МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»
Навчально-науковий інститут електронних та інформаційних технологій
Кафедра інформаційних та комп’ютерних систем

Додаток А

Текст клієнт-серверної програми

Аркушів 38

Виконала:

студентка гр. КІт – 221 Киричок В.В.

Чернігів 2023

Лістинг 1 – Файл Server.h

```
#pragma once
#include "ChatServer.h"
#include "ThreadFunc.h"
#include "resource.h"
#include <windows.h>
#include <iostream>
#include <xlocale>
#include <xstring>
#include <codecvt>
#include <fstream>

HWND hTextBox;

void LoadData();
void RestoreData();
```

Лістинг 2 – Файл ChatServer.h

```
#pragma once
#define _AFXDLL
#include "Definition.h"

class ChatServer
{
public:
ChatServer();
~ChatServer();
bool isConnected();
void startListenClient();
int sendMessageClient(ClientPacket* client, WCHAR* message, int len);
int recClient(SOCKET recSocket);
void setHWND(HWND hwnd);
bool signUp(User* user);
bool logIn(SOCKET socket, User user);
bool isOnlineUser(wstring username);
bool isOnlineGroup(wstring groupname);
void sendMessageGroup(wstring groupname, wstring sender, WCHAR* message, int len, bool isSendToSender =
false);
void addUser(User* user);
list<User*>& getUser();
private:
bool _isConnected;
int _serverPort;
list<ClientPacket*> _clientList;
SOCKET _socClient;
SOCKET _socListenClient;
HWND _hwnd;
list<User*> _userData;
list<GroupChat*> _groupchatList;
};
```

Лістинг 3 – Файл Client.h

```
#pragma once
#include "ChatClient.h"
#include "ThreadFunc.h"
#include "resource.h"
#include <windows.h>
#include <windowsx.h>
#include "PrivateChatBox.h"
#include "GroupChatBox.h"
#include <list>
#include <mmsystem.h>
```

```
#pragma comment(lib, "Winmm.lib")
#pragma comment(linker,"\"/manifestdependency:type='win32' name='Microsoft.Windows.Common-Controls' version='6.0.0.0' processorArchitecture='*' publicKeyToken='6595b64144ccf1df' language='*'\"")


HWND hTextBox;
HWND hMessageBox;
HWND hSend;

HWND hSignUp;
HWND hLogIn;
HWND hUsername;
HWND hPassword;

HWND hInvitedUsername;
HWND hGroupName;
HWND hPrivateChat;
HWND hGroupChat;
HWND hCreate;
HWND hwnd;
HFONT hFont;

int gCurScene;
list<PrivateChatBox*> gPrivateChatBoxList;
list<GroupChatBox*> gGroupChatBoxList;
enum MessageType gCurMessageType;

GdiplusStartupInput gdiplusStartupInput;
ULONG_PTR gdiplusToken;

LPWSTR convertSize(DWORD size);
bool myRegClass(WNDPROC lpfnWndProc, WCHAR *szClassName, HINSTANCE hInst);
BOOL myCreateOpenFile(HWND hwnd, WCHAR* filename);
BOOL myCreateSaveFile(HWND hwnd, WCHAR* filename);

LRESULT CALLBACK ChatBoxProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);
void OnDestroy(HWND hwnd);
BOOL OnCreate(HWND hwnd, LPCREATESTRUCT lpCreateStruct);
void OnCommand(HWND hwnd, int id, HWND hwndCtl, UINT codeNotify);
void OnPaint(HWND hwnd);
void OnDrawItem(HWND hwnd, const DRAWITEMSTRUCT * lpDrawItem);
```

Лістинг 4 – Файл ChatClient.h

```
#pragma once
#define _AFXDLL
#include <Afxwin.h>
#include <stdio.h>
#include <winsock2.h>
#include <conio.h>
#include <iostream>
#include <windows.h>
#include "resource.h"
using namespace std;


class ChatClient
{
public:
ChatClient();
~ChatClient();
void init(string ipAddress, int port);
int sendMessagePort(WCHAR* message, int len);
int recMessagePort();
bool isConnected();
void setHWND(HWND hwnd);
```

```
void setUsername(wstring username);
wstring& getUsername();
private:
bool _isConnected; // true - connected false - not connected
string _serverIPAddress;
int _serverPort;
SOCKET _connect; // socket connected to server
HWND _hwnd;
wstring _username;
};
```

Лістинг 5 – Файл Server.cpp

```
#include "stdafx.h"
#include "Server.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name

// Forward declarations of functions included in this code module:
ATOM             MyRegisterClass(HINSTANCE hInstance);
BOOL             InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                                        _In_opt_ HINSTANCE hPrevInstance,
                                        _In_ LPWSTR    lpCmdLine,
                                        _In_ int       nCmdShow)
{
UNREFERENCED_PARAMETER(hPrevInstance);
UNREFERENCED_PARAMETER(lpCmdLine);

// TODO: Place code here.

// Initialize global strings
LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadStringW(hInstance, IDC_SERVER, szWindowClass, MAX_LOADSTRING);
MyRegisterClass(hInstance);

// Perform application initialization:
if (!InitInstance (hInstance, nCmdShow))
{
        return FALSE;
}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_SERVER));

MSG msg;

// Main message loop:
while (GetMessage(&msg, nullptr, 0, 0))
{
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
        }
}

return (int) msg.wParam;
}
```

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style         = CS_HREDRAW | CS_VREDRAW;
wcex.lpfnWndProc    = WndProc;
wcex.cbClsExtra     = 0;
wcex.cbWndExtra     = 0;
wcex.hInstance      = hInstance;
wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_SERVER));
wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_SERVER);
wcex.lpszClassName  = szWindowClass;
wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

return RegisterClassExW(&wcex);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  hInst = hInstance; // Store instance handle in our global variable

  HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_CAPTION | WS_SYSMENU |
WS_MINIMIZEBOX,
 CW_USEDEFAULT, 0, 500, 500, nullptr, nullptr, hInstance, nullptr);

  if (!hWnd)
  {
 return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
switch (message)
{
case WM_CREATE:
{
        char buf[4096];
        if (!gServerObj.isConnected())
        {
                MessageBox(0, L"\nFailed to initialise server socket.", 0, 0);
                getch();
                return 1;
        }
        gServerObj.setHWND(hWnd);
        AfxBeginThread(listenServerThread, 0);
        LoadData();
        hTextBox = CreateWindow(L"edit", L"", WS_VISIBLE | WS_VSCROLL | WS_CHILD |
ES_AUTOVSCROLL | ES_MULTILINE | ES_READONLY, 0, 0, 480, 500, hWnd, 0, hInst, 0);
        break;
}
case WM_COMMAND:
        {
                int wmId = LOWORD(wParam);
                // Parse the menu selections:
                switch (wmId)
```

```
                {
                case IDM_ABOUT:
                        DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                        break;
                case IDM_EXIT:
                        DestroyWindow(hWnd);
                        break;
                case ID_RESTORE:
                        RestoreData();
                        break;
                case ID_USER_CONNECT:
                {
                        WCHAR buffer[10000];
                        GetWindowText(hTextBox, buffer, 10000);
                        if (buffer == NULL)
                        {
                                break;
                        }
                        wcscat(buffer, L"\r\n");
                        wcscat(buffer, L"1 user has just connected to server");
                        SetWindowText(hTextBox, buffer);

                        SendMessageA(hTextBox, EM_SETSEL, 0, -1);
                        SendMessageA(hTextBox, EM_SETSEL, -1, -1);
                        SendMessageA(hTextBox, EM_SCROLLCARET, 0, 0);

                        break;
                }
                case ID_USER_LEAVE:
                {
                        WCHAR buffer[10000];
                        GetWindowText(hTextBox, buffer, 10000);
                        if (buffer == NULL)
                        {
                                break;
                        }
                        wcscat(buffer, L"\r\n");
                        wcscat(buffer, L"1 user has just disconnected to server");
                        SetWindowText(hTextBox, buffer);

                        SendMessageA(hTextBox, EM_SETSEL, 0, -1);
                        SendMessageA(hTextBox, EM_SETSEL, -1, -1);
                        SendMessageA(hTextBox, EM_SCROLLCARET, 0, 0);

                        break;
                }
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
                }
        }
        break;
case WM_PAINT:
        {
                PAINTSTRUCT ps;
                HDC hdc = BeginPaint(hWnd, &ps);
                // TODO: Add any drawing code that uses hdc here...
                EndPaint(hWnd, &ps);
        }
        break;
case WM_DESTROY:
        RestoreData();
        PostQuitMessage(0);
        break;
default:
        return DefWindowProc(hWnd, message, wParam, lParam);
}
return 0;
```

