МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Програмне забезпечення для керування ПК через мережу інтернет»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Проценко О.Б.**

Студента групи ІН – 72 — **Рубан Д.І.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи ІН-72 спеціальності "Інформатика" денної форми навчання Рубана Дениса Ігоровича.

Тема: " **Програмне забезпечення для керування ПК через мережу інтернет**"

Затверджена наказом по СумДУ

№_____ от _____2021 г.

Зміст пояснювальної записки: 1) огляд існуючих рішень; 2) вибір засобів реалізації; 3) програмна реалізація;

Дата видачі завдання "_____"_____2021 р.

Керівник випускної роботи _____ Проценко О.Б.

Завдання приняв до виконання _____ Рубан Д.І.

РЕФЕРАТ

Записка: 50 стор., 30 рис., 1 додаток, 15 джерел.

Об’єкт дослідження — програмне забезпечення для керування ПК через мережу інтернет.

Мета роботи — дослідження існуючого програмного забезпечення для керування ПК через мережу інтернет та його функціоналу , а також розробка аналогічного ПЗ та його тестування.

Методи дослідження — метод емпіричного дослідження.

Результати — розроблено клієнт та сервер який використовується для керування з’єднаннями між клієнтами; в клієнті були реалізовані такі функції: трансляція екрану; емуляція натискання клавіш клавіатури та миші; файловий менеджер; командний рядок.

PYTHON , ДИСТАНЦІЙНЕ КЕРУВАННЯ КОМП’ЮТЕРОМ , C# , WPF.

ЗМІСТ

ВСТУП 5
1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ 6
1.1 Програмне забезпечення TeamViewer 6
1.2 Програмне забезпечення AnyDesk 7
1.3 Програмне забезпечення UltraVNC 8
1.4 Постановка задачі 10
2 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ 11
2.1 Засіб реалізації серверу 11
2.2 Засіб реалізації клієнту 11
2.3 Вибір середовища розробки IDE 11
3 ПРОГРАМНА РЕАЛІЗАЦІЯ 13
3.1 Проектування алгоритму обміну даними між клієнтами та сервером 13
3.2 Розробка серверу 16
3.3 Розробка клієнту 17
3.4 Тестування розробленого програмного забезпечення 24
ВИСНОВОК 30
СПИСОК ЛІТЕРАТУРИ 31
ДОДАТКИ 32

ВСТУП

Кожного дня кількість персональних комп’ютерів збільшується і все більше людей починає ними користуватися , а доступ до мережі інтернет стає доступним майже по всьому світу. Нерідко виникає потреба отримати доступ до комп’ютеру який знаходиться далеко від вас. Наприклад потрібно допомогти людині яка не має достатньо досвіду з комп’ютерами налаштувати якусь програму. В таких ситуаціях можна скористатися спеціалізованим програмним забезпеченням.

Зазвичай головною функцією такого програмного забезпечення є трансляція всього що відбувається на екрані підконтрольного комп’ютеру та можливість взаємодіяти з його клавіатурою та мишею. Така функція дозволяє робити майже все що можна було б при фізичному контакті з комп’ютером. Проте зазвичай однією функцією такі програми не обмежуються. Додатковими функціями можуть бути: передача файлів , чат , відеозв’язок.

Актуальність випускної роботи полягає в тому що доступність користування розроблюваним програмним забезпеченням буде тільки збільшуватися також як і потреба в ньому.

Об’єктом даного дослідження є програмне забезпечення для керування ПК через мережу інтернет.

Предметом даного дослідження є програмне забезпечення для керування ПК через мережу інтернет.

Метою даної роботи є дослідження існуючого програмного забезпечення для керування ПК через мережу інтернет та його функціоналу , а також розробка аналогічного ПЗ та його тестування.

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Програмне забезпечення для віддаленого керування існувало ще на початку 2000-х. На даний момент існують різні варіації таких програм. Раніше була розповсюджена архітектура коли комп’ютер до якого потрібно було підключитися виступав сервером. Для того щоб бути сервером потрібно заздалегідь налаштувати комп’ютер тому це може визвати складність у звичайних користувачів. У наш час великої популярності набули програми для віддаленого керування в яких існує головний сервер , він виступає посередником між комп’ютером що керує та тим що керують. Це відкидає проблему з необхідністю налаштування серверу для отримання можливості підключення до нього. Окрім основної функції керування комп’ютером часто додають додатковий функціонал такий як: перегляд та передача файлів , чат , відео та аудіо зв’язок між користувачами. Основним призначенням цих програм є надання комп’ютерної підтримки на відстані.

1.1 Програмне забезпечення TeamViewer

Одна з самих популярних програм для віддаленого доступу[7]. Мільйони користувачів обирають її через простоту та надійність. Перший випуск стався ще в далекому 2005. Перша версія базувалася на системі VNC. Працівник IT сервісу хотів уникнути непотрібних поїздок до клієнтів для налаштування ПЗ. Так і з’явилася перша версія яка виявилася дуже успішною.

Після запуску програми у себе на комп’ютері можна навіть не реєструватися , адже вам надається тимчасовий ідентифікатор та пароль за допомогою яких можна підключитися до вашого комп’ютеру. З’єднання комп’ютерів проводиться через сервери TeamViewer , тому додаткове налаштування не потрібне.

Для некомерційного користування програмою платити не потрібно , але для комерційного використання є різні типи ліцензій під різні потреби. Безкоштовною версією програми можна користуватися скільки завгодно проте

деякий функціонал залишиться недоступним для вас , як наприклад web версія для керування з браузеру. Нижче можна побачити як програма виглядає (рис 1.1).

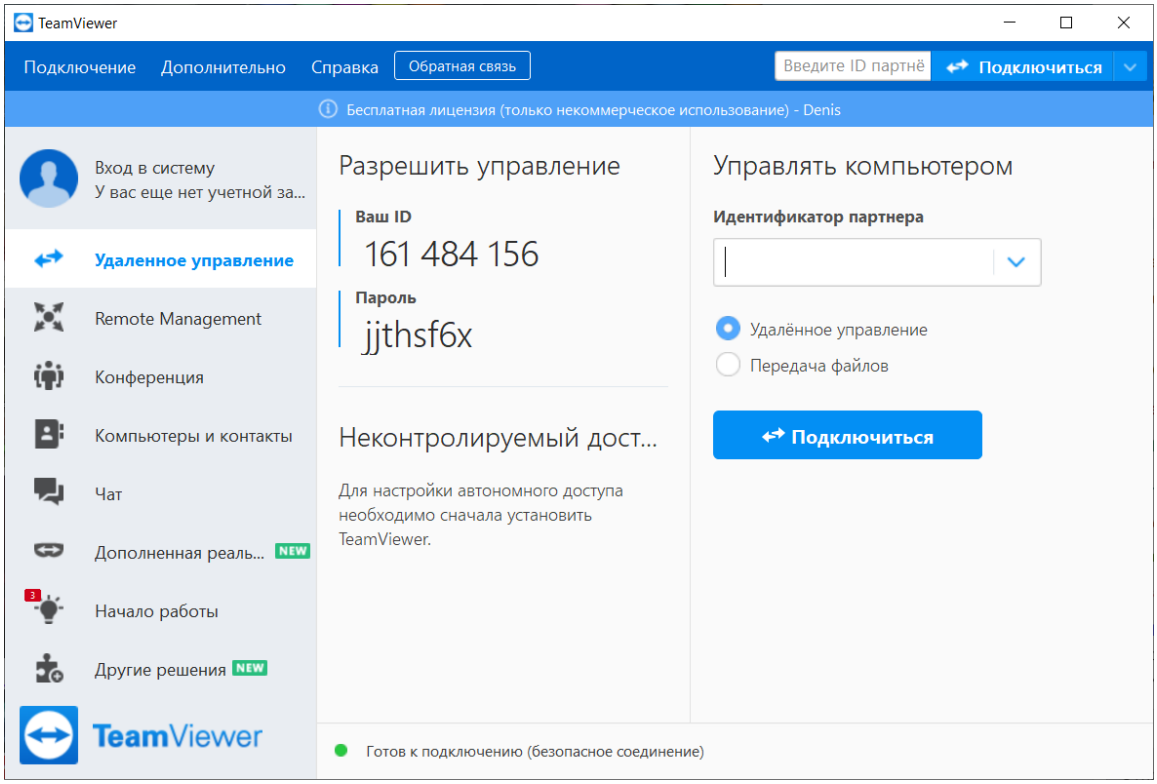

Рисунок 1.1 – Графічний інтерфейс TeamViewer

TeamViewer можна запустити на будь-якій сучасній ОС. Він співпрацює з 127 виробниками мобільних пристроїв , операційних систем , та IoT пристроїв , що є найбільшою кількістю серед конкурентів. Також він являється лідером в якості зображення та швидкості передачі файлів. Всі з’єднання між клієнтами захищені 256-бітним шифруванням по стандарту AES.

1.2 Програмне забезпечення AnyDesk

Ще одна програма яка займає лідируючі позиції серед програм для віддаленого доступу. Компанія була заснована в 2014 в Германії і швидко стала успішною.

Команда AnyDesk розробила кодек DeskRT спеціально під їхні потреби[8]. Це надає можливість користуватися програмою навіть при

швидкості інтернету менше 100 кБ/с , а також найбільшу кількість кадрів за секунду серед конкурентів.

Як і в випадку з TeamViewer з'єднання між клієнтами проводяться через сервери компанії без додаткового налаштування вашого комп'ютеру , а реєстрація не обов'язкова. Некомерційне користування також безкоштовне. Хоча в наявності є декілька тарифних планів але відрізняються вони в основному лише кількістю одночасних сесій. Нижче можна побачити як програма виглядає (рис 1.2).

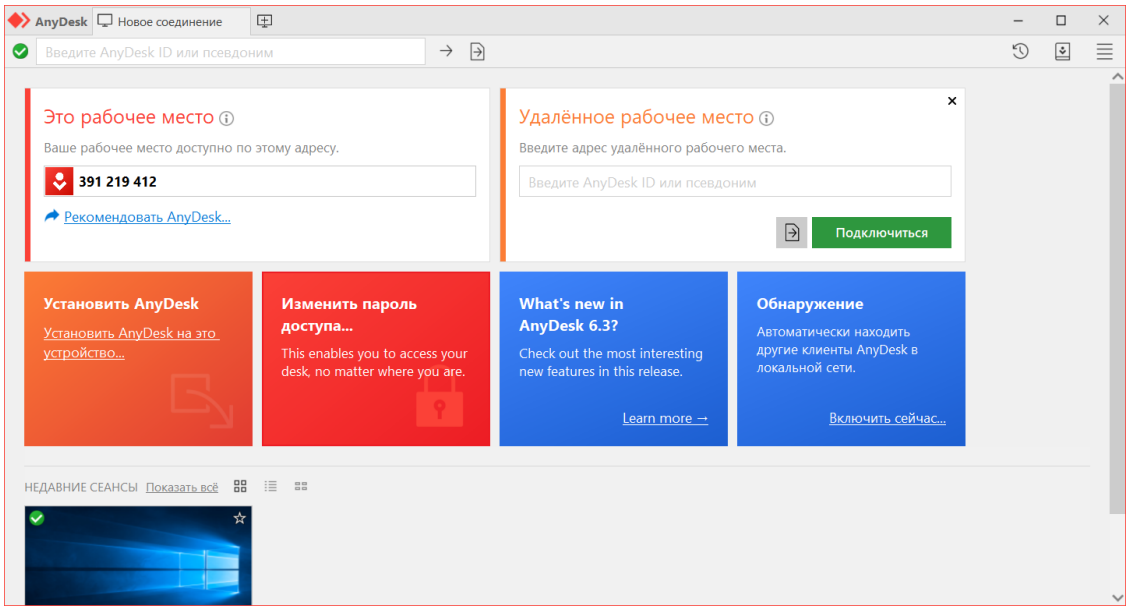

Рисунок 1.2 – Графічний інтерфейс AnyDesk

Підтримка всіх сучасних ОС наявна. Розмір самої програми AnyDesk значно меньше конкурентів що є приємним бонусом. Шифрування даних між клієнтами відбувається за допомогою протоколу TLS версії 1.2. Також присутня функція запису екрану контрольованого комп'ютеру.

1.3 Програмне забезпечення UltraVNC

Одна із багатьох програм для віддаленого керування комп'ютером яка базується на VNC. Вона являється продуктом з відкритим програмним кодом[9]. Кожен бажаючий може скомпілювати програму у себе на комп'ютері , а при необхідності відредагувати в код під його потреби.

Перша версія була випущена в 2005 і з того часу не переставала розвиватися. Підключення між клієнтами проходить напряму , так як виділених серверів немає. Сервером завжди виступає той кого контролюють. Для успішного підключення клієнту до серверу через мережу інтернет потрібно щоб у сервера були відкриті порти і доступні ззовні.

