HW1: Mid-term assignment report

Pedro Alexandre Gonçalves Marques [92926], 2021-05-14

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. With this web application, a web client can access weather information about a specific region. The integrated cache allows the application to store this data and retrieve it when necessary. The client can also see a few statistics about the web application as well as get the information available in the cache. There is also a REST API available that can be used for the same purposes.

1.2 Current limitations

The current project supports many operations for a client. However, some of these operations are not available in the web browser but just for API usage. Even though it has not been implemented, these features should have been available for a web browser user just as well as an API client. The list of available locations on the web browser is not static, as such it increases every time a client uses the REST API. That being said the same is not applicable to the web application. There is no way of adding new locations to the select menu besides using the API.

2 Product specification

2.1 Functional scope and supported interactions

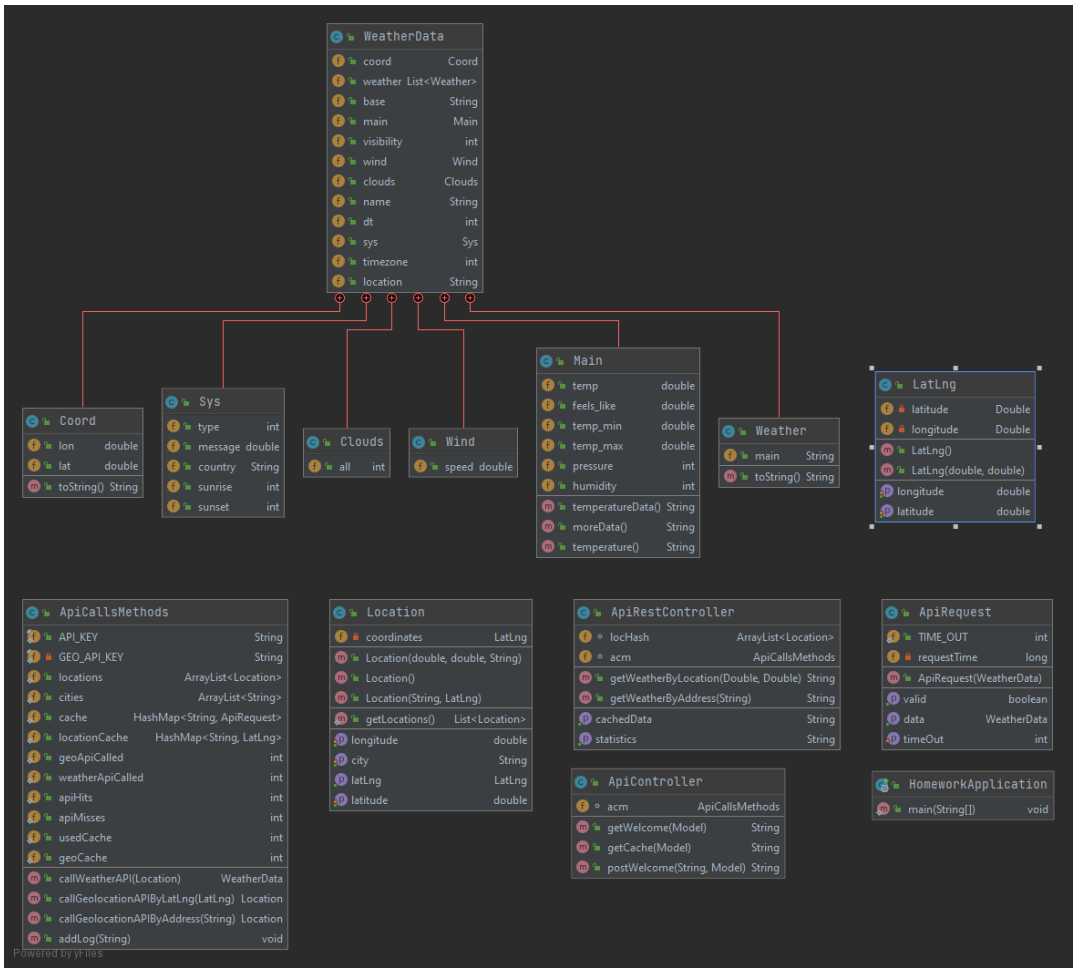
With this web application, a client can check up to date weather data about a certain region by address or location/zone input.

