

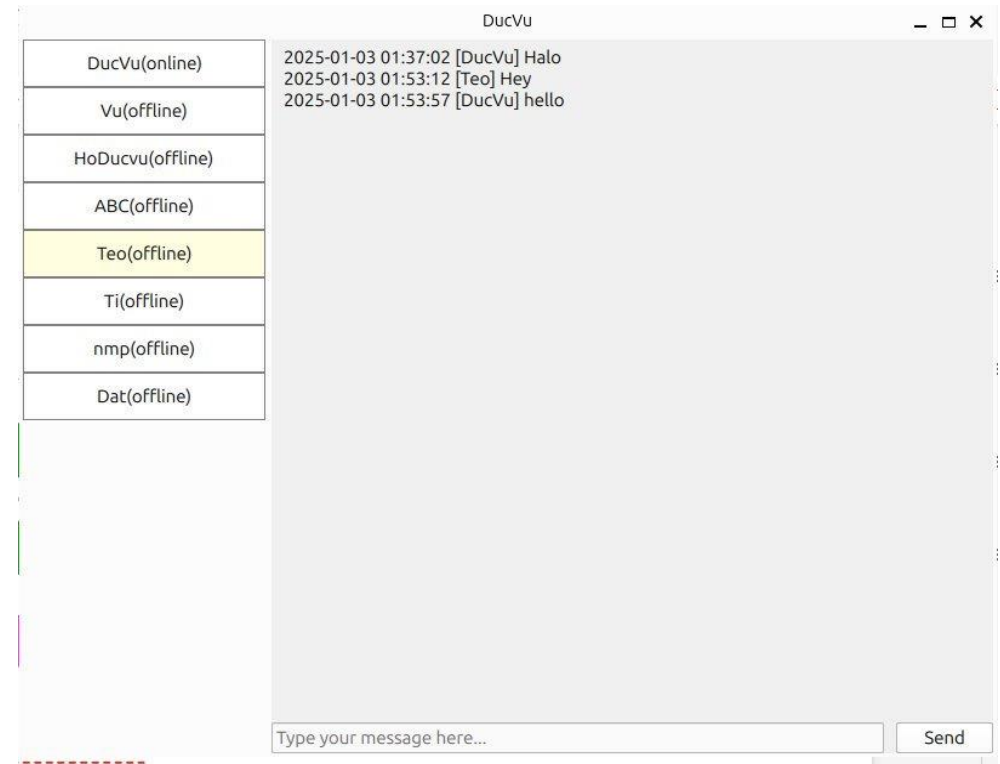
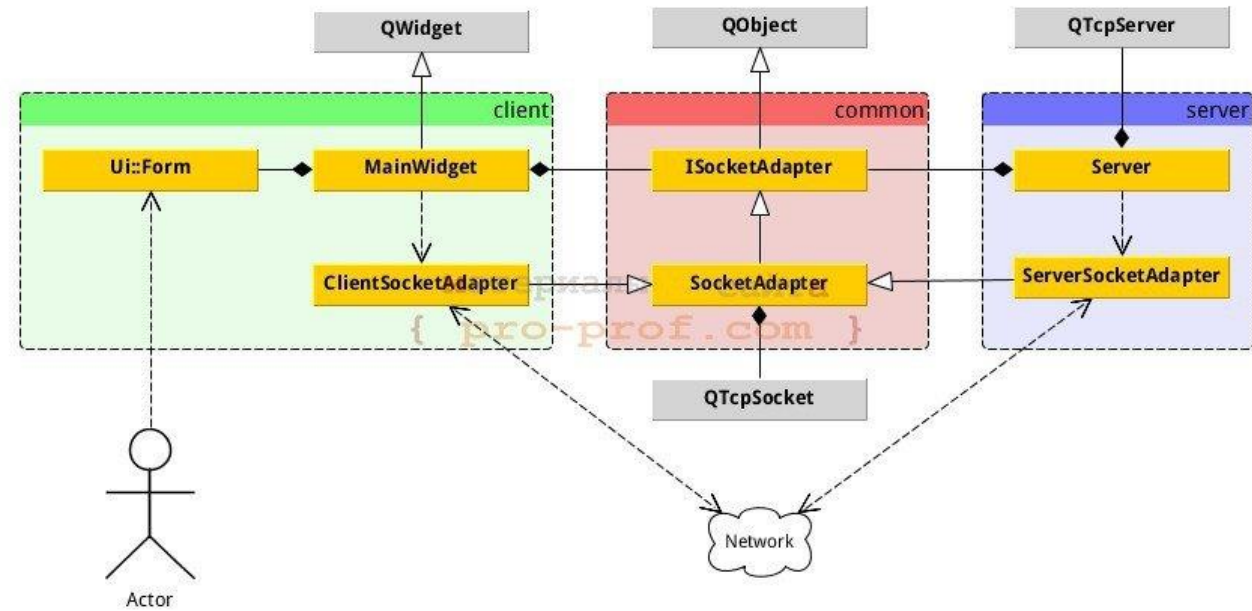
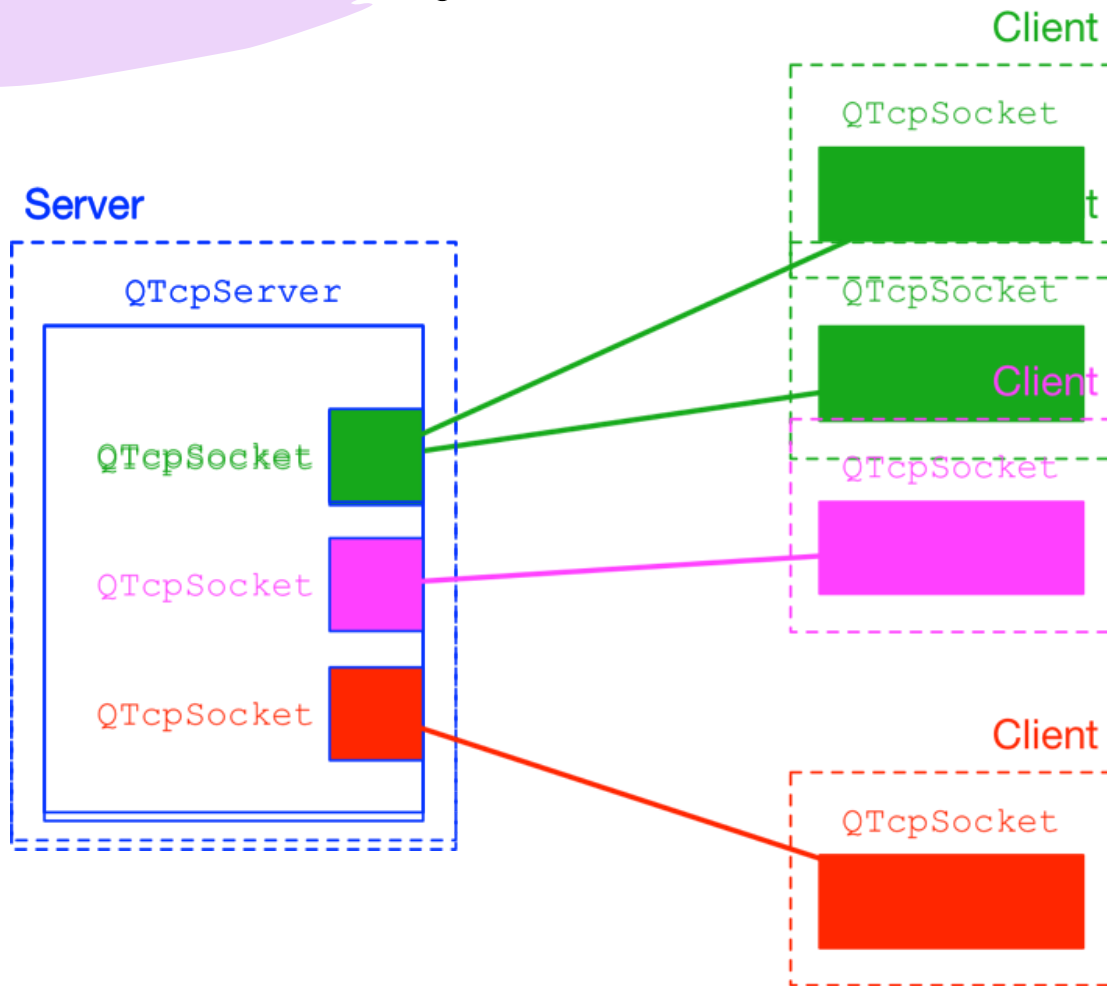
Chat App (Qt)