UltraVNC є повністю безкоштовною для комерційного чи некомерційного використання. Великим мінусом є те що сервер доступний тільки для Windows , тобто контролювати можна тільки комп'ютери під ОС Windows. Значною перевагою варто відмітити гнучкі можливості налаштування які можна побачити на рисунку нижче (рис 1.3) такі як: колір , компресія , якість зображення. Також присутня можливість додавати свої плагіни для шифрування трафіку.

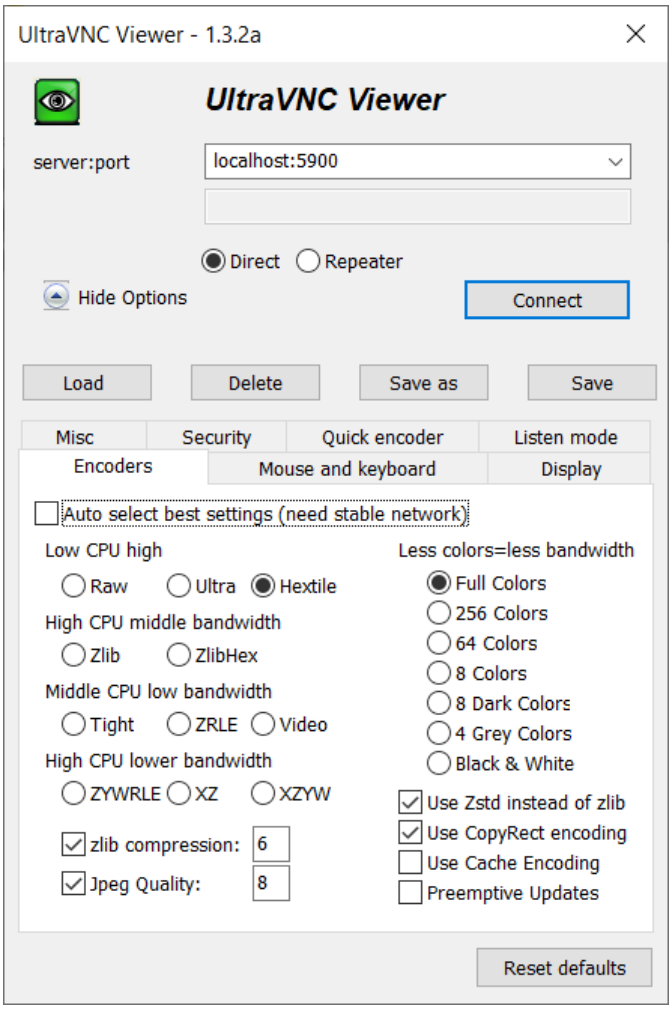

Рисунок 1.3 – Графічний інтерфейс UltraVNC

1.4 Постановка задачі

Був проведений аналіз існуючих рішень для віддаленого керування комп’ютером. На основі отриманої інформації було сформовано список необхідного функціоналу майбутнього додатку:

- трансляція екрану
- емуляція натискання клавіш клавіатури та миші
- файловий менеджер
- командний рядок
- сервер який буде виступати посередником між клієнтами

2 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Засіб реалізації серверу

Для розробки серверу було обрано мову програмування Python. Існують 2 основні версії: Python 2 та Python 3. Python 2 вже не підтримується хоча багато програм якими користуються і по цей день було створено на ньому. Так як Python 3 більш сучасний та регулярно оновлюється то його було обрано для розробки серверу.

Для облегшення та прискорення швидкості розробки було вирішено використовувати фреймворк. Вибір пав на Tornado[3]. Він дозволяє розроблювати веб додатки різної складності. Один із його плюсів це асинхронність , що дозволяє значно покращити його продуктивність.

2.2 Засіб реалізації клієнту

Для розробки клієнту була обрана мова програмування C# разом з .NET Framework. Вони були обрані через високу швидкість розробки програмного забезпечення для Windows.

Клієнт матиме графічний інтерфейс тому для його побудови буде використано графічну систему WPF[1].

Також коли клієнт буде виступати у ролі комп’ютера якого контролюють він повинен буде емулювати натискання клавіш миші та клавіатури. З цим завданням допоможе справитись бібліотека WindowsInput[5].

2.3 Вибір середовища розробки IDE

У якості інтегрованого середовища розробки буде використано Visual Studio 2019. З її допомогою можна швидко і комфортно створювати дизайн графічного інтерфейсу для WPF , приклад можна побачити нижче (рис 2.1). Також вона має велику кількість інструментів для відладки що може значно зекономити час на знаходження помилок.

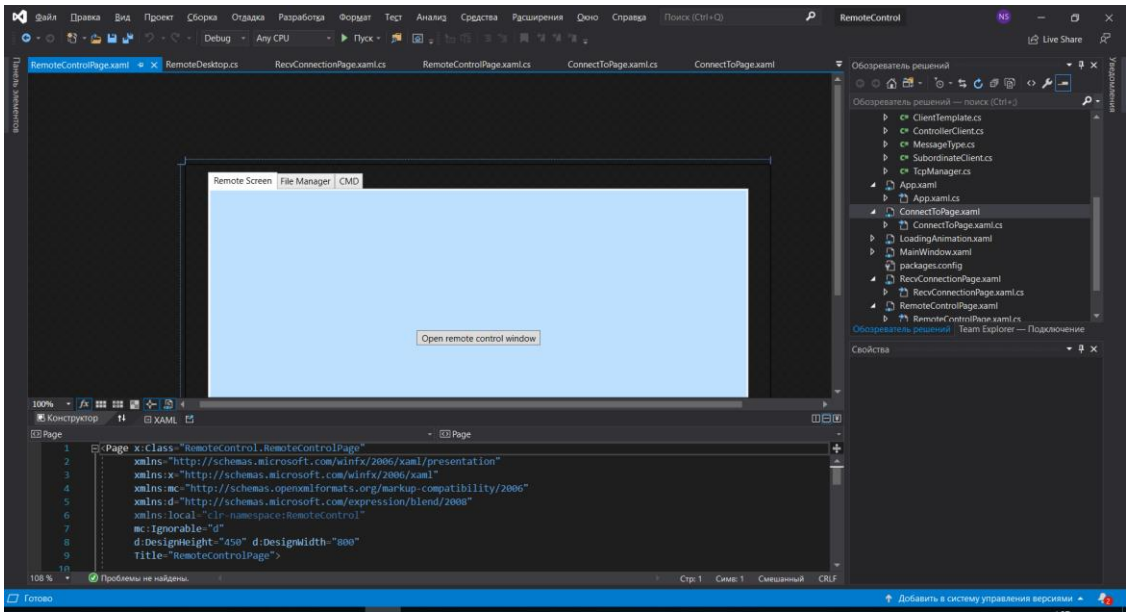

Рисунок 2.1 – Конструктор графічного інтерфейсу WPF в Visual Studio 2019

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Проектування алгоритму обміну даними між клієнтами та сервером

При виборі протоколу за яким будуть обмінюватися даними клієнти з сервером стояв вибір між TCP та UDP. Зваживши всі за і проти був обраний протокол TCP через його надійність , адже при виборі UDP потрібно було б імплементувати перевірку на правильність порядку отриманих пакетів та вирішувати що робити з втраченими пакетами[6].

Сервер буде виступати посередником між клієнтами , тобто буде пересилати данні від одного клієнта іншому , це можна побачити на рисунку 3.1. Для того щоб з'єднати одного клієнта з іншим , клієнт який під'єднується повинен знати ID клієнта до якого буде підключатися. ID генеруються на стороні клієнту , сервер тільки перевіряє щоб не було однакових ID.

Рисунок 3.1 – Передача повідомлень між клієнтами

Клієнт з сервером будуть спілкуватися використовуючи заздалегідь вигадані повідомлення. Заголовок кожного повідомлення буде складатися з 8 байт. Перші 4 байти повідомлення будуть відповідати за його тип , потім 4 байти які будуть відповідати за розмір даних повідомлення. Вигляд повідомлення в схематично зображено на рисунку 3.2.

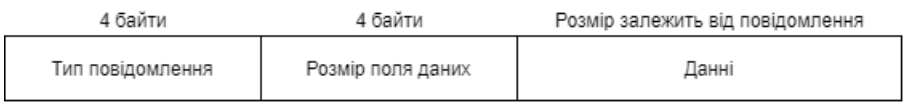

Рисунок 3.2 – Схема повідомлення

Після отримання заголовка повідомлення якщо в байтах які відповідають за розмір знаходиться не 0 то отримана кількість байт також зчитується так як вона являється частиною повідомлення. Поки не буде отримано заголовок повідомлення та кількість даних зазначених в заголовкові повідомлення не буде вважатися отриманим. Ось весь список повідомлень , контролер – той хто контролює , підопічний – той кого контролюють , сервер – посередник між клієнтами , номер перед назвою повідомлення це цифрове значення яке буде відправлятися в заголовкові:

Контролер → Сервер

1) ConnectById - запит на підключення по ID , в полі данних передається ID в виді рядку

Сервер → Контролер

2) WrongConnectId - неправильний ID при запиті на підключення по ID , поле данних відсутньо
3) Connected - успішне підключення до підопічного , також повідомляє підопічному що до нього підключився контролер , поле данних відсутньо

Підопічний → Сервер

4) CheckConnectId - перевірка чи ID не зайнятий іншим підопічним , в полі данних передається ID в виді рядку

Сервер → Підопічний

5) CheckResult – результат перевірки ID підопічним , в полі даних передається 1 байт в якому при значенні 1 означає що результат перевірки позитивний(ID доступний для використання) інакше результат негативний(оберіть інший ID)

Сервер → * (контролер і підопічний)

6) DisconnectMsg – індикує відключення контролера або підопічного , поле данних відсутнє

Контролер → Підопічний

7) StartRemoteDesktop – запускає віддалений робочий стіл , поле даних відсутнє

8) StopRemoteDesktop – зупиняє віддалений робочий стіл , поле даних відсутнє

10) KeyPress – емулює натискання або відпускання клавіші на клавіатурі , в полі даних 2 байти перший з яких відповідає за те натиснута чи відпущена клавіша , 1 – натиснута , 2 - відпущена , а другий за keycode клавіші

11) MousePress – емулює натискання правої або лівої клавіші миші , в полі даних 2 байти перший з яких відповідає за те натиснута чи відпущена клавіша , 1 – натиснута , 2 – відпущена , а другий за клавішу , 1 – ліва , 2 – права

12) MouseMove – емулює рух миші , в полі даних 8 байт , перші 4 байти в типі float відповідають за x координату миші де 0 – ліва частина екрану , а 1 – права частина екрану , інші 4 байти в типі float відповідають за y координату миші де 0 – верхня частина екрану , а 1 – нижня частина екрану

13) StartCMD – запускає командний рядок , поле даних відсутнє

14) StopCMD – зупиняє командний рядок , поле даних відсутнє

15) CMDInput – данні які вписуються в командний рядок , в полі даних передаються команди в виді рядку

Підопічний → Контролер

9) RemoteDesktopFrame – скріншот екрану , в полі даних передається скріншот екрану в форматі JPG

16) CMDOutput – вихідний текст з командного рядку , в полі даних передається вихідний текст в виді рядку

3.2 Розробка серверу

До серверу будуть підключатися 2 типи клієнтів , ті хто буде контролювати іншим клієнтом та ті хто буде контролювати іншого клієнта. Сервер повинен буде визначити після налаштування з'єднання який тип клієнту до нього під'єднався. Визначення проводитиметься в залежності від першого повідомлення відправленого від клієнту , далі в залежності від визначеного типу буде 2 різних алгоритми за якими буде поводити себе сервер. Нижче можна побачити як це реалізовано в коді (рис 3.3).

```
packet_header = await stream.read_bytes(PacketHeaderSize)
message_type , size = PacketHeader.unpack(packet_header)

message_data = None
if size != 0:
    message_data = await stream.read_bytes(size)

if message_type == MessageTypes["ConnectById"]:
    await self.handle_connect_by_id(stream , message_data)

elif message_type == MessageTypes["CheckConnectId"]:
    await self.handle_check_connect_id(stream , message_data)
```

Рисунок 3.3 – Визначення типу клієнту

Якщо клієнтом буде той хто буде контролюватися іншим клієнтом то в першому повідомленні він надішле ID на перевірку чи не зайнятий він іншим клієнтом. У випадку коли такий ID вже занятий сервер повідомить про це клієнта і буде знову чекати на те ж саме повідомлення з іншим ID. Сервер буде приймати повідомлення з ID до того моменту поки вони будуть зайняті , коли це станеться буде надіслана відповідь що перевірка прошла успішно , нижче можна побачити реалізацію описаного алгоритму в коді (рис. 3.4). Обраний ID буде запам'ятовано і сервер буде чекати сигналу доки інший клієнт не захоче підключитися для керування. Коли сигнал буде отримано всі дані що будуть отримані від клієнту будуть відправлені тому хто ним керує.

```
while connect_id in ConnectIdDict:
    await self.send_message(stream , MessageTypes["CheckResult"] , b"\x00")

    packet_header = await stream.read_bytes(PacketHeaderSize)

    message_type , size = PacketHeader.unpack(packet_header)
    if message_type != MessageTypes["CheckConnectId"]:
        stream.close()
        return

    message_data = await stream.read_bytes(size)

    connect_id = message_data.decode("utf-8")
await self.send_message(stream , MessageTypes["CheckResult"] , b"\x01")
```

Рисунок 3.4 – Очікування доступного ID

Якщо клієнтом буде той хто контролює іншого клієнта то він в першому повідомленні надішле ID клієнта якого він буде контролювати. Сервер спершу перевірить чи існує на даний момент підключений клієнт з таким ID. Якщо такого клієнта не знайдено то відповідне повідомлення буде надіслано і сервер розірве з’єднання. У випадку коли шуканий клієнт знайдено і результат перевірки відправлено сервер сигналізує клієнту який знаходиться в стані очікування щоб "розбудити" його. Далі всі отримані дані від клієнту будуть відправлені тому ком він керує.

