SecureFastChat Server Program

Khushang Singla, Mridul Agarwal, Arhaan Ahmad

Nov 26, 2022

CONTENTS:

CHAPTER

ONE

SERVER-SIDE

1.1 DatabaseRequestHandler module

`DatabaseRequestHandler.`**`checkuid`**(*uid*)

`DatabaseRequestHandler.`**`setpass`**(*uid*, *pwd*)

`DatabaseRequestHandler.`**`check_login_uid`**(*uid*)

`DatabaseRequestHandler.`**`check_login_pwd`**(*uid*, *pwd*)

`DatabaseRequestHandler.`**`getUnsentMessages`**(*uid*)

1.2 Message module

`class` `Message.`**`Message`**(*conn_socket*, *status*, *request*, *sel*, *loggedClients*, *lbsock*, *selector*, *encrypt=True*)

Bases: `object`

This is the class to handle Encryption of messages. The format in which the message is sent to client is determined in this class

Parameters

- **`socket`** (*`socket.socket`*) – The socket used for connection with Server
- **`request_content`** (*`dict`*) – Content to include in the request to send to server
- **`_data_to_send`** (*`bytes`*) – Contains the data to send to the server
- **`_recvd_msg`** (*`bytes`*) – Content received from server is stored here

Constructor Object

Parameters

- **`conn_socket`** (*`socket.socket`*) – Socket which has a connection with client
- **`request`** (*`str`*) – Content to send to server
- **`encrypt`** – Whether the received data would be encrypted

`classmethod fromSelKey`(*selectorKey*, *loggedClients*, *lbsock*, *selector*, *encrypt=True*)

Custom constructor to initialise a message given just the selector key

Parameters **`selectorKey`** (*`SelectorKey`*) – the selector key containitnall the data

Returns Message

Return type *Message*

`_send_data_to_client`()

Function to send the string to the client. It sends content of _send_data_to_client to the client.

`encrypt`(*data: bytes*) → bytes

summary

Parameters **`data`** (*`bytes`*) – the data to encrypt

Returns Encrypted Message

Return type bytes

`_recv_data_from_client`(*size: int*, *encrypted=True*) → int

Function to recv data from client. Stores the bytes recieved in a variable named _recvd_msg.

Parameters

- **`size`** (*`int`*) – the size of data to receive
- **`encrypted`** (*`bool, optional`*) – Whether the incoming data is supposed to be encrypted, defaults to True

Returns code to see if something works. Returns -1 if the connection closed

Return type int

`_send_msg_to_reciever`(*rcvr_sock*)

Function to send message to a reciever

Parameters **`rcvr_sock`** (*`Socket`*) – The socket to which to send

`_json_encode`(*obj*, *encoding*)

Function to encode dictionary to bytes object

Parameters

- **`obj`** (*`dict`*) – dictionary to encode
- **`encoding`** (*`str`*) – Encoding to use

Returns Encoded obj

Return type bytes

`_json_decode`(*obj*, *encoding*)

Function to decode bytes object to dictionary

Parameters

- **`obj`** (*`bytes`*) – Encoded json data
- **`encoding`** (*`str`*) – Encoding used

Returns Decoded json object

Return type json

`processTask`()

Processes the task to do

Returns Returns int to represent result of the process. The details of return values are given in the corresponding functions handling the actions.

Return type int

`_handle_leave_group_request`(*grp_uid*, *uid*)
Function to remove member from group based on leave request

Parameters

- **`guid`** (*`str`*) – Group to remove from
- **`uid`** (*`str`*) – user to remove

`_rem_grp_mem`(*grp_uid*, *uid*)
Removes a member from group, checking the validity of group id, user id and connecting with the database

Parameters

- **`grp_uid`** (*`str,`*) – group id,
- **`uid`** (*`str`*) – user id,

`_send_grp_message`(*grp_uid*, *msg_type*, *content*)
Send messages in a group

Parameters

- **`grp_uid`** (*`str`*) – Id of the group in which message is to be sent
- **`msg_type`** (*`str`*) – Type of message to be send, text or file object
- **`content`** (*`str`*) – message to be sent

