WillDo

GTD System with Clients in Multi-platforms

Prepared by LHearen

LHearen@126.com

April 22, 2015

CONTENTS

1. Introduction

1.1 Purpose

With the development of the mobile phones, more and more attention and manpower poured into mobile applications development. Nowadays lots of applications have been produced but there are still some ideas that can be further fulfilled and polished, considering this situation I chose a topic about ToDoList which is called formally GTD system abbreviated from Get Things Done; though there are many developers at home and abroad, I still want to have a try to challenge myself in this field with bran-new developing skills (for me only), mobile development in Android, Windows client using WPF as presenter and server using C++ network programming which are completely strange for me who only tried C, java, Win Form, Windows Phone and have some basic project experience. In this report I will try my best to display my current work concretely including server side, Windows client and Android client.

1.2 Project Scope

There are several parts about this system and in each part we can have so many features to accomplish and polish but due to the time limitation and personal reason I will only achieve four main parts in this system including a server accomplished by C++ which I plan to use Node.js to replace in the near future, a Windows client by C# using WPF to accomplish the presenter layer one of whose outstanding features is binding which will be displayed more vividly in the section three, an Android client by java and a Web client which will be finished in the near future too.

As for the server, I planned to use non-block multi-thread mechanism as the basic model to achieve multi-users and multi-clients network communication and the realization details will be revealed in the realization part in section three.

When it comes to the Windows client programmed in C# whose presenter layer is accomplished by WPF using binding mechanism; in this client the user can freely use add, delete, update, sort, filter, modify and the other features. For example, the user can simply click an add button and the to-do list will contain a wholly new to-do task with default values according to the situation and when the user tries to edit it, he or she can simply select it and then edit at the bottom of the main panel, changing its priority using different color to indicate different levels, changing its due time using date picker, changing its progress any time he or she wants, customizing the color of the whole item, changing the category and location of the task and there are still more which will be detailed in section three.

Given the Android client, due to its simplicity, only some of the features are accomplished compared to the Windows client except for one specialty, the location which is to be retrieved by GPS or network provider instead of manually input in

Windows client and also there are still some features that the Windows client cannot provide, let’s see the specifics in the section three.

In this report, section one and section two will be used to explain some basic ideas and terms used in the system; section three is arranged to discuss the details of the features of the whole system currently including server side, Windows client and Android client and the corresponding realization details; after section three we will further discuss the performance, feasibility, robustness, portability and future improvements about this system; before we end this report in section five there will be several Gantt graphs used to present the whole developing process and arrangements and of course references in section six.

2. Terms

Term	Specification
Blocking & Non-blocking socket[1]	One of the first issues that you'll encounter when developing your Windows Sockets applications is the difference between blocking and non-blocking sockets. Whenever you perform some operation on a socket, it may not be able to complete immediately and return control back to your program. For example, a read on a socket cannot complete until some data has been sent by the remote host. If there is no data waiting to be read, one of two things can happen: the function can wait until some data has been written on the socket, or it can return immediately with an error that indicates that there is no data to be read. The first case is called a blocking socket. In other words, the program is "blocked" until the request for data has been satisfied. When the remote system does write some data on the socket, the read operation will complete and execution of the program will resume. The second case called a non-blocking socket requires that the application recognize the error condition and handle the situation appropriately. Programs that use non-blocking sockets typically use one of two methods when sending and receiving data. The first method, called polling, is when the program periodically attempts to read or write data from the socket (typically using a timer). The second, and preferred method, is to use what is called asynchronous notification. This means that the program is notified whenever a socket event takes place, and in turn can respond to that event. For example, if the remote program writes some data to the socket, a "read event" is generated so that program knows it can read the data from the socket at that point.
WPF[2]	Windows Presentation Foundation (WPF) is a graphical subsystem for rendering user interfaces in Windows-based applications by Microsoft. WPF attempts to provide a consistent programming model for building applications and separates the user interface from business logic. It resembles similar XML-oriented object models, such as those implemented in XUL and SVG. WPF employs XAML, an XML-based language, to define and link various interface elements.[1] WPF applications can also be deployed as standalone desktop programs, or hosted as an embedded object in a website. WPF aims to unify a number of common user interface elements, such as 2D/3D rendering, fixed and adaptive documents, typography, vector graphics, runtime animation, and pre-rendered media. These elements can then be linked and manipulated based on various events, user interactions, and data bindings.
ZeroMQ	ØMQ[3] (also known as ZeroMQ, ØMQ, or zmq) looks like an embeddable networking library but acts like a concurrency framework. It gives you sockets that carry atomic messages across various transports like in-process, inter-process, TCP, and multicast. You can connect sockets N-to-N with patterns like fan-

ZeroMQ	out, pub-sub, task distribution, and request-reply. It’s fast enough to be the fabric for clustered products. Its asynchronous I/O model gives you scalable multicore applications, built as asynchronous message-processing tasks. It has a score of language APIs and runs on most operating systems. ØMQ is from iMatix1 and is LGPLv3 open source.
Node.js[4]	Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

3. System Features

In this section we will focus on the features of the whole system including three parts server side currently realized by C++ using non-blocking network programming, Windows client realized by C# using WPF to implement the presenter layer and Android client realized by java, which will be presented in three different parts; this section will exclude these affairs, the performance, scalability, feasibility, robustness and future improvements that will be partly explained in section four.

3.1 Sever

3.1.1 Sequence diagrams

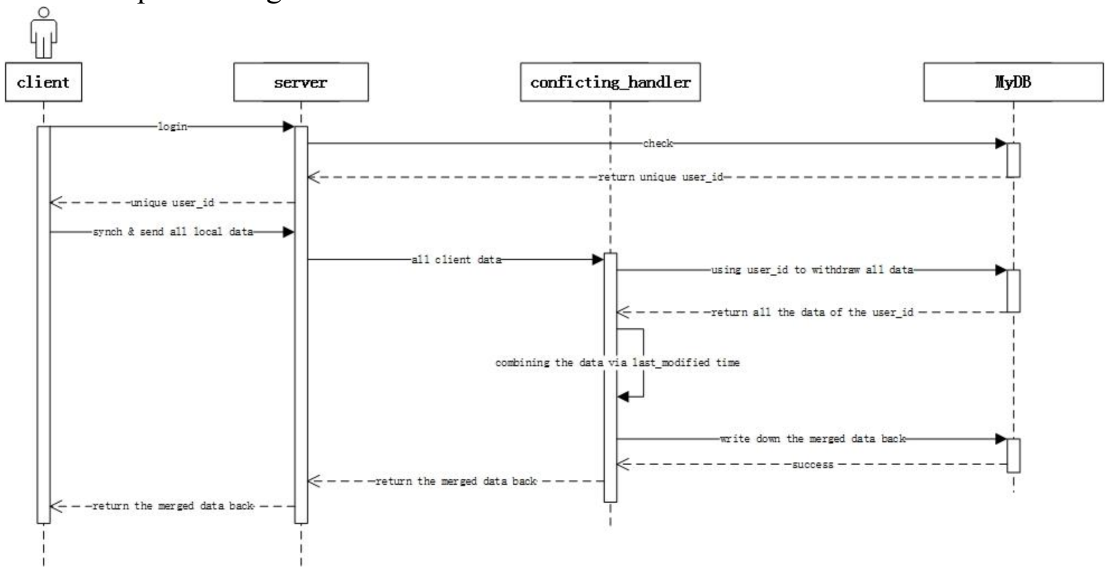

client
server
conficting_handler
MyDB
login
check
return unique user_id
unique user_id
synch & send all local data
all client data
using user_id to withdraw all data
return all the data of the user_id
combining the data via last_modified time
write down the merged data back
success
return the merged data back
return the merged data back

Fig. 1 Major Process Communicating with Server

In the above diagram shown in Fig. 1, we can clearly see what exactly happens when the client connecting to the server synchronizing data. Before the user can do anything further, he or she should firstly login in using personal registered account after which he or she can synchronize with the server storing data in server which will ensure the security of the tasks records and enable him or her to access data either in Windows client or Android client. After login and granted authority, the client will pass all its local data to the server which will invoke a conflicting method to handle the conflicted components between client and server according to the last modified attribute to get the final results and then store it back into the database and return it back to the client too.

3.1.2 Details in Sequence diagrams

As is shown in Fig.2, a very typical example of the whole process of the client interacting with the server to complete synchronization; once the user logins, the server will start a new thread to handle all the requests from the client inside which there will be two other threads generated to handle receiving events and responding affairs; in receiving process the thread will complete the conflicting data handling and make sure the data between client and server is updated and consistent after which the receiving thread will return and inform the parent thread to create another thread sending data back to the client before which of course the receiving thread will store the updated data back into the database for future use. As for the details, we will take a deep look at it in the realization part.

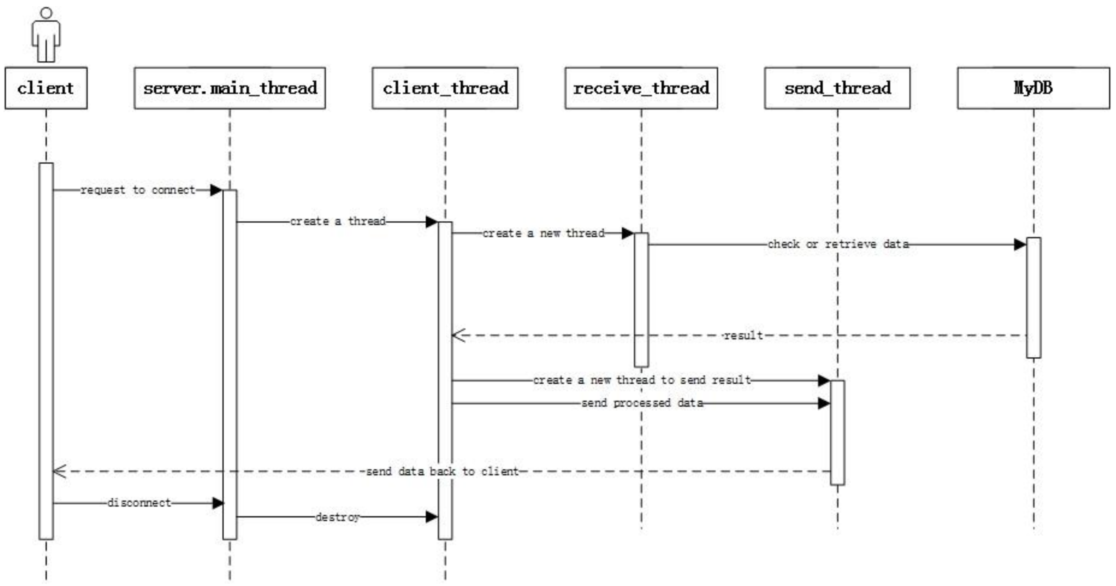


Fig. 2 Details of the Server Side

3.1.3 Realization

In this part, we will focus on the realization details of the server side which of course will include some corresponding database operations, the structure shown in Fig. 3 and to make it more clearly, I will try to only mention mere the most significant functions and implementations here. Due to the simplicity, I ignore some auxiliary features in server including user registration and login part; for the time being only the certain user can be used but that can be easily expanded in the near future and just some trivial stuff and now let’s pay attention to the critical parts of the server.

