# SOEN 6441 Advanced Programming Practices (Fall 2020)

# Project Assignment #1 (v1.1)

**Due date #1 (Group Information): Monday, October 26**
**Due date #2 (System Submission): Monday, November 9**

This is the first of two programming assignments in our course. Note that the assignments are not 'stand-alone', but rather build on top of each other to form a bigger project.

The assignments focus on applying the concepts covered in the lectures, such as functional programming, lambdas, streams, unit testing, asynchronous programming with futures, and actor-based programming, in the context of a web application.

The assignments have to be developed based on the *Play Framework*, a full-stack reactive framework for the JVM, which is also used in large-scale commercial deployments by companies, such as LinkedIn, Morgan Stanley or Walmart.

Note that this assignment has both a group part and an individual part. Your group must submit a single application. Your own marks will be based on the group work (50%) and your individual contribution (50%).

**Play Application: "TweeterLytics" (Group Part–50%).** The overall goal is to develop a web application that can analyze the live feed of the Twitter API.

The group part contains the setup of the overall framework for this web app, as well as a simple web interface that takes a string with one or multiple search keywords (input text field at top of the page, see Figure 1) and then displays the latest 10 tweets matching the keyword(s) below the search field. Each match must include the name of the tweet's author and (if available) the geolocation. Note that you have to search for the whole phrase (and not each keyword individually).

```
+--------------------------------------------------------------------+
|                       Welcome to TweeterLytics                     |
|                                                                    |
|                                                                    |
|      +--------------------------------------------+     +---+     |
|      |  (enter search terms)                      |     |Go!|     |
|      +--------------------------------------------+     +---+     |
|                                                                    |
|                                                                    |
+--------------------------------------------------------------------+
```

Figure 1: TweeterLytics page when first started

The user has to be able to enter new search terms on the output page, which will result in 10 more tweets being displayed (i.e., a second search will add 10 more results above the 10 tweets from the first search and so on – see Figure 2). Keep at most 10 search queries.

You have to use the (free) Twitter API to access the tweet information, see https://developer.twitter.com.

```
+-----------------------------------------------------------------------+
|                          Welcome to TweeterLytics                     |
|                                                                       |
|                                                                       |
|      +----------------------------------------------+      +---+     |
|      |  (enter search terms)                        |      |Go!|     |
|      +----------------------------------------------+      +---+     |
|                                                                       |
|                                                                       |
|-----------------------------------------------------------------------|
|      Search terms: Montreal Sushi :-(                                 |
|                                                                       |
|      1. User A (loc): "tweet 1 #sushi"                                |
|      2. User B (loc): "tweet 2 #fun"                                  |
|         ...                                                           |
|     10. ...                                                           |
|-----------------------------------------------------------------------|
|      Search terms: cat cafe tokyo :-)                                 |
|                                                                       |
|      1. User C (loc): "tweet 1 #cat"                                  |
|      2. User B (loc): "tweet 2"                                       |
|         ...                                                           |
|     10. ...                                                           |
+-----------------------------------------------------------------------+
```

Figure 2: TweeterLytics page after two searches, showing two sets of results with 10 tweets each

Process the results using the Java 8+ Streams API to add hyperlinks to user names, locations (if available), and hashtags (used in the individual parts detailed below).

**Note:** for this assignment, you do not need to stream updates to the user interface. In other words, the front-end page is static until a "refresh" (or new search) is triggered.

**Individual Part (50%).** In addition to the group part above, each team member has to work on *one* of the following features. Your team has to decide who works on which part – two members cannot work on the same task!

If your team has less than four members, only pick one individual task per team member.

Note that the individual part counts for 50% of the overall assignment marks.

**a) User Profile:** Create a web page containing all available profile information about a user, as well as the last 10 tweets of that user. This profile page must be hyperlinked with the user name from the tweets on the main search page.

**b) Tweet Words:** For a search query (one or multiple keywords), display a word-level statistics for the 250 latest tweets (less if fewer are available), counting all unique words in descending order (by frequency of the words). This statistics page must be hyperlinked from the search terms above the results. You must use the Java 8 Streams API to process the tweets.

**c) Hashtags:** For a given hashtag (linked from the tweets in the search results), display the 10 latest tweets containing this hashtag, in the same format as the tweets on the main search page.

