

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Информационная безопасность систем и технологий»

Курсовая работа  
по дисциплине «Технологии и методы программирования»  
на тему «Программная реализация сетевого сервера»

ПГУ.100502.С.1.О.23.КР.22ПТ115.01.ПЗ

Специальность — 10.05.02 Информационная безопасность  
телекоммуникационных система

Специализация - 7 Разработка защищенных телекоммуникационных систем

Выполнил студент: \_\_\_\_\_ Черный М.В.

Группа: 22ПТ1

Руководитель: \_\_\_\_\_ Лупанов М.Ю.

Работа защищена с оценкой \_\_\_\_\_

Дата защиты \_\_\_\_\_

УТВЕРЖДАЮ  
Зав. кафедрой ИБСТ

к.т.н., доцент

Зефилов С.Л.

18.09.2023 г.

## ЗАДАНИЕ

на курсовую работу

по теме: Программная реализация сетевого сервера

1 Дисциплина: Технологии и методы программирования.

2 Студент: Черный Михаил Вячеславович

3 Группа: 22ПТ1.

4 Исходные данные на работу:

4.1 Цель: разработка серверной программы для клиент-серверной системы обработки данных

4.2 Требования к программе:

4.2.1. Требования к выполняемым функциям

– программа должна выполнять следующие функции:

- чтение базы пользователей при старте программы;
- обеспечение возможности подключения клиента в течение всего времени функционирования;
- обработка запросов клиентов в однопоточном режиме, в том числе:
  - идентификацию и аутентификацию подключаемого клиента;
  - выполнение вычислений над данными, передаваемыми клиентом;
  - операция, выполняемая над данными — произведение вектора.

4.2.2. Требования к базе клиентов

- хранение базы клиентов в файле текстового формата;
- хранение в базе клиентов пар значений «идентификатор:пароль»;
- загрузка базы клиентов при старте сервера.

4.2.3. Требования к взаимодействию с клиентом

- реализация сеанса взаимодействия с клиентом должна состоять из следующих операций:
  - установка сеанса взаимодействия с сервером;
  - аутентификация клиента на сервере;
  - передача числовых векторов на сервер;

- получение результатов расчетов с сервера;
  - завершение сеанса;
  - для сеанса связи должен использоваться протокол TCP;
  - для аутентификации должна использоваться хэш-функция MD5
  - аутентификация должна проводиться в текстовой форме;
  - передача данных должна выполняться в двоичной форме;
  - протокол взаимодействия с сервером должен быть следующим:
    1. клиент устанавливает соединение
    2. клиент передает свой идентификатор ID
    - 3а. сервер передает случайное число  $SALT_{16}$  (при успешной идентификации)
    - 3б. сервер передает строку "ERR" и разрывает соединение(при ошибке идентификации)
    4. клиент передает  $HASH_{MD5}(SALT_{16} || PASSWORD)$
    - 5а. сервер передает ОК при успешной аутентификации
    - 5б. сервер передает строку "ERR" и разрывает соединение(при ошибке аутентификации)
  - начиная с шага 6 обмен в двоичном формате
  - 6. клиент посылает количество векторов ;
  - 7. клиент посылает размер первого вектора;
  - 8. клиент посылает все значения первого вектора одним блоком данных;
  - 9. сервер возвращает результат вычислений по первому вектору;
  - 10. шаги 7-9 повторяются для всех векторов
  - 11. клиент завершает соединение
- 4.2.4. Требования к функции аутентификации
- разрядность случайного числа  $SALT$  — 64 бита;
  - длина строки  $SALT_{16}$  — 16 шестнадцатеричных цифр;
  - метод обеспечения постоянного размера строки  $SALT_{16}$  при различных значениях числа  $SALT$  — дополнение слева цифрами «0»;
  - используемая хеш-функция — MD5;
  - представление результата хеширования — в шестнадцатеричном формате
  - допустимые шестнадцатеричные цифры — 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- 4.2.5 Требования к функции обработки данных, пересылаемых клиентом
- формат принимаемых данных — двоичный:
    - первое поле 4 байта, число типа `uint32_t`, количество векторов;

- поле «размер вектора», 4 байта, число типа `uint32_t`;
  - поле «значения вектора», последовательность полей по 8 байт, числа типа `uint32_t`, значения вектора;
  - формат передаваемых данных — двоичный:
    - поле «результат вычислений по вектору» 8 байт, число типа `uint32_t`;
  - функция обработки — произведение значений вектора;
    - при переполнении вверх возвращать значение  $2^{31} - 1$ ;
    - при переполнении вниз возвращать значение  $-2^{31}$ ;
- 4.2.6. Требования к пользовательскому интерфейсу и обработке ошибок в процессе