Коли два клієнти налаштували між собою зв’язок через сервер вони можуть відправляти будь які повідомлення між собою , сервер буде зразу ж пересилати їх від одного клієнта іншому. Це буде продовжуватися доки один із клієнтів не розірве з’єднання. У цьому випадку сервер повідомить про це клієнта який був з’єднаний з ним.

3.3 Розробка клієнту

Клієнт матиме головне вікно в якому в залежності від дій користувача буде відображатися ті чи інші елементи. Для того щоб було легше групувати елементи вони будуть розділені на сторінки. Користувач буде взаємодіяти з елементами і таким чином переходити з однієї сторінки на іншу. Далі будуть

розглянуті всі сторінки з їх скріншотами в конструкторі Visual Studio та алгоритмом роботи.

Стартова сторінка – це сама перша сторінка яка з’являється відразу після запуску. В ній користувач обирає буде він підключатися для керування іншим комп’ютером чи чекати на підключення іншого клієнту. Її можна побачити на рисунку 3.5.

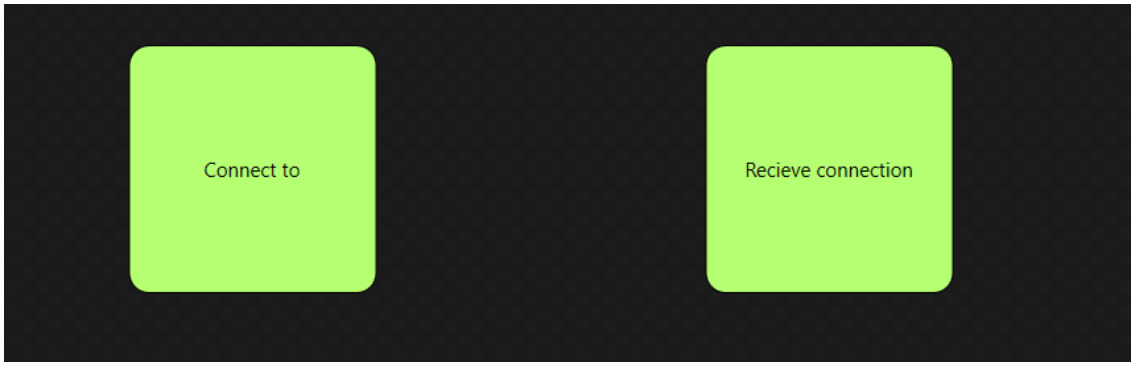

Рисунок 3.5 – Стартова сторінка

При переході на сторінку очікування підключення клієнт згенерує ID та відправить на перевірку серверу , якщо ID зайнятий клієнт буде генерувати ID доки не знайде придатний для використання. Генерація ID відбувається за допомогою структури GUID. Вона генерує 128 бітне число яке потім перетворюється в текстовий рядок(рис. 3.6).

```
if (UID == null) UID = Guid.NewGuid().ToString();
while (!SubordinateClient.CheckConnectId(UID)) {
    UID = Guid.NewGuid().ToString();
}
```

Рисунок 3.6 – Сторінка очікування підключення

Коли підключення до серверу пройшло успішно у текстовому полі буде знаходитися ID який можна буде передати іншому клієнту щоб він зміг під’єднатися. Проте якщо підключитися до срверу не вдасться то відповідне повідомлення виведеться в полі статусу щоб можна було зрозуміти що щось пішло не так. Сторінку можна побачити на рисунку 3.7.

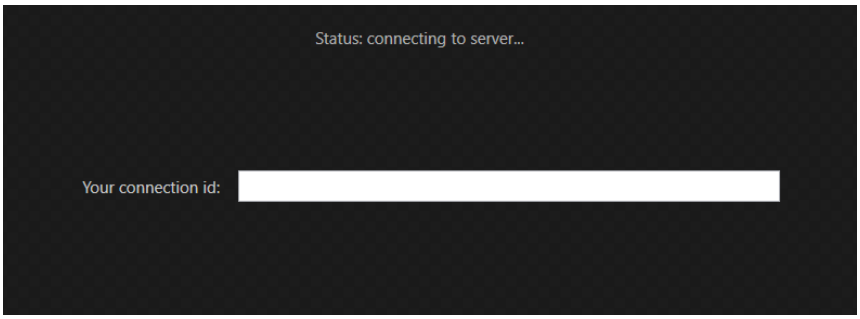

Рисунок 3.7 – Сторінка очікування підключення

На сторінці підключення до іншого клієнту потрібно ввести ID. Якщо клієнту з таким ID не існує то буде показано повідомлення в якому про це буде говоритися. У випадку успішного підключення поточну сторінку буде змінено на сторінку керування віддаленим комп’ютером. Сторінку можна побачити на рисунку 3.8.

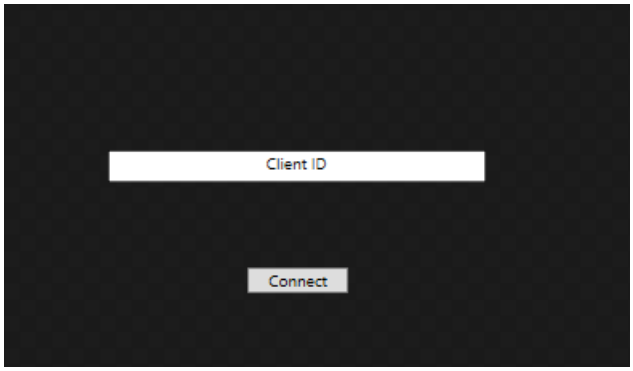

Рисунок 3.8 – Сторінка очікування підключення

Після успішного підключення до іншого клієнту буде відкрито сторінку керування віддаленим комп’ютером. Тут розташовані три вкладки , кожна з яких відповідає за частину функціоналу для контролю віддаленого клієнту. Перша вкладка відповідає за перегляд екрану , керування мишею та клавіатурою клієнту.

Зображення екрану передається у форматі JPEG. Для зменшення розміру зображення перед відправкою його якість зменшується до 35 відсотків (рис. 3.9).

```
ImageCodecInfo jpgEncoder = null;
ImageCodecInfo[] codecs = ImageCodecInfo.GetImageEncoders();
foreach (ImageCodecInfo codec in codecs) {
    if (codec.FormatID == ImageFormat.Jpeg.Guid) {
        jpgEncoder = codec;
    }
}

Encoder myEncoder = Encoder.Quality;
EncoderParameters myEncoderParameters = new EncoderParameters(1);
EncoderParameter myEncoderParameter = new EncoderParameter(myEncoder , 35L);
myEncoderParameters.Param[0] = myEncoderParameter;

bitmap.Save(ms , jpgEncoder , myEncoderParameters);
```

Рисунок 3.9 – Код зміни якості зображення

При натисканні клавіш клавіатури або миші та її руху всі дії будуть відсилатися до віддаленого клієнту та виконуватися. Для емуляції натискання клавіш та руху миші використовується бібліотека WindowsInput (рис. 3.10).

```
public static void KeyDown(byte keyCode) {
    InputSim.Keyboard.KeyDown((VirtualKeyCode) keyCode);
}

public static void KeyUp(byte keyCode) {
    InputSim.Keyboard.KeyUp((VirtualKeyCode) keyCode);
}

public static void MouseMove(float x , float y) {
    InputSim.Mouse.MoveMouseTo(x * 65535 , y * 65535);
}

public static void MouseButtonDown(byte button) {
    if (button == 1) InputSim.Mouse.LeftButtonDown();
    else if (button == 2) InputSim.Mouse.RightButtonDown();
}

public static void MouseButtonUp(byte button) {
    if (button == 1) InputSim.Mouse.LeftButtonUp();
    else if (button == 2) InputSim.Mouse.RightButtonUp();
}
```

Рисунок 3.10 – Код емуляції клавіатури та миші

Після натискання по кнопці що розташована посередині з’явиться нове вікно в якому і буде відображатися екран клієнту. Скріншот першої вкладки можна побачити на рисунку 3.11.

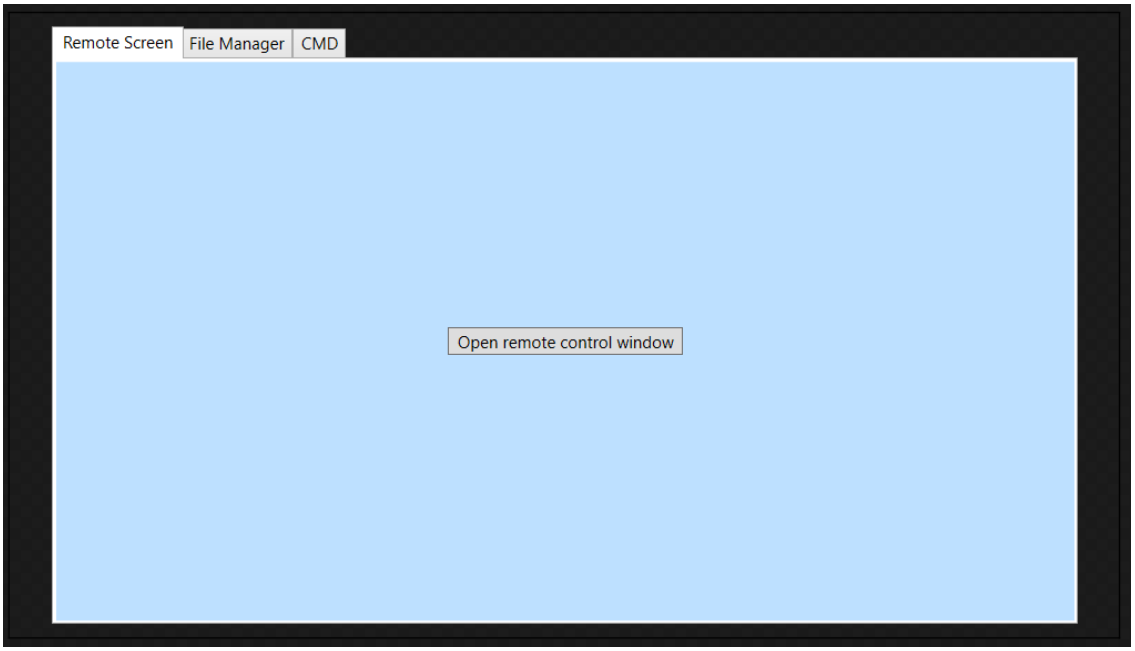

Рисунок 3.11 – Сторінка керування віддаленим комп’ютером з першою вкладкою

Друга вкладка відповідає за перегляд , скачування та завантаження файлів. Для переходу по папкам потрібно зробити подвійний клік по папці в яку треба перейти. Щоб скачати файл потрібно вибрати його та натиснути кнопку «Download file». Якщо потрібно завантажити файл на комп’ютер віддаленого клієнту то потрібно натиснути кнопку «Upload file» і в файловому діалозі що відкрився обрати файл для завантаження. Для відкриття файлового діалогу використовується функції із простору імен Win32 (рис. 3.12).

```
Microsoft.Win32.OpenFileDialog fileDialog = new Microsoft.Win32.OpenFileDialog();
Nullable<bool> result = fileDialog.ShowDialog();
if (result == true) {
    UploadStatusLabel.Content = "Status: uploading...";
```

