HW1: Mid-term assignment report

Maria Inês Seabra Rocha [93320], v2021-05-15

1	**Introduction**	**1**
1.1	Overview of the work	1
1.2	Current limitations	1
2	**Product specification**	**2**
2.1	Functional scope and supported interactions	2
2.2	System architecture	2
2.3	API for developers	4
3	**Quality assurance**	**4**
3.1	Overall strategy for testing	4
3.2	Unit and integration testing	5
3.3	Functional testing	5
3.4	Static code analysis	5
4	**References & resources**	6

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.

Healthy Breeze is a multilayered spring boot application that offers an interface where the user can consult his current location information about air quality or pollen concentration and can search for a specific location or city name. Furthermore, the project includes an API that not only shows the same results shown in interface, but also cache statistics.

1.2 Current limitations

Unfortunately there are a lot of things that were not implemented as it was supposed to.

By now, it is not possible to see any history or forecast air or pollen information.

The calls that occur to external APIs are not very efficient which results in increased time response waiting to user requests, something that should be improved later.

Another identified problem is related to the fact that the main API (ambee api) free account only accepts 100 requests per day, which is very limited.

Regarding the tests, some were left to be implemented.

2 Product specification

2.1 Functional scope and supported interactions

The main objective of the application as it was pointed before, is to show air quality and pollen information for a place. As the user opens the webpage it is shown is current air quality information based on is IP location. However sometimes, the external API may not find user current location information and an error message is shown instead.

When a user scrolls down a webpage he finds a section where he can search by city name or city coordinates and iif every input is correct, then in the table of results the information will appear.

In case that some input field is not correctly filled, a modal with message error is shown to give user feedback of his errors and help him to interact with the application in the correct way.

When the user searches for a city then the application service component will query Ambee API about that city and return its information when the query is successful. When the first API does not respond as it was supposed to then the second API (wAQI) is called, following the same process.

A message modal is shown to the user when both API requests fail.

It is also possible to interact with the internal API. This one returns data on JSON format and allows extra queries about cache statistics.

2.2 System architecture

As we can see in the following diagram, the client can interact directly with the API and to each request, the PlaceRestController class annotated with @RestController will request data to the service class and then return the response to the user. The @Service annotated class will consult cache to see if that request was already made and if the object was not created yet or expired then it will call the external APIs.

In the previous section it was already explained the order of external API requests, if none of them returns the supposed information then nothing is returned to the restcontroller and a message about the problem is returned. Along with the normal response entity message with specific status code in case of error, it was also implemented log support, Log4j, that writes to console and to file spring.log at the same time. Some logs are also written whenever some error in service occurs during the parse of APIs response to Place object or in APIConnection class when the main API reaches the daily requests bound.

On the interface side, the request is made to PagesController and the flow starts when the page is loaded and thymeleaf is used to load the user's location data requested from PageController to service. After that Ajax is used to call the internal API whenever a search is made and the flow proceeds as it was explained earlier.

