HW1: Mid-term assignment report

Tomás dos Santos Batista [89296], 2020-04-15

1 Introduction

1.1 Overview of the work

AirQuality WebApp is a website to consult the quality of the air in Lisbon and Madrid. It contains multiple parameters (air quality information, PM2.5, temperature, pressure, and many others). It uses a public API to obtain the values.

1.2 Limitations

The only limitations I fought with were some initial Spring-Boot problems. Other than that, everything went fine.

2 Product specification

2.1 Functional scope and supported interactions

The users can consult the information through the webapp or through the API of the project. The webapp have a clean view of each's city information.

AirQuality WebApp Lisbon Madrid

Lisbon

Station: Entrecampos, Lisboa, Portugal

- **AQI:** 41.0
- **PM2.5:** 15.0
- **PM10:** 6.0
- **O3:** 40.7
- **No2:** 5.5
- **SO2:** 0.6
- **Temperature:** 16.0
- **Pressure:** 1009.0
- **Humidity:** 72.0
- **Wind:** 8.2
- ***Measure timestamp:*** 2020-04-15 14:00:00

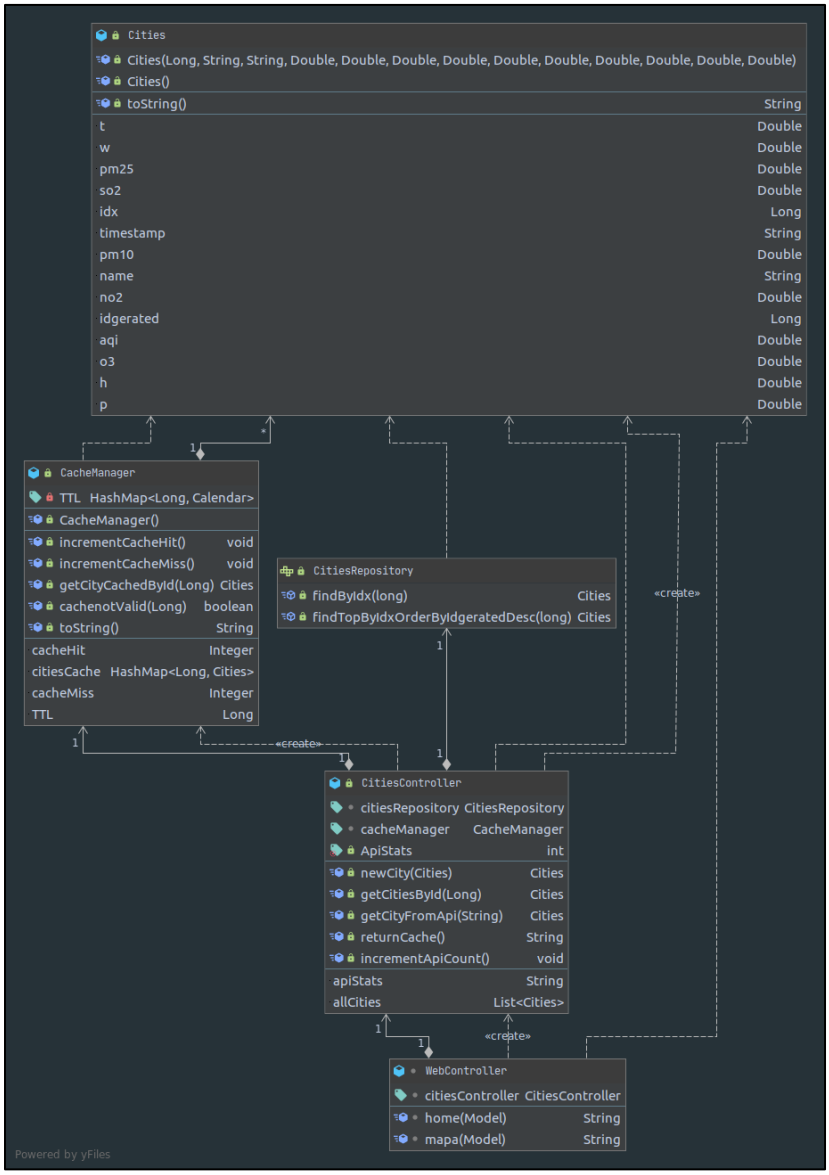
AirQuality WebApp Lisbon Madrid

Madrid

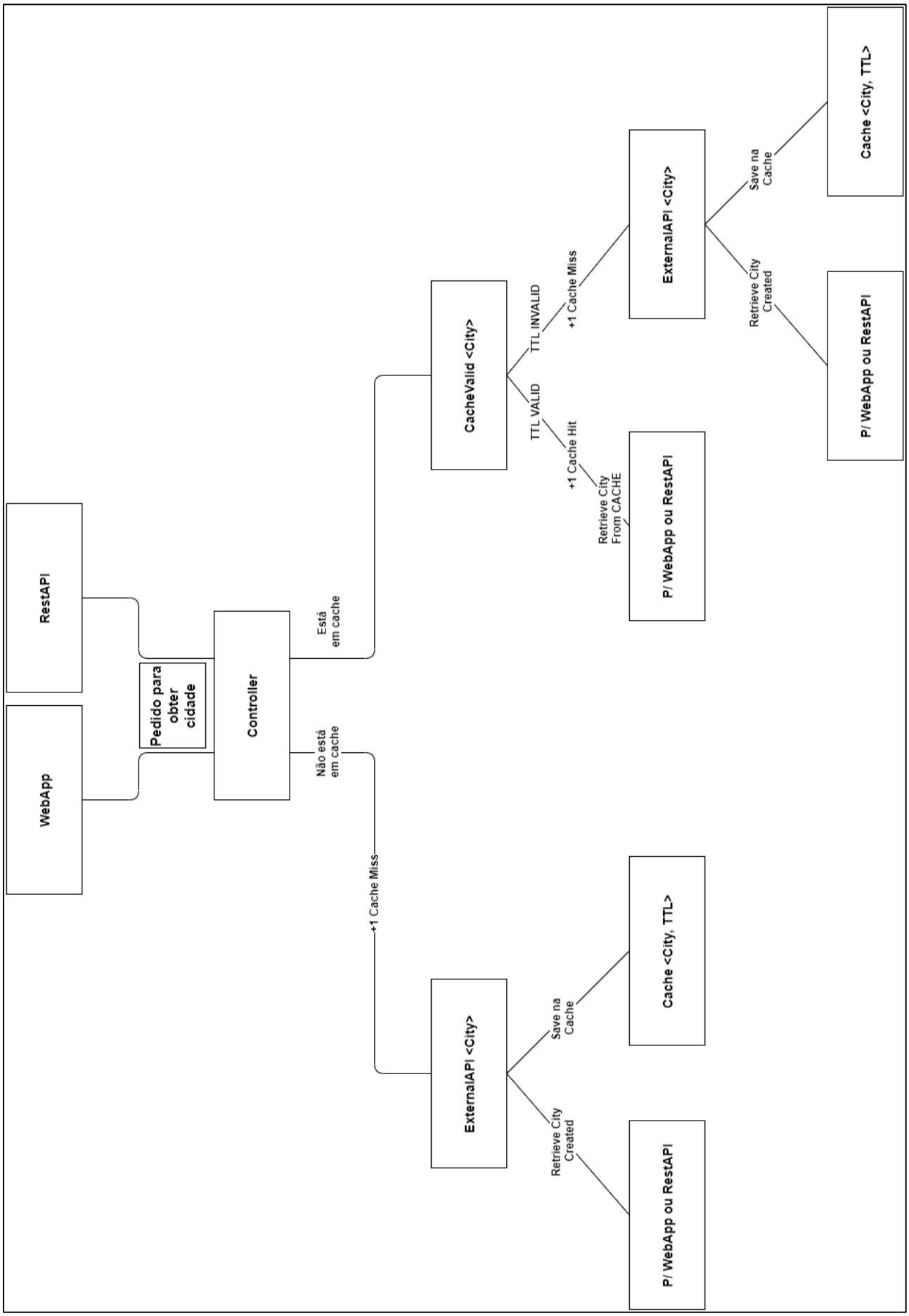
Station: Madrid

- **AQI:** 30.0
- **PM2.5:** 30.0
- **PM10:** 8.0
- **O3:** 32.1
- **No2:** 3.7
- **SO2:** 3.6
- **Temperature:** 13.8
- **Pressure:** 1011.6
- **Humidity:** 76.0