Рисунок 3.12 – Код відкриття файлового діалогу

Завантажений файл з’явиться на комп’ютері клієнту в тій папці яка була відкрита на момент початку завантаження. Вигляд другої вкладки можна побачити на рисунку 3.13

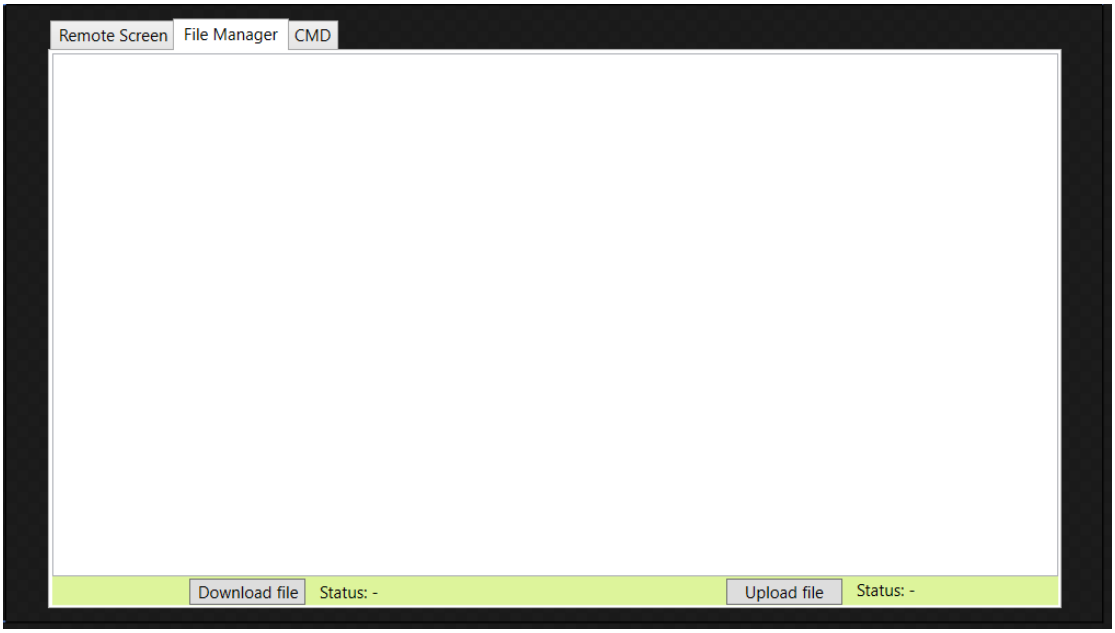

Рисунок 3.13 – Вкладка файлового менеджеру

Третя вкладка відповідає за командний рядок. Для забезпечення повноцінного командного рядку за допомогою класу Process запускається новий процес програми cmd.exe (рис. 3.14). Стандартний вивід , ввід та вивід помилок перенаправляються для того щоб можна було зчитувати що виводиться в командний рядок та потім відправляти , а також вводити команди.

```
cmdProcess = new Process();
cmdProcess.StartInfo.FileName = "cmd.exe";
cmdProcess.StartInfo.CreateNoWindow = true;
cmdProcess.StartInfo.UseShellExecute = false;
cmdProcess.StartInfo.RedirectStandardOutput = true;
cmdProcess.StartInfo.RedirectStandardInput = true;
cmdProcess.StartInfo.RedirectStandardError = true;
cmdProcess.OutputDataReceived += new DataReceivedEventHandler(CmdOutputDataHandler);
cmdProcess.ErrorDataReceived += new DataReceivedEventHandler(CmdOutputDataHandler);
cmdProcess.Start();
```

Рисунок 3.14 – Запуск командного рядку

В нижньому текстовому полі потрібно писати команду і по натисненню кнопки Enter вона буде відправлена клієнту. Приклад виконаної команди можна побачити на рисунку 3.15.

Рисунок 3.15 – Вкладка командного рядку

В верхньому лівому краю буде знаходитися кнопка для повернення на попередню сторінку (рис. 3.16). Вона може понадобитися якщо користувач захотів змінити режим який обрав першого разу в меню без перезапуску програми , або підключитися до іншого клієнту якщо він уже підключений.

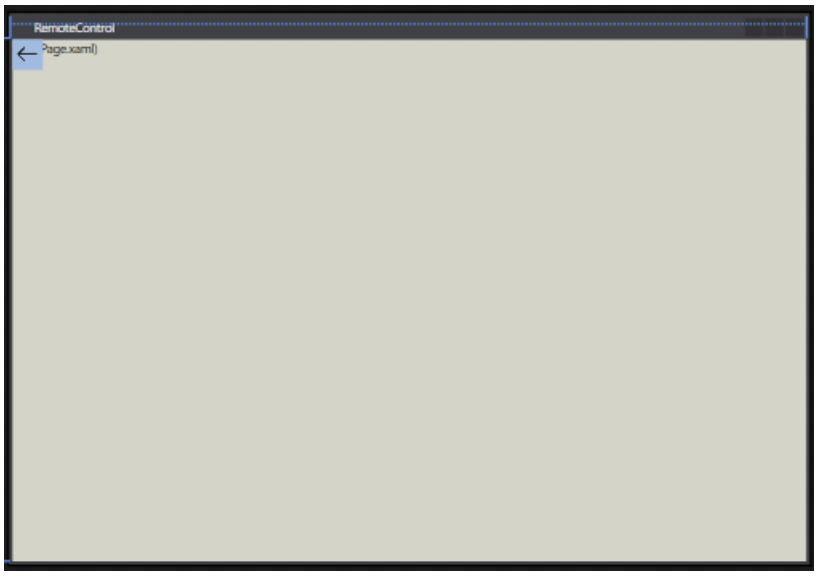

Рисунок 3.16 – Кнопка повернення до попередньї сторінки

3.4 Тестування розробленого програмного забезпечення

Для початку потрібно запустити сервер. На комп’ютері який буде підконтрольним треба запустити клієнт та в стартовому меню обирати «Receive connection». Скріншот стартового меню знаходиться на рисунку 3.17.

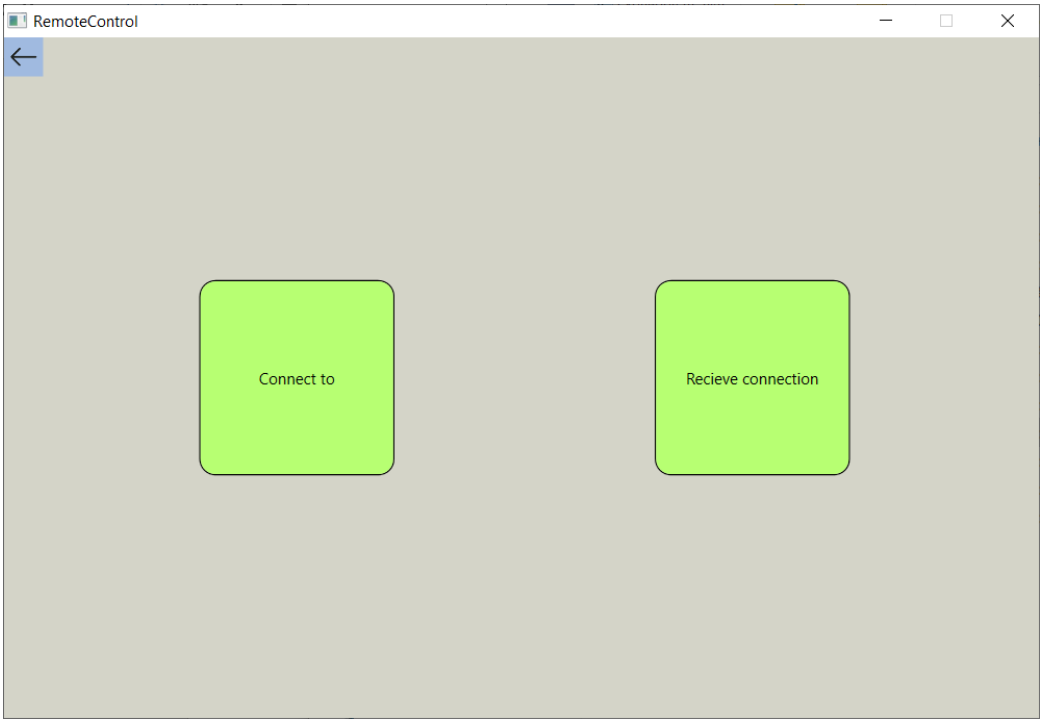

Рисунок 3.17 – Стартове меню

Після успішного підключення до серверу у відповідному текстовому полі з’явився ID по якому буде проводитися підключення , що і можна побачити на рисунку 3.18. Тепер все що треба зробити це передати отриманий ID іншому клієнту для підключення.

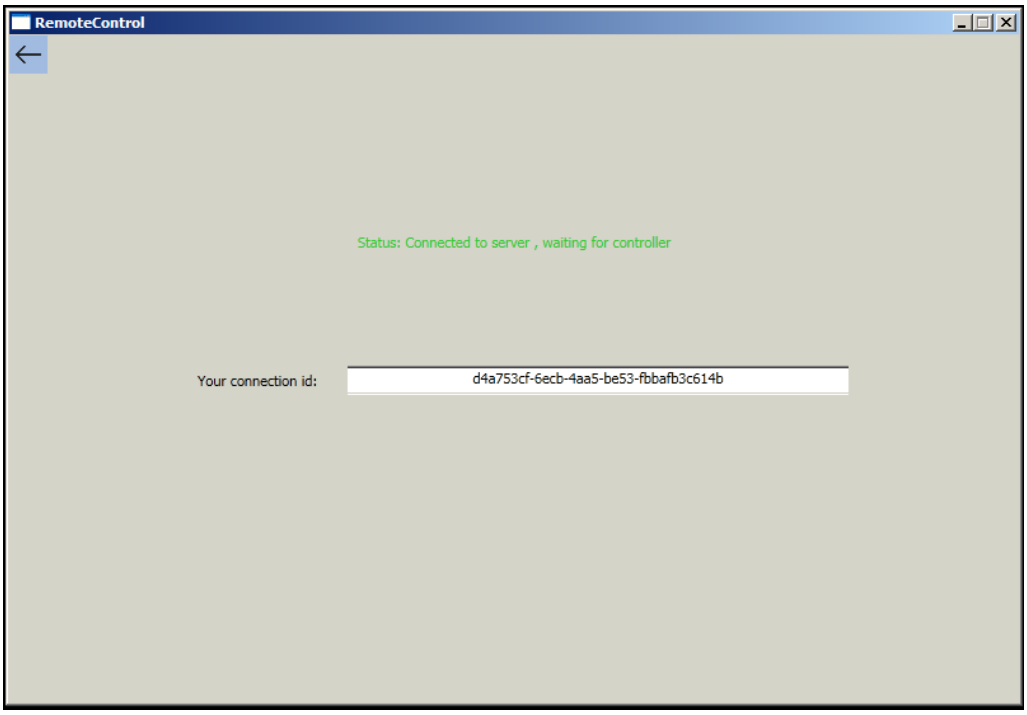

Рисунок 3.18 – Очікування підключення клієнту

Далі на іншому комп’ютері потрібно запустити клієнт та в меню обрати «Connect to» , потім ввести ID для підключення який знаходиться у іншого клієнту , це можна побачити на рисунку 3.19. Тепер треба натиснути кнопку «Connect».

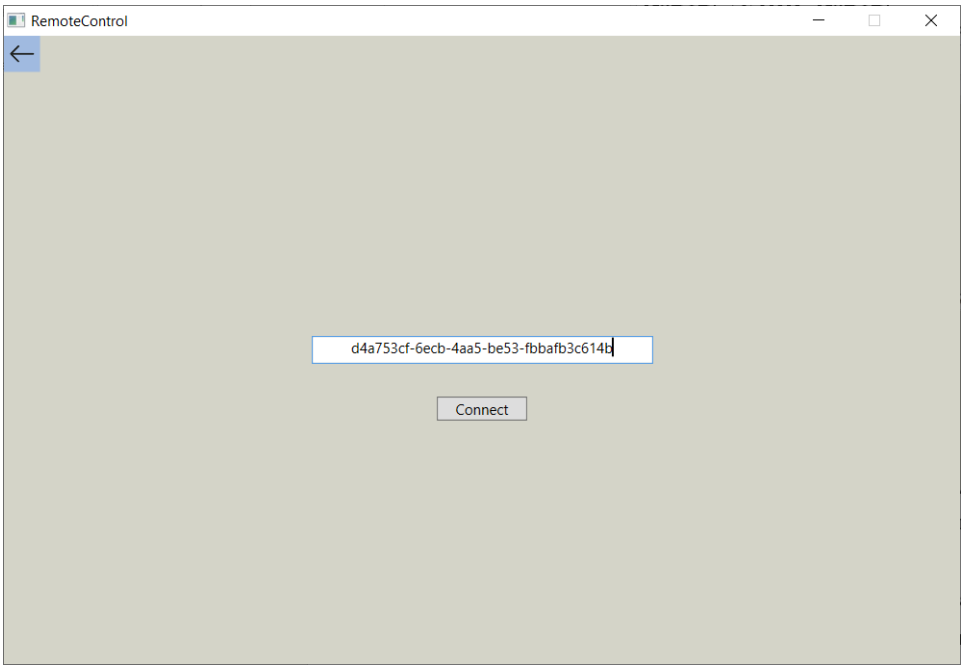

Рисунок 3.19 – Підключення по ID

Після успішного підключення статус в вікні підконтрольного клієнту змінився на «controller connected» , що можна побачити на рисунку 3.20. Це свідчить про те що інший клієнт під’єднався і тепер може контролювати ваш комп’ютер.