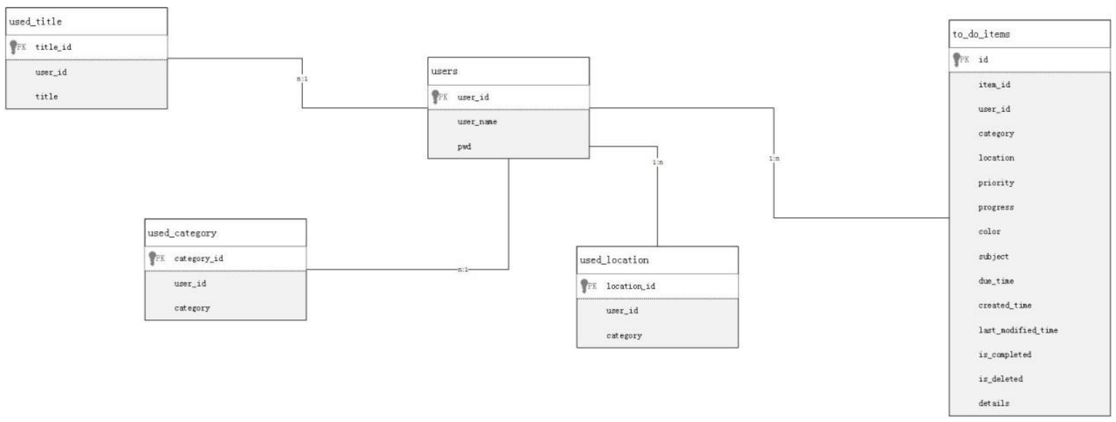


Fig. 3 E-R Diagram of Database

As for the database, all the necessary operations are encapsulated into several major methods which can be accessible outside the MyDB class including:

User retrieve_info(string name, string pwd);

```
int add_user(string user_name, string pwd);

void update_user(User user);
```

all the related codes can be checked in source file along with this specification. via retrieve_info using the user_name and password which will invoke verify_user to check whether the user and its password are correct to withdraw the data stored in database and synchronize, the server can then get all the info of the user including its most frequently used searching titles, locations and categories and all the to-do tasks. Adding a user via add_user is okay with the database though right now the clients have not yet realize registration and the related functions. After synchronization, the server will write down the consistent data back to the database, via update_user, server can easily realize that feature.

As for the server itself, after constructing a server socket and set its non-blocking status, it will start to listen at the certain port and once the client connect to it, it will create a new thread to handle all the related stuff including receiving and sending data.

The following codes is about waiting the remote clients requesting connection with the server and once the request occurs, a new thread will be built and handle all the rest – one prerequisite, the request should be valid otherwise the clientSocket will not be returned owing to the ReceiveConnect method which will make sure the return value is valid otherwise it will listen at the predefined port eternally.

```
while (1)

        {

                SOCKET clientSocket = ReceiveConnect(serverSocket);

                DWORD dwThreadId;

                DWORD WINAPI AnswerThread(LPVOID lparam);

                CreateThread(NULL, NULL, AnswerThread, (LPVOID)clientSocket,
0, &dwThreadId);

        }
```

Here is a major function used to handle the data from client. To make sure all the data have been received, there is a length before the whole data; once the server receive the data, it will firstly find the length and then continue to receive all the rest data until the length of the received data reaches the total length it gets before, after which the server will invoke the parseForWindowClient to get the user from a json string and after that the server will then call the retrieve_info to get the info of the user in database and according to the info from database, the server will further call getConsistentUser to handle the inconsistency between them via last modified time, after which the updated data will be written back to the database and the refreshed user will also be returned to the client via json string.

```
///this simple function is used to start the server socket and accept the
```

```
///remote the connection and then start a new thread to handle the requests from
clients;

int init()

{


        WSADATA wsd;

        SOCKET sServer;

        SOCKET sClient;

        int retVal;

        char buf[BUF_SIZE];


        //initialize the environment;

        if (WSAStartup(MAKEWORD(2, 2), &wsd) != 0)

        {

                printf("WSAStartup failed!\n");

                return 1;

        }

        //initialize the server socket;

        sServer = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

        if (INVALID_SOCKET == sServer)

        {

                printf("socket failed!\n");

                WSACleanup();

                return -1;

        }

        //initialize the basic attributes for the server socket;

        SOCKADDR_IN addrServ;

        addrServ.sin_family = AF_INET;

        addrServ.sin_port = htons(8888);
```

```
	addrServ.sin_addr.S_un.S_addr = htonl(INADDR_ANY);


	//binding to the local ip and certain port;
	retVal = bind(sServer, (const struct sockaddr*)&addrServ,
sizeof(SOCKADDR_IN));
	if (SOCKET_ERROR == retVal)
	{
		printf("bind failed!\n");
		closesocket(sServer);
		WSACleanup();
		return -1;
	}


	//start listenning;
	retVal = listen(sServer, 1);
	if (SOCKET_ERROR == retVal)
	{
		printf("listen failed!\n");
		closesocket(sServer);
		WSACleanup();
		return -1;
	}


	//set non-blocking mode;
	int iMode = 1;
	retVal = ioctlsocket(sServer, FIONBIO, (u_long FAR*) &iMode);
	if (retVal == SOCKET_ERROR)
	{
		printf("ioctlsocket failed!\n");
		WSACleanup();
		return -1;
	}

	//start accepting;
	printf("TCPServer start...\n");
	sockaddr_in addrClient;
	int addrClientlen = sizeof(addrClient);
	while (true)
	{
		SOCKET sClient = accept(sServer, (sockaddr FAR*)&addrClient,
&addrClientlen);
		if (INVALID_SOCKET == sClient)
		{
			int err = WSAGetLastError();
			if (err == WSAEWOULDBLOCK)
			{
				Sleep(5);
				continue;
			}
			else
			{
				printf("accept failed!\n");
				closesocket(sServer);
				WSACleanup();
				return -1;
			}
		}
		cout << "Accepted Client IP: "<<inet_ntoa(addrClient.sin_addr) << "
PORT: " << addrClient.sin_port << endl;
```

```
                //once connected, create a thread to handle the rest of it;
                DWORD dwThreadId;
                //DWORD WINAPI AnswerThread(LPVOID lparam);

                CreateThread(NULL, NULL, AnswerThread, (LPVOID)sClient, 0,
&dwThreadId);
        }
        system("pause");
}
```

We have to know the details of the User before we thoroughly understand all the logics mentioned later.

```
class User
{
private:
        int user_id;
        string user_name;
        string pwd;

        //frequently used info;
        vector<string> used_titles;
        vector<string> used_locations;
        vector<string> used_categories;

        vector<ToDoItem> items_vector;//store all the to-do items;

public:
        //default constructor;
        User():user_id(0){};

        //dispose;
```

```
~User()
{
        this->used_categories.clear();
        this->used_locations.clear();
        this->items_vector.clear();
        this->used_titles.clear();
}
int get_user_id()
{
        return user_id;
}

string get_user_name()
{
        return user_name;
}

string get_password()
{
        return pwd;
}

vector<ToDoItem> getItems()
{
        return items_vector;
}

vector<string> getTitles()
{
        return used_titles;
}

vector<string> getCategories()
{
        return used_categories;
}

vector<string> getLocations()
{
        return used_locations;
}

void set_id(int id)
{
        this->user_id = id;
}

void set_name(string name)
{
        this->user_name = name;
}

void set_password(string pwd)
{
        this->pwd = pwd;
}
//set attributes for the user;
void set_titles(vector<string> titles)
```

```
	{
		this->used_titles = titles;
	}
	void set_locations(vector<string> &used_locations)
	{
		this->used_locations = used_locations;
	}
	void set_categories(vector<string> &used_categories)
	{
		this->used_categories = used_categories;
	}
	void set_items(vector<ToDoItem> &items)
	{
		this->items_vector = items;
	}
};
```

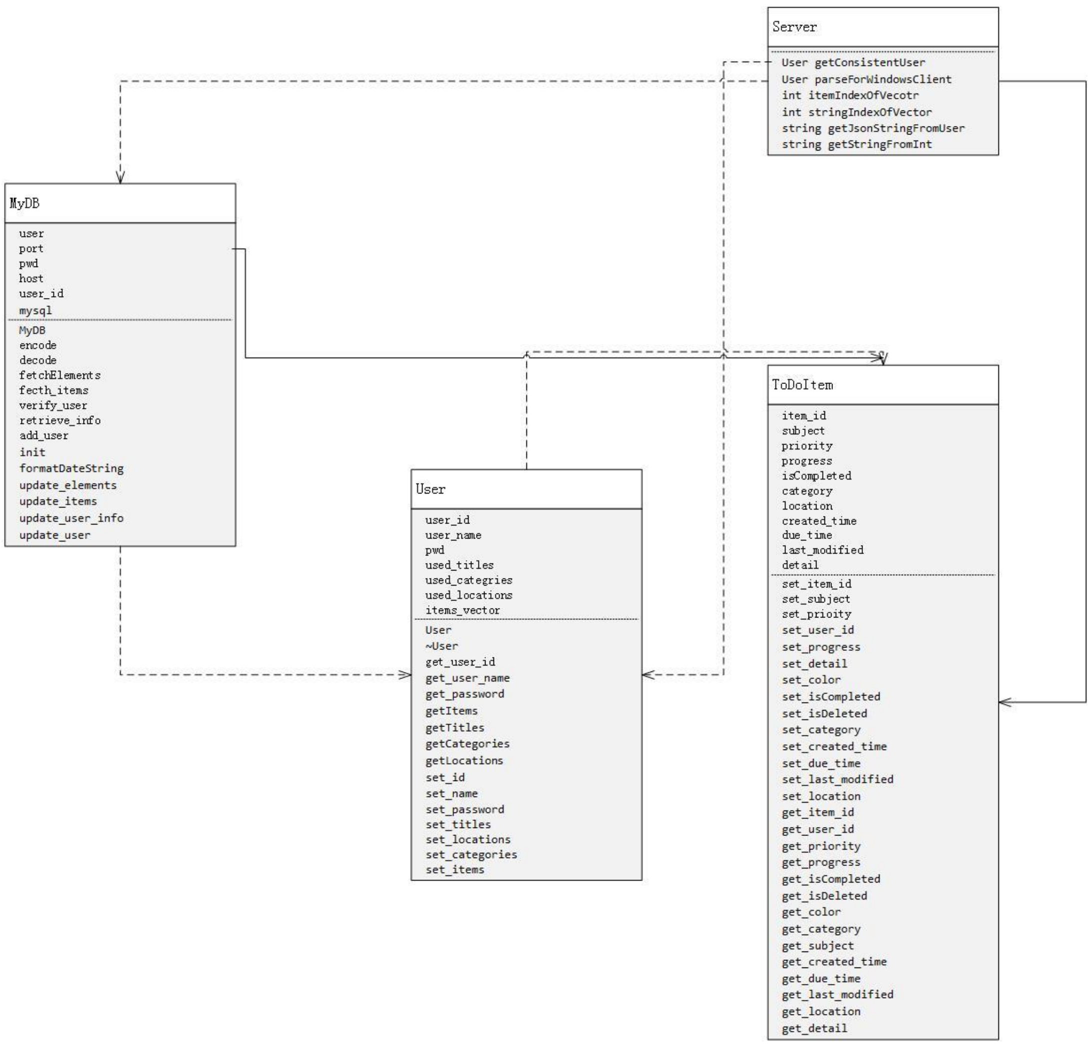


Fig. 4 Server Class Diagram

To make it easier to understand, there is a class diagram to point out the relationships among these four classes shown in Fig. 4. Though there are only four main classes, there is still other accidental functions in Server.cpp to facilitate the whole opereations like function init to initialize the server socket, set non-blocking mode and start listening and at the same time once the remote client connects to the server, it will create a new thread using CreateThread and then it comes to the AnswerThread function which will create its own object of class Server to first handle the data coming from client and second parse them and third use getConsistentUser to get the consistent user according to the last modified time and then fourth return the json string of the consistent user and then return to end the current thread, in this version the more the better rule will be fully applied in the getConsistentUser function which means once the server and the client have different item ids they will all be included comprising the most frequently used titles, locations, categories and to-do items; of course, if they have the same id, it will select the latest modified to-do item.

