Software Quality and Tests
Homework 1 : Test Development

Author: **André Pedrosa**
N. Mec.: **85098**
Date: **06/05/2019**
Repository: `https://github.com/aspedrosa/weather_forecast/`

Index

1 Intro

This homework consisted in developing a web project that allows to query the meteorology for the next days, for a specific location, which the main goal was to develop the different classes of tests (unitary, unitary with dependency isolation, integration and functional) along with the development of the project.

The main components of the solution are:

- **Web page**, allowing the user to select and search for locations and also giving the choice to the user to choose how many days he wants to see.
- **Rest API**, that can be called by external clients allowing to query the meteorology for a specific location.
- **Integration with an external source**, from where the real meteorologic data will be obtained, so the app built is a client of another source.
- **Caching mechanism**, where stores the last retrieved results from recent calls to the external sources, having also some statistics about its functioning (number of requests, hits and misses)

For the website just jQuery and Bootstrap was used. To implement the server side logic a web application was built using SpringBoot.

2 Solution

2.1 Front End

The front end consists on a single minimalist page, where the user can do every thing that the system allows on one page.

At the start the user can either search from a specific location or use the "I'm Feeling Lucky" feature where meteorology for the first location found on the results of the search mechanism is displayed. At his time the user also can choose how much days of meteorologic data he wants to see.

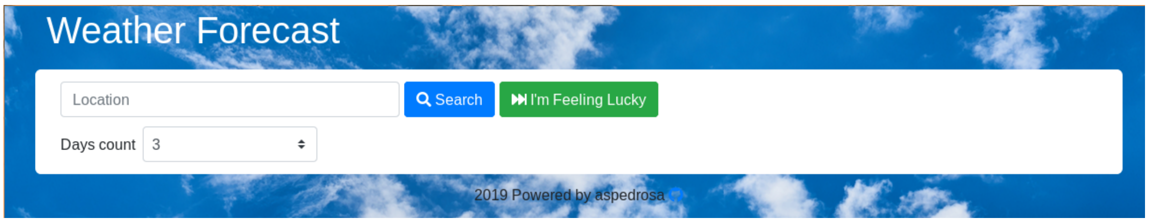
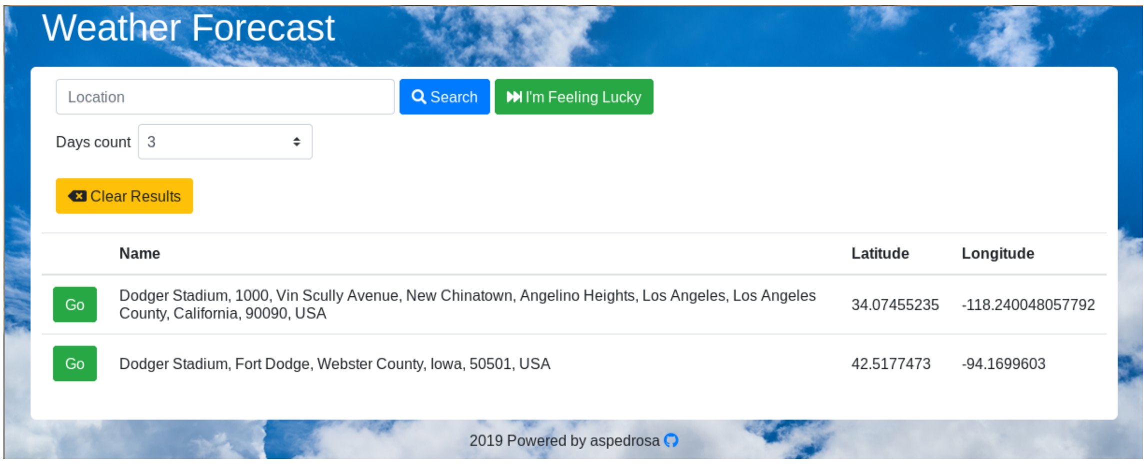


