Java 3

Project AT3

Question 2

Product Design Specification

Version 0.1

Student Name: Alan Pedersen

Student ID: P225139

Date: 22/05/2020

Version History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version # | Implemented By | Version Date | Approved By | Approval Date | Reason |
| 0.1 | A Pedersen | 22/05/20 |  |  | Initial Draft Design Document |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Table of Contents

[Table of Figures ii](#_Toc41129935)

[Table of Tables ii](#_Toc41129936)

[Introduction 1](#_Toc41129937)

[Design Guidelines 1](#_Toc41129938)

[Coding Conventions 1](#_Toc41129939)

[GUI Design 1](#_Toc41129940)

[System Architecture 2](#_Toc41129941)

[Introduction 2](#_Toc41129942)

[Server Application 2](#_Toc41129943)

[Server Application Class Definitions 4](#_Toc41129944)

[The MainForm class 4](#_Toc41129945)

[The Client Class 5](#_Toc41129946)

[The ClientList Class 5](#_Toc41129947)

[The PasswordManager Class 7](#_Toc41129948)

[The DataStore Class 8](#_Toc41129949)

[The ListenForClient Class 8](#_Toc41129950)

[The SocketServer Class 9](#_Toc41129951)

[Client Application 9](#_Toc41129952)

[Client Application Class Definitions 11](#_Toc41129953)

[The MainForm class 11](#_Toc41129954)

[The ClientListener Class 12](#_Toc41129955)

[System Design 13](#_Toc41129956)

[Proposed GUI 13](#_Toc41129957)

[Database Design 15](#_Toc41129958)

[Testing 16](#_Toc41129959)

[Server Controls Testing 16](#_Toc41129960)

[User Controls Testing 16](#_Toc41129961)

[Client Login Testing 17](#_Toc41129962)

[Client Disconnect Testing 17](#_Toc41129963)

[Messaging Function Testing 17](#_Toc41129964)

[Design Specification Approval 18](#_Toc41129965)

# Table of Figures

[Figure 1: MessageServer class diagram 2](#_Toc41129966)

[Figure 2: Client class diagram 5](#_Toc41129967)

[Figure 3: ClientList class diagram 6](#_Toc41129968)

[Figure 4: PasswordManager class diagram 7](#_Toc41129969)

[Figure 5: DataStore class diagram 8](#_Toc41129970)

[Figure 6: ListenForClient class diagram 8](#_Toc41129971)

[Figure 7: SocketServer class diagram 9](#_Toc41129972)

[Figure 8: MessageClient class diagram 10](#_Toc41129973)

[Figure 9: server application design 13](#_Toc41129974)

[Figure 10: client application design 14](#_Toc41129975)

# Table of Tables

[Table 1: server GUI button actions 4](#_Toc41129976)

[Table 2: server GUI get methods 4](#_Toc41129977)

[Table 3: server GUI miscellaneous controls 5](#_Toc41129978)

[Table 4: client GUI button actions 11](#_Toc41129979)

[Table 5: client GUI get methods 11](#_Toc41129980)

[Table 6: client GUI miscellaneous controls 12](#_Toc41129981)

[Table 7: client states summary 14](#_Toc41129982)

[Table 8: user login table 15](#_Toc41129983)

[Table 9: event log table 15](#_Toc41129984)

[Table 10: user controls test table 16](#_Toc41129985)

[Table 11: client login test table 17](#_Toc41129986)

# Introduction

Jupiter Mining Corporation have requested the development of a messaging system that will allow the exchange of messages between users and a central administrator.

The proposed system will use a server application providing overall control of the system, combined with a client application that will be distributed to users. Access to the system will be controlled using passworded user accounts.

# Design Guidelines

## Coding Conventions

Standard Java coding conventions as recommended by Sun Microsystems should be adopted.

## GUI Design

The GUI should be designed to present an intuitive interface to the user.

GUI elements should be visibly grouped into logical sets.

GUI elements should only be enabled when their use is allowed and disabled when their use becomes illegal.

All input should be validated before use and feedback provided immediately explaining any exceptions.

# System Architecture

## Introduction

The system management functions are provided by a server application supporting:

* Creation of new users
* Validation of user login attempts
* Distribution of messages

A client application will enable users to:

* Login to the system
* Send and receive messages

## Server Application

The server application uses a series of classes to support its operation. The details of each class are discussed in the following section.

