Phase 1:

This phase has been made in multiple files. Including : main.c , cJSON.c , userMenu.c , mainMenu.c , chatMenu.c.

Main.c:

In main, there is the WSAStartup function which will tell the user if a usable Winsock DLL can be found or not. The system function changes the background color in CMD. And then the userMenu function is launched.

userMenu.c:

In userMenu, we have an infinite loop (while(1)) followed by a scanf, which will act as a menu, and the user can press buttons from 1 to 5, to enter the needed section. According to the number the user enters, it will have a unique response.

1 : Register 2 : Login 3: Clear Screen Every Time 4 : Undo '3' 5 : Exit

In "Register" (line 35), the program asks for a username and password. Then with the help of the sprintf function builds the message which will be sent to the server in the correct format. Since the server is stateless, it asks for connection with the connect function and if it was accepted, it will send the string with the send function. Next it will receive the answer, and if the type value was "Successful" it will tell the user to go to the "Login" part. This part can only have one error, and that is, if the user already exists. So if the type value wasn't "Successful" it will go into this second part.

In "Login" (line 72), username and password will be asked. Sprintf will make the message, and it will be sent with the send function. Answer will be received with the recv function, and 3 scenarios will be tested. 1. If the type value was "AuthToken", 2.If the content value was "The user <username> is already logged in.", 3. If the content value was "Wrong password." In the first scenario, the mainMenu function will be called, and AuthToken will be saved for next occurances.

The "Clear Screen Every Time" is a sort of a flag and if it had a true value, in specific places the system function will be called, to clear the terminal.

mainMenu.c:

This section has a while(1) like the previous function, And 3 sections :

1 : Create Channel 2 : Join Channel 3 : Logout

In "Create Channel" (line 27), the user will be asked for a name. Sprintf and send and recv are like before. The program will give a feedback according to the type value being successful or not. If it was successful, the chatMenu function will be called.

In “Join Channel” (line 54), the user will again be asked for a name. Sprintf and send and recv are like before. And if the type value of the received string was successful, the chatMenu function will be called.

In “Logout” (line 88), the message will be sent for the server and the user will be logged out of their account. And the mainMenu function will return the void value, and return to userMenu.

chatMenu.c:

This function has a while(1) and a scanf, like the rest of the functions of this phase. This function consists of 4 sections :

1 : Send Message 2 : Refresh 3 : Channel Members 4 : Leave

In “Send Message”, the program will scan the string the user enters, as long as the user doesn’t press the “enter” key. This is possible by getting the string character by character with the getchar() function. (line 38). This section has snprintf instead of sprintf, because of the whitespaces that the string contains.

In “Refresh”, a relatively big message will be received from the server. And it will be parsed into cJSON. This way the “content” part can be accessed. A “for” loop has been written and in this loop, each array item has been extracted and printed right away, by using functions :

cJSON_GetArraySize, cJSON_GetArrayItem, cJSON_GetObjectItem

“Channel Members” work exactly like the “Refresh” part, except that in each loop of for, only one item has to be extracted.

In “Leave”, the leaving message will be sent for the server, and the user will return to the mainMenu.

Phase 2:

This phase consists of main.c and cJSON.c.

When the program is run, WSAStartup will be checked, server socket function will also be checked, and after that binding and listening will be checked.

After that, the number of existing channels and usernames will be checked, with the channelInroduction function and the userIntroduction function. I will only explain channelIntroduction here, because the other one acts similarly.

This function is possible by using the libraries : windows.h and dirent.h. the opendir function will try to open the defined address, and if it returns a NULL, that means the directory doesn’t exist. If it existed, the opendir function and readdir function will go to the next file continuously until there isn’t any unread files. Meanwhile the channel name will be extracted with ent->d_name. And therefore their names and number of channels can be printed in the terminal.

This phase consists of a while (true) with 9 “if” functions, the first one being “if”, and the rest of them are “else if”.

