# Chat App

## Objectives (initial version)

* Provide a C#.NET WPF desktop application that serves as a bi-directional chat window.
* Use System.Net.Sockets API for tcp and udp interaction between client and server.
  + tcp unicast for internal LAN-based use or for communication over internet.
  + udp multicast only for internal LAN-based use, not recommended over internet.
* App is not cloud-based and does not write to any database.
* There is no user login, user tracking or telemetry.
* Chat logs are not persisted or exported. All chat communication is deleted on close of application.
* (future version) Internal LAN network communication and/or internet-based communication privacy to be provided via two user-selectable options: symmetric encryption and asymmetric encryption/PKI.

# A. Functional Specifications

## 1. Launch App, Establish Connectivity

Two windows must run at the same time, with different roles.

* Window #1 - Run a tcp server which binds to local port and accepts socket connections. The tcp server must be launched first (before the tcp client).

TCP Connection State will change from Red <unassigned> to Green <connected>

* Window #2 - Run a tcp client which initiates a connection to the tcp server host ip/port. The tcp client must be launched second (after the tcp server).

TCP Connection State will change from Red <unassigned> to Green <connected>

**On launch, before connection is established:**

**TCP Server**

A screenshot of a computer

Description automatically generated

**TCP Client**

A screenshot of a computer

Description automatically generated

**Initiate tcp connection between “server” and “client”:**

User A must select TCP radio button, “Run as tcp Server” checkbox, then click “Intiate Connection” and then “Begin Listening”:

A screenshot of a computer

Description automatically generated

User B must select TCP radio button, “Run as tcp Client” checkbox, then click “Initiate Connection” and then “Begin Listening”:

A screenshot of a computer

Description automatically generated

A TCP connection will be established.

**After connection is established:**

**TCP Server**

A screenshot of a computer

Description automatically generated

**TCP Client**

A screenshot of a computer

Description automatically generated

## 2. Initiate Message Transmission

Fill-out TextBox window with desired message under “Type Message Here” label.

Click “Send Msg”

**Sending window chat Log is updated with outgoing message.**

A screenshot of a computer

Description automatically generated

**Receiving window chat log is updated with incoming message:**

A screenshot of a computer

Description automatically generated

# 3. Privacy/Encryption

Message Encryption will be supported in Chat.exe in a future release version. This will be a configurable option.

Two encryption options will be available to the user:

* symmetric (initial version)
* asymmetric / PKI (future version)

# B. System Design

Software Design: The activity following requirements specification and before programming. The stylized software engineering process.

This is a “simple” application so using MVVM is not necessary. The basics of the design pattern is presented here for future consideration as the application complexity grows with the addition of new features.

## Design pattern: Model-View-View-Model (MVVM)

MVVM NuGet Package: “CommunityToolkit.Mvvm”

* Model (DATA): Business Objects that encapsulate data and behavior of application domain
* View (User Interface): what the user sees
* Viewmodel (LOGIC): Link between Model and View OR It Retrieves data from Model and exposes it to the View. This is the model specifically designed for the View.

A diagram of a model

Description automatically generated

**FEATURES:**

* Life Cycle state of Application will be maintained.
* The application will be in the same position as where the user left it.
* UI Components are kept away from Business Logic.
* Business Logic is kept away from Database operations.
* Easy to understand and read.

**BASIC EXAMPLE:**We want to display Name in Purple Color (not written in the proper format, proper length) or Display Purple Color if Age of a person is > 18 years, Display Pink Color if Age of a person is < 18 years, Then the Logic of Purple and Pink Color would be present in ViewModel.

**SUMMARY:** From Server, Get Data(available in Model Objects), View Model reads Model Objects and then facilitates the easy presentation of data on the view.

The primary differences between MVVM AND MVC are as follows:

|  |  |
| --- | --- |
| **MVVM** | **MVC** |
| The Model is somewhat similar to MVC but here we have ViewModels which are passed to the view and all the logic is in the ViewModel and hence no controller is there. Example: Knockout.js | In this pattern, we have models which are basic objects with no code and just properties, views that contribute to presentation items (HTML, WinForms, etc), client-side deletes, and Controllers that focus on the logic part. Examples: <http://ASP.NET> MVC, Angular |
| In MVVM your DeletePerson would be called off of your view model | We have a PersonController with an Action DeletePerson that delete a person |
| We are on the client side so we can hold on to objects and do a lot more logic in a non-disconnected state. | MVC is typically used when things are transactional and disconnected as is the case with server-side web. In ASP MVC we send the view through the wire and then the transaction with the client is over. |

**ADVANTAGES:**

* Maintainability – Can remain agile and keep releasing successive versions quickly.
* Extensibility – Have the ability to replace or add new pieces of code.
* Testability – Easier to write unit tests against a core logic.
* Transparent Communication – The view model provides a transparent interface to the view controller, which it uses to populate the view layer and interact with the model layer, which results in a transparent communication between the layers of your application.

