**Приложение клиента**

**Файл ChatMessageInfo.cpp:**

#include "ChatMessageInfo.h"

#include "ui\_ChatMessageInfo.h"

#include <random>

QHash<QString, QColor> ChatMessageInfo::userColorMap;

ChatMessageInfo::ChatMessageInfo(QWidget \*parent)

: QWidget(parent)

, ui(new Ui::ChatMessageInfo) {

ui->setupUi(this); }

ChatMessageInfo::~ChatMessageInfo() {

delete ui; }

void ChatMessageInfo::displayMessage(QString message, QString username, bool isUserMessage) {

qDebug() << "Displaying message:" << message << "from user:" << username << "isUserMessage:" << isUserMessage;

setMessageColor(getUserColor(username));

setMessageAlignment(isUserMessage);

ui->labelMessage->setText(message);

ui->labelTime->setText(QDateTime::currentDateTime().toString("HH:mm")); }

QColor ChatMessageInfo::getUserColor(QString username) {

if (username.isEmpty()) {

return Qt::black; }

else {

if (!userColorMap.contains(username)) {

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_int\_distribution<> dis(0, 255);

QColor newUserColor = QColor::fromRgb(dis(gen), dis(gen), dis(gen));

userColorMap.insert(username, newUserColor); }

return userColorMap.value(username); } }

void ChatMessageInfo::setMessageAlignment(bool isUserMessage) {

if (isUserMessage) {

ui->labelMessage->setAlignment(Qt::AlignRight); }

else {

ui->labelMessage->setAlignment(Qt::AlignLeft); } }

void ChatMessageInfo::setMessageColor(QColor color) {

QString css = QString("color: %1").arg(color.name());

ui->labelMessage->setStyleSheet(css); }

**Файл ChatMessageInfo.h:**

#ifndef CHATMESSAGEINFO\_H

#define CHATMESSAGEINFO\_H

#include <QWidget>

#include <QDateTime>

#include <QtGlobal>

namespace Ui {

class ChatMessageInfo; }

class ChatMessageInfo : public QWidget {

Q\_OBJECT

public:

explicit ChatMessageInfo(QWidget \*parent = nullptr);

~ChatMessageInfo();

void displayMessage(QString message, QString username, bool isUserMessage);

private:

Ui::ChatMessageInfo \*ui;

static QHash<QString, QColor> userColorMap;

QColor getUserColor(QString username);

void setMessageColor(QColor color);

void setMessageAlignment(bool isUserMessage);

};

#endif // CHATMESSAGEINFO\_H

**Файл ClientManager.cpp:**

#include "ClientManager.h"

ClientManager::ClientManager(QHostAddress ip, ushort port, QObject \*parent)

: QObject{parent},

ip(ip),

port(port) {

setupClient(); }

void ClientManager::connectToServer() {

socket->connectToHost(ip, port); }

void ClientManager::composeAndSendMessage(QString message, QString receiver) {

socket->write(protocol.composeTextMessage(message, receiver)); }

void ClientManager::composeAndSendName(QString name) {

socket->write(protocol.composeNameMessage(name)); }

void ClientManager::disconnectFromServer() {

socket->disconnectFromHost(); }

void ClientManager::readyRead() {

auto data = socket->readAll();

protocol.parseData(data);

switch (protocol.getMessageType()) {

case ClientProtocol::CHAT\_MESSAGE:

emit chatMessageReceived(protocol.getMessageSender(), protocol.getChatMessage());

break;

case ClientProtocol::SEND\_NAME:

emit newNameReceived(protocol.getNewName());

break;

case ClientProtocol::CONNECTION\_ACK:

emit connectionAcknowledged(protocol.getMyName(), protocol.getClientNames());

break;

case ClientProtocol::NEW\_CLIENT\_CONNECTED:

emit newClientConnectedToServer(protocol.getCurrentClientName());

break;

case ClientProtocol::CLIENT\_DISCONNECTED:

emit clientDisconnected(protocol.getCurrentClientName());

break;

case ClientProtocol::UPDATE\_NAME:

emit clientNameUpdated(protocol.getPreviousName(), protocol.getCurrentClientName());

break;

default:

break; } }

void ClientManager::setupClient() {

socket = new QTcpSocket(this);

connect(socket, &QTcpSocket::connected, this, &ClientManager::connected);

connect(socket, &QTcpSocket::disconnected, this, &ClientManager::disconnected);

connect(socket, &QTcpSocket::readyRead, this, &ClientManager::readyRead);

connect(socket, QOverload<QAbstractSocket::SocketError>::of(&QAbstractSocket::errorOccurred),

