Program documentation:

How to run program:

* To start ***server*** run *serverStartup.py*
* To start ***client*** run *clientLoad.py*

These Python files can be run in idle or by double clicking the file itself

Program details:

* Client works by initially running *clientLoad*.py, program will load successful ONLY if the server is active, if the server is offline the client application will produce an appropriate error message
* Server works on its own by running *serverStartup.py* and will run until it is stopped
* If a client logs out or is disconnected the server will still run, however if a client is running when the server is offline the client will be forced to close the program with an appropriate error message being displayed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Description | Files involved | Values tested and expectation | Test results |
| SERVER:  Server application starts when run | Double click *serverStartup.py* to start the program | serverStartup.py | Server console displays “Server running” | **Pass** |
| SERVER:  Server closes | Server disconnects/closes when the window exit button is clicked | serverStartup.py | Application closes | **Pass** |
| CLIENT:  Start client application when server is offline | Double click *clientLoad.py* to start the program, with the server being offline | clientLoad.py | Appropriate error message is displayed | **Pass** |
| CLIENT:  Start client application when server is online | Double click *clientLoad.py* to start the program, with the server being online | clientLoad.py  clientData.py  clientConnection.py  clientMain.py | If the server is successfully running, when *clientLoad.py* is running*, clientConnection.py* and *clientMain.py* are also called and run, with *clientMain.py* being responsible for displaying the programs main menu window and *clientConnection.py* being responsible for checking if the server is still connected/reachable/online | **Pass** |
| SERVER:  Check server console window to see a successful client connection | If the server is running successfully, when a client application is running the client should successfully connect to the server | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  serverStartup.py | If the server is successfully running, when the client connects the servers console window should display the connected address and port of the client as well as the command “PING” to signify a successful connection | **Pass** |
| CLIENT:  Check if variables are imported from *clientData.py* | Check to see if the client file *clientLoad.py* loads all the data from *clientData.py* | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  serverStartup.py | If clientLoad.py successfully loads the data from clientData.py the background of the application should be a grey colour and the program should be named ‘Chatsy’ | **Pass** |
| CLIENT:  Check login page loads | Click the ‘Login’ button on the Main Menu window | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  serverStartup.py | If the server is successfully running, when the ‘Login’ button is clicked the main menu window should load the Login page window | **Pass** |
| CLIENT:  Check login page loads (when server is offline) | Click the ‘Login’ button on the Main Menu window | clientLoad.py  clientData.py  clientConnection.py  clientMain.py | If the server is not running, when the ‘login’ button is clicked an appropriate error message should be displayed and when acknowledged the application should close | **Pass** |
| CLIENT:  Check registration page loads | Click the ‘Register’ button on the Main Menu window | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  serverStartup.py | If the server is successfully running, when the ‘Register’ button is clicked the main menu window should load the Registration page window | **Pass** |
| CLIENT:  Check registration page loads (when server is offline) | Click the ‘Register’ button on the Main Menu window | clientLoad.py  clientData.py  clientConnection.py  clientMain.py | If the server is not running, when the ‘Register’ button is clicked an appropriate error message should be displayed and when acknowledged the application should close | **Pass** |
| CLIENT:  Check ‘Exit’ button | Click the ‘Exit’ button on the Main Menu window | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  serverStartup.py | When the ‘Exit’ button is clicked, a confirmation message box pops up, asking the user to confirm their decision. If user clicks ‘Yes’ the application closes and if ‘No’ the application continues to run | **Pass** |
| CLIENT:  Test ‘Hide Password’ checkbox within the registration form | From the Main Menu, the ‘Register’ button is clicked, and the registration page is loaded. ‘Hide Password’ checkbox is clicked | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  serverStartup.py | If the checkbox is checked the password is hidden, unchecked the password can be seen | **Pass** |
| CLIENT:  Use the registration form to create a new chat account  (Valid details) | From the Main Menu, the ‘Register’ button is clicked, and the registration page is loaded. Valid details are entered (username and password characters between length of 5-14) | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | If all details entered are valid, once the ‘Register’ button is pressed the registration window opens the chat window, running *clientChat.py* file and the user can start to chat  Username: RealUser  Password Password | **Pass** |
| CLIENT:  Test public and private key distribution between client and server for the registration of an account  (Diffie Hellman key exchange) | If the server is successfully running and the user enters a valid (unused) username and valid password and clicks the ‘Register’ button, the client and server exchange public keys to both simultaneously generate their own respected partial keys which are exchange and generate private keys which they both privately keep and store | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Once the ‘Register’ button is clicked the server and client partake in a public key exchange, then generating partial keys which are exchanged which are then used to generate the private keys. If all is successful, the *clientChat.py* file is loaded, and the chat client window is displayed  Username: test1  Password: test1 | **Pass**  (Client console is shown here to visually demonstrate the key exchange)  CLIENT CONSOLE:    SERVER CONSOLE:    Both generate an equal full encryption as part of the diffie-hellman key exchange. This generation of fully key will be used as part of the encryption and decryption process for both client and server. (NOTE: smaller primary key values are used for demonstration purposes and testability of each key generation algorithm) |
