ParuGramm_server_and_database

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 Database

Класс для управления SQLite базой данных чат-мессенджера.

#include <database.h>

Открытые члены

- QSqlDatabase & getDb ()
  Возвращает объект базы данных для выполнения запросов.

Открытые статические члены

- static Database & instance ()
  Получает единственный экземпляр базы данных (Singleton).

4.1.1 Подробное описание

Класс для управления SQLite базой данных чат-мессенджера.

См. определение в файле database.h строка 12

4.1.2 Методы

4.1.2.1 getDb()

QSqlDatabase & Database::getDb ()

Возвращает объект базы данных для выполнения запросов.

Возвращает объект базы данных.

Возвращает

Ссылка на QSqlDatabase.

См. определение в файле database.cpp строка 43

4.1.2.2 instance()

Database & Database::instance () [static]

Получает единственный экземпляр базы данных (Singleton).

Получает единственный экземпляр базы данных.

Возвращает

Ссылка на экземпляр Database.

См. определение в файле database.cpp строка 9

Объявления и описания членов классов находятся в файлах:

- E:/ParuGramm/my_proj/SUBD/database.h
- E:/ParuGramm/my_proj/SUBD/database.cpp

4.2 Класс DatabaseServer

Основной класс сервера чат-мессенджера, управляющий клиентскими подключениями и комнатами.

#include <databaseserver.h>

Открытые члены

- void start (int port)

  Запускает сервер на указанном порту.
- void stop ()

  Останавливает сервер и закрывает все соединения.
- ~DatabaseServer ()

  Деструктор, освобождающий ресурсы сервера.

Открытые статические члены

- static DatabaseServer & getInstance ()

  Получает единственный экземпляр класса (Singleton).

Друзья

- class SingletonDestroyer

4.2.1 Подробное описание

Основной класс сервера чат-мессенджера, управляющий клиентскими подключениями и комнатами.

См. определение в файле databaseserver.h строка 41

4.2.2 Конструктор(ы)

4.2.2.1 ~DatabaseServer()

DatabaseServer::~DatabaseServer ()

Деструктор, освобождающий ресурсы сервера.

Деструктор DatabaseServer. Останавливает сервер и освобождает ресурсы.

См. определение в файле databaseserver.cpp строка 47

4.2.3 Методы

4.2.3.1 getInstance()

DatabaseServer & DatabaseServer::getInstance () [static]

Получает единственный экземпляр класса (Singleton).

Получает единственный экземпляр DatabaseServer.

Возвращает

Ссылка на экземпляр DatabaseServer.

См. определение в файле databaseserver.cpp строка 29

4.2.3.2 start()

void DatabaseServer::start (
int port)

Запускает сервер на указанном порту.

Аргументы

port	Порт для прослушивания подключений.

См. определение в файле databaseserver.cpp строка 71

4.2.3.3 stop()

void DatabaseServer::stop ()

Останавливает сервер и закрывает все соединения.

См. определение в файле databaseserver.cpp строка 115

4.2.4 Друзья класса и относящимся к классу обозначения

4.2.4.1 SingletonDestroyer

friend class SingletonDestroyer [friend]

См. определение в файле databaseserver.h строка 67

Объявления и описания членов классов находятся в файлах:

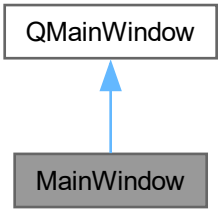
- E:/ParuGramm/my_proj/SUBD/databaseserver.h
- E:/ParuGramm/my_proj/SUBD/databaseserver.cpp

4.3 Класс MainWindow

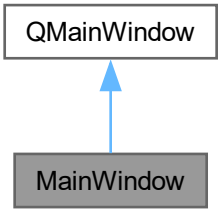
Основное окно приложения для взаимодействия с базой данных.

#include <mainwindow.h>

Граф наследования:MainWindow:



Граф связей класса MainWindow:



Открытые члены

- MainWindow (QWidget *parent=nullptr)
  Конструктор MainWindow.

4.3.1 Подробное описание

Основное окно приложения для взаимодействия с базой данных.

См. определение в файле mainwindow.h строка 13

4.3.2 Конструктор(ы)

4.3.2.1 MainWindow()

```
MainWindow::MainWindow (
        QWidget * parent = nullptr)
```

Конструктор MainWindow.

Конструктор MainWindow. Инициализирует интерфейс и подключает сигналы.

Аргументы

parent	Родительский виджет.

См. определение в файле mainwindow.cpp строка 15

Объявления и описания членов классов находятся в файлах:

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

4.4 Класс SingletonDestroyer

Вспомогательный класс для корректного уничтожения одиночки DatabaseServer.

#include <databaseserver.h>

Открытые члены

- SingletonDestroyer ()=default
  Конструктор по умолчанию.
- ~SingletonDestroyer ()
  Деструктор, освобождающий ресурсы DatabaseServer.
- void initialize (DatabaseServer *p)
  Инициализирует указатель на экземпляр DatabaseServer.

4.4.1 Подробное описание

Вспомогательный класс для корректного уничтожения одиночки DatabaseServer.

См. определение в файле databaseserver.h строка 15

4.4.2 Конструктор(ы)

4.4.2.1 SingletonDestroyer()

SingletonDestroyer::SingletonDestroyer () [default]

Конструктор по умолчанию.

4.4.2.2 ~SingletonDestroyer()

SingletonDestroyer::~SingletonDestroyer ()

Деструктор, освобождающий ресурсы DatabaseServer.

Деструктор SingletonDestroyer. Освобождает ресурсы экземпляра DatabaseServer.

См. определение в файле databaseserver.cpp строка 55

4.4.3 Методы

4.4.3.1 initialize()

void SingletonDestroyer::initialize (
DatabaseServer * p)

Инициализирует указатель на экземпляр DatabaseServer.

Инициализирует SingletonDestroyer указателем на DatabaseServer.

Аргументы

p	Указатель на экземпляр DatabaseServer.

См. определение в файле databaseserver.cpp строка 63

Объявления и описания членов классов находятся в файлах:

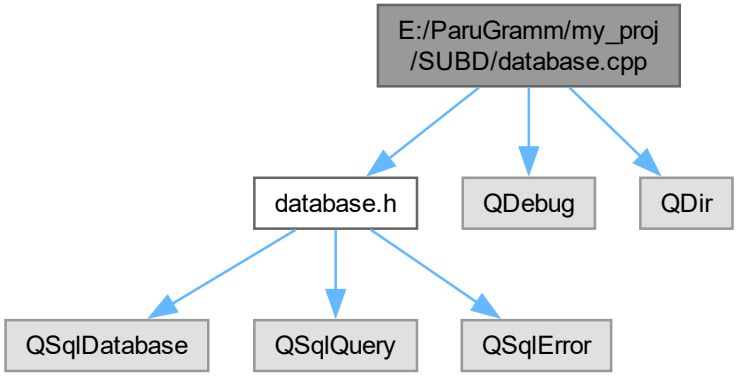
- E:/ParuGramm/my_proj/SUBD/databaseserver.h
- E:/ParuGramm/my_proj/SUBD/databaseserver.cpp

Глава 5

Файлы

5.1 Файл E:/ParuGramm/my_proj/SUBD/database.cpp

#include "database.h"
#include <QDebug>
#include <QDir>
Граф включаемых заголовочных файлов для database.cpp:



5.2 database.cpp

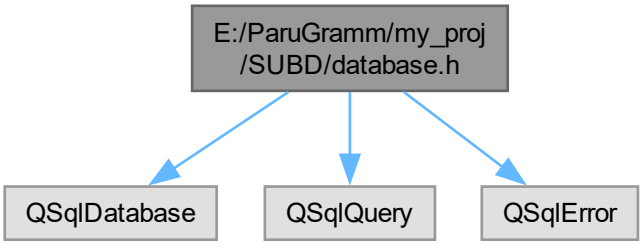
См. документацию.

```
00001 #include "database.h"
00002 #include <QDebug>
00003 #include <QDir>
00004
00009 Database& Database::instance() {
00010     static Database db;
00011     return db;
00012 }
00013
00018 Database::Database() {
```

```
00019     db = QSqlDatabase::addDatabase("QSQLITE", "chatDB");
00020     QString dbPath = QDir::currentPath() + "/chat.db";
00021     qDebug() « "Database path:" « dbPath;
00022     db.setDatabaseName(dbPath);
00023     if (!db.open()) {
00024         qDebug() « "Database error:" « db.lastError().text();
00025     } else {
00026         qDebug() « "Database initialized successfully";
00027     }
00028     createTables();
00029 }
00030
00035 Database::~Database() {
00036     db.close();
00037 }
00038
00043 QSqlDatabase& Database::getDb() {
00044     return db;
00045 }
00046
00050 void Database::createTables() {
00051     QSqlQuery query(db);
00052     if (query.exec("CREATE TABLE IF NOT EXISTS rooms ("
00053                    "id INTEGER PRIMARY KEY AUTOINCREMENT, "
00054                    "name TEXT NOT NULL, "
00055                    "password TEXT)")) {
00056         qDebug() « "Created table 'rooms'";
00057     } else {
00058         qDebug() « "Error creating table 'rooms':" « query.lastError().text();
00059     }
00060
00061     if (query.exec("CREATE TABLE IF NOT EXISTS messages ("
00062                    "id INTEGER PRIMARY KEY AUTOINCREMENT, "
00063                    "room_id INTEGER, "
00064                    "sender_id INTEGER, "
00065                    "message TEXT, "
00066                    "file_path TEXT, "
00067                    "timestamp DATETIME DEFAULT CURRENT_TIMESTAMP)")) {
00068         qDebug() « "Created table 'messages'";
00069     } else {
00070         qDebug() « "Error creating table 'messages':" « query.lastError().text();
00071     }
00072 }
```

5.3 Файл E:/ParuGramm/my_proj/SUBD/database.h

#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>
Граф включаемых заголовочных файлов для database.h:



Классы

- class Database

  Класс для управления SQLite базой данных чат-мессенджера.

5.4 database.h

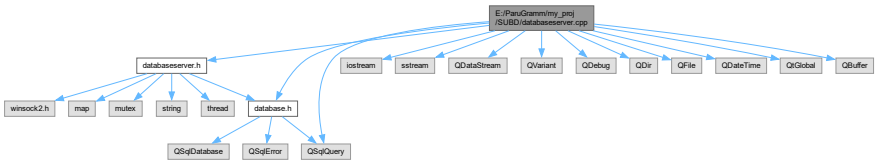
См. документацию.

```
00001 #ifndef DATABASE_H
00002 #define DATABASE_H
00003
00004 #include <QSqlDatabase>
00005 #include <QSqlQuery>
00006 #include <QSqlError>
00007
00012 class Database {
00013 public:
00018     static Database& instance();
00019
00024     QSqlDatabase& getDb();
00025
00026 private:
00030     Database();
00031
00035     ~Database();
00036
00037     Database(const Database&) = delete;
00038     Database& operator=(const Database&) = delete;
00039
00043     void createTables();
00044
00045     QSqlDatabase db;
00046 };
00047
00048 #endif // DATABASE_H
```

5.5 Файл E:/ParuGramm/my_proj/SUBD/databaseserver.cpp

```
#include "databaseserver.h"
#include "database.h"
#include <iostream>
#include <sstream>
#include <QSqlQuery>
#include <QDataStream>
#include <QVariant>
#include <QDebug>
#include <QDir>
#include <QFile>
#include <QDateTime>
#include <QtGlobal>
#include <QBuffer>
```

Граф включаемых заголовочных файлов для databaseserver.cpp:



5.6 databaseserver.cpp

См. документацию.

```
00001 #include "databaseserver.h"
00002 #include "database.h"
```

```
00003 #include <iostream>
00004 #include <sstream>
00005 #include <QSqlQuery>
00006 #include <QDataStream>
00007 #include <QVariant>
00008 #include <QDebug>
00009 #include <QDir>
00010 #include <QFile>
00011 #include <QDateTime>
00012 #include <QtGlobal>
00013 #include <QBuffer>
00014
00018 DatabaseServer* DatabaseServer::p_instance = nullptr;
00019
00023 SingletonDestroyer DatabaseServer::destroyer;
00024
00029 DatabaseServer& DatabaseServer::getInstance() {
00030     if (!p_instance) {
00031         p_instance = new DatabaseServer();
00032         destroyer.initialize(p_instance);
00033     }
00034     return *p_instance;
00035 }
00036
00041 DatabaseServer::DatabaseServer() : running_(false), serverSocket_(INVALID_SOCKET), nextClientId_(1) {}
00042
00047 DatabaseServer::~DatabaseServer() {
00048     stop();
00049 }
00050
00055 SingletonDestroyer::~SingletonDestroyer() {
00056     delete p_instance;
00057 }
00058
00063 void SingletonDestroyer::initialize(DatabaseServer* p) {
00064     p_instance = p;
00065 }
00066
00071 void DatabaseServer::start(int port) {
00072     qDebug() << "Server starting on port" << port;
00073     std::lock_guard<std::mutex> lock(mutex_);
00074     if (running_) return;
00075
00076     WSADATA wsaData;
00077     if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
00078         std::cerr << "WSAStartup failed.\n";
00079         return;
00080     }
00081
00082     serverSocket_ = socket(AF_INET, SOCK_STREAM, 0);
00083     if (serverSocket_ == INVALID_SOCKET) {
00084         std::cerr << "Socket creation failed.\n";
00085         WSACleanup();
00086         return;
00087     }
00088
00089     sockaddr_in serverAddr;
00090     serverAddr.sin_family = AF_INET;
00091     serverAddr.sin_addr.s_addr = INADDR_ANY;
00092     serverAddr.sin_port = htons(port);
00093
00094     if (bind(serverSocket_, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
00095         std::cerr << "Bind failed.\n";
00096         closesocket(serverSocket_);
00097         WSACleanup();
00098         return;
00099     }
00100
00101     if (listen(serverSocket_, SOMAXCONN) == SOCKET_ERROR) {
00102         std::cerr << "Listen failed.\n";
00103         closesocket(serverSocket_);
00104         WSACleanup();
00105         return;
00106     }
00107
00108     running_ = true;
00109     std::thread(&DatabaseServer::handleConnections, this).detach();
00110 }
00111
00115 void DatabaseServer::stop() {
00116     std::lock_guard<std::mutex> lock(mutex_);
00117     if (!running_) return;
00118
00119     running_ = false;
00120     closesocket(serverSocket_);
00121     WSACleanup();
00122 }
```

```
00123
00127 void DatabaseServer::handleConnections() {
00128     while (running_) {
00129         SOCKET clientSocket = accept(serverSocket_, nullptr, nullptr);
00130         if (clientSocket == INVALID_SOCKET) {
00131             std::cerr « "Accept failed.\n";
00132             continue;
00133         }
00134         std::lock_guard<std::mutex> lock(mutex_);
00135         clients_[nextClientId_] = clientSocket;
00136         std::string response = "SUCCESS:Client ID:" + std::to_string(nextClientId_) + "\r\n";
00137         send(clientSocket, response.c_str(), response.size(), 0);
00138         qDebug() « "Assigned client ID:" « nextClientId_ « "to socket";
00139         std::thread(&DatabaseServer::handleClient, this, clientSocket).detach();
00140         nextClientId_++;
00141     }
00142 }
00143
00148 void DatabaseServer::handleClient(SOCKET clientSocket) {
00149     int clientId = -1;
00150     {
00151         std::lock_guard<std::mutex> lock(mutex_);
00152         for (auto& [id, socket] : clients_) {
00153             if (socket == clientSocket) {
00154                 clientId = id;
00155                 break;
00156             }
00157         }
00158     }
00159
00160     std::string readBuffer;
00161     char tempBuffer[65536];
00162     int bytesReceived;
00163     while ((bytesReceived = recv(clientSocket, tempBuffer, sizeof(tempBuffer), 0)) > 0) {
00164         readBuffer.append(tempBuffer, bytesReceived);
00165         qDebug() « "Client" « clientId « "received" « bytesReceived « "bytes, buffer size:" « readBuffer.size();
00166
00167         while (!readBuffer.empty()) {
00168             if (readBuffer.size() >= sizeof(uint32_t)) {
00169                 QDataStream ds(QByteArray::fromRawData(readBuffer.data(), readBuffer.size()));
00170                 ds.setVersion(QDataStream::Qt_5_15);
00171                 ds.setByteOrder(QDataStream::LittleEndian);
00172                 uint32_t magic;
00173                 ds » magic;
00174                 if (magic == 0xFA57F11E) {
00175                     qDebug() « "Detected file packet for client" « clientId;
00176                     if (processFile(clientId, clientSocket, readBuffer)) {
00177                         continue; // Файл обработан, продолжаем
00178                     } else {
00179                         break; // Ждём больше данных
00180                     }
00181                 } else {
00182                     qDebug() « "No file packet, magic:" « QString::number(magic, 16);
00183                 }
00184             }
00185
00186             size_t newlinePos = readBuffer.find('\n');
00187             if (newlinePos == std::string::npos) {
00188                 if (readBuffer.size() > 10 * 1024 * 1024) { // Ограничение 10 МБ
00189                     qDebug() « "Buffer too large without newline, clearing";
00190                     readBuffer.clear();
00191                     std::string response = "ERROR:Buffer overflow\r\n";
00192                     send(clientSocket, response.c_str(), response.size(), 0);
00193                 }
00194                 break;
00195             }
00196
00197             std::string line = readBuffer.substr(0, newlinePos);
00198             if (!line.empty() && line.back() == '\r') {
00199                 line.pop_back();
00200             }
00201             if (!line.empty()) {
00202                 qDebug() « "Processing text command from client" « clientId « ":" « QString::fromStdString(line);
00203                 std::string response = processRequest(clientId, clientSocket, line) + "\r\n";
00204                 send(clientSocket, response.c_str(), response.size(), 0);
00205             }
00206             readBuffer.erase(0, newlinePos + 1);
00207         }
00208     }
00209
00210     {
00211         std::lock_guard<std::mutex> lock(mutex_);
00212         clients_.erase(clientId);
00213         clientRooms_.erase(clientId);
00214     }
00215     qDebug() « "Client" « clientId « "disconnected";
00216     shutdown(clientSocket, SD_BOTH);
```

```
00217     closesocket(clientSocket);
00218 }
00219
00227 std::string DatabaseServer::processRequest(int clientId, SOCKET clientSocket, const std::string& request) {
00228     qDebug() « "Processing request from client" « clientId « ":" « QString::fromStdString(request);
00229     std::stringstream ss(request);
00230     std::string command;
00231     std::getline(ss, command, ':');
00232
00233     if (command == "CREATE_ROOM") {
00234         std::string name, password;
00235         std::getline(ss, name, ':');
00236         std::getline(ss, password);
00237         return handleCreateRoom(name, password);
00238     } else if (command == "JOIN_ROOM") {
00239         std::string roomIdStr, password;
00240         std::getline(ss, roomIdStr, ':');
00241         std::getline(ss, password);
00242         int roomId;
00243         try {
00244             roomId = std::stoi(roomIdStr);
00245         } catch (...) {
00246             return "ERROR:Invalid room ID";
00247         }
00248         return handleJoinRoom(clientId, roomId, password);
00249     } else if (command == "MESSAGE") {
00250         std::string roomIdStr, message;
00251         std::getline(ss, roomIdStr, ':');
00252         std::getline(ss, message);
00253         int roomId;
00254         try {
00255             roomId = std::stoi(roomIdStr);
00256         } catch (...) {
00257             return "ERROR:Invalid room ID";
00258         }
00259         return handleMessage(clientId, roomId, message);
00260     } else if (command == "LIST_ROOMS") {
00261         return handleListRooms();
00262     } else if (command == "GET_MESSAGES") {
00263         std::string roomIdStr;
00264         std::getline(ss, roomIdStr);
00265         int roomId;
00266         try {
00267             roomId = std::stoi(roomIdStr);
00268         } catch (...) {
00269             return "ERROR:Invalid room ID";
00270         }
00271         QSqlQuery query(Database::instance().getDb());
00272         query.prepare("SELECT sender_id, message, file_path FROM messages WHERE room_id = ? ORDER BY
    timestamp");
00273         query.addBindValue(roomId);
00274         if (query.exec()) {
00275             std::string response = "MESSAGES:" + std::to_string(roomId) + ":";
00276             bool first = true;
00277             while (query.next()) {
00278                 if (!first) response += ";";
00279                 int senderId = query.value(0).toInt();
00280                 std::string message = query.value(1).toString().toStdString();
00281                 std::string filePath = query.value(2).toString().toStdString();
00282                 response += std::to_string(senderId) + "," + (filePath.empty() ? message : "File: " + filePath);
00283                 first = false;
00284             }
00285             return response.empty() ? "MESSAGES:" + std::to_string(roomId) + ":None" : response;
00286         } else {
00287             return "ERROR:Failed to fetch messages:" + query.lastError().text().toStdString();
00288         }
00289     } else if (command == "GET_USERS") {
00290         std::string roomIdStr;
00291         std::getline(ss, roomIdStr);
00292         int roomId;
00293         try {
00294             roomId = std::stoi(roomIdStr);
00295         } catch (...) {
00296             return "ERROR:Invalid room ID";
00297         }
00298         std::lock_guard<std::mutex> lock(mutex_);
00299         std::string response = "USER_LIST:";
00300         bool first = true;
00301         for (const auto& [id, socket] : clients_) {
00302             if (clientRooms_.find(id) != clientRooms_.end() && clientRooms_[id] == roomId) {
00303                 if (!first) response += ";";
00304                 response += std::to_string(id);
00305                 first = false;
00306             }
00307         }
00308         return response.empty() ? "USER_LIST:None" : response;
00309     } else if (command == "LEAVE_ROOM") {
```

```
00310         std::string roomIdStr;
00311         std::getline(ss, roomIdStr);
00312         int roomId;
00313         try {
00314             roomId = std::stoi(roomIdStr);
00315         } catch (...) {
00316             return "ERROR:Invalid room ID";
00317         }
00318         return handleLeaveRoom(clientId);
00319     }
00320     else {
00321         qDebug() « "Unknown command from client" « clientId « ":" « QString::fromStdString(command);
00322         return "ERROR:Unknown command";
00323     }
00324 }
00325
00332 std::string DatabaseServer::handleCreateRoom(const std::string& name, const std::string& password) {
00333     QSqlQuery query(Database::instance().getDb());
00334     query.prepare("INSERT INTO rooms (name, password) VALUES (:name, :password)");
00335     query.bindValue(":name", QVariant(QString::fromStdString(name)));
00336     query.bindValue(":password", QVariant(QString::fromStdString(password)));
00337     if (query.exec()) {
00338         int roomId = query.lastInsertId().toInt();
00339         return "SUCCESS:Room created:" + std::to_string(roomId);
00340     } else {
00341         return "ERROR:Failed to create room:" + query.lastError().text().toStdString();
00342     }
00343 }
00344
00352 std::string DatabaseServer::handleJoinRoom(int clientId, int roomId, const std::string& password) {
00353     QSqlQuery query(Database::instance().getDb());
00354     query.prepare("SELECT id, name, password FROM rooms WHERE id = ?");
00355     query.addBindValue(roomId);
00356
00357     if (!query.exec() || !query.next()) {
00358         return "ERROR:Room not found";
00359     }
00360
00361     QString dbPassword = query.value("password").toString();
00362     if (!dbPassword.isEmpty() && dbPassword != QString::fromStdString(password)) {
00363         return "ERROR:Invalid password";
00364     }
00365
00366     {
00367         std::lock_guard<std::mutex> lock(mutex_);
00368         if (clientRooms_.count(clientId)) {
00369             int oldRoomId = clientRooms_[clientId];
00370             clientRooms_.erase(clientId);
00371             notifyUserLeft(oldRoomId, clientId);
00372         }
00373         clientRooms_[clientId] = roomId;
00374     }
00375
00376     // Отправляем сначала подтверждение входа
00377     std::string response = "SUCCESS:Joined room\r\n";
00378     send(clients_[clientId], response.c_str(), response.size(), 0);
00379
00380     // Затем отправляем список пользователей
00381     std::string userList;
00382     {
00383         std::lock_guard<std::mutex> lock(mutex_);
00384         bool first = true;
00385         for (const auto& [id, rId] : clientRooms_) {
00386             if (rId == roomId) {
00387                 if (!first) userList += ";";
00388                 userList += std::to_string(id);
00389                 first = false;
00390             }
00391         }
00392         if (first) userList = "None";
00393     }
00394
00395     std::string userListResponse = "USER_LIST:" + userList + "\r\n";
00396     send(clients_[clientId], userListResponse.c_str(), userListResponse.size(), 0);
00397
00398     notifyUserJoined(roomId, clientId);
00399     return ""; // Мы уже отправили ответы вручную
00400 }
00401
00409 std::string DatabaseServer::handleMessage(int clientId, int roomId, const std::string& message) {
00410     {
00411         std::lock_guard<std::mutex> lock(mutex_);
00412         if (clientRooms_.find(clientId) == clientRooms_.end() || clientRooms_[clientId] != roomId) {
00413             return "ERROR:Not in room";
00414         }
00415     }
00416
```

```
00417     QSqlQuery query(Database::instance().getDb());
00418     query.prepare("INSERT INTO messages (room_id, sender_id, message) VALUES (:roomId, :senderId, :message)");
00419     query.bindValue(":roomId", QVariant(roomId));
00420     query.bindValue(":senderId", QVariant(clientId));
00421     query.bindValue(":message", QVariant(QString::fromStdString(message)));
00422     if (query.exec()) {
00423         broadcastMessage(roomId, clientId, message);
00424         return "SUCCESS:Message sent";
00425     } else {
00426         return "ERROR:Failed to save message:" + query.lastError().text().toStdString();
00427     }
00428 }
00429
00434 std::string DatabaseServer::handleListRooms() {
00435     QSqlQuery query(Database::instance().getDb());
00436     // Получаем список всех комнат с их названиями и наличием пароля
00437     query.prepare("SELECT id, name, password FROM rooms");
00438     if (!query.exec()) {
00439         qDebug() << "Failed to fetch rooms:" << query.lastError().text();
00440         return "ERROR:Failed to fetch room list";
00441     }
00442
00443     // Собираем статистику по участникам в комнатах
00444     std::map<int, int> userCounts;
00445     {
00446         std::lock_guard<std::mutex> lock(mutex_);
00447         for (const auto& [clientId, roomId] : clientRooms_) {
00448             userCounts[roomId]++;
00449         }
00450     }
00451
00452     std::string result = "ROOM_LIST:";
00453     bool first = true;
00454     while (query.next()) {
00455         if (!first) result += ";";
00456
00457         int roomId = query.value("id").toInt();
00458         QString name = query.value("name").toString();
00459         QString password = query.value("password").toString();
00460         int participants = userCounts[roomId];
00461
00462         result += QString("%1:%2:%3:%4")
00463                       .arg(roomId)
00464                       .arg(name)
00465                       .arg(password.isEmpty() ? "No" : "Yes")
00466                       .arg(participants)
00467                       .toStdString();
00468
00469         first = false;
00470     }
00471     return result.empty() ? "ROOM_LIST:None" : result;
00472 }
00473
00480 void DatabaseServer::broadcastMessage(int roomId, int senderId, const std::string& message) {
00481     std::string response = "MESSAGE:" + std::to_string(roomId) + ":" + std::to_string(senderId) + ":" + message +
      "\r\n";
00482     std::lock_guard<std::mutex> lock(mutex_);
00483     for (auto& [id, socket] : clients_) {
00484         if (clientRooms_.find(id) != clientRooms_.end() && clientRooms_[id] == roomId) {
00485             send(socket, response.c_str(), response.size(), 0);
00486         }
00487     }
00488 }
00489
00497 bool DatabaseServer::processFile(int clientId, SOCKET clientSocket, std::string& buffer) {
00498     if (buffer.size() < 3 * sizeof(int32_t)) {
00499         qDebug() << "Not enough data for file header, need" << (3 * sizeof(int32_t)) << ", have" << buffer.size();
00500         return false;
00501     }
00502
00503     QDataStream ds(QByteArray::fromRawData(buffer.data(), buffer.size()));
00504     ds.setVersion(QDataStream::Qt_5_15);
00505     ds.setByteOrder(QDataStream::LittleEndian);
00506
00507     int32_t magic, nameSize, dataSize;
00508     ds >> magic >> nameSize >> dataSize;
00509
00510     if (magic != 0xFA57F11E) {
00511         qDebug() << "Invalid magic number:" << QString::number(magic, 16);
00512         buffer.clear();
00513         std::string response = "ERROR:Invalid file packet\r\n";
00514         send(clientSocket, response.c_str(), response.size(), 0);
00515         return true;
00516     }
00517
00518     if (nameSize <= 0 || nameSize > 1024 || dataSize < 0 || dataSize > 10 * 1024 * 1024) {
00519         qDebug() << "Invalid file packet: nameSize=" << nameSize << ", dataSize=" << dataSize;
```

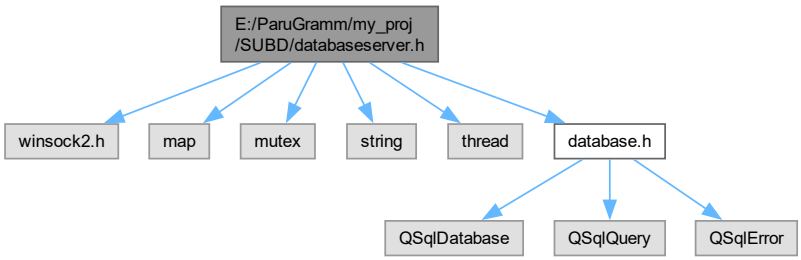
```
00520         buffer.clear();
00521         std::string response = "ERROR:Invalid file packet\r\n";
00522         send(clientSocket, response.c_str(), response.size(), 0);
00523         return true;
00524     }
00525 
00526     size_t totalSize = sizeof(int32_t) * 3 + nameSize + dataSize;
00527     if (buffer.size() < totalSize) {
00528         qDebug() << "Incomplete file data for client" << clientId << ", need" << totalSize << ", have" << buffer.size();
00529         return false; // Ждём больше данных
00530     }
00531 
00532     QByteArray nameData = QByteArray(buffer.data() + 12, nameSize);
00533     QByteArray fileData = QByteArray(buffer.data() + 12 + nameSize, dataSize);
00534     qDebug() << "nameData (hex):" << nameData.toHex();
00535     qDebug() << "fileData size:" << fileData.size();
00536 
00537     QString fileName = QString::fromUtf8(nameData);
00538     qDebug() << "Parsed fileName:" << fileName;
00539     if (fileName.isEmpty()) {
00540         qDebug() << "Invalid file name (empty after UTF-8 conversion)";
00541         buffer.erase(0, totalSize);
00542         std::string response = "ERROR:Invalid file name\r\n";
00543         send(clientSocket, response.c_str(), response.size(), 0);
00544         return true;
00545     }
00546 
00547     qDebug() << "Processing file from client" << clientId << ":" << fileName << ", size:" << dataSize;
00548 
00549     int roomId = -1;
00550     {
00551         std::lock_guard<std::mutex> lock(mutex_);
00552         if (clientRooms_.find(clientId) != clientRooms_.end()) {
00553             roomId = clientRooms_[clientId];
00554         }
00555     }
00556 
00557     if (roomId != -1) {
00558         saveFile(roomId, clientId, fileName, fileData);
00559         std::string fileBuffer = buffer.substr(0, totalSize);
00560         broadcastFile(roomId, clientId, fileBuffer);
00561         std::string response = "SUCCESS:File received\r\n";
00562         qDebug() << "Sent response bytes:" << send(clientSocket, response.c_str(), response.size(), 0);
00563     } else {
00564         qDebug() << "Client" << clientId << "not in any room, ignoring file";
00565         std::string response = "ERROR:Not in room\r\n";
00566         send(clientSocket, response.c_str(), response.size(), 0);
00567     }
00568 
00569     buffer.erase(0, totalSize);
00570     qDebug() << "File processed, remaining buffer size:" << buffer.size();
00571     return true;
00572 }
00573 
00581 void DatabaseServer::saveFile(int roomId, int senderId, const QString& fileName, const QByteArray& fileData) {
00582     QDir().mkpath("uploads");
00583     QString timestamp = QDateTime::currentDateTime().toString("yyyyMMdd_hhmmss_zzz");
00584     QString uniqueFileName = QString("uploads/%1_%2_%3").arg(senderId).arg(timestamp).arg(fileName);
00585 
00586     QFile file(uniqueFileName);
00587     if (file.open(QIODevice::WriteOnly)) {
00588         file.write(fileData);
00589         file.close();
00590         qDebug() << "File saved:" << uniqueFileName;
00591 
00592         QSqlQuery query(Database::instance().getDb());
00593         query.prepare("INSERT INTO messages (room_id, sender_id, message, file_path) "
00594                       "VALUES (:roomId, :senderId, :message, :filePath)");
00595         query.bindValue(":roomId", roomId);
00596         query.bindValue(":senderId", senderId);
00597         query.bindValue(":message", QString("File: %1").arg(fileName));
00598         query.bindValue(":filePath", uniqueFileName);
00599         if (!query.exec()) {
00600             qDebug() << "Error saving file metadata:" << query.lastError().text();
00601         } else {
00602             qDebug() << "File metadata saved for" << fileName;
00603         }
00604     } else {
00605         qDebug() << "Failed to save file:" << uniqueFileName;
00606     }
00607 }
00608 
00615 void DatabaseServer::broadcastFile(int roomId, int senderId, const std::string& buffer) {
00616     QBuffer newBuffer;
00617     newBuffer.open(QIODevice::WriteOnly);
00618     QDataStream ds(&newBuffer);
00619     ds.setVersion(QDataStream::Qt_5_15);
```

```
00620     ds.setByteOrder(QDataStream::LittleEndian);
00621
00622     QByteArray originalData = QByteArray::fromRawData(buffer.data(), buffer.size());
00623     QDataStream origDs(originalData);
00624     origDs.setVersion(QDataStream::Qt_5_15);
00625     origDs.setByteOrder(QDataStream::LittleEndian);
00626
00627     int32_t magic, nameSize, dataSize;
00628     origDs » magic » nameSize » dataSize;
00629
00630     ds « magic « senderId « nameSize « dataSize;
00631     newBuffer.write(originalData.mid(12));
00632
00633     QByteArray packet = newBuffer.data();
00634     std::lock_guard<std::mutex> lock(mutex_);
00635     for (auto& [id, socket] : clients_) {
00636         if (clientRooms_.find(id) != clientRooms_.end() && clientRooms_[id] == roomId && id != senderId) {
00637             int totalSent = 0;
00638             while (totalSent < packet.size()) {
00639                 int sent = send(socket, packet.data() + totalSent, packet.size() - totalSent, 0);
00640                 if (sent == SOCKET_ERROR) {
00641                     qDebug() « "Broadcast file failed for client" « id « "with error:" « WSAGetLastError();
00642                     break;
00643                 }
00644                 totalSent += sent;
00645             }
00646             qDebug() « "Broadcasted file to client" « id « "from sender" « senderId « ", sent:" « totalSent;
00647         }
00648     }
00649 }
00650
00656 std::string DatabaseServer::handleLeaveRoom(int clientId) {
00657     int roomId = -1;
00658     {
00659         std::lock_guard<std::mutex> lock(mutex_);
00660         if (clientRooms_.count(clientId)) {
00661             roomId = clientRooms_[clientId];
00662             clientRooms_.erase(clientId);
00663         }
00664     }
00665
00666     if (roomId != -1) {
00667         notifyUserLeft(roomId, clientId);
00668         return "SUCCESS:Left room";
00669     }
00670     return "ERROR:Not in any room";
00671 }
00672
00678 void DatabaseServer::notifyUserJoined(int roomId, int userId) {
00679     std::string response = "USER_JOINED:" + std::to_string(roomId) + ":" + std::to_string(userId) + "\r\n";
00680     std::lock_guard<std::mutex> lock(mutex_);
00681     for (auto& [id, socket] : clients_) {
00682         if (clientRooms_.find(id) != clientRooms_.end() && clientRooms_[id] == roomId) {
00683             send(socket, response.c_str(), response.size(), 0);
00684         }
00685     }
00686 }
00687
00693 void DatabaseServer::notifyUserLeft(int roomId, int userId) {
00694     std::string response = "USER_LEFT:" + std::to_string(roomId) + ":" + std::to_string(userId) + "\r\n";
00695     std::lock_guard<std::mutex> lock(mutex_);
00696     for (auto& [id, socket] : clients_) {
00697         if (clientRooms_.find(id) != clientRooms_.end() && clientRooms_[id] == roomId) {
00698             send(socket, response.c_str(), response.size(), 0);
00699         }
00700     }
00701 }
00702
```

5.7 Файл E:/ParuGramm/my_proj/SUBD/databaseserver.h

```
#include <winsock2.h>
#include <map>
#include <mutex>
#include <string>
#include <thread>
#include "database.h"
```

Граф включаемых заголовочных файлов для databaseserver.h:



Классы

- class SingletonDestroyer

  Вспомогательный класс для корректного уничтожения одиночки DatabaseServer.
- class DatabaseServer

  Основной класс сервера чат-мессенджера, управляющий клиентскими подключениями и комнатами.

5.8 databaseserver.h

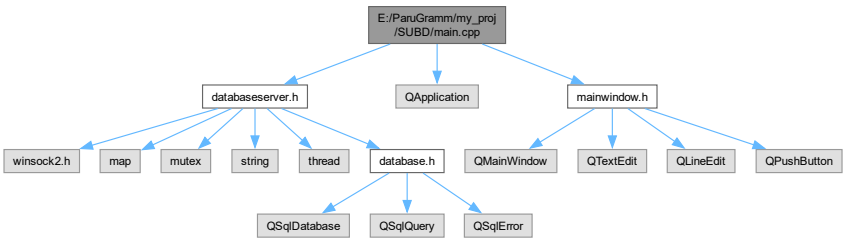
См. документацию.

```
00001 #ifndef DATABASESERVER_H
00002 #define DATABASESERVER_H
00003
00004 #include <winsock2.h>
00005 #include <map>
00006 #include <mutex>
00007 #include <string>
00008 #include <thread>
00009 #include "database.h"
00010
00015 class SingletonDestroyer {
00016 public:
00020     SingletonDestroyer() = default;
00021
00025     ~SingletonDestroyer();
00026
00031     void initialize(DatabaseServer* p);
00032
00033 private:
00034     DatabaseServer* p_instance = nullptr;
00035 };
00036
00041 class DatabaseServer {
00042 public:
00047     static DatabaseServer& getInstance();
00048
00053     void start(int port);
00054
00058     void stop();
00059
00063     ~DatabaseServer();
00064
00065 private:
00066     DatabaseServer();
00067     friend class SingletonDestroyer;
00068
00069     static DatabaseServer* p_instance;
00070     static SingletonDestroyer destroyer;
00071
00072     bool running_;
```

```
00073     SOCKET serverSocket_;
00074     std::map<int, SOCKET> clients_;
00075     std::map<int, int> clientRooms_;
00076     std::mutex mutex_;
00077     int nextClientId_;
00078     std::map<int, int> roomUserCounts_;
00079     std::mutex roomCountsMutex_;
00080
00084     void handleConnections();
00085
00090     void handleClient(SOCKET clientSocket);
00091
00099     std::string processRequest(int clientId, SOCKET clientSocket, const std::string& request);
00100
00107     std::string handleCreateRoom(const std::string& name, const std::string& password);
00108
00116     std::string handleJoinRoom(int clientId, int roomId, const std::string& password);
00117
00125     std::string handleMessage(int clientId, int roomId, const std::string& message);
00126
00131     std::string handleListRooms();
00132
00139     void broadcastMessage(int roomId, int senderId, const std::string& message);
00140
00148     bool processFile(int clientId, SOCKET clientSocket, std::string& buffer);
00149
00157     void saveFile(int roomId, int senderId, const QString& fileName, const QByteArray& fileData);
00158
00165     void broadcastFile(int roomId, int senderId, const std::string& buffer);
00166
00172     void notifyUserJoined(int roomId, int userId);
00173
00179     void notifyUserLeft(int roomId, int userId);
00180
00186     std::string handleLeaveRoom(int clientId);
00187 };
00188
00189 #endif // DATABASESERVER_H
```

5.9 Файл E:/ParuGramm/my_proj/SUBD/main.cpp

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



Функции

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

  Точка входа в приложение. Запускает сервер и графический интерфейс.

5.9.1 Функции

5.9.1.1 main()

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

Точка входа в приложение. Запускает сервер и графический интерфейс.

Аргументы

argc	Количество аргументов командной строки.
argv	Массив аргументов командной строки.

Возвращает

Код завершения приложения.

См. определение в файле main.cpp строка 12

5.10 main.cpp

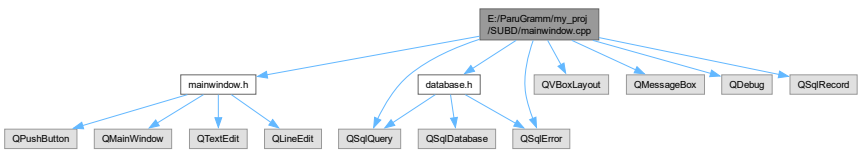
См. документацию.

```
00001 #include "databaseserver.h"
00002 #include <QApplication>
00003 #include "mainwindow.h"
00004
00012 int main(int argc, char *argv[]) {
00013     QApplication a(argc, argv);
00014     DatabaseServer::getInstance().start(12346);
00015     MainWindow w;
00016     w.show();
00017     return a.exec();
00018 }
```

5.11 Файл E:/ParuGramm/my_proj/SUBD/mainwindow.cpp

```
#include "mainwindow.h"
#include "database.h"
#include <QVBoxLayout>
#include <QMessageBox>
#include <QDebug>
#include <QSqlQuery>
#include <QSqlError>
#include <QSqlRecord>
```

Граф включаемых заголовочных файлов для mainwindow.cpp:



5.12 mainwindow.cpp

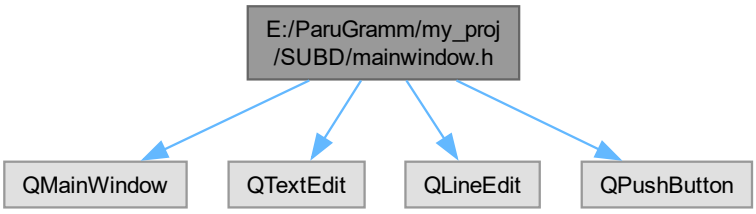
См. документацию.

```
00001 #include "mainwindow.h"
00002 #include "database.h"
00003 #include <QVBoxLayout>
00004 #include <QMessageBox>
00005 #include <QDebug>
00006 #include <QSqlQuery>
00007 #include <QSqlError>
00008 #include <QSqlRecord>
00009
00015 MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
00016     // Инициализация интерфейса
00017     QWidget *centralWidget = new QWidget(this);
00018     QVBoxLayout *layout = new QVBoxLayout(centralWidget);
00019
00020     output = new QTextEdit(this);
00021     output->setReadOnly(true);
00022
00023     input = new QLineEdit(this);
00024     input->setPlaceholderText("Введите SQL-запрос...");
00025
00026     btnExecute = new QPushButton("Выполнить", this);
00027     btnMessages = new QPushButton("Показать сообщения", this);
00028     btnRooms = new QPushButton("Показать комнаты", this);
00029     btnClear = new QPushButton("Очистить историю", this);  // Новая кнопка
00030
00031     layout->addWidget(output);
00032     layout->addWidget(input);
00033     layout->addWidget(btnExecute);
00034     layout->addWidget(btnMessages);
00035     layout->addWidget(btnRooms);
00036     layout->addWidget(btnClear);  // Добавляем кнопку в layout
00037
00038     setCentralWidget(centralWidget);
00039     resize(600, 400);
00040
00041     // Соединение сигналов
00042     connect(btnExecute, &QPushButton::clicked, this, &MainWindow::executeQuery);
00043     connect(input, &QLineEdit::returnPressed, this, &MainWindow::executeQuery);
00044     connect(btnMessages, &QPushButton::clicked, this, &MainWindow::showMessages);
00045     connect(btnRooms, &QPushButton::clicked, this, &MainWindow::showRooms);
00046     connect(btnClear, &QPushButton::clicked, this, &MainWindow::clearHistory);  // Новый коннект
00047
00048     // Вывод информации о базе
00049     output->append("База данных инициализирована");
00050 }
00051
00055 void MainWindow::executeQuery() {
00056     QString sql = input->text().trimmed();
00057     if (sql.isEmpty()) return;
00058
00059     QSqlQuery query(Database::instance().getDb());
00060     if (query.exec(sql)) {
00061         output->append("\n> " + sql);
00062
00063         if (query.isSelect()) {
00064             // Вывод результатов SELECT
00065             QString result;
00066             while (query.next()) {
00067                 for (int i = 0; i < query.record().count(); i++) {
00068                     result += query.value(i).toString() + "\t";
00069                 }
00070                 result += "\n";
00071             }
00072             output->append(result.isEmpty() ? "Нет данных" : result);
00073         } else {
00074             output->append("Выполнено успешно. Затронуто строк: " +
00075                            QString::number(query.numRowsAffected()));
00076         }
00077     } else {
00078         output->append("\nОШИБКА: " + query.lastError().text());
00079     }
00080
00081     input->clear();
00082 }
00083
00087 void MainWindow::showMessages() {
00088     QSqlQuery query(Database::instance().getDb());
00089     if (query.exec("SELECT * FROM messages")) {
00090         output->append("\n> SELECT * FROM messages");
00091         QString result;
00092         while (query.next()) {
00093             for (int i = 0; i < query.record().count(); i++) {
```

```
00094             result += query.value(i).toString() + "\t";
00095         }
00096         result += "\n";
00097     }
00098     output->append(result.isEmpty() ? "Нет данных" : result);
00099 } else {
00100     output->append("\nОШИБКА: " + query.lastError().text());
00101 }
00102 }
00103
00107 void MainWindow::showRooms() {
00108     QSqlQuery query(Database::instance().getDb());
00109     if (query.exec("SELECT * FROM rooms")) {
00110         output->append("\n> SELECT * FROM rooms");
00111         QString result;
00112         while (query.next()) {
00113             for (int i = 0; i < query.record().count(); i++) {
00114                 result += query.value(i).toString() + "\t";
00115             }
00116             result += "\n";
00117         }
00118         output->append(result.isEmpty() ? "Нет данных" : result);
00119     } else {
00120         output->append("\nОШИБКА: " + query.lastError().text());
00121     }
00122 }
00123
00127 void MainWindow::clearHistory() {
00128     output->clear();  // Очищаем текстовое поле
00129     output->append("База данных инициализирована");  // Восстанавливаем начальное сообщение
00130 }
```

5.13 Файл E:/ParuGramm/my_proj/SUBD/mainwindow.h

#include <QMainWindow>
#include <QTextEdit>
#include <QLineEdit>
#include <QPushButton>
Граф включаемых заголовочных файлов для mainwindow.h:



Классы

- class MainWindow

  Основное окно приложения для взаимодействия с базой данных.

5.14 mainwindow.h

См. документацию.

```
00001 #ifndef MAINWINDOW_H
00002 #define MAINWINDOW_H
00003
00004 #include <QMainWindow>
00005 #include <QTextEdit>
00006 #include <QLineEdit>
00007 #include <QPushButton>
00008
00013 class MainWindow : public QMainWindow {
00014     Q_OBJECT
00015 public:
00020     MainWindow(QWidget *parent = nullptr);
00021
00022 private slots:
00026     void executeQuery();
00027
00031     void showMessages();
00032
00036     void showRooms();
00037
00041     void clearHistory();
00042
00043 private:
00044     QTextEdit *output;
00045     QLineEdit *input;
00046     QPushButton *btnExecute;
00047     QPushButton *btnMessages;
00048     QPushButton *btnRooms;
00049     QPushButton *btnClear;
00050 };
00051
00052 #endif // MAINWINDOW_H
```

Предметный указатель