

## Zusammenfassung der Arbeit

### Abstract of Thesis

Fachbereich: **Electrical Engineering and Computer Science**

Department:

Studiengang: **Information Technology**

University course:

Thema: **Investigation and Test of Xamarin.Forms Client to Client**

Subject: **communication methods**

Zusammenfassung:

Abstract:

With the rapid development of mobile internet technology and the popularity of smart mobile terminals, live chatting with mobile devices based on different platforms has become an indispensable part of people's lives. Cross-platform development is becoming a trend. Xamarin.Forms provide a solution for cross-platform programming which is integrated in Visual Studio 2017. The aim of this thesis is to create a real-time communication application between Android, IOS, windows devices and raspberry pi 3 by using Xamarin.Forms. To reach the final goal, we select MQTT protocol which is a lightweight agent-based publish/subscribe messaging protocol to realize the communication function for its simple, lightweight nature and is well suited for use on cross-platform mobile devices. This thesis will introduce the basic knowledge of Visual Studio 2017, Xamarin.Forms, MQTT protocol and the programming language(C#). With the analysis and study of the format characteristics of the MQTT protocol and how the protocol works, this paper designed and implemented a instant messaging application for cross-platform mobile terminals which includes Android, IOS, UWP and raspberry PI, and this application achieved log/ register, instant message, state rendering function.

Verfasser: **Zun Yuan**

Author:

Betreuender Professor/in: **Prof. Dr. Jörg Bayerlein**

Attending Professor:

WS / SS : **SS <2019>**

# Table of Contents

<b>Abstract of Thesis .....</b>	<b>i</b>
<b>Chapter 1 Introduction .....</b>	<b>1</b>
1.1 Motivation .....	1
1.2 Goal .....	2
1.3 Thesis structure .....	2
<b>Chapter 2 Basic Knowledge.....</b>	<b>3</b>
2.1 Visual Studio .....	3
2.2 Xamarin.Forms .....	3
2.3 SQLite.....	4
2.4 Raspberry Pi .....	5
<b>Chapter 3 Installation and Configuration .....</b>	<b>7</b>
3.1 Visual Studio Installation .....	7
3.2 Emulator Configuration .....	9
3.2.1 UWP Emulator Configuration .....	9
3.2.2 Android Emulator Configuration .....	10
3.3 IOS Compilation and Connection .....	11
<b>Chapter 4 MQTT protocol.....</b>	<b>13</b>
4.1 Introduction and Working Principle.....	13
4.2 Publish and Subscribe models .....	14
4.3 Quality of Service.....	15
4.4 MQTTnet .....	17
<b>Chapter 5 Project Programming .....</b>	<b>18</b>
5.1 Directory Description.....	18
5.2 Create a Project and Import the Class Library.....	19
5.3 Source Code Description .....	20
5.3.1 Local Storage API .....	21

5.3.2	Pop-up Box API .....	21
5.3.3	Interface Design .....	22
5.3.4	Client .....	26
5.3.5	MQTT Server .....	29
5.4	SQLite Programming .....	31
5.4.1	Building Library .....	32
5.4.2	Additions, Deletions, Changes, etc .....	33
5.4.3	File Call .....	34
5.5	Software Operation Guide.....	35
<b>Chapter 6</b>	<b>Summary .....</b>	<b>38</b>
6.1	Conclusion and Evaluation.....	38
6.2	Outlook .....	39
<b>Acknowledgment</b>	<b>.....</b>	<b>41</b>
<b>Appendix A – List of figures</b>	<b>.....</b>	<b>42</b>
<b>Appendix B – List of code</b>	<b>.....</b>	<b>44</b>
<b>Appendix C – Content of USB Striker</b>	<b>.....</b>	<b>45</b>
<b>Bibliography</b>	<b>.....</b>	<b>46</b>

# Chapter 1 Introduction

## 1.1 Motivation

With the development of mobile internet technology, instant messaging has become one of the most commonly used communication tools. It's necessary for people to exchange message on different platform.

The implementation of the mobile instant messaging system mainly uses the XMPP protocol and the SIMPLE protocol except for a few proprietary protocols in current market. The XMPP protocol is an XML-based open instant messaging protocol which is mature, secure and scalable, but it has the disadvantages of complicated protocol, repeated message forwarding, power consumption and fee flow [1]. This is a design and implementation of a high energy consumption agreement. The SIMPLE protocol is extended on the basis of the SIP protocol [2] and is one of the mainstream instant messaging protocols. The mature audio and video standard support various types of instant messaging, but it also has problems such as large traffic consumption and complicated expansion. These two protocols are not designed with the characteristics of the cross-platform terminals, so they do not perform well in practical applications. MQTT is a protocol designed for devices with limited computing power and working in low-bandwidth and unreliable networks. Its characteristic of small transmission, consuming less power, reducing network traffic and minimizing data packets give us a suitable option for cross-platform instant messaging system on mobile devices.

## **1.2 Goal**

The former research is on developing a communication method via email exchange and data base connection, but these way does not work fast enough. MQTT (Message Queuing Telemetry), a lightweight messaging/delivery-based messaging protocol, is mobile ideal for end applications. This thesis is to introduce the basic content and features of MQTT and test it on UWP local machine, Raspberry, Android and iPhone. Besides, it is to use mobile phones as remote control for control systems. As final App a communication App whose functions include register/login, instant messaging and status display should be the result.

## **1.3 Thesis structure**

This thesis is divided into six parts to help the reader understand the research and content to be expressed throughout the paper.

The first chapter is to introduce the motivation and purpose of doing this research.

In chapter 2 is to presents the basic knowledge of Visual Studio, Xamarin, SQLite and Rasberry Pi so that readers can understand the research and content afterwards better.

Then in chapter 3, there is the guide of Visual Studio installation and the emulator configuration of different platform.

Chapter 4 aims to introduce MQTT protocol which is most critical in this paper. It is to help readers to understand the content and logic of MQTT protocol and pave the way for the practical application of MQTT protocol in instant messaging application in next chapter

Chapter 5 will specifically introduce how each part of the program is constructed and its role.

Finally, in the chapter the result and conclusion are summarized and the limitation and possibilities are demonstrated. After that there is a outlook for future research.

## **Chapter 2 Basic Knowledge**

This chapter is mainly introducing the basic knowledge of the integrated development environment, cross-platform application development framework, database and hardware in order to provide a primary impression of the technology and let the reader understand what the thesis is writing about better.

### **2.1 Visual Studio**

Microsoft Visual Studio is an integrated development environment created by Microsoft which is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio supports 36 different programming languages such as C, C++, JavaScript, HTML and Python. Besides, the Community edition is available free of charge which is the most basic edition of Visual Studio. [3] The Visual Studio version used in this paper is 2017.

### **2.2 Xamarin.Forms**

In our inherent thinking, we believe it's a reliable and conservative way to hire developers who has specific skills to be responsible for the part where they are good at. For example, some developers use Objective-C to develop IOS version, some developers use Java to develop Android version and some developers use C# to develop windows version. However, if we develop a cross-platform in such way, it will lead to a lot of troubles and mistakes such as inconsistence and non-compatibility. This is a huge loss and waste of resources and time. To solve this kind of problem, a cross-platform technology which called Xamarin is created as the key to productivity

and the old programming model which isolate IOS, Android and Windows is gradually eliminated. [4]

Xamarin.Forms provides a complete cross-platform UI product for .NET developers and uses the C# in Visual Studio to build completely native Android, iOS and universal windows platform applications [5].

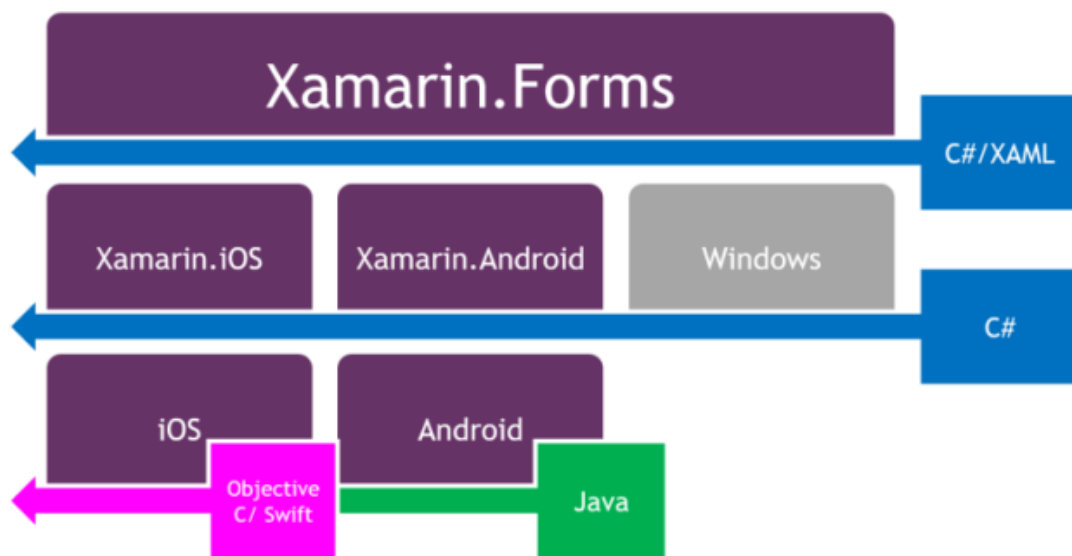


Figure 2-1 Xamarin.Forms [7]

## 2.3 SQLite

SQLite is a process library which implements a self-contained, serverless, zero-configuration and transactional SQL database engine [6]. SQLite is free for use for any purposes such as private and business since the code for SQLite domain is in the public domain.

Unlike most other SQL databases, SQLite is an embedded SQL database engine and does not have a separate server process which enable it to directly read and write ordinary disk files. Besides, SQLite is a compact library whose size may be less than

600KiB depending on the settings of target platform and compiler and it runs faster when giving it more memory. SQLite perform quite good in a low memory environment [6].

## 2.4 Raspberry Pi

Raspberry Pi is a mini computer for computer amateurs, teachers, students and small companies. It is pre-installed with Linux system and its size is as big as a credit card. It is equipped with an ARM architecture processor and its computing performance is similar to a smartphone.



Figure 2-2 Raspberry Pi 3 Model B [8]

The version of Raspberry Pi using in this thesis is Raspberry Pi 3 Model B+ and it is the latest revision of third-generation single-board computer which contains 1.4Hz



64-bit quad-core processor, dual-band wireless LAN, Bluetooth 4.2/BLE, faster Ethernet and Power-over-Ethernet support [8]. All in all, this product is the perfect foundation for building professional applications or interesting projects.

If we want to connect to raspberry, we could use the help of IoT (internet of things) . If IoT system has already been installed on the raspberry, we can connect to the IP address of the raspberry its own and make the preparation for next work.

## Chapter 3 Installation and Configuration

Microsoft Visual Studio 2017 which is integrated with Xamarin.Forms is the key tool to develop the cross-platform program in this thesis. This chapter mainly introduces the installation of the Visual Studio 2017 and the configuration in each emulator.

### 3.1 Visual Studio Installation

The first step we should do is downloading the Visual Studio Installer and use it to install the Visual Studio 2017 which is available in the Microsoft Visual Studio webpage [9]. The following pictures show the version and the components we need to be installed.

