

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



COMPUTER NETWORK

COMPUTER REMOTE CONTROL VIA EMAIL

Nhóm 3

GVHD	Đỗ Hoàng Cường Huỳnh Thụy Bảo Trân
23122009	Bàng Mỹ Linh
23122040	Nguyễn Thị Mỹ Kim

TP. HỒ CHÍ MINH, THÁNG 12/2024

Mục lục

1	Giao diện	3
1.1	Welcome Screen	3
1.2	Menu	3
1.3	Login/Register Form	4
1.4	About	5
1.5	Help	5
1.6	Working Screen	6
2	Công cụ	7
2.1	Các thư viện	7
2.1.1	EASendMail và EAGetMail	7
2.1.2	Qt	7
2.1.3	FFmpeg	8
2.2	Mô hình quá trình kết nối User - Client - Server	9
3	Giải thích mã nguồn	9
3.1	Client side	9
3.1.1	Class ClientService - Quản lý Request và Email	9
3.1.2	Class LoginDialog - Đăng nhập và xử lý xác thực người dùng	11
3.1.3	Class RegisterDialog - Đăng ký tài khoản người dùng	12
3.1.4	Class MainWindow - Giao diện chính quản lý dịch vụ Remote Control	12
3.1.5	Một số Class xử lý giao diện khác	13
3.2	Server side	14
3.2.1	Quản lý Application trong máy tính Server	14
3.2.2	Quản lý Service trong máy tính Server	16
3.2.3	File - Quản lý file trong máy tính Server	16
3.2.4	Screenshot - Chụp ảnh màn hình máy tính	17
3.2.5	Record Video - Quay màn hình Webcam máy tính	17
3.2.6	ShutDown - Tắt máy tính	18



Báo cáo Hoạt động Nhóm

Phân công nhiệm vụ

- **23122009 Bàn Mỹ Linh:** Phụ trách viết mã nguồn liên quan đến giao tiếp email giữa Client và User, kết nối Client với Server, chức năng Quản lý File, thiết kế giao diện.
- **23122040 Nguyễn Thị Mỹ Kim:** Phụ trách viết mã nguồn toàn bộ chức năng còn lại của Server, tìm kiếm và cài đặt các thư viện cần thiết cho đồ án.

Quá trình làm việc nhóm

- **Giai đoạn 1:** Nghiên cứu yêu cầu dự án và phân tích công nghệ. Cả nhóm đã cùng thảo luận và chọn các công cụ phù hợp như Qt, WinSock.
- **Giai đoạn 2:** Phân công nhiệm vụ và triển khai. Hai thành viên làm việc song song, thường xuyên họp nhóm qua Google Meet để trao đổi tiến độ.
- **Giai đoạn 3:** Kiểm thử và hoàn thiện. Cả hai tiến hành kiểm thử từng chức năng, sửa lỗi và tối ưu.

Khó khăn và giải pháp

- **Khó khăn:** Thiếu kinh nghiệm làm việc với thư viện xử lý email, sự cố khi cài đặt thư viện, xảy ra lỗi khi chạy source code trên máy khác.
- **Giải pháp:** Tìm kiếm tài liệu hướng dẫn trên Internet, thảo luận trực tiếp với nhau.

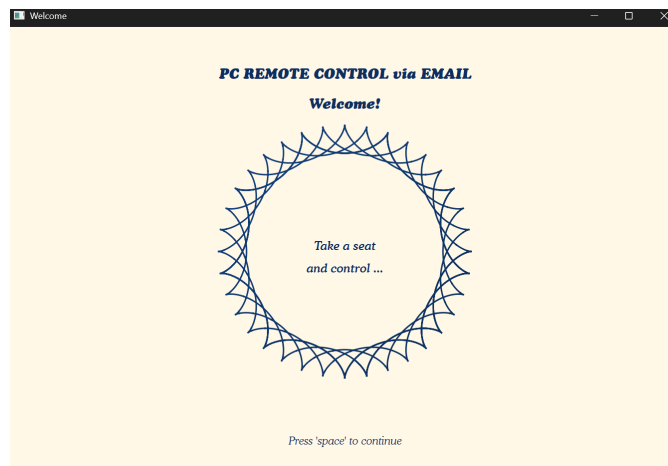
Kết quả đạt được

Nhóm đã hoàn thành đúng tiến độ với các chức năng hoạt động ổn định:

- Gửi và nhận email giữa Client và User.
- Điều khiển được Server từ xa qua email, thực hiện được các chức năng theo yêu cầu đồ án.