THÀNH VIÊN: HỒ ĐỨC VŨ - NGUYỄN MINH PHƯƠNG

Outline

1. Giới thiệu
2. Phân tích ứng dụng (Server side)
3. Phân tích ứng dụng (Client side)
4. Kết nối Server – Client
5. Hướng ứng dụng Chat App vào đề tài "Giám sát và kiểm tra tải hệ thống IVI"
6. Kết luận và Câu hỏi

Giới thiệu



Server Interfaces

Lưu trữ thông tin người dùng và nội dung chat

Danh sách Client kết nối với Server

Khởi tạo và kết nối database

Xử lý messages client gửi đến Server

Gửi danh sách clients đang kết nối (online) và có trong database (offline)

Gửi messages đến client

```
ChatServer.cpp

struct ClientInfo {
    QString username;
    QTcpSocket *socket;
};

class ChatServer : public QTcpServer {
    Q_OBJECT
public:
    ChatServer(QObject *parent = nullptr);
    bool startServer(quint16 port);

protected:
    void incomingConnection(qintptr socketDescriptor) override;
private slots:
    void onReadyRead();
    void onDisconnected();
private:
    QSqlDatabase db;
    QMap<QTcpSocket*, ClientInfo> clients;
    bool initializeDatabase();
    void handleMessage(QTcpSocket *client, const QString &message);
    void sendClientList();
    void sendMessageToClient(QTcpSocket *client, const QString &message);
    void broadcastMessage(const QString &message, QTcpSocket *excludeClient = nullptr);
    QString authenticateUser(QTcpSocket *client, const QStringList &credentials);
    void sendChatHistory(QTcpSocket *client, const QString &withUser);
};
```

Server Interfaces

Nhận messages từ client thông qua kết nối signal **readyRead** của **QTcpSocket**

Nhận tín hiệu ngắt kết nối đến client thông qua kết nối signal **disconnected** của **QTcpSocket**, thực hiện giải phòng kết nối client và cập nhập database

Lọc socket null và client chưa xác thực

Xác thực và đăng ký tài khoản người dùng

Gửi lịch sử chat đến người dùng

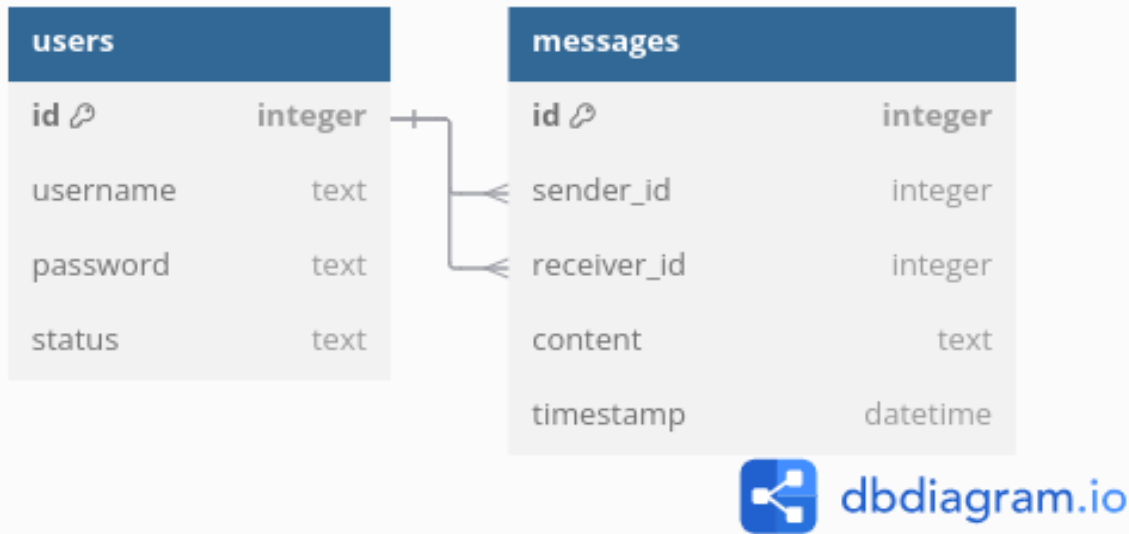
```
ChatServer.cpp

struct ClientInfo {
    QString username;
    QTcpSocket *socket;
};

class ChatServer : public QTcpServer {
    Q_OBJECT
public:
    ChatServer(QObject *parent = nullptr);
    bool startServer(quint16 port);

protected:
    void incomingConnection(qintptr socketDescriptor) override;
private slots:
    void onReadyRead();
    void onDisconnected();
private:
    QSqlDatabase db;
    QMap<QTcpSocket*, ClientInfo> clients;
    bool initializeDatabase();
    void handleMessage(QTcpSocket *client, const QString &message);
    void sendClientList();
    void sendMessageToClient(QTcpSocket *client, const QString &message);
    void broadcastMessage(const QString &message, QTcpSocket *excludeClient = nullptr);
    QString authenticateUser(QTcpSocket *client, const QStringList &credentials);
    void sendChatHistory(QTcpSocket *client, const QString &withUser);
};
```

