fort secure chat – documentation

This document outlines the unique APIs present in my implementation of a P2P networking service. APIs already documented in the Application Protocol are ***not*** *included, as per specification.*

HTML-serving functions

/index

This function connects and serves "index.html", the initial landing page of the client. A prompt is displayed on the screen based on the user's current login status – if the login failed due to incorrect credentials, a prompt will display accordingly. This page is also redirected to on logoff, so a visual prompt will display upon successfully logging off.

/home

This function serves "home.html", containing the main messaging client. This function is in charge of serving the page and replacing data with important Python-controlled data, displaying output from functions such as /getList, /listAllUsers and /getChatConvo. This helps display information about who is currently online and the correct chat logs.

/viewProfiles

This function serves "myProfile.html", containing profile display, profile search and the ability to view and edit one's own profile. The function replaces data in HTML with the /displayProfile function.

/files

This function serves "files.html", which has the interface for sending files and viewing received files. Functions such as /displayFile and /getList are called here to provide visual output.

/logs

This function services "logs.html", which contains a user interface for the user to search message logs of sent and receives messages. The user can also search for messages on this page.

/twoFA & /twoFAcode

These functions prompt the user for a 6-digit Google Authenticator code. A form will be submitted which calls another exposed function /validate2FA (see next), to verify the value entered. /twoFA is the landing page for new users, so a QR code is displayed to scan in a new secret as well as provide input for the user's code.

Back-end functions
All parameters are required, unless otherwise stated.

/validate2FA
Description: This API is called when a user enters their six-digit code from Google Authenticator for authentication on either the /twoFA or /twoFAcode page. The function first checks whether the user is new or returning – if the user is new, a new base32 string is created based on the user's username using encrypt.py's /generateBase32 function. If the user is returning, a call to the user_credentials table is made to retrieve the secret given to the user upon his/her initial login – this means the user does not have to re-scan another QR code and merely has to enter his/her current TOTP again.
The function then validates the entered code with the algorithm found in the *encrypt* helper file, based off TOTP (Time-based One Time Password). The function also checks for the edge case where the most significant digit in the code is a 0, which causes the server-side code to return a 5-digit number. The most significant digit is removed before comparison to circumvent this.
Parameters: code, string
Returns: None – on success, HTTPRedirect to /home is called. On failure, HTTPRedirect to /index is called with the 'login failed' flag raised.

/blackList
Description: This API is called if the blacklisting form present on /home is submitted. The function manages the insertion and deletion from the blacklist table, based on the 'choice' selected by the user and the username chosen to be blocked. Implementation on how communication is blocked is not found here – refer to the non-exposed /limitReached function instead.
Parameters: choice, a string (limited by front-end to either "Block" or "Unblock")
username, a string
Returns: None – on success, the target user is blocked/unblocked and an HTTPRedirect to /home is called. On failure, an error message is displayed.

/setStatus
Description: Called internally in /viewProfiles to set the status of the current user by updating the appropriate SQLite table. The user has predefined options for status as determined by the front end- Online, Away, Idle, Do Not Disturb, and Offline.
Parameters: status, a string
Returns: None - /setStatus is called inside another function (/updateProfile) which is responsible for returning.

/grabStatus
Description: Is called internally to grab other people's profile information. A JSON-encoded request to the target user's /getStatus function is executed, and the JSON-decoded response is stored in the user_status SQLite table.
Parameters: username, a string
Returns: Numerical Error Code

/getChatConvo
Description: This function is called internally to populate the chat window with the appropriate messages in /home. In terms of our database, the msg table contains all messages, sent or received. getChatConvo's primary role is to sort through these messages and select a conversation between the logged in user and the target user passed in through the 'username' string, in chronological order with the most recent messages at the bottom. getChatConvo is also responsible for converting Markdown text if messages are received in Markdown, and also to apply some CSS-defined styling to give a chat-bubble like appearance to sender and receiver, similar to modern messaging clients. Lastly, getChatConvo's username string is optional and pre-assigned to 'entry', a fake user which allows the client to run a tutorial-like introduction to fort secure chat.
Parameters: username, *optional* string
Returns: conversation, a string of formatted messages

/sendMessage
Description: The 'opposite' to receiveMessage. This function calls other people's receiveMessage. The function takes in information about the target user, looks up their ip and port information, and sends a JSON-encoded request to the target user's /receiveMessage. A response is then read and the appropriate numerical error code is returned. If the transmission was successful, the message is stored in the local msg table. Lastly, sendMessage has a try/except clause to first attempt to send a message with markdown arguments encoded, but will revert to not using markdown arguments if the request was unsuccessful.
Parameters: destination, a string
message, a string
markdown, an integer (0 or 1)
Returns: Numerical Error Code, HTTPRedirect to /home

/grabProfile
Description: The 'opposite' of getProfile. This function calls other people's getProfile. The function takes in a target username, and sends a JSON-encoded request to the target user's /getProfile. The response is received and an appropriate numerical error code is returned. If successful, the response is JSON-decoded and stored in the profiles table in the database.
Parameters: profile_username, a string
sender, a string
Returns: The result of /displayProfile

