

# Сервер обработки данных

## 1.0

Создано системой Doxygen 1.9.8



---

1 Документация сервера обработки данных	1
1.1 Введение	1
1.2 Основные возможности:	1
1.3 Использование:	1
2 README	3
2.0.1 Запуск	3
2.0.2 Справка	3
3 Алфавитный указатель классов	5
3.1 Классы	5
4 Список файлов	7
4.1 Файлы	7
5 Классы	9
5.1 Класс AuthManager	9
5.1.1 Подробное описание	10
5.1.2 Методы	10
5.1.2.1 authenticate()	10
5.1.2.2 compute_md5_hash()	10
5.1.2.3 generate_salt()	11
5.1.2.4 get_password()	11
5.1.2.5 get_users()	11
5.1.2.6 load_users()	12
5.1.2.7 user_exists()	12
5.2 Класс CommandLineParser	12
5.2.1 Подробное описание	13
5.2.2 Методы	13
5.2.2.1 get_log_file()	13
5.2.2.2 get_port()	14
5.2.2.3 get_user_db_file()	14
5.2.2.4 parse()	14
5.2.2.5 validate()	14
5.3 Класс DataCalculator	15
5.3.1 Подробное описание	15
5.3.2 Методы	16
5.3.2.1 calculate_sum_of_squares()	16
5.3.2.2 handle_overflow()	16
5.3.2.3 process_client_data()	16
5.3.2.4 read_exact()	18
5.3.2.5 send_exact()	18
5.4 Класс ErrorHandler	19
5.4.1 Подробное описание	20
5.4.2 Конструктор(ы)	20

5.4.2.1 ErrorHandler()	20
5.4.3 Методы	20
5.4.3.1 handle_auth_error()	20
5.4.3.2 handle_calculation_error()	21
5.4.3.3 handle_critical_error()	21
5.4.3.4 handle_error()	21
5.4.3.5 handle_exception()	22
5.4.3.6 handle_io_error()	22
5.4.3.7 handle_network_error()	22
5.5 Класс Logger	22
5.5.1 Подробное описание	23
5.5.2 Конструктор(ы)	23
5.5.2.1 Logger()	23
5.5.3 Методы	24
5.5.3.1 get_current_time()	24
5.5.3.2 log()	24
5.5.3.3 log_auth()	24
5.5.3.4 log_connection()	25
5.5.3.5 log_data()	25
5.5.3.6 log_debug()	25
5.5.3.7 log_error()	25
5.6 Класс Server	26
5.6.1 Подробное описание	27
5.6.2 Конструктор(ы)	27
5.6.2.1 Server()	27
5.6.3 Методы	28
5.6.3.1 get_client_ip()	28
5.6.3.2 handle_client()	28
5.6.3.3 recv_string()	28
5.6.3.4 send_string()	29
5.6.3.5 start()	29
5.7 Класс ServerInterface	30
5.7.1 Подробное описание	30
5.7.2 Методы	30
5.7.2.1 run()	30
6 Файлы	33
6.1 Файл src/AuthManager.cpp	33
6.1.1 Подробное описание	33
6.2 AuthManager.h	34
6.3 Файл src/CommandLineParser.cpp	34
6.3.1 Подробное описание	35
6.4 CommandLineParser.h	35

---

6.5	Файл <code>src/DataCalculator.cpp</code> . . . . .	35
6.5.1	Подробное описание . . . . .	36
6.6	<code>DataCalculator.h</code> . . . . .	36
6.7	Файл <code>src/ErrorHandler.cpp</code> . . . . .	37
6.7.1	Подробное описание . . . . .	37
6.8	<code>ErrorHandler.h</code> . . . . .	37
6.9	Файл <code>src/Logger.cpp</code> . . . . .	38
6.9.1	Подробное описание . . . . .	38
6.10	<code>Logger.h</code> . . . . .	39
6.11	Файл <code>src/Server.cpp</code> . . . . .	39
6.11.1	Подробное описание . . . . .	40
6.12	<code>Server.h</code> . . . . .	40
6.13	Файл <code>src/ServerInterface.cpp</code> . . . . .	41
6.13.1	Подробное описание . . . . .	41
6.14	<code>ServerInterface.h</code> . . . . .	41
	Предметный указатель . . . . .	43



# Глава 1

## Документация сервера обработки данных

### 1.1 Введение

Сервер для вычисления суммы квадратов векторов с аутентификацией пользователей.

### 1.2 Основные возможности:

- Аутентификация пользователей по MD5 хешам
- Прием и обработка векторов данных
- Вычисление суммы квадратов с проверкой переполнения
- Логирование всех событий
- Обработка ошибок через исключения

### 1.3 Использование:

```
./server --port 33333 --users users.txt --log server.log
```

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ





## Глава 2

# README

#Курсовой проект: Сервер обработки данных

#Описание Серверная часть клиент-серверной системы для обработки векторов данных с аутентификацией по MD5.

#Функциональность

- Аутентификация клиентов по логину/паролю с использованием MD5 и соли
- Прием бинарных данных (векторов чисел double)
- Вычисление суммы квадратов для каждого вектора
- Защита от переполнения
- Логирование всех событий в файл
- Конфигурация через параметры командной строки

#Сборка и запуск

Сборка  
make all

### 2.0.1 Запуск

```
cd build
#По умолчанию параметры: Порт=33333, Файл с пользователями = data/users.txt, Журнал = sever.log
./server
./server -p 33333 -u data/users.txt -l server.log
./server --port 33333 --users data/users.txt --log server.log
```

### 2.0.2 Справка

```
cd build
./server --help
./server -h
```



## Глава 3

# Алфавитный указатель классов

### 3.1 Классы

Классы с их кратким описанием.

