cna project Reflection

Hayden Walton | 20015319

Table of Contents

[Glossary 1](#_Toc62115845)

[Key Words 1](#_Toc62115846)

[Introduction 2](#_Toc62115847)

[Methodology / Literature Review 3](#_Toc62115848)

[Findings and Discussions / Results 4](#_Toc62115849)

[Conclusion 5](#_Toc62115850)

[Recommendations / Future Work 6](#_Toc62115851)

[Bibliography 7](#_Toc62115852)

[Appendices 8](#_Toc62115853)

[Appendix 1: Title 8](#_Toc62115854)

[Appendix 2: Title 8](#_Toc62115855)

# Key Words:

* WPF (Windows Presentation Foundation)
* C#
* Concurrency
* Networking

# Introduction

The project aim was to create a simple chat system networking application, using introduced concepts of Concurrency, WPF & C#. This report describes and reflects upon project development.

# Results:

### Strategy:

The initial strategy was to base the program off the given material, experiment, then draw conclusions regarding technology and techniques utilised.

This strategy was chosen because pre-defined facets are key components in developing the application, worth exploring and expanding upon. The development started with a simple check list paraphrased from the assignment mark scheme. Each component was highlighted, helping to direct my division of work and time.

### Technology:

Technologies include WPF, C#, Threading and Concurrency techniques, and Networking. Each play a crucial role in the application.

#### Networking:

Networking determines the execution of the artifact. There is a co-dependant relationship between networked clients and the server. This initialisation occurs with the creation and execution of network related objects to use the TCP Networking Protocol, such as the TCP Client and Listener, Binary Reader and Writer, and network stream.

The system is created using TCP methodology. The server acts as the “listener” so it is the authoritative node within the local network; clients act as the “Clients” since this provides the data the “Listener” will respond to.

Having initialised the backend system foundation, introducing packets was crucial to functionality, being the only optimised method of distributing data along a network. Different packet types were created to write data to stream, for the server to receive and read this data using a reader, applied from the Write and Read function located in a packet class.

Utilising packet methodology to distribute data, each packet type was assigned to handle client-side data, from which when sent, the server processes packet type to generate appropriate responses to that data type. For example, Global message packet when processed, distributes message data with the client’s name to all other connected clients stored in the concurrent dictionary. This sends packet data to other clients, and they update accordingly.

Since clients’ instances are stored within the server, using an interface class “connectedclient” that acts as the client, accessing the client’s instances is achievable through the concurrent dictionary, utilising a dictionary key added to each client to send data to. Instantiation of new packet types presented no issues; logic is transferable due to minimal data storage and processing, which are processed using the same methods.

Sending the encrypted packet type was done by using the public key encryption system provided by RSA Encryption services to encrypt data upon the closed local network. With the encryption limits based on key size, packet data undergoing encryption had to be below 1024bits, since encryption increases size of packet data to fit the key size, 2048 bits.

Decrypting encrypted packets required server adjustment to gain access to packet type for standard processing. Data had to be encrypted and decrypted in data segments, of 128 bytes or 1024 bits, and 256 bytes or 2048 bits, respectively. This knowledge was adapted into the creation of two functions to encrypt and decrypt packets. To ensure requirements are met, all data encrypted was small, being delegated to string and int list types.

#### Threading:

Threading is required for safe, correct execution of the application. This is because the network is hosted locally, with multiple instances of the same client project on a single computer. Assigning each client to a new thread stores that client instance bound to a thread within a concurrent dictionary, in the thread safe storage system to handle the correct execution of multiple threads.

With resources and functions, packet read and write, being accessed by multiple clients, locks are required to prevent risks of race-conditioning issues.

Furthermore, to correctly execute WPF controls, update client interactive data, and invoke statements for each instance in which WPF Objects are updated, “Dispatcher.Invoke” must be called upon to ensure the thread bound to the client is correctly called to allow all clients invoked from the server with the distributed packet to update to the client accordingly.

#### WPF & Event Driven Programming:

WPF was used to handle client side and event driven interactions, a design drafted based upon chat system requirements, keeping the design simple since changes to integral parts of the application could have been a possibility. However, eventually, with improved understanding of WPF, event driven invoked functions were included, giving controls true functionality.

The first function was a global chat response when a client application had connected to the server. With this success, the addition of a chat system, both private and global, a disconnect and reconnect system, client list system, and change username system were all included.