/displayProfile
Description: This function is in charge of populating the profiles window with the appropriate profile information, based on the user string passed in. Like /getChatConvo, the username input parameter is optional to run a spoof user called 'entry', allowing the function to be initially called with static text. The static text in this case includes the user's personal profile – displaying the logged in user's picture, name, position, description and location. The static text also has a form to edit the user's profile and set his/her status.
If another profile is clicked on the UI, /displayProfile assists /grabProfile in displaying the target user's profile information.
Lastly, /displayProfile is also called by the front-end's search module. An AJAX request is sent to search for a specific user profile, which calls /displayProfile to do so.
Parameters: user, optional string
Returns: A formatted HTML string (replaced in /viewProfile)

/updateProfile
Description: This function is called internally to update the logged in user's profile details. The input parameters passed in are updated in the profiles SQite table.
Parameters: fullname, string
position, string
description, string
location, string
picture, URL string
status, string
Returns: A call to /setStatus and an HTTPRedirect to /viewProfiles

/sendFile
Description: The opposite of receiveFile. This function calls other user's receiveFile in order to send a base64-encoded file to them. The request is sent in JSON format and the response is read. If the response is successful (error code 0 detected), an HTTPRedirect is raised to /files.
Parameters: destination, string
File, a base64 encoded string
Returns: HTTPRedirect to /files

/displayFile
Description: Displays the contents of the files database. Only shows received files, as sent files are not stored in the database. The function retrieves all files sent and formats them in HTML. In this way /displayFile also contains the implementation for the embedded media player, as different HTML tags are called for different content_types. Image-based content is displayed with <img src>, Video-based content is displayed with <video> and Audio-based content with <audio>. Lastly, if the mimetype does not conform to any of these above formats (e.g. a PDF or executable), a download link is provided.
Parameters: None
Returns: files, a HTML formatted string

/displayFileForm
Description: Created so sending files could run through an AJAX request on the JQuery side. The function contains a form which when submitted will call /sendFile.
Parameters: None
Returns: None, used in conjunction with /sendFile to separate concerns

/displayReceivedMessage
Description: Called in message logs to display all messages ever received by the logged in user, by traversing through the message table. This function also calls on helper functions in time_formatting.py to retrieve human-readable timestamps based on the epoch time integer produced by python's int(time.time()) call, and message timeStamps in comparison to the current time.
Parameters: None
Returns: messages, a HTML formatted string

/displaySentMessage
Description: Similar to /displayReceivedMessages, but for sent messages instead.
Parameters: None
Returns: messages, a HTML formatted string

/searchMessage
Description: This function is called upon a search query being submitted in the message logs screen, /logs. This function takes in a msg_phrase and returns all messages, sent or received, containing the search query. The search engine is case insensitive, and does not require an exact word match. This is the most practical for message searching purposes.
Parameters: msg_phrase, string
Returns: messages, a HTML formatted string

Non-exposed Important APIs
Please refer to the time_formatting.py, encrypt.py and db_calls.py for additional (non-exposed) helper functions that help with the project's implementation.

/resetTimer
Description: This non-exposed function is called at the beginning of server setup to start a thread that resets a global variable every sixty seconds.
Parameters: None
Returns: None

/LimitReached
Description: This function is called as the very first thing in /receiveMessage, /receiveFile, /getProfile and /getStatus. This is because these are calls that other people make to my client, and so there is risk involved with server instability should many users concurrently call my API. To assist with rate limiting, /LimitReached increments the global variable mentioned in /resetTimer everytime the function is called. The function also checks whether the global variable has exceeded the value of 14 – if it has done so, True is returned to alert all other functions to stop serving content to other users, until /resetTimer is inevitably called and the global variable is reset.
This function also checks through the blacklist mentioned in /blackList to block communication with users on this list.
Parameters: None
Returns: Boolean, true (block communication) or false (allow communication)

/reportThreaded
Description: This function uses data found in the user credentials table (which is added on /report) to continually call /report to the login server. This function is called every 30 seconds using a separate thread.
Parameters: None
Returns: Numerical Error Code

/logoffForced
Description: This function is called in a 'finally' clause in the __main__ section of login.py. Using the same user credentials table mentioned in /reportThreaded, this function aims to forcefully log out the user on application exit – that is, when the server crashes or is exited in any way. In conjunction with /reportThreaded, this means that users get extremely accurate data on whether the client is logged in or not, instead of waiting for a 5 minute timer to expire or log off to be manually called.

/getTotpToken
Description: This helper function is found in encrypt.py and simulates the algorithm for TOTP, according to RFC6238 documentation. The function takes in a secret (which in itself is randomly generated with the current username as the seed) and outputs a 6-digit 'token'. This token is then compared with user input on the /twoFA and /twoFAcode pages, where users enter their six-digit code from the Google Authenticator app. As Google Authenticator employs the same algorithm, provided the same initial secret (which our implementation stores, see /validate2FA) the output token is identical:
Parameters: secret, a string
Returns: token, an integer