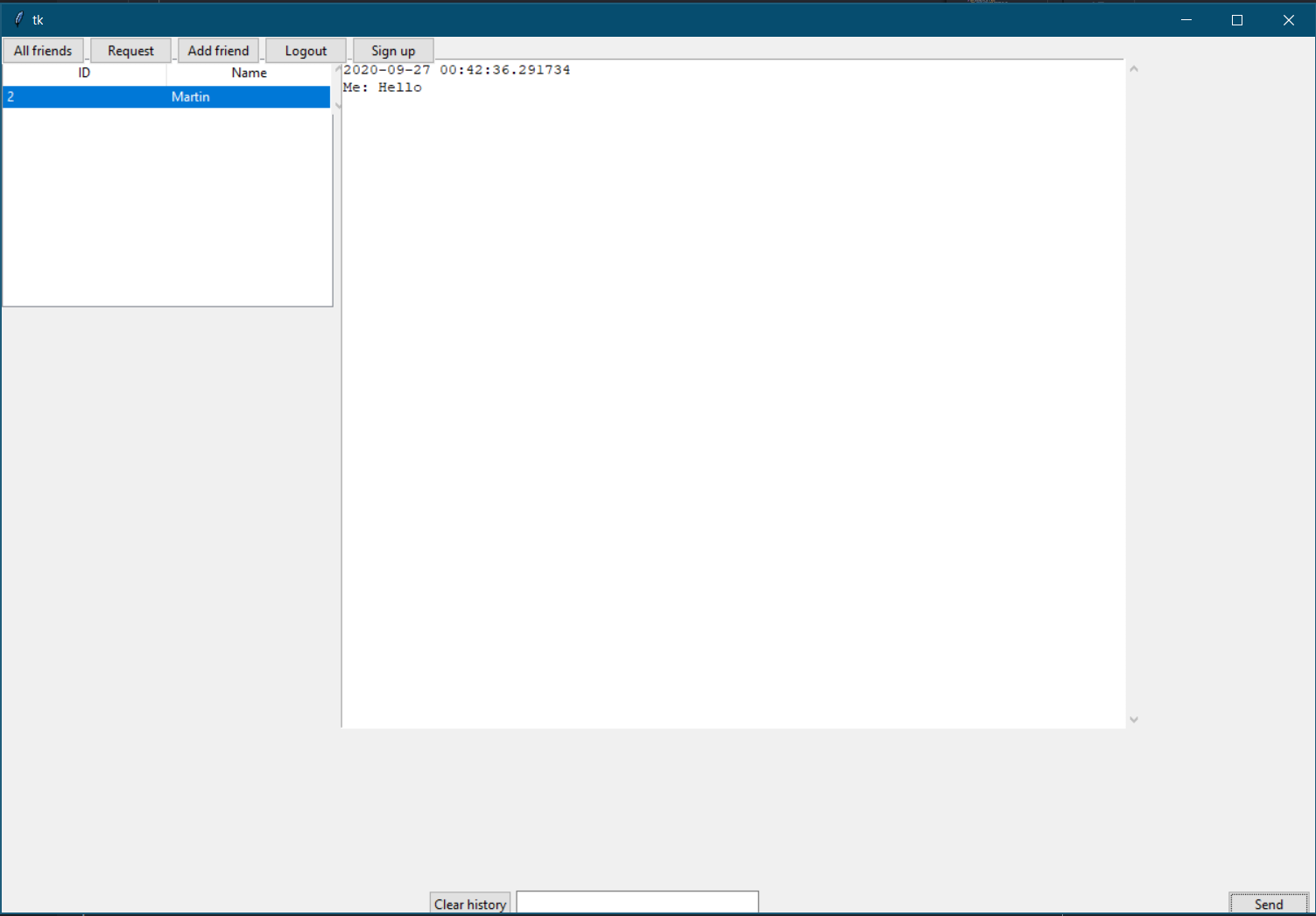
CS MINI IA

Author: DP2-1 Eric Jin

Date: September.26

Subject: CS HL



Contents

[Problem description 3](#_Toc52059840)

[Solution description 4](#_Toc52059841)

[Requirements 5](#_Toc52059842)

[Rationale 6](#_Toc52059843)

[Tools 7](#_Toc52059844)

[Design 8](#_Toc52059845)

[Communication design 8](#_Toc52059846)

[Database design 9](#_Toc52059847)

[Server end 9](#_Toc52059848)

[Client end 10](#_Toc52059849)

[UI design 11](#_Toc52059850)

[Top down design 12](#_Toc52059851)

[Logic Flowcharts 14](#_Toc52059852)

[Server end 14](#_Toc52059853)

[Client end 26](#_Toc52059854)

[Implementation 29](#_Toc52059855)

[Testing 48](#_Toc52059856)

[Reference 57](#_Toc52059857)

# Problem description

In this project, I designed a multi user online message software.

# Solution description

# Requirements

* The user can use the program with GUI interface
* The user can sign up with a username and password and get a random ID
* The user can sign in with the username and password
* After sign in
  + The user can change password
  + The user can request to add friends with friend’s ID
  + The user can choose to accept or refuse friend requests
  + The user can send and receive text message to friends
  + When messages are sent to off-line users, they will be stored on cloud, and when the receiver get online, he or she can get message to local. Then the message will be stored in local and removed in cloud.
  + The user can choose to remove message history from local database.

# Rationale

# Tools

* Develop language: Python3
* GUI Lib: Tkinter
* Database: sqlite3
* Communicate protocol: TCP

# Design

## Communication design

Each time client send a message to server in dictionary.

It have key “mode”, server end respond according to “mode” key’ value.

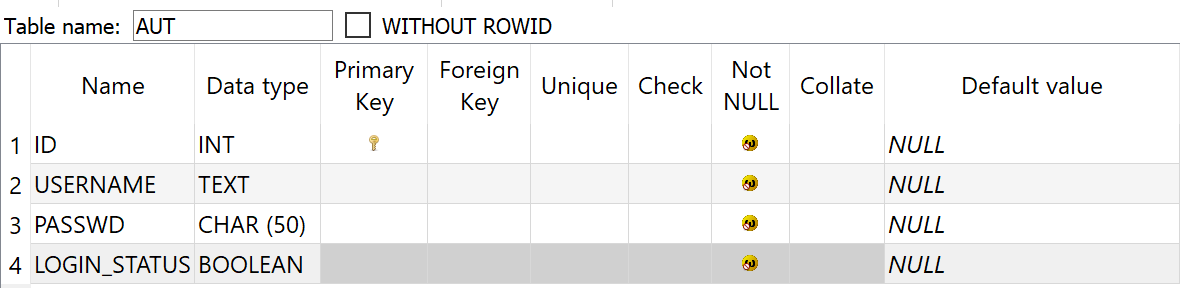
It have other keys and values needed for the mode

|  |  |  |
| --- | --- | --- |
| Mode | Keys | Server respond |
| “sign\_up” | “name”, “pwd” | Success: auto-assigned user\_id  Fail: 0 |
| “sign\_in” | “id”, “pwd” | Success: user\_name  Fail: 0 |
| “sign\_out” | None | Success: 1  Fail: 0 |
| “add\_friend” | “friend\_id”, “req\_note” | Success: 1  Fail: 0 |
| “del\_friend” | “friend\_id” | Success: 1  Fail: 0 |
| “accept\_friend” | “friend\_id” | Success: 1  Fail: 0 |
| “refuse\_friend” | “friend\_id” | Success: 1  Fail: 0 |
| “send\_msg” | “friend\_id”, “msg” | Success: 1  Fail: 0 |
| “refresh” | None | Success: {msg: [[sender\_id, time, msg], [], …], req: [], fl} |