`_add_grp_mem`(*grp_uid*, *new_uid*, *user_grp_key*)
Function to add a new member in the group

Parameters

- **`grp_uid`** (*`str`*) – id of the group in which member is to be added
- **`new_uid`** (*`str`*) – user id of the new user which is to be added in the group
- **`user_grp_key`** (*`str`*) – Public key of the group

`_create_grp`(*grp_uid: str*, *grp_key: str*)
Creates a new group

Parameters

- **`grp_uid`** (*`str`*) – name of the group
- **`grp_key`** (*`str`*) – the key used for encrypting messages

Returns 1 if there was an error, 0 otherwise

Return type int

`_send_group_key`(*grp_name: str*, *username: str*) → None
Sends a json response containing the group key for a particular user, which can be decrypted by only that user to get the actual private key

Parameters

- **`grp_name`** (*`str`*) – name of the group
- **`username`** (*`str`*) – name of the user

`_send_msg`(*rcvr_uid*, *msg_type*, *content*, *grp_uid=None*, *sender=None*, *timestamp=None*, *save=False*)
Sends messages to the specified user

Parameters

- **`rcvr_uid`** (*`str`*) – User ID of the reciever client
- **`msg_type`** (*`str`*) – Type of message, text or file
- **`content`** (*`str`*) – Encrypted message to be sent
- **`grp_uid`** (*`str`*) – GroupId in case of group message
- **`sender`** (*`str`*) – name of the message sender, in case of group chat, it is grp_id::user_id, otherwise it is username of self
- **`save`** (*`bool`*) – whether to save if the receiver is not directly connected to the server

`_send_rcvr_key`(*rcvr_uid: str*) → None
Gets the public key of a given user

Parameters **`rcvr_uid`** (*`str`*) – User id of the user whose public key is requested

`keyex`() → str
Does key exchange. First waits for request from the client, then sends a response with its own public key. Returns a string containing the public key of the client

Returns public key of the client, encoded to base64

Return type str

`_process_login`(*username*, *password*)
Processes Login Request On successful login sends pending messages

Parameters

- **`username`** (*`str`*) – Username of the Client to be logged in
- **`password`** (*`str`*) – Password of the Client to be logged in

`_login_failed`() → bytes
Returns the response to send after a failed login attempt

Returns reponse after failed login

Return type bytes

`_login_successful`() → bytes
Returns the response after a succesful login

Returns response after a succesful login

Return type bytes

`_signup_failed`() → bytes
Returns the response to send after a failed signup attempt

Returns reponse after failed signup

Return type bytes

`_successfully_signed_up`() → bytes
Returns the response to send after a succesful login attempt

Returns reponse after succesful login

Return type bytes

`_process_signup_uid`(*uid: str*) → None
Processes Signup Request by validating if requested Uid already exists or not

Parameters **`uid`** (*`str`*) – User ID of new user

`_invalid_uid_type``()`
Returns the response to send if the username is of the wrong type

Returns protoheader + a json header which does not containt the availability key

Return type bytes

`checkValidityOfUID``(`*uid*`)`
Function to check if the uid is valid. A valid uid is one which has only a-z,A-Z,0-9,_ characters

Parameters **`uid`** (*`str`*) – User id to check for

Returns Return True if valid

Return type bool

`_signup_uid_not_available``()` → bytes
Returns the response to send if the username is already taken

Returns protoheader + a json header saying that the availability is 0

Return type bytes

`_signup_uid_available``()` → bytes
Returns the response to send if the username is free

Returns protoheader + a json header saying that the availability is 1

Return type bytes

`_process_signup_pass``(`*password: str*, *e2eKey: str*`)` → None
Process the command for signing up the user and storing the password

