CS MINI IA

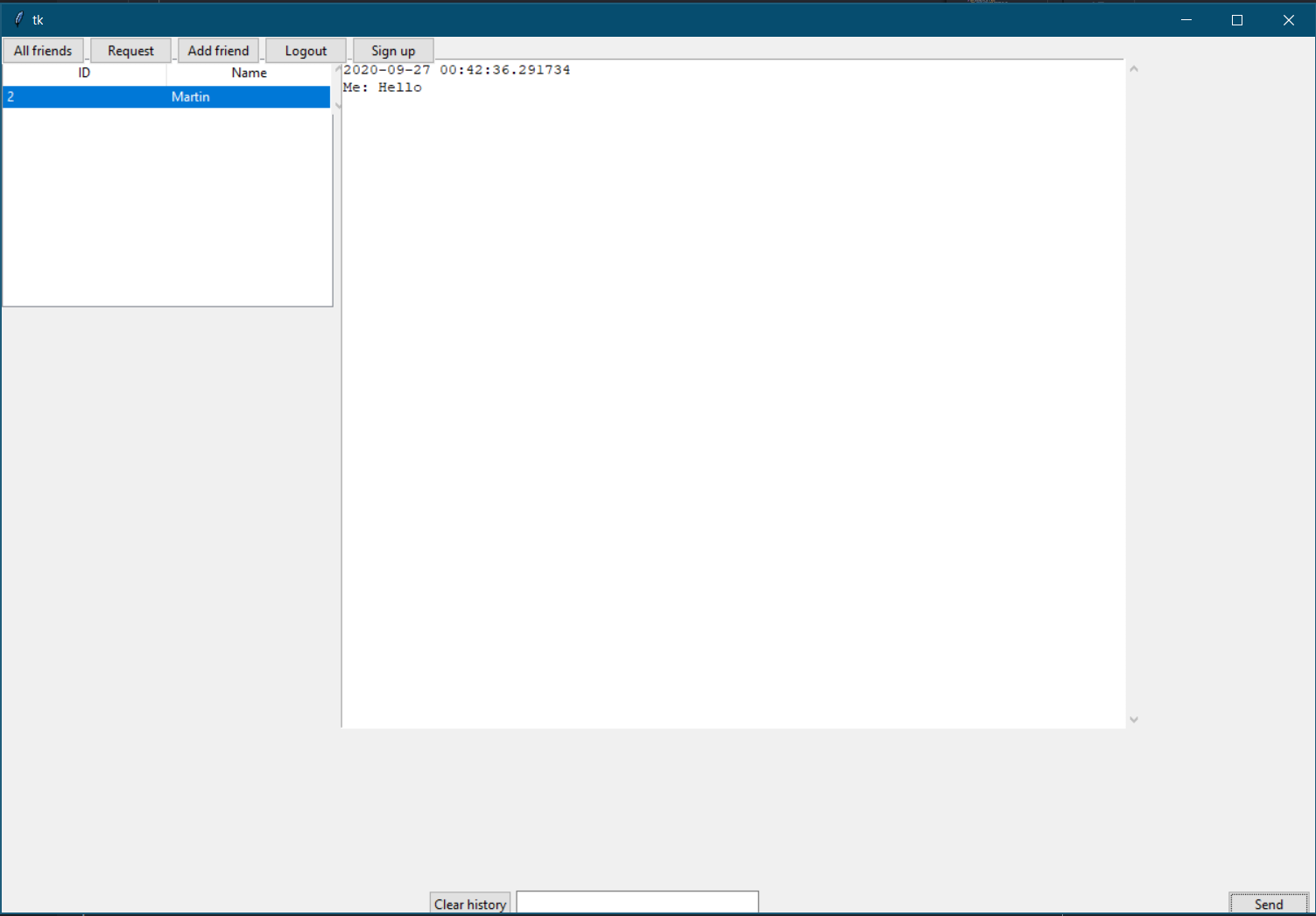
Multi User Client-Server Chatting software

Author: DP2-1 Eric Jin

Date: September.26.2020

Subject: IB CS HL

Github Repository link: <https://github.com/EricEricEricJin/CS_MINI_IA>



Contents

[Problem description 4](#_Toc52139664)

[Requirements 4](#_Toc52139665)

[Functional 4](#_Toc52139666)

[Non-functional 4](#_Toc52139667)

[Tools choice 4](#_Toc52139668)

[Design 5](#_Toc52139669)

[Communication design 5](#_Toc52139670)

[Database design 6](#_Toc52139671)

[Server end 6](#_Toc52139672)

[Client end 7](#_Toc52139673)

[UI design 8](#_Toc52139674)

[Main window 8](#_Toc52139675)

[Login Popup window 8](#_Toc52139676)

[Sign up Popup window 8](#_Toc52139677)

[Add friend Popup window 9](#_Toc52139678)

[Request Popup window 9](#_Toc52139679)

[Friend Popup window 9](#_Toc52139680)

[Top down design 10](#_Toc52139681)

[Server end 10](#_Toc52139682)

[Client end 10](#_Toc52139683)

[Logic Flowcharts 11](#_Toc52139684)

[Server end 11](#_Toc52139685)

[Client end 19](#_Toc52139686)

[Pseudocode 22](#_Toc52139687)

[Server end 22](#_Toc52139688)

[Client end 29](#_Toc52139689)

[Implementation 34](#_Toc52139690)

[Identifier table 34](#_Toc52139691)

[Testing 35](#_Toc52139692)

[Testing plan 35](#_Toc52139693)

[Testing 36](#_Toc52139694)

[Sign up with valid input 36](#_Toc52139695)

[Sign up with invalid input 36](#_Toc52139696)

[Login with valid input 36](#_Toc52139697)

[Login with invalid input 37](#_Toc52139698)

[Login with wrong password 37](#_Toc52139699)

[Login with not exist ID 37](#_Toc52139700)

[Add friend which do not exist 38](#_Toc52139701)

[Add friend that exist 38](#_Toc52139702)

[Delete friend 38](#_Toc52139703)

[Accept friend 39](#_Toc52139704)

[Refuse friend 39](#_Toc52139705)

[Send message to friend 39](#_Toc52139706)

[Receive & reply friends 40](#_Toc52139707)

[Run client program when server is not available 40](#_Toc52139708)

[TODOs & points should be improved 41](#_Toc52139709)

[Reflection 41](#_Toc52139710)

[Reference 42](#_Toc52139711)

# Problem description

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

This software should have a server end and a client end. User should be able to login, add friends, and chat with each other.

# Requirements

## Functional

* 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 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.

## Non-functional

* Text field of message display can always seek to bottom for latest message.

# Tools choice

|  |  |  |
| --- | --- | --- |
| Item | Choice | Reason |
| Develop language | Python3 | Less code, cross platform, simple |
| GUI Lib | Tkinter | Integrated in Python3 release, easy to use, widely used. |
| Database | sqlite3 | Integrated in Python3 release, doesn’t require complex configurations, light |
| Communication protocol | TCP | Don’t need to write code to check data. |

# 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.

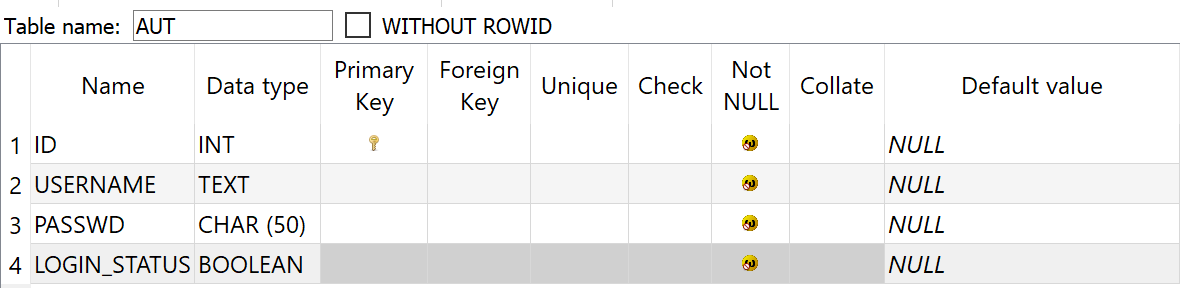
It have other keys and values needed for the mode

|  |  |  |
| --- | --- | --- |
| 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: {friend\_id: request\_note}, {friend\_id: [friend\_username, friend\_loginstatus]}} |

## Database design

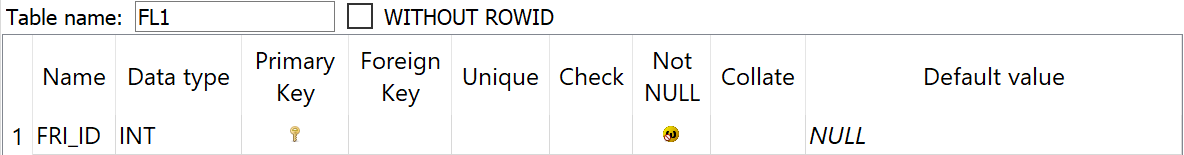
### Server end

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

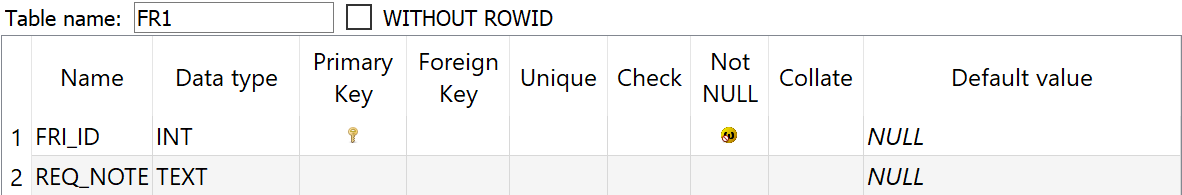


One **FL<ID>** table for each user store friends’ ID

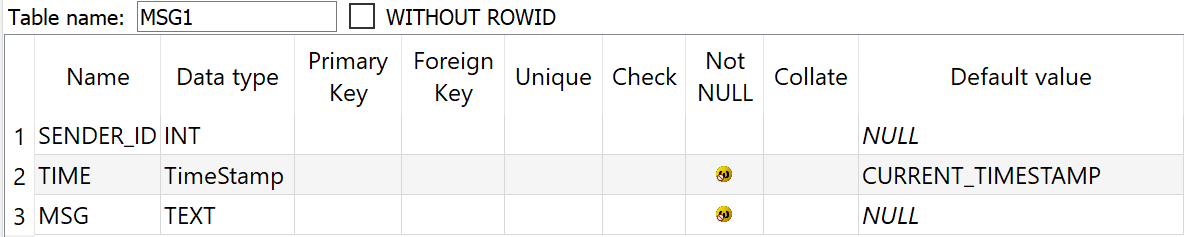
\*FL<ID> , for example ID is 114, the table name is FL114