[this](QAbstractSocket::SocketError socketError) {

Q\_UNUSED(socketError)

emit errorOccurred(socket->errorString());

}); }

**Файл ClientManager.h:**

#ifndef CLIENTMANAGER\_H

#define CLIENTMANAGER\_H

#include "ClientProtocol.h"

#include <QObject>

#include <QTcpSocket>

class ClientManager : public QObject {

Q\_OBJECT

public:

explicit ClientManager(QHostAddress ip = QHostAddress("195.181.246.125"), ushort port = 8080, QObject \*parent = nullptr);

void connectToServer();

void composeAndSendMessage(QString message, QString receiver);

void composeAndSendName(QString name);

void disconnectFromServer();

signals:

void connected();

void disconnected();

void chatMessageReceived(QString sender, QString message);

void newNameReceived(QString name);

void connectionAcknowledged(QString myName, QStringList clientsName);

void newClientConnectedToServer(QString clienName);

void clientNameUpdated(QString prevName, QString clientName);

void clientDisconnected(QString clientName);

void errorOccurred(const QString &errorString);

private slots:

void readyRead();

private:

QTcpSocket \*socket;

QHostAddress ip;

ushort port;

ClientProtocol protocol;

private:

void setupClient();

};

#endif // CLIENTMANAGER\_H

**Файл ClientProtocol.cpp:**

#include "ClientProtocol.h"

#include <QFileInfo>

#include <QIODevice>

ClientProtocol::ClientProtocol() { }

QByteArray ClientProtocol::composeTextMessage(QString message, QString receiver) {

QByteArray ba;

QDataStream out(&ba, QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_0);

out << CHAT\_MESSAGE << receiver << message;

return ba; }

QByteArray ClientProtocol::composeNameMessage(QString name) {

return prepareData(SEND\_NAME, name); }

void ClientProtocol::parseData(QByteArray data) {

QDataStream in(&data, QIODevice::ReadOnly);

in.setVersion(QDataStream::Qt\_5\_0);

in >> messageType;

switch (messageType) {

case CHAT\_MESSAGE:

in >> messageSender >>messageReceiver >> chatMessage;

break;

case SEND\_NAME:

in >> newName;

break;

case UPDATE\_NAME:

in >> previousName >> currentClientName;

break;

case NEW\_CLIENT\_CONNECTED:

case CLIENT\_DISCONNECTED:

in >> currentClientName;

break;

case CONNECTION\_ACK:

in >> myName >> clientNames;

break;

default:

break; } }

QByteArray ClientProtocol::prepareData(MessageType type, QString data) {

QByteArray ba;

QDataStream out(&ba, QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_0);

out << type << data;

return ba; }

QString ClientProtocol::getMessageSender() const {

return messageSender; }

const QString &ClientProtocol::getMyName() const {

return myName; }

const QStringList &ClientProtocol::getClientNames() const {

return clientNames; }

const QString &ClientProtocol::getPreviousName() const {

return previousName; }

const QString &ClientProtocol::getCurrentClientName() const {

return currentClientName; }

QString ClientProtocol::getMessageReceiver() const {

return messageReceiver; }

ClientProtocol::MessageType ClientProtocol::getMessageType() const {

return messageType; }

const QString &ClientProtocol::getNewName() const {

return newName; }

const QString &ClientProtocol::getChatMessage() const {

return chatMessage; }

**Файл ClientProtocol.h:**

#ifndef CLIENTPROTOCOL\_H

#define CLIENTPROTOCOL\_H

#include <QByteArray>

#include <QString>

#include <QStringList>

class ClientProtocol {

public:

enum MessageType{

CHAT\_MESSAGE,

SEND\_NAME,

UPDATE\_NAME,

CONNECTION\_ACK,

NEW\_CLIENT\_CONNECTED,

CLIENT\_DISCONNECTED

};

ClientProtocol();

QByteArray composeTextMessage(QString message, QString receiver);

