HW1: Mid-term assignment report

Paulo Pinto *[103234]*, v2023-04-11

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.
This report presents and brief explanation on the work done for the Homework given in TQS.

The objective of this project was to create an **REST-API service** with frontend and implement various types of tests on them. The tests requested were:

- Unit Tests
- Integration Tests
- Service-Level Tests
- Functional Testing (on web interface)

The air quality data used is received from three different APIs, the main one is the WeatherAPI, if this one fails the API-Ninjas and Big Data Cloud APIs are used to get the data.

1.2. Current limitations

One of the biggest limitations of this project is the range of data that is given, it was supposed to display air quality data of the moment and forecast but this app only allows the user to see the more recent (or the cache version) of the air quality.
Another problems comes from the same city can have different names in different APIs, for example in the first API the city Porto only appears if written “Oporto”, but in the second API it is called “Porto”, which can cause data duplication and unnecessary misses on the cache.
Due to some problems between *pom* dependencies the Selenium and Cucumber with Selenium test needed to be done in a maven project of their own.
If the city does not exist there is no good response or error response defined, the frontend will just not display anything new, like if the the search button was never used.

2. Product specification

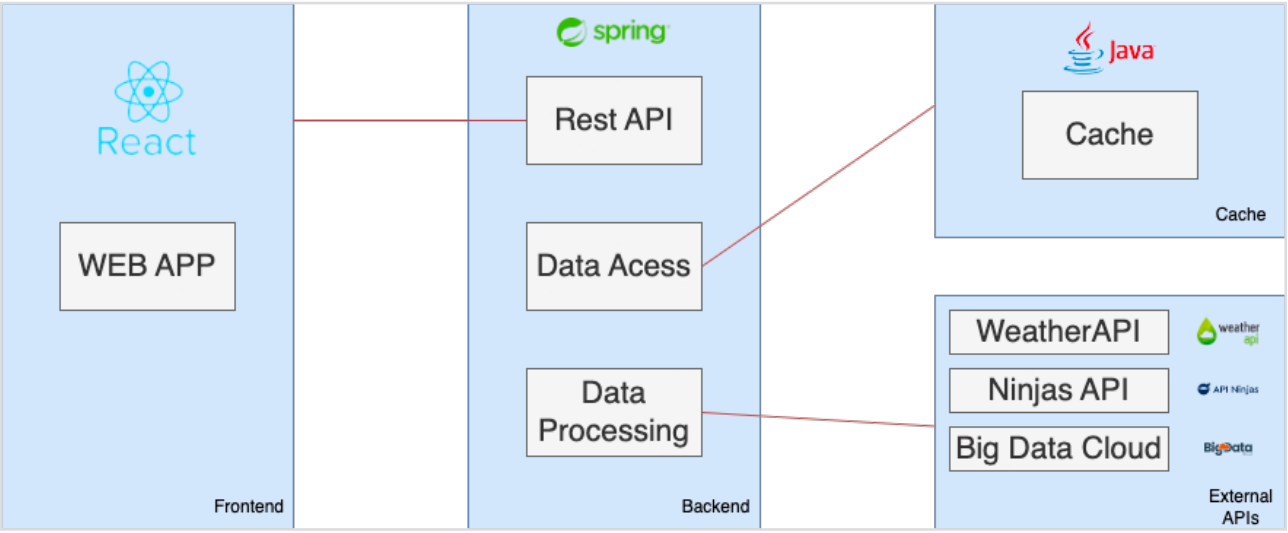
2.1. Functional scope and supported interactions

This application allows the user to get the air quality information of a city, more precisely, elements concentration. The user has the option to search by city name or using coordinates. the user can see in real time how the cache interacted with request, if the hits count was elevated it means the city information came from the cache, if it was the miss that was elevated, it means the city information came from the API. An brief explanation of an search with city name is:

- Enter the site
- Click the name input (The predefined search method is by name)
- Input the city name
- Click the search button
- See the table show the city information

2.2. System architecture

The project has three main parts, The frontend developed in react, the backend developed in spring boot and the tests, that use many different frameworks, like Junit, Mockito and Selenium.



2.3. API for developers

The API has three endpoints, all for *get* requests:

- ***/name*** , this endpoint receives one parameter called name, and calls the service who replies with a City object.
- ***/coords*** , this endpoint receives two parameters called lat and lon, and calls the service who replies with a City object.
- ***/cacheDetails*** , this endpoint calls the service who replies with a Map where the key is a String (Requests, Hits, Misses) and the value is how many time each happened.