The MessageServer class contains the Main method and provides overall control of the functionality of the application. The application is event driven using a graphical user interface. The methods used by the MessageServer class are tied to button click events generated by the GUI. The main method launches the GUI using the start method to initialise the user interface elements.

The following class diagram shows the proposed structure for the class:

|  |
| --- |
| MessageServer Class |
| Attributes:  + testingFlag : Boolean  + DISCONNECT : String  + LOGIN\_PASS : String  + LOGIN\_FAIL : String  + ADMIN\_USER : String  + ADMIN\_PW : String  + mainStage : Stage  - ObservableList<String> clientList  - ObservableList<String> messageList  - ClientManager : ClientList  + PORT\_NUMBER : int  - listener : ListenForClients |
| Methods:  + void Main()  + void start( Stage primaryStage)  + void CreateNewUser()  + void UserLogIn()  + void UserLogOut()  + void StartSocketServer()  + void BroadcastMessage()  + void DisconnectSelectedUser() |

Figure : MessageServer class diagram

A description of the attributes of the class follows.

The testingFlag is a semaphore that will provide increased feedback for testing when set to true.

DISCONNECT is a string constant defining the disconnection message.

LOGIN\_PASS and LOGIN\_FAIL are string constants defining the successful and unsuccessful login messages.

ADMIN\_USER and ADMIN\_PW are string constants defining the default admin username and password.

The Stage object mainStage is a public reference to the main stage used for the GUI.

Two observable lists are used by the application. clientList is bound to a ListView element in the GUI and is used to display client names and login details. It is populated by the ListNodes method in the ClientList object instantiated as ClientManager. messageList is bound to a ListView element in the GUI and displays messages received from clients and system messages.

ClientManager is an instantiation of the ClientList object and is used to provide client management functions.

PORT\_NUMBER is an integer constant defining the port number for the socket.

An instance of the ListenForClients class is referenced as listener.

The methods included in the class perform the following functions.

The main method starts the GUI running using the start method to build the interface.

The start method builds the GUI and starts it running.

The CreateNewUser method is used to create a new user account. The username and password are retrieved from the main form and the AddClient method of the ClientManager is used to create the new account and add it to the tree structure. The method traps for exceptions thrown by the main form and the AddClient method. The admin user must be logged in before an account can be created.

The UserLogIn method is used to enable the admin functions and to test user logins. The username and password are retrieved from the main form and the username used to search for a matching Client object. If a matching client object is found the password is checked. If the password is validated the user is flagged as active. If any of the tests fail the user is flagged as inactive and an error message given. If the successful login is by the admin user, the restricted functions of the interface are enabled. The method traps for exceptions thrown by the main form. The login event is recorded in the system log table.

The UserLogOut method is used to log a user out of the system. The username is retrieved from the main form and the username used to search for a matching Client object. If a matching client is found it is flagged as inactive. No matching client will generate an error message. If the admin account logs out the restricted functions are disabled. The method traps for exceptions thrown by the main form. The log out event is recorded in the system log table.

The StartSocketServer method is used to start the listener object running.

The BroadcastMessage method is used to send a message to all active clients. The message is retrieved from the main form. The main form will throw an exception if no message is found. The ClientManager object is used to send the message.

The DisconnectSelectedUser is used to disconnect one user from the server. A record must be selected in the client list on the main form. The selected record is retrieved, and the client name found in the string. The client name is used to find the Client object and a disconnect message is sent to the client. The Clients stream objects are closed and the client flagged as inactive.

## Server Application Class Definitions

### The MainForm class

The MainForm class contains a set of static functions that are used to build and to interact with the user interface.

Six methods are used to build the user interface:

* buildGui main controlling method
* addServerControlsToGui adds the server user interface elements
* addUserControlsToGui adds the user control interface elements
* addMessageContolsToGui add the messaging control interface elements
* addCloseToGui add the close button and error message box
* addEventListeners add the event listener code to the interface elements