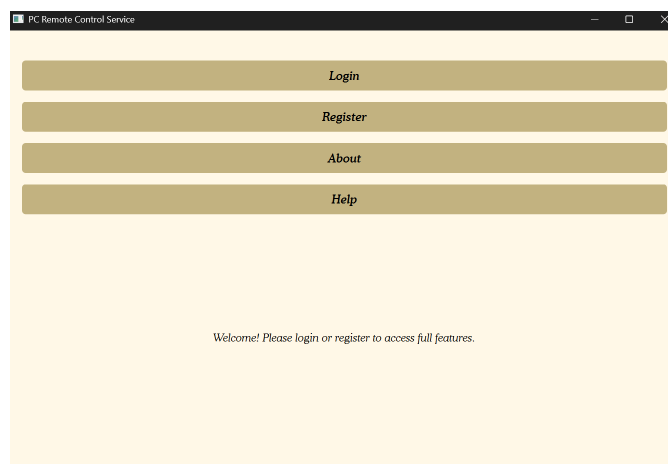
1 Giao diện

1.1 Welcome Screen



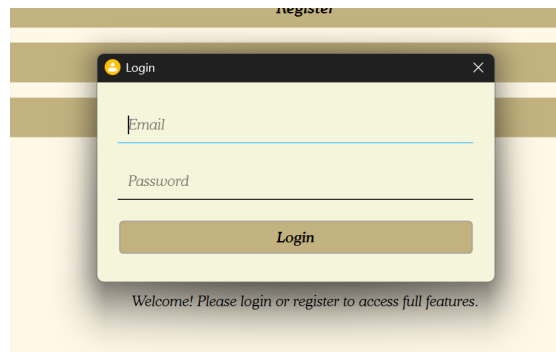
Hình 1: Welcome Screen

1.2 Menu



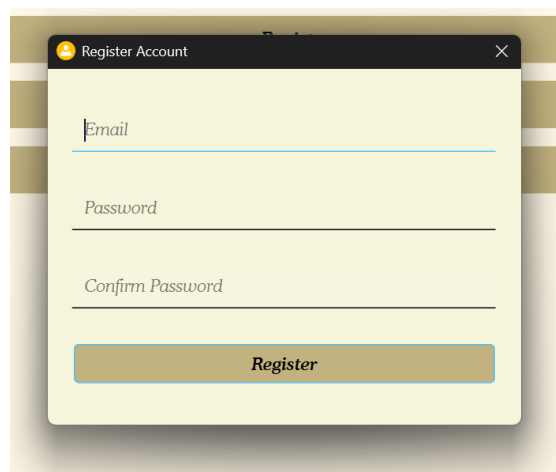
Hình 2: Main Menu

1.3 Login/Register Form



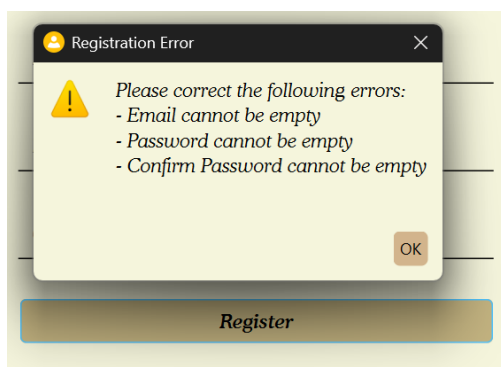
The image shows a 'Login' dialog box with a title bar containing a user icon and a close button. The dialog has two input fields labeled 'Email' and 'Password'. Below these fields is a 'Login' button. At the bottom of the dialog, there is a message: 'Welcome! Please login or register to access full features.'

Hình 3: Login Form



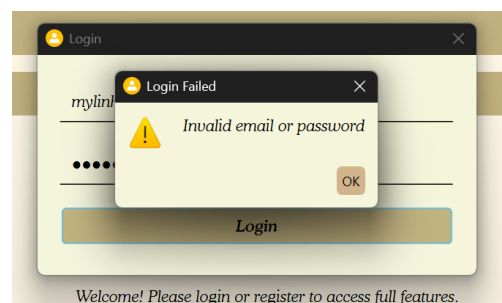
The image shows a 'Register Account' dialog box with a title bar containing a user icon and a close button. The dialog has three input fields labeled 'Email', 'Password', and 'Confirm Password'. Below these fields is a 'Register' button.

Hình 4: Register Form



The image shows a 'Registration Error' dialog box with a title bar containing a user icon and a close button. The dialog contains a yellow warning icon and the following text: 'Please correct the following errors:' followed by a list of errors: '- Email cannot be empty', '- Password cannot be empty', and '- Confirm Password cannot be empty'. There is an 'OK' button at the bottom right of the dialog.