Each WPF control differed in suitability for the requirements. For instance, the client list was originally intended to be a textbox, however after finding the alternative control “ListBox”, this was determined more suitable with its interactive abilities regarding its item list and was used instead.

Since WPF objects invoke event driven responses, based on the relevant control used, these are set according to context to its name. Buttons typically hold the most influence in decision making, since upon invoking, an action is to occur, for example, clicking the “send message” button sends a message based upon parameters, clicking “disconnect” ends client connectivity, and “reconnect” sends someone back to the login form to start a new client instance.

These all act using the packet system, where packets are sent while updating locally and accordingly, to save bandwidth sending redundant data. Packets are sent based upon the control and event invoked, a message sent will invoke either a private or global chat packet. All these objects are therefore integral to the connectivity to client-to-client interactivity.

# Conclusion

Overall, the development went well, with few issues arising from incorrect processing of client data within the server, especially processing disconnection. Further issues encountered during the development of the graphical solution include processing incorrect synchronization of new connected game players within the canvas and encryptions errors, frequent bad length errors.

Things that went well were the ease of repeatable logic in system creation, particularly within the packet system. Moreover, the simple text-based game only encountered minor issues , being the processing of the guess & correct output update from the guess. The dynamic client list, while challenging to keep consistent, also worked.

In the future, I would change the system for the graphical solution, as I am still figuring out the process of adding and synchronising the correct movement of objects between clients. Furthermore, I can include the addition of identifiable traits. With the last minute attempt being poorly designed with major errors.

# Bibliography

**Website(RSA Encryption Solution)**:

**StackOverflow (Oct 2010) [Online] Available From:**

<https://stackoverflow.com/questions/1496793/rsa-encryption-getting-bad-length>

**[Accessed December 27th 2021]**

**Website(Public Key Encryption)**:

**Tutorials Point [Online] Available From:**

<https://www.tutorialspoint.com/cryptography/public_key_encryption.html>

**[Accessed December 26th 2021]**

**Video(WASD Rectangle Movement & Canvas Tutorial)**:

**Moo ICT [Online] Available From:**

<https://www.youtube.com/watch?v=kSoVL6MuL5o&t=760s>

**[Accessed January 2nd 2022]**

**Website(List Box Tutorial)**:

**Mahesh Chand [Online] Available From:**

[**https://www.c-sharpcorner.com/uploadfile/mahesh/listbox-in-wpf/#:~:text=WPF%20ListBox%20Tutorial%201%20Introduction%20to%20WPF%20ListBox,Control.%20...%209%20Sorting%20a%20ListBox%20Items.%20**](https://www.c-sharpcorner.com/uploadfile/mahesh/listbox-in-wpf/#:~:text=WPF%20ListBox%20Tutorial%201%20Introduction%20to%20WPF%20ListBox,Control.%20...%209%20Sorting%20a%20ListBox%20Items.%20)

**[Accessed December 21th 2021]**

# Appendices

## How to use Application:

CHANGE THE SOLUTIONS STARTUP PROJECTS TO START UP BOTH CLIENT/SERVERPROJ.

To add new client, right click on “ClientProj” click debug and start new instance.

Main Form Buttons:

List of button with their given purposes:

* “Send” – Sends messages contained in textbox left hand side of button placement
* Disconnect – Terminates Client Service
* Reconnect – Instantiate new Connect Form instance to allow connection to service
* Change Username – Opens Form with the ability to change username
* Play Game – Opens canvas to move square and other objects

PLAY GAME NOTE (FOR THIS TO WORK IN ITS OPTIMAL WAY , THE FIRST CLIENT OPENED SHOULD CONNECT TO GAME FIRST)!

* Exit Game - Ends Canvas instance

Text Commands:

* “/Play Hangman” – Invokes hangman game to start
* “/Guess ‘single character’ ” – Process Hangman Guess
* “/Help” – Opens Help form , used for additional support

Other Features:

Send Private Messages – Click username listed on Client List (List Box) , this invokes private chat mode, must send message.

## Appendix 1: RUDIMENTARY SOFTWARE DOCUMENTATION

I always create rudimentary software documentation, I use this to outline features, functions that I implement into the project.   


NOTE – NOT FULLY UP TO DATE WITH ALL FEATURES PRESENT.