Database



- Sử dụng SQLite để triển khai database cho hệ thống
- Ứng dụng framework `<QtSql/QtSqlDatabase>`, `<QtSql/QtSqlQuery>` và `<QtSql/QtSqlError>` để tương tác giữa SQLite với Qt framework.

```
QtSqlQuery query;
query.prepare("UPDATE users SET status=:status
              WHERE username=:username");
query.bindValue(":status", "offline");
query.bindValue(":username", username);
query.exec();
```

```
QtSqlQuery query;
query.prepare("SELECT username, status
              FROM users WHERE status=:status");
query.bindValue(":status", "offline");
query.exec();
while (query.next())
{
    QJsonObject userObj;
    userObj["username"] = query.value(0).toString();
    userObj["status"] = query.value(1).toString();
    array.append(userObj);
}
```

Khởi động Server

B1: Server khởi tạo/kết nối đến database

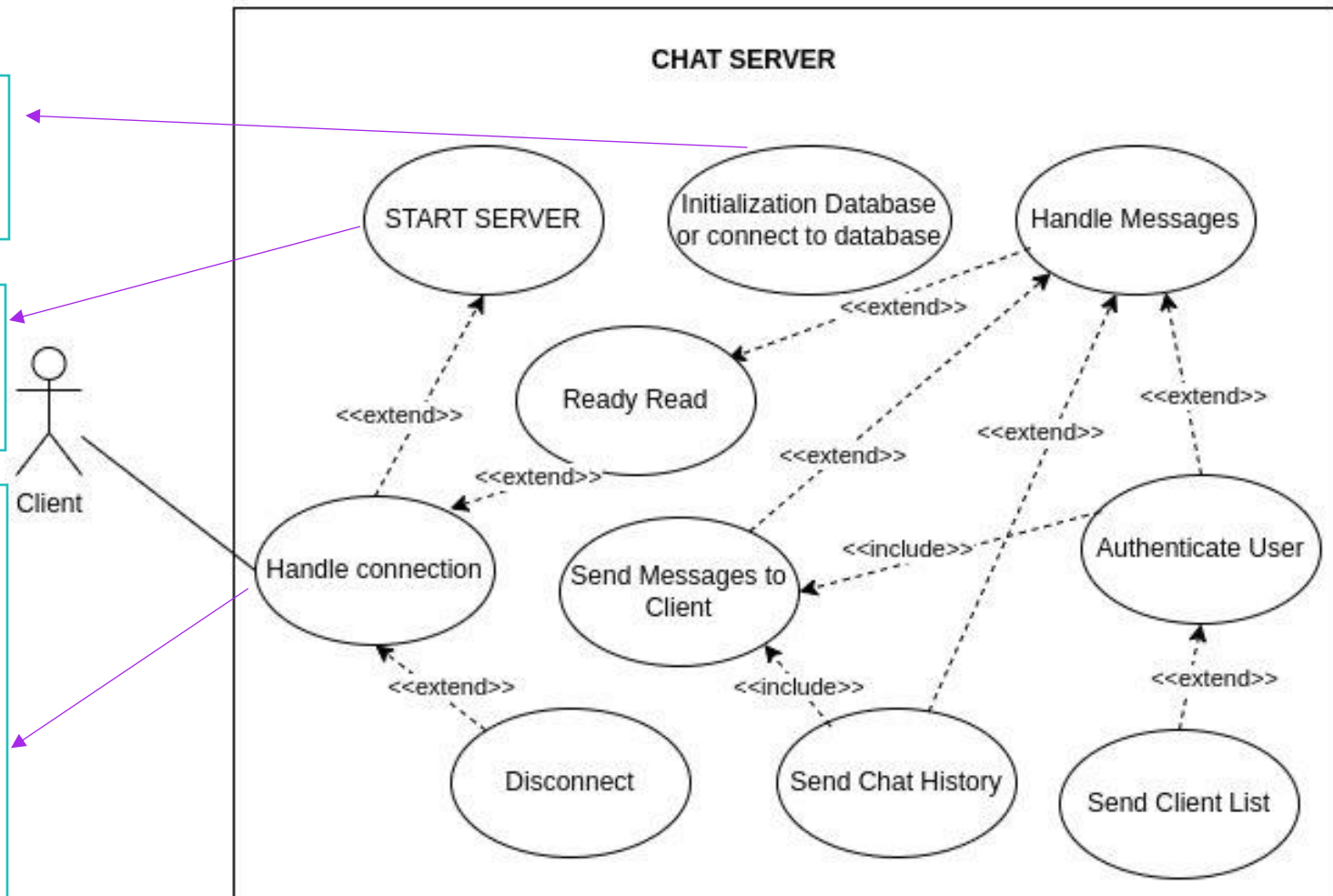
```
db = QSqlDatabase::addDatabase("QSQLITE");  
db.setDatabaseName("chat_server.db");
```

