Сервер аутентификации и векторных вычислений

1.0

Создано системой Doxygen 1.9.8

Глава 1

Сервер аутентификации и векторных вычислений

Сервер принимает подключения клиентов, проводит аутентификацию и выполняет вычисления над векторами.

1.1 Использование

./server [параметры]
./server -p 33333 -c vcalc.conf -l vcalc.log

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Глава 3

Список файлов

3.1 Файлы

Полный список документированных файлов.

Глава 4

Классы

4.1 Класс ArgsParser

Класс для разбора аргументов командной строки

#include <args_parser.h>

Открытые статические члены

- static ServerConfig parse (int argc, char *argv[])
  Разбирает аргументы командной строки
- static void printHelp ()
  Выводит справку по использованию программы
- static bool validatePort (int port)
  Проверяет корректность номера порта

4.1.1 Подробное описание

Класс для разбора аргументов командной строки

Реализует парсинг параметров запуска сервера и их валидацию. Поддерживает короткие (-p) и длинные (–port) формы аргументов.

4.1.2 Методы

4.1.2.1 parse()

ServerConfig ArgsParser::parse (
int argc,
char * argv[]) [static]

Разбирает аргументы командной строки

Аргументы

argc	Количество аргументов
argv	Массив аргументов

Возвращает

ServerConfig Структура с параметрами сервера

Исключения

std::invalid_argument	При неверных аргументах

Поддерживаемые аргументы:

- -h, –help - показать справку
- -p, –port PORT - установить порт (1024-65535)
- -c, –config FILE - файл конфигурации
- -l, –log FILE - файл логов

  Проходит по всем аргументам, распознает опции и их значения. Устанавливает значения по умолчанию если опции не указаны.

Аргументы

argc	Количество аргументов
argv	Массив аргументов

Возвращает

ServerConfig Конфигурация сервера

Исключения

std::invalid_argument	при неизвестной опции или неверном порте

См. также

ServerConfig
validatePort

4.1.2.2 printHelp()

void ArgsParser::printHelp ( ) [static]

Выводит справку по использованию программы

Форматированный вывод доступных опций и их описания. Вызывается при указании -h или –help, или при ошибке парсинга.

4.1.2.3 validatePort()

```
bool ArgsParser::validatePort (
                int port )   [static]
```

Проверяет корректность номера порта

Аргументы

port	Номер порта для проверки

Возвращает

true если порт корректен, false в противном случае

Порт должен быть в диапазоне 1024-65535. Порты 0-1023 зарезервированы системой.

Аргументы

port	Номер порта для проверки

Возвращает

true если порт в допустимом диапазоне

false если порт зарезервирован или вне диапазона

Заметки

Системные порты (0-1023) требуют прав администратора

Объявления и описания членов классов находятся в файлах:

- src/args_parser.h
- src/args_parser.cpp

4.2 Класс Auth

Класс для аутентификации клиентов

#include <auth.h>

Открытые статические члены

- static bool authenticate (int clientSocket)

  Выполняет аутентификацию клиента

4.2.1 Подробное описание

Класс для аутентификации клиентов

Обеспечивает проверку учетных данных клиентов, используя механизм соли и хэширования SHA-224.

Заметки

Все методы статические - не требует создания экземпляра

4.2.2 Методы

4.2.2.1 authenticate()

```
bool Auth::authenticate (
            int clientSocket )   [static]
```

Выполняет аутентификацию клиента

Выполняет полный процесс аутентификации клиента

Аргументы

clientSocket	Дескриптор сокета клиента

Возвращает

true Аутентификация успешна

false Аутентификация не удалась

Протокол аутентификации:

1. Клиент отправляет: LOGIN(4) + SALT(16 hex) + HASH(56 hex)
2. Сервер проверяет формат
3. Сервер вычисляет и сравнивает хэш
4. Сервер отправляет OK или ERR

Аргументы

clientSocket	Дескриптор сокета подключенного клиента

Возвращает

true Клиент успешно аутентифицирован

false Ошибка аутентификации

Процесс аутентификации:

1. Получение данных от клиента (76 байт)
2. Разбор на компоненты: логин, соль, хэш
3. Валидация формата
4. Проверка учетных данных
5. Отправка результата клиенту

Заметки

Формат сообщения: LOGIN(4) + SALT(16 hex) + HASH(56 hex)
При ошибке отправляет "ERR", при успехе - "OK"

Объявления и описания членов классов находятся в файлах:

- src/auth.h
- src/auth.cpp

4.3 Класс Database

Класс для работы с базой данных пользователей

#include <database.h>

Открытые статические члены

- static bool load (const std::string &filename)
  Загружает базу пользователей из файла
- static std::string getPassword (const std::string &login)
  Получает пароль пользователя по логину
- static bool checkUser (const std::string &login, const std::string &salt, const std::string &hash)
  Проверяет учетные данные пользователя

4.3.1 Подробное описание

Класс для работы с базой данных пользователей

Хранит логины и пароли пользователей в памяти. Обеспечивает загрузку из файла и проверку учетных данных.