## Database design

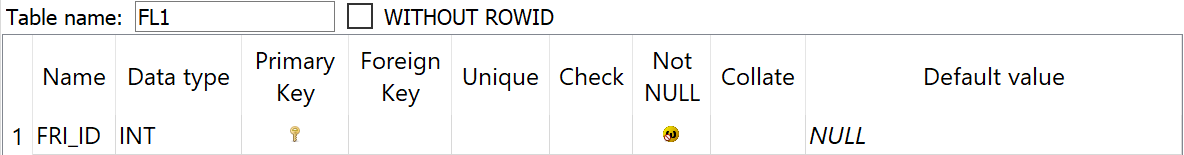
### Server end

One **AUT** table store the all users’ information

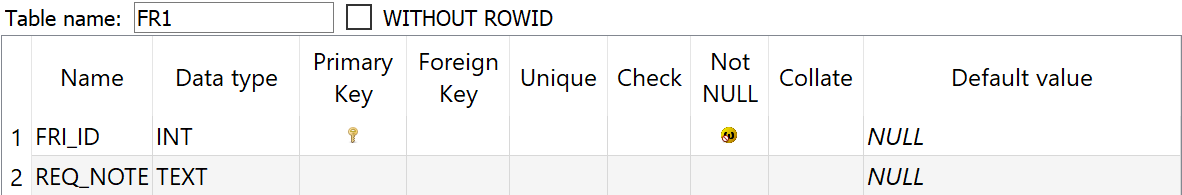


One **FL<ID>** table for each user store friends’ ID

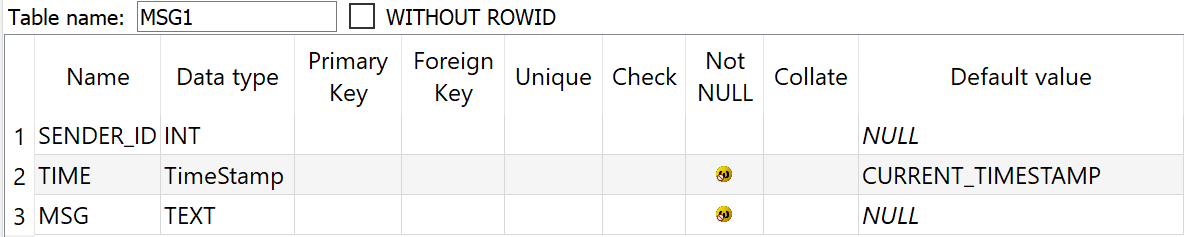
\*FL<ID> , for example ID is 114, the table name is FL114



One **FR<ID>** table for each user store friend requests info

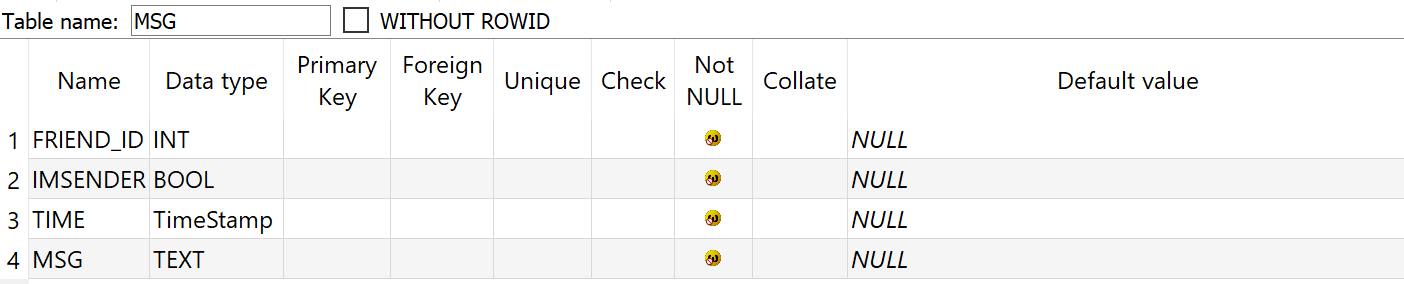


One **MSG<ID>** table for each user store unreceived messages



### Client end

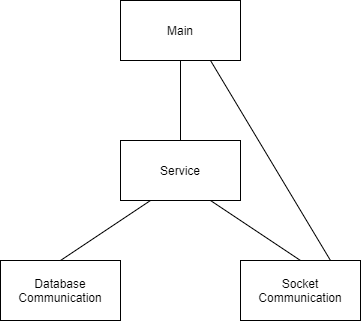
In client end, each user has one database. Each database has one table