```
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
UNREFERENCED_PARAMETER(lParam);
switch (message)
{
case WM_INITDIALOG:
	return (INT_PTR)TRUE;

case WM_COMMAND:
	if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
	{
		EndDialog(hDlg, LOWORD(wParam));
		return (INT_PTR)TRUE;
	}
	break;
}
return (INT_PTR)FALSE;
}


void RestoreData()
{
std::wofstream fos(L"userdata.ini");
std::locale loc(std::locale(), new std::codecvt_utf8<wchar_t>);
fos.imbue(loc);
if (!fos.is_open())
	return;
for (auto user : gServerObj.getUser())
{
	fos << user->username << L";";
	fos << user->password << L"\n";
}
fos.close();
}

void LoadData()
{
std::locale loc(std::locale(), new std::codecvt_utf8<wchar_t>);
std::wifstream fin(L"userdata.ini");
fin.imbue(loc);
WCHAR buffer[200];
WCHAR username[100];
WCHAR password[100];

int i = 0;

while (!fin.eof())
{
	fin.getline(buffer, 200);
	if (buffer[0] == NULL)
	{
		break;
	}
	int len = wcslen(buffer);
	int j, k;
	for (j = 0; j < len; j++)
	{
		if (buffer[j] == L';')
		{
			j++;
			break;
		}
		username[j] = buffer[j];
	}
```

```
		username[j - 1] = NULL;

		for (k = j; k < len; k++)
		{
			password[k - j] = buffer[k];
		}
		password[k - j] = NULL;
		auto user = new User;
		user->username = username;
		user->password = password;
		gServerObj.addUser(user);
}

fin.close();
}
```

Лістинг 6 – Файл ChatServer.cpp

```
#include "stdafx.h"
#include "ChatServer.h"
#include "ThreadFunc.h"
#include "Resource.h"
ChatServer::ChatServer()
{
cout << "Starting up TCP Chat server\n";
_isConnected = false;

WSADATA wsaData;

sockaddr_in local;

int wsaret = WSAStartup(0x101, &wsaData);

if (wsaret != 0)
{
	return;
}

local.sin_family = AF_INET;
local.sin_addr.s_addr = INADDR_ANY;
local.sin_port = htons((u_short)8084);

_socListenClient = socket(AF_INET, SOCK_STREAM, 0);


if (_socListenClient == INVALID_SOCKET)
{
	return;
}


if (bind(_socListenClient, (sockaddr*)&local, sizeof(local)) != 0)
{
	return;
}


if (listen(_socListenClient, 10) != 0)
{
	return;
}

_isConnected = true;
return;
}
```

```
ChatServer::~ChatServer()
{
closesocket(_socListenClient);

WSACleanup();

if (_clientList.size() != 0)
{
	for (auto client: _clientList)
	{
		delete client;
	}
}

if (_groupchatList.size() != 0)
{
	for (auto group : _groupchatList)
	{
		delete group;
	}
}
}

bool ChatServer::isConnected()
{
return _isConnected;
}

void ChatServer::startListenClient()
{

sockaddr_in from;
int fromlen = sizeof(from);

_socClient = accept(_socListenClient,
	(struct sockaddr*)&from, &fromlen);
auto packet = new ClientPacket;
packet->socket = _socClient;

if (_socClient != INVALID_SOCKET)
{
	SendMessage(_hwnd, WM_COMMAND, ID_USER_CONNECT, 0);
	_clientList.push_back(packet);

}

AfxBeginThread(recServerThread, (void *)_socClient);

}



int ChatServer::sendMessageClient(ClientPacket* client, WCHAR* message, int len)
{
int iStat = 0;

iStat = send(client->socket, (char*)message, len * 2 + 2, 0);
if (iStat == -1)
	_clientList.remove(client);
if (iStat == -1)
	return 1;

return 0;

}

int ChatServer::recClient(SOCKET recSocket)
```

```
{
WCHAR* message;
WCHAR temp[4096];
int iStat;
int len;
iStat = recv(recSocket, (char*)temp, 4096, 0);
list<ClientPacket*>::iterator itl;
for (itl = _clientList.begin(); itl != _clientList.end(); itl++)
{
	if ((*itl)->socket == recSocket)
	{
		break;
	}
}

if (iStat == -1)
{
	for (auto gc : _groupchatList)
	{
		gc->username.remove((*itl)->username);
	}
	_clientList.remove((*itl));
	SendMessage(_hwnd, WM_COMMAND, ID_USER_LEAVE, 0);
	return 1;
}
else
{
	message = temp;
	switch (message[0])
	{
	case MessageType::PRIVATE_CHAT:
	{
		/*
		* receive:	message = [FLAG | receiver | NULL | sender | NULL | content | NULL]
		* receive:	message = [FLAG | receiver | NULL | sender | NULL | content | NULL]
		*/

		WCHAR* partner;
		partner = message + 1;


		for (auto client : _clientList)
		{
			if (wcscmp(client->username.c_str(), partner) == 0)
			{
				sendMessageClient(client, (WCHAR*)message, iStat / 2);
				break;
			}
		}
		break;
	}
	case MessageType::GROUP_CHAT:
	{
		/*
		 * receive: message = [FLAG | group name | NULL | sender | NULL | content | NULL]
		 * send:  message = [FLAG | group name | NULL | sender | NULL | content | NULL]
		 */
		WCHAR* groupname;
		len = wcslen(message);
		groupname = message + 1;

		sendMessageGroup(groupname, (*itl)->username, message, iStat / 2);
		break;
	}
	case MessageType::END_PRIVATE_CHAT:
	{
		/*
```

```
                * receive:          message = [FLAG | receiver | NULL | sender | NULL]
                * send:             message = [FLAG | receiver | NULL | sender | NULL]
                */
                WCHAR* receiver = message + 1;

                for (auto client : _clientList)
                {
                        if (wcscmp(client->username.c_str(), receiver) == 0)
                        {
                                sendMessageClient(client, (WCHAR*)message, iStat / 2);
                                break;
                        }
                }
                break;
        }
        case MessageType::END_GROUP_CHAT:
        {
                /*
                * receive:          message = [FLAG | group name | NULL | sender | NULL]
                * send:             message = [FLAG | group name | NULL | sender | NULL]
                */

                WCHAR* groupname;
                GroupChat* gc = NULL;

                groupname = message + 1;
                for (auto group : _groupchatList)
                {

                        if (wcscmp(group->name.c_str(), groupname) == 0)
                        {
                                for (auto user : group->username)
                                {

                                        if (user != (*itl)->username)
                                        {

                                                for (auto client : _clientList)
                                                {

                                                        if (client->username == user)
                                                        {

                                                                sendMessageClient(client, message, iStat
/ 2);
                                                                break;
                                                        }
                                                }
                                        }
                                }
                                gc = group;
                                break;
                        }
                }
                if (!gc)
                {
                        break;
                }
                if (gc->username.size() > 0)
                {
                        gc->username.remove((*itl)->username);
                }

                if (gc->username.size() == 0)
                {
                        if (_groupchatList.size() > 0)
                        {
```

```
                        _groupchatList.remove(gc);
                    }
                }
                break;
            }
            case MessageType::SIGNUP:
            {
                /*
                * receive:      message = [FLAG | user name | NULL | password | NULL]
                * send:         message = [FLAG]
                */

                WCHAR* username;
                WCHAR* password;

                username = message + 1;
                password = message + wcslen(message) + 1;

                auto user = new User;
                user->username = username;
                user->password = password;
                int result = signUp(user);
                if (result == true)
                {
                    message[0] = MessageType::SU_SUCCESS;
                    SendMessage(_hwnd, WM_COMMAND, ID_RESTORE, 0);
                }
                else
                {
                    message[0] = MessageType::SU_FAILURE;
                }
                sendMessageClient((*itl), (WCHAR*)message, 1);
                break;
            }
            case MessageType::LOGIN:
            {
                /*
                * receive:      message = [FLAG | user name | NULL | password | NULL]
                * send:         message = [FLAG]
                */

                WCHAR* username;
                WCHAR* password;
                username = message + 1;
                password = message + wcslen(message) + 1;
                User user;
                user.username = username;
                user.password = password;
                int result = logIn(recSocket, user);
                if (result == true)
                {
                    message[0] = MessageType::LI_SUCCESS;
                }
                else
                {
                    message[0] = MessageType::LI_FAILURE;
                }

                sendMessageClient((*itl), (WCHAR*)message, 1);
                break;
            }
            case MessageType::CREATE_PRIVATE_CHAT:
            {
                /*
                * receive:      message = [FLAG | partner | NULL]
                * send:         message = [FLAG | parter | NULL] - Succeed
                *                                                 [FLAG]
```

