sumOfSquares

Создано системой Doxygen 1.9.4

Глава 1

Math_server_sumOfSquares

Разработка сервера для курсовой.

Глава 2

Иерархический список классов

2.1 Иерархия классов

Иерархия классов.

Глава 3

Алфавитный указатель классов

3.1 Классы

Классы с их кратким описанием.

Глава 4

Список файлов

4.1 Файлы

Полный список файлов.

Глава 5

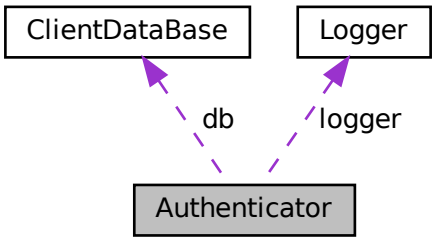
Классы

5.1 Класс Authenticator

Обрабатывает аутентификацию пользователей по базе данных.

#include <Auth.h>

Граф связей класса Authenticator:



Открытые члены

- Authenticator (const std::string &db_filename, Logger &logger)
  Конструктор объекта Authenticator.
- bool isLoginExists (const std::string &login)
  Проверяет, существует ли логин в базе данных.
- std::string generateSalt ()
  Генерирует случайную соль длиной 16 символов в шестнадцатеричном формате.
- std::string hashPassword (const std::string &salt_16, const std::string &password)
  Хэширует пароль с использованием SHA224 и заданной солью.
- bool compareHashes (const std::string &clientHash, const std::string &salt, const std::string &login)
  Сравнивает полученный от клиента хэш с сгенерированным на сервере.
- bool verifyPassword (const std::string &login, const std::string &password, const std::string &salt)
  Проверяет пароль пользователя в базе данных.

Закрытые данные

- ClientDataBase db

  База данных для хранения учетных данных пользователей.
- Logger & logger

  Объект логгера для записи событий аутентификации.

5.1.1 Подробное описание

Обрабатывает аутентификацию пользователей по базе данных.

5.1.2 Конструктор(ы)

5.1.2.1 Authenticator()

```
Authenticator::Authenticator (
            const std::string & db_filename,
            Logger & logger )
```

Конструктор объекта Authenticator.

Аргументы

db_filename	Путь к файлу базы данных.
logger	Ссылка на объект логгера.

```
7     : db(db_filename), logger(logger)
8 {
9 }
```

5.1.3 Методы

5.1.3.1 compareHashes()

```
bool Authenticator::compareHashes (
            const std::string & clientHash,
            const std::string & salt,
            const std::string & login )
```

Сравнивает полученный от клиента хэш с сгенерированным на сервере.

Аргументы

clientHash	Хэш, полученный от клиента.
salt	Соль, использованная для хэширования.
login	Логин пользователя.

Возвращает

True, если хэши совпадают, иначе false.

```
46 {
47     string password = db[login];
48     logger.log(INFO, "Хэш от клиента: " + clientHash);
49     std::string generatedHash = hashPassword(salt, password);
50      logger.log(INFO, "Сравниваю хэши (Сервер == клиент): " + generatedHash + " == " + clientHash );
51     return generatedHash == clientHash;
52 }
```

5.1.3.2 generateSalt()

string Authenticator::generateSalt ( )

Генерирует случайную соль длиной 16 символов в шестнадцатеричном формате.

Возвращает

Сгенерированная соль.

```
16 {
17     namespace CPP = CryptoPP;
18     constexpr int salt_size = 8; // 64 бита = 8 байт
19     CPP::byte salt[salt_size];
20     CPP::AutoSeededRandomPool prng;
21     prng.GenerateBlock(salt, salt_size);
22     std::string salt_16;
23     CPP::ArraySource(salt, salt_size, true, new CPP::HexEncoder(new CPP::StringSink(salt_16)));
24 
25     //тут  дополняем слева нулями до 16 символов
26     while(salt_16.length() < 16) {
27         salt_16.insert(salt_16.begin(), '0');
28     }
29     return salt_16;
30 }
```

5.1.3.3 hashPassword()

string Authenticator::hashPassword (
const std::string & salt_16,
const std::string & password)

Хэширует пароль с использованием SHA224 и заданной солью.

Аргументы

salt_16	Соль длиной 16 символов в шестнадцатеричном формате.
password	Пароль для хэширования.

Возвращает

Хэш SHA224 пароля, конкатенированного с солью.

```
33 {
34     namespace CPP = CryptoPP;
35     string HashAndPass = salt_16 + password;
```

```
36     CPP::SHA224 hash;
37     CPP::byte digest[CryptoPP::SHA224::DIGESTSIZE];
38     hash.CalculateDigest(digest, (const CPP::byte*)HashAndPass.data(), HashAndPass.size());
39     string Hash;
40     CryptoPP::StringSource(digest, sizeof(digest), true, new CryptoPP::HexEncoder(new CryptoPP::StringSink(Hash)));
41     logger.log(INFO, "Сервер сгенерировал хэш: "+ Hash);
42     return Hash;
43 }
```

5.1.3.4 isLoginExists()

```
bool Authenticator::isLoginExists (
                const std::string & login )
```

Проверяет, существует ли логин в базе данных.

Аргументы

login	Логин пользователя для проверки.

Возвращает

True, если логин существует, иначе false.

```
12 {
13     return db.isLoginExists(login);
14 }
```

5.1.3.5 verifyPassword()

```
bool Authenticator::verifyPassword (
                const std::string & login,
                const std::string & password,
                const std::string & salt )
```

Проверяет пароль пользователя в базе данных.

Аргументы

login	Логин пользователя.
password	Пароль пользователя.
salt	Соль, связанная с паролем пользователя.

Возвращает

True, если пароль валиден, иначе false.

5.1.4 Данные класса

5.1.4.1 db

ClientDataBase Authenticator::db [private]

База данных для хранения учетных данных пользователей.

5.1.4.2 logger

Logger& Authenticator::logger [private]

Объект логгера для записи событий аутентификации.

Объявления и описания членов классов находятся в файлах:

- Auth.h
- Auth.cpp

5.2 Класс ClientDataBase

Класс для работы с базой данных клиентов, хранящей логины и пароли.

#include <ClientDataBase.h>

Открытые члены

- ClientDataBase (const std::string &filename)
  Конструктор класса ClientDataBase.
- bool isLoginExists (const std::string &login)
  Проверяет существование логина в базе данных.
- std::string & operator[] (const std::string &login)
  Возвращает ссылку на пароль по заданному логину.
- const std::string & operator[] (const std::string &login) const
  Возвращает константную ссылку на пароль по заданному логину.

Закрытые члены

- void loadDatabase (const std::string &filename)
  Загружает данные из файла в базу данных.

Закрытые данные

- std::unordered_map< std::string, std::string > database
  Внутренняя база данных, хранящая пары логин-пароль.

5.2.1 Подробное описание

Класс для работы с базой данных клиентов, хранящей логины и пароли.

База данных загружается из файла при создании объекта. Обеспечивает проверку существования логина и доступ к паролю по логину.

5.2.2 Конструктор(ы)

5.2.2.1 ClientDataBase()

```
ClientDataBase::ClientDataBase (
            const std::string & filename )
```

Конструктор класса ClientDataBase.

Аргументы

filename	Имя файла базы данных.

Исключения

DatabaseException	Если в файле обнаружен логин или пароль без пары.
std::runtime_error	Если база данных пуста после загрузки.
std::system_error	Если не удалось открыть файл.

```
10 {
11     loadDatabase(filename);
12 }
```

5.2.3 Методы

5.2.3.1 isLoginExists()

```
bool ClientDataBase::isLoginExists (
            const std::string & login )
```

Проверяет существование логина в базе данных.

Аргументы

login	Проверяемый логин.

Возвращает

true, если логин существует, false в противном случае.

```
15 {
16     return database.find(login) != database.end();
17 }
```

5.2.3.2 loadDatabase()

void ClientDataBase::loadDatabase (
const std::string & filename) [private]

Загружает данные из файла в базу данных.

Аргументы

filename	Имя файла базы данных.

Исключения

DatabaseException	Если в файле обнаружен логин или пароль без пары.
std::runtime_error	Если база данных пуста после загрузки.
std::system_error	Если не удалось открыть файл.

```
19                                          {
20     ifstream file(filename);
21     if (file.is_open()) {
22         string line;
23         while (getline(file, line, ':')) {
24             string login = line;
25             string password;
26             getline(file, password);
27             password.erase(password.find_last_not_of(" \t\r\n") + 1);
28             if (login.empty()) {
29                 std::string message = "Предупреждение: Обнаружен пароль без логина в базе данных";
30                 throw DatabaseException(message);
31             }
32             if (password.empty()) {
33                 string message =  "Предупреждение: Обнаружен логин без пароля в базе данных";
34                 throw DatabaseException(message);
35             }
36 
37             if (login.empty() || password.empty()) {
38                 continue;
39             }
40 
41             database[login] = password;
42         }
43 
44         file.close();
45         if (database.empty()) {
46             throw std::runtime_error("Критическая ошибка: База данных пуста после загрузки из файла " + filename);
47         }
48     } else {
49         cerr « "Ошибка: Не удалось открыть файл " « filename « endl;
50         std::string error_message = strerror(errno);
51         std::string exception_message = "Критическая ошибка: Не удалось открыть файл " + filename + " " +
       error_message;
52         throw std::system_error(errno, std::generic_category(), exception_message);
53     }
54 }
```

5.2.3.3 operator[]() [1/2]

```
std::string & ClientDataBase::operator[] (
            const std::string & login )
```

Возвращает ссылку на пароль по заданному логину.

