# CSE3310 Project

The Social Network

# Iteration II

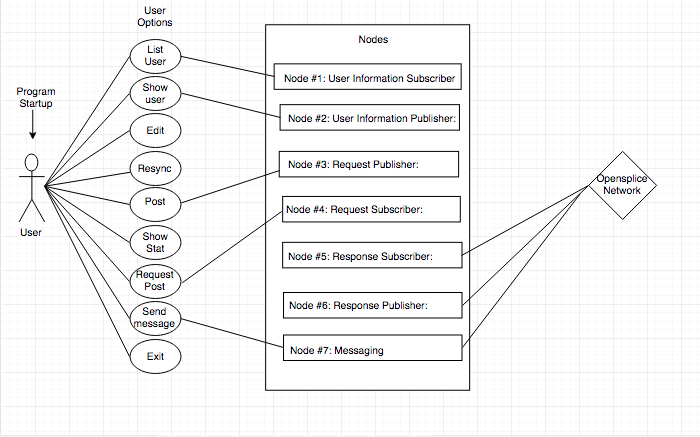
## Requirements and testing of all functional requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REQID | F/NF | Requirements | Action | Result | Status |
| 1 | F | The list user option shall list all the users in an orderly fashion. | List User option Entered | All users listed | Pass |
| 2 | F | The show users option shall show the specific user and the posts they have made. | Show User option Entered | User and posts displayed | Pass |
| 3 | F | The edit option shall let the user edit the information entered, by default it is saved to a disk file and read in on program start. | Edit option Entered | Able to edit user info | Pass |
| 4 | F | The resync option shall forget all saved data and delete all .tsn files. | Resync option Entered | Deleted all .tsn files | Pass |
| 5 | F | The post option should let user post their content. | Post option entered | User was able to post | Pass |
| 6 | F | On start the program should print user’s personal information which will be published not more than once per 30 seconds. | Program startup | On startup user personal info printed | Pass |
| 7 | F | After receiving request, data will be sent out 1 per minute, which contains vector of UUID and serial numbers. | Request Subscriber activated | Posts send out using responses. | Pass |
| 8 | F | The serial number of the post will be the unique identifier for the post. | Post is made | Received the serial number for the post | Pass |
| 9 | F | In order to make a request there must be nodes online with data you do not have. | Request | Received Post | Pass |
| 10 | F | The program shall ask for the user information only for the first time. | Program was run multiple times | Prompt for user info was made only once | Pass |
| 11 | F | A user must be able to post, exit, and rejoin at any time they want. | Exit was entered | Program was terminated | Pass |
| 12 | F | Posts are to be sent only when requested to save network traffic. | Request Post | Post Received | Pass |
| 13 | NF | User id must be of length 37 generated by Boost stored in a char array | UUID generated | UUID was of 37 characters | Pass |
| 14 | F | All post must be stored on local disk | Post was made | Post stored on local disk | Pass |
| 15 | F | The Stats option shall print out how many nodes are known and how much content is available in this node listed as a percentage | Stat option was entered | Stats displays total nodes and percentage | Pass |
| 16 | F | Local disk file must be encrypted | Not implemented | Not implemented | Fail |
| 17 | F | The menu will display the user commands. | Entered the menu | Menu displayed user commands | Pass |
| 18 | F | All serial numbers generated will start at 0 then generate new one every time post is made | Multiple post made | Serial number increased as post increased | Pass |
| 19 | F | UUID generated will be stored in a file. | UUID generated | UUID was stored in a file | Pass |
| 20 | NF | This program should be compiled and run in Linux OS. | Program tested on Linux OS | Program compiled and worked on a Linux OS | Pass |
| 21 | F | An application will only satisfy requests for data it owns. | Post requested from user | Only local data is sent from users.tsn | Pass |
| 22 | F | Program should support command to list all the users | List user option | Program displayed all users on the network | Pass |
| 23 | F | Program should support command to reset the sending of data. (resync) | Resync is executed | All .tsn files are deleted | Pass |
| 24 | NF | TSN will be coded in C++11 Language |  |  |  |
| 25 | NF | The IDL for the program must be IDL provided by the instructor. |  |  |  |
| 26 | F | Notification should be displayed if new post is received. | Post is requested | Received post is printed out | Pass |
| 27 |  | Notification should be displayed if new user is online. | A new user comes online | Notification is put in queue and displayed. | Pass |
| 28 | F | The posts will be matched based on the interests the user selects. | Not implemented |  | Fail |
| 29 | F | The program shall support Direct messaging. | Direct message option entered | Able to send and receive direct message | Pass |
| 30 | NF | The program should be built upon the functionality provided in iteration I. |  |  |  |
| 31 | NF | The user Interface shall be fun and user friendly to use. |  |  |  |
| 32 | F | The direct message shall be displayed as soon as it arrives. | Message sent | Message arrived as it was sent | Pass |
| 33 | F | The user shall be able to go back to menu options after getting out of one option. | Multiple options were entered | User was able to go back to menu | Pass |
| 34 | NF | Boost-Test shall be used for testing functions. |  |  |  |
| 35 | F | Unit test shall show pass/fail without user intervention. | Make test is run | All tests pass | Pass |
| 36 | F | New user\_information shall be pushed to User HashMap | User\_information subscriber is launched | HashMap is populated | Pass |
| 37 | F | If user\_information is not unique, new posts should be updated in the hashmap | New Post is made by a user already in the HashMap. | New post is appended | Pass |
| 38 | F | All subscriber objects shall be put in a queue and displayed only at the starting of the loop. | Start subscribers | Queued output is displayed | Pass |
| 39 | F | A request shall be made only when new information is available | - | - | - |
| 40 | NF | No compile time warnings should be displayed |  |  |  |
| 41 | NF | All user information shall be stored in text files with .tsn extension. |  |  |  |
| 42 | NF | All user input shall be taken with getline instead of std::cin |  |  |  |
| 43 | F | All subscribers should be launched by a single thread in the program | Program is started | All subscribers are active | Pass |
| 44 | F | The Statistics function shall not include our node in any metrics. | Stats function is started using menu (isolated) | Known nodes are 0 | Pass |
| 45 | F | A post will only have a parent field node if it has a parent. | Not implemented |  | Fail |