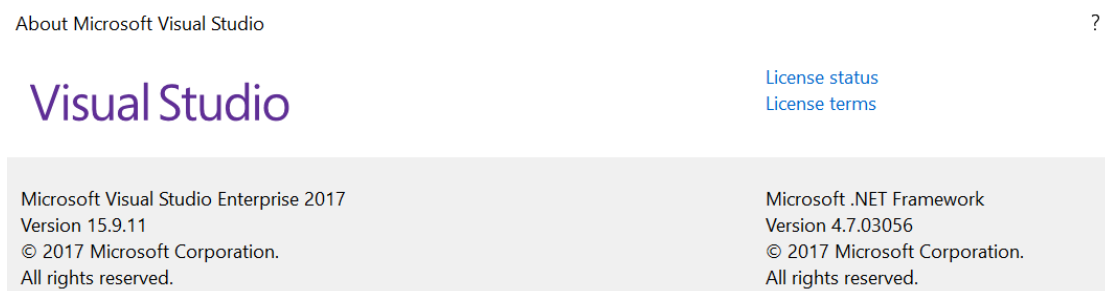


Figure 3-1 Version of Visual Studio

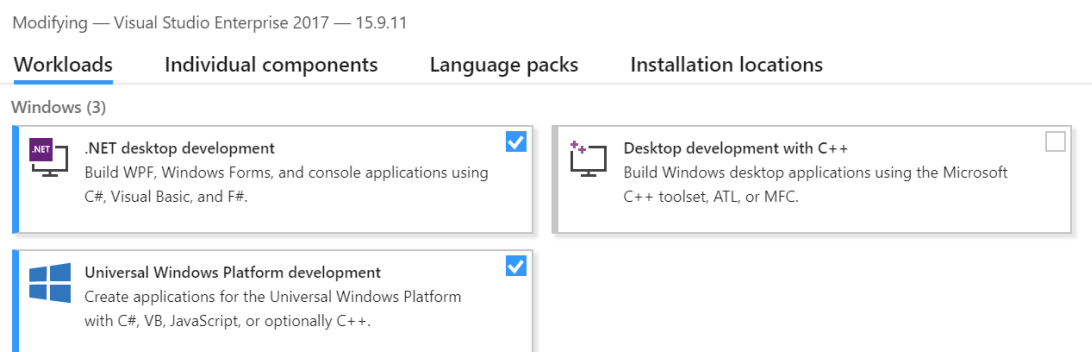


Figure 3-2 Installed Component in Visual Studio 2017 1

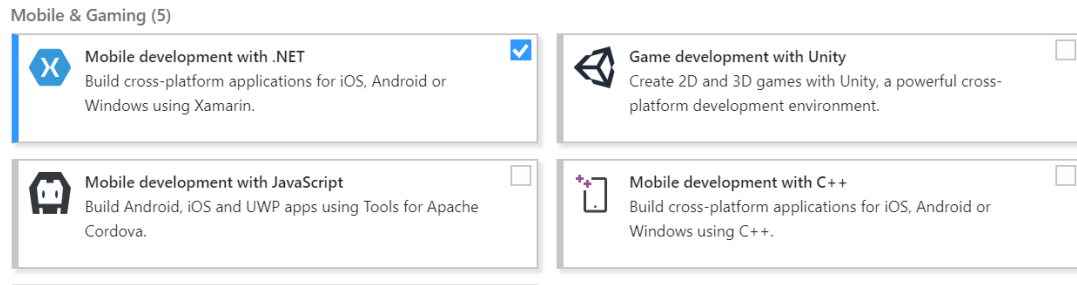


Figure 3-3 Installed Component in Visual Studio 2017 2

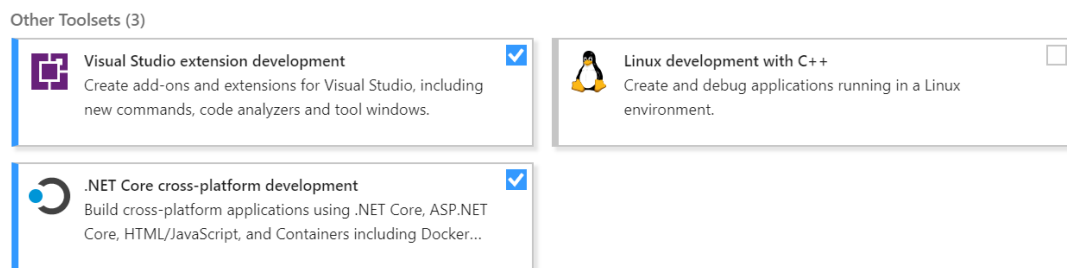


Figure 3-4 Installed Component in Visual Studio 2017 3

## Installation details

- > Visual Studio core editor
- > .NET desktop development
- > Universal Windows Platform development
- > Mobile development with .NET
- > Visual Studio extension development
- > .NET Core cross-platform development
- ✓ Individual components
  - ✓ C# and Visual Basic Roslyn compilers
  - ✓ Static analysis tools
  - ✓ C# and Visual Basic
  - ✓ Windows Universal CRT SDK
  - ✓ Windows 10 SDK (10.0.16299.0) for UWP: C#, VB, JS
  - ✓ Windows 10 SDK (10.0.15063.0) for UWP: C#, VB, JS
  - ✓ Windows 10 SDK (10.0.14393.0)
  - ✓ Windows 10 SDK (10.0.10586.0)
  - ✓ Windows 10 SDK (10.0.10240.0)
  - ✓ .NET Core runtime
  - ✓ .NET Compiler Platform SDK

Figure 3-1-5 Installation Details of Visual Studio 2017

Then the Visual Studio 2017 can be used to develop cross-platform software when creating the project as “Mobile App (Xamarin.Forms)” in “Cross-Platform” section in “Visual C#”.

## 3.2 Emulator Configuration

Suitable and powerful emulators helps us to verify our application at lightning speed depending on its powerful simulation to imitate the actual situation through various device sensors. The X86 emulator starts and runs at almost the speed of a physical device, making it easy to debug graphics-intensive, processor-intensive applications [10].

### 3.2.1 UWP Emulator Configuration

Before starting to develop UWP applications, we need to find “Settings” – “Updates and Security” – “For developers” on Win 10 PC, first switch the model under “Use developer features” to “Develop mode”, and then turn “Device Portal” status is changed to “On”.

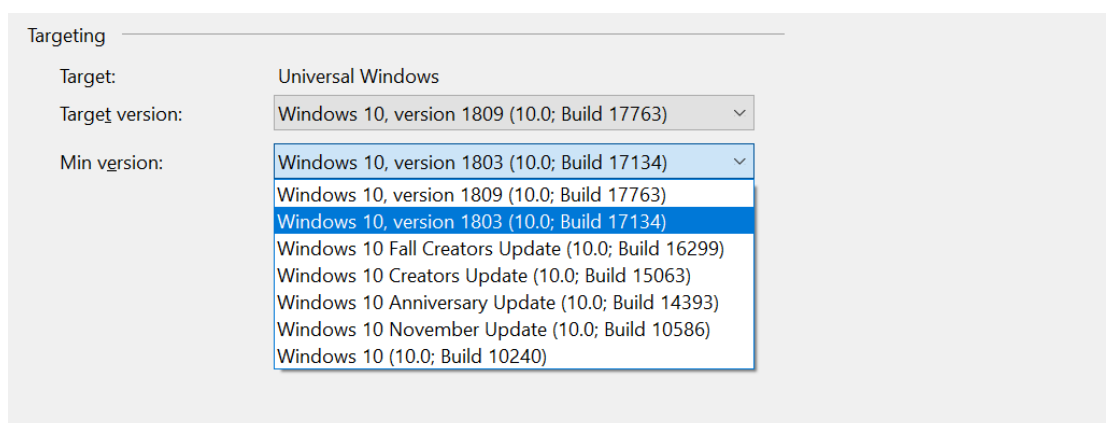


Figure 3-5 UWP Targeting SDK

In this thesis, we set target version as the current latest version of Windows 10, version 1809(10.0; Build 17763) and use Windows 10, version 1803(10.0; Build 17134) as the

min version.

### 3.2.2 Android Emulator Configuration

Win10 system has a standalone version of the Android simulator Visual Studio Emulator for Android. I will give detailed introduction about this emulator settings and how to use it.

First, turn on Hyper-V virtualization technology which in the “Start or Shut Down Windows Features” in “Uninstall a Program” in “Control Panel. Then download the Visual Studio Emulator for Android on the website: <https://visualstudio.microsoft.com/vs/msft-android-emulator/> [22].

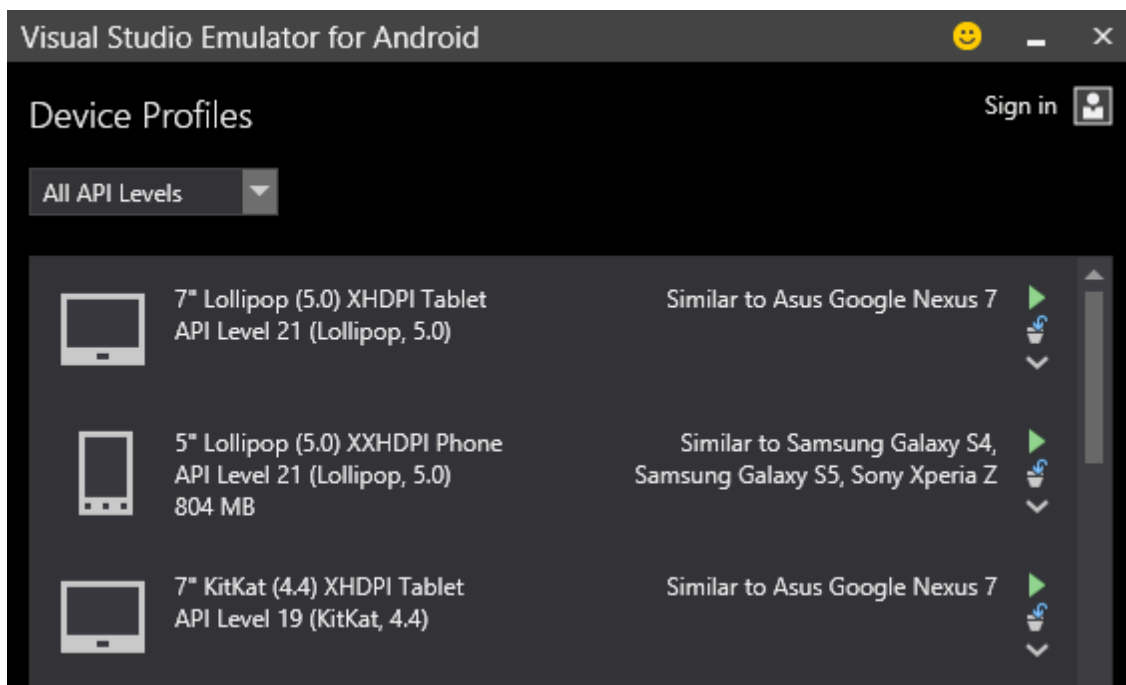


Figure 3-6 Visual Studio Android Emulator

After downloading, it is a 40M vs\_emulatorsetup.exe file, which does not contain the file of the Android emulator. The x86 file of API 19 (Android 4.4) will be downloaded online by default. We can also download other APIs separately. In addition, Visual Studio Emulator for Android is not dependent on Visual Studio, which means that it can

be installed separately. Android Studio uses Android Debug Bridge as a bridge to connect Visual Studio Emulator for Android.

Then look into Android SDK manager, there is a list of Android platforms and I choose the Android 8.1-Oreo and click apply changes. Now I can run the application of Android version in my Android emulator.