Within this project, two external APIs are being used although only one of them is available in the web browser and while using the REST API. For weather information purposes, I'm using openWeatherMap API as it provides a lot of information like pressure, humidity and temperature. However, this API revolves around coordinates for information retrieval, as such, the OpenCage Geocoder API and library for address conversion into a set of coordinates. The requests made to these APIs are being cached. The weather API's requests have a default TTL of 2 minutes, after that the cache data turns invalid and the next request will update the cache by making a new call to the API. However, when it comes to the Geocoder API these data are static and don't change overtime, as such there is no TTL on the Geocoder data cache.

This project's web page is very simple. An input box is available for a user to enter a location. After submitting said request the web application returns and displays information about the weather conditions of the selected zone. Under this input box, various statistics are provided as the number of calls for both APIs and the number of times the cache was used instead of executing those calls. As I have mentioned before the web application is more limited than the REST API. There's another web page available at "/cache". On this page, a set of weather data is provided of previously cached locations. As such, these sets of data can be validated by the field 'Is Valid' in the top left corner of each card.

When it comes to the REST API, all the previously mentioned features are available and we are also able to get the weather information by providing a set of coordinates.

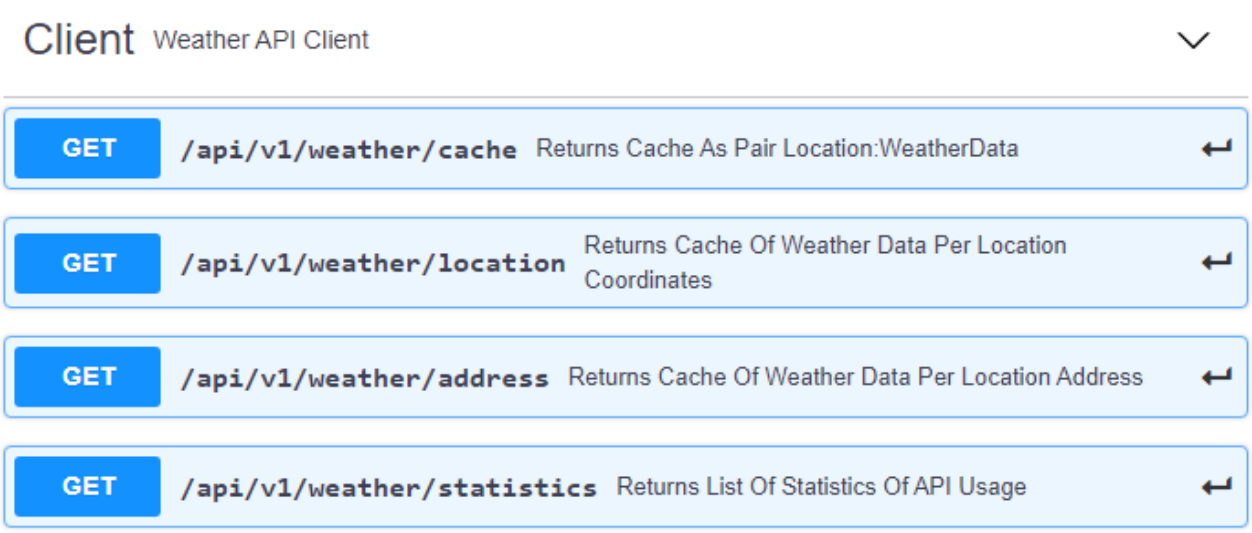
2.2 System architecture



The project is composed by 2 controllers, one for the frontend and a RESTController for the REST API. The statistics and cache are maintained in the ApiCallsMethods as well as the methods that make the calls to the external APIs. The weather API returns a JSON structure that is transformed into a WeatherData structure thanks to the Gson library. The mapping of the JSON structure to a Java Class was generated by the website JSON2CSHARP.
The Location class is used in order to create an association between a set of latitude and longitude coordinates and a location/address. The ApiRequest class allows for cache validation of data. As such, the TIME_OUT field (2 minutes by default) determines that data is invalid if it is displayed after a certain time of being retrieved from the API. This class only applies to WeatherData because geolocation information does not expire.

