PARALLEL AND CLOUD COMPUTING

REPORT

LAB ASSIGNMENT: 4

Student Name: Shijie Chen

Student ID: 11612028

Student E-mail: 11612028@mail.sustech.edu.cn

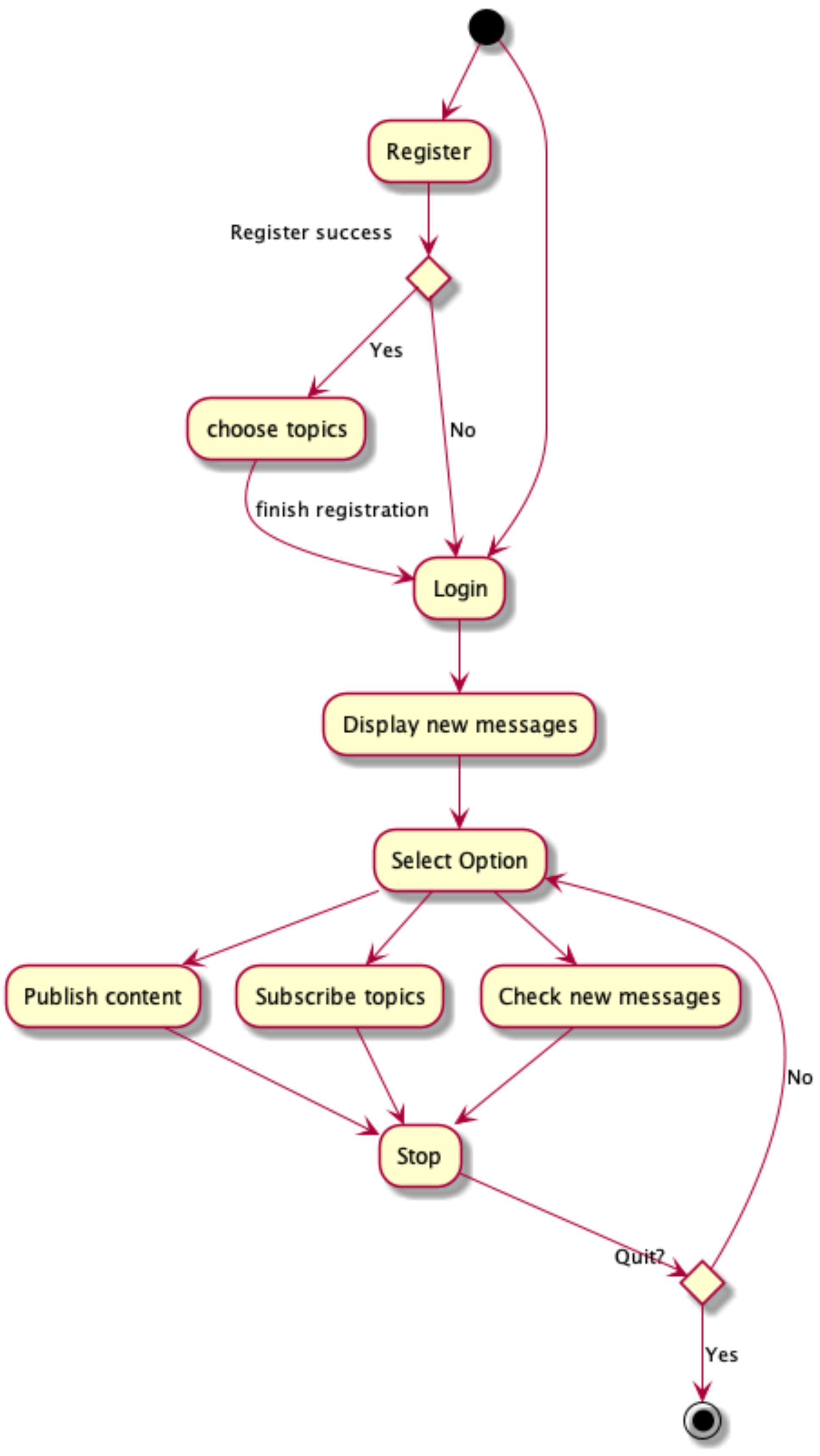
System Design

Module Design

Client Activities

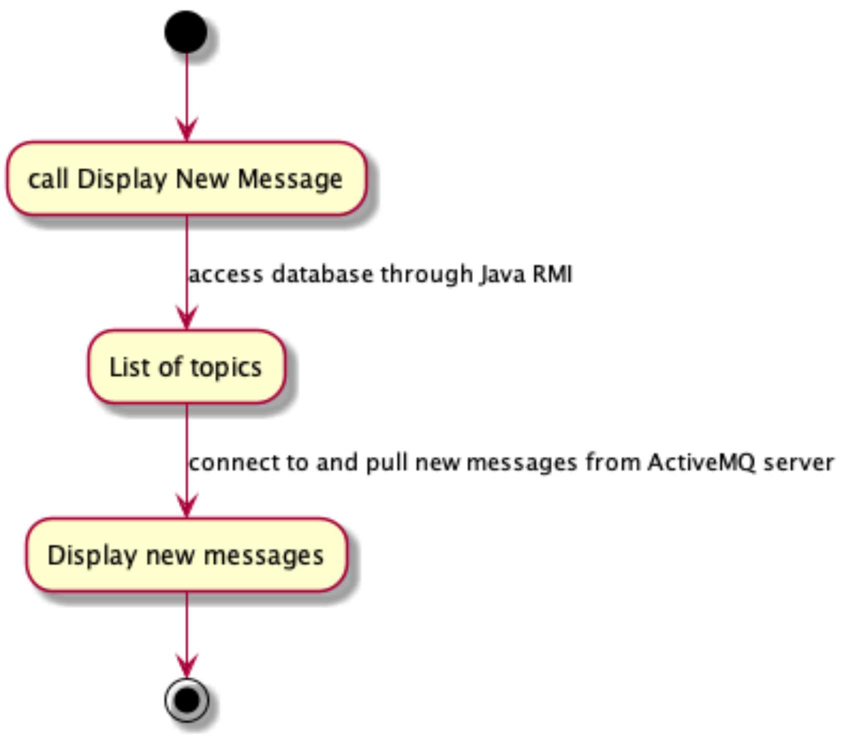
The following diagram demonstrates the client app workflow.

When subscribing or publishing content to a non-existing topic, the system will create the topic automatically. The subscription is **durable** subscription. The id of a user is automatically generated by the database of user information and is graranteed to be globally unique.



Display new messages

When we want to display all new messages for a user, the client will get the list of all subscribed topics from the app server through Java RMI. Then query and pull messages from the ActiveMQ server with the topics.