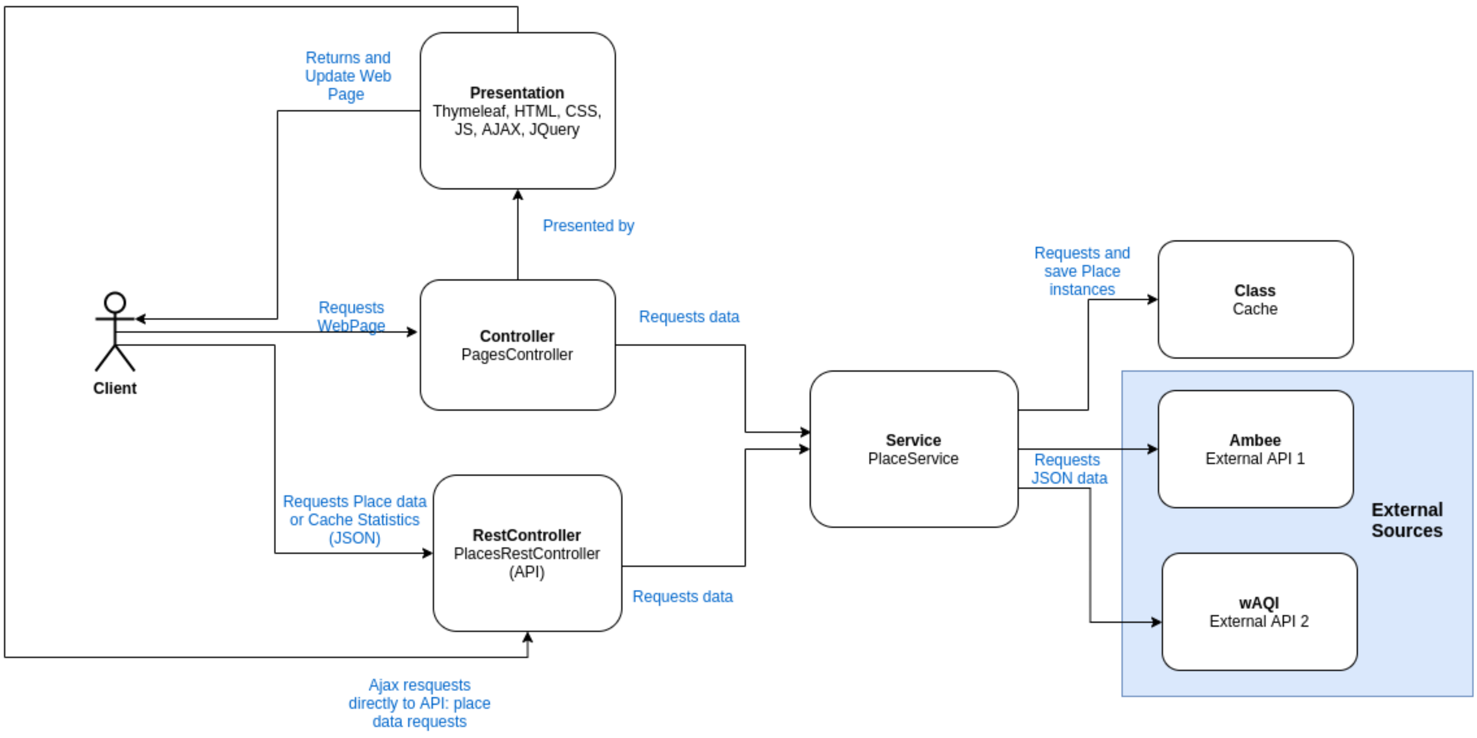

Returns and Update Web Page
Presentation
Thymeleaf, HTML, CSS, JS, AJAX, JQuery
Presented by
Requests and save Place instances
Class
Cache
Requests WebPage
Controller
PagesController
Requests data
Client
Service
PlaceService
Ambee
External API 1
Requests JSON data
External Sources
Requests Place data or Cache Statistics (JSON)
RestController
PlacesRestController (API)
Requests data
wAQI
External API 2
Ajax resquests directly to API: place data requests