1. Register

Recieves string from client. Extracts username and password with sscanf. Checks if the file of the username exists or not, with fopen “r” mode. If the filePointer was NULL, it hadn’t existed previously, but if the filePointer wasn’t NULL, that counts as an error. Success message is sent to client.

2. Login

Recieves string from client. Extracts username and password with sscanf. Checks if the file of the username exists or not, with fopen “r” mode. If the filePointer was NULL, it hadn’t existed previously and this is the error in this part, but if the filePointer wasn’t NULL, that counts as a success. Now fscanf is used to retrieve the password of the existing user, if the password entered by client was correct, the process has been successful, otherwise, a “wrong password” message will be sent to client. This message should be in cJSON format, so cJSON_AddStringToObject has been used two times, one for type and type value, and another one for content and content value. Then we should check if the user is already in any channel or not by checking the user structure, if the user was in a channel, an error should be sent to client. Otherwise, a 32 character token is created randomly, by using time.h and srand and rand. And just in case, the token is then compared with every other made token just to make sure that it is a unique token. After that the AuthToken will be sent to the client.

3. Create Channel

The server receives a string from the client, which contains a channel name and an AuthToken. By using the AuthToken, we can find the user’s number in the array of user structures. If the token was unacceptable, a cJSON will be made and the error will be sent to client. Otherwise the existence of the channel will be checked with fopen “r” mode. And if the filePointer wasn’t NULL, the channel exists and a cJSON should be made and the error should be sent. This is done by cJSON_CreateObject and cJSON_AddStringToObject. Otherwise everything has been ok and the information should be saved. We will use these functions :

cJSON_CreateObject, cJSON_CreateArray, cJSON_AddItemToObject

cJSON_CreateString, cJSON_Print

The first message that will be created in this channel will be “<username> created <channel_name>”. This will be written to the file by using fprintf.

4. Join Channel

Acts similar to the previous section, the only difference is that the channel already exists. So we fscanf from the file, and use cJSON_Parse, cJSON_GetObjectItem. And create a string

containing the message “<username> joined the channel” and add that to the array of messages, and use cJSON_Print and then fprintf followed by a fclose.

5. Send

To extract the AuthToken we take the last characters of the string with a for loop. And then we check for AuthToken’s validation. By using a for loop we figure out the user’s number in the user structures array, and we can see if they’re in any channel or not. If everything was ok, we read the JSON from the file, by using fscanf and then add an object containing “sender” and “content” to the “messages” part. Then we convert it to a string with cJSON_Print and then fopen “w”, fprintf and fclose. Success message is sent to the client.

6. Refresh

The errors will be the same as the previous section, if there weren’t any errors, we read the file with fscanf, and grab the “messages” object from the json, and rename it to “content” and add a string “type” with the value of “List” before that. Then we use cJSON_Print. And send it to client.

7. Channel Members

The errors are like the previous two sections. If everything was ok, we extract the user’s number and their channel’s number and because we have the channel struct, we know the names of the people inside the channel currently. So we create an object with cJSON_CreateObject, then add a string and an array to it. After that there is a “for” loop which adds the channel members’ names to the array, and after that cJSON_Print gives us a string to send to the client.

8. Leave

The errors are like the previous sections. We use fscanf to get the current messages in the channel, then parse it with cJSON_Parse, and add a string saying “<username> left the channel.” And use fprintf and fclose. Success message is sent to the client. The most important part in this section is editing the structures, the person doesn’t exist in any channels anymore, and the respective channel has one less person inside. Since we can find the user’s number and their channel’s number, this will be possible.

9. Logout

The errors are like the previous sections. If there weren’t any errors, the number of users will decrease by one, and a success message will be sent to the client. And in the user structures array, the people after that user will take the user’s place and fill up the void.

Phase 3:

In this phase, all cJSON usages have been deleted from the codes. And instead, they have been made using sscanf and sprintf. Knowing the specific syntax of sscanf and sprintf comes in handy. We use [^\0] to scan until the end of the string, and we use %*s or %*c to ignore specific items.