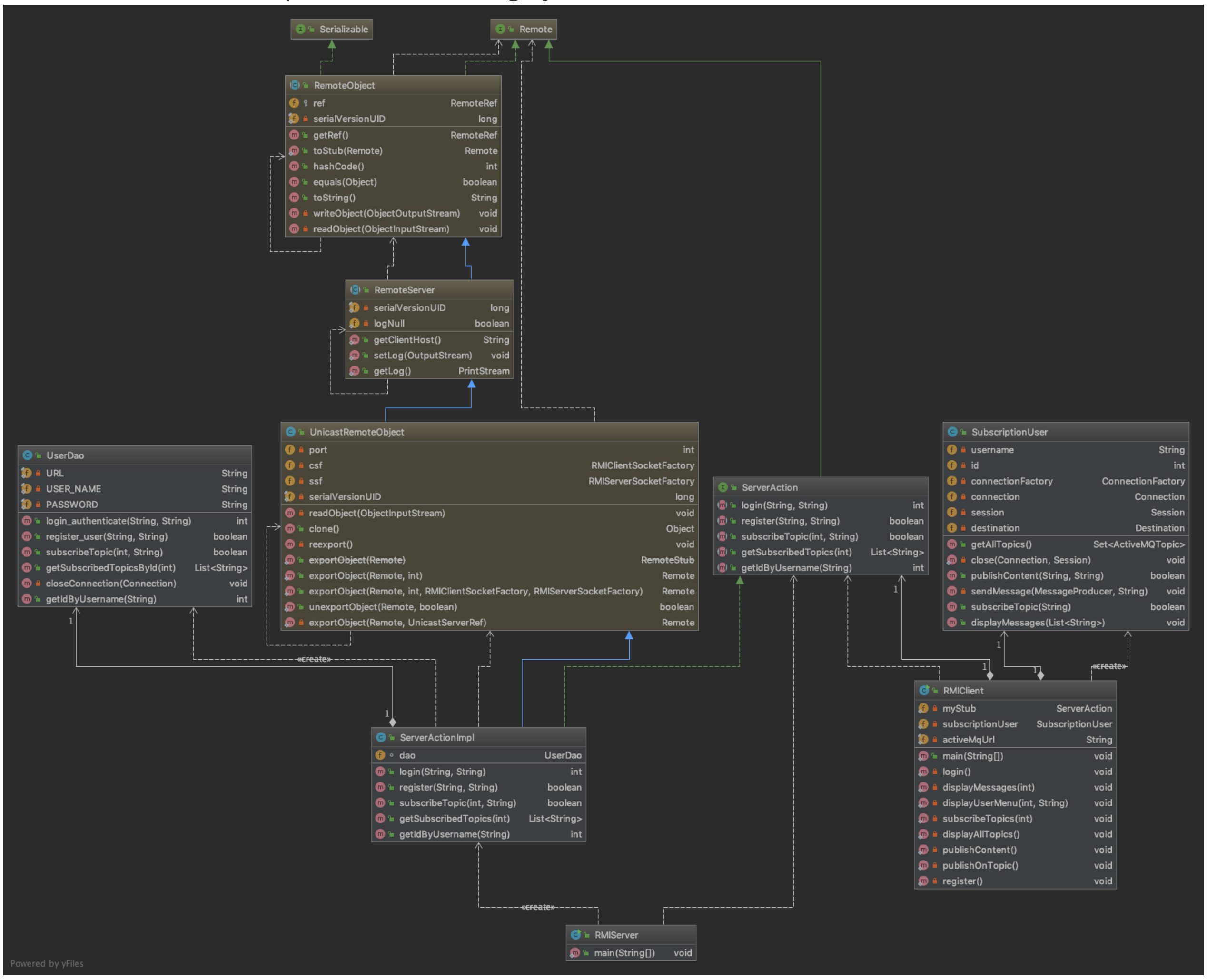

call Display New Message
access database through Java RMI
List of topics
connect to and pull new messages from ActiveMQ server
Display new messages

Class Design

The main difference between this lab's design and the former labs is the added `SubscriptionUser` class. This class packs the interaction between ActiveMQ Server and the client app.

Note that a user can be a subscriber and a publisher at the same time. Therefore I pack the methods for subscription and publication into a single class. A single connection to ActiveMQ server is created for each `SubscriptionUser` object but different sessions are used for different tasks.

The `ServerAction` interface, the `UserDao` class and `ServerImpl` class are also modified to include methods that get `userId` and `topics` (all the topics that a user subscribed) from the remote database. This part is done through java RMI.



Serializable
Remote
RemoteObject
ref RemoteRef
serialVersionUID long
getRef() RemoteRef
toStub(Remote) Remote
hashCode() int
equals(Object) boolean
toString() String
writeObject(ObjectOutputStream) void
readObject(ObjectInputStream) void
RemoteServer
serialVersionUID long
logNull boolean
getClientHost() String
setLog(OutputStream) void
getLog() PrintStream
UnicastRemoteObject
port int
csf RMIClientSocketFactory
ssf RMIServerSocketFactory
serialVersionUID long
readObject(ObjectInputStream) void
clone() Object
reexport() void
~~exportObject(Remote)~~ ~~RemoteStub~~
exportObject(Remote, int) Remote
exportObject(Remote, int, RMIClientSocketFactory, RMIServerSocketFactory) Remote
unexportObject(Remote, boolean) boolean