<a href="#">AuthManager</a>	Класс для управления аутентификацией пользователей . . . . .	9
<a href="#">CommandLineParser</a>	Класс для разбора аргументов командной строки . . . . .	12
<a href="#">DataCalculator</a>	Класс для вычисления суммы квадратов векторов . . . . .	15
<a href="#">ErrorHandler</a>	Класс для обработки ошибок сервера . . . . .	19
<a href="#">Logger</a>	Класс для логирования событий сервера . . . . .	22
<a href="#">Server</a>	Основной класс сервера . . . . .	26
<a href="#">ServerInterface</a>	Интерфейс для запуска сервера . . . . .	30



## Глава 4

# Список файлов

### 4.1 Файлы

Полный список документированных файлов.

src/ <a href="#">AuthManager.cpp</a>	
Реализация класса <a href="#">AuthManager</a>	33
src/ <a href="#">AuthManager.h</a>	34
src/ <a href="#">CommandLineParser.cpp</a>	
Реализация класса <a href="#">CommandLineParser</a>	34
src/ <a href="#">CommandLineParser.h</a>	35
src/ <a href="#">DataCalculator.cpp</a>	
Реализация класса <a href="#">DataCalculator</a>	35
src/ <a href="#">DataCalculator.h</a>	36
src/ <a href="#">ErrorHandler.cpp</a>	
Реализация класса <a href="#">ErrorHandler</a>	37
src/ <a href="#">ErrorHandler.h</a>	37
src/ <a href="#">Logger.cpp</a>	
Реализация класса <a href="#">Logger</a>	38
src/ <a href="#">Logger.h</a>	39
src/ <a href="#">Server.cpp</a>	
Реализация класса <a href="#">Server</a>	39
src/ <a href="#">Server.h</a>	40
src/ <a href="#">ServerInterface.cpp</a>	
Реализация класса <a href="#">ServerInterface</a>	41
src/ <a href="#">ServerInterface.h</a>	41



## Глава 5

# Классы

### 5.1 Класс AuthManager

Класс для управления аутентификацией пользователей

```
#include <AuthManager.h>
```

Открытые члены

- AuthManager ()=default  
Конструктор по умолчанию
- bool `load_users` (const std::string &filename)  
Загрузить пользователей из файла
- bool `user_exists` (const std::string &login)  
Проверить существование пользователя
- std::string `get_password` (const std::string &login)  
Получить пароль пользователя
- std::string `generate_salt` ()  
Сгенерировать случайную соль
- bool `authenticate` (const std::string &login, const std::string &client\_hash, const std::string &salt, `Logger` &logger, const std::string &client\_ip)  
Аутентифицировать пользователя
- const std::unordered\_map< std::string, std::string > & `get_users` () const  
Получить копию базы пользователей

Открытые статические члены

- static std::string `compute_md5_hash` (const std::string &salt, const std::string &password)  
Вычислить MD5 хеш

Закрытые данные

- std::unordered\_map< std::string, std::string > users  
База пользователей (логин → пароль)

### 5.1.1 Подробное описание

Класс для управления аутентификацией пользователей

Загружает базу пользователей, генерирует соль, проверяет хеши MD5

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

### 5.1.2 Методы

#### 5.1.2.1 authenticate()

```
bool AuthManager::authenticate (
    const std::string & login,
    const std::string & client_hash,
    const std::string & salt,
    Logger & logger,
    const std::string & client_ip )
```

Аутентифицировать пользователя

Аргументы

in	login	Логин пользователя
in	client_hash	Хеш от клиента
in	salt	Соль использованная для хеширования
in	logger	Логгер для записи событий
in	client_ip	IP адрес клиента

Возвращает

true если аутентификация успешна

#### 5.1.2.2 compute\_md5\_hash()

```
std::string AuthManager::compute_md5_hash (
    const std::string & salt,
    const std::string & password ) [static]
```

Вычислить MD5 хеш



## Аргументы

in	salt	Соль
in	password	Пароль

## Возвращает

MD5 хеш в верхнем регистре

## 5.1.2.3 generate\_salt()

```
std::string AuthManager::generate_salt ( )
```

Сгенерировать случайную соль

## Возвращает

Соль в шестнадцатеричном формате (16 символов)

## Исключения

std::runtime_error	При ошибке генерации
--------------------	----------------------

## 5.1.2.4 get\_password()

```
std::string AuthManager::get_password (
    const std::string & login )
```

Получить пароль пользователя

## Аргументы

in	login	Логин пользователя
----	-------	--------------------

## Возвращает

Пароль пользователя или пустая строка если пользователь не найден

## 5.1.2.5 get\_users()

```
const std::unordered_map< std::string, std::string > & AuthManager::get_users ( ) const [inline]
```

Получить копию базы пользователей

## Возвращает

Константная ссылка на базу пользователей

### 5.1.2.6 load\_users()

```
bool AuthManager::load_users (
    const std::string & filename )
```

Загрузить пользователей из файла

Аргументы

in	filename	Имя файла с базой пользователей
----	----------	---------------------------------

Возвращает

true если успешно

Исключения

std::runtime_error	При ошибке чтения файла или отсутствии пользователей
--------------------	--

### 5.1.2.7 user\_exists()

```
bool AuthManager::user_exists (
    const std::string & login )
```

Проверить существование пользователя

Аргументы

in	login	Логин пользователя
----	-------	--------------------

Возвращает

true если пользователь существует

Объявления и описания членов классов находятся в файлах:

- src/AuthManager.h
- src/[AuthManager.cpp](#)

## 5.2 Класс CommandLineParser

Класс для разбора аргументов командной строки

```
#include <CommandLineParser.h>
```