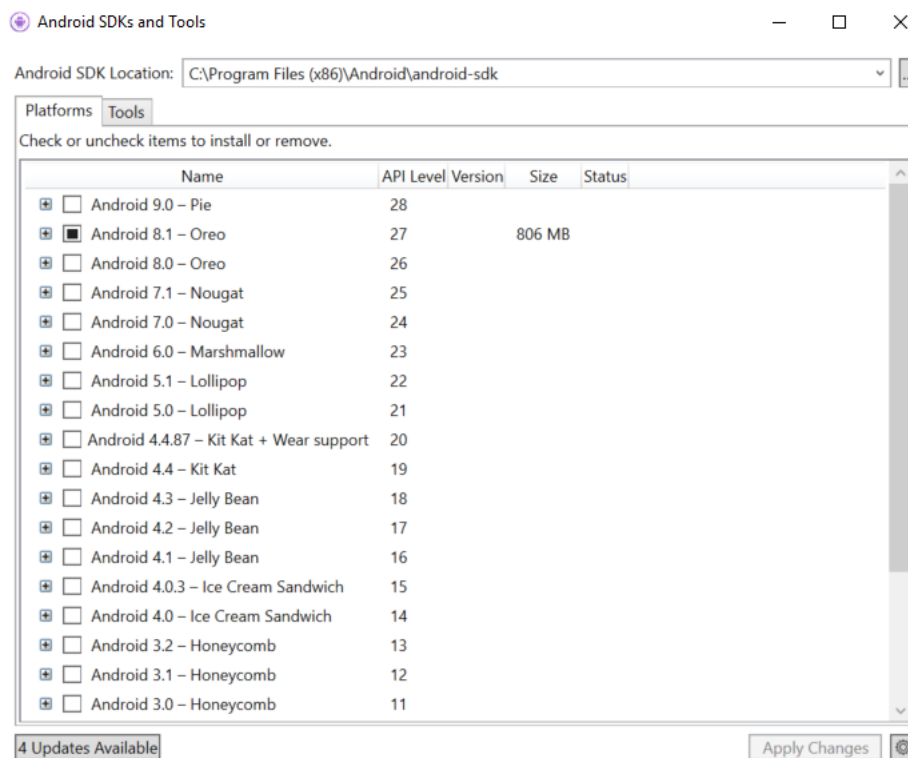


Figure 3-7 Android SDK Manager Platform

### 3.3 IOS Compilation and Connection

We need a Mac computer and Xcode installed on it to compile the IOS version of the software. If there is difficult on compiling on a real mac computer, you can try to install a virtual machine and install the Mac system on it to compile the software.

We need to make sure that this computer has already installed the Xcode and Visual

Studio 2017 for Mac, then we can start connecting. First is to open the Info.plist in the Visual Studio to set the application name and bundle identifier. After that confirm and choose the Mac computer we are going to use.

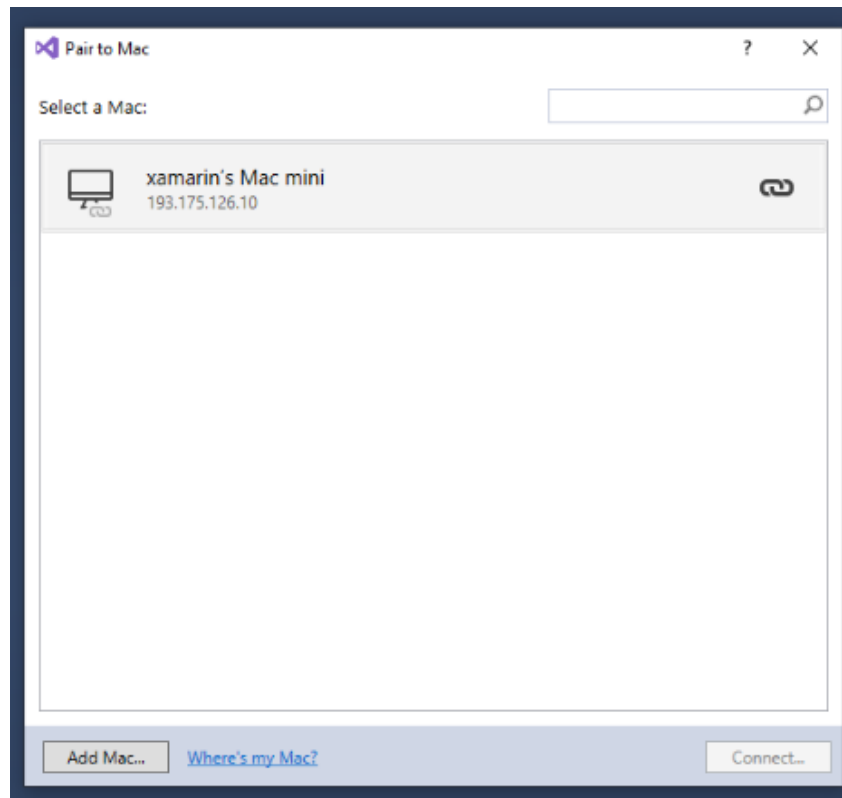


Figure 3-8 Pair to Mac

In the end, we just need to run the application of IOS by clicking the run button.

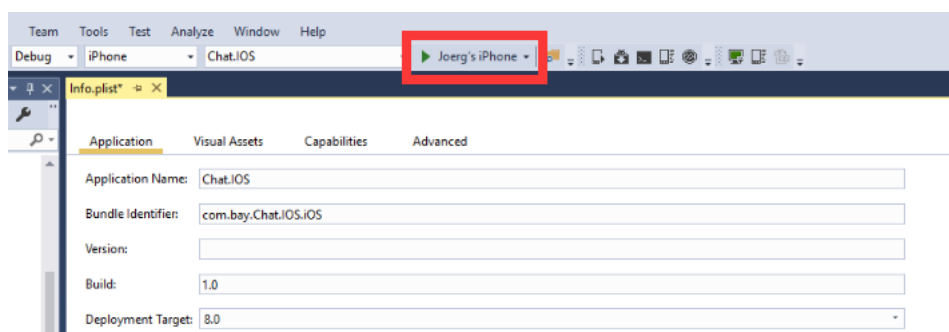


Figure 3-9 Connect Real Device

# Chapter 4 MQTT protocol

## 4.1 Introduction and Working Principle

The MQTT protocol (Message Queuing Telemetry Transport) was proposed by IBM in 1999 and the latest version is 5.0 [11]. MQTT is a message protocol based on binary message-based publish/subscribe programming mode. Nowadays it has become the OASIS specification. Because the specification is very simple, the original purpose of the design is to provide extremely limited memory devices and unreliable communication with low network bandwidth. Suitable for IoT scenarios that require low power consumption and limited network bandwidth [12].

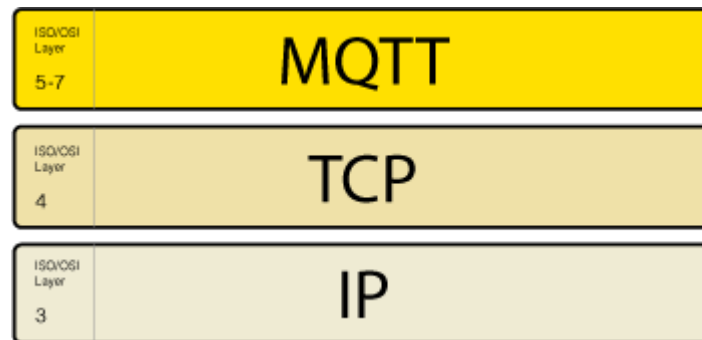


Figure 4-1 MQTT Protocol Stack [13]

The MQTT protocol provides one-to-many message publishing, which can decouple application coupling and reduce information redundancy. The protocol requires a client and a server, and there are three main identities in the protocol: Publisher, Broker, Server, and Subscriber. Among them, the publisher and subscriber of the message are both clients, the message proxy is the server, and the message publisher can be the subscriber at the same time, which realizes the decoupling between the producer and the consumer.



## 4.2 Publish and Subscribe models

The MQTT protocol defines two entity types in the network: a message broker and some clients. A proxy is a sever that receives all messages from the client and routes them to the relevant target client. A client is anything that can interact with an agent to send and receive messages. The client can be a live IoT sensor or an application that processes IoT data in the data center. First the client connects to the proxy and it can subscribe to any message “subject” in the agent. Then the client publishes messages within a subject by sending messages and topics to the agent. At last the agent forwards the message to all clients that subscribe to the topics [12] and we can see the procedure in the following picture.

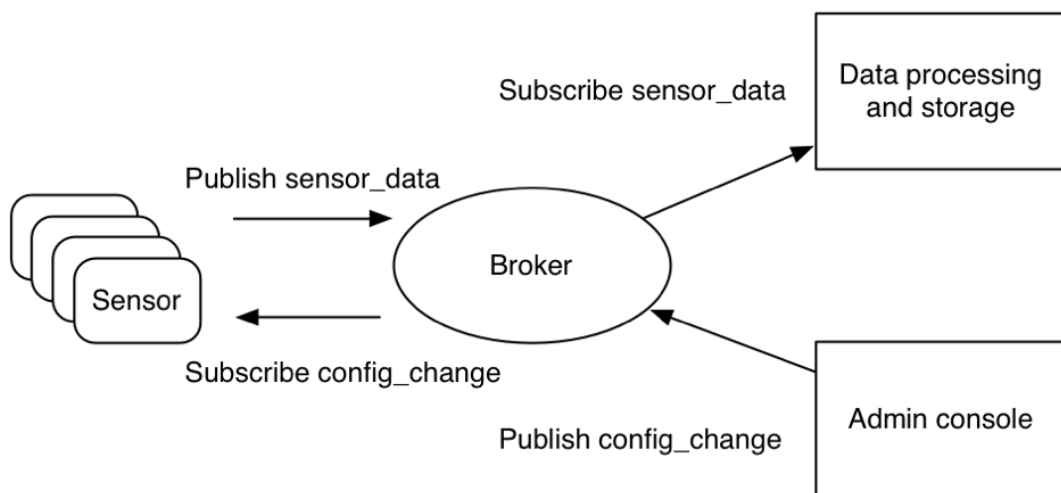


Figure 4-2 MQTT Publish and Subscribe Model for IoT Sensors [12]

The publish/subscribe model implements decoupling between publishers and subscribers and can be distinguished from multiple dimensions. The first one is spatial decoupling which means publishers and subscribers do not need to know each other's existence (such as the other party's IP address and port). The second decoupling is time decoupling which means publishers and subscribers do not need to run at the same time. The third one is synchronous decoupling which means the operation of both components is not suspended during the release or reception. In general, the

publish/subscribe model decouples publishers and subscribers from messages, and by filtering the messages, only certain clients can receive the corresponding messages. Decoupling consists of three dimensions: space, time and synchronization [14].

### 4.3 Quality of Service

MQTT supports three QoS levels:

QoS 0: “At most once”, message publishing relies entirely on the underlying TCP/IP network. Messages distributed may be lost or duplicated [15]. For example, this level can be used for environmental sensor data and a single data loss does not matter, as there will be a second transmission in the near future.



Figure 4-3 QoS 0 [16]

QoS 1: “At least once” to ensure that the message can arrive, but the message may be repeated [15].

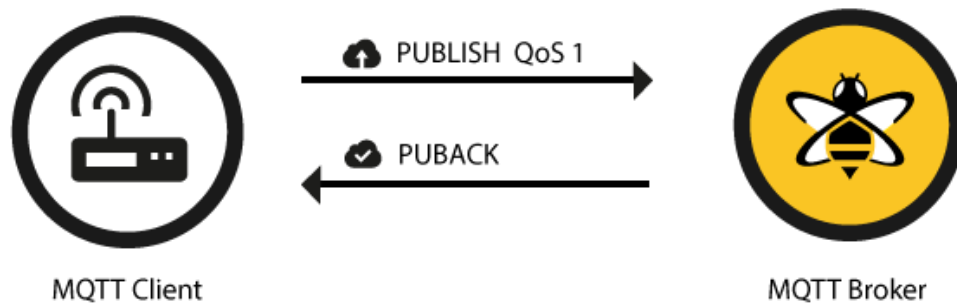


Figure 4-4 QoS 1 [16]

QoS 2:” Only once”, ensuring that the message arrives only once [15]. For example, the level can be used in a billing system where incorrect or lost messages can result in incorrect charges.