QByteArray composeNameMessage(QString name);

void parseData(QByteArray data);

const QString &getChatMessage() const;

const QString &getNewName() const;

MessageType getMessageType() const;

QString getMessageReceiver() const;

const QString &getCurrentClientName() const;

const QString &getPreviousName() const;

const QStringList &getClientNames() const;

const QString &getMyName() const;

QString getMessageSender() const;

private:

QByteArray prepareData(MessageType type, QString data);

private:

MessageType messageType;

QString chatMessage;

QString newName;

QString messageReceiver;

QString currentClientName;

QString previousName;

QStringList clientNames;

QString myName;

QString messageSender;

};

#endif // CLIENTPROTOCOL\_H

**Файл ClientWindow.cpp:**

#include "ClientWindow.h"

#include "ui\_ClientWindow.h"

#include <QDebug>

#include "ChatMessageInfo.h"

#include <QLineEdit>

#include <QMessageBox>

#include <QInputDialog>

#include <QDir>

#include "ClientWindow.h"

#include "ui\_ClientWindow.h"

#include <QInputDialog>

#include <QDir>

ClientWindow::ClientWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::ClientWindow)

, loginWindow(new LoginWindow()) {

ui->setupUi(this);

setupClient();

client->connectToServer();

connect(client, &ClientManager::errorOccurred, [this](const QString &errorString) {

loginWindow->close();

QMessageBox::critical(this, tr("Error"), tr("Server not running. Details: ") + errorString);

QApplication::quit();

exit(EXIT\_FAILURE);

});

if (loginWindow->exec() == QDialog::Rejected) {

client->disconnectFromServer();

QApplication::quit();

exit(EXIT\_SUCCESS); }

ui->nameEdit->setText(loginWindow->getNickname());

client->composeAndSendName(loginWindow->getNickname()); }

ClientWindow::~ClientWindow() {

loginWindow->removeNickname(loginWindow->getNickname());

delete ui;

delete loginWindow; }

void ClientWindow::setupClient() {

client = new ClientManager();

connect(client , &ClientManager::connected, this, [this](){

ui->btnSend->setEnabled(true);

ui->editMessage->setEnabled(true);

});

connect(client, &ClientManager::disconnected, this, [this](){

ui->btnSend->setEnabled(false);

ui->editMessage->setEnabled(false);

});

connect(client, &ClientManager::chatMessageReceived, this, &ClientWindow::receiveChatMessage);

connect(client, &ClientManager::connectionAcknowledged, this, &ClientWindow::onConnectionAcknowledgement);

connect(client, &ClientManager::newClientConnectedToServer, this, &ClientWindow::onNewClientConnectedToServer);

connect(client, &ClientManager::clientDisconnected, this, &ClientWindow::onClientDisconnected);

connect(client, &ClientManager::clientNameUpdated, this, &ClientWindow::onClientNameUpdated); }

void ClientWindow::on\_btnSend\_clicked() {

processMessageAndSend(); }

void ClientWindow::createMessage(const QString& username, const QString& message, bool isMyMessage) {

auto chatMessageInfo = new ChatMessageInfo();

chatMessageInfo->displayMessage(message, username, isMyMessage);

auto listItemWidget = new QListWidgetItem();

listItemWidget->setSizeHint(QSize(0,65));

ui->messages->addItem(listItemWidget);

if (isMyMessage) {

listItemWidget->setBackground(QColor(227,225,225)); }

ui->messages->setItemWidget(listItemWidget, chatMessageInfo); }

void ClientWindow::processMessageAndSend() {

auto data = ui->editMessage->text().trimmed();

if (data.isEmpty()) {

QMessageBox::warning(this, tr("Warning"), tr("Message cannot be empty"));

return; }

client->composeAndSendMessage(data, ui->receiverBox->currentText());

ui->editMessage->setText("");

createMessage("", data.toUtf8(), true); }

void ClientWindow::receiveChatMessage(QString sender, QString message) {

createMessage(sender, sender + ": " + message, false); }

void ClientWindow::onConnectionAcknowledgement(QString myName, QStringList clientsName) {

ui->receiverBox->clear();

clientsName.prepend("Server");