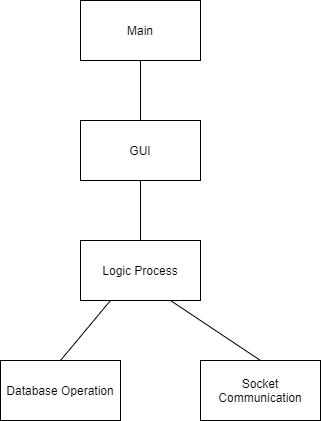
## UI design

## Top down design

### Server end



### Client end



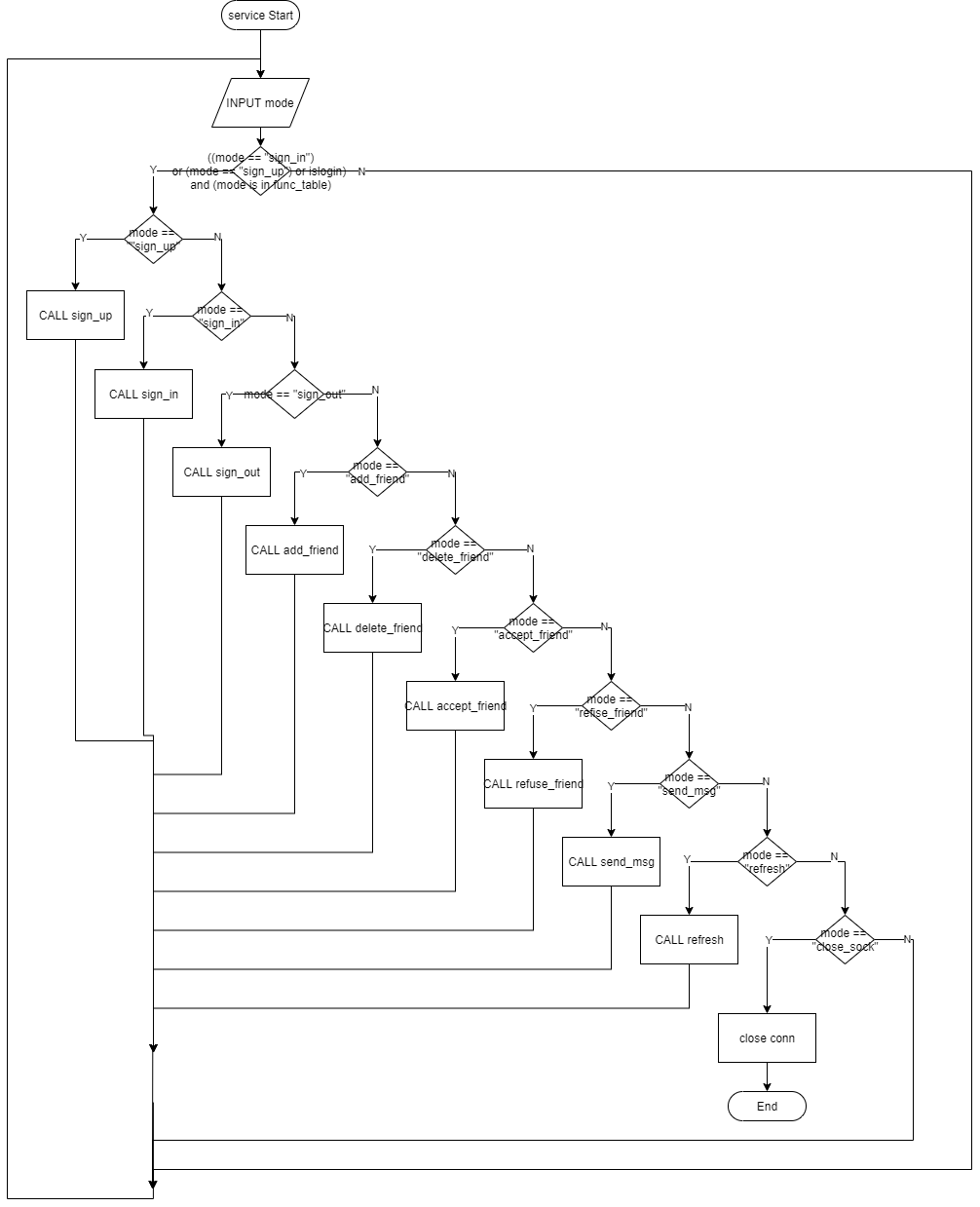
## Logic Flowcharts

### Server end

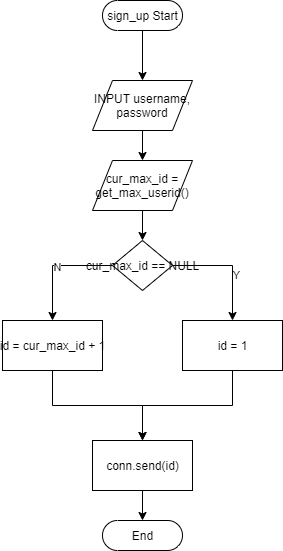
#### Main



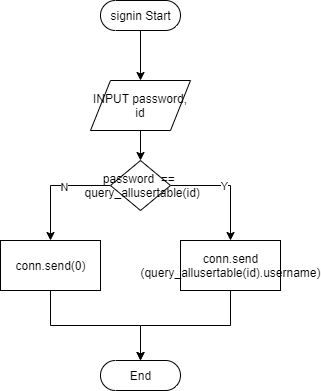
#### Service



#### Sign up



#### Sign in

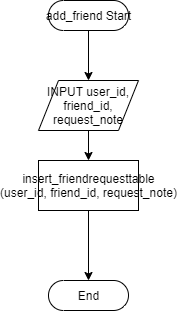
.password

Login\_status, xxx, modify

#### Sign out



#### Add friend



#### Delete friend



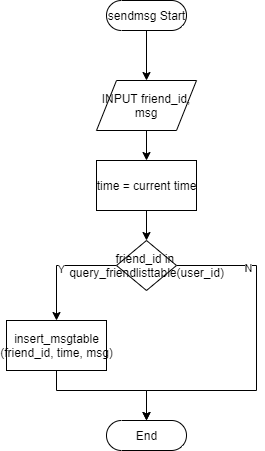
#### Accept friend



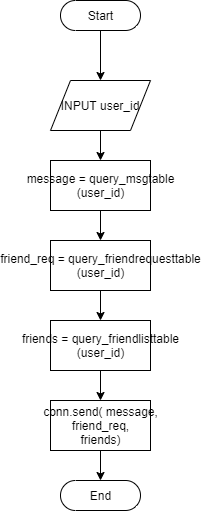
#### Refuse friend



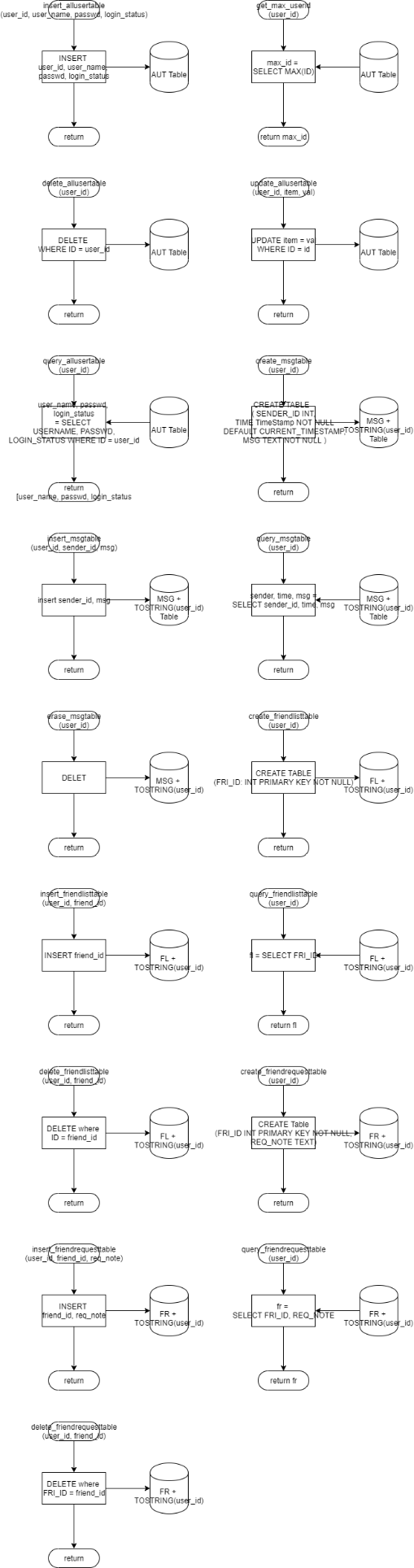
#### Send msg



#### Refresh

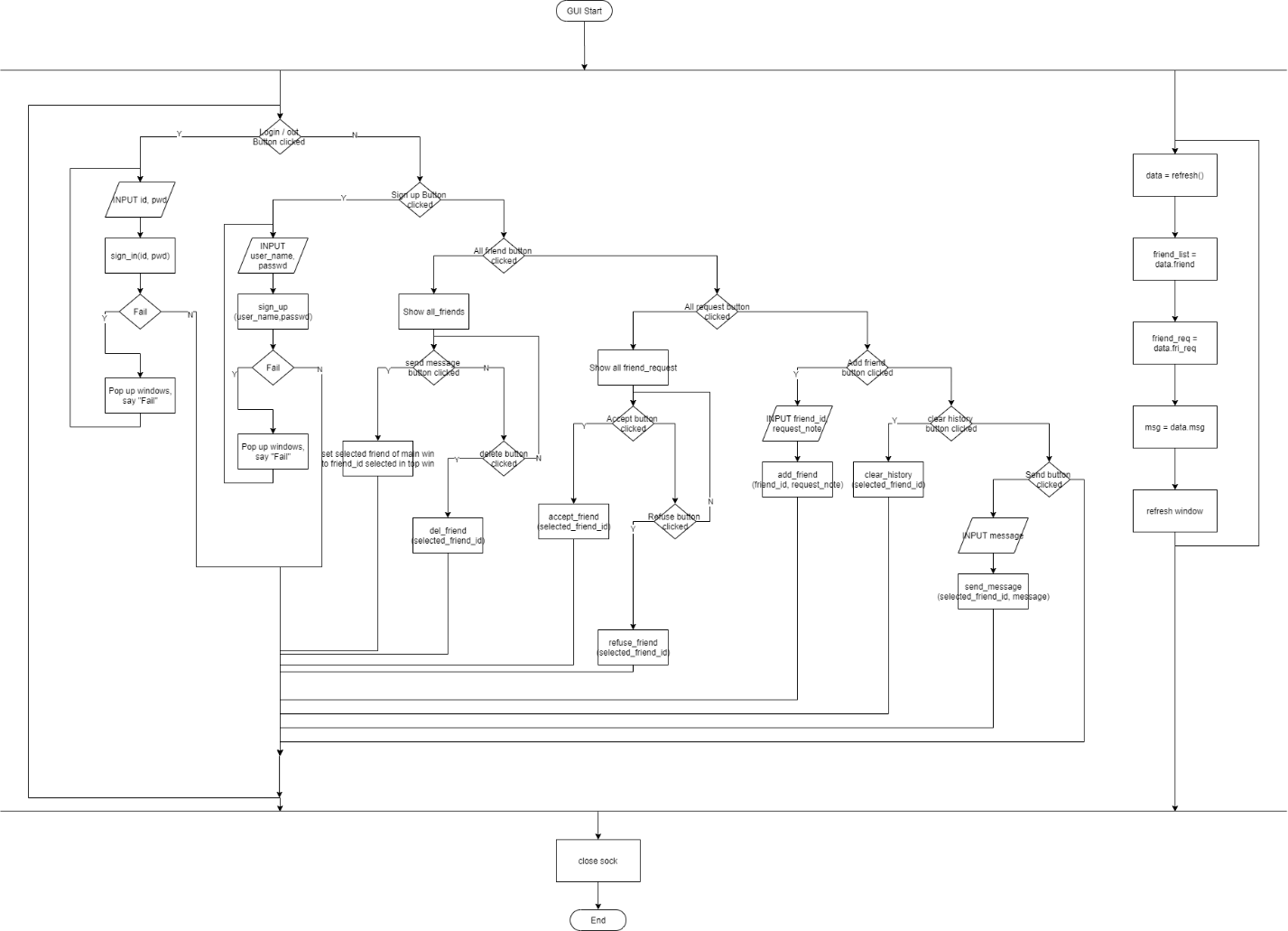


#### Database operation

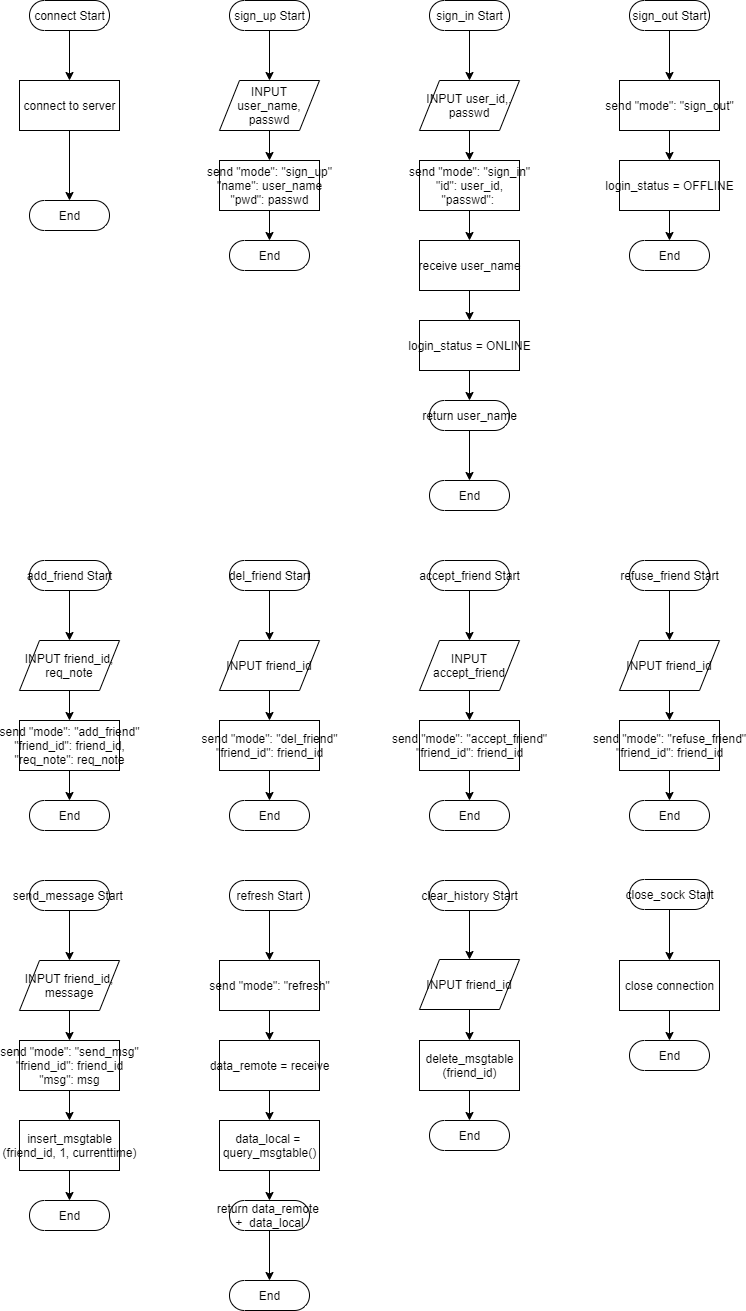


### Client end

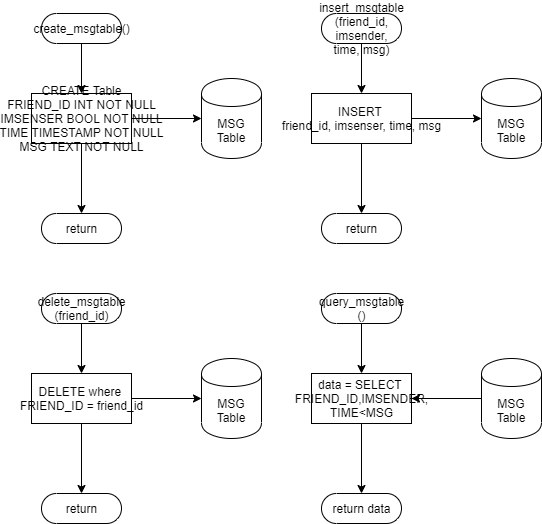
#### User Interface



#### Logic Process



#### Database operation



## Pseudocode

### Server end

#### Main

#### Service

PROC service

DECLARE mode: STRING

CONSTANT FUNC\_TABLE <- {