```
- Fail
                */

                WCHAR* partner;
                partner = message + 1;
                bool result = isOnlineUser(partner);
                if (result == true)
                {

                        message[0] = MessageType::C_PC_SUCCESS;
                        sendMessageClient((*itl), (WCHAR*)message, iStat / 2);
                }
                else
                {
                        message[0] = MessageType::C_PC_FAILURE;
                        sendMessageClient((*itl), (WCHAR*)message, 1);
                }
                break;
        }
        case MessageType::CREATE_GROUP_CHAT:
        {
                /*
                * receive:      message = [FLAG | group name | NULL]
                * send:         message = [FLAG | group name | NULL]     - Succeed
                *                                                       [FLAG]
- Fail
                */

                WCHAR buffer[150];
                buffer[0] = C_GC_SUCCESS;
                buffer[1] = NULL;
                wcscat(buffer, (*itl)->username.c_str());
                wcscat(buffer, L";");
                wcscat(buffer, message + 1);
                bool result = isOnlineGroup(buffer + 1);
                if (result == false)
                {
                        auto gc = new (std::nothrow) GroupChat;
                        gc->name = buffer + 1;
                        gc->username.push_back((*itl)->username);
                        _groupchatList.push_back(gc);
                        len = wcslen(buffer);
                        sendMessageClient((*itl), (WCHAR*)buffer, len);
                }
                else
                {
                        buffer[0] = MessageType::C_GC_FAILURE;
                        sendMessageClient((*itl), (WCHAR*)buffer, 1);
                }
                break;
        }
        case MessageType::GC_ADD_USER:
        {
                /*
                * receive:      message = [FLAG | group name | NULL | added user | NULL]
                * send:         message = [FLAG | group name | NULL | added user | NULL]
                */
                WCHAR* username;
                WCHAR* groupname;
                len = wcslen(message);

                groupname = message + 1;
                username = message + len + 1;

                bool result = isOnlineUser(username);
                if (result == true)
                {
```

```
                        for (auto group : _groupchatList)
                        {
                                if (wcscmp(group->name.c_str(), groupname) == 0)
                                {
                                        for (auto user : group->username)
                                        {
                                                if (wcscmp(user.c_str(), username) == 0)
                                                {
                                                        message[0] = MessageType::GC_AU_FAILURE;
                                                        len += group->username.size() + 1;
                                                        sendMessageClient((*itl), (WCHAR*)message,
len);

                                                        return 0;
                                                }
                                        }
                                        group->username.push_back(username);

                                        message[0] = MessageType::GC_AU_SUCCESS;

                                        len += wcslen(username) + 1;

                                        sendMessageGroup(groupname, L"user", message, len, true);
                                        //SendMessageClient((*itl), (WCHAR*)buffer, len);
                                        return 0;
                                }
                        }
                }
                else
                {
                        message[0] = MessageType::GC_AU_FAILURE;
                        sendMessageClient((*itl), (WCHAR*)message, len);
                }
                break;
        }
        case MessageType::SF_ACCEPT:
        case MessageType::SF_CANCEL:
        {
                /*
                * receive:      message = [FLAG | file name | NULL | file size | NULL | sender | NULL|
receiver | NULL]
                * send:         message = [FLAG | file name | NULL | file size | NULL | sender | NULL|
receiver | NULL]
                */
                WCHAR* partner;
                partner = message + wcslen(message) + 4;
                partner += wcslen(partner) + 1;
                //buffer[len] = NULL;
                for (auto client : _clientList)
                {
                        if (wcscmp(client->username.c_str(), partner) == 0)
                        {
                                sendMessageClient(client, (WCHAR*)message, iStat / 2);
                                break;
                        }
                }
                break;
        }
        case MessageType::SEND_FILE:
        {
                /*
                * receive:      message = [FLAG | file name | NULL | file size | NULL | receiver | NULL|
sender | NULL]
                * send:         message = [FLAG | file name | NULL | file size | NULL | receiver | NULL|
sender | NULL]
                */
                WCHAR* partner;
                partner = message + wcslen(message) + 4;
```

```
                        //buffer[len] = NULL;
                        for (auto client : _clientList)
                        {
                                if (wcscmp(client->username.c_str(), partner) == 0)
                                {
                                        sendMessageClient(client, (WCHAR*)message, iStat / 2);
                                        break;
                                }
                        }
                        break;
                }
                case MessageType::FILE_DATA:
                {
                        /*
                        * receive:      message = [FLAG | file size | NULL | receiver | NULL | sender | NULL |
content]
                        * send:         message = [FLAG | file size | NULL | receiver | NULL | sender | NULL |
content]
                        */

                        WCHAR* receiver;
                        receiver = message + 3;

                        //buffer[len] = NULL;
                        for (auto client : _clientList)
                        {
                                if (wcscmp(client->username.c_str(), receiver) == 0)
                                {
                                        sendMessageClient(client, (WCHAR*)message, iStat / 2);
                                        break;
                                }
                        }
                        break;
                }
                case MessageType::STOP:
                case MessageType::CONTINUE:
                {
                        /*
                        * receive:      message = [FLAG | receiver | NULL | sender | NULL]
                        * send:         message = [FLAG | receiver | NULL | sender | NULL]
                        */
                        WCHAR* receiver = message + 1;
                        for (auto client : _clientList)
                        {
                                if (wcscmp(client->username.c_str(), receiver) == 0)
                                {
                                        sendMessageClient(client, (WCHAR*)message, iStat / 2);
                                        break;
                                }
                        }

                        break;
                }
                }

                return 0;
        }
        return 0;

        }

        void ChatServer::setHWND(HWND hwnd)
        {
        _hwnd = hwnd;
        }
```

```
bool ChatServer::signUp(User* user)
{
for (auto userdata : _userData)
{
	if (user->username == userdata->username)
	{
		return false;
	}
}
_userData.push_back(user);
return true;
}

bool ChatServer::logIn(SOCKET socket, User user)
{
for (auto userdata : _userData)
{
	if (user.username == userdata->username && user.password == userdata->password)
	{
		for (auto client : _clientList)
		{
			if (client->socket == socket)
			{
				client->username = user.username;
				return true;
			}
		}
	}
}
return false;
}

bool ChatServer::isOnlineUser(wstring username)
{
for (auto client : _clientList)
{
	if (username == client->username)
	{
		return true;
	}
}
return false;
}

bool ChatServer::isOnlineGroup(wstring groupname)
{
for (auto group : _groupchatList)
{
	if (group->name == groupname)
	{
		return true;
	}
}
return false;
}

void ChatServer::sendMessageGroup(wstring groupname, wstring sender, WCHAR* message, int len, bool
isSendToSender)
{
for (auto group : _groupchatList)
{

	if (group->name == groupname)
	{

		for (auto user : group->username)
		{
```

```
                        if (user != sender || isSendToSender)
                        {

                                for (auto client : _clientList)
                                {

                                        if (client->username == user)
                                        {

                                                sendMessageClient(client, (WCHAR*)message, len);
                                                break;
                                        }
                                }

                        }
                }

                return;
        }
}
}

void ChatServer::addUser(User* user)
{
_userData.push_back(user);
}

list<User*>& ChatServer::getUser()
{
return _userData;
}
```

Лістинг 7 – Файл Client.cpp

```
#include "stdafx.h"
#include "Client.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                        // current instance
WCHAR szTitle[MAX_LOADSTRING];          // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];    // the main window class name

// Forward declarations of functions included in this code module:
ATOM            MyRegisterClass(HINSTANCE hInstance);
BOOL            InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                                        _In_opt_ HINSTANCE hPrevInstance,
                                        _In_ LPWSTR    lpCmdLine,
                                        _In_ int       nCmdShow)
{
UNREFERENCED_PARAMETER(hPrevInstance);
UNREFERENCED_PARAMETER(lpCmdLine);

// TODO: Place code here.