Each button in the interface has an attached listener that calls a method. The details are listed in the following table:

|  |  |  |
| --- | --- | --- |
| Button | Method | Action performed |
| btnStartServer | StartServer() | Calls MessageServer.StartSocketMethod() |
| btnLogIn | UserLogIn() | Calls MessageServer.UserLogIn() |
| btnLogOut | UserLogOut() | Calls MessageServer.UserLogOut() |
| btnDisconnect | DisconnectUser() | Calls MessageServer.DisconnectSelectedUser() |
| btnNewUser | NewUser() | Calls MessageServer.CreateNewUser() |
| btnClearReceivedMessages | ClearMessageReceived() | Clears the messages received list view |
| btnClearSendMessages | ClearMessageSent() | Clears the send message text area |
| btnSendMessage | SendMessage() | Calls MessageServer.BroadcastMessage() |
| btnClose | CloseApplication() | Closes the application |

Table : server GUI button actions

The TestSelection method is run when a record is selected in the client list view interface element. The method check the selected record and if the client is flagged as active and is not the admin account the disconnect button is enabled, otherwise the disconnect button is disabled.

A series of methods are provided to get values from the interface elements. The elements will throw an exception if no value is found in the element. The methods are summarised in the following table:

|  |  |  |
| --- | --- | --- |
| Method | GUI element | Comment |
| GetSelectedClient | clientListView | Can only be called when a record has been selected in the ListView |
| GetUserName | tfUserName | Throws an exception if field is empty |
| GetUserPw | tfPassword | Throws an exception if field is empty |
| GetSendMessage | tfMessage | Throws an exception if field is empty |

Table : server GUI get methods

The remaining methods are used to manage the user interface:

|  |  |
| --- | --- |
| Method | Comment |
| ClearUserFields | Clears the user name and password fields |
| AdminLogin | Enables the new user button |
| AdminLogout | Disables the new user button |
| ShowReceivedMessage | Adds a received message to the observable list |
| SetSocketStatus | Show the current status of the socket |
| DisableServerStartButton | Disables the server start button |
| EnableServerStartButton | Enables the server start button |

Table : server GUI miscellaneous controls

### The Client Class

The details for each user will be stored using the Client class. The Client class is designed to simply store the data required for each client.

The following class diagram shows the proposed structure for the class:

|  |
| --- |
| Client Class |
| Attributes:  - name : String  - passwordSalt : byte[]  - passwordHash : String  - activeFlag : boolean  - ServerSocket : Socket  - readStream : DataInputStream  - writeStream : DataOutputStream  - left : Client  - right : Client |
| Methods:  + Client( String name, String passwordHash, byte[] passwordSalt, Boolean activeFlag)  + Getter & Setter methods  + string ToString() |

Figure : Client class diagram

A separate class will be used to manage password salting, hashing and validation. The ToString method is used to generate the records displayed for the client list displayed in the server application and simply returns the client name and a flag indicating if the client is currently logged into the system.

### The ClientList Class

The user objects will be stored in a self-balancing binary tree structure. The ClientList class provides the functions necessary to manage the binary tree structure and to distribute messages. The Client name will be used as the key value when ordering the list.

The following class diagram shows the proposed structure for the class:

|  |
| --- |
| ClientList Class |
| Attributes:  - rootNode : Client |
| Methods:  + void ClientList()  + void AddClient( String newName, String newPw)  + void AddNewNode( Client newNode)  - Client AddNodes( Client thisNode, Client newNode)  - int FindHeight( Client thisNode)  - int CalcBalanceFactor( Client thisNode)  - Client RightRotation( Client node2)  - Client LeftRotation( Client node1)  - Client LeftRightRotation( Client node3)  - Client RightLeftRotation( Client node3)  + Client FindNode( String searchValue)  - Client SearchNodes( Client thisNode, String searchValue)  + void ListNodes()  - void ListNodesAdd( Client thisNode, ArrayList<Client> nodeList)  + void Delete( String searchValue)  - Client DeleteNode( Client thisNode, String searchValue)  - Client FindSmallestNode( Client testNode)  + void MessageAllClients( String message)  - void FindActiveClients( Client thisNode, String message)  + void MessageOneClient( Client cl, String message)  + String GetClientNameFromListRecord( String clientRecord) |

Figure : ClientList class diagram

The constructor method creates initialises the root node of the binary tree structure to null and sets a reference to the observable list of clients.

The AddClient method creates a new Client object and adds it into the binary tree. The method receives the name and password for the new client. It checks that the name is unique and that the password is different to the name, failing either test will cause an exception to be thrown. The password manager is used to generate a salting value and this is used with the password to create a salted, hashed password string. A new Client object is created and added to the tree using the AddNewNode method. The new Client object will be passed to the DataStore class to create a new record in the database.