To make sure the data will be correctly transferred and received, there is a rule between server and clients; due to the structure of the json string transferring among server and clients, the first character is always left brace bracket, they will add the total length of the json string before the json string; when the opposite received the json string, it will firstly get the length character using findIndexOf or split method and once it received enough string according to the predefined length, it will automatically quit receiving and continue with its own stuff. The following function named AnswerThread is the function we will need to create a thread to handle the rest of the communication tasks and the related comments are the best material to explain them.

```
///the accepted socket is its parameter, use this socket to receive and send data;

///creating an object of class Server to handle the data process - parse, update and serialize;

DWORD WINAPI AnswerThread(LPVOID lparam)

{

        char buf[BUF_SIZE];

        int retVal;

        ///get the accepted sClient from main thread;

        SOCKET sClient = (SOCKET)lparam;

        ///if we want to use the functions of the Server, we have to new an object of its class inside the thread function;

        ///the child thread will have its own space, we cannot use the parent's space to share an object of class Server;

        Server server = Server();

        //used to get the current time;

        SYSTEMTIME systemTime;

        GetLocalTime(&systemTime);

        string stringReceived = "";

        bool isJustConnected = true;

        int startPos = 0;

        int receivedLength = 0;

        while (true)

        {

                //receive data from sClient;

                if ((retVal = recv(sClient, buf, sizeof(buf), 0)) == SOCKET_ERROR)

                {
```

```
            if (SOCKET_ERROR == retVal)
            {
                int err = WSAGetLastError();
                if (err == WSAEWOULDBLOCK)
                {
                    Sleep(100);
                    continue;
                }
                else if (err == WSAETIMEDOUT || err ==
WSAENETDOWN)
                {
                    cout << "recv failed " << endl;
                    closesocket(sClient);
                    return -1;
                }
            }
            if (retVal <= 0)
                return -1;
        }
        buf[retVal] = '\0';//form a string;
        stringReceived.append(buf);
        if (isJustConnected)///only the first '{' will be crucial;
        {
            ///json string of an object will be started with a left brace
bracket;
            ///to find the header added length, we need to find the position
of the first '{';
            startPos = stringReceived.find_first_of("{");
            if (startPos > 0 && (receivedLength =
atoi(stringReceived.substr(0, startPos).c_str()) != 0))///presume that the length
characters will at most 20 characters long;
```

```
                        {///there is no int.tryparse function; so we presume that;
                                //cout << stringReceived.substr(0, startPos).c_str() <<
endl;
                                isJustConnected = false;///only the first '{' will be that
curcial;
                                stringReceived                                          =
stringReceived.substr(startPos);///substract the length header;
                        }
                        else//error;
                                return -1;
                }
                if (int(stringReceived.length()) == receivedLength)
                        break;
        }
        ///according to the json string, get the user from client;
        User user = server.parseForWindowsClient(stringReceived);
        ///according to the client user, withdraw all its data from database and handle
the conflicting items;
        User user2 = server.getConsistentUser(user);
        MyDB myDB;
        ///using the updated user info to update the database;
        myDB.update_user(user2);
        ///get the json string via the updated user;
        string stringForSend = server.getJsonStringFromUser(user2);
        ///add the total length before the json string;
        stringForSend      =      server.getStringFromInt(stringForSend.length())      +
stringForSend;
        while (true)
        {
                ///using the consistent data to form a formatted data which the client
will be able to handle
```

```
                ///using TCP protocol;
               retVal = send(sClient, stringForSend.c_str(), stringForSend.length(), 0);
                if (SOCKET_ERROR == retVal)
                {
                        int err = WSAGetLastError();
                        if (err == WSAEWOULDBLOCK)
                        {
                                Sleep(100);
                                continue;
                        }
                        else
                        {
                                cout << "send failed" << endl;
                                closesocket(sClient);
                                return -1;
                        }
                }
                break;
        }
        closesocket(sClient);
}
```

3.2 Windows Client

This part is mainly used to discuss the overall design decisions made for better user interaction, which will specifically consist of four major parts, filtering panel used to filter the current list presented, body panel containing the to-do and cancelled list, left-side panel controlling the most significant features of the program and the bottom edition panel for reediting the selected to-do item. There is a simple use case diagram shown in Fig. 5 which may simplify the understanding of the whole framework of the Windows client which is also partly applicable to the other clients.

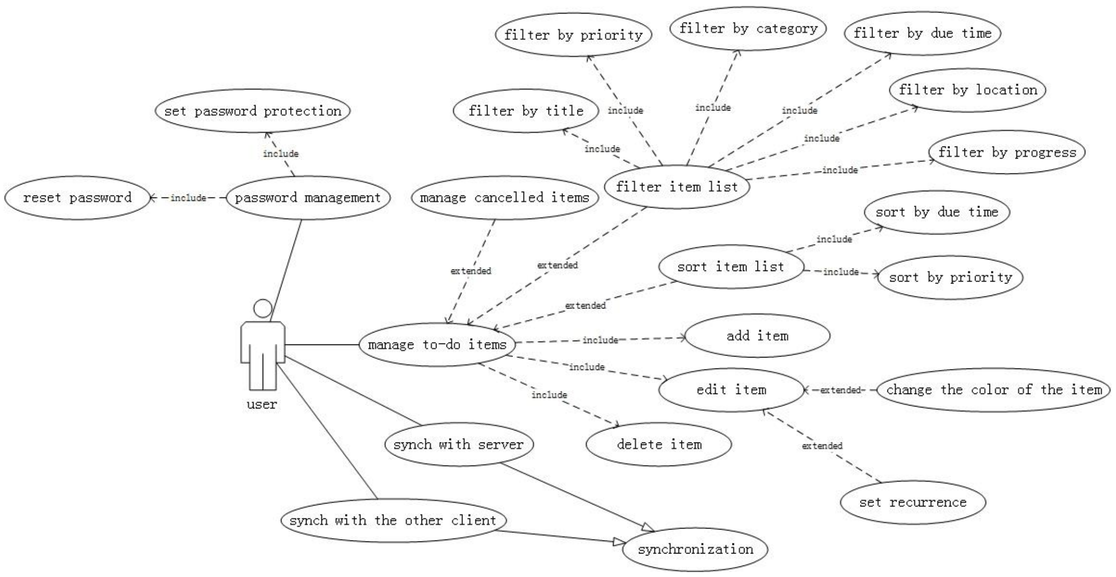
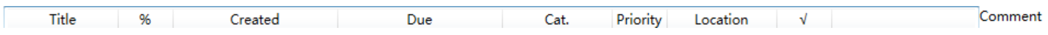


Fig. 5 Use Case Diagram

3.2.1 Features

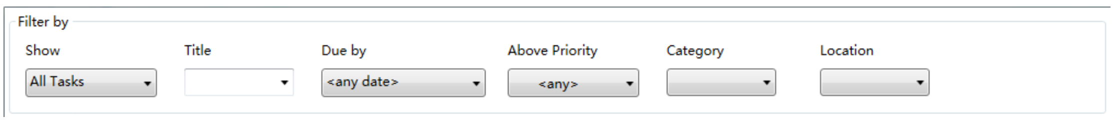
Before we get started on our journey, we need to first know what we are going to achieve here, a to-do list program which will run in windows and act as a reminder to record the to-do items for users. And the basic functions we have to provide here comprising title which will remind the user what the topic is all that task about, created and due time used to help the user to control the progress and of course a progress attribute will also be included to customize the overall status of the task, priority used to assist the user to pay more attention to the more critical affairs, location and category used to facilitate filtering and reminding the details of the to-do task and some other auxiliary attributes as shown in the following Fig. 6.



Title	%	Created	Due	Cat.	Priority	Location	√	Comment

Fig. 6 Some Attributes of the To-Do Task

As for the control of these attributes mentioned above in this application, filtering panel which is displayed in Fig. 7 is specifically used to filter the to-do list and the filtering factors consist of the following six choices.



Filter by

Show	Title	Due by	Above Priority	Category	Location
All Tasks		<any date>	<any>		

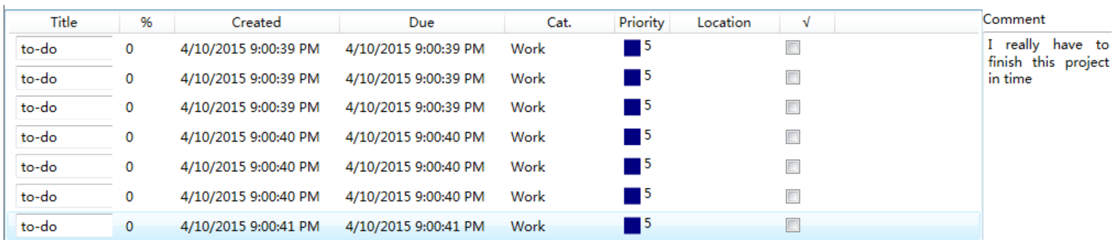
Fig. 7 the Filtering Panel

1. Show including all – show all the to-do list, complete – only present the complete ones, incomplete – merely display all the incomplete to-do items, recently modified – show all the items modified in the last one hour;
2. Title used to filter the to-do items via the title using RE (Regular Expression) and

while using this feature the program will automatically record the input sought titles for later convenience, even after closing the application;

3. Due by – to select the items before a certain date including the most frequently used date like now which means within an hour, today – before the midnight, tomorrow – before the midnight of tomorrow, the end of this week and the end of this month; besides when the user tries to select another date out of this range, the user can even select the <specific date> to use a date picker to choose a personally selected date to filter the items in the list;
4. Above priority – used to select most significant items whose priority is higher than or equal to the priority of the selected to-do task;
5. Category – when the user tries to select a specific type of the to-do tasks, this feature will help a lot;
6. Location – personally when using a mobile phone to record a to-do task, I will prefer to get the location of the current position and correspondingly this feature should be accomplished in Windows client, reaching coherence among different clients.