// Initialize global strings
LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
LoadStringW(hInstance, IDC_CLIENT, szWindowClass, MAX_LOADSTRING);
   // MyRegisterClass(hInstance);
if (myRegClass(WndProc, szWindowClass, hInst) == false)
```

```
		return 0;

	if (myRegClass(ChatBoxProc, L"chatbox", hInst) == false)
		return 0;
	// Perform application initialization:
	if (!InitInstance (hInstance, nCmdShow))
	{
		return FALSE;
	}

	HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_CLIENT));

	MSG msg;

	// Main message loop:
	while (GetMessage(&msg, nullptr, 0, 0))
	{
		if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return (int) msg.wParam;
	}

	bool myRegClass(WNDPROC lpfnWndProc, WCHAR *szClassName, HINSTANCE hInst)
	{
	WNDCLASSEX wincl;        /* Data structure for the windowclass */

                                                        /* The Window structure */
	wincl.hInstance = hInst;
	wincl.lpszClassName = szClassName;
	wincl.lpfnWndProc = lpfnWndProc;      /* This function is called by windows */
	wincl.style = CS_HREDRAW | CS_VREDRAW;                /* Catch double-clicks */
	wincl.cbSize = sizeof(WNDCLASSEX);

	/* Use default icon and mouse-pointer */
	wincl.hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_CLIENT));
	wincl.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
	wincl.hCursor = LoadCursor(NULL, IDC_ARROW);
	wincl.hbrBackground = (HBRUSH)(COLOR_BTNHILIGHT + 1);
	wincl.lpszMenuName = NULL;                /* No menu */
	wincl.cbClsExtra = 0;                    /* No extra bytes after the window class */
	wincl.cbWndExtra = 0;                      /* structure or the window instance */
                                                                        /* Use Windows's
default colour as the background of the window */
	wincl.hIconSm = LoadIcon(wincl.hInstance, MAKEINTRESOURCE(IDI_SMALL));

	/* Register the window class, and if it fails quit the program */
	if (RegisterClassEx(&wincl))
		return true;
	else
		return false;
	}

	ATOM MyRegisterClass(HINSTANCE hInstance)
	{
	WNDCLASSEXW wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style         = CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc    = WndProc;
	wcex.cbClsExtra     = 0;
	wcex.cbWndExtra     = 0;
```

```
	wcex.hInstance      = hInstance;
	wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_CLIENT));
	wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
	wcex.hbrBackground  = (HBRUSH)(COLOR_BTNFACE+1);
	wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_CLIENT);
	wcex.lpszClassName  = szWindowClass;
	wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

	return RegisterClassExW(&wcex);
	}

	BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
	{
	  hInst = hInstance; // Store instance handle in our global variable

	  HWND hWnd = CreateWindowW(szWindowClass, L"Chat Client", WS_CAPTION | WS_SYSMENU |
WS_MINIMIZEBOX,
	   CW_USEDEFAULT, 0, 400, 600, nullptr, nullptr, hInstance, nullptr);

	  if (!hWnd)
	  {
	  return FALSE;
	  }

	  ShowWindow(hWnd, nCmdShow);
	  UpdateWindow(hWnd);

	  return TRUE;
	}

	LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
	{
	switch (message)
	{
		HANDLE_MSG(hWnd, WM_CREATE, OnCreate);
		HANDLE_MSG(hWnd, WM_COMMAND, OnCommand);
		HANDLE_MSG(hWnd, WM_DESTROY, OnDestroy);
		HANDLE_MSG(hWnd, WM_PAINT, OnPaint);
		HANDLE_MSG(hWnd, WM_DRAWITEM, OnDrawItem);
		case WM_CTLCOLORSTATIC:
		{
			HWND hStatic = (HWND)lParam;
			HDC hdc = (HDC)wParam;

			//SetTextColor(hdc, RGB(0, 215, 194));
			SetBkMode(hdc, TRANSPARENT);
			//return (LRESULT)CreateSolidBrush(RGB(255, 255, 255));
			return (LRESULT)GetStockObject(DC_BRUSH);
		}
	}
	return DefWindowProc(hWnd, message, wParam, lParam);
	}

	void OnDestroy(HWND hwnd)
	{
	GdiplusShutdown(gdiplusToken);
	PostQuitMessage(0);
	}

	BOOL OnCreate(HWND hWnd, LPCREATESTRUCT lpCreateStruct)
	{
	INITCOMMONCONTROLSEX icex;

	icex.dwSize = sizeof(icex);
	icex.dwICC = ICC_DATE_CLASSES;

	InitCommonControlsEx(&icex);
```

```
	GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

	char buf[4096];
	gCurScene = 0;

	FILE *fp = fopen("server.ini", "r");
	if (fp == NULL)
	{
		MessageBox(0, L"\nUnable to open server.ini. Please specify server IPsddress in server.ini", 0, 0);
		return 1;
	}
	string sServerAddress;
	while ((fgets(buf, 4096, fp)) != NULL)
	{
		if (buf[0] == '#')
			continue;
		sServerAddress = buf;

	}
	fclose(fp);

	if (sServerAddress.size() == 0)
	{
		MessageBox(hWnd, L"\nUnable to connect to the IP address specified in server.ini\r\nPlease check
server IP address.", 0, 0);
		return 0;
	}

	gClientObj.init(sServerAddress.c_str(), 8084);
	if (!gClientObj.isConnected())
	{
		MessageBox(hWnd, L"\nUnable to connect to the IP address specified in server.ini\r\nPlease check
server IP address.", 0, 0);
		return 0;
	}
	gClientObj.setHWND(hWnd);
	AfxBeginThread(recMessageThread, 0);


	hFont = CreateFont(17, 0, 0, 0, FW_DONTCARE, FALSE, FALSE, FALSE, ANSI_CHARSET,
		OUT_TT_PRECIS, CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
		DEFAULT_PITCH | FF_DONTCARE, L"Arial");

	if (hFont == NULL)
	{
		LOGFONT lf;
		GetObject(GetStockObject(DEFAULT_GUI_FONT), sizeof(LOGFONT), &lf);
		CreateFont(lf.lfHeight*1.6, lf.lfWidth*1.6,
			lf.lfEscapement, lf.lfOrientation, lf.lfWeight,
			lf.lfItalic, lf.lfUnderline, lf.lfStrikeOut, lf.lfCharSet,
			lf.lfOutPrecision, lf.lfClipPrecision, lf.lfQuality,
			lf.lfPitchAndFamily, lf.lfFaceName);
	}
	SetWindowFont(hwnd, hFont, true);
	hUsername = CreateWindowEx(0, L"edit", L"", WS_VISIBLE | WS_CHILD, 50, 160, 280, 35, hWnd, 0, hInst,
0);
	SetWindowFont(hUsername, hFont, true);
	hPassword = CreateWindowEx(0, L"edit", L"", WS_VISIBLE | WS_CHILD | ES_PASSWORD, 50, 250, 280,
35, hWnd, (HMENU)99, hInst, 0);
	SetWindowFont(hPassword, hFont, true);
	hSignUp = CreateWindowEx(0, L"button", L"Sign Up", WS_VISIBLE | WS_CHILD | BS_OWNERDRAW, 50,
320, 135, 30, hWnd, (HMENU)IDC_SIGNUP, hInst, 0);
	hLogIn = CreateWindowEx(0, L"button", L"Log In", WS_VISIBLE | WS_CHILD | BS_OWNERDRAW, 195,
320, 135, 30, hWnd, (HMENU)IDC_LOGIN, hInst, 0);
	}

	void OnCommand(HWND hWnd, int id, HWND hwndCtl, UINT codeNotify)
```

```
	{
	switch (id)
	{
	case IDM_ABOUT:
		DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
		break;
	case IDM_EXIT:
		DestroyWindow(hWnd);
		break;
	case IDC_RECEIVE:
	{
		WCHAR* message = (WCHAR*)hwndCtl;
		switch (message[0])
		{
		case MessageType::PRIVATE_CHAT:
		{
			/*
			* receive:		message = [FLAG | receiver | NULL | sender | NULL | content | NULL]
			*/
			PlaySound(MAKEINTRESOURCE(IDR_WAVE1), NULL, SND_RESOURCE |
SND_ASYNC);
			message++;
			int	len = wcslen(message);
			WCHAR* partner;
			WCHAR* content;

			partner = message + len + 1;
			content = partner + wcslen(partner) + 1;

			for (auto chatbox : gPrivateChatBoxList)
			{
				if (wcscmp(chatbox->getPartner().c_str(), partner) == 0)
				{
					chatbox->receiveMessage(content);
					SetForegroundWindow(chatbox->getHWND());
					SetActiveWindow(chatbox->getHWND());
					return;
				}
			}
			auto cb = PrivateChatBox::create(hWnd, hInst, Point(CW_USEDEFAULT,
CW_USEDEFAULT), Size(500, 500), partner);
			cb->setFont(hFont);
			cb->setUsername(gClientObj.getUsername());
			gPrivateChatBoxList.push_back(cb);
			cb->receiveMessage(content);
			SetForegroundWindow(cb->getHWND());
			SetActiveWindow(cb->getHWND());
			break;
		}
		case MessageType::GROUP_CHAT:
		{
			/*
			* receive:		message = message = [FLAG | group name | NULL | sender | NULL | content
| NULL]
			*/
			PlaySound(MAKEINTRESOURCE(IDR_WAVE1), NULL, SND_RESOURCE |
SND_ASYNC);
			message++;
			int	len = wcslen(message);
			WCHAR buffer[10000];
			WCHAR* groupname;
			WCHAR* sendername;
			WCHAR* content;
			groupname = message;
			sendername = message + len + 1;
			content = sendername + wcslen(sendername) + 1;
```