The AddNewNode method is used to add a new Client to the tree. The method calls AddNodes passing a reference to the root node and the new Client. The method searches the tree recursively until it finds an unoccupied location where it adds the new Client. At each node the Client name stored in that node is compared with the client name from the new Client. If the new name precedes the current name alphabetically the left node is recursively searched, if the new name succeeds the current name the right node is recursively searched. As the recursion unwinds the height of each branch is calculated at each node. If the height of the branches differs by two or more levels the tree is balanced. The method calls the ListNodes method to update the list of users displayed in the GUI.

The FindNode method is used to search for a Client object using the Client name. The method calls the SearchNodes method passing the root node and the Client name. The tree is searched recursively until a match is found or an unoccupied location is found. If a match is found a reference to the node is returned otherwise a null value is returned.

The ListNodes method populates the observable list of clients using the ToString method of each Client. The method calls ListNodesAdd passing the list and the root node. The tree structure is searched recursively adding each record in alphabetical order.

The MessageAllClients method will send the submitted message to all active clients. The method calls the FindActiveClients method passing the root node and the message. The tree is searched recursively, the activeFlag of each client is checked if true the message is sent to the Client using the MessageOneClient method.

The MessageOneClient method is used to send a message to the selected Client. The method is passed a reference to the Client object and the message to be sent. The method uses the DataOutputStream referenced in the Client object.

### The PasswordManager Class

The password manager class provides password services to the application. The class provides a set of static methods that can be accessed as required.

The following class diagram shows the proposed structure for the class:

|  |
| --- |
| PasswordManager Class |
| Attributes: |
| Methods:  + byte[] GetSalt()  + String GetHashedPassword( String passwordToHash, byte[] salt)  + Boolean CheckPassword( String checkWord  , String hashCode, byte[] salt) |

Figure : PasswordManager class diagram

The GetSalt method generates a 16 byte random number to use when salting passwords.

The GetHashedPassword method accepts a password and a salt code and returns a string containing a salted hashed version of the submitted password.

The CheckPassword method accepts a password to test, a salted hashed password and a salt value. A salted hashed password string is generated from the submitted password and salt code. If this string matches the stored value the password is validated and true returned to the calling function. Otherwise a false value is returned.

### The DataStore Class

The DataStore class provides access to the data base for storage and retrieval of records.

The following class diagram shows the proposed structure for the class:

|  |
| --- |
| DataStore Class |
| Attributes:  - String connectionString  - String dbUserName  - String dbUserPw |
| Methods:  - Connection GetConnection()  + void LoadUsers()  + void NewUser( Client newUser)  + void AddLogRecord( String log) |

Figure : DataStore class diagram

The GetConnection method is called by the other methods to open a connection to the MySQL database.

The LoadUsers method is called when the application is started. It retrieves a list of usernames, salt codes and hashed salted passwords from the database. It loops though the list creating a new Client object for each record and calling the AddNewNode method to store the Client object.

The NewUser method is used to add the details for a new user to the user table. The method uses an INSERT SQL query to add the user details to the table.

The AddLogRecord method is used to record user logins, log outs and disconnections. The method uses an INSERT SQL query to add the user details to the table.

### The ListenForClient Class

The ListenForClient class is started in its own thread when the server is started. The thread runs a continuous loop listening for connections from client applications. When a connection request is received a SocketServer object is created and the connection passed to it.

The following class diagram shows the proposed structure for the class:

|  |
| --- |
| ListenForClient Class |
| Attributes:  - int ServerPortNumber  - ClientList ClientManager |
| Methods:  + ListenForClient( ClientList newClientList, int serverPort)  + void run() |

Figure : ListenForClient class diagram

### The SocketServer Class

A SocketServer object is created to manage each user connection.

When a SocketServer object is instantiated, it calls its own start method to start the thread running. Input and output data streams are opened, and a username and password string read from the input stream.

The username is used to search for a Client record. If no match is found a login fail message is returned, the connections closed and the thread terminates.

The Client record is used to validate the submitted password. If the password validation fails a login fail message is sent, the connections are closed and the thread terminates. If the password is valid a login success message is sent, the socket and data stream references are set in the Client record and the active flag set to true.

The SocketServer runs a continuous loop listening for messaged from the attached client. Received messages are displayed in the main form. If the thread received a disconnect message the connections are closed and the thread terminated.

