Server for Attestation System

Создано системой Doxygen 1.13.2

Глава 1

Документация на TCP-сервер экзаменационной системы

Данный проект представляет собой серверную часть платформы для проведения онлайн-экзаменов. Сервер реализован с использованием библиотеки Qt и взаимодействует с клиентами по протоколу TCP. Все данные хранятся в базе данных PostgreSQL, а бизнес-логика обработки команд сосредоточена в модульных классах.

Сервер поддерживает следующие функции:

- регистрация и авторизация пользователей;
- получение списка доступных экзаменов и их вопросов;
- приём и сохранение результатов прохождения тестов;
- просмотр профиля пользователя;
- изменение пароля;
- предоставление статистики об успешно пройденных экзаменах.

1.1 Основные компоненты:

- MyTcpServer — основной TCP-сервер, обрабатывающий подключения клиентов, принимающий команды и направляющий их в обработку.
- Functions — модуль бизнес-логики, реализующий обработку команд клиента, таких как:
  - LOGIN / REGISTER;
  - GET_EXAMS, GET_QUESTIONS;
  - SAVE_RESULTS, GET_STATISTICS;
  - GET_PROFILE, UPDATE_PROFILE, CHANGE_PASSWORD.
- Database — класс-одиночка (Singleton), реализующий подключение к базе данных PostgreSQL и предоставляющий интерфейс для выполнения SQL-запросов.
- DatabaseDestroyer — вспомогательный класс, отвечающий за корректное удаление экземпляра Database при завершении работы приложения (паттерн RAII).

1.2 Архитектура

Проект построен по принципам модульности и однозначного разделения ответственности:

- Сетевое взаимодействие — MyTcpServer;
- Логика обработки — Functions;
- Работа с базой — Database.

1.3 Хранилище данных

Сервер подключается к базе данных PostgreSQL с помощью драйвера QPSQL. Подключение настраивается автоматически при старте и закрывается в деструкторе. Все запросы выполняются через QSqlQuery.

1.4 Документация

Данная документация сгенерирована с помощью Doxygen, включает полное описание классов, методов, параметров и взаимодействий.

Глава 2

Иерархический список классов

2.1 Иерархия классов

Иерархия классов.

Глава 3

Алфавитный указатель структур данных

3.1 Структуры данных

Структуры данных с их кратким описанием.

Глава 4

Список файлов

4.1 Файлы

Полный список файлов.

Глава 5

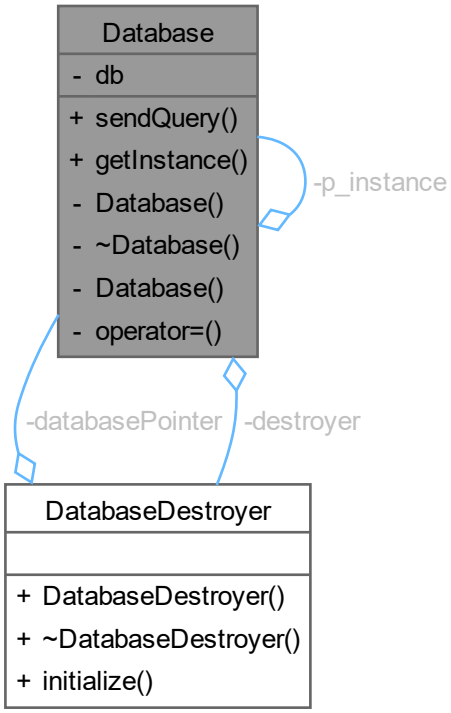
Структуры данных

5.1 Класс Database

Класс реализует шаблон Singleton для подключения к базе данных PostgreSQL.

#include <database.h>

Граф связей класса Database:



Открытые члены

- QString sendQuery (const QString &query)

  Выполняет произвольный SQL-запрос.

Открытые статические члены

- static Database * getInstance ()

  Возвращает единственный экземпляр класса Database (Singleton).

Закрытые члены

- Database ()

  Приватный конструктор.
- ~Database ()

  Приватный деструктор.
- Database (const Database &)=delete
- Database & operator= (const Database &)=delete

Закрытые данные

- QSqlDatabase db

  Объект подключения к базе данных.

Закрытые статические данные

- static Database * p_instance = nullptr

  Указатель на Singleton-экземпляр.
- static DatabaseDestroyer destroyer

  Объект для корректного удаления экземпляра Database.

Друзья

- class DatabaseDestroyer

  Предоставляет DatabaseDestroyer доступ к приватному деструктору.

5.1.1 Подробное описание

Класс реализует шаблон Singleton для подключения к базе данных PostgreSQL.

Класс предоставляет единый глобальный интерфейс для выполнения SQL-запросов и работы с базой данных. Подключение настраивается в конструкторе и автоматически закрывается при завершении программы. Уничтожение объекта реализовано через вспомогательный класс DatabaseDestroyer.

Используется в проекте TCP-сервера как backend-слой для взаимодействия с базой данных.

5.1.2 Конструктор(ы)

5.1.2.1 Database() [1/2]

Database::Database () [private]

Приватный конструктор.

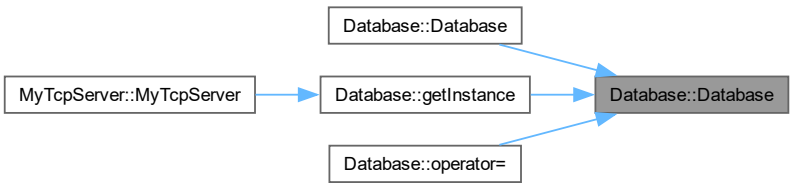
Конструктор класса Database.

Настраивает подключение к PostgreSQL с предопределёнными параметрами. Используется только внутри getInstance().

Настраивает параметры подключения к PostgreSQL:

- драйвер: QPSQL
- хост: localhost
- порт: 5432
- имя базы данных: exam_system
- имя пользователя: postgres
- пароль: 230405

При неудачном подключении выводит сообщение об ошибке в отладочный вывод. Граф вызова функции:



5.1.2.2 ~Database()

Database::~Database () [private]

Приватный деструктор.