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

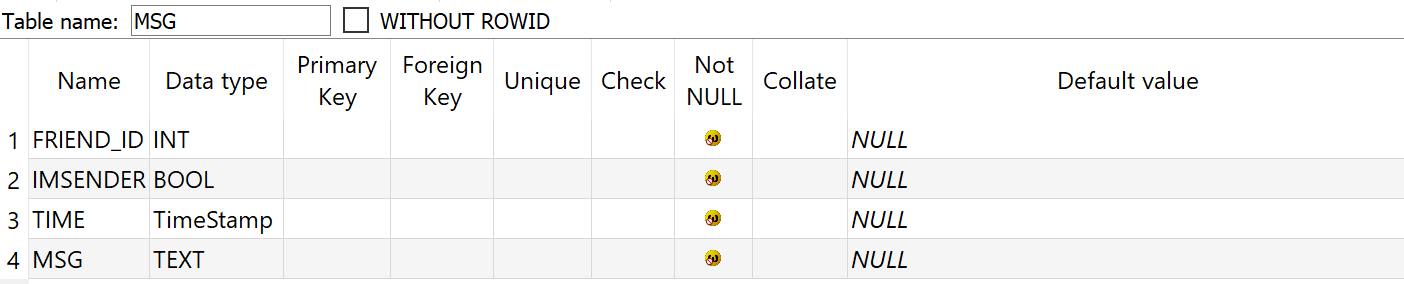


One **MSG<ID>** table for each user store unreceived messages



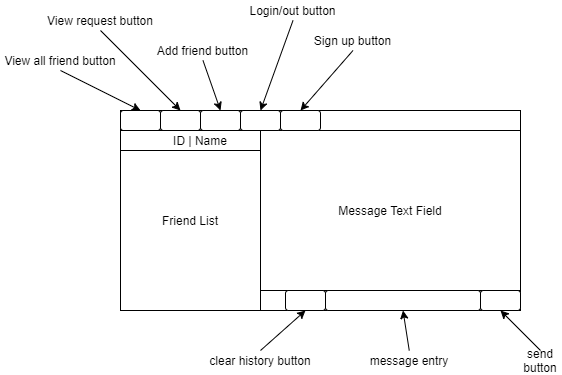
### Client end

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

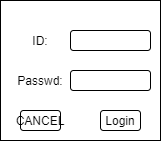


## UI design

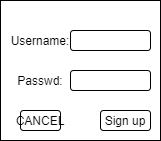
### Main window



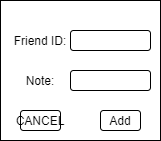
### Login Popup window



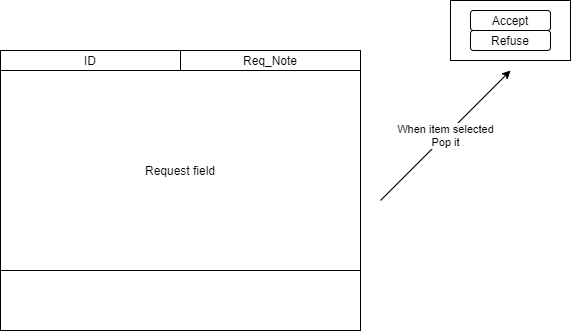
### Sign up Popup window



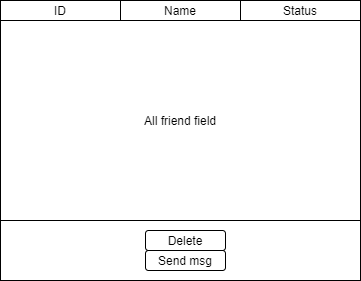
### Add friend Popup window



### Request Popup window

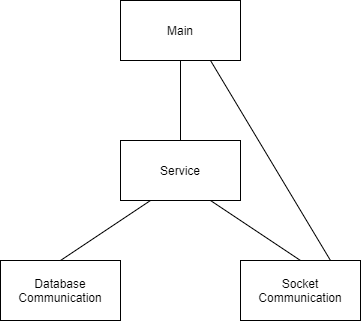