Fig 1: Architecture Diagram

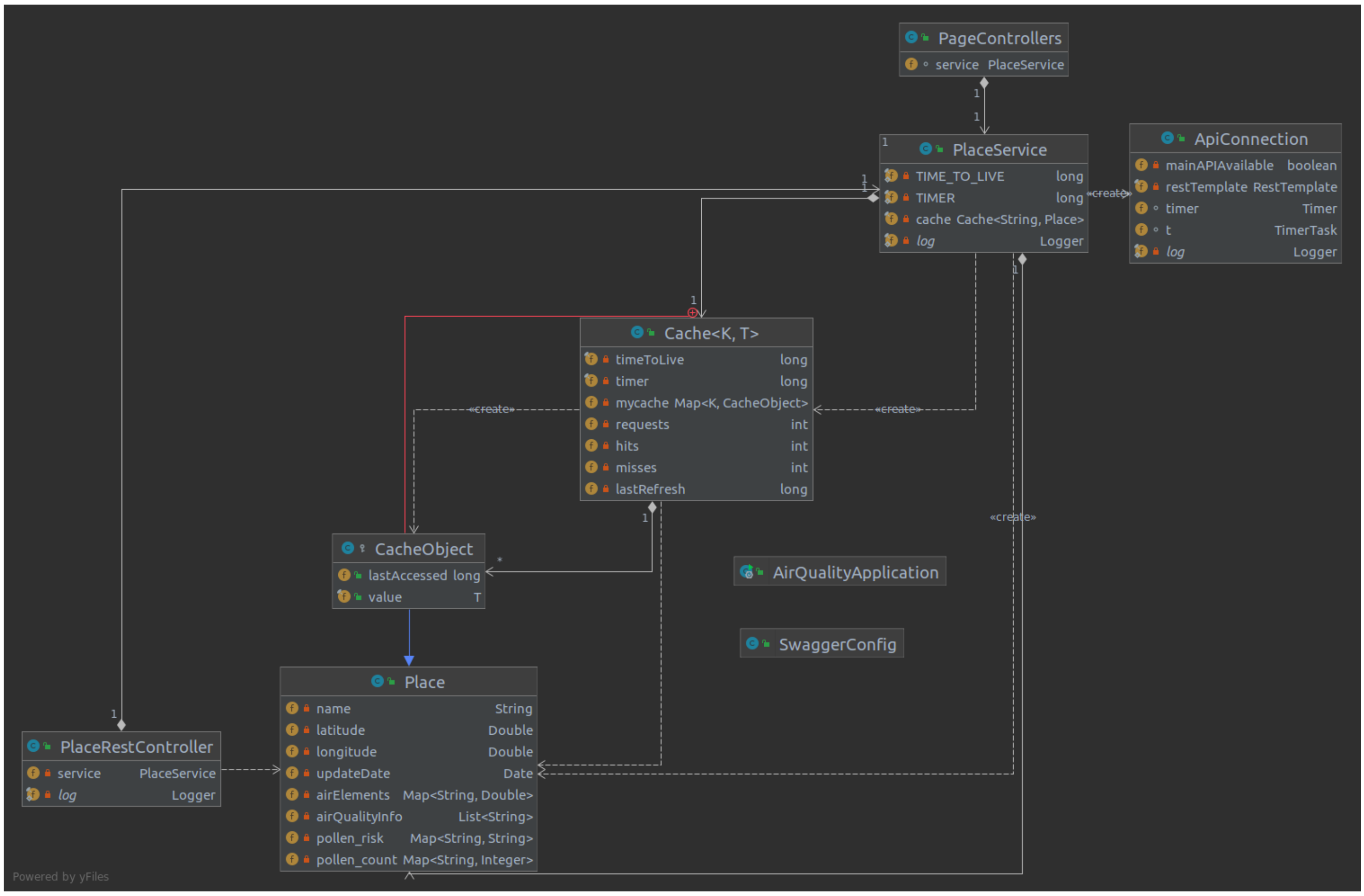