- **Wind:** 12.5
- ***Measure timestamp:*** 2020-04-15 17:00:00

2.2 System architecture



The project was built using Spring-Boot and ThymeLeaf. I implemented a PostgreSQL database running on a docker container to save the data persistently.
The communication between database ⇔ Spring-Boot is made using a JPARepository. The communication between Frontend/ThymeLeaf ⇔ Spring-Boot is made through a Controller and Models. I also implemented a Cache to save the last searched values.
The way of working of my system can is the following:


WebApp
RestAPI
Pedido para obter cidade
Controller
Está em cache
Não está em cache
CacheValid <City>
TTL VALID
TTL INVALID
+1 Cache Hit
+1 Cache Miss
Retrieve City From CACHE
P/ WebApp ou RestAPI
ExternalAPI <City>
Save na Cache
Retrieve City Created
Cache <City, TTL>
P/ WebApp ou RestAPI
+1 Cache Miss
ExternalAPI <City>
Save na Cache
Retrieve City Created
Cache <City, TTL>
P/ WebApp ou RestAPI

2.3 API for developers

I made a documentation to support my API. It can be found at: [Postman Documentation - AirQuality WebApp](#).

3 Quality assurance

3.1 Overall strategy for testing

I implemented unit testing, Mockito and Selenium web testing. Got an overall of 100% classes and 92% lines covered (due to main not be tested and WebController tests were in a different package).