## Открытые члены

- `CommandLineParser ()`  
Конструктор парсера командной строки
- `bool parse (int argc, char *argv[])`  
Разобрать аргументы командной строки
- `bool validate () const`  
Проверить валидность параметров
- `int get_port () const`  
Получить порт сервера
- `std::string get_user_db_file () const`  
Получить файл базы пользователей
- `std::string get_log_file () const`  
Получить файл журнала

## Закрытые данные

- `int port`  
Порт сервера
- `std::string user_db_file`  
Файл базы пользователей
- `std::string log_file`  
Файл журнала

## 5.2.1 Подробное описание

Класс для разбора аргументов командной строки

Парсит и валидирует параметры запуска сервера

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

## 5.2.2 Методы

5.2.2.1 `get_log_file()`

`std::string CommandLineParser::get_log_file () const` [inline]

Получить файл журнала

Возвращает

Путь к файлу журнала

#### 5.2.2.2 get\_port()

```
int CommandLineParser::get_port ( ) const [inline]
```

Получить порт сервера

Возвращает

Номер порта

#### 5.2.2.3 get\_user\_db\_file()

```
std::string CommandLineParser::get_user_db_file ( ) const [inline]
```

Получить файл базы пользователей

Возвращает

Путь к файлу базы пользователей

#### 5.2.2.4 parse()

```
bool CommandLineParser::parse (
    int argc,
    char * argv[] )
```

Разобрать аргументы командной строки

Аргументы

in	argc	Количество аргументов
in	argv	Массив аргументов

Возвращает

true если разбор успешен, false при ошибке или выводе справки

Исключения

boost::program_options::error	При ошибке парсинга
-------------------------------	---------------------

#### 5.2.2.5 validate()

```
bool CommandLineParser::validate ( ) const
```

Проверить валидность параметров

Возвращает

true если параметры валидны

Исключения

std::runtime_error	При ошибке валидации
--------------------	----------------------

Объявления и описания членов классов находятся в файлах:

- src/CommandLineParser.h
- src/CommandLineParser.cpp

## 5.3 Класс DataCalculator

Класс для вычисления суммы квадратов векторов

```
#include <DataCalculator.h>
```

Открытые статические члены

- static bool [read\\_exact](#) (int sock, void \*buffer, size\_t size)  
Прочитать точное количество байт из сокета
- static bool [send\\_exact](#) (int sock, const void \*buffer, size\_t size)  
Отправить точное количество байт в сокет
- static bool [process\\_client\\_data](#) (int client\_sock, class [Logger](#) &logger, const std::string &client\_ip)  
Обработать данные от клиента
- static double [calculate\\_sum\\_of\\_squares](#) (const std::vector< double > &vec)  
Вычислить сумму квадратов вектора
- static double [handle\\_overflow](#) (double value)  
Обработать переполнение значения

Закрытые статические данные

- static constexpr uint32\_t MAX\_REASONABLE\_VECTORS = 1000  
Максимальное разумное количество векторов
- static constexpr uint32\_t MAX\_REASONABLE\_VECTOR\_SIZE = 1000000  
Максимальный разумный размер вектора

### 5.3.1 Подробное описание

Класс для вычисления суммы квадратов векторов

Обрабатывает данные от клиентов, вычисляет сумму квадратов с проверкой переполнения

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

### 5.3.2 Методы

#### 5.3.2.1 calculate\_sum\_of\_squares()

```
double DataCalculator::calculate_sum_of_squares (
    const std::vector< double > & vec ) [static]
```

Вычислить сумму квадратов вектора

Аргументы

in	vec	Вектор значений
----	-----	-----------------

Возвращает

Сумма квадратов элементов вектора

Исключения

std::overflow_error	При переполнении вычислений
---------------------	-----------------------------

#### 5.3.2.2 handle\_overflow()

```
double DataCalculator::handle_overflow (
    double value ) [static]
```

Обработать переполнение значения

Аргументы

in	value	Проверяемое значение
----	-------	----------------------

Возвращает

Значение с ограничением по диапазону

Исключения

std::overflow_error	При недопустимом значении (inf/nan)
---------------------	-------------------------------------

#### 5.3.2.3 process\_client\_data()

```
bool DataCalculator::process_client_data (
    int client_sock,
    class Logger & logger,
    const std::string & client_ip ) [static]
```

Обработать данные от клиента

## Аргументы

in	client_sock	Сокет клиента
in	logger	Логгер для записи событий
in	client_ip	IP адрес клиента

## Возвращает

true если обработка успешна, false при ошибке

## 5.3.2.4 read\_exact()

```
bool DataCalculator::read_exact (
    int sock,
    void * buffer,
    size_t size ) [static]
```

Прочитать точное количество байт из сокета

## Аргументы

in	sock	Сокет для чтения
out	buffer	Буфер для данных
in	size	Количество байт для чтения

## Возвращает

true если успешно, false при ошибке

## Исключения

std::runtime_error	При ошибке чтения
--------------------	-------------------

## 5.3.2.5 send\_exact()

```
bool DataCalculator::send_exact (
    int sock,
    const void * buffer,
    size_t size ) [static]
```

Отправить точное количество байт в сокет

## Аргументы

in	sock	Сокет для отправки
in	buffer	Буфер с данными
in	size	Количество байт для отправки



Возвращает

true если успешно, false при ошибке

Исключения

std::runtime_error	При ошибке отправки
--------------------	---------------------

Объявления и описания членов классов находятся в файлах:

- src/DataCalculator.h
- src/DataCalculator.cpp

## 5.4 Класс ErrorHandler

Класс для обработки ошибок сервера

```
#include <ErrorHandler.h>
```

Открытые члены

- [ErrorHandler](#) (std::shared\_ptr< [Logger](#) > logger)  
Конструктор обработчика ошибок
- void [handle\\_network\\_error](#) (const std::string &context, int error\_code=0)  
Обработать сетевую ошибку
- void [handle\\_auth\\_error](#) (const std::string &client\_ip, const std::string &login, const std::string &reason)  
Обработать ошибку аутентификации
- void [handle\\_calculation\\_error](#) (const std::string &client\_ip, const std::string &operation)  
Обработать ошибку вычислений
- void [handle\\_io\\_error](#) (const std::string &filename, const std::string &operation)  
Обработать I/O ошибку
- void [handle\\_exception](#) (const std::exception &e, const std::string &context)  
Обработать исключение
- void [handle\\_critical\\_error](#) (const std::string &error\_message)  
Обработать критическую ошибку
- void [handle\\_error](#) (const std::string &error\_type, const std::string &message, bool is\_critical=false)  
Общий метод обработки ошибок

Закрытые данные

- std::shared\_ptr< class [Logger](#) > logger  
Логгер для записи ошибок

### 5.4.1 Подробное описание

Класс для обработки ошибок сервера

Обрабатывает различные типы ошибок: сетевые, аутентификации, вычислений, I/O

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

### 5.4.2 Конструктор(ы)

#### 5.4.2.1 ErrorHandler()

```
ErrorHandler::ErrorHandler (
    std::shared_ptr< Logger > logger ) [explicit]
```

Конструктор обработчика ошибок

Аргументы

in	logger	Общий логгер для записи ошибок
----	--------	--------------------------------

### 5.4.3 Методы

#### 5.4.3.1 handle\_auth\_error()

```
void ErrorHandler::handle_auth_error (
    const std::string & client_ip,
    const std::string & login,
    const std::string & reason )
```

Обработать ошибку аутентификации

Аргументы

in	client↔ _ip	IP клиента
in	login	Логин пользователя
in	reason	Причина ошибки

## 5.4.3.2 handle\_calculation\_error()

```
void ErrorHandler::handle_calculation_error (
    const std::string & client_ip,
    const std::string & operation )
```

Обработать ошибку вычислений

Аргументы

in	client_ip	IP клиента
in	operation	Операция, вызвавшая ошибку

## 5.4.3.3 handle\_critical\_error()

```
void ErrorHandler::handle_critical_error (
    const std::string & error_message )
```

Обработать критическую ошибку

Аргументы

in	error_message	Сообщение об ошибке
----	---------------	---------------------

Исключения

std::runtime_error	Всегда выбрасывает исключение
--------------------	-------------------------------

## 5.4.3.4 handle\_error()

```
void ErrorHandler::handle_error (
    const std::string & error_type,
    const std::string & message,
    bool is_critical = false )
```

Общий метод обработки ошибок

Аргументы

in	error_type	Тип ошибки
in	message	Сообщение об ошибке
in	is_critical	Флаг критичности ошибки (по умолчанию false)

Исключения

std::runtime_error	Если is_critical == true
--------------------	--------------------------

#### 5.4.3.5 handle\_exception()

```
void ErrorHandler::handle_exception (
    const std::exception & e,
    const std::string & context )
```

Обработать исключение

Аргументы

in	e	Исключение
in	context	Контекст, где произошло исключение

#### 5.4.3.6 handle\_io\_error()

```
void ErrorHandler::handle_io_error (
    const std::string & filename,
    const std::string & operation )
```

Обработать I/O ошибку

Аргументы

in	filename	Имя файла
in	operation	Операция (чтение/запись)

#### 5.4.3.7 handle\_network\_error()

```
void ErrorHandler::handle_network_error (
    const std::string & context,
    int error_code = 0 )
```

Обработать сетевую ошибку

Аргументы

in	context	Контекст ошибки
in	error_code	Код ошибки (0 для использования errno)

Объявления и описания членов классов находятся в файлах:

- src/ErrorHandler.h
- src/[ErrorHandler.cpp](#)

## 5.5 Класс Logger

Класс для логирования событий сервера

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

## Открытые члены

- `Logger` (const std::string &filename)  
Конструктор логгера
- `~Logger` ()  
Деструктор логгера
- void `log` (const std::string &message)  
Записать общее сообщение
- void `log_auth` (const std::string &ip, const std::string &login, bool success, const std::string &details="")  
Записать событие аутентификации
- void `log_connection` (const std::string &ip, bool connected)  
Записать событие подключения/отключения
- void `log_data` (const std::string &ip, const std::string &operation)  
Записать событие обработки данных
- void `log_error` (const std::string &error)  
Записать сообщение об ошибке
- void `log_debug` (const std::string &msg)  
Записать отладочное сообщение

## Закрытые члены

- std::string `get_current_time` ()  
Получить текущее время в формате строки

## Закрытые данные

- std::ofstream `log_file`  
Файл журнала

## 5.5.1 Подробное описание

Класс для логирования событий сервера

Записывает логи в файл и выводит в консоль

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

## 5.5.2 Конструктор(ы)

5.5.2.1 `Logger()`

```
Logger::Logger (  
    const std::string & filename ) [explicit]
```

Конструктор логгера

## Аргументы

in	filename	Имя файла журнала
----	----------	-------------------

## Исключения

std::runtime_error	При ошибке открытия файла
--------------------	---------------------------

## 5.5.3 Методы

## 5.5.3.1 get\_current\_time()

```
std::string Logger::get_current_time ( ) [private]
```

Получить текущее время в формате строки

Возвращает

Строка с текущим временем

## 5.5.3.2 log()

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

Записать общее сообщение

## Аргументы

in	message	Сообщение для записи
----	---------	----------------------

## 5.5.3.3 log\_auth()

```
void Logger::log_auth (
    const std::string & ip,
    const std::string & login,
    bool success,
    const std::string & details = "" )
```