| CLIENT:  Use the registration form to create a new chat account  (Invalid details) | From the Main Menu, the ‘Register’ button is clicked, and the registration page is loaded. Invalid details are entered (username and password characters are not between length of 5-14) | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  serverStartup.py | If all details entered are invalid, once the ‘Register’ button is pressed information messages are displayed indicating that the username and password entered are invalid  Username: no  Password: pass | **Pass** |
| CLIENT:  Use the registration form to create a new chat account  (Existing details) | From the Main Menu, the ‘Register’ button is clicked, and the registration page is loaded. An existing username is entered. | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  serverStartup.py  User\_Data.db | If the username entered is invalid, once the ‘Register’ button is pressed an error message is displayed indicating that the username already exists  Username: RealUser  Password: Password | **Pass** |
| SERVER:  The server should be able to handle the clients request to register a new account | Once the user clicks the ‘Register’ button within the client program, the server will receive a ‘CONN’ command, this is for the CONNection between the client and server and the request for an initial key exchange.  Once this is complete another command is sent by the client, the ‘REGD’ command. This command corresponds to REGister Data, in which the clients’ requests to have it’s data stored within the *User\_Data.db* database. Through this the data is sent encrypted to ensure maximum security and the server will need to decrypt the data. If the server can successfully store the clients username and password the server will return a boolean response ‘True’ or ‘False’ indicating its success. If the username already exists for example.  If successfully, return ‘True’, the user is allowed to login and the client chat window is displayed.  Within the server console once successful and the data is stored within the database the server receives a ‘USER’ command from the clientChat.py file which is now loaded, with ‘USER’ indicating the client is ready to chat with the server and wants to share its username.  Another key exchange is partaken between the client (this time from *clientChat.py*) and the server to establish a secure connection. (For demonstration these keys are the same but can be changed to ensure different ciphertext generation for chat messages compared to login/registration) | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Once the ‘Register’ button is clicked the server and client partake in a public key exchange, then generating partial keys which are exchanged which are then used to generate the private keys. If all is successful, the *clientChat.py* file is loaded, and the chat client window is displayed in which another key exchange is established to encrypt/decrypt the chat messages  Username: test2  Password: test2 | **Pass**  SERVER CONSOLE: |
| CLIENT:  Enter details into login form  (Valid existing details) | From the Main Menu, the ‘Login’ button is clicked, and the login page is loaded. Valid details are entered | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | If all details entered are valid, once the ‘Login’ button is pressed the login window opens the chat window, running *clientChat.py* file and the user can start to chat  Username: RealUser  Password: Password | **Pass** |
| SERVER:  The server should be able to handle the clients request to login to an existing account | Once the user clicks the ‘Login’ button within the client program (login page), the server will receive a ‘CONN’ command, this is for the CONNection between the client and server and the request for an initial key exchange.  Once this is complete another command is sent by the client, the ‘LOGD’ command. This command corresponds to LOGin Data, in which the clients’ requests to have its data retrieved from within the *User\_Data.db* database from the user\_details table. Through this the data is sent by the client encrypted to ensure maximum security and the server will need to decrypt this data. If the server can successfully retrieve the data corresponding to the clients requested username and password, the server will send back the username and password, again encrypted, to the client. If the decrypted username and password sent by the client is the same as the received username and password the client will successfully log the user in.  However, if the clients sent username and password doesn’t match an “ERROR” string is encrypted and sent to the client. If this is the case an error message box is displayed to the user displaying an appropriate error message.  Within the server console once successful the database the server receives the string ‘USER’ command from the clientChat.py file which is now loaded, with ‘USER’ indicating the client is ready to chat with the server and wants to share its username.  Another key exchange is partaken between the client (this time from *clientChat.py*) and the server to establish a secure connection. (For demonstration these keys are the same but can be changed to ensure different ciphertext generation for chat messages compared to login/registration) | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Once the ‘Login’ button is clicked the server and client partake in a public key exchange, then generating partial keys which are exchanged which are then used to generate the private keys. If all is successful, the *clientChat.py* file is loaded, and the chat client window is displayed in which another key exchange is established to encrypt/decrypt the chat messages  Username: RealUser  Password: Password | **Pass**  CLIENT CONSOLE:    SERVER CONSOLE: |
| CLIENT:  Check to see if the online users list is being updated | When the user is logged in and the *clientChat.py* file is loaded with the Chat client being displayed the Online users list should be updated with all online connected users  This list is refreshed within a loop that checks every 5 seconds to see who is currently connected | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Username: RealUser  Password: Password  Connected users should be displayed on the online users list | **Pass**  (Result highlighted in a red for this demonstration purpose) |