Деструктор класса Database.

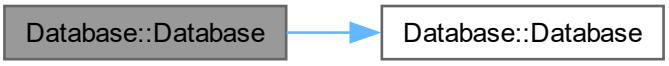
Закрывает соединение с базой данных. Вызывается автоматически при завершении программы.

При уничтожении объекта закрывает соединение с базой данных. Вызывается автоматически через DatabaseDestroyer.

5.1.2.3 Database() [2/2]

Database::Database (
const Database &) [private], [delete]

Граф вызовов:



5.1.3 Методы

5.1.3.1 getInstance()

Database * Database::getInstance () [static]

Возвращает единственный экземпляр класса Database (Singleton).

Возвращает указатель на единственный экземпляр класса Database.

При первом вызове создаёт объект и сохраняет его в статическую переменную. Повторные вызовы возвращают уже созданный экземпляр.

Возвращает

Указатель на объект Database.

Если экземпляр не был создан ранее, он инициализируется, а затем передаётся в объект DatabaseDestroyer для автоматического удаления при завершении программы.

Возвращает

Указатель на глобальный объект Database.

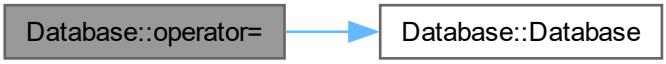
Граф вызовов:

Граф вызова функции:

5.1.3.2 operator=()

Database & Database::operator= (
const Database &) [private], [delete]

Граф вызовов:



5.1.3.3 sendQuery()

QString Database::sendQuery (
const QString & query)

Выполняет произвольный SQL-запрос.

Выполняет произвольный SQL-запрос и возвращает строку с результатом.

Выполняет запрос и возвращает сообщение об успехе или об ошибке. Подходит для административных или отладочных целей.

Аргументы

query	Строка SQL-запроса.

Возвращает

QString Сообщение о результате выполнения запроса.

Функция выполняет запрос, переданный как текстовая строка. Если выполнение не удалось, возвращается сообщение об ошибке.

Аргументы

query	Строка с SQL-запросом (например, SELECT * FROM table).

Возвращает

QString Содержит сообщение "Query executed successfully." или описание ошибки.

5.1.4 Друзья класса и относящимся к классу обозначения

5.1.4.1 DatabaseDestroyer

friend class DatabaseDestroyer [friend]

Предоставляет DatabaseDestroyer доступ к приватному деструктору.

5.1.5 Поля

5.1.5.1 db

QSqlDatabase Database::db [private]

Объект подключения к базе данных.

5.1.5.2 destroyer

DatabaseDestroyer Database::destroyer [static], [private]

Объект для корректного удаления экземпляра Database.

Инициализация объекта-уничтожителя, обеспечивающего корректное удаление экземпляра Database.

5.1.5.3 p_instance

Database * Database::p_instance = nullptr [static], [private]

Указатель на Singleton-экземпляр.

Инициализация указателя на Singleton-экземпляр базы данных.

Объявления и описания членов классов находятся в файлах:

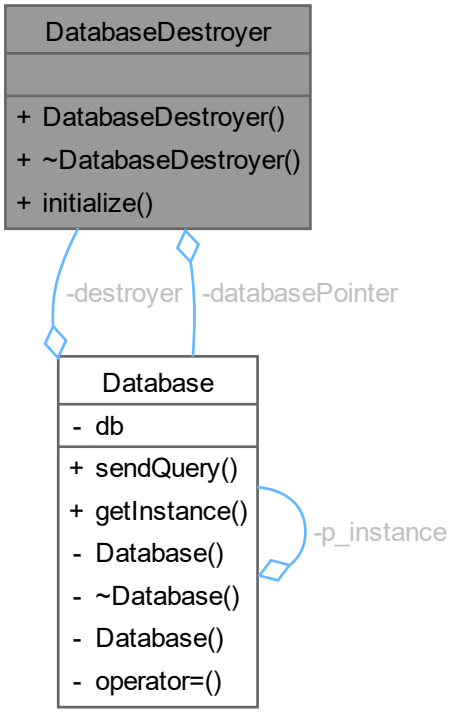
- echo_server/database.h
- echo_server/database.cpp

5.2 Класс DatabaseDestroyer

Класс DatabaseDestroyer управляет автоматическим удалением Singleton-экземпляра класса Database.

#include <databasedestroyer.h>

Граф связей класса DatabaseDestroyer:



Открытые члены

- DatabaseDestroyer ()

  Конструктор по умолчанию.
- ~DatabaseDestroyer ()

  Деструктор.
- void initialize (Database *p)

  Сохраняет указатель на экземпляр Singleton-базы данных.

Закрытые данные

- Database * databasePointer

  Указатель на экземпляр класса Database, подлежащий удалению.

5.2.1 Подробное описание

Класс DatabaseDestroyer управляет автоматическим удалением Singleton-экземпляра класса Database.

Используется как часть реализации паттерна Singleton.

В норме Singleton должен быть удалён в момент завершения работы приложения. Так как Database создаётся динамически, объект DatabaseDestroyer выступает в роли "умного указателя", который гарантирует вызов delete при завершении программы.

Этот подход позволяет обойти проблему «порядка уничтожения глобальных объектов» в C++.

5.2.2 Конструктор(ы)

5.2.2.1 DatabaseDestroyer()

DatabaseDestroyer::DatabaseDestroyer ()

Конструктор по умолчанию.

Инициализирует внутренний указатель как nullptr. Уничтожение экземпляра будет происходить только если initialize() был вызван ранее.

5.2.2.2 ~DatabaseDestroyer()

DatabaseDestroyer::~DatabaseDestroyer ()

Деструктор.

Деструктор класса DatabaseDestroyer.

