ECS713P Functional Programming

# Individual Coursework

## Haskell Social Network Simulation


School of Electronic Engineering and Computer Science
Queen Mary University of London

Name: **Sayed Sohail Pasha Peerzade**
Student Id**: 220541549**
MSc Computer Science

## Aim

The Haskell social network simulator is an application that simulates a social network. Ten simulated users are generated, each of whom can send random messages to other users at random intervals until a total of 100 messages have been sent. At the end, the program will display how many messages each user has received. Additionally this application also displays a user's inbox and outbox displaying all the messages sent or received by the user.

## Design

### User

| User | |
|---|---|
| user_id | Int |
| messages_received | Int |
| received_messages | [Message] |
| no_of_messages_sent | Int |
| sent_messages | [Message] |

The information related to a user is represented by User Data Type. Where a unique userId is stored as an Int, messages_received and no_of_messages_sent are used to store the count of messages sent or received. The received_messages and sent_messages are of type [Message] which stores the message contents in a list.

### Message

| Messages | |
|---|---|
| from | Int |
| to | Int |
| content | DataText.Text |
| time | String |
| date | String |

All information regarding a message sent between two users is stored in the message record of Message Data Type.  Both the "from" and "to" fields store the user_id of the specific user. The

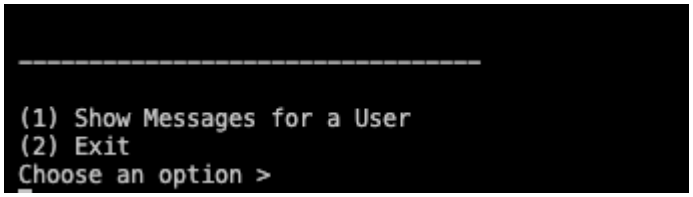
“content” field stores the random message generated. And the “time” and “date” fields store the date and time when the message was sent. Messages are stored inside the user records.

### Challenges faced

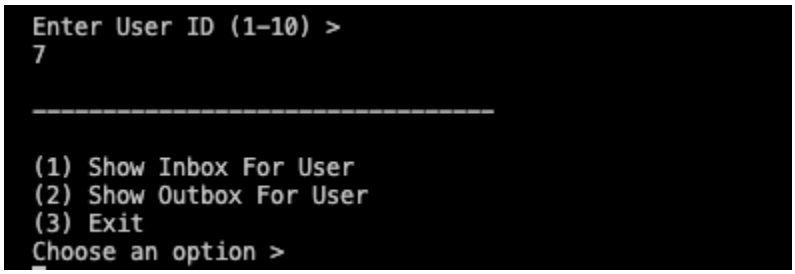
- Managing concurrent access to shared resources, this required the use of a mechanism to ensure that all threads can safely and consistently access shared resources. The “total” MVars was used to keep track of the messages sent and to stop sending messages once 100 messages limit was reached and “wait” MVar was used to notify the end of simulation
- Ensuring data consistency when multiple threads access and modify shared data, it becomes difficult to guarantee the integrity of the data. This can lead to race conditions, where two or more threads modify the same variable simultaneously and the resulting data is unpredictable.
- Memory Management: Handling memory utilization in Haskell can be difficult due to the fact that it is a memory-managed language, which requires each thread to control its own memory. When a lot of threads are running concurrently, the collective memory usage can easily become out of control.

