SOEN 6441 Advanced Programming Practices (Fall 2021)

Assignment #2

Due date (Moodle Submission): Tuesday, December 7

This is the second (and final) part of our course project, building on the results of your first assignment. Note that you must also fix any issues that came up with your first version for your new submission.

Play Application: "Reactive Gitterific" (Group Part: 50%). Your task is to modify your Play application from Assignment #1 to make it a *reactive server-push app*, i.e., stream live updates to the user interface. Thus, instead of having a static list of information for the search keywords, your application will now dynamically update, by showing new incoming information. When starting a new search, immediately populate the 10 latest results, then append any new incoming information matching the search keywords. *Note:* when searching repositories, "newest" refers to the `updated` sort option.

You **must** implement this reactive behavior as an asynchronous *server-push* solution, using WebSockets and Akka Actors. In particular, a solution where you simply refresh (parts of) a page using, e.g., AJAX on the client side will not be accepted. Depending on your solution, you might have to filter incoming information for duplicates (that is, make sure you do not show the same information multiple times in case there are no new information for a set of search keywords).

Individual Part (50%). The individual part is a continuation of your work from Assignment #1. You have to create a new *Akka Actor* to perform the same operation as defined in Assignment #1, together with appropriate message classes and unit tests (see the coding guidelines below for details). Note: here you can, but do not have to, define an additional supervisor for your actor (in the group part, you must have a supervisor actor).

Additional Coding guidelines. In addition to the coding guidelines from Assignment #1, your submission must satisfy the following new requirements:

Reactive Push. Additional guidelines for working with WebSockets and Akka:

1. You must create one actor instance for each user (WebSocket connection). Store the user's information, like the search history, in this user actor.
2. Handle any errors (e.g., from the GitHub API) in a supervisor actor, with a suitable failure handling strategy (e.g., restarting the child actors).
3. Create a separate actor class for each individual task (User Profile, Repository Profile, etc.)

Testing. Additional testing requirements:

1. For testing your mock GitHub API, you must use *dependency injection* (DI) with Google Guice. If you are already using a mocking framework in A1, like *Mockito*, this is also acceptable for A2.
2. You must add JUnit tests for your actors and ActorSystem, using the Java Akka Testkit shown in the lectures.

Submission. You must submit your code electronically on Moodle by the due date (late submission will incur a penalty, see Moodle for details). If your group has changed from A1, include a new, single, signed (by all team members) *Expectation of originality* form (see https://www.concordia.ca/ginacody/students/academic-services/expectation-of-originality.html) 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 group part of the code, as well as their individual work.