NawaabChat

Release 2022

Yash Ruhatiya, Aryan Mathe, Yashwanth Reddy Challa

Nov 26, 2022

CONTENTS:

1 FASTCHAT **1**

1.1 DM module 1
1.2 client module 1
1.3 interface module 5
1.4 loadBalancer module 5
1.5 performance module 6
1.6 server module 6
1.7 serverDatabase module 9
1.8 signIn module 9
1.9 signUp module 10

2 Indices and tables **11**

Python Module Index **13**

Index **15**

CHAPTER

ONE

FASTCHAT

1.1 DM module

`DM.handleDM(MY_USERNAME, OTHER_USERNAME, client_sockets, proxy, isGroup)`

To be used when MY_USERNAME sends a message to OTHER_USERNAME. Handles the keywords REMOVE_PARTICIPANT, LEAVE GROUP, and SEND IMAGE.

Parameters

- **[MY_USERNAME]** (*str*) – username of the client who sent DM
- **[OTHER_USERNAME]** (*str*) – username of the receiving client/group
- **[client_socket]** (*str*) – username of the receiving client/group
- **[proxy]** (*ServerProxy*) – proxy server for remote call to receive_message
- **[isGroup]** (*bool*) – whether the receiver is a group

1.2 client module

`client.HEADER_LENGTH = 10`

In order to communicate large messages over a socket, they are broken into multiple smaller messages. The first HEADER_LENGTH characters of the initial message inform the listener how many bytes of data to receive, so that they may stop listening once these many bytes have been received.

`client.addNewDM(MY_USERNAME, username, proxy)`

Adding a new DM to username as requested by MY_USERNAME

Parameters

- **[MY_USERNAME]** (*str*) – username of the client who requested DM
- **[username]** (*str*) – username of the other client

Returns
True for success and False for failure

Return type
bool

`client.checkSocketReady(socket)`

Parameters
[socket] (*socket*) – socket in question

Returns
return the socket if it is ready to be read, otherwise return false

Return type
bool

client.**connectMydb**(*dbName*)

Parameters
[dbName] (*str*) – username of the client whose database we need to connect to

Returns
cursor pointing to that user's local database

Return type
_Cursor

client.**createGroup**(*grpName*, *ADMIN*, *proxy*)

Create a new group by updating the database

Parameters

- **[grpName]** (*str*) – name of the new group
- **[ADMIN]** (*str*) – username of the creator
- **[proxy]** (*ServerProxy*) – the proxy server, used for a remote call to createGroupAtServer

client.**decryptMessage**(*message*, *cur*, *MY_USERNAME*)

Parameters

- **[message]** (*str*) – encrypted message
- **[cur]** (*_Cursor*) – cursor pointing to the user's local database
- **[MY_USERNAME]** (*str*) – username of the client in question

Returns
the decrypted message

Return type
str

client.**getAllUsers**(*MY_USERNAME*)

Parameters
[MY_USERNAME] (*str*) – username of the client whose connections we need to check

Returns
lists DM, group of all the users and groups in MY_USERNAME's connections

Return type
list, list

client.**getOwnPrivateKey**(*sender*)

Get the sender's private key from local database

Parameters
[sender] (*str*) – username of the sender

Returns
sender's private key

Return type
rsa.key.PrivateKey

client.**getOwnPublicKey**(*sender*)

Get the sender's public key from local database

Parameters
[sender] (*str*) – username of the sender

Returns
sender's public key

Return type
rsa.key.PublicKey

client.**getPrivateKey**(*group*, *sender*)

Parameters

- **[group]** (*str*) – group of which the sender is a participant
- **[sender]** (*str*) – username of the sender of the message

Returns
parameters the private key of the group

Return type
tuple

client.**getPublicKey**(*reciever*, *sender*)

Parameters

- **[reciever]** (*str*) – username of the receiver of the message
- **[sender]** (*str*) – username of the sender of the message

Returns
parameters n and e of the public key of the receiver

Return type
list

client.**goOnline**(*username*, *IP*, *PORT*)

Parameters

- **[username]** (*str*) – group of which the sender is a participant
- **[IP]** (*str*) – IP address of the server
- **[PORT]** (*int*) – PORT of the server

Returns
parameters the private key of the group

Return type
tuple

client.**handlePendingMessages**(*client_pending_socket*, *proxy*)

Handles the sending of pending messages. Called every time the client logs in.