Parameters **`password`** (*`str`*) – The password

`isOnline``()` → bool
Returns if the user is online

Returns Is the user online

Return type bool

`get_uid_selKey``()` → Tuple[str, selectors.SelectorKey, bool]
Helper function to get the username and selectorkey

Returns A tuples containing the username and selectorkey and whether this message led to a new login

Return type tuple[str, selectors.SelectorKey, bool]

1.3 db module

`db.`**`deleteOldMessages`**`()`
Delete messages older than 7 days (can change later) This is called when we add some new message to the db

`db.`**`checkIfUsernameFree`**`(`*username: str*`)` → bool
Check if a given username is already in use

Parameters **`username`** (*`str`*) – The username to check

Returns Whether the name is in use or not

Return type bool

`db.`**`createUser`**(*username: str*, *password: str*, *e2ePublicKey: str*) → bool
Adds a user with the given username and password to the database. Assumes that the checkIfUsernameFree has already been called before. We hash the password here. Returns true if the user generation happened without any error

Parameters

- **`username`** (*`str`*) – username
- **`password`** (*`str`*) – password (hashed)

Returns Whether the user creation happened succesfully

Return type bool

`db.`**`db_login`**(*username: str*, *password: str*) → bool
Checks if a given username password pair is present in the db

Parameters

- **`username`** (*`str`*) – username
- **`password`** – password

Returns True if the user is authenticated by this

Return type bool

`db.`**`storeMessageInDb`**(*sender: str*, *receiver: str*, *message: str*, *timestamp: str*, *content_type: str*)
stores the encrypted message in the database, in case it is not possible to send them the message directly

Parameters

- **`sender`** (*`str`*) – sender username
- **`receiver`** (*`str`*) – receiver username
- **`message`** (*`str`*) – the enecrypted message

`db.`**`getE2EPublicKey`**(*user: str*) → str
Takes the username and outputs the e2e public key of that user

Parameters **`user`** (*`str`*) – username of the user

Returns the e2ekey in base64

Return type str

`db.`**`getUnsentMessages`**(*username: str*) → list
Get the unsent messages to a particular user, ordered by timestamp

Parameters **`username`** (*`str`*) – username of receiver

Returns list of tuples containing the data about the messages

Return type list

`db.`**`checkIfGroupNameFree`**(*groupName: str*) → bool
Check if a given groupname is already in use

Parameters **`groupName`** – The groupname to check

Returns Whether the name is in use or not

Return type bool

`db.`**`createGroup`**(*groupname: str*, *key: str*, *creatorUsername: str*, *creatorE2Ekey: str*) → bool
Creates a new group in the database

Parameters

- **`groupname`** (*`str`*) – name of the group
- **`key`** (*`str`*) – key used for encrypting messages for this group. Note that this is encrypted by the creators e2e encrypted key
- **`creatorUsername`** (*`str`*) – username of creator

Returns _description_

Return type bool

`db.`**`isGroupAdmin`**(*groupName: str*, *username: str*) → bool

Checks if a particular user is the admin of a group

Parameters

- **`groupName`** (*`str`*) – name of the group
- **`username`** (*`str`*) – username to check

Returns whether username is an admin of the group

Return type bool

`db.`**`addUserToGroup`**(*groupname: str*, *username: str*, *usersGroupKey: str*)

`db.`**`getGroupMembers`**(*groupname: str*) → List[str]

`db.`**`getUsersGroupKey`**(*groupname: str*, *username: str*) → Tuple[str, str]

`db.`**`removeGroupMember`**(*groupname: str*, *username: str*)

Remove a user from the db of a group

Parameters

- **`groupname`** (*`str`*) – name of the group
- **`username`** (*`str`*) – username to remove

`db.`**`deleteMessage`**(*receiver: str*, *sender: str*, *content: bytes*)

Delete a particular message from the database

Parameters

- **`receiver`** (*`str`*) – username of the receiver
- **`sender`** (*`str`*) – username of the sender
- **`content`** (*`bytes`*) – content of the message

1.4 startServer module

`startServer.`**`accept`**(*sel*, *sock*)