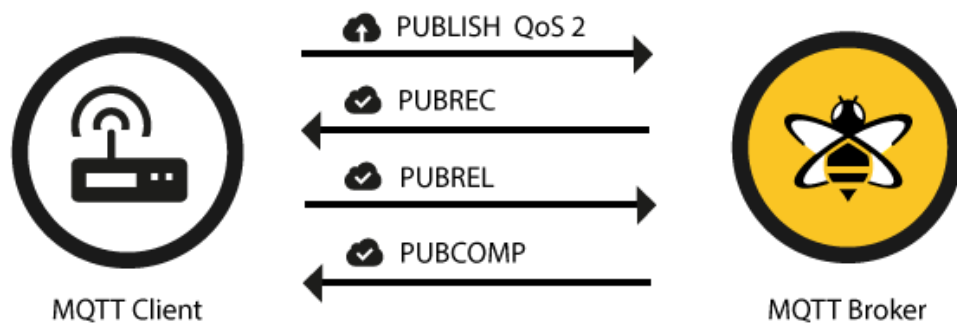


Figure 4-5 QoS 2 [16]

QoS is a major feature of MQTT. It makes the exchange of information in an unstable network environment much simpler, because the protocol controls the relay and guarantees the delivery of information, ignoring the unreliable underlying interaction. Moreover, it authorizes the client to determine the QoS level based on the client’s program logic and network reliability.

## **4.4 MQTTnet**

MQTTnet is a high-performance .NET open source library based on MQTT communication that supports both MQTT server and client [20]. And the author is also updated, currently supporting the new version of the .NET core, which is why MQTTnet is chosen. MQTTnet is not the most downloaded .NET MQTT open source library in Github, others are MqttDotNet, nMQTT, M2MQTT, etc.

## Chapter 5 Project Programming

In this chapter, I will start with the catalog. Then explain how to create a project and import important class libraries. After that, I will analyze the code and explain the role of each part of the code, the use of the database and how to build MQTT server. Finally, I will provide the guide to run the program in this paper.

### 5.1 Directory Description

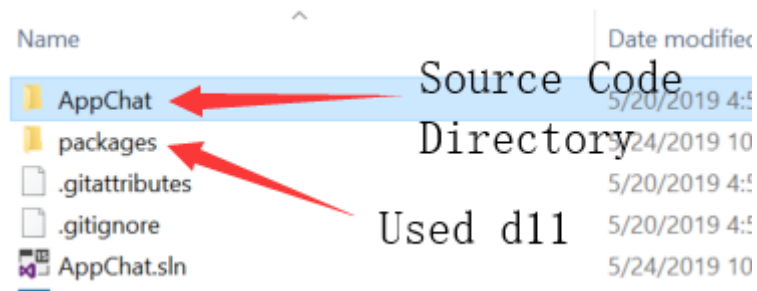


Figure 5-1 Source Code Directory 1

Folder “AppChat” contains all the source code and folder “packages” contains all the used d11.

Name	Date modified	Type
Chat.Android	5/20/2019 7:35 PM	File folder
Chat.IOS	5/20/2019 6:50 PM	File folder
Chat.Model	5/20/2019 10:18 PM	File folder
Chat.Uwp	5/20/2019 10:18 PM	File folder
Content.Core	5/21/2019 12:07 AM	File folder
MqttNetServer	5/20/2019 6:50 PM	File folder

Figure 5-2 Source Code Directory 2

The folder “Chat.Android” defines the Android interface while the folder “Chat.IOS” and

“Chat.Uwp” respectively defines the IOS and Uwp interfaces. The folder “Chat.Model” includes the entity class definition used by the program. Besides, the folder “Content.Core” contains the common interface and core logic. The folder “MqttNetServer” contains the sever source code which also includes http and application programming interface (API) .

## 5.2 Create a Project and Import the Class Library

Here we use Visual Studio 2017 to create a project which is supported by Xamarin.Forms and add projects which include a server and clients. The server project template selects the latest .NET Core console application and client projects select the traditional form application.

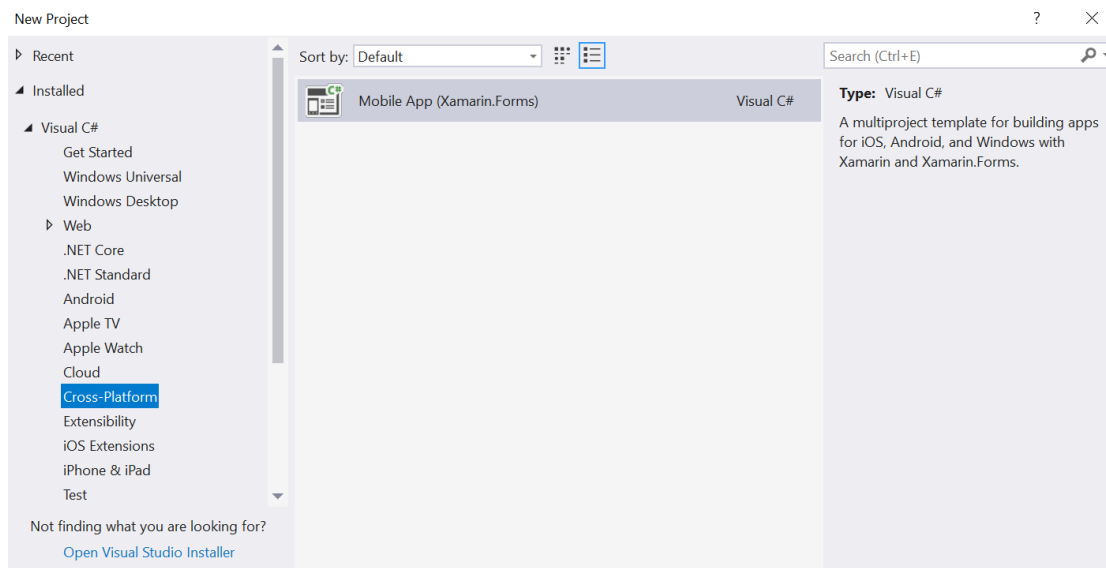


Figure 5-3 Create a New Project

Then in the solution right click and select “Manage NuGet Packages for Solutions” then search for “MQTTnet” under “Browse” tab, install MQTTnet library for both server project and client project, the latest stable version is 2.8.5.

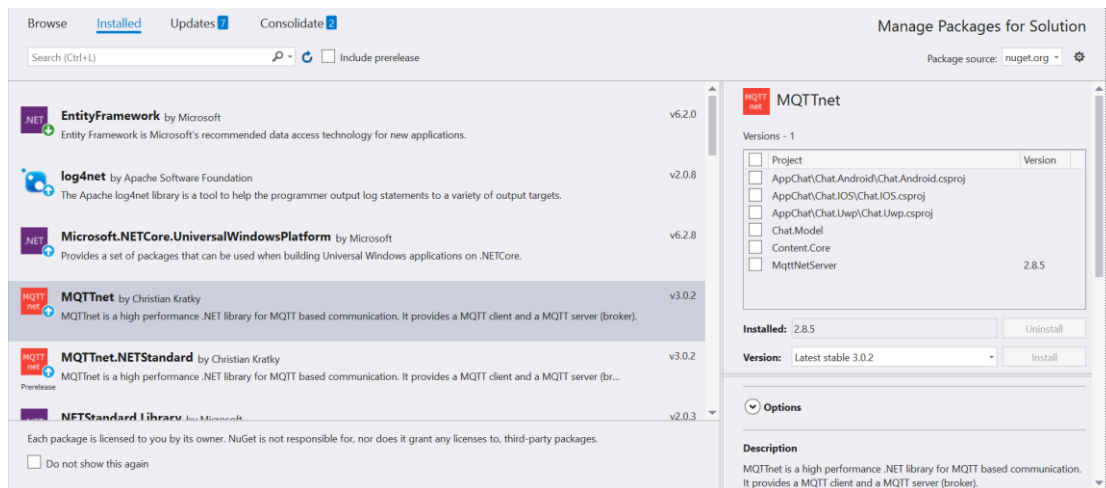


Figure 5-4 Install MQTTnet

## 5.3 Source Code Description

MQTTnet is a high-performance .NET open source library based on MQTT communication that supports both MQTT server and client. The author also updates, currently supporting the new version of .NET core, which is why MQTTnet is chosen. MQTTnet is not the most download .NET MQTT open source library in Github, others are MqttDotNet, nMQTT, M2MQTT, etc.

Client simple demo can refer to the official documentation:  
<https://github.com/chkr1011/MQTTnet/wiki/Client> [16]

Simple communication sample program can refer to github:  
<https://github.com/landbroken/MQTTLearning> [17]

### 5.3.1 Local Storage API

```
5 namespace Content.Core
6 {
7     5 references
8     public interface IUserPreferences
9     {
10         0 references
11         void SetDefaultValue();
12
13         1 reference
14         void SetObj<T>(string key, T obj);
15         0 references
16         T GetObj<T>(string key);
17         2 references
18         void SetString(string key, string value);
19         10 references
20         string GetString(string key);
21         0 references
22         void DeleteString(string key);
23
24         1 reference
25         void SetInt(string key, int value);
26         2 references
27         int GetInt(string key);
28     }
29 }
```

Code 5-1 Local Storage API

IUserPreferences in Content.Core is to establish user configuration information local storage interface. Besides, UserPreferencesAndroid, UserPreferencesIOS and UserPreferencesUwp all inherited it and implemented this interface on Android, IOS and UWP respectively.

### 5.3.2 Pop-up Box API



```

23     private static IToast instance;
24     10 references
25     public static IToast Instance
26     {
27         get
28         {
29             if (instance == null)
30             {
31                 instance = DependencyService.Get<IToast>();
32             }
33             return instance;
34         }
35     }
36

```

Code 5-2 Pop-up Box API

IToast is the pop-up prompt box interface and Toast\_Uwp, Toast\_IOS and Toast\_Android implement the interfaces on UWP, IOS and Android separately.

### 5.3.3 Interface Design

The language to design the interfaces in this paper is XAML (Extensible Application Markup Language) which is a markup language for instantiating .NET objects. It simplifies the process of creating UI for .NET framework applications, making programming interface programming simpler and clearer. XAML represents the instantiation of an object directly in a specific set of fallback types defined in the assembly. Each element in a XAML file represents a class in .NET, and each property in a XAML file represents a property, method, or event in a .NET class [18]. And it is very easy to start a xaml project and start to program.

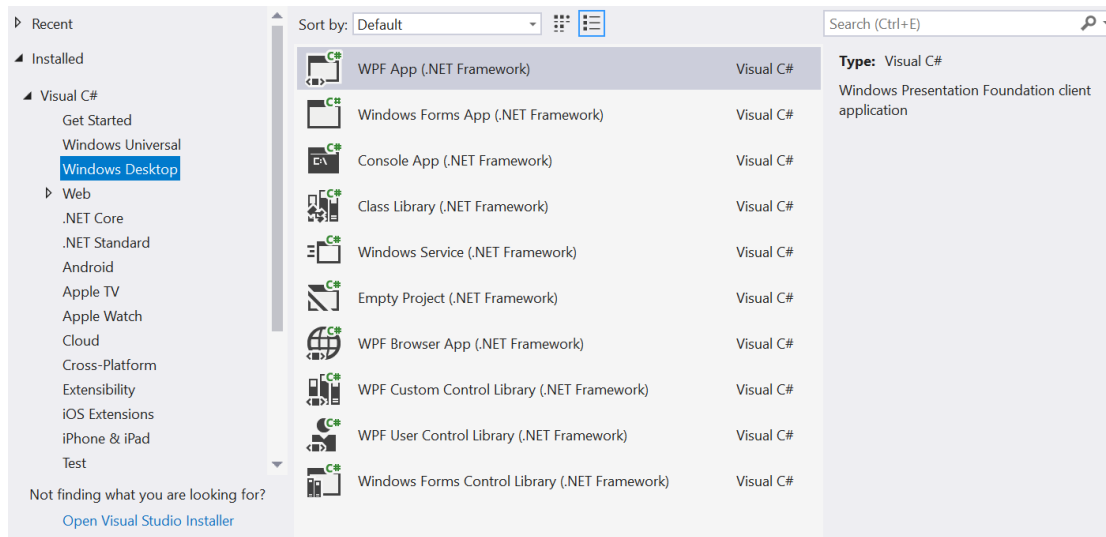


Figure 5-5 Create a WPF File

Most general global temporary configuration of the program is stored in App.Xaml.cs. Login.xaml is the login interface file which provides the inputs for the account number and password and an entry for registration.