выполнения программы

- программа должна записывать информацию об ошибках в специальный файл журнала;
- запись файла журнала должна содержать:
  - дату и время ошибки;
  - критичность ошибки (критическая или нет);
  - параметры ошибки.
- не критические ошибки не должны приводить к аварийному завершению программы;
- управление программой должно осуществляться через параметры командной строки, передаваемые ей при старте;
- интерфейс должен обеспечивать передачу программе следующей информации:
  - имя файла с базой клиентов, необязательный;
    - значение по умолчанию `/etc/vcalc.conf`;
  - имя файла с журналом работы, необязательный;
    - значение по умолчанию `/var/log/vcalc.log`;
  - номер порта, на котором будет работать сервер, необязательный;
    - значение по умолчанию `33333`;
- программа не должна интерактивно взаимодействовать с пользователем;
- программа должна предусматривать выдачу справки о пользовательском интерфейсе
  - при запуске без параметров;
  - при запуске с параметром `-h`;

#### 4.2.7. Требования к среде функционирования программы

- программа должна функционировать в операционной системе с ядром Linux

### 5 Структура работы:

#### 5.1 Пояснительная записка (содержание работы):

- анализ требований к программе;
- построение UML-диаграммы вариантов использования и проектирование пользовательского интерфейса;
- построение UML-диаграммы классов и планирование модулей;
- построение UML-диаграммы последовательностей;
- построение UML-диаграммы деятельности;
- разработка программы;
- разработка модульных тестов и модульное тестирование;
- разработка функциональных тестов и приемочное тестирование;
- документирование кода программы с использованием Doxygen.

6 Календарный план выполнения работы:

- оформление ТЗ *4 сентября 2023 г.;*
- разработка диаграмм вариантов использования *18 сентября 2023 г.;*  
и диаграмм классов
- разработка диаграмм последовательностей и *2 октября 2023г.;*  
диаграмм деятельности
- разработка кода программы *28 октября 2023 г.;*
- разработка модульных тестов и выполнение *11 ноября 2023 г.;*  
тестирования
- разработка функциональных тестов и *25 ноября 2023 г.;*  
приемочное тестирование
- разработка документации *9 декабря 2023 г.;*
- оформление отчета о курсовой работе *11 декабря 2023 г.;*
- защита курсовой работы *23 декабря 2023 г.;*

Руководитель работы

Задание получил « »

2023г.

Студент

Нормоконтролер



М.Ю. Лупанов



М.В. Черный

М.Ю. Лупанов

## Оглавление

Введение.....	7
1 Анализ требований к программе.....	9
2 Построение UML-диаграммы вариантов использования и проектирование пользовательского интерфейса.....	11
3 Построение UML-диаграммы классов и планирование модулей.....	13
4 Построение UML-диаграммы последовательностей.....	14
5 Построение UML-диаграммы деятельности.....	16
6 Разработка программы.....	18
7 Разработка модульных тестов и модульное тестирование.....	20
8 Разработка функциональных тестов и приемочное тестирование.....	23
9 Документирование кода программы с использованием Doxygen.....	24
Заключение.....	25
Список используемых источников.....	26
ПРИЛОЖЕНИЕ А.....	27
ПРИЛОЖЕНИЕ Б.....	34
ПРИЛОЖЕНИЕ В.....	35

## Введение

С ускоренным развитием современных информационных технологий и ростом объема обрабатываемых данных в сетевых системах, необходимость эффективного взаимодействия между клиентами и серверами становится более актуальной. Разработка программного обеспечения для сетевого сервера становится ключевым элементом создания распределенных приложений, обеспечивая надежное и эффективное взаимодействие между удаленными узлами в компьютерной сети.

Объектом изучения данной курсовой работы является детальное исследование и программная реализация сетевого сервера. В рамках работы рассматриваются основные аспекты проектирования и разработки серверных приложений, включая выбор сетевого протокола, обработку запросов клиентов, обеспечение безопасности и эффективное управление ресурсами.

Используемый в данной курсовой работе язык программирования - C++, предоставляет возможность создания быстрых и эффективных приложений. Основной целью работы является разработка серверной программы для системы обработки данных в клиент-серверной архитектуре.

На первом этапе произведен анализ требований к разрабатываемой программе. Затем, на втором этапе, созданы UML-диаграммы вариантов использования и пользовательского интерфейса, что включает в себя анализ требований к программе и ее пользовательскому интерфейсу.