2.3 API for developers

As we have mentioned before this project comes with a REST API that can be accessed to gather information just has the web page. With this API a developer is capable of accessing Weather information by providing an address or a set of coordinates (this last method is not provided via web browser). The API's methods are documented below and the communication with the external APIs has been detailed previously.



https://app.swaggerhub.com/apis/PedroMarques27/TQS_Weather_API/1.0.0

To get weather information by providing a set of coordinates the URL should have the following format.

/api/v1/weather/location?lat={latitude}&lng={longitude}

To get weather information by providing a set of coordinates the URL should have the following format.

/api/v1/weather/address?q={query}

Both calls return a weather data structure as defined in code in the WeatherData class.
The statistics and cache endpoints don't require any parameters.
The first one provides information about calls to geocoder and weather API, number of times data was retrieved from geolocator cache and weather cache as well as the number of hits and misses on external API calls.
The cache endpoint returns all available data in the weather cache as a map of Address: WeatherData. All responses are returned as a JSON structure, thanks to the GSON library.

3 Quality assurance

3.1 Overall strategy for testing

For code development I started to define simple features and requirements that I would like to see in my final product. I proceeded to create some tests for these features in order to ensure everything was working as intended. Overtime with development more features where created and more tests originated in order to accompany the code development and assure the well function of the web application.

3.2 Unit and integration testing

Unit tests where conducted in order to ensure that the data provided by the client was valid. The ApiCallsMethod class was the most tested in order to ensure that the data provided by the external API would be correctly formatted to be used by the web application.

```
@Test
void callGeolocationAPIByLatLng() throws IOException{
    Location data = acm.callGeolocationAPIByLatLng(point2);
    System.out.println(data.getLatitude());
    assertThat(data.getCity(), equalTo( operand: "15948 Petón do Currumil, Espanha"));
}

@Test
void callGeolocationAPIByAddress() throws IOException {
    Location data = acm.callGeolocationAPIByAddress( name: "15948 Petón do Currumil, Espanha");
    assertThat(data.getLatitude(), equalTo( operand: 42.6446276));
    assertThat(data.getLongitude(), equalTo( operand: -8.9490691));

}
```

For example, these tests guarantee that the OpenCage Geolocator API returns the location for the specific set of coordinates or address provided.

```
@Test
void isValidBefore() throws InterruptedException {

    apiRequest = new ApiRequest(new WeatherData());
    apiRequest.setTimeOut(500);

    Thread.sleep( l: 200);
    assertThat(apiRequest.isValid(), equalTo( operand: true));
}

@Test
void isValidAfter() throws InterruptedException {


    apiRequest = new ApiRequest(new WeatherData());
    apiRequest.setTimeOut(500);

    Thread.sleep( l: 1000);
    assertThat(apiRequest.isValid(), equalTo( operand: false));
}
```

The ApiRequest class determines whether a certain weather data is valid or not. Two tests are conducted: one where the validity is evaluated before the defined time out and one after.

```java
@Test
@Order(7)
void cacheDoesntAddIfExisting() throws IOException, ParseException {...}

@Test
@Order(8)
void cacheNotValidUpdate() throws IOException, ParseException, InterruptedException {...}

@Test
@Order(9)
void getGeoCachedData() throws Exception {...}
```

The first method in the image above verifies that when a request for valid weather information of a location that was already previously requested is made, the cache returns valid information instead of making a call to the external API. Furthermore, the second method verifies that the said call is done if the information provided by the cache is not valid anymore and that the weather cache is updated. Finally, the last method verifies that instead of making a Geolocator API call for a previously requested location, the cache returns the needed data.

When it comes to the frontend of the web application, tests we're conducted using the Selenium IDE recorder and using some unit tests it was possible to assess the implementation of the cache mechanism and the API calls made. In the test, the user started by entering 'Aveiro' in the input text box, followed by 'Porto' and 'Aveiro' again. That being said, the statistics where also being displayed on screen and they were updated correctly. The first two requests, as expected, made calls to the external APIs and, also as expected, the last one just retrieved the data from the geolocator and weather cache. After that the user was redirected to */cache,* and, as expected, all the information that was previously searched for and was in cache was retrieved and display, even if invalid.