**d) Tweet Sentiment:** Given a stream of (up to 250) tweets, determine if these tweets are overall happy `':-)'`, sad `':-('` or neutral `':-|'`. This *sentiment* will be displayed on the main search page for each search query (see Figure 2). Process the stream of tweets, finding happy and sad words, emojis, and emoticons and counting them (create your own "happy/sad" lists). If a tweet contains more than 70% "happy" strings, return a happy sentiment `:-)`, for more than 70% sad, return an unhappy emoticon `:-(`, otherwise a neutral one `:-|`. For a stream of tweets, compute the average of the individual tweets' results. You must use the Java 8 Streams API to process the tweets.

**Coding guidelines.** Your submission must satisfy the following requirements (these apply to both the group work and the individual work):

***Play Framework.*** Note the following general guidelines for using the Play Framework and building the application:

1. Your application must be based on the Play framework.
2. It must be possible to build and run your app using the standard `sbt run` command.
3. Any required third-party libraries must be automatically resolved through `sbt`.
4. The standard build targets (`sbt run/test/jacoco/javadoc/clean`) must all work correctly.
5. You can optionally use the `twitter4J` (or any other) library, but you have to wrap any synchronous APIs to make them asynchronous (see below).
6. Do **not** put business logic into the controller. Use dedicated model classes (MVC pattern) for your application functions.
7. Write *a single controller* for your app, integrating the individual parts defined below.
8. Make sure your server correctly handles *user sessions* for multiple searches – i.e., if you open your app from a second browser, you must not see the search results from an ongoing session!
9. Do **not** fetch the same tweets multiple times (e.g., for passing them between different tasks) – develop an appropriate caching strategy for your appplication.

You can setup a DevOps server for your team (e.g., using Jenkins), but this is not required for the project.

***Documentation.*** The following guidelines concern your documentation:

1. Document all your classes and methods with *Javadoc*.
2. This includes private methods and unit tests!
3. Make sure all classes and methods include an `@author` tag (both for group and individual work).
4. Include the generated HTML Javadoc in your submission (make sure you generate it for the private methods as well).

***Reactive Programming.*** The following guidelines refer to using reactive programming techniques. These will become especially important for the second assignment (second part of the project), so you should do your best to get it right the first time.

1. Your controller actions must be *asynchronous* (non-blocking), using Java 8's `CompletionStage<T>` `/CompletableFuture<T>`.
2. Do not use `.get()/.join()`, `thread.sleep` or similar to block for a future's result anywhere in your code (only exception are unit tests, see below)!

Note: any blocking code in your app will lead to significant marks reduction!

***Testing.*** Testing your code is a very important part of the project. Make sure you write your unit tests at the same time as you implement your features. **Code that is not tested will count as not implemented!** – In other words, if you completely implement the whole project, but do not have a single test, your submission will receive 0 marks!

1. Create JUnit tests for all your classes. You must have tests for every method in your controller and every controller action, as well as any additional classes you wrote. Compute your test coverage with JaCoCo. You must have 100% test coverage of your classes, methods, and lines (excluding classes automatically generated by Play).
2. Within a test, you can use `.get()/.join()` to wait for a result.
3. **Never** call the live Twitter API from a unit test, use a mock class for testing instead. For testing your classes using the Twitter API, you must use either your own, factory-generated mock class (you can use Dependency Injection (DI) with the Google Guice library) or use the *Mockito* framework.
4. Your unit tests must be properly designed by finding the equivalence classes and doing partition testing. A unit test that does not properly check the result values (e.g., only checking if a result is non-empty, rather than comparing the values), will count as *not written* for the purpose of test coverage!

**Submission.** There are two deadlines, the first for submitting the information which developer handles the individual part and the second for the system. Note that for each submission, only one upload per group is required.

***Submission 1: Group Information.*** You must submit the following team information on Moodle by the first due date:

**Github:** The URL of your Github (or Gitlab) repository containing the project.

**Team setup:** List all team members, indicating which of the individual parts above are assigned to each team member. An individual task can only be assigned to one team member and the marks (50% of the total marks) for that task will go to the designated team member only.

***Submission 2: System.*** You must submit your code electronically on Moodle by the due date (late submission will incur a penalty, see Moodle for details). Include a signed (by all team members) *Expectation of originality* form[1] with your submission.

**Demo.** You must also demo your code to your TA during a Zoom session (time slots for the demo will be reserved through Moodle). Note that **all** group members must be present for the demo and ready to answer questions about the code.

[1]See https://www.concordia.ca/ginacody/students/academic-services/expectation-of-originality.html