**DISADVANTAGES:**

* Some people think that for simple UIs, MVVM can be overkill.
* In bigger cases, it can be hard to design the ViewModel.
* Debugging would be a bit difficult when we have complex data bindings.

Ref: [Introduction to Model View View Model (MVVM) - GeeksforGeeks](https://www.geeksforgeeks.org/introduction-to-model-view-view-model-mvvm/)

## ****Design pattern: Model-View-Controller (MVC)****

* is this for web apps only? if so, this does not apply here (for now)

# UI/UX

Under construction.

# Threading model

Thread: UI (main)

* for GUI logic and updating/interacting with GUI controls

Thread: receive\_message()

* used by tcp server and tcp client to enable bi-directional messaging.
* NetworkStream.Read() is a blocking operation so this thread runs an infinite loop. This is why this must run as a separate thread.

Thread safety considerations

* under construction

# Classes

public class tcpClass

{

// data

string role { get; set; }

string client\_ip\_and\_port { get; set; }

string client\_ip;

string client\_port;

string server\_port { get; set; }

bool result\_read;

bool result\_write;

public Socket handler = null;

public TcpListener server = null;

public Int32 port = 1000;

public IPAddress localAddr = IPAddress.Parse("192.168.0.54");

public Byte[] bytes = new byte[256];

public String data = null;

public TcpClient client = null;

// methods

public tcpClass(string role, string client\_ip\_and\_port) { }

public ~tcpClass() {}

public async void initiate\_connection\_tcp\_server(){ }

public bool initiate\_connection\_tcp\_client() { }

public bool send\_message() { }

public bool receive\_message( ) { }

}

Chat.exe app only requires a single class for implementing all required aspects of the System.Net.Sockets API. A single object is instantiated for either of the tcp server or tcp client roles.

# GUI component accessibility to business logic

Using static variables as aliases now. Need to consider 2-way binding in a future release.

Description:

public partial class MainWindow : Window

{

public static Button StatusButton;

public static TextBox TextBox\_Msg;

public static TextBlock TB\_local;

}

public MainWindow()

{

InitializeComponent();

StatusButton = Button\_CxnState;

TextBox\_Msg = TextBox\_enterMessage;

TB\_local = TextBlock\_messages;

}

# C. Development

C# v7.0

.NET Framework 4.7.2

### Development testing considerations

* Development must adopt a test-driven approach (“TDD”). This is in progress.
* Development must include a robust unit-test framework. This to be created using “MSTest.” This is in progress.

# D. Testing/Validation

* GUI test automation to be created for this project, using the Microsoft UI Automation (MUIA) library.
* Continuous Integration (CI) to be created using git hooks so that any commit or branch merge will automatically trigger building and checkout of the app, and execution of the GUI automation framework.

# E. Deployment

### Application Delivery

* The software delivery mechanism will be Sync which provides secure delivery of the Chat.exe binary through email via a secure link.
* The end user will provide their email address. A sync secure link email will be sent to the user by Sync.
* Here is a sample email notification that is provided from Sync:

A screenshot of a computer

Description automatically generated

When link is selected for download here is the URL that presents to the user:

A screenshot of a computer

Description automatically generated

### <<<IMPORTANT >>> Expect Windows Defender, Norton and other Internet Security systems to flag the executable when the download link is selected.

* Make sure your internet security software is configured to recognize and allow (trust) use of the executable.
* Make sure your internet security software is configured to allow outbound and inbound tcp activity to/from the target server.

### <<<IMPORTANT>>> setup instructions only for GUIs running in the tcp “server” role

* The host running the tcp server role must setup a port forwarding rule in the cable modem/router (public IP to host ip/port) to allow inbound tcp connection requests to the target host.

Example port forwarding rule setup in a Motorola Cable modem/router:

A screenshot of a computer

Description automatically generated

### Licensing

* Option A: EULA
* Option B: Develop a license server that contains a registration list of all internal IPs. Chat.exe app to retrieve the internal host IP and pass it in an initial handshake message to the license server. License server will check the IP against the internal list.
  + If present, an “OK” reply is sent and user can continue with use of the app.
  + If not in list, a “NOT OK” response is sent and the GUI will present a pop-up indicating unlicensed use of app was attempted.