Parameters

- **[client_pending_socket]** (*socket*) – socket belonging to the client having pending messages

- **[proxy]** (*ServerProxy*) – proxy server for rpc

client.**isAdminOfGroup**(*grpName*, *MY_USERNAME*)

Parameters

- **[grpName]** (*str*) – name of the group
- **[MY_USERNAME]** (*str*) – admin username

Returns
whether MY_USERNAME is an admin of grpName

Return type
bool

client.**isInConnections**(*MY_USERNAME*, *username*)

Parameters

- **[MY_USERNAME]** (*str*) – username of the client whose connections we need to check
- **[username]** (*str*) – username of the other client

Returns
whether username is in MY_USERNAME's connections

Return type
bool

client.**receive_message**(*data*, *proxy*)

Handle the reception of normal messages as well as the SEND_IMAGE and ADD_PARTICIPANT keywords along with updating the user-side database

Parameters

- **[data]** (*dict*) – dictionary containing all details of the message received
- **[proxy]** (*ServerProxy*) – proxy server for remote calls

client.**replace_quote**(*msg*, *fernet*)

Duplicate all occurences of both double and single quotes

Parameters

- **[msg]** (*str*) – message string
- **[fernet]** (*str*) – fernet string

Returns
return the string with duplicated quotes

Return type
str,str

client.**sendAck**(*client_socket*, *messageId*, *isImage*)

Send an acknowledgement to the server on receipt of a message over socket

Parameters

- **[client_socket]** (*socket*) – the socket that is sending the ack
- **[messageId]** (*int*) – unique id used to identify the message
- **[isImage]** (*bool*) – whether the message is an image or not

client.**unpack_message**(*client_socket*)

Receive as many bytes as specified by the header in units of 16 bytes

Parameters
[client_socket] (*socket*) – the socket that is receiving data

Returns
Dictionary of the received json data. False if any exception occured

Return type
dict/bool

1.3 interface module

interface.**HEADER_LENGTH = 10**

In order to communicate large messages over a socket, they are broken into multiple smaller messages. The first HEADER_LENGTH characters of the initial message inform the listener how many bytes of data to receive, so that they may stop listening once these many bytes have been received.

1.4 loadBalancer module

class loadBalancer.**LoadBalancer**(*ip*, *PORT*, *algorithm='random'*)

Bases: object

Socket implementation of a load balancer.

Flow Diagram: +——————+ +————————————————+ +——————+ | client socket | <==> | client-side socket | server-side socket | <==> | server socket | | <client> | | < load balancer > | | <server> | +——————+ +————————————————+ +——————+

startServers()

Start each server on a separate thread

class loadBalancer.**ServerThread**(*IP*)

Bases: Thread

run()

Serve Forever

loadBalancer.**assignPid**()

Initialize the dicts pid_serverId and serverId_pid, so that we can easily find process id from server id and vice versa

loadBalancer.**getFreeServerId**()

Return the free-est server

Returns
index of the server and corresponding port

Return type
int,int

loadBalancer.**pid_serverId = {}**

dictionary mapping process id to server id

`loadBalancer.runServer(IP, PORT)`

Run a new server process

Parameters

- **[IP]** (*str*) – IP address of network
- **[PORT]** (*int*) – PORT that the server is to be run on

`loadBalancer.serverId_pid = {}`

dictionary mapping server id to process id

`loadBalancer.strategy(algorithm)`

Return a server based on the load balancing algorithm requested

Parameters
[algorithm] (*str*) – 'round-robin'/'random'/'memory'/'cpu'

