CPS3222 Assignment

16th November 2017

Instructions

- This is a group (pairs) assignment. This assignment will be marked out of 100 and accounts for 60% of the final CPS3222 grade.
- Students will be marked individually based on their contribution to the assignment, as verified during an interview.
- While it is strongly recommended that you start working on the tasks as soon as related material is covered in class, the firm submission deadline is Wednesday 17th January 2018 at noon. Hard copies are to be handed in to Mr Kevin Cortis, the departmental secretary of the Department of Computer Science.
- You are to allocate approximately 40 to 50 hours of work to this assignment.
- A soft-copy of the report and all related files (including code) must be uploaded to the VLE by the indicated deadline. All files must be archived into a single .zip file. IT is the student's responsibility to ensure that the uploaded zip file and all contents are valid. Since this is a group assignment, all members of the group should upload the assignment.
- Reports (and code) that are difficult to follow due to low quality in writing-style/organization/presentations will be penalized.

Additional Instructions

- Work on the assignment is to be carried out by both people at the same time using pair programming on the same machine.
- Please organize yourself in pairs and send an e-mail to mark.micallef@um.edu.mt indicating the members of your group by Friday 24th November 2017.
- You will be required to produce a short report about the assignment and will be asked to defend your work in a 10 minute interview.

The country of Kanfalia is currently being plagued by a series of intelligence leaks with the most recent involving the leaking of a recipe of the country's world-famous Zondia dish. The uniqueness of this dish is credited with bringing millions of tourists to the country each year. The government is assembling a crack team of agents to locate and retrieve the recipe before it is leaked, something which could ruin the country's tourism sector.

Your team has been hired to support this mission by developing and testing a secure messaging system that will be used by the agents of the mission. These are the specifications, which you have been provided with:

1. Every time an agent wants to use the system, they must first contact their supervisor who will decide if it is safe for them to use the system at that point in time. If it is safe, then the supervisor will give them a special login key consisting of a 10-character string, which they must use to log into the system within 1 minute or the key will become invalid.
2. Once logged in, the agent can send and receive a maximum of 25 messages, at which point they will be logged out of the system and have to log in again.
3. Messages can only be a maximum of 140 characters long and should not contain any blocked words. Blocked words are words which should not be said between agents and for the scope of this assignment, a list of blocked words can be hard coded into your system as ["recipe", "ginger", "nuclear"]
4. Users of the system will also be logged out after 10 minutes of having logged in, regardless of whether they have sent/received their quota of messages.
5. When agents send a message to another agent, it is placed in a mailbox, which the receiver can access. Messages that are in the inbox for more than 30 minutes will be deleted.

Initial Architectural Overview

The system architect has created an initial design of the system's core business logic classes to help you get started. You are not obliged to adhere to this design and should feel free to change it in order to make things work. Figure 1 depicts a class diagram of the system whilst ***the Javadocs for the entities in the diagram have been uploaded to the VLE***.

