

Программная реализация сетевого сервера

Создано системой Doxygen 1.9.4

1 Сетевой сервер обработки данных	1
1.1 Использование	1
2 Иерархический список классов	3
2.1 Иерархия классов	3
3 Алфавитный указатель классов	5
3.1 Классы	5
4 Список файлов	7
4.1 Файлы	7
5 Классы	9
5.1 Класс auth_error	9
5.1.1 Подробное описание	10
5.1.2 Конструктор(ы)	10
5.1.2.1 auth_error()	10
5.2 Класс Authenticator	11
5.2.1 Подробное описание	11
5.2.2 Методы	11
5.2.2.1 isValidHex()	11
5.2.2.2 verify()	12
5.2.3 Данные класса	13
5.2.3.1 SALT16_LENGTH	13
5.2.3.2 SHA1_HEX_LENGTH	14
5.3 Класс DataProcessor	14
5.3.1 Подробное описание	14
5.3.2 Методы	14
5.3.2.1 calculateAverage()	14
5.4 Класс Interface	16
5.4.1 Подробное описание	16
5.4.2 Конструктор(ы)	17
5.4.2.1 Interface()	17
5.4.3 Методы	17
5.4.3.1 getParams()	17
5.4.3.2 Parser()	17
5.4.3.3 printHelp()	18
5.4.4 Данные класса	18
5.4.4.1 desc	18
5.4.4.2 params	19
5.4.4.3 vm	19
5.5 Класс Logger	19
5.5.1 Подробное описание	19
5.5.2 Методы	20
5.5.2.1 init()	20

5.5.2.2 logError()	20
5.5.2.3 logInfo()	20
5.5.3 Данные класса	21
5.5.3.1 logPath	21
5.6 Структура Params	21
5.6.1 Подробное описание	21
5.6.2 Данные класса	22
5.6.2.1 dbFile	22
5.6.2.2 logFile	22
5.6.2.3 port	22
5.7 Класс Server	22
5.7.1 Подробное описание	24
5.7.2 Конструктор(ы)	24
5.7.2.1 Server()	24
5.7.2.2 ~Server()	24
5.7.3 Методы	25
5.7.3.1 handleClient()	25
5.7.3.2 processVectors()	25
5.7.3.3 readTextMessage()	26
5.7.3.4 run()	27
5.7.3.5 sendError()	28
5.7.3.6 startListening()	28
5.7.3.7 validatePort()	28
5.7.4 Данные класса	29
5.7.4.1 authenticator	29
5.7.4.2 foreign_addr	29
5.7.4.3 listen_sock	29
5.7.4.4 logger	30
5.7.4.5 MAX_PORT	30
5.7.4.6 MIN_PORT	30
5.7.4.7 port	30
5.7.4.8 processor	30
5.7.4.9 self_addr	31
5.7.4.10 userDb	31
5.8 Класс server_error	31
5.8.1 Подробное описание	32
5.8.2 Конструктор(ы)	32
5.8.2.1 server_error()	32
5.9 Класс UserDatabase	33
5.9.1 Подробное описание	33
5.9.2 Методы	33
5.9.2.1 getPassword()	33
5.9.2.2 load()	34

5.9.3 Данные класса	35
5.9.3.1 users	35
5.10 Класс vector_error	35
5.10.1 Подробное описание	36
5.10.2 Конструктор(ы)	36
5.10.2.1 vector_error()	36
6 Файлы	39
6.1 Файл include/Authenticator.h	39
6.1.1 Подробное описание	40
6.2 Authenticator.h	40
6.3 Файл include/DataProcessor.h	40
6.3.1 Подробное описание	41
6.4 DataProcessor.h	41
6.5 Файл include/Interface.h	41
6.5.1 Подробное описание	42
6.6 Interface.h	43
6.7 Файл include/Logger.h	43
6.7.1 Подробное описание	44
6.8 Logger.h	44
6.9 Файл include/Server.h	44
6.9.1 Подробное описание	45
6.10 Server.h	45
6.11 Файл include/UserDatabase.h	46
6.11.1 Подробное описание	47
6.12 UserDatabase.h	47
6.13 Файл src/Authenticator.cpp	47
6.13.1 Подробное описание	48
6.14 Authenticator.cpp	48
6.15 Файл src/DataProcessor.cpp	49
6.15.1 Подробное описание	49
6.16 DataProcessor.cpp	50
6.17 Файл src/Interface.cpp	50
6.17.1 Подробное описание	50
6.18 Interface.cpp	51
6.19 Файл src/Logger.cpp	51
6.19.1 Подробное описание	51
6.20 Logger.cpp	52
6.21 Файл src/main.cpp	52
6.21.1 Подробное описание	53
6.21.2 Функции	53
6.21.2.1 isValidPort()	53
6.21.2.2 main()	54

6.22 main.cpp	56
6.23 Файл src/Server.cpp	56
6.23.1 Подробное описание	57
6.23.2 Макросы	57
6.23.2.1 AUTH_DATA_LENGTH	57
6.23.2.2 BUflen	57
6.23.2.3 QLEN	58
6.24 Server.cpp	58
6.25 Файл src/UserDatabase.cpp	60
6.25.1 Подробное описание	60
6.26 UserDatabase.cpp	61
Предметный указатель	63

Глава 1

Сетевой сервер обработки данных

Программа реализует сетевой сервер для аутентификации клиентов и вычисления среднего арифметического значений векторов.

1.1 Использование

Запуск сервера: `./server [-file FILE] [-log FILE] [-port PORT]`

Вывод справки: `./server -help ./server -h`

Глава 2

Иерархический список классов

2.1 Иерархия классов

Иерархия классов.

Authenticator	11
DataProcessor	14
Interface	16
Logger	19
Params	21
std::runtime_error	
server_error	31
auth_error	9
vector_error	35
Server	22
UserDatabase	33

Глава 3

Алфавитный указатель классов

3.1 Классы

Классы с их кратким описанием.

auth_error	Исключение для ошибок аутентификации	9
Authenticator	Предварительное объявление класса Logger	11
DataProcessor	Предварительное объявление класса Logger	14
Interface	Класс для обработки параметров командной строки	16
Logger	Класс для ведения журнала работы сервера	19
Params	Структура для хранения параметров сервера	21
Server	Основной класс сервера	22
server_error	Базовый класс исключений сервера	31
UserDatabase	Предварительное объявление класса Logger	33
vector_error	Исключение для ошибок обработки векторов	35

Глава 4

Список файлов

4.1 Файлы

Полный список документированных файлов.

include/Authenticator.h	Заголовочный файл модуля <code>Authenticator</code> - аутентификация пользователей	39
include/DataProcessor.h	Заголовочный файл модуля <code>DataProcessor</code> - обработка числовых данных	40
include/Interface.h	Заголовочный файл модуля <code>Interface</code> - обработка параметров командной строки . .	41
include/Logger.h	Заголовочный файл модуля <code>Logger</code> - журналирование работы сервера	43
include/Server.h	Заголовочный файл модуля <code>Server</code> - основной серверный модуль	44
include/UserDatabase.h	Заголовочный файл модуля <code>UserDatabase</code> - работа с базой пользователей	46
src/Authenticator.cpp	Реализация класса <code>Authenticator</code> для аутентификации пользователей	47
src/DataProcessor.cpp	Реализация класса <code>DataProcessor</code> для обработки числовых данных	49
src/Interface.cpp	Реализация класса <code>Interface</code> для обработки параметров командной строки	50
src/Logger.cpp	Реализация класса <code>Logger</code> для ведения журнала работы сервера	51
src/main.cpp	Точка входа серверной программы	52
src/Server.cpp	Реализация основного серверного модуля	56
src/UserDatabase.cpp	Реализация класса <code>UserDatabase</code> для работы с базой данных пользователей	60

Глава 5

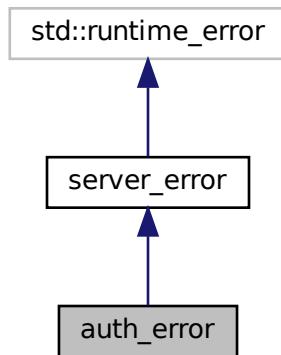
Классы

5.1 Класс auth_error

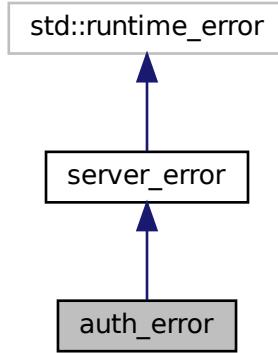
Исключение для ошибок аутентификации

```
#include <Server.h>
```

Граф наследования:auth_error:



Граф связей класса auth_error:



Открытые члены

- [auth_error](#) (const std::string &message)
Конструктор исключения аутентификации

5.1.1 Подробное описание

Исключение для ошибок аутентификации

См. определение в файле [Server.h](#) строка 35

5.1.2 Конструктор(ы)

5.1.2.1 auth_error()

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

Конструктор исключения аутентификации

Аргументы

message	Сообщение об ошибке аутентификации
---------	------------------------------------

См. определение в файле [Server.h](#) строка 41

Объявления и описания членов класса находятся в файле:

- include/[Server.h](#)

5.2 Класс Authenticator

Предварительное объявление класса [Logger](#).

```
#include <Authenticator.h>
```

Открытые члены

- bool [verify](#) (const std::string &login, const std::string &salt_hash_client, [UserDatabase](#) &db, [Logger](#) &logger) const
Проверка аутентификационных данных

Закрытые члены

- bool [isValidHex](#) (const std::string &str) const
Проверка строки на соответствие шестнадцатеричному формату

Закрытые статические данные

- static const int [SALT16_LENGTH](#) = 16
Длина строки SALT в hex-формате
- static const int [SHA1_HEX_LENGTH](#) = 40
Длина хеша SHA-1 в hex-формате

5.2.1 Подробное описание

Предварительное объявление класса [Logger](#).