Рисунок 3.20 – Підключення пройшло успішно

Після підключення відкрився інтерфейс для керування іншим клієнтом. Необхідно натиснути на кнопку «Open remote control window» і з’явиться нове вікно в якому відображається екран іншого клієнту , результат можна побачити на рисунку 3.21. Після користування даною функцією можна просто закрити вікно що відкрилося ріниіше.

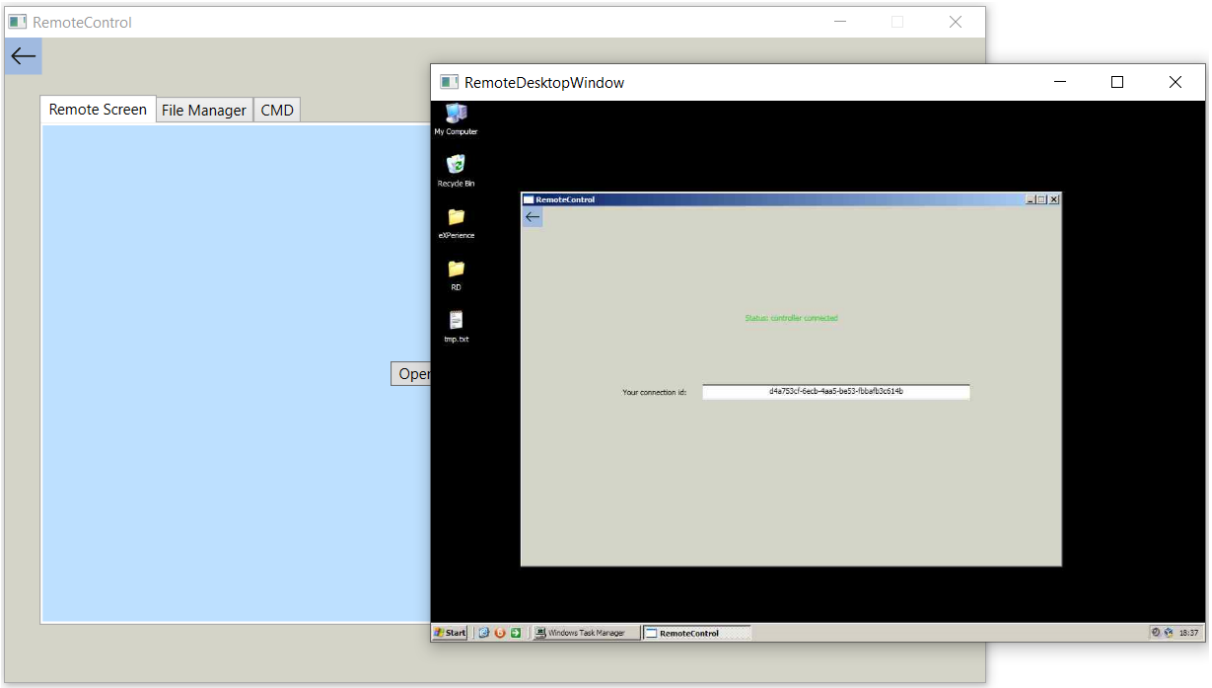

Рисунок 3.21 – Вікно віддаленого керування

Тепер можна протестувати файловий менеджер. Для слід перейти в його вкладку , тут знаходиться список папок. Для перезоду в якусь папку треба зробити подвійний клік по ній , список папок можна побачити на рисунку 3.22.

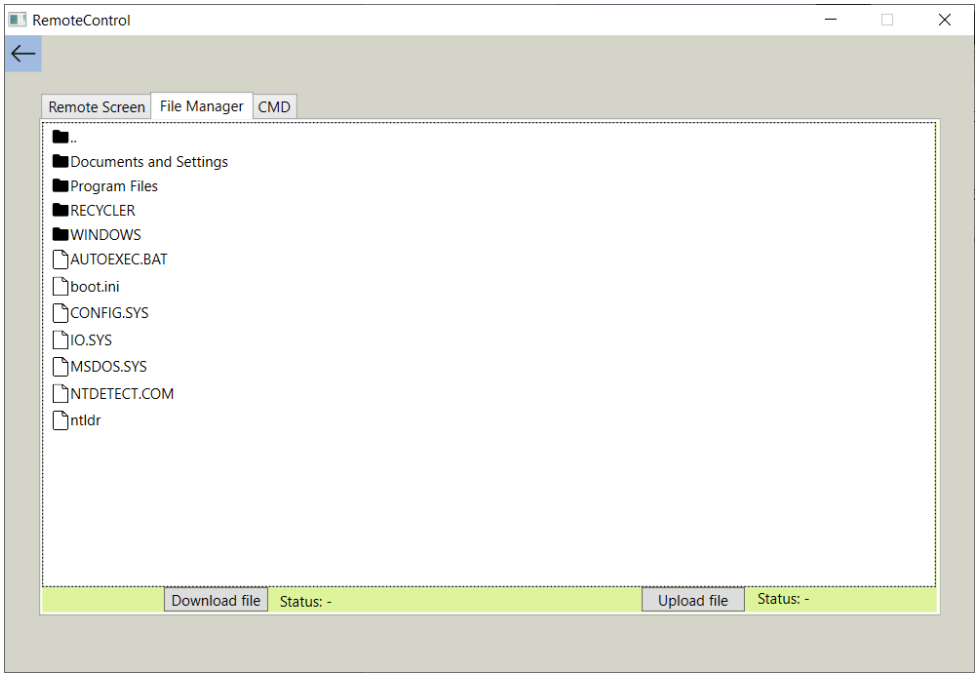

Рисунок 3.22 – Список папок файлового менеджеру

Для скачування файлу потрібно обрати його в списку і натиснути на кнопку «Download file». Статус біля кнопки зміниться на «downloading…» , і після завершення він зміниться знову на «completed». Фінальний статус можна побачити на рисунку 3.23.

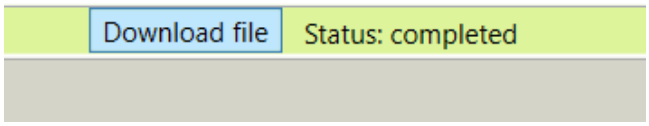

Рисунок 3.23 – Статус скачаного файлу

Для завантаження файлу на комп’ютер іншого клієнту слід натиснути на кнопку «Upload file». З’явиться вікно вибору файлу , що можна побачити на рисунку 3.24.

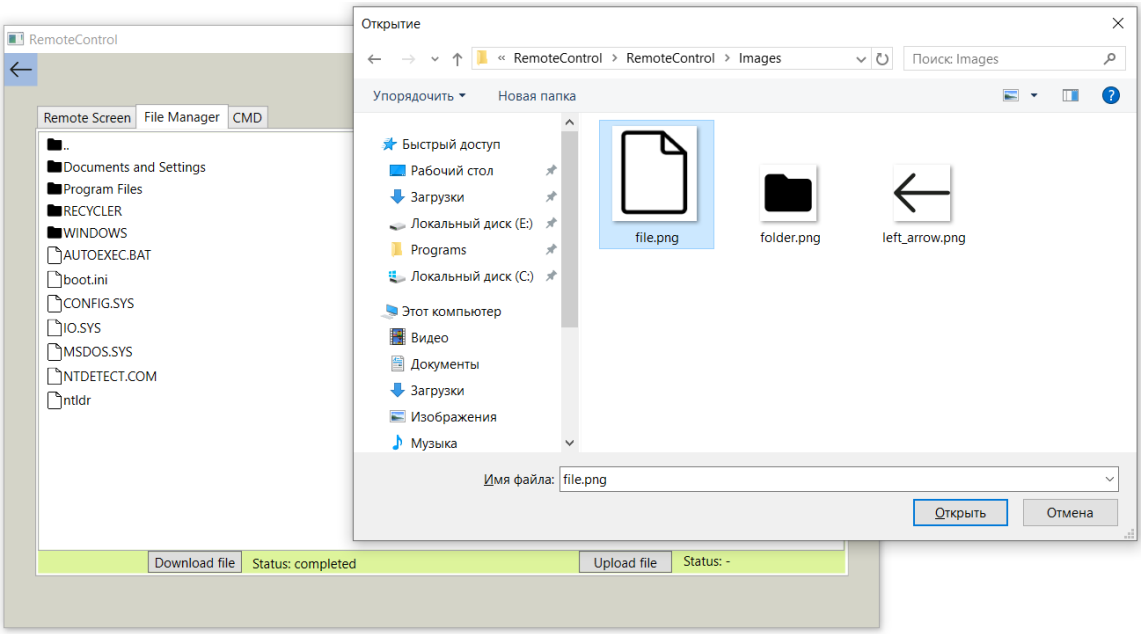

Рисунок 3.24 – Вибір файлу для завантаження

Далі треба обирати файл і по завершенню завантаження можна побачити що файл з’явився в файловому менеджері , а також що статус завантаження файлу змінився на «completed». Це можна побачити на рисунку 3.25.

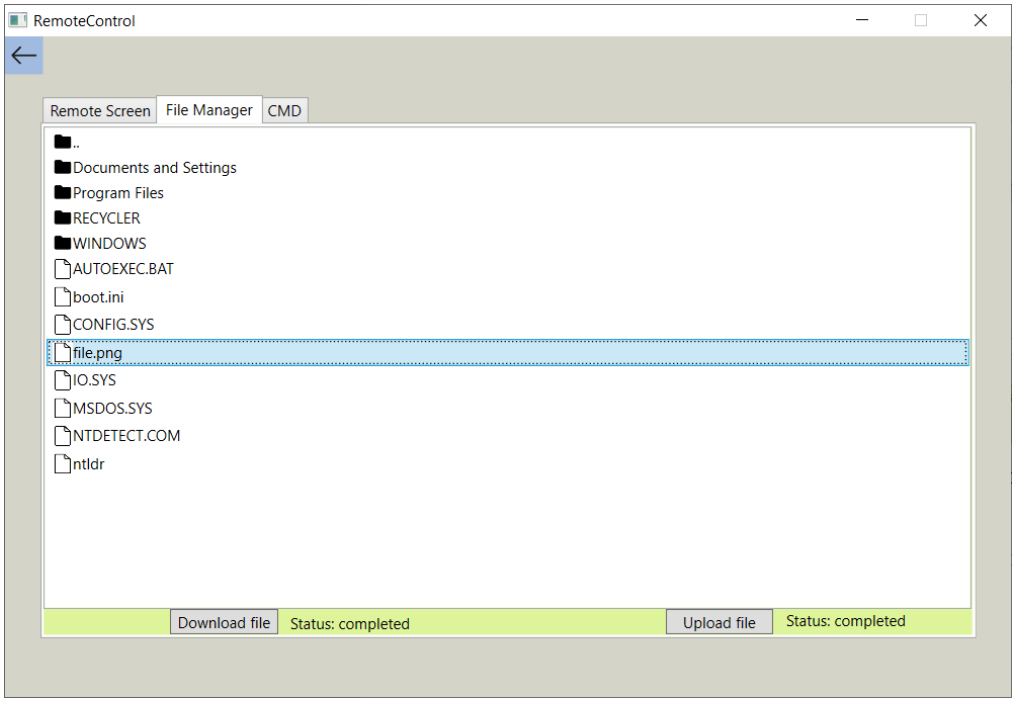

Рисунок 3.25 – Результат завантаження файлу

Щоб протестувати командний рядок необхідно перейти у відповідну вкладку та ввести яку небудь команду. Результат введення команди можна побачити на рисунку 3.26.