(a) Hộp thoại báo lỗi đăng ký

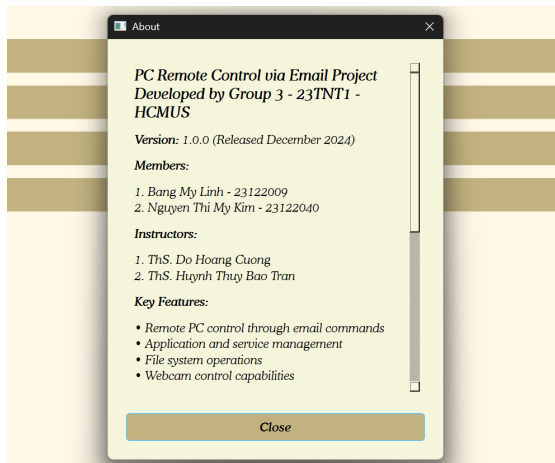


The image shows a 'Login Failed' dialog box with a title bar containing a user icon and a close button. The dialog contains a yellow warning icon and the text: 'Invalid email or password'. There is an 'OK' button at the bottom right of the dialog. In the background, a 'Login' dialog box is visible with the same 'Welcome! Please login or register to access full features.' message.

(b) Hộp thoại báo lỗi đăng nhập



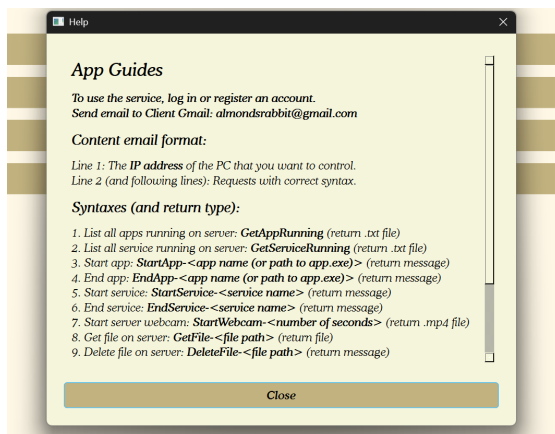
1.4 About



Hình 6: About Box

1.5 Help

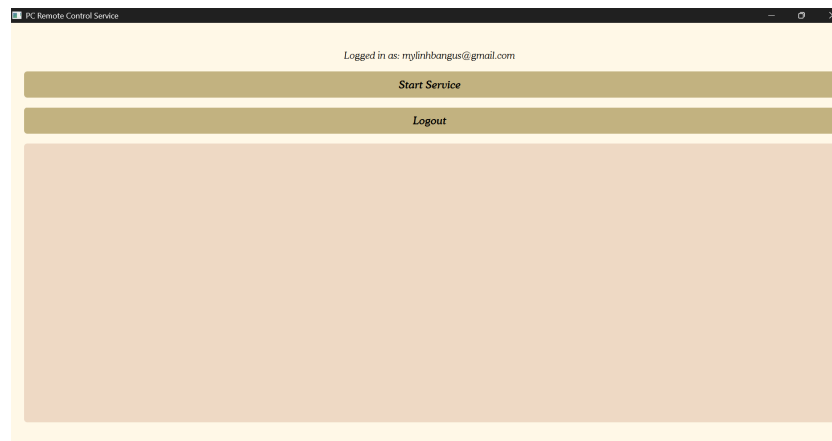
Help Box cung cấp địa chỉ gmail của client các cú pháp gửi mail.



Hình 7: Help Box

1.6 Working Screen

- Màn hình làm việc theo dõi quá trình thực thi request từ user và hiện lên tracking log.
- Có thể Stop/Start Service nếu cần.



Hình 8: Working Screen

Màn hình sau khi thực thi request:



Hình 9: Working Screen



2 Công cụ

2.1 Các thư viện

2.1.1 EASendMail và EAGetMail

Nhóm dùng thư viện EASendMail, EAGetMail để xử lý các tác vụ liên quan đến email phía Client, cụ thể:

- EASendMail hỗ trợ các giao thức và tính năng gửi email qua các máy chủ SMTP.
- EAGetMail hỗ trợ nhận và quản lý email từ các máy chủ email sử dụng giao thức POP3, IMAP hoặc EWS (Exchange Web Services).