"sign\_up", "sign\_in", "sign\_out",

"add\_friend", "delete\_friend", "accept\_friend", "refuse\_friend",

"send\_msg", "refuse\_friend", "close\_sock"

}

WHILE TRUE

mode <- INPUT

IF ((mode == "sign\_in") or (mode == "sign\_up") \

or (isLogIn)) and (mode in FUNC\_TABLE) THEN

IF mode == "sign\_up" THEN

CALL sign\_up

ELIF mode == "sign\_in" THEN

CALL sign\_in

ELIF mode == "sign\_out" THEN

CALL sign\_out

ELIF mode == "add\_friend" THEN

CALL add\_friend

ELIF mode == "delete\_friend" THEN

CALL delete\_friend

ELIF mode == "accept\_friend" THEN

CALL accept\_friend

ELIF mode == "refuse\_friend" THEN

CALL refuse\_friend

ELIF mode == "send\_msg" THEN

CALL send\_msg

ELIF mode == "refresh" THEN

CALL refuse\_friend

ELIF mode == "close\_sock" THEN

CLOSE CONNECTION

BREAK

ENDIF

ENDIF

ENDPROC

#### Sign up

PROC sign\_up

DECLARE user\_name: STRING

DECLARE password: STRING

DECLARE cur\_max\_id: INT

DECLARE id: INT

user\_name <- INPUT

password <- INPUT

cur\_max\_id <- get\_max\_userid()