B2: Server được khởi động trong mạng cục bộ với port xác định và lắng nghe kết nối của client:

```
listen(QHostAddress::Any, port)
```

B3: Khi client kết nối đến Server, Server sẽ tạo ra một kết nối client mới và thực hiện kết nối các signals và slots.

```
clients.insert(clientSocket, info);  
connect(clientSocket, &QTcpSocket::readyRead,  
this, &ChatServer::onReadyRead);  
connect(clientSocket, &QTcpSocket::disconnected,  
this, &ChatServer::onDisconnected);
```



Server - Xử lý logic

```
handleMessages.json

messages: {
  type: ["AUTH", "MESSAGE", "GET_HISTORY"],
  type["AUTH"]: {
    username,
    password,
    action: ["REGISTER", "LOGIN"]
  },
  type["MESSAGE"]: {
    to, // id user
    content, // content of message
  }
  type["GET_HISTORY"]: {
    with, // username
  }
}
```

If type == AUTH

if action == REGISTOR

B1: Lưu username và password vào user table

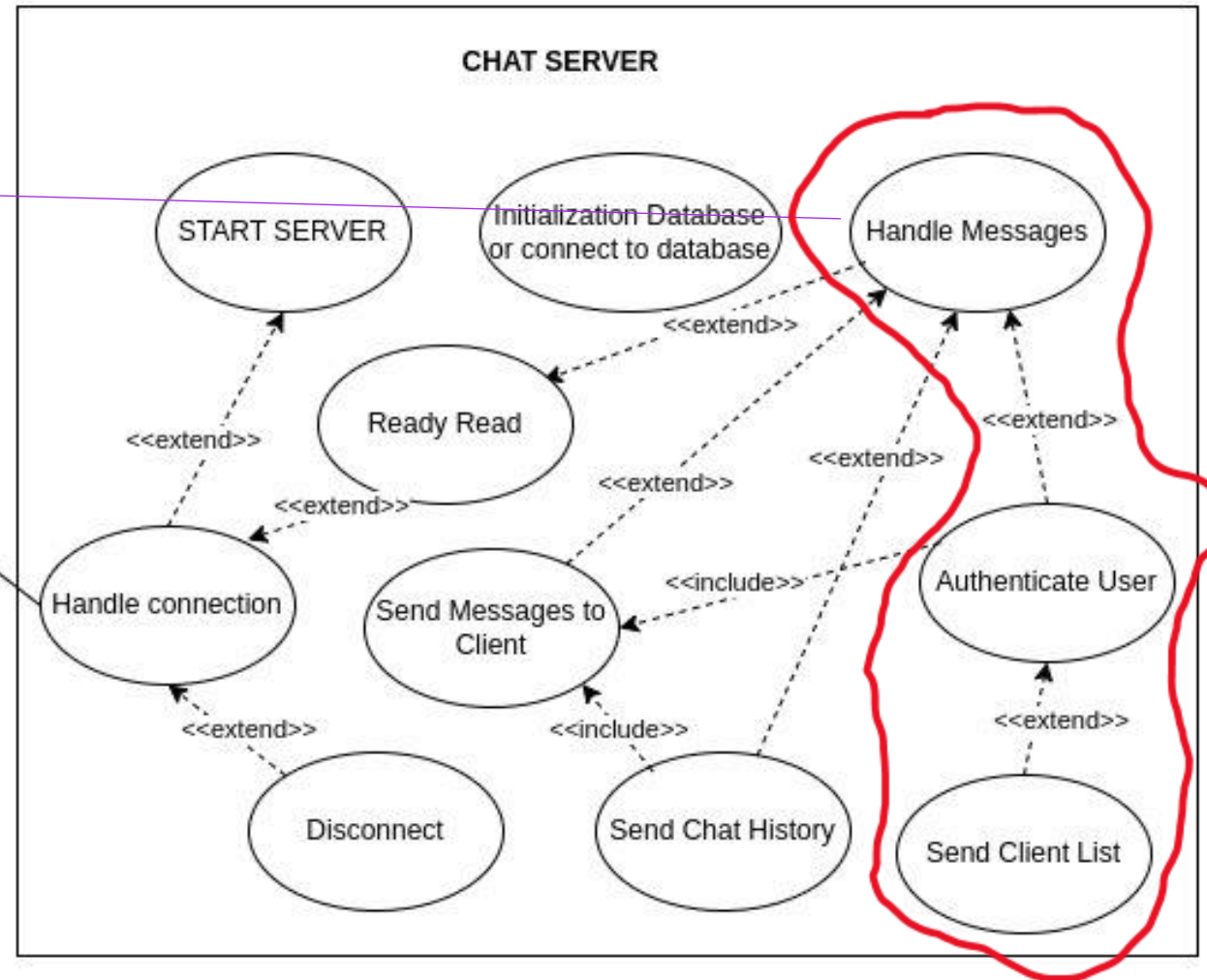
B2: Gửi thông điệp đăng ký thành công đến client

else if acction == LOGIN

B1: Truy xuất username và password trong users table

B2: Nếu tồn tại thì gửi thông điệp đăng nhập thành công đến client

SendClientList()



Server - Xử lý logic

If type == MESSAGE

B1: Truy xuất sender_id và receiver_id từ users table
// Với sender_id là client và receiver_id là "to"