### Friend Popup window

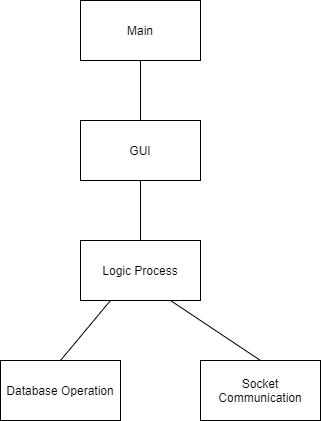


## Top down design

### Server end



### Client end



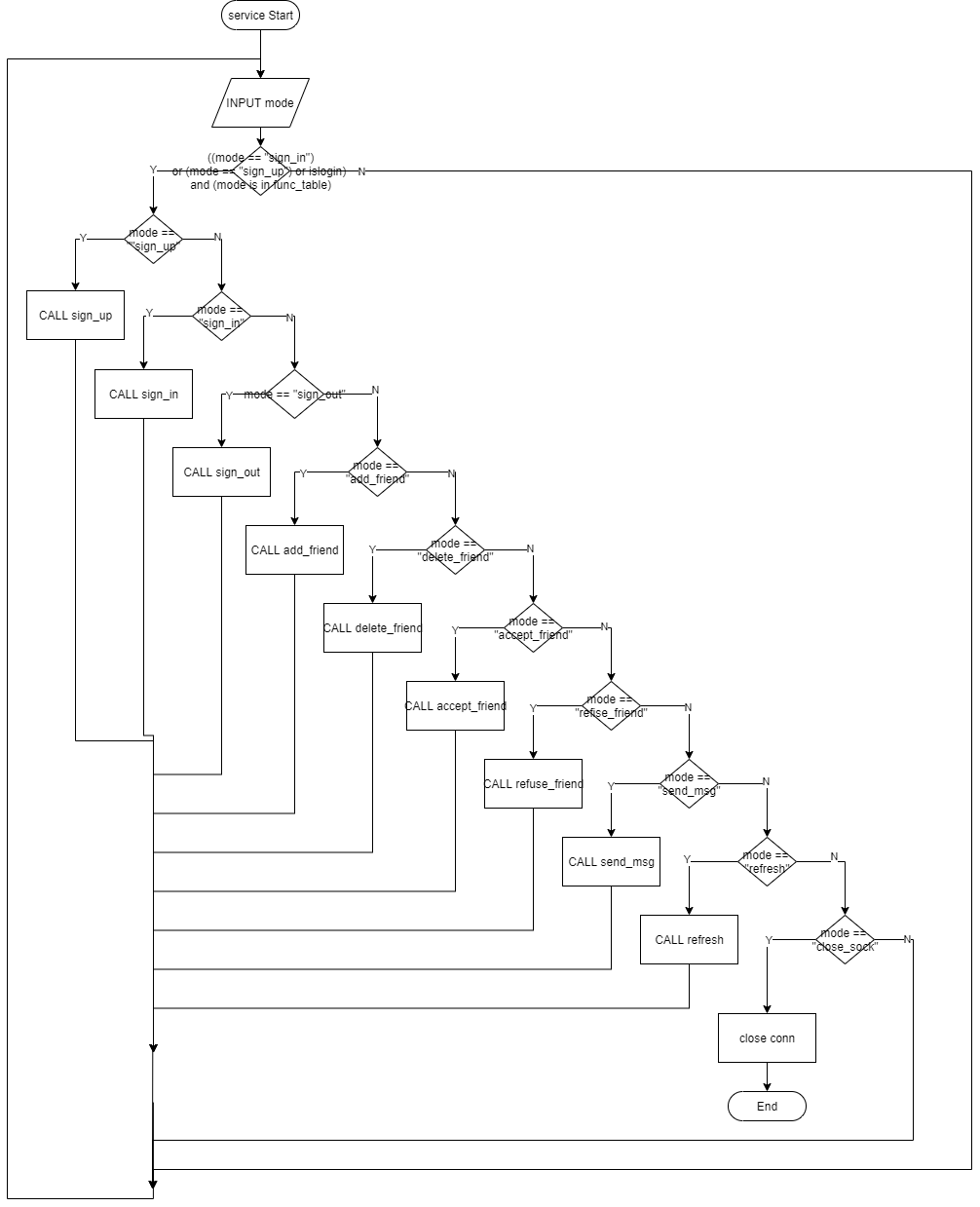
## Logic Flowcharts

### Server end

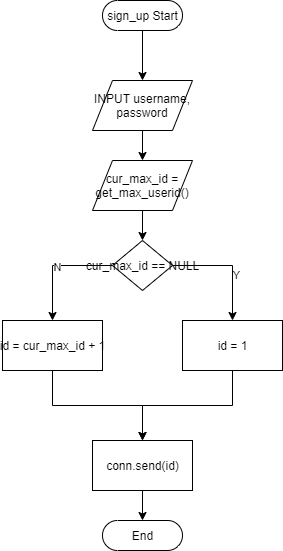
#### Main



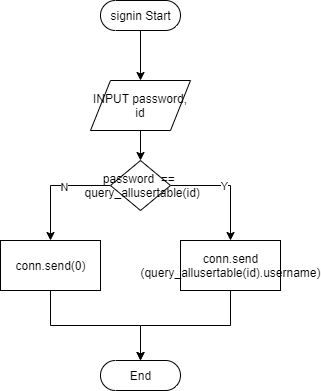
#### Service



#### Sign up



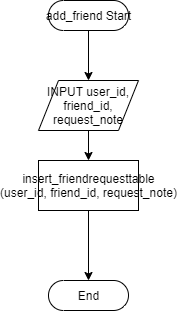
#### Sign in



#### Sign out



#### Add friend



#### Delete friend



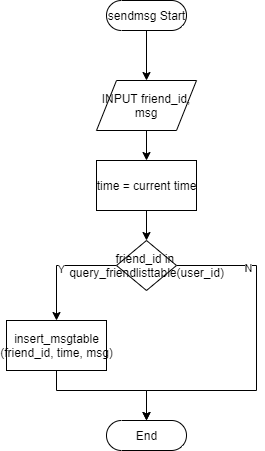
#### Accept friend



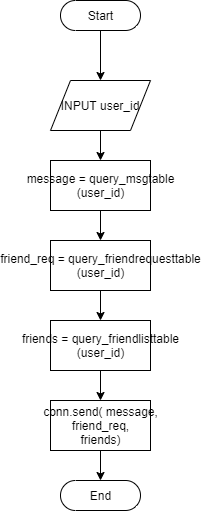
#### Refuse friend



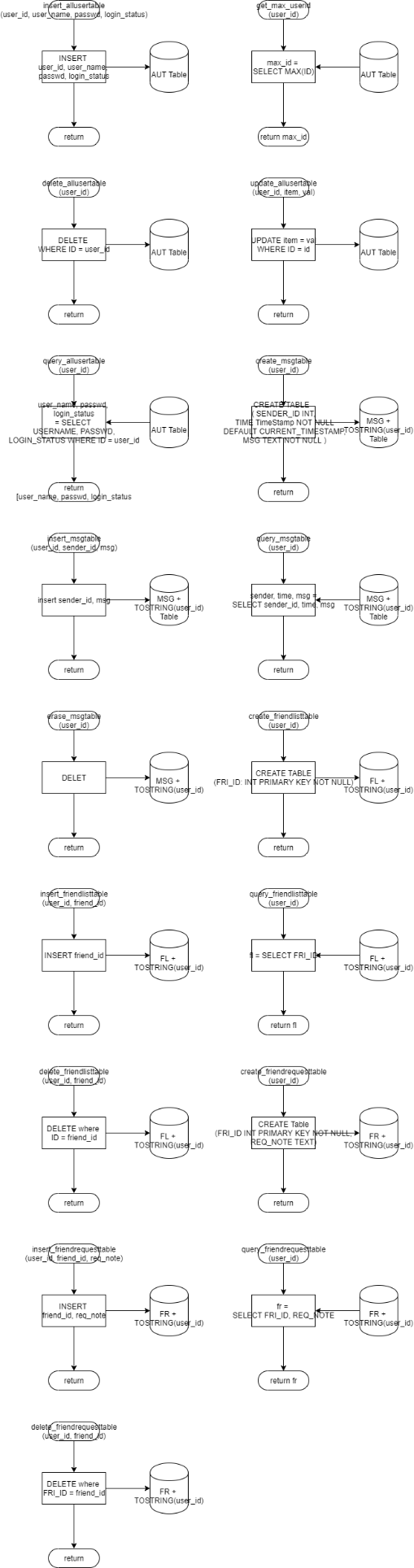
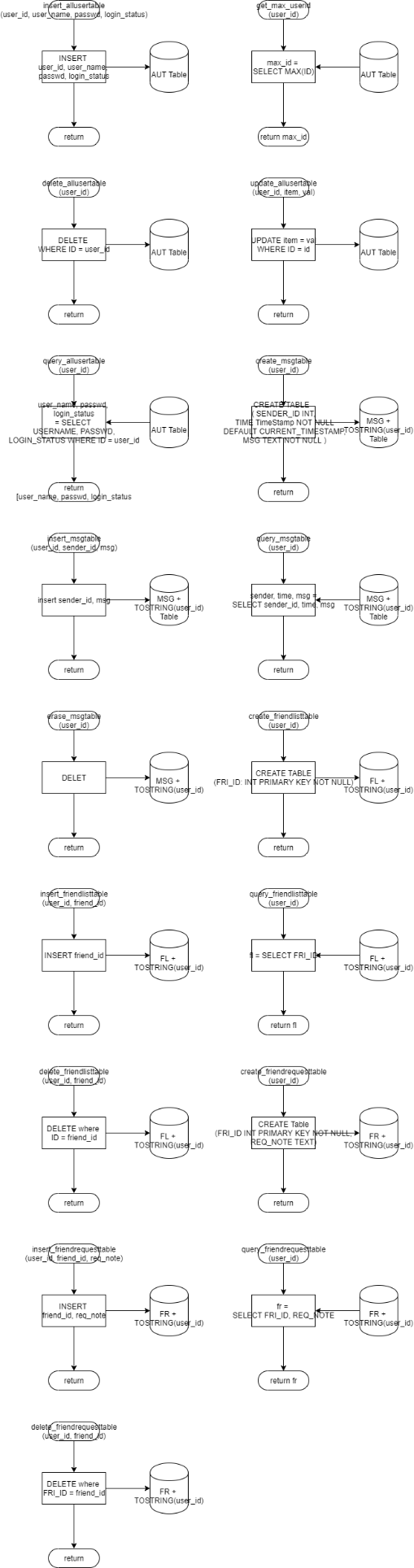
#### Send msg