Записать событие аутентификации

## Аргументы

in	ip	IP адрес клиента
in	login	Логин пользователя
in	success	Результат аутентификации
in	details	Детали аутентификации (по умолчанию пустая строка)

## 5.5.3.4 log\_connection()

```
void Logger::log_connection (
    const std::string & ip,
    bool connected )
```

Записать событие подключения/отключения

Аргументы

in	ip	IP адрес клиента
in	connected	Флаг подключения (true) или отключения (false)

## 5.5.3.5 log\_data()

```
void Logger::log_data (
    const std::string & ip,
    const std::string & operation )
```

Записать событие обработки данных

Аргументы

in	ip	IP адрес клиента
in	operation	Операция с данными

## 5.5.3.6 log\_debug()

```
void Logger::log_debug (
    const std::string & msg )
```

Записать отладочное сообщение

Аргументы

in	msg	Текст отладочного сообщения
----	-----	-----------------------------

## 5.5.3.7 log\_error()

```
void Logger::log_error (
    const std::string & error )
```

Записать сообщение об ошибке

Аргументы

in	error	Текст ошибки
----	-------	--------------

Объявления и описания членов классов находятся в файлах:

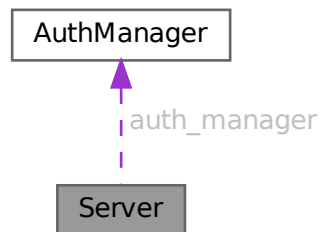
- `src/Logger.h`
- `src/Logger.cpp`

## 5.6 Класс Server

Основной класс сервера

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

Граф связей класса Server:



Открытые члены

- `std::string get_client_ip (int client_sock)`  
Получить IP адрес клиента
- `bool send_string (int sock, const std::string &str)`  
Отправить строку клиенту
- `bool recv_string (int sock, std::string &str, size_t max_len=1024)`  
Принять строку от клиента
- `bool handle_client (int client_sock)`  
Обработать подключение клиента
- `Server (int port, const std::string &user_db_file, const std::string &log_file)`  
Конструктор сервера
- `~Server ()`  
Деструктор сервера
- `bool start ()`  
Запустить сервер
- `void stop ()`  
Остановить сервер
- `void run ()`  
Основной цикл работы сервера



## Закрытые данные

- `int port`  
Порт сервера
- `std::string user_db_file`  
Файл базы пользователей
- `std::string log_file`  
Файл журнала
- `std::shared_ptr< class Logger > logger`  
Логгер
- `std::shared_ptr< class ErrorHandler > error_handler`  
Обработчик ошибок
- `class AuthManager auth_manager`  
Менеджер аутентификации
- `int server_socket`  
Сокет сервера
- `bool running`  
Флаг работы сервера

### 5.6.1 Подробное описание

#### Основной класс сервера

Управляет подключениями клиентов, аутентификацией и обработкой данных

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

### 5.6.2 Конструктор(ы)

#### 5.6.2.1 Server()

```
Server::Server (  
    int port,  
    const std::string & user_db_file,  
    const std::string & log_file )
```

Конструктор сервера

## Аргументы

in	port	Порт для прослушивания
in	user_db_file	Файл базы пользователей
in	log_file	Файл журнала

## Исключения

std::runtime_error	При ошибке инициализации
--------------------	--------------------------

## 5.6.3 Методы

## 5.6.3.1 get\_client\_ip()

```
std::string Server::get_client_ip (
    int client_sock )
```

Получить IP адрес клиента

## Аргументы

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

## Возвращает

IP адрес клиента или "unknown" при ошибке

## 5.6.3.2 handle\_client()

```
bool Server::handle_client (
    int client_sock )
```

Обработать подключение клиента

## Аргументы

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

## Возвращает

true если обработка завершена (успешно или с ошибкой)

## 5.6.3.3 recv\_string()

```
bool Server::recv_string (
    int sock,
```

```
std::string & str,  
size_t max_len = 1024 )
```

Принять строку от клиента

Аргументы

in	sock	Сокет клиента
out	str	Принятая строка
in	max_len	Максимальная длина строки

Возвращает

true если успешно, false при ошибке

#### 5.6.3.4 send\_string()

```
bool Server::send_string (  
    int sock,  
    const std::string & str )
```

Отправить строку клиенту

Аргументы

in	sock	Сокет клиента
in	str	Строка для отправки

Возвращает

true если успешно, false при ошибке

#### 5.6.3.5 start()

```
bool Server::start ( )
```

Запустить сервер

Возвращает

true если успешно, false при ошибке

Объявления и описания членов классов находятся в файлах:

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

## 5.7 Класс ServerInterface

Интерфейс для запуска сервера

```
#include <ServerInterface.h>
```

Открытые статические члены

- static int `run` (int argc, char \*argv[])  
Запустить сервер с параметрами командной строки

### 5.7.1 Подробное описание

Интерфейс для запуска сервера

Обрабатывает аргументы командной строки и запускает сервер

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

### 5.7.2 Методы

#### 5.7.2.1 `run()`

```
int ServerInterface::run (  
    int argc,  
    char * argv[] ) [static]
```

Запустить сервер с параметрами командной строки

Аргументы

in	argc	Количество аргументов
in	argv	Массив аргументов

Возвращает

Код завершения (0 - успешно, 1 - ошибка)

Объявления и описания членов классов находятся в файлах:

- `src/ServerInterface.h`
- `src/ServerInterface.cpp`



## Глава 6

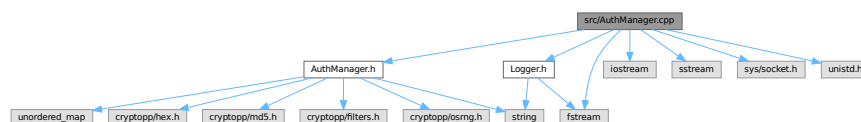
# Файлы

### 6.1 Файл src/AuthManager.cpp

Реализация класса [AuthManager](#).

```
#include "AuthManager.h"  
#include "Logger.h"  
#include <iostream>  
#include <fstream>  
#include <sstream>  
#include <sys/socket.h>  
#include <unistd.h>
```

Граф включаемых заголовочных файлов для AuthManager.cpp:



#### 6.1.1 Подробное описание

Реализация класса [AuthManager](#).

