****

|  |  |
| --- | --- |
| **Họ và tên** | **MSSV** |
| Đỗ Đức Minh Triết | 23521650 |
| Nguyễn Quốc Việt | 23521782 |
| Nguyễn Thanh Tùng | 23521744 |

GVHD: **Phan Trung Phát**

Lớp: **NT106.P13.1**

3

**Working with Sockets in C#**

**ĐÁNH GIÁ KHÁC (\*):**

|  |  |
| --- | --- |
| **Nội dung** | **Kết quả** |
| Tổng thời gian thực hiện bài thực hành trung bình (1) | 5 ngày |
| Link Video thực hiện (2)  *(nếu có)* | [Bai3](https://drive.google.com/file/d/1M0bxj3wmpjwixVq9CNYFOFDNexjrI34F/view?usp=sharing) [[Bai4](https://drive.google.com/file/d/1EMlwEjpewxnK6g0GpC5BY-VtBB144x9q/view?usp=sharing)](https://drive.google.com/file/d/1EMlwEjpewxnK6g0GpC5BY-VtBB144x9q/view?usp=sharing) [[Bai5](https://8gcf1d-my.sharepoint.com/:f:/g/personal/posu29_8gcf1d_onmicrosoft_com/EiZ_uYs7YHlDlwqbLUrwWjcB8DMeo_CUpabNa45SLl0z3w?e=ZlQbwJ)](https://8gcf1d-my.sharepoint.com/:f:/g/personal/posu29_8gcf1d_onmicrosoft_com/EiZ_uYs7YHlDlwqbLUrwWjcB8DMeo_CUpabNa45SLl0z3w?e=ZlQbwJ) |
| Ý kiến (3) *(nếu có)*  + Khó khăn  + Đề xuất … |  |
| Điểm tự đánh giá (4) | 8/10 |
| (\*): phần (1) và (4) bắt buộc thực hiện. | |

# Bài 3

1. **Minh chứng:**

A screenshot of a computer

Description automatically generated

Hình . Bài 3

A screenshot of a computer

Description automatically generated

Hình . Form Server bài 3

A screenshot of a computer

Description automatically generated

Hình . Form Client bài 3

* 1. **Code Server:**

A screenshot of a computer program

Description automatically generated

Hình . Code server(1)

A screenshot of a computer code

Description automatically generated

Hình . Code server(2)

* 1. **Code client:**

A screenshot of a computer code

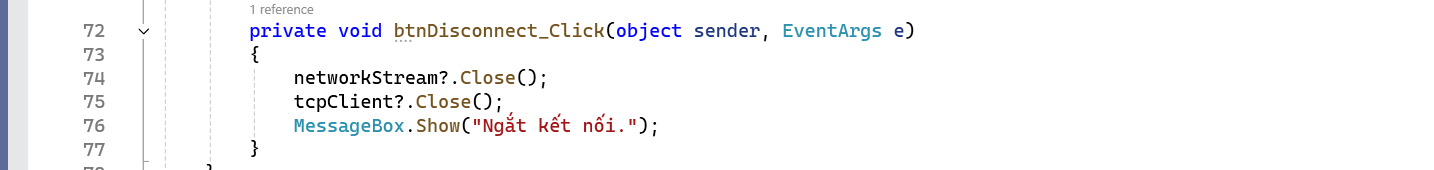
Description automatically generated

Hình . Code client(1)

A screen shot of a computer code

Description automatically generated

Hình . Code client(2)



Hình . Code client(3)

## Giải thích:

**Server:**

* Viết hàm **btnListen\_Click** để bắt đầu quá trình lắng nghe kết nối từ client khi người dùng nhấn nút “Listen”.
  + Khởi tạo **serverThread**, một luồng nền để chạy server mà không làm gián đoạn giao diện người dùng.
  + Gọi hàm **StartServer** để server bắt đầu lắng nghe.
  + Vô hiệu quá nút **“Listen”** sau khi server được khởi động để ngắn việc khởi động lại server nhiều lần.
* Viết hàm **StartServer** để khởi tạo Server và lắng nghe kết nối từ client.
  + Tạo đối tượng **tcpListener** tại địa chỉ **127.0.0.1** và cổng **8080**.
  + Bắt đầu **tcpListener** để lắng nghe kết nối từ client.
  + Trong một vòng lặp, gọi **AccpetTCPClient()** để chấp nhận client kết nối.
  + Khi có kết nối mới từ **client**:
  + Lấy **địa chỉ IP** và cổng của client và hiển thị thông tin này trong textbox.
  + Tạo một luồng mới **(clientThread)** để xử lý kết nối của **client**, sau đó gọi hàm **HandleClient** để quản lý giao tiếp với **client** này.
* Viết hàm **HandleClient** để quản lý kết nối với từng client riêng lẻ.
  + Mở luồng dữ liệu (**NetworkStream**) để trao đổi thông tin với client.
  + Sử dụng một vòng lặp để nhận và xử lý dữ liệu từ **client**:
  + Đọc dữ liệu vào **buffer** và chuyển thành chuỗi **message**.
  + Hiển thị tin nhắn từ client trong **txtServerMessages**.
  + Khi client đóng kết nối, luồng **stream** sẽ không còn dữ liệu, vòng lặp kết thúc, đóng client, và cập nhật **txtServerMessages** với thông báo ngắt kết nối.
* Viết hàm **OnFormClosing** để dừng server 1 cách an toàn khi form bị đóng
  + Kiểm tra nếu **tcpListener** tồn tại thì gọi Stop để đóng server và ngừng lắng nghe.
  + Gọi Join trên **serverThread** để đảm bảo luồng server dừng hẳn trước khi ứng dụng đóng.
  + Gọi hàm gốc **OnFormClosing** để xử lý các thao tác đóng form khác (nếu có).