3. Quality assurance

3.1. Overall strategy for testing

The strategy used for the backend tests consisted on thinking how we wanted the API to work, writing the tests names on paper that were taught as needed to test what was going to be built. Then the API was created and it's components implemented. After that the tests were written and the API was adapted in order for the tests to pass, in other words, fist the API was implemented, tests were written with some consideration how it worked but testing if what it does it giving the results we wanted, if the tests failed the API was corrected in order for them to pass.

3.2. Unit and integration testing

Unit tests were used for basically the whole backend, and the implementation was based on testing each functionality in its own test.

For the cache tests some of them had the time mocked, making it lock like sixty one seconds have passed, to test how the cache reacted.

In the controller tests integration tests were used to test the output of each endpoint.

```java
@WebMvcTest(CityController.class)
class ControllerTest {
    3 usages
    @Autowired
    private MockMvc mvc;

    3 usages
    @MockBean
    private CityService cityService;

    @Test
    void getCityByNameTest() throws Exception {
        City porto = new City( name: "Oporto", country: "Portugal", lat: 41.15,  lon: -8.62,  co: 360.5,  no2: 14.89,  o3: 46.5,  so2: 1.89,  pm2_5: 21.29

        when(cityService.getCityByName("Oporto")).thenReturn(porto);

        mvc.perform(get( urlTemplate: "/name")
                        .param( name: "name",  ...values: "Oporto"))
                        .andExpect(status().isOk());
    }

    @Test
    void getCityByLatAndLonTest() throws Exception {
        City porto = new City( name: "Oporto", country: "Portugal", lat: 41.15,  lon: -8.62,  co: 360.5,  no2: 14.89,  o3: 46.5,  so2: 1.89,  pm2_5: 21.29

        when(cityService.getCityByLatAndLon( lat: 41.15,  lon: -8.62)).thenReturn(porto);

        mvc.perform(get( urlTemplate: "/coords")
                        .param( name: "lat",  ...values: "41.15")
                        .param( name: "lon",  ...values: "-8.62"))
                .andExpect(status().isOk());

    }
```

3.3. Functional testing

The functional tests were written in Selenium and in Cucumber. Two test used only selenium and then we used exactly the same tests but merged with the Cucumber approach.

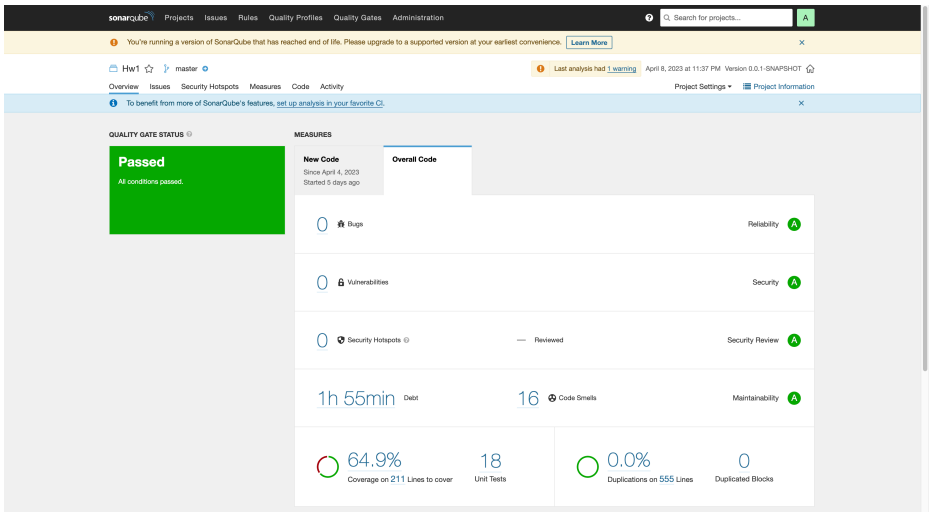
```
Feature: City Air quailty search
  To allow a client to search for a city air quality.

  Scenario: Search by Name
    Given a website 'http://127.0.0.1:3000/'
    Then inital cache values are checked
    Then a city with name 'Aveiro' is searched
    Then the recieved values for the name and the cache values are checked
    Then the city is searched again
    Then the name cache values are checked again
    Then the website is closed


  Scenario: Search by Latitude and Longitude
    Given a website 'http://127.0.0.1:3000/'
    Then inital cache values are checked
    Then a city latitute '41.15' and longitude '-8.62' is searched
    Then the recieved values for the coords and the cache values are checked
    Then the city is searched again
    Then the coords cache values are checked again
    Then the website is closed
```

3.4. Code quality analysis

The code analysis tool used was SonarQube with Jacoco integrated, the code coverage was only 64,9 % due to the lack of tests for the HTTP Client which we were not able to test without errors, if we exclude this file the rest had an 88+ % of coverage.