Аргументы

login	Логин.

Возвращает

Ссылка на строку, содержащую пароль.

```
57 {
58     return database[login];
59 }
```

5.2.3.4 operator[]() [2/2]

```
const std::string & ClientDataBase::operator[] (
            const std::string & login ) const
```

Возвращает константную ссылку на пароль по заданному логину.

Аргументы

login	Логин.

Возвращает

Константная ссылка на строку, содержащую пароль.

```
62 {
63     return database.at(login);
64 }
```

5.2.4 Данные класса

5.2.4.1 database

```
std::unordered_map<std::string, std::string> ClientDataBase::database  [private]
```

Внутренняя база данных, хранящая пары логин-пароль.

Объявления и описания членов классов находятся в файлах:

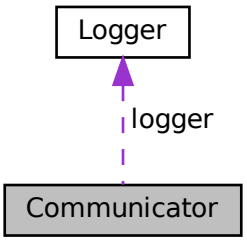
- ClientDataBase.h
- ClientDataBase.cpp

5.3 Класс Communicator

Этот класс отвечает за обработку коммуникации с клиентом. Он принимает векторы данных от клиента, обрабатывает их и отправляет результат обратно.

#include <Communicator.h>

Граф связей класса Communicator:



Открытые члены

- Communicator (int client_socket, const std::string &dataFile, Logger &logger)

  Конструктор класса Communicator. Инициализирует объект, устанавливая связь с клиентом и настраивая логирование.
- void handleClient ()

  Основной метод обработки запроса клиента. Принимает данные от клиента, обрабатывает их и отправляет ответ.
- ~Communicator ()

  Деструктор класса Communicator. Закрывает сокет после завершения работы.
- void processVectors ()

  Обрабатывает полученные от клиента векторы данных.
- void sendResultToClient (int32_t resultValue)

  Отправляет результат обработки векторов клиенту.

Закрытые члены

- void sendResponse (const std::string &response)

  Отправляет ответ клиенту через сокет.

Закрытые данные

- std::string dataFile

  Путь к файлу данных, используемому в процессе обработки (если требуется).
- Logger & logger

  Ссылка на объект Logger для ведения логов.
- int client_socket

  Дескриптор сокета, используемого для общения с клиентом.

5.3.1 Подробное описание

Этот класс отвечает за обработку коммуникации с клиентом. Он принимает векторы данных от клиента, обрабатывает их и отправляет результат обратно.

5.3.2 Конструктор(ы)

5.3.2.1 Communicator()

```
Communicator::Communicator (
            int client_socket,
            const std::string & dataFile,
            Logger & logger )
```

Конструктор класса Communicator. Инициализирует объект, устанавливая связь с клиентом и настраивая логирование.

Аргументы

client_socket	Дескриптор сокета, через который происходит общение с клиентом.
dataFile	Путь к файлу данных (если используется).
logger	Ссылка на объект Logger для записи логов.

```
16     : dataFile(dataFile)
17     , logger(logger)
18     , client_socket(socket)
19 {
20 }
```

5.3.2.2 ~Communicator()

```
Communicator::~Communicator ( )
```

Деструктор класса Communicator. Закрывает сокет после завершения работы.
143 { close(client_socket); }

5.3.3 Методы

5.3.3.1 handleClient()

```
void Communicator::handleClient ( )
```

Основной метод обработки запроса клиента. Принимает данные от клиента, обрабатывает их и отправляет ответ.

```
23 {
24     char login[1024] = { 0 };
25     ssize_t recvlogin = recv(client_socket, login, sizeof(login), 0);
26     if(recvlogin == 0) {
27 
28         std::string error_message = strerror(errno);
29         std::string exception_message = "Клиент закрыл соединение: " + error_message;
30         logger.log(ERROR, exception_message);
31     } else if(recvlogin < 0) {
32         std::string error_message = strerror(errno);
33         std::string exception_message = "Ошибка получения логина: " + error_message;
34         throw std::system_error(errno, std::generic_category(), exception_message);
35     }
36     for(ssize_t i = 0; i < recvlogin; ++i) {
37         if(login[i] == '\n') {
38             login[i] = '\0';
39             break;
40         }
41     }
42     logger.log(INFO, "Клиент прислал логин: " + string(login));
43     Authenticator authenticator(dataFile, logger);
44     if(authenticator.isLoginExists(login) == false) {
45         logger.log(ERROR, "Клиент не прошел аутентификацию");
46         const string errMsg = "ERR";
47         send(client_socket, errMsg.c_str(), errMsg.size(), 0);
48         close(client_socket);
49     } else {
50         string salt = authenticator.generateSalt();
51         send(client_socket, salt.c_str(), salt.size(), 0);
52         logger.log(INFO, "Клиенту отправлена соль: " + salt);
53         char clientHash[1024] = { 0 };
54         ssize_t recvlclientHash = recv(client_socket, clientHash, sizeof(clientHash), 0);
55         if(recvlogin == 0) {
56             std::string error_message = strerror(errno);
57             std::string exception_message = "Клиент закрыл соединение: " + error_message;
58             logger.log(ERROR, exception_message);
59         } else if(recvlogin < 0) {
60             std::string error_message = strerror(errno);
61             std::string exception_message = "Ошибка получения хеша: " + error_message;
62             throw std::system_error(errno, std::generic_category(), exception_message);
63         }
64         for(ssize_t i = 0; i < recvlclientHash; ++i) {
65             if(clientHash[i] == '\n') {
66                 clientHash[i] = '\0';
67                 break;
68             }
69         }
70         if(authenticator.compareHashes(clientHash, salt, login)) {
71             logger.log(INFO, "Клиент прошел аутентификацию");
72             const string successMsg = "OK\n";
73             send(client_socket, successMsg.c_str(), successMsg.size(), 0);
74             processVectors();
75         } else {
76             logger.log(ERROR, "Клиент не прошел аутентификацию");
77             const string errMsg = "ERR\n";
78             send(client_socket, errMsg.c_str(), errMsg.size(), 0);
79         }
80     }
81 }
```

5.3.3.2 processVectors()

void Communicator::processVectors ( )

Обрабатывает полученные от клиента векторы данных.

```
85                          {
86     logger.log(INFO, "Начинаю обработку векторов");
87     uint32_t numVectors{0};
88     uint32_t vectorSize{0};
89 
90     ssize_t recvBytes = recv(client_socket, &numVectors, sizeof(numVectors), 0);
91     if (recvBytes != sizeof(numVectors)) {
92         std::string error_message = strerror(errno);
93         std::string exception_message = "Ошибка получения количества векторов: " + error_message;
94         logger.log(ERROR, exception_message);
95         close(client_socket);
96         return;
```

```
97     }
98
99     logger.log(INFO, "От клиента пришло " + std::to_string(numVectors) + " векторов");
100
101     for (uint32_t i = 0; i < numVectors; ++i) {
102         int rc = recv(client_socket, &vectorSize, sizeof(vectorSize), 0);
103         if (rc == -1) {
104             std::string error_message = strerror(errno);
105             std::string exception_message = "Ошибка получения размера вектора: " + error_message;
106             logger.log(ERROR, exception_message);
107             close(client_socket);
108             return;
109         }
110         std::unique_ptr<int32_t[]> data(new int32_t[vectorSize]);
111         int rc2 = recv(client_socket, data.get(), sizeof(int32_t) * vectorSize, 0);
112         if (rc2 == -1) {
113             std::string error_message = strerror(errno);
114             std::string exception_message = "Ошибка получения данных вектора: " + error_message;
115             logger.log(ERROR, exception_message);
116             close(client_socket);
117             return;
118         } else if (static_cast<size_t>(rc2) != sizeof(int32_t) * vectorSize) {
119             std::string error_message = "Ошибка получения данных: количество полученных байт не совпадает с
    ожидаемым";
120             logger.log(ERROR, error_message);
121             close(client_socket);
122             return;
123         }
124 std::vector<int32_t> vectorValues(data.get(), data.get() + vectorSize);
125         int32_t result = HandlerVector::processVector(vectorValues);
126         logger.log(INFO, "Результат обработки вектора " + std::to_string(i) + ": " + std::to_string(result));
127          sendResultToClient(result);
128     }
129 }
```

5.3.3.3 sendResponse()

void Communicator::sendResponse (
const std::string & response) [private]

Отправляет ответ клиенту через сокет.

Аргументы

response	Строка, содержащая ответ, который нужно отправить клиенту.

5.3.3.4 sendResultToClient()

void Communicator::sendResultToClient (
int32_t resultValue)

Отправляет результат обработки векторов клиенту.