Cài đặt

1. Tải EAGetMail và EASendMail Installer

- Vào link: [Download](#)
- Chọn tải EASendMail SMTP Component và EAGetMail POP3/IMAP4 Component

2. Sau khi cài đặt, di chuyển vào thư mục /EASendMail/Include/tlh

3. Copy 2 tệp easendmailobj.tlh và easendmailobj.tli vào project folder.

4. Di chuyển vào thư mục /EAGetMail/Include/tlh

5. Copy 2 tệp EAGetMailobj.tlh và EAGetMailobj.tli vào project folder.

2.1.2 Qt

Qt là một framework hỗ trợ tạo giao diện người dùng đồ họa (Graphical User Interface - GUI), cung cấp các module có sẵn để xử lý các tác vụ đồ họa. Nhóm sử dụng Qt để phát triển giao diện bên phía Client.

Cài đặt mã nguồn Qt vào máy tính

1. Tạo folder Qt tại ổ C với đường dẫn như sau: C:\Qt
2. Vào link: [ZIP File](#) để tải file 6.8.1.zip về máy, đặt 6.8.1.zip trong C:\Qt và thực hiện giải nén tại thư mục này.
3. Kiểm tra đường dẫn cuối cùng phải là: C:\Qt\6.8.1\msvc2022_64, trong thư mục msvc2022_64 là các thư mục chứa mã nguồn của Qt

Cài đặt Qt Extensions trên Visual Studio

1. Mở Solution clientapp.sln trên Visual Studio.
2. Chọn Extension trên thanh công cụ, sau đó chọn Manage Extension
3. Trang Extension Manager sẽ hiện lên, trong mục Search, nhập "Qt", chọn Install extension Qt Vs CMake Tools và chọn Install extension LEGACY Qt Visual Studio Tools
4. Sau khi cài đặt xong, tắt tất cả cửa sổ Visual Studio. Sau đó, cửa sổ VSIX Installer sẽ hiện lên, chọn Modify, End các tasks VSIX Install yêu cầu để hệ thống thực hiện Modifications xong thì chọn Close.



5. Khởi động lại máy

Sau các bước trên, ta thêm Qt Versions vào Qt Extension:

1. Mở Solution `clientapp.sln` trên Visual Studio.
2. Chọn Extension trên thanh công cụ, trỏ con trỏ chuột vào Qt VS Tools và chọn Qt Versions
3. Cửa sổ Options cho Qt Versions hiện lên, chọn Biểu tượng dấu "+", sau đó chọn Biểu tượng "thư mục" - Path.
4. Cửa sổ Qt VS Tools - Select qmake sẽ hiện lên, chọn file `qmake.exe` ở đường dẫn `C:\Qt\6.8.1\msvc2022_64\bin\qmake.exe`, chọn Open.
5. Màn hình tự động quay trở lại cửa sổ Options, chọn OK

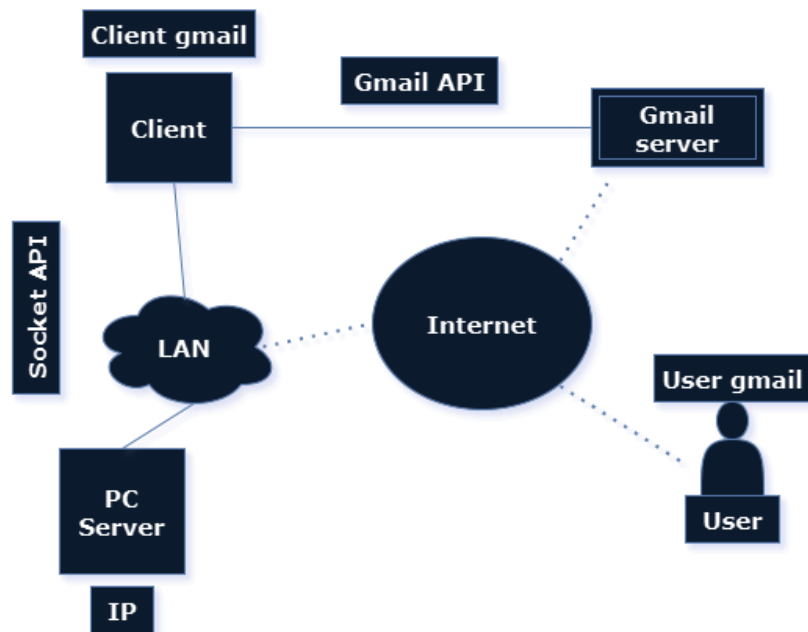
2.1.3 FFMpeg

FFmpeg là mã nguồn mở được sử dụng để xử lý các tác vụ liên quan đến video, âm thanh, ... Nhóm sử dụng FFMpeg để quay màn hình Webcam bên phía Server.