Figure 1: How the web page is displayed at the beginning

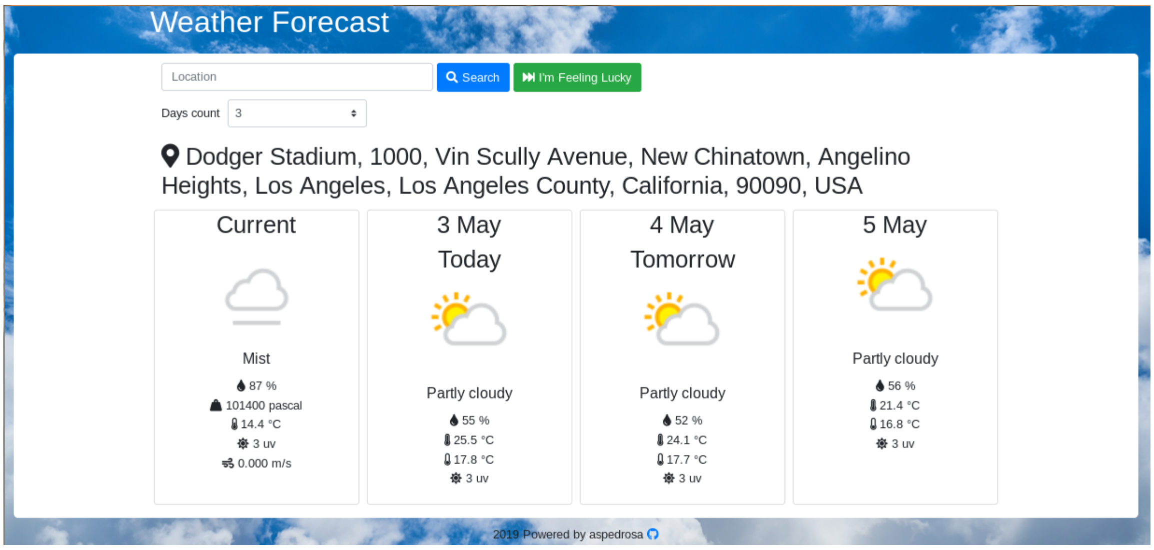
Once the user searches for a location, a table appears with the names of related location as well as the respective geographic coordinates.



	Name	Latitude	Longitude
Go	Dodger Stadium, 1000, Vin Scully Avenue, New Chinatown, Angelino Heights, Los Angeles, Los Angeles County, California, 90090, USA	34.07455235	-118.240048057792
Go	Dodger Stadium, Fort Dodge, Webster County, Iowa, 50501, USA	42.5177473	-94.1699603

Figure 2: Search results for dodger stadium

When the user chooses a location two types of forecast data is displayed. Current and for future days, being the number of days the same as the value on the "Days Count" select.



Dodger Stadium, 1000, Vin Scully Avenue, New Chinatown, Angelino Heights, Los Angeles, Los Angeles County, California, 90090, USA

Current	3 May Today	4 May Tomorrow	5 May
Mist	Partly cloudy	Partly cloudy	Partly cloudy
87 %	55 %	52 %	56 %
101400 pascal	25.5 °C	24.1 °C	21.4 °C
14.4 °C	17.8 °C	17.7 °C	16.8 °C
3 uv	3 uv	3 uv	3 uv
0.000 m/s			

Figure 3: Forecast data for Dodger Stadium, California, USA

2.2 Back End

As mentioned before, back end uses SpringBoot that divides the system in three layers. Controllers that process HTTP requests and pass the received data to the service layer. The service