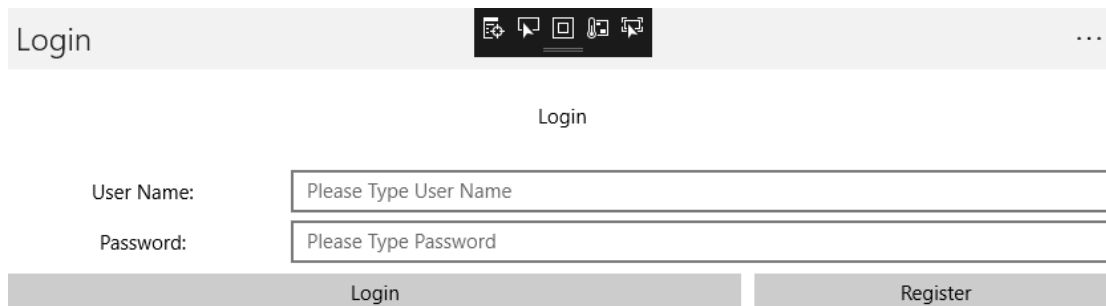
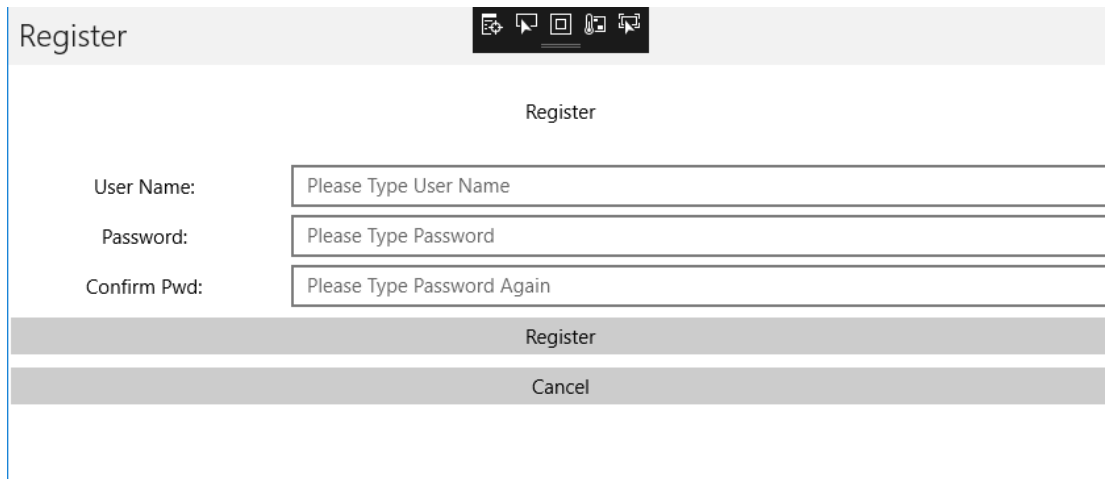


Figure 5-6 Login Interface on UWP

Register.xaml is the registration interface file which help the user to create their account and input and store their account number and password. When they create their account in this page successfully, they can login in and start to chat with others.



The image shows a Windows Universal Platform (UWP) application window titled "Register". The window has a standard Windows title bar with icons for maximize, minimize, and close. The main content area is titled "Register" and contains three input fields for registration:

- User Name:** A text box with the placeholder text "Please Type User Name".
- Password:** A text box with the placeholder text "Please Type Password".
- Confirm Pwd:** A text box with the placeholder text "Please Type Password Again".

Below the input fields are two buttons: "Register" and "Cancel".

Figure 5-7 Register Interface on UWP

MainPage.xaml is the friend list interface file which help the user to check other user in this software and whether they are online or not.

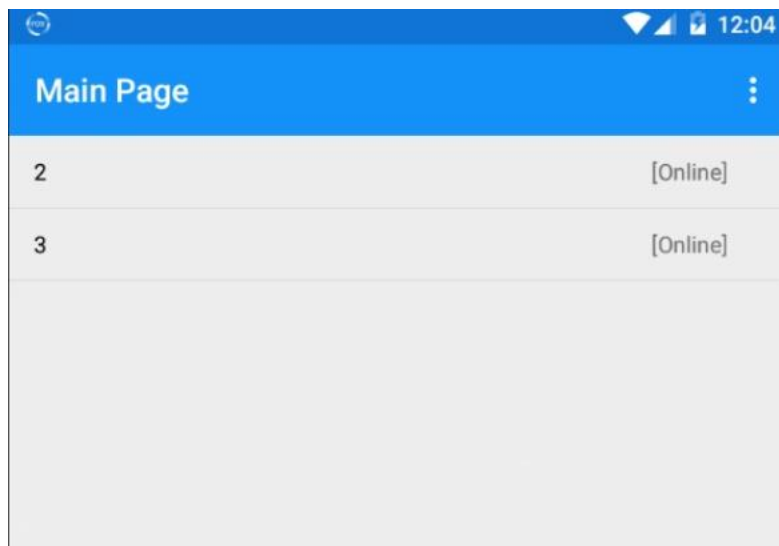


Figure 5-8 Main Page Interface on Android

ChatPage.xaml provides a page file for the chat interface which can allow the user to send and receive message immediately after they choose the user in their friend list.

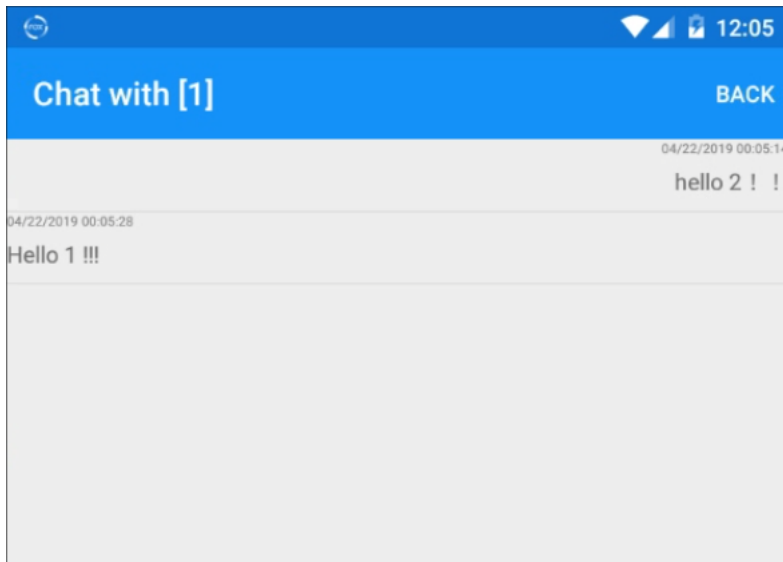


Figure 5-9 Chat Page Interface on Android 1

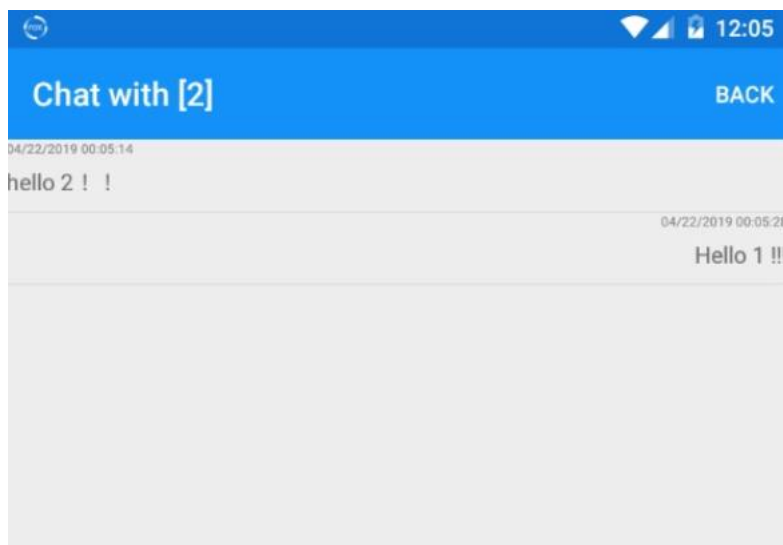


Figure 5-10 Chat Page Interface on Android 2

FrmMqttServer.cs provides the main server interface which let the user to change the IP address and ports. Besides it can also give the button to make the server run or stop and allow the user to check the records in the server

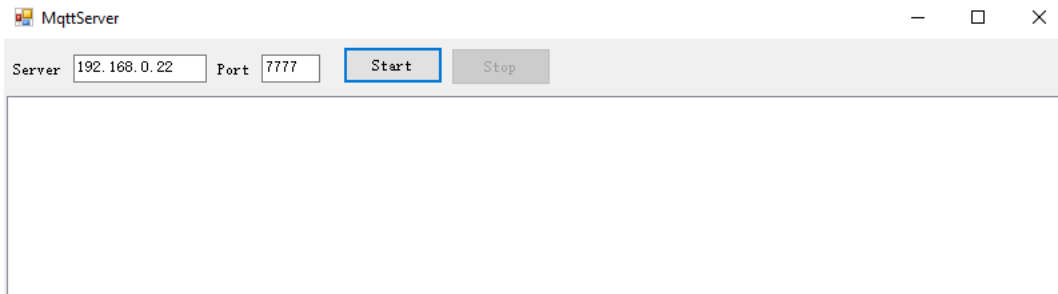


Figure 5-11 Server Interface

### 5.3.4 Client

MQTT is different from HTTP, which is based on request/response. The server cannot send data directly to the client. The MQTT is based on the publish/subscribe mode, and all clients remain connected to the server. Clients subscribe to the server for messages, others send messages or topics to the server and the server matches the subscribed and published topic or messages, then forward the message to Match the passed client.

HttpClientHelper.cs is the http client helper class which provides http request function and encapsulates registration, login, message sending, online, offline and get buddy list function.

```

15  public async Task<int> Register(UserInfo userInfo)
16  {
17      try
18      {
19          var retuanObj = 0;
20          var content = new StringContent(JsonConvert.SerializeObject(userInfo), Encoding.UTF8);
21          var response = await client.PostAsync(
22              App.UserPreferences.GetString(EnumUserPreferences.ServerAddress.ToString()) + "Register", content);
23          var txt = await response.Content.ReadAsStringAsync();
24          var result = JsonConvert.DeserializeObject<ExchangeData>(txt);
25          if (result.IsSuccess)
26          {
27              int.TryParse(result.Data.ToString(), out retuanObj);
28          }
29          return retuanObj;
30      }
31      catch
32      {
33          return 0;
34      }
35  }
36  }

```

1 reference

class System.Text.Encoding  
Represents a character encoding.

Code 5-3 Encapsulate Registration Function

The code above is an example to encapsulate registration.

