ParuGramm_client

1.0.0

Создано системой Doxygen 1.13.2

Глава 1

Иерархический список классов

1.1 Иерархия классов

Иерархия классов.

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Глава 3

Список файлов

3.1 Файлы

Полный список файлов.

Глава 4

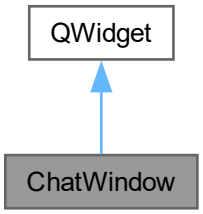
Классы

4.1 Класс ChatWindow

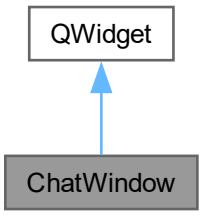
Класс для управления окном чата в комнате.

#include <chatwindow.h>

Граф наследования:ChatWindow:



Граф связей класса ChatWindow:



Открытые члены

- ChatWindow (int roomId, Client *client, QWidget *parent=nullptr)
  Конструктор окна чата.

4.1.1 Подробное описание

Класс для управления окном чата в комнате.

См. определение в файле chatwindow.h строка 22

4.1.2 Конструктор(ы)

4.1.2.1 ChatWindow()

```
ChatWindow::ChatWindow (
            int roomId,
            Client * client,
            QWidget * parent = nullptr)
```

Конструктор окна чата.

Аргументы

roomId	ID комнаты.
client	Указатель на клиентский объект.
parent	Родительский виджет.

См. определение в файле chatwindow.cpp строка 31

Объявления и описания членов классов находятся в файлах:

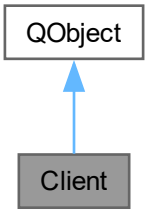
- E:/ParuGramm/my_proj/client/chatwindow.h
- E:/ParuGramm/my_proj/client/chatwindow.cpp

4.2 Класс Client

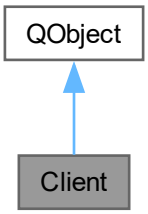
Класс для управления клиентской частью мессенджера с TCP-сокетами.

#include <client.h>

Граф наследования:Client:



Граф связей класса Client:



Сигналы

- void messageReceived (int roomId, int senderId, const QString &message)

  Сигнал, испускаемый при получении сообщения.
- void roomCreated (int roomId)

  Сигнал, испускаемый при создании комнаты.
- void joinedRoom ()

  Сигнал, испускаемый при успешном присоединении к комнате.
- void roomListReceived (const QStringList &rooms)

  Сигнал, испускаемый при получении списка комнат.
- void errorReceived (const QString &error)

  Сигнал, испускаемый при получении ошибки.
- void fileReceived (int roomId, int senderId, const QString &fileName, const QByteArray &fileData)

  Сигнал, испускаемый при получении файла.
- void fileSent (const QString &fileName, const QByteArray &fileData)

  Сигнал, испускаемый при отправке файла.
- void fileSentConfirmed ()

  Сигнал, испускаемый при подтверждении отправки файла.
- void usersReceived (const QStringList &users)

Сигнал, испускаемый при получении списка пользователей.

- void userJoined (int roomId, int userId)

  Сигнал, испускаемый при присоединении пользователя к комнате.
- void userLeft (int roomId, int userId)

  Сигнал, испускаемый при выходе пользователя из комнаты.
- void leftRoom ()

  Сигнал, испускаемый при выходе из комнаты.

Открытые члены

- bool connectToServer (const std::string &ip, int port)

  Подключается к серверу по указанному IP и порту.
- void disconnect ()

  Отключается от сервера.
- void sendRequest (const std::string &request)

  Отправляет запрос на сервер.
- void sendFile (const std::string &filePath)

  Отправляет файл на сервер.
- void setRoomId (int id)

  Устанавливает ID текущей комнаты.
- int getRoomId () const

  Получает ID текущей комнаты.
- int getClientId () const

  Получает ID клиента.
- bool isConnected () const

  Проверяет, подключен ли клиент к серверу.
- void leaveRoom ()

  Покидает текущую комнату.

Открытые статические члены

- static Client & getInstance ()

  Получает единственный экземпляр Client (паттерн Singleton).

Друзья

- class SingletonDestroyer

4.2.1 Подробное описание

Класс для управления клиентской частью мессенджера с TCP-сокетами.

См. определение в файле client.h строка 40

4.2.2 Методы

4.2.2.1 connectToServer()

```
bool Client::connectToServer (
            const std::string & ip,
            int port)
```

Подключается к серверу по указанному IP и порту.

Аргументы

ip	IP-адрес сервера.
port	Порт сервера.

Возвращает

true, если подключение успешно, иначе false.

См. определение в файле client.cpp строка 60

4.2.2.2 disconnect()

```
void Client::disconnect ()
```

Отключается от сервера.

См. определение в файле client.cpp строка 94

4.2.2.3 errorReceived

```
void Client::errorReceived (
            const QString & error)   [signal]
```

Сигнал, испускаемый при получении ошибки.

Аргументы

error	Текст ошибки.

4.2.2.4 fileReceived

```
void Client::fileReceived (
            int roomId,
            int senderId,
            const QString & fileName,
            const QByteArray & fileData)   [signal]
```

Сигнал, испускаемый при получении файла.

Аргументы

roomId	ID комнаты.
senderId	ID отправителя.
fileName	Имя файла.
fileData	Данные файла.

4.2.2.5 fileSent

```
void Client::fileSent (
            const QString & fileName,
            const QByteArray & fileData)   [signal]
```

Сигнал, испускаемый при отправке файла.

Аргументы

fileName	Имя файла.
fileData	Данные файла.

4.2.2.6 fileSentConfirmed

void Client::fileSentConfirmed () [signal]

Сигнал, испускаемый при подтверждении отправки файла.

4.2.2.7 getClientId()

int Client::getClientId () const [inline]

Получает ID клиента.

Возвращает

ID клиента.

См. определение в файле client.h строка 90

4.2.2.8 getInstance()

Client & Client::getInstance () [static]

Получает единственный экземпляр Client (паттерн Singleton).

Возвращает

Ссылка на экземпляр Client.

См. определение в файле client.cpp строка 30

4.2.2.9 getRoomId()

int Client::getRoomId () const [inline]

Получает ID текущей комнаты.

Возвращает

ID текущей комнаты.

См. определение в файле client.h строка 84

4.2.2.10 isConnected()

bool Client::isConnected () const [inline]

Проверяет, подключен ли клиент к серверу.

Возвращает

true, если подключен, иначе false.

См. определение в файле client.h строка 96

4.2.2.11 joinedRoom

void Client::joinedRoom () [signal]

Сигнал, испускаемый при успешном присоединении к комнате.

4.2.2.12 leaveRoom()

void Client::leaveRoom ()

Покидает текущую комнату.

См. определение в файле client.cpp строка 478

4.2.2.13 leftRoom

void Client::leftRoom () [signal]

Сигнал, испускаемый при выходе из комнаты.

4.2.2.14 messageReceived

void Client::messageReceived (
int roomId,
int senderId,
const QString & message) [signal]

Сигнал, испускаемый при получении сообщения.

Аргументы

roomId	ID комнаты.
senderId	ID отправителя.
message	Текст сообщения.

4.2.2.15 roomCreated

void Client::roomCreated (
int roomId) [signal]

Сигнал, испускаемый при создании комнаты.

Аргументы

roomId	ID созданной комнаты.

4.2.2.16 roomListReceived

void Client::roomListReceived (
const QStringList & rooms) [signal]

Сигнал, испускаемый при получении списка комнат.

Аргументы

rooms	Список комнат.

4.2.2.17 sendFile()

void Client::sendFile (
const std::string & filePath)

Отправляет файл на сервер.

Аргументы

filePath	Путь к файлу.

См. определение в файле client.cpp строка 142

4.2.2.18 sendRequest()

void Client::sendRequest (
const std::string & request)

Отправляет запрос на сервер.

Аргументы

request	Текст запроса.

См. определение в файле client.cpp строка 112

4.2.2.19 setRoomId()

void Client::setRoomId (
int id) [inline]

Устанавливает ID текущей комнаты.

Аргументы

id	ID комнаты.

См. определение в файле client.h строка 78

4.2.2.20 userJoined

void Client::userJoined (
int roomId,
int userId) [signal]

Сигнал, испускаемый при присоединении пользователя к комнате.

Аргументы

roomId	ID комнаты.
userId	ID пользователя.

4.2.2.21 userLeft

void Client::userLeft (
int roomId,
int userId) [signal]

Сигнал, испускаемый при выходе пользователя из комнаты.

Аргументы

roomId	ID комнаты.
userId	ID пользователя.

4.2.2.22 usersReceived

void Client::usersReceived (
const QStringList & users) [signal]

Сигнал, испускаемый при получении списка пользователей.

Аргументы

users	Список пользователей.

4.2.3 Друзья класса и относящимся к классу обозначения

4.2.3.1 SingletonDestroyer

friend class SingletonDestroyer [friend]

См. определение в файле client.h строка 207

Объявления и описания членов классов находятся в файлах:

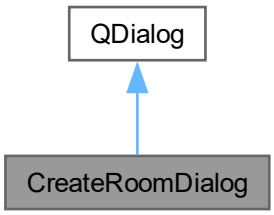
- E:/ParuGramm/my_proj/client/client.h
- E:/ParuGramm/my_proj/client/client.cpp

4.3 Класс CreateRoomDialog

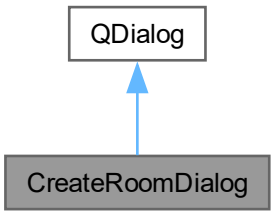
Класс для управления диалогом создания комнаты мессенджера.

#include <createroomdialog.h>

Граф наследования:CreateRoomDialog:



Граф связей класса CreateRoomDialog:



Открытые члены

- CreateRoomDialog (QWidget *parent=nullptr)
  Конструктор диалога создания комнаты.
- QString getPassword () const
  Получает пароль созданной комнаты.

4.3.1 Подробное описание

Класс для управления диалогом создания комнаты мессенджера.

См. определение в файле createroomdialog.h строка 17

4.3.2 Конструктор(ы)

4.3.2.1 CreateRoomDialog()

```
CreateRoomDialog::CreateRoomDialog (
            QWidget * parent = nullptr)   [explicit]
```

Конструктор диалога создания комнаты.

Аргументы

parent	Родительский виджет.

См. определение в файле createroomdialog.cpp строка 8

4.3.3 Методы

4.3.3.1 getPassword()

```
QString CreateRoomDialog::getPassword () const
```

Получает пароль созданной комнаты.

Возвращает

Пароль комнаты.

См. определение в файле createroomdialog.cpp строка 111

Объявления и описания членов классов находятся в файлах:

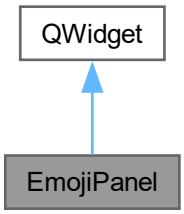
- E:/ParuGramm/my_proj/client/createroomdialog.h
- E:/ParuGramm/my_proj/client/createroomdialog.cpp

4.4 Класс EmojiPanel

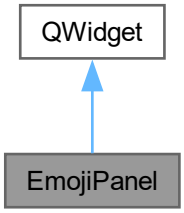
Класс для управления панелью выбора эмодзи в мессенджере.

#include <emojipanel.h>

Граф наследования:EmojiPanel:



Граф связей класса EmojiPanel:



Сигналы

- void emojiSelected (const QString &emoji)
  Сигнал, испускаемый при выборе эмодзи.

Открытые члены

- EmojiPanel (QWidget *parent=nullptr)
  Конструктор панели выбора эмодзи.

4.4.1 Подробное описание

Класс для управления панелью выбора эмодзи в мессенджере.

См. определение в файле emojipanel.h строка 17

4.4.2 Конструктор(ы)

4.4.2.1 EmojiPanel()

EmojiPanel::EmojiPanel (
QWidget * parent = nullptr) [explicit]

Конструктор панели выбора эмодзи.

Аргументы

parent	Родительский виджет.

См. определение в файле emojipanel.cpp строка 9

4.4.3 Методы

4.4.3.1 emojiSelected

void EmojiPanel::emojiSelected (
const QString & emoji) [signal]

Сигнал, испускаемый при выборе эмодзи.

Аргументы

emoji	Выбранный эмодзи.

Объявления и описания членов классов находятся в файлах:

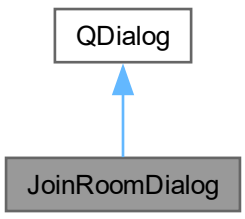
- E:/ParuGramm/my_proj/client/emojipanel.h
- E:/ParuGramm/my_proj/client/emojipanel.cpp

4.5 Класс JoinRoomDialog

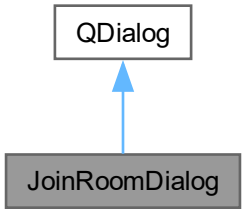
Класс для управления диалогом присоединения к комнате мессенджера.

#include <joinroomdialog.h>

Граф наследования:JoinRoomDialog:



Граф связей класса JoinRoomDialog:



Открытые члены

- JoinRoomDialog (QWidget *parent=nullptr)
  Конструктор диалога присоединения к комнате.
- void setRoomId (const QString &id)
  Устанавливает ID комнаты для присоединения.
- void showPasswordField ()
  Показывает поле ввода пароля.
- void hidePasswordField ()
  Скрывает поле ввода пароля.

4.5.1 Подробное описание

Класс для управления диалогом присоединения к комнате мессенджера.

См. определение в файле joinroomdialog.h строка 17

4.5.2 Конструктор(ы)

4.5.2.1 JoinRoomDialog()

```
JoinRoomDialog::JoinRoomDialog (
            QWidget * parent = nullptr)   [explicit]
```

Конструктор диалога присоединения к комнате.

Аргументы

parent	Родительский виджет.

См. определение в файле joinroomdialog.cpp строка 8

4.5.3 Методы

4.5.3.1 hidePasswordField()

void JoinRoomDialog::hidePasswordField ()

Скрывает поле ввода пароля.

См. определение в файле joinroomdialog.cpp строка 146

4.5.3.2 setRoomId()

void JoinRoomDialog::setRoomId (
const QString & id)

Устанавливает ID комнаты для присоединения.

Аргументы

id	ID комнаты.

См. определение в файле joinroomdialog.cpp строка 137

4.5.3.3 showPasswordField()

void JoinRoomDialog::showPasswordField ()

Показывает поле ввода пароля.

См. определение в файле joinroomdialog.cpp строка 142

Объявления и описания членов классов находятся в файлах:

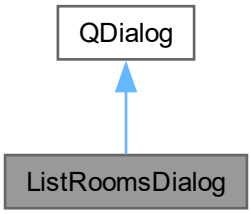
- E:/ParuGramm/my_proj/client/joinroomdialog.h
- E:/ParuGramm/my_proj/client/joinroomdialog.cpp

4.6 Класс ListRoomsDialog

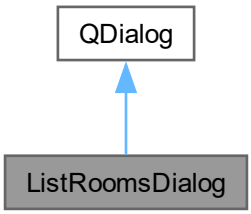
Класс для управления диалогом списка комнат.

#include <listroomsdialog.h>

Граф наследования:ListRoomsDialog:



Граф связей класса ListRoomsDialog:



Открытые члены

- ListRoomsDialog (QWidget *parent=nullptr)
  Конструктор диалога списка комнат.

4.6.1 Подробное описание

Класс для управления диалогом списка комнат.

См. определение в файле listroomsdialog.h строка 18

4.6.2 Конструктор(ы)

4.6.2.1 ListRoomsDialog()

```
ListRoomsDialog::ListRoomsDialog (
            QWidget * parent = nullptr)   [explicit]
```

Конструктор диалога списка комнат.

Аргументы

parent	Родительский виджет.

См. определение в файле listroomsdialog.cpp строка 11

Объявления и описания членов классов находятся в файлах:

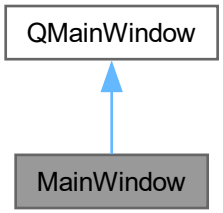
- E:/ParuGramm/my_proj/client/listroomsdialog.h
- E:/ParuGramm/my_proj/client/listroomsdialog.cpp

4.7 Класс MainWindow

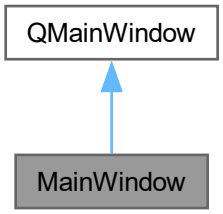
Класс для управления главным окном мессенджера.

#include <mainwindow.h>

Граф наследования:MainWindow:



Граф связей класса MainWindow:



Открытые члены

- MainWindow (QWidget *parent=nullptr)
  Конструктор главного окна.
- ~MainWindow ()
  Деструктор.

4.7.1 Подробное описание

Класс для управления главным окном мессенджера.

См. определение в файле mainwindow.h строка 17

4.7.2 Конструктор(ы)

4.7.2.1 MainWindow()

```
MainWindow::MainWindow (
            QWidget * parent = nullptr)   [explicit]
```

Конструктор главного окна.

Аргументы

parent	Родительский виджет.

См. определение в файле mainwindow.cpp строка 15

4.7.2.2 ~MainWindow()

```
MainWindow::~MainWindow ()
```

Деструктор.

См. определение в файле mainwindow.cpp строка 89

Объявления и описания членов классов находятся в файлах:

- E:/ParuGramm/my_proj/client/mainwindow.h
- E:/ParuGramm/my_proj/client/mainwindow.cpp

4.8 Класс SingletonDestroyer

Класс для уничтожения единственного экземпляра Client.

#include <client.h>

Открытые члены

- ~SingletonDestroyer ()

  Деструктор, освобождающий экземпляр Client.
- void initialize (Client *p)

  Инициализирует указатель на экземпляр Client.

4.8.1 Подробное описание

Класс для уничтожения единственного экземпляра Client.

См. определение в файле client.h строка 20

4.8.2 Конструктор(ы)

4.8.2.1 ~SingletonDestroyer()

SingletonDestroyer::~SingletonDestroyer ()

Деструктор, освобождающий экземпляр Client.

См. определение в файле client.cpp строка 38

4.8.3 Методы

4.8.3.1 initialize()

void SingletonDestroyer::initialize (
Client * p)

Инициализирует указатель на экземпляр Client.

Аргументы

p	Указатель на Client.

См. определение в файле client.cpp строка 42

Объявления и описания членов классов находятся в файлах:

- E:/ParuGramm/my_proj/client/client.h
- E:/ParuGramm/my_proj/client/client.cpp

Глава 5

Файлы

5.1 Файл E:/ParuGramm/my_proj/client/chatwindow.cpp

```
#include "chatwindow.h"
#include "emojipanel.h"
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QFileDialog>
#include <QMessageBox>
#include <QApplication>
#include <QPushButton>
#include <QDebug>
#include <QLabel>
#include <QScreen>
#include <QUrl>
#include <QTextBrowser>
#include <QTextCursor>
#include <QScrollBar>
#include <QStringList>
#include <QDateTime>
#include <QTimer>
#include <mainwindow.h>
#include <QRegularExpression>
```

Граф включаемых заголовочных файлов для chatwindow.cpp:

5.2 chatwindow.cpp

См. документацию.

```
00001 #include "chatwindow.h"
00002 #include "emojipanel.h"
00003 #include <QVBoxLayout>
00004 #include <QHBoxLayout>
00005 #include <QFileDialog>
```

```
00006 #include <QMessageBox>
00007 #include <QApplication>
00008 #include <QPushButton>
00009 #include <QDebug>
00010 #include <QLabel>
00011 #include <QScreen>
00012 #include <QUrl>
00013 #include <QTextBrowser>
00014 #include <QTextCursor>
00015 #include <QScrollBar>
00016 #include <QStringList>
00017 #include <QDateTime>
00018 #include <QTimer>
00019 #include <mainwindow.h>
00020 #include <QRegularExpression>
00021
00022 static QStringList colorList = {
00023     "#90EE90", // light green
00024     "#87CEEB", // sky blue
00025     "#FFA500", // orange
00026     "#FF69B4", // pink
00027     "#DA70D6", // orchid
00028     "#00CED1"  // turquoise
00029 };
00030
00031 ChatWindow::ChatWindow(int roomId, Client *client, QWidget *parent)
00032     : QWidget(parent), roomId(roomId), client(client), lastSenderId(-1), lastWasEvent(false)
00033 {
00034     setWindowTitle("Чат: Комната " + QString::number(roomId));
00035     setFixedSize(800, 875);
00036     setWindowIcon(QIcon(":/icons/snake.png"));
00037
00038     QScreen *screen = QApplication::primaryScreen();
00039     QRect screenGeometry = screen->geometry();
00040     int x = (screenGeometry.width() - width()) / 2;
00041     int y = (screenGeometry.height() - height()) / 2;
00042     move(x, y);
00043
00044     QVBoxLayout *mainLayout = new QVBoxLayout(this);
00045
00046     // --- Header: Logo, Title, Leave Button ---
00047     QHBoxLayout *headerLayout = new QHBoxLayout();
00048     logoLabel = new QLabel(this);
00049     QPixmap logoPixmap(":/icons/snake.png");
00050     logoLabel->setPixmap(logoPixmap.scaled(50, 50, Qt::KeepAspectRatio, Qt::SmoothTransformation));
00051     titleLabel = new QLabel("ParuGramm Chat", this);
00052     titleLabel->setStyleSheet("font-size: 28px; font-weight: bold; color: #27ec2f; margin-left: 12px;");
00053     leaveButton = new QPushButton("Выйти", this);
00054     leaveButton->setFixedSize(100, 40);
00055     leaveButton->setStyleSheet(
00056         "QPushButton { background-color: #d5394e; color: #FFFFFF; border: 1px solid #E0E0E0; border-radius: 5px;
      padding: 8px; font-weight: bold; font-size: 16px; }"
00057         "QPushButton:hover { background-color: #a5182a; border: 2px solid #E0E0E0; }"
00058         "QPushButton:pressed { background-color: #8b1423; }"
00059         );
00060     headerLayout->addWidget(logoLabel);
00061     headerLayout->addWidget(titleLabel);
00062     headerLayout->addStretch();
00063     headerLayout->addWidget(leaveButton);
00064     mainLayout->addLayout(headerLayout);
00065
00066     // --- User List Section ---
00067     userListLabel = new QLabel(this);
00068     userListLabel->setStyleSheet("font-size: 18px; color: #AFFF99; font-weight: bold; margin-bottom: 2px; margin-left:
      3px;");
00069     userList = new QListWidget(this);
00070     userList->setMinimumHeight(105);
00071     userList->setMaximumHeight(120);
00072     userList->setStyleSheet(
00073         "QListWidget { background-color: #263238; border: 1px solid #4CAF50; border-radius: 7px; font-size: 16px;
      padding-left: 5px; color: #E0E0E0; }"
00074         "QListWidget::item { padding: 7px 0 7px 10px; border-radius: 5px; height: 32px; }"
00075         "QListWidget::item:selected { background: #4CAF50; color: #E0E0E0; border-radius: 3px; }"
00076         );
00077     mainLayout->addSpacing(2);
00078     mainLayout->addWidget(userListLabel);
00079     mainLayout->addWidget(userList);
00080
00081     // --- Chat Area ---
00082     chatArea = new QTextBrowser(this);
00083     chatArea->setReadOnly(true);
00084     chatArea->setMinimumHeight(400);
00085     chatArea->setStyleSheet("font-size: 20px; background-color: #1A2526; border: none; padding: 10px;");
00086     chatArea->setOpenExternalLinks(false);
00087     chatArea->setOpenLinks(false);
00088     mainLayout->addSpacing(10);
00089     mainLayout->addWidget(chatArea);
```

```
00090
00091     // --- Message Input/Buttons ---
00092     QHBoxLayout *inputLayout = new QHBoxLayout();
00093     messageInput = new QLineEdit(this);
00094     messageInput->setMinimumHeight(40);
00095     messageInput->setPlaceholderText("Написать сообщение...");
00096     messageInput->setStyleSheet(
00097         "QLineEdit { background-color: #1e292e; border: 1px solid #4CAF50; border-radius: 5px; padding: 5px; font-size:
      16px; color: #E0E0E0; }"
00098         "QLineEdit:placeholder { color: #A0A0A0; }"
00099         );
00100     sendButton = new QPushButton("Отправить", this);
00101     sendButton->setFixedSize(125, 40);
00102     emojiButton = new QPushButton("", this);
00103     emojiButton->setFixedSize(50, 40);
00104     fileButton = new QPushButton("", this);
00105     fileButton->setFixedSize(50, 40);
00106
00107     inputLayout->addWidget(messageInput);
00108     inputLayout->addWidget(sendButton);
00109     inputLayout->addWidget(emojiButton);
00110     inputLayout->addWidget(fileButton);
00111
00112     mainLayout->addLayout(inputLayout);
00113
00114     setLayout(mainLayout);
00115     setupConnections();
00116
00117
00118     setStyleSheet(R"(
00119         QWidget {
00120             background-color: #1A2526;
00121             color: #E0E0E0;
00122         }
00123         QLineEdit {
00124             background-color: #263238;
00125             border: 1px solid #4CAF50;
00126             border-radius: 5px;
00127             padding: 5px;
00128             font-size: 16px;
00129             color: #E0E0E0;
00130         }
00131         QPushButton {
00132             background-color: #4CAF50;
00133             color: #E0E0E0;
00134             border: 1px solid #E0E0E0;
00135             border-radius: 5px;
00136             padding: 8px;
00137             font-weight: bold;
00138             font-size: 16px;
00139         }
00140         QPushButton:hover {
00141             background-color: #388E3C;
00142             border: 2px solid #E0E0E0;
00143         }
00144         QPushButton:pressed {
00145             background-color: #2E7D32;
00146         }
00147     )");
00148
00149     qDebug() << "ChatWindow created for room ID:" << roomId;
00150     show();
00151
00152
00153     if (client && roomId != -1) {   //хз можно убрать удмаю
00154         client->sendRequest("GET_USERS:" + std::to_string(roomId));
00155     }
00156 }
00157
00158 QColor ChatWindow::getUserColor(int userId) const {
00159     if (userId == client->getClientId()) {
00160         qDebug() << "getUserColor: userId =" << userId << ", returning #FFFFFF (client)";
00161         return QColor("#FFFFFF");
00162     }
00163     if (userId < 0) {
00164         qDebug() << "getUserColor: userId =" << userId << ", returning #A0A0A0 (invalid)";
00165         return QColor("#A0A0A0");
00166     }
00167     QString color = colorList[userId % colorList.size()];
00168     qDebug() << "getUserColor: userId =" << userId << ", returning" << color;
00169     return QColor(color);
00170 }
00171
00172 QString ChatWindow::getUserName(int userId) const {
00173     if (userId == client->getClientId())
00174         return "Я";
00175     if (userId < 0)
```

```
00176         return "User?";
00177     return QString("User%1").arg(userId);
00178 }
00179
00180 void ChatWindow::setupConnections() {
00181     connect(sendButton, &QPushButton::clicked, this, &ChatWindow::sendMessage);
00182     connect(messageInput, &QLineEdit::returnPressed, this, &ChatWindow::sendMessage);
00183     connect(emojiButton, &QPushButton::clicked, this, &ChatWindow::selectEmoji);
00184     connect(fileButton, &QPushButton::clicked, this, &ChatWindow::sendFile);
00185     connect(chatArea, &QTextBrowser::anchorClicked, this, &ChatWindow::handleAnchorClicked);
00186
00187     connect(leaveButton, &QPushButton::clicked, this, &ChatWindow::onLeaveRoomClicked);
00188     connect(client, &Client::userJoined, this, &ChatWindow::handleUserJoined);
00189     connect(client, &Client::userLeft, this, &ChatWindow::handleUserLeft);
00190     connect(client, &Client::leftRoom, this, &ChatWindow::handleLeftRoom);
00191
00192     if (client) {
00193         connect(client, &Client::messageReceived, this, [this](int roomId, int senderId, const QString& message) {
00194             if (this->roomId == roomId) {
00195                 lastSenderId = senderId;
00196
00197                 QString senderName = getUserName(senderId);
00198                 QColor color = getUserColor(senderId);
00199                 qDebug() << "messageReceived: roomId =" << roomId << ", senderId =" << senderId << ", color =" << color.name() <<
      ", message =" << message;
00200
00201                 QString timeStr = QDateTime::currentDateTime().toString("hh:mm");
00202                 QTextCursor cursor = chatArea->textCursor();
00203                 cursor.movePosition(QTextCursor::End);
00204
00205                 if (lastWasEvent) {
00206                     chatArea->append("");
00207                     qDebug() << "messageReceived: Added empty line due to lastWasEvent";
00208                 }
00209
00210                 // Set block alignment to left
00211                 QTextBlockFormat blockFormat;
00212                 blockFormat.setAlignment(Qt::AlignLeft);
00213                 blockFormat.setBottomMargin(15); // Add margin between messages
00214                 cursor.insertBlock(blockFormat);
00215
00216                 // Set color and font for sender name
00217                 QTextCharFormat format;
00218                 format.setForeground(color);
00219                 format.setFont(QFont("Arial", 16, QFont::Bold));
00220                 cursor.setCharFormat(format);
00221                 cursor.insertText(senderName + ": ");
00222
00223                 // Message text
00224                 format.setFont(QFont("Arial", 16));
00225                 cursor.setCharFormat(format);
00226                 cursor.insertText(message + " ");
00227
00228                 // Time in grey
00229                 QTextCharFormat timeFormat;
00230                 timeFormat.setForeground(QColor("#A0A0A0"));
00231                 timeFormat.setFont(QFont("Arial", 10));
00232                 cursor.setCharFormat(timeFormat);
00233                 cursor.insertText(timeStr);
00234
00235                 chatArea->setTextCursor(cursor);
00236                 chatArea->verticalScrollBar()->setValue(chatArea->verticalScrollBar()->maximum());
00237                 lastWasEvent = false;
00238             }
00239         });
00240         connect(client, &Client::errorReceived, this, [this](const QString& error) {
00241             QMessageBox::warning(this, "Ошибка", error);
00242         });
00243         connect(client, &Client::fileReceived, this, [this](int roomId, int senderId, const QString& fileName, const
      QByteArray& fileData) {
00244             if (this->roomId == roomId) {
00245                 QString timestamp = QDateTime::currentDateTime().toString("yyyyMMdd_hhmmss_zzz");
00246                 QString fileId = QString("%1_%2_%3").arg(fileName).arg(senderId).arg(timestamp);
00247                 FileData fileDataStruct;
00248                 fileDataStruct.fileName = fileName;
00249                 fileDataStruct.data = fileData;
00250                 fileDataMap[fileId] = fileDataStruct;
00251                 lastSenderId = senderId;
00252                 QString senderName = getUserName(senderId);
00253                 QColor color = getUserColor(senderId);
00254                 qDebug() << "fileReceived: roomId =" << roomId << ", senderId =" << senderId << ", color =" << color.name() << ",
      fileName =" << fileName;
00255
00256                 QString timeStr = QDateTime::currentDateTime().toString("hh:mm");
00257                 QTextCursor cursor = chatArea->textCursor();
00258                 cursor.movePosition(QTextCursor::End);
00259
```

```
00260             if (lastWasEvent) {
00261                 chatArea->append("");
00262                 qDebug() « "fileReceived: Added empty line due to lastWasEvent";
00263             }
00264 
00265             // Set block alignment to left
00266             QTextBlockFormat blockFormat;
00267             blockFormat.setAlignment(Qt::AlignLeft);
00268             blockFormat.setBottomMargin(15); // Add margin between messages
00269             cursor.insertBlock(blockFormat);
00270 
00271             // Set color and font for sender name
00272             QTextCharFormat format;
00273             format.setForeground(color);
00274             format.setFont(QFont("Arial", 16, QFont::Bold));
00275             cursor.setCharFormat(format);
00276             cursor.insertText(senderName + " отправил файл: ");
00277 
00278             // File link
00279             QString link = QString("<a href=\"file://%1\" style=\"color: #64B5F6; text-decoration: underline;
    font-weight: bold;\">%2</a>")
00280                              .arg(fileId)
00281                              .arg(fileName);
00282             cursor.insertHtml(link + " ");
00283 
00284             // Time in grey
00285             QTextCharFormat timeFormat;
00286             timeFormat.setForeground(QColor("#A0A0A0"));
00287             timeFormat.setFont(QFont("Arial", 10));
00288             cursor.setCharFormat(timeFormat);
00289             cursor.insertText(timeStr);
00290 
00291             chatArea->setTextCursor(cursor);
00292             chatArea->verticalScrollBar()->setValue(chatArea->verticalScrollBar()->maximum());
00293             lastWasEvent = false;
00294         }
00295     });
00296     connect(client, &Client::fileSent, this, &ChatWindow::handleFileSent);
00297     connect(client, &Client::fileSentConfirmed, this, []() {});
00298     connect(client, &Client::usersReceived, this, [this](const QStringList& users) {
00299         lastUserList = users;
00300         updateUserList(users);
00301     });
00302 }
00303 }
00304 
00305 void ChatWindow::updateUserList(const QStringList& users) {
00306     userList->clear();
00307     int count = users.size();
00308     userListLabel->setText(QString("Список участников (%1)").arg(count));
00309     for (const QString& userStr : users) {
00310         int userId = -1;
00311         QString name = userStr;
00312         if (userStr == "User" + QString::number(client->getClientId())) {
00313             userId = client->getClientId();
00314             name = "Я";
00315         } else {
00316             QRegularExpression re("User(\\d+)");
00317             auto match = re.match(userStr);
00318             if (match.hasMatch()) {
00319                 userId = match.captured(1).toInt();
00320                 if (userId == client->getClientId()) {
00321                     name = "Я";
00322                 } else {
00323                     name = QString("User%1").arg(userId);
00324                 }
00325             }
00326         }
00327         QColor dotColor = getUserColor(userId);
00328         QString dot = QString("<span style='color:%1; font-size:18px; font-weight:bold;
    vertical-align:middle;'></span>").arg(dotColor.name());
00329         QString userHtml = QString("%1 <span style='font-weight:bold; color:%2;'>%3</span>")
00330                                .arg(dot)
00331                                .arg((name == "Я") ? "#FFFFFF" : dotColor.name())
00332                                .arg(name);
00333         QListWidgetItem *item = new QListWidgetItem();
00334         item->setText("");
00335         item->setData(Qt::DisplayRole, "");
00336         item->setData(Qt::UserRole, name);
00337         item->setData(Qt::UserRole + 1, dotColor);
00338         item->setData(Qt::UserRole + 2, userHtml);
00339         userList->addItem(item);
00340         userList->setItemWidget(item, new QLabel(userHtml));
00341     }
00342 }
00343 
00344 void ChatWindow::sendMessage() {
```

```
00345     QString message = messageInput->text().trimmed();
00346     if (message.isEmpty() || roomId == -1 || !client) return;
00347     client->sendRequest("MESSAGE:" + std::to_string(roomId) + ":" + message.toStdString());
00348     messageInput->clear();
00349 }
00350
00351 void ChatWindow::selectEmoji() {
00352     EmojiPanel *emojiPanel = new EmojiPanel(this);
00353     connect(emojiPanel, &EmojiPanel::emojiSelected, this, [this](const QString& emoji) {
00354         messageInput->insert(emoji);
00355     });
00356     emojiPanel->show();
00357 }
00358
00359 void ChatWindow::sendFile() {
00360     QString filePath = QFileDialog::getOpenFileName(this, "Выберите файл");
00361     if (!filePath.isEmpty() && client) client->sendFile(filePath.toStdString());
00362 }
00363
00364 void ChatWindow::handleFileSent(const QString& fileName, const QByteArray& fileData) {
00365     QString timestamp = QDateTime::currentDateTime().toString("yyyyMMdd_hhmmss_zzz");
00366     QString fileId = QString("%1_%2_%3").arg(fileName).arg(client->getClientId()).arg(timestamp);
00367     FileData fileDataStruct;
00368     fileDataStruct.fileName = fileName;
00369     fileDataStruct.data = fileData;
00370     fileDataMap[fileId] = fileDataStruct;
00371     lastSenderId = client->getClientId();
00372     QString senderName = "Я";
00373     QColor color = getUserColor(client->getClientId());
00374     qDebug() « "handleFileSent: senderId =" « client->getClientId() « ", color =" « color.name() « ", fileName =" «
    fileName;
00375
00376     QString timeStr = QDateTime::currentDateTime().toString("hh:mm");
00377     QTextCursor cursor = chatArea->textCursor();
00378     cursor.movePosition(QTextCursor::End);
00379
00380     if (lastWasEvent) {
00381         chatArea->append("");
00382         qDebug() « "handleFileSent: Added empty line due to lastWasEvent";
00383     }
00384
00385     // Set block alignment to left
00386     QTextBlockFormat blockFormat;
00387     blockFormat.setAlignment(Qt::AlignLeft);
00388     blockFormat.setBottomMargin(15); // Add margin between messages
00389     cursor.insertBlock(blockFormat);
00390
00391     // Set color and font for sender name
00392     QTextCharFormat format;
00393     format.setForeground(color);
00394     format.setFont(QFont("Arial", 16, QFont::Bold));
00395     cursor.setCharFormat(format);
00396     cursor.insertText(senderName + " отправил файл: ");
00397
00398     // File link
00399     QString link = QString("<a href=\"file://%1\" style=\"color: #64B5F6; text-decoration: underline; font-weight:
    bold;\">%2</a>")
00400                        .arg(fileId)
00401                        .arg(fileName);
00402     cursor.insertHtml(link + " ");
00403
00404     // Time in grey
00405     QTextCharFormat timeFormat;
00406     timeFormat.setForeground(QColor("#A0A0A0"));
00407     timeFormat.setFont(QFont("Arial", 10));
00408     cursor.setCharFormat(timeFormat);
00409     cursor.insertText(timeStr);
00410
00411     chatArea->setTextCursor(cursor);
00412     chatArea->verticalScrollBar()->setValue(chatArea->verticalScrollBar()->maximum());
00413     lastWasEvent = false;
00414 }
00415
00416 void ChatWindow::handleAnchorClicked(const QUrl& url) {
00417     QString fileId = url.toString().mid(7);
00418     if (fileDataMap.contains(fileId)) {
00419         QString fileName = fileDataMap[fileId].fileName;
00420         QString savePath = QFileDialog::getSaveFileName(this, "Сохранить файл", fileName, "Все файлы (*.*)");
00421         if (!savePath.isEmpty()) {
00422             QFile file(savePath);
00423             if (file.open(QIODevice::WriteOnly)) {
00424                 file.write(fileDataMap[fileId].data);
00425                 file.close();
00426                 QString timeStr = QDateTime::currentDateTime().toString("hh:mm");
00427                 QTextCursor cursor = chatArea->textCursor();
00428                 cursor.movePosition(QTextCursor::End);
00429
```

```
00430             if (lastWasEvent) {
00431                 chatArea->append("");
00432                 qDebug() « "handleAnchorClicked: Added empty line due to lastWasEvent";
00433             }
00434 
00435             // Set block alignment to left
00436             QTextBlockFormat blockFormat;
00437             blockFormat.setAlignment(Qt::AlignLeft);
00438             blockFormat.setBottomMargin(15); // Add margin between messages
00439             cursor.insertBlock(blockFormat);
00440 
00441             // Set color and font for message
00442             QTextCharFormat format;
00443             format.setForeground(QColor("#FFFFFF"));
00444             format.setFont(QFont("Arial", 16, QFont::Bold));
00445             cursor.setCharFormat(format);
00446             cursor.insertText("я сохранил файл: ");
00447             cursor.insertText(fileName + " ");
00448 
00449             // Time in grey
00450             QTextCharFormat timeFormat;
00451             timeFormat.setForeground(QColor("#A0A0A0"));
00452             timeFormat.setFont(QFont("Arial", 10));
00453             cursor.setCharFormat(timeFormat);
00454             cursor.insertText(timeStr);
00455 
00456             chatArea->setTextCursor(cursor);
00457             chatArea->verticalScrollBar()->setValue(chatArea->verticalScrollBar()->maximum());
00458             lastWasEvent = false;
00459         }
00460     }
00461 }
00462 }
00463 
00464 void ChatWindow::handleUserJoined(int roomId, int userId) {
00465     if (this->roomId == roomId) {
00466         QString message = QString("%1 вошел в чат").arg(getUserName(userId));
00467         QTextCursor cursor = chatArea->textCursor();
00468         cursor.movePosition(QTextCursor::End);
00469 
00470         QTextBlockFormat blockFormat;
00471         blockFormat.setAlignment(Qt::AlignCenter);
00472         blockFormat.setBottomMargin(2);
00473         cursor.insertBlock(blockFormat);
00474 
00475         QTextCharFormat format;
00476         format.setForeground(QColor("#8f8f8f"));
00477         format.setFont(QFont("Arial", 16));
00478         cursor.setCharFormat(format);
00479         cursor.insertText(message);
00480 
00481         chatArea->setTextCursor(cursor);
00482         chatArea->verticalScrollBar()->setValue(chatArea->verticalScrollBar()->maximum());
00483         lastWasEvent = true;
00484         client->sendRequest("GET_USERS:" + std::to_string(roomId));
00485     }
00486 }
00487 
00488 void ChatWindow::handleUserLeft(int roomId, int userId) {
00489     if (this->roomId == roomId) {
00490         QString message = QString("%1 вышел из чата").arg(getUserName(userId));
00491         QTextCursor cursor = chatArea->textCursor();
00492         cursor.movePosition(QTextCursor::End);
00493 
00494         QTextBlockFormat blockFormat;
00495         blockFormat.setAlignment(Qt::AlignCenter);
00496         blockFormat.setBottomMargin(2);
00497         cursor.insertBlock(blockFormat);
00498 
00499         QTextCharFormat format;
00500         format.setForeground(QColor("#8f8f8f"));
00501         format.setFont(QFont("Arial", 16));
00502         cursor.setCharFormat(format);
00503         cursor.insertText(message);
00504 
00505         chatArea->setTextCursor(cursor);
00506         chatArea->verticalScrollBar()->setValue(chatArea->verticalScrollBar()->maximum());
00507         lastWasEvent = true;
00508         client->sendRequest("GET_USERS:" + std::to_string(roomId));
00509     }
00510 }
00511 
00512 void ChatWindow::onLeaveRoomClicked() {
00513     if (client && roomId != -1) {
00514         client->leaveRoom();
00515     }
00516     QApplication::quit();
```

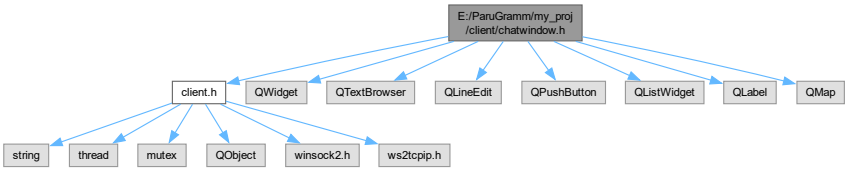
```
00517 }
00518
00519 void ChatWindow::handleLeftRoom() {
00520     this->hide();
00521     MainWindow *mainWindow = new MainWindow();
00522     mainWindow->show();
00523     QTimer::singleShot(100, this, [this]() {
00524         client->disconnect();
00525         this->deleteLater();
00526     });
00527 }
```

5.3 Файл E:/ParuGramm/my_proj/client/chatwindow.h

Заголовочный файл для окна чата мессенджера.

```
#include "client.h"
#include <QWidget>
#include <QTextBrowser>
#include <QLineEdit>
#include <QPushButton>
#include <QListWidget>
#include <QLabel>
#include <QMap>
```

Граф включаемых заголовочных файлов для chatwindow.h:



Классы

- class ChatWindow

  Класс для управления окном чата в комнате.

5.3.1 Подробное описание

Заголовочный файл для окна чата мессенджера.

См. определение в файле chatwindow.h

5.4 chatwindow.h

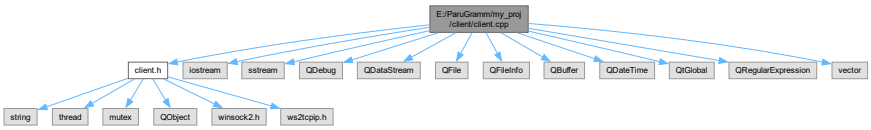
См. документацию.

```
00001 #ifndef CHATWINDOW_H
00002 #define CHATWINDOW_H
00003
00004 #include "client.h"
00005 #include <QWidget>
00006 #include <QTextBrowser>
00007 #include <QLineEdit>
00008 #include <QPushButton>
00009 #include <QListWidget>
00010 #include <QLabel>
00011 #include <QMap>
00012
00017
00022 class ChatWindow : public QWidget {
00023     Q_OBJECT
00024
00025 public:
00032     ChatWindow(int roomId, Client *client, QWidget *parent = nullptr);
00033
00034 private slots:
00038     void sendMessage();
00039
00043     void selectEmoji();
00044
00048     void sendFile();
00049
00055     void handleFileSent(const QString& fileName, const QByteArray& fileData);
00056
00061     void handleAnchorClicked(const QUrl& url);
00062
00066     void onLeaveRoomClicked();
00067
00073     void handleUserJoined(int roomId, int userId);
00074
00080     void handleUserLeft(int roomId, int userId);
00081
00085     void handleLeftRoom();
00086
00087 private:
00091     void setupConnections();
00092
00097     void updateUserList(const QStringList& users);
00098
00104     QColor getUserColor(int userId) const;
00105
00111     QString getUserName(int userId) const;
00112
00113     int roomId;
00114     Client *client;
00115     QTextBrowser *chatArea;
00116     QListWidget *userList;
00117     QLabel *userListLabel;
00118     QLineEdit *messageInput;
00119     QPushButton *sendButton;
00120     QPushButton *emojiButton;
00121     QPushButton *fileButton;
00122     QPushButton *leaveButton;
00123     QLabel *logoLabel;
00124     QLabel *titleLabel;
00125     struct FileData {
00126         QString fileName;
00127         QByteArray data;
00128     };
00129     QMap<QString, FileData> fileDataMap;
00130     int lastSenderId;
00131     bool lastWasEvent;
00132     QStringList lastUserList;
00133 };
00134
00135 #endif // CHATWINDOW_H
```

5.5 Файл E:/ParuGramm/my_proj/client/client.cpp

```
#include "client.h"
#include <iostream>
```

#include <sstream>
#include <QDebug>
#include <QDataStream>
#include <QFile>
#include <QFileInfo>
#include <QBuffer>
#include <QDateTime>
#include <QtGlobal>
#include <QRegularExpression>
#include <vector>
Граф включаемых заголовочных файлов для client.cpp:



Функции

- void initLogging ()

5.5.1 Функции

5.5.1.1 initLogging()

void initLogging ()

См. определение в файле client.cpp строка 14

5.6 client.cpp

См. документацию.

```
00001 #include "client.h"
00002 #include <iostream>
00003 #include <sstream>
00004 #include <QDebug>
00005 #include <QDataStream>
00006 #include <QFile>
00007 #include <QFileInfo>
00008 #include <QBuffer>
00009 #include <QDateTime>
00010 #include <QtGlobal>
00011 #include <QRegularExpression>
00012 #include <vector>
00013
00014 void initLogging() {
00015     QFile logFile("client.log");
00016     if (logFile.open(QIODevice::WriteOnly | QIODevice::Text)) {
00017         qInstallMessageHandler([](QtMsgType, const QMessageLogContext&, const QString& msg) {
00018             QFile logFile("client.log");
00019             if (logFile.open(QIODevice::Append | QIODevice::Text)) {
00020                 QTextStream out(&logFile);
00021                 out « QDateTime::currentDateTime().toString() « ": " « msg « "\n";
00022             }
00023         });
00024     }
00025 }
00026
```

```
00027 Client* Client::p_instance = nullptr;
00028 SingletonDestroyer Client::destroyer;
00029
00030 Client& Client::getInstance() {
00031     if (!p_instance) {
00032         p_instance = new Client();
00033         destroyer.initialize(p_instance);
00034     }
00035     return *p_instance;
00036 }
00037
00038 SingletonDestroyer::~SingletonDestroyer() {
00039     delete p_instance;
00040 }
00041
00042 void SingletonDestroyer::initialize(Client* p) {
00043     p_instance = p;
00044 }
00045
00046 Client::Client() : is_connected_(false), socket_(INVALID_SOCKET), currentRoomId(-1), pendingRoomId(-1),
      clientId(-1) {
00047     initLogging();
00048     WSADATA wsaData;
00049     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
00050         qDebug() << "WSAStartup failed";
00051         std::cerr << "WSAStartup failed.\n";
00052     }
00053 }
00054
00055 Client::~Client() {
00056     disconnect();
00057     WSACleanup();
00058 }
00059
00060 bool Client::connectToServer(const std::string& ip, int port) {
00061     qDebug() << "Attempting to connect to" << ip.c_str() << ":" << port;
00062     std::lock_guard<std::mutex> lock(mutex_);
00063     if (is_connected_) {
00064         qDebug() << "Already connected";
00065         return true;
00066     }
00067
00068     socket_ = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
00069     if (socket_ == INVALID_SOCKET) {
00070         qDebug() << "Socket creation failed with error:" << WSAGetLastError();
00071         std::cerr << "Socket creation failed.\n";
00072         return false;
00073     }
00074
00075     sockaddr_in serverAddr;
00076     serverAddr.sin_family = AF_INET;
00077     serverAddr.sin_port = htons(port);
00078     serverAddr.sin_addr.s_addr = inet_addr(ip.c_str());
00079
00080     if (::connect(socket_, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
00081         qDebug() << "Connection failed with error:" << WSAGetLastError();
00082         closesocket(socket_);
00083         socket_ = INVALID_SOCKET;
00084         std::cerr << "Connection failed.\n";
00085         return false;
00086     }
00087
00088     qDebug() << "Connected successfully";
00089     is_connected_ = true;
00090     receive_thread_ = std::thread(&Client::receiveThread, this);
00091     return true;
00092 }
00093
00094 void Client::disconnect() {
00095     std::lock_guard<std::mutex> lock(mutex_);
00096     if (!is_connected_) return;
00097
00098     is_connected_ = false;
00099     if (socket_ != INVALID_SOCKET) {
00100         shutdown(socket_, SD_BOTH);
00101         closesocket(socket_);
00102         socket_ = INVALID_SOCKET;
00103     }
00104     if (receive_thread_.joinable()) {
00105         receive_thread_.join(); // БЛОКИРУЕМСЯ пока поток не завершится!
00106     }
00107     clientId = -1;
00108     currentRoomId = -1;
00109     pendingRoomId = -1;
00110 }
00111
00112 void Client::sendRequest(const std::string& request) {
```

```
00113     std::lock_guard<std::mutex> lock(mutex_);
00114     if (!is_connected_) {
00115         qDebug() « "Cannot send request: not connected";
00116         emit errorReceived("Клиент не подключен к серверу");
00117         return;
00118     }
00119 
00120     if (request.find("JOIN_ROOM:") == 0) {
00121         std::stringstream ss(request.substr(10));
00122         std::string roomIdStr;
00123         std::getline(ss, roomIdStr, ':');
00124         try {
00125             pendingRoomId = std::stoi(roomIdStr);
00126             qDebug() « "Pending room ID set to:" « pendingRoomId;
00127         } catch (...) {
00128             qDebug() « "Invalid room ID in JOIN_ROOM";
00129         }
00130     }
00131 
00132     std::string data = request + "\n";
00133     qDebug() « "Sending request:" « request.c_str();
00134     int sent = send(socket_, data.c_str(), data.size(), 0);
00135     if (sent == SOCKET_ERROR) {
00136         qDebug() « "Send failed with error:" « WSAGetLastError();
00137         disconnect();
00138         emit errorReceived("Ошибка отправки запроса");
00139     }
00140 }
00141 
00142 void Client::sendFile(const std::string& filePath) {
00143     QFile file(QString::fromStdString(filePath));
00144     if (!file.open(QIODevice::ReadOnly)) {
00145         qDebug() « "Failed to open file:" « filePath.c_str();
00146         emit errorReceived("Не удалось открыть файл");
00147         return;
00148     }
00149 
00150     QByteArray fileData = file.readAll();
00151     if (fileData.size() > 10 * 1024 * 1024) { // Ограничение 10 МБ
00152         qDebug() « "File too large:" « filePath.c_str();
00153         emit errorReceived("Файл слишком большой (макс. 10 МБ)");
00154         return;
00155     }
00156 
00157     QString fileName = QFileInfo(file).fileName();
00158     if (fileName.isEmpty()) {
00159         qDebug() « "Invalid file name (empty):" « fileName;
00160         emit errorReceived("Недопустимое имя файла");
00161         return;
00162     }
00163 
00164     QByteArray fileNameUtf8 = fileName.toUtf8();
00165 
00166     QBuffer buffer;
00167     buffer.open(QIODevice::WriteOnly);
00168     QDataStream ds(&buffer);
00169     ds.setVersion(QDataStream::Qt_5_15);
00170     ds.setByteOrder(QDataStream::LittleEndian);
00171 
00172     uint32_t magic = 0xFA57F11E;
00173     int32_t nameSize = fileNameUtf8.size();
00174     int32_t dataSize = fileData.size();
00175 
00176     ds « magic « nameSize « dataSize;
00177     buffer.write(fileNameUtf8);
00178     buffer.write(fileData);
00179 
00180     QByteArray packet = buffer.data();
00181     qDebug() « "Packet size:" « packet.size();
00182     qDebug() « "Packet header (hex):" « packet.left(12).toHex();
00183     qDebug() « "nameSize:" « nameSize « ", dataSize:" « dataSize;
00184     qDebug() « "fileNameUtf8 (hex):" « fileNameUtf8.toHex();
00185 
00186     std::lock_guard<std::mutex> lock(mutex_);
00187     if (is_connected_) {
00188         qDebug() « "Sending file:" « fileName « "size:" « fileData.size();
00189         int totalSent = 0;
00190         while (totalSent < packet.size()) {
00191             int sent = send(socket_, packet.constData() + totalSent, packet.size() - totalSent, 0);
00192             if (sent == SOCKET_ERROR) {
00193                 qDebug() « "Send file failed with error:" « WSAGetLastError();
00194                 disconnect();
00195                 emit errorReceived("Ошибка отправки файла");
00196                 break;
00197             }
00198             totalSent += sent;
00199         }
```

```
00200         if (totalSent == packet.size()) {
00201             emit fileSent(fileName, fileData);
00202         }
00203     } else {
00204         qDebug() « "Cannot send file: not connected";
00205         emit errorReceived("Клиент не подключен к серверу");
00206     }
00207 }
00208
00209 void Client::receiveThread() {
00210     std::string readBuffer;
00211     char tempBuffer[65536];
00212     int bytesReceived;
00213
00214     qDebug() « "Receive thread started";
00215     while (is_connected_) {
00216         bytesReceived = recv(socket_, tempBuffer, sizeof(tempBuffer), 0);
00217         if (bytesReceived == SOCKET_ERROR) {
00218             int error = WSAGetLastError();
00219             qDebug() « "Receive failed with error:" « error;
00220             disconnect();
00221             emit errorReceived("Ошибка получения данных: " + QString::number(error));
00222             break;
00223         }
00224         if (bytesReceived == 0) {
00225             qDebug() « "Connection closed by server";
00226             disconnect();
00227             emit errorReceived("Сервер закрыл соединение");
00228             break;
00229         }
00230
00231         qDebug() « "Received" « bytesReceived « "bytes";
00232         readBuffer.append(tempBuffer, bytesReceived);
00233
00234         if (readBuffer.size() >= sizeof(uint32_t)) {
00235             QDataStream ds(QByteArray::fromRawData(readBuffer.data(), readBuffer.size()));
00236             ds.setVersion(QDataStream::Qt_5_15);
00237             ds.setByteOrder(QDataStream::LittleEndian);
00238             uint32_t magic;
00239             ds » magic;
00240             if (magic == 0xFA57F11E) {
00241                 qDebug() « "Processing file data";
00242                 processFile(readBuffer);
00243                 continue;
00244             }
00245         }
00246
00247         size_t newlinePos;
00248         while ((newlinePos = readBuffer.find('\n')) != std::string::npos) {
00249             std::string line = readBuffer.substr(0, newlinePos);
00250             if (!line.empty() && line.back() == '\r') {
00251                 line.pop_back();
00252             }
00253             if (!line.empty()) {
00254                 qDebug() « "Processing response:" « QString::fromStdString(line);
00255                 processResponse(line);
00256             }
00257             readBuffer.erase(0, newlinePos + 1);
00258         }
00259     }
00260     qDebug() « "Receive thread stopped";
00261 }
00262
00263 void Client::processResponse(const std::string& response) {
00264     qDebug() « "Received raw response:" « QString::fromStdString(response);
00265
00266     // Разделяем ответ на отдельные строки
00267     std::vector<std::string> lines;
00268     size_t start = 0;
00269     size_t end = response.find('\n');
00270
00271     while (end != std::string::npos) {
00272         lines.push_back(response.substr(start, end - start));
00273         start = end + 1;
00274         end = response.find('\n', start);
00275     }
00276     lines.push_back(response.substr(start));
00277
00278     // Обрабатываем каждую строку
00279     for (const auto& line : lines) {
00280         if (line.empty()) continue;
00281
00282         std::stringstream ss(line);
00283         std::string command;
00284         std::getline(ss, command, ':');
00285         qDebug() « "Processing command:" « QString::fromStdString(command);
00286
```

```
00287         if (command == "SUCCESS") {
00288             std::string action;
00289             std::getline(ss, action, ':');
00290 
00291             if (action == "Room created") {
00292                 std::string roomIdStr;
00293                 std::getline(ss, roomIdStr);
00294                 try {
00295                     int roomId = std::stoi(roomIdStr);
00296                     currentRoomId = roomId;
00297                     qDebug() << "Emitting roomCreated signal with ID:" << roomId;
00298                     emit roomCreated(roomId);
00299                 } catch (...) {
00300                     qDebug() << "Invalid room ID in response";
00301                     emit errorReceived("Некорректный ID комнаты");
00302                 }
00303             }
00304             else if (action == "Joined room") {
00305                 currentRoomId = pendingRoomId;
00306                 pendingRoomId = -1;
00307                 qDebug() << "Emitting joinedRoom signal with room ID:" << currentRoomId;
00308                 emit joinedRoom();
00309 
00310                 // После входа в комнату запросим список пользователей
00311                 if (currentRoomId != -1) {
00312                     sendRequest("GET_USERS:" + std::to_string(currentRoomId));
00313                 }
00314             }
00315             else if (action == "Left room") {
00316                 qDebug() << "Left room confirmed by server";
00317                 emit leftRoom();
00318             }
00319             else if (action == "Message sent") {
00320                 qDebug() << "Message sent confirmed";
00321             }
00322             else if (action == "File received") {
00323                 qDebug() << "File received confirmed by server";
00324                 emit fileSentConfirmed();
00325             }
00326             else if (action == "Client ID") {
00327                 std::string clientIdStr;
00328                 std::getline(ss, clientIdStr);
00329                 try {
00330                     clientId = std::stoi(clientIdStr);
00331                     qDebug() << "Assigned client ID:" << clientId;
00332                 } catch (...) {
00333                     qDebug() << "Invalid client ID in response";
00334                     emit errorReceived("Некорректный ID клиента");
00335                 }
00336             }
00337             else {
00338                 qDebug() << "Unknown success action:" << QString::fromStdString(action);
00339                 emit errorReceived("Неизвестное действие: " + QString::fromStdString(action));
00340             }
00341         }
00342         else if (command == "MESSAGE") {
00343             std::string roomIdStr, senderIdStr, message;
00344             std::getline(ss, roomIdStr, ':');
00345             std::getline(ss, senderIdStr, ':');
00346             std::getline(ss, message);
00347             try {
00348                 int roomId = std::stoi(roomIdStr);
00349                 int senderId = std::stoi(senderIdStr);
00350                 qDebug() << "Emitting messageReceived for room" << roomId << "from sender" << senderId;
00351                 emit messageReceived(roomId, senderId, QString::fromStdString(message));
00352             } catch (...) {
00353                 qDebug() << "Invalid message format";
00354                 emit errorReceived("Некорректный формат сообщения");
00355             }
00356         }
00357         else if (command == "MESSAGES") {
00358             std::string roomIdStr, messages;
00359             std::getline(ss, roomIdStr, ':');
00360             std::getline(ss, messages);
00361             try {
00362                 int roomId = std::stoi(roomIdStr);
00363                 if (messages != "None") {
00364                     std::stringstream ms(messages);
00365                     std::string messageEntry;
00366                     while (std::getline(ms, messageEntry, ';')) {
00367                         std::stringstream me(messageEntry);
00368                         std::string senderIdStr, message;
00369                         std::getline(me, senderIdStr, ',');
00370                         std::getline(me, message);
00371                         int senderId = std::stoi(senderIdStr);
00372                         qDebug() << "Emitting messageReceived for room" << roomId << "from sender" << senderId;
00373                         emit messageReceived(roomId, senderId, QString::fromStdString(message));
```

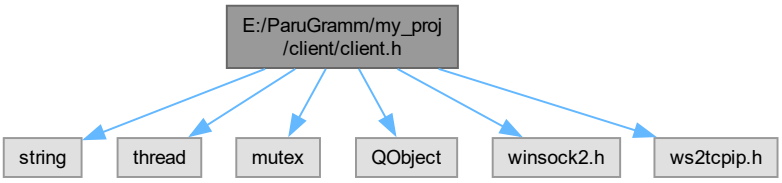
```
00374                 }
00375             } else {
00376                 qDebug() « "No messages for room" « roomId;
00377             }
00378         } catch (...) {
00379             qDebug() « "Invalid messages format";
00380             emit errorReceived("Некорректный формат сообщений");
00381         }
00382     }
00383     else if (command == "ROOM_LIST") {
00384         std::string roomsStr;
00385         std::getline(ss, roomsStr);
00386         QStringList rooms;
00387         if (roomsStr != "None") {
00388             std::stringstream roomSs(roomsStr);
00389             std::string room;
00390             while (std::getline(roomSs, room, ';')) {
00391                 rooms « QString::fromStdString(room);
00392             }
00393         }
00394         qDebug() « "Emitting roomListReceived with" « rooms.size() « "rooms";
00395         emit roomListReceived(rooms);
00396     }
00397     else if (command == "USER_LIST") {
00398         std::string usersStr;
00399         std::getline(ss, usersStr);
00400         QStringList users;
00401         if (usersStr != "None") {
00402             std::stringstream userSs(usersStr);
00403             std::string user;
00404             while (std::getline(userSs, user, ';')) {
00405                 users « "User" + QString::fromStdString(user);
00406             }
00407         }
00408         qDebug() « "Emitting usersReceived with" « users.size() « "users";
00409         emit usersReceived(users);
00410     }
00411     else if (command == "ERROR") {
00412         std::string error;
00413         std::getline(ss, error);
00414         qDebug() « "Emitting errorReceived:" « QString::fromStdString(error);
00415         emit errorReceived(QString::fromStdString(error));
00416     }
00417     else if (command == "USER_JOINED") {
00418         std::string roomIdStr, userIdStr;
00419         std::getline(ss, roomIdStr, ':');
00420         std::getline(ss, userIdStr);
00421         try {
00422             int roomId = std::stoi(roomIdStr);
00423             int userId = std::stoi(userIdStr);
00424             emit userJoined(roomId, userId);
00425         } catch (...) {
00426             qDebug() « "Invalid USER_JOINED format";
00427         }
00428     }
00429     else if (command == "USER_LEFT") {
00430         std::string roomIdStr, userIdStr;
00431         std::getline(ss, roomIdStr, ':');
00432         std::getline(ss, userIdStr);
00433         try {
00434             int roomId = std::stoi(roomIdStr);
00435             int userId = std::stoi(userIdStr);
00436             emit userLeft(roomId, userId);
00437         } catch (...) {
00438             qDebug() « "Invalid USER_LEFT format";
00439         }
00440     }
00441     else {
00442         qDebug() « "Unknown response:" « QString::fromStdString(line);
00443         emit errorReceived("Неизвестный ответ от сервера: " + QString::fromStdString(line));
00444     }
00445 }
00446 }
00447
00448 void Client::processFile(std::string& buffer) {
00449     QDataStream ds(QByteArray::fromRawData(buffer.data(), buffer.size()));
00450     ds.setVersion(QDataStream::Qt_5_15);
00451     ds.setByteOrder(QDataStream::LittleEndian);
00452
00453     uint32_t magic, nameSize, dataSize;
00454     int32_t senderId;
00455     ds » magic » senderId » nameSize » dataSize;
00456
00457     if (buffer.size() < static_cast<size_t>(sizeof(uint32_t) * 3 + sizeof(int32_t) + nameSize + dataSize)) {
00458         qDebug() « "Incomplete file data, waiting for more";
00459         return;
00460     }
```

```
00461
00462     QByteArray nameData = QByteArray(buffer.data() + 16, nameSize);
00463     QByteArray fileData = QByteArray(buffer.data() + 16 + nameSize, dataSize);
00464     qDebug() « "Received nameData (hex):" « nameData.toHex();
00465     qDebug() « "Received fileData size:" « fileData.size();
00466     qDebug() « "Sender ID:" « senderId;
00467
00468     QString fileName = QString::fromUtf8(nameData);
00469     qDebug() « "Parsed fileName:" « fileName;
00470
00471     qDebug() « "Emitting fileReceived for file:" « fileName;
00472     emit fileReceived(currentRoomId, senderId, fileName, fileData);
00473
00474     buffer.erase(0, sizeof(uint32_t) * 3 + sizeof(int32_t) + nameSize + dataSize);
00475 }
00476
00477 // Важно: leaveRoom НЕ вызывает disconnect()
00478 void Client::leaveRoom() {
00479     if (currentRoomId != -1) {
00480         sendRequest("LEAVE_ROOM:" + std::to_string(currentRoomId));
00481         currentRoomId = -1;
00482         // disconnect(); // НЕ вызываем здесь!
00483     }
00484 }
```

5.7 Файл E:/ParuGramm/my_proj/client/client.h

Заголовочный файл для клиентской части мессенджера.

#include <string>
#include <thread>
#include <mutex>
#include <QObject>
#include <winsock2.h>
#include <ws2tcpip.h>
Граф включаемых заголовочных файлов для client.h:



Классы

- class SingletonDestroyer

  Класс для уничтожения единственного экземпляра Client.
- class Client

  Класс для управления клиентской частью мессенджера с TCP-сокетами.

5.7.1 Подробное описание

Заголовочный файл для клиентской части мессенджера.

См. определение в файле client.h

5.8 client.h

См. документацию.

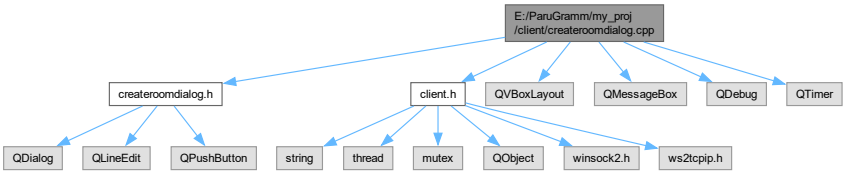
```
00001 #ifndef CLIENT_H
00002 #define CLIENT_H
00003
00004 #include <string>
00005 #include <thread>
00006 #include <mutex>
00007 #include <QObject>
00008 #include <winsock2.h>
00009 #include <ws2tcpip.h>
00010
00015
00020 class SingletonDestroyer {
00021 private:
00022     Client* p_instance;
00023 public:
00027     ~SingletonDestroyer();
00028
00033     void initialize(Client* p);
00034 };
00035
00040 class Client : public QObject {
00041     Q_OBJECT
00042 public:
00047     static Client& getInstance();
00048
00055     bool connectToServer(const std::string& ip, int port);
00056
00060     void disconnect();
00061
00066     void sendRequest(const std::string& request);
00067
00072     void sendFile(const std::string& filePath);
00073
00078     void setRoomId(int id) { currentRoomId = id; }
00079
00084     int getRoomId() const { return currentRoomId; }
00085
00090     int getClientId() const { return clientId; }
00091
00096     bool isConnected() const { return is_connected_; }
00097
00101     void leaveRoom();
00102
00103 signals:
00111     void messageReceived(int roomId, int senderId, const QString& message);
00112
00118     void roomCreated(int roomId);
00119
00124     void joinedRoom();
00125
00131     void roomListReceived(const QStringList& rooms);
00132
00138     void errorReceived(const QString& error);
00139
00148     void fileReceived(int roomId, int senderId, const QString& fileName, const QByteArray& fileData);
00149
00156     void fileSent(const QString& fileName, const QByteArray& fileData);
00157
00162     void fileSentConfirmed();
00163
00169     void usersReceived(const QStringList& users);
00170
00177     void userJoined(int roomId, int userId);
00178
00185     void userLeft(int roomId, int userId);
00186
00191     void leftRoom();
00192
00193 private:
00197     Client();
00198
00202     ~Client();
00203
00204     Client(const Client&) = delete;
00205     Client& operator=(const Client&) = delete;
00206
00207     friend class SingletonDestroyer;
00208
00209     static Client* p_instance;
00210     static SingletonDestroyer destroyer;
00211
00215     void receiveThread();
```

```
00216
00221     void processResponse(const std::string& response);
00222
00227     void processFile(std::string& buffer);
00228
00229     bool is_connected_;
00230     SOCKET socket_;
00231     std::mutex mutex_;
00232     std::thread receive_thread_;
00233     int currentRoomId;
00234     int pendingRoomId;
00235     int clientId;
00236 };
00237
00238 #endif // CLIENT_H
```

5.9 Файл E:/ParuGramm/my_proj/client/createroomdialog.cpp

```
#include "createroomdialog.h"
#include "client.h"
#include <QVBoxLayout>
#include <QMessageBox>
#include <QDebug>
#include <QTimer>
```

Граф включаемых заголовочных файлов для createroomdialog.cpp:



5.10 createroomdialog.cpp

См. документацию.

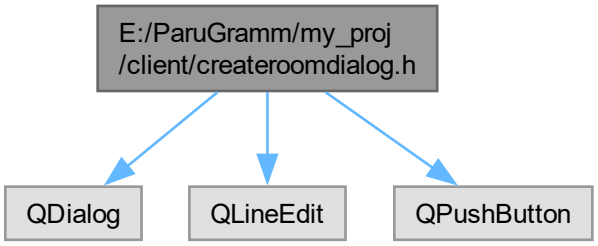
```
00001 #include "createroomdialog.h"
00002 #include "client.h"
00003 #include <QVBoxLayout>
00004 #include <QMessageBox>
00005 #include <QDebug>
00006 #include <QTimer>
00007
00008 CreateRoomDialog::CreateRoomDialog(QWidget *parent) : QDialog(parent) {
00009     setWindowTitle("Создать комнату");
00010     QVBoxLayout *layout = new QVBoxLayout(this);
00011
00012     roomNameEdit = new QLineEdit(this);
00013     roomNameEdit->setPlaceholderText("Название комнаты");
00014     roomNameEdit->setStyleSheet("padding: 12px; font-family: 'Roboto';");
00015
00016     passwordEdit = new QLineEdit(this);
00017     passwordEdit->setPlaceholderText("Пароль (опционально)");
00018     passwordEdit->setEchoMode(QLineEdit::Password);
00019     passwordEdit->setStyleSheet("padding: 12px; font-family: 'Roboto';");
00020
00021     createButton = new QPushButton("Создать", this);
00022     createButton->setStyleSheet("padding: 12px; font-family: 'Roboto';");
00023
00024     layout->addWidget(roomNameEdit);
00025     layout->addWidget(passwordEdit);
00026     layout->addWidget(createButton);
00027
00028     connect(createButton, &QPushButton::clicked, this, &CreateRoomDialog::onCreateClicked);
```

```
00029
00030     setFixedSize(300, 500);
00031     setWindowIcon(QIcon(":/icons/snake.png"));
00032     setStyleSheet(R"(
00033         QWidget {
00034             background-color: #12171c;
00035             color: #ECF0F1;
00036         }
00037         QLineEdit {
00038             background-color: #34495E;
00039             border: 1px solid #2ECC71;
00040             border-radius: 5px;
00041             padding: 5px;
00042             font-size: 14px;
00043             font: bold;
00044             color: #ECF0F1;
00045         }
00046         QPushButton {
00047             background-color: #2ECC71;
00048             color: #2C3E50;
00049             border: 1px solid #ECF0F1;
00050             border-radius: 5px;
00051             padding: 8px 15px;
00052             font-weight: bold;
00053             font-size: 14px;
00054         }
00055         QPushButton:hover {
00056             background-color: #27AE60;
00057             border: 1px solid #ECF0F1;
00058         }
00059     )");
00060 }
00061
00062 void CreateRoomDialog::onCreateClicked() {
00063     QString name = roomNameEdit->text().trimmed();
00064     password = passwordEdit->text(); // Сохраняем введённый пароль
00065     if (name.isEmpty()) {
00066         qDebug() << "Room name is empty";
00067         QMessageBox::warning(this, "Ошибка", "Введите название комнаты");
00068         return;
00069     }
00070
00071     Client& client = Client::getInstance();
00072     bool isConnected = client.connectToServer("127.0.0.1", 12346);
00073     if (!isConnected) {
00074         qDebug() << "Failed to connect to server";
00075         QMessageBox::warning(this, "Ошибка", "Не удалось подключиться к серверу");
00076         return;
00077     }
00078
00079     qDebug() << "Creating room:" << name << "with password:" << password;
00080
00081     bool responseReceived = false;
00082     connect(&client, &Client::roomCreated, this, [&responseReceived, this, name]() {
00083         responseReceived = true;
00084         int roomId = Client::getInstance().getRoomId();
00085         qDebug() << "Room creation successful, room ID:" << roomId;
00086         // Присоединяемся с правильным паролем
00087         Client::getInstance().sendRequest("JOIN_ROOM:" + std::to_string(roomId) + ":" + this->password.toStdString());
00088         accept();
00089     });
00090     connect(&client, &Client::errorReceived, this, [&responseReceived, this](const QString& error) {
00091         responseReceived = true;
00092         qDebug() << "Error creating room:" << error;
00093         QMessageBox::warning(this, "Ошибка", error);
00094         reject();
00095     });
00096
00097     client.sendRequest("CREATE_ROOM:" + name.toStdString() + ":" + password.toStdString());
00098
00099     QTimer timer;
00100     timer.setSingleShot(true);
00101     connect(&timer, &QTimer::timeout, this, [&responseReceived, this]() {
00102         if (!responseReceived) {
00103             qDebug() << "No response from server for CREATE_ROOM";
00104             QMessageBox::warning(this, "Ошибка", "Сервер не отвечает");
00105             reject();
00106         }
00107     });
00108     timer.start(10000);
00109 }
00110
00111 QString CreateRoomDialog::getPassword() const {
00112     return password;
00113 }
```

5.11 Файл E:/ParuGramm/my_proj/client/createroomdialog.h

Заголовочный файл для диалога создания комнаты.

#include <QDialog>
#include <QLineEdit>
#include <QPushButton>
Граф включаемых заголовочных файлов для createroomdialog.h:



Классы

- class CreateRoomDialog

  Класс для управления диалогом создания комнаты мессенджера.

5.11.1 Подробное описание

Заголовочный файл для диалога создания комнаты.

См. определение в файле createroomdialog.h

5.12 createroomdialog.h

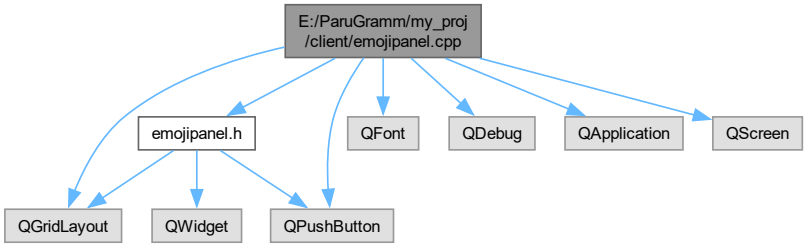
См. документацию.

```
00001 #ifndef CREATEROOMDIALOG_H
00002 #define CREATEROOMDIALOG_H
00003
00004 #include <QDialog>
00005 #include <QLineEdit>
00006 #include <QPushButton>
00007
00012
00017 class CreateRoomDialog : public QDialog {
00018     Q_OBJECT
00019 public:
00024     explicit CreateRoomDialog(QWidget *parent = nullptr);
00025
00030     QString getPassword() const;
00031
00032 private slots:
00036     void onCreateClicked();
00037
00038 private:
00039     QLineEdit *roomNameEdit;
00040     QLineEdit *passwordEdit;
00041     QPushButton *createButton;
00042     QString password;
00043 };
00044
00045 #endif // CREATEROOMDIALOG_H
```

5.13 Файл E:/ParuGramm/my_proj/client/emojipanel.cpp

```
#include "emojipanel.h"
#include <QPushButton>
#include <QGridLayout>
#include <QFont>
#include <QDebug>
#include <QApplication>
#include <QScreen>
```

Граф включаемых заголовочных файлов для emojipanel.cpp:



5.14 emojipanel.cpp

См. документацию.

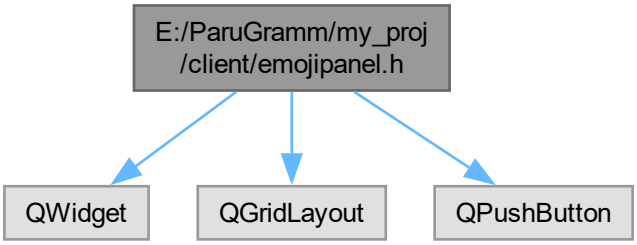
```
00001 #include "emojipanel.h"
00002 #include <QPushButton>
00003 #include <QGridLayout>
00004 #include <QFont>
00005 #include <QDebug>
00006 #include <QApplication>
00007 #include <QScreen>
00008
00009 EmojiPanel::EmojiPanel(QWidget *parent) : QWidget(parent) {
00010     setupUI();
00011     setWindowFlags(Qt::Popup);
00012     setFixedSize(520, 280); // 10x5 кнопок (40x40) + отступы
00013     setStyleSheet("background: #0c1217; border: 2px solid #0ef16e; border-radius: 5px;");
00014     setWindowIcon(QIcon(":/icons/snake.png"));
00015
00016     // Центрирование на экране
00017     QScreen *screen = QApplication::primaryScreen();
00018     QRect screenGeometry = screen->geometry();
00019     int x = (screenGeometry.width() - width()) / 2;
00020     int y = (screenGeometry.height() - height()) / 2;
00021     move(x, y);
00022
00023     qDebug() « "EmojiPanel created";
00024 }
00025
00026 void EmojiPanel::setupUI() {
00027     QGridLayout *layout = new QGridLayout(this);
00028     layout->setSpacing(5);
00029
00030     const QStringList emojis = {
00031         "", "", "", "", "", "", "", "", "", "",
00032         "", "", "", "", "", "", "", "", "", "",
00033         "", "", "", "", "", "", "", "", "", "",
00034         "", "", "", "", "", "", "", "", "", "",
00035         "", "", "", "", "", "", "", "", "", ""
00036     };
00037
00038     int index = 0;
00039     for (int row = 0; row < 5; ++row) {
00040         for (int col = 0; col < 10; ++col) {
00041             if (index >= emojis.size()) break;
```

```
00042           QPushButton *btn = new QPushButton(emojis[index], this);
00043           btn->setFont(QFont("Segoe UI Emoji", 48));
00044           btn->setFixedSize(48, 48);
00045           btn->setStyleSheet("QPushButton:hover { background: #2ECC71; }");
00046           connect(btn, &QPushButton::clicked, [this, btn]() {
00047               qDebug() << "Emoji button clicked:" << btn->text();
00048               emit emojiSelected(btn->text());
00049           });
00050           layout->addWidget(btn, row, col);
00051           index++;
00052       }
00053   }
00054 }
```

5.15 Файл E:/ParuGramm/my_proj/client/emojipanel.h

Заголовочный файл для панели выбора эмодзи.

#include <QWidget>
#include <QGridLayout>
#include <QPushButton>
Граф включаемых заголовочных файлов для emojipanel.h:



Классы

- class EmojiPanel

  Класс для управления панелью выбора эмодзи в мессенджере.

5.15.1 Подробное описание

Заголовочный файл для панели выбора эмодзи.

См. определение в файле emojipanel.h

5.16 emojipanel.h

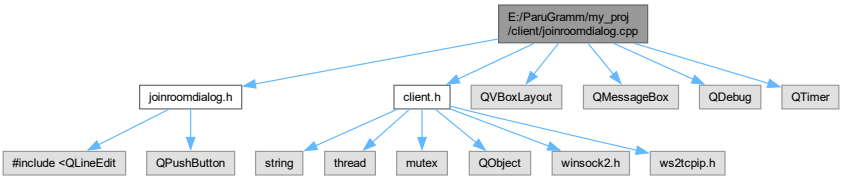
См. документацию.

```
00001 #ifndef EMOJIPANEL_H
00002 #define EMOJIPANEL_H
00003
00004 #include <QWidget>
00005 #include <QGridLayout>
00006 #include <QPushButton>
00007
00012
00017 class EmojiPanel : public QWidget {
00018     Q_OBJECT
00019
00020 public:
00025     explicit EmojiPanel(QWidget *parent = nullptr);
00026
00027 signals:
00033     void emojiSelected(const QString &emoji);
00034
00035 private:
00039     void setupUI();
00040 };
00041
00042 #endif // EMOJIPANEL_H
```

5.17 Файл E:/ParuGramm/my_proj/client/joinroomdialog.cpp

```
#include "joinroomdialog.h"
#include "client.h"
#include <QVBoxLayout>
#include <QMessageBox>
#include <QDebug>
#include <QTimer>
```

Граф включаемых заголовочных файлов для joinroomdialog.cpp:



5.18 joinroomdialog.cpp

См. документацию.

```
00001 #include "joinroomdialog.h"
00002 #include "client.h"
00003 #include <QVBoxLayout>
00004 #include <QMessageBox>
00005 #include <QDebug>
00006 #include <QTimer>
00007
00008 JoinRoomDialog::JoinRoomDialog(QWidget *parent) : QDialog(parent) {
00009     setWindowTitle("Присоединиться к комнате");
00010     QVBoxLayout *layout = new QVBoxLayout(this);
00011
00012     ipEdit = new QLineEdit(this);
00013     ipEdit->setPlaceholderText("IP сервера");
00014     ipEdit->setText("127.0.0.1");
00015
```

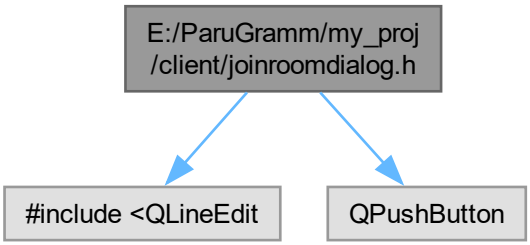
```
00016     portEdit = new QLineEdit(this);
00017     portEdit->setPlaceholderText("Порт");
00018     portEdit->setText("12346");
00019 
00020     roomIdEdit = new QLineEdit(this);
00021     roomIdEdit->setPlaceholderText("ID комнаты");
00022 
00023     passwordEdit = new QLineEdit(this);
00024     passwordEdit->setPlaceholderText("Пароль (опционально)");
00025     passwordEdit->setEchoMode(QLineEdit::Password);
00026 
00027     joinButton = new QPushButton("Присоединиться", this);
00028 
00029     layout->addWidget(ipEdit);
00030     layout->addWidget(portEdit);
00031     layout->addWidget(roomIdEdit);
00032     layout->addWidget(passwordEdit);
00033     layout->addWidget(joinButton);
00034 
00035     connect(joinButton, &QPushButton::clicked, this, &JoinRoomDialog::onJoinClicked);
00036 
00037     // Применяем стиль и размер
00038     setFixedSize(300, 500);
00039     setWindowIcon(QIcon(":/icons/snake.png"));
00040     setStyleSheet(R"(
00041         QWidget {
00042             background-color: #12171c;
00043             color: #ECF0F1;
00044         }
00045         QLineEdit {
00046             background-color: #34495E;
00047             border: 1px solid #2ECC71;
00048             border-radius: 5px;
00049             padding: 5px;
00050             font-size: 14px;
00051             font: bold;
00052             color: #ECF0F1;
00053         }
00054         QPushButton {
00055             background-color: #2ECC71;
00056             color: #2C3E50;
00057             border: 1px solid #ECF0F1;
00058             border-radius: 5px;
00059             padding: 8px 15px;
00060             font-weight: bold;
00061             font-size: 14px;
00062         }
00063         QPushButton:hover {
00064             background-color: #27AE60;
00065             border: 1px solid #ECF0F1;
00066         }
00067     )");
00068 
00069     qDebug() « "JoinRoomDialog created";
00070 }
00071 
00072 void JoinRoomDialog::onJoinClicked() {
00073     QString ip = ipEdit->text().trimmed();
00074     QString portStr = portEdit->text().trimmed();
00075     QString roomIdStr = roomIdEdit->text().trimmed();
00076     QString password = passwordEdit->text();
00077 
00078     if (ip.isEmpty() || portStr.isEmpty() || roomIdStr.isEmpty()) {
00079         qDebug() « "Missing input: IP, port, or room ID";
00080         QMessageBox::warning(this, "Ошибка", "Введите IP, порт и ID комнаты");
00081         return;
00082     }
00083 
00084     bool ok;
00085     int port = portStr.toInt(&ok);
00086     if (!ok || port <= 0 || port > 65535) {
00087         qDebug() « "Invalid port:" « portStr;
00088         QMessageBox::warning(this, "Ошибка", "Некорректный порт");
00089         return;
00090     }
00091 
00092     int roomId = roomIdStr.toInt(&ok);
00093     if (!ok || roomId <= 0) {
00094         qDebug() « "Invalid room ID:" « roomIdStr;
00095         QMessageBox::warning(this, "Ошибка", "Некорректный ID комнаты");
00096         return;
00097     }
00098 
00099     Client& client = Client::getInstance();
00100     qDebug() « "Attempting to connect to" « ip « ":" « port;
00101     bool isConnected = client.connectToServer(ip.toStdString(), port);
00102     if (!isConnected) {
```

```
00103         qDebug() « "Connection failed";
00104         QMessageBox::warning(this, "Ошибка", "Не удалось подключиться к серверу");
00105         return;
00106     }
00107
00108     qDebug() « "Joining room:" « roomId « "with password:" « password;
00109
00110     bool responseReceived = false;
00111     connect(&client, &Client::joinedRoom, this, [&responseReceived, this, roomId]() {
00112         responseReceived = true;
00113         qDebug() « "Room join successful, room ID:" « roomId;
00114         accept();
00115     });
00116     connect(&client, &Client::errorReceived, this, [&responseReceived, this](const QString& error) {
00117         responseReceived = true;
00118         qDebug() « "Error joining room:" « error;
00119         QMessageBox::warning(this, "Ошибка", error);
00120         reject();
00121     });
00122
00123     client.sendRequest("JOIN_ROOM:" + roomIdStr.toStdString() + ":" + password.toStdString());
00124
00125     QTimer timer;
00126     timer.setSingleShot(true);
00127     connect(&timer, &QTimer::timeout, this, [&responseReceived, this]() {
00128         if (!responseReceived) {
00129             qDebug() « "No response from server for JOIN_ROOM";
00130             QMessageBox::warning(this, "Ошибка", "Сервер не отвечает");
00131             reject();
00132         }
00133     });
00134     timer.start(10000);
00135 }
00136
00137 void JoinRoomDialog::setRoomId(const QString& id) {
00138     roomIdEdit->setText(id);
00139     roomIdEdit->setEnabled(false); // Запрещаем редактирование, так как комната уже выбрана
00140 }
00141
00142 void JoinRoomDialog::showPasswordField() {
00143     passwordEdit->setVisible(true);
00144 }
00145
00146 void JoinRoomDialog::hidePasswordField() {
00147     passwordEdit->setVisible(false);
00148 }
```

5.19 Файл E:/ParuGramm/my_proj/client/joinroomdialog.h

Заголовочный файл для диалога присоединения к комнате.

#include <#include <QLineEdit>
#include <QPushButton>
Граф включаемых заголовочных файлов для joinroomdialog.h:



Классы

- class JoinRoomDialog

  Класс для управления диалогом присоединения к комнате мессенджера.

5.19.1 Подробное описание

Заголовочный файл для диалога присоединения к комнате.

См. определение в файле joinroomdialog.h

5.20 joinroomdialog.h

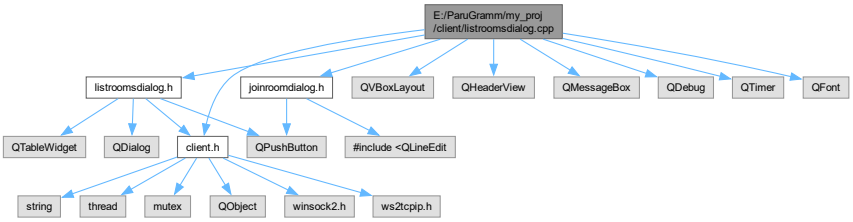
См. документацию.

```
00001 #ifndef JOINROOMDIALOG_H
00002 #define JOINROOMDIALOG_H
00003
00004 #include <QDialog botanical
00005 #include <QLineEdit>
00006 #include <QPushButton>
00007
00012
00017 class JoinRoomDialog : public QDialog {
00018     Q_OBJECT
00019 public:
00024     explicit JoinRoomDialog(QWidget *parent = nullptr);
00025
00030     void setRoomId(const QString& id);
00031
00035     void showPasswordField();
00036
00040     void hidePasswordField();
00041
00042 private slots:
00046     void onJoinClicked();
00047
00048 private:
00049     QLineEdit *ipEdit;
00050     QLineEdit *portEdit;
00051     QLineEdit *roomIdEdit;
00052     QLineEdit *passwordEdit;
00053     QPushButton *joinButton;
00054 };
00055
00056 #endif // JOINROOMDIALOG_H
```

5.21 Файл E:/ParuGramm/my_proj/client/listroomsdialog.cpp

```
#include "listroomsdialog.h"
#include <QVBoxLayout>
#include <QHeaderView>
#include <QMessageBox>
#include <QDebug>
#include <joinroomdialog.h>
#include <QTimer>
#include <client.h>
```

#include <QFont>
Граф включаемых заголовочных файлов для listroomsdialog.cpp:



5.22 listroomsdialog.cpp

См. документацию.

```
00001 #include "listroomsdialog.h"
00002 #include <QVBoxLayout>
00003 #include <QHeaderView>
00004 #include <QMessageBox>
00005 #include <QDebug>
00006 #include <joinroomdialog.h>
00007 #include <QTimer>
00008 #include <client.h>
00009 #include <QFont>
00010
00011 ListRoomsDialog::ListRoomsDialog(QWidget *parent)
00012     : QDialog(parent), client(Client::getInstance()) {
00013     setWindowTitle("Список комнат");
00014     setMinimumSize(900, 650); // Увеличили размер окна
00015     setWindowFlags(windowFlags() & ~Qt::WindowContextHelpButtonHint); // Убираем кнопку помощи
00016
00017     QVBoxLayout *layout = new QVBoxLayout(this);
00018     layout->setContentsMargins(15, 15, 15, 15); // Отступы по краям
00019     layout->setSpacing(15); // Расстояние между элементами
00020
00021     // Настройка таблицы
00022     roomsTable = new QTableWidget(this);
00023     roomsTable->setColumnCount(4);
00024     roomsTable->setHorizontalHeaderLabels({"ID", "Название комнаты", "Пароль", "Участники"});
00025
00026     // Настройка заголовков
00027     QFont headerFont = roomsTable->horizontalHeader()->font();
00028     headerFont.setBold(true);
00029     headerFont.setPointSize(10);
00030     roomsTable->horizontalHeader()->setFont(headerFont);
00031
00032     roomsTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch); // Растягиваем на всю ширину
00033     roomsTable->horizontalHeader()->setStretchLastSection(true); // Последняя колонка растягивается
00034     roomsTable->setSelectionBehavior(QAbstractItemView::SelectRows);
00035     roomsTable->setEditTriggers(QAbstractItemView::NoEditTriggers); // Запрещаем редактирование
00036     roomsTable->setSortingEnabled(true);
00037     roomsTable->sortByColumn(3, Qt::DescendingOrder);
00038     roomsTable->verticalHeader()->setVisible(false); // Скрываем вертикальные заголовки
00039     roomsTable->setSelectionMode(QAbstractItemView::SingleSelection); // Выбор только одной строки
00040
00041     // Настройка кнопок
00042     QHBoxLayout *buttonLayout = new QHBoxLayout();
00043     buttonLayout->setSpacing(50); // Расстояние между кнопками
00044
00045     refreshButton = new QPushButton("Обновить", this);
00046     refreshButton->setFixedWidth(150); // Фиксированная ширина
00047     refreshButton->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
00048
00049     joinButton = new QPushButton("Войти", this);
00050     joinButton->setFixedWidth(150); // Фиксированная ширина
00051     joinButton->setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
00052
00053     buttonLayout->addStretch(); // Добавляем растягивающееся пространство
00054     buttonLayout->addWidget(refreshButton);
00055     buttonLayout->addWidget(joinButton);
00056     buttonLayout->addStretch(); // Добавляем растягивающееся пространство
00057
```

```
00058     layout->addWidget(roomsTable);
00059     layout->addLayout(buttonLayout);
00060
00061     connect(refreshButton, &QPushButton::clicked, this, &ListRoomsDialog::onRefreshClicked);
00062     connect(joinButton, &QPushButton::clicked, this, &ListRoomsDialog::onJoinClicked);
00063     connect(&client, &Client::roomListReceived, this, &ListRoomsDialog::onRoomListReceived);
00064
00065     // Таймер для автообновления
00066     QTimer *refreshTimer = new QTimer(this);
00067     connect(refreshTimer, &QTimer::timeout, this, &ListRoomsDialog::onRefreshClicked);
00068     refreshTimer->start(30000);
00069
00070     // Первоначальная загрузка
00071     onRefreshClicked();
00072
00073     // Стилизация
00074     setStyleSheet(R"(
00075         QDialog {
00076             background-color: #1c2935;
00077         }
00078
00079         QTableWidget {
00080             background-color: #34495E;
00081             border: 2px solid #2C3E50;
00082             color: #ECF0F1;
00083             font: bold;
00084             font-size: 12px;
00085             selection-background-color: #2980B9;
00086             selection-color: white;
00087             gridline-color: #2C3E50;
00088         }
00089
00090         QHeaderView::section {
00091             background-color: #2C3E50;
00092             color: #ECF0F1;
00093             padding: 8px;
00094             border: none;
00095             font-weight: bold;
00096         }
00097
00098         QPushButton {
00099             border-radius: 4px;
00100             padding: 8px 15px;
00101             font-weight: bold;
00102             font-size: 14px;
00103             min-width: 80px;
00104         }
00105
00106         QPushButton#refreshButton {
00107             background-color: #3498DB;
00108             color: white;
00109         }
00110
00111         QPushButton#refreshButton:hover {
00112             background-color: #2980B9;
00113         }
00114
00115         QPushButton#refreshButton:pressed {
00116             background-color: #1B6CA8;
00117         }
00118
00119         QPushButton#joinButton {
00120             background-color: #2ECC71;
00121             color: #2C3E50;
00122         }
00123
00124         QPushButton#joinButton:hover {
00125             background-color: #27AE60;
00126         }
00127
00128         QPushButton#joinButton:pressed {
00129             background-color: #219653;
00130         }
00131     )");
00132
00133     // Присваиваем имена для стилизации
00134     refreshButton->setObjectName("refreshButton");
00135     joinButton->setObjectName("joinButton");
00136 }
00137
00138 // Остальные методы остаются без изменений
00139 void ListRoomsDialog::onRefreshClicked() {
00140     if (!client.isConnected()) {
00141         bool connected = client.connectToServer("127.0.0.1", 12346);
00142         if (!connected) {
00143             QMessageBox::warning(this, "Ошибка", "Не удалось подключиться к серверу");
00144             return;
```

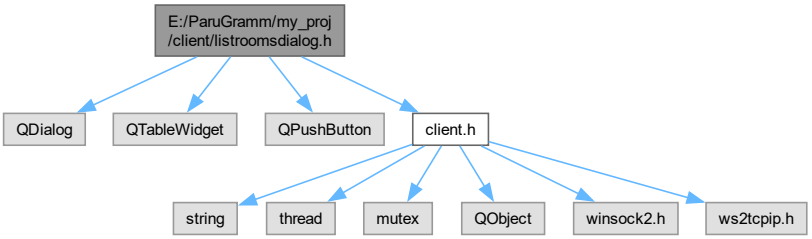
```
00145         }
00146     }
00147     client.sendRequest("LIST_ROOMS");
00148 }
00149
00150 void ListRoomsDialog::onRoomListReceived(const QStringList& rooms) {
00151     roomsTable->setRowCount(0);
00152     roomsTable->setSortingEnabled(false);
00153
00154     for (const QString& roomStr : rooms) {
00155         QStringList parts = roomStr.split(':');
00156         if (parts.size() == 4) {
00157             int row = roomsTable->rowCount();
00158             roomsTable->insertRow(row);
00159
00160             // ID
00161             QTableWidgetItem *idItem = new QTableWidgetItem(parts[0]);
00162             idItem->setTextAlignment(Qt::AlignCenter);
00163             roomsTable->setItem(row, 0, idItem);
00164
00165             // Название
00166             QTableWidgetItem *nameItem = new QTableWidgetItem(parts[1]);
00167             nameItem->setTextAlignment(Qt::AlignLeft | Qt::AlignVCenter);
00168             roomsTable->setItem(row, 1, nameItem);
00169
00170             // Пароль
00171             QTableWidgetItem *passItem = new QTableWidgetItem(parts[2] == "Yes" ? "" : "");
00172             passItem->setTextAlignment(Qt::AlignCenter);
00173             roomsTable->setItem(row, 2, passItem);
00174
00175             // Участники
00176             QTableWidgetItem *partItem = new QTableWidgetItem(parts[3]);
00177             partItem->setTextAlignment(Qt::AlignCenter);
00178             roomsTable->setItem(row, 3, partItem);
00179         }
00180     }
00181
00182     roomsTable->setSortingEnabled(true);
00183     roomsTable->sortByColumn(3, Qt::DescendingOrder);
00184 }
00185
00186 void ListRoomsDialog::onJoinClicked() {
00187     QModelIndexList selected = roomsTable->selectionModel()->selectedRows();
00188     if (selected.isEmpty()) {
00189         QMessageBox::warning(this, "Ошибка", "Выберите комнату из списка");
00190         return;
00191     }
00192
00193     int row = selected.first().row();
00194     QString roomId = roomsTable->item(row, 0)->text();
00195     bool hasPassword = (roomsTable->item(row, 2)->text() == "");
00196
00197     JoinRoomDialog joinDialog(this);
00198     joinDialog.setRoomId(roomId);
00199
00200     if (hasPassword) {
00201         joinDialog.showPasswordField();
00202     } else {
00203         joinDialog.hidePasswordField();
00204     }
00205
00206     if (joinDialog.exec() == QDialog::Accepted) {
00207         accept();
00208     }
00209 }
```

5.23 Файл E:/ParuGramm/my_proj/client/listroomsdialog.h

Заголовочный файл для диалога списка комнат.

```
#include <QDialog>
#include <QTableWidget>
#include <QPushButton>
#include "client.h"
```

Граф включаемых заголовочных файлов для listroomsdialog.h:



Классы

- class ListRoomsDialog

  Класс для управления диалогом списка комнат.

5.23.1 Подробное описание

Заголовочный файл для диалога списка комнат.

См. определение в файле listroomsdialog.h

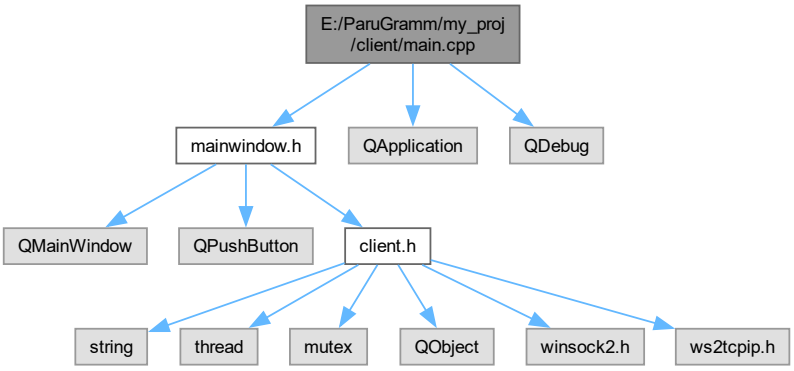
5.24 listroomsdialog.h

См. документацию.

```
00001 #ifndef LISTROOMSDIALOG_H
00002 #define LISTROOMSDIALOG_H
00003
00004 #include <QDialog>
00005 #include <QTableWidget>
00006 #include <QPushButton>
00007 #include "client.h"
00008
00013
00018 class ListRoomsDialog : public QDialog {
00019     Q_OBJECT
00020 public:
00025     explicit ListRoomsDialog(QWidget *parent = nullptr);
00026
00027 private slots:
00031     void onRefreshClicked();
00032
00036     void onJoinClicked();
00037
00042     void onRoomListReceived(const QStringList& rooms);
00043
00044 private:
00045     QTableWidget *roomsTable;
00046     QPushButton *refreshButton;
00047     QPushButton *joinButton;
00048     Client& client;
00049 };
00050
00051 #endif // LISTROOMSDIALOG_H
```

5.25 Файл E:/ParuGramm/my_proj/client/main.cpp

#include "mainwindow.h"
#include <QApplication>
#include <QDebug>
Граф включаемых заголовочных файлов для main.cpp:



Функции

- int main (int argc, char *argv[])

5.25.1 Функции

5.25.1.1 main()

```
int main (
                int argc,
                char * argv[])
```

См. определение в файле main.cpp строка 5

5.26 main.cpp

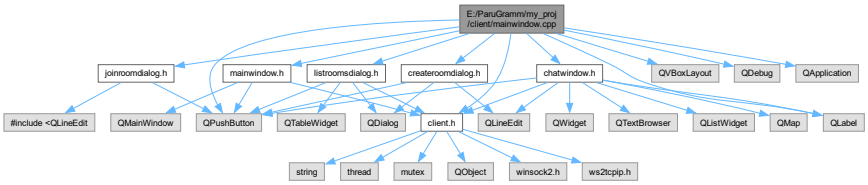
См. документацию.

```
00001 #include "mainwindow.h"
00002 #include <QApplication>
00003 #include <QDebug>
00004
00005 int main(int argc, char *argv[])
00006 {
00007     QApplication a(argc, argv);
00008     qDebug() << "Application started";
00009     MainWindow w;
00010     w.show();
00011     return a.exec();
00012 }
```

5.27 Файл E:/ParuGramm/my_proj/client/mainwindow.cpp

```
#include "mainwindow.h"
#include "createroomdialog.h"
#include "joinroomdialog.h"
#include "chatwindow.h"
#include "client.h"
#include <QVBoxLayout>
#include <QPushButton>
#include <QDebug>
#include <QApplication>
#include <QLabel>
#include <listroomsdialog.h>
```

Граф включаемых заголовочных файлов для mainwindow.cpp:



5.28 mainwindow.cpp

См. документацию.

```
00001 #include "mainwindow.h"
00002 #include "createroomdialog.h"
00003 #include "joinroomdialog.h"
00004 #include "chatwindow.h"
00005 #include "client.h"
00006 #include <QVBoxLayout>
00007 #include <QPushButton>
00008 #include <QDebug>
00009 #include <QApplication>
00010
00011 #include <QLabel>
00012
00013 #include <listroomsdialog.h>
00014
00015 MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
00016     setWindowTitle("ParuGramm");
00017     setWindowIcon(QIcon(":/icons/snake.png"));
00018
00019     QWidget *centralWidget = new QWidget(this);
00020     setCentralWidget(centralWidget);
00021     QVBoxLayout *layout = new QVBoxLayout(centralWidget);
00022
00023     // Логотип
00024     QLabel *logoLabel = new QLabel(this);
00025     QPixmap logoPixmap(":/icons/snake.svg");
00026     logoLabel->setPixmap(logoPixmap.scaled(300, 300, Qt::KeepAspectRatio, Qt::SmoothTransformation));
00027     logoLabel->setAlignment(Qt::AlignCenter);
00028
00029     // Надпись ParuGramm
00030     QLabel *titleLabel = new QLabel("ParuGramm", this);
00031     titleLabel->setAlignment(Qt::AlignCenter);
00032     titleLabel->setStyleSheet("font-size: 70px; font-weight: bold; color: #0BDA51;");
00033
00034     //кнопочки
00035     createRoomButton = new QPushButton("Создать комнату", this);
00036     joinRoomButton = new QPushButton("Присоединиться", this);
00037
00038     QPushButton *listRoomsButton = new QPushButton("Список комнат", this);
00039
00040
00041
```

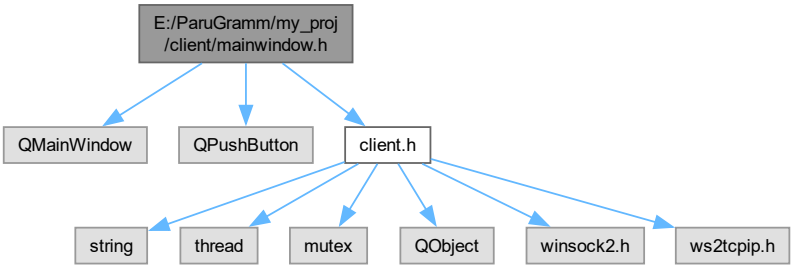
```
00042     layout->addSpacing(30);
00043     layout->addWidget(logoLabel);
00044     layout->addWidget(titleLabel);
00045     layout->addSpacing(90); // Отступ
00046     layout->addWidget(createRoomButton);
00047     layout->addSpacing(20); // Отступ перед кнопками
00048     layout->addWidget(joinRoomButton);
00049     layout->addSpacing(20); // Отступ перед кнопками
00050     layout->addWidget(listRoomsButton);
00051
00052     layout->addStretch();
00053
00054
00055
00056     connect(createRoomButton, &QPushButton::clicked, this, &MainWindow::onCreateRoomClicked);
00057     connect(joinRoomButton, &QPushButton::clicked, this, &MainWindow::onJoinRoomClicked);
00058     connect(listRoomsButton, &QPushButton::clicked, this, &MainWindow::onListRoomsClicked);
00059
00060     Client& client = Client::getInstance();
00061     connect(&client, &Client::roomCreated, this, &MainWindow::onRoomCreated);
00062     connect(&client, &Client::joinedRoom, this, &MainWindow::onJoinedRoom);
00063
00064     setFixedSize(600, 750);
00065     setStyleSheet(R"(
00066         QWidget {
00067             background-color: #2C3E50;
00068           _training
00069             color: #ECF0F1;
00070         }
00071         QPushButton {
00072             background-color: #2ECC71;
00073             color: #2C3E50;
00074             border: 1px solid #ECF0F1;
00075             border-radius: 5px;
00076             padding: 8px 15px;
00077             font-weight: bold;
00078             font-size: 14px;
00079         }
00080         QPushButton:hover {
00081             background-color: #27AE60;
00082             border: 1px solid #ECF0F1;
00083         }
00084     )");
00085
00086     qDebug() « "MainWindow created:" « this;
00087 }
00088
00089 MainWindow::~MainWindow() {
00090     qDebug() « "MainWindow destroyed";
00091 }
00092
00093 void MainWindow::onCreateRoomClicked() {
00094     qDebug() « "Create room button clicked";
00095     CreateRoomDialog *dialog = new CreateRoomDialog(this);
00096     dialog->exec();
00097     delete dialog;
00098 }
00099
00100 void MainWindow::onJoinRoomClicked() {
00101     qDebug() « "Join room button clicked";
00102     JoinRoomDialog *dialog = new JoinRoomDialog(this);
00103     dialog->exec();
00104     delete dialog;
00105 }
00106
00107 void MainWindow::onRoomCreated(int roomId) {
00108     qDebug() « "Room created with ID:" « roomId;
00109     currentRoomId = roomId;
00110     // Присоединение теперь в createroomdialog.cpp
00111 }
00112
00113 void MainWindow::onJoinedRoom() {
00114     Client& client = Client::getInstance();
00115     int roomId = client.getRoomId();
00116     qDebug() « "Joined room with ID:" « roomId;
00117     currentRoomId = roomId;
00118
00119     // Закрываем предыдущее окно чата, если оно есть
00120     for (QWidget* widget : QApplication::topLevelWidgets()) {
00121         if (widget != this && widget->inherits("ChatWindow")) {
00122             widget->close();
00123             widget->deleteLater();
00124         }
00125     }
00126
00127     ChatWindow *chatWindow = new ChatWindow(roomId, &client, nullptr);
00128     chatWindow->setAttribute(Qt::WA_DeleteOnClose);
```

```
00129     qDebug() « "ChatWindow created for room ID:" « roomId;
00130     this->hide();
00131 }
00132
00133 void MainWindow::onListRoomsClicked() {
00134     ListRoomsDialog *dialog = new ListRoomsDialog(this);
00135     connect(dialog, &ListRoomsDialog::accepted, this, [this]() {
00136         // После успешного входа в комнату
00137         onJoinedRoom();
00138     });
00139     dialog->exec();
00140     delete dialog;
00141 }
```

5.29 Файл E:/ParuGramm/my_proj/client/mainwindow.h

Заголовочный файл для главного окна мессенджера.

#include <QMainWindow>
#include <QPushButton>
#include "client.h"
Граф включаемых заголовочных файлов для mainwindow.h:



Классы

- class MainWindow

  Класс для управления главным окном мессенджера.

5.29.1 Подробное описание

Заголовочный файл для главного окна мессенджера.

См. определение в файле mainwindow.h

5.30 mainwindow.h

См. документацию.

```
00001 #ifndef MAINWINDOW_H
00002 #define MAINWINDOW_H
00003
00004 #include <QMainWindow>
00005 #include <QPushButton>
00006 #include "client.h"
00007
00012
00017 class MainWindow : public QMainWindow {
00018     Q_OBJECT
00019
00020 public:
00025     explicit MainWindow(QWidget *parent = nullptr);
00026
00030     ~MainWindow();
00031
00032 private slots:
00036     void onCreateRoomClicked();
00037
00041     void onJoinRoomClicked();
00042
00047     void onRoomCreated(int roomId);
00048
00052     void onJoinedRoom();
00053
00057     void onListRoomsClicked();
00058
00059 private:
00060     QPushButton *createRoomButton;
00061     QPushButton *joinRoomButton;
00062     QPushButton *ListRoomsButton;
00063     Client& client;
00064     int currentRoomId;
00065 };
00066
00067 #endif // MAINWINDOW_H
```

Предметный указатель