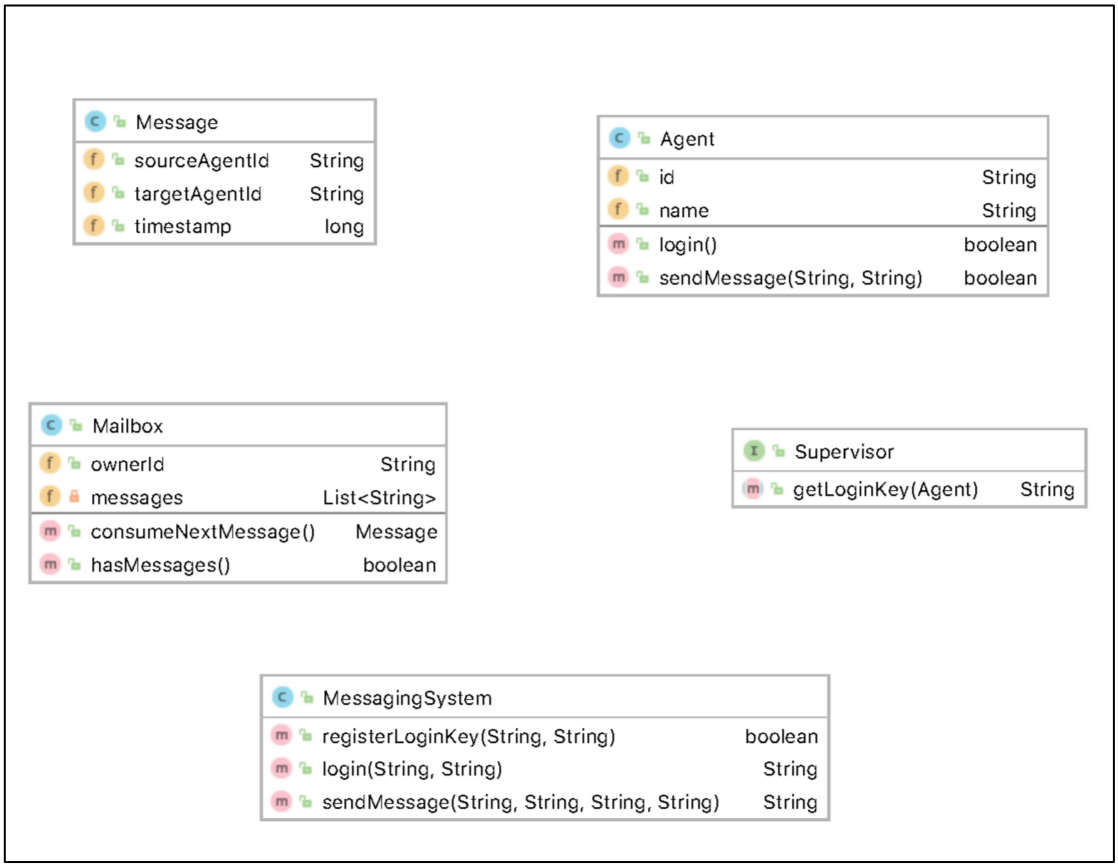

Figure 1 - Initial class diagram of the system.

Task 1: Unit Testing and Test Driven Development (28%)

You are required to build and unit-test the system outlined above using a test driven approach whilst ensuring maximum test coverage. Document your statement coverage analysis, any test patterns you use, your use of test doubles and any aspects of interest when it comes to coding for testability. Please note that at this stage, you are not required to produce any implementations of the `Supervisor` interface.

The design presented above is a first draft by the architect and may need adjustments in order to function and be fully testable. Feel free to make the design your own but please document any assumptions you make in your documentation.

Task 2: Cucumber and Automated Web Testing (28%)

Develop an appropriate test double for a `Supervisor`, which will enable you to set up an end-to-end functioning messaging system. The `Supervisor` you implement should always allow agents to log in unless their ID starts with "spy-". Implement a rudimentary web interface, which allows a user to log in and send and receive messages. Make any assumptions you deem necessary in order to make the system function. Please note that your web app will be assessed from a testability standpoint, so you should not waste time making the web application "look good".

Once you've developed your application, develop automated tests using Cucumber and Selenium/Webdriver to demonstrate that the following scenarios work.

Scenario 1: *Successful Login*

```
Given I am an agent trying to log in
When I obtain a key from the supervisor using a valid id
Then the supervisor should give me a valid key
When I log in using that key
Then I should be allowed to log in
```

Scenario 2: *Login timeout*

```
Given I am an agent trying to log in
When I obtain a key from the supervisor using a valid id
Then the supervisor should give me a valid key
When I wait for 65 seconds
And I log in using that key
Then I should not be allowed to log in
```

Scenario 3: *Surpassing message limit*

```
Given I am a logged in agent
When I attempt to send 25 messages
Then the messages should be successfully sent
When I try to send another message
Then the system will inform me that I have exceeded my quota
And I will be logged out
```

Scenario 4: *Blocked words*

```
Given I am a logged in agent
When I attempt to send the message <message> to another agent
Then the other agent should receive the message <new-message>

Examples:
|message                  | new-message |
|Hello there              | Hello there |
|Send recipe now          | Send now    |
|Nuclear recipe is ready | ready       |
|GinGer nuclear RECipE.   | .           |
```

Scenario 5: Logging out

```
Given I am a logged in agent
When I click on "Log out"
Then I should be logged out
```

Task 3: Model-Based Testing (24%)

Create and document a model of the web application which incorporates all your knowledge of how it is expected to function, including the rules provided in the introduction to the assignment. Without modifying your system, use this model to automate the generation and execution of test cases against the web application. Document any issues you find. Iteratively implement any missing functionality (if any) in your system, run your model-based tests again, and document. Repeat until you cover all transitions of your model and your system is able to pass a sequence of tests generated by your models without failing for 15 minutes.

Task 4: Performance Testing OR Mobile Testing (20%)

You are to choose ONE of these tasks and complete them as Task 4 of the assignment.

Option 1: Performance Testing

It is expected that the number of agents using the system will continue to grow. Document a performance testing plan that will address key functionality in your

system (you can propose which functionalities are important for testing). The plan should include details about what metrics will be collected and how they would be interpreted. Implement your performance test using either a custom built test driver in your language of choice, or a 3rd party tool such as JMeter.

Option 2: Mobile Browser Testing

Appium is a technology which uses the Selenium/Webdriver API but enables you to create automated tests for mobile applications or websites running in mobile browsers. Investigate the use of Appium and create a version of the tests from Task 3 which run on a mobile browser of your choice (IOS or Android). You can use simulators/emulators to demonstrate this working.