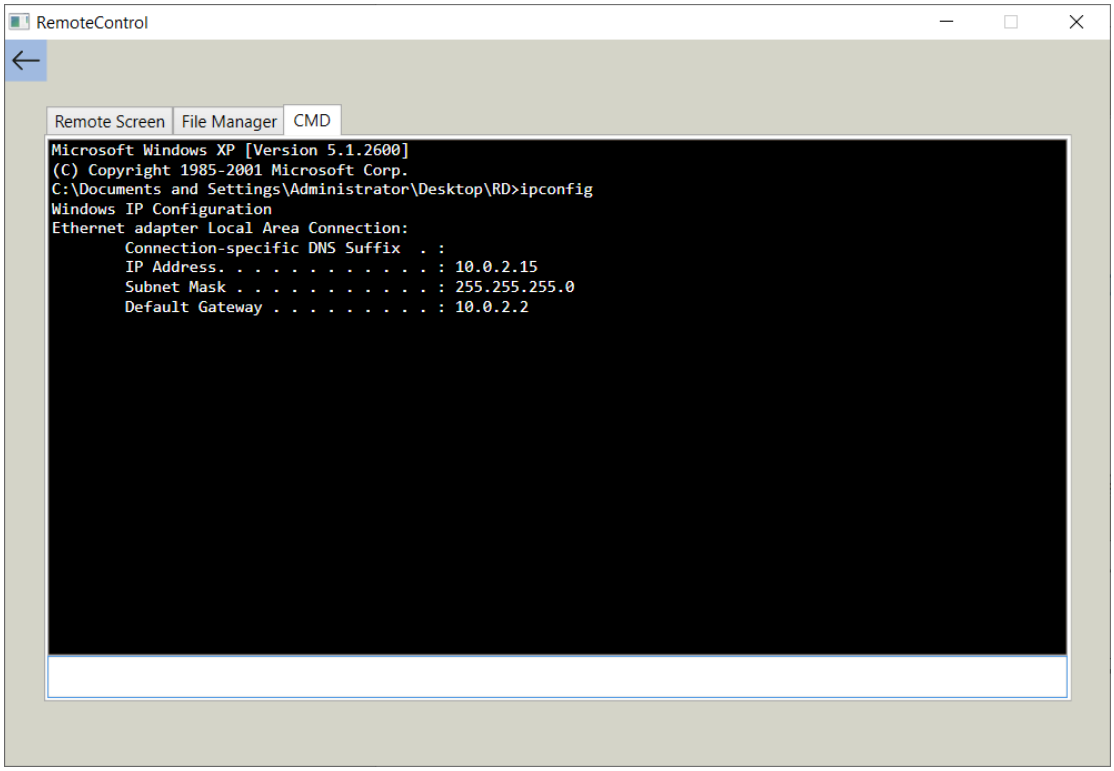

Рисунок 3.26 – Командний рядок в дії

В результаті тестування розробленого програмного забезпечення було перевірено увесь основний функціонал. Можна зробити висновок що всі функції програмного забезпечення працюють як і було заплановано.

ВИСНОВОК

Під час виконання випускної роботи були проаналізовані аналоги розроблюваного програмного забезпечення. На основі здобутої інформації було створено постановку задачі.

Так як для розробки серверу було вирішено використовувати асинхронний фреймворк Tornado то була проведена робота по пошуку та вивченню літератури по асинхронності в мові Python.

В результаті було розроблено сервер та клієнт. Сервер використовується для керування з’єднаннями між клієнтами. Клієнт має графічний інтерфейс для зручного користування. В клієнті були реалізовані такі функції:

- трансляція екрану
- емуляція натискання клавіш клавіатури та миші
- файловий менеджер
- командний рядок

Також було проведено тестування розроблених серверу та клієнту для перевірки працездатності всього функціоналу.

СПИСОК ЛІТЕРАТУРИ

1. Документація WPF [Електронний ресурс] – режим доступу https://docs.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-5.0
2. Асинхронність в Python [Електронний ресурс] – режим доступу https://habr.com/ru/post/421625/
3. Документація Tornado [Електронний ресурс] – режим доступу https://www.tornadoweb.org/en/stable/
4. Марк Лутц. Изучаем Python / Марк Лутц // СПб.: ООО “Диалектика” , 2019. — с. 491-598.
5. Репозиторій бібліотеки WindowsInput [Електронний ресурс] - режим доступу https://github.com/MediatedCommunications/WindowsInput
6. Ногл М. TCP/IP. Иллюстрированный учебник / Ногл М. // «ДМК Пресс», 2003 – с. 0-109.
7. Сайт TeamViewer [Електронний ресурс] – режим доступу https://www.teamviewer.com/
8. Сайт AnyDesk [Електронний ресурс] – режим доступу https://anydesk.com/
9. Сайт UVNC [Електронний ресурс] – режим доступу https://www.uvnc.com/
10. StackOverflow [Електронний ресурс] – режим доступу https://stackoverflow.com/
11. Mark P. Dive into Python 3/ Mark Pilgrim// APRESS ISBN – 2016 – 237 с.
12. Jessica McKellar , Abe Fettig. Twisted Network Programming Essentials / Jessica McKellar , Abe Fettig // O`Reilly Media – 2013 – 163 с.
13. Эндрю Троелсен и Филипп Джепикс. Язык программирования C# 7 и платформы .NET и .NET Core / Эндрю Троелсен, Филипп Джепикс // СПб.: ООО “Диалектика” – 2018 – с. 0-230.
14. Jamie Chan. Learn C# in One Day and Learn It Well / Jamie Chan // CreateSpace Independent Publishing Platform – 2015 – 164 с.
15. Matthew MacDonald. Pro WPF 4.5 in C# / Matthew MacDonald // Apress – 2012 – с. 0-195.

ДОДАТОК

Лістинг коду

Сервер

```
import tornado
from tornado.tcpserver import TCPServer
import struct , asyncio

PacketHeader = struct.Struct("II")
PacketHeaderSize = struct.calcsize("II")

MessageTypes = {
	"ConnectById" : 1,
	"WrongConnectId" : 2,
	"Connected" : 3,

	"CheckConnectId" : 4,
	"CheckResult" : 5,

	"DisconnectMsg":6
}

ConnectIdDict = {}

class TestServer(TCPServer):
	async def handle_stream(self , stream , address):
		try:
			packet_header = await stream.read_bytes(PacketHeaderSize)
			message_type , size = PacketHeader.unpack(packet_header)

			message_data = None
			if size != 0:
				message_data = await stream.read_bytes(size)

			if message_type == MessageTypes["ConnectById"]:
				await self.handle_connect_by_id(stream , message_data)

			elif message_type == MessageTypes["CheckConnectId"]:
				await	self.handle_check_connect_id(stream	,
message_data)

			else:
				stream.close()
				return
		except Exception as e:
			stream.close()
			return

	async def send_message(self , stream , message_type , data = b""):
		packet = PacketHeader.pack(message_type , len(data)) + data
		await stream.write(packet)

	async def handle_connect_by_id(self , stream , message_data):
		connect_id = message_data.decode("utf-8")
		print("connect by id" , connect_id)
		if connect_id in ConnectIdDict and ConnectIdDict[connect_id][2] ==
None:
			ConnectIdDict[connect_id][2] = stream
			endpoint_stream = ConnectIdDict[connect_id][1]
```

```
                ConnectIdDict[connect_id][0].set()

                stream.set_close_callback(self.generate_on_close(connect_id))

                await self.send_message(stream , MessageTypes["Connected"])
                while 1:
                        data = await stream.read_bytes(4096 , True)
                        endpoint_stream.write(data)
        else:
                await                self.send_message(stream                ,
MessageTypes["WrongConnectId"])
                raise

    async def handle_check_connect_id(self , stream , message_data):
        connect_id = message_data.decode("utf-8")
        print("check connect id" , connect_id)

        while connect_id in ConnectIdDict:
                await self.send_message(stream , MessageTypes["CheckResult"] ,
b"\x00")

                packet_header = await stream.read_bytes(PacketHeaderSize)

                message_type , size = PacketHeader.unpack(packet_header)
                if message_type != MessageTypes["CheckConnectId"]:
                        stream.close()
                        return

                message_data = await stream.read_bytes(size)

                connect_id = message_data.decode("utf-8")
        await  self.send_message(stream  ,  MessageTypes["CheckResult"]  ,
b"\x01")
        connect_event = asyncio.Event()
        ConnectIdDict[connect_id] = [connect_event , stream , None]

        done  ,  futures  =  await  asyncio.wait([connect_event.wait()  ,
stream.read_bytes(4096 , True)] , return_when = asyncio.FIRST_COMPLETED)
        for completed in done:
                if               type(completed.exception())               is
tornado.iostream.StreamClosedError:
                        del ConnectIdDict[connect_id]
                        raise Exception("Disconnected")
                elif completed.result() == True:
                        pass
                else:
                        del ConnectIdDict[connect_id]
                        raise Exception("Unknown behavior")

        await self.send_message(stream , MessageTypes["Connected"])
        endpoint_stream = ConnectIdDict[connect_id][2]

        stream.set_close_callback(self.generate_on_close(connect_id))

        done , futures = await asyncio.wait(futures)
        for completed in done:
                await endpoint_stream.write(completed.result())

        while 1:
                data = await stream.read_bytes(4096 , True)
                await endpoint_stream.write(data)
```

```
      def generate_on_close(self , connect_id):
            async def on_close():
                  if connect_id in ConnectIdDict:
                        try:
                              await
self.send_message(ConnectIdDict[connect_id][1] , MessageTypes["DisconnectMsg"])
                        except:
                              pass

                        try:
                              await
self.send_message(ConnectIdDict[connect_id][2] , MessageTypes["DisconnectMsg"])
                        except:
                              pass

                        ConnectIdDict[connect_id][1].close()
                        ConnectIdDict[connect_id][2].close()
                        del ConnectIdDict[connect_id]
                  print("dict" , len(ConnectIdDict))

            return on_close


if __name__ == "__main__":
      tcpserver = TestServer()
      tcpserver.listen(6888)
      tornado.ioloop.IOLoop.current().start()
```

Клієнт

Файл App.xaml.cs

```
using System.Windows;

namespace RemoteControl {
      public partial class App : Application {
            public static string SERVER_IP = "192.168.0.105";
            public static int SERVER_PORT = 6888;
      }
}
```

Файл ConnectToPage.xaml

```
<Page x:Class="RemoteControl.ConnectToPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:RemoteControl"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="ConnectToPage">

    <Grid>
            <Label HorizontalAlignment="Center" Margin="0,100" Foreground="Red"
x:Name="ErrorMessage"></Label>
            <TextBox     x:Name="ConnectIdInput"     HorizontalAlignment="Center"
Height="23" TextAlignment="Center" Text="Client ID" VerticalAlignment="Center"
Width="284" Background="White" BorderBrush="Black" SelectionBrush="#FF2796FF"
Foreground="Black"/>
```

```
            <Button        Content="Button"        HorizontalAlignment="Center"
Margin="0,300,0,0"   VerticalAlignment="Top"   Width="75"   Name="TestPageButton"
Click="ConnectToButtonClick" />
      </Grid>
</Page>
```

Файл ConnectToPage.xaml.cs

```
using System.Windows;
using System.Windows.Controls;
using System.Threading.Tasks;
using System.Threading;

namespace RemoteControl {
      public partial class ConnectToPage : Page {
            MainWindow   MainWindowObj   =   Application.Current.MainWindow   as
MainWindow;

            public ConnectToPage() {
                  InitializeComponent();
            }

        private void ConnectToButtonClick(object sender , RoutedEventArgs e) {
                  MainWindowObj.ShowLoadingAnimation();

                  Task task = new Task(() => {
                        bool                    ConnectResult                    =
Network.ControllerClient.ConnectToServer();

      Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal
, new ThreadStart(() => {
                              ConnectResultHandler(ConnectResult);
                        }));
                  });

                  task.Start();

        }

            private void ConnectResultHandler(bool ConnectResult) {
                  MainWindowObj.HideLoadingAnimation();
                  if (ConnectResult) {
                        ConnectResult                                              =
Network.ControllerClient.ConnectById(ConnectIdInput.Text);
                        if (ConnectResult) {
                              MainWindowObj.NavTo(new RemoteControlPage());

                        } else {
                              ErrorMessage.Content = "Wrong connect id";
                        }
                  } else {
                        ErrorMessage.Content = "Cannot connect to server";
                  }
            }

    }
}
```

Файл MainWindow.xaml

```
<Window
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:RemoteControl"
        x:Class="RemoteControl.MainWindow"
        mc:Ignorable="d"
        Title="MainWindow"   Height="550"   Width="800"   Background="#FFD4D4C8"
Closed="Window_Closed">

      <Window.ResizeMode>
            <ResizeMode>CanMinimize</ResizeMode>
      </Window.ResizeMode>
      <Grid>
            <local:LoadingAnimation   x:Name="LoadingPanel"   Visibility="Hidden"
Panel.ZIndex="100"/>
            <Frame           Source="StartPage.xaml"           Name="PageFrame"
NavigationUIVisibility="Hidden"></Frame>
            <Border     Width="30"     Height="30"     HorizontalAlignment="Left"
VerticalAlignment="Top"                                    Background="#FFA0BAE0"
MouseLeftButtonUp="BackButtonMouseUp">
                  <Label HorizontalAlignment="Center" VerticalAlignment="Center"
FontSize="16">□</Label>
            </Border>
      </Grid>
</Window>
```