Cài đặt

1. Tạo folder `ffmpeg` tại ổ C với đường dẫn như sau: `C:\ffmpeg`
2. Tải FFMpeg phiên bản cho Window: [ZIP File](#)
3. Đưa file `ffmpeg-master-latest-win64-gpl.zip` vào đường dẫn `C:\ffmpeg` và thực hiện giải nén
4. Vào `C:\ffmpeg\ffmpeg-master-latest-win64-gpl\bin` di chuyển 3 file sau vào `C:\ffmpeg` gồm:
 - `ffmpeg.exe`
 - `ffplay.exe`
 - `ffprobe.exe`
5. Nhập và chọn "Edit enviroment variables for your account" trên thanh tìm kiếm của Window.
6. Cửa sổ Enviroment Variables sẽ hiện lên, trong mục User variables for Administrator_Name, chọn Variables là Path, sau đó chọn Edit.
7. Cửa sổ Edit enviroment variable sẽ hiện lên, chọn New, sau đó nhập "`C:\ffmpeg`", sau đó chọn OK.
8. Màn hình tự động quay lại cửa sổ Enviroment Variables, chọn OK.

2.2 Mô hình quá trình kết nối User - Client - Server



Hình 10: Sơ đồ hoạt động

3 Giải thích mã nguồn

3.1 Client side

3.1.1 Class ClientService - Quản lý Request và Email

Các hàm xử lý email

1. Hàm `split`: Tách chuỗi đầu vào thành một vector các chuỗi con dựa trên ký tự phân tách.

```
vector<string> split(const string& str, char delimiter)
```

Tham số:

- `str`: Chuỗi đầu vào cần được tách.
 - `delimiter`: Ký tự phân tách dùng để chia chuỗi.
2. Hàm `_getCurrentPath`: Lấy đường dẫn hiện tại của tệp thực thi và trả về chiều dài đường dẫn.

```
DWORD _getCurrentPath(LPTSTR lpPath, DWORD nSize)
```



Tham số:

- **lpPath**: Con trỏ đến mảng ký tự nơi lưu trữ đường dẫn của tệp thực thi.
 - **nSize**: Kích thước tối đa của mảng **lpPath**.
3. Hàm **parseEmail**: Đọc nội dung Email và trích xuất thông tin người gửi, địa chỉ IP và các requests.

```
vector<string> parseEmail (const string& filePath, vector<string>& requests);
```

Tham số

- **filePath**: Đường dẫn đến file email cần đọc
 - **requests**: Tham chiếu đến vector lưu trữ các requests để cập nhật các requests
4. Hàm **sendEmailWith**: Gửi email phản hồi đính kèm kết quả tới user, sử dụng giao thức SMTP và các object được hỗ trợ bởi **EASendMail**.

```
bool sendEmailWith (std::string Recipient, std::string Subject,  
std::string& BodyText, const std::string& AttachmentPath="");
```

Tham số:

- **Recipient**: Địa chỉ mail user (người nhận)
 - **Subject**: Tiêu đề mail - mặc định là **RESULT: <tên request>**
 - **BodyText**: Nội dung mail
 - **AttachmentPath**: Đường dẫn đến file kết quả đính kèm
5. Hàm **receiveEmail**: Kiểm tra và nhận email sử dụng giao thức IMAP và các object hỗ trợ bởi **EAGetMail**.

```
void receiveEmail();
```

Hoạt động chính: Kết nối đến máy chủ **IMAP**, cấu hình tài khoản gmail cho client với gmail và app password đã tạo trước đó, nhận email gửi đến và tải email về thư mục **inbox** trong project folder.

6. Hàm **processEmail**: Xử lý các email đến, xác thực người gửi và thực thi các requests được yêu cầu qua email.

```
void processEmail();
```

Các hàm xử lý Request

1. Hàm **executeRequest**: Khởi tạo Socket kết nối tới Server, gửi các requests đến server và xử lý kết quả trả về.

```
int executeRequest (const string& serverIP, const string& request,
                    string& messageResult, string& resultFilePath, string& fileRequest);
```

Tham số:

- **serverIP**: Địa chỉ IP của server
- **requests**: Vector chứa các lệnh mà user yêu cầu (user có thể yêu cầu nhiều lệnh trong 1 email)
- **messageResult**: Message trả về trạng thái thực thi trên server đối với các lệnh `start/end app`, `start/end service`, `delete file`
- **resultFilePath**: Đường dẫn để lưu kết quả dạng file đối với các lệnh `start webcam`, `get file`, `screenshot`, `get app running`, `get service running`
- **fileRequest**: tên file cụ thể cần lấy (đối với lệnh `get file`)

2. Hàm `getFileFromServer`: Xử lý lấy file từ server qua SOCKET và lưu vào file cục bộ.

```
bool getFileFromServer (SOCKET clientSock, string& fileResult, string& filename);
```

Tham số:

- **clientSock**: Socket kết nối giữa client và server
- **fileResult**: Tên hoặc đường dẫn của file sẽ lưu vào máy của client sau khi nhận dữ liệu file từ server
- **filename**: Là tên hoặc đường dẫn của file được yêu cầu. Nếu giá trị không rỗng (đối với lệnh `get file`), nó được gửi đến server để chỉ định tệp cần tải xuống.