Заметки

Все методы класса статические - используется как синглтон

4.3.2 Методы

4.3.2.1 checkUser()

```
bool Database::checkUser (
            const std::string & login,
            const std::string & salt,
            const std::string & hash )   [static]
```

Проверяет учетные данные пользователя

Аргументы

login	Логин пользователя
salt	Соль для хэширования
hash	Предоставленный хэш пароля

Возвращает

true Учетные данные верны

false Неверный логин или пароль

Сравнивает SHA-224(salt + password) с предоставленным хэшем Сравнение регистронезависимое для hex-строк

Аргументы

login	Логин пользователя
salt	Соль для хэширования (16 hex символов)
hash	Хэш для проверки (56 hex символов)

Возвращает

true Хэш совпадает с ожидаемым

false Логин не найден или хэш не совпадает

Алгоритм проверки:

1. Получение пароля из базы
2. Вычисление SHA-224(salt + password)
3. Сравнение с предоставленным хэшем (регистронезависимо)

См. также

SHA224::hashWithSalt

4.3.2.2 getPassword()

```
std::string Database::getPassword (
            const std::string & login )  [static]
```

Получает пароль пользователя по логину

Получает пароль пользователя

Аргументы

login	Логин пользователя

Возвращает

std::string Пароль пользователя или пустая строка если не найден

Аргументы

login	Логин пользователя для поиска

Возвращает

std::string Пароль или пустая строка если пользователь не найден

Предупреждения

Возвращаемая строка может быть пустой если пользователь не существует

4.3.2.3 load()

```
bool Database::load (
            const std::string & filename )   [static]
```

Загружает базу пользователей из файла

Загружает пользователей из текстового файла

Аргументы

filename	Путь к файлу конфигурации

Возвращает

true Файл успешно загружен
false Ошибка загрузки файла

Формат файла:

- Каждая строка: логин:пароль
- Пробелы вокруг разделителя игнорируются
- Строки, начинающиеся с #, игнорируются
- Пустые строки игнорируются

Аргументы

filename	Путь к файлу конфигурации

Возвращает

true Файл успешно загружен

false Ошибка открытия или чтения файла

Пример содержимого файла:

```
user1:password1
user2:password2
# комментарий
user3:password3
```

Заметки

Тримит пробелы и табуляции вокруг логина и пароля

Очищает предыдущие данные перед загрузкой

Объявления и описания членов классов находятся в файлах:

- src/database.h
- src/database.cpp

4.4 Класс Logger

Класс логгера (синглтон) для записи событий сервера

#include <logger.h>

Открытые члены

- bool init (const std::string &filename)

  Инициализирует логгер с указанным файлом
- void log (const std::string &message, bool isCritical=false)

  Записывает сообщение в лог
- void close ()

  Закрывает файл логов

Открытые статические члены

- static Logger & getInstance ()

  Возвращает единственный экземпляр логгера

4.4.1 Подробное описание

Класс логгера (синглтон) для записи событий сервера

Обеспечивает потокобезопасное логирование в файл с метками времени. Реализован как синглтон - только один экземпляр на программу.

Заметки

Для использования вызовите Logger::getInstance()

4.4.2 Методы

4.4.2.1 close()

void Logger::close ( )

Закрывает файл логов

Вызывается автоматически в деструкторе

Вызывается при завершении работы программы или при повторной инициализации логгера.

4.4.2.2 getInstance()

Logger & Logger::getInstance ( ) [static]

Возвращает единственный экземпляр логгера

Возвращает

Logger& Ссылка на экземпляр логгера

Реализовано через статическую локальную переменную (Meyer's singleton), что обеспечивает потокобезопасность в C++11+

Возвращает

Logger& Ссылка на статический экземпляр

Заметки

Использует идиому "Meyer's singleton" Потокобезопасна в стандарте C++11 и выше

4.4.2.3 init()

bool Logger::init (
const std::string & filename)

Инициализирует логгер с указанным файлом

Аргументы

filename	Путь к файлу логов

Возвращает

true Файл успешно открыт

false Ошибка открытия файла

Заметки

Открывает файл в режиме добавления (append)

Если файл не открывается, логи выводятся в консоль

Аргументы

filename	Путь к файлу для записи логов

Возвращает

true Файл успешно открыт

false Не удалось открыть файл

Открывает файл в режиме std::ios::app (добавление в конец). Если файл не открывается, логи будут выводиться в std::cout.