Файл MainWindow.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Input;

namespace RemoteControl {
      public partial class MainWindow : Window {

            public MainWindow() {
                  InitializeComponent();
            }

            public void NavTo(object page) {
                  PageFrame.NavigationService.Navigate(page);

            }

            private void PrevPage() {
                  if (PageFrame.Content is RemoteControlPage) {
                        Network.ControllerClient.Disconnect();
                  } else if (PageFrame.Content is RecvConnectionPage) {
                        Network.SubordinateClient.Disconnect();
                  } else if (PageFrame.Content is StartPage) {
                        return;
                  }

                  try {
                        PageFrame.NavigationService.GoBack();
                  } catch(InvalidOperationException) { }
            }


            private void BackButtonMouseUp(object sender , MouseButtonEventArgs
e) {
                  PrevPage();
            }
```

```
            public void ShowLoadingAnimation() {
                  LoadingPanel.Visibility = Visibility.Visible;
            }

            public void HideLoadingAnimation() {
                  LoadingPanel.Visibility = Visibility.Hidden;
            }

        private void Window_Closed(object sender , EventArgs e) {
                  if (PageFrame.Content is RemoteControlPage) {
                        (PageFrame.Content                                    as
RemoteControlPage).RDWindow.Close();
                  }
                  Network.ControllerClient.Disconnect();
                  Network.SubordinateClient.Disconnect();
        }
    }
}
```

Файл RecvConnectionPage.xaml

```
<Page x:Class="RemoteControl.RecvConnectionPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:RemoteControl"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="RecvConnectionPage">

      <Grid>
            <Label Content="Your connection id: " HorizontalAlignment="Center"
Margin="146,221,254,203.6" VerticalAlignment="Center" Width="400"/>
            <Label       x:Name="StatusLabel"       HorizontalAlignment="Center"
VerticalAlignment="Center"     Margin="0,0,0,200"     Foreground="Black">Status:
connecting to server...</Label>
            <TextBox    x:Name="ConnectIdTextBox"    HorizontalAlignment="Center"
Height="23"  Width="400"  Margin="266,222,134,204.6"  TextWrapping="Wrap"  Text=""
VerticalAlignment="Center" TextAlignment="Center" IsReadOnly="True" />
      </Grid>
</Page>
```

Файл RecvConnectionPage.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Threading.Tasks;
using System.Threading;
using System.Windows.Media;
using RemoteControl.Network;
using RemoteControl.Modules;
using System.IO;

namespace RemoteControl {
      public partial class RecvConnectionPage : Page {
            private Task MessageHandlerTask;
            MainWindow   MainWindowObj   =   Application.Current.MainWindow   as
MainWindow;

            public RecvConnectionPage() {
```

```
                InitializeComponent();

                ConnectToServer();
            }

            private void ConnectToServer() {
                Task task = new Task(() => {
                    bool                ConnectResult                =
SubordinateClient.ConnectToServer();

      Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal
, new ThreadStart(() => {
                        ConnectResultHandler(ConnectResult);
                    }));
                });

                task.Start();
            }

            private void ConnectResultHandler(bool ConnectResult) {
                if (ConnectResult) {
                    string uid = Guid.NewGuid().ToString();
                    while (!SubordinateClient.CheckConnectId(uid)) {
                        uid = Guid.NewGuid().ToString();
                    }


                    StatusLabel.Content = "Status: Connected to server ,
waiting for controller";
                    StatusLabel.Foreground = Brushes.LimeGreen;

                    ConnectIdTextBox.Text = uid;
                    MessageHandlerTask = new Task(MessageHandler);
                    MessageHandlerTask.Start();

                } else {
                    StatusLabel.Content = "Status: Cannot connect to
server";
                    StatusLabel.Foreground = Brushes.Red;
                }
            }

            private void MessageHandler() {
                MessageType messageType;
                byte[] data;
                while (true) {
                    try {
                        data = SubordinateClient.Receive(out messageType);
                    } catch (IOException) {
                        SubordinateClient.Disconnect();
                        RemoteDesktop.Stop();

      Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal
, new ThreadStart(() => {
                                    if (!(MainWindowObj.PageFrame.Content is
RecvConnectionPage)) {
                                        return;
                                    }

                                    MessageBox.Show("Controller disconnected",
"Disconnected", MessageBoxButton.OK, MessageBoxImage.Exclamation , 0 ,
MessageBoxOptions.DefaultDesktopOnly);
```

```
                                    StatusLabel.Content = "Status: connecting to
server...";
                                    StatusLabel.Foreground = Brushes.Black;
                                    ConnectToServer();
                              }));
                              return;
                        }


                        switch (messageType) {
                              case MessageType.Connected:

      Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal
, new ThreadStart(() => {
                                          StatusLabel.Content     =     "Status:
controller connected";
                                    }));
                                    break;
                              case MessageType.DisconnectMsg:
                                    SubordinateClient.Disconnect();

      Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal
, new ThreadStart(() => {
                                          MessageBox.Show("Controller
disconnected", "Disconnected", MessageBoxButton.OK, MessageBoxImage.Exclamation
, 0 , MessageBoxOptions.DefaultDesktopOnly);
                                          StatusLabel.Content     =     "Status:
connecting to server...";
                                          StatusLabel.Foreground               =
Brushes.Black;
                                          ConnectToServer();
                                    }));
                                    break;
                              case MessageType.StartRemoteDesktop:
                                    RemoteDesktop.Start();
                                    break;
                              case MessageType.KeyPress:
                                    if (data[0] == 1) {
                                          RemoteDesktop.KeyDown(data[1]);
                                    } else if (data[0] == 2) {
                                          RemoteDesktop.KeyUp(data[1]);
                                    }
                                    break;
                              case MessageType.MousePress:
                                    if (data[0] == 1) {

      RemoteDesktop.MouseButtonDown(data[1]);
                                    } else if (data[0] == 2) {
                                          RemoteDesktop.MouseButtonUp(data[1]);
                                    }
                                    break;
                              case MessageType.MouseMove:
                                    float x = BitConverter.ToSingle(data , 0);
                                    float y = BitConverter.ToSingle(data , 4);
                                    RemoteDesktop.MouseMove(x , y);
                                    break;
                              case MessageType.StartCMD:
                                    CMD.Start();
                                    break;
                              case MessageType.StopCMD:
                                    CMD.Stop();
                                    break;
```

```
                                case MessageType.CMDInput:
                                      CMD.WriteToCmd(data);
                                      break;

                                }

                          }
                    }

      }
}
```

Файл RemoteControlPage.xaml

```
<Page x:Class="RemoteControl.RemoteControlPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:RemoteControl"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="RemoteControlPage">

      <Grid>
            <TabControl         HorizontalAlignment="Center"         Height="430"
Margin="0,0,0,0"              VerticalAlignment="Center"              Width="740"
RenderTransformOrigin="0.575,0.407"
SelectionChanged="TabControl_SelectionChanged">
                  <TabItem Header="Remote Screen">
                        <Grid Background="#FFBDE0FF">
                              <Button           Width="170"           Height="20"
Click="StartRemoteDesktopClick">Open remote control window</Button>
                        </Grid>
                  </TabItem>
                  <TabItem Header="File Manager">
                        <Grid Background="#FFDDF49B">
                        </Grid>
                  </TabItem>
                  <TabItem x:Name="CMDTab" Header="CMD">
                        <Grid Background="#FFF4CE9B">
                              <TextBox  x:Name="CMDoutput"  AcceptsReturn="True"
Margin="0,0,0,30"              Background="Black"              Foreground="White"
FontFamily="Consolas"></TextBox>
                              <TextBox   Height="30"   VerticalAlignment="Bottom"
Background="White"        KeyDown="CMDInputKeyDown"        FontFamily="Consolas"
FontSize="20"></TextBox>
                        </Grid>
                  </TabItem>
            </TabControl>

      </Grid>
</Page>
```

Файл RemoteControlPage.xaml.cs

```
using System.IO;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using RemoteControl.Network;
using System.Threading;
```

```
using System.Threading.Tasks;

namespace RemoteControl {
      public partial class RemoteControlPage : Page {
            private Task MessageHandlerTask;
            private MainWindow MainWindowObj = Application.Current.MainWindow as
MainWindow;
            public RemoteDesktopWindow RDWindow = new RemoteDesktopWindow();

            public RemoteControlPage() {
                  InitializeComponent();
                  MessageHandlerTask = new Task(MessageHandler);
                  MessageHandlerTask.Start();

            }

            private void StartRemoteDesktopClick(object sender , RoutedEventArgs
e) {
                  ControllerClient.Send(MessageType.StartRemoteDesktop , null);
                  RDWindow.Show();
            }

            private void MessageHandler() {
                  MessageType messageType;
                  byte[] data;
                  while (true) {
                        try {
                              data = ControllerClient.Receive(out messageType);

                        } catch (IOException) {
                              ControllerClient.Disconnect();

      Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal
, new ThreadStart(() => {
                                    MessageBox.Show("Disconnected from client",
"Disconnected",   MessageBoxButton.OK,   MessageBoxImage.Exclamation   ,   0   ,
MessageBoxOptions.DefaultDesktopOnly);
                                    MainWindowObj.NavTo(new StartPage());
                              }));
                              return;
                        }

                        switch (messageType) {
                              case MessageType.RemoteDesktopFrame:
                                    RDWindow.setImage(data);
                                    break;
                              case MessageType.CMDOutput:
                                    string str = Encoding.UTF8.GetString(data) +
"\n";

      Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal
, new ThreadStart(() => {
                                          CMDoutput.Text += str;
                                    }));
                                    break;
                              case MessageType.DisconnectMsg:
                                    ControllerClient.Disconnect();

      Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal
, new ThreadStart(() => {
                                    MessageBox.Show("Disconnected     from
client", "Disconnected", MessageBoxButton.OK, MessageBoxImage.Exclamation);
```

```
                                        MainWindowObj.NavTo(new StartPage());
                                  }));
                                  break;

                        }

                  }
            }

            private    void    TabControl_SelectionChanged(object    sender    ,
SelectionChangedEventArgs e) {
                  if (CMDTab.IsSelected) {
                        ControllerClient.Send(MessageType.StartCMD , null);
                  }

            }

            private void CMDInputKeyDown(object sender , KeyEventArgs e) {
                  if (e.Key == Key.Enter) {
                        TextBox input = sender as TextBox;
                        byte[] data = Encoding.UTF8.GetBytes(input.Text);
                        ControllerClient.Send(MessageType.CMDInput , data);
                        input.Text = "";
                  }
            }
      }
}
```

Файл RemoteDesktopWindow.xaml

```
<Window x:Class="RemoteControl.RemoteDesktopWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:RemoteControl"
        mc:Ignorable="d"
        Title="RemoteDesktopWindow"           Height="450"           Width="800"
KeyDown="Window_KeyDown" KeyUp="Window_KeyUp" Closed="Window_Closed">
    <Grid>
            <Image              x:Name="DesktopImage"              Stretch="Fill"
MouseMove="DesktopImage_MouseMove"            MouseDown="DesktopImage_MouseDown"
MouseUp="DesktopImage_MouseUp"></Image>
    </Grid>
</Window>
```

Файл RemoteDesktopWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media.Imaging;
using RemoteControl.Network;

namespace RemoteControl {
      public partial class RemoteDesktopWindow : Window {
            int DesktopWidth = 0;
            int DesktopHeight = 0;

            public RemoteDesktopWindow() {
```

```
                InitializeComponent();
            }

            public void setImage(byte[] jpeg) {

      Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal
, new ThreadStart(() => {
                        MemoryStream memoryStream = new MemoryStream(jpeg);
                        BitmapImage bitmapImage = new BitmapImage();
                        bitmapImage.BeginInit();
                        bitmapImage.StreamSource = memoryStream;
                        bitmapImage.EndInit();
                        DesktopWidth = bitmapImage.PixelWidth;
                        DesktopHeight = bitmapImage.PixelHeight;

                        DesktopImage.Source = bitmapImage;
                }));