Now let's move to the body, the to-do or cancelled list in the center of the panel which is shown in Fig. 8. After discussing so much about the basic features of the filtering panel, it's time to move further to the list under the filtering panel containing the to-do tasks. In this simple list, as a designer, what I mainly considered is how to make it look better and simpler for the user; so as a result, there are only the most significant factors presented in the header of the list, furthermore, the user can sort the list via simply clicking the corresponding header title to fit his or her needs. I also prepared several other features to assist the user to more easily control his or her tasks list which will contain the following four aspects.



Title	%	Created	Due	Cat.	Priority	Location	√
to-do	0	4/10/2015 9:00:39 PM	4/10/2015 9:00:39 PM	Work	🟦 5		☐
to-do	0	4/10/2015 9:00:39 PM	4/10/2015 9:00:39 PM	Work	🟦 5		☐
to-do	0	4/10/2015 9:00:39 PM	4/10/2015 9:00:39 PM	Work	🟦 5		☐
to-do	0	4/10/2015 9:00:40 PM	4/10/2015 9:00:40 PM	Work	🟦 5		☐
to-do	0	4/10/2015 9:00:40 PM	4/10/2015 9:00:40 PM	Work	🟦 5		☐
to-do	0	4/10/2015 9:00:40 PM	4/10/2015 9:00:40 PM	Work	🟦 5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	🟦 5		☐

Comment

I really have to finish this project in time

Fig. 8 the To-Do List

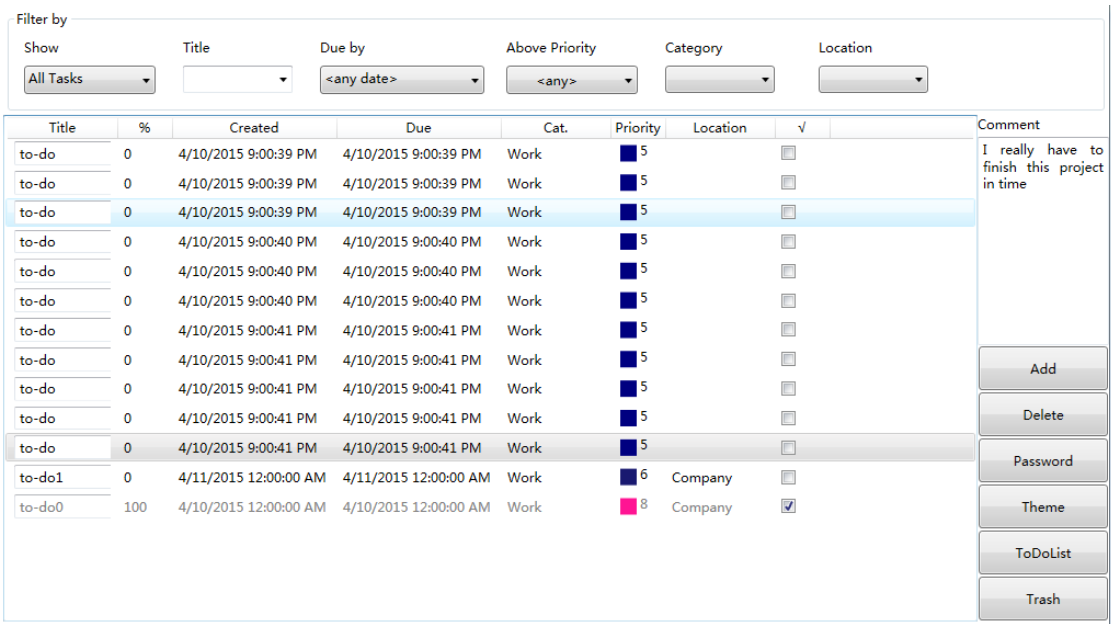
1. Editable title – via this feature the user can very easily and conveniently edit the title of the selected to-do task as he or she wishes;
2. Colorful priority – not only with a number but with a colorful rectangle, which may seem quite normal but with this simple and seemingly stupid feature the user can quite easily catch and spot the most significant task in a tedious list;
3. Checking box – this checking box designed to make the ignoring or finishing operation handled easier for the user; when the user tries to set a task to complete status or just wants to ignore that task but not delete it, he or she can just check it and the program will handle the rest of it setting the progress of the checked task to 100 and make the color of that task gray to make it less outstanding; besides the user even can sort the checking box to make the checked tasks at the bottom

of the list – ignoring it, finishing it simply and completely!

4. Side comment box – once the user selected a to-do task in the list, the comment box will show the details of it; especially this comment box is to make the task more detailed and specific, some users may want to add some comments to the task to make it more vivid, readable and understandable, as a result such comment box designed to meet this kind of requirement.

After discussing the main body of the program, it's time to know more about its controlling functions which can control the whole process of the application, the left-side panel containing about seven buttons, which is presented partly in Fig. 9.

1. Add – used to add a bran-new task to the list for further edition by the edition panel;
2. Delete – used to cancel a task from the to-do list and when deleting a task from the trash, that item will be permanently deleted, of course after clicking Ok in the confirmation dialog;
3. Restore – used to restore a cancelled task back to to-do list saving some inconvenience for the user especially the cancelled one is quite similar to the wanted one – needless to add a bran-new task to edit it for further use;
4. Password – this function is prepared to protect personal privacy via network checking process; with this feature the user can protect his or her personal to-do list from others peeking;
5. Theme – this feature is designed to make the application more human to fit different users' taste;
6. TodoList – the default showing list of the program, but when changing to the trash list only via clicking this button the user can go back to the to-do list;
7. Trash – clicking this button will guide the user to the cancelled to-do tasks where the user can further delete the items or restore them to reuse them.


Filter by

Show	Title	Due by	Above Priority	Category	Location
All Tasks		<any date>	<any>		

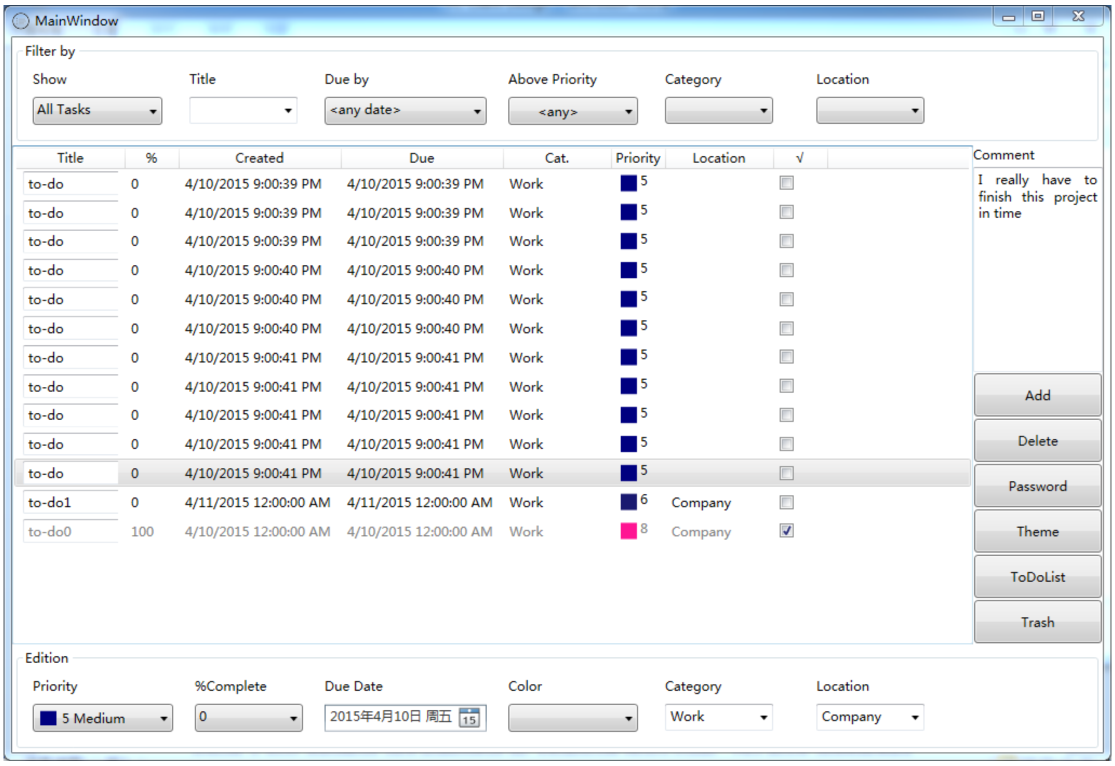
Title	%	Created	Due	Cat.	Priority	Location	√
to-do	0	4/10/2015 9:00:39 PM	4/10/2015 9:00:39 PM	Work	5		☐
to-do	0	4/10/2015 9:00:39 PM	4/10/2015 9:00:39 PM	Work	5		☐
to-do	0	4/10/2015 9:00:39 PM	4/10/2015 9:00:39 PM	Work	5		☐
to-do	0	4/10/2015 9:00:40 PM	4/10/2015 9:00:40 PM	Work	5		☐
to-do	0	4/10/2015 9:00:40 PM	4/10/2015 9:00:40 PM	Work	5		☐
to-do	0	4/10/2015 9:00:40 PM	4/10/2015 9:00:40 PM	Work	5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	5		☐
to-do1	0	4/11/2015 12:00:00 AM	4/11/2015 12:00:00 AM	Work	6	Company	☐
to-do0	100	4/10/2015 12:00:00 AM	4/10/2015 12:00:00 AM	Work	8	Company	☑

Comment

I really have to finish this project in time

Fig. 9 Left-side Buttons

Before we end the specification of the Windows client, we should not forget about edition feature used to edit the detailed contents of the selected to-do task instead of just changing some simple attributes in the list. The edition part is designed into the edition panel at the bottom of the panel, shown in Fig. 10, before the selection operation of the list, that panel is gray which means the user is unable to edit for the time being; however when the user selects a to-do item, the panel will be activated and at the same time the details of the selected to-do item will also be copied in the corresponding UI elements except for color, and now the user can do whatever he or she wants from the application, changing the priority of the currently selected task, the progress in %Complete, the due date, the category, the location and even the color of that selected task which makes this application more colorful and friendly, what's even more friendly here is that the category and location element can be manually input to add new values, not only selecting the existed values.