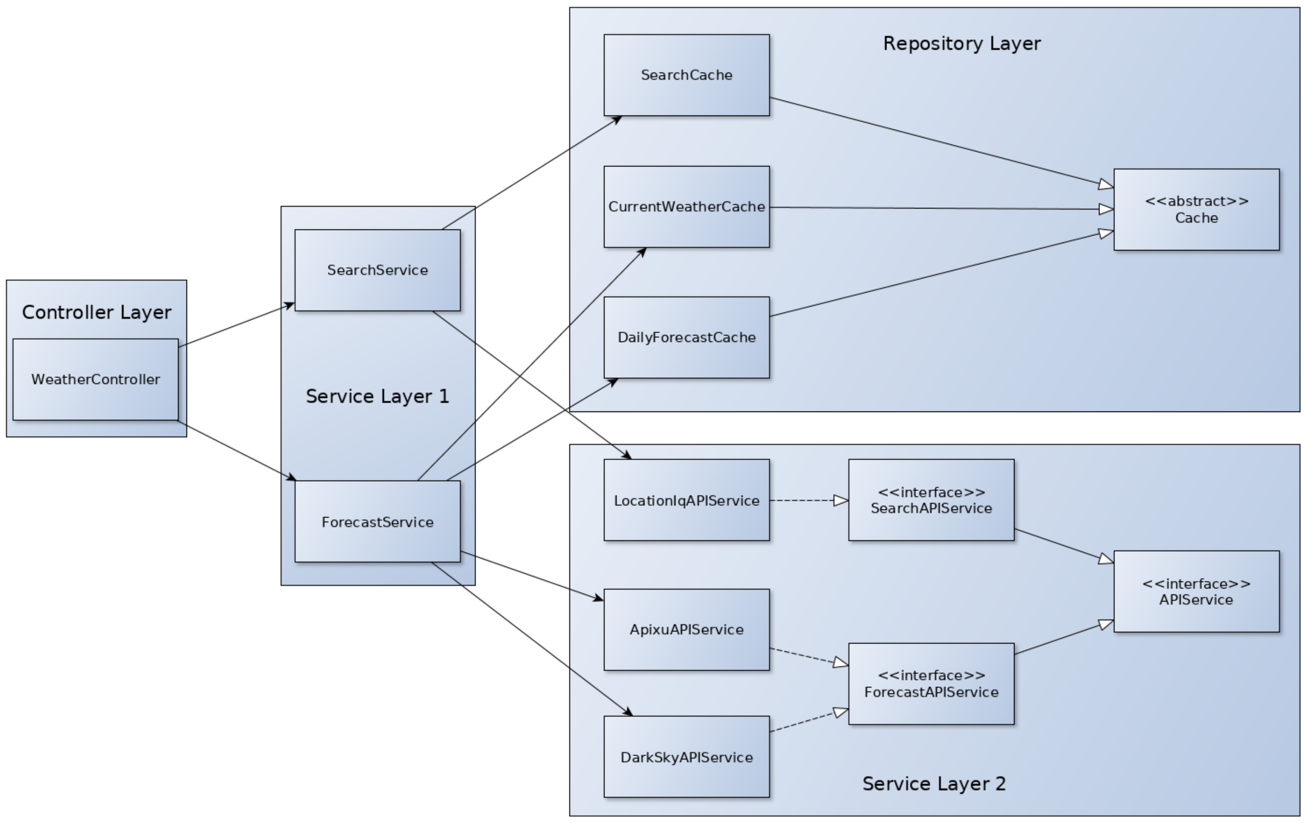


Figure 4: Class diagram for the solution implemented

layer imposes the business logic. The last layers is the repositories where the service layers can store data.

On figure 4 is shown the several layers of the application as well as the classes in each one. Abstraction where used to allow extensibility to the application such as caching new types of data, using other external apis and using other external apis for new types of data.