Содержит реализацию методов аутентификации, работы с солью и хеш-функциями

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

## 6.2 AuthManager.h

```

00001 #ifndef AUTHMANAGER_H
00002 #define AUTHMANAGER_H
00003
00004 #include <string>
00005 #include <unordered_map>
00006
00007 #define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1
00008 #include <cryptopp/hex.h>
00009 #include <cryptopp/md5.h>
00010 #include <cryptopp/filters.h>
00011 #include <cryptopp/osrng.h>
00012
00013 // Предварительное объявление
00014 class Logger;
00015
00021 class AuthManager {
00022 private:
00023     std::unordered_map<std::string, std::string> users;
00024
00025 public:
00027     AuthManager() = default;
00028
00033     bool load_users(const std::string& filename);
00034
00038     bool user_exists(const std::string& login);
00039
00043     std::string get_password(const std::string& login);
00044
00048     std::string generate_salt();
00049
00057     bool authenticate(const std::string& login,
00058                     const std::string& client_hash,
00059                     const std::string& salt,
00060                     Logger& logger,
00061                     const std::string& client_ip);
00062
00067     static std::string compute_md5_hash(const std::string& salt, const std::string& password);
00068
00071     const std::unordered_map<std::string, std::string>& get_users() const { return users; }
00072 };
00073
00074 #endif // AUTHMANAGER_H

```

## 6.3 Файл src/CommandLineParser.cpp

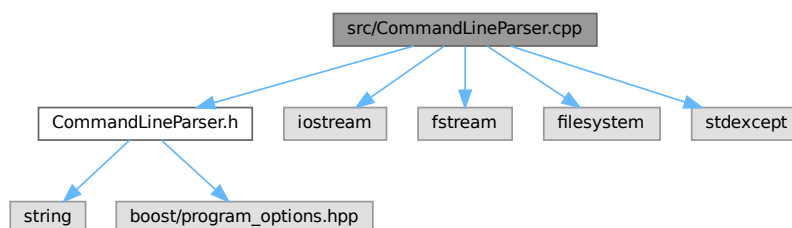
Реализация класса `CommandLineParser`.

```

#include "CommandLineParser.h"
#include <iostream>
#include <fstream>
#include <filesystem>
#include <stdexcept>

```

Граф включаемых заголовочных файлов для `CommandLineParser.cpp`:





### 6.3.1 Подробное описание

Реализация класса [CommandLineParser](#).

Содержит реализацию методов парсинга командной строки

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

## 6.4 CommandLineParser.h

```
00001 #ifndef COMMANDLINEPARSER_H
00002 #define COMMANDLINEPARSER_H
00003
00004 #include <string>
00005 #include <boost/program_options.hpp>
00006
00012 class CommandLineParser {
00013 private:
00014     int port;
00015     std::string user_db_file;
00016     std::string log_file;
00017
00018 public:
00020     CommandLineParser();
00021
00027     bool parse(int argc, char* argv[]);
00028
00032     bool validate() const;
00033
00036     int get_port() const { return port; }
00037
00040     std::string get_user_db_file() const { return user_db_file; }
00041
00044     std::string get_log_file() const { return log_file; }
00045 };
00046
00047 #endif // COMMANDLINEPARSER_H
```

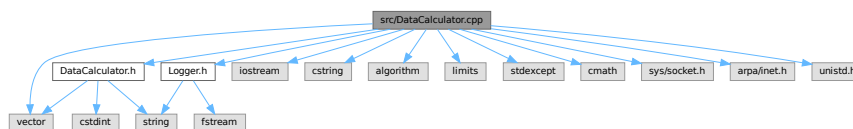
## 6.5 Файл src/DataCalculator.cpp

Реализация класса [DataCalculator](#).

```
#include "DataCalculator.h"
#include "Logger.h"
#include <iostream>
#include <cstring>
#include <algorithm>
#include <limits>
#include <vector>
#include <stdexcept>
#include <cmath>
```

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
```

Граф включаемых заголовочных файлов для DataCalculator.cpp:



## 6.5.1 Подробное описание

Реализация класса [DataCalculator](#).

Содержит реализацию методов обработки клиентских данных и их передачу

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

## 6.6 DataCalculator.h

```

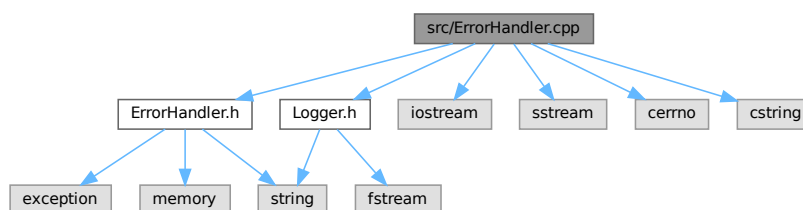
00001 #ifndef DATACALCULATOR_H
00002 #define DATACALCULATOR_H
00003
00004 #include <cstdint>
00005 #include <vector>
00006 #include <string>
00007
00013 class DataCalculator {
00014 private:
00015     static constexpr uint32_t MAX_REASONABLE_VECTORS = 1000;
00016     static constexpr uint32_t MAX_REASONABLE_VECTOR_SIZE = 1000000;
00017
00018 public:
00025     static bool read_exact(int sock, void* buffer, size_t size);
00026
00033     static bool send_exact(int sock, const void* buffer, size_t size);
00034
00040     static bool process_client_data(int client_sock, class Logger& logger, const std::string& client_ip);
00041
00046     static double calculate_sum_of_squares(const std::vector<double>& vec);
00047
00052     static double handle_overflow(double value);
00053 };
00054
00055 #endif // DATACALCULATOR_H
```

## 6.7 Файл src/ErrorHandler.cpp

Реализация класса [ErrorHandler](#).