Filter by

Show	Title	Due by	Above Priority	Category	Location
All Tasks		<any date>	<any>		

Title	%	Created	Due	Cat.	Priority	Location	√
to-do	0	4/10/2015 9:00:39 PM	4/10/2015 9:00:39 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:39 PM	4/10/2015 9:00:39 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:39 PM	4/10/2015 9:00:39 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:40 PM	4/10/2015 9:00:40 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:40 PM	4/10/2015 9:00:40 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:40 PM	4/10/2015 9:00:40 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	■ 5		☐
to-do	0	4/10/2015 9:00:41 PM	4/10/2015 9:00:41 PM	Work	■ 5		☐
to-do1	0	4/11/2015 12:00:00 AM	4/11/2015 12:00:00 AM	Work	■ 6	Company	☐
to-do0	100	4/10/2015 12:00:00 AM	4/10/2015 12:00:00 AM	Work	🟥 8	Company	☑

Comment

I really have to finish this project in time

Edition

Priority	%Complete	Due Date	Color	Category	Location
■ 5 Medium	0	2015年4月10日 周五		Work	Company

Fig. 10 the Whole Panel of the Application

3.2.2 Windows Client Realization

After introducing the features and functions of this application, we have to also try to understand how to realize all these and what the challenges are. This part is going to mainly discuss the details of the realization of the application which may seem tedious but some can be pretty challenging and thought-provoking especially for the computer programmers.

The presenter layer of this application is completely completed by WPF (Windows Presentation Foundation) which a tool designed and populated by Microsoft these years. The most outstanding feature of WPF is called data-binding, a quite different mechanism to use data to control the UI elements compared with MFC and java swing/awt using message-driven or event-driven mechanism. The most two outstanding examples applied in this application will be explained – edition and filtering.

3.2.2.1 Edition

Editing selected to-do task, as the feature part mentioned above, we know that there are several ways for the user to edit the selected to-do task in list, in comment box and in the edition panel. Here I will try to explain them in these three aspects.

1. Edition in list – due to the binding mechanism each task presented is bond to an existing task object; once the user edit the item in the list like changing the title which will directly change the title in the bond to-do object, checking the checking box which will directly set the complete status true and indirectly set the progress to 100 via another bond;
2. Comment box – as for comment of a currently selected to-do item, I use a lost focus event handler to cope with the edited comment for that task; after completing the edition of the comment, the user will get out of the comment box – whatever he or she will do, then the lost focus event handler will be invoked and indirectly set the comment of the currently selected task stored globally with the current value of the comment box – as with globally stored problem, I will explain further in the next passage;
3. Edition panel – as I have mentioned in the previous passage, I stored the currently selected to-do task in a global object for easier access especially for the UI elements in edition panel; after selecting an task in to-do list, the current global current task will be set to the selected task and the edition panel will be refreshed according to the current to-do task with its values including priority, progress, due time, category and location; and now if the user tries to change the priority of the item, he or she can easily change the priority in priority ComboBox when the priority selection changed, the selection changed event handler will be invoked and indirectly set the current selected priority to the current to-do task all these rules can be applied to the progress, due time and color; what's even more interesting is that due to the binding mechanism, the changed value will be instantly passed to the UI element in the list presenting different priority and priority color, showing with different color and due time and the like; however there is a little difference in category and location where the user can input new value into the ComboBox and at the same time also because of the binding mechanism the filtering category and location will also have the new input values, due to binding to the same list.

3.2.2.2 Filtering

Filtering operation, as we have discussed before, there are six factors to filter the current list including to-do list and cancelled list; because of the independence of the six factors, each time the user select a filtering condition the application will record globally and each time it will invoke the filter method to filter the current list, which will filter the list by show type, by title, by due, by priority, by category and by location in order, though some values may be absent but for robustness and convenience, I choose to filter by all these factors once one of the six filtering conditions changed; besides just like the category ComboBox in edition panel, the title ComboBox will also record each input filtering values for later use and further for more friendly interaction experience.

Before we finally end this part, some other details we should also pay some attention to. When starting the program, the application itself will automatically use the stored xml file to withdraw all the user's information to initialize itself and in the end when the user wants to close the application, the program itself again will automatically store all the modified data in the local xml file, the same file mentioned earlier, which accomplished by catching the closing event of the application. Another little trick is about the cold start which will cause some problems when first using this application, I handle this by using a try catch block when trying to read from a xml file which accidentally does not exist in the expected directory, the program will catch the exception and then automatically create a xml file and write some formatted data in it for further reading. There are still more specifics here, but considering the limitation of this report, only the most outstanding ones are picked to display its main features and realization.

Along with all realization details, the class MyConnector is used to communicate with server to synchronize and just like server there is a class named user containing all the related info a user should have similar to the user in server side; also the sending and receiving process both need json serialization and deserialization which will be completed by JsonConver.SerializeObject and getUserFromJson separately with the second method realized by myself to parse the json string to a user object. To make the whole realization more understandable, there is a class diagram shown in Fig. 11 which will show the overall structure of the whole application in Windows client.

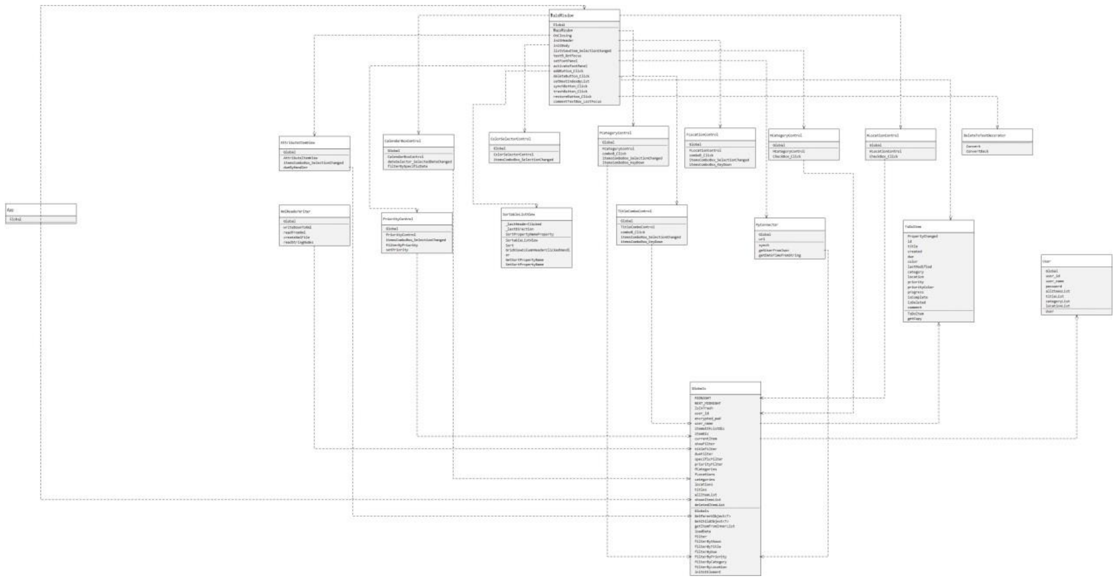


Fig. 11 Windows Client Class Diagram

To make the item id unique, in Windows Client I store the biggest id in the xml file and when loading the data, the application will check the stored max id making sure that it's the biggest id and once it's less than the id of the stored to-do item, we need to reset it then; to allocate a unique item id in Windows Client is pretty complex, while the Android Client can just use UUID to get the unique id simply. To better understand the workflow of the class, MyConnector in Window Client, there is a critical function to handle the most of its interaction with the server.

```
public void synch(User user)
    {
        TcpClient tcpClient = null;
        NetworkStream stream = null;
        try
        {
            tcpClient = new TcpClient("127.0.0.1", 8888);
            stream = tcpClient.GetStream();
            //string stringJson = JsonConvert.SerializeObject(Global.allItemList);
            string stringJson = JsonConvert.SerializeObject(user);
            string stringForSend = stringJson.Length + stringJson;
            byte[] dataForSend = Encoding.UTF8.GetBytes(stringForSend);

            stream.Write(dataForSend, 0, dataForSend.Length);
            stream.Flush();

            byte[] dataReceived = new byte[1024 * 60];
            string stringReceived;
            while (true)
            {
                int receivetCount = tcpClient.Client.Receive(dataReceived);
                dataReceived[receivetCount] = (byte)'\0';
                stringReceived = Encoding.UTF8.GetString(dataReceived);
                Array.Clear(dataReceived, 0, dataReceived.Length);
                if (stringReceived.Contains('{'))//only this formatted string can be further
processed;
                {
                    int startPos = stringReceived.IndexOf('{');
```

```
                        int stringLength;
                        if (int.TryParse(stringReceived.Substring(0, startPos), out stringLength))
                        {
                            stringReceived = stringReceived.Substring(startPos);
                        }
                        while (true)///receiving data from server;
                        {
                            if (stringLength <= stringReceived.Length)
                            {
                                break;
                            }
                            tcpClient.Client.Receive(dataReceived);
                            stringReceived += Encoding.UTF8.GetString(dataReceived);
                        }

                        User user1 = getUserFromJson(stringReceived);
                        Global.refreshAllItems(user1);
                        break;
                    }
                    Console.WriteLine(stringReceived);
                    break;
                }
            }
            catch(Exception e)
            {
                Console.WriteLine("Synchronization process error!**************" + e.ToString());
            }
            finally
```

```
        {
           if(stream != null)
              stream.Close();
           if(tcpClient != null && tcpClient.Client != null)
              tcpClient.Client.Close();
           if(tcpClient != null)
              tcpClient.Close();
        }
    }
```

3.3 Android Client

In this part, we are going to focus on the design, functional features and some essential realization details about the application in Android client. Unlike the Windows client, the Android client is a function extension of this application, so only the basic and a little different features will be included in this client. The basics are title, due time, created time, details and solved status and the like; as for the extension due to the specialty of current mobile phones, the location retrieved from nearest AP which takes the advantage of the GPS system of the phone or from the local network provider to get the current accurate position, dislike the traditional uneasy input method in Windows client.

After the overall discussion about the Android client, let's get back to its specific features. First we need to install the application in an Android mobile phone and then find its launching icon with the name WillDo and start it as is shown in Fig. 12.

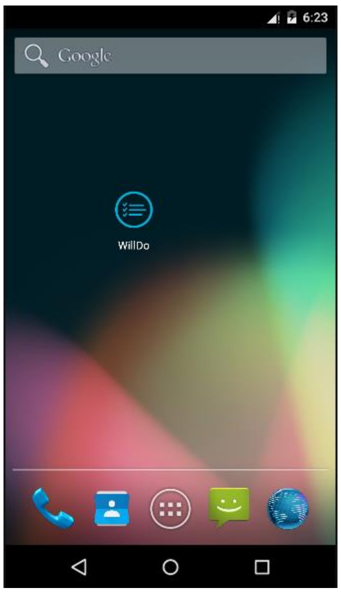
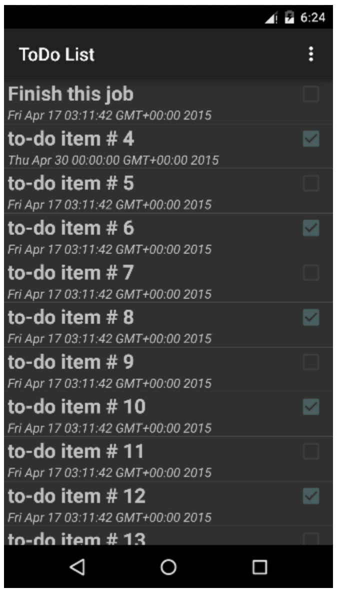


Fig. 12 Launch Icon of the Application

Now the main panel comes to our sight shown in Fig. 13, it’s time for us to get something done here where we can delete to-do tasks by long press and select the items we want to delete presented in Fig. 14 in which case the application will also prompt you to confirm your choice displayed in Fig. 15 and after your second-time confirmation she will help you to delete the items permanently which is quite different from the Windows client containing a trash to contain all the deleted tasks.