4.4.2.4 log()

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

Записывает сообщение в лог

Аргументы

message	Текст сообщения
isCritical	Флаг критичности сообщения

Формат записи: "ВРЕМЯ | УРОВЕНЬ | СООБЩЕНИЕ" Уровень: INFO или CRITICAL в зависимости от isCritical

Аргументы

message	Текст сообщения для записи
isCritical	Флаг, указывающий на критичность сообщения

Формат записи: [YYYY-MM-DD HH:MM:SS | LEVEL | message]

Уровни:

- INFO (isCritical = false)
- CRITICAL (isCritical = true)

Если файл не открыт, выводит в консоль с префиксом [LOG]

Объявления и описания членов классов находятся в файлах:

- src/logger.h
- src/logger.cpp

4.5 Класс Processor

Класс для обработки векторных данных от клиентов

#include <processor.h>

Открытые статические члены

- static bool processVectors (int clientSocket)
  Обрабатывает векторные данные от клиента

4.5.1 Подробное описание

Класс для обработки векторных данных от клиентов

Вычисляет произведение элементов векторов с обработкой переполнения. Работает с данными, полученными по сети от аутентифицированных клиентов.

4.5.2 Методы

4.5.2.1 processVectors()

bool Processor::processVectors (
int clientSocket) [static]

Обрабатывает векторные данные от клиента

Аргументы

clientSocket	Дескриптор сокета клиента

Возвращает

true Обработка завершена успешно

false Ошибка при обработке

Протокол обмена:

1. Получает количество векторов (uint32_t)
2. Для каждого вектора:
   - Получает размер (uint32_t)
   - Получает данные вектора (double[])
   - Вычисляет произведение элементов
   - Отправляет результат обратно

Аргументы

clientSocket	Дескриптор сокета подключенного клиента

Возвращает

true Все векторы успешно обработаны

false Ошибка при чтении/записи данных

Алгоритм обработки:

1. Чтение количества векторов (uint32_t)
2. Для каждого вектора в цикле:
   - Чтение размера вектора (uint32_t)
   - Чтение данных вектора (массив double)
   - Вычисление произведения элементов
   - Отправка результата клиенту

Заметки

Использует MSG_WAITALL для гарантированного чтения всего блока данных

Все данные передаются в сетевом порядке байт

Объявления и описания членов классов находятся в файлах:

- src/processor.h
- src/processor.cpp

4.6 Класс Server

Класс сервера для обработки клиентских подключений

#include <server.h>

Открытые члены

- bool start (int port, const std::string &configFile)
  Запускает сервер

4.6.1 Подробное описание

Класс сервера для обработки клиентских подключений

Отвечает за:

- Создание и настройку сокета
- Ожидание входящих подключений
- Обработку клиентов в отдельных сессиях
- Загрузку базы данных пользователей

4.6.2 Методы

4.6.2.1 start()

```
bool Server::start (
            int port,
            const std::string & configFile )
```

Запускает сервер

Запускает сервер и начинает прослушивание порта

Аргументы

port	Порт для прослушивания
configFile	Путь к файлу конфигурации

Возвращает

true Сервер успешно запущен
false Ошибка запуска сервера

Последовательность действий:

1. Загрузка базы данных
2. Создание сокета
3. Привязка к порту
4. Начало прослушивания
5. Обработка входящих подключений

Аргументы

port	Порт для прослушивания
configFile	Путь к файлу с базой пользователей

Возвращает

true Сервер успешно запущен
false Ошибка при запуске сервера

Метод выполняет полную инициализацию сервера:

1. Загружает базу данных пользователей
2. Создает сокет сервера
3. Настраивает и привязывает сокет
4. Начинает прослушивание порта
5. Входит в бесконечный цикл обработки клиентов

Заметки

Использует TCP сокеты с адресом INADDR_ANY (все интерфейсы)
Включает опцию SO_REUSEADDR для быстрого перезапуска

См. также

Database::load
handleClient

Объявления и описания членов классов находятся в файлах:

- src/server.h
- src/server.cpp

4.7 Структура ServerConfig

Структура конфигурации сервера

#include <args_parser.h>

Открытые атрибуты

- int port
  Порт для прослушивания (1024-65535)
- std::string configFile
  Файл конфигурации с логинами/паролями
- std::string logFile
  Файл для записи логов
- bool showHelp
  Флаг показа справки

4.7.1 Подробное описание

Структура конфигурации сервера

Содержит параметры, полученные из аргументов командной строки.