```
#include "ErrorHandler.h"
#include "Logger.h"
#include <iostream>
#include <sstream>
#include <cerrno>
#include <cstring>
```

Граф включаемых заголовочных файлов для ErrorHandler.cpp:



### 6.7.1 Подробное описание

Реализация класса [ErrorHandler](#).

Содержит реализацию методов обработки ошибок

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

## 6.8 ErrorHandler.h

```
00001 #pragma once
00002 #include <string>
00003 #include <exception>
00004 #include <memory>
00005
00011 class ErrorHandler {
00012 private:
00013     std::shared_ptr<class Logger> logger;
00014
00015 public:
00018     explicit ErrorHandler(std::shared_ptr<Logger> logger);
00019
```

```

00023 void handle_network_error(const std::string& context, int error_code = 0);
00024
00029 void handle_auth_error(const std::string& client_ip,
00030                        const std::string& login,
00031                        const std::string& reason);
00032
00036 void handle_calculation_error(const std::string& client_ip,
00037                              const std::string& operation);
00038
00042 void handle_io_error(const std::string& filename,
00043                    const std::string& operation);
00044
00048 void handle_exception(const std::exception& e, const std::string& context);
00049
00053 void handle_critical_error(const std::string& error_message);
00054
00060 void handle_error(const std::string& error_type,
00061                 const std::string& message,
00062                 bool is_critical = false);
00063 };

```

## 6.9 Файл src/Logger.cpp

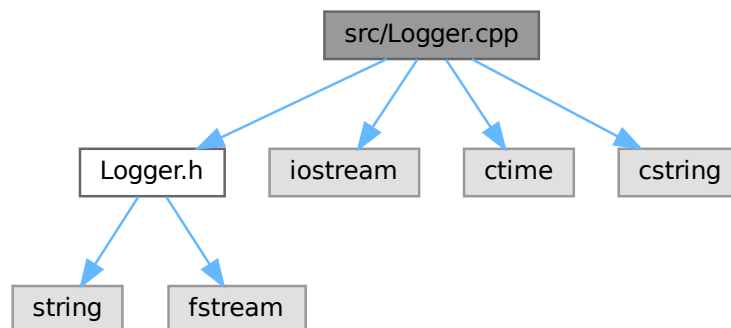
Реализация класса `Logger`.

```

#include "Logger.h"
#include <iostream>
#include <ctime>
#include <cstring>

```

Граф включаемых заголовочных файлов для `Logger.cpp`:



### 6.9.1 Подробное описание

Реализация класса `Logger`.

Содержит реализацию методов логирования работы сервера

Автор

Осетров М.С.

Дата

2025

АВТОРСТВО

ПГУ

## 6.10 Logger.h

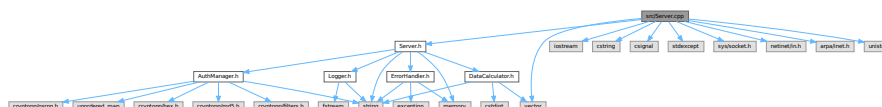
```
00001 #pragma once
00002 #include <string>
00003 #include <fstream>
00004
00010 class Logger {
00011 private:
00012     std::ofstream log_file;
00013
00016     std::string get_current_time();
00017
00018 public:
00022     explicit Logger(const std::string& filename);
00023
00025     ~Logger();
00026
00029     void log(const std::string& message);
00030
00036     void log_auth(const std::string& ip, const std::string& login,
00037                  bool success, const std::string& details = "");
00038
00042     void log_connection(const std::string& ip, bool connected);
00043
00047     void log_data(const std::string& ip, const std::string& operation);
00048
00051     void log_error(const std::string& error);
00052
00055     void log_debug(const std::string& msg);
00056 };
```

## 6.11 Файл src/Server.cpp

### Реализация класса `Server`.

```
#include "Server.h"
#include <iostream>
#include <cstring>
#include <csignal>
#include <stdexcept>
#include <vector>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
```

Граф включаемых заголовочных файлов для Server.cpp:



### 6.11.1 Подробное описание

Реализация класса [Server](#).

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

## 6.12 Server.h

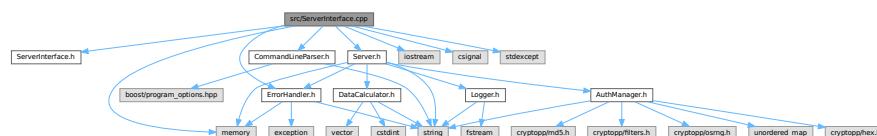
```
00001 #ifndef SERVER_H
00002 #define SERVER_H
00003
00004 #include <string>
00005 #include <memory>
00006 #include "Logger.h"
00007 #include "AuthManager.h"
00008 #include "DataCalculator.h"
00009 #include "ErrorHandler.h"
00010
00016 class Server {
00017 private:
00018     int port;
00019     std::string user_db_file;
00020     std::string log_file;
00021
00022     std::shared_ptr<class Logger> logger;
00023     std::shared_ptr<class ErrorHandler> error_handler;
00024     class AuthManager auth_manager;
00025     int server_socket;
00026     bool running;
00027
00028 public:
00032     std::string get_client_ip(int client_sock);
00033
00038     bool send_string(int sock, const std::string& str);
00039
00045     bool recv_string(int sock, std::string& str, size_t max_len = 1024);
00046
00050     bool handle_client(int client_sock);
00051
00057     Server(int port, const std::string& user_db_file, const std::string& log_file);
00058
00060     ~Server();
00061
00064     bool start();
00065
00067     void stop();
00068
00070     void run();
00071 };
00072
00073 #endif // SERVER_H
```

## 6.13 Файл src/ServerInterface.cpp

Реализация класса `ServerInterface`.