3.1.2 Class `LoginDialog` - Đăng nhập và xử lý xác thực người dùng

Các hàm chính

1. Hàm `validateLogin`: Kiểm tra trạng thái của ô nhập email và mật khẩu.

```
void LoginDialog::validateLogin();
```

2. Hàm `handleLogin`: Xác thực thông tin đăng nhập và hiển thị thông báo

```
void LoginDialog::handleLogin();
```

3. Hàm `authenticateUser`: Kiểm tra thông tin đăng nhập dựa trên cơ sở dữ liệu người dùng trong tệp JSON.

```
bool LoginDialog::authenticateUser(const QString& email, const QString& password);
```



3.1.3 Class RegisterDialog - Đăng ký tài khoản người dùng

Các hàm chính

1. Hàm loadRegisteredUsers: Đọc danh sách người dùng đã đăng ký từ file `registered_users.json`.

```
void RegisterDialog::loadRegisteredUsers();
```

2. Hàm isValidEmail: Kiểm tra định dạng email hợp lệ.

```
bool RegisterDialog::isValidEmail(const QString& email) const;
```

3. Hàm isValidPassword: Kiểm tra mật khẩu hợp lệ dựa trên các tiêu chí: độ dài, số, và ký tự đặc biệt.

```
bool RegisterDialog::isValidPassword(const QString& password) const;
```

4. Hàm handleRegistration: Xử lý đăng ký tài khoản.

```
void RegisterDialog::handleRegistration();
```

3.1.4 Class MainWindow - Giao diện chính quản lý dịch vụ Remote Control

Các hàm chính

1. Hàm showLoginDialog: Hiển thị hộp thoại đăng nhập và xử lý thông tin đăng nhập.

```
void MainWindow::showLoginDialog();
```

2. Hàm showRegisterDialog: Hiển thị hộp thoại và xử lý thông tin đăng ký.

```
void MainWindow::showRegisterDialog();
```

3. Hàm showAboutDialog: Hiển thị thông tin "About" từ file `about.txt`

```
void MainWindow::showAboutDialog();
```

4. Hàm showHelpDialog: Hiển thị thông tin "Help" từ file `help.txt`

```
void MainWindow::showHelpDialog();
```

5. Hàm onLogoutClicked: Xử lý sự kiện đăng xuất, làm sạch thông tin đăng nhập và cập nhật giao diện.

```
void MainWindow::onLogoutClicked();
```



6. Hàm `updateLog`: Thêm nội dung vào buffer ghi log trong quá trình xử lý request.

```
void MainWindow::flushLogBuffer();
```

7. Hàm `onStartServiceClicked`: Khởi động dịch vụ Remote Control và cập nhật giao diện.

```
void MainWindow::onStartServiceClicked();
```

8. Hàm `onStopServiceClicked`: Dừng dịch vụ Remote Control và cập nhật giao diện.

```
void MainWindow::onStopServiceClicked();
```

9. Hàm `updateUIBasedOnLoginStatus`: Cập nhật giao diện dựa trên trạng thái đăng nhập (ẩn/hiện các thành phần giao diện).

```
void MainWindow::updateUIBasedOnLoginStatus();
```

3.1.5 Một số Class xử lý giao diện khác

- Class `CurveWidget`: Vẽ đường cong, tạo hiệu ứng động cho màn hình Welcome.
- Class `FontManager`: Cài đặt Font chữ đồng bộ cho cả Service.
- Class `WelcomeScreen`: Màn hình Welcome User.
- Class `AboutDialog`: Giao diện và nội dung About Box.
- Class `HelpDialog`: Giao diện và nội dung Help Box.