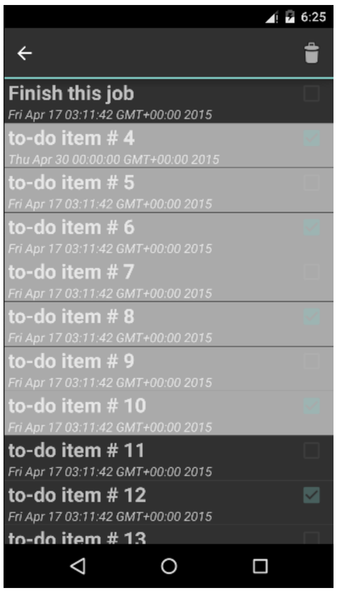


6:24

ToDo List

Finish this job
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 4
Thu Apr 30 00:00:00 GMT+00:00 2015
to-do item # 5
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 6
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 7
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 8
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 9
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 10
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 11
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 12
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 13

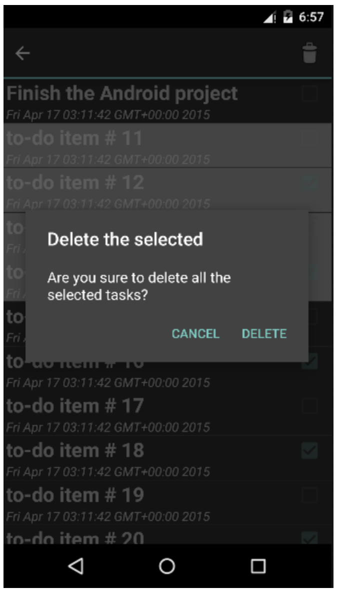
Fig. 13 Main Panel of WillDo



6:25

Finish this job
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 4
Thu Apr 30 00:00:00 GMT+00:00 2015
to-do item # 5
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 6
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 7
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 8
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 9
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 10
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 11
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 12
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 13

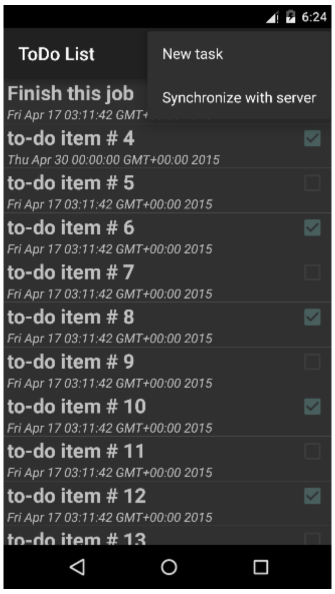
Fig. 14 Selecting Several Items to Delete



6:57
Finish the Android project
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 11
to-do item # 12
Delete the selected
Are you sure to delete all the selected tasks?
CANCEL
DELETE
to-do item # 17
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 18
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 19
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 20

Fig. 15 Let the User to Confirm the Deleting Operation

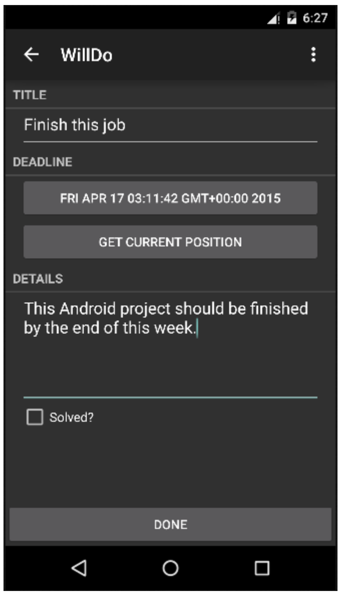
Of course, we have to add new to-do task to our list displayed in Fig. 16, so there it is in the top menu by which we can add a new item to get more organized. In the main panel not only can we add and delete, in fact at the same time there is a small thread running background to store all the changed values back into the local storage sandbox called in Android mobile phone.



6:24
ToDo List
New task
Synchronize with server
Finish this job
Fri Apr 17 03:11:42 GMT+
to-do item # 4
Thu Apr 30 00:00:00 GMT+00:00 2015
to-do item # 5
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 6
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 7
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 8
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 9
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 10
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 11
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 12
Fri Apr 17 03:11:42 GMT+00:00 2015
to-do item # 13

Fig. 16 Add New Task

After so many edition operations and changes to each to-do task, maybe the application someday can collapse or break down or the user just want to use the to-do list in a computer instead of mobile phone environment; at that time to handle this kind of situation, a server designed for the this application to store the clients' data in the remote database which will ensure the data will be kept safely and sound by manual synchronization at the top of the main panel shown in Fig. 16 too, which will enable the user to use it more conveniently anywhere he or she wants. The user's requests are the criteria; all we do is just for one purpose, the user's convenience and betterment.

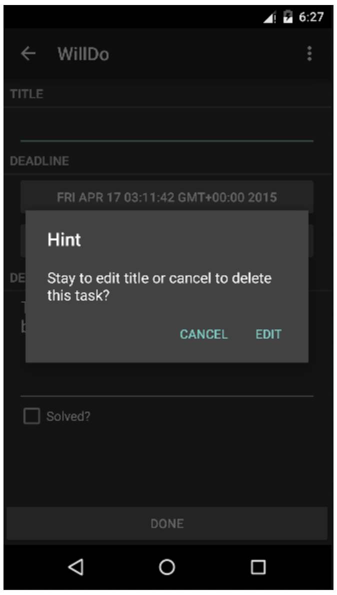


Finish this job

FRI APR 17 03:11:42 GMT+00:00 2015

This Android project should be finished by the end of this week.

Fig. 17 Details of the To-Do Task



Hint

Stay to edit title or cancel to delete this task?

Fig. 18 Encounter Empty Title Problem

As mentioned earlier, after adding one bran-new to-do task the application will directly guide the user to edit the newly created task shown in Fig. 17 and at the same time she will automatically store the new one into local storage. In the detail panel the user can edit the title whose default value is to-do item, the due time, the current position retrieving by GPS or network provider, the details of the task and its status whether handled or not which will be presented in the main panel also to remind the user of the current status of the task, after all these edition the user may 'done' the process by clicking the done button at the bottom of the panel or just click go back menu at the top of the panel and at that time the application will check the title and if the title is empty she will demand the user to add it or just cancel to delete the whole new task displayed in Fig. 18, a task without title is meaningless in this application.

To make the whole process friendly and convenient enough, the user can also delete the to-do task in the detail panel by clicking the top menu of the panel presented in Fig. 19 and of course to make the process more secure, the application will again need the user to confirm his or her decision as is shown in Fig. 20.

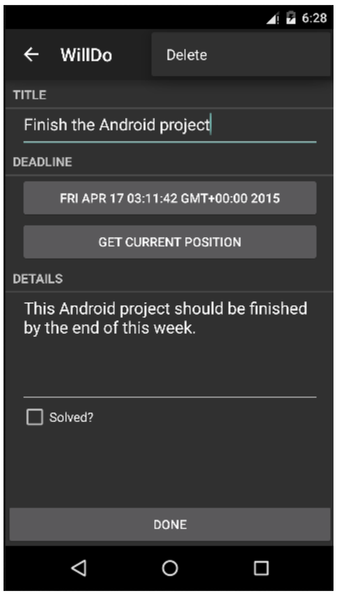

6:28
WillDo
Delete
TITLE
Finish the Android project
DEADLINE
FRI APR 17 03:11:42 GMT+00:00 2015
GET CURRENT POSITION
DETAILS
This Android project should be finished by the end of this week.
Solved?
DONE

Fig. 19 Delete To-Do Task in Detail Panel

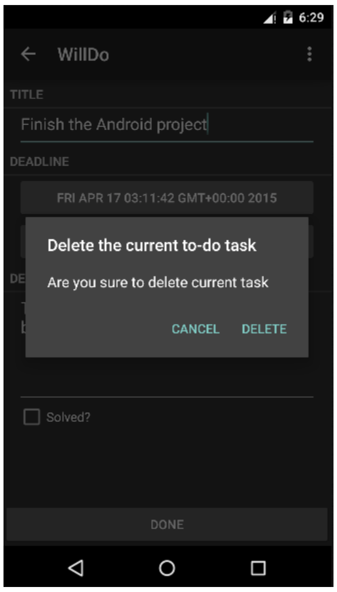

6:29
WillDo
TITLE
Finish the Android project
DEADLINE
FRI APR 17 03:11:42 GMT+00:00 2015
Delete the current to-do task
Are you sure to delete current task
CANCEL
DELETE
Solved?
DONE

Fig. 20 Let the User to Confirm Deleting

We have discussed these earlier but now let’s get some details about setting due time, adding details and locating our position. As long as the user is in the detail panel coming from by clicking the item in the list in main panel or by adding new task, the user can edit each element in detail panel except for the created time which is set as soon as the item created. To set the due time, the user can simply click the due time button and select the specific date he or she prefers as shown in Fig. 21; about the details part, the user can just click the detail text field and add whatever text he or she wants; but there is a little different about location, when the user tries to get the current position by clicking the ‘GET CURRENT POSITION’ button, he or she must ensure the GPS or network works right otherwise the application will prompt the user about this kind of error shown at the bottom of the panel in Fig. 22 and she will fail to get the current location.

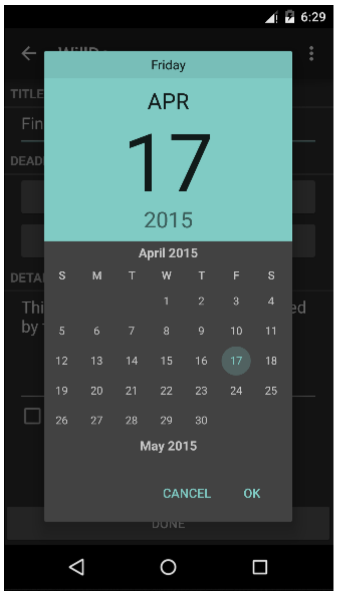
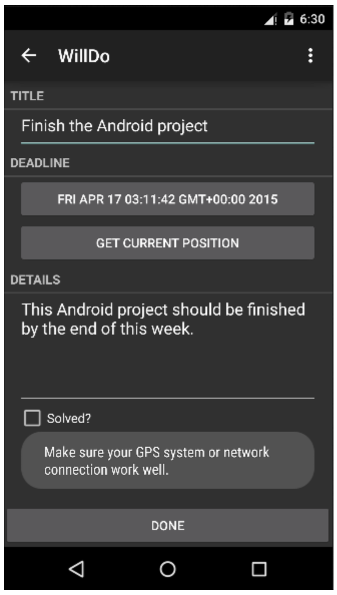


Fig. 21 Selecting Due Time



Finish the Android project

FRI APR 17 03:11:42 GMT+00:00 2015

This Android project should be finished by the end of this week.