Аргументы

resultValue	Значение результата, которое нужно отправить клиенту. Тип int32_t.

```
133 {
134     ssize_t sentBytes = send(client_socket, &resultValue, sizeof(resultValue), 0);
135     logger.log(INFO, "Отправил клиенту результаты вычислений");
136     if(sentBytes != sizeof(resultValue)) {
```

```
137         std::string error_message = strerror(errno);
138         std::string exception_message = "Ошибка чтения из сокета: " + error_message;
139         logger.log(CRITICAL, exception_message);
140         throw std::system_error(errno, std::generic_category(), exception_message);
141     }
142 }
```

5.3.4 Данные класса

5.3.4.1 client_socket

int Communicator::client_socket [private]

Дескриптор сокета, используемого для общения с клиентом.

5.3.4.2 dataFile

std::string Communicator::dataFile [private]

Путь к файлу данных, используемому в процессе обработки (если требуется).

5.3.4.3 logger

Logger& Communicator::logger [private]

Ссылка на объект Logger для ведения логов.

Объявления и описания членов классов находятся в файлах:

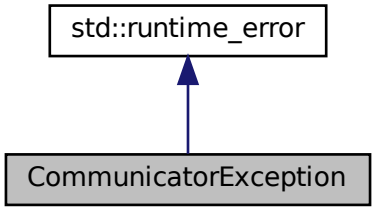
- Communicator.h
- Communicator.cpp

5.4 Класс CommunicatorException

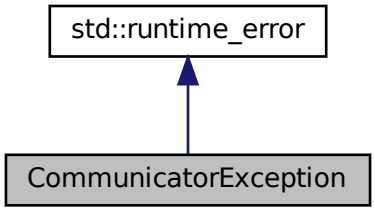
Исключение, возникающее при ошибках в процессе коммуникации.

#include <Errors.h>

Граф наследования:CommunicatorException:



Граф связей класса CommunicatorException:



Открытые члены

- CommunicatorException (const std::string &message)
  Конструктор CommunicatorException.

5.4.1 Подробное описание

Исключение, возникающее при ошибках в процессе коммуникации.

5.4.2 Конструктор(ы)

5.4.2.1 CommunicatorException()

```
CommunicatorException::CommunicatorException (
            const std::string & message )   [inline], [explicit]
```

Конструктор CommunicatorException.

Аргументы

message	Текстовое сообщение, описывающее ошибку.

```
45      : std::runtime_error(message) {}
```

Объявления и описания членов класса находятся в файле:

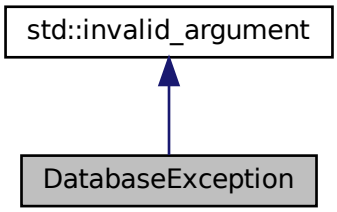
- Errors.h

5.5 Класс DatabaseException

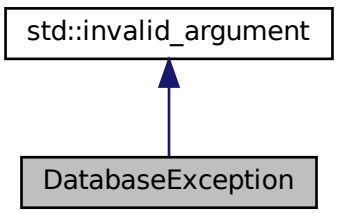
Исключение, возникающее при ошибках работы с базой данных.

#include <Errors.h>

Граф наследования:DatabaseException:



Граф связей класса DatabaseException:



Открытые члены

- DatabaseException (const std::string &what_arg)
  Конструктор DatabaseException.
- DatabaseException (const char *what_arg)
  Конструктор DatabaseException.

5.5.1 Подробное описание

Исключение, возникающее при ошибках работы с базой данных.

5.5.2 Конструктор(ы)

5.5.2.1 DatabaseException() [1/2]

```
DatabaseException::DatabaseException (
            const std::string & what_arg )   [inline], [explicit]
```

Конструктор DatabaseException.

Аргументы

what_arg	Текстовое сообщение, описывающее ошибку.

```
24      : std::invalid_argument(what_arg) {}
```

5.5.2.2 DatabaseException() [2/2]

```
DatabaseException::DatabaseException (
            const char * what_arg )   [inline], [explicit]
```

Конструктор DatabaseException.

Аргументы

what_arg	Текстовое сообщение, описывающее ошибку.

```
31      : std::invalid_argument(what_arg) {}
```

Объявления и описания членов класса находятся в файле:

- Errors.h

5.6 Класс HandlerVector

Класс для обработки целочисленных векторов.

#include <HandlerVector.h>

Открытые статические члены

- static int32_t processVector (const std::vector< int32_t > &vectorValues)

  Вычисляет сумму квадратов элементов вектора.

5.6.1 Подробное описание

Класс для обработки целочисленных векторов.

Предоставляет статический метод processVector для вычисления суммы квадратов элементов вектора. Обрабатывает возможные переполнения.

5.6.2 Методы

5.6.2.1 processVector()

int32_t HandlerVector::processVector (
const std::vector< int32_t > & vectorValues) [static]

Вычисляет сумму квадратов элементов вектора.

Аргументы

vectorValues	Входной вектор целых чисел.

Возвращает

Сумма квадратов элементов вектора, преобразованная в int32_t. Возвращает максимальное или минимальное значение int32_t в случае переполнения.

```
10                                                  {
11     int64_t sumOfSquares = 0;
12     const int64_t maxVal = std::numeric_limits<int32_t>::max();
13     const int64_t minVal = std::numeric_limits<int32_t>::min();
14
15     for (int32_t value : vectorValues) {
16         int64_t square = static_cast<int64_t>(value) * value;
17         // Проверка на переполнение до сложения
18         if (sumOfSquares > maxVal - square) {
19             return static_cast<int32_t>(maxVal);
20         }
21         sumOfSquares += square;
22         // Проверка на переполнение после сложения
23         if (sumOfSquares > maxVal) {
24             return static_cast<int32_t>(maxVal);
25         }
```

```
26         if (sumOfSquares < minVal) {
27             return static_cast<int32_t>(minVal);
28         }
29     }
30     return static_cast<int32_t>(sumOfSquares);
31 }
```

Объявления и описания членов классов находятся в файлах:

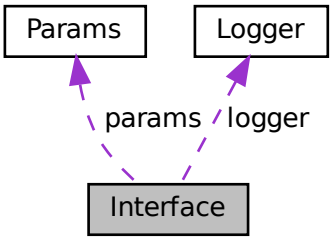
- HandlerVector.h
- HandlerVector.cpp

5.7 Класс Interface

Класс для обработки параметров командной строки и инициализации сервера.

#include <StartInterface.h>

Граф связей класса Interface:



Открытые члены

- Interface ()

  Конструктор класса Interface. Инициализирует параметры по умолчанию.
- bool Parser (int argc, const char **argv)

  Парсит параметры командной строки.
- void processCommands (int argc, const char **argv)

  Обрабатывает параметры командной строки, инициализирует логгер и проверяет корректность параметров.
- const Params & getParams () const

  Возвращает константную ссылку на структуру параметров.
- Logger & getLogger ()

  Возвращает ссылку на объект Logger.
- int getPort () const

  Возвращает номер порта.
- const std::string & getClientDataFile () const

  Возвращает имя файла с данными клиентов.
- string getDescription ()

  Возвращает строковое описание параметров командной строки.

Закрытые данные

- Params params

  Структура для хранения параметров
- Logger logger

  Объект для логирования
- po::options_description desc

  Описание параметров командной строки
- po::variables_map vm

  Хранилище для распарсенных параметров

5.7.1 Подробное описание

Класс для обработки параметров командной строки и инициализации сервера.

Использует библиотеку Boost.Program_options для парсинга аргументов командной строки.

5.7.2 Конструктор(ы)

5.7.2.1 Interface()

Interface::Interface ( )

Конструктор класса Interface. Инициализирует параметры по умолчанию.

```
12                  : logger("/var/log/vcalc.log") {
13     params.dataFileName = "/etc/vcalc.conf";
14     params.logFileName = "/var/log/vcalc.log";
15     params.port = 33333;
16 
17     desc.add_options()
18         ("help,h", "Получить справку")
19         ("data,d", po::value<std::string>(&params.dataFileName)->default_value("/etc/vcalc.conf"), "Файл с базой данных")
20         ("log,l", po::value<std::string>(&params.logFileName)->default_value("/var/log/vcalc.log"), "Журнал событий")
21         ("port,p", po::value<int>(&params.port)->default_value(33333), "Порт");
22 }
```

5.7.3 Методы

5.7.3.1 getClientDataFile()

const std::string & Interface::getClientDataFile ( ) const

Возвращает имя файла с данными клиентов.

Возвращает

Имя файла с данными клиентов.

```
74                                      {
75     return params.dataFileName;
76 }
```

5.7.3.2 getDescription()

std::string Interface::getDescription ( )

Возвращает строковое описание параметров командной строки.

Возвращает

Строка с описанием параметров.

```
78                                  {
79     std::ostringstream ss;
80     ss « desc;
81     return ss.str();
82 }
```

5.7.3.3 getLogger()

Logger & Interface::getLogger ( )

Возвращает ссылку на объект Logger.

Возвращает

Ссылка на объект Logger.

```
84 { return logger; }
```

5.7.3.4 getParams()

const Params & Interface::getParams ( ) const

Возвращает константную ссылку на структуру параметров.

Возвращает

Константная ссылка на структуру Params.

```
68 { return params; }
```

5.7.3.5 getPort()

int Interface::getPort ( ) const

Возвращает номер порта.