```
                wcscpy(buffer, L"$[");
                wcscat(buffer, sendername);
                wcscat(buffer, L"]:\r\n");
                wcscat(buffer, content);
                for (auto chatbox : gGroupChatBoxList)
                {
                        if (wcscmp(chatbox->getGroupName().c_str(), groupname) == 0)
                        {
                                chatbox->receiveMessage(buffer);
                                SetForegroundWindow(chatbox->getHWND());
                                SetActiveWindow(chatbox->getHWND());
                                return;
                        }
                }
                auto cb = GroupChatBox::create(hWnd, hInst, Point(CW_USEDEFAULT,
CW_USEDEFAULT), Size(500, 500), groupname);
                cb->setFont(hFont);
                gGroupChatBoxList.push_back(cb);
                cb->receiveMessage(buffer);
                cb->setUsername(gClientObj.getUsername());
                SetForegroundWindow(cb->getHWND());
                SetActiveWindow(cb->getHWND());
                break;
        }
        case MessageType::END_PRIVATE_CHAT:
        {
                /*
                * receive:          message = [FLAG | receiver | NULL | sender | NULL]
                */
                message++;
                WCHAR* sender;
                sender = message + wcslen(message) + 1;

                for (auto chatbox : gPrivateChatBoxList)
                {
                        if (wcscmp(chatbox->getPartner().c_str(), sender) == 0)
                        {
                                chatbox->onEndChat();
                                break;
                        }
                }
                break;
        }
        case MessageType::END_GROUP_CHAT:
        {
                /*
                * receive:          message = [FLAG | group name | NULL | sender | NULL]
                */
                message++;
                int       len = wcslen(message);
                WCHAR* sender;
                sender = message + len + 1;
                for (auto chatbox : gGroupChatBoxList)
                {
                        if (wcscmp(chatbox->getGroupName().c_str(), message) == 0)
                        {
                                chatbox->onUserLeft(sender);
                                break;
                        }
                }
                break;
        }
        case MessageType::SU_SUCCESS:
        {
                /*
                * receive:          message = [FLAG]
                */
```

```
                MessageBox(hWnd, L"Sign Up Successfull!", L"Congratulations", 0);
                break;
            }
            case MessageType::SU_FAILURE:
            {
                /*
                * receive:          message = [FLAG]
                */
                MessageBox(hWnd, L"Sign Up Failed", 0, 0);
                break;
            }
            case MessageType::LI_SUCCESS:
            {
                /*
                * receive:          message = [FLAG]
                */
                MessageBox(hWnd, L"Log In Successful!", L"Congratulations", 0);
                DestroyWindow(hUsername);
                DestroyWindow(hPassword);
                DestroyWindow(hSignUp);
                DestroyWindow(hLogIn);
                gCurScene = 1;
                InvalidateRect(hWnd, 0, TRUE);

                hInvitedUsername = CreateWindowEx(0, L"edit", L"", WS_VISIBLE | WS_CHILD, 50, 190,
280, 35, hWnd, 0, hInst, 0);
                SetWindowFont(hInvitedUsername, hFont, true);
                hPrivateChat = CreateWindowEx(0, L"button", L"Private Chat", WS_VISIBLE | WS_CHILD |
BS_AUTORADIOBUTTON, 50, 120, 280, 35, hWnd, (HMENU)IDC_PRIVATECHAT, hInst, 0);
                SetWindowFont(hPrivateChat, hFont, true);
                hGroupName = CreateWindowEx(0, L"edit", L"", WS_VISIBLE | WS_CHILD, 50, 330, 280,
35, hWnd, 0, hInst, 0);
                SetWindowFont(hGroupName, hFont, true);
                hGroupChat = CreateWindowEx(0, L"button", L"Group Chat", WS_VISIBLE | WS_CHILD |
BS_AUTORADIOBUTTON, 50, 260, 280, 35, hWnd, (HMENU)IDC_GROUPCHAT, hInst, 0);
                SetWindowFont(hGroupChat, hFont, true);
                hCreate = CreateWindowEx(0, L"button", L"Create", WS_VISIBLE | WS_CHILD |
BS_OWNERDRAW, 50, 400, 280, 35, hWnd, (HMENU)IDC_CREATE, hInst, 0);
                SendMessage(hPrivateChat, BM_SETCHECK, BST_CHECKED, 0);
                break;
            }
            case MessageType::LI_FAILURE:
            {
                /*
                * receive:          message = [FLAG]
                */
                MessageBox(hWnd, L"Username or password is incorrect.", 0, 0);
                break;
            }
            case MessageType::C_PC_FAILURE:
            {
                /*
                * receive:          message = [FLAG]
                */
                MessageBox(hWnd, L"The user is offline.", L"Create private chat", 0);
                break;
            }
            case MessageType::C_PC_SUCCESS:
            {
                /*
                * receive:          message = [FLAG | partner | NULL]
                */
                message++;
                int         len = wcslen(message);
                MessageBox(hWnd, L"The user is online.", L"Create private chat", 0);
                for (auto chatbox : gPrivateChatBoxList)
                {
```

```
                        if (wcscmp(chatbox->getPartner().c_str(), message) == 0)
                        {
                                return;
                        }
                }
                auto cb = PrivateChatBox::create(hWnd, hInst, Point(100, 100), Size(500, 500), message);
                cb->setFont(hFont);
                cb->setUsername(gClientObj.getUsername());
                gPrivateChatBoxList.push_back(cb);
                break;
        }
        case MessageType::C_GC_FAILURE:
        {
                /*
                * receive:      message = [FLAG]
                */
                MessageBox(hWnd, L"You created a group chat by this name. Please use another group's
name!", L"Create group chat", 0);
                break;
        }
        case MessageType::C_GC_SUCCESS:
        {
                /*
                * receive:      message = [FLAG | group name | NULL]
                */
                message++;
                int     len = wcslen(message);
                MessageBox(0, L"The group chat was created.", L"Create group chat", 0);
                for (auto chatbox : gGroupChatBoxList)
                {
                        if (wcscmp(chatbox->getGroupName().c_str(), message) == 0)
                        {
                                return;
                        }
                }
                auto cb = GroupChatBox::create(hWnd, hInst, Point(100, 100), Size(500, 500), message);
                cb->setFont(hFont);
                cb->setUsername(gClientObj.getUsername());
                gGroupChatBoxList.push_back(cb);
                break;
        }
        case MessageType::GC_AU_FAILURE:
        {
                /*
                * receive:      message = [FLAG | group name | NULL]
                */
                MessageBox(hWnd, L"The user was offline or has been added to the group.", L"Add user", 0);
                break;
        }
        case MessageType::GC_AU_SUCCESS:
        {
                /*
                * receive:      message = [FLAG | group name | NULL | added user | NULL]
                */
                message++;
                int     len = wcslen(message);
                WCHAR* username;
                username = message + len + 1;
                for (auto group : gGroupChatBoxList)
                {
                        if (wcscmp(group->getGroupName().c_str(), message) == 0)
                        {
                                group->onUserJoin(username);
                        }
                }
                //MessageBox(hWnd, L"Adding user to the group successfully.", 0, 0);
                break;
```

```
		}
		case MessageType::SEND_FILE:
		{
			/*
			* receive:		message = [FLAG | file name | NULL | file size | NULL | receiver | NULL | sender | NULL]
			* send:		message = [FLAG | file name | NULL | file size | NULL | receiver | NULL | sender | NULL]
			*/
			message++;
			int	len = wcslen(message);
			WCHAR* partner;
			DWORD size;
			len++;
			size = *(message + len) << 16;
			len++;
			size += *(message + len);
			len += 2;
			partner = message + len;
			int len2 = wcslen(partner);
			len += len2 + 1;
			partner += len2 + 1;
			PrivateChatBox* prChat = NULL;
			for (auto chatbox : gPrivateChatBoxList)
			{
				if (wcscmp(chatbox->getPartner().c_str(), partner) == 0)
				{
					prChat = chatbox;
					break;
				}
			}

			if (!prChat)
			{
				prChat = PrivateChatBox::create(hWnd, hInst, Point(CW_USEDEFAULT, CW_USEDEFAULT), Size(500, 500), partner);
				prChat->setFont(hFont);
				prChat->setUsername(gClientObj.getUsername());
				gPrivateChatBoxList.push_back(prChat);
			}
			WCHAR text[1000];
			WCHAR* pSize = convertSize(size);
			wsprintf(text, L"%lS send to you a file:\nFile name: %lS\nSize: %lS\nDo you want to receive it?", partner, message, pSize);
			int result = MessageBox(prChat->getHWND(), text, L"File", MB_YESNO);

			message--;
			len += prChat->getPartner().size() + 1;
			if (result == IDYES)
			{
				message[0] = SF_ACCEPT;
			}
			else
			{
				message[0] = SF_CANCEL;
				gClientObj.sendMessagePort(message, len);
				break;
			}