Объявления и описания членов структуры находятся в файле:

- src/args_parser.h

4.8 Класс SHA224

Класс для вычисления SHA-224 хэшей

#include <sha224.h>

Открытые статические члены

- static std::string hash (const std::string &data)
  Вычисляет SHA-224 хэш от данных
- static std::string hashWithSalt (const std::string &salt, const std::string &password)
  Вычисляет SHA-224 от конкатенации соли и пароля
- static std::string toHex (const unsigned char *data, size_t length)
  Конвертирует бинарные данные в hex строку
- static bool isValidHex (const std::string &str)
  Проверяет, является ли строка корректной hex строкой

4.8.1 Подробное описание

Класс для вычисления SHA-224 хэшей

Предоставляет статические методы для:

- Вычисления хэшей от произвольных данных
- Вычисления хэшей с солью
- Преобразования в hex строки
- Валидации hex строк

Заметки

Использует OpenSSL EVP API (рекомендовано для OpenSSL 3.0+)

4.8.2 Методы

4.8.2.1 hash()

std::string SHA224::hash (
const std::string & data) [static]

Вычисляет SHA-224 хэш от данных

Аргументы

data	Входные данные для хэширования

Возвращает

std::string Hex строка хэша (56 символов)

Предупреждения

Возвращает пустую строку при ошибке инициализации контекста

Аргументы

data	Входные данные для хэширования

Возвращает

std::string Hex строка хэша длиной 56 символов

Пустая строка в случае ошибки создания контекста

Использует EVP API из OpenSSL, что является рекомендуемым способом в OpenSSL 3.0 и выше. Длина выходного хэша - 28 байт (224 бита), что соответствует 56 hex символам.

Заметки

Требуется линковка с libcrypto (-lcrypto)

4.8.2.2 hashWithSalt()

```
std::string SHA224::hashWithSalt (
            const std::string & salt,
            const std::string & password )   [static]
```

Вычисляет SHA-224 от конкатенации соли и пароля

Аргументы

salt	Соль для хэширования
password	Пароль пользователя

Возвращает

std::string Hex строка хэша (56 символов)

Вычисляет SHA-224(salt || password) Используется для безопасного хранения и проверки паролей

Аргументы

salt	Соль для добавления к паролю
password	Пароль пользователя

Возвращает

std::string Hex строка хэша

Конкатенирует соль и пароль, затем вычисляет хэш. Это защищает от атак с использованием радужных таблиц.

4.8.2.3 isValidHex()

bool SHA224::isValidHex (
const std::string & str) [static]

Проверяет, является ли строка корректной hex строкой

Проверяет корректность hex строки

Аргументы

str	Строка для проверки

Возвращает

true Строка содержит только hex символы (0-9, a-f, A-F)

false Строка содержит не-hex символы

Аргументы

str	Строка для проверки

Возвращает

true Строка содержит только hex символы (0-9, a-f, A-F)

false Строка пуста или содержит не-hex символы

Использует std::isxdigit для проверки каждого символа. Пустая строка считается некорректной.

4.8.2.4 toHex()

std::string SHA224::toHex (
const unsigned char * data,
size_t length) [static]

Конвертирует бинарные данные в hex строку

Аргументы

data	Указатель на бинарные данные
length	Длина данных в байтах

Возвращает

std::string Hex представление данных

Заметки

Каждый байт представляется двумя hex символами

Объявления и описания членов классов находятся в файлах:

- src/sha224.h
- src/sha224.cpp

Глава 5

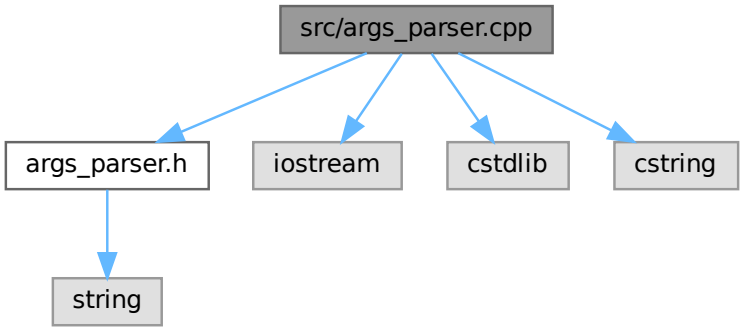
Файлы

5.1 Файл src/args_parser.cpp

Реализация парсера аргументов командной строки

```
#include "args_parser.h"
#include <iostream>
#include <cstdlib>
#include <cstring>
```

Граф включаемых заголовочных файлов для args_parser.cpp:



5.1.1 Подробное описание

Реализация парсера аргументов командной строки

Автор

Мелькаев Евгений

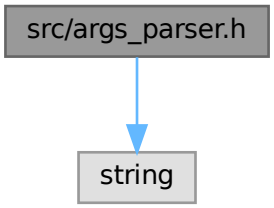
Дата

2025

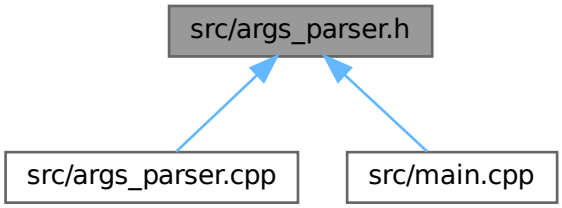
5.2 Файл src/args_parser.h

Заголовочный файл для парсера аргументов командной строки

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



Граф файлов, в которые включается этот файл:



Классы

- struct ServerConfig

  Структура конфигурации сервера
- class ArgsParser

  Класс для разбора аргументов командной строки

5.2.1 Подробное описание

Заголовочный файл для парсера аргументов командной строки

Автор

Мелькаев Евгений

Дата

2025

5.3 args_parser.h

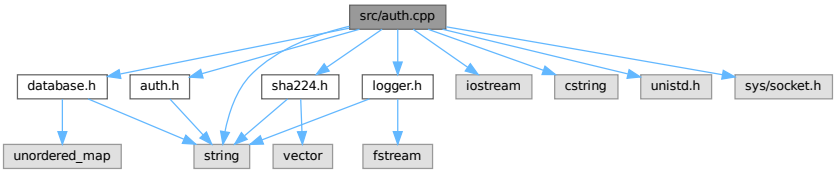
См. документацию.

```
00001
00008 #ifndef ARGS_PARSER_H
00009 #define ARGS_PARSER_H
00010
00011 #include <string>
00012
00018 struct ServerConfig {
00019     int port;
00020     std::string configFile;
00021     std::string logFile;
00022     bool showHelp;
00023 };
00024
00031 class ArgsParser {
00032 public:
00046     static ServerConfig parse(int argc, char* argv[]);
00047
00051     static void printHelp();
00052
00061     static bool validatePort(int port);
00062 };
00063
00064 #endif
```

5.4 Файл src/auth.cpp

Реализация класса аутентификации

#include "auth.h"
#include "database.h"
#include "sha224.h"
#include "logger.h"
#include <iostream>
#include <cstring>
#include <string>
#include <unistd.h>
#include <sys/socket.h>
Граф включаемых заголовочных файлов для auth.cpp:



5.4.1 Подробное описание

Реализация класса аутентификации

Автор

Мелькаев Евгений

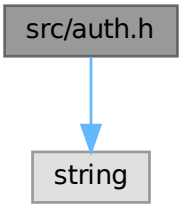
Дата

2025

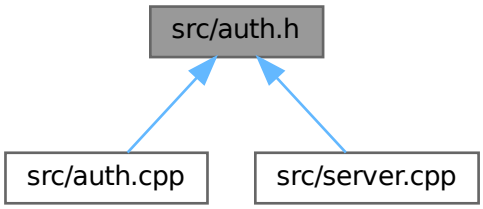
5.5 Файл src/auth.h

Заголовочный файл класса аутентификации

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



Граф файлов, в которые включается этот файл:



Классы

- class Auth

  Класс для аутентификации клиентов

5.5.1 Подробное описание

Заголовочный файл класса аутентификации

Автор

Мелькаев Евгений

Дата

2025

5.6 auth.h

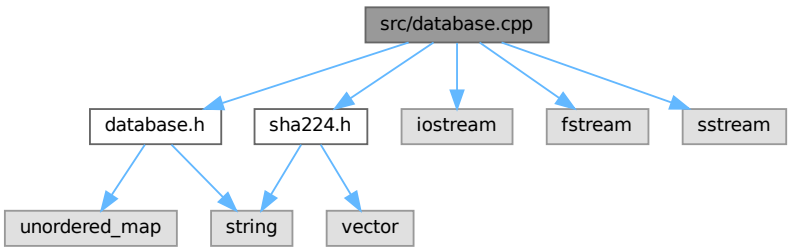
См. документацию.

```
00001
00008 #ifndef AUTH_H
00009 #define AUTH_H
00010
00011 #include <string>
00012
00021 class Auth {
00022 public:
00035     static bool authenticate(int clientSocket);
00036
00037 // Делаем методы публичными для тестов
00038 #ifdef UNIT_TESTS
00039 public:
00040 #else
00041 private:
00042 #endif
00057     static bool validateFormat(const std::string& login,
00058                                const std::string& salt,
00059                                const std::string& hash);
00060
00071     static bool verifyCredentials(const std::string& login,
00072                                   const std::string& salt,
00073                                   const std::string& receivedHash);
00074 };
00075
00076 #endif
```

5.7 Файл src/database.cpp

Реализация класса базы данных пользователей

#include "database.h"
#include "sha224.h"
#include <iostream>
#include <fstream>
#include <sstream>
Граф включаемых заголовочных файлов для database.cpp:



5.7.1 Подробное описание

Реализация класса базы данных пользователей

Автор

Мелькаев Евгений

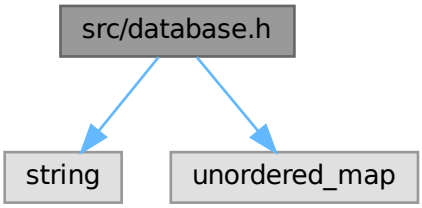
Дата

2025

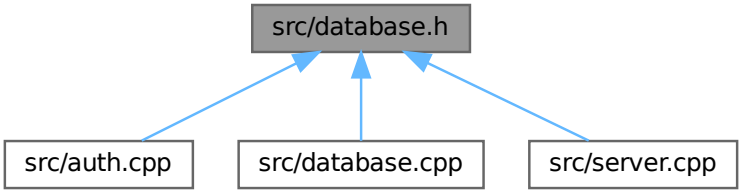
5.8 Файл src/database.h

Заголовочный файл класса базы данных пользователей

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



Граф файлов, в которые включается этот файл:



Классы

- class Database

  Класс для работы с базой данных пользователей

5.8.1 Подробное описание

Заголовочный файл класса базы данных пользователей

Автор

Мелькаев Евгений

Дата

2025

5.9 database.h

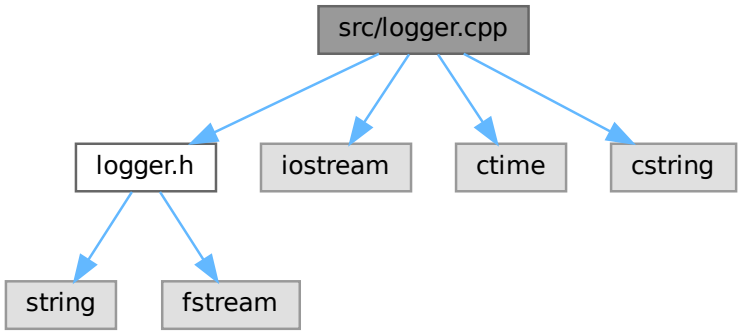
См. документацию.

```
00001
00008 #ifndef DATABASE_H
00009 #define DATABASE_H
00010
00011 #include <string>
00012 #include <unordered_map>
00013
00022 class Database {
00023 public:
00036     static bool load(const std::string& filename);
00037
00043     static std::string getPassword(const std::string& login);
00044
00056     static bool checkUser(const std::string& login,
00057                           const std::string& salt,
00058                           const std::string& hash);
00059
00060 private:
00061     static std::unordered_map<std::string, std::string> users;
00062 };
00063
00064 #endif
```

5.10 Файл src/logger.cpp

Реализация класса логгера

#include "logger.h"
#include <iostream>
#include <ctime>
#include <cstring>
Граф включаемых заголовочных файлов для logger.cpp:



5.10.1 Подробное описание

Реализация класса логгера

Автор

Мелькаев Евгений

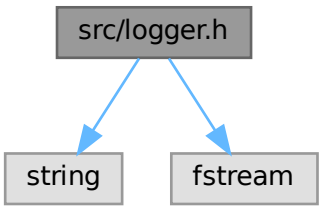
Дата

2025

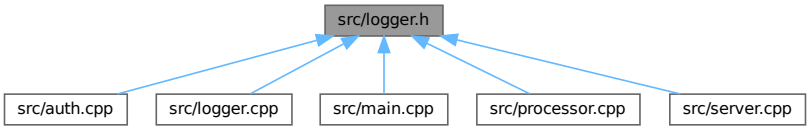
5.11 Файл src/logger.h

Заголовочный файл класса логгера

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



Граф файлов, в которые включается этот файл:



Классы

- class Logger

  Класс логгера (синглтон) для записи событий сервера

5.11.1 Подробное описание

Заголовочный файл класса логгера

Автор

Мелькаев Евгений

Дата

2025

5.12 logger.h

См. документацию.

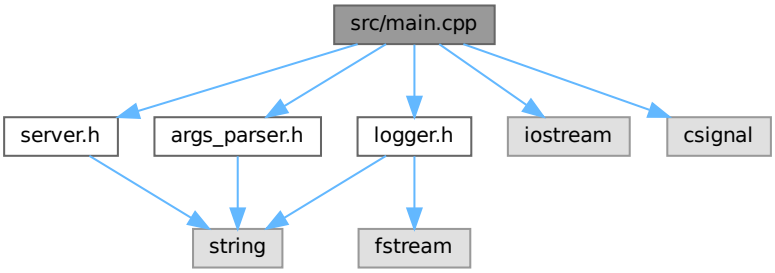
```
00001
00008 #ifndef LOGGER_H
00009 #define LOGGER_H
00010
00011 #include <string>
00012 #include <fstream>
00013
00022 class Logger {
00023 public:
00031     static Logger& getInstance();
00032
00042     bool init(const std::string& filename);
00043
00052     void log(const std::string& message, bool isCritical = false);
00053
00059     void close();
00060
00061 private:
00062     Logger() = default;
00063     ~Logger();
00064     Logger(const Logger&) = delete;
00065     Logger& operator=(const Logger&) = delete;
00066
00067     std::ofstream logFile;
00068 };
00069
00070 #endif
```