The following class diagram shows the proposed structure for the class:

|  |
| --- |
| SocketServer Class |
| Attributes:  - Socket ServerSocket  - DataInputStream readStream  - DataOutputStream writeStream  - ClientList ClientManager  - String username  - String userPw  - Client testClient  - Boolean validPassword  - String message  - String readMessage |
| Methods:  + SocketServer( Socket newSocket, ClientList newClientList)  + void run() |

Figure : SocketServer class diagram

## Client Application

The client application allows users to connect to the server application and exchange messages.

The application uses three classes, MessageClient is the main application class, MainForm contains a set of static methods for creating and interacting with the user interface, ClientListener listens for messages from the server.

The MessageClient class contains the main method and provides overall control for the application. The application presents a form to the user and is driven by events generated by the form.

The following class diagram shows the proposed structure for the class:

|  |
| --- |
| MessageClient Class |
| Attributes:  + testingFlag : Boolean  + DISCONNECT : String  + LOGIN\_PASS : String  + LOGIN\_FAIL : String  - messageReceiver : ClientListener  + clientSocket : Socket  + readStream : DataInputStream  + writeStream : DataOutputStream  + HOST\_NAME : String  + HOST\_PORT : int  + mainStage : Stage |
| Methods:  + void main()  + void start( Stage primaryStage)  + void LogInToServer()  + void LogOutOfServer()  + void SendMessageToServer() |

Figure : MessageClient class diagram

A description of the attributes of the class follows.

The testingFlag is a semaphore that will provide increased feedback for testing when set to true.

DISCONNECT is a string constant defining the disconnection message.

LOGIN\_PASS and LOGIN\_FAIL are string constants defining the successful and unsuccessful login messages.

An instance of the ClientListener class is created and referred to as messageReceiver.

The message exchange with the server is managed using the clientSocket, readStream and writeStream objects.

HOST\_NAME and HOST\_PORT are constants setting the parameters for the socket connection.

The GUI can be accessed via the mainStage object reference.

The methods included in the class perform the following functions.

The main method starts the GUI running using the start method to build the interface.

The start method builds the GUI and starts it running.

The LogInToServer method establishes a socket connection and allows the user to log in to the server. The method gets the client name and password from the user interface. A socket connection is requested, and input and output data streams opened. The method sends the username and password to the server and waits for a login confirmation message. The server will return a string indicating that the success of the login attempt and a message conforming success or explaining failure. The message is displayed in the received messages list.

If the login is successful a ClientListener object is created to listen for messages from the server and the user interface is updated to reflect the connected status.

If the login attempt fails, the data streams and socket are closed and the user interface is set to the unconnected status.

The LogOutOfServer method sends a disconnect message to the server, the data streams and socket are closed and the user interface is set to the unconnected status.

The SendMessageToServer method reads the message from the interface and sends it to the server.

## Client Application Class Definitions

### The MainForm class

The MainForm class contains a set of static functions that are used to build and to interact with the user interface.

Five methods are used to build the user interface:

* buildGui main controlling method
* addServerControlsToGui adds the server user interface elements
* addMessageContolsToGui add the messaging control interface elements
* addCloseToGui add the close button and error message box
* addEventListeners add the event listener code to the interface elements

Each button in the interface has an attached listener that calls a method. The details are listed in the following table:

|  |  |  |
| --- | --- | --- |
| Button | Method | Action performed |
| btnLogIn | UserLogIn() | Calls MessageClient.LogInToServer() |
| btnLogOut | UserLogOut() | Calls MessageClient.LogOutOfServer() |
| btnClearReceivedMessages | ClearMessageReceived() | Clears the messages received list view |
| btnClearSendMessages | ClearMessageSent() | Clears the send message text area |
| btnSendMessage | SendMessage() | Calls MessageClient.SendMessageToServer() |
| btnClose | CloseApplication() | Closes the application |

Table : client GUI button actions

A series of methods are provided to get values from the interface elements. The elements will throw an exception if no value is found in the element. The methods are summarised in the following table:

|  |  |  |
| --- | --- | --- |
| Method | GUI element | Comment |
| GetUserName | tfUserName | Throws an exception if field is empty |
| GetUserPw | tfPassword | Throws an exception if field is empty |
| GetSendMessage | tfMessage | Throws an exception if field is empty |