Класс для аутентификации пользователей

Реализует аутентификацию с использованием хеш-функции SHA-1 и соли, формируемой клиентом

См. определение в файле [Authenticator.h](#) строка 16

5.2.2 Методы

5.2.2.1 [isValidHex\(\)](#)

```
bool Authenticator::isValidHex (  
    const std::string & str ) const [private]
```

Проверка строки на соответствие шестнадцатеричному формату

Проверка строки на соответствие шестнадцатеричному формату

Аргументы

str	Проверяемая строка
-----	--------------------

Возвращает

true - строка содержит только hex-символы, false - строка содержит недопустимые символы

Аргументы

str	Проверяемая строка
logger	Ссылка на журнал для записи ошибок

Возвращает

true - строка содержит только hex-символы, false - строка содержит недопустимые символы

Допустимые символы: 0-9, A-F, a-f

См. определение в файле [Authenticator.cpp](#) строка [26](#)

5.2.2.2 verify()

```
bool Authenticator::verify (
    const std::string & login,
    const std::string & salt_hash_client,
    UserDatabase & db,
    Logger & logger ) const
```

Проверка аутентификационных данных

Проверка аутентификационных данных пользователя

Аргументы

login	Логин пользователя
salt_hash_client	Строка формата SALT16 + HASH (56 символов)
db	Ссылка на базу данных пользователей
logger	Ссылка на журнал для записи событий

Возвращает

true - аутентификация успешна, false - аутентификация не пройдена

Аргументы

login	Логин пользователя
salt_hash_client	Строка формата SALT16 + HASH (56 символов)
db	Ссылка на базу данных пользователей
logger	Ссылка на журнал для записи ошибок

Возвращает

true - аутентификация успешна, false - аутентификация не пройдена

Алгоритм проверки:

1. Проверка длины сообщения ($16+40=56$ символов)
2. Валидация hex-формата SALT и HASH
3. Поиск пользователя в базе данных
4. Вычисление хеша SHA-1 от (SALT + PASSWORD)
5. Сравнение вычисленного хеша с полученным

Заметки

Используется схема: $\text{HASH} = \text{SHA1}(\text{SALT} \parallel \text{PASSWORD})$

Предупреждения

При несовпадении хешей в журнал записывается ошибка аутентификации

См. определение в файле [Authenticator.cpp](#) строка 49

5.2.3 Данные класса

5.2.3.1 SALT16_LENGTH

```
const int Authenticator::SALT16_LENGTH = 16 [static], [private]
```

Длина строки SALT в hex-формате

См. определение в файле [Authenticator.h](#) строка 39

5.2.3.2 SHA1_HEX_LENGTH

const int Authenticator::SHA1_HEX_LENGTH = 40 [static], [private]

Длина хеша SHA-1 в hex-формате

См. определение в файле [Authenticator.h](#) строка 40

Объявления и описания членов классов находятся в файлах:

- include/[Authenticator.h](#)
- src/[Authenticator.cpp](#)

5.3 Класс DataProcessor

Предварительное объявление класса [Logger](#).

#include <DataProcessor.h>

Открытые члены

- int32_t [calculateAverage](#) (const std::vector< int32_t > &vector, [Logger](#) &logger)

Вычисление среднего арифметического значений вектора

5.3.1 Подробное описание

Предварительное объявление класса [Logger](#).

Класс для обработки числовых данных

Выполняет вычисления над векторами целых чисел

См. определение в файле [DataProcessor.h](#) строка 16

5.3.2 Методы

5.3.2.1 calculateAverage()

```
int32_t DataProcessor::calculateAverage (
    const std::vector< int32_t > & vector,
    Logger & logger )
```

Вычисление среднего арифметического значений вектора

Аргументы

vector	Вектор целых чисел для обработки
logger	Ссылка на объект журнала для записи ошибок

Возвращает

Среднее арифметическое значений вектора

Заметки

при переполнении возвращает INT32_MAX или INT32_Min

Предупреждения

При пустом векторе возвращает 0

Аргументы

vector	Вектор целых чисел для обработки
logger	Ссылка на журнал для записи ошибок

Возвращает

Среднее арифметическое значений вектора

Алгоритм:

1. Проверка на пустой вектор (возвращает 0)
2. Суммирование значений с использованием int64_t для предотвращения переполнения
3. Деление суммы на количество элементов
4. Проверка границ

Заметки

При переполнении возвращает:

2147483647 (INT32_MAX) при переполнении вверх

- -2147483647 (INT32_MIN) при переполнении вниз

Предупреждения

При пустом векторе возвращает 0 и записывает предупреждение в журнал

См. определение в файле [DataProcessor.cpp](#) строка 24

Объявления и описания членов классов находятся в файлах:

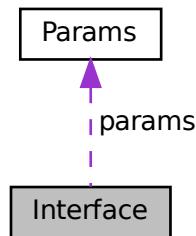
- include/[DataProcessor.h](#)
- src/[DataProcessor.cpp](#)

5.4 Класс Interface

Класс для обработки параметров командной строки

```
#include <Interface.h>
```

Граф связей класса Interface:



Открытые члены

- **Interface ()**
Конструктор класса **Interface**.
- **bool Parser (int argc, char **argv)**
Парсинг аргументов командной строки
- **Params getParams () const**
Получение параметров
- **void printHelp () const**
Вывод справки по использованию

Закрытые данные

- **po::options_description desc**
Описание доступных опций командной строки
- **po::variables_map vm**
Карта для хранения результатов парсинга
- **Params params**
Экземпляр структуры с параметрами

5.4.1 Подробное описание

Класс для обработки параметров командной строки

Использует библиотеку Boost.program_options для парсинга аргументов

См. определение в файле [Interface.h](#) строка 25

5.4.2 Конструктор(ы)

5.4.2.1 Interface()

Interface::Interface ()

Конструктор класса **Interface**.

Инициализирует парсер опциями: help, file, log, port

Инициализирует парсер командной строки опциями: help, file, log, port

См. определение в файле [Interface.cpp](#) строка 14

5.4.3 Методы

5.4.3.1 getParams()

Params Interface::getParams () const [inline]

Получение параметров

Возвращает

Структура **Params** с заполненными значениями

См. определение в файле [Interface.h](#) строка 52

5.4.3.2 Parser()

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

Парсинг аргументов командной строки

Аргументы

argc	Количество аргументов
argv	Массив аргументов

Возвращает

true - успешный парсинг рабочих параметров, false - запрошена справка (help)

Исключения

po::error	при ошибках парсинга
-----------	----------------------

Аргументы

argc	Количество аргументов
argv	Массив аргументов

Возвращает

true - успешный парсинг рабочих параметров, false - запрошена справка (help)

Исключения

po::error	при ошибках парсинга
std::exception	при других ошибках

См. определение в файле [Interface.cpp](#) строка [31](#)

5.4.3.3 printHelp()

void Interface::printHelp () const

Вывод справки по использованию

Вывод справки по использованию программы

Выводит описание всех доступных параметров и их значений по умолчанию

См. определение в файле [Interface.cpp](#) строка [49](#)

5.4.4 Данные класса

5.4.4.1 desc

po::options_description Interface::desc [private]

Описание доступных опций командной строки

См. определение в файле [Interface.h](#) строка [27](#)

5.4.4.2 params

Params Interface::params [private]

Экземпляр структуры с параметрами

См. определение в файле [Interface.h](#) строка 29

5.4.4.3 vm

po::variables_map Interface::vm [private]

Карта для хранения результатов парсинга

См. определение в файле [Interface.h](#) строка 28

Объявления и описания членов классов находятся в файлах:

- include/[Interface.h](#)
- src/[Interface.cpp](#)

5.5 Класс Logger

Класс для ведения журнала работы сервера

```
#include <Logger.h>
```

Открытые члены

- void [init](#) (const std::string &log_path)
Инициализация журнала
- void [logError](#) (const std::string &message, bool isCritical=false)
Запись сообщения об ошибке
- void [logInfo](#) (const std::string &message)
Запись информационного сообщения

Закрытые данные

- std::string [logPath](#)
Путь к файлу журнала

5.5.1 Подробное описание

Класс для ведения журнала работы сервера

Обеспечивает запись информационных сообщений и ошибок в файл

См. определение в файле [Logger.h](#) строка 14

5.5.2 Методы

5.5.2.1 init()

```
void Logger::init (
    const std::string & log_path )
```

Инициализация журнала

Инициализация журнала с указанием пути к файлу журнала

Аргументы

log_path	Путь к файлу журнала
----------	----------------------

См. определение в файле [Logger.cpp](#) строка 16

5.5.2.2 logError()

```
void Logger::logError (
    const std::string & message,
    bool isCritical = false )
```

Запись сообщения об ошибке

Запись сообщения об ошибке в журнал

Аргументы

message	Текст сообщения об ошибке
isCritical	Флаг критичности ошибки
message	Текст сообщения об ошибке
isCritical	Флаг критичности ошибки

Формат записи: "YYYY-MM-DD HH:MM:SS; УРОВЕНЬ; СООБЩЕНИЕ"

См. определение в файле [Logger.cpp](#) строка 26

5.5.2.3 logInfo()

```
void Logger::logInfo (
    const std::string & message )
```

Запись информационного сообщения

Запись информационного сообщения в журнал

Аргументы

message	Текст информационного сообщения
message	Текст информационного сообщения

Формат записи: "YYYY-MM-DD HH:MM:SS; INFO; СООБЩЕНИЕ"

См. определение в файле [Logger.cpp](#) строка [58](#)

5.5.3 Данные класса

5.5.3.1 logPath

```
std::string Logger::logPath [private]
```

Путь к файлу журнала

См. определение в файле [Logger.h](#) строка [16](#)

Объявления и описания членов классов находятся в файлах:

- include/[Logger.h](#)
- src/[Logger.cpp](#)

5.6 Структура Params

Структура для хранения параметров сервера

```
#include <Interface.h>
```

Открытые атрибуты

- std::string **dbFile**
Путь к файлу базы данных пользователей
- std::string **logFile**
Путь к файлу журнала работы сервера
- unsigned short **port**
Порт

5.6.1 Подробное описание

Структура для хранения параметров сервера

Содержит пути к файлам базы данных и журнала, а также номер порта

См. определение в файле [Interface.h](#) строка [15](#)

5.6.2 Данные класса

5.6.2.1 dbFile

`std::string Params::dbFile`

Путь к файлу базы данных пользователей

См. определение в файле [Interface.h](#) строка 16

5.6.2.2 logfile

`std::string Params::logfile`

Путь к файлу журнала работы сервера

См. определение в файле [Interface.h](#) строка 17

5.6.2.3 port

`unsigned short Params::port`

Порт

См. определение в файле [Interface.h](#) строка 18

Объявления и описания членов структуры находятся в файле:

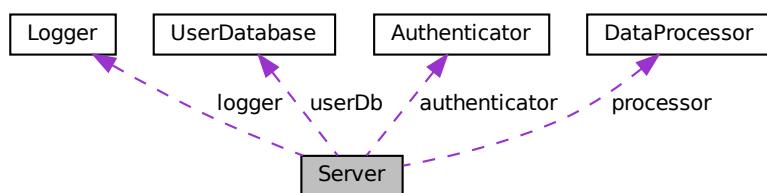
- `include/Interface.h`

5.7 Класс Server

Основной класс сервера

`#include <Server.h>`

Граф связей класса Server:



Открытые члены

- `Server (unsigned short port, Logger &logger, UserDatabase &userDb, Authenticator &authenticator, DataProcessor &processor)`
Конструктор сервера
- `~Server ()`
Деструктор сервера
- `void run ()`
Основной метод запуска сервера

Статические открытые данные

- `static const unsigned short MIN_PORT = 1024`
Минимальный допустимый порт
- `static const unsigned short MAX_PORT = 49151`
Максимальный допустимый порт

Закрытые члены

- `void validatePort (unsigned short p) const`
Проверка валидности номера порта
- `void startListening ()`
Инициализация и запуск сокета
- `void handleClient (int client_sock)`
Обработка одного клиента
- `void processVectors (int client_sock)`
Обработка векторов данных от клиента
- `std::string readTextMessage (int sock) const`
Чтение текстового сообщения от клиента
- `void sendError (int client_sock, const std::string &message) const`
Отправка сообщения об ошибке клиенту

Закрытые данные

- `unsigned short port`
Порт сервера
- `Logger & logger`
Ссылка на журнал
- `UserDatabase & userDb`
Ссылка на базу данных пользователей
- `Authenticator & authenticator`
Ссылка на аутентификатор
- `DataProcessor & processor`
Ссылка на обработчик данных
- `int listen_sock`
Сокет
- `std::unique_ptr<sockaddr_in> self_addr`
Адрес сервера
- `std::unique_ptr<sockaddr_in> foreign_addr`
Адрес клиента

5.7.1 Подробное описание

Основной класс сервера

Реализует сетевой сервер с аутентификацией и обработкой данных

См. определение в файле [Server.h](#) строка 62

5.7.2 Конструктор(ы)

5.7.2.1 Server()

```
Server::Server (
    unsigned short port,
    Logger & logger,
    UserDatabase & userDb,
    Authenticator & authenticator,
    DataProcessor & processor )
```

Конструктор сервера

Аргументы

port	Порт
logger	Ссылка на журнал
userDb	Ссылка на базу данных пользователей
authenticator	Ссылка на аутентификатор
processor	Ссылка на обработчик данных

Исключения

std::runtime_error	при невалидном порте
--------------------	----------------------

См. определение в файле [Server.cpp](#) строка 30

5.7.2.2 ~Server()

```
Server::~Server ( )
```

Деструктор сервера

Закрывает сокеты и освобождает ресурсы

См. определение в файле [Server.cpp](#) строка 43

5.7.3 Методы

5.7.3.1 handleClient()

```
void Server::handleClient (
    int client_sock ) [private]
```

Обработка одного клиента

Аргументы

client_sock	Сокет подключенного клиента
-------------	-----------------------------

Исключения

auth_error	при ошибках аутентификации
vector_error	при ошибках обработки векторов

Аргументы

client_sock	Сокет подключенного клиента
-------------	-----------------------------

Исключения

auth_error	при ошибках аутентификации
vector_error	при ошибках обработки векторов

Выполняет полный цикл взаимодействия:

1. Чтение аутентификационного сообщения
2. Проверка аутентификации
3. Обработка векторов данных

См. определение в файле [Server.cpp](#) строка 171

5.7.3.2 processVectors()

```
void Server::processVectors (
    int sock ) [private]
```

Обработка векторов данных от клиента

Аргументы

client_sock	Сокет клиента
-------------	---------------

Исключения

vector_error	при ошибках обработки векторов
--------------	--------------------------------

Аргументы

client_sock	Сокет клиента
-------------	---------------

Исключения

vector_error	при ошибках обработки векторов
--------------	--------------------------------

Протокол обработки векторов:

1. Получение количества векторов (uint32_t)
2. Для каждого вектора: а. Получение размера вектора (uint32_t) б. Получение данных вектора (int32_t[]) в. Вычисление среднего арифметического г. Отправка результата клиенту

Заметки

Проверяет корректность размера вектора

См. определение в файле [Server.cpp](#) строка [214](#)

5.7.3.3 readTextMessage()

```
std::string Server::readTextMessage (
    int sock ) const [private]
```

Чтение текстового сообщения от клиента

Аргументы

sock	Сокет клиента
------	---------------

Возвращает

Прочитанная строка

Исключения

std::system_error	при ошибках чтения
-------------------	--------------------

Аргументы

sock	Сокет клиента
------	---------------

Возвращает

Прочитанная строка

Исключения

std::system_error	при ошибках чтения
-------------------	--------------------

Удаляет символы перевода строки из полученного сообщения

См. определение в файле [Server.cpp](#) строка [84](#)

5.7.3.4 run()

void Server::run ()

Основной метод запуска сервера

Запускает бесконечный цикл обработки подключений

Заметки

Метод не возвращает управление

Исключения

std::system_error	при ошибках сетевого взаимодействия
-------------------	-------------------------------------

Запускает бесконечный цикл обработки подключений

Заметки

Метод не возвращает управление в нормальных условиях

Исключения

std::system_error	при ошибках сетевого взаимодействия
-------------------	-------------------------------------

См. определение в файле [Server.cpp](#) строка [137](#)

5.7.3.5 sendError()

```
void Server::sendError (
    int client_sock,
    const std::string & message ) const [private]
```

Отправка сообщения об ошибке клиенту

Аргументы

client_sock	Сокет клиента
message	Текст сообщения для записи в журнал
client_sock	Сокет клиента
message	Текст сообщения для записи в журнал

Отправляет строку "ERR" клиенту и записывает ошибку в журнал

См. определение в файле [Server.cpp](#) строка [71](#)

5.7.3.6 startListening()

```
void Server::startListening () [private]
```

Инициализация и запуск сокета

Исключения

std::system_error	при ошибках создания сокета
std::system_error	при ошибках создания сокета

Создает TCP сокет, привязывает к указанному порту, устанавливает флаг SO_REUSEADDR

См. определение в файле [Server.cpp](#) строка [105](#)

5.7.3.7 validatePort()

```
void Server::validatePort (
    unsigned short p ) const [private]
```

Проверка валидности номера порта

Аргументы

p	Проверяемый порт
---	------------------

Исключения

std::runtime_error	при невалидном порте
--------------------	----------------------

Аргументы

p	Проверяемый порт
---	------------------

Исключения

std::runtime_error	при невалидном порте
--------------------	----------------------

Допустимый диапазон портов: 1024-49151

См. определение в файле [Server.cpp](#) строка [56](#)

5.7.4 Данные класса

5.7.4.1 authenticator

[Authenticator&](#) Server::authenticator [private]

Ссылка на аутентификатор

См. определение в файле [Server.h](#) строка [97](#)

5.7.4.2 foreign_addr

std::unique_ptr<sockaddr_in> Server::foreign_addr [private]

Адрес клиента

См. определение в файле [Server.h](#) строка [102](#)

5.7.4.3 listen_sock

int Server::listen_sock [private]

Сокет

См. определение в файле [Server.h](#) строка [100](#)

5.7.4.4 logger

`Logger& Server::logger [private]`

Ссылка на журнал

См. определение в файле [Server.h](#) строка [95](#)

5.7.4.5 MAX_PORT

`const unsigned short Server::MAX_PORT = 49151 [static]`

Максимальный допустимый порт

См. определение в файле [Server.h](#) строка [65](#)

5.7.4.6 MIN_PORT

`const unsigned short Server::MIN_PORT = 1024 [static]`

Минимальный допустимый порт

См. определение в файле [Server.h](#) строка [64](#)

5.7.4.7 port

`unsigned short Server::port [private]`

Порт сервера

См. определение в файле [Server.h](#) строка [94](#)

5.7.4.8 processor

`DataProcessor& Server::processor [private]`

Ссылка на обработчик данных

См. определение в файле [Server.h](#) строка [98](#)

5.7.4.9 self_addr

std::unique_ptr<sockaddr_in> Server::self_addr [private]

Адрес сервера

См. определение в файле [Server.h](#) строка 101

5.7.4.10 userDb

UserDatabase& Server::userDb [private]

Ссылка на базу данных пользователей

См. определение в файле [Server.h](#) строка 96

Объявления и описания членов классов находятся в файлах:

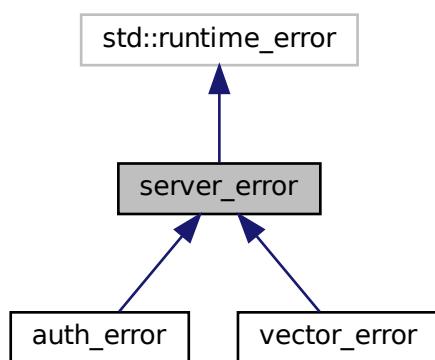
- include/[Server.h](#)
- src/[Server.cpp](#)

5.8 Класс server_error

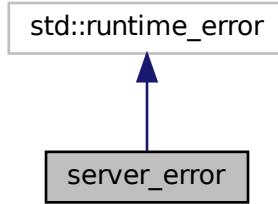
Базовый класс исключений сервера

#include <Server.h>

Граф наследования:server_error:



Граф связей класса server_error:



Открытые члены

- `server_error (const std::string &message)`
Конструктор исключения

5.8.1 Подробное описание

Базовый класс исключений сервера

Наследуется от `std::runtime_error`

См. определение в файле [Server.h](#) строка 22

5.8.2 Конструктор(ы)

5.8.2.1 `server_error()`

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

Конструктор исключения

Аргументы

message	Сообщение об ошибке
---------	---------------------

См. определение в файле [Server.h](#) строка 28

Объявления и описания членов класса находятся в файле:

- `include/Server.h`

5.9 Класс UserDatabase

Предварительное объявление класса [Logger](#).

```
#include <UserDatabase.h>
```

Открытые члены

- `bool load (const std::string &db_path, Logger &logger)`
Загрузка базы пользователей из файла
- `bool getPassword (const std::string &login, std::string &out_password) const`
Получение пароля пользователя по логину

Закрытые данные

- `std::map< std::string, std::string > users`
Контейнер для хранения пользователей

5.9.1 Подробное описание

Предварительное объявление класса [Logger](#).