Tabular Use-Case for user Interface:

|  |  |
| --- | --- |
| **USER ACTION** | **SYSTEM RESPONSE** |
| List User | Lists all the user in orderly fashion |
| Show User | Show specific user and post they have made |
| Edit | Ability to edit personal information |
| Resync | Deletes all the user info and start from the beginning |
| Post | Gives ability to post |
| Show Statistics | Shows the number of times post is made |
| Send message | Send Private message to selected user |
| Exit | Exits out of the program |

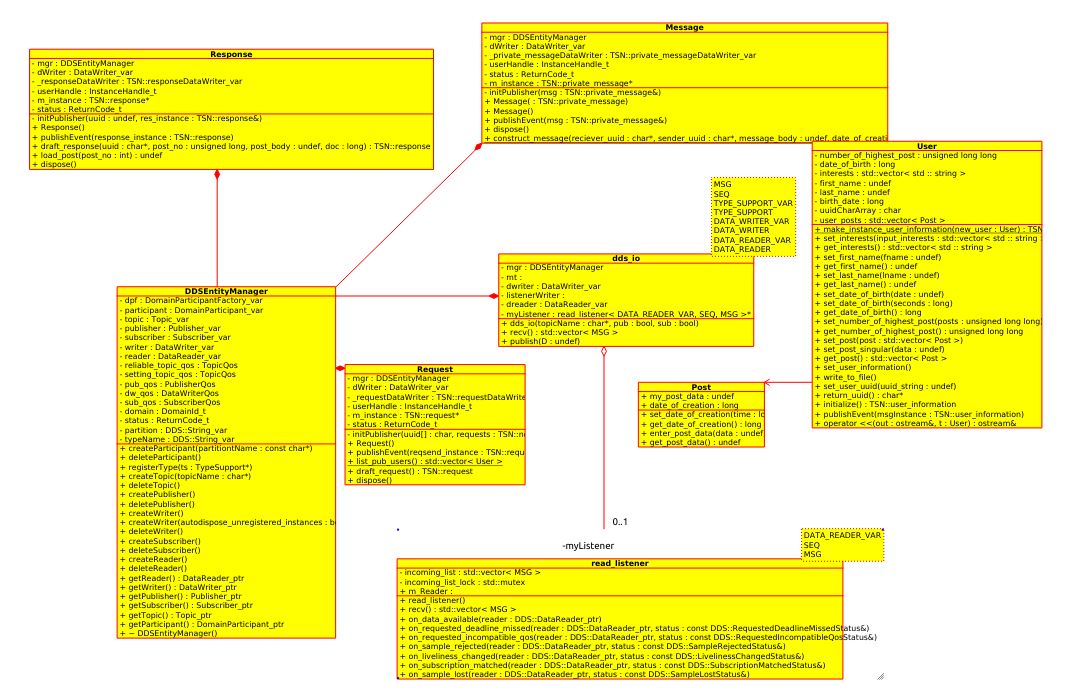
Use-case with stick figures:



## Design (4 points)

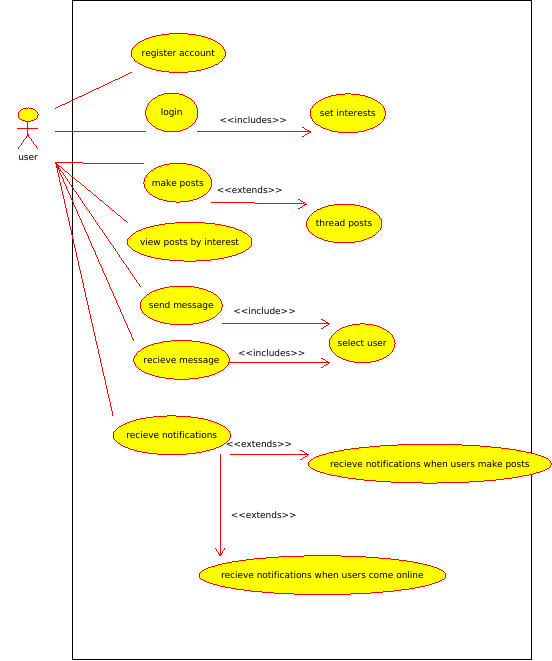
Provide:

* A class diagram.