exportObject(Remote, UnicastServerRef) Remote
UserDao
URL String
USER_NAME String
PASSWORD String
login_authenticate(String, String) int
register_user(String, String) boolean
subscribeTopic(int, String) boolean
getSubscribedTopicsById(int) List<String>
closeConnection(Connection) void
getIdByUsername(String) int
ServerAction
login(String, String) int
register(String, String) boolean
subscribeTopic(int, String) boolean
getSubscribedTopics(int) List<String>
getIdByUsername(String) int
SubscriptionUser
username String
id int
connectionFactory ConnectionFactory
connection Connection
session Session
destination Destination
getAllTopics() Set<ActiveMQTopic>
close(Connection, Session) void
publishContent(String, String) boolean
sendMessage(MessageProducer, String) void
subscribeTopic(String) boolean
displayMessages(List<String>) void
«create»
1
ServerActionImpl
dao UserDao
login(String, String) int
register(String, String) boolean
subscribeTopic(int, String) boolean
getSubscribedTopics(int) List<String>
getIdByUsername(String) int
RMIClient
myStub ServerAction
subscriptionUser SubscriptionUser
activeMqUrl String
main(String[]) void
login() void
displayMessages(int) void
displayUserMenu(int, String) void
subscribeTopics(int) void
displayAllTopics() void
publishContent() void
publishOnTopic() void
register() void
RMIServer
main(String[]) void
Powered by yFiles

Major Design Decisions & Problems

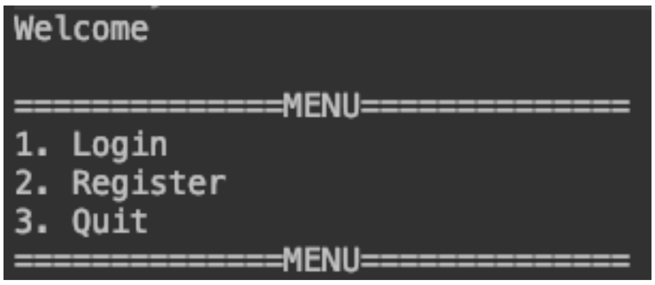
The major problem in this lab is the **implementation of one user subscribing multiple topics**. The user needs to provide a unique `user name` to ActiveMQ server. Otherwise previous subscription will be overriden.