Возвращает

Номер порта.

```
70                          {
71     return params.port;
72 }
```

5.7.3.6 Parser()

bool Interface::Parser (
int argc,
const char ** argv)

Парсит параметры командной строки.

Аргументы

argc	Количество аргументов.
argv	Массив аргументов.

Возвращает

true, если параметры успешно спарсены, false в противном случае (например, если был передан ключ –help).

```
24                                          {
25     po::store(po::parse_command_line(argc, argv, desc), vm);
26     if (argc == 1) {
27         std::cout « desc « std::endl;
28     }
29     if (vm.count("help")) {
30         std::cout « desc « std::endl;
31         return false;
32     }
33
34     po::notify(vm);
35     return true;
36 }
```

5.7.3.7 processCommands()

```
void Interface::processCommands (
                int argc,
                const char ** argv )
```

Обрабатывает параметры командной строки, инициализирует логгер и проверяет корректность параметров.

Аргументы

argc	Количество аргументов.
argv	Массив аргументов.

Исключения

std::runtime_error	При некорректном порте или отсутствии файла с данными клиентов.

```
38                                              {
39     if (!Parser(argc, argv)) {
40         exit(1);
41     }
42
43     logger.setLogFile(params.logFileName);
44     logger.open();
45
46     if (params.port < 0 || params.port > 65535) {
47         std::string error_message = "Неверный порт: " + std::to_string(params.port) + ". Порт должен быть в диапазоне от
       0 до 65535";
48         logger.log(CRITICAL, error_message);
49         throw std::runtime_error(error_message);
50     }
51
52     std::ifstream clientData(params.dataFileName);
53     if (!clientData.is_open()) {
54         std::string error_message = "Файл с базой клиентов " + params.dataFileName + " не найден.";
55         logger.log(CRITICAL, error_message);
56     }
```

```
57
58     if (clientData.peek() == std::ifstream::traits_type::eof()) {
59         std::string error_message = "Файл с базой клиентов " + params.dataFileName + " пустой.";
60         logger.log(CRITICAL, error_message);
61     }
62
63     logger.log(INFO, "Файл с базой клиентов: " + params.dataFileName);
64     logger.log(INFO, "Файл с журналом работы: " + logger.getLogFile());
65     logger.log(INFO, "Порт: " + std::to_string(params.port));
66 }
```

5.7.4 Данные класса

5.7.4.1 desc

po::options_description Interface::desc [private]

Описание параметров командной строки

5.7.4.2 logger

Logger Interface::logger [private]

Объект для логирования

5.7.4.3 params

Params Interface::params [private]

Структура для хранения параметров

5.7.4.4 vm

po::variables_map Interface::vm [private]

Хранилище для распарсенных параметров

Объявления и описания членов классов находятся в файлах:

- StartInterface.h
- StartInterface.cpp

5.8 Класс Logger

Класс для ведения логов в файл.

#include <Logger.h>

Открытые члены

- Logger (const std::string &filename)

  Конструктор класса Logger.
- void open ()

  Открывает файл для записи логов. Если файл не может быть открыт, выводится сообщение об ошибке и создается файл "vcalc.log".
- void close ()

  Закрывает файл для записи логов.
- ~Logger ()

  Деструктор класса Logger. Закрывает файл логов.
- void log (LogLevel level, const std::string &message)

  Записывает сообщение в лог-файл.
- void setLogFile (const std::string &filename)

  Устанавливает новое имя файла для логирования.
- std::string getLogFile () const

  Возвращает текущее имя файла для логирования.
- bool GetIsOpen () const

  Проверяет открыт ли файл для логирования.

Закрытые члены

- void handleError (const std::string &error_message)

  Обрабатывает ошибку, выводя сообщение на стандартный поток ошибок и выбрасывая исключение.
- std::string currentDateTime ()

  Возвращает текущую дату и время в формате YYYY-MM-DD HH:MM:SS.
- std::string logLevelToString (LogLevel level)

  Преобразует уровень логирования в строковое представление.

Закрытые данные

- std::string filename

  Имя файла для логирования
- bool is_open

  Флаг, указывающий открыт ли файл
- std::ofstream log_file

  Поток для записи в файл
- std::mutex mutex_

  Мьютекс для обеспечения потокобезопасности

5.8.1 Подробное описание

Класс для ведения логов в файл.

Обеспечивает потокобезопасное логирование с возможностью указать файл для логов и несколькими уровнями серьезности сообщений. При невозможности записи в указанный файл, пытается создать файл "vcalc.log" в директории запуска.

5.8.2 Конструктор(ы)

5.8.2.1 Logger()

```
Logger::Logger (
                const std::string & filename )
```

Конструктор класса Logger.

Аргументы

filename	Имя файла для логирования.

Исключения

std::invalid_argument	Если filename пустой.

```
14                                  {
15     setLogFile(filename);
16     is_open = false;
17 }
```

5.8.2.2 ~Logger()

```
Logger::~Logger ( )
```

Деструктор класса Logger. Закрывает файл логов.

```
27             {
28     close();
29 }
```

5.8.3 Методы

5.8.3.1 close()

void Logger::close ( )

Закрывает файл для записи логов.

```
55                {
56     std::lock_guard<std::mutex> lock(mutex_);
57     if (is_open) {
58         log_file.close();
59         is_open = false;
60     }
61 }
```

5.8.3.2 currentDateTime()

std::string Logger::currentDateTime ( ) [private]

Возвращает текущую дату и время в формате YYYY-MM-DD HH:MM:SS.

Возвращает

Текущая дата и время в формате строки.

```
84                                {
85     time_t now = time(0);
86     struct tm tstruct;
87     char buf[80];
88     localtime_r(&now, &tstruct);
89 
90     strftime(buf, sizeof(buf), "%Y-%m-%d %X", &tstruct);
91     return buf;
92 }
```

5.8.3.3 GetIsOpen()

bool Logger::GetIsOpen ( ) const

Проверяет открыт ли файл для логирования.

Возвращает

true если файл открыт, false иначе.

```
104 { return is_open; }
```

5.8.3.4 getLogFile()

std::string Logger::getLogFile ( ) const

Возвращает текущее имя файла для логирования.

Возвращает

Текущее имя файла.

```
106                                {
107     return filename;
108 }
```

5.8.3.5 handleError()

void Logger::handleError (
const std::string & error_message) [private]

Обрабатывает ошибку, выводя сообщение на стандартный поток ошибок и выбрасывая исключение.

Аргументы

error_message	Текст сообщения об ошибке.

```
79                                          {
80     std::cerr « error_message « std::endl;
81     throw std::runtime_error(error_message);
82 }
```

5.8.3.6 log()

```
void Logger::log (
                LogLevel level,
                const std::string & message )
```

Записывает сообщение в лог-файл.

Аргументы

level	Уровень серьезности сообщения.
message	Текст сообщения.

```
63                                              {
64     std::lock_guard<std::mutex> lock(mutex_);
65
66     if (!is_open) {
67         handleError("Файл журнала закрыт: " + filename);
68         return;
69     }
70
71     log_file « currentDateTime() « " [" « logLevelToString(level) « "] " « message « std::endl;
72     std::cout « currentDateTime() « " [" « logLevelToString(level) « "] " « message « std::endl;
73
74     if (level == CRITICAL) {
75         handleError("Критическая ошибка: " + message);
76     }
77 }
```

5.8.3.7 logLevelToString()

```
std::string Logger::logLevelToString (
                LogLevel level )    [private]
```

Преобразует уровень логирования в строковое представление.

Аргументы

level	Уровень логирования.

Возвращает

Строковое представление уровня логирования.

```
94                                          {
95     switch (level) {
```

```
96         case INFO: return "INFO";
97         case WARNING: return "WARNING";
98         case ERROR: return "ERROR";
99         case CRITICAL: return "CRITICAL";
100         default: return "UNKNOWN";
101     }
102 }
```

5.8.3.8 open()

void Logger::open ( )

Открывает файл для записи логов. Если файл не может быть открыт, выводится сообщение об ошибке и создается файл "vcalc.log".

```
31                 {
32     std::lock_guard<std::mutex> lock(mutex_);
33     if (!is_open) {
34         log_file.open(filename, std::ios::app);
35         if (!log_file.is_open()) {
36             std::string error_message = "Не удалось получить доступ к " + filename + ": " + strerror(errno) +
37                                         ". Попытка создания файла с логами в /tmp.";
38             std::cout « error_message « std::endl;
39 
40 
41             std::string tmp_filename = "/tmp/vcalc.log";
42             log_file.open(tmp_filename, std::ios::app);
43             if (!log_file.is_open()) {
44                 handleError("Не удалось открыть файл журнала: " + tmp_filename);
45             } else {
46                 is_open = true;
47                 this->filename = tmp_filename;
48             }
49         } else {
50             is_open = true;
51         }
52     }
53 }
```

5.8.3.9 setLogFile()

void Logger::setLogFile (
const std::string & filename)

Устанавливает новое имя файла для логирования.

Аргументы

filename	Новое имя файла.