            }

            private void Window_KeyDown(object sender , KeyEventArgs e) {
                  e.Handled = true;
                  byte[] data = new byte[2];
                  int keyCode = KeyInterop.VirtualKeyFromKey(e.Key);
                  data[0] = 1;
                  data[1] = (byte) keyCode;
                  ControllerClient.Send(MessageType.KeyPress , data);
            }

            private void Window_KeyUp(object sender , KeyEventArgs e) {
                  e.Handled = true;
                  byte[] data = new byte[2];
                  int keyCode = KeyInterop.VirtualKeyFromKey(e.Key);
                  data[0] = 2;
                  data[1] = (byte) keyCode;
                  ControllerClient.Send(MessageType.KeyPress , data);
            }

            private void DesktopImage_MouseMove(object sender , MouseEventArgs
e) {
                  Point point = e.GetPosition(DesktopImage);
                  float      normalizedX      =      (float)      (point.X      /
DesktopImage.ActualWidth);
                  float      normalizedY      =      (float)      (point.Y      /
DesktopImage.ActualHeight);
                  byte[] x = BitConverter.GetBytes(normalizedX);
                  byte[] y = BitConverter.GetBytes(normalizedY);
                  byte[] data = new byte[8];
                  x.CopyTo(data , 0);
                  y.CopyTo(data , 4);
                  ControllerClient.Send(MessageType.MouseMove , data);
            }

            private     void     DesktopImage_MouseDown(object     sender     ,
MouseButtonEventArgs e) {
                  byte[] data = new byte[2];
                  data[0] = 1;
                  data[1] = (byte) (e.ChangedButton == MouseButton.Right ? 2 :
1);
                  ControllerClient.Send(MessageType.MousePress , data);
            }
```

```
            private      void      DesktopImage_MouseUp(object      sender      ,
MouseButtonEventArgs e) {
                  byte[] data = new byte[2];
                  data[0] = 2;
                  data[1] = (byte) (e.ChangedButton == MouseButton.Right ? 2 :
1);
                  ControllerClient.Send(MessageType.MousePress , data);
            }

            private void Window_Closed(object sender , EventArgs e) {
                  ControllerClient.Send(MessageType.StopRemoteDesktop , null);
            }
      }
}
```

Файл StartPage.xaml

```
<Page x:Class="RemoteControl.StartPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:local="clr-namespace:RemoteControl"
      mc:Ignorable="d"
      d:DesignHeight="450" d:DesignWidth="800"
      Title="StartPage">

      <Grid>
            <Border      MouseDown="ConnectToMouseDown"      BorderBrush="Black"
BorderThickness="1"     HorizontalAlignment="Left"     VerticalAlignment="Center"
Height="150"      Margin="150,0,0,0"      Width="150"      Background="#FFB7FF72"
CornerRadius="12">
                  <Label                              HorizontalAlignment="Center"
VerticalAlignment="Center">Connect to</Label>
            </Border>
            <Border    MouseDown="RecvConnectionMouseDown"    BorderBrush="Black"
BorderThickness="1"     HorizontalAlignment="Left"     VerticalAlignment="Center"
Height="150"      Margin="500,0,0,0"      Width="150"      Background="#FFB7FF72"
CornerRadius="12">
                  <Label                              HorizontalAlignment="Center"
VerticalAlignment="Center">Recieve connection</Label>
            </Border>
      </Grid>
</Page>
```

Файл StartPage.xaml.cs

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;

namespace RemoteControl {
      public partial class StartPage : Page {
            MainWindow   MainWindowObj   =   Application.Current.MainWindow   as
MainWindow;

            public StartPage() {
                  InitializeComponent();
            }

            private void ConnectToMouseDown(object sender , MouseButtonEventArgs
e) {
                  MainWindowObj.NavTo(new ConnectToPage());
```

```
            }

            private     void     RecvConnectionMouseDown(object     sender     ,
MouseButtonEventArgs e) {
                  MainWindowObj.NavTo(new RecvConnectionPage());
            }
      }
}
```

Файл Modules/CMD.cs

```
using System;
using System.Diagnostics;
using System.Text;
using RemoteControl.Network;

namespace RemoteControl.Modules {
      class CMD {
            private static Process cmdProcess;
            private static bool startFlag = false;

            public static void Start() {
                  startFlag = true;
                  cmdProcess = new Process();
                  cmdProcess.StartInfo.FileName = "cmd.exe";
                  cmdProcess.StartInfo.CreateNoWindow = true;
                  cmdProcess.StartInfo.UseShellExecute = false;
                  cmdProcess.StartInfo.RedirectStandardOutput = true;
                  cmdProcess.StartInfo.RedirectStandardInput = true;
                  cmdProcess.StartInfo.RedirectStandardError = true;
                  cmdProcess.OutputDataReceived              +=              new
DataReceivedEventHandler(CmdOutputDataHandler);
                  cmdProcess.Start();
                  cmdProcess.BeginOutputReadLine();
            }

            public static void Stop() {
                  startFlag = false;
                  cmdProcess.Kill();
            }

            private  static  void  CmdOutputDataHandler(object  sendingProcess,
DataReceivedEventArgs ev) {
                  if (!startFlag) return;
                  if (!String.IsNullOrEmpty(ev.Data)) {
                        try {
                              byte[] data = Encoding.UTF8.GetBytes(ev.Data);
                              SubordinateClient.Send(MessageType.CMDOutput     ,
data);
                        } catch (Exception err) { }
                  }
            }

            public static void WriteToCmd(byte[] data) {
                  if (!startFlag) return;
                  string strData = Encoding.UTF8.GetString(data);
                  cmdProcess.StandardInput.WriteLine(strData);
            }

      }
}
```

Файл Modules/RemoteDesktop.cs

```
using System.Threading.Tasks;
using System.Threading;
using System.Windows.Forms;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using RemoteControl.Network;
using WindowsInput;
using WindowsInput.Native;


namespace RemoteControl.Modules {
      class RemoteDesktop {
            private static Task CaptureDesktopTask;
            private static bool CapturingFlag = false;
            private static InputSimulator InputSim= new InputSimulator();

            public static void Start() {
                  CapturingFlag = true;
                  CaptureDesktopTask = new Task(() => {
                        Rectangle bounds = Screen.GetBounds(Point.Empty);
                        Bitmap bitmap = new Bitmap(bounds.Width, bounds.Height);

                        while (CapturingFlag) {
                              Graphics graphics = Graphics.FromImage(bitmap);
                              graphics.CopyFromScreen(Point.Empty,  Point.Empty,
bounds.Size);

                              MemoryStream ms = new MemoryStream();

                              ImageCodecInfo jpgEncoder = null;
                              ImageCodecInfo[]             codecs             =
ImageCodecInfo.GetImageEncoders();
                              foreach (ImageCodecInfo codec in codecs) {
                                    if (codec.FormatID == ImageFormat.Jpeg.Guid)
{
                                          jpgEncoder = codec;
                                    }
                              }

                              Encoder myEncoder = Encoder.Quality;

                              EncoderParameters   myEncoderParameters   =   new
EncoderParameters(1);

                              EncoderParameter    myEncoderParameter    =    new
EncoderParameter(myEncoder , 35L);
                              myEncoderParameters.Param[0] = myEncoderParameter;

                              bitmap.Save(ms        ,        jpgEncoder        ,
myEncoderParameters);

                              byte[] imgBytes = ms.ToArray();

      SubordinateClient.Send(MessageType.RemoteDesktopFrame , imgBytes);
                              Thread.Sleep(250);
                        }

                  });

                  CaptureDesktopTask.Start();
```

```
            }

            public static void Stop() {
                  CapturingFlag = false;
            }

            public static void KeyDown(byte keyCode) {
                  InputSim.Keyboard.KeyDown((VirtualKeyCode) keyCode);
            }

            public static void KeyUp(byte keyCode) {
                  InputSim.Keyboard.KeyUp((VirtualKeyCode) keyCode);
            }

            public static void MouseMove(float x , float y) {
                  InputSim.Mouse.MoveMouseTo(x * 65535 , y * 65535);
            }

            public static void MouseButtonDown(byte button) {
                  if (button == 1) InputSim.Mouse.LeftButtonDown();
                  else if (button == 2) InputSim.Mouse.RightButtonDown();
            }

            public static void MouseButtonUp(byte button) {
                  if (button == 1) InputSim.Mouse.LeftButtonUp();
                  else if (button == 2) InputSim.Mouse.RightButtonUp();
            }

      }
}
```

Файл Network/ClientTemplate.cs

```
namespace RemoteControl.Network {
      abstract class ClientTemplate {
            protected static TcpManager tcpManager = new TcpManager();

            public static bool ConnectToServer() {
                  return       tcpManager.ConnectToServer(App.SERVER_IP       ,
App.SERVER_PORT);
            }

            public static byte[] Receive(out MessageType messageType) {
                  return tcpManager.Receive(out messageType);
            }

            public static bool Send(MessageType messageType , byte[] data) {
                  return tcpManager.Send(messageType , data);
            }

            public static void Disconnect() {
                  tcpManager.Disconnect();
            }

      }
}
```

Файл Network/ControllerClient.cs

```
using System.Text;

namespace RemoteControl.Network {
      class ControllerClient : ClientTemplate {
```

```
            public static bool ConnectById(string connectId) {
                  byte[] connectIdBytes = Encoding.ASCII.GetBytes(connectId);
                  tcpManager.Send(MessageType.ConnectById , connectIdBytes);
                  MessageType messageType;
                  tcpManager.Receive(out messageType);

                  if (messageType == MessageType.Connected) {
                        return true;
                  }

                  return false;
            }

      }
}
```

Файл Network/MessageType.cs

```
namespace RemoteControl.Network {
      enum MessageType {
            ConnectById = 1,
            WrongConnectId,
            Connected,
            CheckConnectId,
            CheckResult,
            DisconnectMsg,

            StartRemoteDesktop,
            StopRemoteDesktop,
            RemoteDesktopFrame,
            KeyPress,
            MousePress,
            MouseMove,
            StartCMD,
            StopCMD,
            CMDInput,
            CMDOutput
      }

      static class MessageTypeMethods {
            public static uint ToUInt32(this MessageType mType) {
                  return (uint) mType;
            }
      }
}
```

Файл Network/SubordinateClient.cs

```
using System;
using System.Text;

namespace RemoteControl.Network {
      class SubordinateClient : ClientTemplate {
            public static bool CheckConnectId(string connectId) {
                  byte[] connectIdBytes = Encoding.ASCII.GetBytes(connectId);
                  tcpManager.Send(MessageType.CheckConnectId , connectIdBytes);
                  MessageType messageType;
                  byte[] data = tcpManager.Receive(out messageType);
                  if (messageType != MessageType.CheckResult) throw new
Exception("CheckId result wrong message");
                  return data[0] == 1;
            }
```

```
	}
}
```

Файл Network/TcpManager.cs

```
using System;
using System.Net.Sockets;

namespace RemoteControl.Network {
	class TcpManager {
		private TcpClient tcpClient = new TcpClient();
		private NetworkStream ioStream;
		private const int HeaderLength = 8;

		public bool ConnectToServer(string ip , int port) {
			try {
				if (tcpClient.Connected) {
					tcpClient.Close();
				}
				tcpClient = new TcpClient();
				tcpClient.Connect(ip , port);
				ioStream = tcpClient.GetStream();
				return true;
			} catch (System.IO.IOException) {
				return false;
			}
		}

		public bool Send(MessageType mType , byte[] data) {
			if (ioStream == null) return false;

			try {
				uint size = (uint) (data != null ? data.Length : 0);
				byte[] packetHeader = CreatePacketHeader(mType , size);
				ioStream.Write(packetHeader , 0 , packetHeader.Length);
				if (size != 0) ioStream.Write(data , 0 , data.Length);

				return true;
			} catch (System.IO.IOException) {
				return false;
			}
		}

		public byte[] Receive(out MessageType mType) {
			byte[] packetHeader = ReceiveBytes(HeaderLength);
			mType = (MessageType) BitConverter.ToUInt32(packetHeader , 0);
			uint size = BitConverter.ToUInt32(packetHeader , 4);

			byte[] data = null;
			if (size != 0) data = ReceiveBytes(size);
			return data;
		}

		private byte[] ReceiveBytes(uint byteAmount) {
			int readed = 0;
			byte[] data = new byte[byteAmount];
			do {
				readed += ioStream.Read(data , readed , (int)byteAmount
- readed);
			} while (readed < byteAmount);
```

```
                return data;
            }

            private byte[] CreatePacketHeader(MessageType mType , uint size) {
                byte[] header = new byte[HeaderLength];

                byte[] mTypeBytes = BitConverter.GetBytes(mType.ToUInt32());
                Buffer.BlockCopy(mTypeBytes , 0 , header , 0 , 4);

                byte[] sizeBytes = BitConverter.GetBytes(size);
                Buffer.BlockCopy(sizeBytes , 0 , header , 4 , 4);

                return header;
            }

            public void Disconnect() {
                tcpClient.Close();
            }

        }
    }
```