SOEN 6441 Advanced Programming Practices (Fall 2021)

Project Assignment #1

Due date #1 (Group Information): Monday, November 1
Due date #2 (System Submission): Wednesday, November 17

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 various 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 including both parts. Your own marks will be based on the group work (50%) and your individual contribution (50%), as detailed below.

Play Application: "Gitterific" (Group Part—50%). The overall goal is to develop a web application that can analyze the content on GitHub using its REST API.[1]

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 repositories matching the keyword(s) below the search field. Each match must include the name of the repository's author and (if available) topics for the repository. Note that you have to search for the whole phrase (and not each keyword individually).

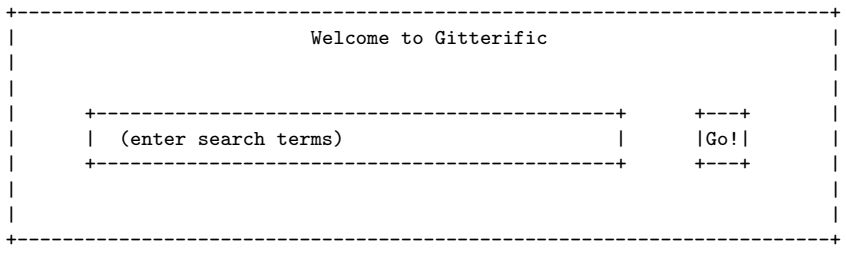

Figure 1: Gitterific page when first started

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

[1] GitHub REST API, https://docs.github.com/en/rest

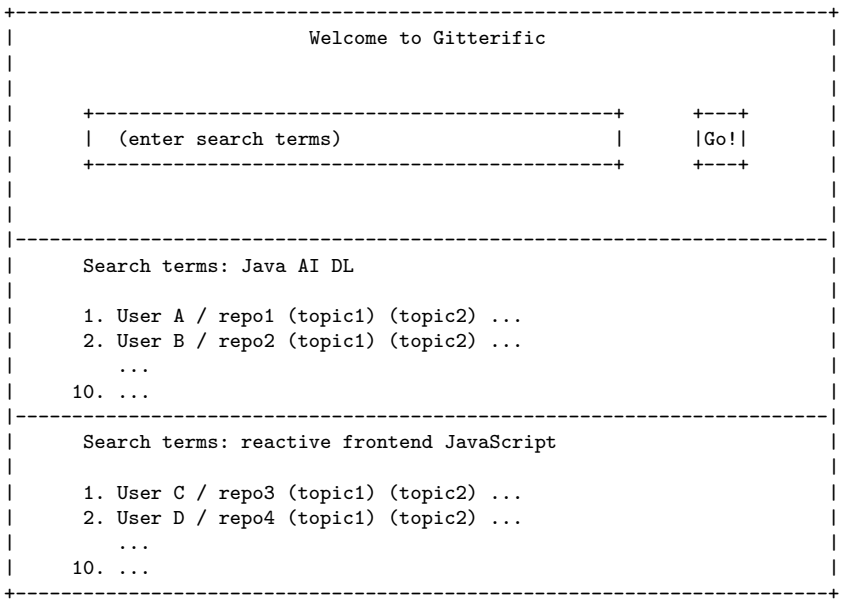

Figure 2: Gitterific page after two searches, showing two sets of results with 10 repositories each

You have to use the (free) GitHub REST API to access the information, see https://docs.github.com/en/rest.

Process the results using the Java 8+ Streams API to add hyperlinks to user profiles, repository locations, and topics (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 five members, only pick one individual task per team member (e.g., if your team has four members, only work on four tasks).

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

a) User Profile: Create a web page containing all available public profile information about a user, as well as all the other repositories of that user. This profile page must be hyperlinked from the user name from the results on the main search page. The repositories must be hyperlinked to their profile page from part b) below.

b) Repository Profile: Create a web page containing all available details for a repository. List the 20 latest issues of that repository with their information and hyperlink the issues statistics page from part c) below. Also hyperlink the names of the collaborators of the repository to their user profile page from part a) above and add a hyperlink to the commit page from part e) below.

c) Repository Issues: For a repository, fetch the available issues and compute a word-level statistics of the issue titles, counting all unique words in descending order (by frequency of the words). This

statistics page must be hyperlinked from the repository profile page from task b) above. You must use the Java 8 Streams API to process the issues.

d) Topics: For a topic[2] (hyperlinked from the results in the search results), display the 10 latest repositories containing this topic, in the same format as the results on the main search page.

e) Repository Commits: For a repository (hyperlinked from the repository page of task b) above), fetch the 100 newest commits and compile statistics about: (i) the top-10 committers, i.e., users who had the most commits, where you hyperlink the user to the profile page from task a) and show the total number of commits of that user as a number (n) next to the name; (ii) the minimum, maximum, and average number of additions and deletions across all these commits. You must use the Java 8 Streams API to compute the commits statistics.

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 an (open source) wrapping library for the GitHub API, but you have to wrap any synchronous APIs to make them asynchronous (see below). Also, all tasks above must use the same library in this case.
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 above.
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 results multiple times (e.g., for passing them between different tasks) – develop an appropriate caching strategy for your application.

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/Scaladoc 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.

[2]Examples for topics are information like "framework", "frontend", "blockchain" etc., see https://github.com/topics

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 a 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 GitHub API from a unit test, use a mock class for testing instead. For testing your classes using the GitHub 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 each of the individual parts 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 your 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[3] 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. Projects will not receive marks until they have been demoed.

Happy coding!

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