There are 16 code smells, but 15 are do the the package/methods names convections, which we deemed as not a problem and one that warned us about the city constructor having more

that 7 arguments (10), we also did not change that due to the way we wanted to implemented the Model.



Coverage **64.9%**

New Code: Since April 4, 2023

	Coverage	Uncovered Lines	Uncovered Conditions
src/main/java/tqs/Hw1/Services/HTTPClient.java	4.1%	51	20
src/main/java/tqs/Hw1/Hw1Application.java	33.3%	2	–
src/main/java/tqs/Hw1/Services/CityServiceImpl.java	88.6%	7	3
src/main/java/tqs/Hw1/Models/City.java	91.2%	3	–
src/main/java/tqs/Hw1/Controllers/CityController.java	100%	0	–
src/main/java/tqs/Hw1/Cache/TTLCache.java	100%	0	0

6 of 6 shown

3.5. Continuous integration pipeline [optional]

The Integration pipeline was created with git actions and has two jobs, one is test if the package was build correctly, passing all the backend tests. The second was also the package test but with communication with the SonarQube, sending it the data automatically.

Since SonarQube is running on localhost we used ngrok to create a safe connection between my port and the internet.

```yaml
name: CI Script

# Controls when the action will run.
on:
  # Triggers the workflow on push or pull request events but only for the main branch
  push:
    branches: [ main ]

  # Allows you to run this workflow manually from the Actions tab
  workflow_dispatch:

# A workflow run is made up of one or more jobs that can run sequentially or in parallel
jobs:
  # This workflow contains a single job called "build"
  build:
    name: Build Maven Project
    # The type of runner that the job will run on
    runs-on: ubuntu-latest

    # Steps represent a sequence of tasks that will be executed as part of the job
    steps:
      # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
      - name: Step 1 - Checkout main branch from Github
        uses: actions/checkout@v2

      - name: Step 2 - Set up JDK 17
        uses: actions/setup-java@v1
        with:
          java-version: 17

      - name: Step 3 - Have Github Actions build Maven project
        run: |
          ls -la
          mvn -B package --file HW1/Backend/pom.xml

      - name: Step 4 - List the current directory
        run: ls -la

      - name: Step 5 - Show files inside the target/ folder
        run: |
          cd HW1/Backend/target
          ls -la
  sonar:
      name: SonarCloud Code Inspection
      runs-on: ubuntu-latest
      steps:
        - uses: actions/checkout@v2
          with:
            fetch-depth: 0  # Shallow clones should be disabled for a better relevancy of analysis
        - name: Set up JDK 17
          uses: actions/setup-java@v1
          with:
            java-version: 17
        - name: Cache SonarCloud packages
          uses: actions/cache@v1
          with:
            path: ~/.sonar/cache
            key: ${{ runner.os }}-sonar
            restore-keys: ${{ runner.os }}-sonar
        - name: Cache Maven packages
          uses: actions/cache@v1
          with:
            path: ~/.m2
            key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
            restore-keys: ${{ runner.os }}-m2
        - name: Build and analyze
          env:
            GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}  # Needed to get PR information, if any
            SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
            SONAR_HOST_URL: ${{secrets.SONAR_HOST_URL}}
          run: |
              cd HW1/Backend
              mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar
```

4. References & resources

Project resources

Resource:	URL/location:
Git repository	https://github.com/Pjnp5/TQS_103234/tree/main/HW1
Video demo	https://github.com/Pjnp5/TQS_103234/blob/main/HW1/VideoHW1.mp4
QA dashboard (online)	[**optional**; if you have a quality dashboard available online (e.g.: sonarcloud), place the URL here]
CI/CD pipeline	https://github.com/Pjnp5/TQS_103234/blob/main/.github/workflows/build.yml
Deployment ready to use	[**optional**; if you have the solution deployed and running in a server, place the URL here]

Reference materials

- ChatGPT
- SonarQube with Github
- WeatherAPI
- API-Ninjas