3.3 Functional testing

It was important to assure that all functionalities where working for in my application. As such a test was developed for each endpoint of the REST API. Furthermore Selenium was utilized for testing of the web application.
In regards to the rest API, the ApiRestTest class is responsible for the tests to the created API and the cache usage. However, all methods used to access the API are defined inside the class APICallsMethods, as well as the cache itself. As such, MockMvc was utilized in order to mock calls to the created REST API.

```java
@Test
@Order(2)
void getWeatherByLocation() throws Exception {
    mockMvc
            .perform(get( urlTemplate: "/api/v1/weather/location").param( name: "lat", String.valueOf(40.6446276)).param( name: "lng", ...values: "-8.6498691"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString( substring: "Churrasqueira Don Torradinho, Rua do Gravito, 3800-196 Aveiro, Portugal")));
}
@Test
@Order(3)
void getWeatherByAddress() throws Exception {
    mockMvc
            .perform(get( urlTemplate: "/api/v1/weather/address").param( name: "q",  ...values: "Churrasqueira Don Torradinho, Rua do Gravito, 3800-196 Aveiro, Portugal"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString( substring: "Churrasqueira Don Torradinho, Rua do Gravito, 3800-196 Aveiro, Portugal")));

}
```

In the image above, I used mockMvc to make sure the methods of getting weather information by a pair of coordinates or by an address where working. As such I mocked a call to the API and expected a specific value in return as I had defined in the setup() method.

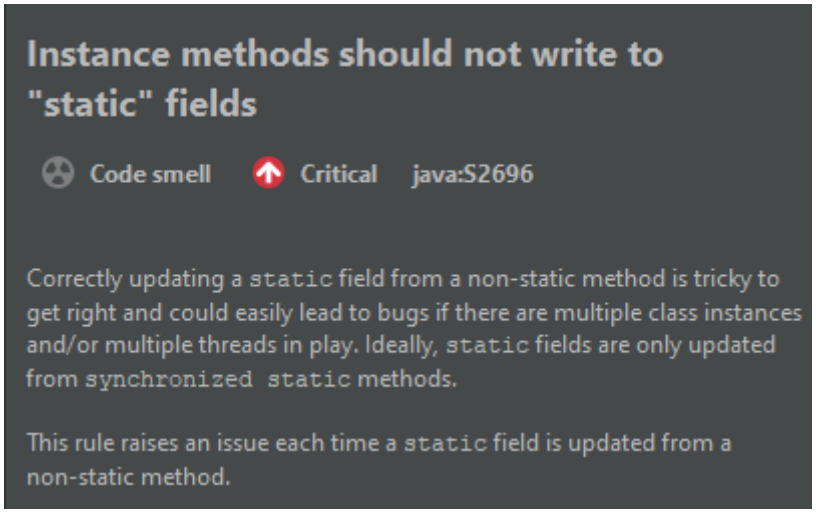
The method getStatistics() was also intended to test the return of a user who calls the API to get statistics as presented in the frontend along with number of hits and misses on external API calls. It also allowed to test the cache.

```
assertThat(statistics.get("hits"), equalTo( operand: 2));
assertThat(statistics.get("GeolocationApiCalls"), equalTo( operand: 2));
assertThat(statistics.get("WeatherApiCalls"), equalTo( operand: 2));
assertThat(statistics.get("weatherCacheUsage"), equalTo( operand: 1));
assertThat(statistics.get("geolocatorCacheUsage"), equalTo( operand: 1));
assertThat(statistics.get("misses"), equalTo( operand: 0));
```

Many more methods were developed in order to test the cache functionality of the application.

3.4 Static code analysis

In regards to static code analysis the Intellij SonarLint plugin was installed in order to reduce code smells and bugs. As we can see in the images below many low level code smells were detected. Furthermore, some critical code smells and bugs were also encountered mainly in the ApiCallsMethods class.



Even though many of the errors were fixed and solutions were found, some of them were left untouched as they would affect the application's functionalities. As such, the error above is one of these code smells. Even though it might be a critical code smell, these specific static fields are supposed to be able to be changed by other methods in other classes.

4 References & resources

Project resources

- Demonstration video was stored in the GitHub repository

Reference materials

OpenWeatherApi, https://openweathermap.org/current

OpenCageGeolocator, https://opencagedata.com/api#quickstart

Gson Deserialization Cookbook, https://www.baeldung.com/gson-deserialization-guide

Hamcrest Collections Cookbook, https://www.baeldung.com/hamcrest-collections-arrays

Integration Testing in Spring, https://www.baeldung.com/integration-testing-in-spring

JSON2CSharp, https://json2csharp.com/json-to-pojo