Исключения

std::invalid_argument	Если filename пустой.

```
19                                     {
20     std::lock_guard<std::mutex> lock(mutex_);
21     if (filename.empty()) {
22         throw std::invalid_argument("Путь к журналу событий не может быть пустым");
23     }
24     this->filename = filename;
25 }
```

5.8.4 Данные класса

5.8.4.1 filename

std::string Logger::filename [private]

Имя файла для логирования

5.8.4.2 is_open

bool Logger::is_open [private]

Флаг, указывающий открыт ли файл

5.8.4.3 log_file

std::ofstream Logger::log_file [private]

Поток для записи в файл

5.8.4.4 mutex_

std::mutex Logger::mutex_ [private]

Мьютекс для обеспечения потокобезопасности

Объявления и описания членов классов находятся в файлах:

- Logger.h
- Logger.cpp

5.9 Структура Params

Структура для хранения параметров, полученных из командной строки.

#include <StartInterface.h>

Открытые атрибуты

- std::string dataFileName

  Имя файла с данными клиентов.
- std::string logFileName

  Имя файла журнала событий.
- int port

  Порт сервера.

5.9.1 Подробное описание

Структура для хранения параметров, полученных из командной строки.

5.9.2 Данные класса

5.9.2.1 dataFileName

std::string Params::dataFileName

Имя файла с данными клиентов.

5.9.2.2 logFileName

std::string Params::logFileName

Имя файла журнала событий.

5.9.2.3 port

int Params::port

Порт сервера.

Объявления и описания членов структуры находятся в файле:

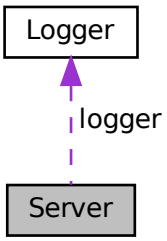
- StartInterface.h

5.10 Класс Server

Класс сетевого сервера.

#include <server.h>

Граф связей класса Server:



Открытые члены

- Server (int port, const std::string &data, Logger &logger)
  Конструктор класса Server.
- Server ()=delete
- ~Server ()
- void start ()
  Запускает сервер. В бесконечном цикле принимает соединения от клиентов и обрабатывает их.

Закрытые члены

- void handleClient (int client_socket)
  Обрабатывает соединение с клиентом.
- void listen_socket ()
  Начинает прослушивание сокета на наличие входящих соединений.

Закрытые данные

- int port
  Порт сервера
- std::string data
  Данные для отправки клиентам
- Logger & logger
  Объект для логирования
- int server_socket
  Дескриптор серверного сокета

5.10.1 Подробное описание

Класс сетевого сервера.

Принимает соединения от клиентов, обрабатывает их и отправляет данные. Использует класс Logger для логирования событий.

5.10.2 Конструктор(ы)

5.10.2.1 Server() [1/2]

```
Server::Server (
            int port,
            const std::string & data,
            Logger & logger )
```

Конструктор класса Server.

Аргументы

port	Порт сервера.
data	Данные для отправки клиентам.
logger	Объект для логирования.

Исключения

std::system_error	При ошибках создания или привязки сокета.
std::system_error	При указании неверного номера порта.

```
23     : port(port)
24     , data(data)
25     , logger(logger)
26 {
27     if (port < 1 || port > 65535) {
28         throw std::system_error(EINVAL, std::generic_category(), "Неправильный порт: " + to_string(port));
29     }
30     logger.log(INFO, "Создание сервера на порту " + std::to_string(port));
31 
32     server_socket = socket(AF_INET, SOCK_STREAM, 0);
33     if(server_socket == -1) {
34         std::string error_message = strerror(errno);
35         std::string exception_message = "Ошибка создания сокета: " + error_message;
36         logger.log(CRITICAL, exception_message);
37         throw std::system_error(errno, std::generic_category(), exception_message);
38     }
39 
40     sockaddr_in server_addr{};
41     server_addr.sin_family = AF_INET;
42     server_addr.sin_addr.s_addr = INADDR_ANY;
43     server_addr.sin_port = htons(port);
44 
45     if(bind(server_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) == -1) {
46         std::string error_message = strerror(errno);
47         std::string exception_message = "Ошибка привязки сокета: " + error_message;
48         logger.log(CRITICAL, exception_message);
49         throw std::system_error(errno, std::generic_category(), exception_message);
50     }
51 
52     listen_socket();
53 }
```

5.10.2.2 Server() [2/2]

Server::Server ( ) [delete]

5.10.2.3 ~Server()

Server::~Server ( ) [inline]
57 {}

5.10.3 Методы

5.10.3.1 handleClient()

void Server::handleClient (
int client_socket) [private]

Обрабатывает соединение с клиентом.

Аргументы

client_socket	Дескриптор сокета клиента.

5.10.3.2 listen_socket()

void Server::listen_socket ( ) [private]

Начинает прослушивание сокета на наличие входящих соединений.

```
16 {
17     if(listen(server_socket, 5) == -1) {
18         throw std::system_error(errno, std::generic_category(), "Ошибка прослушивания сокета");
19     }
20 }
```

5.10.3.3 start()

void Server::start ( )

Запускает сервер. В бесконечном цикле принимает соединения от клиентов и обрабатывает их.

Исключения

std::system_error	При ошибках приема соединения.

```
56 {
57     while(true) {
58         sockaddr_in client_addr{};
59         socklen_t client_addr_size = sizeof(client_addr);
60 
61         int client_socket = accept(server_socket, (struct sockaddr*)&client_addr, &client_addr_size);
62         if(client_socket == -1) {
63             std::string error_message = strerror(errno);
64             std::string exception_message = "Ошибка при приёме соединения: " + error_message;
65             logger.log(CRITICAL, exception_message);
66             throw std::system_error(errno, std::generic_category(), exception_message);
67         }
68         std::string client_ip = inet_ntoa(client_addr.sin_addr);
69         logger.log(INFO, "Новое подключение от " + client_ip);
70         Communicator communicator(client_socket, data, logger);
71         communicator.handleClient();
72         close(client_socket);
73     }
74 }
```

5.10.4 Данные класса

5.10.4.1 data

std::string Server::data [private]

Данные для отправки клиентам

5.10.4.2 logger

Logger& Server::logger [private]

Объект для логирования

5.10.4.3 port

int Server::port [private]

Порт сервера

5.10.4.4 server_socket

int Server::server_socket [private]

Дескриптор серверного сокета

Объявления и описания членов классов находятся в файлах:

- server.h
- server.cpp

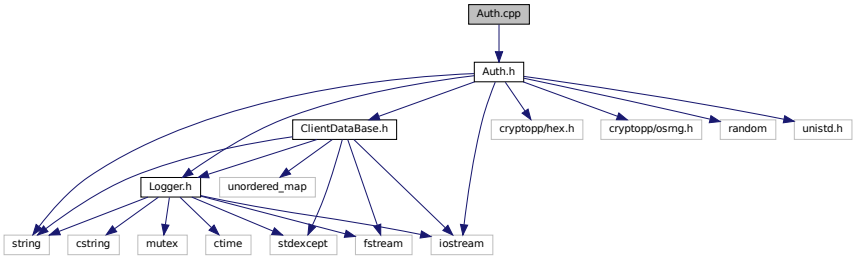
Глава 6

Файлы

6.1 Файл Auth.cpp

Файл содержит реализацию класса Authenticator для аутентификации пользователей.

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



6.1.1 Подробное описание

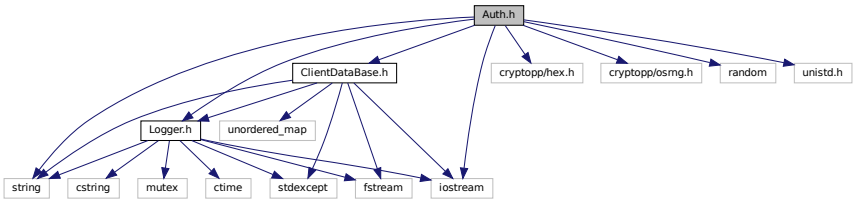
Файл содержит реализацию класса Authenticator для аутентификации пользователей.

6.2 Файл Auth.h

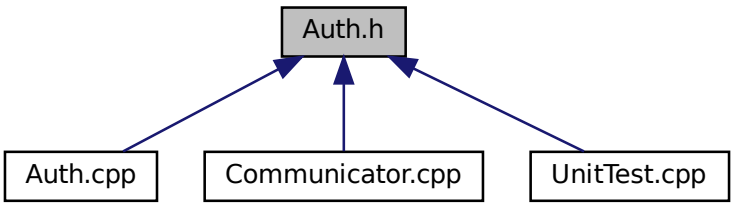
Файл содержит класс Authenticator для аутентификации пользователей.

#include "ClientDataBase.h"
#include "Logger.h"
#include <cryptopp/hex.h>
#include <cryptopp/osrng.h>
#include <iostream>
#include <random>
#include <string>

#include <unistd.h>
Граф включаемых заголовочных файлов для Auth.h:



Граф файлов, в которые включается этот файл:



Классы

- class Authenticator

  Обрабатывает аутентификацию пользователей по базе данных.

6.2.1 Подробное описание

Файл содержит класс Authenticator для аутентификации пользователей.

Автор

Грачев В.В.