To solve this problem, I set the `user name` to be `user_id_topic` where `id` is the id of the user and topic is the name of the subscribed topic.

Running Result

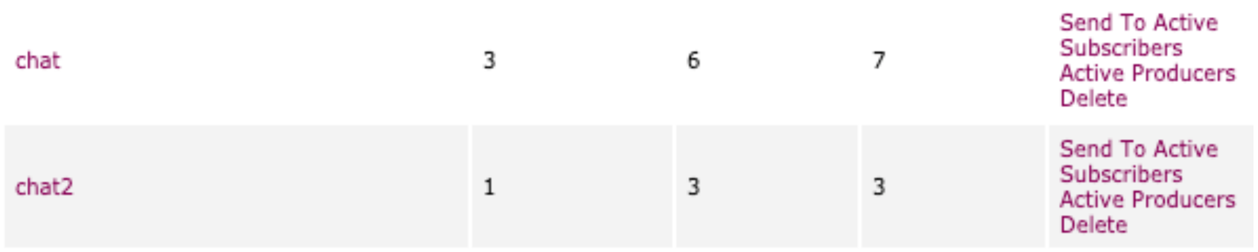
To run the code, please run `SimpleRegistryServer` followed by `RMIServer`. Then start the ActiveMQ service on your machine. Then `RMIClient` is ready to go.

Main UI

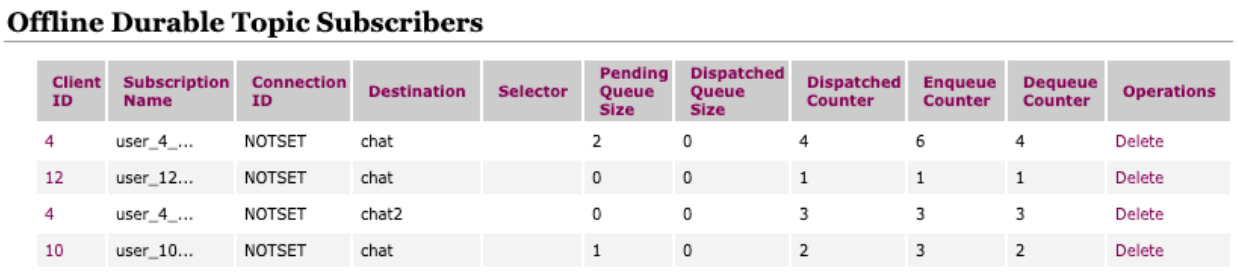

```
Welcome

==============MENU==============
1. Login
2. Register
3. Quit
==============MENU==============
```

Publishing and Subscribing

Existing Topics and Subscribers



chat	3	6	7	Send To Active Subscribers Active Producers Delete
chat2	1	3	3	Send To Active Subscribers Active Producers Delete