| CLIENT:  Check to see if the message/chat window is being updated with the users’ messages | When the user is logged in and the *clientChat.py* file is loaded with the Chat client window being displayed the chat frame should be updated to display every connected users’ messages when the ‘send’ button is clicked | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Username: RealUser  Password: Password  Message: Hello, this is a real test message, enjoy!  Message sent by the client should be displayed to all connected users on the chat frame | **Pass**  (Result highlighted in a red for this demonstration purpose) |
| SERVER:  The communication of chat messages between client and server should be encrypted using the Diffie Hellman key exchange | Once the user clicks the ‘Send’ button to send a message, the client creates a new string with the first 4 letters being the command, MSGR, followed by the username of the client and then followed by the users’ message. This new string should read “MSGRRealUser: Hello, this is a real test message, enjoy!” for example. This message is then encrypted and is then sent to the server as ciphertext using the clients private key, in this case it looks like “°¬· ¾°½k°··ºwk¿³´¾k´¾k¬k½°¬·k¿°¾¿k¸°¾¾¬²°wk°¹µºÄlU”, a bunch of random letters and characters. If the command is ciphertext the server will detect this and then decrypts this message back into plaintext. Once this message has been decrypted the server looks at the first 4 characters of the string to find the command, the command for a message is “MSGR” for MeSsaGe Received. As the command is MSGR the message plaintext is then encrypted back into ciphertext using the server’s private key encryption and is broadcasted to ALL connected client addresses/users. Within each client program this message is received and decrypted from cipher text back into plaintext using the client’s private key system. The command is then retrieved from the 4 characters of the string, again it’s “MSGR”. The command is then removed from the string and the message is displayed on the clients’ chat frame | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Username: RealUser  Password: Password  Message: Hello, this is a real test message, enjoy!  Message sent by the client should be displayed to all connected users on the chat frame | **Pass**  CLIENT CONSOLE:  (Client console is shown here to visually demonstrate the key exchange)    SERVER CONSOLE: |
| CLIENT:  Check to see if the message/chat window is being updated with the users’ messages | When a message is typed into the message box and the ‘Send’ button is clicked | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Message: Hello everyone!  Message: How are we all?  Message: I’m doing great, how is everyone else? | **Pass** |
| CLIENT:  Check to see if more then one user can connect to the server at the same time | Another client will connect to the same server under a different username | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Username: test1  Password: test1  If a new user connects a message will be broadcasted to all clients announcing that a new user has connected | **Pass**  (Result highlighted in a red for this demonstration purpose) |
| CLIENT:  Check to see if the online users list is being updated when new users connect | When the user is logged in and the *clientChat.py* file is loaded with the Chat client being displayed the Online users list should be updated with all online connected users  This list is refreshed within a loop that checks every 5 seconds to see who is currently connected | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Username: test1  Password: test1  Connected users should be displayed on the online users list | **Pass**  (Result highlighted in a red for this demonstration purpose) |
| CLIENT:  Check to see if more then one users messages can be sent and displayed on each clients chat frame | This test will demonstrate if more then one client can chat at the same time and communicate with each other, each client should see the other clients’ messages | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | User: test1:  Message: Hi RealUser, test1 here! I’m doing great thanks for asking!  Message: This chat program is really cool! | **Pass**  RealUser client:    test1 client: |
| CLIENT:  Check to see if a client can disconnect successfully | Once a client successfully logouts/disconnects an appropriate message should be displayed to all other connected users | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Test1 client clicks ‘Logout’ button | **Pass**  (Result highlighted in a red for this demonstration purpose) |
| CLIENT:  Check to see if the online users list is being updated when new users connect | When the user is logged in and the *clientChat.py* file is loaded with the Chat client being displayed the Online users list should be updated with all online connected users  This list is refreshed within a loop that checks every 5 seconds to see who is currently connected | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Client test1 logout of client and disconnects from the server, Online users frame should update with test1 being removed from the list and RealUser should remain | **Pass**  (Result highlighted in a red for this demonstration purpose) |
| SERVER:  Test the functionality of the server when a client disconnects | Once the user clicks the ‘Logout’ button, the client sends the “DISC” command to the server, this command means DISConnect. Once the server receives this command the client then sends the server the username that needs to be removed from the online users list, this is also encrypted and sent as ciphertext.  Once this ciphertext username is received by the user it is decrypted back into plaintext.  The username is collected and removed from the connected users array that holds all the connected users. The index is also found of the user in the array and this index is used to remove the address of the disconnected user from the connected addresses array.  A message is then broadcasted to all users indicating that the user has disconnected. | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | Client clicks ‘Logout’ button | **Pass**  SERVER CONSOLE: |
| CLIENT:  Test the functionality of the ‘Logout’ button on the Chat client window | Click the ‘Logout’ button | clientLoad.py  clientData.py  clientConnection.py  clientMain.py  clientChat.py  serverStartup.py  User\_Data.db | When the ‘Logout’ button is clicked, a confirmation message box pops up, asking the user to confirm their decision. If user clicks ‘Yes’ the Chat client window closes and the Main Menu window is displayed and if ‘No’ the application continues to run | **Pass** |