2.2.1 Controller

The controller layer is composed by a single controller where the REST API is defined. Has three paths:

- api/forecast : Obtains meteorologic data of a location
  - latitude : double : request parameter : latitude of the location
  - longitude : double : request parameter : longitude of the location
  - days_count : int : request parameter : how much days of forecast data to retrieve

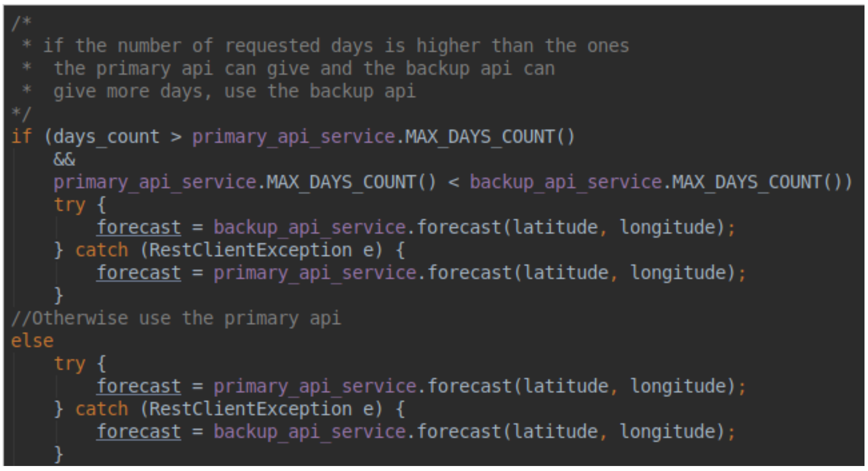
- api/search : searches for a location. Used to obtain geographic coordinates
  - location : String : request parameter : location to search for

The controller is also responsible to validate to parameters received before sending them to the first service layer (i.e. latitude must be a double).

2.2.2 Services

Is composed by the up layer (Service Layer 1) and the lower layer (Service Layer 2). The upper layers is used by the WeatherController and its role is to check the cache for the requested data and if is not there should interact with the lower layer. The role of the lower layer is to retrieve meteorologic from the external api. Whenever the upper layer contacts the lower layer to get data successfully, must store that received data in the cache.

The upper layer is composed by two services, Search and Forecast. Both follow the flow mentioned previously, however the Forecast service as a complexer logic to when query the lower layer. First the Forecast service is in charge of two types of data, current and daily forecast, leading to two caches with different expiration times. Secondly the Forecast service uses two services of the lower layer. One used as a primary and other as backup in case the primary fails. Furthermore, if the client asks for a high number of days of forecast data, the Forecast service tries to use the one that offers the higher number of days of forecast data.



```
/*
 * if the number of requested days is higher than the ones
 *  the primary api can give and the backup api can
 *  give more days, use the backup api
*/
if (days_count > primary_api_service.MAX_DAYS_COUNT()
    &&
    primary_api_service.MAX_DAYS_COUNT() < backup_api_service.MAX_DAYS_COUNT())
    try {
        forecast = backup_api_service.forecast(latitude, longitude);
    } catch (RestClientException e) {
        forecast = primary_api_service.forecast(latitude, longitude);
    }
//Otherwise use the primary api
else
    try {
        forecast = primary_api_service.forecast(latitude, longitude);
    } catch (RestClientException e) {
        forecast = backup_api_service.forecast(latitude, longitude);
    }
```