Третий этап включает в себя создание диаграммы классов UML и планирование модулей, а на четвертом этапе разработаны диаграммы последовательностей UML, соответствующие техническому заданию.

Пятый этап включает в себя разработку кода программы на языке C++, а на шестом этапе созданы модульные тесты для функций, отвечающих за обработку данных, работу с файлами и идентификацию пользователей.

Седьмой этап включает в себя проведение тестовых сценариев и приемочное тестирование, с использованием функциональных тестов, написанных на языке программирования bash.

На восьмом этапе осуществлена документация программы с использованием инструмента Doxygen.



## 1 Анализ требований к программе

На текущем этапе работы над курсовым проектом был проведен анализ требований к разрабатываемому программному продукту. Основные функциональные возможности программы включают в себя чтение базы данных пользователей при запуске программы, обеспечение бесперебойной работы клиента в течение всего функционирования, обработку запросов клиентов в однопоточном режиме. Эта обработка включает в себя следующие аспекты:

- идентификацию и аутентификацию подключаемого клиента;
- выполнение вычислений над данными, передаваемыми клиентом;
- выполнение операции над данными, которая, в данном случае, представляет собой суммирование элементов векторов.

Требования к базе клиентов включают хранение ее в текстовом файле в формате "идентификатор:пароль". Загрузка базы клиентов происходит при старте сервера.

Взаимодействие между сервером и клиентом представляет собой последовательность операций:

- установка сеанса взаимодействия с сервером;
- аутентификация клиента на сервере;
- передача числовых векторов на сервер;
- получение результатов расчетов с сервера;
- завершение сеанса.

Управление программой осуществляется через параметры командной строки, передаваемые при старте. В интерфейсе программы реализована возможность указания информации о:

- адресе сервера;
- порте сервера;
- пути к файлу с базой данных клиентов;

- пути к файлу журнала ошибок.

Программа не взаимодействует с пользователем интерактивно и автоматически записывает информацию об ошибках в специальный файл журнала. Запись в файл журнала включает дату и время ошибки, критичность ошибки (критическая или нет), а также параметры ошибки. Некритические ошибки не приводят к аварийному завершению программы.

Кроме того, предусмотрена возможность получения справки о пользовательском интерфейсе программы при запуске без параметров или с параметром -h.

## 2 Построение UML-диаграммы вариантов использования и проектирование пользовательского интерфейса

В процессе разработки курсовой работы, при создании UML-диаграммы вариантов использования и разработке пользовательского интерфейса, проводился анализ требований к программе и взаимодействию пользователей с ней. На данном этапе выполнялось выявление потребностей программы, определение различных сценариев взаимодействия пользователя с программой, а также описывались действия, выполняемые пользователем, и взаимодействие с сервером. Создание UML-диаграммы вариантов использования позволяло более глубоко понять требования пользователей и определить, как программное обеспечение должно быть спроектировано для эффективного удовлетворения этих потребностей.

На рисунке 1-2 представлен результат создания UML-диаграммы.