3.2 Server side

3.2.1 Quản lí Application trong máy tính Server

1. Hàm chuyển đổi chuỗi kí tự từ `wstring` thành `string` và từ `string` thành `wstring`.

```
std::string wstringToString(const std::wstring& wstr);  
std::wstring stringToWstring(const std::string& str);
```

Tham số:

- `wstring& wstr`: tham chiếu tới chuỗi Unicode muốn chuyển đổi.
 - `string& str`: tham chiếu tới chuỗi kí tự ASCII muốn chuyển đổi.
2. Hàm `BOOL CALLBACK EnumWindowsProc`: dùng khi liệt kê tất cả các cửa sổ đang mở trên hệ thống, kiểm tra xem cửa sổ hiện tại `hwnd` có thuộc về tiến trình có ID là `processID` không. Nếu có thì trả về `FALSE` để dừng việc liệt kê khi tìm thấy cửa sổ thuộc về tiến trình yêu cầu.

```
BOOL CALLBACK EnumWindowsProc(HWND hwnd, LPARAM processID);
```

Tham số:

- `HWND hwnd`: Handle của cửa sổ hiện tại.
 - `LPARAM processID`: ID của tiến trình cần tìm.
3. Hàm `saveProcessNameAndId`: mở tiến trình với ID là `processID` để lấy tên của tiến trình đó, liệt kê tất cả các cửa sổ liên quan tới ID tiến trình đó và ghi thông tin về tên, process ID, và tiêu đề cửa sổ của tiến trình đó vào file `outfile`.

```
void saveProcessNameAndID(DWORD processID, std::ofstream& outfile);
```

Tham số:

- `DWORD processID`: ID của tiến trình
 - `ofstream& outfile`: Tham chiếu tới file để ghi kết quả
4. Hàm `bool getAppRunning`: Sử dụng `EnumProcesses` để lấy danh sách các ID tiến trình, dùng hàm `saveProcessNameAndID` với tham số là danh sách các ID tiến trình đó để lưu các thông tin cần thiết vào output file.

```
bool getAppRunning(const std::string& filename);
```

Tham số:

- `const std::string& filename`: Tên file để lưu danh sách tiến trình
5. Hàm `startApp` khởi chạy một ứng dụng với đường dẫn tuyệt đối bằng cách sử dụng `ShellExecute` của Windows, hàm trả về `false` nếu có lỗi khi thực thi.



```
bool startApp(std::string& appName);
```

Tham số:

- `std::string& appName`: Đường dẫn tuyệt đối tới ứng dụng cần khởi chạy.

6. Hàm `getProcessName` lấy tên của tiến trình từ ID tiến trình, trả kết quả dưới dạng chuỗi Unicode `wstring`.

```
std::wstring getProcessName(DWORD processID);
```

Tham số:

- `DWORD processID`: ID của tiến trình muốn biết tên.

7. Hàm `std::vector<ProcessInfo> listProcessByName`: Hàm dùng `getProcessName` để lấy tên từ danh sách các ID tiến trình, sau đó so khớp tên với `targetProcessName` và trả về một danh sách các `ProcessInfo` có tên tiến trình trùng với nó.

```
struct ProcessInfo {  
    DWORD processId;  
    std::wstring processName;  
};  
std::vector<ProcessInfo> listProcessesByName(std::wstring& targetProcessName);
```

Tham số:

- `struct ProcessInfo` lưu thông tin ID của tiến trình `processID` và tên ở dạng chuỗi Unicode `processName` của tiến trình.
- `wstring& targetProcessName`: Tên của tiến trình (application) cần tìm.

8. Hàm `terminateProcessById` kết thúc một tiến trình dựa trên ID của nó `processId`.

```
bool terminateProcessById(DWORD processId);
```

Tham số:

- `processId`: ID của tiến trình muốn kết thúc.

9. `endApp`: Hàm liệt kê thông tất cả các tiến trình có tên trùng với tên ứng dụng muốn dừng hoạt động `appName`. Sau đó, xóa tiến trình có ID trùng với `pid` hoặc xóa hết các tiến trình khi `pid` là -1.

```
bool endApp(std::string& appName, DWORD pid = -1);
```

Tham số:

- `string& appName`: Tên ứng dụng muốn dừng hoạt động.
- `DWORD pid`: một ID tiến trình của ứng dụng muốn xóa, nếu muốn xóa hết các tiến trình liên quan tới ứng dụng này thì giá trị mặc định đặt là -1.