#### Refresh

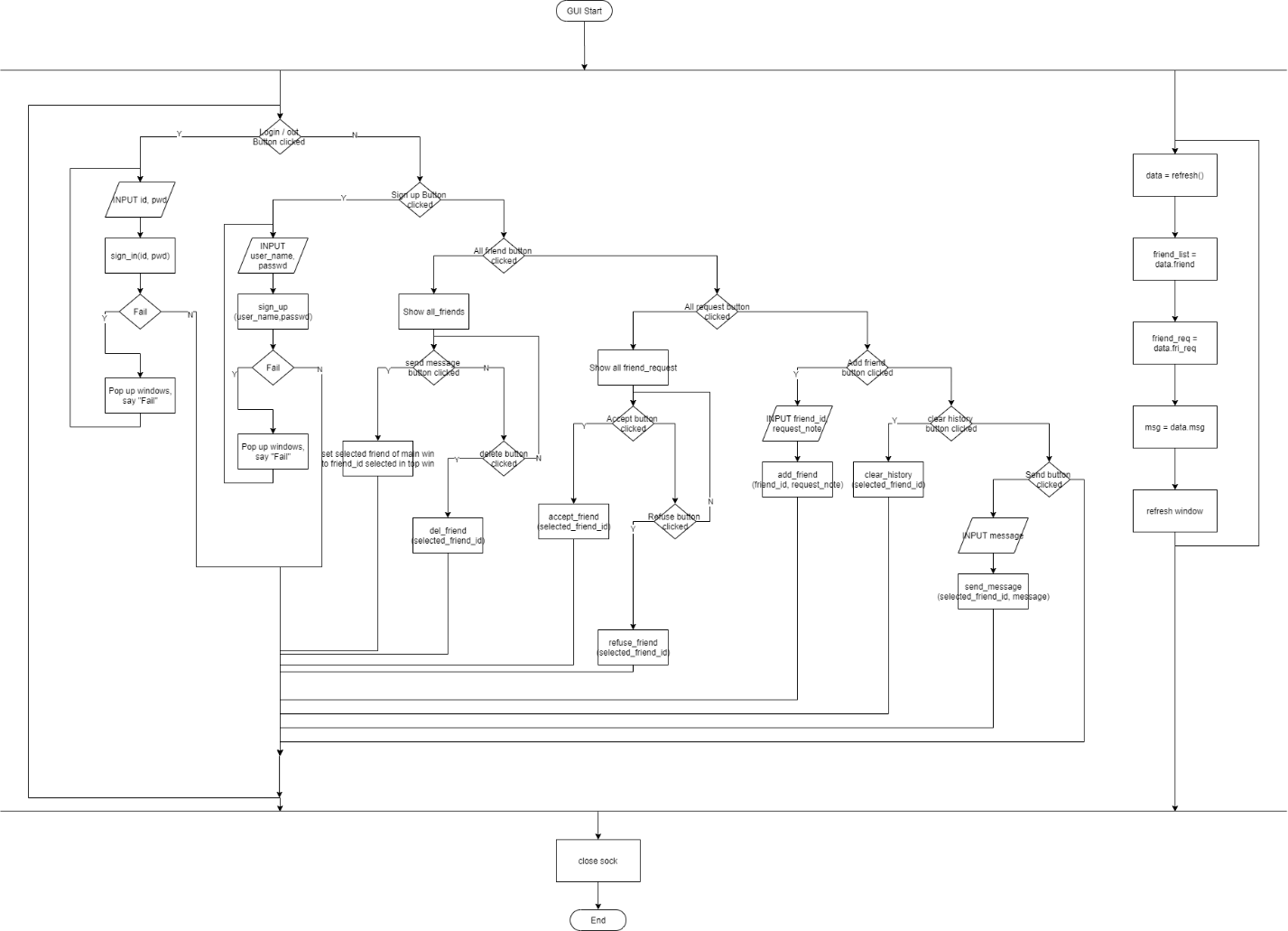


#### Database operation

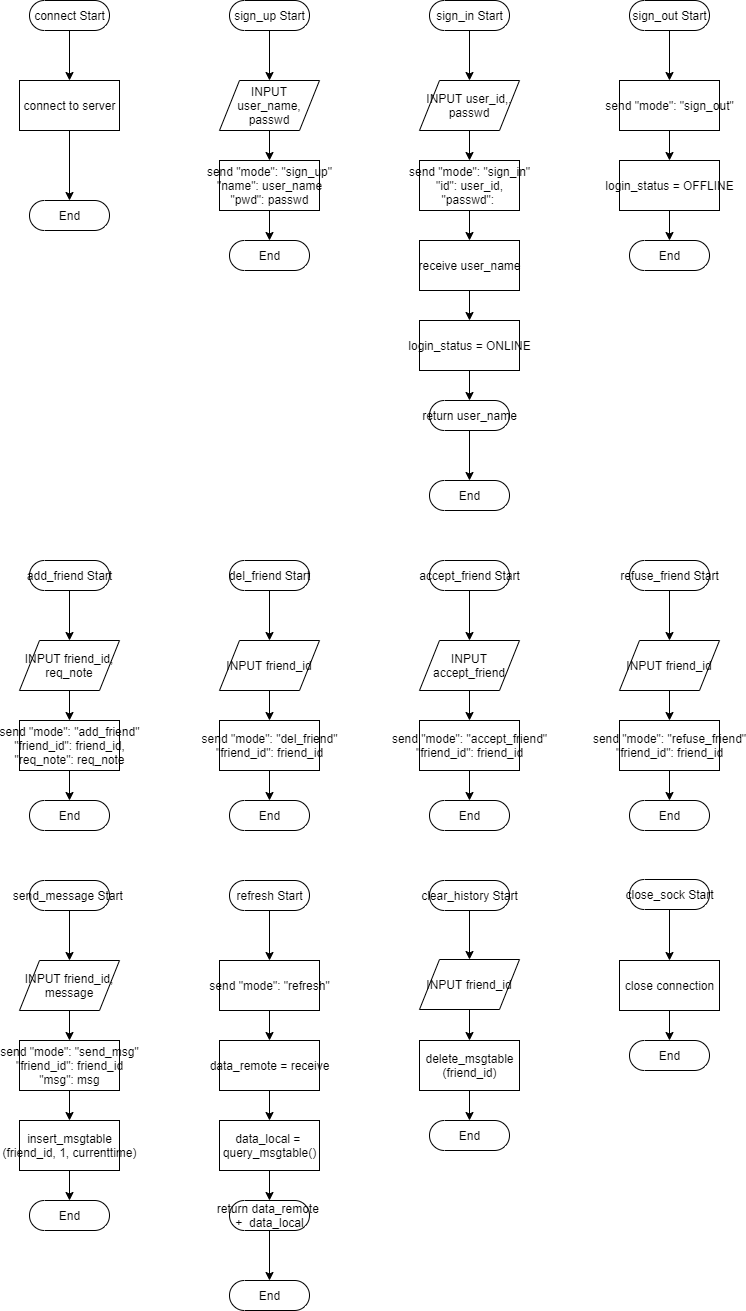
 

### Client end

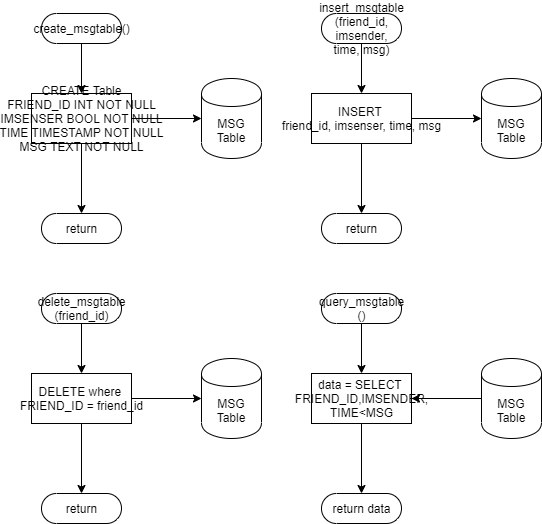
#### User Interface



#### Logic Process



#### Database operation



## Pseudocode

### Server end

#### Main

WHILE TRUE

DECLARE CONNECTION

WAIT FOR CONNECTION

IF CONNECT THEN

CALL service IN NEW THREAD

ENDIF

ENDWHILE

#### 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

DECLARE insert\_allusertable(user\_id, user\_name, passwd, login\_status) <- FUNC

INSERT user\_id, user\_name, passwd, login\_status \

INTO AUT Table

ENDFUNC

DECLARE get\_max\_userid(user\_id) <- FUNC

DECLARE max\_id: INT

max\_id <- SELECT MAX(ID) FROM AUT Table

RETURN max\_id

ENDFUNC

DECLARE delete\_allusertable(user\_id) <- FUNC

DELETE AUT Table WHERE ID = user\_id

ENDFUNC

DECLARE update\_allusertable(user\_id, item, val) <- FUNC

UPDATE AUT Table item = val where ID = user\_id

ENDFUNC

DECLARE query\_allusertable(user\_id) <- FUNC

DECLARE user\_name: STRING

DECLARE passwd: STRING

DECLARE login\_status: BOOL

DECLARE data" ARRAY

data <- SELECT USERNAME, PASSWD, LOGIN\_STATUS WHERE ID = user\_id

user\_name <- data.user\_name

passwd <- data.passwd

login\_status <- data.login\_status

RETURN user\_name, passwd, login\_status

ENDFUNC

DECLARE create\_msgtable(user\_id) <- FUNC

CREATE Table "MSG" + TOSTRING(user\_id) \

SENDER\_ID, TIME, MSG

ENDFUNC

DECLARE insert\_msgtable(user\_id, sender\_id, msg) <- FUNC

INSERT INTO "MSG" + TOSTRING(user\_id) \

sender\_id, CURRENT\_TIME, msg

ENDFUNC

DECLARE query\_msgtable(user\_id) <- FUNC

DECLARE data: ARRAY

DECLARE sender\_id: INT

DECLARE time: TIMESTAMP

DECLARE msg: STRING

data = SELECT sender\_id, time, msg FROM MSH + TOSTRING(user\_id)

sender\_id <- data.sender\_id

time <- data.time

msg <- data.msg

RETURN sender\_id, time, msg

ENDFUNC

DECLARE erase\_msgtable(user\_id) <- FUNC

DELETE Table "MSG" + TOSTRING(user\_id)

ENDFUNC

DECLARE create\_friendlisttable(user\_id) <- FUNC

CREATE Table "FL" + TOSTRING(user\_id) \

FRI\_ID

ENDFUNC

DECLARE insert\_friendlisttable(user\_id, friend\_id) <- FUNC

INSERT INTO Table "FL" + TOSTRING(user\_id) \

friend\_id

ENDFUNC

DECLARE query\_friendlisttable(user\_id) <- FUNC

DECLARE friend\_list: ARRAY

friend\_list <- SELECT FROM Table "FL" + TOSTRING(user\_id)

RETURN friend\_list

ENDFUNC

DECLARE delete\_friendlisttable(user\_id, friend\_id) <- FUNC

DELETE FROM Table "FL" + TOSTRING(user\_id) WHERE ID = friend\_id

ENDFUNC

DECLARE create\_friendrequesttable(user\_id) <- FUNC

CREATE Table "FR" + TOSTRING(user\_id) \

FRI\_ID, REQ\_NOTE

ENDFUNC

DECLARE insert\_friendrequesttable(user\_id, friend\_id, req\_note) <- FUNC

INSERT INTO Table "FR" + TOSTRING(user\_id) friend\_id, req\_note

ENDFUNC

DECLARE query\_friendrequesttable(user\_id) <- FUNC

DECLARE friend\_req: ARRAY

friend\_req <- SELECT FRI\_ID, REQ\_NOTE FROM Table "FL" + TOSTRING(user\_id)

RETURN friend\_req

ENDFUNC

DECLARE delete\_friendrequesttable(user\_id, friend\_id) <- FUNC

DELETE "FR" + TOSTRING(user\_id) WHERE FRI\_ID = friend\_id

ENDFUNC

### 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

Python code attached in appendix.

## Identifier table

(Client end)

|  |  |  |
| --- | --- | --- |
| Variable name | Type / Structure | Meaning |
| msg\_list | {fri\_id\_0(INT): [sender(INT), time(STRING), msg(STRING)], …} | Dictionary of all friends and message send to / receive from them and time of messages |
| friend\_list | {fri\_id\_0(INT): [name(STRING), login\_status(BOOL)], …} | Dictionary of all friends and their name, status |
| friend\_req\_list | {fri\_id\_0(INT): req\_note(STRING), …} | Dictionary of all friend-requests |
| id2item\_friend\_treeview\_table | {friend\_id(INT): treeview\_item(Tkinter\_WidgetNo), …} | Dictionary of all treeview items |
| selected\_fri\_id | friend\_id(INT) | ID of the friend selected in friend treeview |

# Testing

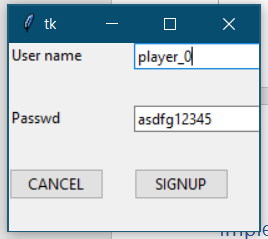
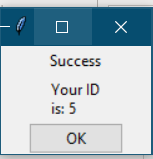
## Testing plan

The testing will be done under following scenarios:

1. Sign up with valid input
2. Sign up with invalid input
3. Login with valid input
4. Login with invalid input
5. Login with wrong password
6. Login with not-exist ID
7. Add friend which do not exist
8. Add friend that exist
9. Delete friend
10. Accept friend
11. Refuse friend
12. Send message to friend
13. Receive and reply to friends
14. Run program when server not available