```
#include "ServerInterface.h"
#include "CommandLineParser.h"
#include "Server.h"
#include "ErrorHandler.h"
#include <iostream>
#include <csignal>
#include <memory>
#include <stdexcept>
```

Граф включаемых заголовочных файлов для `ServerInterface.cpp`:



### 6.13.1 Подробное описание

Реализация класса `ServerInterface`.

Автор

Осетров М.С.

Дата

2025

Авторство

ПГУ

## 6.14 ServerInterface.h

```
00001 #ifndef SERVERINTERFACE_H
00002 #define SERVERINTERFACE_H
00003
00009 class ServerInterface {
00010 public:
00015     static int run(int argc, char* argv[]);
00016 };
00017
00018 #endif // SERVERINTERFACE_H
```





# Предметный указатель

- authenticate
  - AuthManager, [10](#)
- AuthManager, [9](#)
  - authenticate, [10](#)
  - compute\_md5\_hash, [10](#)
  - generate\_salt, [11](#)
  - get\_password, [11](#)
  - get\_users, [11](#)
  - load\_users, [11](#)
  - user\_exists, [12](#)
- calculate\_sum\_of\_squares
  - DataCalculator, [16](#)
- CommandLineParser, [12](#)
  - get\_log\_file, [13](#)
  - get\_port, [13](#)
  - get\_user\_db\_file, [14](#)
  - parse, [14](#)
  - validate, [14](#)
- compute\_md5\_hash
  - AuthManager, [10](#)
- DataCalculator, [15](#)
  - calculate\_sum\_of\_squares, [16](#)
  - handle\_overflow, [16](#)
  - process\_client\_data, [16](#)
  - read\_exact, [18](#)
  - send\_exact, [18](#)
- ErrorHandler, [19](#)
  - ErrorHandler, [20](#)
  - handle\_auth\_error, [20](#)
  - handle\_calculation\_error, [21](#)
  - handle\_critical\_error, [21](#)
  - handle\_error, [21](#)
  - handle\_exception, [21](#)
  - handle\_io\_error, [22](#)
  - handle\_network\_error, [22](#)
- generate\_salt
  - AuthManager, [11](#)
- get\_client\_ip
  - Server, [28](#)
- get\_current\_time
  - Logger, [24](#)
- get\_log\_file
  - CommandLineParser, [13](#)
- get\_password
  - AuthManager, [11](#)
- get\_port
  - CommandLineParser, [13](#)
- get\_user\_db\_file
  - CommandLineParser, [14](#)
- get\_users
  - AuthManager, [11](#)
- handle\_auth\_error
  - ErrorHandler, [20](#)
- handle\_calculation\_error
  - ErrorHandler, [21](#)
- handle\_client
  - Server, [28](#)
- handle\_critical\_error
  - ErrorHandler, [21](#)
- handle\_error
  - ErrorHandler, [21](#)
- handle\_exception
  - ErrorHandler, [21](#)
- handle\_io\_error
  - ErrorHandler, [22](#)
- handle\_network\_error
  - ErrorHandler, [22](#)
- handle\_overflow
  - DataCalculator, [16](#)
- load\_users
  - AuthManager, [11](#)
- log
  - Logger, [24](#)
- log\_auth
  - Logger, [24](#)
- log\_connection
  - Logger, [25](#)
- log\_data
  - Logger, [25](#)
- log\_debug
  - Logger, [25](#)
- log\_error
  - Logger, [25](#)
- Logger, [22](#)
  - get\_current\_time, [24](#)
  - log, [24](#)
  - log\_auth, [24](#)
  - log\_connection, [25](#)
  - log\_data, [25](#)
  - log\_debug, [25](#)
  - log\_error, [25](#)
  - Logger, [23](#)
- parse

- CommandLineParser, [14](#)
- process\_client\_data
  - DataCalculator, [16](#)
- read\_exact
  - DataCalculator, [18](#)
- README, [3](#)
- recv\_string
  - Server, [28](#)
- run
  - ServerInterface, [30](#)
- send\_exact
  - DataCalculator, [18](#)
- send\_string
  - Server, [29](#)
- Server, [26](#)
  - get\_client\_ip, [28](#)
  - handle\_client, [28](#)
  - recv\_string, [28](#)
  - send\_string, [29](#)
  - Server, [27](#)
  - start, [29](#)
- ServerInterface, [30](#)
  - run, [30](#)
- src/AuthManager.cpp, [33](#)
- src/AuthManager.h, [34](#)
- src/CommandLineParser.cpp, [34](#)
- src/CommandLineParser.h, [35](#)
- src/DataCalculator.cpp, [35](#)
- src/DataCalculator.h, [36](#)
- src/ErrorHandler.cpp, [37](#)
- src/ErrorHandler.h, [37](#)
- src/Logger.cpp, [38](#)
- src/Logger.h, [39](#)
- src/Server.cpp, [39](#)
- src/Server.h, [40](#)
- src/ServerInterface.cpp, [41](#)
- src/ServerInterface.h, [41](#)
- start
  - Server, [29](#)
- user\_exists
  - AuthManager, [12](#)
- validate
  - CommandLineParser, [14](#)
- Документация сервера обработки данных, [1](#)