Table : client GUI get methods

The remaining methods are used to manage the user interface:

|  |  |
| --- | --- |
| Method | Comment |
| ClearUserFields | Clears the user name and password fields |
| SetUserLogin | Sets the interface to logged in status |
| SetUserLogout | Sets the interface to logged out status |
| ShowReceivedMessage | Adds a received message to the observable list |
| SetSocketStatusDisconnect | Sets the socket status to disconnected |
| SetSocketStatusConnected | Sets the socket status to connected |

Table : client GUI miscellaneous controls

### The ClientListener Class

An instance of the ClientListener class is created when the user successfully logs into the server. The class listens for messages received from the server and displays them in the message received list. The object runs in its own thread allowing the user to continue to use the interface.

The thread runs a continuous loop listening for messages and displaying them when received. Execution of the loop is stopped by receiving a disconnect message when the data streams and socket are closed and the thread terminated.

# System Design

## Proposed GUI

The system will use two applications, a server application, and a client application.

The proposed design for the server application is presented below:

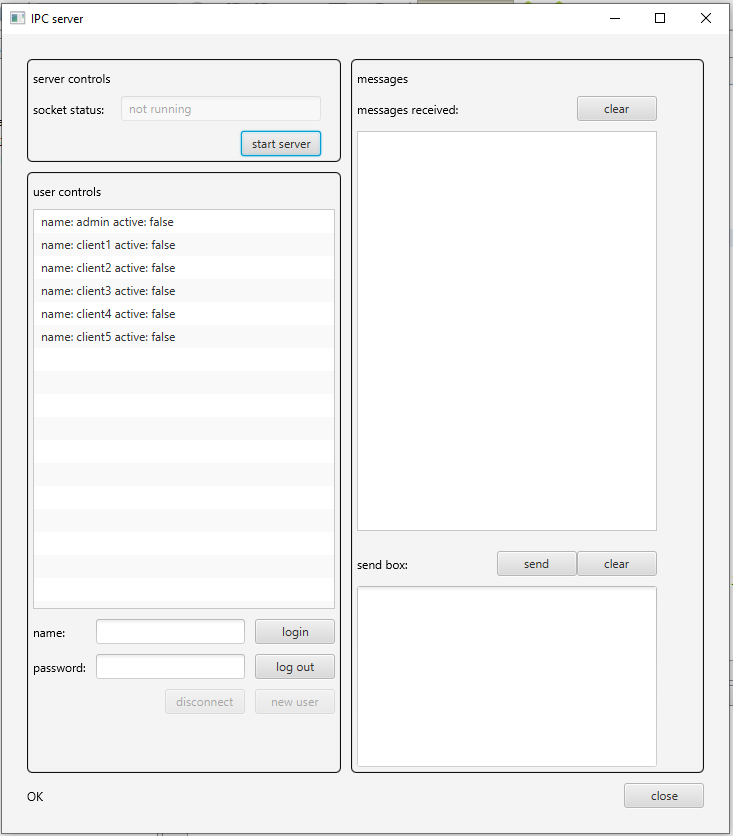


Figure : server application design

The GUI will dynamically change to reflect the status of the application.

The “start server” button will be disabled once the server has been started. The “socket status” text will display “not running” or “running on port ####”.

The “new user” button will only be enabled when the admin account is active.

The “disconnect” button will only be active when an active user has been selected in the “user controls” list.

The proposed design for the client application is presented below:

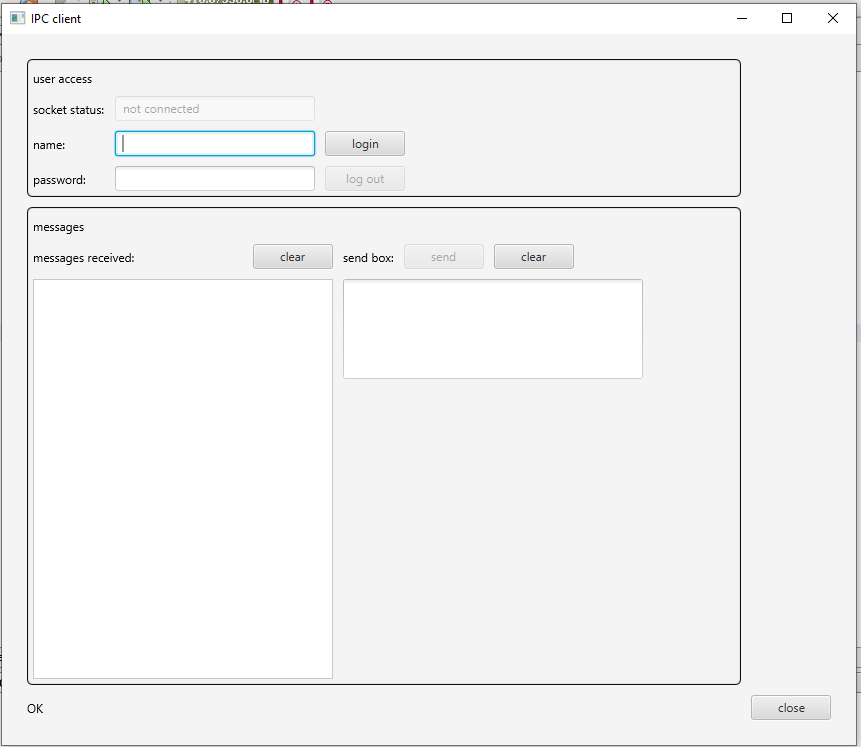


Figure : client application design

The GUI will dynamically change to reflect the status of the application.

The “socket status” text will display “not connected” or “connected on port ####”.

The client application presents two states to the user, logged in and logged out. The status of the affected controls are summarised in the following table:

|  |  |  |
| --- | --- | --- |
| Control | Logged Out | Logged In |
| login button | Enabled | Disabled |
| log out button | Disabled | Enabled |
| send button | Disabled | Enabled |

Table : client states summary

## Database Design

Two tables are required for the database, one to store the user login details and one to store the log records.

The table definition for the user login details is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field name | Field type | Field size | Null permitted | Key value |
| userName | Varchar | 50 | No | Yes |
| userSalt | Varchar | 100 | No |  |
| userHash | Varchar | 100 | no |  |

Table : user login table

The table definition for the log table is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field name | Field type | Field size | Null permitted | Key value |
| logTime | DateTime |  | No | Yes |
| logMessage | Varchar | 100 | No |  |

Table : event log table

## Testing

The sub-systems within the two applications will be tested individually.

### Server Controls Testing

The server will be started and its operation verified.

### User Controls Testing

The following test table will be used to test the user controls:

|  |  |  |
| --- | --- | --- |
| Test | Expected Outcome | Actual Outcome |
| Admin Login | Admin account is active  “new user” button enabled  Logging record created |  |
| Create a new user | A new user is added to the client list  Logging record created |  |
| Login test missing data | Error messages shown in message box |  |
| Login test invalid user | Error message shown in message box  Logging record created |  |
| Login test wrong password | Error message shown in message box  Logging record created |  |
| Successful login test | Client account is flagged as active  Login message displayed in message box  Logging record created |  |
| Test user creation with missing values | Error message shown in message box |  |
| Test user creation username = password | Error message shown in message box |  |
| Client logout | Client status is inactive  Logging record created |  |
| Admin Logout | Admin account is inactive  “new user” button disabled  Logging record created |  |

Table : user controls test table

### Client Login Testing

The following test table will be used to test the client login functions:

|  |  |  |
| --- | --- | --- |
| Test | Expected Outcome | Actual Outcome |
| Login with missing data | Error message displayed in the status bar |  |
| Login with an invalid username | Login failure message  Client not connected  Message displayed on the server  Logging record created |  |
| Login with an invalid password | Login failure message  Client not connected  Message displayed on the server  Logging record created |  |
| Successful login | Login successful message  Client connected  Message displayed on the server  Logging record created |  |

Table : client login test table

### Client Disconnect Testing

The client disconnect function will be tested on an active and an inactive client.

### Messaging Function Testing

Message sending and receiving will be tested on both the server and the client applications.

# Design Specification Approval

The undersigned acknowledge they have reviewed the Product Design Specification document and agree with the approach it presents. Any changes to this Requirements Definition will be coordinated with and approved by the undersigned or their designated representatives.

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: |  |  |  |
| Title: |  |  |  |
| Role: |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: |  |  |  |
| Title: |  |  |  |
| Role: |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: |  |  |  |
| Title: |  |  |  |
| Role: |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Signature: |  | Date: |  |
| Print Name: |  |  |  |
| Title: |  |  |  |
| Role: |  |  |  |