Make sure your GPS system or network connection work well.

Fig. 22 Getting Current Position Fails

There are still some details which have not been mentioned in previous context and we should also know about this small application which makes her more robust, simpler and more convenient to use. First, when the user rotate the mobile phone the current situation will not change a little bit for the saved instance values before the rotation being promptly applied by the phone after the rotation; second, when the user tries to check the next or previous to-do task, he or she can easily slide the current task to the left for the next and to the right for the previous; third, when the user adding a new task, he or she will be directly guided to the detail panel for further edition instead of staying the same main panel containing the new task at the bottom of the list and waiting for the user to click it and then edit its details; fourth, each time the user tries to delete some tasks, the application will prompt the user to confirm the operation again in case of some accidents; fifth, to make the to-do list more clear, the application require the user to add title for the task and once he or she forgets about it, the application will force him or her to add it; sixth, when sliding the detail panel, the user can spot the title much obviously due to the title of the current task being displayed at the top of the detail panel by the application; seventh, each time when the application encounters some problems including the failure of getting location or synchronizing with the server, she will prompt the user about it and tries to overcome it herself first. Of course, still there are lots of problems and inconveniences inside this version of the small application, but I will try to further polish and optimize it in UI design, robustness, feasibility and portability.

3.3.1 Android Client Realization

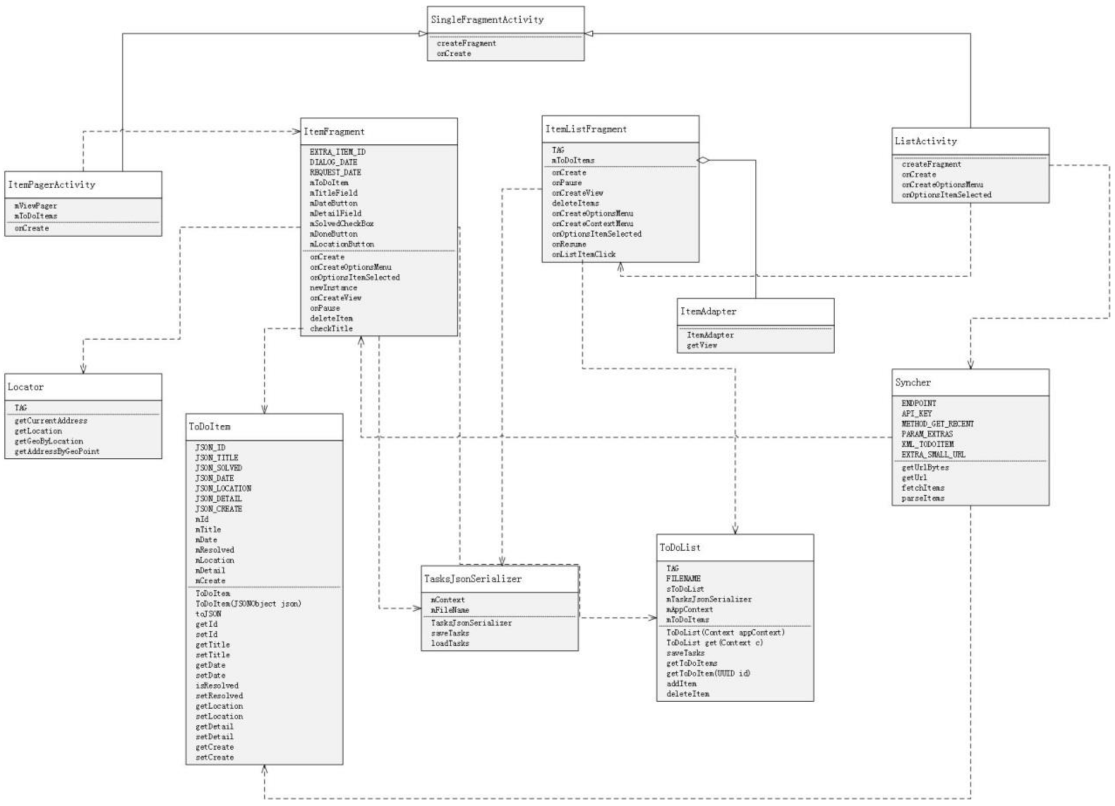


Fig. 23 Android Client Class Diagram

Unlike the skeleton of the description in Windows client part, the class diagram here is displayed in Fig. 23 firstly to make it easier to understand the overall structure. To make full use of the commonness between ItemPagerActivity and ListActivity, there is an abstract class named SingleFragmentActivity which will contain a single fragment as its name suggests presenting the whole panel to the user; and to make it further simple, the fragment is feasible that can be changed at run time to make the UI more colorful and friendly, what's more meaningful is that the whole application is easy to be constructed and reconstructed in the near future for new requirements; in the ItemPagerActivity, I took advantage of ViewPager to make the panel can be slide to the left and right to get the next and previous task panel. To make the detail more specific, there are several lines of codes to show themselves.

```
///using ViewPager for sliding feature;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        ///using ViewPager to slide
        mViewPager = new ViewPager(this);
        mViewPager.setId(R.id.viewPager);
        setContentView(mViewPager);
        mToDoItems = ToDoList.get(this).getToDoItems();
        FragmentManager fm = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fm) {
            @Override
            public Fragment getItem(int i) {
                ToDoItem item = mToDoItems.get(i);
                return ItemFragment.newInstance(item.getId());///passing data by this
method ensure further security;
            }


            @Override
            public int getCount() {
                return mToDoItems.size();
            }
        });
        mViewPager.setOnPageChangeListener(new
ViewPager.OnPageChangeListener()
        {
            public void onPageScrollStateChanged(int state){}
            public void onPageScrolled(int pos, float posOffset, int posOffsetPixels){}
            public void onPageSelected(int pos)
            {
                ToDoItem item = mToDoItems.get(pos);
                if(item.getTitle() != null)
                {
                    setTitle(item.getTitle());
                }
            }
        });


        UUID itemId =
(UUID)getIntent().getSerializableExtra(ItemFragment.EXTRA_ITEM_ID);
        for(int i = 0; i < mToDoItems.size(); i++)
        {
            if(mToDoItems.get(i).getId().equals(itemId))
            {
                mViewPager.setCurrentItem(i);
                break;
            }
        }
        //////using ViewPager to slide
    }
```

The codes above are used to realize the sliding page feature, which is a typical example to show the implementation details of the UI part. As for the underlying data process and transferring part, we have ToDoItem, ToDoList and TasksJsonSerializer to handle the storing, loading and transferring function, while the Syncher cope with the data transformation between client and server. Now let's check some details about it in the following codes including three methods named getUrlBytes, a simplified version of transferring data method in Syncher, saveTasks and loadTasks used to handle the data storing back to the underlying sandbox and retrieving from it.

```
private byte[] getUrlBytes(String urlSpec) throws IOException
```

```
  {
    URL url = new URL(urlSpec);
    HttpURLConnection connection = (HttpURLConnection)url.openConnection();
    try
    {
      ByteArrayOutputStream out = new ByteArrayOutputStream();
      InputStream in = connection.getInputStream();
      if(connection.getResponseCode() != HttpURLConnection.HTTP_OK)
        return null;

      int bytesRead = 0;
      byte[] buffer = new byte[1024];
      while ((bytesRead = in.read(buffer)) > 0)
      {
        out.write(buffer, 0, bytesRead);
      }
      out.close();
      return out.toByteArray();
    }finally {
      connection.disconnect();
    }
  }

  public void saveTasks(ArrayList<ToDoItem> toDoItems) throws JSONException,
IOException
    {
      JSONArray array = new JSONArray();
      for(ToDoItem item : toDoItems)
      {
            array.put(item.toJSON());
        }


        Writer writer = null;
        try
        {
            OutputStream out = mContext.openFileOutput(mFileName,
Context.MODE_PRIVATE);
            writer = new OutputStreamWriter(out);
            writer.write(array.toString());
        }
        finally {
            if(writer != null)
                writer.close();
        }
    }


    public ArrayList<ToDoItem> loadTasks() throws IOException, JSONException
    {
        ArrayList<ToDoItem> toDoItems = new ArrayList<ToDoItem>();
        BufferedReader reader = null;
        try
        {
            InputStream in = mContext.openFileInput(mFileName);
            reader = new BufferedReader(new InputStreamReader(in));


            StringBuilder jsonString = new StringBuilder();
            String line = null;
            while((line = reader.readLine()) != null)
            {
               jsonString.append(line);
            }


            JSONArray array = (JSONArray) new
JSONTokener(jsonString.toString()).nextValue();


            for(int i = 0; i < array.length(); i++)
               toDoItems.add(new ToDoItem(array.getJSONObject(i)));
         }catch (FileNotFoundException e)
         { }
         finally {
            if(reader != null)
               reader.close();
         }
         return toDoItems;
      }
```

As we can see from the two methods above, saveTasks and loadTasks, there are two strange usages of class ToDoItem, item.toJSON() used to serialize the object using json format, new ToDoItem(array.getJSONObject(i)) used to deserialize the json string back to the object which is quite awesome compared with that process in C++ and C# in which case we have to manually handle the tedious operations.

There are still a lot details can be presented here, but the format here is quite different from that in IDE, so if you really want to check the specifics, the source code along with the report is recommended; besides the current version of the whole application is pretty coarse and crude, not only the whole framework but also the realization details; I will pay more attention to this kind of issues after the forthcoming deadline and exams.

4. Improvements

In this section, I will try my best to lay out an overview of the system and make some future plans and suggestions which cannot be realized before deadline but can be further polished in the near future not only in GUI of clients but also some essential underlying realization methods. For example, I will try to use Node.js to replace C++ to realize the server side in which case the Node.js is a bran-new tool and more convenient in http network programming to accomplish asynchronization and light-weighted server for web application; by the way, a web client will also be accomplished in this semester to accompany the Windows client and Android client in this system. There will be three main parts to discuss the improvements, server side, Windows client and Android client.

4.1 Server

Currently the server is completely realized by C++ socket which is quite hard to use and pretty difficult to achieve a high network communication load; as we know there is a tool called ZeroMQ which is an expert tool package used to handle all the network programming affairs in C++, ensuring high network load, network security and so many effective features which will dramatically improve the whole server performance and security. Before I give up on C++ I will try to use ZeroMQ to reimplement the whole server to try to test the whole performance of the system in the end. Of course, we should also consider the database operations but due to the stupidity of the current design, the whole database is quite simple and not too much stuff we need to think about.

After all the previous improvements using C++, I will also try to reimplement the whole server using a completely tool named Node.js as mentioned before which will, I believe, dramatically decrease the workload; as for the details of it, I will try to add it up after finishing it in person.