IF cur\_max\_id == NULL THEN

id <- 1

ELSE

id <- cur\_max\_id + 1

ENIF

RETURN id

ENDPROC

#### Sign in

PROC sign\_in

DECLARE password: STRING

DECLARE id: INT

password <- INPUT

id <- INPUT

IF password == query\_allusertable(id).password THEN

SEND (query\_allusertable(id).username)

GLOBAL.isLogIn <- TRUE

GLOBAL.user\_id <- id

ELSE

SEND 0

ENDIF

END\_PROC

#### Sign out

PROC sign\_out

DECLARE id: INT

id <- INPUT

update\_allusertable(id, "LOGIN\_STATUS", 0)

SEND 1

ENDPROC

#### Add friend

PROC add\_friend

DECLARE user\_id: INT

DECLARE friend\_id: INT

DECLARE request\_note: INT

user\_id <- INPUT

friend\_id <- INPUT

request\_note <- INPUT

insert\_friendrequesttable(user\_id, friend\_id, request\_note)

ENDPROC

#### Delete friend

PROC del\_friend

DECLARE user\_id: INT

DECLARE friend\_id: INT

user\_id <- INPUT

friend\_id <- INPUT

delete\_friendlisttable(user\_id, friend\_id)

delete\_friendlisttable(friend\_id, user\_id)

END\_PROC

#### Accept friend

PROC accept\_friend

DECLARE user\_id: INT

DECLARE friend\_id: INT

user\_id <- INPUT

friend\_id <- INPUT

insert\_friendlisttable(friend\_id, user\_id)

insert\_friendlisttable(friend\_id, user\_id)

delete\_friendrequesttable(user\_id, friend\_id)

ENDPROC

#### Refuse friend

PROC refuse\_friend

DECLARE user\_id: INT

DECLARE friend\_id: INT

delete\_friendrequesttable(user\_id, friend\_id)

ENDPROC

#### Send msg

PROC send\_msg

DECLARE friend\_id: INT

DECLARE msg: STRING

DECLARE user\_id: INT

DECLARE time: TIMESTAMP

DECLARE all\_user\_id: ARRAY[INT]

friend\_id <- INPUT

msg <- INPUT

user\_id <- GLOBAL.user\_id

time <- CURRENT\_TIME

all\_user\_id <- query\_friendlisttable(user\_id)

IF friend\_id in all\_user\_id THEN

insert\_msgtable(friend\_id, time, msg)

ENDIF

ENDPROC

#### Refresh

PROC refresh

DECLARE user\_id: INT

DECLARE message: ARRAY

DECLARE friend\_req: ARRAY

DECLARE friends: ARRAY

user\_id <- INPUT

message <- query\_msgtable(user\_id)

friend\_req <- query\_friendrequesttable(user\_id)

friends <- query\_friendlisttable(user\_id)

SEND message, friend\_req, friends

ENDPROC

#### Database operation

### Client end

#### User Interface

PROC GUI

DECLARE data

DECLARE msg\_list

DECLARE friend\_list

DECLARE friend\_req

SUBPROC react

IF LoginoutButtonClicked THEN

DECLARE id: INT

DECLARE pwd: STRING

WHILE Failed == TRUE

id <- INPUT "id"

pwd <- INPUT "pwd"

sign\_in(id, pwd)

IF Failed THEN

Pop Window Say "Fail"

ENDIF

ENDWHILE

ELSE IF SignUpButtonClicked THEN

DECLARE user\_name: STRING

DECLARE passwd: STRING

WHILE Failed == TRUE

user\_name <- INPUT "user\_name"

passwd <- INPUT "password"

sign\_up(user\_name, passwd)

IF Failed THEN

Pop Window Say "Fail"

ENDIF

ENDWHILE

ELSE IF AllFriendButtonClicked THEN

Show All Friends

WHILE TRUE

IF SendMessageButtonClicked THEN

SET selected friend in main window \

to friend\_id selected in top window

BREAK

ELSE IF DeleteButtonClicked THEN

del\_friend(selected\_friend\_id)

BREAK

ENDIF

ENDWHILE

ELSE IF AllRequestButtonClicked THEN

Show All FriendRequests

WHILE TRUE

IF AcceptButtonClicked THEN

accept\_friend(selected\_friend\_id)

BREAK

ELSE IF RefuseButtonClicked THEN

refuse\_friend(SendButtonClicked)

BREAK

ENDIF

ENDWHILE

ELSE IF AddFriendButtonClicked THEN

DECLARE friend\_id: INT

DECLARE request\_note: STRING

friend\_id <- INPUT "friend id"

request\_note <- INPUT "request note"

add\_friend(selected\_friend\_id)

ELSE IF ClearHistoryButtonClicked THEN

clear\_history(selected\_friend\_id)

ELSE IF SendButtonClicked THEN

DECLARE msg: STRING

msg <- INPUT(GetFromEntry)

send\_message(selected\_friend\_id, msg)

ENDIF

ENDSUBPROC

SUBPROC refresh

data <- refresh()

friend\_list <- data.friend

friend\_req <- data.fri\_req

msg <- data.msg

refresh window

ENDSUBPROC

CALL refresh IN NEW THREAD

CALL react

ENDPROC

#### Logic Process

DECLARE connect() <- FUNC

CONNECT TO SERVER

ENDFUNC

DECLARE sign\_up() <- FUNC

DECLARE user\_name: STRING

DECLARE passwd: STRING

user\_name <- INPUT "Input username"

passwd <- INPUT "Input password"

SEND (

"mode": "sign\_up",

"name": user\_name,

"pwd": passwd

)

ENDFUNC

DECLARE sign\_in() <- FUNC

DECLARE user\_id: INT

DECLARE passwd: STRING

DECLARE user\_name: STRING

user\_id <- INPUT "Input id"

passwd <- INPUT "Input passwd"

SEND (

"mode": "sign\_in",

"id": user\_id,

"passwd": passwd

)

user\_name <- RECEIVE

GLOBAL.LOGIN\_STATUS <- ONLINE

RETURN user\_name

ENDFUNC

DECLARE sign\_out() <- FUNC

SEND ("mode" = "sign\_out")

GLOBAL.LOGIN\_STATUS <- OFFLINE

ENDFUNC

DECLARE add\_friend() <- FUNC

DECLARE friend\_id: INT

DECLARE req\_note: STRING

friend\_id <- INPUT "friend id"

req\_note <- "request note"

SEND (

"mode": "add\_friend",

"friend\_id": friend\_id,

"req\_note": req\_note

)

ENDFUNC

DECLARE del\_friend() <- FUNC

DECLARE friend\_id

friend\_id <- INPUT "friend id"

SEND (

"mode": "del\_friend",

"friend\_id": friend\_id

)

ENDFUNC

DECLARE accept\_friend() <- FUNC

DECLARE friend\_id: INT

friend\_id <- INPUT "friend id"

SEND (

"mode": "accept\_friend",

"friend\_id": friend\_id

)

ENDFUNC

DECLARE refuse\_friend() <- FUNC

DECLARE friend\_id: INT

friend\_id <- INPUT "friend id"

SEND (

"mode": "refuse\_friend",

"friend\_id": "friend\_id"

)

ENDFUNC

DECLARE send\_message() <- FUNC

DECLARE friend\_id: INT

DECLARE message: STRING

friend\_id <- INPUT "friend id"

message <- INPUT "message"

SEND (

"mode": "send\_msg",

"friend\_id": friend\_id,

"msg": msg

)

insert\_msgtable(friend\_id, 1, CURRENT\_TIME)

ENDFUNC

DECLARE refresh() <- FUNC

DECLARE data\_remote: DICT

DECLARE data\_local: DICT

SEND ("mode": "refresh")

data\_remote <- RECEIVE

data\_local <- query\_msgtable()

RETURN COMBINE(data\_remote, data\_local)