			WCHAR buffer[1000];
			wcscpy(buffer, message + 1);
			bool result2 = myCreateSaveFile(prChat->getHWND(), buffer);
			if (!result2)
			{
				message[0] = SF_CANCEL;
				gClientObj.sendMessagePort(message, len);
```

```
                        break;
                }

                prChat->preReceiveFile(buffer, size);
                SetForegroundWindow(prChat->getHWND());
                SetActiveWindow(prChat->getHWND());
                gClientObj.sendMessagePort(message, len);
                break;
        }
        case MessageType::SF_CANCEL:
        {
                /*
                * receive:      message = [FLAG | file name | NULL | file size | NULL | sender | NULL|
receiver | NULL]
                */
                message++;
                int     len = wcslen(message);
                WCHAR* partner;

                partner = message + len + 4;
                for (auto chatbox : gPrivateChatBoxList)
                {
                        if (wcscmp(chatbox->getPartner().c_str(), partner) == 0)
                        {
                                chatbox->onRefuseReceiveFile();
                                break;
                        }
                }
                break;
        }
        case MessageType::SF_ACCEPT:
        {
                /*
                * receive:      message = [FLAG | file name | NULL | file size | NULL | partner | NULL]
                * send:         message = [FLAG | file name | NULL | file size | NULL | partner | NULL]
                */

                message++;
                int     len = wcslen(message);
                WCHAR* partner;

                partner = message + len + 4;
                for (auto chatbox : gPrivateChatBoxList)
                {
                        if (wcscmp(chatbox->getPartner().c_str(), partner) == 0)
                        {
                                chatbox->onAcceptReceiveFile();
                                WCHAR buffer[1000];
                                int len = chatbox->onSendFile(buffer);

                                gClientObj.sendMessagePort(buffer, len);
                                break;
                        }
                }
                break;
        }
        case MessageType::FILE_DATA:
        {
                /*
                * receive:      message = [FLAG | file size | NULL | receiver | NULL | sender | NULL |
content]
                */
                message++;
                WCHAR* partner;
                WCHAR* content;
                DWORD size = *message;
                int len = 4 + gClientObj.getUsername().size();
```

```
                partner = message + len;

                for (auto chatbox : gPrivateChatBoxList)
                {
                        if (wcscmp(chatbox->getPartner().c_str(), partner) == 0)
                        {
                                content = message + len + chatbox->getPartner().size() + 2;
                                bool result = chatbox->receiveFile(content, size);

                                WCHAR messageReply[100];
                                WCHAR* sender;
                                messageReply[0] = MessageType::CONTINUE;
                                wcscpy(messageReply + 1, chatbox->getPartner().c_str());
                                sender = messageReply + chatbox->getPartner().size() + 2;
                                wcscpy(sender, chatbox->getUsername().c_str());

                                int len = chatbox->getPartner().size() + chatbox->getUsername().size() + 3;

                                if (!result)
                                {
                                        messageReply[0] = MessageType::STOP;
                                }
                                gClientObj.sendMessagePort(messageReply, len);

                                break;
                        }
                }
                break;
        }
        case MessageType::CONTINUE:
        {
                /*
                * receive:      message = [FLAG | receiver | NULL | sender | NULL]
                */
                WCHAR* partner;
                partner = message + wcslen(message) + 1;
                for (auto chatbox : gPrivateChatBoxList)
                {
                        if (wcscmp(chatbox->getPartner().c_str(), partner) == 0)
                        {
                                WCHAR buffer[600];
                                int len = chatbox->onSendFile(buffer);
                                gClientObj.sendMessagePort(buffer, len);
                        }
                }
                break;
        }
        case MessageType::STOP:
        {
                /*
                * receive:      message = [FLAG | receiver | NULL | sender | NULL]
                */
                WCHAR* partner;
                partner = message + wcslen(message) + 1;
                for (auto chatbox : gPrivateChatBoxList)
                {
                        if (wcscmp(chatbox->getPartner().c_str(), partner) == 0)
                        {
                                chatbox->onStop();
                        }
                }
                break;
        }
        break;
        }
        break;
}
```

```
case IDC_SIGNUP:
{
	/*
	* send:		message = [FLAG | username | NULL | password | NULL]
	*/
	WCHAR username[50];
	WCHAR password[50];
	WCHAR message[100];
	message[0] = MessageType::SIGNUP;
	message[1] = NULL;
	GetWindowText(hUsername, username, 50);
	if (username[0] == NULL)
	{
		MessageBox(hWnd, L"Username is empty.", 0, 0);
		return;
	}
	for (int i = 0; username[i] != NULL; i++)
	{
		if (username[i] == L';')
		{
			MessageBox(hWnd, L"The username has invalid characters: ;", 0, 0);
			return;
		}
	}
	GetWindowText(hPassword, password, 50);
	if (password[0] == NULL)
	{
		MessageBox(hWnd, L"Password is empty.", 0, 0);
		return;
	}
	wcscat(message, username);
	int len = wcslen(message);
	len++;
	int i;
	for (i = 0; password[i] != NULL; i++)
	{
		message[len + i] = password[i];
	}
	len += i;
	message[len] = NULL;
	gClientObj.sendMessagePort(message, len);
	break;
}
case IDC_LOGIN:
{
	/*
	* send:		message = [FLAG | username | NULL | password | NULL]
	*/
	WCHAR username[50];
	WCHAR password[50];
	WCHAR message[100];
	message[0] = MessageType::LOGIN;
	message[1] = NULL;
	GetWindowText(hUsername, username, 50);
	if (username[0] == NULL)
	{
		MessageBox(hWnd, L"Uername is empty.", 0, 0);
		return;
	}
	for (int i = 0; username[i] != NULL; i++)
	{
		if (username[i] == L';')
		{
			MessageBox(hWnd, L"Username has an invalid character: ;", 0, 0);
			return;
		}
	}
```

```
		GetWindowText(hPassword, password, 50);
		if (password[0] == NULL)
		{
			MessageBox(hWnd, L"Password is empty.", 0, 0);
			return;
		}
		wcscat(message, username);
		int len = wcslen(message);
		len++;
		int i;
		for (i = 0; password[i] != NULL; i++)
		{
			message[len + i] = password[i];
		}
		len += i;
		message[len] = NULL;
		gClientObj.sendMessagePort(message, len);
		gClientObj.setUsername(username);
		break;
	}
	case IDC_CREATE:
	{
		if (IsDlgButtonChecked(hWnd, IDC_PRIVATECHAT))
		{
			/*
			* send:		message = [FLAG | partner | NULL]
			*/
			WCHAR message[50];
			WCHAR buffer[50];

			message[0] = MessageType::CREATE_PRIVATE_CHAT;
			message[1] = NULL;
			GetWindowText(hInvitedUsername, buffer, 50);
			if (buffer[0] == NULL)
			{
				MessageBox(hWnd, L"Username is empty!", 0, 0);
				return;
			}
			if (wcscmp(buffer, gClientObj.getUsername().c_str()) == 0)
			{
				MessageBox(hWnd, L"You can't create the private chat by your username!", 0, 0);
				return;
			}
			wcscat(message, buffer);
			int len = wcslen(message);
			gClientObj.sendMessagePort(message, len);
			SetWindowText(hInvitedUsername, L"");
		}
		else
		{
			/*
			* send:		message = [FLAG | group name | NULL]
			*/
			WCHAR message[50];
			WCHAR buffer[50];
			message[0] = MessageType::CREATE_GROUP_CHAT;
			GetWindowText(hGroupName, buffer, 50);
			if (buffer[0] == NULL)
			{
				MessageBox(hWnd, L"Group name is empty!", 0, 0);
				return;
			}
			wcscpy(message + 1, buffer);
			int len = wcslen(message);
			gClientObj.sendMessagePort(message, len);
			SetWindowText(hGroupName, L"");
		}
```