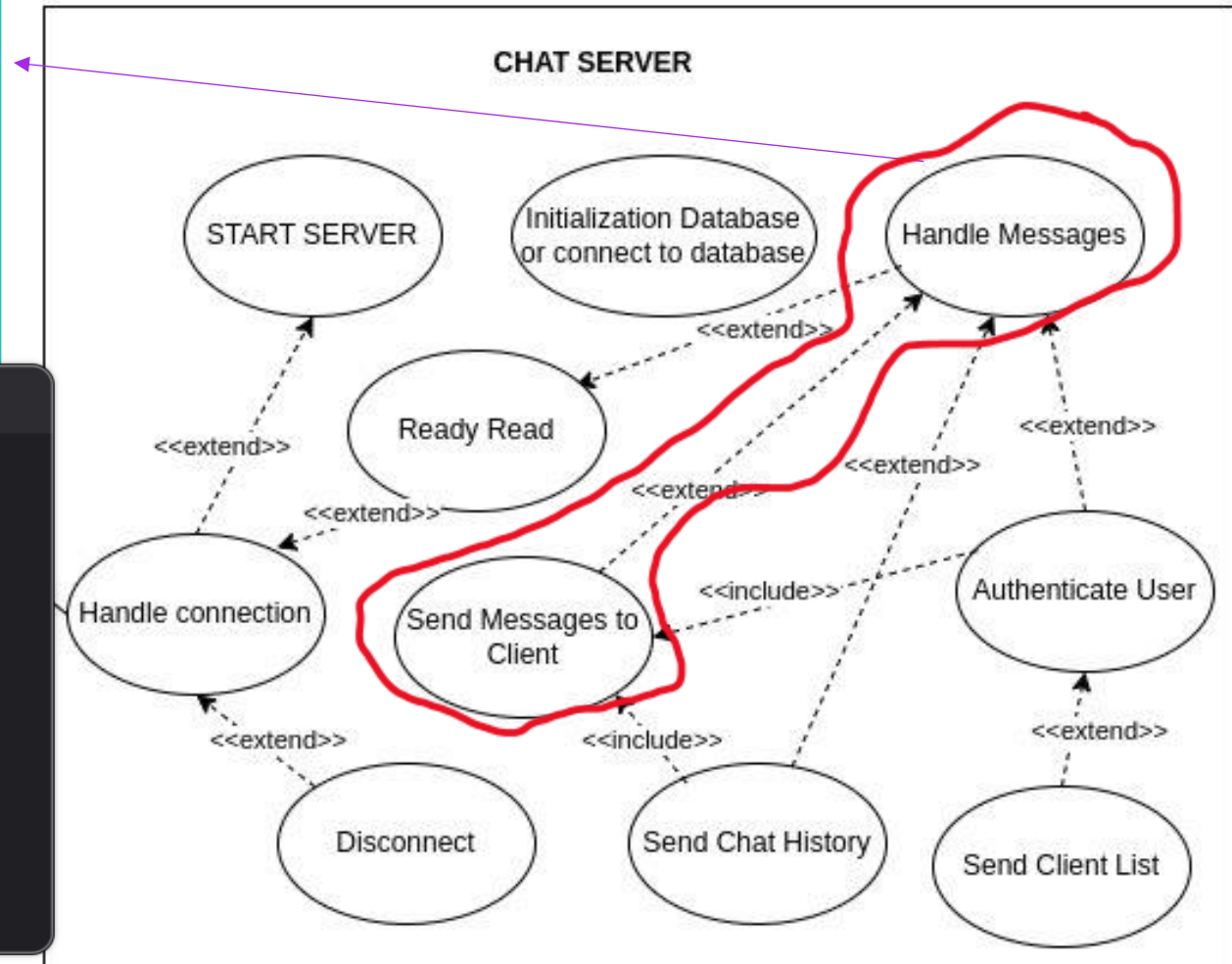
B2: Lưu message vào messages table với 3 thuộc tính:
sender_id, receiver_id và content

B3: Kiểm tra các clients đang kết nối == "to"

```
msg: {  
  "type": "MESSAGE"  
  "from": sender  
  "content": content  
}
```

B4: Gửi msg đến "to"

```
handleMessages.json  
  
messages: {  
  type: ["AUTH", "MESSAGE", "GET_HISTORY"],  
  type["AUTH"]: {  
    username,  
    password,  
    action: ["REGISTER", "LOGIN"]  
  },  
  type["MESSAGE"]: {  
    to, // id user  
    content, // content of message  
  },  
  type["GET_HISTORY"]: {  
    with, // username  
  }  
}
```



Server - Xử lý logic

If type == GET_HISTORY

B1: Truy xuất current user id và with user id từ users table
// current user id = client, with user = "with"

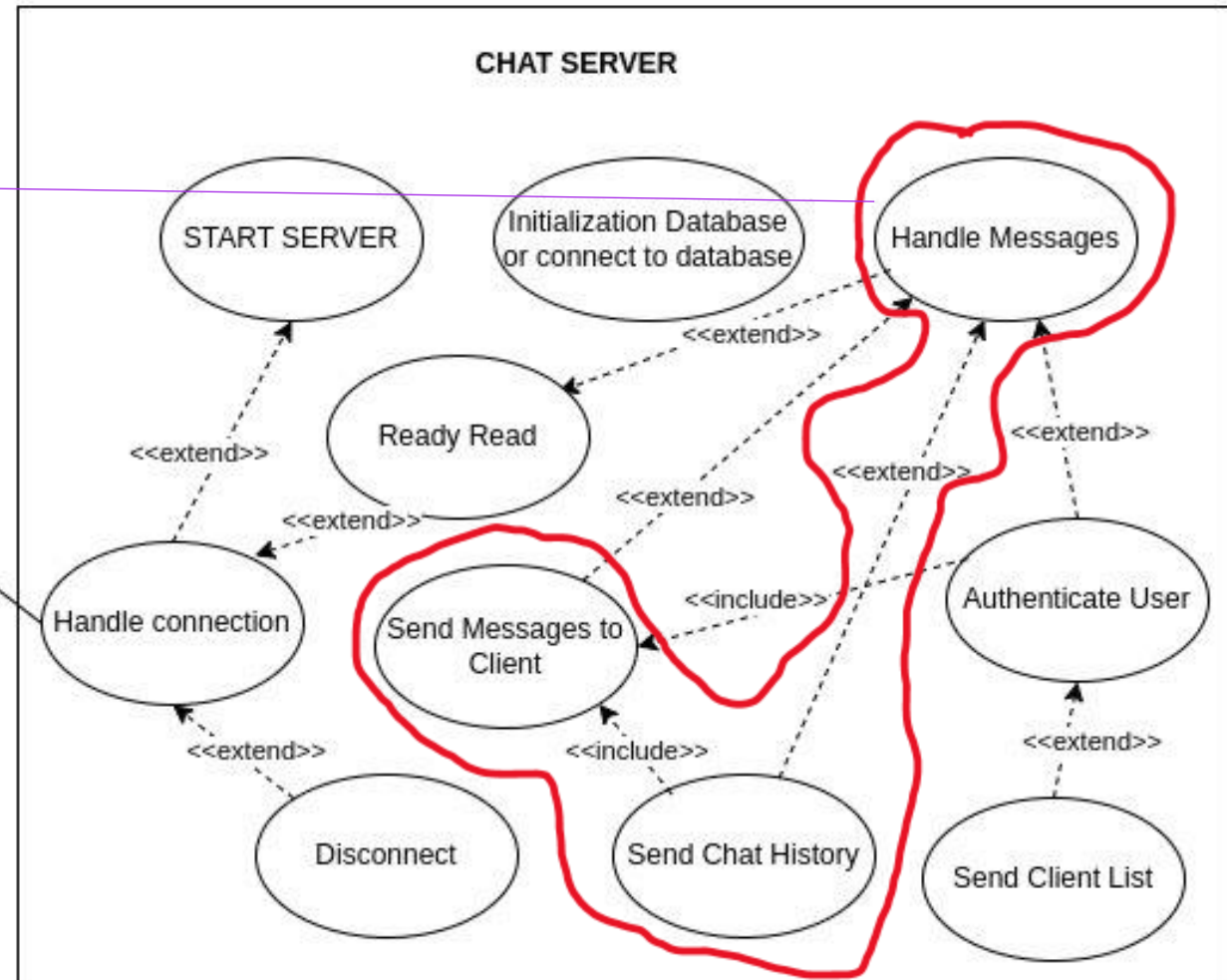
B2: Truy xuất username, content, timestamp từ messages table với sender_id = client.id, receiver_id = "with".id hoặc đổi lại.