При завершении программы автоматически вызывает delete для объекта Database, на который ссылается internal указатель databasePointer.

Отвечает за освобождение памяти, занятой экземпляром класса Database. Вызывается автоматически при завершении работы программы.

Если до этого был вызван метод initialize(), то указатель databasePointer будет указывать на выделенный объект Database и будет корректно освобождён.

5.2.3 Методы

5.2.3.1 initialize()

void DatabaseDestroyer::initialize (
Database * p)

Сохраняет указатель на экземпляр Singleton-базы данных.

Инициализирует уничтожитель указателем на объект базы данных.

Вызывается из Database::getInstance() для передачи ответственности за удаление.

Аргументы

p	Указатель на ранее созданный объект Database.

Данный метод вызывается из метода Database::getInstance() и сохраняет указатель на экземпляр Singleton, чтобы затем освободить его в деструкторе.

Аргументы

p	Указатель на объект класса Database.

5.2.4 Поля

5.2.4.1 databasePointer

Database* DatabaseDestroyer::databasePointer [private]

Указатель на экземпляр класса Database, подлежащий удалению.

Объявления и описания членов классов находятся в файлах:

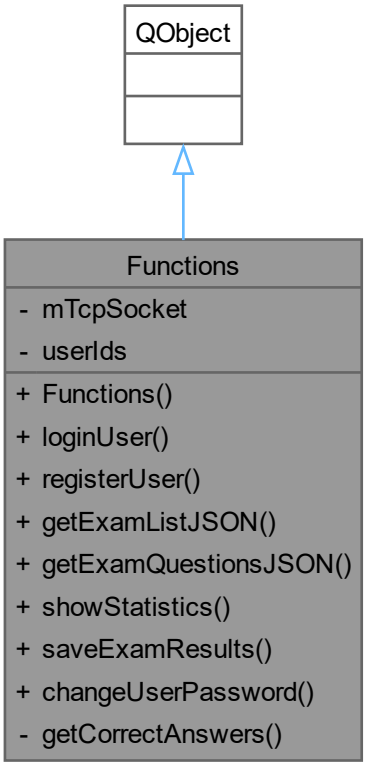
- echo_server/databasedestroyer.h
- echo_server/databasedestroyer.cpp

5.3 Класс Functions

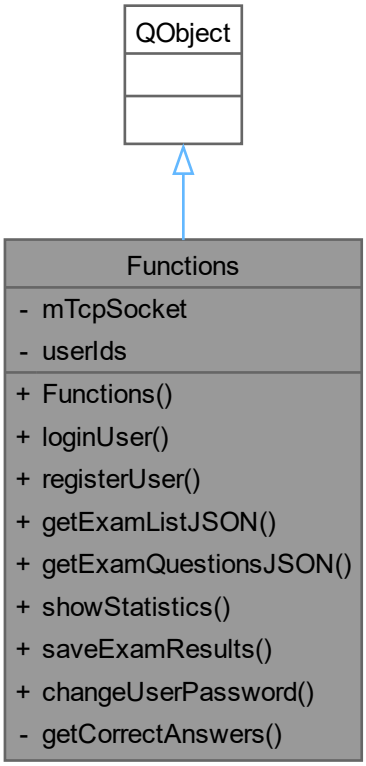
Класс реализует логику обработки клиентских команд: авторизация, регистрация, экзамены, статистика. Используется в связке с TCP-сервером. Взаимодействует с базой данных PostgreSQL.

#include <functions.h>

Граф наследования:Functions:



Граф связей класса Functions:



Открытые члены

- Functions (QTcpSocket *socket, QMap< QTcpSocket *, int > &userIds, QObject *parent=nullptr)

  Конструктор класса Functions.
- void loginUser (const QString &username, const QString &password)

  Выполняет вход пользователя.
- void registerUser (const QString &username, const QString &password)

  Регистрирует нового пользователя.
- QJsonArray getExamListJSON ()

  Получает список всех экзаменов.
- QJsonArray getExamQuestionsJSON (int examId)

  Получает список вопросов экзамена.
- void showStatistics ()

  Отправляет клиенту статистику прохождения экзаменов.
- void saveExamResults (int userId, int examId, int score, const QJsonArray &answers)

  Сохраняет результаты прохождения экзамена в базу данных.
- void changeUserPassword (int userId, const QString &oldPassword, const QString &newPassword)

  Изменяет пароль пользователя.

Закрытые члены

- QJsonArray getCorrectAnswers (int examId)

  Извлекает правильные ответы на вопросы определённого экзамена.

Закрытые данные

- QTcpSocket * mTcpSocket

  Указатель на текущий клиентский сокет.
- QMap< QTcpSocket *, int > & userIds

  Ссылка на глобальную карту: сокет <-> ID пользователя.

5.3.1 Подробное описание

Класс реализует логику обработки клиентских команд: авторизация, регистрация, экзамены, статистика. Используется в связке с TCP-сервером. Взаимодействует с базой данных PostgreSQL.

Класс Functions предоставляет реализацию бизнес-логики сервера.

Обрабатывает команды, полученные от клиентов через TCP-сокеты, включая:

- авторизацию и регистрацию пользователей,
- получение списка экзаменов и вопросов,
- сохранение результатов,
- просмотр статистики,
- изменение пароля.

Для взаимодействия с базой данных используется QSqlQuery и подключение PostgreSQL, а данные передаются в формате JSON.

5.3.2 Конструктор(ы)

5.3.2.1 Functions()

```
Functions::Functions (
            QTcpSocket * socket,
            QMap< QTcpSocket *, int > & userIds,
            QObject * parent = nullptr)    [explicit]
```

Конструктор класса Functions.

Инициализирует обработчик команд клиента, сохраняет сокет и карту соответствия сокет-ID.

Аргументы

socket	Указатель на клиентский сокет.
userIds	Ссылка на QMap, сопоставляющий сокеты и ID пользователей.
parent	Родительский объект QObject (по умолчанию nullptr).
socket	Указатель на сокет клиента.
userIds	Ссылка на отображение сокетов в ID пользователей.
parent	Родительский QObject (по умолчанию nullptr).

5.3.3 Методы

5.3.3.1 changeUserPassword()

void Functions::changeUserPassword (
int userId,
const QString & oldPassword,
const QString & newPassword)

Изменяет пароль пользователя.

Изменяет пароль пользователя, если старый пароль введён правильно.

Сравнивает хеш старого пароля с хранимым значением, и если совпадает — обновляет на новый.

Аргументы

userId	ID пользователя.
oldPassword	Старый пароль (открытый текст).
newPassword	Новый пароль (открытый текст).

Выполняется сравнение старого пароля с хранящимся хешем. В случае успеха — обновляется запись в БД.

Аргументы

userId	ID пользователя.
oldPassword	Старый пароль (в открытом виде).
newPassword	Новый пароль (в открытом виде).

Граф вызова функции:

5.3.3.2 getCorrectAnswers()

QJsonArray Functions::getCorrectAnswers (
int examId) [private]

Извлекает правильные ответы на вопросы определённого экзамена.

Возвращает массив правильных ответов на основе экзамена.

Используется при проверке результатов.

Аргументы

examId	Идентификатор экзамена.

Возвращает

QJsonArray Массив правильных ответов.

Аргументы

examId	ID экзамена.

Возвращает

QJsonArray Массив правильных ответов.

Граф вызова функции:

5.3.3.3 getExamListJSON()

QJsonArray Functions::getExamListJSON ()

Получает список всех экзаменов.

Возвращает список всех доступных экзаменов в формате JSON.

Извлекает данные из таблицы exams и возвращает в виде массива JSON.

Возвращает

QJsonArray Список экзаменов (id и название).
QJsonArray Список экзаменов с полями id и name.

Граф вызова функции:

5.3.3.4 getExamQuestionsJSON()

QJsonArray Functions::getExamQuestionsJSON (
int examId)

Получает список вопросов экзамена.

Возвращает список вопросов экзамена.

Извлекает текст, правильные ответы и варианты выбора по ID экзамена.

Аргументы

examId	Идентификатор экзамена.

Возвращает

QJsonArray Массив вопросов в формате JSON.

Включает текст вопроса, массив правильных ответов и варианты выбора.

Аргументы

examId	Идентификатор экзамена.

Возвращает

QJsonArray Массив JSON-объектов с вопросами.

Граф вызова функции:

5.3.3.5 loginUser()

```
void Functions::loginUser (
                const QString & username,
                const QString & password)
```

Выполняет вход пользователя.

Пытается выполнить вход пользователя по логину и паролю.

Проверяет наличие пользователя в БД по имени и хешу пароля. В случае успеха добавляет ID в карту userIds и отправляет OK клиенту.

Аргументы

username	Имя пользователя.
password	Пароль в открытом виде.

Пароль хешируется с использованием SHA-256. Если пользователь найден — ID добавляется в user↩ Ids, клиенту отправляется "OK".

Аргументы

username	Имя пользователя.
password	Пароль в открытом виде.

Граф вызова функции:

5.3.3.6 registerUser()

void Functions::registerUser (
const QString & username,
const QString & password)

Регистрирует нового пользователя.

Хеширует пароль и записывает нового пользователя в базу данных.

Аргументы

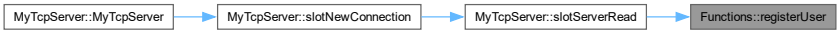
username	Имя пользователя.
password	Пароль в открытом виде.

Пароль хешируется и сохраняется. Если регистрация прошла успешно, ID пользователя сохраняется в userIds.

Аргументы

username	Имя пользователя.
password	Пароль в открытом виде.

Граф вызова функции:



5.3.3.7 saveExamResults()

```
void Functions::saveExamResults (
                int userId,
                int examId,
                int score,
                const QJsonArray & answers)
```

Сохраняет результаты прохождения экзамена в базу данных.

Сохраняет результаты экзамена в базу данных.

Сохраняет ID пользователя, экзамена, балл, ответы и дату прохождения. После сохранения отправляет клиенту JSON-ответ с правильными ответами.

Аргументы

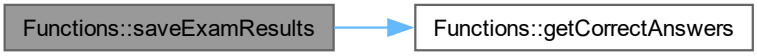
userId	ID пользователя.
examId	ID экзамена.
score	Баллы, набранные пользователем.
answers	Массив JSON с ответами пользователя.

Включает оценку, ответы пользователя, правильные ответы и дату прохождения. Отправляет клиенту JSON с полем correct_answers.

Аргументы

userId	ID пользователя.
examId	ID экзамена.
score	Количество баллов.
answers	Массив пользовательских ответов.

Граф вызовов:



Граф вызова функции:

5.3.3.8 showStatistics()

void Functions::showStatistics ()

Отправляет клиенту статистику прохождения экзаменов.

Отправляет клиенту статистику по всем пройденным экзаменам.

Включает оценки, дату, название экзамена и детали (ответы).

Включает названия экзаменов, оценки, дату прохождения, а также правильные и пользовательские ответы. Граф вызова функции:

5.3.4 Поля

5.3.4.1 mTcpSocket

QTcpSocket* Functions::mTcpSocket [private]

Указатель на текущий клиентский сокет.

5.3.4.2 userIds

QMap<QTcpSocket*, int>& Functions::userIds [private]

Ссылка на глобальную карту: сокет <-> ID пользователя.

Объявления и описания членов классов находятся в файлах:

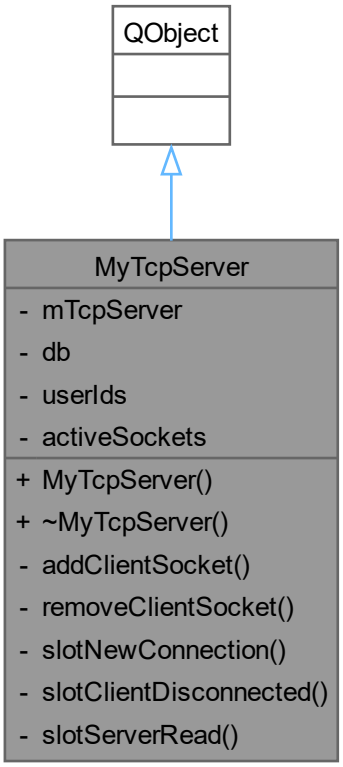
- echo_server/functions.h
- echo_server/functions.cpp

5.4 Класс MyTcpServer

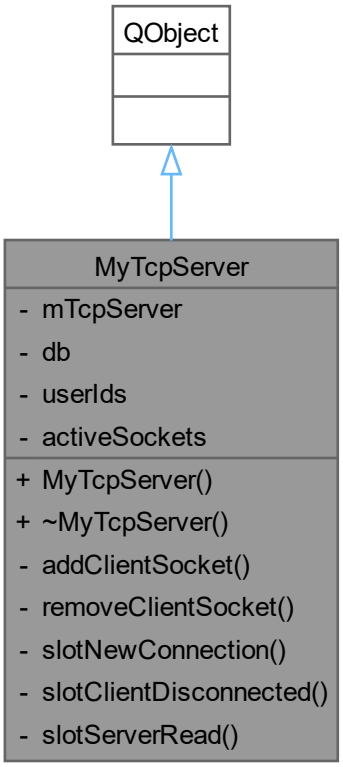
Класс реализует TCP-сервер для обработки клиентских соединений и команд.

#include <mytcpserver.h>

Граф наследования:MyTcpServer:



Граф связей класса MyTcpServer:



Открытые члены

- MyTcpServer (QObject *parent=nullptr)
  Конструктор класса MyTcpServer.
- ~MyTcpServer ()
  Деструктор класса.

Закрытые слоты

- void slotNewConnection ()
  Слот для обработки новых подключений клиентов.
- void slotClientDisconnected ()
  Слот для обработки отключения клиента.
- void slotServerRead ()
  Слот для обработки входящих данных от клиента.

Закрытые члены

- void addClientSocket (QTcpSocket *socket)
  Добавляет сокет клиента в список активных соединений.
- void removeClientSocket (QTcpSocket *socket)
  Удаляет сокет клиента из списка активных соединений.

Закрытые данные

- QTcpServer * mTcpServer

  Указатель на объект TCP-сервера, прослушивающего подключения.
- QSqlDatabase db

  Объект подключения к базе данных (не используется напрямую, оставлен на будущее).
- QMap< QTcpSocket *, int > userIds

  Ассоциативная карта: сокет клиента → ID пользователя.
- QVector< QTcpSocket * > activeSockets

  Список всех активных клиентских сокетов.

5.4.1 Подробное описание

Класс реализует TCP-сервер для обработки клиентских соединений и команд.

Класс MyTcpServer реализует TCP-сервер, обрабатывающий подключения клиентов и команды по сети.

Сервер принимает подключения от клиентов по TCP, читает команды в текстовом формате, и передаёт их на обработку классу Functions. В ответ клиенту отправляются данные или сообщения статуса.

Сервер работает на порту 33333 и поддерживает множество одновременных клиентов.

Сервер принимает команды в текстовом или JSON-формате от подключённых клиентов, передаёт их на обработку в класс Functions, управляет соединениями и поддерживает карту привязки сокетов к ID пользователей.

Сервер работает на порту 33333 и обслуживает команды: LOGIN, REGISTER, GET_↩ EXAMS, GET_QUESTIONS, SAVE_RESULTS, GET_PROFILE, UPDATE_PROFILE, CHANGE↩ _PASSWORD, GET_STATISTICS.

5.4.2 Конструктор(ы)

5.4.2.1 MyTcpServer()

```
MyTcpServer::MyTcpServer (
            QObject * parent = nullptr)   [explicit]
```

Конструктор класса MyTcpServer.

Конструктор MyTcpServer.

Инициализирует объект TCP-сервера и запускает прослушивание порта. Также инициализирует соединение с базой данных через Singleton Database.

Аргументы

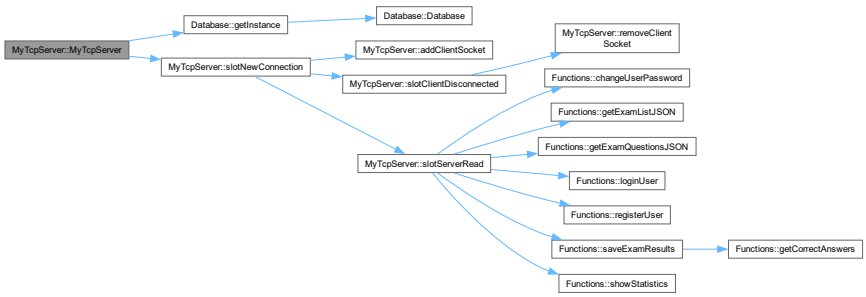
parent	Родительский объект QObject (по умолчанию nullptr).

Инициализирует подключение к базе данных, создаёт TCP-сервер и запускает прослушивание входящих соединений.

Аргументы

parent	Родительский объект Qt.

Граф вызовов:



5.4.2.2 ~MyTcpServer()

MyTcpServer::~MyTcpServer ()

Деструктор класса.

Деструктор. Закрывает TCP-сервер и освобождает ресурсы.

Закрывает TCP-сервер и очищает ресурсы.

5.4.3 Методы

5.4.3.1 addClientSocket()

void MyTcpServer::addClientSocket (
QTcpSocket * socket) [private]

Добавляет сокет клиента в список активных соединений.

Добавляет сокет клиента в вектор активных подключений.

Аргументы

socket	Указатель на клиентский сокет.
socket	Указатель на сокет клиента.

Граф вызова функции:

5.4.3.2 removeClientSocket()

void MyTcpServer::removeClientSocket (
QTcpSocket * socket) [private]

Удаляет сокет клиента из списка активных соединений.

Удаляет сокет клиента из вектора активных подключений.

Аргументы

socket	Указатель на клиентский сокет.
socket	Указатель на сокет клиента.

Граф вызова функции:

5.4.3.3 slotClientDisconnected

void MyTcpServer::slotClientDisconnected () [private], [slot]

Слот для обработки отключения клиента.

Удаляет клиентский сокет из списка активных, очищает карту userIds и удаляет сокет.

Удаляет сокет из списка активных и очищает карту идентификаторов пользователей. Граф вызовов:

Граф вызова функции:

5.4.3.4 slotNewConnection

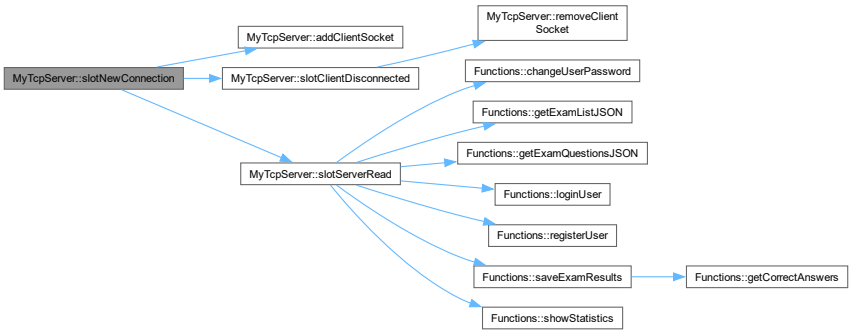
void MyTcpServer::slotNewConnection () [private], [slot]

Слот для обработки новых подключений клиентов.

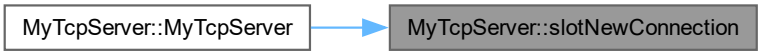
Слот для обработки нового подключения клиента.

При подключении нового клиента создаётся сокет, привязываются сигналы readyRead и disconnected. Сокет добавляется в список активных соединений.

Устанавливает соединения с сигналами клиента: readyRead и disconnected. Добавляет сокет в список активных подключений. Граф вызовов:



Граф вызова функции:



5.4.3.5 slotServerRead

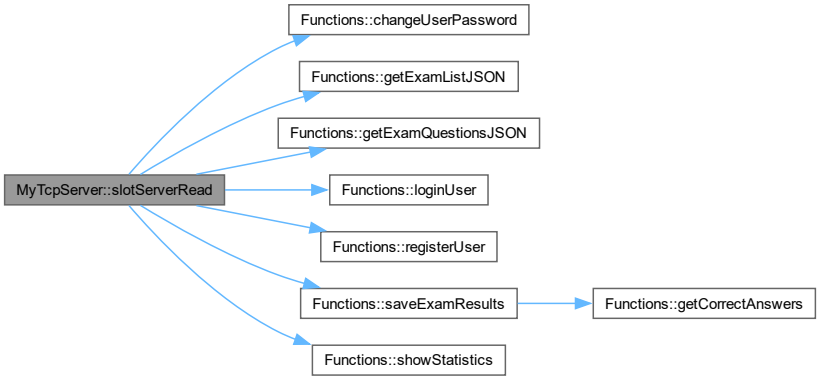
void MyTcpServer::slotServerRead () [private], [slot]

Слот для обработки входящих данных от клиента.

Слот для обработки входящих сообщений от клиента.

Читает команду и данные от клиента, парсит её, вызывает соответствующую функцию из класса Functions, и отправляет ответ клиенту.

Получает команду от клиента, парсит её и вызывает соответствующий метод класса Functions. Поддерживаются команды: LOGIN, REGISTER, GET_EXAMS, GET_QUESTIONS, SAVE_↩ RESULTS, GET_PROFILE, UPDATE_PROFILE, CHANGE_PASSWORD, GET_STATISTICS. Граф вызовов:



Граф вызова функции:

5.4.4 Поля

5.4.4.1 activeSockets

QVector<QTcpSocket*> MyTcpServer::activeSockets [private]

Список всех активных клиентских сокетов.

5.4.4.2 db

QSqlDatabase MyTcpServer::db [private]

Объект подключения к базе данных (не используется напрямую, оставлен на будущее).

5.4.4.3 mTcpServer

QTcpServer* MyTcpServer::mTcpServer [private]

Указатель на объект TCP-сервера, прослушивающего подключения.

5.4.4.4 userIds

QMap<QTcpSocket*, int> MyTcpServer::userIds [private]

Ассоциативная карта: сокет клиента → ID пользователя.

Объявления и описания членов классов находятся в файлах:

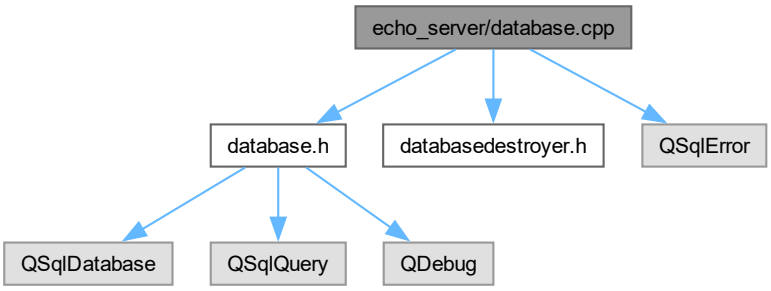
- echo_server/mytcpserver.h
- echo_server/mytcpserver.cpp

Глава 6

Файлы

6.1 Файл echo_server/database.cpp

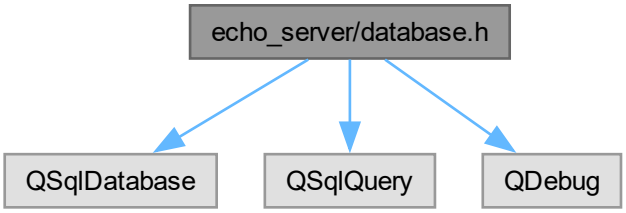
#include "database.h"
#include "databasedestroyer.h"
#include <QSqlError>
Граф включаемых заголовочных файлов для database.cpp:



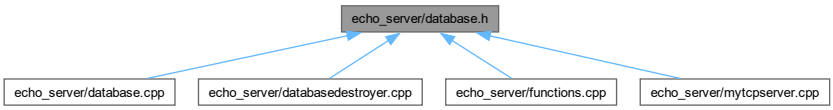
6.2 Файл echo_server/database.h

#include <QSqlDatabase>
#include <QSqlQuery>

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



Граф файлов, в которые включается этот файл:



Структуры данных

- class Database

  Класс реализует шаблон Singleton для подключения к базе данных PostgreSQL.

6.3 database.h

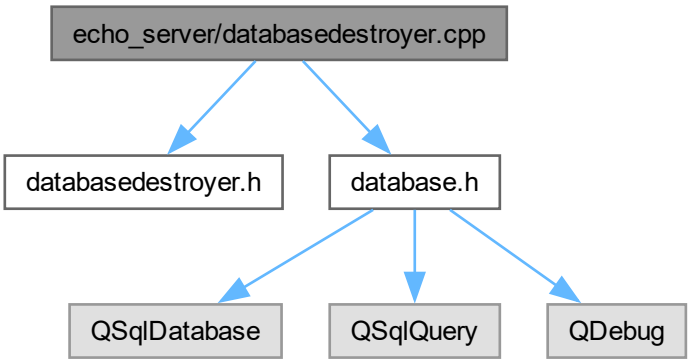
См. документацию.

```
00001 #ifndef DATABASE_H
00002 #define DATABASE_H
00003
00004 #include <QSqlDatabase>
00005 #include <QSqlQuery>
00006 #include <QDebug>
00007
00008 class DatabaseDestroyer;
00009
00021 class Database
00022 {
00023 public:
00032     static Database* getInstance();
00033
00043     QString sendQuery(const QString& query);
00044
00045 private:
00052     Database();
00053
00059     ~Database();
00060
00061     // Удаляем копирование и присваивание для соблюдения Singleton-паттерна.
00062     Database(const Database&) = delete;
00063     Database& operator=(const Database&) = delete;
00064
```

```
00065     QSqlDatabase db;
00066
00067     static Database* p_instance;
00068     static DatabaseDestroyer destroyer;
00069
00070     friend class DatabaseDestroyer;
00071 };
00072
00073 #endif // DATABASE_H
```

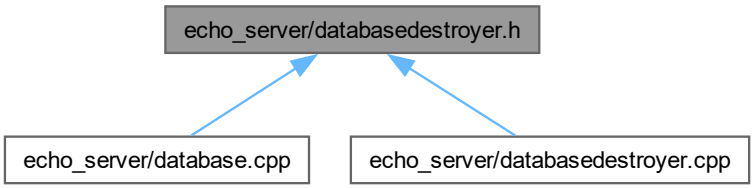
6.4 Файл echo_server/databasedestroyer.cpp

#include "databasedestroyer.h"
#include "database.h"
Граф включаемых заголовочных файлов для databasedestroyer.cpp:



6.5 Файл echo_server/databasedestroyer.h

Граф файлов, в которые включается этот файл:



Структуры данных

- class DatabaseDestroyer

  Класс DatabaseDestroyer управляет автоматическим удалением Singleton-экземпляра класса Database.

6.6 databasedestroyer.h

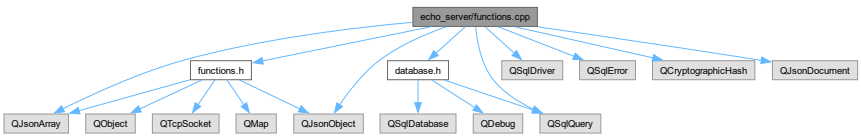
См. документацию.

```
00001 #ifndef DATABASEDESTROYER_H
00002 #define DATABASEDESTROYER_H
00003
00004 class Database;
00005
00018 class DatabaseDestroyer
00019 {
00020 public:
00027     DatabaseDestroyer();
00028
00035     ~DatabaseDestroyer();
00036
00044     void initialize(Database* p);
00045
00046 private:
00047     Database* databasePointer;
00048 };
00049
00050 #endif // DATABASEDESTROYER_H
```

6.7 Файл echo_server/functions.cpp

```
#include "functions.h"
#include "database.h"
#include <QSqlQuery>
#include <QSqlDriver>
#include <QSqlError>
#include <QCryptographicHash>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
```

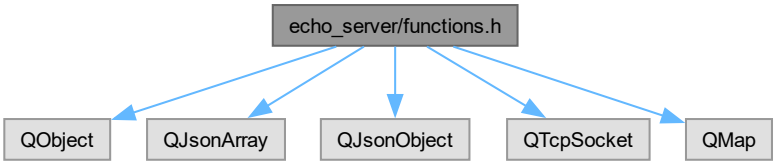
Граф включаемых заголовочных файлов для functions.cpp:



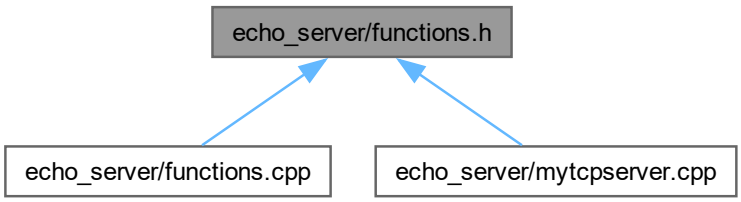
6.8 Файл echo_server/functions.h

```
#include <QObject>
#include <QJsonArray>
#include <QJsonObject>
#include <QTcpSocket>
```

#include <QMap>
Граф включаемых заголовочных файлов для functions.h:



Граф файлов, в которые включается этот файл:



Структуры данных

- class Functions

  Класс реализует логику обработки клиентских команд: авторизация, регистрация, экзамены, статистика. Используется в связке с TCP-сервером. Взаимодействует с базой данных PostgreSQL.

6.9 functions.h

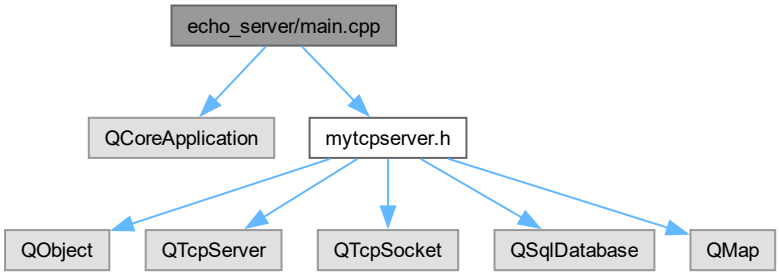
См. документацию.

```
00001 #ifndef FUNCTIONS_H
00002 #define FUNCTIONS_H
00003
00004 #include <QObject>
00005 #include <QJsonArray>
00006 #include <QJsonObject>
00007 #include <QTcpSocket>
00008 #include <QMap>
00009
00024 class Functions : public QObject
00025 {
00026     Q_OBJECT
00027
00028 public:
00038     explicit Functions(QTcpSocket *socket, QMap<QTcpSocket*, int> &userIds, QObject *parent = nullptr);
00039
00049     void loginUser(const QString &username, const QString &password);
00050
00059     void registerUser(const QString &username, const QString &password);
00060
```

```
00068     QJsonArray getExamListJSON();
00069
00078     QJsonArray getExamQuestionsJSON(int examId);
00079
00085     void showStatistics();
00086
00098     void saveExamResults(int userId, int examId, int score, const QJsonArray &answers);
00099
00109     void changeUserPassword(int userId, const QString &oldPassword, const QString &newPassword);
00110
00111 private:
00120     QJsonArray getCorrectAnswers(int examId);
00121
00122     QTcpSocket *mTcpSocket;
00123     QMap<QTcpSocket*, int> &userIds;
00124 };
00125
00126 #endif // FUNCTIONS_H
```

6.10 Файл echo_server/main.cpp

#include <QCoreApplication>
#include "mytcpserver.h"
Граф включаемых заголовочных файлов для main.cpp:



Функции

- int main (int argc, char *argv[])
  Точка входа в серверное приложение.

6.10.1 Функции

6.10.1.1 main()

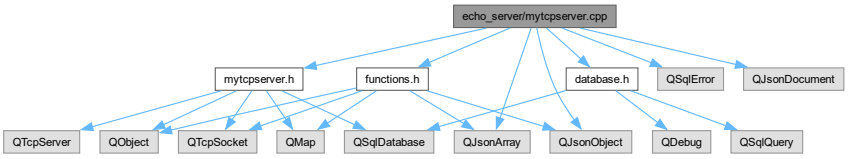
int main (
int argc,
char * argv[])

Точка входа в серверное приложение.

6.11 Файл echo_server/mytcpserver.cpp

```
#include "mytcpserver.h"
#include "functions.h"
#include "database.h"
#include <QSqlError>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
```

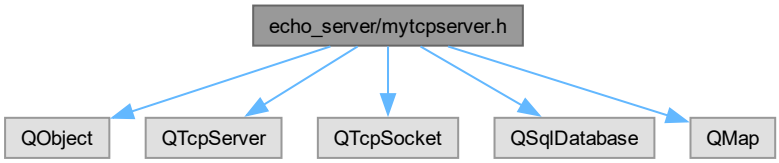
Граф включаемых заголовочных файлов для mytcpserver.cpp:



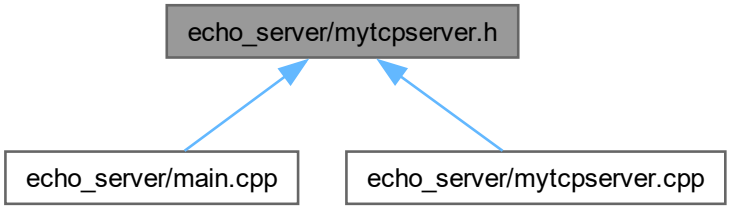
6.12 Файл echo_server/mytcpserver.h

```
#include <QObject>
#include <QTcpServer>
#include <QTcpSocket>
#include <QSqlDatabase>
#include <QMap>
```

Граф включаемых заголовочных файлов для mytcpserver.h:



Граф файлов, в которые включается этот файл:



Структуры данных

- class MyTcpServer

  Класс реализует TCP-сервер для обработки клиентских соединений и команд.

6.13 mytcpserver.h

См. документацию.

```
00001 #ifndef MYTCPSERVER_H
00002 #define MYTCPSERVER_H
00003
00004 #include <QObject>
00005 #include <QTcpServer>
00006 #include <QTcpSocket>
00007 #include <QSqlDatabase>
00008 #include <QMap>
00009
00020 class MyTcpServer : public QObject
00021 {
00022     Q_OBJECT
00023
00024 public:
00033     explicit MyTcpServer(QObject *parent = nullptr);
00034
00040     ~MyTcpServer();
00041
00042 private slots:
00049     void slotNewConnection();
00050
00056     void slotClientDisconnected();
00057
00064     void slotServerRead();
00065
00066 private:
00067     QTcpServer *mTcpServer;
00068     QSqlDatabase db;
00069     QMap<QTcpSocket*, int> userIds;
00070     QVector<QTcpSocket*> activeSockets;
00071
00077     void addClientSocket(QTcpSocket *socket);
00078
00084     void removeClientSocket(QTcpSocket *socket);
00085 };
00086
00087 #endif // MYTCPSERVER_H
```

6.14 Файл mainpage.md