**Client:**

* Viết hàm **btnConnect\_Click** để Kết nối client đến server khi người dùng nhấn nút "**Connect**".
  + Kiểm tra xem client đã được kết nối đến server chưa. Nếu chưa:
    - Gọi **tcpClient.Connect("127.0.0.1", 8080)** để kết nối đến server tại địa chỉ **IP 127.0.0.1** và cổng **8080.**
    - Lấy **networkStream** từ tcpClient để quản lý luồng dữ liệu trao đổi với server.
    - Hiển thị thông báo kết nối thành công.
  + Nếu đã kết nối trước đó, hiển thị thông báo cho biết client đã kết nối.
  + Xử lý ngoại lệ **SocketException** và hiển thị thông báo nếu có lỗi khi kết nối.
* Viết hàm **btnSend\_Click** để gửi tin nhắn từ client đến server.
  + Kiểm tra nếu **client** đã kết nối **(tcpClient.Connected)**.
  + Nếu kết nối:
    - Lấy nội dung tin nhắn từ **txtClientMessages** và mã hóa sang byte bằng **Encoding.ASCII.GetBytes**.
    - Ghi dữ liệu **(networkStream.Write)** vào **networkStream** để gửi đến server.
    - Hiển thị thông báo rằng tin nhắn đã được gửi thành công.
  + Nếu chưa kết nối, hiển thị thông báo yêu cầu kết nối trước khi gửi.
  + Xử lý ngoại lệ nếu có lỗi khi gửi tin nhắn.
* Viết hàm **btnDisconnect\_Click** để ngắt kết nối client khỏi server.
  + Đóng **networkStream** (nếu tồn tại) để ngừng luồng truyền dữ liệu.
  + Đóng **tcpClient** (nếu tồn tại) để ngắt kết nối hoàn toàn.
  + Hiển thị thông báo xác nhận ngắt kết nối thành công.

**Link Google Drive:**

[https://drive.google.com/file/d/1M0bxj3wmpjwixVq9CNYFOFDNexjrI34F/view?usp=drive\_link](https://drive.google.com/file/d/1M0bxj3wmpjwixVq9CNYFOFDNexjrI34F/view?usp=drive_link&fbclid=IwZXh0bgNhZW0CMTAAAR1vS12MkKCKEfAAlFuC710QiTUdlxobqMu2Ld4hse3wSQecpC_0NL5EFC8_aem_SgW5DHdz0h7tLplZlfByFA)

# Câu hỏi 4

## 1. Minh chứng:

## A screenshot of a computer Description automatically generated

Hình . Hình ảnh giao diện bên phía client

A screen shot of a computer program

Description automatically generated

Hình . Class MovieTicket

A black background with white text

Description automatically generated

Hình . Khai báo List luồng, hàm cancel, IP, Socket

A computer screen shot of a program

Description automatically generated

Hình . Hàm Connect đến server

A black screen with white text

Description automatically generated

Hình . Hàm đóng kết nối

A computer screen shot of a program

Description automatically generated

Hình . Hàm Gửi thông tin

A computer screen shot of a program code

Description automatically generated

Hình . Hàm gửi thông tin khi đặt vé

A computer screen shot of a program code

Description automatically generated

Hình . Hàm Deserialize

A computer screen shot of a program

Description automatically generated

Hình . Hàm Receive thông tin từ server

A screen shot of a computer program

Description automatically generated

Hình . Initilize các giá trị, hàm khi bắt đầu chương trình

A screen shot of a computer program

Description automatically generated

Hình . Class MovieStatistics

A screen shot of a computer program

Description automatically generated

Hình . Class ThongTin

A computer screen shot of a program code

Description automatically generated

Hình . Class Hall, List my\_list với type là ThongTin, Dictionary my\_hall

A screen shot of a computer program

Description automatically generated

Hình . Function Lưu thông tin

A computer screen shot of a program code

Description automatically generated

Hình . Nội dung button Lưu và hàm Deserialize Thông Tin

A computer screen shot of a black background

Description automatically generated

Hình . Hàm kiểm tra dấu phẩy ở input

A screen shot of a computer program

Description automatically generated

A computer screen with colorful text

Description automatically generated

Hình . Nội dung button Add

A screen shot of a computer code

Description automatically generated

Hình . Button Lấy thông tin

A computer screen shot of a program code

Description automatically generated

Hình . Activity trong comboBox Chọn phim

A screen shot of a computer program

Description automatically generated

Hình . Activity trong comboBox Chọn rạp

A screen shot of a computer program

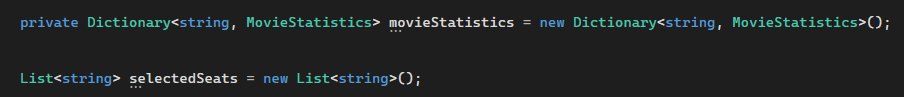
Description automatically generated

Hình . Activity trong Event Chọn ghế checklistBox

A computer screen shot of a program code

Description automatically generated

Hình . Hàm kiểm tra ghế đã được đặt hay chưa



Hình . Dictionary MovieStatistics và List selectedSeats

A screen shot of a computer program

Description automatically generated

A computer screen with many colorful text

Description automatically generated

Hình . Activity trong button Đặt vé

A computer screen shot of a program code

Description automatically generated

Hình . Nội dung button XuatThongTin

A screen shot of a computer program

Description automatically generated

Hình . Đóng kết nối khi form đóng

A computer screen shot of a program

Description automatically generated

Hình . Kiểm tra Checked Seats khi chọn ghế

A computer screen shot of a program code

Description automatically generated

Hình . Nội dung Function SendCheckedSeats

A screenshot of a computer

Description automatically generated

Hình . Giao diện server

A screen shot of a computer program

Description automatically generated

Hình . Khai báo các kết nối cần thiết, Initialize khi chương trình bắt đầu

A computer screen shot of a program

Description automatically generated

Hình . Hàm Khởi động kết nối cho server

A screen shot of a computer program

Description automatically generated A screen shot of a computer screen

Description automatically generated

Hình . Nội dung hàm Receive bên server

A screen shot of a computer program

Description automatically generated

Hình . Hàm kiểm tra Rạp phim đã có trong Database hay chưa

A screen shot of a computer program

Description automatically generated

Hình . Hàm Insert ghế của rạp

A screen shot of a computer code

Description automatically generated

Hình . Hàm Update trạng thái ghế sau khi mua

A computer screen with colorful text

Description automatically generated

Hình . Hàm Deserialize và đóng kết nối bên server khi đóng form

## 2. Giải thích:

Class **MovieTicket**:

- **Nội dung**: Kiểu dữ liệu này sẽ đóng vai trò thao tác với **Server.**

**- Trình bày:** Khai báo các biến Tên người dùng, tên phim, rạp đã chọn, số ghế đã chọn, tổng giá tiền, **IsInitialInfo** (kiểm tra xem **dữ liệu** được gửi đi có phải đã dữ liệu **khởi tạo** thông tin hay không), **IsOccupied** (Check xem dữ liệu ghế được gửi đi là khai báo **nhằm lưu** vào cơ sở dữ liệu hay **cập nhật** dữ liệu trên cơ sở dữ liệu).

List **Thread**:

- **Nội dung**: Theo dõi các luồng hoạt động => nhằm **quản lý** luồng khi đóng form, tránh luồng chạy ngầm.

Lớp **CancellationTokenSource**:

- **Nội dung**: Cơ chế báo hủy trong các hoạt động đa luồng hoặc không đồng bộ.

**IPEndPoint, Socket**:

- **Nội dung**: Khởi tạo IP, socket để kết nối đến server.

Hàm **ConnectToServer()**:

- **Nội dung**: Khai báo địa chỉ IP đích, client, kết nối đến Server.

- **Trình bày**:

- Khai báo địa chỉ đích của server **127.0.0.1 – địa chỉ LoopBack**, có thể đổi sang địa chỉ khác ví dụ **192.168.1.15** nếu server ở máy tính khác.

- Thread **listen** = new Thread:

* Khởi tạo luồng, chạy độc lập so với luồng chính.
* **Lamda Expression**: Định nghĩa anonymous method và truyền token **cancellationTokenSource.Token** làm tham số.
* Phương thức **Receive:** Xử lý dữ liệu đến từ server, CancellationToken cho phép dừng **Receive** khi cần thiết.
* **Threads.Add(listen)**: theo dõi tất cả các luồng được tạo.

Hàm **CloseConnection()**:  
- **Nội dung**: đóng kết nối phía client.

Hàm **Send\_ThongTin()**:

- **Nội dung**: Gửi nội dung khởi tạo nhằm để **server** cập nhật dữ liệu lên **database**.

- **Trình bày**: Sử dụng class **MovieTicket** để dễ dàng thống nhất dữ liệu gửi đi, Serialize dữ liệu bằng **JSON** tránh lỗi Input format ghi dùng **BinaryFormatter.**

Ở đây, **IsInitialInfo** được set bằng **true** và **IsOccupied** được set bằng **false** vì đây là hàm gửi nhằm khởi tạo thông tin trên cơ sở dữ liệu.

Hàm **Send\_datVe()**:

- **Nội dung**: Update dữ liệu sau khi người dùng click **datVe button**.

- **Trình bày**: Truyền các đối tượng được người dùng chọn như tên phim, rạp phim, ghế được chọn sau đó gửi đến **server** để cập nhật thông tin trên **database.** Ở đây, **IsInitialInfo** được set bằng **false** và **IsOccupied** được set bằng **true** vì đây là hàm gửi nhằm cập nhật thông tin không phải khởi tạo thông tin.

Hàm **Deserialize()**:

- **Nội dung**: **Deserialize** thông tin được truyền đến từ server (dạng **JSON**).

Hàm **Receive()**:

- **Nội dung**: Lắng nghe nội dung truyền đến từ server.

- **Trình bày**: Kiểm tra thông tin được nhận có phải là **Cancel Request** từ server hay không (**!token.IsCancellationRequested**) nếu không phải thì **Deserialize** dữ liệu. **Catch** các lỗi do **socket** hoặc lỗi khác xảy ra nếu có:

* **information != null && information.Sign == 3:** Kiểm tra **thông tin** nhận được có rỗng hay không, **Sign = 3** => thông tin về seats đã chọn ở các client khác.
* Kiểm tra ghế vừa **tick** nếu ghế trùng thì **hủy bỏ trạng thái checked** ở client đầu tiên kèm **thông báo** “Ghế … đã được đặt”.

Class **MovieStatistics**:

- **Nội dung**: Đối tượng dùng để **lưu thông tin** được người dùng chọn => nhằm thống kê doanh thu, xếp hạng phim khi xuất ra **file output**.

Class **ThongTin**:

- **Nội dung**: Đối tượng nhận các thông tin khởi tạo mặc định cho rạp.

Class **Hall:**

**- Nội dung:** Đối tượng khởi tạo **ghế và trạng thái** ghế trong 1 rạp (mặc định là **false** - ghế chưa được chọn).

List<**ThongTin**> **my\_list**:  
- **Nội dung**: Tạo list cho mỗi phim được nhập vào trước khi lưu **file input**.

Dictionary<**string, Hall**> **my\_hall**:

- **Nội dung**: Nhằm quản lý nhiều rạp và trạng thái ghế tương ứng vào rạp đó.

Hàm **Lưu**:

- **Nội dung**: **Serialize** thông tin (mảng **ThongTin**) ở screen sau khi add và lưu vào file input.

Button **Lưu**:

- **Nội dung**: gọi hàm **Lưu**.