```
		break;
}
}
}
void OnDrawItem(HWND hwnd, const DRAWITEMSTRUCT * lpDrawItem)
{
switch (lpDrawItem->CtlID)
{
case IDC_SIGNUP:
{
		auto graphics = new Graphics(lpDrawItem->hDC);
		Gdiplus::SolidBrush brush(Gdiplus::Color(255, 254, 88, 136));
		graphics->FillRectangle(&brush, 0, 0, 135, 30);

		Gdiplus::FontFamily  fontFamily(L"Arial");
		Gdiplus::Font        font(&fontFamily, 15, Gdiplus::FontStyleBold, Gdiplus::UnitPixel);
		Gdiplus::PointF      pointF(35, 6);
		Gdiplus::SolidBrush  solidBrush(Gdiplus::Color(255, 255, 255, 255));

		graphics->DrawString(L"Sign Up", -1, &font, pointF, &solidBrush);

		if (graphics)
			delete graphics;
		break;
}
case IDC_LOGIN:
{
		auto graphics = new Graphics(lpDrawItem->hDC);
		Gdiplus::SolidBrush brush(Gdiplus::Color(255, 69, 215, 194));
		graphics->FillRectangle(&brush, 0, 0, 135, 30);

		Gdiplus::FontFamily  fontFamily(L"Arial");
		Gdiplus::Font        font(&fontFamily, 15, Gdiplus::FontStyleBold, Gdiplus::UnitPixel);
		Gdiplus::PointF      pointF(43, 6);
		Gdiplus::SolidBrush  solidBrush(Gdiplus::Color(255, 255, 255, 255));
		graphics->DrawString(L"Log In", -1, &font, pointF, &solidBrush);

		if (graphics)
			delete graphics;
		break;
}
case IDC_CREATE:
{
		auto graphics = new Graphics(lpDrawItem->hDC);
		Gdiplus::SolidBrush brush(Gdiplus::Color(255, 69, 215, 194));
		graphics->FillRectangle(&brush, 0, 0, 280, 35);

		Gdiplus::FontFamily  fontFamily(L"Arial");
		Gdiplus::Font        font(&fontFamily, 15, Gdiplus::FontStyleBold, Gdiplus::UnitPixel);
		Gdiplus::PointF      pointF(110, 8);
		Gdiplus::SolidBrush  solidBrush(Gdiplus::Color(255, 255, 255, 255));
		graphics->DrawString(L"Create", -1, &font, pointF, &solidBrush);

		if (graphics)
			delete graphics;
		break;
}
}
}

void OnPaint(HWND hWnd)
{
PAINTSTRUCT ps;
HDC hdc = BeginPaint(hWnd, &ps);

auto graphics = new Graphics(hdc);
Gdiplus::Pen pen(Gdiplus::Color(255, 37, 156, 236));
```

```
switch (gCurScene)
{
case 0:
{
	graphics->DrawRectangle(&pen, 49, 159, 282, 37);
	graphics->DrawRectangle(&pen, 49, 249, 282, 37);

	Gdiplus::FontFamily  fontFamily(L"Arial");
	Gdiplus::Font	font(&fontFamily, 15, Gdiplus::FontStyleBold, Gdiplus::UnitPixel);
	Gdiplus::PointF	pointF(46.0f, 135.0f);
	Gdiplus::SolidBrush  solidBrush(Gdiplus::Color(255, 37, 156, 236));

	graphics->DrawString(L"Username:", -1, &font, pointF, &solidBrush);

	pointF = Gdiplus::PointF(46.0f, 225.0f);
	graphics->DrawString(L"Password:", -1, &font, pointF, &solidBrush);
	break;
}
case 1:
{
	Gdiplus::FontFamily  fontFamily(L"Arial");
	Gdiplus::Font	font(&fontFamily, 15, Gdiplus::FontStyleBold, Gdiplus::UnitPixel);
	Gdiplus::PointF	pointF(46.0f, 165.0f);
	Gdiplus::SolidBrush  solidBrush(Gdiplus::Color(255, 171, 210, 237));
	graphics->DrawString(L"Username:", -1, &font, pointF, &solidBrush);
	graphics->DrawRectangle(&pen, 49, 189, 282, 37);

	pointF = Gdiplus::PointF(46.0f, 305.0f);
	graphics->DrawString(L"Group name:", -1, &font, pointF, &solidBrush);
	graphics->DrawRectangle(&pen, 49, 329, 282, 37);
	break;
}
}

EndPaint(hWnd, &ps);
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
UNREFERENCED_PARAMETER(lParam);
switch (message)
{
case WM_INITDIALOG:
	return (INT_PTR)TRUE;

case WM_COMMAND:
	if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
	{
		EndDialog(hDlg, LOWORD(wParam));
		return (INT_PTR)TRUE;
	}
	break;
}
return (INT_PTR)FALSE;
}


LRESULT CALLBACK ChatBoxProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
switch (message)		/* handle the messages */
{
case WM_CTLCOLORSTATIC:
{
	HWND hStatic = (HWND)lParam;
	HDC hdc = (HDC)wParam;
```

```
		SetBkMode(hdc, TRANSPARENT);
		return (LRESULT)GetStockObject(DC_BRUSH);
}
case WM_DRAWITEM:
{
		auto dit = (DRAWITEMSTRUCT *)(lParam);
		switch (dit->CtlID)
		{
		case IDC_SEND_GROUP:
		case IDC_SEND:
		{
			auto graphics = new Graphics(dit->hDC);
			Gdiplus::SolidBrush brush(Gdiplus::Color(255, 69, 215, 194));
			graphics->FillRectangle(&brush, 0, 0, 75, 75);

			Gdiplus::FontFamily  fontFamily(L"Arial");
			Gdiplus::Font        font(&fontFamily, 17, Gdiplus::FontStyleBold, Gdiplus::UnitPixel);
			Gdiplus::PointF      pointF(15, 30);
			Gdiplus::SolidBrush  solidBrush(Gdiplus::Color(255, 255, 255, 255));

			graphics->DrawString(L"Send", -1, &font, pointF, &solidBrush);

			if (graphics)
				delete graphics;
			break;
		}
		case IDC_ATTACH:
		{
			auto graphics = new Graphics(dit->hDC);
			Gdiplus::SolidBrush brush(Gdiplus::Color(255, 160, 160, 160));
			graphics->FillRectangle(&brush, 0, 0, 75, 75);

			Gdiplus::FontFamily  fontFamily(L"Arial");
			Gdiplus::Font        font(&fontFamily, 17, Gdiplus::FontStyleBold, Gdiplus::UnitPixel);
			Gdiplus::PointF      pointF(10, 30);
			Gdiplus::SolidBrush  solidBrush(Gdiplus::Color(255, 255, 255, 255));

			graphics->DrawString(L"Attach", -1, &font, pointF, &solidBrush);

			if (graphics)
				delete graphics;
			break;
		}
		case IDC_ADD:
		{
			auto graphics = new Graphics(dit->hDC);
			Gdiplus::SolidBrush brush(Gdiplus::Color(255, 160, 160, 160));
			graphics->FillRectangle(&brush, 0, 0, 75, 25);

			Gdiplus::FontFamily  fontFamily(L"Arial");
			Gdiplus::Font        font(&fontFamily, 15, Gdiplus::FontStyleBold, Gdiplus::UnitPixel);
			Gdiplus::PointF      pointF(20, 3);
			Gdiplus::SolidBrush  solidBrush(Gdiplus::Color(255, 255, 255, 255));

			graphics->DrawString(L"Add", -1, &font, pointF, &solidBrush);

			if (graphics)
				delete graphics;
			break;
		}
		}
		break;
}
case WM_CREATE:
		break;

case WM_DESTROY:
```

```
		/*
		* send:		message = [FLAG | receiver | NULL | sender | NULL]
		*/
		for (auto chatbox : gPrivateChatBoxList)
		{
			if (chatbox->getHWND() == hwnd)
			{
				gPrivateChatBoxList.remove(chatbox);
				WCHAR message[101];
				WCHAR* sender;
				message[0] = MessageType::END_PRIVATE_CHAT;
				message[1] = NULL;
				wcscat(message, chatbox->getPartner().c_str());
				sender = message + chatbox->getPartner().size() + 2;
				wcscpy(sender, gClientObj.getUsername().c_str());
				int len = chatbox->getPartner().size() + gClientObj.getUsername().size() + 3;
				gClientObj.sendMessagePort(message, len);
				break;
			}
		}