XMI of the class diagram is included in the tar ball.

Use case diagram:



Overall description of the program and how it works (text).

TITLE: The Social Network

**Iteration II**

The second iteration of the project will add more functionality and easy user interface in the program. The two primary functionalities are user being able to send private message to another user on the network and getting notification if any user is online. This version of TSN is the modification on the previous version which is better and easy to use.

Basically there are 7 major nodes involved in sending and receiving information over the OpenDDS network. It is important to note that all of these nodes are derived from the types defined in the IDL provided by the instructor and using dds\_io classes. The description of the Nodes are given below.

Node #1: User Information Subscriber:

Responsible for listening for any user information published on the network and hence creating a user for them through the User class and write to file.

Node #2: User Information Publisher:

Launches upon program startup, publishes our user information every 30 seconds to the network. Automatically loads user\_info if the program is started after the first time.

Node #3: Request Publisher:

Publishes an object of the Request class to the network.

Node #4: Request Subscriber:

Looks for any requests received over the network. Initiates a response automatically if the request uuid is our user uuid.

Node #5: Response Subscriber:

If any responses are received the posts are extracted from it using response subscriber

Node #6: Response Publisher:

If there is a request and the uuid matches, a response is published with the requested post as data.

Node #7: Messaging

Responsible for publishing messages out to the network and receiving them.

All the subscriber nodes are launched using a thread upon startup of the program, so they are always listening.

Based on the UML diagram above, some important classes and the datatypes are explained below:

Based on the class diagram and nature of the program, Once the program executes in the command line interface and the user has never signed in before several prompts for personal information will show up, where the name of the users will be stored in the first\_name, last\_name variables. Similarly, the date of birth will be stored in birth\_date as long and the interest will be stored in a vector of string called interests which are inside the User class as shown in the UML diagram above. As soon as this information are received the user gets a ID which is stored in uuidCharArray and if any posts are made by the user the number of posts are saved in the variable number\_of\_highest \_post, and the post itself is stored in a vector of post called user\_posts which are saved in a file locally. In the same User class, we have several getter functions. Similarly, In the class Post, we have my\_post\_data to save the post initially before saving to a file, also the variable date\_of\_creation saves the time and date of the post made and lastly, we have some setter and getter functions in this class. To handle request and response we have made two classes one for each. The request class will handle publishing of object to the network and it also looks for any requests received over the network. Then, It will initiate a response automatically if the conditions are fulfilled. Accordingly, the response class will extract posts if any responses are received over the network. If there is a request and the UUID matches, a response is published with the requested post as data.

The message class is responsible for receiving and sending messages over the network. In this class we have functions to publish the message and receive the messages.

**Testing:**

The system testing of the project is given below:

Steps to compile and run the project:

1. Open command prompt and go to open splice directory, do source ./release.com
2. Go to the program directory and locate make file, do make and ./main
3. User interface will show up, and testing begins

We conducted the test in two computers both running on Linux OS and on same network.

After the compilation of the code on both the computers, we executed the code. The program prompted for personal information on both computers. After entering the personal information on both computers, we received UUID and user number for both users.

The Program was tested on two computers that fulfilled the above specifications and requirements:

|  |  |
| --- | --- |
| Computer1- | Computer2- |
| First Name- User  Last Name- One  Date of birth- 11/12/1995  Interests- Hockey, Dance | First Name- User  Last Name- Two  Date of birth- 12/22/1995  Interests- Study, Programming |

After the generation of UUID, the user options were tested by User One on Computer1

1. **List Users**-

Listed names of both users, and the UUID numbers.

1. **Show User**-

Displayed information about selected user along with the number of posts made.

1. **Edit-**

User one was able to edit the personal information entered on startup.

1. **Resync-**

Deleted all the .tsn files stored on computer1.

1. **Post-**

User one added a post “Hello everyone, this is a test” which was saved to local machine.

1. **Show statistics-**

Ran the code with it being the only node and it returned statistics with number of known nodes a 0.

1. **Request Post-**

User two requested a post from user one, User two received the post made by user one.

1. **Send message-**

User one selected user number two to send message, User one sent a message “Hello user two”.

User two received the message.

1. **Exit-**

User one exited out of the program and went offline.