MqttApplicationMessageReceivedHandler is the Mqtt message receiving processing class which providing chat message, processing function of online and offline. And it also the class to subscribe topics and all topic subscriptions are defined here.

```
12 public class MqttApplicationMessageReceivedHandler : IMqttApplicationMessageReceivedHandler
13 {
14     public event Action<string> ReceiveMsg;
15     public event Action<int> ReceiveOnline;
16     public event Action<int> ReceiveOffline;
17     public async Task HandleApplicationMessageReceivedAsync(MqttApplicationMessageReceivedEventArgs eventArgs)
18     {
19         var topic = eventArgs.ApplicationMessage.Topic;
20         if (topic == MQTTTopic.Msg.ToString() + "-" + App.CurrentUser.Id)
21         {
22             var content = Encoding.UTF8.GetString(eventArgs.ApplicationMessage.Payload);
23             ReceiveMsg?.Invoke(content);
24         }
25         else if (topic == MQTTTopic.Online.ToString())
26         {
27             var content = Encoding.UTF8.GetString(eventArgs.ApplicationMessage.Payload);
28             var userinfo = JsonConvert.DeserializeObject<UserInfo>(content);
29             if (userinfo != null && userinfo.Id > 0)
30             {
31                 ReceiveOnline?.Invoke(userinfo.Id);
32             }
33         }
34         else if (topic == MQTTTopic.Offline.ToString())
35         {
36             var content = Encoding.UTF8.GetString(eventArgs.ApplicationMessage.Payload);
37             var userinfo = JsonConvert.DeserializeObject<UserInfo>(content);
38             if (userinfo != null && userinfo.Id > 0)
39             {
40                 ReceiveOffline?.Invoke(userinfo.Id);
41             }
42         }
43     }
44 }
```

Code 5-4 Deal with Received Message

In ChatPage.xaml, there will be more specific action about dealing with the messages.

```
public ChatPage(UserInfo remoteUser)
{
    InitializeComponent();
    RemoteUser = remoteUser;
    this.Title = $"Chat with [{remoteUser.Name}]";

    MsgList.ItemsSource = Messages;
    MsgList.Refreshing += MsgList_Refreshing;
    var handler = MQTTHelper.Instance.MqttClient.ApplicationMessageReceivedHandler as MqttApplicationMessageReceivedHandler;
    if (handler != null)
    {
        handler.ReceiveMsg += Handler_ReceiveMsg;
    }
    LoadMsg();
}
```

Code 5-5 Subscribe for Received Message

This is the event subscription for the received message.

```
public async void LoadMsg()
{
    var list = await client.GetMessage(RemoteUser.Id);
    if (list != null && list.Count > 0)
    {
        Messages = new ObservableCollection<MsgInfo>(list);
        MsgList.ItemsSource = Messages;
    }
}
```

Code 5-6 Load History Message

LoadMsg is used to load history message.

```
private async void SendMsg_Button_Clicked(object sender, EventArgs e)
{
    var msg = SendMsg.Text;
    if (!string.IsNullOrEmpty(msg))
    {
        MsgInfo msgInfo = new MsgInfo();
        msgInfo.Content = msg;
        msgInfo.ReceiveId = RemoteUser.Id;
        msgInfo.SendId = App.CurrentUser.Id;
        var result = await MQTTHelper.Instance.SendMsg(msgInfo);
        client.Message(msgInfo);
        if (result)
        {
            SendMsg.Text = "";
            Messages.Add(msgInfo); ScollerToButtom();
            //Toast_Android.Instance.ShortAlert("Send Succeeded!");
        }
        else
        {
            ToastHelper.Instance.ShortAlert("Send Failed!");
        }
    }
}
```

Code 5-7 Send Message

This part is the route of the message being sent. The message will be sent to the mqttserver. After that the message will be sent to http server and this server will save the message into the SQLite.

MQTTHelper.cs is the mqtt help class provides initialization of MQTT client (including subscription related topic function), send message, registration, go online, offline and message push function. MqttMsgSourceConverter.cs is providing a conversion tool for providing Xaml pages. It provides color location and online offline mark conversion.

```
try
{
    // var options = new MqttClientOptions() { ClientId = userInfo.Guid + "-" + userInfo.Name };
    var options = new MqttClientOptions() { ClientId = Guid.NewGuid().ToString("D") };
    options.ChannelOptions = new MqttClientTcpOptions()
    {
        Server = ServerIp,
        Port = Port
    };
    options.Credentials = new MqttClientCredentials()
    {
        Username = "admin",
        Password = "public"
    };
    options.CleanSession = true;
    options.KeepAlivePeriod = TimeSpan.FromSeconds(100.5);
    options.KeepAliveSendInterval = TimeSpan.FromSeconds(20000);
}
```

Code 5-8 Initialize MQTT Client 1

```
var msgTopicFilter = new TopicFilterBuilder().WithTopic(MQTTTopic.Msg.ToString() + "-" + App.CurrentUser.Id)
    .WithQualityOfServiceLevel(MqttQualityOfServiceLevel.ExactlyOnce).Build();

var onlineTopicFilter = new TopicFilterBuilder().WithTopic(MQTTTopic.Online.ToString())
    .WithQualityOfServiceLevel(MqttQualityOfServiceLevel.ExactlyOnce).Build();
var offlineTopicFilter = new TopicFilterBuilder().WithTopic(MQTTTopic.Offline.ToString())
    .WithQualityOfServiceLevel(MqttQualityOfServiceLevel.ExactlyOnce).Build();

var currentOptions = new MqttClientSubscribeOptions();
currentOptions.TopicFilters.Add(msgTopicFilter);
currentOptions.TopicFilters.Add(onlineTopicFilter);
currentOptions.TopicFilters.Add(offlineTopicFilter);
var cancel = new CancellationToken();
await _mqttClient.SubscribeAsync(currentOptions);
return true;
```

Code 5-9 Initialize MQTT Client 2

### 5.3.5 MQTT Server

The server includes two functions, one is the MQTT service function, which provides the subscription and release functions of the message; the other function is the web API interface function, which provides functions such as user registration, login and



extraction of historical messages. When the client sends data to the server through the http request, it can implement related functions such as registration and login.

MQTT is a link protocol that specifies how data bytes are organized and transmitted over a TCP/IP network [21]. But in fact, developers don't need to link to the specifics of this link protocol. I only need to know that each message has a command and data payload. This command defines the message type (such as a CONNECT message or a SUBSCRIBE message). All MQTT libraries and tools provide a basic way to handle these messages directly and can automatically populate some of the necessary fields such as messages and client IDs.

In MQTT part, I have chosen the MQTTnet middleware which is very easy to use and there is open source in the website: <https://github.com/chkr1011/MQTTnet> [20].

The "private async void MqttServer()" method in the "FrmMqttServer.cs" file is to initialize the MQTT server.

```
private async void MqttServer()
{
    if (null != _mqttServer)
    {
        return;
    }

    var optionBuilder =
        new MqttServerOptionsBuilder().WithConnectionBacklog(1000).WithDefaultEndpointPort(Convert.ToInt32(TxbPort.Text));

    if (!TxbServer.Text.IsNullOrEmpty())
    {
        optionBuilder.WithDefaultEndpointBoundIPAddress(IPAddress.Parse(TxbServer.Text));
    }

    var options = optionBuilder.Build();

    (options as MqttServerOptions).ConnectionValidator += context =>
    {
        if (context.ClientId.Length < 10)
        {
            context.ReturnCode = MqttConnectReturnCode.ConnectionRefusedIdentifierRejected;
            return;
        }
        if (!context.Username.Equals("admin"))
        {
            context.ReturnCode = MqttConnectReturnCode.ConnectionRefusedBadUsernameOrPassword;
            return;
        }
        if (!context.Password.Equals("public"))
        {
            context.ReturnCode = MqttConnectReturnCode.ConnectionRefusedBadUsernameOrPassword;
        }
    }
}
```

Code 5-10 Initialize MQTT Server

I customized an HttpListener service. Its listening port is “7778” and the user can only pass <http://ip:port/action> method to access.

The server supports checking client connections, disconnection and receiving messages from clients.

```
3 references
public class RequestHelper
{
    private HttpListenerRequest request;
    1 reference
    public RequestHelper(HttpListenerRequest request)
    {
        this.request = request;
    }
    1 reference
    public Stream RequestStream { get; set; }
    0 references
    public void ExtracHeader()
    {
        RequestStream = request.InputStream;
    }

    public delegate void ExecutingDispatch(ExchangeData data);
    1 reference
    public void DispatchResources(ExecutingDispatch action)
    {
        ExchangeData data = new ExchangeData();
        try
        {
            var rawUrl = request.RawUrl;
            if (request.Url.Segments.Length == 3 && request.Url.Segments[0] == "/" && request.Url.Segments[1].ToLower() == "api/")
            {
                //Filter url
                UserInfo userInfo = null;
                MsgInfo msgInfo = null;
            }
        }
    }
}
```

Code 5-11 Process of Web API

The specific processing logic and process of this web API are in the class RequestHelper.

## 5.4 SQLite Programming

There are some reasons let me to choose SQLite to store data in this program, First is that SQLite is free and no copyright restriction where SQLite claims itself as the most widely deployed and used database engine [6]. Second reason is SQLite is fast, even 35% faster than the file system [19]. The last one is light weight which make us use SQLite generally only need to bring a dynamic library of it and we can enjoy its full functionality.

### 5.4.1 Building Library

```
public void InitDb(string dbPath)
{
    this.dbPath = dbPath;
    if (!File.Exists(dbPath))
    {
        File.Delete(dbPath);
        Thread.Sleep(2);
    }
    SQLiteConnection.CreateFile(dbPath);
    CreateUserInfoTable();
    CreateMessageTable();
    CreateRelationshipTable();
}
```

Code 5-12 Rebuild Library

```
public void CreateUserInfoTable()
{
    var sql = @"CREATE TABLE UserInfo
    (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        Name NVARCHAR(50) NOT NULL,
        Password NVARCHAR(50) NOT NULL,
        Online INTEGER NOT NULL DEFAULT 0,
        DateTimeStamp INTEGER NOT NULL
    )";
    SQLiteHelper.ExecuteScalar(sql);
}
```

Code 5-13 Create a New User Information Table

```
public void CreateMessageTable()
{
    var sql = @"CREATE TABLE Message
    (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        SendId INTEGER NOT NULL,
        ReceiveId INTEGER NOT NULL,
        Content NVARCHAR(500) NOT NULL,
        DateTimeStamp INTEGER NOT NULL
    )";
    SQLiteHelper.ExecuteScalar(sql);
}
```

Code 5-14 Create a New Message Table

```
public void CreateRelationshipTable()
{
    var sql = @"CREATE TABLE Relationship
    (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        MasterId INTEGER NOT NULL,
        SlaverId INTEGER NOT NULL,
        DateTimeStamp INTEGER NOT NULL
    )";
    SQLiteHelper.ExecuteScalar(sql);
}
```

Code 5-15 Create a New Relationship Table

“InitDbHelper.cs” is used when need to rebuild the library for the first time or later. “public void InitDb(string dbPath)” is used to create a new SQLite database file (the existing file will be deleted and rebuild again). “public void CreateUserInfoTable()” is used to create a user information table. “public void CreateMessageTable()” is used to

create a message table. “public void CreateRelationshipTable()” is used to establish a friend relationship table.

## 5.4.2 Additions, Deletions, Changes, etc

“SQLiteHelper.cs” contains an access class for operating SQLite data and provides basic methods such as adding, deleting and updating.

```
b references
public static object ExecuteScalar(string commandText, CommandType commandType = CommandType.Text)
{
    object result = 0;
    if (connectionString == null || connectionString.Length == 0)
        throw new ArgumentNullException("connectionString");
    if (commandText == null || commandText.Length == 0)
        throw new ArgumentNullException("commandText");
    SQLiteCommand cmd = new SQLiteCommand();
    using (SQLiteConnection con = new SQLiteConnection(connectionString))
    {
        SQLiteTransaction trans = null;
        PrepareCommand(cmd, con, ref trans, true, commandType, commandText);
        try
        {
            result = cmd.ExecuteScalar();
            trans.Commit();
        }
        catch (Exception ex)
        {
            trans.Rollback();
            throw ex;
        }
    }
    return result;
}
```

Code 5-16 Return the Data in the First Column of the First Row

For example, “ExecuteScalar” in the picture return the data in the first column of the first row after executing the SQL statement. Other like “ExecuteNonQuery” returns the number of rows affected after executing the SQL statement. “ExecuteReader” returns the DbDataReader object returned after executing the SQL statement. “ExecuteDataTable” returns the dataset object returned after executing the SQL statement. “ExecuteDataSet” returns the datatable object returned after executing the SQL statement.