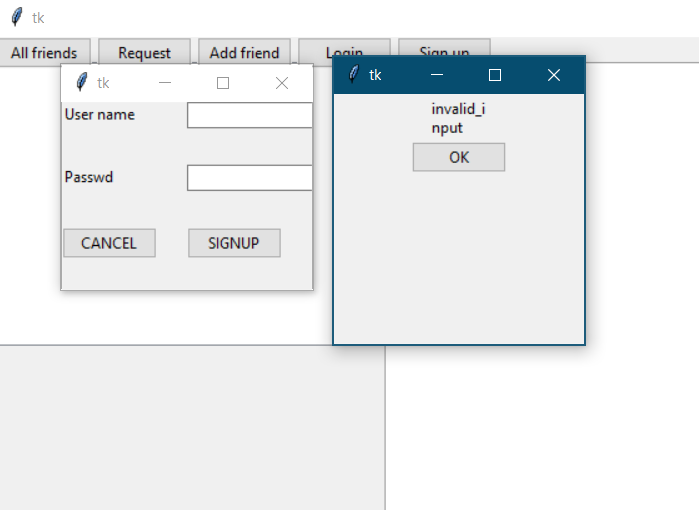
## Testing

### Sign up with valid input

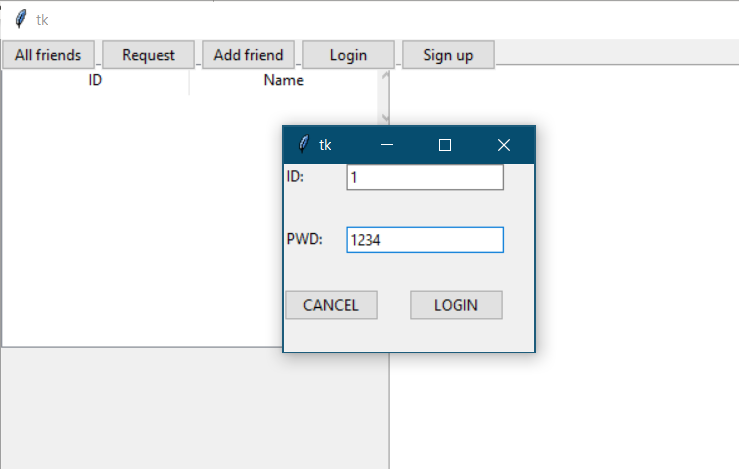
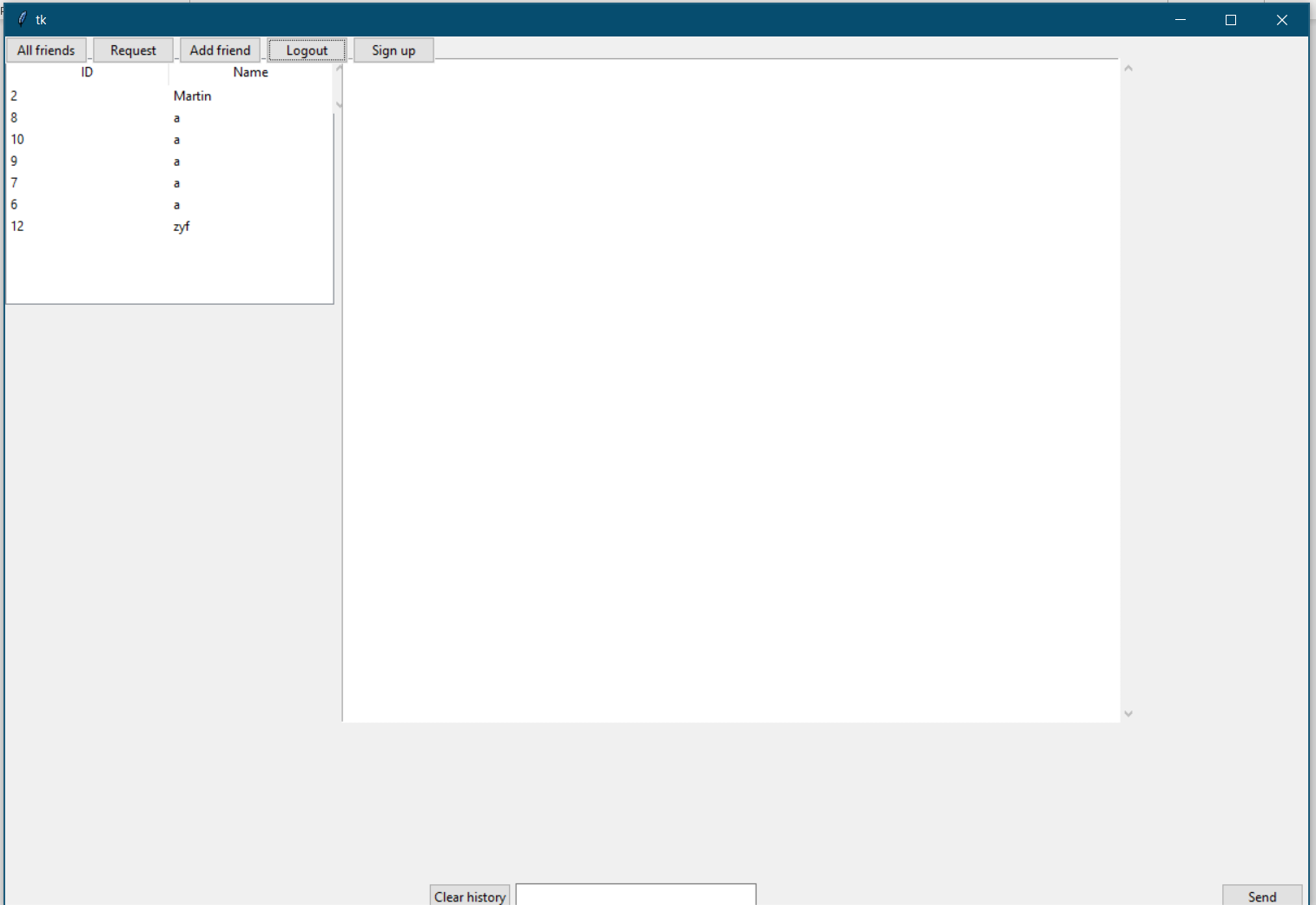
 

### Sign up with invalid input

Give empty input.

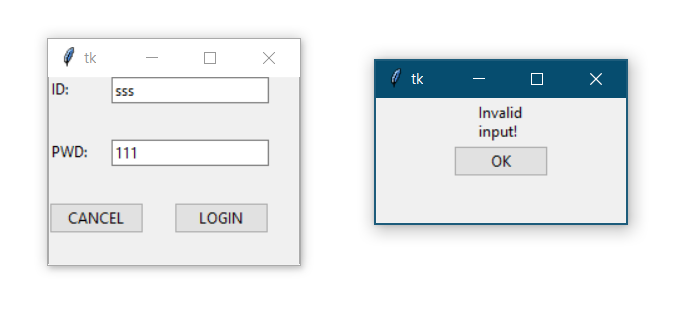


### Login with valid input

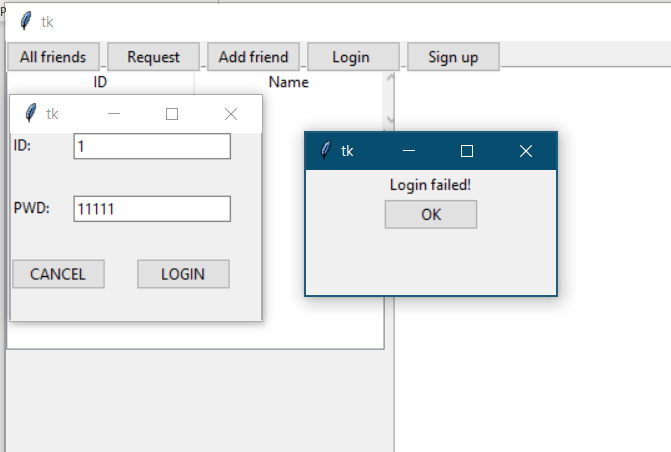
 

### Login with invalid input

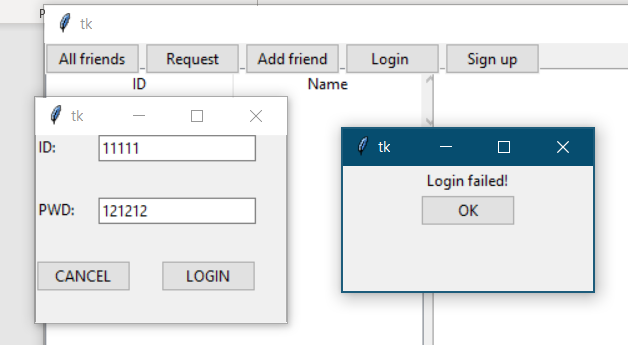
Input non-integer as ID



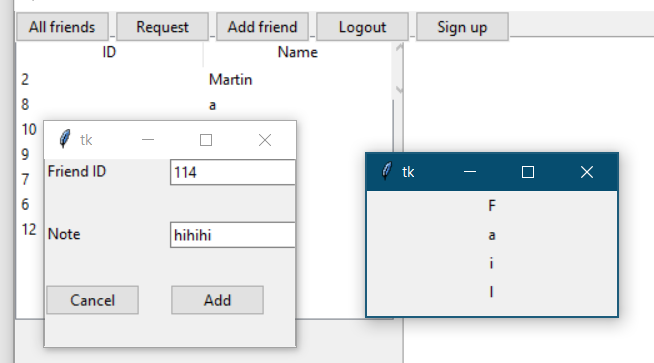
### Login with wrong password



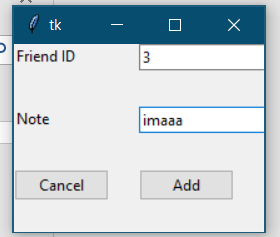
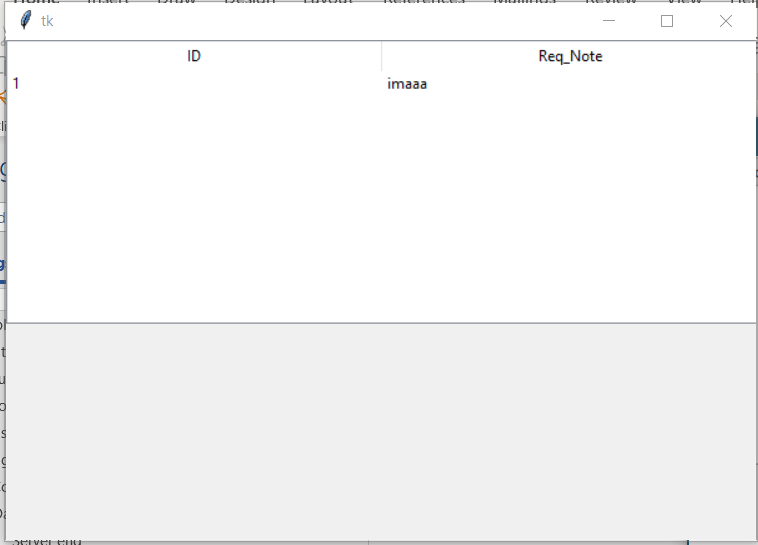
### Login with not exist ID