ENDFUNC

DECLARE clear\_history() <- FUNC

DECLARE friend\_id

friend\_id <- INPUT "friend id"

delete\_msgtable(friend\_id)

ENDFUNC

DECLARE close\_sock() <- FUNC

CLOSE CONNECTION

ENDFUNC

#### Database Operation

DECLAR create\_mgtable() <- FUNC

CREATE TABLE

FRIEND\_ID INT NOT NULL,

IMSENDER BOOL NOT NULL,

TIME TIMESTAMP NOT NULL,

MSG TEXT NOT NULL

ENDFUNC

DECLARE insert\_msgtable(friend\_id, imsender, time, msg) = FUNC

INSERT friend\_id, imsender, time, msg INTO MSG

ENDFUNC

DECLARE delete\_msgtable(friend\_id) <- FUNC

DELETE where FRIEND\_ID = friend\_id

ENDFUNC

DECLARE query\_msgtable() <- FUNC

data = SELECT FRIEND\_ID, IMSENDER, TIME, MSG

RETURN data

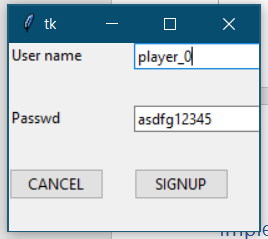
ENDFUNC

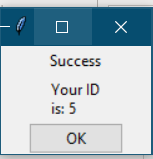
# Implementation

Code in appendix.