B3: Truy xuất toàn bộ messages với cấu trúc:

```
msg: {  
  sender,  
  content,  
  timestamp  
}
```

B4: Gửi đến client

```
handleMessages.json  
  
messages: {  
  type: ["AUTH", "MESSAGE", "GET_HISTORY"],  
  type["AUTH"]: {  
    username,  
    password,  
    action: ["REGISTER", "LOGIN"]  
  },  
  type["MESSAGE"]: {  
    to, // id user  
    content, // content of message  
  },  
  type["GET_HISTORY"]: {  
    with, // username  
  }  
}
```



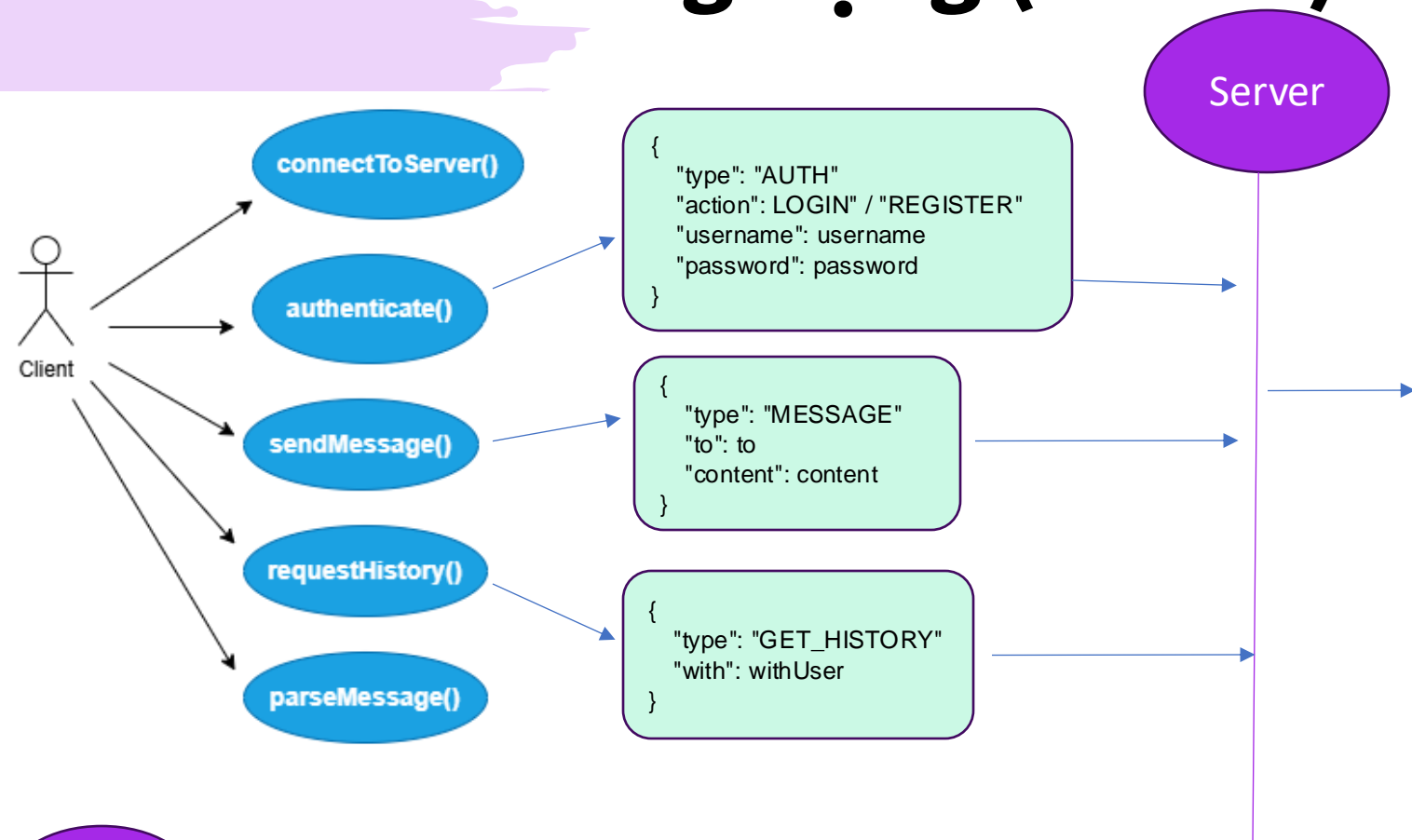
Client Interfaces

Những thuộc tính này thường được sử dụng trong **QML** để liên kết (binding) giao diện người dùng với dữ liệu trong C++

```
class ChatClient : public QObject
{
    Q_OBJECT
    Q_PROPERTY(QStringList messages READ messages NOTIFY messagesChanged FINAL)
    Q_PROPERTY(QStringList users READ users NOTIFY usersChanged FINAL)
public:
    explicit ChatClient(QObject *parent = nullptr);
    Q_INVOKABLE void connectToServer(const QString &host, quint16 port);
    Q_INVOKABLE void sendMessage(const QString &to, const QString &content);
    Q_INVOKABLE void authenticate(const QString &action, const QString &username,
                                   const QString &password);
    Q_INVOKABLE void requestHistory(const QString &withUser);

    QStringList messages() const;
    QStringList users() const;
signals:
    void messagesChanged();
    void usersChanged();
    void connectedToServer();
    void disconnectedFromServer();
    void authenticationResult(bool success, const QString &message);