### Extension of the given specification

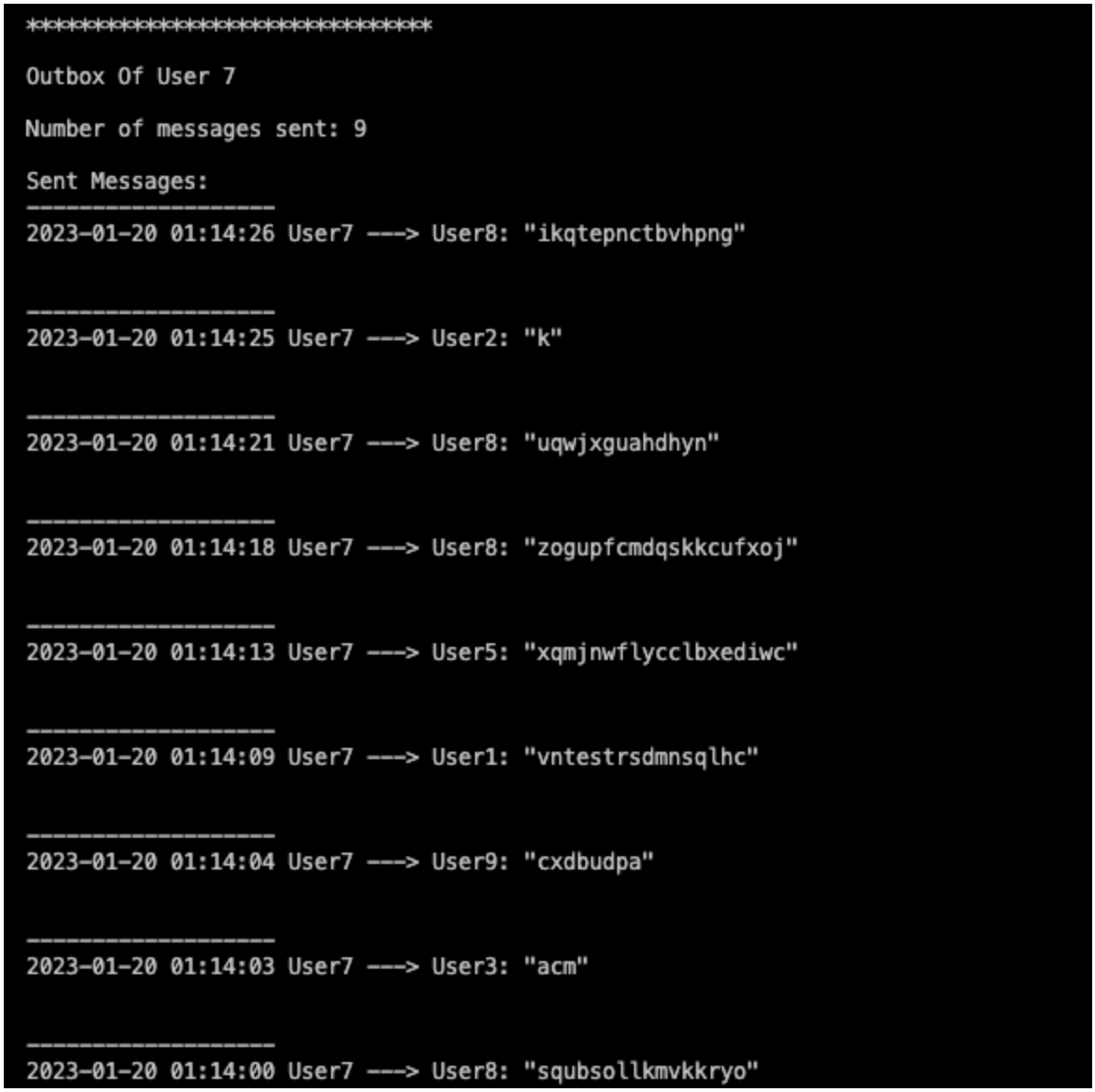
As an extension to the given specification, an option to display a user's inbox and outbox has been implemented.
After the simulation is completed the application prompts to show user’s message




On selecting the first option the application prompts to enter the user id which is a number between 1 - 10 (for 10 users).




On entering the user id the application prompts to either show the inbox or the out box of the user.




The application then displays the user's inbox or outbox with the total number of messages sent or received and also displays the messages.

### User Manual

- How to compile and run the code:
    - stack run
- How to generate haddock documentation:
    - stack build && cabal haddock --enable-documentation --open
- Haddock Doc Home file path:
    - dist-newstyle/build/aarch64-osx/ghc-9.4.4/social-network-simulation-0.1.0.0/doc/html/social-network-simulation
      **or**
    - /haddock-documentation