Hàm **Deserialize**:  
- **Nội dung**: **Deserialize** thông tin trước khi lấy thông tin từ file và cập nhật lên giao diện

Hàm **IsCommaSeparated**:

- **Nội dung**: Kiểm tra thông tin số **rạp chiếu** được nhập từ người dùng với **rằng buộc** là nếu số lượng rạp nhiều hơn 1 thì phải ngăn cách bằng **dấu phẩy**.

Button **Add**:

- **Nội dung**: Kiểm tra thông tin được nhập và add vào screen để chuẩn bị cho việc lưu trữ.

- **Trình bày**:

* Kiểm tra dấu phẩy bằng **IsCommaSeparated(rapChieu\_box.Text)**
* Áp dụng **Regular Expression** cho **Giá vé** **System.Text.RegularExpressions.Regex.IsMatch(giaVe\_box.Text, @"^[1-9]\d\*$") – Giá vé** phải là số không bắt đầu từ 0 và không chứa ký tự.
* Kiểm tra đối chiếu với các thông tin đã có trong **Screen** để không xảy ra hiện tượng trùng lặp => nếu không có **trùng lặp** xảy ra thì dữ liệu được add vào **my\_list** trước khi lưu.

Button **layInfo**:

- **Nội dung**: Deserialize dữ liệu từ file input và cập nhật thông tin lên giao diện

ComboBox **phimSelection**:

- **Nội dung**: Clear các giá trị ghế được tick trước đó khi chọn phim khác và cập nhật giá vé cơ sở đối với đối với mỗi phim được chọn.

ComboBox **chonRap**:

- **Nội dung**: Làm sạch các giá trị ghế được **tick** trước đó khi chuyển sang **rạp mới** và gửi thông tin đến **server** để kiểm tra rạp hiện tại, nếu không có trong **cơ sở dữ liệu** => thực hiện câu lệnh **SQL** để tạo rạp mới trong cơ sở dữ liệu.

- **Trình bày**: gọi hàm **Send\_ThongTin()** đã được định nghĩa (hàm được set bất đồng bộ để không ảnh hưởng đến việc cập nhật giao diện).

Event **SeatSelect**:

- **Nội dung**: Kiểm tra xem các ghế chọn đã được đặt hay chưa, nếu đã được đặt rồi thì khôi phục lại trạng thái **unchecked**

**- Trình bày:** gọi hàm **IsSeatOccupiedInDatabase()** để check giá trị trong database.

Hàm **IsSeatOccupiedInDatabase**

- **Nội dung**: chọn các seat trong **database** dựa trên tên rạp và ghế được truyền vào, sau đó kiểm tra => nếu ghế đã được đặt (**IsOccupied** = 1) thì báo lỗi.

Button **Đặt Vé**:

- **Nội dung**: Hiển thị thông tin mua của người dùng (tên, phim đã chọn, số ghế đã chọn, tổng số tiền cần thanh toán), gửi thông tin đặt vé đến server.

- **Trình bày**:

* Sử dụng lệnh **If** để kiểm tra người dùng đã chọn đủ thông tin hay không.
* Tính toán giá vé dựa trên chỗ ngồi **đã chọn** => Sử dụng vòng **for**, sau đó tính toán số vé **còn lại**.
* **Send\_datVe()**: Gửi thông tin đặt vé đến server để cập nhật giá trị ghế.

Button **Xuất Thông Tin**:

- **Nội dung**: Hiển thị thông tin của phim đã mua (tên phim, số lượng vé bán ra, số lượng vé tồn, tỉ lệ vé bán ra, doanh thu, xếp hạng phòng vé).

- **Trình bày**:

* Sắp xếp thứ tự thông tin phim lưu vào file dựa trên doanh thu được tính toán từ mỗi lần mua vé => chuyển thành **List** và lưu vào biến **sortedStatistics.**
* Ghi vào file **output** các thông tin theo đề yêu cầu với mỗi đối tượng **MovieStatics**.
* Thanh **Progress Bar** giúp người dùng theo dõi tình trạng xuất file.

void **Bai4\_FormClosed()**:

- **Nội dung**: Đóng kết nối và kiểm tra có thread nào chạy nền hay không.

void **StartServer()**:

- **Nội dung**: Chạy server, nhận địa chỉ từ bất kỳ IP nào trong nội mạng.

Hàm **Receive()**:

- **Nội dung**: Lắng nghe các data được gửi đến từ phía client, sau đó thực hiện **Deserialize** dưới dạng **MovieTicket** tưởng ứng với định dạng được gửi đi sau đó bắt đầu xử lý dữ liệu.

- **Trình bày**:

* **ticketInfo.Sign == 1**: Kiểm tra dữ liệu truyền vào có phải là dữ liệu khởi tạo rạp được gửi từ phía client hay không (hàm **Send\_ThongTin**). Nếu có thì kiểm tra xem rạp đã tồn tại hay chưa **IsHallInDatabase(ticketInfo)** rồi thực hiện thêm rạp **InsertSeatAvailability(ticketInfo).**
* Nếu **Sign == 2** thì dữ liệu gửi đến sẽ được coi như là dữ liệu nhằm update trạng thái rạp từ **0 => 1** (Occupied). Sau đó thêm thông tin của vé vừa được mua vào **screen.**
* Nếu **Sign == 3** thì dữ liệu được gửi đến sẽ được hiểu là **server** có nhiệm vụ **broadcast** đến các **client khác** về ghế đã chọn của **client đầu tiên**.

Hàm **IsHallInDatabase()**:  
- **Nội dung**: Khởi tạo kết nối với **database**. Kiểm tra dữ liệu đã có trong cơ sở dữ liệu hay chưa.

Hàm **InsertSeatAvailability()**:

- **Nội dung**: Thực hiện không đồng bộ việc thêm rạp vào trong cơ sở dữ liệu với query đã được định nghĩa.

