Программная реализация сетевого сервера

Создано системой Doxygen 1.9.4

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.

Глава 3

Алфавитный указатель классов

3.1 Классы

Классы с их кратким описанием.

Глава 4

Список файлов

4.1 Файлы

Полный список документированных файлов.

Глава 5

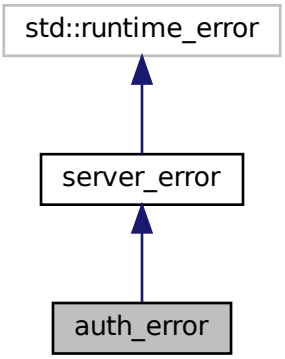
Классы

5.1 Класс auth_error

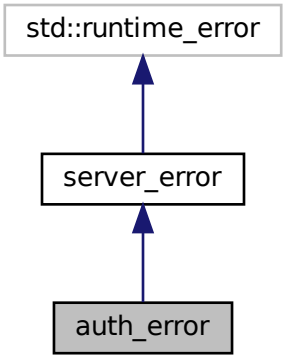
Исключение для ошибок аутентификации

#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 соотвествие 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 соотвествие 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: HASH = SHA1(SALT || 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:

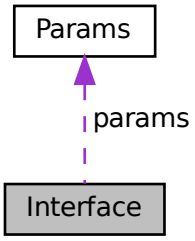
- include/DataProcessor.h
- src/DataProcessor.cpp

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:

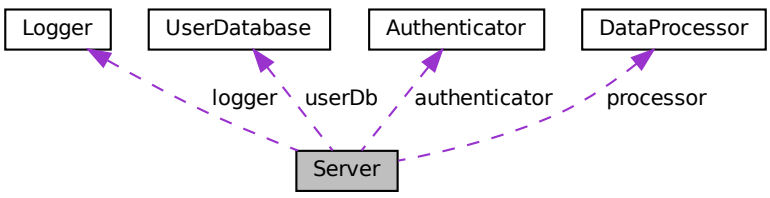
- include/Interface.h

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 коректность 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:

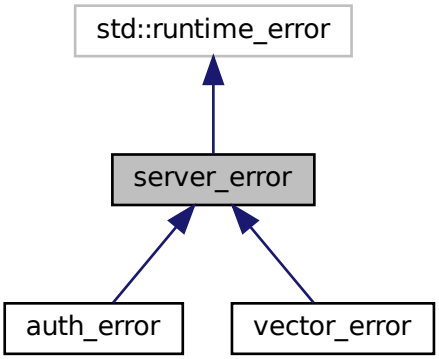
- include/Server.h
- src/Server.cpp

5.8 Класс server_error

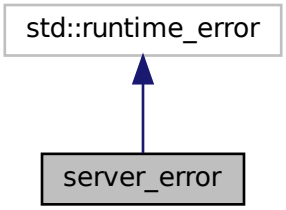
Базовый класс исключений сервера

#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

login	Логин пользователя
out_password	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:

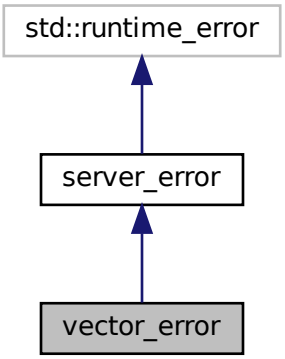
- include/UserDatabase.h
- src/UserDatabase.cpp

5.10 Класс vector_error

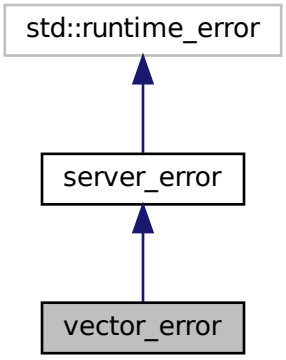
Исключение для ошибок обработки векторов

#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

message	Сообщение об ошибке обработки векторов

См. определение в файле Server.h строка 54

Объявления и описания членов класса находятся в файле:

- include/Server.h

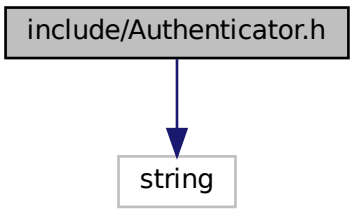
Глава 6

Файлы

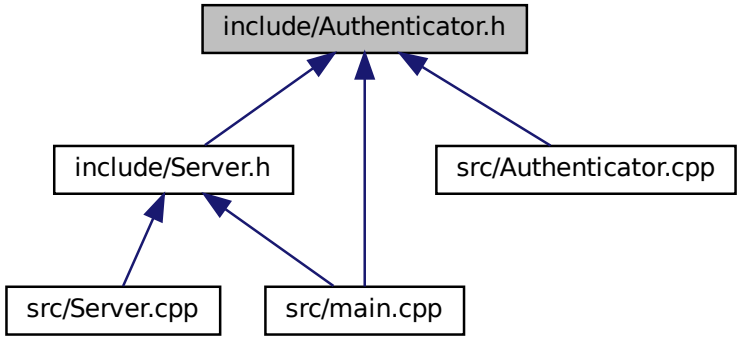
6.1 Файл include/Authenticator.h

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

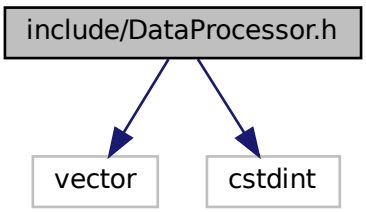
См. документацию.

```
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 };
```

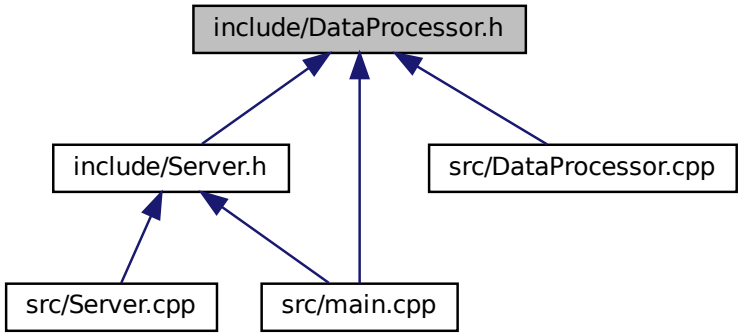
6.3 Файл include/DataProcessor.h

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

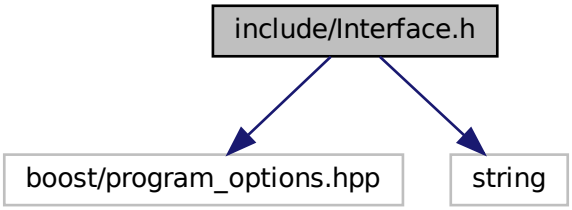
См. документацию.

```
00001
00006 #pragma once
00007 #include <vector>
00008 #include <cstdint>
00009
00010 class Logger;
00011
00016 class DataProcessor {
00017 public:
00026     int32_t calculateAverage(const std::vector<int32_t>& vector, Logger& logger);
00027 };
```

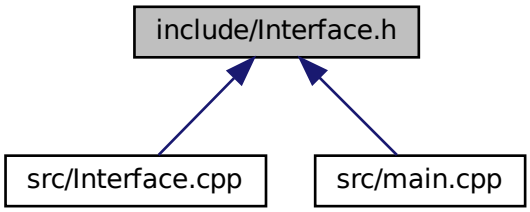
6.5 Файл include/Interface.h

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

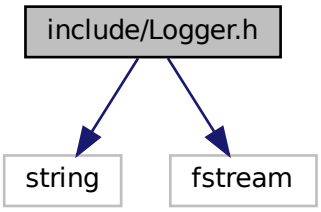
См. документацию.

```
00001
00006 #pragma once
00007 #include <boost/program_options.hpp>
00008 #include <string>
00009 namespace po = boost::program_options;
00010
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:
00036     Interface();
00037
00046     bool Parser(int argc, char** argv);
00047
00052     Params getParams()const {
00053         return params;
00054     }
00055
00059     void printHelp() const;
00060 };
```

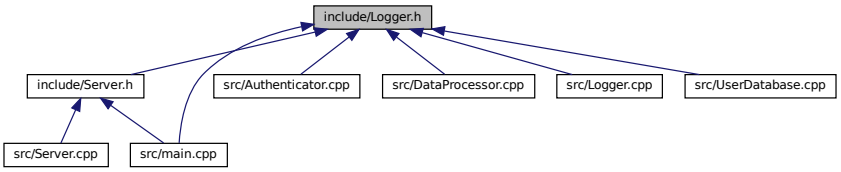
6.7 Файл include/Logger.h

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.

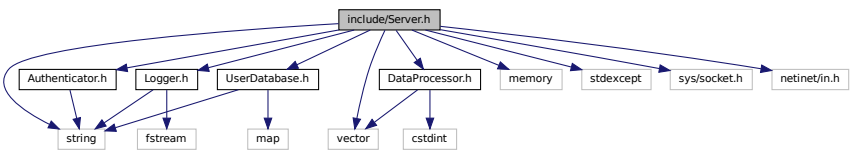
```
00001
00006 #pragma once
00007 #include <string>
00008 #include <fstream>
00009
00014 class Logger {
00015 private:
00016     std::string logPath;
00017
00018 public:
00023     void init(const std::string& log_path);
00024
00030     void logError(const std::string& message, bool isCritical = false);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

6.9 Файл include/Server.h

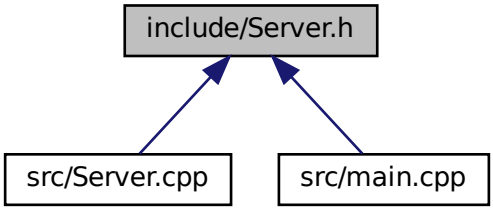
Заголовочный файл модуля Server - основной серверный модуль

```
#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.

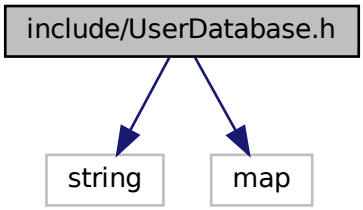
```
00001
00006 #pragma once
00007 #include "UserDatabase.h"
00008 #include "Authenticator.h"
00009 #include "DataProcessor.h"
00010 #include "Logger.h"
00011 #include <memory>
00012 #include <vector>
00013 #include <stdexcept>
00014 #include <string>
00015 #include <sys/socket.h>
00016 #include <netinet/in.h>
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 };
```

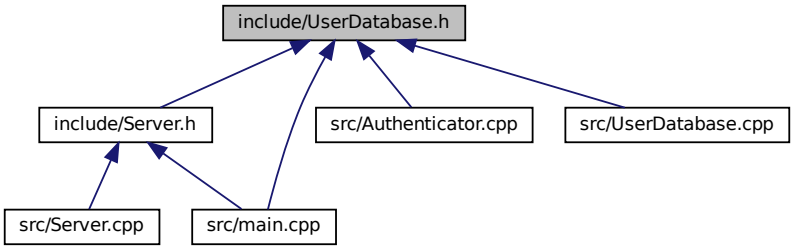
6.11 Файл include/UserDatabase.h

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.

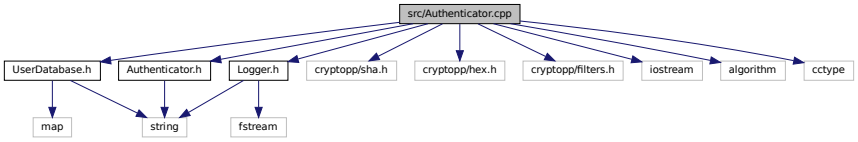
```
00001
00006 #pragma once
00007 #include <string>
00008 #include <map>
00009
00010 class Logger;
00011
00016 class UserDatabase {
00017 private:
00018     std::map<std::string, std::string> users;
00019
00020 public:
00028     bool load(const std::string& db_path, Logger& logger);
00029
00037    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

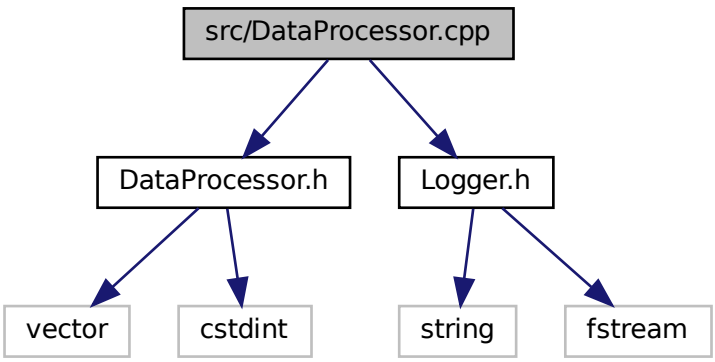
```
00001
00006 #include "Authenticator.h"
00007 #include "UserDatabase.h"
00008 #include "Logger.h"
00009 #include <cryptopp/sha.h>
00010 #include <cryptopp/hex.h>
00011 #include <cryptopp/filters.h>
00012 #include <iostream>
00013 #include <algorithm>
00014 #include <cctype>
00015
00016 namespace CPP = 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

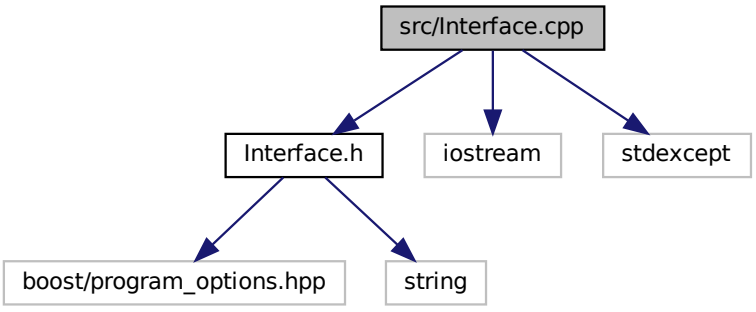
См. документацию.

```
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

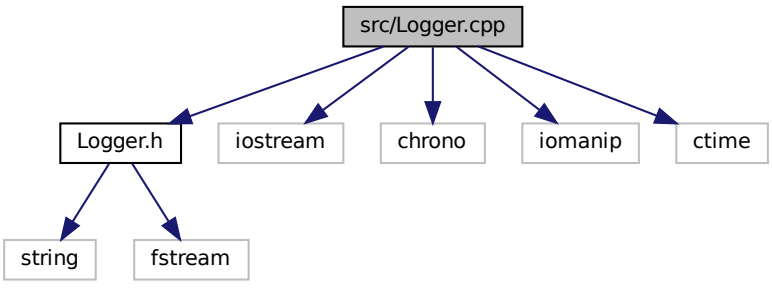
См. документацию.

```
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(33333), "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.

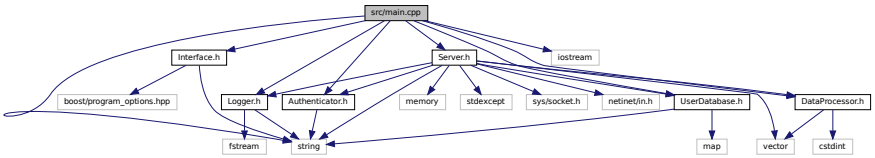
```
00001
00006 #include "Logger.h"
00007 #include <iostream>
00008 #include <chrono>
00009 #include <iomanip>
00010 #include <ctime>
00011
00016 void Logger::init(const std::string& log_path) {
00017     logPath = log_path;
00018 }
00019
00026 void Logger::logError(const std::string& message, bool isCritical) {
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.

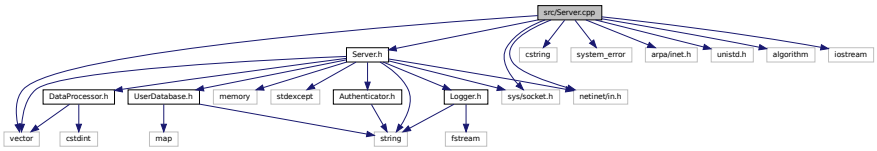
```
00001
00022 #include "Interface.h"
00023 #include "Logger.h"
00024 #include "UserDatabase.h"
00025 #include "DataProcessor.h"
00026 #include "Authenticator.h"
00027 #include "Server.h"
00028 #include <iostream>
00029 #include <string>
00030
00038 bool isValidPort(unsigned short port) {
00039     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 BUFLEN 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 BUFLEN