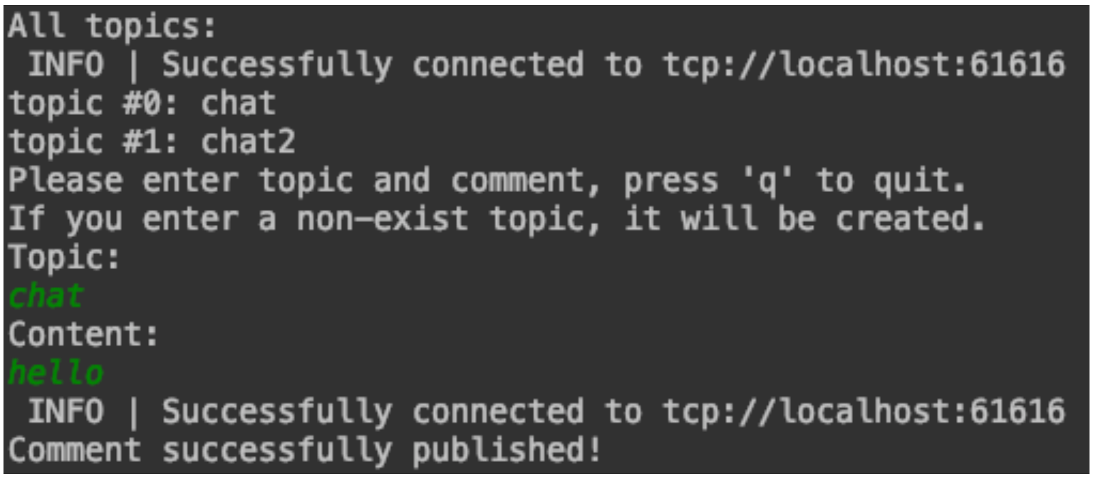
Offline Durable Topic Subscribers

Client ID	Subscription Name	Connection ID	Destination	Selector	Pending Queue Size	Dispatched Queue Size	Dispatched Counter	Enqueue Counter	Dequeue Counter	Operations
4	user_4_...	NOTSET	chat		2	0	4	6	4	Delete
12	user_12...	NOTSET	chat		0	0	1	1	1	Delete
4	user_4_...	NOTSET	chat2		0	0	3	3	3	Delete
10	user_10...	NOTSET	chat		1	0	2	3	2	Delete

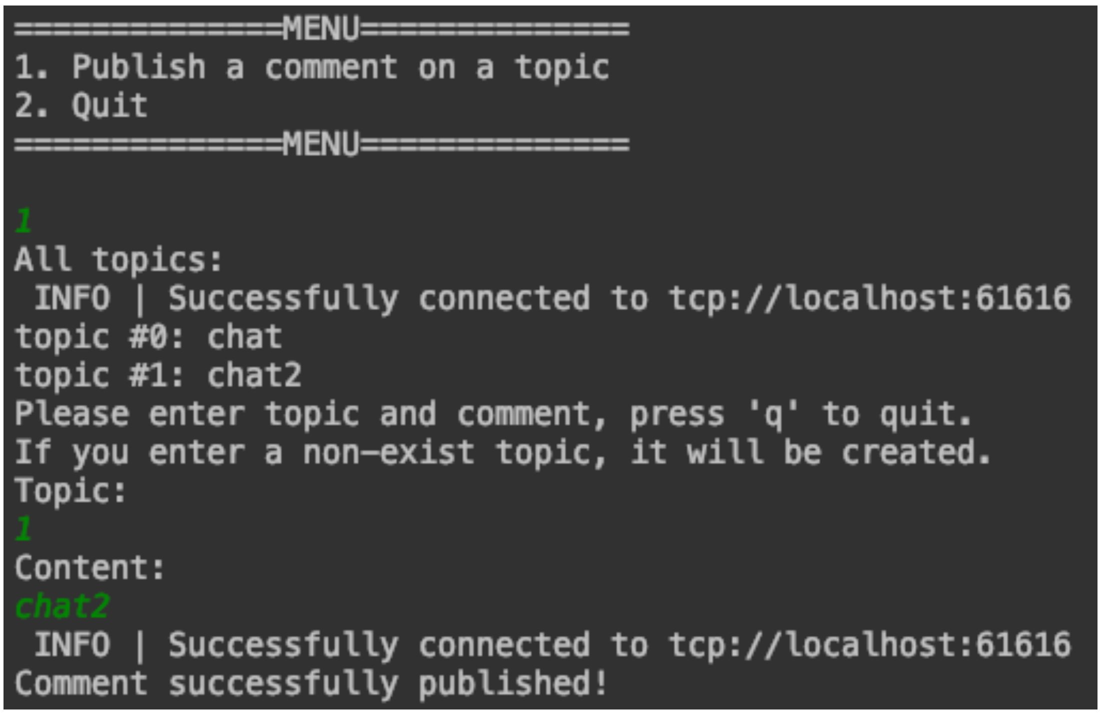
Here, the user `user` (with id user_4_...) has subsribed to both topic `chat` and `chat2`.

User Story

We login as user `noob1` and publish some content to these topics:

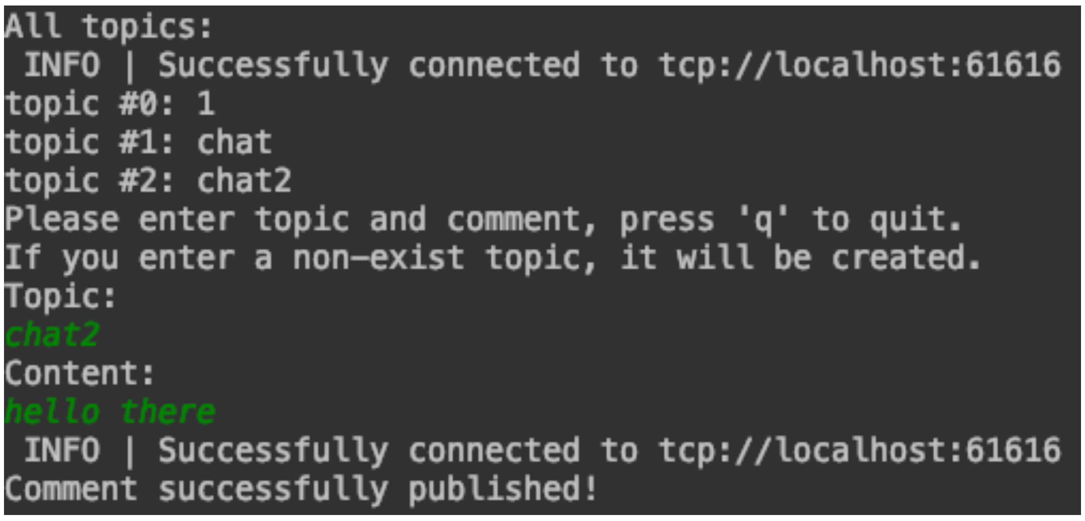

```
All topics:
 INFO | Successfully connected to tcp://localhost:61616
topic #0: chat
topic #1: chat2
Please enter topic and comment, press 'q' to quit.
If you enter a non-exist topic, it will be created.
Topic:
chat
Content:
hello
 INFO | Successfully connected to tcp://localhost:61616
Comment successfully published!
```

When publish content to a non-existing topic, that topic is created automatically. (This is accually an operation by mistake, as you may have noticed, but demonstrates this feature just fine :).)



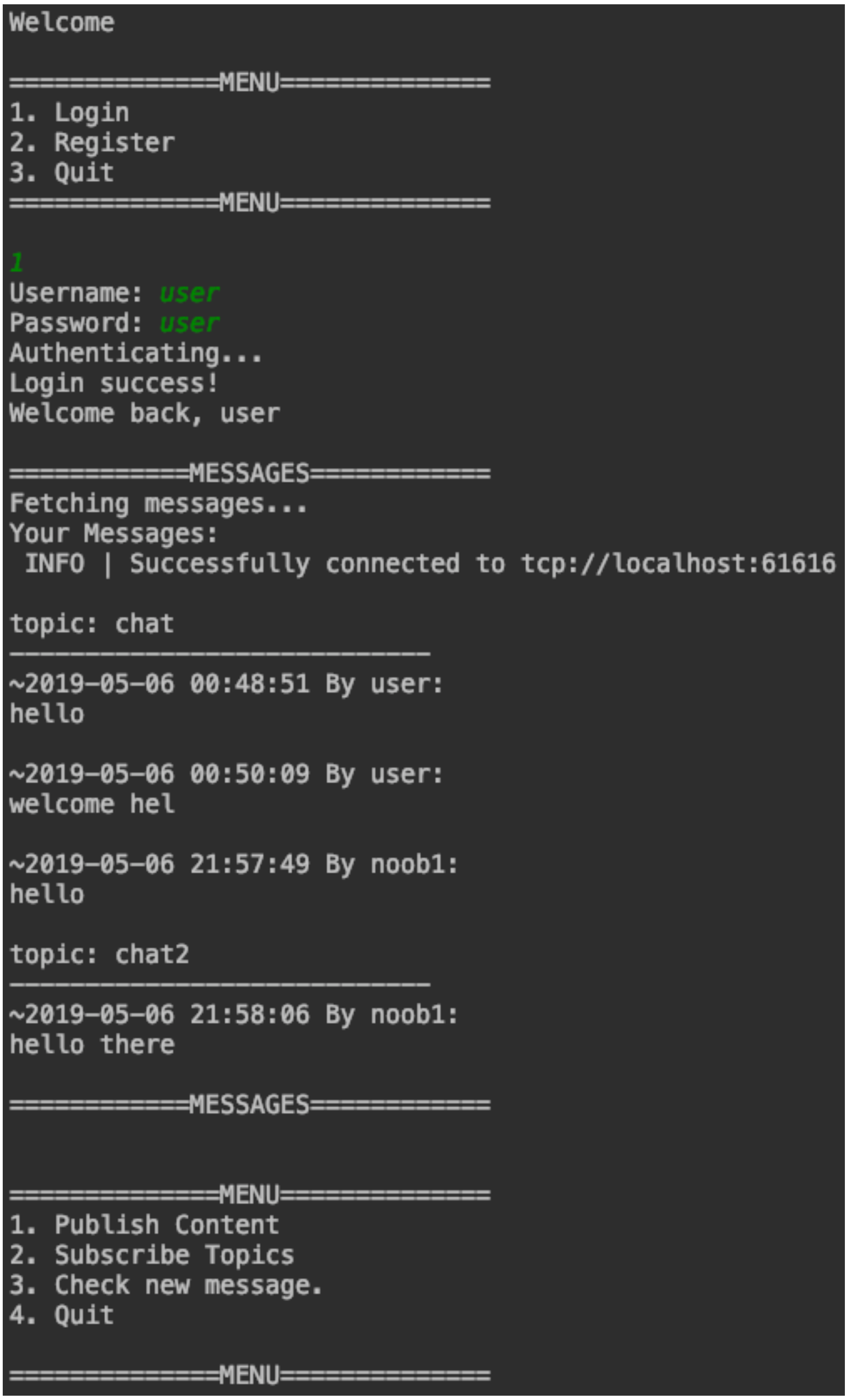
```
===============MENU===============
1. Publish a comment on a topic
2. Quit
===============MENU===============

1
All topics:
 INFO | Successfully connected to tcp://localhost:61616
topic #0: chat
topic #1: chat2
Please enter topic and comment, press 'q' to quit.
If you enter a non-exist topic, it will be created.
Topic:
1
Content:
chat2
 INFO | Successfully connected to tcp://localhost:61616
Comment successfully published!
```