Класс для работы с базой данных пользователей

Загружает пары "логин:пароль" из текстового файла и предоставляет доступ к ним для аутентификации

См. определение в файле [UserDatabase.h](#) строка 16

5.9.2 Методы

5.9.2.1 getPassword()

```
bool UserDatabase::getPassword (
    const std::string & login,
    std::string & out_password ) const
```

Получение пароля пользователя по логину

Аргументы

<code>login</code>	Логин пользователя
<code>out_password</code>	Строка для записи пароля

Возвращает

true - пользователь найден, false - пользователь не найден

Аргументы

login	Логин пользователя
out_password	Строка для записи пароля

Возвращает

true - пользователь найден, пароль записан в out_password, false - пользователь не найден

См. определение в файле [UserDatabase.cpp](#) строка 65

5.9.2.2 load()

```
bool UserDatabase::load (
    const std::string & db_path,
    Logger & logger )
```

Загрузка базы пользователей из файла

Загрузка базы данных пользователей из текстового файла

Аргументы

db_path	Путь к файлу базы данных
logger	Ссылка на объект журнала для записи ошибок

Возвращает

true - база успешно загружена, false - произошла ошибка при загрузке

Аргументы

db_path	Путь к файлу базы данных
logger	Ссылка на журнал для записи ошибок

Возвращает

true - база успешно загружена, false - произошла критическая ошибка

Формат файла: каждая строка "логин:пароль" Пустые строки и строки, начинающиеся с '#', игнорируются

Заметки

Не критические ошибки (неверный формат строки) записываются в журнал, но не прерывают загрузку

См. определение в файле [UserDatabase.cpp](#) строка 21

5.9.3 Данные класса

5.9.3.1 users

```
std::map<std::string, std::string> UserDatabase::users [private]
```

Контейнер для хранения пользователей

См. определение в файле [UserDatabase.h](#) строка 18

Объявления и описания членов классов находятся в файлах:

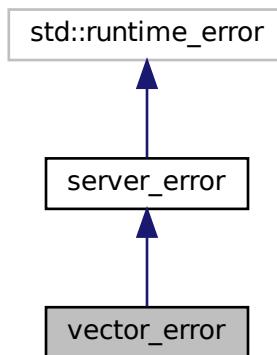
- [include/UserDatabase.h](#)
- [src/UserDatabase.cpp](#)

5.10 Класс vector_error

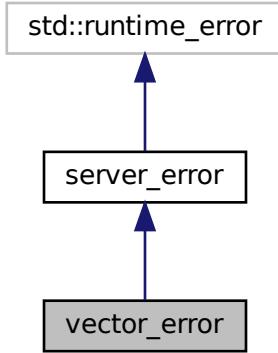
Исключение для ошибок обработки векторов

```
#include <Server.h>
```

Граф наследования:vector_error:



Граф связей класса `vector_error`:



Открытые члены

- `vector_error` (const std::string &message)
Конструктор исключения ошибок обработки векторов

5.10.1 Подробное описание

Исключение для ошибок обработки векторов

См. определение в файле [Server.h](#) строка 48

5.10.2 Конструктор(ы)

5.10.2.1 `vector_error()`

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

Конструктор исключения ошибок обработки векторов

Аргументы

<code>message</code>	Сообщение об ошибке обработки векторов
----------------------	--

См. определение в файле [Server.h](#) строка 54

Объявления и описания членов класса находятся в файле:

- include/[Server.h](#)

Глава 6

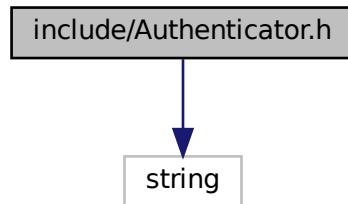
Файлы

6.1 Файл include/Authenticator.h

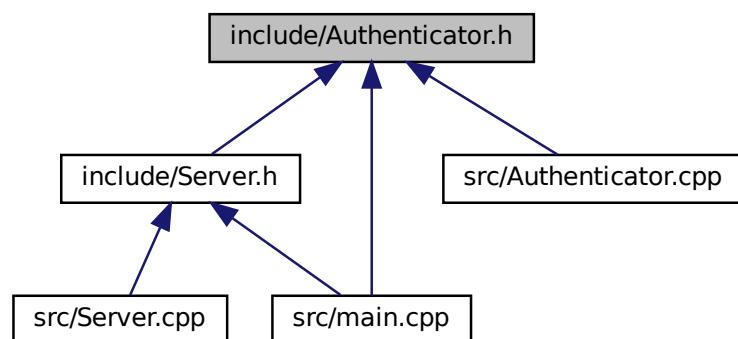
Заголовочный файл модуля [Authenticator](#) - аутентификация пользователей

```
#include <string>
```

Граф включаемых заголовочных файлов для Authenticator.h:



Граф файлов, в которые включается этот файл:



Классы

- class [Authenticator](#)

Предварительное объявление класса [Logger](#).

6.1.1 Подробное описание

Заголовочный файл модуля [Authenticator](#) - аутентификация пользователей

См. определение в файле [Authenticator.h](#)

6.2 Authenticator.h

[См. документацию.](#)

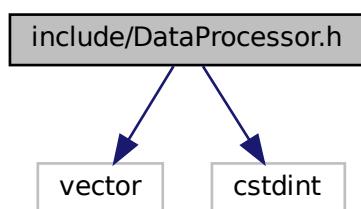
```
00001
00006 #pragma once
00007 #include <string>
00008
00009 class UserDatabase;
00010 class Logger;
00011
00016 class Authenticator {
00017 public:
00027     bool verify(const std::string& login, const std::string& salt_hash_client,
00028                 UserDatabase& db, Logger& logger) const;
00029
00030 private:
00037     bool isValidHex(const std::string& str) const;
00038
00039     static const int SALT16_LENGTH = 16;
00040     static const int SHA1_HEX_LENGTH = 40;
00041 };
```

6.3 Файл include/DataProcessor.h

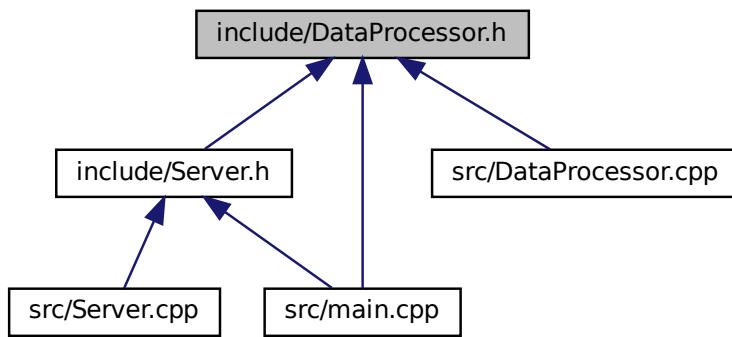
Заголовочный файл модуля [DataProcessor](#) - обработка числовых данных

```
#include <vector>
#include <cstdint>
```

Граф включаемых заголовочных файлов для DataProcessor.h:



Граф файлов, в которые включается этот файл:



Классы

- class [DataProcessor](#)
Предварительное объявление класса [Logger](#).

6.3.1 Подробное описание

Заголовочный файл модуля [DataProcessor](#) - обработка числовых данных

См. определение в файле [DataProcessor.h](#)

6.4 DataProcessor.h

[См. документацию.](#)

```

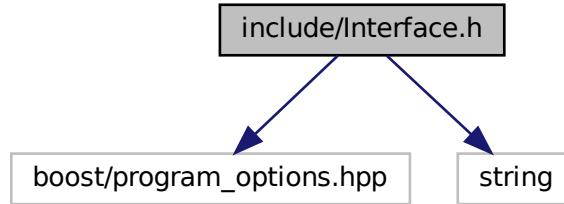
00001
00002 #pragma once
00003 #include <vector>
00004 #include <cstdint>
00005
00006 class Logger;
00007
00016 class DataProcessor {
00017 public:
00018     int32_t calculateAverage(const std::vector<int32_t>& vector, Logger& logger);
00019 };
  
```

6.5 Файл include/Interface.h

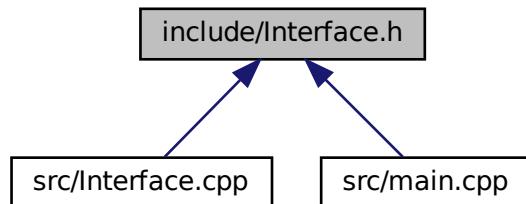
Заголовочный файл модуля [Interface](#) - обработка параметров командной строки

```
#include <boost/program_options.hpp>
#include <string>
```

Граф включаемых заголовочных файлов для Interface.h:



Граф файлов, в которые включается этот файл:



Классы

- struct [Params](#)
Структура для хранения параметров сервера
- class [Interface](#)
Класс для обработки параметров командной строки

6.5.1 Подробное описание

Заголовочный файл модуля [Interface](#) - обработка параметров командной строки

См. определение в файле [Interface.h](#)

6.6 Interface.h

[См. документацию.](#)

```

00001
00002 #pragma once
00003 #include <boost/program_options.hpp>
00004 #include <string>
00005 namespace po = boost::program_options;
00006
00015 struct Params {
00016     std::string dbFile;
00017     std::string logFile;
00018     unsigned short port;
00019 };
00020
00025 class Interface {
00026 private:
00027     po::options_description desc;
00028     po::variables_map vm;
00029     Params params;
00030
00031 public:
00032     Interface();
00033
00046     bool Parser(int argc, char** argv);
00047
00052     Params getParams() const {
00053         return params;
00054     }
00055
00059     void printHelp() const;
00060 };

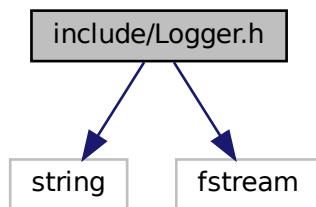
```