foreach (auto client, clientsName) {

ui->receiverBox->addItem(client); }

setWindowTitle(myName); }

void ClientWindow::onNewClientConnectedToServer(QString clienName) {

ui->receiverBox->addItem(clienName); }

void ClientWindow::onClientNameUpdated(QString prevName, QString clientName) {

for (int i = 0; i < ui->receiverBox->count(); ++i) {

if (ui->receiverBox->itemText(i) == prevName) {

ui->receiverBox->setItemText(i, clientName);

return; } } }

void ClientWindow::onClientDisconnected(QString clientName) {

for (int i = 0; i < ui->receiverBox->count(); ++i) {

if (ui->receiverBox->itemText(i) == clientName) {

ui->receiverBox->removeItem(i);

return; } } }

void ClientWindow::on\_nameEdit\_returnPressed() {

auto newName = ui->nameEdit->text().trimmed();

if (loginWindow->doesNicknameExist(newName)) {

QMessageBox::warning(this, tr("Error"), tr("This nickname is already taken"));

ui->nameEdit->setText(loginWindow->getNickname());

return; }

loginWindow->removeNickname(loginWindow->getNickname());

loginWindow->addNickname(newName);

loginWindow->setNickName(newName);

client->composeAndSendName(newName); }

void ClientWindow::on\_editMessage\_returnPressed() {

processMessageAndSend(); }

**Файл ClientWindow.h:**

#ifndef CLIENTWINDOW\_H

#define CLIENTWINDOW\_H

#include "LoginWindow.h"

#include <QMainWindow>

#include <ClientManager.h>

QT\_BEGIN\_NAMESPACE

namespace Ui {

class ClientWindow; }

QT\_END\_NAMESPACE

class ClientWindow : public QMainWindow {

Q\_OBJECT

public:

ClientWindow(QWidget \*parent = nullptr);

~ClientWindow();

void connectToServer();

private slots:

void on\_btnSend\_clicked();

void receiveChatMessage(QString sender, QString message);

void onConnectionAcknowledgement(QString myName, QStringList clientsName);

void onNewClientConnectedToServer(QString clienName);

void onClientNameUpdated(QString prevName, QString clientName);

void onClientDisconnected(QString clientName);

void on\_nameEdit\_returnPressed();

void on\_editMessage\_returnPressed();

private:

Ui::ClientWindow \*ui;

ClientManager \*client;

LoginWindow \*loginWindow;

void setupClient();

void createMessage(const QString& username, const QString& message, bool isMyMessage);

void processMessageAndSend();

};

#endif // CLIENTWINDOW\_H

Файл client\_main.cpp:

#include "LoginWindow.h"

#include "ClientWindow.h"

#include <QDebug>

#include <QApplication>

int main(int argc, char \*argv[]) {

QApplication a(argc, argv);

LoginWindow login;

ClientWindow client;

// QObject::connect(&login, &LoginWindow::loginSuccessful, [&]() {

// login.close();

// client.updateUserNameAndNotifyServer(login.getNickname());

// client.connectToServer();

// client.setWindowTitle(login.getNickname());

// client.show();

// });

// QObject::connect(&a, &QApplication::aboutToQuit, [&]() {

// login.removeNickname(login.getNickname());

// });

client.show();

return a.exec(); }

**Файл LoginWindow.cpp:**

#include "LoginWindow.h"

#include "ui\_LoginWindow.h"

#include <QMessageBox>

LoginWindow::LoginWindow(QWidget \*parent)

: QDialog(parent)

, ui(new Ui::LoginWindow) {

ui->setupUi(this);

setupDatabase(); }

LoginWindow::~LoginWindow() {

delete ui;

db.close(); }

void LoginWindow::setupDatabase() {

db = QSqlDatabase::addDatabase("QPSQL");

db.setHostName("195.181.246.125");

db.setDatabaseName("users");

db.setUserName("postgres");

db.setPassword("admin");

if (!db.open()) {

qCritical() << "Cannot open database:" << db.lastError();

return; }

QSqlQuery query(db);

if (!query.exec("CREATE TABLE IF NOT EXISTS nicknames (nickname TEXT UNIQUE)")) {

qCritical() << "Cannot create table:" << query.lastError(); } }

bool LoginWindow::doesNicknameExist(const QString &nickname) {

QSqlQuery query(db);

query.prepare("SELECT 1 FROM nicknames WHERE nickname = ?");

query.addBindValue(nickname);