5.13 Файл src/main.cpp

Главный файл сервера аутентификации и обработки векторов

```
#include "server.h"
#include "logger.h"
#include "args_parser.h"
#include <iostream>
```

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



Функции

- void signalHandler (int signum)

  Обработчик сигналов завершения работы
- int main (int argc, char *argv[])

  Точка входа в программу

Переменные

- Server server

  Глобальный экземпляр сервера

5.13.1 Подробное описание

Главный файл сервера аутентификации и обработки векторов

5.13.2 Функции

5.13.2.1 main()

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

Точка входа в программу

Аргументы

argc	Количество аргументов командной строки
argv	Массив аргументов командной строки

Возвращает

Код завершения программы (0 - успешно, 1 - ошибка)

Выполняет:

1. Парсинг аргументов командной строки
2. Инициализацию логгера
3. Установку обработчика сигналов
4. Запуск сервера

Исключения

std::exception	При ошибках парсинга аргументов

5.13.2.2 signalHandler()

```
void signalHandler (
                int signum )
```

Обработчик сигналов завершения работы

Аргументы

signum	Номер сигнала

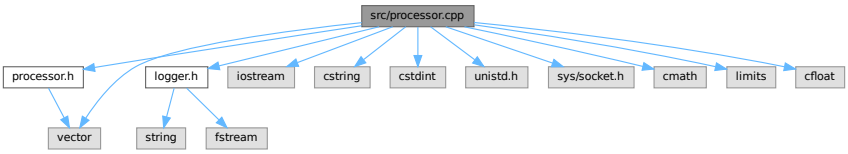
Обрабатывает сигналы SIGINT (Ctrl+C) для корректного завершения сервера. Записывает сообщение в лог и выводит информацию в консоль.

5.14 Файл src/processor.cpp

Реализация обработки векторных данных

```
#include "processor.h"
#include "logger.h"
#include <iostream>
#include <cstring>
#include <cstdint>
#include <vector>
#include <unistd.h>
#include <sys/socket.h>
#include <cmath>
#include <limits>
```

#include <cfloat>
Граф включаемых заголовочных файлов для processor.cpp:



Переменные

- const double OVERFLOW_UP = 9223372036854775807.0
  2^63 - 1 (максимальное значение при переполнении)
- const double OVERFLOW_DOWN = -9223372036854775808.0
  -2^63 (минимальное значение при переполнении)

5.14.1 Подробное описание

Реализация обработки векторных данных

Автор

Мелькаев Евгений

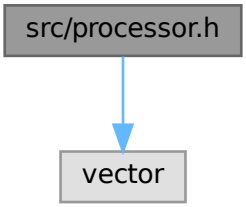
Дата

2025

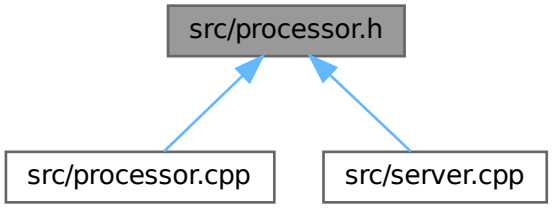
5.15 Файл src/processor.h

Заголовочный файл класса обработки векторных данных

#include <vector>
Граф включаемых заголовочных файлов для processor.h:



Граф файлов, в которые включается этот файл:



Классы

- class Processor

  Класс для обработки векторных данных от клиентов

5.15.1 Подробное описание

Заголовочный файл класса обработки векторных данных

Автор

Мелькаев Евгений

Дата

2025

5.16 processor.h

См. документацию.

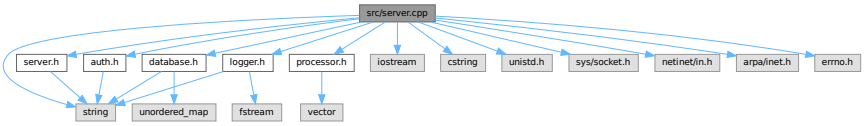
```
00001
00008 #ifndef PROCESSOR_H
00009 #define PROCESSOR_H
00010
00011 #include <vector>
00012
00019 class Processor {
00020 public:
00035     static bool processVectors(int clientSocket);
00036
00037 // Делаем методы публичными для тестов
00038 #ifdef UNIT_TESTS
00039 public:
00040 #else
00041 private:
00042 #endif
00055     static double calculateProduct(const std::vector<double>& vector);
00056 };
00057
00058 #endif
```