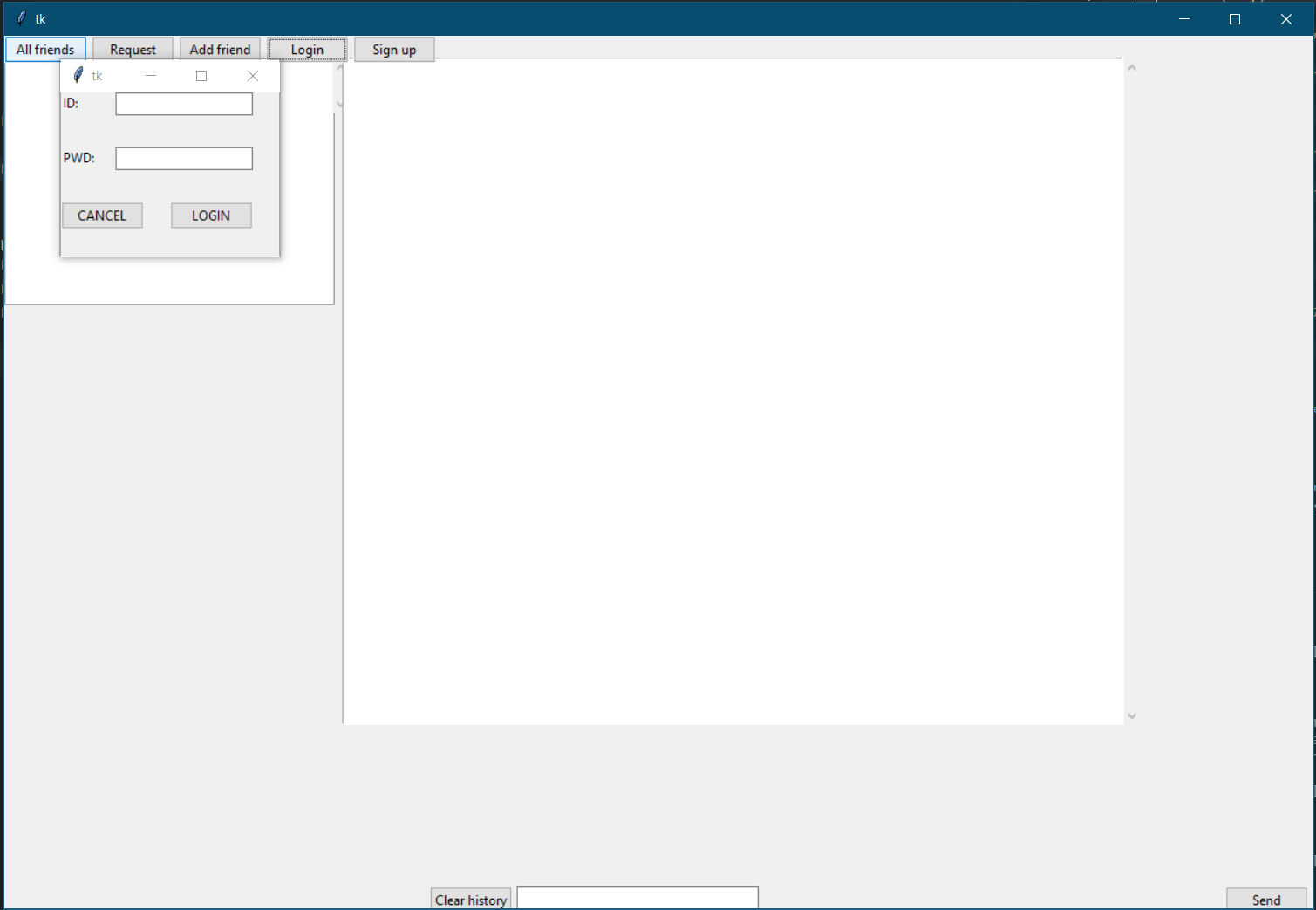
# Testing

Sign up

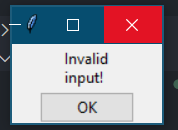




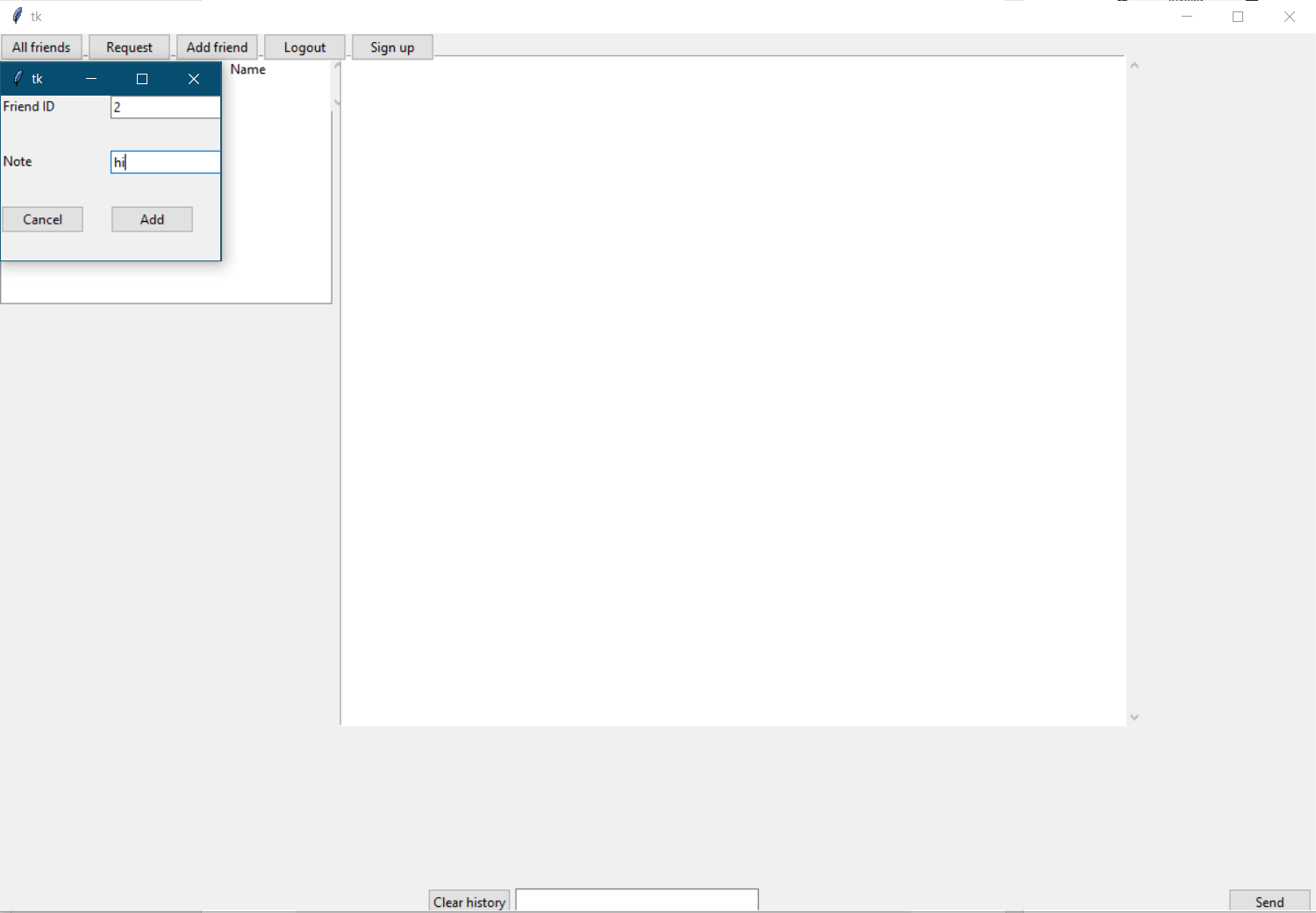
Login

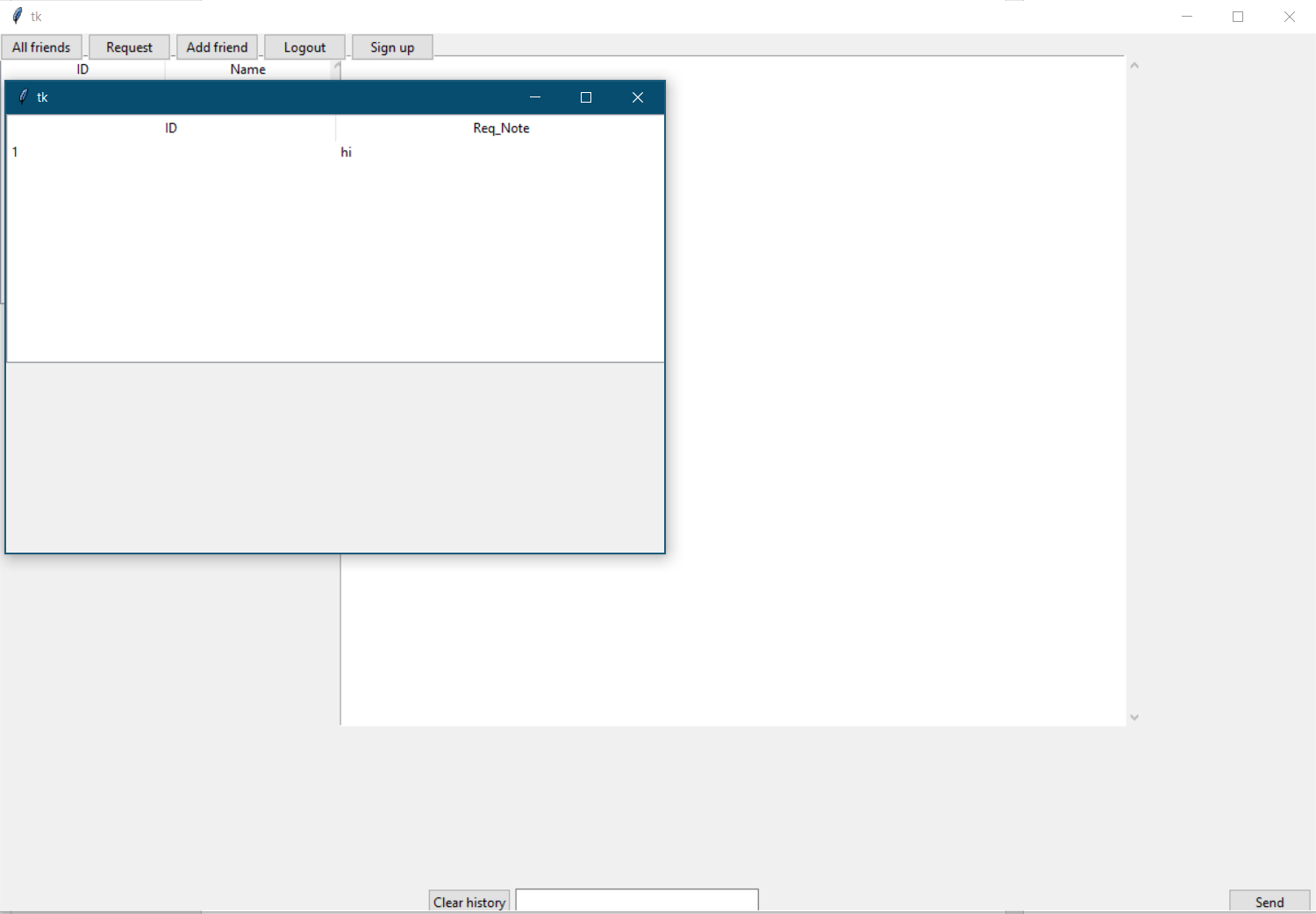


Empty / invalid input

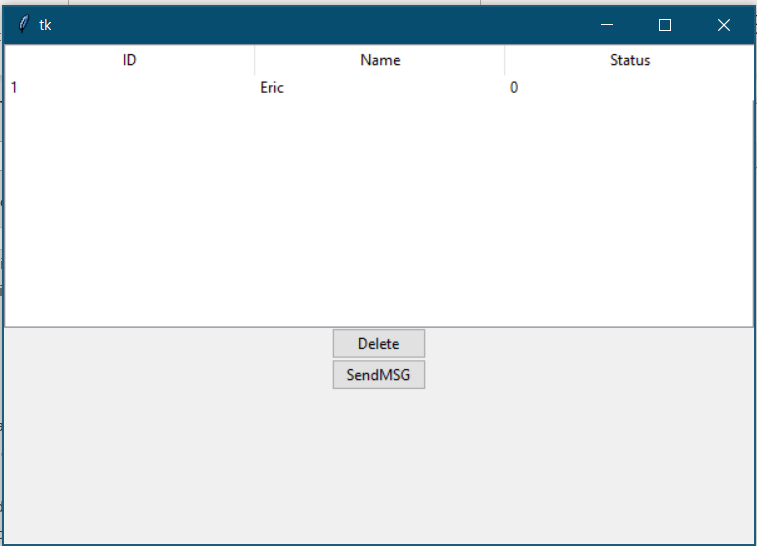


Add friend

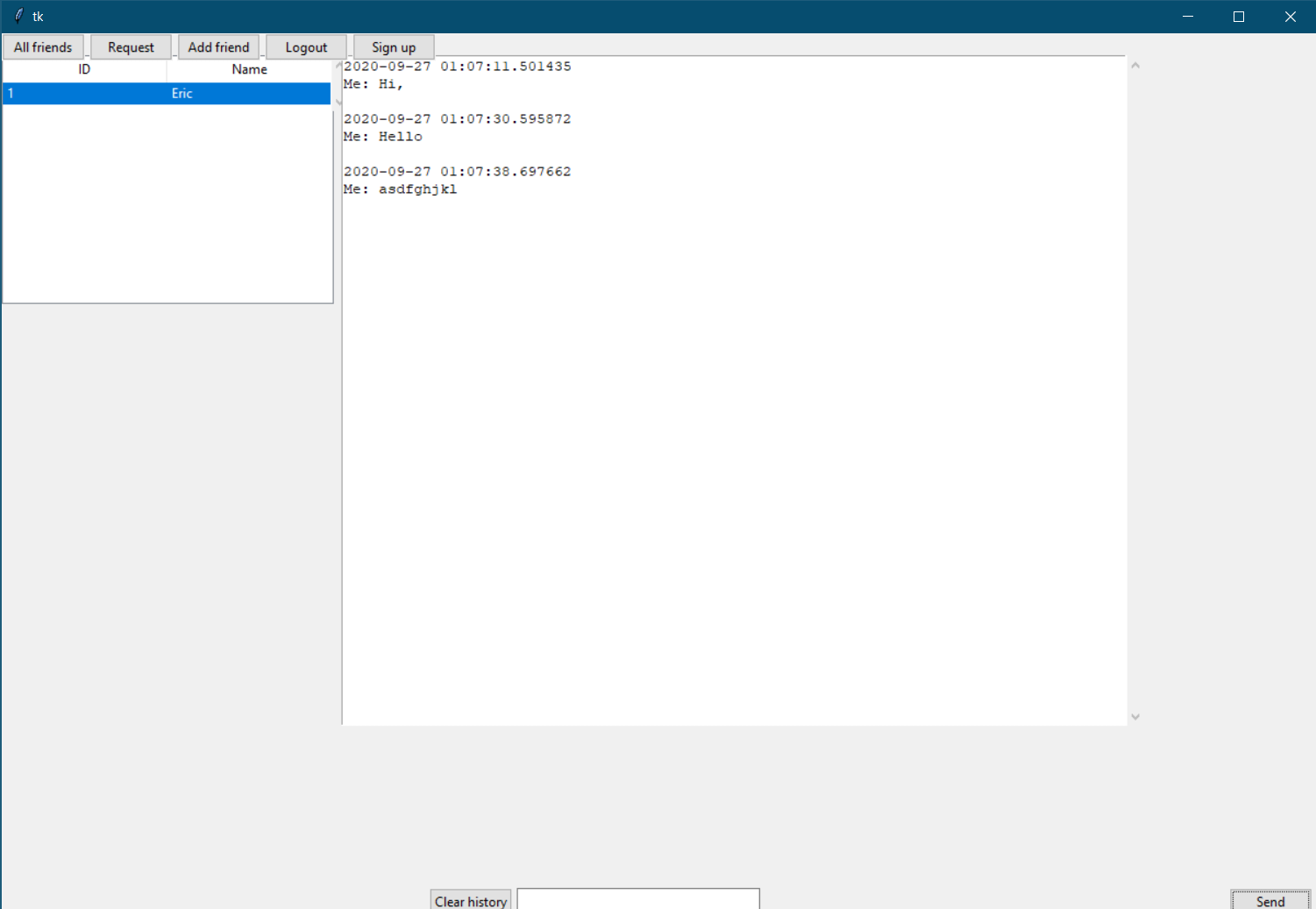


User receive the friend request

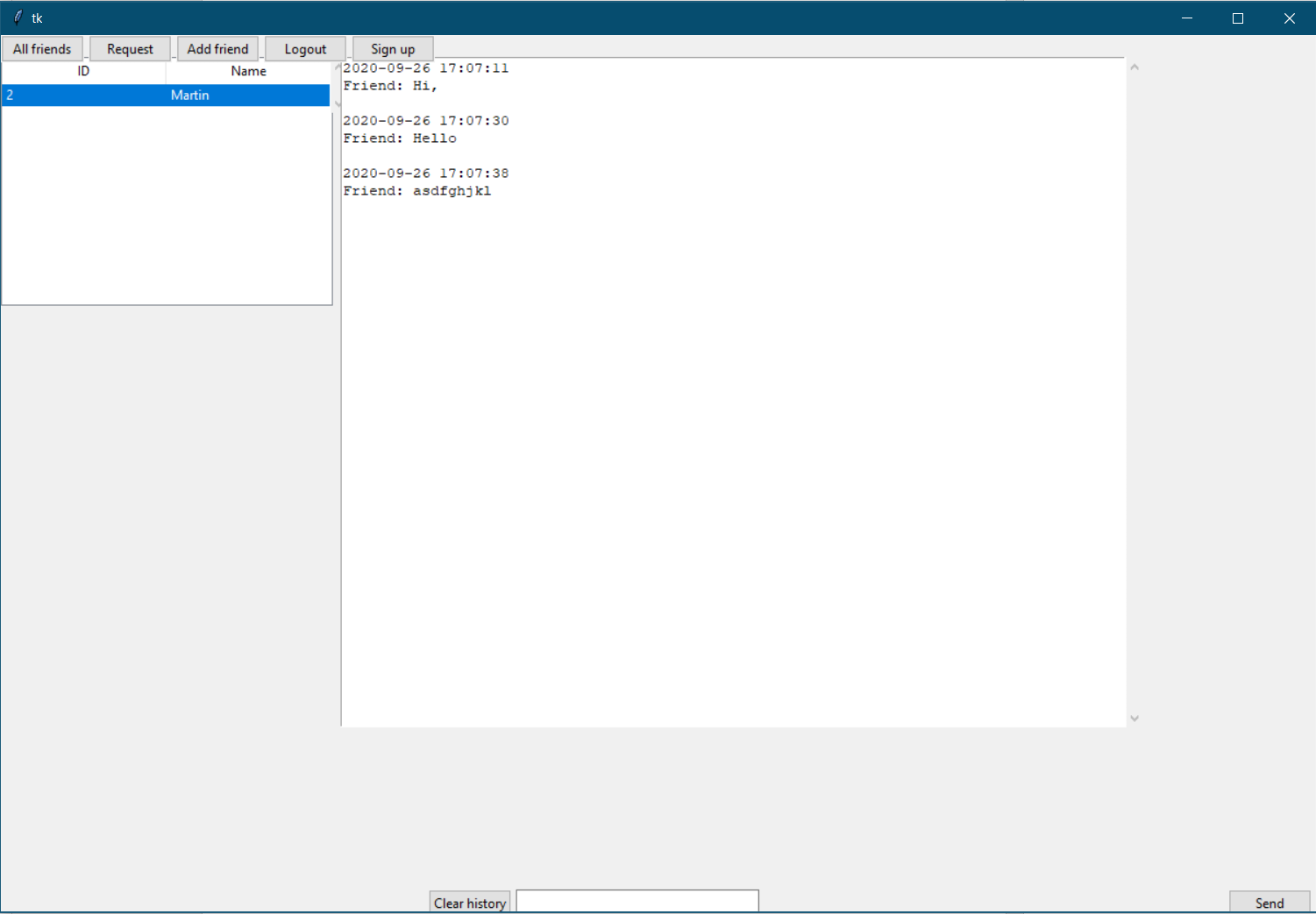
User accept the request



Send message to friend



Friend receive message



# TODOs & points should be improved

* If the receiver user is online, the server should send the data from sender to the receiver directly instead of putting into database and then fetch out.
* Should add scrollbar for friend tree view. Otherwise, when the number of friends exceed a given value, they can’t be displayed.
* Should have “change name” functionality so that end user can change username when needed.
* More intuitive and good-look GUI
* Shell scripts to configure the server end program and install the client end program

# Reference

1. Eric Jin . multi-user-chatting-software . <https://github.com/EricEricEricJin/multi-user-chatting-software>
2. Eric Jin . Python DIY简易多人聊天软件 . <https://blog.csdn.net/weixin_43528943/article/details/104721649>
3. Runoob . SQLite – Python . <https://www.runoob.com/sqlite/sqlite-python.html>