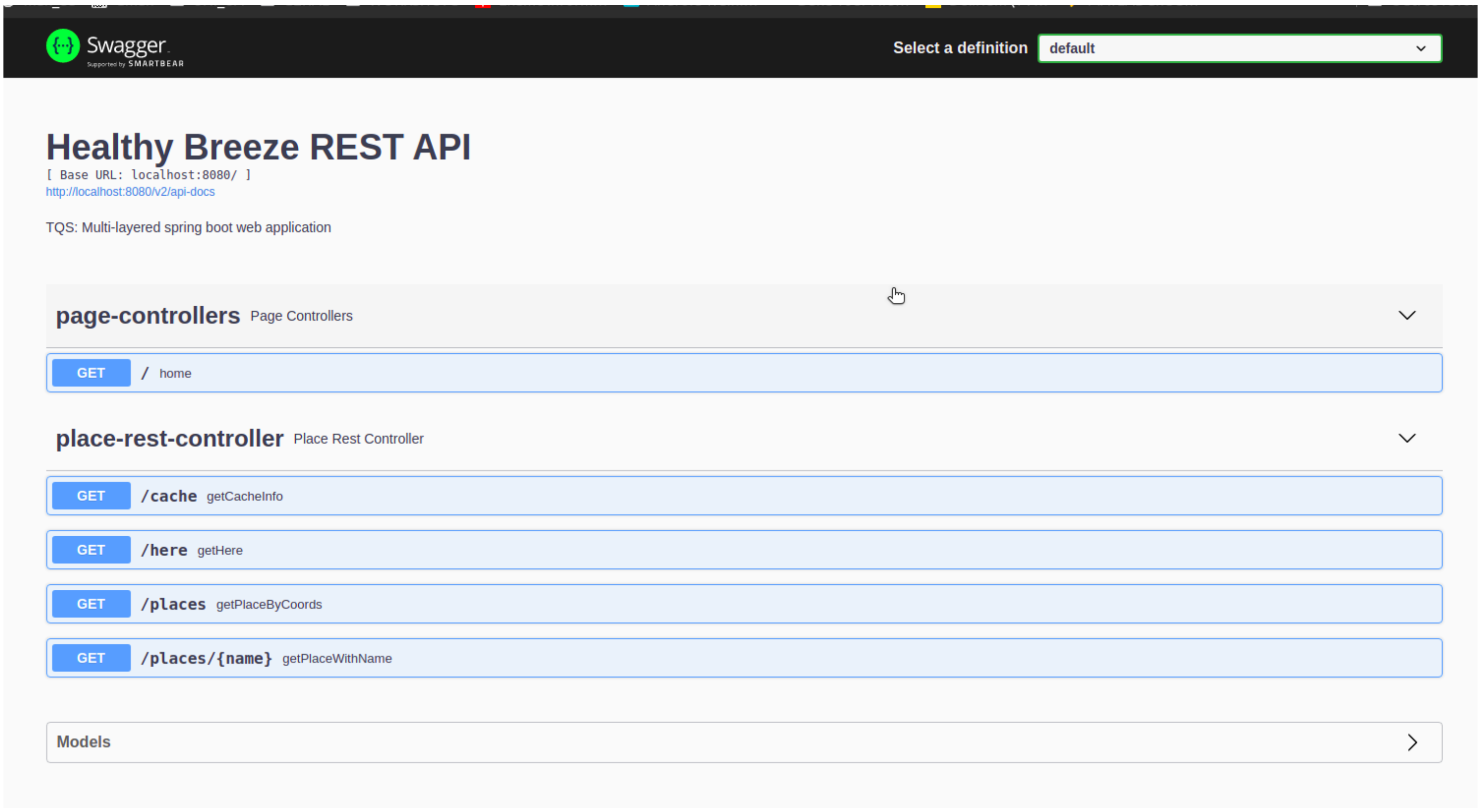