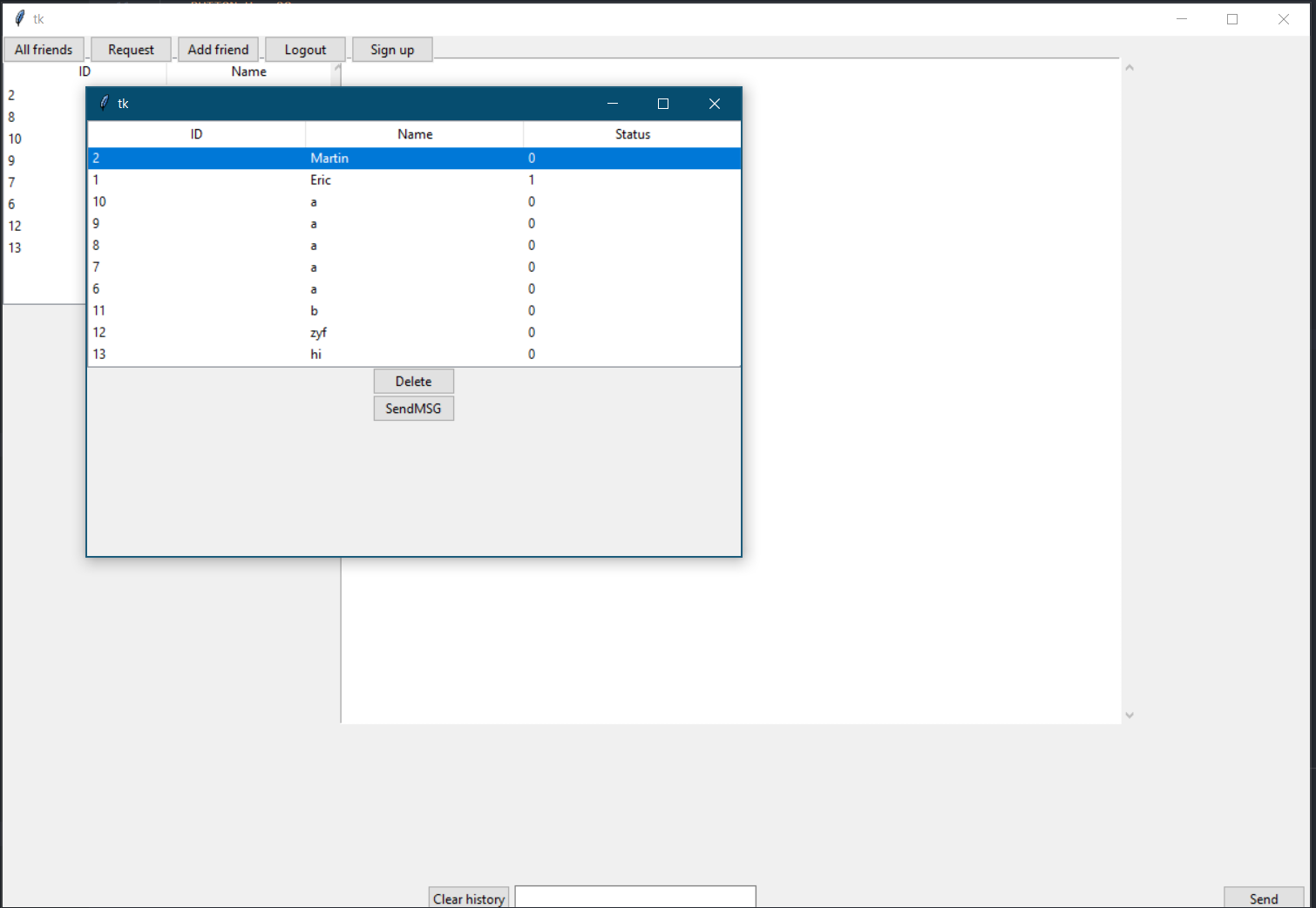
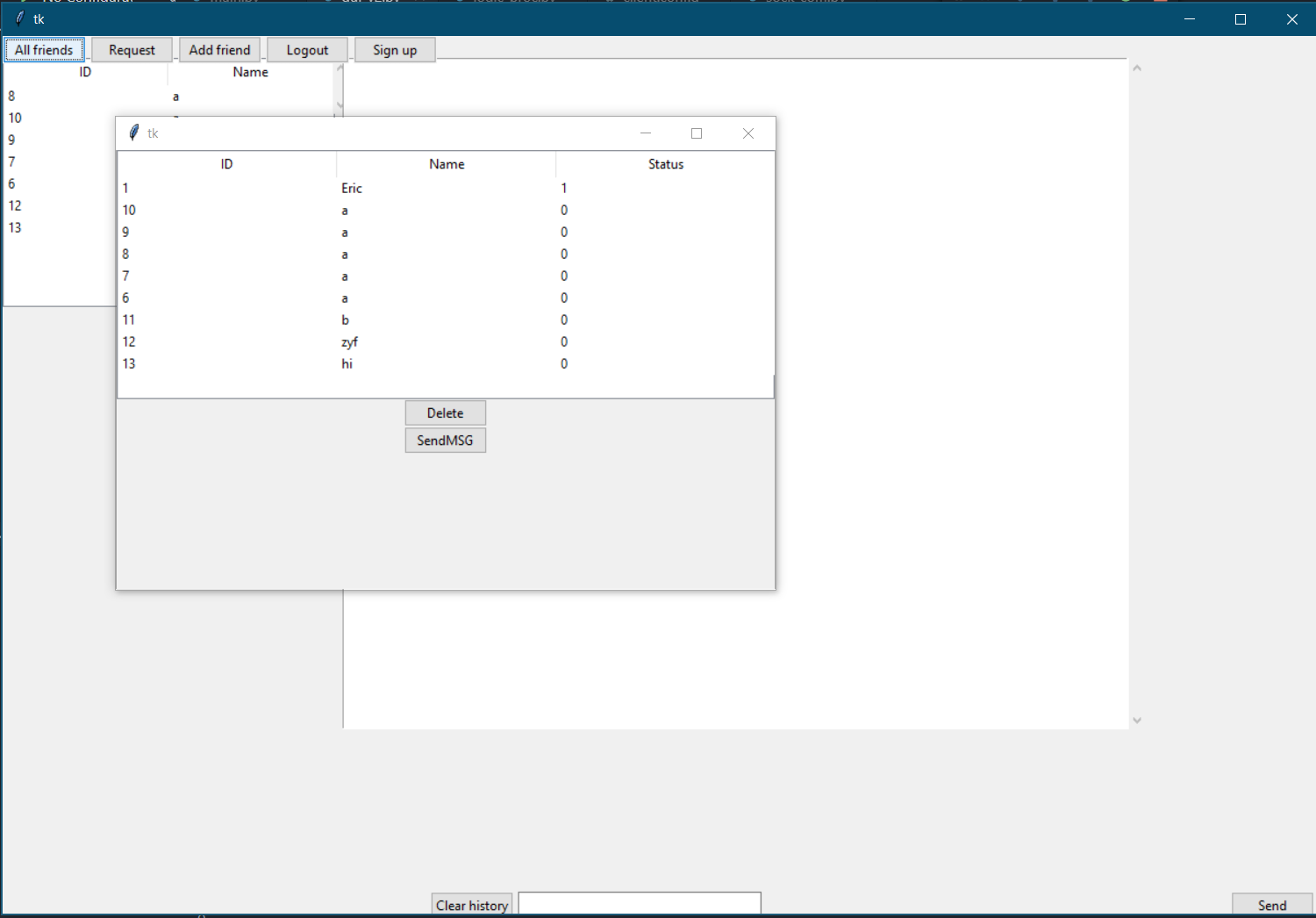
### Add friend which do not exist



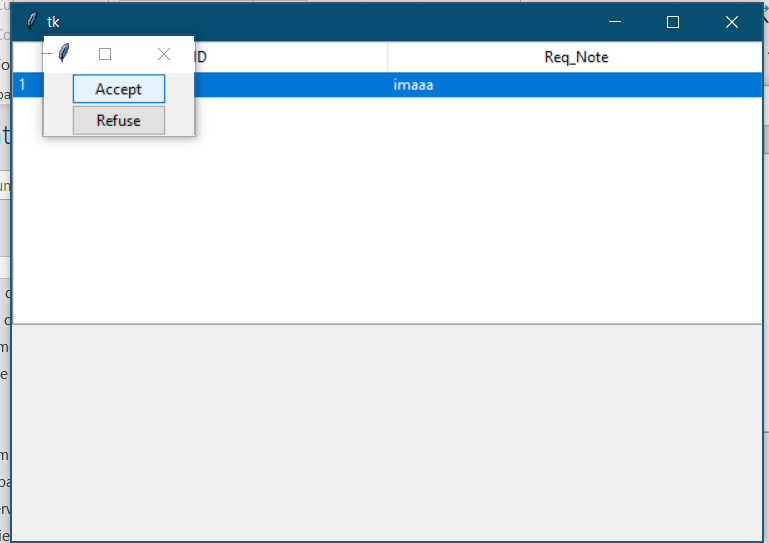
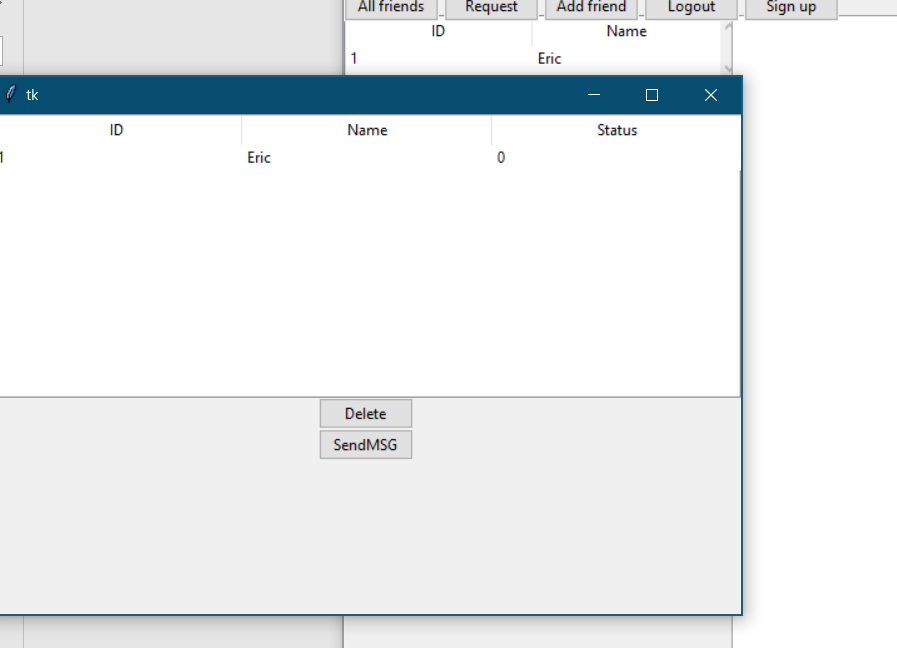
### Add friend that exist