Hàm **UpdateSeatAvailability():**

**- Nội dung:** Khởi tạo kết nối với **database**. Cập nhật trạng thái chỗ được đặt trong **database** với những ghế có trong data **foreach (string seat in data.Seats)**.

Hàm **Deserialize()**:  
- **Nội dung**: **Deserialize** thông tin được nhận (**JSON Format**).

void **server\_FormClosed()**:

- **Nội dung**: Thực hiện việc ngắt kết nối khi form được đóng.

CheckListBox **seatSelect()**:

- **Nội dung**: Kiểm tra ghế sau khi check đã được chọn hay chưa

- **Trình bày**: Với mỗi lần **check/uncheck** các ghế, **thông tin** sẽ được gửi đến **server** bằng **SendCheckedSeats()**, sau đó **Server** sẽ **broadcast** cho các **client** khác về tình trạng ghế. Khi các **client** khác **check** vào ghế đã chọn thì sẽ hiện ra **thông báo**.

void **SendCheckedSeats():**

- **Nội dung**: Gửi thông tin mỗi lần check ghế đến server (sau khi chuyển sang dạng MovieTicket).

- **Trình bày**: Sử dụng **async** để xử lý **dữ liệu bất đồng** bộ với sự kiện check từ **người dùng (**dữ liệu gửi đi được Serialize dạng **JSON** sau đó mã hóa sang **byte)**.

**Link Google Drive**: [**https://drive.google.com/file/d/1EMlwEjpewxnK6g0GpC5BY-VtBB144x9q/view?usp=sharing**](https://drive.google.com/file/d/1EMlwEjpewxnK6g0GpC5BY-VtBB144x9q/view?usp=sharing)

# Bài 5

## Minh chứng:

## Server:

A screenshot of a computer program

Description automatically generated

Hình 45 Code server side phần 1

A computer screen shot of a program

Description automatically generated

Hình 46 Cover server side phần 2

A screenshot of a computer program

Description automatically generated

Hình 47 Cover server side phần 3

A computer screen shot of a program

Description automatically generated

Hình 48 Cover server side phần 4

A screenshot of a computer program

Description automatically generated

Hình 49 Code server side

A screenshot of a computer program

Description automatically generated

Hình 50 Code server side phần 6

A screen shot of a computer program

Description automatically generated

Hình 51 Code server side phần 7

A computer screen shot of a program

Description automatically generated

Hình 52 Code server side phần 8



Hình 53 Code server side phần 9

A screenshot of a computer

Description automatically generated

Hình 54 Form server

## Client:

A screenshot of a computer program

Description automatically generated

Hình 55 Code client side phần 1

A screenshot of a computer program

Description automatically generated

Hình 56 Code client side

A screenshot of a computer program

Description automatically generated

Hình 57 Code client side phần 3

A computer screen shot of a program

Description automatically generated

Hình 58 Code client side phần 4

A computer screen shot of a program

Description automatically generated

Hình 59 Code client side phần 5

A screenshot of a computer program

Description automatically generated

Hình 60 Code client side phần 6

A screen shot of a computer program

Description automatically generated

Hình 61 Code client phần 7

A computer screen shot of a program

Description automatically generated

Hình 62 Code client side phần 8

A screenshot of a computer

Description automatically generated

Hình 63 Code client side phần 9

A screenshot of a computer

Description automatically generated

Hình 64 Client form

## Giải thích:

## Server:

- Biến **connectedClients** là danh sách **List** chứa các đối tượng **ClientInfo**, mỗi đối tượng này đại diện cho một **client** đã kết nối với **server**. Danh sách này giúp **server** gửi tin nhắn đến tất cả các **client** hoặc **tìm kiếm** một **client** theo tên người dùng.

- Phương thức **Lab03\_Bai05\_Server\_Load** thực hiện:

* **CheckForIllegalCrossThreadCalls = false**: Cài đặt này cho phép truy cập vào các điều khiển giao diện từ các luồng phụ mà **không gây ra lỗi**. Đây là cách đơn giản **để tránh lỗi** khi cập nhật giao diện (ví dụ **richTextBox\_chat**) từ một luồng không phải **luồng chính**.
* Tạo **luồng serverThread**: Luồng này được khởi chạy từ phương thức **StartUnsafeThread**, đảm bảo **server** có thể lắng nghe kết nối từ nhiều client cùng lúc mà không làm treo giao diện người dùng.
* Gán sự kiện **KeyDown** cho **textBox\_send\_chat**: Mỗi khi người dùng nhấn phím trong **textBox\_send\_chat**, sự kiện **textBox\_send\_chat\_KeyDown** sẽ được gọi, hỗ trợ việc gửi tin nhắn nhanh qua phím **Enter**.

- Lớp **ClientInfo** chứa hai thuộc tính: **ClientSocket** đại diện cho kết nối mạng của **client**, **Username** là tên người dùng của **client**. Lớp này dùng để lưu trữ và truy cập dễ dàng thông tin của mỗi **client**, giúp xử lý các kết nối, tìm kiếm **client** theo tên người dùng và phân biệt các **client**.

- Phương thức **check\_username** kiểm tra xem tên người dùng có bị trùng với **client** đã **kết nối** trước đó hay không. Dùng **foreach** để duyệt qua danh sách **connectedClients** để so sánh **Username** và **ClientSocket** của từng **client** theo **điều kiện**: **client\_info.Username == sus\_client.Username** kiểm tra xem có client nào đang dùng tên giống với **sus\_client** không , **client\_info.ClientSocket != sus\_client.ClientSocket** đảm bảo client có tên trùng phải là client khác (**ClientSocket không trùng**), để không so sánh với chính **sus\_client**. Nếu có **tên trùng**, phương thức sẽ trả về **true**, cho biết tên này đã được dùng, giúp các **Client** có tên phân biệt.