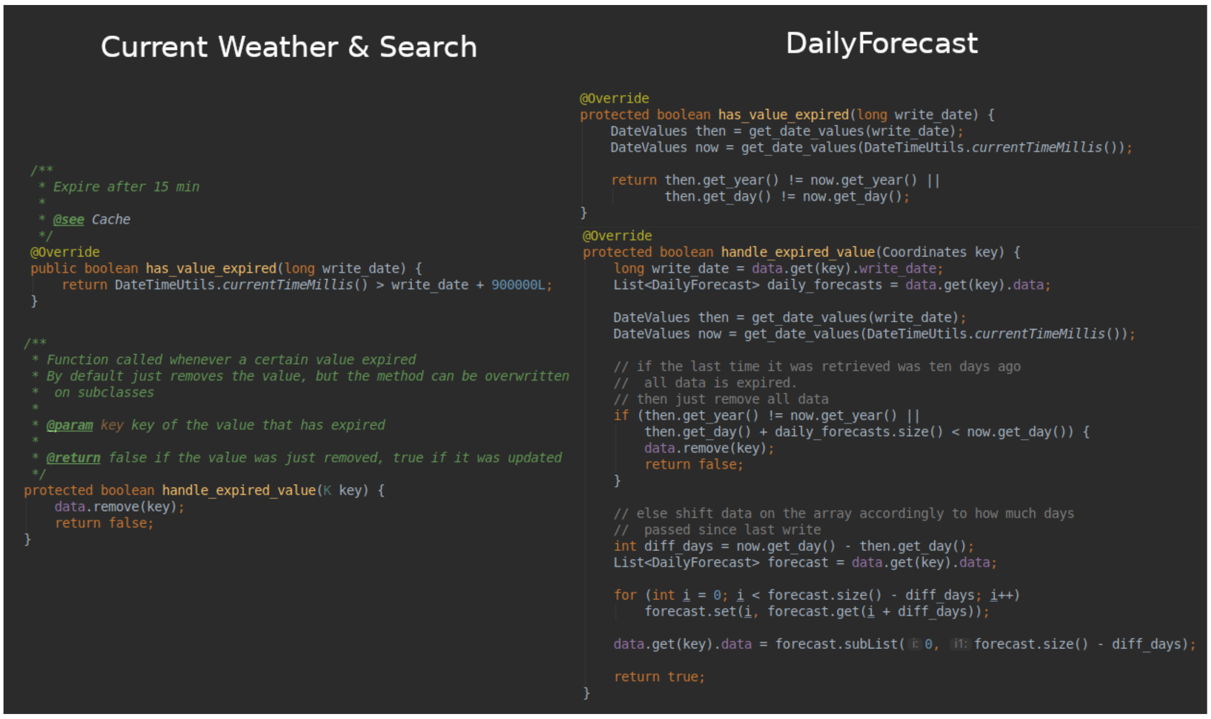
Figure 5: Block of code of the function retrieve_from_apis of the class ForecastService where the choice of which api to use happens

Third both current and daily data are retrieved on one call to an external api. So, if the current data expires, both current and daily data will be stored in cache after the call and overwriting of existing data may occur but because the new data is more recent, is more accurate, so overwriting ends up being a good thing.

2.2.3 Repositories

The repository layer is in charge implementing caching mechanisms for the application. This is done associating data to a key, in other words, a map was used. Each cached data has a time-to-live after which the data stored expires and if clients asks for that data, should consider as not not having that data cached.

Different types of data have different strategies to see if that expired. For current meteorologic data and search data is just checking, whenever its requested, if a certain time has passed since the last write (30 days for search data, because location data it's pretty stable and 15 min for current meteorologic data). In this cases the data associated is just removed if its considered as expired. For daily forecast all data isn't removed since daily forecast data can be associated with several days, witch means is a list of values each position associated with a day. So the data "expires" the current date is in a different data comparing to the date on the last write (i.e. if data is stored at 12/05/2019 10:00, at 12/05/2019 15:00 is still valid but at 13/05/2019 10:00 isn't). When this happens the data needs to be shifted the difference of days since of last write, however if the number of days that had pass is higher than the number of days stored than the data is just removed.



```java
/**
 * Expire after 15 min
 *
 * @see Cache
 */
@Override
public boolean has_value_expired(long write_date) {
    return DateTimeUtils.currentTimeMillis() > write_date + 900000L;
}

/**
 * Function called whenever a certain value expired
 * By default just removes the value, but the method can be overwritten
 *  on subclasses
 *
 * @param key key of the value that has expired
 *
 * @return false if the value was just removed, true if it was updated
 */
protected boolean handle_expired_value(K key) {
    data.remove(key);
    return false;
}
```

```java
@Override
protected boolean has_value_expired(long write_date) {
    DateValues then = get_date_values(write_date);
    DateValues now = get_date_values(DateTimeUtils.currentTimeMillis());

    return then.get_year() != now.get_year() ||
           then.get_day() != now.get_day();
}
@Override
protected boolean handle_expired_value(Coordinates key) {
    long write_date = data.get(key).write_date;
    List<DailyForecast> daily_forecasts = data.get(key).data;

    DateValues then = get_date_values(write_date);
    DateValues now = get_date_values(DateTimeUtils.currentTimeMillis());

    // if the last time it was retrieved was ten days ago
    //  all data is expired.
    // then just remove all data
    if (then.get_year() != now.get_year() ||
        then.get_day() + daily_forecasts.size() < now.get_day()) {
        data.remove(key);
        return false;
    }

    // else shift data on the array accordingly to how much days
    //  passed since last write
    int diff_days = now.get_day() - then.get_day();
    List<DailyForecast> forecast = data.get(key).data;

    for (int i = 0; i < forecast.size() - diff_days; i++)
        forecast.set(i, forecast.get(i + diff_days));

    data.get(key).data = forecast.subList( i: 0,  i1: forecast.size() - diff_days);

    return true;
}
```

