Software Design 236700 — Assignment 3

Teaching Assistant: Dor Brekhman <brekhman.d@campus.technion.ac.il>

June 13, 2022

1 General remarks

- For any questions that are not personal please use Piazza: `piazza.com/technion.ac.il/spring2022/236700`.
  If you want to e-mail the TA please include ”software design hw3” and your name in the subject of the e-mail.
  You are REQUIRED to follow the piazza for announcements.
- Please think of the environment and *do not* print out this document!
- This assignment is written in English. If you have any trouble understanding the intention, please contact the TA.
- We encourage you to read all relevant documents & files from the course websites before starting work on this assignment. Please also note the existence of an FAQ, I'll try to keep it updated.
- Submission is electronic via the course website.
- You *must* work in pairs. The number of team-members is two (2) for each team. A team with a single member is not enough, and three members are too much—four is right out. Two students for each submission, no more and no less.
- You are allowed to switch homework partners, simply correct `README.md` .
- The course staff are most familiar with IntelliJ, therefore will only able to provide technical support for that environment. However, we will still attempt to assist you if you choose to forgo an IDE and use only the command-line tools.
- Any updates and changes to the assignment will appear *in this document*, and will be tracked in section A on page 6. You should occasionally refresh this document from the course website, and be sure to go through it before your final submission. Also note the date shown at the top of this page.
- The purpose of this assignment is to use an API of a different pair and learn about considerations of API design.
- This

Contents

2 The new MessagingClient

You will implement a **new** system for messaging between users. This system is not related to SifriTaub but uses a library written in the previous assignment. You will not use your library, but a library of some other pair, that can be found in the course materials tab in the course website. The most used library will get bonus points.

2.1 Time limits & limitations

All the time limits discussed here are in regards to the time it takes the future to complete. A future will be returned immediately.

2.2 External library

The external library has the same properties as in previous assignments.

2.2.1 Test time limits

Make sure your code reads and writes to SecureStorage in linear time with respect to the written / read data of the high-level methods (when writing - consider the size of the parameters, and when reading consider the size of the output).

2.2.2 Concurrency

The tests will *not* be concurrent, i.e., the test will call a method, wait for it to complete (using the `CompletableFuture` ,) and only then call another method. Methods will not be called simultaneously from different threads. The provided sample tests are written to guarantee this. Note however that if the work continues after the returned `CompletableFuture` is ready, you may *still* have concurrency related bugs.

2.3 Dummy implementation

For your benefit, a dummy implementation of the persistent storage layer is provided. It acts the same as before. You are not required to mock the storage layer and are free to implement your tests however you wish.

2.4 Dependency injection and Guice

You must use Guice in this assignment, as before. Note that the new tests only use `MessagingClientModule` , and you will need to configure it to support all the needed dependencies.

2.5 Library

You are not allowed to use the library you've written in the previous assignments. Instead, you must use one of the libraries written by other students. You may fix any bugs you find, and add new features if you need them.
You can find all the other libraries in the course material.

Remember to note in your `README.md` , what library you chose (By noting the <id1>-<id2> pair associated with it). A small bonus will be given to the students who submitted the most-used library.

3 Dry part

You will review two APIs:

1. The SifriTaub API (all method signatures of the class + signatures of related classes, e.g Obtained-Loan).
2. The API of the library that you chose.

You are required to write at least a paragraph about each of them and include:

- What you liked any why.
- What would you change and why.

Use terms and ideas learnt in class about design generally and API design specifically.
You will get full grade if you wrote at least a paragraph, wrote what you liked and what would you change, and explained that using relevant terms from the course.

4 Tips & Tricks & Hints

1. This assignment in inherently asynchronous, and if you're not careful you'll accidentally operate in parallel, creating race condition bugs. Make sure you understand monads and use monad composition. Specifically, you should never let a `CompletableFuture` drop out of scope unless you've either composed it into another `CompletableFuture` or are somehow sure that it's completed.
2. Read the `CompletableFuture` documentation. There are a lot of useful things there. The Kotlin extension method library linked in the tutorial is also nice.
3. Parallelism is nice when reading from the storage since you can wait for things in parallel.
4. Your laptop might not have enough cores to exhibit failures due to race conditions. Try running on `csl3` .
5. Don't use `join()` or `get()` , except in tests.
6. Add a logger to your code. It's a very useful thing to have, and you can just turn off the messages before submitting.
7. Try to get Guice to do things for you by smart configuration instead of doing things manually.

5 Administrivia

5.1 Late submission

The late submission penalty is $n \cdot 5$, where n is the number of days after the submission date. Late submission closes after 5 days.
The exceptions to this rule are reserve duty (Miluim), and very exceptional circumstances to be decided by the TA. In any case, special extensions must be requested well in advance. Last minute extensions are unlikely to be granted, and extensions *after* the submission date will *not be granted* at all, even for reserve duty.

5.2 Your grade

You will be graded based on the following criteria:

1. Number of tests passed

2. Usage of the library you got from other students (Manual.)
3. Light code-review. If your design is deeply problematic, you will be penalized for that. However, if your design could be better, you will not be penalized, and you'll get feedback (Manual).
4. The existence and quality of your tests, and that they pass (Manual).

6 Submission Instructions

The project must be submitted as a ZIP file. We have provided automation for this purpose in the form of a GRADLE task that zips the entire project directory into a file named `submission.zip`. To use it, run the following command from the root directory of your project (the `base` directory that contains everything else) or from INTELLIJ in the prompt that appears after pressing "Ctrl" twice:

```
gradle submission
```

In addition to writing your code, you should fill out `README.md` with your names, ID numbers, and project documentation, and add your answers to the dry questions as a PDF file named `dry.pdf`. Everything should be in the project root directory, and subsequently in the `submission.zip` file.
You *must* make sure the submission file is ***less than 1MB***. Larger submissions will be silently ignored! Use GRADLE to decrease the size of your submission. Make sure you do not submit any log files, compiled binary files (e.g., .CLASS or .JAR files) or other spurious baggage. Look at the `exclude` rules in the `build.gradle.kts` to change what is not included with the submission.
Make sure to *test* your submission after creating the zip file!

6.1 Submission checklist

- Did you correctly merge your old code into the new skeleton?
- Did you add all bindings to the GUICE module?
- Did you use the provided external library, remembering not to use any other persistence methods?
- Did you write tests for everything? How's your coverage?
- Do the provided minimal tests pass (with an implementation of the external library)?
- Did you fill out `README.md`?
- Did you extract the submission zip to a different location and make sure it compiles, including running `gradle test`

If you answered yes to all, you're good to go! ☺

And most importantly, have fun!

A Changelog