- Phương thức **UsernameError** gửi thông báo lỗi nếu tên người dùng bị trùng đến client bị trùng với **byte header** là “4”.

- Phương thức **StartUnsafeThread** giúp **server** luôn luôn lắng nghe kết nối từ **client** mỗi khi **khởi động** server:

- Khởi tạo **listenerSocket**: Đây là **socket lắng nghe**, được cấu hình với **AddressFamily.InterNetwork, SocketType.Stream, và ProtocolType.Tcp** để kết nối TCP qua mạng IPv4.

* **IPEndPoint chứa IP (127.0.0.1 - localhost)** và **cổng (8080)**, giúp server xác định nơi để lắng nghe.
* Khi **kết nối thành công**, socket của **client** được truyền đến một luồng mới để xử lý client, giúp **server tiếp tục lắng nghe** các kết nối khác mà không bị gián đoạn.

- Phương thức **HandleClient** xử lý các **hoạt động** từ **client** và là nơi **server** nhận và xử lý các thông điệp từ **client**:

* Khởi tạo **recv**: Mảng **byte recv** có kích thước đủ lớn để nhận dữ liệu từ **client**, bao gồm cả tin nhắn văn bản và hình ảnh.
* Biến **bytesReceived**: Số lượng byte dữ liệu đã nhận từ **client**.
* Biến **username** và **remove\_username**: Tên người dùng mặc định là "**Unknown**" và **remove\_username** đánh dấu xem có cần xóa tên người dùng này khỏi danh sách khi ngắt kết nối hay không.
* **Vòng lặp while**: Đảm bảo **server liên tục nhận** và **xử lý dữ liệu từ client** cho đến khi kết nối bị ngắt hoặc gặp lỗi.
* Trong vòng lặp while, **server** nhận dữ liệu từ **client** qua **clientSocket.Receive(recv)**, server xác định loại dữ liệu theo **header** là byte đầu tiên trong **mảng recv.**
* **Header 3** - **Ngắt kết nối**: **Server** đọc tên người dùng từ **client** (bắt đầu từ byte thứ hai của recv). Sau khi ngắt kết nối, **server** gửi thông báo người dùng đã rời khỏi chat.
* **Header 2** - **Kết nối mới**: **Server** đọc tên người dùng từ **client**, tạo một đối tượng **ClientInfo** mới, thêm vào **connectedClients** và kiểm tra trùng tên. Nếu trùng, gọi **UsernameError**, nếu không thì gửi thông báo người dùng đã vào chat.
* **Header 6 - Tin nhắn riêng tư**: Server **phân tích nội dung** tin nhắn để gửi riêng tư với phương thức **SendPrivateChat**.
* **Header 0 - Tin nhắn công khai**: Server gọi phương thức **BroadcastMessage** để gửi tin nhắn đến **tất cả client**.
* **Không có header - Dữ liệu là hình ảnh**: **Server** gửi ảnh đến tất cả **client** qua phương thức **BroadcastImage**.
* Khi vòng lặp **while** dừng lại (**ngắt kết nối**), client sẽ bị xóa khỏi **connectedClients** và **listBox\_participants** sẽ được cập nhật, đóng **socket client**

- Phương thức **BroadcastMessage** gửi tin nhắn đến tất cả **client** đã kết nối:

* **formattedMessage**: Format **nội dung tin nhắn** với thời gian và tên người gửi nếu username là "**Server**" hoặc **tin nhắn rỗng** đối với tin nhắn gửi ảnh. Nếu không, tin nhắn sẽ **được giữ** nguyên **nội dung**.
* **messageToSend**: Để gửi tin nhắn đến các **client,** tin nhắn được mã hóa sang dạng **UTF-8** thành **messageBytes**. Byte đầu tiên của **messageToSend** được gán “0”, là **header** xác định đây là tin nhắn văn bản công khai.
* **Vòng lặp gửi tin**: Gửi **messageToSend** đến tất cả **client** đã kết nối. Nếu **client** gặp lỗi kết nối (**SocketException**), **MessageBox** thông báo lỗi sẽ xuất hiện.
* Cập nhật **richTextBox\_chat**: Tin nhắn cũng được hiển thị trên giao diện **server** bằng **AppendText**, giúp người dùng dễ dàng theo dõi.

- Phương thức **BroadcastImage** gửi ảnh đến **tất cả client**:

* **clientInfo.ClientSocket.Send(imageBytes)**: trong vòng lặp để gửi **imageBytes** (**dữ liệu ảnh**) đến tất cả **client** đã kết nối.
* Hiển thị ảnh trên **richTextBox\_chat**: Phương thức **InsertImageToChatBox** hiển thị ảnh lên giao diện **server**. Gọi **BroadcastMessage** để gửi tin nhắn **rỗng** giúp cập nhật tên người gửi là **username**.

- Phương thức **button\_send\_Click** xử lý sự kiện khi người dùng **server** nhấn nút gửi tin nhắn văn bản từ **server**:

* Kiểm tra **tin nhắn rỗng**: Chỉ gửi tin nhắn nếu nội dung textBox\_send\_chat không trống.
* Gửi **tin nhắn**: **BroadcastMessage** được gọi với **username** là "**Server**" và nội dung **chat\_text**.
* Xóa nội dung **textBox\_send\_chat**: Sau khi gửi, **textBox\_send\_chat** sẽ được xóa để người dùng **nhập nội dung mới**.