if (!query.exec() || !query.next()) {

return false; }

return true; }

void LoginWindow::addNickname(const QString &nickname) {

QSqlQuery query(db);

query.prepare("INSERT INTO nicknames (nickname) VALUES (?)");

query.addBindValue(nickname);

if (!query.exec()) {

qCritical() << "Cannot insert nickname:" << query.lastError(); } }

void LoginWindow::removeNickname(const QString &nickname) {

QSqlQuery query(db);

query.prepare("DELETE FROM nicknames WHERE nickname = :nickname");

query.bindValue(":nickname", nickname);

if (!query.exec()) {

qWarning() << "Не удалось удалить никнейм:" << query.lastError(); } }

void LoginWindow::on\_nickname\_returnPressed() {

QString nickname = ui->nickname->text();

if (nickname.isEmpty()) {

QMessageBox::warning(this, tr("Warning"), tr("Nickname cannot be empty"));

return; }

if (doesNicknameExist(nickname)) {

QMessageBox::warning(this, tr("Error"), tr("This nickname is already taken"));

return; }

nickName = nickname;

addNickname(nickname);

accept();

close(); }

void LoginWindow::setNickName(const QString &newNickName) {

nickName = newNickName; }

QString LoginWindow::getNickname() const {

return nickName; }

**Файл LoginWindow.h:**

#ifndef LOGINWINDOW\_H

#define LOGINWINDOW\_H

#include <QWidget>

#include <QSqlQuery>

#include <QSqlError>

#include <QDebug>

#include <QDialog>

namespace Ui {

class LoginWindow; }

class LoginWindow : public QDialog {

Q\_OBJECT

public:

explicit LoginWindow(QWidget \*parent = nullptr);

~LoginWindow();

QString getNickname() const;

void removeNickname(const QString &nickname);

bool doesNicknameExist(const QString &nickname);

void addNickname(const QString &nickname);

void setNickName(const QString &newNickName);

private slots:

void on\_nickname\_returnPressed();

private:

Ui::LoginWindow \*ui;

QSqlDatabase db;

QString nickName;

private:

void setupDatabase();

};

#endif // LOGINWINDOW\_H

**Приложение сервера**

**Файл ChatWindow.cpp:**

#include "ChatWindow.h"

#include "ui\_ChatWindow.h"

ChatWindow::ChatWindow(QTcpSocket \* \_client, QWidget \*parent)

: QWidget(parent)

, ui(new Ui::ChatWindow) {

ui->setupUi(this);

client = new ServerClientManager(\_client, this);

connect(client, &ServerClientManager::disconnected, this, &ChatWindow::clientDisconnected);

connect(client, &ServerClientManager::chatMessageReceived, this, &ChatWindow::chatMessageReceived);

connect(client, &ServerClientManager::clientNameUpdated, this, &ChatWindow::onClientNameChanged); }

void ChatWindow::disconnect() {

client->disconnectFromServer(); }

ChatWindow::~ChatWindow() {

delete ui; }

void ChatWindow::clientDisconnected() { }

void ChatWindow::on\_btnSend\_clicked() {

auto message = ui->editMessage->text().trimmed();

client->composeAndSendMessage(message);

ui->editMessage->setText("");

ui->listMessages->addItem(message); }

void ChatWindow::chatMessageReceived(QString message, QString receiver, QString sender) {

if (receiver == "Server") {

ui->listMessages->addItem(QString("%1: %2").arg(sender, message)); }

else {

emit textForOtherClients(message, receiver, client->getClientName()); } }

void ChatWindow::onClientNameChanged(QString prevName, QString name) {

emit clientNameUpdated(prevName, name); }

**Файл ChatWindow.h:**

#ifndef CHATWINDOW\_H

#define CHATWINDOW\_H

#include <QWidget>

#include <QTcpSocket>

#include "ServerClientManager.h"

namespace Ui {

class ChatWindow; }

class ChatWindow : public QWidget {

Q\_OBJECT

public:

explicit ChatWindow(QTcpSocket \* \_client, QWidget \*parent = nullptr);

void disconnect();

~ChatWindow();

private slots:

void clientDisconnected();

void on\_btnSend\_clicked();

void chatMessageReceived(QString message, QString receiver, QString sender);

void onClientNameChanged(QString prevName, QString name);

signals:

void clientNameUpdated(QString prevName, QString name);

void textForOtherClients(QString message, QString receiver, QString sender);

private:

Ui::ChatWindow \*ui;

ServerClientManager \*client;

};

#endif // CHATWINDOW\_H

**Файл ServerClientManager.cpp:**

#include "ServerClientManager.h"

#include <QDir>

ServerClientManager::ServerClientManager(QHostAddress ip, ushort port, QObject \*parent)

: QObject{parent},

ip(ip),

port(port) {

socket = new QTcpSocket(this);

setupClient(); }

ServerClientManager::ServerClientManager(QTcpSocket \*client, QObject \*parent)

: QObject{parent},

socket(client) {

setupClient(); }

void ServerClientManager::connectToServer() {

socket->connectToHost(ip, port); }

void ServerClientManager::disconnectFromServer() {

socket->disconnectFromHost(); }

void ServerClientManager::composeAndSendMessage(QString message) {

socket->write(protocol.composeChatMessage(message, getClientName(), "Server")); }

void ServerClientManager::composeAndSendName(QString name) {

socket->write(protocol.composeNameMessage(name)); }

QString ServerClientManager::getClientName() const {

auto id = socket->property("id").toInt();

auto name = protocol.getNewName().length() > 0 ? protocol.getNewName() : QString("Client (%1)").arg(id);

return name; }

void ServerClientManager::readyRead() {

auto data = socket->readAll();

protocol.parseData(data);

switch (protocol.getMessageType()) {

case ServerProtocol::CHAT\_MESSAGE:

emit chatMessageReceived(protocol.getChatMessage(), protocol.getMessageReceiver(), getClientName());

break;

case ServerProtocol::SEND\_NAME:{

auto prevName = socket->property("clientName").toString();

socket->setProperty("clientName", getClientName());

emit clientNameUpdated(prevName, getClientName());

break; }

default:

break; } }

void ServerClientManager::setupClient() {

connect(socket, &QTcpSocket::connected, this, &ServerClientManager::connected);

connect(socket, &QTcpSocket::disconnected, this, &ServerClientManager::disconnected);

connect(socket, &QTcpSocket::readyRead, this, &ServerClientManager::readyRead); }

**Файл ServerClientManager.h:**

#ifndef SERVERCLIENTMANAGER\_H

#define SERVERCLIENTMANAGER\_H

#include "ServerProtocol.h"

#include <QObject>

#include <QTcpSocket>

#include <QHostAddress>

class ServerClientManager : public QObject {

Q\_OBJECT

public:

explicit ServerClientManager(QHostAddress ip = QHostAddress("195.181.246.125"), ushort port = 8080, QObject \*parent = nullptr);

explicit ServerClientManager(QTcpSocket \*client, QObject \*parent = nullptr);

void connectToServer();

void disconnectFromServer();

void composeAndSendMessage(QString message);

void composeAndSendName(QString name);

QString getClientName() const;

signals:

void connected();

void disconnected();

void chatMessageReceived(const QString message, QString receiver, QString sender);

void clientNameUpdated(QString prevName, QString name);

private slots:

void readyRead();

private:

QTcpSocket \*socket;

QHostAddress ip;

ushort port;

ServerProtocol protocol;

private:

void setupClient();

};

#endif // SERVERCLIENTMANAGER\_H

**Файл ServerManager.cpp:**

#include "ServerManager.h"

ServerManager::ServerManager(ushort port, QObject \*parent)

: QObject{parent} {

setupServer(port); }

void ServerManager::informClientsAboutNameChange(QString prevName, QString name) {

auto message = protocol.composeUpdateNameMessage(prevName, name);

foreach (auto cl, \_clients) {

auto clientName = cl->property("clientName").toString();

if (clientName != name) {

cl->write(message); } } }

void ServerManager::onTextForOtherClients(QString message, QString receiver, QString sender) {

auto msg = protocol.composeChatMessage(message, receiver, sender);

foreach (auto cl, \_clients) {

auto clientName = cl->property("clientName").toString();

if (clientName == receiver) {

cl->write(msg);

return; } } }

void ServerManager::onNewClientConnection() {

auto client = server->nextPendingConnection();

auto id = \_clients.count() + 1;

auto clientName = QString("Client (%1)").arg(id);

client->setProperty("id", id);

client->setProperty("clientName", clientName);

connect(client, &QTcpSocket::disconnected, this, &ServerManager::onClientDisconnected);

emit newClientConnected(client);

if (id > 1) {

auto message = protocol.composeConnectionAckMessage(clientName, \_clients.keys());

client->write(message);

auto newClientMessage = protocol.composeNewClientMessage(clientName);

foreach (auto cl, \_clients) {

cl->write(newClientMessage); } }

\_clients[clientName] = client; }

void ServerManager::onClientDisconnected() {

auto client = qobject\_cast<QTcpSocket \*>(sender());

auto clientName = client->property("clientName").toString();