Function to accept a new client connection

`startServer.`**`doKeyex`**(*conn*, *selkey*)

`startServer.`**`service`**(*key*, *mask*, *HOST*, *PORT*)

`startServer.`**`send_lb_logout_info`**(*uid*)

Tells the ;pad balancer about which user has logged out

Parameters **`uid`** (*`str`*) – user id of the user that has logged out,

`startServer.`**`send_lb_new_login_info`**(*uid*, *HOST*, *PORT*)
Tells the load balancer about which user has logged in

Parameters

- **`uid`** (*`str,`*) – User id of the user that has logged in,
- **`HOST`** (*`str,`*) – host of the load balancer,
- **`PORT`** (*`int`*) – port of the load balancer,

`startServer.`**`startServer`**(*pvtKey*, *HOST='127.0.0.1'*, *PORT=8000*)
Server starts and connects to the socket of the load balancer

Parameters

- **`pvtKey`** (*`nacl.public.PrivateKey,`*) – private key of the server,
- **`HOST`** (*`str,`*) – host of the loadbalancer,
- **`PORT`** (*`int`*) – port of the loadbalancer,

1.5 loadbalancer module

`loadbalancer.`**`accept`**(*sel*, *sock*)
Function to accept a new client connection

`loadbalancer.`**`registerServer`**(*addr: Tuple[str, int]*, *index: int*)
Registers a server with listening socket of the load balancer

Parameters

- **`addr`** (*`Tuple[str, int]`*) – tuple of host, port
- **`index`** (*`int`*) – index of the server

`loadbalancer.`**`serverComm`**(*key*, *mask*)
Process the communication between server and loadbalancer

Parameters **`key`** (*`selector key`*) – server key

1.6 lb_msg module

`class` `lb_msg.`**`LoadBalancerMessage`**(*socket*, *sel*, *strategy='random'*)
Bases: `object`

Class for conversation over sockets

Parameters

- **`socket`** (*`socket.socket`*) – Connection Socket to talk on
- **`strategy`** (*`str`*) – Loadbalancing algorithm to use
- **`_msg_to_send`** (*`bytes`*) – the message to send to client
- **`sel`** –

Constructor object

Parameters **`socket`** (*`socket.socket`*) – Connection socket

`_json_encode`(*obj*, *encoding='utf-8'*)
Function to encode dictionary to bytes object

Parameters

- **`obj`** (*`dict`*) – dictionary to encode
- **`encoding`** (*`str`*) – (Optional)Encoding to use

Returns Encoded obj

Return type bytes

`_getAvailableServerID`(*strategy*)
This function finds the server with least number of connections and returns the corresponding id.

Returns The id of server to use

Return type int

`_getLeastConnServer`()
This function implements the least connection server Distribution Returns the server_id with least number of connections

Returns server id

Return type int

`_getRoundRobinServer`()
This function implements the round robin server Distribution

Returns server id

Return type int

`_getRandomServer`()
This function implements the random server Distribution

Returns server id

Return type int

`_getLsockHostPortFromID`(*server_id*)
Get the listening socket details from id

Returns Listening socket details as (host,port) tuple

Return type tuple(str,int)

`_getSocketFromID`(*server_id*)
Get the listening socket from id

Returns Listening socket details as (host,port) tuple

Return type tuple(str,int)

`_prepareMessage`(*json_header*, *content=b''*, *encrypt=True*)
Prepare the string to send from header and content and encrypt by default

Parameters

- **`json_header`** (*`dict`*) – Json Header with important headers. content-len and byteorder are added to header in the function
- **`content`** (*`bytes`*) – The content of the message(Optional,default = b”)
- **`encrypt`** (*`bool`*) – If encryption is to be done(Optional, default = True)

readFromSocket()
: Returns the json_header and content of the message recieved

processTask()
: Processes and redirects requests

_send_data_to_client()
: Sends the content of _msg_to_send through the socket

processClient()
: Function to redirect client

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX