Сервер для курсовой

Создано системой Doxygen 1.9.8

Глава 1

Курсовая работа по Технологиям и методам программирования (ТИМП)

1.0.1 Структура проекта

- server/ — директория с модулями сервера.
- Documentation/ — директория с модулями сервера с блоками документирования для DOXYGEN.
- README.md — описание проекта, сборки и запуска.

1.0.2 Клонирование репозитория

Чтобы клонировать проект, выполните команды:
```
git clone <URL-репозитория>
cd <папка с проектом>
```

1.0.3 Запуск сервера

Для запуска сервера перейдите в директорию server и используйте следующие команды:
```
make
./server -l log.txt -b user_db.txt [-p port]
```

1.0.4 Аргументы:

1.0.4.1 -l — путь к файлу лога.

1.0.4.2 -b — путь к базе данных пользователей.

1.0.4.3 -p — порт для подключения клиентов (по умолчанию: 33333).

1.0.5 Запуск модульных тестов

Для запуска модульных тестов перейдите в директорию server и выполните команду:
```
make test
```

1.0.6 Очистка проекта

Для удаления скомпилированных файлов выполните:
```
make clean
```

1.0.7 Структура базы данных пользователей

Файл базы данных user_db.txt содержит пары логин и пароль для каждого пользователя, разделённые пробелом. Пример:
```
user1 password1
user2 password2
```

1.0.8 Создание документации

Для создания документации перейдите в директорию Documentation и выполните:
```
make
cd DOXYGEN/latex
make
```

Файл refman.pdf - созданная документация.

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Глава 3

Список файлов

3.1 Файлы

Полный список документированных файлов.

Глава 4

Классы

4.1 Класс Calculator

Класс Calculator предоставляет функциональность для вычисления суммы квадратов элементов вектора и обработки векторов, полученных через сетевое соединение.

#include <Calculator.h>

Открытые члены

- uint16_t processVectors (int socket)
  Обрабатывает векторы, полученные от клиента через сокет.

Открытые статические члены

- static uint32_t calculateSumOfSquares (const std::vector< uint16_t > &vec)
  Вычисляет сумму квадратов элементов вектора.

4.1.1 Подробное описание

Класс Calculator предоставляет функциональность для вычисления суммы квадратов элементов вектора и обработки векторов, полученных через сетевое соединение.

Класс содержит два основных метода:

- processVectors: обрабатывает векторы, полученные от клиента через сокет.
- calculateSumOfSquares: вычисляет сумму квадратов элементов вектора.

Класс разработан для использования в серверном приложении, которое принимает данные от клиентов, обрабатывает их и возвращает результат.

4.1.2 Методы

4.1.2.1 calculateSumOfSquares()

uint32_t Calculator::calculateSumOfSquares (
const std::vector< uint16_t > & vec) [static]

Вычисляет сумму квадратов элементов вектора.

Функция принимает вектор 16-битных беззнаковых целых чисел (uint16_t) и вычисляет сумму квадратов его элементов. В функции реализована проверка на переполнение:

- Если в процессе вычисления произойдет переполнение 32-битного беззнакового целого (uint32_t), функция вернет 65535 (максимальное значение uint16_t).

Аргументы

vec	Вектор 16-битных беззнаковых целых чисел.

Возвращает

Сумму квадратов элементов вектора или 65535 в случае переполнения.

Заметки

Функция использует статическое приведение типов для предотвращения переполнения при промежуточных вычислениях.

Функция принимает вектор 16-битных беззнаковых целых чисел (uint16_t) и вычисляет сумму квадратов его элементов. В функции реализована проверка на переполнение:

- Если в процессе вычисления произойдет переполнение 32-битного беззнакового целого (uint32_t), функция вернет 65535 (максимальное значение uint16_t).

Аргументы

vec	Входной вектор, для которого необходимо рассчитать сумму квадратов.

Возвращает

Сумма квадратов элементов вектора. Возвращает 65535 в случае переполнения.

Заметки

Функция использует статическое приведение типов для предотвращения переполнения при промежуточных вычислениях.

Функция использует цикл for на основе диапазона (range-based for loop) для итерации по элементам вектора.

4.1.2.2 processVectors()

```
uint16_t Calculator::processVectors (
            int socket )
```

Обрабатывает векторы, полученные от клиента через сокет.

Обрабатывает данные векторов, полученные от клиента через сокет.

Функция принимает дескриптор сокета, через который происходит взаимодействие с клиентом. Сначала от клиента ожидается получение количества векторов (numberOfVectors) в сетевом порядке байт (big-endian). Затем для каждого вектора:

1. Получает размер вектора (также в сетевом порядке байт).
2. Получает данные вектора. Элементы вектора (uint16_t) также ожидаются в сетевом порядке байт.
3. Вычисляет сумму квадратов элементов вектора с помощью функции calculateSumOfSquares.
4. Отправляет результат обратно клиенту в сетевом порядке байт.

Аргументы

socket	Дескриптор сокета для взаимодействия с клиентом.

Возвращает

0 в случае успешного выполнения, -1 при ошибке получения данных или если recv вернул 0 (соединение закрыто клиентом).

Заметки

Функция использует функции ntohl и ntohs для преобразования порядка байт из сетевого в порядок байт хоста и htonl для обратного преобразования.

Функция не обрабатывает ошибки отправки данных (send).

Функция не закрывает сокет после завершения работы.

Функция принимает дескриптор сокета, через который происходит взаимодействие с клиентом. Сначала от клиента ожидается получение количества векторов (numberOfVectors) в сетевом порядке байт (big-endian). Затем для каждого вектора:

1. Получает размер вектора (также в сетевом порядке байт).
2. Получает данные вектора. Элементы вектора (uint16_t) также ожидаются в сетевом порядке байт.
3. Вычисляет сумму квадратов элементов вектора с помощью функции calculateSumOfSquares.
4. Отправляет результат обратно клиенту в сетевом порядке байт.

Аргументы

socket	Сокет, через который происходит обмен данными с клиентом.

Возвращает

0 в случае успешной обработки, -1 в случае ошибки при получении данных или если recv вернул 0 (соединение закрыто клиентом).

Заметки

Функция использует функции ntohl и ntohs для преобразования порядка байт из сетевого в порядок байт хоста и htonl для обратного преобразования.

Функция не обрабатывает ошибки отправки данных (send).

Функция не закрывает сокет после завершения работы.

Объявления и описания членов классов находятся в файлах:

- Calculator.h
- Calculator.cpp

4.2 Класс ClientCommunicate

Класс ClientCommunicate отвечает за взаимодействие с клиентом.

#include <ClientCommunicate.h>

Открытые члены

- void communicate (int socket, const std::string &userDbFileName, const std::string &logFileName)

  Осуществляет взаимодействие с клиентом.

Открытые статические члены

- static ParsedMessage parseMessage (const std::string &message)

  Разбирает сообщение от клиента на составные части.

4.2.1 Подробное описание

Класс ClientCommunicate отвечает за взаимодействие с клиентом.

Класс предоставляет методы для парсинга сообщений от клиента и для обработки запросов клиента, включая аутентификацию и обработку данных.

4.2.2 Методы

4.2.2.1 communicate()

```
void ClientCommunicate::communicate (
                int socket,
                const std::string & userDbFileName,
                const std::string & logFileName )
```

Осуществляет взаимодействие с клиентом.

Функция принимает дескриптор сокета, имя файла базы данных пользователей и имя файла логов.

Последовательность действий:

1. Получает сообщение от клиента через сокет.
2. Парсит полученное сообщение с помощью функции parseMessage.
3. Если парсинг прошел успешно (структура ParsedMessage не пуста):
   - Выполняет аутентификацию пользователя с помощью класса ConnectToBase.
   - В случае успешной аутентификации:
     - Отправляет клиенту сообщение "OK".
     - Логирует успешную аутентификацию с помощью Interface::logMessage.
     - Вызывает функцию processVectors класса Calculator для обработки векторов от клиента.
     - Если при обработке векторов произошла ошибка, логирует ошибку с помощью Interface::logError.
   - В случае неудачи при аутентификации:
     - Отправляет клиенту сообщение "ERR".
     - Логирует ошибку аутентификации с помощью Interface::logError.
4. Если парсинг сообщения не удался:
   - Логирует ошибку парсинга с помощью Interface::logError.
   - Отправляет клиенту сообщение "ERR".

Аргументы

socket	Дескриптор сокета для взаимодействия с клиентом.
userDbFileName	Имя файла базы данных пользователей.
logFileName	Имя файла логов.

Заметки

Функция не закрывает сокет после завершения работы.

Функция не обрабатывает ошибки отправки данных (send).

Функция предполагает, что файл базы данных пользователей и файл логов доступны для записи.

Функция получает сообщение от клиента, парсит его, аутентифицирует пользователя и, в случае успеха, обрабатывает данные от клиента.

Аргументы

socket	Сокет для взаимодействия с клиентом.
userDbFileName	Имя файла базы данных пользователей.
logFileName	Имя файла для записи логов.

Заметки

Функция использует классы ConnectToBase, Calculator и Interface для выполнения соответствующих задач.

Функция не обрабатывает ошибки отправки данных клиенту.

Функция не закрывает соединение с клиентом.

4.2.2.2 parseMessage()

ParsedMessage ClientCommunicate::parseMessage (
const std::string & message) [static]

Разбирает сообщение от клиента на составные части.

Парсит сообщение, полученное от клиента.

Функция принимает строку сообщения от клиента и разбивает ее на логин, соль и хеш. Предполагается, что сообщение имеет формат: <логин><соль><хеш>, где:

- длина соли фиксирована и равна 16 символам.
- длина хеша фиксирована и равна 40 символам.
- логин - это строка переменной длины, расположенная в начале сообщения.

Аргументы

message	Строка сообщения от клиента.

Возвращает

Структуру ParsedMessage, содержащую логин, соль и хеш. В случае ошибки парсинга (например, некорректный формат сообщения) возвращается пустая структура (все поля будут пустыми строками).

Заметки

Функция не проверяет корректность логина, соли и хеша (например, не проверяет, что хеш соответствует паролю и соли).

Функция использует фиксированные длины для соли и хеша. Изменение этих длин потребует изменения кода функции.

Функция извлекает из сообщения логин, соль и хеш пароля.

Аргументы

message	Сообщение от клиента в формате <логин><соль><хеш>. Длина соли - 16 символов, длина хеша - 40 символов.

Возвращает

Структура ParsedMessage, содержащая логин, соль и хеш. В случае ошибки парсинга возвращается пустая структура.

Заметки

Функция использует фиксированные длины для соли и хеша.

Функция не проверяет корректность формата сообщения.

Объявления и описания членов классов находятся в файлах:

- ClientCommunicate.h
- ClientCommunicate.cpp

4.3 Класс ConnectToBase

Класс ConnectToBase отвечает за взаимодействие с базой данных пользователей.

#include <ConnectToBase.h>

Открытые члены

- bool authenticateUser (const std::string &login, const std::string &salt, const std::string &clientHash, const std::string &dbFileName)

  Аутентифицирует пользователя по логину, соли, хешу и файлу базы данных.

Закрытые члены

- std::string hashPassword (const std::string &password, const std::string &salt)

  Вычисляет хеш пароля с использованием соли.
- bool compareHashes (const std::string &serverHash, const std::string &clientHash)

  Сравнивает два хеша на равенство.

4.3.1 Подробное описание

Класс ConnectToBase отвечает за взаимодействие с базой данных пользователей.

Класс предоставляет методы для аутентификации пользователей, хеширования паролей и сравнения хешей.

4.3.2 Методы

4.3.2.1 authenticateUser()

```
bool ConnectToBase::authenticateUser (
                const std::string & login,
                const std::string & salt,
                const std::string & clientHash,
                const std::string & dbFileName )
```

Аутентифицирует пользователя по логину, соли, хешу и файлу базы данных.

Аутентифицирует пользователя по логину, соли и хешу пароля.

Функция считывает данные пользователей из файла базы данных, который должен быть в формате: <логин> <пароль>\n Для каждой записи в файле:

1. Сравнивает логин из файла с предоставленным логином.
2. Если логины совпадают, вычисляет хеш пароля из файла с использованием предоставленной соли.
3. Сравнивает вычисленный хеш с предоставленным хешем.
4. Если хеши совпадают, аутентификация считается успешной.

Аргументы

login	Логин пользователя.
salt	Соль, используемая при хешировании пароля.
clientHash	Хеш пароля пользователя, полученный от клиента.
dbFileName	Имя файла базы данных пользователей.

Возвращает

true, если аутентификация прошла успешно, false в противном случае.

Заметки

Функция не обрабатывает ошибки чтения из файла.

Функция использует hashPassword для вычисления хеша пароля и compareHashes для сравнения хешей.

Функция предполагает, что файл базы данных существует и доступен для чтения.

Функция завершает работу после нахождения первой совпадающей записи.

Аргументы

login	Логин пользователя.
salt	Соль, используемая при хешировании.
clientHash	Хеш пароля, полученный от клиента.
dbFileName	Имя файла базы данных, содержащей логины и пароли пользователей.

Возвращает

true, если аутентификация прошла успешно, false в противном случае.

Заметки

Функция считывает файл базы данных построчно.

Функция использует hashPassword для вычисления хеша пароля и compareHashes для сравнения хешей.

4.3.2.2 compareHashes()

```
bool ConnectToBase::compareHashes (
            const std::string & serverHash,
            const std::string & clientHash )   [private]
```

Сравнивает два хеша на равенство.

Сравнивает два хеша, приведенных к нижнему регистру.

Функция приводит обе строки к нижнему регистру перед сравнением.

Аргументы

serverHash	Хеш, вычисленный на сервере.
clientHash	Хеш, полученный от клиента.

Возвращает

true, если хеши совпадают, false в противном случае.

Заметки

Функция использует std::transform для приведения строк к нижнему регистру.

Аргументы

serverHash	Хеш, сгенерированный на сервере.
clientHash	Хеш, полученный от клиента.

Возвращает

true, если хеши равны, false в противном случае.

4.3.2.3 hashPassword()

```
std::string ConnectToBase::hashPassword (
            const std::string & password,
            const std::string & salt )   [private]
```

Вычисляет хеш пароля с использованием соли.

Вычисляет SHA1 хеш пароля с добавлением соли.

Функция использует алгоритм SHA1 для вычисления хеша.

Аргументы

password	Пароль пользователя.
salt	Соль.

Возвращает

Строка с хешем пароля в шестнадцатеричном формате (40 символов).

Заметки

Функция использует функции SHA1 из библиотеки OpenSSL.

Аргументы

password	Пароль, который необходимо хешировать.
salt	Соль, добавляемая к паролю перед хешированием.

Возвращает

Строка, представляющая собой SHA1 хеш пароля с солью.

Заметки

Функция использует библиотеку OpenSSL для вычисления SHA1 хеша.

Объявления и описания членов классов находятся в файлах:

- ConnectToBase.h
- ConnectToBase.cpp

4.4 Класс Error

Класс Error предоставляет функциональность для логирования ошибок.

#include <Error.h>

Открытые статические члены

- static void logError (const std::string &message, bool isCritical=false)

  Логирует сообщение об ошибке.

4.4.1 Подробное описание

Класс Error предоставляет функциональность для логирования ошибок.

Класс Error предоставляет статический метод logError, который выводит сообщение об ошибке в стандартный поток ошибок (cerr). Сообщение об ошибке может быть помечено как критическое.

4.4.2 Методы

4.4.2.1 logError()

```
void Error::logError (
                const std::string & message,
                bool isCritical = false )   [static]
```

Логирует сообщение об ошибке.

Выводит сообщение об ошибке в поток ошибок.

Функция принимает сообщение об ошибке и флаг, указывающий, является ли ошибка критической. Сообщение об ошибке выводится в стандартный поток ошибок (cerr) с префиксом "Error: " или "Critical Error: " в зависимости от значения флага isCritical.

Аргументы

message	Сообщение об ошибке.
isCritical	Флаг, указывающий, является ли ошибка критической. Если true, ошибка считается критической.

Заметки

Функция не записывает сообщения об ошибках в файл.

Функция не генерирует исключений.

Аргументы

message	Сообщение об ошибке.
isCritical	Флаг, указывающий, является ли ошибка критической. Если true, то сообщение будет выведено с префиксом "Critical Error: ", иначе с префиксом "Error: ".

Объявления и описания членов классов находятся в файлах:

- Error.h
- Error.cpp

4.5 Структура FileTestFixture

Открытые атрибуты

- const std::string TEST_DB_FILE = "test_db.txt"

- const std::string TEST_LOG_FILE = "test_log.txt"

Объявления и описания членов структуры находятся в файле:

- tests.cpp

4.6 Класс Interface

Класс Interface предоставляет интерфейс для работы сервера, включая разбор аргументов командной строки, запуск сервера, логирование сообщений и ошибок.

#include <Interface.h>

Открытые статические члены

- static void printUsage ()
  Выводит справку по использованию сервера.
- static void logMessage (const std::string &logFileName, const std::string &message)
  Логирует сообщение в файл лога.
- static void logError (const std::string &logFileName, const std::string &message, bool isCritical)
  Логирует ошибку в файл лога и в поток ошибок.
- static int runServer (int argc, char *argv[])
  Запускает сервер.
- static int getParseResult (int argc, char **argv, std::stringstream &buffer)
  Разбирает аргументы командной строки и возвращает результат.

Закрытые статические члены

- static int parseCommandLine (int argc, char *argv[])
  Разбирает аргументы командной строки.
- static int startServer ()
  Запускает сервер и начинает прослушивание порта.

Закрытые статические данные

- static std::string logFileName = ""
  Имя файла лога.
- static std::string userDbFileName = ""
  Имя файла базы данных пользователей.
- static int port = 33333
  Порт, на котором сервер будет принимать соединения.

4.6.1 Подробное описание

Класс Interface предоставляет интерфейс для работы сервера, включая разбор аргументов командной строки, запуск сервера, логирование сообщений и ошибок.

Класс предоставляет следующие статические методы:

- printUsage: выводит справку по использованию сервера.
- logMessage: логирует сообщение в файл лога.
- logError: логирует ошибку в файл лога и в поток ошибок.
- runServer: запускает сервер.
- getParseResult: разбирает аргументы командной строки, записывая ошибки в буфер.

Класс хранит следующие статические данные:

- logFileName: имя файла лога.
- userDbFileName: имя файла базы данных пользователей.
- port: порт, на котором запускается сервер.

4.6.2 Методы

4.6.2.1 getParseResult()

```
int Interface::getParseResult (
                int argc,
                char ** argv,
                std::stringstream & buffer )   [static]
```

Разбирает аргументы командной строки и возвращает результат.

Разбирает командную строку и возвращает результат.

Функция разбирает аргументы командной строки, устанавливая значения полей logFileName, user↩DbFileName и port. Ошибки, возникшие при разборе, записываются в переданный буфер buffer.

Аргументы

argc	Количество аргументов командной строки.
argv	Аргументы командной строки.
buffer	Буфер для записи сообщений об ошибках.

Возвращает

0 в случае успешного разбора аргументов, ненулевое значение в случае ошибки:

- 1, если не удалось разобрать аргументы (например, пропущен обязательный аргумент).
- -1, если передан некорректный номер порта.

Заметки

Функция использует parseCommandLine для разбора аргументов.

Функция перенаправляет std::cerr в buffer на время разбора аргументов.

Аргументы

argc	Количество аргументов командной строки.
argv	Аргументы командной строки.
buffer	Буфер для записи сообщений об ошибках.

Возвращает

Результат разбора командной строки (0 в случае успеха, ненулевое значение в случае ошибки).

4.6.2.2 logError()

```
void Interface::logError (
        const std::string & logFileName,
        const std::string & message,
        bool isCritical )   [static]
```

Логирует ошибку в файл лога и в поток ошибок.

Записывает сообщение об ошибке в лог-файл.

Функция добавляет строку-разделитель с текущим временем, после чего записывает переданное сообщение об ошибке в файл. Также выводит сообщение в std::cerr.

Аргументы

logFileName	Имя файла лога.
message	Сообщение об ошибке.
isCritical	Флаг, указывающий, является ли ошибка критической.

Заметки

Функция не обрабатывает ошибки открытия файла.

Аргументы

logFileName	Имя лог-файла.
message	Сообщение об ошибке.
isCritical	Флаг, указывающий, является ли ошибка критической.

4.6.2.3 logMessage()

```
void Interface::logMessage (
```

const std::string & logFileName,
const std::string & message) [static]

Логирует сообщение в файл лога.

Записывает информационное сообщение в лог-файл.

Функция добавляет строку-разделитель с текущим временем, после чего записывает переданное информационное сообщение в файл.

Аргументы

logFileName	Имя файла лога.
message	Сообщение для логирования.

Заметки

Функция не обрабатывает ошибки открытия файла.

Аргументы

logFileName	Имя лог-файла.
message	Информационное сообщение.

4.6.2.4 parseCommandLine()

int Interface::parseCommandLine (
int argc,
char * argv[]) [static], [private]

Разбирает аргументы командной строки.

Парсит командную строку.

Функция устанавливает значения полей logFileName, userDbFileName и port на основе переданных аргументов.

Аргументы

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

Возвращает

0 в случае успешного разбора, 1 в случае ошибки (например, пропущен обязательный аргумент или указан неверный формат).

Заметки

Функция использует printUsage для вывода справки по использованию в случае ошибки.

Функция проверяет, что обязательные аргументы -l и -b указаны.

Функция проверяет, что порт находится в допустимом диапазоне (1-65535).

Аргументы

argc	Количество аргументов командной строки.
argv	Аргументы командной строки.

Возвращает

0 в случае успеха, 1 в случае ошибки.

4.6.2.5 printUsage()

```
void Interface::printUsage ( )  [static]
```

Выводит справку по использованию сервера.

Выводит информацию об использовании программы.

Функция выводит в стандартный поток вывода (std::cout) информацию о том, как запустить сервер, какие аргументы командной строки он принимает и какие значения по умолчанию используются для этих аргументов.

4.6.2.6 runServer()

```
int Interface::runServer (
            int argc,
            char * argv[] )  [static]
```

Запускает сервер.

Запускает сервер с заданными параметрами командной строки.

Функция осуществляет разбор командной строки, проверку доступности файлов базы данных и логов, создание и настройку сокета, ожидание и обработку клиентских подключений.

Аргументы

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

Возвращает

0 в случае успешного выполнения, ненулевое значение в случае ошибки:

- 1, если не удалось разобрать аргументы командной строки.

- -1, если произошла ошибка при запуске сервера (например, не удалось создать сокет, привязать его к адресу или начать прослушивание).

Заметки

Функция использует parseCommandLine для разбора аргументов командной строки.

Функция использует startServer для запуска сервера.

Функция блокирует выполнение до тех пор, пока сервер не завершит работу.

Аргументы

argc	Количество аргументов командной строки.
argv	Аргументы командной строки.

Возвращает

0 в случае успеха, ненулевое значение в случае ошибки.

4.6.2.7 startServer()

int Interface::startServer ( ) [static], [private]

Запускает сервер и начинает прослушивание порта.

Запускает сервер.

Функция выполняет следующие действия:

1. Проверяет доступность файлов логов и базы данных.
2. Создает сокет.
3. Устанавливает параметры сокета (переиспользование адреса и порта).
4. Привязывает сокет к адресу и порту.
5. Начинает прослушивание порта.
6. В бесконечном цикле ожидает входящие соединения.
7. При поступлении соединения принимает его.
8. Создает объект ClientCommunicate для обработки клиента.
9. Вызывает ClientCommunicate::communicate для взаимодействия с клиентом.
10. Закрывает соединение с клиентом.
11. Повторяет шаги 6-10.

Возвращает

0 в случае успешного запуска, -1 в случае ошибки на любом из этапов.

Заметки

Функция использует Interface::logError для логирования ошибок.

Функция использует Interface::logMessage для логирования информационных сообщений.

Функция использует ClientCommunicate для обработки клиентских подключений.

Функция блокирует выполнение, пока сервер не завершит работу.

Возвращает

0 в случае успеха, -1 в случае ошибки.

4.6.3 Данные класса

4.6.3.1 logFileName

std::string Interface::logFileName = "" [static], [private]

Имя файла лога.

Статическая переменная, хранящая имя файла, в который будут записываться логи. Значение устанавливается функцией parseCommandLine.

4.6.3.2 port

int Interface::port = 33333 [static], [private]

Порт, на котором сервер будет принимать соединения.

Статическая переменная, хранящая номер порта, на котором сервер будет ожидать входящие соединения. Значение устанавливается функцией parseCommandLine, значение по умолчанию - 33333.

4.6.3.3 userDbFileName

std::string Interface::userDbFileName = "" [static], [private]

Имя файла базы данных пользователей.

Статическая переменная, хранящая имя файла, содержащего данные пользователей (логины и пароли). Значение устанавливается функцией parseCommandLine.

Объявления и описания членов классов находятся в файлах:

- Interface.h
- Interface.cpp

4.7 Структура ParsedMessage

Структура для хранения разобранного сообщения от клиента.

#include <ClientCommunicate.h>

Открытые атрибуты

- std::string login

  Логин пользователя.
- std::string salt

  Соль, используемая при хешировании пароля.
- std::string hash

  Хеш пароля пользователя.

4.7.1 Подробное описание

Структура для хранения разобранного сообщения от клиента.

Структура содержит поля для хранения логина, соли и хеша, извлеченных из сообщения клиента. Эти данные используются для аутентификации пользователя.

4.7.2 Данные класса

4.7.2.1 hash

std::string ParsedMessage::hash

Хеш пароля пользователя.

Строка, содержащая хеш пароля, извлеченный из сообщения клиента. Хеш используется для проверки подлинности пользователя без необходимости хранения пароля в открытом виде.

4.7.2.2 login

std::string ParsedMessage::login

Логин пользователя.

Строка, содержащая логин пользователя, извлеченный из сообщения клиента.

4.7.2.3 salt

std::string ParsedMessage::salt

Соль, используемая при хешировании пароля.

Строка, содержащая соль, извлеченную из сообщения клиента. Соль используется для повышения безопасности хранения паролей.

Объявления и описания членов структуры находятся в файле:

- ClientCommunicate.h

Глава 5

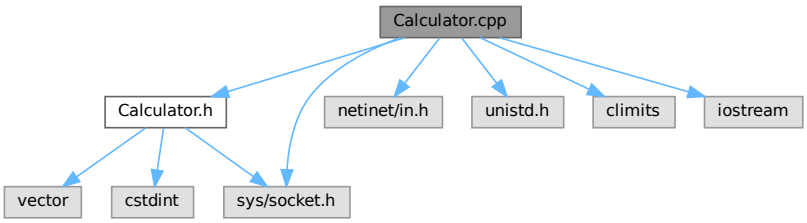
Файлы

5.1 Файл Calculator.cpp

Содержит реализацию класса Calculator.

```
#include "Calculator.h"
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>
#include <climits>
#include <iostream>
```

Граф включаемых заголовочных файлов для Calculator.cpp:



5.1.1 Подробное описание

Содержит реализацию класса Calculator.

Автор

Бренинг Иван

Дата

13.12.24

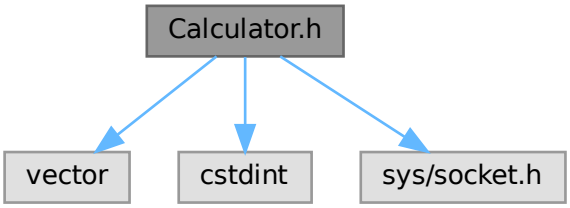
Версия

1.0

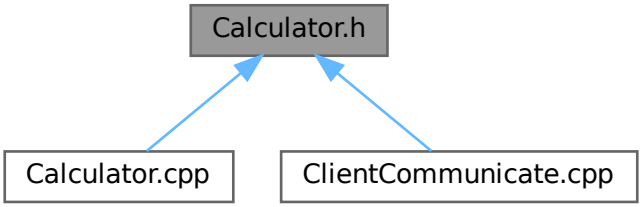
5.2 Файл Calculator.h

Содержит объявление класса Calculator.

#include <vector>
#include <cstdint>
#include <sys/socket.h>
Граф включаемых заголовочных файлов для Calculator.h:



Граф файлов, в которые включается этот файл:



Классы

- class Calculator

  Класс Calculator предоставляет функциональность для вычисления суммы квадратов элементов вектора и обработки векторов, полученных через сетевое соединение.

5.2.1 Подробное описание

Содержит объявление класса Calculator.

Автор

Бренинг Иван

Дата

13.12.24

Версия

1.0

5.3 Calculator.h

См. документацию.

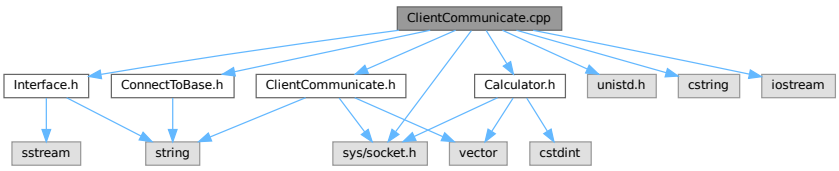
```
00001 #ifndef CALCULATOR_H
00002 #define CALCULATOR_H
00003
00004 #include <vector>
00005 #include <cstdint>
00006 #include <sys/socket.h>
00007
00025 class Calculator {
00026 public:
00045     uint16_t processVectors(int socket);
00046
00059     static uint32_t calculateSumOfSquares(const std::vector<uint16_t>& vec);
00060
00061 private:
00062
00063 };
00064
00065 #endif
```

5.4 Файл ClientCommunicate.cpp

Содержит реализацию класса ClientCommunicate.

```
#include "ClientCommunicate.h"
#include "ConnectToBase.h"
#include "Calculator.h"
#include "Interface.h"
#include <sys/socket.h>
#include <unistd.h>
#include <cstring>
#include <iostream>
```

Граф включаемых заголовочных файлов для ClientCommunicate.cpp:



5.4.1 Подробное описание

Содержит реализацию класса ClientCommunicate.

Автор

Бренинг Иван

Дата

13.12.24

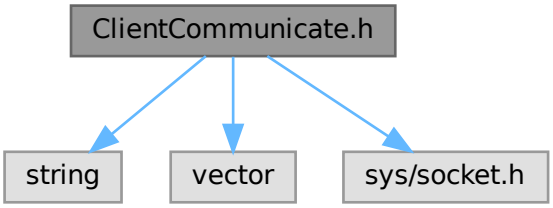
Версия

1.0

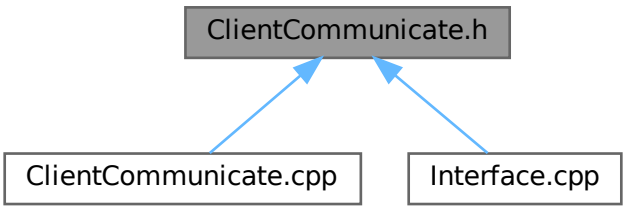
5.5 Файл ClientCommunicate.h

Содержит объявление класса ClientCommunicate и структуры ParsedMessage.

#include <string>
#include <vector>
#include <sys/socket.h>
Граф включаемых заголовочных файлов для ClientCommunicate.h:



Граф файлов, в которые включается этот файл:



Классы

- struct ParsedMessage

  Структура для хранения разобранного сообщения от клиента.
- class ClientCommunicate

  Класс ClientCommunicate отвечает за взаимодействие с клиентом.

5.5.1 Подробное описание

Содержит объявление класса ClientCommunicate и структуры ParsedMessage.

Автор

Бренинг Иван

Дата

13.12.24

Версия

1.0

5.6 ClientCommunicate.h

См. документацию.

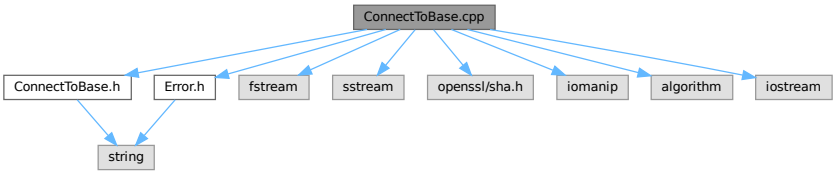
```
00001 #ifndef CLIENTCOMMUNICATE_H
00002 #define CLIENTCOMMUNICATE_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include <sys/socket.h>
00007
00022 struct ParsedMessage {
00028     std::string login;
00035     std::string salt;
00042     std::string hash;
00043 };
00044
00051 class ClientCommunicate {
00052 public:
00083     void communicate(int socket, const std::string& userDbFileName, const std::string& logFileName);
00084
00100     static ParsedMessage parseMessage(const std::string& message);
00101
00102 private:
00103
00104 };
00105
00106 #endif
```

5.7 Файл ConnectToBase.cpp

Содержит реализацию класса ConnectToBase.

```
#include "ConnectToBase.h"
#include "Error.h"
#include <fstream>
#include <sstream>
#include <openssl/sha.h>
#include <iomanip>
#include <algorithm>
#include <iostream>
```

Граф включаемых заголовочных файлов для ConnectToBase.cpp:



5.7.1 Подробное описание

Содержит реализацию класса ConnectToBase.

Автор

Бренинг Иван

Дата

13.12.24

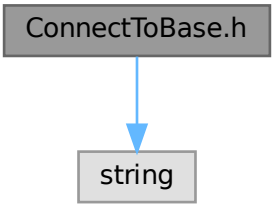
Версия

1.0

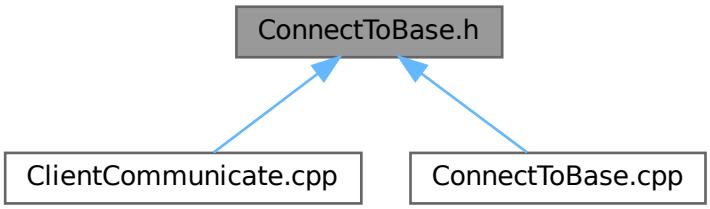
5.8 Файл ConnectToBase.h

Содержит объявление класса ConnectToBase.

#include <string>
Граф включаемых заголовочных файлов для ConnectToBase.h:



Граф файлов, в которые включается этот файл:



Классы

- class ConnectToBase

  Класс ConnectToBase отвечает за взаимодействие с базой данных пользователей.

5.8.1 Подробное описание

Содержит объявление класса ConnectToBase.

Автор

Бренинг Иван

Дата

13.12.24

Версия

1.0

5.9 ConnectToBase.h

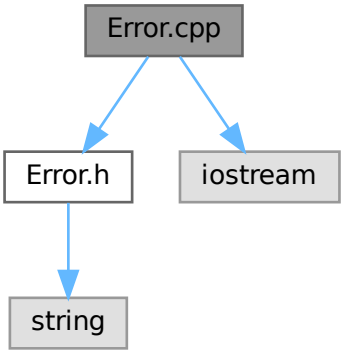
См. документацию.

```
00001 #ifndef CONNECTTOBASE_H
00002 #define CONNECTTOBASE_H
00003
00004 #include <string>
00005
00019 class ConnectToBase {
00020 public:
00043     bool authenticateUser(const std::string& login, const std::string& salt, const std::string& clientHash, const std::string&
      dbFileName);
00044
00045 private:
00057     std::string hashPassword(const std::string& password, const std::string& salt);
00058
00070     bool compareHashes(const std::string& serverHash, const std::string& clientHash);
00071 };
00072
00073 #endif
```

5.10 Файл Error.cpp

Содержит реализацию класса Error.

#include "Error.h"
#include <iostream>
Граф включаемых заголовочных файлов для Error.cpp:



5.10.1 Подробное описание

Содержит реализацию класса Error.

Автор

Бренинг Иван

Дата

13.12.24

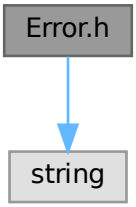
Версия

1.0

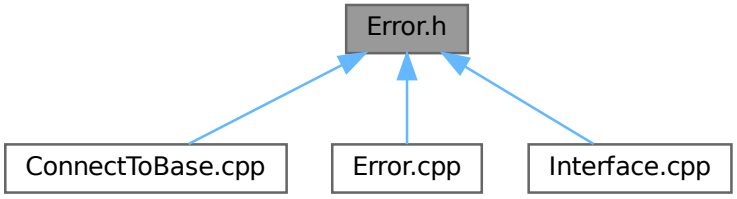
5.11 Файл Error.h

Содержит объявление класса Error.

#include <string>
Граф включаемых заголовочных файлов для Error.h:



Граф файлов, в которые включается этот файл:



Классы

- class Error
  Класс Error предоставляет функциональность для логирования ошибок.

5.11.1 Подробное описание

Содержит объявление класса Error.

Автор

Бренинг Иван

Дата

13.12.24

Версия

1.0

5.12 Error.h

См. документацию.

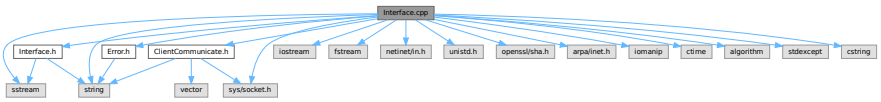
```
00001 #ifndef ERROR_H
00002 #define ERROR_H
00003
00004 #include <string>
00005
00020 class Error {
00021 public:
00034     static void logError(const std::string& message, bool isCritical = false);
00035 };
00036
00037 #endif
```

5.13 Файл Interface.cpp

Содержит реализацию класса Interface.

```
#include "Interface.h"
#include "ClientCommunicate.h"
#include "Error.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <netinet/in.h>
#include <sys/socket.h>
#include <unistd.h>
#include <openssl/sha.h>
#include <arpa/inet.h>
#include <iomanip>
#include <ctime>
#include <algorithm>
#include <stdexcept>
#include <cstring>
```

Граф включаемых заголовочных файлов для Interface.cpp:



5.13.1 Подробное описание

Содержит реализацию класса Interface.

Автор

Бренинг Иван

Дата

13.12.24

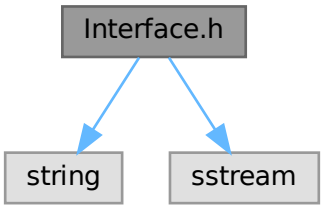
Версия

1.0

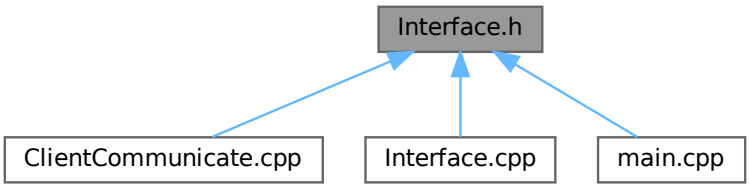
5.14 Файл Interface.h

Содержит объявление класса Interface.

#include <string>
#include <sstream>
Граф включаемых заголовочных файлов для Interface.h:



Граф файлов, в которые включается этот файл:



Классы

- class Interface

  Класс Interface предоставляет интерфейс для работы сервера, включая разбор аргументов командной строки, запуск сервера, логирование сообщений и ошибок.

5.14.1 Подробное описание

Содержит объявление класса Interface.

Автор

Бренинг Иван

Дата

13.12.24

Версия

1.0

5.15 Interface.h

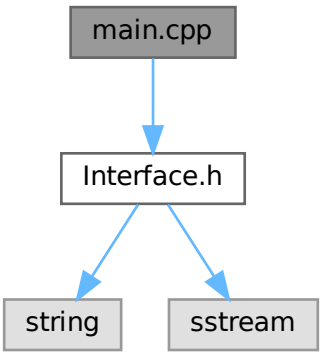
См. документацию.

```
00001 #ifndef INTERFACE_H
00002 #define INTERFACE_H
00003
00004 #include <string>
00005 #include <sstream>
00006
00031 class Interface {
00032 public:
00039     static void printUsage();
00040
00052     static void logMessage(const std::string& logFileName, const std::string& message);
00053
00066     static void logError(const std::string& logFileName, const std::string& message, bool isCritical);
00067
00085     static int runServer(int argc, char* argv[]);
00086
00103     static int getParseResult(int argc, char** argv, std::stringstream& buffer);
00104
00105 private:
00112     static std::string logFileName;
00113
00120     static std::string userDbFileName;
00121
00128     static int port;
00129
00143     static int parseCommandLine(int argc, char* argv[]);
00144
00168     static int startServer();
00169 };
00170
00171 #endif
```

5.16 Файл main.cpp

Главный файл приложения сервера.

#include "Interface.h"
Граф включаемых заголовочных файлов для main.cpp:



Функции

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

5.16.1 Подробное описание

Главный файл приложения сервера.

Автор

Бренинг Иван

Дата

13.12.24

Версия

1.0

Файл содержит точку входа в приложение. Инициирует запуск сервера с помощью вызова Interface::runServer.

5.16.2 Функции

5.16.2.1 main()

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

Точка входа в приложение.

Аргументы

argc	Количество аргументов командной строки.
argv	Аргументы командной строки.

Возвращает

0 в случае успешного завершения, ненулевое значение в случае ошибки.

Предметный указатель