4.2 Windows Client

There are still a lot more for me to accomplish including first password protection which can be activated and deactivated by the user any time he or she prefers – when password protection activated the user will be required to enter the password before editing and even checking any to-do tasks, protecting personal privacy, second theme customization using not just the default style but with personal picture as the background and third synchronization with not only server but a mobile phone in which case the user can use the Windows client as a server to let the Android client to connect and synchronize data between them. There are still few features I will try to finish before the deadline.

Of course, as we have expected the whole application in Windows is still rather simple and can be more delicate including more features to manage more stuff like in a company; but considering that kind of aspect will not be our business here, we will try to make it more convenient and useful in daily personal use. Maybe after a short-time use of this application, more ideas will come to me and I will try to accomplish them at that time.

4.3 Android Client

Compared to the Windows client, there are too much to reimplement in Android including first the most severe part GUI which is too coarse for the time being, second the sorting feature which is pretty useful in Windows client, third filtering by different factors, fourth editing the progress of the task, fifth customizable to-do item in the list which will make the whole style of the application more colorful and friendly, sixth trash which can be used to accommodate all the cancelled tasks that can be restored in future saving inconvenience of renewing a new one and the like. The following content will detail the whole imperfection of this small application and tries to specify the overall solutions in the near future.

GUI is a very critical element to determine the success of an application; but right now the outlook of the Android client is rather simple and crude. First, adding a quite different refreshing launching icon is the first thing we can do to change the first impression of the application; second decorating the main panel will also play an important role in attracting the user to use this application more frequently, as for the details that we can enable the user to change the background of the main panel with his or her favorite picture, make the items in the list more cute or cool; third adding some animations in essential process which will definitely affect the user interaction process and further attract the user's attention.

Searching feature is another important function that should be included in an application like this which will enable the user to find the certain items more easily; making the user more conveniently is making the application more popular and charming. And filtering now is kind of an auxiliary feature that will make the functions of this application more optional and meet different requirements from diverse users; sometimes replaceable functions are rather important especially in different circumstances.

Trash is also a quite useful feature in such application which will make the user easier to renew a similar to-do task from the cancelled one which is called reusability. Compared with the factors mentioned above, progress is another element we should not ignore which will affect the controlling feeling of the user, a predominating factor of the popularity and success of an application of the current world, together with this factor the priority should also be considered to make different tasks have different attentions and as a result the sorting should definitely be included to make them perform better by which the user can find out the most quickly finished tasks, the slowest tasks and the most important ones and then make specific arrangements for better efficiency. Too much can be detailed, but to make this report terse enough only the most critical are collected and presented here and due to the limitation of personal writing skills, the logics and overall framework may be not well arranged for readers to easily understand the whole application and its design, later on I will pay more attention to the whole report and make it more delicate, more readable and more understandable.

5. Work Arrangements

The following Gantt graphs are used to exemplify the arrangements in the whole project. Though lots of inconsistency happened and even till now there is still a bunch of work for me to do for further betterment, the basic situation is clearly presented in the diagrams below for April.

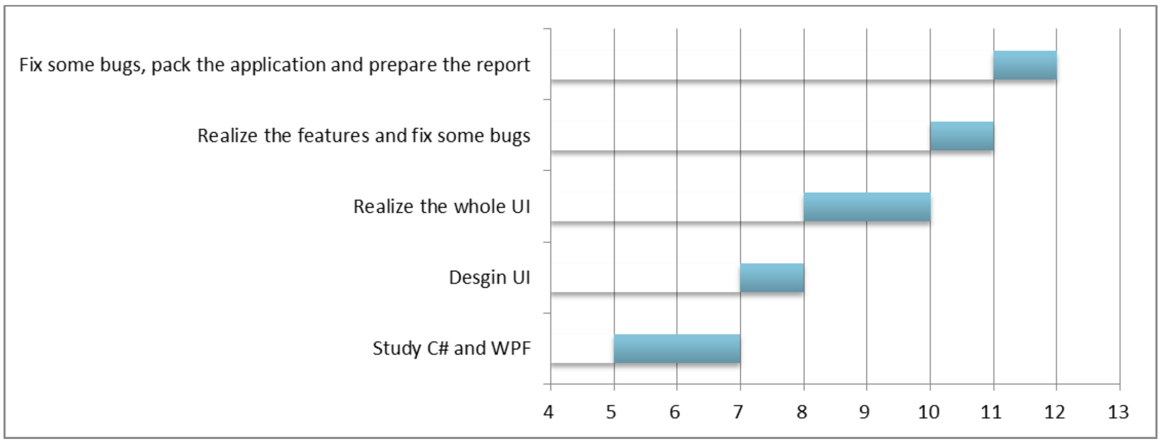


Fig. 1 Apr. Arrangements for Windows Client

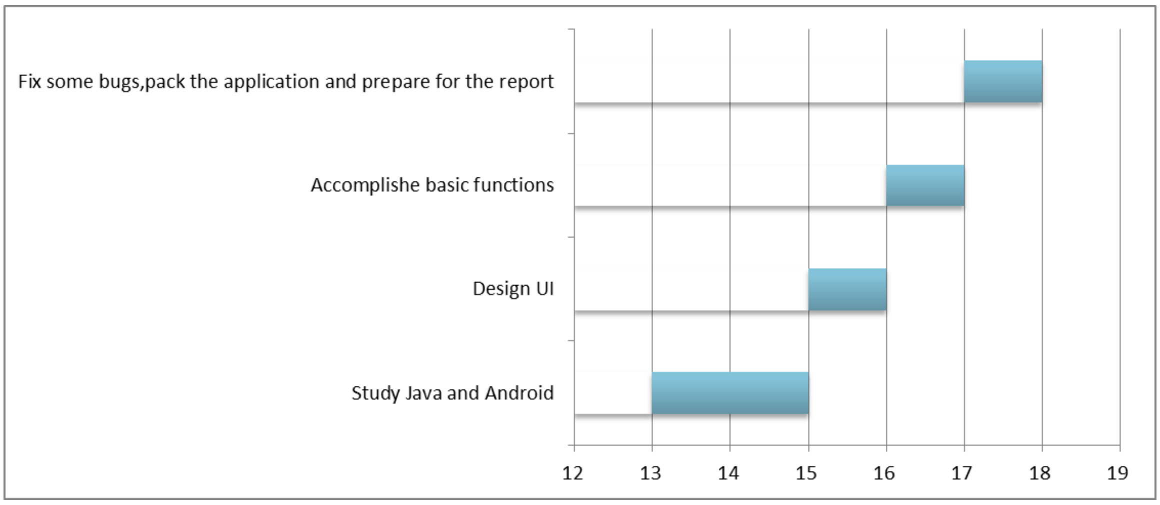


Fig. 2 Apr. Arrangements for Android client

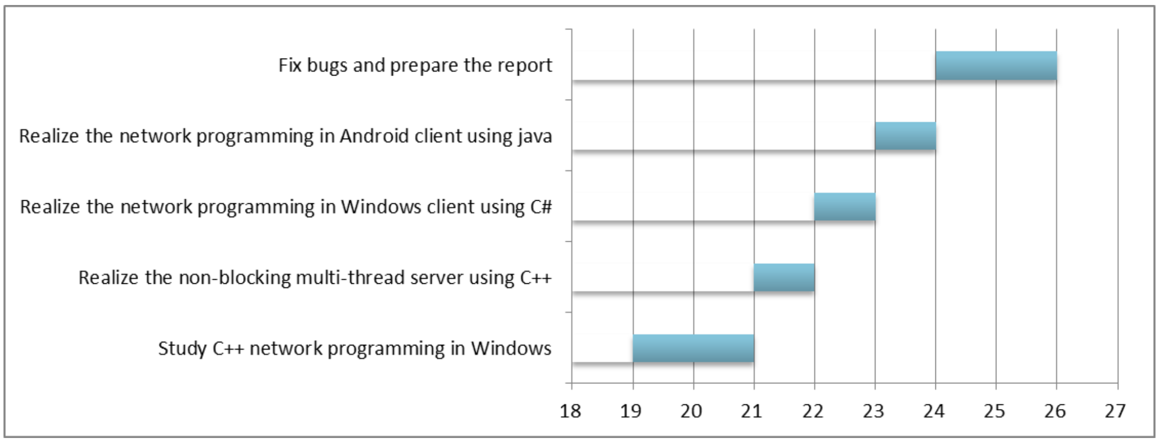


Fig. 3 Apr. Arrangements for Server and Fixing Bugs

As for the arrangements shown in these three diagrams, I have to confess that the real work is far from this type and lots of stamina have been poured into some meaningless stuff like selecting the books to refer to, worrying about the overall performance, learning bran-new skills and as a result the real time spent in realizing this system is pretty limited, about a week at most with each side two days; what's even worse, the result is rather out of expectation and some severe problems have not been solved, though which I will handle in the near future.

6. References

[1] http://en.wikipedia.org/wiki/Windows_Presentation_Foundation

[2] http://blog.csdn.net/henhen2002/article/details/4368300

[3] Pieter H.(2013), Code Connected Volume 1: Learning ZeroMQ Professional Edition for C/C++, Published by iMatix Corporation

[4] http://www.nodejs.org/

[5] Frederick P.Brookds.Jr.(1975), The Mythical Man-Month, published by Addison Wesley

[6] A.P. Ershov, Aesthetics and the human factor in programming, CACM, 15, 7(July,1972), pp.501-505

[7] Wolverton, R.W., The cost of developing large-scale software, IEEE Trans. On Computers, C-23, 6(June, 1974) PP.615-636.

[8] Sackman, H., W. J. Erikson, and E. E. Grant, Exploratory experimental studies comparing online and offline programming performance, CACM, 11, 1(Jan., 1968), pp. 3-11.

[9] Mills, H., Chief programmer teams, principles, and procedures, IBM Federal Systems Division Report FSC 715108, Gaithbersburg, Md., 1971.

[10]Baker, F. T., Chief programmer team management of production programming, IBM Sys. J. 11, 1(1972)

[11]Brooks, F. P., Architechtural philosophy, in W. Buchholz(ed.), Planning A Computer System. New York: McGraw-Hill, 1962.

[12]Blaauw, G. A., Hardware requirements for the fourth generation, in F. Gruenberger (ed.), Fourth Generation Computers, Englewood Cliffs, N. J.:Prentice-Hall, 1970.

[13]Glegg, G.L., The Design of Design. Cambridge: Cambridge Univ.

[14]C. H. Reynolds, What’s wrong with computer programming management? In G.F. Weinwurm(ed.). On the Management of Computer Programming. Philadelphia: Auerbach, 1971, pp. 35-42.

[15]Parnas, D. L., Information distribution aspects of design methodology, Carnegie-Mellon Univ., Dept. of Computer Science Technical Report, February, 1971.

[16]Information Processing 1986, the Proceedings of the IFIP Tenth World Computing Conference, edited by H. – J. Kugler (1986), pp. 1069-1076.