### 5.4.3 File Call

“RequestHelper.cs” provides read and write calls to SQLite files.

```
public void DispatchResources(ExecutingDispatch action)
{
    ExchangeData data = new ExchangeData();
    try
    {
        var rawUrl = request.RawUrl;
        if (request.Url.Segments.Length == 3 && request.Url.Segments[0] == "/" && request.Url.Segments[1].ToLower() == "api/")
        {
            //Filter url
            UserInfo userInfo = null;
            MsgInfo msgInfo = null;
            if (request.HttpMethod == "POST" && request.InputStream.CanRead)
            {
                StreamReader reader = new StreamReader(request.InputStream);
                string bodyContent = reader.ReadToEnd();
                userInfo = JsonConvert.DeserializeObject<UserInfo>(bodyContent);
                msgInfo = JsonConvert.DeserializeObject<MsgInfo>(bodyContent);
            }
            if (userInfo == null && msgInfo == null && request.HttpMethod == "POST")
            {
                data.Data = "No Data!";
            }
            else
            {
                data.IsSuccess = true;
                switch (request.Url.Segments[2].ToLower())
                {
                    case "register":
                        var obj = SQLiteHelper.ExecuteScalar(string.Format("Select Id From UserInfo Where Name='{0}'", userInfo.Name));
                        if (obj != null)
                        {

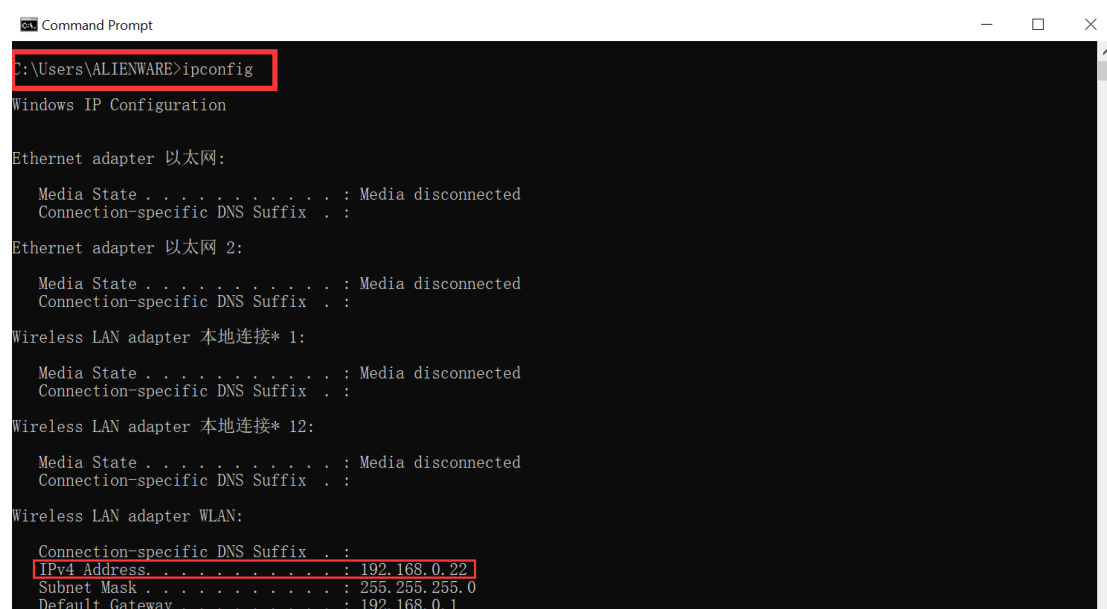
```

Code 5-17 Read and Write the SQLite File

The method “public void DispatchResources(ExecutingDispatch action)” internally provides a write operation to the SQLite file, including storage of information such as operating friend table, registration, online/offline, message and login.

## 5.5 Software Operation Guide

Step 1: Insert “ipconfig” in the Command Prompt to check the ip address



```
Command Prompt
C:\Users\ALIENWARE>ipconfig

Windows IP Configuration

Ethernet adapter 以太网:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Ethernet adapter 以太网 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter 本地连接* 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter 本地连接* 12:

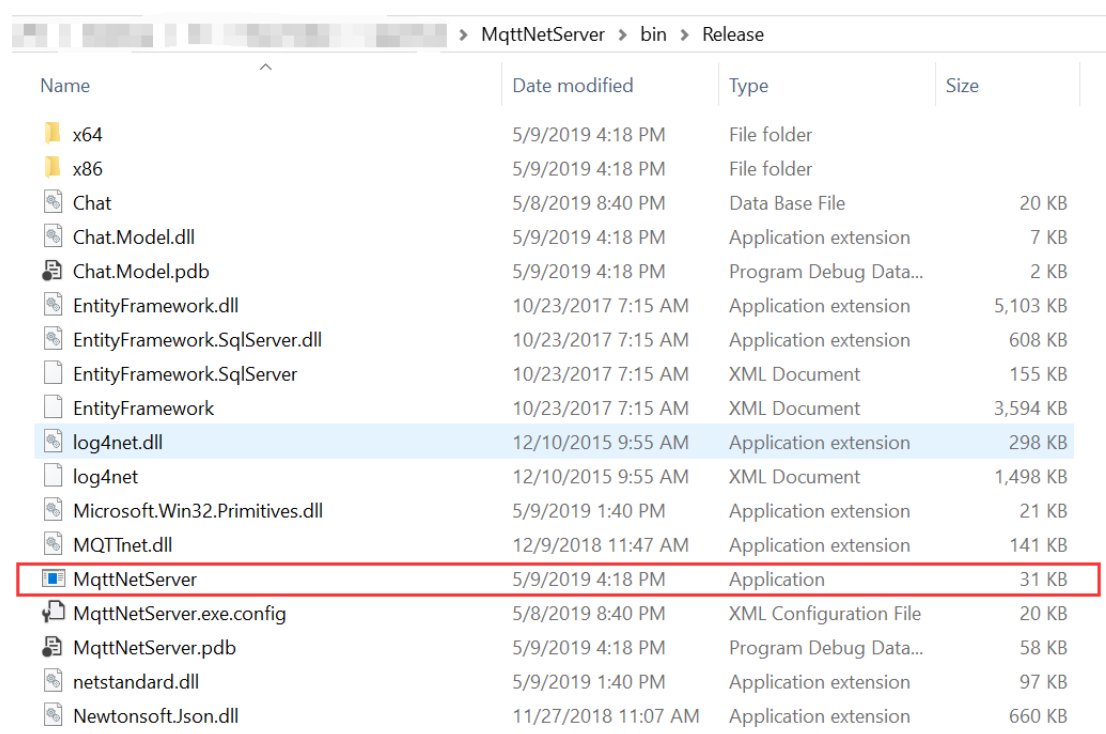
    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . :

Wireless LAN adapter WLAN:

    Connection-specific DNS Suffix  . :
    IPv4 Address. . . . . : 192.168.0.22
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.0.1
```

Figure 5-12 Check the IP Address

Step 2: Open the application “MqttNetServer” in “MqttNetServer\bin\Release”



MqttNetServer > bin > Release				
Name	Date modified	Type	Size	
x64	5/9/2019 4:18 PM	File folder		
x86	5/9/2019 4:18 PM	File folder		
Chat	5/8/2019 8:40 PM	Data Base File	20 KB	
Chat.Model.dll	5/9/2019 4:18 PM	Application extension	7 KB	
Chat.Model.pdb	5/9/2019 4:18 PM	Program Debug Data...	2 KB	
EntityFramework.dll	10/23/2017 7:15 AM	Application extension	5,103 KB	
EntityFramework.SqlServer.dll	10/23/2017 7:15 AM	Application extension	608 KB	
EntityFramework.SqlServer	10/23/2017 7:15 AM	XML Document	155 KB	
EntityFramework	10/23/2017 7:15 AM	XML Document	3,594 KB	
log4net.dll	12/10/2015 9:55 AM	Application extension	298 KB	
log4net	12/10/2015 9:55 AM	XML Document	1,498 KB	
Microsoft.Win32.Primitives.dll	5/9/2019 1:40 PM	Application extension	21 KB	
MQTTnet.dll	12/9/2018 11:47 AM	Application extension	141 KB	
MqttNetServer	5/9/2019 4:18 PM	Application	31 KB	
MqttNetServer.exe.config	5/8/2019 8:40 PM	XML Configuration File	20 KB	
MqttNetServer.pdb	5/9/2019 4:18 PM	Program Debug Data...	58 KB	
netstandard.dll	5/9/2019 1:40 PM	Application extension	97 KB	
Newtonsoft.Json.dll	11/27/2018 11:07 AM	Application extension	660 KB	

Figure 5-13 Open the MqttNetServer Application

Step 3: Insert ip address in the input box for server address in “MqttNetServer” and set a port for it. After that press the “start button” and now the server is running.

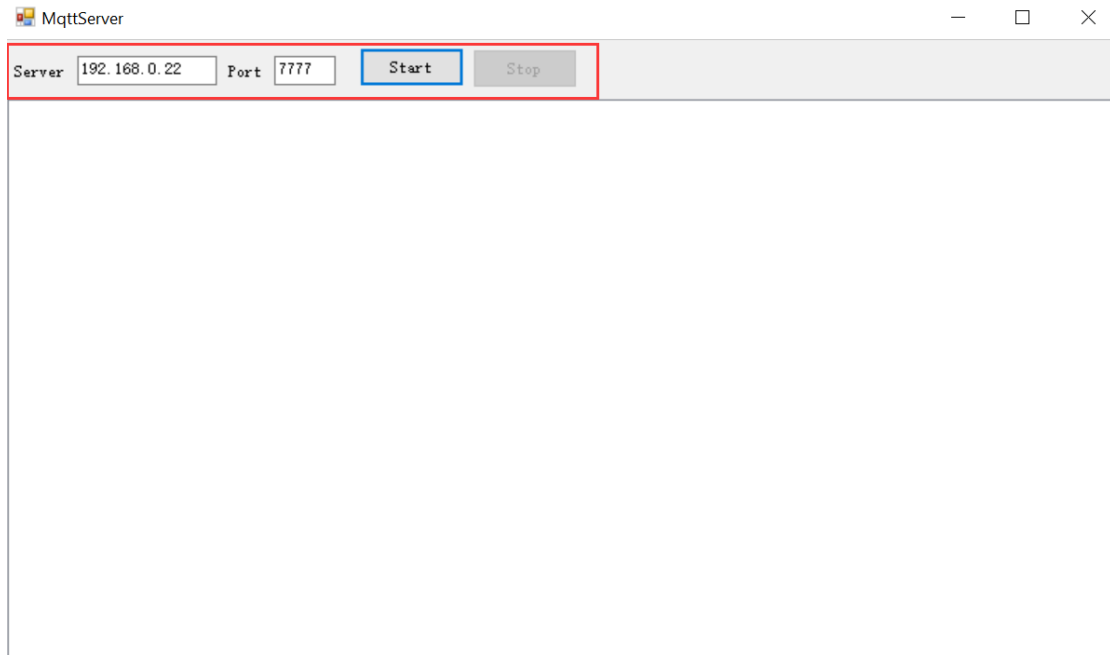


Figure 5-14 Insert IP Address and Port in Server

Step 4: Run the application on IOS/Android/UWP/Raspberry. Press “setting” button and insert the ip address into “ServerIp” and set the port and api. After that press the “Confirm”.