- Phương thức **button\_send\_files\_Click** xử lý khi người dùng server chọn gửi file từ server đến client:

* **Lọc file**: **openFileDialog** chỉ hiển thị file văn bản .txt và các định dạng ảnh (**jpeg, jpg, png, gif**).
* **Gửi file**: File văn bản được đọc và gửi dưới dạng tin nhắn với **BroadcastMessage**, file ảnh được đọc toàn bộ thành **imageBytes** và gửi với **BroadcastImage**.
* Phương thức **InsertImageToChatBox** chèn ảnh vào **richTextBox\_chat** của server:
* **InvokeRequired**: Kiểm tra nếu cần thực hiện **Invoke** khi đang chạy trong một luồng khác ngoài luồng giao diện.
* Chèn ảnh vào **Clipboard**: Sao chép ảnh vào bộ nhớ tạm (**Clipboard**) rồi **Paste** vào **richTextBox\_chat**, tạo cảm giác ảnh được gửi trực tiếp vào khung chat.

- Phương thức **textBox\_send\_chat\_KeyDown** xử lý sự kiện khi người dùng server nhấn phím trong **textBox\_send\_chat**: Gọi **button\_send.PerformClick()** để gửi tin nhắn khi nhấn Enter. Nếu **Shift + Enter**, con trỏ sẽ xuống dòng thay vì gửi tin.

- Phương thức **UpdateParticipantList** dùng để cập nhật danh sách **participant** cho các client hiển thị trên UI mỗi khi có một **client** kết nối hoặc **client** ngắt kết nối, các **client** có sự cố kết nối hoặc trùng tên sẽ không được cập nhật:

* Tạo một chuỗi danh sách người dùng hiện tại bằng cách lặp qua các clientInfo trong **connectedClients**. Phương thức **AppendLine** thêm tên người dùng vào chuỗi và xuống dòng sau mỗi tên.
* Sử dụng vòng lặp để gửi **messageToSend** với header là “5” đến tất cả client đang kết nối. Nếu xảy ra lỗi (**SocketException**), một thông báo sẽ hiển thị để báo lỗi gửi tin nhắn.

- Phương thức **SendPrivateChat** thực hiện gửi tin nhắn riêng đến một client cụ thể. Duyệt qua **connectedClients** để tìm **client** có tên người dùng (**Username**) trùng với target. Gửi tin nhắn với header là “6” cho client có tên trùng với target.

## Client

- Khởi tạo **form client**:

* Với các biến: **clientSocket, receiveThread, privateChatTarget, private\_message**
* Phương thức **Lab03\_Bai05\_Client\_Load** gán sự kiện nhấn nút **enter** để gửi chat và khi đóng form thực hiện phương thức **Lab03\_Bai05\_Client\_FormClosing** để ngắt kết nối tới server

- Phương thức **SendImage** tương tự phương thức **BroadcastImage** của **Server** nhưng chỉ gửi hình cho **server**.

- Phương thức **button\_send\_Click** tương tụ **button\_send\_Click** của **server** nhưng chỉ gửi tin nhắn cho server qua phương thức **SendMessage**.

- Phương thức **button\_send\_files\_Click** tương tự phương thức **button\_send\_files\_Click** của server nhưng chỉ gửi hình cho server qua phương thức SendImage.

- Phương thức **InsertImageToChatBox** tương tự phương thức **InsertImageToChatBox** của server.

- Phương **thức textBox\_send\_chat\_KeyDown** tương tự phương thức **textBox\_send\_chat\_KeyDown** cửa server.

- Phương thức **UpdateParticipantList** cập nhật danh sách những người tham gia trong **listBox\_participants**:

* Nếu phương thức được gọi từ một **luồng khác** luồng giao diện, **InvokeRequired** sẽ được dùng để thực hiện cập nhật trên luồng giao diện.
* **Danh sách người dùng** tham gia được xóa và các tên người dùng mới sẽ được thêm vào từ dữ liệu nhận từ server (**participants**), mỗi dòng một tên.

- Phương thức **button\_private\_chat\_Click** để gửi tin nhắn riêng tư khi nhấn nút “**PM**”:

* **Kiểm tra** xem người dùng **đã chọn** một người tham gia trong **listBox\_participants** và đã nhập nội dung trong **textBox\_send\_chat** chưa.
* Nếu thỏa mãn, **biến privateChatTarget** sẽ được đặt **tên** của người dùng được chọn làm mục tiêu cho tin nhắn riêng tư.
* Phương thức **SendMessage** sẽ gửi tên người dùng với **mã "2"** để gửi tin nhắn **riêng tư** đến người **tham gia** đã chọn.
* Cuối cùng, một **thông báo** sẽ xuất hiện để **xác nhận** rằng tin nhắn đã được gửi **riêng tư**. Nếu người dùng chưa chọn người **tham gia** hoặc chưa nhập tin nhắn, một cảnh báo sẽ xuất hiện.

- Phương thức **button\_connect\_Click** để kết nối client với server khi nhấn nút “**Connect**”:

* Phương thức tạo một kết nối **socket TCP** giữa **client và server** bằng cách sử dụng địa chỉ **IP (127.0.0.1)** và cổng **(8080)**.
* Sau khi kết nối thành công, một luồng (**receiveThread**) sẽ được khởi chạy để nhận dữ liệu từ server.
* Nếu tên người dùng (**textBox\_username.Text**) trống, tên sẽ được mặc định là "**Unknown**".
* Phương thức **SendMessage** được gọi để gửi tên người dùng đến với **mã "1"** nhằm **thông báo** rằng một **người dùng mới** đã kết nối.
* Giao diện của người dùng sẽ được cập nhật: nút kết nối bị vô hiệu hóa, không được đổi tên người dùng khi đã kết nối, và các nút **"Ngắt kết nối," "Gửi tin nhắn," "Gửi file," "Chat riêng"** được kích hoạt.

- Phương thức **button\_disconnect\_Click** để ngắt **kết nối client với server** khi nhấn nút “**Disconnect**”:

* Phương thức này **ngắt kết nối socket** với **server** nếu nó đang được kết nối.
* Luồng nhận (**receiveThread**) sẽ đóng và một tin nhắn với **mã "0"** được gửi đến **SendMessage** để thông báo rằng người dùng cần ngắt kết nối.
* **Socket** sẽ được đóng hoàn toàn và đặt về null cho an toàn.
* Sau khi ngắt kết nối, giao diện của người dùng sẽ được cập nhật để kích hoạt lại nút kết nối, cho phép người dùng thay đổi tên và vô hiệu hóa các nút liên quan đến tin nhắn và chat.