6.7 Файл include/Logger.h

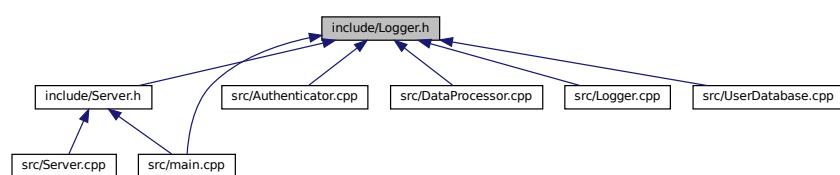
Заголовочный файл модуля [Logger](#) - журналирование работы сервера

```
#include <string>
#include <fstream>
```

Граф включаемых заголовочных файлов для `Logger.h`:



Граф файлов, в которые включается этот файл:



Классы

- class [Logger](#)

Класс для ведения журнала работы сервера

6.7.1 Подробное описание

Заголовочный файл модуля [Logger](#) - журналирование работы сервера

См. определение в файле [Logger.h](#)

6.8 Logger.h

[См. документацию.](#)

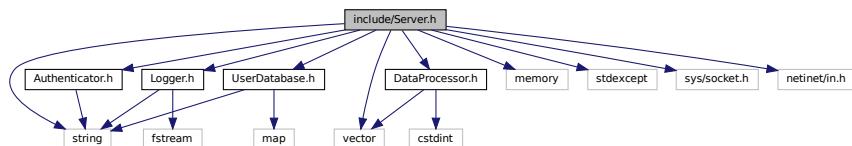
```
00001
00006 #pragma once
00007 #include <string>
00008 #include <fstream>
00009
00014 class Logger {
00015 private:
00016     std::string logPath;
00017
00018 public:
00023     void init(const std::string& log_path);
00024
00030     void logError(const std::string& message, bool isCritical = false);
00031
00036     void logInfo(const std::string& message);
00037 };
```

6.9 Файл include/Server.h

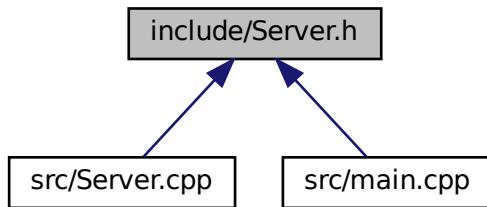
Заголовочный файл модуля [Server](#) - основной серверный модуль

```
#include "UserDatabase.h"
#include "Authenticator.h"
#include "DataProcessor.h"
#include "Logger.h"
#include <memory>
#include <vector>
#include <stdexcept>
#include <string>
#include <sys/socket.h>
#include <netinet/in.h>
```

Граф включаемых заголовочных файлов для Server.h:



Граф файлов, в которые включается этот файл:



Классы

- class [server_error](#)
Базовый класс исключений сервера
- class [auth_error](#)
Исключение для ошибок аутентификации
- class [vector_error](#)
Исключение для ошибок обработки векторов
- class [Server](#)
Основной класс сервера

6.9.1 Подробное описание

Заголовочный файл модуля [Server](#) - основной серверный модуль

См. определение в файле [Server.h](#)

6.10 Server.h

[См. документацию.](#)

```

00001
00006 #pragma once
00007 #include "UserDatabase.h"
00008 #include "Authenticator.h"
00009 #include "DataProcessor.h"
00010 #include "Logger.h"
00011 #include <memory>
00012 #include <vector>
00013 #include <stdexcept>
00014 #include <string>
00015 #include <sys/socket.h>
00016 #include <netinet/in.h>
00017
00022 class server_error : public std::runtime_error {
00023 public:
00028     explicit server_error(const std::string& message)
00029         : std::runtime_error(message) {}
00030 };
00031
00035 class auth_error : public server_error {
00036 public:
00041     explicit auth_error(const std::string& message)
  
```

```

00042     : server_error("Auth error: " + message) {}
00043 };
00044
00048 class vector_error : public server_error {
00049 public:
00054     explicit vector_error(const std::string& message)
00055     : server_error("Vector error: " + message) {}
00056 };
00057
00062 class Server {
00063 public:
00064     static const unsigned short MIN_PORT = 1024;
00065     static const unsigned short MAX_PORT = 49151;
00066
00076     Server(unsigned short port, Logger& logger, UserDatabase& userDb,
00077           Authenticator& authenticator, DataProcessor& processor);
00078
00083     ~Server();
00084
00091     void run();
00092
00093 private:
00094     unsigned short port;
00095     Logger& logger;
00096     UserDatabase& userDb;
00097     Authenticator& authenticator;
00098     DataProcessor& processor;
00099
00100    int listen_sock;
00101    std::unique_ptr<sockaddr_in> self_addr;
00102    std::unique_ptr<sockaddr_in> foreign_addr;
00103
00109    void validatePort(unsigned short p) const;
00110
00115    void startListening();
00116
00123    void handleClient(int client_sock);
00124
00130    void processVectors(int client_sock);
00131
00138    std::string readTextMessage(int sock) const;
00139
00145    void sendError(int client_sock, const std::string& message) const;
00146 };

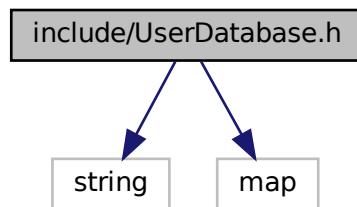
```

6.11 Файл include/UserDatabase.h

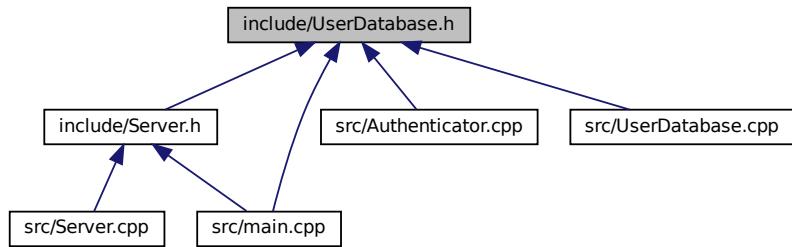
Заголовочный файл модуля **UserDatabase** - работа с базой пользователей

```
#include <string>
#include <map>
```

Граф включаемых заголовочных файлов для UserDatabase.h:



Граф файлов, в которые включается этот файл:



Классы

- class [UserDatabase](#)

Предварительное объявление класса [Logger](#).

6.11.1 Подробное описание

Заголовочный файл модуля [UserDatabase](#) - работа с базой пользователей

См. определение в файле [UserDatabase.h](#)

6.12 UserDatabase.h

[См. документацию.](#)

```

00001
00006 #pragma once
00007 #include <string>
00008 #include <map>
00009
00010 class Logger;
00011
00016 class UserDatabase {
00017 private:
00018     std::map<std::string, std::string> users;
00019
00020 public:
00028     bool load(const std::string& db_path, Logger& logger);
00029
00037     bool getPassword(const std::string& login, std::string& out_password) const;
00038 };
  
```

6.13 Файл src/Authenticator.cpp

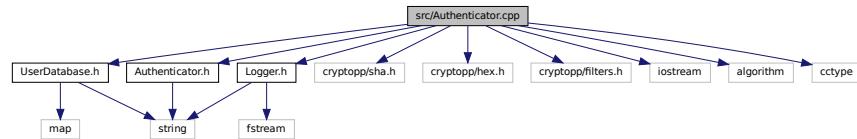
Реализация класса [Authenticator](#) для аутентификации пользователей

```

#include "Authenticator.h"
#include "UserDatabase.h"
#include "Logger.h"
#include <cryptopp/sha.h>
  
```

```
#include <cryptopp/hex.h>
#include <cryptopp/filters.h>
#include <iostream>
#include <algorithm>
#include <cctype>
```

Граф включаемых заголовочных файлов для Authenticator.cpp:



6.13.1 Подробное описание

Реализация класса [Authenticator](#) для аутентификации пользователей

См. определение в файле [Authenticator.cpp](#)

6.14 Authenticator.cpp

[См. документацию.](#)

```

00001
00006 #include "Authenticator.h"
00007 #include "UserDatabase.h"
00008 #include "Logger.h"
00009 #include <cryptopp/sha.h>
00010 #include <cryptopp/hex.h>
00011 #include <cryptopp/filters.h>
00012 #include <iostream>
00013 #include <algorithm>
00014 #include <cctype>
00015
00016 namespace CPP = CryptoPP;
00017
00026 bool Authenticator::isValidHex(const std::string& str) const {
00027     return std::all_of(str.begin(), str.end(), [](char c) {
00028         return (c >= '0' && c <= '9') || (c >= 'A' && c <= 'F') || (c >= 'a' && c <= 'f');
00029     });
00030 }
00031
00049 bool Authenticator::verify(const std::string& login, const std::string& salt_hash_client,
00050                                 UserDatabase& db, Logger& logger) const {
00051
00052     if (salt_hash_client.length() != SALT16_LENGTH + SHA1_HEX_LENGTH) {
00053         logger.logError("Authenticator: Message length mismatch. Expected: " +
00054             std::to_string(SALT16_LENGTH + SHA1_HEX_LENGTH) +
00055             ", got: " + std::to_string(salt_hash_client.length()), false);
00056         return false;
00057     }
00058
00059     std::string salt16 = salt_hash_client.substr(0, SALT16_LENGTH);
00060     std::string clientHash16 = salt_hash_client.substr(SALT16_LENGTH);
00061
00062     if (!isValidHex(salt16)) {
00063         logger.logError("Authenticator: Invalid hex format in SALT16: " + salt16, false);
00064         return false;
00065     }
00066
00067     if (!isValidHex(clientHash16)) {
00068         logger.logError("Authenticator: Invalid hex format in client hash", false);
00069         return false;
00070     }
00071
00072     std::string password;
00073     if (!db.getPassword(login, password)) {

```

```

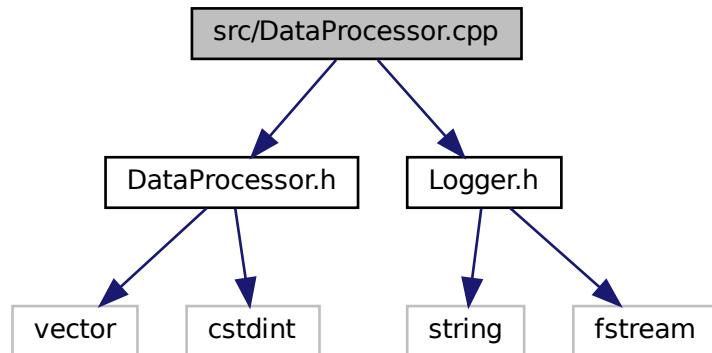
00074     logger.logError("Authenticator: Login " + login + " not found", false);
00075     return false;
00076 }
00077
00078 std::string input = salt16 + password;
00079 std::string serverHash16;
00080
00081 try {
00082     CPP::SHA1 hash;
00083
00084     CPP::StringSource(input, true,
00085         new CPP::HashFilter(hash,
00086             new CPP::HexEncoder(
00087                 new CPP::StringSink(serverHash16))));
00088
00089 } catch (const CPP::Exception& e) {
00090     logger.logError(std::string("Crypto++ error: ") + e.what(), true);
00091     return false;
00092 }
00093
00094 std::transform(clientHash16.begin(), clientHash16.end(), clientHash16.begin(), ::toupper);
00095 std::transform(serverHash16.begin(), serverHash16.end(), serverHash16.begin(), ::toupper);
00096
00097 if (serverHash16 == clientHash16) {
00098     logger.logInfo("Authenticator: Success for login " + login);
00099     return true;
00100 } else {
00101     logger.logError("Authenticator: Password mismatch for login " + login, false);
00102     return false;
00103 }
00104 }
```

6.15 Файл src/DataProcessor.cpp

Реализация класса [DataProcessor](#) для обработки числовых данных

```
#include "DataProcessor.h"
#include "Logger.h"
```

Граф включаемых заголовочных файлов для DataProcessor.cpp:



6.15.1 Подробное описание

Реализация класса [DataProcessor](#) для обработки числовых данных

См. определение в файле [DataProcessor.cpp](#)

6.16 DataProcessor.cpp

См. документацию.

```

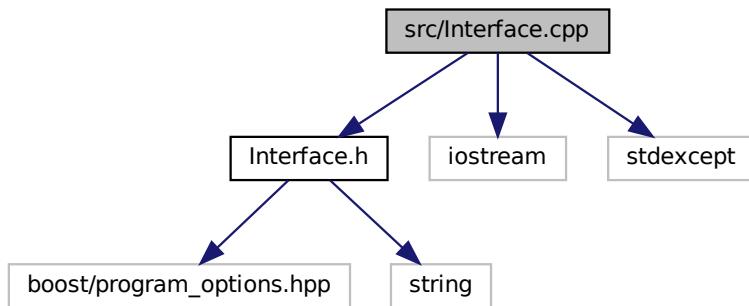
00001
00006 #include "DataProcessor.h"
00007 #include "Logger.h"
00008
00024 int32_t DataProcessor::calculateAverage(const std::vector<int32_t>& vector, Logger& logger) {
00025     if (vector.empty()) {
00026         logger.logError("Vector is empty", false);
00027         return 0;
00028     }
00029
00030     int64_t sum = 0; //предотвращение промежуточного переполнения
00031     for (int32_t val : vector) {
00032         sum += val;
00033     }
00034
00035     int64_t avrg = sum / static_cast<int64_t>(vector.size());
00036
00037     if (avrg > 2147483647) { // 2^(31-1)
00038         logger.logError("Overflow detected (upwards)", false);
00039         return 2147483647;
00040     }
00041     if (avrg < -2147483648) { // -2^31
00042         logger.logError("Overflow detected (downwards)", false);
00043         return -2147483648;
00044     }
00045
00046     return static_cast<int32_t>(avrg);
00047 }
```

6.17 Файл src/Interface.cpp

Реализация класса [Interface](#) для обработки параметров командной строки

```
#include "Interface.h"
#include <iostream>
#include <stdexcept>
```

Граф включаемых заголовочных файлов для Interface.cpp:



6.17.1 Подробное описание

Реализация класса [Interface](#) для обработки параметров командной строки

См. определение в файле [Interface.cpp](#)

6.18 Interface.cpp

[См. документацию.](#)

```

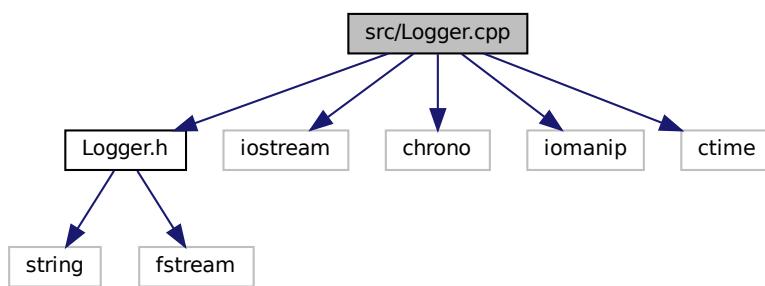
00001
00006 #include "Interface.h"
00007 #include <iostream>
00008 #include <stdexcept>
00009
00014 Interface::Interface() : desc("Allowed options") {
00015     desc.add_options()
00016     ("help,h", "Show help")
00017     ("file,f", po::value<std::string>(&params.dbFile)->default_value("etc/vcalc.conf"), "User database file")
00018     ("log,l", po::value<std::string>(&params.logFile)->default_value("var/log/vcalc.log"), "Log file")
00019     ("port,p", po::value<unsigned short>(&params.port)->default_value(3333), "Server port");
00020 }
00021
00031 bool Interface::Parser(int argc, char** argv) {
00032     try {
00033         po::store(po::parse_command_line(argc, argv, desc), vm);
00034         if (vm.count("help")) {
00035             return false; //печать справки
00036         }
00037         po::notify(vm);
00038     } catch (const std::exception& e) {
00039         std::cerr << "Error parsing arguments: " << e.what() << std::endl;
00040         return false;
00041     }
00042     return true;
00043 }
00044
00049 void Interface::printHelp() const {
00050     std::cout << desc << std::endl;
00051 }
```

6.19 Файл src/Logger.cpp

Реализация класса [Logger](#) для ведения журнала работы сервера

```
#include "Logger.h"
#include <iostream>
#include <chrono>
#include <iomanip>
#include <ctime>
```

Граф включаемых заголовочных файлов для Logger.cpp:



6.19.1 Подробное описание

Реализация класса [Logger](#) для ведения журнала работы сервера

[См. определение в файле Logger.cpp](#)

6.20 Logger.cpp

[См. документацию.](#)

```

00001
00006 #include "Logger.h"
00007 #include <iostream>
00008 #include <chrono>
00009 #include <iomanip>
00010 #include <ctime>
00011
00016 void Logger::init(const std::string& log_path) {
00017     logPath = log_path;
00018 }
00019
00026 void Logger::logError(const std::string& message, bool isCritical) {
00027     std::string level;
00028     if (isCritical) {
00029         level = "CRITICAL";
00030     } else {
00031         level = "ERROR";
00032     }
00033
00034     std::ofstream file(logPath, std::ios::app);
00035     if (file.is_open()) {
00036         auto now = std::chrono::system_clock::now();
00037         auto time_t = std::chrono::system_clock::to_time_t(now);
00038         file << std::put_time(std::localtime(&time_t), "%Y-%m-%d %H:%M:%S")
00039             << ":" << level << " " << message << std::endl;
00040         file.flush();
00041         file.close();
00042     } else {
00043         std::cerr << "LOGGER ERROR: Cannot write log to " << logPath << std::endl;
00044     }
00045
00046     if (isCritical) {
00047         std::cerr << "CRITICAL ERROR: " << message << std::endl;
00048     } else {
00049         std::cerr << "ERROR: " << message << std::endl;
00050     }
00051 }
00052
00058 void Logger::logInfo(const std::string& message) {
00059     std::ofstream file(logPath, std::ios::app);
00060     if (file.is_open()) {
00061         auto now = std::chrono::system_clock::now();
00062         auto time_t = std::chrono::system_clock::to_time_t(now);
00063         file << std::put_time(std::localtime(&time_t), "%Y-%m-%d %H:%M:%S")
00064             << ":" << INFO << " " << message << std::endl;
00065         file.flush();
00066         file.close();
00067     } else {
00068         std::cerr << "LOGGER ERROR: Cannot write log to " << logPath << std::endl;
00069     }
00070 }
```

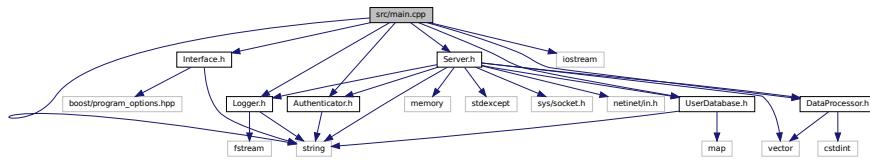
6.21 Файл src/main.cpp

Точка входа серверной программы

```

#include "Interface.h"
#include "Logger.h"
#include "UserDatabase.h"
#include "DataProcessor.h"
#include "Authenticator.h"
#include "Server.h"
#include <iostream>
#include <string>
```

Граф включаемых заголовочных файлов для main.cpp:



Функции

- bool `isValidPort` (`unsigned short port`)

Проверка валидности номера порта

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

Главная функция программы

6.21.1 Подробное описание

Точка входа серверной программы

Автор

Юдашкина В.О.

Версия

1.0

Авторство

ИБСТ ПГУ

См. определение в файле `main.cpp`

6.21.2 Функции

6.21.2.1 isValidPort()

```
bool isValidPort (
    unsigned short port )
```

Проверка валидности номера порта

Аргументы

port	Номер порта для проверки
------	--------------------------

Возвращает

true - порт в допустимом диапазоне, false - порт вне диапазона

Заметки

Диапазон допустимых портов: 1024-49151

См. определение в файле [main.cpp](#) строка [38](#)

6.21.2.2 main()

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

Главная функция программы

Аргументы

argc	Количество аргументов командной строки
argv	Массив аргументов командной строки

Возвращает

0 - успешное завершение, 1 - ошибка инициализации, 2 - ошибка выполнения

Выполняет инициализацию всех модулей и запускает сервер

Исключения

std::exception	при критических ошибках выполнения
----------------	------------------------------------

Заметки

При запуске без параметров выводит справку и запускает сервер

< Объект для ведения журнала работы сервера

< Объект для обработки параметров командной строки

Парсинг аргументов командной строки

Если парсинг не удался или запрошена справка, выводит справку и завершает работу

Получение параметров конфигурации

Извлекает распарсенные параметры для дальнейшего использования

Инициализация системы журналирования

Настраивает журнал на запись в указанный файл

Загрузка базы данных пользователей

Загружает пары "логин:пароль" из указанного файла

< Критическая ошибка: не удалось загрузить базу данных

Инициализация модулей обработки

Создает объекты для аутентификации и обработки данных

Проверка валидности порта

Убеждается, что указанный порт находится в допустимом диапазоне

< Критическая ошибка: неверный порт

Вывод информации о конфигурации

Отображает в консоли успешно загруженные параметры

Создание и запуск сервера

Основной блок выполнения программы

< Запуск основного цикла сервера

Обработка критических ошибок выполнения

Записывает в журнал ошибку и завершает работу

См. определение в файле [main.cpp](#) строка 53

6.22 main.cpp

[См. документацию.](#)

```

00001
00022 #include "Interface.h"
00023 #include "Logger.h"
00024 #include "UserDatabase.h"
00025 #include "DataProcessor.h"
00026 #include "Authenticator.h"
00027 #include "Server.h"
00028 #include <iostream>
00029 #include <string>
00030
00038 bool isValidPort(unsigned short port) {
00039     return port >= 1024 && port <= 49151;
00040 }
00041
00053 int main(int argc, char** argv) {
00054     Logger logger;
00055     Interface iface;
00056
00057     if (argc == 1) {
00058         iface.printHelp(); // вывод справки и запуск сервера
00059     }
00060
00065     if (!iface.Parser(argc, argv)) {
00066         iface.printHelp(); // вывод справки и завершение программы
00067         return 0;
00068     }
00069
00074     Params params = iface.getParams();
00075
00080     logger.init(params.logFile);
00081     logger.logInfo("Server configuration parsing completed");
00082
00087     UserDatabase userDb;
00088     if (!userDb.load(params.dbFile, logger)) {
00089         return 1;
00090     }
00091
00096     Authenticator auth;
00097     DataProcessor processor;
00098     logger.logInfo("Authenticator and DataProcessor initialized");
00099
00104     if (!isValidPort(params.port)) {
00105         std::string error_msg = "Invalid port number: " + std::to_string(params.port);
00106         logger.logError(error_msg, true);
00107         return 1;
00108     }
00109
00114     std::cout << "Server configuration loaded successfully" << std::endl;
00115     std::cout << "Database file: " << params.dbFile << std::endl;
00116     std::cout << "Log file: " << params.logFile << std::endl;
00117     std::cout << "Port: " << params.port << std::endl;
00118
00123     try {
00124         Server server(params.port, logger, userDb, auth, processor);
00125         server.run();
00126     } catch (const std::exception& e) {
00131         logger.logError("Server fatal error: " + std::string(e.what()), true);
00132         std::cerr << "Server error: " << e.what() << std::endl;
00133         return 1;
00134     }
00135     return 0;
00136 }
```

6.23 Файл src/Server.cpp

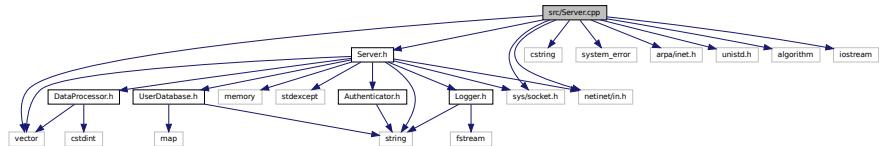
Реализация основного серверного модуля

```

#include "Server.h"
#include <cstring>
#include <system_error>
#include <arpa/inet.h>
#include <vector>
#include <unistd.h>
```

```
#include <algorithm>
#include <iostream>
#include <sys/socket.h>
#include <netinet/in.h>
```

Граф включаемых заголовочных файлов для Server.cpp:



Макросы

- `#define BUFSIZE 1024`
Максимальный размер буфера для текстового сообщения аутентификации
- `#define QLEN 10`
Стандартная очередь для listen.
- `#define AUTH_DATA_LENGTH (16 + 40)`
16 символов SALT + 40 символов HASH

6.23.1 Подробное описание

Реализация основного серверного модуля

См. определение в файле [Server.cpp](#)

6.23.2 Макросы

6.23.2.1 AUTH_DATA_LENGTH

```
#define AUTH_DATA_LENGTH (16 + 40)
```

16 символов SALT + 40 символов HASH

См. определение в файле [Server.cpp](#) строка 19

6.23.2.2 BUFSIZE

```
#define BUFSIZE 1024
```

Максимальный размер буфера для текстового сообщения аутентификации

См. определение в файле [Server.cpp](#) строка 17

6.23.2.3 QLEN

```
#define QLEN 10
```

Стандартная очередь для listen.

См. определение в файле [Server.cpp](#) строка 18

6.24 Server.cpp

[См. документацию.](#)

```
00001
00006 #include "Server.h"
00007 #include <cstring>
00008 #include <system_error>
00009 #include <arpa/inet.h>
00010 #include <vector>
00011 #include <unistd.h>
00012 #include <algorithm>
00013 #include <iostream>
00014 #include <sys/socket.h>
00015 #include <netinet/in.h>
00016
00017 #define BUFSIZE 1024
00018 #define QLEN 10
00019 #define AUTH_DATA_LENGTH (16 + 40)
00020
00030 Server::Server(unsigned short port, Logger& logger, UserDatabase& userDb,
00031             Authenticator& authenticator, DataProcessor& processor)
00032     : port(port), logger(logger), userDb(userDb),
00033       authenticator(authenticator), processor(processor),
00034       listen_sock(-1), self_addr(new sockaddr_in), foreign_addr(new sockaddr_in)
00035 {
00036     validatePort(port);
00037 }
00038
00043 Server::~Server() {
00044     if (listen_sock != -1) {
00045         close(listen_sock);
00046         logger.logInfo("Server socket closed");
00047     }
00048 }
00049
00056 void Server::validatePort(unsigned short p) const {
00057     if (p < MIN_PORT || p > MAX_PORT) {
00058         std::string err_msg = "Port (" + std::to_string(p) + ") out of range " +
00059                               std::to_string(MIN_PORT) + "-" + std::to_string(MAX_PORT);
00060         logger.logError(err_msg, true);
00061         throw std::runtime_error(err_msg);
00062     }
00063 }
00064
00071 void Server::sendError(int client_sock, const std::string& message) const {
00072     const char* err_msg = "ERR";
00073     send(client_sock, err_msg, strlen(err_msg), 0);
00074     logger.logError("Error sent to client: " + message, false);
00075 }
00076
00084 std::string Server::readTextMessage(int sock) const {
00085     char buffer[BUFSIZE];
00086     ssize_t rc = recv(sock, buffer, BUFSIZE - 1, 0);
00087     if (rc == -1) {
00088         throw std::system_error(errno, std::generic_category(), "recv error reading MSG");
00089     }
00090     if (rc == 0) return "";
00091     buffer[rc] = '\0';
00092     std::string message(buffer);
00093
00095     message.erase(std::remove_if(message.begin(), message.end(),
00096                                 [](char c){ return c == '\n' || c == '\r'; }), message.end());
00097     return message;
00098 }
00099
00105 void Server::startListening() {
00106     listen_sock = socket(AF_INET, SOCK_STREAM, 0);
00107     if (listen_sock == -1) {
00108         throw std::system_error(errno, std::generic_category(), "socket creation failed");
```

```

00109     }
00110
00111     int on = 1;
00112     if (setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR, (const char*)&on, sizeof on) == -1) {
00113         logger.logError("setsockopt SO_REUSEADDR failed, continuing", false);
00114     }
00115
00116     self_addr->sin_family = AF_INET;
00117     self_addr->sin_port = htons(port);
00118     self_addr->sin_addr.s_addr = INADDR_ANY;
00119
00120     if (bind(listen_sock, reinterpret_cast<const sockaddr*>(self_addr.get()), sizeof(sockaddr_in)) == -1) {
00121         close(listen_sock);
00122         throw std::system_error(errno, std::generic_category(), "bind failed");
00123     }
00124     if (listen(listen_sock, QLEN) == -1) {
00125         close(listen_sock);
00126         throw std::system_error(errno, std::generic_category(), "listen failed");
00127     }
00128     logger.logInfo("Server started and listening on port " + std::to_string(port));
00129 }
00130
00131 void Server::run() {
00132     startListening();
00133     socklen_t socklen = sizeof(sockaddr_in);
00134     while(true) {
00135         int work_sock = -1;
00136         try {
00137             logger.logInfo("Waiting for new client...");
00138             work_sock = accept(listen_sock, reinterpret_cast<sockaddr*>(foreign_addr.get()), &socklen);
00139             if (work_sock == -1) {
00140                 logger.logError("Accept error: " + std::string(strerror(errno)), false); continue;
00141             }
00142             std::string ip_addr(inet_ntoa(foreign_addr->sin_addr));
00143             logger.logInfo("Connection established with " + ip_addr);
00144             handleClient(work_sock);
00145         } catch (const std::exception& e) {
00146             logger.logError("Error in server loop: " + std::string(e.what()), false);
00147         }
00148         if (work_sock != -1) {
00149             close(work_sock);
00150             logger.logInfo("Connection closed");
00151         }
00152     }
00153 }
00154
00155 void Server::handleClient(int client_sock) {
00156     try {
00157         std::string full_msg = readTextMessage(client_sock);
00158         if (full_msg.empty()) {
00159             logger.logError("Client disconnected during authentication", false);
00160             return;
00161         }
00162         if (full_msg.length() < AUTH_DATA_LENGTH) {
00163             throw auth_error("Auth message too short");
00164         }
00165         std::string auth_data = full_msg.substr(full_msg.length() - AUTH_DATA_LENGTH);
00166         std::string login = full_msg.substr(0, full_msg.length() - AUTH_DATA_LENGTH);
00167
00168         if (authenticator.verify(login, auth_data, userDb, logger)) {
00169             const char* ok_msg = "OK";
00170             send(client_sock, ok_msg, strlen(ok_msg), 0);
00171             logger.logInfo("Client '" + login + "' authenticated successfully");
00172         } else {
00173             throw auth_error("Authentication failed for login " + login);
00174         }
00175         processVectors(client_sock);
00176     } catch (const auth_error& e) {
00177         sendError(client_sock, e.what());
00178         throw;
00179     } catch (const std::exception& e) {
00180         sendError(client_sock, "Protocol processing error");
00181         throw;
00182     }
00183 }
00184
00185 void Server::processVectors(int sock) {
00186     uint32_t num_vectors;
00187     ssize_t rc;
00188
00189     rc = recv(sock, &num_vectors, sizeof(num_vectors), 0);
00190     if (rc != sizeof(num_vectors)) {
00191         throw vector_error("Failed to receive number of vectors");
00192     }
00193     logger.logInfo("Receiving " + std::to_string(num_vectors) + " vectors");
00194     for (uint32_t i = 0; i < num_vectors; ++i) {
00195         uint32_t vector_len;
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
015310
015311
015312
015313
015314
015315
015316
015317
015318
015319
015320
015321
015322
015323
015324
015325
015326
015327
015328
015329
015330
015331
015332
015333
015334
015335
015336
015337
015338
015339
015340
015341
015342
015343
015344
015345
015346
015347
015348
015349
015350
015351
015352
015353
015354
015355
015356
015357
015358
015359
015360
015361
015362
015363
015364
015365
015366
015367
015368
015369
015370
015371
015372
015373
015374
015375
015376
015377
015378
015379
015380
015381
015382
015383
015384
015385
015386
015387
015388
015389
015390
015391
015392
015393
015394
015395
015396
015397
015398
015399
0153100
0153101
0153102
0153103
0153104
0153105
0153106
0153107
0153108
0153109
0153110
0153111
0153112
0153113
0153114
0153115
0153116
0153117
0153118
0153119
0153120
0153121
0153122
0153123
0153124
0153125
0153126
0153127
0153128
0153129
0153130
0153131
0153132
0153133
0153134
0153135
0153136
0153137
0153138
0153139
0153140
0153141
0153142
0153143
0153144
0153145
0153146
0153147
0153148
0153149
0153150
0153151
0153152
0153153
0153154
0153155
0153156
0153157
0153158
0153159
0153160
0153161
0153162
0153163
0153164
0153165
0153166
0153167
0153168
0153169
0153170
0153171
0153172
0153173
0153174
0153175
0153176
0153177
0153178
0153179
0153180
0153181
0153182
0153183
0153184
0153185
0153186
0153187
0153188
0153189
0153190
0153191
0153192
0153193
0153194
0153195
0153196
0153197
0153198
0153199
0153200
0153201
0153202
0153203
0153204
0153205
0153206
0153207
0153208
0153209
0153210
0153211
0153212
0153213
0153214
0153215
0153216
0153217
0153218
0153219
0153220
0153221
0153222
0153223
0153224
0153225
0153226
0153227
0153228
0153229
0153230
0153231
0153232
0153233
0153234
0153235
0153236
0153237
0153238
0153239
0153240
0153241
0153242
0153243
01532
```

```

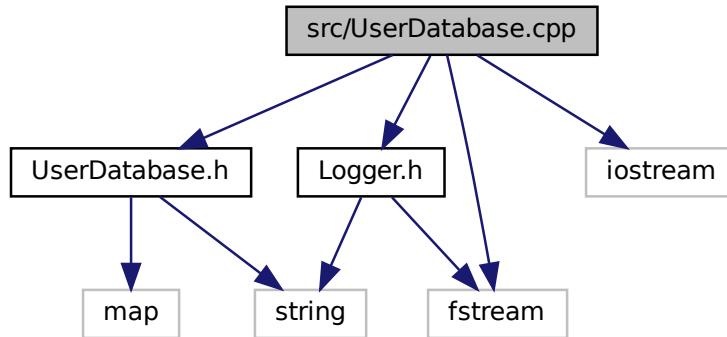
00225     rc = recv(sock, &vector_len, sizeof(vector_len), 0);
00226     if (rc != sizeof(vector_len)) {
00227         throw vector_error("Failed to receive vector length");
00228     }
00229
00230     size_t total_bytes_needed = vector_len * sizeof(int32_t);
00231
00232     if (vector_len == 0 || total_bytes_needed > 4000000000) {
00233         throw vector_error("Vector size invalid or too large");
00234     }
00235     std::vector<int32_t> data(vector_len);
00236     rc = recv(sock, data.data(), total_bytes_needed, 0);
00237     if (rc != (ssize_t)total_bytes_needed) {
00238         throw vector_error("Vector data size mismatch");
00239     }
00240     int32_t result = processor.calculateAverage(data, logger);
00241     int32_t net_result = (result);
00242     send(sock, &net_result, sizeof(net_result), 0);
00243
00244     logger.logInfo("Processed vector " + std::to_string(i+1) + ", result: " + std::to_string(result));
00245 }
00246 }
```

6.25 Файл src/UserDatabase.cpp

Реализация класса [UserDatabase](#) для работы с базой данных пользователей

```
#include "UserDatabase.h"
#include "Logger.h"
#include <fstream>
#include <iostream>
```

Граф включаемых заголовочных файлов для UserDatabase.cpp:



6.25.1 Подробное описание

Реализация класса [UserDatabase](#) для работы с базой данных пользователей

См. определение в файле [UserDatabase.cpp](#)

6.26 UserDatabase.cpp

[См. документацию.](#)

```
00001
00006 #include "UserDatabase.h"
00007 #include "Logger.h"
00008 #include <fstream>
00009 #include <iostream>
00010
00021 bool UserDatabase::load(const std::string& db_path, Logger& logger) {
00022     std::ifstream file(db_path);
00023     if (!file.is_open()) {
00024         logger.logError("Cannot open database file: " + db_path, true);
00025         return false;
00026     }
00027
00028     std::string line;
00029     int line_number = 0;
00030     while (std::getline(file, line)) {
00031         line_number++;
00032         if (line.empty() || line[0] == '#') {
00033             continue;
00034         }
00035         size_t pos = line.find(':');
00036         if (pos == std::string::npos) {
00037             logger.logError("Invalid string format " + std::to_string(line_number), false);
00038             continue;
00039         }
00040         std::string login = line.substr(0, pos);
00041         std::string password = line.substr(pos + 1);
00042         if (login.empty() || password.empty()) {
00043             logger.logError("Empty login or password on line " + std::to_string(line_number), false);
00044             continue;
00045         }
00046         users[login] = password;
00047     }
00048     file.close();
00049
00050     if (users.empty()) {
00051         logger.logError("User database is empty", true);
00052         return false;
00053     }
00054     logger.logInfo("Loaded " + std::to_string(users.size()) + " users from database");
00055     return true;
00056 }
00057
00065 bool UserDatabase::getPassword(const std::string& login, std::string& out_password) const {
00066     auto it = users.find(login);
00067     if (it != users.end()) {
00068         out_password = it->second;
00069         return true;
00070     }
00071     return false;
00072 }
```


Предметный указатель

~Server
 Server, 24

AUTH_DATA_LENGTH
 Server.cpp, 57

auth_error, 9
 auth_error, 10

Authenticator, 11
 isValidHex, 11
 SALT16_LENGTH, 13
 SHA1_HEX_LENGTH, 13
 verify, 12

authenticator
 Server, 29

BUFSIZE
 Server.cpp, 57

calculateAverage
 DataProcessor, 14

DataProcessor, 14
 calculateAverage, 14

dbFile
 Params, 22

desc
 Interface, 18

foreign_addr
 Server, 29

getParams
 Interface, 17

getPassword
 UserDatabase, 33

handleClient
 Server, 25

include/Authenticator.h, 39, 40

include/DataProcessor.h, 40, 41

include/Interface.h, 41, 43

include/Logger.h, 43, 44

include/Server.h, 44, 45

include/UserDatabase.h, 46, 47

init
 Logger, 20

Interface, 16
 desc, 18
 getParams, 17
 Interface, 17

 params
 Interface, 18

Parser
 Interface, 17

port
 Params, 22
 Server, 30

 params, 18
 Parser, 17
 printHelp, 18
 vm, 19

isValidHex
 Authenticator, 11

isValidPort
 main.cpp, 53

listen_sock
 Server, 29

load
 UserDatabase, 34

logError
 Logger, 20

logFile
 Params, 22

Logger, 19
 init, 20
 logError, 20
 logInfo, 20
 logPath, 21

logger
 Server, 29

logInfo
 Logger, 20

logPath
 Logger, 21

main
 main.cpp, 54

main.cpp
 isValidPort, 53
 main, 54

MAX_PORT
 Server, 30

MIN_PORT
 Server, 30

 Params, 21
 dbFile, 22
 logFile, 22
 port, 22

 port
 Params, 22
 Server, 30

printHelp
 Interface, 18
processor
 Server, 30
processVectors
 Server, 25

QLEN
 Server.cpp, 57

readTextMessage
 Server, 26
run
 Server, 27

SALT16_LENGTH
 Authenticator, 13
self_addr
 Server, 30
sendError
 Server, 27
Server, 22
 ~Server, 24
 authenticator, 29
 foreign_addr, 29
 handleClient, 25
 listen_sock, 29
 logger, 29
 MAX_PORT, 30
 MIN_PORT, 30
 port, 30
 processor, 30
 processVectors, 25
 readTextMessage, 26
 run, 27
 self_addr, 30
 sendError, 27
 Server, 24
 startListening, 28
 userDb, 31
 validatePort, 28
Server.cpp
 AUTH_DATA_LENGTH, 57
 BUFSIZE, 57
 QLEN, 57
server_error, 31
 server_error, 32
SHA1_HEX_LENGTH
 Authenticator, 13
src/Authenticator.cpp, 47, 48
src/DataProcessor.cpp, 49, 50
src/Interface.cpp, 50, 51
src/Logger.cpp, 51, 52
src/main.cpp, 52, 56
src/Server.cpp, 56, 58
src/UserDatabase.cpp, 60, 61
startListening
 Server, 28

UserDatabase, 33
 getPassword, 33
 load, 34
 users, 35
userDb
 Server, 31
users
 UserDatabase, 35

validatePort
 Server, 28
vector_error, 35
 vector_error, 36
verify
 Authenticator, 12
vm
 Interface, 19