Fig 2: Model Diagram

2.3 API for developers

- /cache endpoint is where we can find information about cache statistics such as number of requests, misses, hits, last refresh, TTL and number of objects in cache.
- /here endpoint allows to find air quality information based on user IP.
- /places with parameters lat and long, that correspond to latitude and longitude values provide results of search by coordinates in internal APIs.
- /places/{name} is the endpoint that allows search by city name.
- /home is the application webpage endpoint


Select a definition default

Healthy Breeze REST API
[Base URL: localhost:8080/]
http://localhost:8080/v2/api-docs

TQS: Multi-layered spring boot web application

page-controllers Page Controllers
GET / home

place-rest-controller Place Rest Controller
GET /cache getCacheInfo
GET /here getHere
GET /places getPlaceByCoords
GET /places/{name} getPlaceWithName

Models

3 Quality assurance

3.1 Overall strategy for testing

The strategy may not have been the best because all the features were implemented before testing and some errors were detected too late.

Tools used for testing:

> JUnit for cache unit test

> SpringBootTest with TestRestTemplate, RestAssured, JUnit and Hamcrest for API integration test

> MockMvc, JUnit and Hamcrest for API unit test

> Selenium IDE and Selenium+Cucumber for the interface tests

3.2 Unit and integration testing

- ➔ Unit Tests were applied to cache class:
    - ➢ Create cache with valid arguments TTL and TIMER
    - ➢ Put and get from cache
    - ➢ Cache size
    - ➢ Empty cache
    - ➢ Remove from cache
    - ➢ Remove NonExistent Key
    - ➢ Get Expired Object
    - ➢ Get Not Expired Object
    - ➢ Clean cache
    - ➢ Get statistics: get hits, misses, requests

- ➔ API Unit Test
    - ➢ Get city by valid name and return place object
    - ➢ Get non existent city by name and return NOT_FOUND
    - ➢ Get current location information and return place object
    - ➢ Get city by valid coordinates and return place object
    - ➢ Get city by invalid coordinates
    - ➢ Get cache information

- ➔ API Integration Test
    - ➢ Get city by valid name
    - ➢ Get non existent city by name
    - ➢ Get current location information
    - ➢ Get city by valid coordinates
    - ➢ Get city by invalid coordinates

3.3 Functional testing

All the test cases were considered here.

- ➔ Tests about search by city name and coordinates in interface, along with respective error messages were made with cucumber and selenium.

- ➔ The Error Page test, that includes return to home page was made with SeleniumIDE.

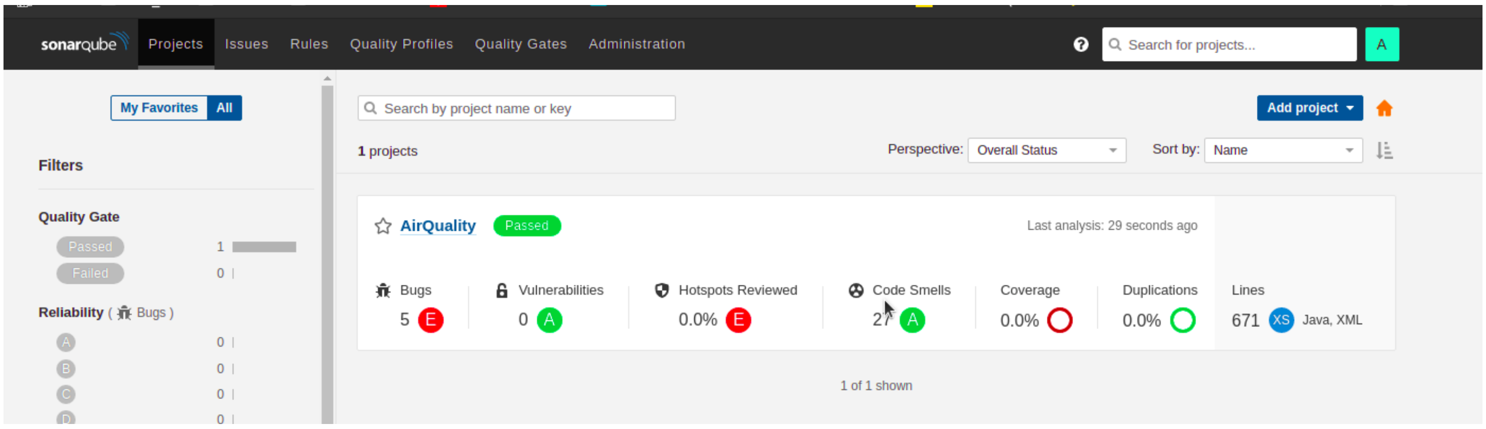
3.4 Static code analysis

This analysis was more complex than what I expected, some of the code smells were corrected later, for example, after replacing system.out.print by logger,.

The bugs, however, are a big problem because three of them are related to the cache class and to correct them would be necessary to rewrite it completely. Although the bugs were detected the class still works perfectly, so I decided to keep it even knowing the risks.

The other two bugs are related to some null pointer exception not caught with try-catch statement directly, but treated the same.

For some reason the tests were not included in this analysis even though I've tried it several times.