3.2.2 Quản lý Service trong máy tính Server

1. Hàm `getServiceRunning` liệt kê các services đang chạy (active) và lưu danh sách các thông tin gồm: Tên service, ID tiến trình (PID), tên hiển thị (Display Name) vào file được chỉ định là `filename`

```
bool getServiceRunning(const std::string& filename);
```

Tham số:

- `const std::string& filename`: tên file text muốn lưu các services đang chạy trên máy.
2. Hàm `startService` khởi chạy một service được chỉ định bởi tên service `serviceName`

```
bool startService(const std::string& serviceName);
```

Tham số:

- `const std::string& serviceName` là tên service muốn khởi chạy
3. Hàm `endService` dừng hoạt động một service được chỉ định bởi tên service `serviceName`

```
bool endService(const std::string& serviceName);
```

Tham số:

- `const std::string& serviceName` là tên service muốn dừng hoạt động

3.2.3 File - Quản lý file trong máy tính Server

1. Hàm `sendFileRequested` gửi một file từ server đến client thông qua socket. Nếu file tồn tại thì gửi mã trạng thái 200(file có sẵn) và kích thước file đến client, sau đó gửi nội dung file từng phần với kích thước cố định là `BUFFER_SIZE`. Nếu file không tồn tại gửi mã trạng thái 404.

```
bool sendFileRequested(SOCKET clientSock, string filename);
```

Tham số:

- `SOCKET clientSock`: Socket đại diện cho kết nối giữa server và client, qua đó dữ liệu sẽ được gửi.
 - `string filename`: Tên file (bao gồm cả đường dẫn nếu cần) mà server cần gửi đến client.
2. Hàm `deleteFile` xóa file được chỉ định là `searchfilename` nếu file có tồn tại.

```
bool deleteFile(string searchfilename);
```

Tham số:

- `string searchfilename`: Đường dẫn tới file cần xóa.



3.2.4 Screenshot - Chụp ảnh màn hình máy tính

1. Hàm `saveBitmapToFile` lưu dữ liệu hình ảnh dưới dạng bitmap (HBITMAP hBitmap) vào file được chỉ định `filename`

```
bool saveBitmapToFile(HBITMAP hBitmap, HDC hDC, const std::string& filename)
```

Tham số:

- HBITMAP hBitmap: Handle của bitmap cần được lưu. Đây là nơi chứa dữ liệu hình ảnh.
 - HDC hDC: Device Context tương thích với bitmap, được sử dụng để trích xuất dữ liệu bitmap.
 - const std::string& filename: Tên và đường dẫn tệp đầu ra nơi bitmap sẽ được lưu.
2. Hàm `screenshot` chụp toàn bộ màn hình và lưu thông tin màn hình vào bitmap, sử dụng `saveBitmapToFile` để lưu kết quả chụp màn hình.

```
bool screenshot(const std::string& filename)
```

Tham số:

- const std::string& filename: Tên và đường dẫn tệp đầu ra để lưu ảnh chụp màn hình.

3.2.5 Record Video - Quay màn hình Webcam máy tính

1. Hàm `bool fileExists` kiểm tra xem file có tồn tại trong hệ thống hay không. Hàm trả về `true` nếu file tồn tại, ngược lại trả về `false`.

```
bool fileExists(const std::string& filename);
```

Tham số:

- const std::string& filename: Tên file cần kiểm tra.
2. Hàm `getVideoDeviceName` được sử dụng khi máy khởi động webcam với tên mặc định camera của Window không thành công. Hàm này sử dụng lệnh `ffmpeg` tìm camera khả dụng trong máy tính Server.

```
std::string getVideoDeviceName();
```

3. Hàm `startWebcam` quản lý việc ghi hình từ webcam và lưu video đầu ra dưới tên `filename`. Hàm kiểm tra nếu `filename` đã tồn tại thì xóa file cũ trước khi thực hiện ghi hình mới. Hàm thực hiện ghi hình video bằng lệnh `ffmpeg` nhờ webcam mặc định của Window (Integrated Camera) hoặc thiết bị khả dụng khác lấy từ `getVideoDeviceName()`.



```
void startWebcam(int durationSeconds, const std::string& filename);
```

Tham số:

- `durationSeconds`: Số giây muốn ghi hình từ webcam.
- `filename`: Tên file đầu ra để lưu video.

3.2.6 ShutDown - Tắt máy tính

Hàm `ShutDownPC` thực hiện tra cứu và kích hoạt quyền tắt máy (`shutdown privilege`), sau đó sử dụng lệnh tắt máy bằng API `ExitWindowEx`.

```
bool ShutDownPC();
```



Tài liệu

- [1] Koolac, "How to install ffmpeg on Windows," *YouTube*, 30 Apr. 2024. [Online]. Available: <https://www.youtube.com/watch?v=JR36oH35Fgg>
- [2] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*, 7th ed.
- [3] L. Peterson and B. Davie, *Computer Networks: A Systems Approach*, 6th ed.
- [4] Tài liệu thực hành mạng máy tính. [Online]. Available: <https://drive.google.com/drive/folders/1Usrw1IE63XeyMnRljp50NWckUqijWgt6>