels in a particular location. This information consists of the specific parameters:
    - Particulate Matter PM2.5
    - Particulate Matter PM10
    - Carbon Monoxide CO
    - Nitrogen Dioxide NO2
2. access and view statistics of the cache's usage:
    - Misses
    - Hits
    - Acesses

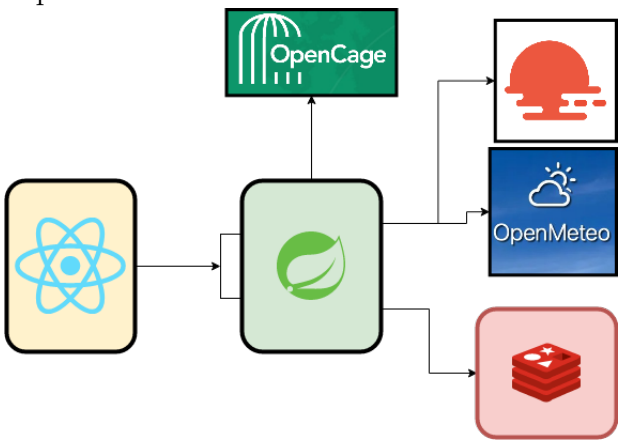
2.2 System architecture

2.2.1 Architecture

The application's Architecture is very basic and allows for easy maintenance and API evolution. The Controller handles communication between the WebApp and the backend. The Service Layer is composed of two distinct services. The Weather Service (responsible for providing the Air Quality for a certain region) utilizes a Geocoding API in order to transform an address or location string into coordinates. It communicates with two different Air Quality API's. In order to provide more constant availability and resilience, one acts as a backup when the main API is down. These API's provide Air Quality data for a certain location based on latitude and longitude. In order to reduce query times, a caching system using a key-value database was also introduced. The objects cached have a specific Time to Live that is easily configurable.

2.2.2 Technologies and API's Used

The very basic User Interface was constructed using ReactJS. It interacts with the Backedn througth a RestAPI. The backend was built using Spring Boot. For an in memory database I used Redis. The Geolocation Api chosen was Open Cage. The primary Air Quality API is the OpenWeather API. The backup Air Quality API is the OpenMeteo API.



2.3 API

The API has two endpoints:

1. **GET** /weather?local={*local*}
   Query Params:

   - **Required** - local: Address, Country or City .

   Response Body:

```
{
 "coords":{
          "lat":40.7275536,
          "lgn":-8.5209649
          },
 "stats":{
          "co":150.0,
          "no2":6.5,
          "pm25":9.4,
          "pm10":11.5
          },
 "location":"aveiro"
}
```

2. **GET** /cacheInfo
   Response Body:

```
{
"hits":5,
"misses":10,
"acesses":15
}
```

3 Quality assurance

3.1 Logging

Logging is a crucial aspect of software development that plays a vital role in supporting production and debugging activities.
Logging was conducted using the slf4j Logger class has it provides an easy integration with the Spring Boot Application.

Each log message contains a timestamp, method invocation details, custom log messages, and other contextual data. Additionally, each log entry is assigned a logging level identifier, providing a way to categorize and prioritize log entries based on their significance.
I followed logging principles and best practices - such as: logging at the correct evel, providing meaningfull messages, etc - in order to provide meaningfull logs.

3.2 Overall strategy for testing

The overall strategy used was for testing was Test-driven development (TDD). The idea behind TDD is to ensure that the code meets the requirements and is free from defects by writing tests that define the expected behavior of the code.

3.3 Unit and integration testing

Unit Tests were used to test the individual units or components of the software application or system to ensure that they are functioning correctly.

- Controller Tests: The goal of these tests is to ensure that the controller behaves correctly and produces the expected HTTP response. This involves creating mock objects for the multiple services that the controllers interact with.

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


    @MockBean
    private CacheService cacheService;

    @MockBean
    private RestTemplate restTemplate;

    @MockBean
    private WeatherService weatherService;

    @MockBean
    private Logger logger;

    @Autowired
    private MockMvc mockMvc;
```

```java
@Test
public void testGetWeather_withValidLocalString() throws Exception {
    Location expectedLocation = new Location();
    expectedLocation.setLocation(address:"gaia");
    expectedLocation.setCoords(new Coords(lat:50, lgn:50));
    expectedLocation.setStats(new Stats(co:13.3, no2:20.0, pm25:20.0, pm10:20.0));

    when(weatherService.getWeather(local:"gaia")).thenReturn(expectedLocation);

    mockMvc.perform(get(urlTemplate:"/weather?local=gaia"))
            .andExpect(status().isOk())
            .andExpect(jsonPath(expression:"$.location").value(expectedValue:"gaia"))
            .andExpect(jsonPath(expression:"$.coords.lat").value(expectedValue:50))
            .andExpect(jsonPath(expression:"$.coords.lgn").value(expectedValue:50))
            .andExpect(jsonPath(expression:"$.stats.co").value(expectedValue:13.3));
}

@Test
    public void testGetWeather_withInvalidLocalString() throws Exception {

        when(weatherService.getWeather(anyString())).thenReturn(value:null);

        mockMvc.perform(get(urlTemplate:"/weather?local=invalid-location"))
                .andExpect(status().isNotFound());
    }
```

- Service Tests: The goal of these tests is to ensure that the Business Logic is implemented as expected. To achive this it is necessary to mock multiple components.

    1. Mock the API calls
    2. Mock the Redis Database
    3. In the Weather Service tests it is also necessary to mock the Cache Service.