Filters

Quality Gate
Passed 1
Failed 0

Reliability (Bugs)
A 0
B 0
C 0
D 0

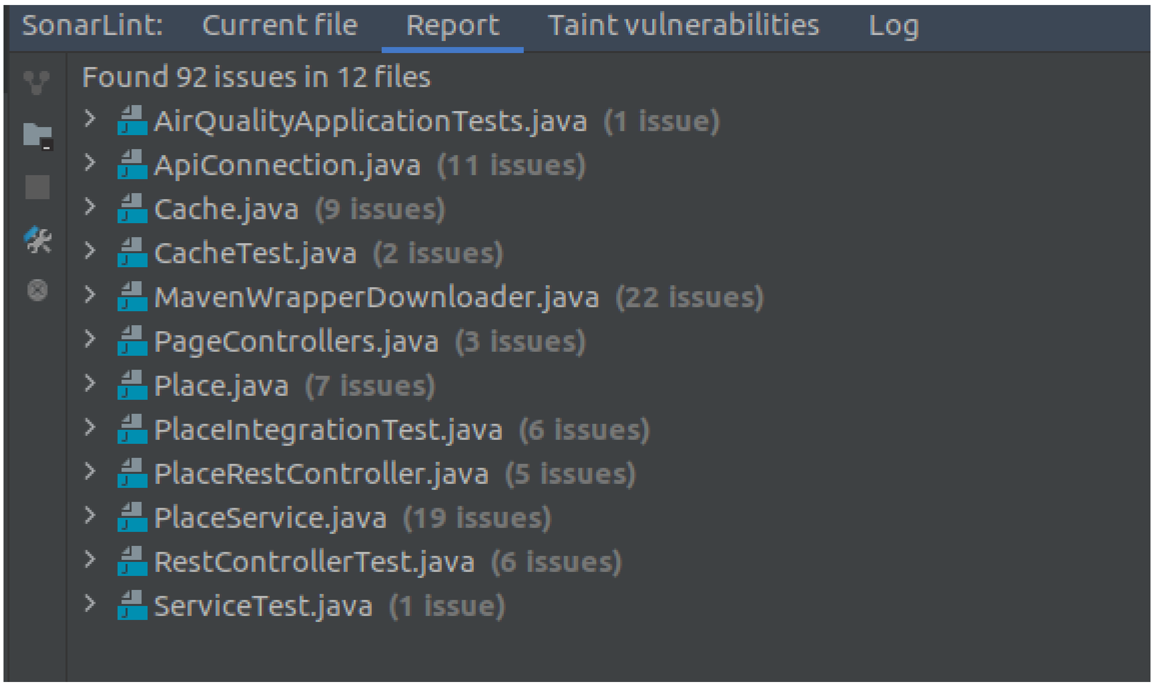
1 projects
Perspective: Overall Status
Sort by: Name

AirQuality Passed
Last analysis: 29 seconds ago

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
5 E	0 A	0.0% E	27 A	0.0%	0.0%	671 XS Java, XML

1 of 1 shown

I've also used SonarLint Plugin to analyse my code and some issues were detected and I corrected most of them.



```
Found 92 issues in 12 files
> AirQualityApplicationTests.java (1 issue)
> ApiConnection.java (11 issues)
> Cache.java (9 issues)
> CacheTest.java (2 issues)
> MavenWrapperDownloader.java (22 issues)
> PageControllers.java (3 issues)
> Place.java (7 issues)
> PlaceIntegrationTest.java (6 issues)
> PlaceRestController.java (5 issues)
> PlaceService.java (19 issues)
> RestControllerTest.java (6 issues)
> ServiceTest.java (1 issue)
```

4 References & resources

Project resources

- Videos demo: https://github.com/Mariainesrocha/TQS_AirQuality/tree/main/FICHEIROS
- Git Repository: https://github.com/Mariainesrocha/TQS_AirQuality

Reference materials

- **Ambee API:** https://docs.ambeedata.com/
- **wAQI API:** https://aqicn.org/api/
- **Swagger:** https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api
- **Cache implementation:** https://crunchify.com/how-to-create-a-simple-in-memory-cache-in-java-lightweight-cache/