Версия

1.0

Дата

18.11.2024

Авторство

ИБСТ ПГУ

6.3 Auth.h

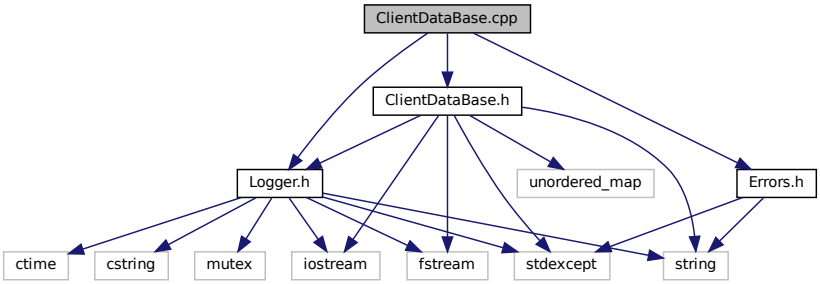
См. документацию.

```
1
9 #pragma once
10
11 #include "ClientDataBase.h"
12 #include "Logger.h"
13
14 #include <cryptopp/hex.h>
15 #include <cryptopp/osrng.h>
16 #include <iostream>
17 #include <random>
18 #include <string>
19 #include <unistd.h>
20 using namespace std;
21
22 namespace CPP = CryptoPP;
23
28 class Authenticator {
29 private:
30     ClientDataBase db;
31     Logger& logger;
32
33 public:
40     Authenticator(const std::string& db_filename, Logger& logger);
41
48     bool isLoginExists(const std::string& login);
49
55     std::string generateSalt();
56
64     std::string hashPassword(const std::string& salt_16, const std::string& password);
65
74     bool compareHashes(const std::string& clientHash, const std::string& salt, const std::string& login);
75
83     bool verifyPassword(const std::string& login, const std::string& password, const std::string& salt);
84 };
```

6.4 Файл ClientDataBase.cpp

Реализация класса ClientDataBase.

#include "ClientDataBase.h"
#include "Logger.h"
#include "Errors.h"
Граф включаемых заголовочных файлов для ClientDataBase.cpp:



6.4.1 Подробное описание

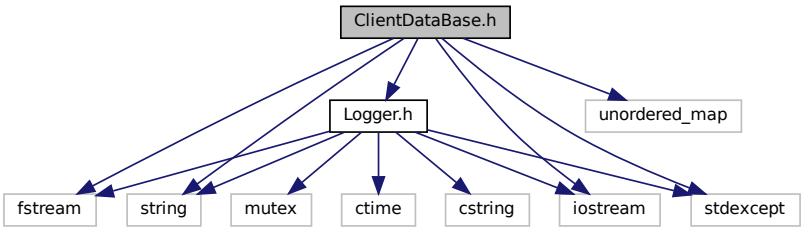
Реализация класса ClientDataBase.

6.5 Файл ClientDataBase.h

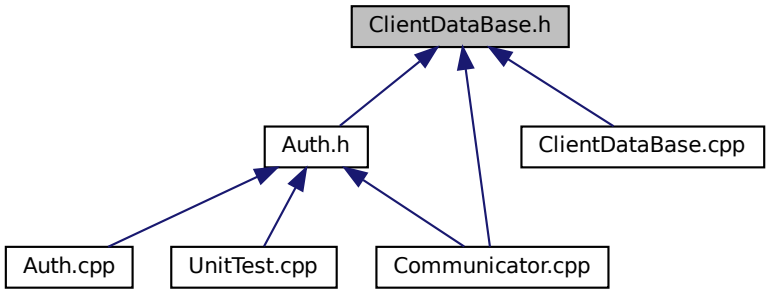
Заголовочный файл класса ClientDataBase, реализующего работу с базой данных клиентов.

```
#include "Logger.h"
#include <fstream>
#include <iostream>
#include <stdexcept>
#include <string>
#include <unordered_map>
```

Граф включаемых заголовочных файлов для ClientDataBase.h:



Граф файлов, в которые включается этот файл:



Классы

- class ClientDataBase

  Класс для работы с базой данных клиентов, хранящей логины и пароли.

6.5.1 Подробное описание

Заголовочный файл класса ClientDataBase, реализующего работу с базой данных клиентов.

Автор

Грачев В.В.

Версия

1.0

Дата

18.11.2024

Авторство

ИБСТ ПГУ

6.6 ClientDataBase.h

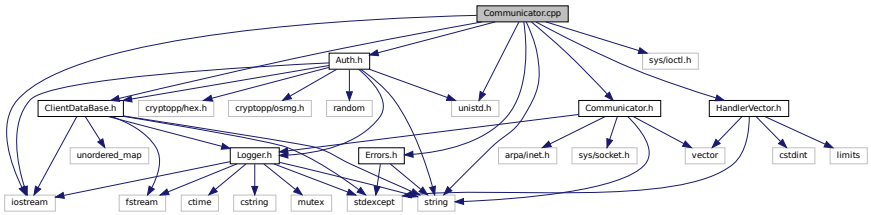
См. документацию.

```
1 #pragma once
10 #include "Logger.h"
11 #include <fstream>
12 #include <iostream>
13 #include <stdexcept>
14 #include <string>
15 #include <unordered_map>
16
23 class ClientDataBase
24 {
25 private:
33     void loadDatabase(const std::string& filename);
37     std::unordered_map<std::string, std::string> database;
38 public:
46     ClientDataBase(const std::string& filename);
52     bool isLoginExists(const std::string& login);
58     std::string& operator[](const std::string& login);
64     const std::string& operator[](const std::string& login) const;
65 };
```

6.7 Файл Communicator.cpp

Реализация класса Communicator, отвечающего за коммуникацию с клиентами.

#include "Auth.h"
#include "ClientDataBase.h"
#include "Communicator.h"
#include "HandlerVector.h"
#include "Errors.h"
#include <iostream>
#include <string>
#include <sys/ioctl.h>
#include <unistd.h>
Граф включаемых заголовочных файлов для Communicator.cpp:



6.7.1 Подробное описание

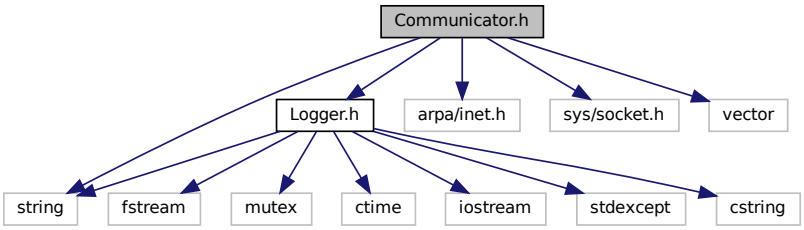
Реализация класса Communicator, отвечающего за коммуникацию с клиентами.

6.8 Файл Communicator.h

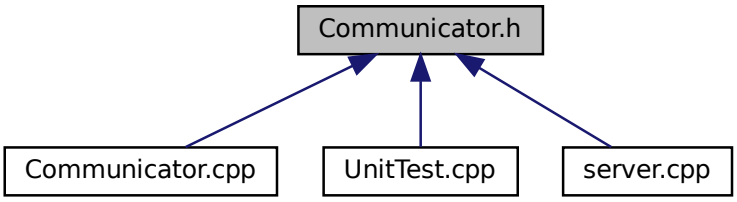
Этот файл содержит определение класса Communicator, отвечающего за коммуникацию с клиентами.

```
#include "Logger.h"
#include <arpa/inet.h>
#include <string>
#include <sys/socket.h>
#include <vector>
```

Граф включаемых заголовочных файлов для Communicator.h:



Граф файлов, в которые включается этот файл:



Классы

- class Communicator

  Этот класс отвечает за обработку коммуникации с клиентом. Он принимает векторы данных от клиента, обрабатывает их и отправляет результат обратно.

6.8.1 Подробное описание

Этот файл содержит определение класса Communicator, отвечающего за коммуникацию с клиентами.

Автор

Грачев В.В.

Версия

1.0

Дата

18.11.2024

Авторство

ИБСТ ПГУ

6.9 Communicator.h

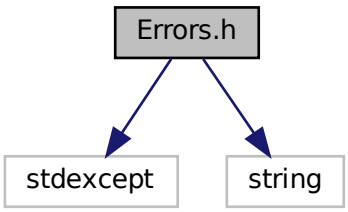
См. документацию.

```
1
9 #pragma once
10 #include "Logger.h"
11
12 #include <arpa/inet.h>
13 #include <string>
14 #include <sys/socket.h>
15 #include <vector>
16
21 class Communicator {
22
23 private:
24     std::string dataFile;
25     Logger& logger;
26     int client_socket;
27
32     void sendResponse(const std::string& response);
33
34
35 public:
42     Communicator(int client_socket, const std::string& dataFile, Logger& logger);
43
47     void handleClient();
48
52     ~Communicator();
53
57     void processVectors();
58
63     void sendResultToClient(int32_t resultValue);
64 };
```

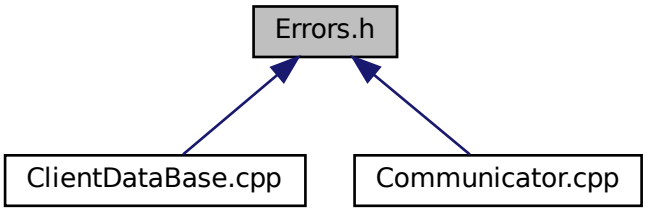
6.10 Файл Errors.h

Этот файл содержит определение класса Communicator, отвечающего за коммуникацию с клиентами.