Returns
index of the server and corresponding port

Return type
int,int

1.5 performance module

`performance.latency()`

Latency in seconds by average of differences between send time and receive time

Returns
the latency in seconds

Return type
float

`performance.throughput(t)`

Throughput by average messages sent/received in total time

Parameters
[t] (*float*) – interval-width

Returns
the throughputs in messages/second

Return type
float,float

1.6 server module

`server.addNewUser(userName, password, n, e)`

Add a new user to the database

Parameters

- **[userName]** (*str*) – username entered by user. This has already been validated by checkUserName

- **[password]** (*str*) – password entered by user
- **[n]** (*int*) – parameter n of the public key of new user
- **[e]** (*int*) – parameter e of the public key of new user

server.**addUserToGroup**(*grpName*, *newuser*)

Add newuser to grpName by an administrator's request

Parameters

- **[grpName]** (*str*) – name of the group to which new participant is to be added
- **[newuser]** (*str*) – new member

Returns
True if successful, False if failure

Return type
bool

server.**checkUserName**(*userName*)

Parameters
[userName] (*str*) – username entered by user

Returns
Whether the username is a registered user (True) or not (False)

Return type
bool

server.**connectToDb**()

server.**createGroupAtServer**(*grpName*, *ADMIN*)

Create a new group by updating the database

Parameters

- **[grpName]** (*str*) – name of the new group
- **[ADMIN]** (*str*) – username of the creator

Returns
the socket corresponding to that username

Return type
socket

server.**getPublicKey**(*username*)

Parameters
[username] (*str*) – username whose public key needs to be extracted from database

Returns
parameters n and e of the public key

Return type
tuple

server.**getSocket**(*username*, *clients*)

Extracts the socket corresponding to a particular username

Parameters

- **[username]** (*str*) – username in question

- **[clients]** (*dict*) – dictionary mapping sockets to usernames

Returns
the socket corresponding to that username

Return type
socket

`server.getUsersList(grpName)`

Get a list of all participants

Parameters
[grpName] (*str*) – name of the group

Returns
list of all participants (each participant is a string)

Return type
list

`server.initialize()`

Initialize the postgreSQL database with the database schema

`server.isValidPassword(userName, password)`

Validate password

Parameters

- **[userName]** (*str*) – username entered by user. This has already been validated by checkUserName
- **[password]** (*str*) – password entered by user

Returns
True if success, False if failure

Return type
bool

`server.receiveAck(client_socket)`

Receive an acknowledgement from the client on receipt of a message over socket

Parameters
[client_socket] (*socket*) – the socket that is sending the ack

Returns
whether the acknowledgement received is meaningful (True) or not (False)

Return type
bool

`server.removeUserFromGroup(grpName, removeuser)`

Remove removeuser from grpName by an administrator's request

Parameters

- **[grpName]** (*str*) – name of the group from which participant is to be removed
- **[removeuser]** (*str*) – participant to be removed

Returns
True if successful, False if failure

Return type
bool

`server.replace_quote(msg, fernet)`

Duplicate all occurences of both double and single quotes

Parameters

- **[msg]** (*str*) – message string
- **[fernet]** (*str*) – fernet string

Returns
return the string with duplicated quotes

Return type
str,str

`server.sendPendingMessages(client_socket, receiverName)`

Send pending messages to reveiverName whenever he/she logs in, over client_socket

Parameters

- **[client_socket]** (*socket*) – the socket that is sending the pending messages
- **[receiverName]** (*str*) – username of the reeceiver

`server.unpack_message(client_socket)`

Receive as many bytes as specified by the header in units of 16 bytes

Parameters
[client_socket] (*socket*) – the socket that is receiving data

Returns
Dictionary of the received json data. False if any exception occured

Return type
dict/bool

`server.updatestatus(isOnline, username)`

Update the status of username on database when they log on/off

Parameters

- **[username]** (*bool*) – username whose status changed
- **[username]** – the current status of username