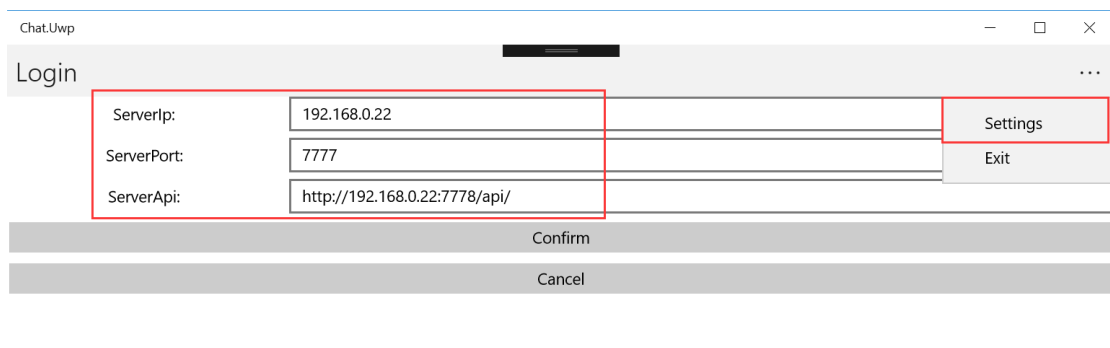


Figure 5-15 Insert IP Address and Port in Client

When we finish the abovementioned steps, we can register an account and login it to use all function in the application in devices.

There are two tips to make sure the software can be operated successfully

First is that we would better run the application “MqttNetServer” as administrator. The second is we would better close firewall in the terminal before we run the application in some situations.

## 🔒 Firewall & network protection

View network connections, specify Windows Defender Firewall settings, and troubleshoot network and Internet problems.

❌ Windows Defender Firewall is using settings that may make your device unsafe.

Restore settings

🏢❌ **Domain network**

Firewall is off.

Turn on

🏠❌ **Private network (active)**

Firewall is off.

Turn on

🌐❌ **Public network**

Firewall is off.

Turn on

Figure 5-16 Close the Firewall



## Chapter 6 Summary

### 6.1 Conclusion and Evaluation

The goal of this thesis is to check the possibility and limitations of communication methods of Xamarin.Forms. Current research is based on a developed communication method via email exchange and database connection. This method can realize communication, but there still is a long delay in message transmission. Therefore, I start trying a new communication method MQTT method which seem to may greatly speed up the message transmission to develop a instant messaging application cross the platforms on Visual Studio with the help of Xamarin.

The result of the research has shown that MQTT protocol can help achieving fast instant message function. The publish/subscribe mode of MQTT protocol allows all clients to stay connected to the server and implement one-to-one instant messaging. MQTT protocol may show a slight delay in the message when the chat information is too dense in the existing program structure. Taking QoS=2 as a example, A message sent by the server to the client needs to be exchanged 4 times: the first time the server sends “publish” to the client; The second time is the client sends “pubrec” to the server; the third time is the server sends “pubrel” to the client; the fourth time is the client sends “pubcomp” to the server. Besides, if there are 10 message which need to be sent, then a total of 40 interactions are required. This may be the main reason for the possible delay.

Xamarin.Forms has many obvious advantages, but it also has its drawbacks. Xamarin is still in the evolutionary stage, bugs are normal and instability is understandable. We must admit that things are not perfect. In this software development, I often encounter Visual Studio hanging or self-flashback, failure of connecting emulator and the code

does not execute as expected. If trying a lot of methods, but the problem has not been solved, restarting the environment and restarting the computer seems to be an unexpected but very effective way.

## **6.2 Outlook**

As for the thesis, the topic of my thesis is about to investigate and test of Xamarin.Forms client to client communication methods. However, The current implementation of instant messaging is only available on the LAN. An interface that allow this instant messaging function to run on the public network is left in the program, but more debugging and operations are required to run successfully. This program is currently successfully tested and run on three platforms, UWP, Android and IOS, with theory of running on the Raspberry Pi possible but not yet tested on a real machine for time reasons.

About the possible delays in the MQTT protocol, In future research, researchers can try to introduce the message transmission protocol of the opportunity version number in the MQTT protocol which means the message is maintained by the version number maintenance order. After the server message arrives, the server is only responsible for the push notification. After the client receives the synchronization request, the server sends msg continuously according to the version. The client tells the server the last version number received. However, this is only a hypothesis and specific solution has to be in real practice.

About the choice of instant messaging protocol, MQTT protocol is selected in this project, but there are still other protocols that may be suitable for instant messaging. At present, OPC UA (Unified Architecture) has great potential in instant messaging application. Instant messaging services also involve many technical points, besides

client-to-client communication, there are multicast, real-time voice and video, etc., which can be added in future development. Different services need to be implemented by different protocols. Like traditional communication protocols, of course, they can also be compared and used. Subsequent experimenters can test and apply these protocols in practical applications.

# Acknowledgment

First and foremost, I would like to express my sincere gratitude and respect to my supervisor Professor Bayerlain who provides me with all necessary facilities and technical guidance. Thanks to his patient teach and continuous suggestion for my project, I can finish it successfully.

Secondly, I want to appreciate sincerely to Professor Heeren who is my second supervisor and gives evaluation and instruction about my project.

Thirdly, I also would like to thank Ms. Hanasova for her detailed guidance of how to solve the problems in thesis writing.

In the end, I want to thank two other students under the guidance of the same supervisor Lu Qing and Wang Yuxuan for their help on equipment and research.

## Appendix A – List of figures

Figure 2-1 Xamarin.Forms [7].....	4
Figure 2-2 Raspberry Pi 3 Model B [8] .....	5
Figure 3-1 Version of Visual Studio .....	7
Figure 3-2 Installed Component in Visual Studio 2017 1 .....	7
Figure 3-3 Installed Component in Visual Studio 2017 2 .....	8
Figure 3-4 Installed Component in Visual Studio 2017 3 .....	8
Figure 3-5 UWP Targeting SDK.....	9
Figure 3-6 Visual Studio Android Emulator .....	10
Figure 3-7 Android SDK Manager Platform .....	11
Figure 3-8 Pair to Mac.....	12
Figure 3-9 Connect Real Device .....	12
Figure 4-1 MQTT Protocol Stack [13].....	13
Figure 4-2 MQTT Publish and Subscribe Model for IoT Sensors [12] .....	14
Figure 4-3 QoS 0 [16].....	15
Figure 4-4 QoS 1 [16].....	16
Figure 4-5 QoS 2 [16].....	16
Figure 5-1 Source Code Directory 1 .....	18
Figure 5-2 Source Code Directory 2 .....	18
Figure 5-3 Create a New Project.....	19
Figure 5-4 Install MQTTnet .....	20
Figure 5-5 Create a WPF File.....	23
Figure 5-6 Login Interface on UWP .....	23
Figure 5-7 Register Interface on UWP.....	24
Figure 5-8 Main Page Interface on Android .....	24
Figure 5-9 Chat Page Interface on Android 1 .....	25
Figure 5-10 Chat Page Interface on Android 2 .....	25

Figure 5-11 Server Interface .....	26
Figure 5-12 Check the IP Address .....	35
Figure 5-13 Open the MqttNetServer Application .....	35
Figure 5-14 Insert IP Address and Port in Server .....	36
Figure 5-15 Insert IP Address and Port in Client.....	36
Figure 5-16 Close the Firewall.....	37

## Appendix B – List of code

Code 5-1 Local Storage API .....	21
Code 5-2 Pop-up Box API .....	22
Code 5-3 Encapsulate Registration Function .....	26
Code 5-4 Deal with Received Message.....	27
Code 5-5 Subscribe for Received Message .....	27
Code 5-6 Load History Message .....	28
Code 5-7 Send Message.....	28
Code 5-8 Initialize MQTT Client 1.....	29
Code 5-9 Initialize MQTT Client 2.....	29
Code 5-10 Initialize MQTT Server .....	30
Code 5-11 Process of Web API .....	31
Code 5-12 Rebuild Library .....	32
Code 5-13 Create a New User Information Table .....	32
Code 5-14 Create a New Message Table .....	32
Code 5-15 Create a New Realationship Table.....	32
Code 5-16 Return the Data in the First Column of the First Row .....	33
Code 5-17 Read and Write the SQLite File .....	34

## **Appendix C – Content of USB Striker**

1. Thesis paper (PDF version):" YuanZun\_Thesis.pdf"
2. Thesis paper (WORD version):" YuanZun\_Thesis.docx"
3. Xamarin app:" AppChat.7z"
4. OperationGuide.pdf



# Bibliography

- [1] Cridland D. XEP-0286: XMPP on Mobile Devices[J]. XMPP Standards Foundation, 2010.
- [2] Niemi A. Session Initiation Protocol(SIP) Extension for Event State Publication[J]. Networking&Communication Engineering, 2004(2).
- [3] Wikipedia, Visual Studio introduction. Retrieved from URL: [https://en.wikipedia.org/wiki/Microsoft\\_Visual\\_Studio](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio) [Accessed: 30.05.2019]
- [4] Kevin Ashley. Cross-Platform Productivity with Xamarin[J]. MSDN magazine, 2016, 31(9): 36-42.
- [5] Microsoft, Xamarin.Forms. Retrieved from URL: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/> [Accessed: 30.05.2019]
- [6] SQLite, About SQLite. Retrieved from URL: <https://www.sqlite.org/about.html> [Accessed: 30.05.2019]
- [7] Qi Ming, Xamarin is free, what can you do? Retrieved from URL: <https://www.cnblogs.com/hobe/p/5385539.html> [Accessed: 30.05.2019]
- [8] Raspberry Pi foundation, Raspberry Pi 3 Model B+. Retrieved from URL: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/> Accessed: 30.05.2019]
- [9] Microsoft Visual Studio, Download. Retrieved from URL: <https://visualstudio.microsoft.com/zh-hans/downloads/> [Accessed: 30.05.2019]
- [10] Microsoft Visual Studio, Visual Studio Emulator. Retrieved from URL: <https://visualstudio.microsoft.com/vs/msft-android-emulator/> [Accessed: 30.05.2019]
- [11] MQTT, MQTT Frequently Asked Questions. Retrieved from URL: <http://mqtt.org/faq> [Accessed: 30.05.2019]
- [12] Michael Yuan, First met MQTT. Retrieved from URL: <https://www.ibm.com/developerworks/cn/iot/iot-mqtt-why-good-for-iot/index.html> [Accessed: 30.05.2019]

- [13] S.Ramana, N Bhaskar, M.V.Ramana Murthy, S China Ramu. Mobile Commerce using ECC and MQTT Protocol. IAPE' 19, Oxford, United Kingdom.
- [14] YunliaoIM, Briefly Introduce the Release/Subscription Mode of MQTT. Retrieved from URL: <http://www.yunliaoim.com/im/919.html> [Accessed: 30.05.2019]
- [15] International Business Machines Corporation (IBM), Eurotech, MQTT V3.1 Protocol Specification. Retrieved from URL: <http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html#qos-flows> [Accessed: 30.05.2019]
- [16] Chirstian, Simple Client Demo of MQTTnet. Retrieved from URL: <https://github.com/chkr1011/MQTTnet/wiki/Client> [Accessed: 30.05.2019]
- [17] landbroken, Learn how to use mqtt with open source lib in C#. Retrieved from URL: <https://github.com/landbroken/MQTTLearning> [Accessed: 30.05.2019]
- [18] Microsoft, XAML overview (WPF). Retrieved from URL: <https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf> [Accessed: 30.05.2019]
- [19] SQLite, Database Speed Comparison. Retrieved from URL: <https://www.sqlite.org/speed.html> [Accessed: 30.05.2019]
- [20] Chirstian, MQTTnet Retrieved from URL: <https://github.com/chkr1011/MQTTnet> [Accessed: 30.05.2019]
- [21] Michael Yuan, Getting to know MQTT. Retrieved from URL: <https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/> [Accessed: 30.05.2019]
- [22] Microsoft, Visual Studio Emulator for Android. Retrieved from URL: <https://visualstudio.microsoft.com/vs/msft-android-emulator/> [Accessed: 30.05.2019]