\_clients.remove(clientName);

auto message = protocol.composeClientDisconnectedMessage(clientName);

foreach (auto cl, \_clients) {

cl->write(message); }

emit clientDisconnected(client); }

void ServerManager::setupServer(ushort port) {

server = new QTcpServer(this);

connect(server, &QTcpServer::newConnection, this, &ServerManager::onNewClientConnection);

server->listen(QHostAddress::Any, port); }

**Файл ServerManager.h:**

#ifndef SERVERMANAGER\_H

#define SERVERMANAGER\_H

#include "ServerProtocol.h"

#include <QObject>

#include <QTcpServer>

#include <QTcpSocket>

class ServerManager : public QObject {

Q\_OBJECT

public:

explicit ServerManager(ushort port = 8080, QObject \*parent = nullptr);

void informClientsAboutNameChange(QString prevName, QString name);

QMap<QString, QTcpSocket \*> \_clients;

public slots:

void onTextForOtherClients(QString message, QString receiver, QString sender);

signals:

void newClientConnected(QTcpSocket \*client);

void clientDisconnected(QTcpSocket \*client);

private slots:

void onNewClientConnection();

void onClientDisconnected();

private:

QTcpServer \*server;

ServerProtocol protocol;

private:

void setupServer(ushort port);

};

#endif // SERVERMANAGER\_H

**Файл ServerProtocol.cpp:**

#include "ServerProtocol.h"

#include <QFileInfo>

#include <QIODevice>

ServerProtocol::ServerProtocol() { }

QByteArray ServerProtocol::composeChatMessage(QString message, QString receiver, QString sender) {

QByteArray ba;

QDataStream out(&ba, QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_0);

out << CHAT\_MESSAGE << sender << receiver << message;

return ba; }

QByteArray ServerProtocol::composeNameMessage(QString name) {

return getData(SEND\_NAME, name); }

QByteArray ServerProtocol::composeUpdateNameMessage(QString prevName, QString name) {

QByteArray ba;

QDataStream out(&ba, QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_0);

out << UPDATE\_NAME << prevName << name;

return ba; }

QByteArray ServerProtocol::composeConnectionAckMessage(QString clientName, QStringList otherClients) {

QByteArray ba;

QDataStream out(&ba, QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_0);

out << CONNECTION\_ACK << clientName << otherClients;

return ba; }

QByteArray ServerProtocol::composeNewClientMessage(QString clientName) {

return getData(NEW\_CLIENT\_CONNECTED, clientName); }

QByteArray ServerProtocol::composeClientDisconnectedMessage(QString clientName) {

return getData(CLIENT\_DISCONNECTED, clientName); }

void ServerProtocol::parseData(QByteArray data) {

QDataStream in(&data, QIODevice::ReadOnly);

in.setVersion(QDataStream::Qt\_5\_0);

qint32 \_type;

in >> \_type;

messageType = static\_cast<MessageType>(\_type);

switch (messageType) {

case CHAT\_MESSAGE:

in >> messageReceiver >> chatMessage;

break;

case SEND\_NAME:

in >> newName;

break;

default:

break; } }

QByteArray ServerProtocol::getData(MessageType type, QString data) {

QByteArray ba;

QDataStream out(&ba, QIODevice::WriteOnly);

out.setVersion(QDataStream::Qt\_5\_0);

out << type << data;

return ba; }

const QString &ServerProtocol::getMessageReceiver() const {

return messageReceiver; }

const QString &ServerProtocol::getNewName() const {

return newName; }

ServerProtocol::MessageType ServerProtocol::getMessageType() const {

return messageType; }

const QString &ServerProtocol::getChatMessage() const {

return chatMessage; }

**Файл ServerProtocol.h:**

#ifndef SERVERPROTOCOL\_H

#define SERVERPROTOCOL\_H

#include <QByteArray>

#include <QString>

#include <QDataStream>

class ServerProtocol {

public:

enum MessageType{

CHAT\_MESSAGE,

SEND\_NAME,

UPDATE\_NAME,

CONNECTION\_ACK,

NEW\_CLIENT\_CONNECTED,

CLIENT\_DISCONNECTED

};