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



Граф файлов, в которые включается этот файл:



Классы

- class DatabaseException

  Исключение, возникающее при ошибках работы с базой данных.
- class CommunicatorException

  Исключение, возникающее при ошибках в процессе коммуникации.

6.10.1 Подробное описание

Этот файл содержит определение класса Communicator, отвечающего за коммуникацию с клиентами.

Автор

Грачев В.В.

Версия

1.0

Дата

18.11.2024

Авторство

ИБСТ ПГУ

6.11 Errors.h

См. документацию.

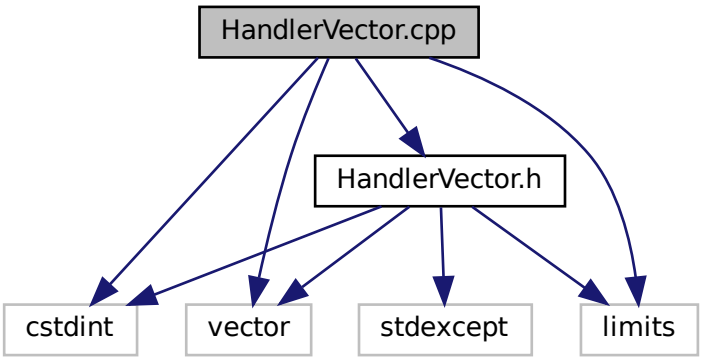
```
1
9 #pragma once
10 #include <stdexcept>
11 #include <string>
12
17 class DatabaseException : public std::invalid_argument {
18 public:
23     explicit DatabaseException(const std::string& what_arg)
24         : std::invalid_argument(what_arg) {}
25
30     explicit DatabaseException(const char* what_arg)
31         : std::invalid_argument(what_arg) {}
32 };
33
38 class CommunicatorException : public std::runtime_error {
39 public:
44     explicit CommunicatorException(const std::string& message)
45         : std::runtime_error(message) {}
46 };
```

6.12 Файл HandlerVector.cpp

Реализация класса HandlerVector.

#include "HandlerVector.h"
#include <vector>
#include <limits>

#include <cstdint>
Граф включаемых заголовочных файлов для HandlerVector.cpp:



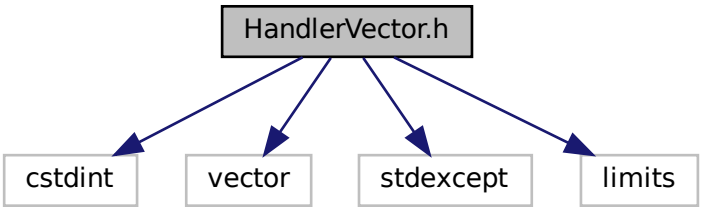
6.12.1 Подробное описание

Реализация класса HandlerVector.

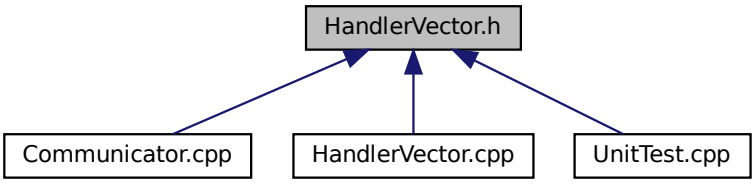
6.13 Файл HandlerVector.h

Заголовочный файл класса HandlerVector.

#include <cstdint>
#include <vector>
#include <stdexcept>
#include <limits>
Граф включаемых заголовочных файлов для HandlerVector.h:



Граф файлов, в которые включается этот файл:



Классы

- class HandlerVector

  Класс для обработки целочисленных векторов.

6.13.1 Подробное описание

Заголовочный файл класса HandlerVector.

Автор

Грачев В.В.

Версия

1.0

Дата

18.11.2024

Авторство

ИБСТ ПГУ

6.14 HandlerVector.h

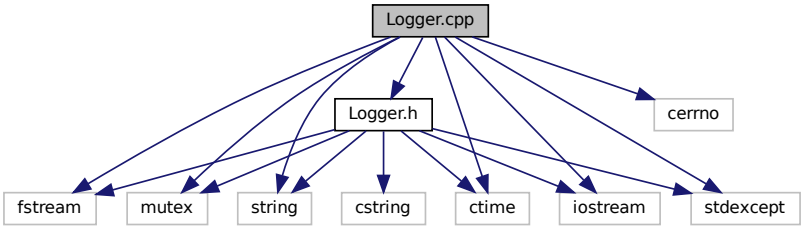
См. документацию.

```
1
9 #include <cstdint>
10 #include <vector>
11 #include <stdexcept>
12 #include <limits>
13
21 class HandlerVector {
22 public:
29     static int32_t processVector(const std::vector<int32_t>& vectorValues);
30 };
```

6.15 Файл Logger.cpp

Реализация класса Logger.

#include "Logger.h"
#include <stdexcept>
#include <iostream>
#include <mutex>
#include <fstream>
#include <string>
#include <cerrno>
#include <ctime>
Граф включаемых заголовочных файлов для Logger.cpp:



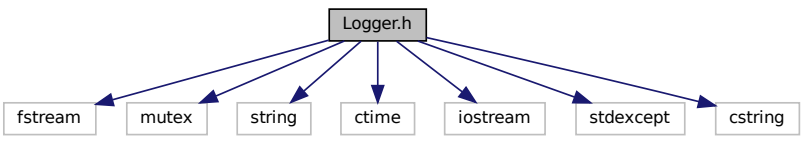
6.15.1 Подробное описание

Реализация класса Logger.

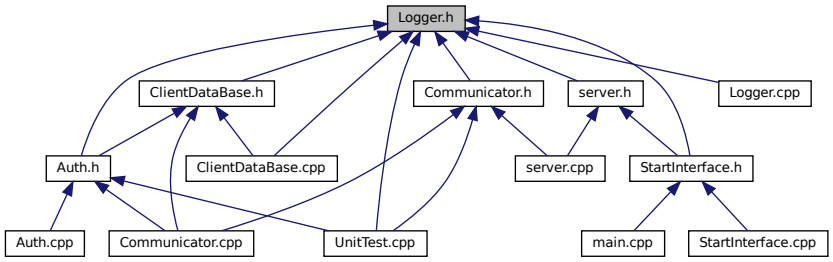
6.16 Файл Logger.h

Заголовочный файл класса Logger для ведения логов.

#include <fstream>
#include <mutex>
#include <string>
#include <ctime>
#include <iostream>
#include <stdexcept>
#include <cstring>
Граф включаемых заголовочных файлов для Logger.h:



Граф файлов, в которые включается этот файл:



Классы

- class Logger

  Класс для ведения логов в файл.

Перечисления

- enum LogLevel { INFO , WARNING , ERROR , CRITICAL }

  Уровни логирования.

6.16.1 Подробное описание

Заголовочный файл класса Logger для ведения логов.

Автор

Грачев В.В.

Версия

1.0

Дата

18.11.2024

Авторство

ИБСТ ПГУ

6.16.2 Перечисления

6.16.2.1 LogLevel

enum LogLevel

Уровни логирования.

Элементы перечислений

INFO	Информационное сообщение
WARNING	Предупреждение
ERROR	Ошибка
CRITICAL	Критическая ошибка

```
22             {
23     INFO,
24     WARNING,
25     ERROR,
26     CRITICAL
27 };
```

6.17 Logger.h

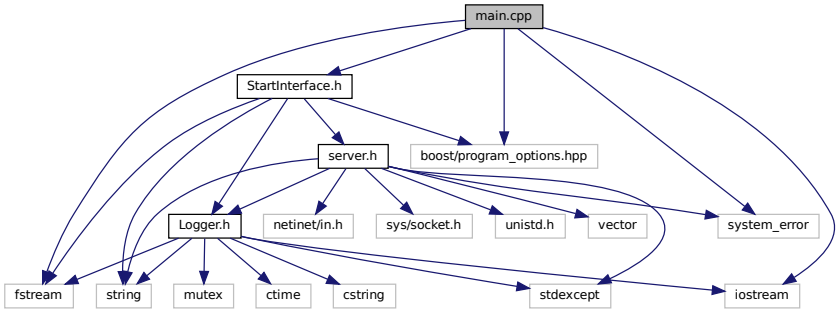
См. документацию.

```
1
9  #pragma once
10 #include <fstream>
11 #include <mutex>
12 #include <string>
13 #include <ctime>
14 #include <iostream>
15 #include <stdexcept>
16 #include <cstring>
17
22 enum LogLevel {
23     INFO,
24     WARNING,
25     ERROR,
26     CRITICAL
27 };
28
36 class Logger {
37 public:
43     Logger(const std::string& filename);
48     void open();
52     void close();
56     ~Logger();
62     void log(LogLevel level, const std::string& message);
68     void setLogFile(const std::string& filename);
73     std::string getLogFile() const;
78     bool GetIsOpen() const;
79 private:
80     std::string filename;
81     bool is_open;
86     void handleError(const std::string& error_message);
91     std::string currentDateTime();
97     std::string logLevelToString(LogLevel level);
98     std::ofstream log_file;
99     std::mutex mutex_;
100 };
```

6.18 Файл main.cpp

```
#include "StartInterface.h"
#include <boost/program_options.hpp>
#include <fstream>
#include <iostream>
```

#include <system_error>
Граф включаемых заголовочных файлов для main.cpp:



Функции

- int main (int argc, const char **argv)

6.18.1 Функции

6.18.1.1 main()

```
int main (
                int argc,
                const char ** argv )
10                              {
11    try{
12    Interface UI;
13    UI.processCommands(argc, argv);
14    Server server(UI.getPort(), UI.getClientDataFile(), UI.getLogger());
15    server.start();
16
17    } catch (const exception& e) {
18       return 1;
19    }
20
21    return 0;
22 }
```

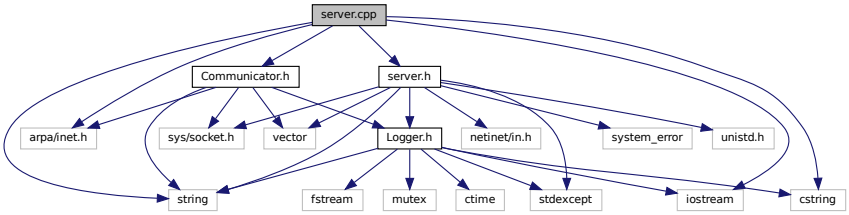
6.19 Файл README.md

6.20 Файл server.cpp

Реализация класса Server.