#define BUFLEN 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 BUFLEN 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[BUFLEN];
00086     ssize_t rc = recv(sock, buffer, BUFLEN - 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
00092     buffer[rc] = '\0';
00093     std::string message(buffer);
00094
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");
```

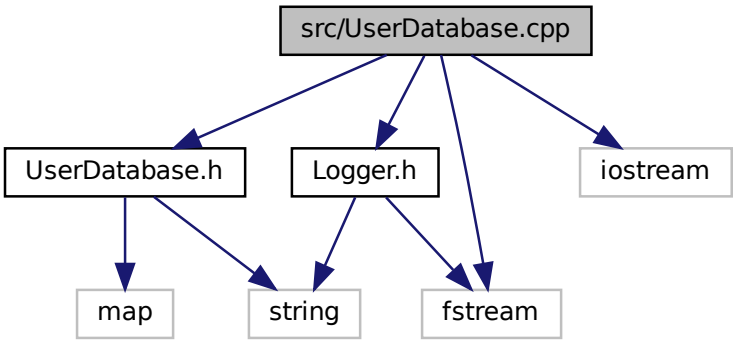
```
00109     }
00110
00111     int on = 1;
00112     if (setsockopt(listen_sock, SOL_SOCKET, SO_REUSEADDR, (const char*)&on, sizeof on) == -1) {
00113         logger.logError("setsockopt SO_REUSEADDR failed, continuing", false);
00114     }
00115
00116     self_addr->sin_family = AF_INET;
00117     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
00137 void Server::run() {
00138     startListening();
00139     socklen_t socklen = sizeof(sockaddr_in);
00140     while(true) {
00141         int work_sock = -1;
00142         try {
00143             logger.logInfo("Waiting for new client...");
00144             work_sock = accept(listen_sock, reinterpret_cast<sockaddr*>(foreign_addr.get()), &socklen);
00145             if (work_sock == -1) {
00146                 logger.logError("Accept error: " + std::string(strerror(errno)), false);continue;
00147             }
00148             std::string ip_addr(inet_ntoa(foreign_addr->sin_addr));
00149             logger.logInfo("Connection established with " + ip_addr);
00150             handleClient(work_sock);
00151         } catch (const std::exception& e) {
00152             logger.logError("Error in server loop: " + std::string(e.what()), false);
00153         }
00154         if (work_sock != -1) {
00155             close(work_sock);
00156             logger.logInfo("Connection closed");
00157         }
00158     }
00159 }
00160
00171 void Server::handleClient(int client_sock) {
00172     try {
00173         std::string full_msg = readTextMessage(client_sock);
00174         if (full_msg.empty()) {
00175             logger.logError("Client disconnected during authentication", false);
00176             return;
00177         }
00178         if (full_msg.length() < AUTH_DATA_LENGTH) {
00179             throw auth_error("Auth message too short");
00180         }
00181         std::string auth_data = full_msg.substr(full_msg.length() - AUTH_DATA_LENGTH);
00182         std::string login = full_msg.substr(0, full_msg.length() - AUTH_DATA_LENGTH);
00183
00184         if (authenticator.verify(login, auth_data, userDb, logger)) {
00185             const char* ok_msg = "OK";
00186             send(client_sock, ok_msg, strlen(ok_msg), 0);
00187             logger.logInfo("Client '" + login + "' authenticated successfully");
00188         } else {
00189             throw auth_error("Authentication failed for login " + login);
00190         }
00191         processVectors(client_sock);
00192     } catch (const auth_error& e) {
00193         sendError(client_sock, e.what());
00194         throw;
00195     } catch (const std::exception& e) {
00196         sendError(client_sock, "Protocol processing error");
00197         throw;
00198     }
00199 }
00200
00214 void Server::processVectors(int sock) {
00215     uint32_t num_vectors;
00216     ssize_t rc;
00217
00218     rc = recv(sock, &num_vectors, sizeof(num_vectors), 0);
00219     if (rc != sizeof(num_vectors)) {
00220         throw vector_error("Failed to receive number of vectors");
00221     }
00222     logger.logInfo("Receiving " + std::to_string(num_vectors) + " vectors");
00223     for (uint32_t i = 0; i < num_vectors; ++i) {
00224         uint32_t vector_len;
```

```
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