Figure 6: Difference of cache strategies between current weather and search and daily forecast. The two methods he shown are called whenever the cached is queried for data. If the method has_value_expired returns true then the method handle_expired_value is called. See method get_cached_data of class Cache

3 Tests

As said in the Intro, the main goal of this homework is to develop a project with test development along. For each class unit tests where made, where they make sense. If a specific class depends on another class developed mocking mechanisms, using Mockito, were used so the individual class could be tested, however integration tests were also made.

3.1 Front End

Functional tests that validate the functional requirements for the application such as:

- give an error if the user inserts an unknown location
- the number of days of forecast data displayed must be the same as the number of days requested
- give an error if the user insert an invalid location
- when the user uses the I'm Feeling Lucky feature, the forecast data should be displayed, without asking to choose a location

Also an integration test is done with all three layers of the web app, controllers, services and repositories, mocking the response of the external api, in other words, a full stack test.

3.2 Back End

3.2.1 Controller

Here is tested the responsibilities of the controllers:

- generate the response accordingly to the response received from the service layer
- validation of the parameters received

The unitests for controllers were done by mocking the service layers and the application context is restricted to Model-View-Controller(MVC) components. Integration tests were done by mocking the web context, testing the integration between the three layers (controllers, services and repositories).

3.2.2 Services

For unitesting the upper layer the lower layer services and repositories are mocked allowing to test:

- ForecastService
  - usage of the cache

  - query external api when one of the cache doesn't satisfies the request (i.e. number of days requested is higher than the days stored or no current data cached)
  - choice between the primary and backup api according to number of days requested
  - choice between the primary and backup api according to availability
- SearchService
  - successful responses
  - not found location

Also for the upper layers integration tests were done by mocking the response to external apis allowing to test the service and repository layers.

For the lower layer only unitests were made mocking the response of the external api. The services that implement the interface ForecastAPIService (DarkSkyAPIService and ApixuAPIService) was tested:

- the conversion of weather metrics to si units
- number of days processed and generated had to be the same as the number of days received from the external api

For the services that implement the SearchAPIService (LocationIqAPIService) was tested:

- handle the user inserting an unknown location leading to no results
- handle receiving multiple entries with the same name from the external api

3.2.3 Repositories

To test the layer of repositories only units were done. Because the class SearchCache and CurrentWeatherCache only change the time-to-live, these classes were not tests however the abstract was tested by creating a new implementation during the tests with a low time-to-live allowing to test:

- receiving null after time-to-live
- receive data after insert
- not receiving data if no inserted were made

For the first test an external library(Awaitility) was used that allows to verify a condition during a certain time. If at the end of that time the condition is not met, the test fails.

Because the DailyForecastCache has a different ways to handle data expiration and the expiration itself is different a separate test was done testing:

- expiration data is correctly handled

- data expires on the correct time
- the action of overriding data is done correctly

To test the expiration of the data an external library (joda-time) was used to allow control the system time.

4 Sonarqube