We can see the new topic `1` right away when we try to publish another content.



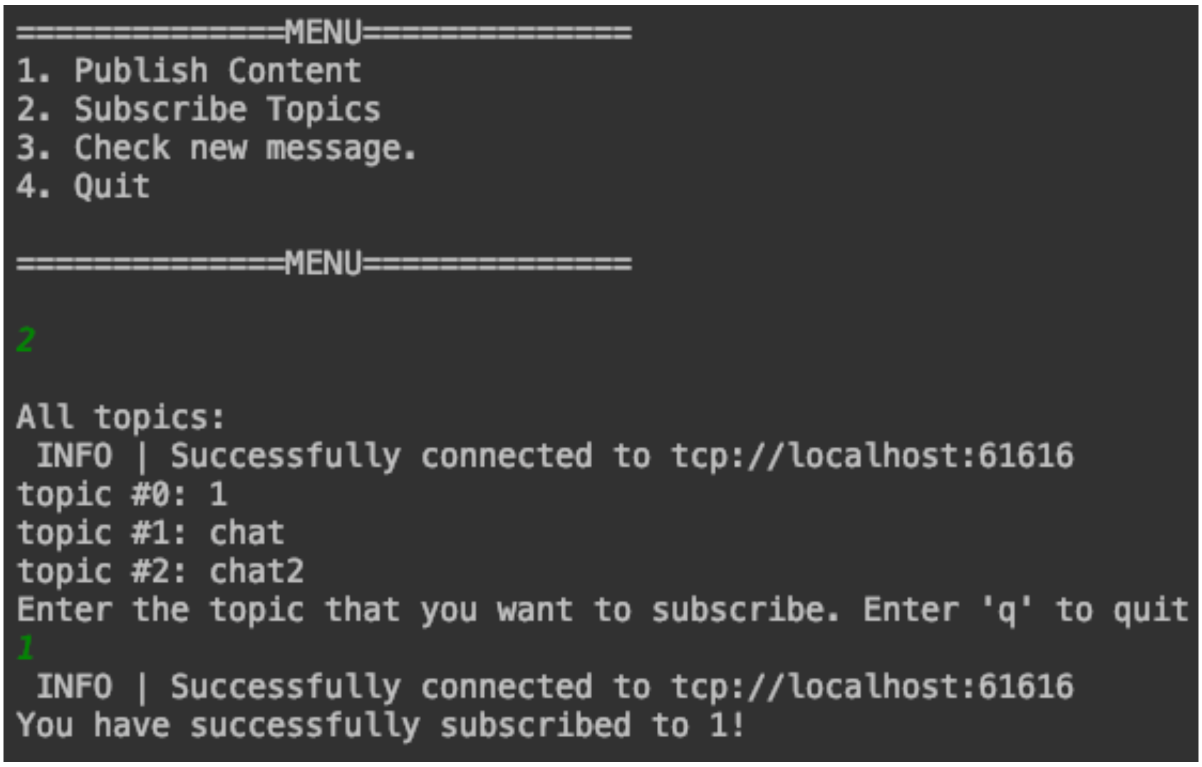
```
All topics:
 INFO | Successfully connected to tcp://localhost:61616
topic #0: 1
topic #1: chat
topic #2: chat2
Please enter topic and comment, press 'q' to quit.
If you enter a non-exist topic, it will be created.
Topic:
chat2
Content:
hello there
 INFO | Successfully connected to tcp://localhost:61616
Comment successfully published!
```

Then we login as `user`. After login successes, we revceive the messages from the subscribed topics including the two comments by `user` himself.