5.17 Файл src/server.cpp

Реализация класса сервера

```
#include "server.h"
#include "auth.h"
#include "database.h"
#include "processor.h"
#include "logger.h"
#include <iostream>
#include <cstring>
#include <string>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
```

Граф включаемых заголовочных файлов для server.cpp:



5.17.1 Подробное описание

Реализация класса сервера

Автор

Мелькаев Евгений

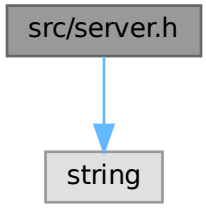
Дата

2025

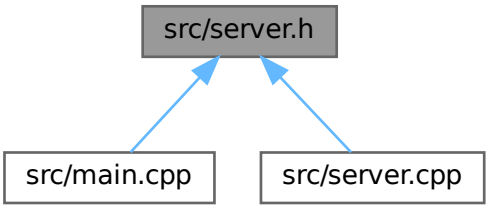
5.18 Файл src/server.h

Заголовочный файл класса сервера

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



Граф файлов, в которые включается этот файл:



Классы

- class Server

  Класс сервера для обработки клиентских подключений

5.18.1 Подробное описание

Заголовочный файл класса сервера

Автор

Мелькаев Евгений

Дата

2025

5.19 server.h

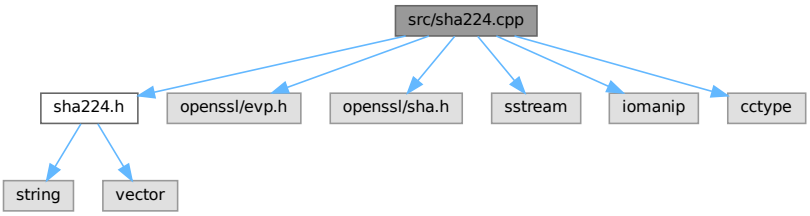
См. документацию.

```
00001
00008 #ifndef SERVER_H
00009 #define SERVER_H
00010
00011 #include <string>
00012
00022 class Server {
00023 public:
00038     bool start(int port, const std::string& configFile);
00039
00040 private:
00050     void handleClient(int clientSocket);
00051 };
00052
00053 #endif
```

5.20 Файл src/sha224.cpp

Реализация SHA-224 хэширования

#include "sha224.h"
#include <openssl/evp.h>
#include <openssl/sha.h>
#include <sstream>
#include <iomanip>
#include <cctype>
Граф включаемых заголовочных файлов для sha224.cpp:



5.20.1 Подробное описание

Реализация SHA-224 хэширования

Автор

Мелькаев Евгений

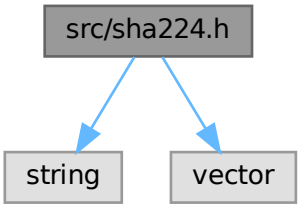
Дата

2025

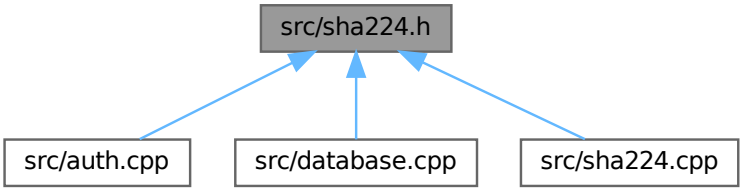
5.21 Файл src/sha224.h

Заголовочный файл для работы с SHA-224 хэшированием

#include <string>
#include <vector>
Граф включаемых заголовочных файлов для sha224.h:



Граф файлов, в которые включается этот файл:



Классы

- class SHA224

  Класс для вычисления SHA-224 хэшей

5.21.1 Подробное описание

Заголовочный файл для работы с SHA-224 хэшированием

Автор

Мелькаев Евгений

Дата

2025

5.22 sha224.h

См. документацию.

```
00001
00008 #ifndef SHA224_H
00009 #define SHA224_H
00010
00011 #include <string>
00012 #include <vector>
00013
00025 class SHA224 {
00026 public:
00034     static std::string hash(const std::string& data);
00035
00045     static std::string hashWithSalt(const std::string& salt, const std::string& password);
00046
00055     static std::string toHex(const unsigned char* data, size_t length);
00056
00063     static bool isValidHex(const std::string& str);
00064 };
00065
00066 #endif
```

Глава 6

Примеры

6.1 toHex

Аргументы

data	Указатель на бинарные данные
length	Длина данных в байтах

Возвращает

std::string Hex представление (длина = length * 2)

({0xAB, 0xCD}, 2) вернет "abcd"

Заметки

Использует std::hex и std::setfill('0') для гарантии двух символов на каждый байт.

Предметный указатель