private slots:
    void onConnected();
    void onReadyRead();
    void onDisconnected();
private:
    QTcpSocket *socket;
    QStringList m_message;
    QStringList m_users;
    void parseMessage(const QString &message);
};
```

Phân tích ứng dụng (Client)



QML

```
function onAuthenticationResult(success) {
    if (success) {
        stackView.push(chatPage)
    }
}
```

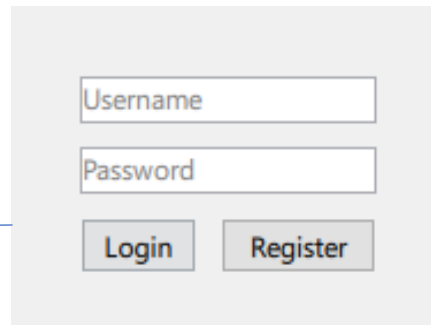
```
function onMessagesChanged(){
    chatClient.requestHistory(username)
}
```

```
if(message == "LOGIN_SUCCESS"){
    emit authenticationResult(true, "Login successful");
}
if(message == "REGISTER_SUCCESS"){
    emit authenticationResult(true, "Register successful");
}
if (type == "CLIENT_LIST") {
    // Lấy tất cả user
    obj["users"]
    // "users": [ { "username": "user1", "status": "online" }, ... ]
    emit usersChanged();
}
if(type == "MESSAGE") {
    // Lấy tên người gửi
    obj["from"]
    // nội dung tin nhắn
    obj["content"]
    // Lưu lại message lấy về từ server
    emit messagesChanged();
}
if(type == "CHAT_HISTORY") {
    // Lấy tất cả message mà server gửi về
    obj["history"]
    // "history": [ { "timestamp": "...", "sender": "..", "content":
    "...", }, ... ]
    emit messagesChanged();
}
```

Phân tích ứng dụng (Client)

authenticate(action, username, password)

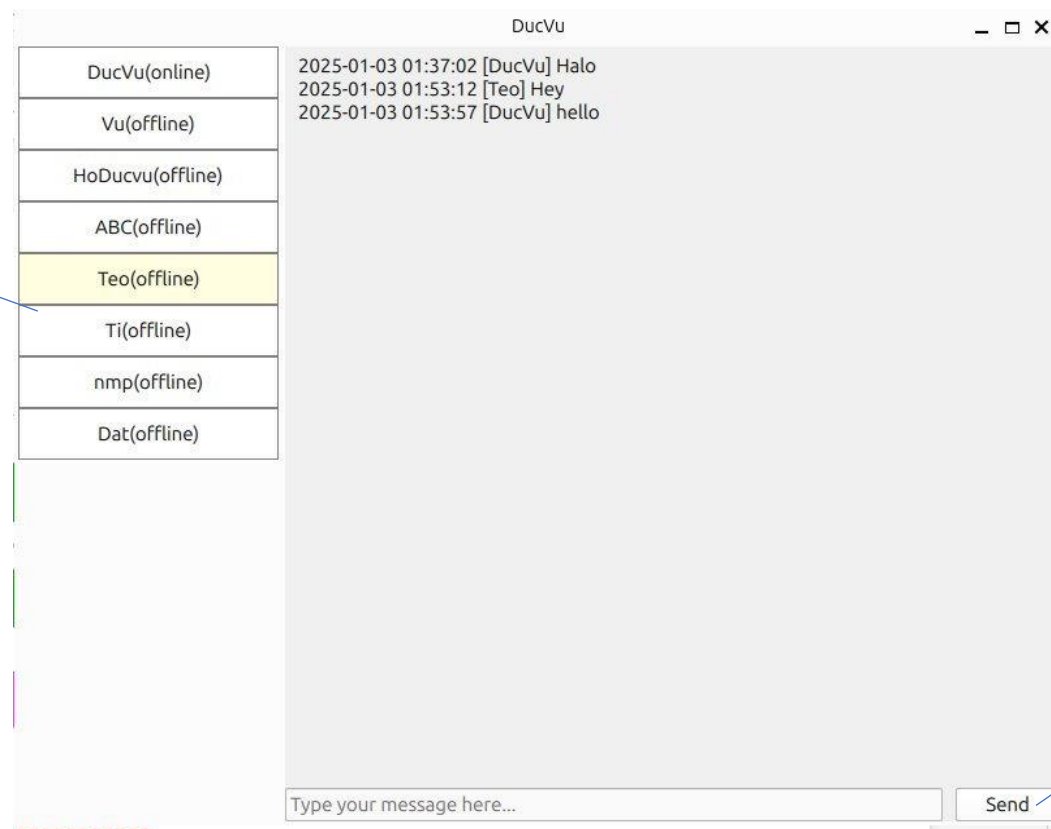
click



A login and registration form with two input fields: 'Username' and 'Password'. Below the fields are two buttons: 'Login' and 'Register'.