```java
@ExtendWith(MockitoExtension.class)
public class WeatherServiceTests {

    @Mock
    private CacheService cacheService;

    @Mock
    private RestTemplate restTemplate;


    @InjectMocks
    private WeatherService weatherService;
```

```java
@ExtendWith(MockitoExtension.class)
public class CacheServiceTests {


    @Mock
    CacheRepository cacheRepository;

    @InjectMocks
    CacheService cacheService;
```

Example of Mocks when *"Location is not in cache but second API responds"*

```
Mockito.when(cacheService.getLocation(local)).thenReturn(value:null);

Mockito.when(restTemplate.getForObject(geocodingUrl, responseType:GeocodingDTO.class))
        .thenReturn(geocodingDTO);


Mockito.when(restTemplate.getForObject(openMetoUrl, responseType:OpenMetoStats.class))
.thenReturn(openMetoStats);


Mockito.when(restTemplate.getForObject(Mockito.anyString(), Mockito.eq(value:AirQualityDTO.class)))
        .thenThrow(new RestClientException(msg:"API not responding"));
```

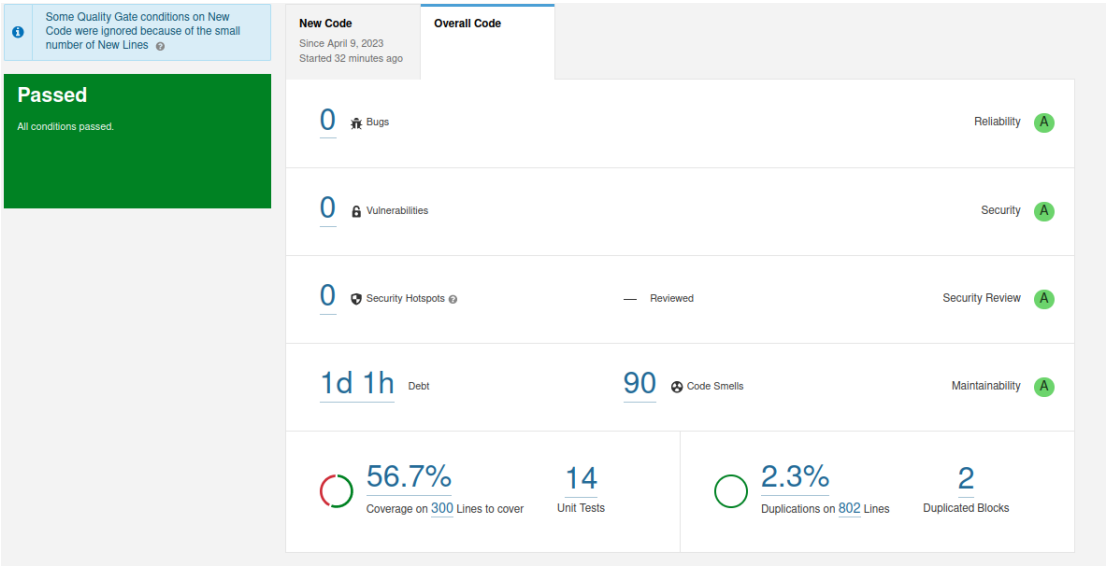
- Repository Tests: The goal of these tests is to ensure that the Database behaves as expected.

```
@Test
@DisplayName("Test findById and test TTL")
void whenfindByID_thenReturnLocation() {
    // arrange a new location and insert into db
    Location paris = new Location(address:"Paris", coords:null, stats:null);
    cacheRepository.save(paris); //ensure data is persisted at this point

    // test the query method of interest
    Optional<Location> found = cacheRepository.findById(paris.getLocation());
    assertThat( found.get() ).isEqualTo(paris);

    try {
        Thread.sleep(7000);
    } catch (Exception e) {
    }

    found = cacheRepository.findById(paris.getLocation());
    assertFalse((found.isPresent()));

}
```

- User Inteface Testing - To automate tests of the Web Application I used the Selenium WebDriver in conjunction with the Selenium IDE.

3.4 Code quality analysis

For static sode analysis I used throughout the entirety of the project the SonarQube platform. It helped me identify potential bugs and vulnerabilities, improve code my code quality, and find code smells and possible anti-patterns. In conjunction with with static code anylisis I also used the SonarQube platform in order to evaluate the code coverage of tests implemented.



4 References & resources

All the code can be found and in my personal TQS repository.
Videos of the working product can also be found on my personal TQS repository.