1.7 serverDatabase module

1.8 signIn module

`signIn.handleSignIn(proxy, IP, PORT)`

Handles sign-in requests

Parameters

- **[proxy]** (*ServerProxy*) – proxy server for calls to checkUserName and isValidPassword
- **[IP]** (*str*) – server IP

- **[PORT]** (*int*) – server PORT

Returns
username and the created socket

Return type
str, socket

1.9 signUp module

signUp.**handleSignUp**(*proxy*, *IP*, *PORT*)

Handles sign-up requests

Parameters

- **[proxy]** (*ServerProxy*) – proxy server for remote calls
- **[IP]** (*str*) – server IP
- **[PORT]** (*int*) – server PORT

Returns
username and the created socket

Return type
str, socket

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

c

client, 1

d

DM, 1

i

interface, 5

l

loadBalancer, 5

p

performance, 6

s

server, 6
serverDatabase, 9
signIn, 9
signUp, 10

INDEX

A

addNewDM() (*in module client*), 1
addNewUser() (*in module server*), 6
addUserToGroup() (*in module server*), 7
assignPid() (*in module loadBalancer*), 5

C

checkSocketReady() (*in module client*), 1
checkUserName() (*in module server*), 7
client
    module, 1
connectMydb() (*in module client*), 2
connectToDb() (*in module server*), 7
createGroup() (*in module client*), 2
createGroupAtServer() (*in module server*), 7

D

decryptMessage() (*in module client*), 2
DM
    module, 1

G

getAllUsers() (*in module client*), 2
getFreeServerId() (*in module loadBalancer*), 5
getOwnPrivateKey() (*in module client*), 2
getOwnPublicKey() (*in module client*), 3
getPrivateKey() (*in module client*), 3
getPublicKey() (*in module client*), 3
getPublicKey() (*in module server*), 7
getSocket() (*in module server*), 7
getUsersList() (*in module server*), 8
goOnline() (*in module client*), 3

H

handleDM() (*in module DM*), 1
handlePendingMessages() (*in module client*), 3
handleSignIn() (*in module signIn*), 9
handleSignUp() (*in module signUp*), 10
HEADER_LENGTH (*in module client*), 1
HEADER_LENGTH (*in module interface*), 5

I

initialize() (*in module server*), 8
interface
    module, 5
isAdminOfGroup() (*in module client*), 4
isInConnections() (*in module client*), 4
isValidPassword() (*in module server*), 8

L

latency() (*in module performance*), 6
loadBalancer
    module, 5
LoadBalancer (*class in loadBalancer*), 5

M

module
    client, 1
    DM, 1
    interface, 5
    loadBalancer, 5
    performance, 6
    server, 6
    serverDatabase, 9
    signIn, 9
    signUp, 10

P

performance
    module, 6
pid_serverId (*in module loadBalancer*), 5

R

receive_message() (*in module client*), 4
receiveAck() (*in module server*), 8
removeUserFromGroup() (*in module server*), 8
replace_quote() (*in module client*), 4
replace_quote() (*in module server*), 9
run() (*loadBalancer.ServerThread method*), 5
runServer() (*in module loadBalancer*), 5

S

sendAck() (*in module client*), 4

sendPendingMessages() (*in module server*), 9
server
    module, 6
serverDatabase
    module, 9
serverId_pid (*in module loadBalancer*), 6
ServerThread (*class in loadBalancer*), 5
signIn
    module, 9
signUp
    module, 10
startServers() (*loadBalancer.LoadBalancer method*), 5
strategy() (*in module loadBalancer*), 6

T

throughput() (*in module performance*), 6

U

unpack_message() (*in module client*), 4
unpack_message() (*in module server*), 9
updatestatus() (*in module server*), 9