ServerProtocol();

QByteArray composeChatMessage(QString message, QString receiver, QString sender);

QByteArray composeNameMessage(QString name);

QByteArray composeUpdateNameMessage(QString prevName, QString name);

QByteArray composeConnectionAckMessage(QString clientName, QStringList otherClients);

QByteArray composeNewClientMessage(QString clientName);

QByteArray composeClientDisconnectedMessage(QString clientName);

void parseData(QByteArray data);

const QString &getChatMessage() const;

const QString &getNewName() const;

MessageType getMessageType() const;

const QString &getMessageReceiver() const;

private:

QByteArray getData(MessageType type, QString data);

MessageType messageType;

QString chatMessage;

QString newName;

QString messageReceiver;

};

#endif // SERVERPROTOCOL\_H

**Файл ServerWindow.cpp:**

#include "ServerWindow.h"

#include "ui\_ServerWindow.h"

ServerWindow::ServerWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::ServerWindow) {

ui->setupUi(this);

setupServerConfiguration(); }

ServerWindow::~ServerWindow() {

delete ui; }

void ServerWindow::newClientConnected(QTcpSocket \*client) {

auto id = client->property("id").toInt();

ui->listClients->addItem(QString("New Client added: %1").arg(id));

auto chatWidget= new ChatWindow(client, ui->tabChats);

ui->tabChats->addTab(chatWidget, QString("Client (%1)").arg(id));

connect(chatWidget, &ChatWindow::clientNameUpdated, this, &ServerWindow::updateClientName);

connect(chatWidget, &ChatWindow::textForOtherClients, server, &ServerManager::onTextForOtherClients); }

void ServerWindow::clientDisconnected(QTcpSocket \*client) {

auto id = client->property("id").toInt();

ui->listClients->addItem(QString("Client with id %1 disconnected").arg(id)); }

void ServerWindow::updateClientName(QString prevName, QString name) {

auto widget = qobject\_cast<QWidget \*>(sender());

auto index = ui->tabChats->indexOf(widget);

ui->tabChats->setTabText(index, name);

if (server->\_clients.contains(prevName)) {

auto clientSocket = server->\_clients.take(prevName);

server->\_clients[name] = clientSocket; }

server->informClientsAboutNameChange(prevName, name); }

void ServerWindow::setupServerConfiguration() {

server = new ServerManager();

connect(server, &ServerManager::newClientConnected, this, &ServerWindow::newClientConnected);

connect(server, &ServerManager::clientDisconnected, this, &ServerWindow::clientDisconnected); }

void ServerWindow::on\_tabChats\_tabCloseRequested(int index) {

auto chatWidget = qobject\_cast<ChatWindow \*>(ui->tabChats->widget(index));

chatWidget->disconnect();

ui->tabChats->removeTab(index); }

#ifndef SERVERWINDOW\_H

#define SERVERWINDOW\_H

#include <QMainWindow>

#include <ServerManager.h>

#include <ChatWindow.h>

QT\_BEGIN\_NAMESPACE

namespace Ui {

class ServerWindow; }

QT\_END\_NAMESPACE

class ServerWindow : public QMainWindow {

Q\_OBJECT

public:

ServerWindow(QWidget \*parent = nullptr);

~ServerWindow();

private slots:

void newClientConnected(QTcpSocket \*client);

void clientDisconnected(QTcpSocket \*client);

void updateClientName(QString prevName, QString name);

void on\_tabChats\_tabCloseRequested(int index);

private:

Ui::ServerWindow \*ui;

ServerManager \* server;

private:

void setupServerConfiguration();

};

#endif // SERVERWINDOW\_H

**Файл server\_main.cpp:**

#include "ServerWindow.h"

#include <QApplication>

int main(int argc, char \*argv[]) {

QApplication a(argc, argv);

ServerWindow w;

w.show();

return a.exec(); }ч