#include "server.h"
#include "Communicator.h"
#include <arpa/inet.h>
#include <cstring>

#include <iostream>
#include <string>
Граф включаемых заголовочных файлов для server.cpp:



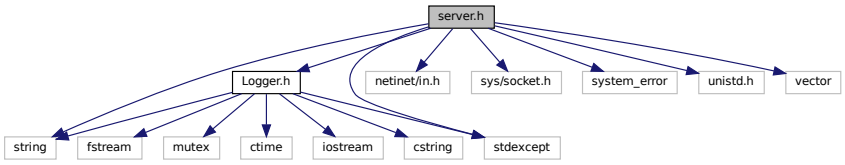
6.20.1 Подробное описание

Реализация класса Server.

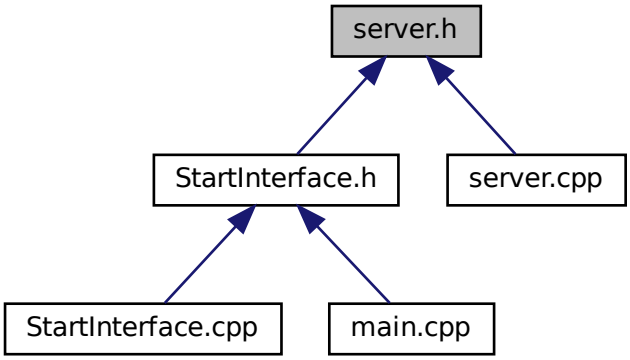
6.21 Файл server.h

Заголовочный файл класса Server.

#include "Logger.h"
#include <netinet/in.h>
#include <stdexcept>
#include <string>
#include <sys/socket.h>
#include <system_error>
#include <unistd.h>
#include <vector>
Граф включаемых заголовочных файлов для server.h:



Граф файлов, в которые включается этот файл:



Классы

- class Server

  Класс сетевого сервера.

6.21.1 Подробное описание

Заголовочный файл класса Server.

Автор

Грачев В.В.

Версия

1.0

Дата

18.11.2024

Авторство

ИБСТ ПГУ

6.22 server.h

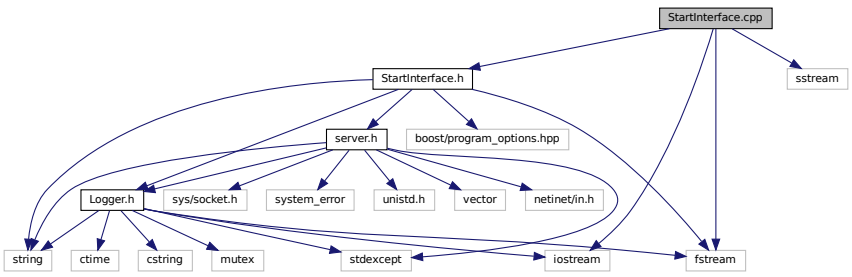
См. документацию.

```
1
9 #pragma once
10
11 #include "Logger.h"
12
13 #include <netinet/in.h>
14 #include <stdexcept>
15 #include <string>
16 #include <sys/socket.h>
17 #include <system_error>
18 #include <unistd.h>
19 #include <vector>
20
21 using namespace std;
22
30 class Server
31 {
32 private:
33     int port;
34     std::string data;
35     Logger& logger;
36     int server_socket;
41     void handleClient(int client_socket);
45     void listen_socket();
46 public:
55     Server(int port, const std::string& data, Logger& logger);
56     Server() = delete;
57     ~Server() {}
63     void start();
64 };
```

6.23 Файл StartInterface.cpp

Реализация класса Interface для обработки параметров командной строки.

#include "StartInterface.h"
#include <sstream>
#include <iostream>
#include <fstream>
Граф включаемых заголовочных файлов для StartInterface.cpp:



6.23.1 Подробное описание

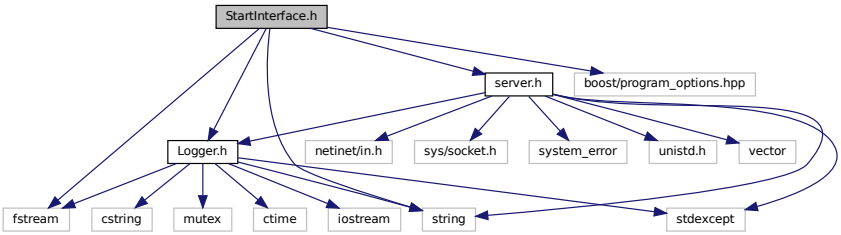
Реализация класса Interface для обработки параметров командной строки.

6.24 Файл StartInterface.h

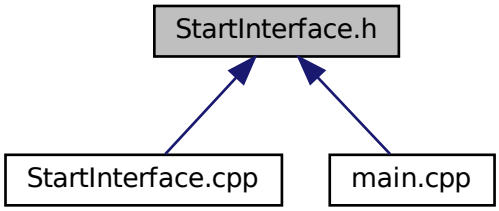
Заголовочный файл класса Interface для обработки параметров командной строки.

```
#include <string>
#include <fstream>
#include <boost/program_options.hpp>
#include "Logger.h"
#include "server.h"
```

Граф включаемых заголовочных файлов для StartInterface.h:



Граф файлов, в которые включается этот файл:



Классы

- struct Params

  Структура для хранения параметров, полученных из командной строки.
- class Interface

  Класс для обработки параметров командной строки и инициализации сервера.

6.24.1 Подробное описание

Заголовочный файл класса Interface для обработки параметров командной строки.

Автор

Грачев В.В.

Версия

1.0

Дата

18.11.2024

Авторство

ИБСТ ПГУ

6.25 StartInterface.h

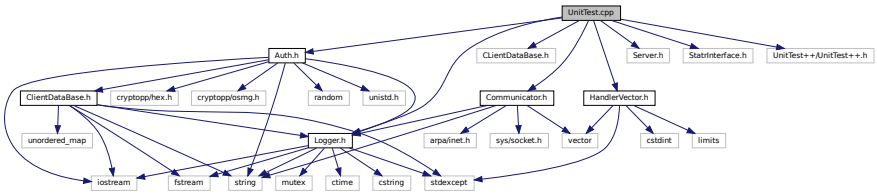
См. документацию.

```
1
9 #pragma once
10
11 #include <string>
12 #include <fstream>
13 #include <boost/program_options.hpp>
14 #include "Logger.h"
15 #include "server.h"
16 namespace po = boost::program_options;
17
18
23 struct Params {
24     std::string dataFileName;
25     std::string logFileName;
26     int port;
27 };
28
35 class Interface {
36 public:
41     Interface();
48     bool Parser(int argc, const char** argv);
55     void processCommands(int argc, const char** argv);
60     const Params& getParams() const;
65     Logger& getLogger();
70     int getPort() const;
75     const std::string& getClientDataFile() const;
80     string getDescription();
81 private:
82     Params params;
83     Logger logger;
84     po::options_description desc;
85     po::variables_map vm;
86 };
```

6.26 Файл UnitTest.cpp

```
#include "Auth.h"
#include "CLientDataBase.h"
#include "Communicator.h"
#include "HandlerVector.h"
#include "Logger.h"
#include "Server.h"
#include "StatrInterface.h"
```

#include <UnitTest++/UnitTest++.h>
Граф включаемых заголовочных файлов для UnitTest.cpp:



Функции

- SUITE (HelpTest)

6.26.1 Функции

6.26.1.1 SUITE()

```
SUITE (
                HelpTest   )
12            {
13 TEST(Help) {
14  Interface UI;
15  int port = 33333;
16  const char* argv[] = {"test", "-h", "-p", port, nullptr};
17  int argc = sizeof argv / sizeof nullptr - 1;
18  REQUIRE CHECK(!iface.Parser(argc, argv));
19  CHECK(!iface.getDescription().empty());
20  }
21 }
```

Предметный указатель