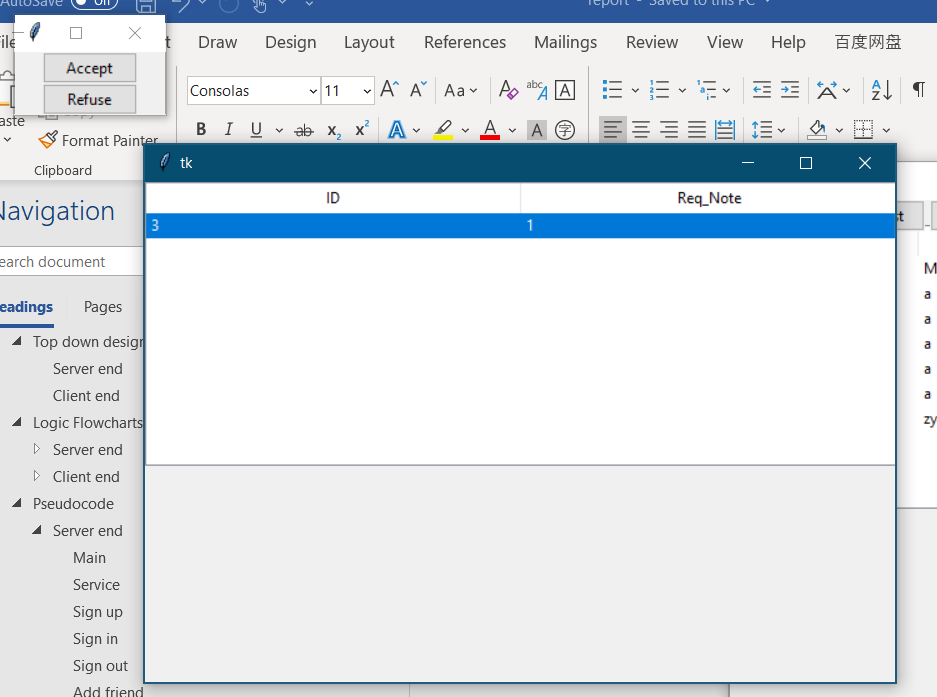
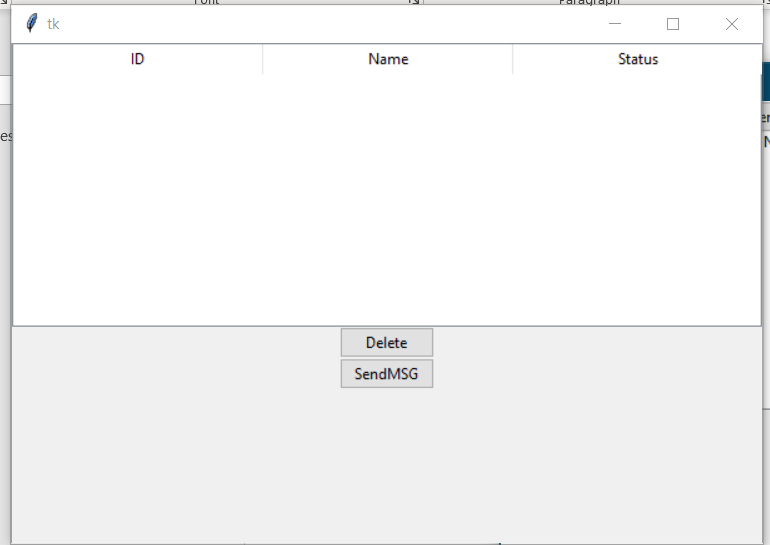
### Delete friend

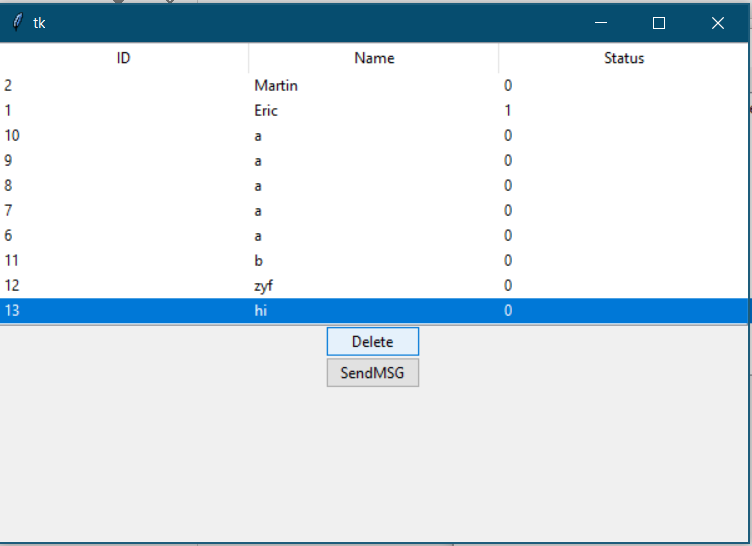
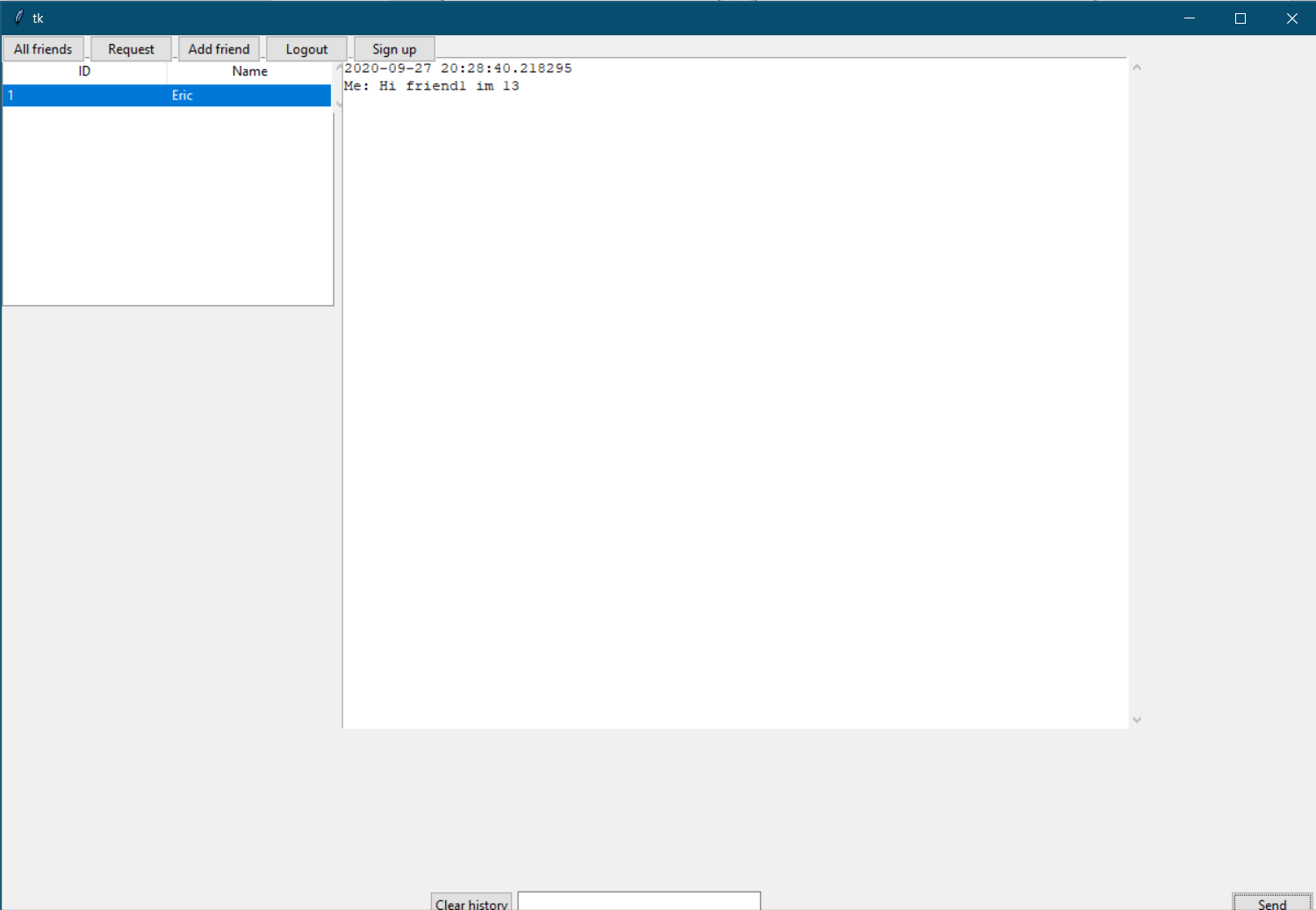
### Accept friend