- Phương thức **SendMessage** để gửi tin nhắn client cho server:

* Tạo một chuỗi **timestamp** để đánh dấu thời gian gửi tin nhắn.
* Xác định kiểu tin nhắn dựa vào **giá trị message**
* Nếu message là **"0"**: Gửi thông báo ngắt kết nối**. messageToSend[0] = 3** là phần đầu (header) báo cho server biết đây là tin nhắn ngắt kết nối.
* Nếu message là **"1":** Gửi tên người dùng khi kết nối, với **messageToSend[0] = 2** là header để thông báo đây là tin nhắn kết nối.
* Nếu message là **"2"**: Gửi tin nhắn riêng tư đến **privateChatTarget**. Nội dung tin nhắn bao gồm tên người nhận và dấu **"<private>"** để đánh dấu đây là tin nhắn riêng tư, và **messageToSend[0] = 6** là header cho biết loại tin nhắn này.
* Các trường hợp khác: Được xử lý như tin nhắn công khai. Tin nhắn sẽ bao gồm dấu thời gian và tên người dùng, với **messageToSend[0] = 0** là header.
* Chuỗi tin nhắn được **mã hóa sang byte** và gửi qua socket. Nếu xảy ra lỗi trong quá trình gửi, thông báo lỗi sẽ hiển thị.

- Phương thức **ReceiveData** để nhận thông điệp từ **server** và được chạy trên một luồng riêng để liên tục nhận dữ liệu từ **server**:

* Tạo một bộ đệm (**recvBuffer**) để lưu dữ liệu nhận từ server và một biến kiểm tra kết nối (**check\_connect**).
* Vòng lặp while hoạt động khi **clientSocket** chưa bị ngắt kết nối.
* Sử dụng **clientSocket.Receive(recvBuffer)** để nhận **byte dữ liệu từ server**.
* Nếu **header là 4**: Tin nhắn ngắt kết nối từ **server** khi username bị trùng. **check\_connect** chuyển thành **false**, hiển thị tin nhắn thông báo **trùng username** và tự động thực hiện ngắt kết nối (**button\_disconnect.PerformClick**).
* Nếu **header là 5**: Cập nhật danh sách người tham gia, gọi phương thức UpdateParticipantList để cập nhật **listBox\_participants**.
* Nếu **header là 0 hoặc 6**: Đây là tin nhắn văn bản (công khai hoặc riêng tư). Tin nhắn sẽ được giải mã từ byte sang chuỗi và thêm vào **richTextBox\_chat** cho người dùng thấy.
* Các **trường hợp khác**: Xử lý dữ liệu ảnh và gọi **InsertImageToChatBox** để hiển thị hình ảnh trong hộp chat.
* Nếu **lần đầu** nhận dữ liệu mà **check\_connect** vẫn là **true**, một thông báo **kết nối** thành công sẽ hiển thị.
* Xử lý **ngoại lệ**: Nếu xảy ra lỗi kết nối (**SocketException**), thông báo **lỗi** sẽ xuất hiện và vòng lặp bị **ngắt**.

**Link Google Drive:**

[**https://8gcf1d-my.sharepoint.com/:f:/g/personal/posu29\_8gcf1d\_onmicrosoft\_com/EiZ\_uYs7YHlDlwqbLUrwWjcB8DMeo\_CUpabNa45SLl0z3w?e=ZlQbwJ**](https://8gcf1d-my.sharepoint.com/:f:/g/personal/posu29_8gcf1d_onmicrosoft_com/EiZ_uYs7YHlDlwqbLUrwWjcB8DMeo_CUpabNa45SLl0z3w?e=ZlQbwJ&fbclid=IwZXh0bgNhZW0CMTAAAR02Smo74Z5L88DDZLp-RlEZwfgmUlOYWLTqqZvM2bpYrDA6JKS2xXSkJF0_aem_jzoG_9XviZfjXp_jwsUMsg)

**YÊU CẦU CHUNG**

1. Đánh giá

* Chuẩn bị tốt các yêu cầu đặt ra trong bài thực hành.
* Sinh viên hiểu và tự thực hiện được bài thực hành, trả lời đầy đủ các yêu cầu đặt ra.
* Nộp báo cáo kết quả chi tiết những đã thực hiện, quan sát thấy và kèm ảnh chụp màn hình kết quả *(nếu có)*; giải thích cho quan sát *(nếu có)*.
* Sinh viên báo cáo kết quả thực hiện và nộp bài.

1. Báo cáo

* Nộp file .docx. Tập trung vào nội dung, giải thích.
* Nội dung trình bày bằng Font chữ Cambria hoặc Times New Roman (*tuy nhiên, phải chuyển đổi hết báo cáo này sang 1 font chữ thống nhất*) – cỡ chữ 13. Canh đều (Justify) cho văn bản. Canh giữa (Center) cho ảnh chụp.
* Đặt tên theo định dạng: Mã lớp-LabX\_MSSV1\_MSSV2. (trong đó X là Thứ tự buổi Thực hành).

Ví dụ: NT106.O21.1-Lab01\_25520001\_25520002

* Nếu báo cáo có nhiều file, nén tất cả file vào file .ZIP với cùng tên file báo cáo.
* Không đặt tên đúng định dạng – yêu cầu, sẽ KHÔNG chấm điểm bài thực hành.
* Nộp file báo cáo trên theo thời gian đã thống nhất tại courses.uit.edu.vn.

Bài sao chép, trễ, … sẽ được xử lý tùy mức độ vi phạm.

**HẾT**