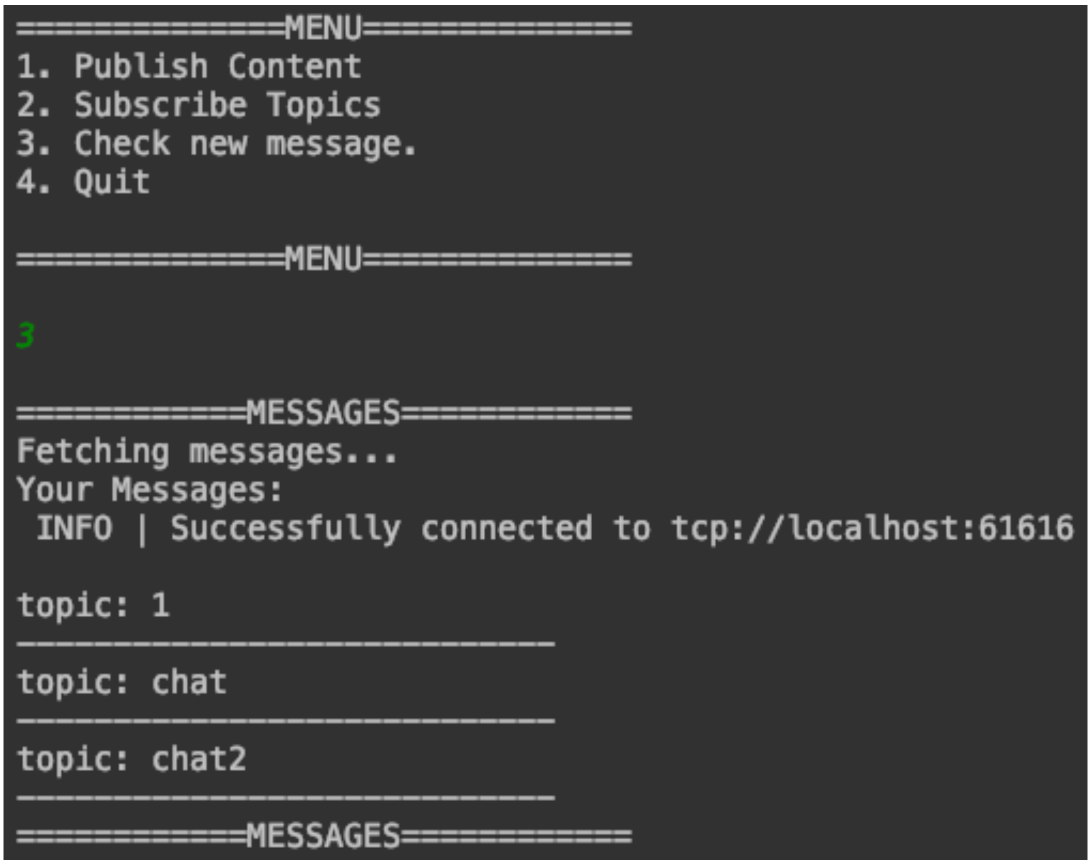
```
Welcome

==============MENU==============
1. Login
2. Register
3. Quit
==============MENU==============

1
Username: user
Password: user
Authenticating...
Login success!
Welcome back, user

============MESSAGES============
Fetching messages...
Your Messages:
 INFO | Successfully connected to tcp://localhost:61616

topic: chat
----------------------------
~2019-05-06 00:48:51 By user:
hello

~2019-05-06 00:50:09 By user:
welcome hel

~2019-05-06 21:57:49 By noob1:
hello

topic: chat2
----------------------------
~2019-05-06 21:58:06 By noob1:
hello there

============MESSAGES============


==============MENU==============
1. Publish Content
2. Subscribe Topics
3. Check new message.
4. Quit

==============MENU==============
```

Later we can subscribe to other topics. In this case, topic `1` is chosen.



```
==============MENU==============
1. Publish Content
2. Subscribe Topics
3. Check new message.
4. Quit

==============MENU==============

2

All topics:
 INFO | Successfully connected to tcp://localhost:61616
topic #0: 1
topic #1: chat
topic #2: chat2
Enter the topic that you want to subscribe. Enter 'q' to quit
1
 INFO | Successfully connected to tcp://localhost:61616
You have successfully subscribed to 1!
```

Now let's check if there's any new messages in the subscribed topics. Since no messages is sent to these topics, we should get empty from all three topics.



```
==============MENU==============
1. Publish Content
2. Subscribe Topics
3. Check new message.
4. Quit

==============MENU==============

3

============MESSAGES============
Fetching messages...
Your Messages:
 INFO | Successfully connected to tcp://localhost:61616

topic: 1
----------------------------
topic: chat
----------------------------
topic: chat2
----------------------------
============MESSAGES============
```