		/*
		* send:		message = [FLAG | group name | NULL | sender | NULL]
		*/
		for (auto chatbox : gGroupChatBoxList)
		{
			if (chatbox->getHWND() == hwnd)
			{
				gGroupChatBoxList.remove(chatbox);
				WCHAR message[101];
				WCHAR* sender;
				message[0] = MessageType::END_GROUP_CHAT;
				message[1] = NULL;
				wcscat(message, chatbox->getGroupName().c_str());

				sender = message + chatbox->getGroupName().size() + 2;
				wcscpy(sender, gClientObj.getUsername().c_str());
				int len = chatbox->getGroupName().size() + gClientObj.getUsername().size() + 3;
				gClientObj.sendMessagePort(message, len);
				break;
			}
		}
		DestroyWindow(hwnd);

		break;
	case WM_COMMAND:
		switch (LOWORD(wParam))
		{
		case IDC_SEND:
		{
			/*
			* send:		message = [FLAG | receiver | NULL | sender | NULL | content | NULL]
			*/
			for (auto chatbox : gPrivateChatBoxList)
			{
				if (chatbox->getHWND() == hwnd)
				{
					WCHAR buffer[1000];
					int len = chatbox->onPressBtnSend(buffer);
					if (len == -1)
					{
						return 0;
					}
					gClientObj.sendMessagePort(buffer, len);
					break;
				}
			}
```

```
			break;
		}
		case IDC_SEND_GROUP:
		{
			/*
			* send:		message = [FLAG | group name | NULL | sender | NULL | content | NULL]
			*/
			for (auto chatbox : gGroupChatBoxList)
			{
				if (chatbox->getHWND() == hwnd)
				{
					WCHAR buffer[1000];
					int len = chatbox->onPressBtnSend(buffer);
					if (len == -1)
					{
						return 0;
					}
					gClientObj.sendMessagePort(buffer, len);
					break;
				}
			}
			break;
		}
		case IDC_ADD:
		{
			/*
			* send:		message = [FLAG | group name | NULL | added user | NULL]
			*/
			for (auto chatbox : gGroupChatBoxList)
			{
				if (chatbox->getHWND() == hwnd)
				{
					WCHAR buffer[1000];
					int len = chatbox->onPressBtnAdd(buffer);
					if (len == -1)
					{
						return 0;
					}
					gClientObj.sendMessagePort(buffer, len);
					break;
				}
			}
			break;
		}
		case IDC_ATTACH:
		{
			/*
			* send:		message = [FLAG | file name | NULL | file size | NULL | receiver | NULL |
sender | NULL]
			*/
			for (auto chatbox : gPrivateChatBoxList)
			{
				if (chatbox->getHWND() == hwnd)
				{
					WCHAR message[1000];
					WCHAR buffer[1000];
					WCHAR* partner;
					WCHAR* sender;
					message[0] = MessageType::SEND_FILE;
					BOOL isOpenFile = myCreateOpenFile(chatbox->getHWND(), buffer);
					if (!isOpenFile)
					{
						break;
					}
					chatbox->onPressBtnAttach(buffer);
					int lenBuf = wcslen(buffer);
					int i;
```

```
                                    for (i = lenBuf - 1; buffer[i] != L'\\' && i >= 0; i--) {}

                                    wcscpy(message + 1, &buffer[i + 1]);
                                    DWORD size = chatbox->getSizeAndOpenFile(buffer);


                                    int len = wcslen(message);
                                    len++;
                                    message[len++] = size >> 16;
                                    message[len++] = size;
                                    message[len++] = NULL;

                                    partner = message + len;
                                    wcscpy(partner, chatbox->getPartner().c_str());
                                    len += chatbox->getPartner().size() + 1;

                                    sender = message + len;
                                    wcscpy(sender, chatbox->getUsername().c_str());
                                    len += chatbox->getUsername().size();

                                    gClientObj.sendMessagePort(message, len);
                                    break;
                            }
                    }
                    break;
            }
            }
            break;
}
return DefWindowProc(hwnd, message, wParam, lParam);
}


BOOL myCreateOpenFile(HWND hwnd, WCHAR* filename)
{
OPENFILENAMEW ofn = { 0 };
WCHAR szFile[260];
ofn.lStructSize = sizeof(ofn);
ofn.hwndOwner = hwnd;
ofn.lpstrFile = szFile;
ofn.lpstrFile[0] = '\0';
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFilter = TEXT("All Files (*.*)\0*.*\0");
ofn.nFilterIndex = 1;
ofn.lpstrFileTitle = NULL;
ofn.nMaxFileTitle = 0;
ofn.lpstrInitialDir = NULL;
ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;
if (!GetOpenFileName(&ofn))
            return FALSE;
wcscpy(filename, ofn.lpstrFile);
return TRUE;
}


BOOL myCreateSaveFile(HWND hwnd, WCHAR* path)
{
OPENFILENAMEW ofn = { 0 };
WCHAR szFile[260];
WCHAR ex[10];
wcscpy(szFile, path);
int j;
for (j = wcslen(szFile) - 1; j > 0 && szFile[j] != L'.'; j--) {}
if (j != 0)
{
            wcscpy(ex, szFile + j);
```

```
	}
	ofn.lStructSize = sizeof(ofn);
	ofn.hwndOwner = hwnd;
	ofn.lpstrFile = szFile;
	ofn.nMaxFile = sizeof(szFile);
	ofn.lpstrFilter = TEXT("All Files (*.*)\0*.*\0");
	ofn.nFilterIndex = 1;
	ofn.lpstrFileTitle = NULL;
	ofn.nMaxFileTitle = 0;
	ofn.lpstrInitialDir = NULL;
	ofn.Flags = OFN_PATHMUSTEXIST | OFN_FILEMUSTEXIST;
	if (GetSaveFileName(&ofn) == FALSE)
		return FALSE;
	wcscpy(path, ofn.lpstrFile);
	int i;
	for (i = wcslen(path) - 1; i > 0 && path[i] != L'.'; i--) {}

	if (i == 0)
	{
		if (j != 0)
		{
			wcscat(path, ex);
		}
	}
	return TRUE;
	}

	LPWSTR convertSize(DWORD size)
	{
	WCHAR *buffer = new WCHAR[9];
	if (size / 1073741824 > 0)
	{
		wsprintf(buffer, L"%d.%d GB", size / 1073741824, (int)((double)(size % 1073741824) /
(double)1073741824)) * 10;
	}
	else if (size / 1048576 > 0)
	{
		wsprintf(buffer, L"%d.%d MB", size / 1048576, (int)((float)(size % 1048576) / (float)1048576)) * 10;
	}
	else if (size / 1024 > 0)
	{
		wsprintf(buffer, L"%d.%d KB", size / 1024, (int)((float)(size % 1024) / (float)1024)) * 10;
	}
	else
	{
		wsprintf(buffer, L"%d Byte", size);
	}
	return buffer;
	}
```

Лістинг 8 – Файл ChatClient.cpp

```
#include "stdafx.h"
#include "ChatClient.h"


ChatClient::ChatClient()
{
_isConnected = false;

}

void ChatClient::init(string ipAddress, int port)
{
```

```
_serverIPAddress = ipAddress;
_serverPort = port;
struct hostent *hp;
unsigned int addr;
struct sockaddr_in server;


WSADATA wsaData;

int wsaret = WSAStartup(0x101, &wsaData);


if (wsaret != 0)
{
	return;
}

_connect = socket(AF_INET, SOCK_STREAM, 0);
if (_connect == INVALID_SOCKET)
	return;

addr = inet_addr(_serverIPAddress.c_str());
hp = gethostbyaddr((char*)&addr, sizeof(addr), AF_INET);

if (hp == NULL)
{
	closesocket(_connect);
	return;
}

server.sin_addr.s_addr = *((unsigned long*)hp->h_addr);
server.sin_family = AF_INET;
server.sin_port = htons(_serverPort);
if(connect(_connect,(struct sockaddr*)&server,sizeof(server)))
{
	closesocket(_connect);
	return;
}
_isConnected = true;
return;
}

ChatClient::~ChatClient()
{
if(_isConnected)
	closesocket(_connect);
}

int ChatClient::sendMessagePort(WCHAR* message, int len)
{
	int iStat = 0;

	iStat = send(_connect, (char*)message, len * 2 + 2, 0);
	if(iStat == -1)
		return 1;

	return 0;

}

int ChatClient::recMessagePort()
{

	char acRetData[4096];
	int iStat = 0;

	iStat = recv(_connect, acRetData, 4096, 0);
```

```
		if(iStat == -1)
			return 1;
		SendMessage(_hwnd, WM_COMMAND, (WPARAM)IDC_RECEIVE, (LPARAM)acRetData);

		return 0;

}

bool ChatClient::isConnected()
{
return _isConnected;
}

void ChatClient::setHWND(HWND hwnd)
{
_hwnd = hwnd;
}

void ChatClient::setUsername(wstring username)
{
_username = username;
}

wstring & ChatClient::getUsername()
{
return _username;
}
```