Gọi hàm **requestHistory(withUser)**

click

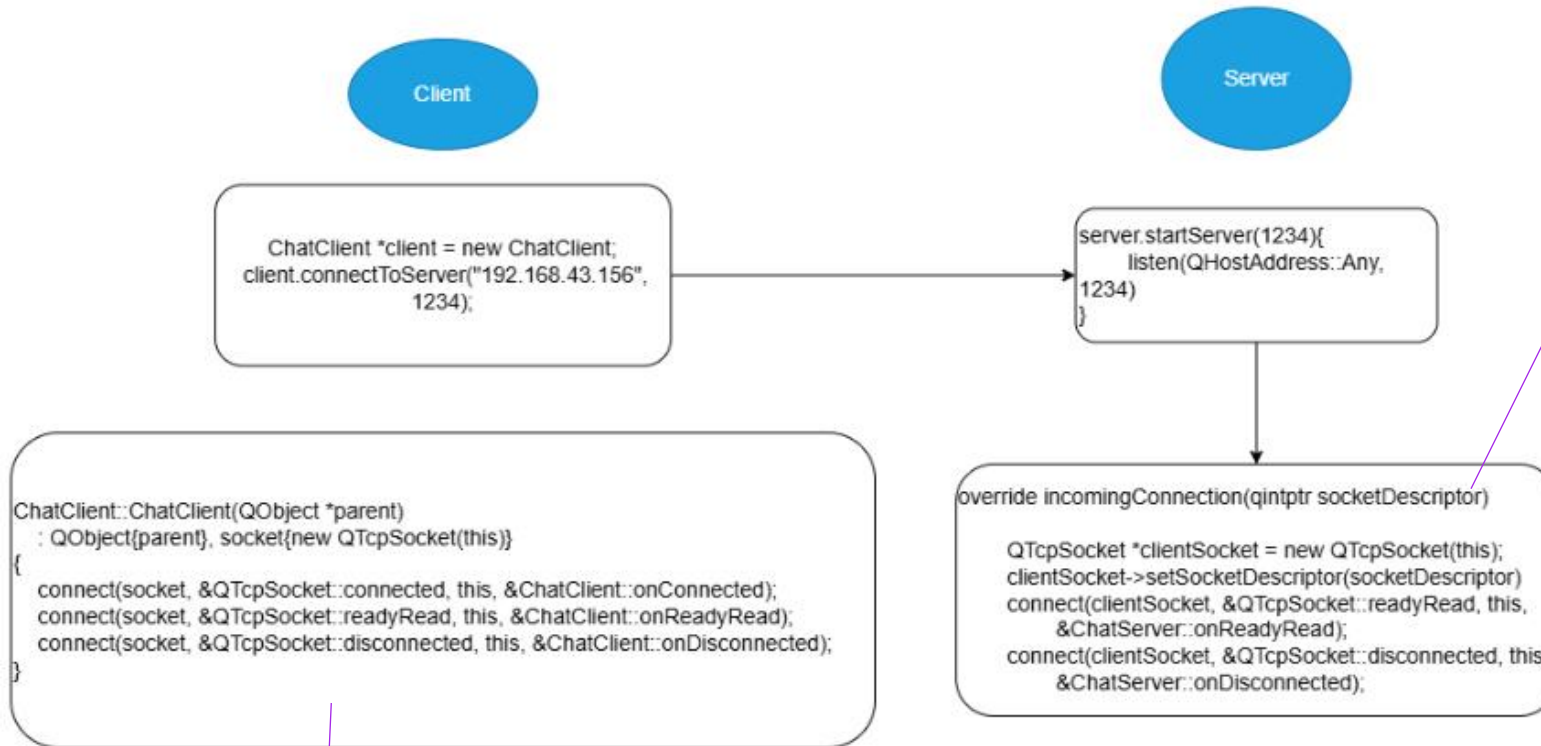


A chat application window titled 'DucVu'. On the left is a list of users: DucVu(online), Vu(offline), HoDucvu(offline), ABC(offline), Teo(offline), Ti(offline), nmp(offline), and Dat(offline). The 'Teo(offline)' user is highlighted in yellow. On the right is a chat history area showing messages: '2025-01-03 01:37:02 [DucVu] Halo', '2025-01-03 01:53:12 [Teo] Hey', and '2025-01-03 01:53:57 [DucVu] hello'. At the bottom is a text input field 'Type your message here...' and a 'Send' button.

Gọi hàm **sendMessage(to, content)**

click

Kết nối giữa Server - Client



Server sẽ nhận một **socket descriptor** -> Server nhận kết nối từ một client thông qua **socket descriptor**, đại diện cho kết nối đó.

phương thức ảo (virtual) trong lớp **QTcpServer** của Qt, -> xử lý các kết nối đến từ client khi một server đang lắng nghe (listening) trên một cổng mạng.

- sử dụng **QTcpSocket::write()** để gửi dữ liệu
- sử dụng **QTcpSocket::readLine()** để đọc dữ liệu

thiết lập một **client TCP** và xử lý các sự kiện liên quan đến việc kết nối client với server

Server:

slot **onReadyRead()**:

- Nhận message từ client -> **handleMessage(client, message);**

Client:

Slot **onReadyRead()**:

- Nhận message từ server -> **parseMessage(message)**



Demo

Hướng ứng dụng Chat App vào đề tài "Giám sát và kiểm tra tải cho hệ thống IVI"

- Sử dụng QTcpServer và QTcpSocket để tương tác giữa client và services
- Sử dụng SQLite để lưu trữ những thông tin quan trọng của services như các thông số, lịch sử trạng thái...
- Đối với client có thể được triển khai bằng Qt Quick Application.

Kết luận

- Ứng dụng được những kiến thức đã học về Qt framework để xây dựng một ứng dụng hoàn thiện.
- Có thể tái sử dụng code và phương pháp để ứng dụng vào đề tài chính.

Câu hỏi về đề tài "Giám sát và kiểm tra tải cho hệ thống IVI"

- Về service:
 - Nguyên lý hoạt động như thế nào?
 - Hoạt động trong môi trường/thiết bị nào?
 - Nhóm có thể xây dựng service và test service như thế nào?
 - Service trong đề tài có thể được xây dựng nhưng cách xây dựng Server Chat App?

Phụ lục

- Code: <https://github.com/HODUCVU/ChatApp-Qt.git>
- QSqlQuery: <https://doc.qt.io/qt-6/qsqlquery.html>
- QTcpSocket: <https://doc.qt.io/qt-6/qtcpsocket.html>
- QTcpServer: <https://doc.qt.io/qt-6/qtcpserver.html>