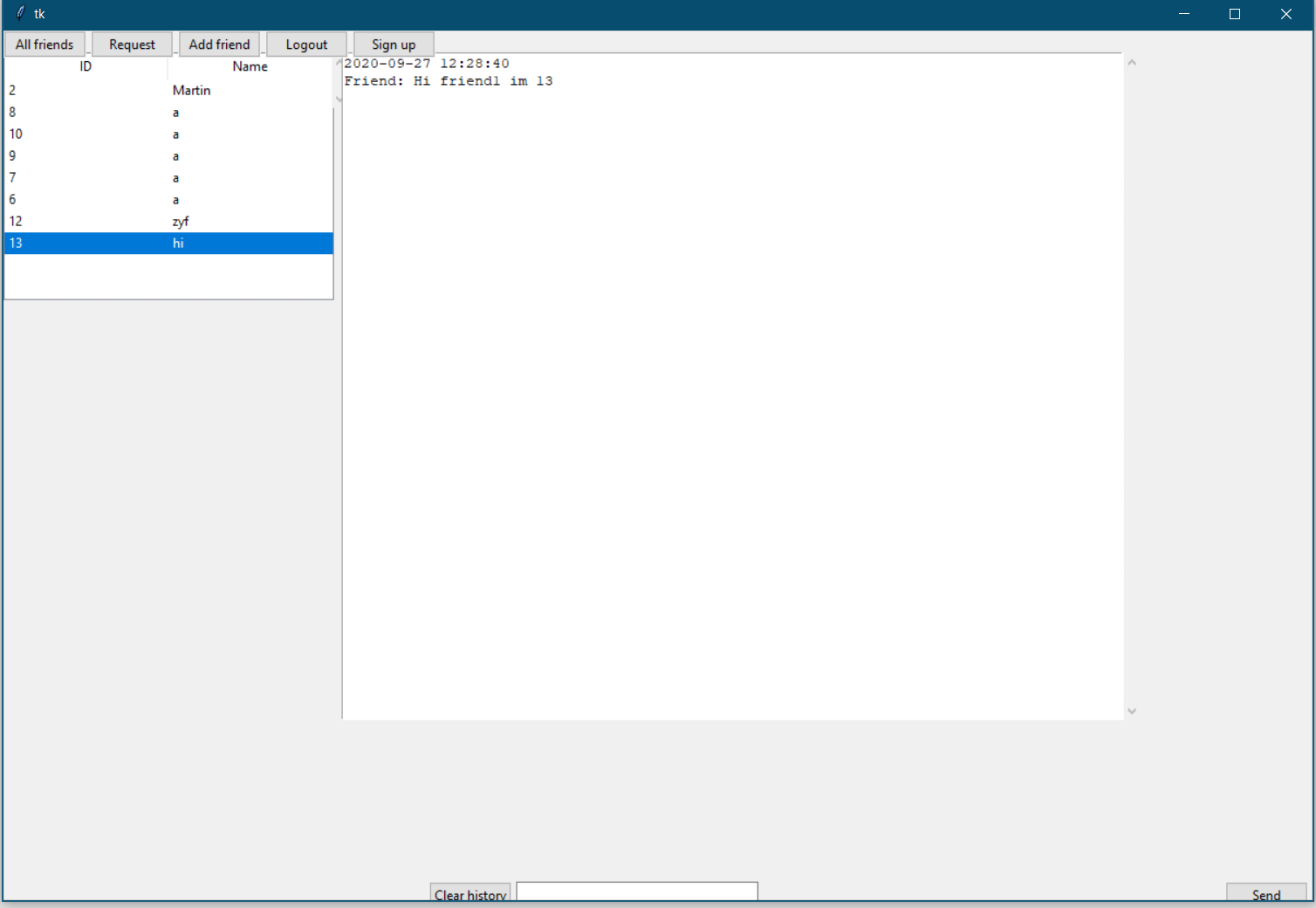
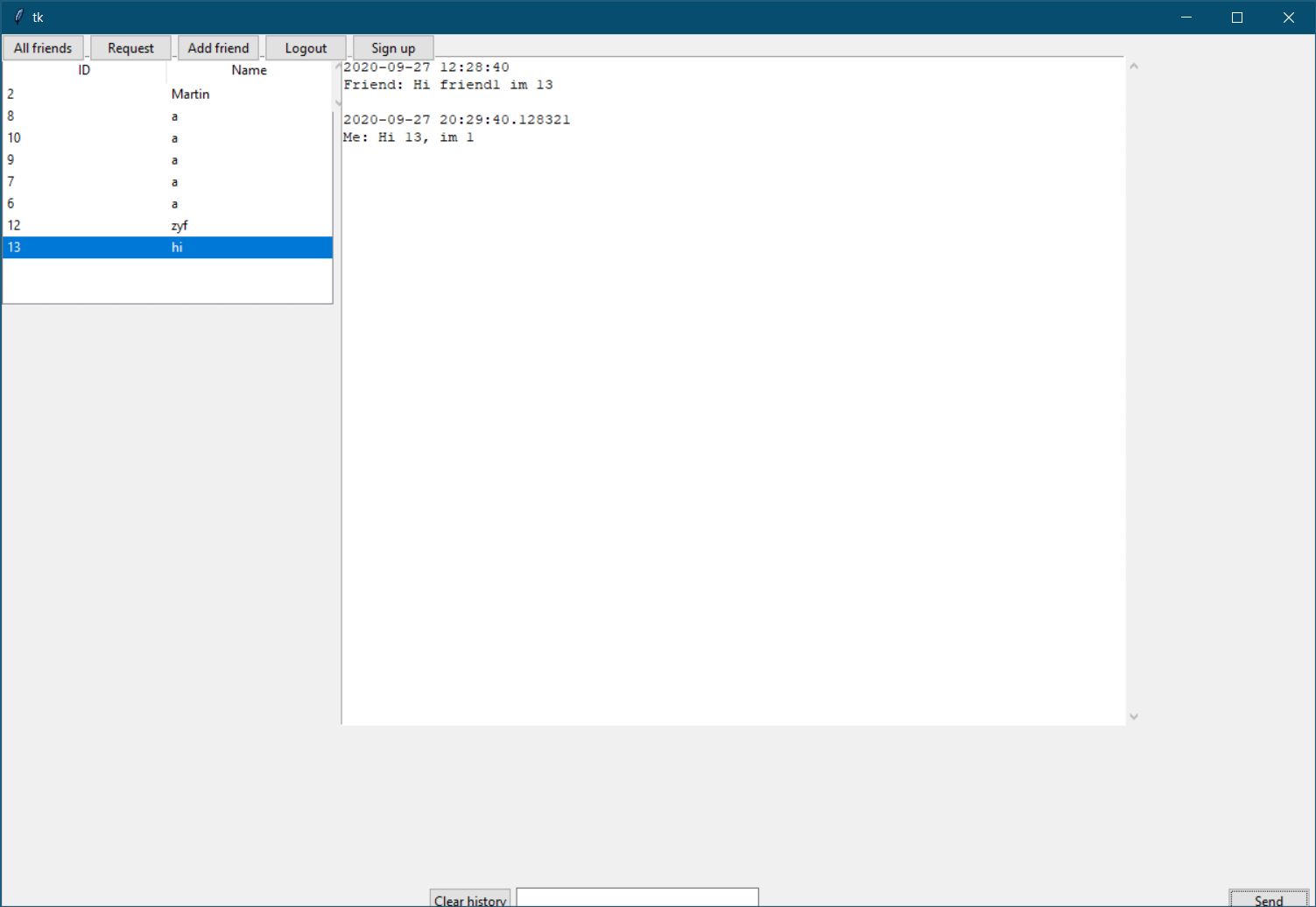
### Refuse friend

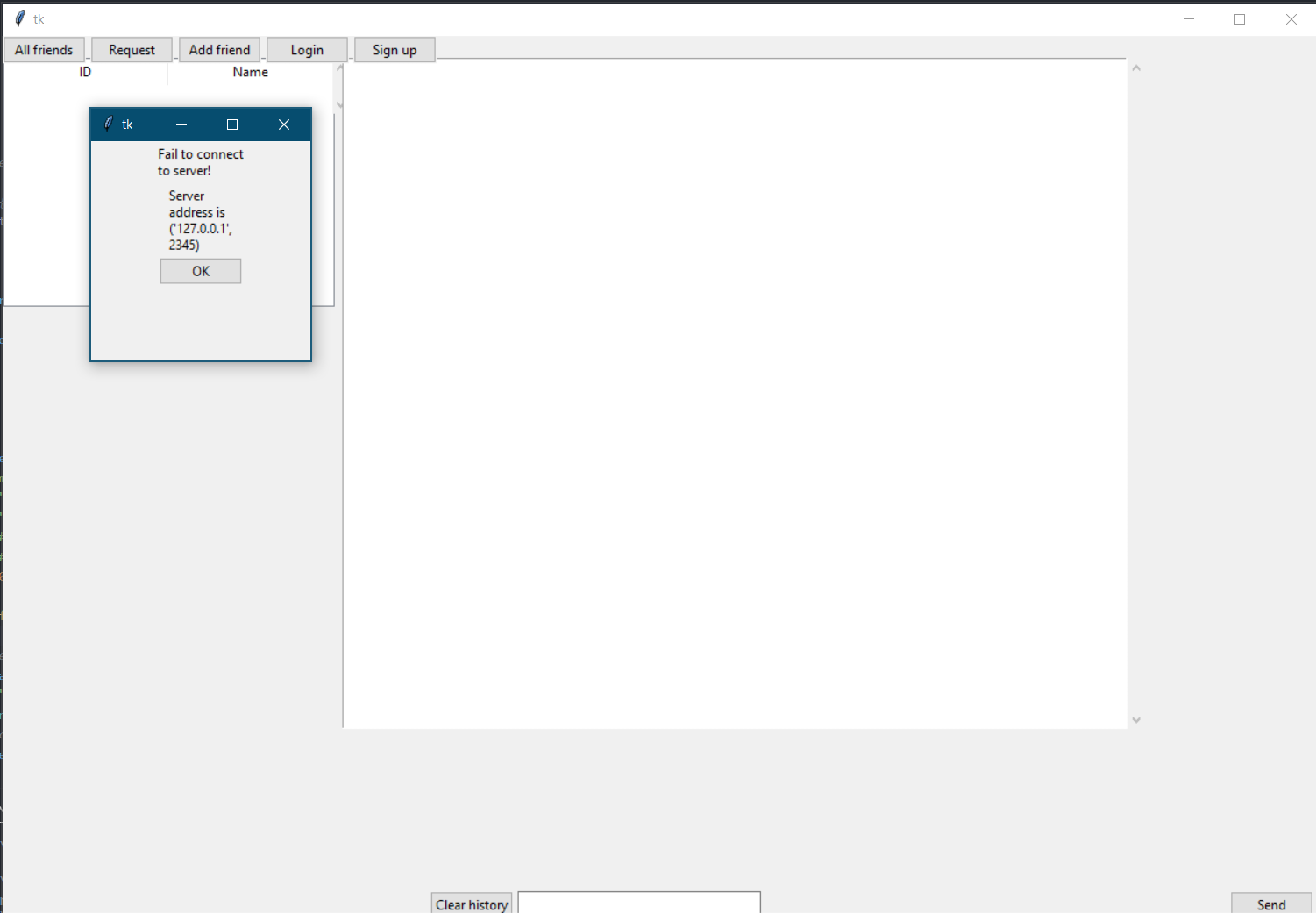
### Send message to friend

### Receive & reply friends

### Run client program when server is not available



# TODOs & points should be improved

|  |  |  |  |
| --- | --- | --- | --- |
| Problem | Why bad | How to improve | Why I make it bad |
| When the receiver is online, the sender still put the data into database and then the receiver fetch the data out. | Frequency access to the database. Increase I/O load. Inefficiency and slow. | Send message directly to receiver through socket if the receiver is online. | Didn’t notice it when designing. Tie Schedule thus no enough time to modify. |
| No scrollbar for friend tree view | If number of items exceed a value, it parts of friends can’t be displayed or clicked. | Add scrollbar. | I did add the code of scrollbar, but it don’t work properly. Too tie schedule so that I didn’t make it. |
| User can’t change username | QQ / WeChat / etc, can do this. | Add change username functionality. | Changing name is not that important. Tie schedule. |
| Ugly interface | Bad GUI design | Design according to standards, spend more time | I have little experience in it.  The schedule is tied. |

# Reflection

In this project, I learnt basic use of database with SQL, and UI design with Tkinter.

# 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>
4. 洪锦魁 . 清华大学出版社 . Python GUI设计：Tkinter菜鸟教程