Coverage: com.tomas.ua.airquality in air-quality ×

100% classes, 92% lines covered in package 'com.tomas.ua.airquality'

Element	Class, %	Method, %	Line, %
AirQualityApplication	100% (1/1)	0% (0/1)	33% (1/3)
controller	100% (2/2)	80% (8/10)	88% (64/72)
repository	100% (0/0)	100% (0/0)	100% (0/0)
config	100% (1/1)	100% (3/3)	100% (15/15)
cache	100% (1/1)	100% (11/11)	93% (29/31)
models	100% (1/1)	100% (17/17)	100% (31/31)

- com.tomas.ua.airquality 100% classes, 92% lines covered
  - cache 100% classes, 93% lines covered
    - CacheManager 100% methods, 93% lines covered
  - config 100% classes, 100% lines covered
    - WebConfig 100% methods, 100% lines covered
  - controller 100% classes, 88% lines covered
    - CitiesController 100% methods, 100% lines covered
    - WebController 0% methods, 20% lines covered
  - models 100% classes, 100% lines covered
    - Cities 100% methods, 100% lines covered
  - repository
  - AirQualityApplication 0% methods, 33% lines covered

3.2 Unit and integration testing

I tested all different components.

CacheManager:

1. Number of hits and misses
2. TTL times were valid
3. Set and Get Cache from certain city

CitiesController:

1. API Stats (number of calls)
2. Cache Stats (number of misses and hits)
3. Call the extern API
4. Get city by certain ID
5. Get All Cities on DB
6. Add new City

CityModel:

1. City added and retrieved
2. Test all methods (Comparing field by field)
3. Save through repository

3.3 Functional testing

Implemented Selenium IDE tests.

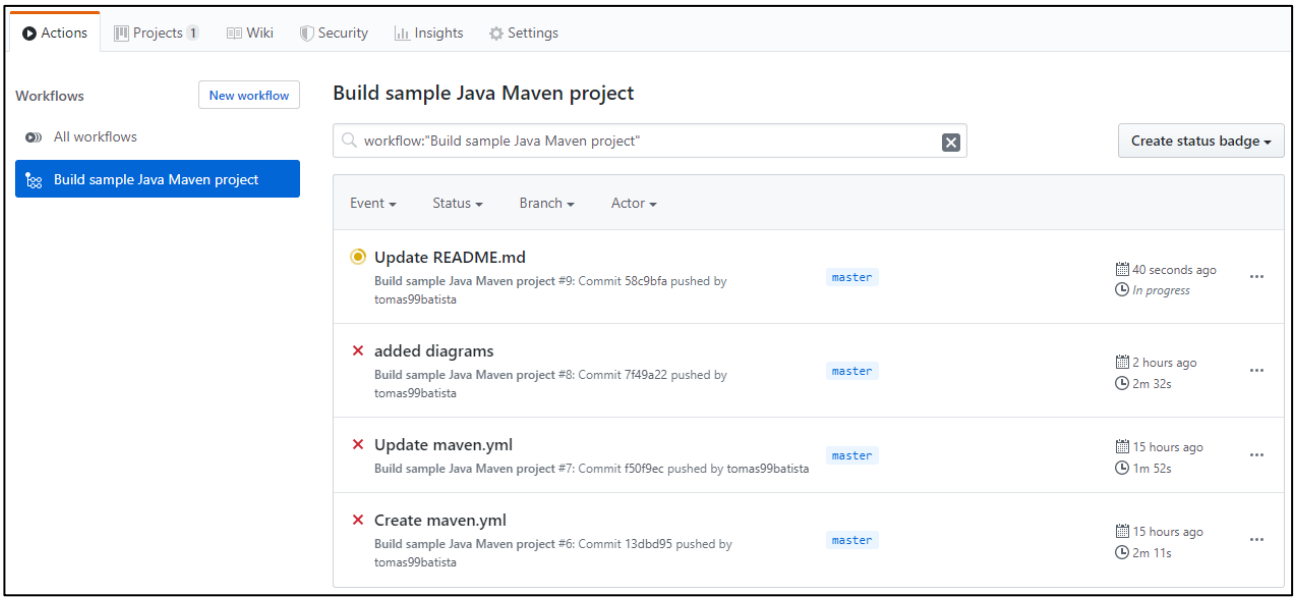
1. Check the info loaded
2. Assert all info loaded
3. Assert that the current page is the specific

3.4 Static code analysis

I implemented Codacy to analyze the code. I picked this software because I have been working with it. It showed me, mostly, unused imports.

3.5 Continuous integration pipeline [optional]

I implemented CI with GitHub actions. Although I used CI, it fails due to not having access to the database (I implemented PostGresSQL running on a docker container).



4 References & resources

Project resources

- Git repository
- Video demo

Reference materials

- WAQI API
- Codacy
- GitHub
- Spring.io Tutorials
- CI/CD Tutorial

How to run the project

- **Configure postgres db on docker:** *docker run --name postgres -d -p 5432:5432 -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=password -d postgres*
- **Run:** *mvn spring-boot:run*
- **WebApp:** localhosti:8080/ || localhost:8080/madrid
- **API Calls:** See endpoints here