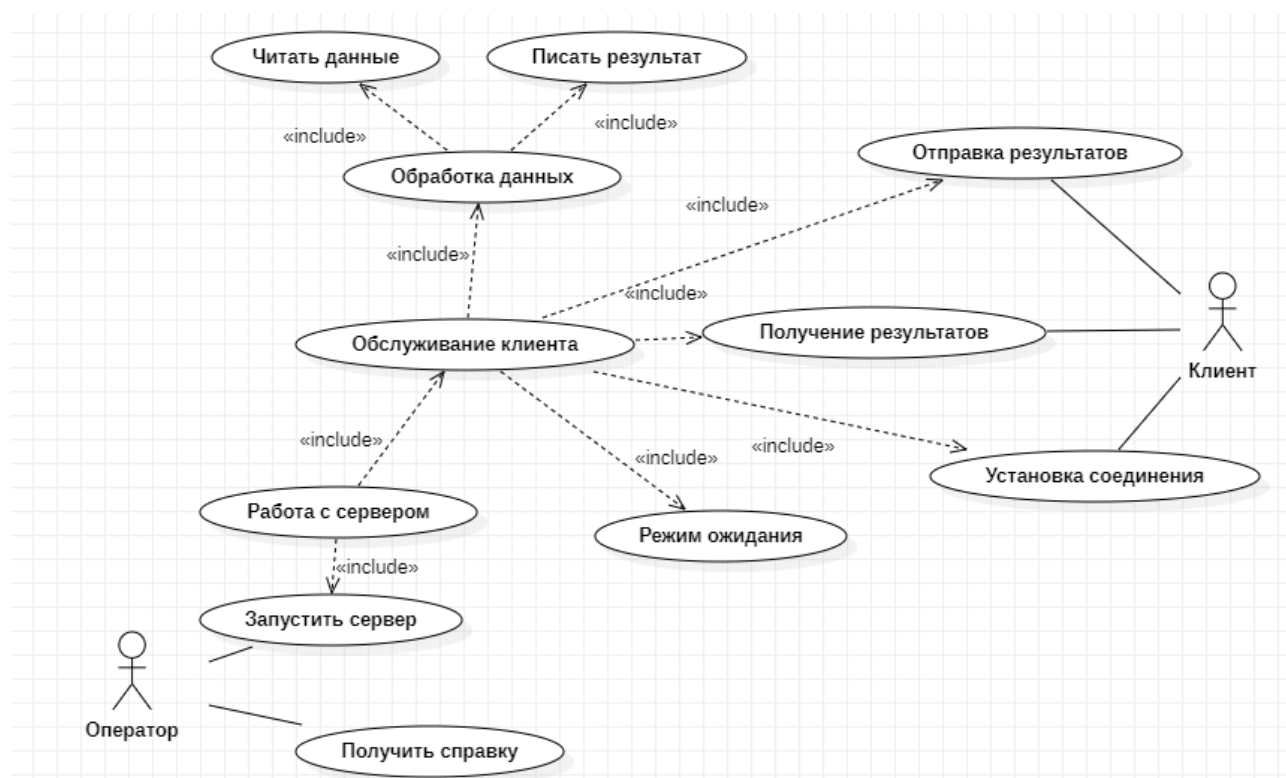


Рисунок 1 - UML-диаграмма вариантов использования

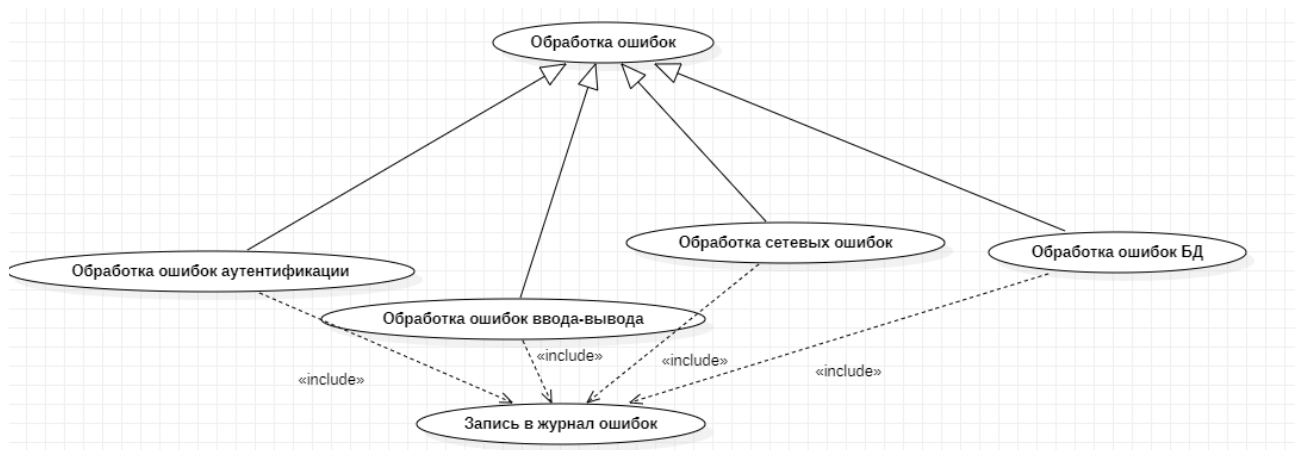


Рисунок 2 - UML-диаграмма ошибок

Также в процессе разработки был создан пользовательский интерфейс программы с целью передачи необходимой информации, такой как сетевой адрес сервера, порт сервера и другие параметры. Этот этап включал в себя разработку соответствующих элементов интерфейса, таких как текстовые поля, кнопки и другие компоненты.

Таким образом, на данном этапе были выявлены требования к программе и ее пользовательскому интерфейсу, а также созданы соответствующие диаграммы и интерфейс. Этот этап играет важную роль в процессе разработки программного обеспечения, поскольку обеспечивает понимание потребностей пользователей и проектирование программы таким образом, чтобы эффективно удовлетворить эти требования.

### 3 Построение UML-диаграммы классов и планирование модулей

На текущем этапе была разработана UML-диаграмма классов, а также выполнено планирование модулей.

На рисунке 3 представлена диаграмма классов.

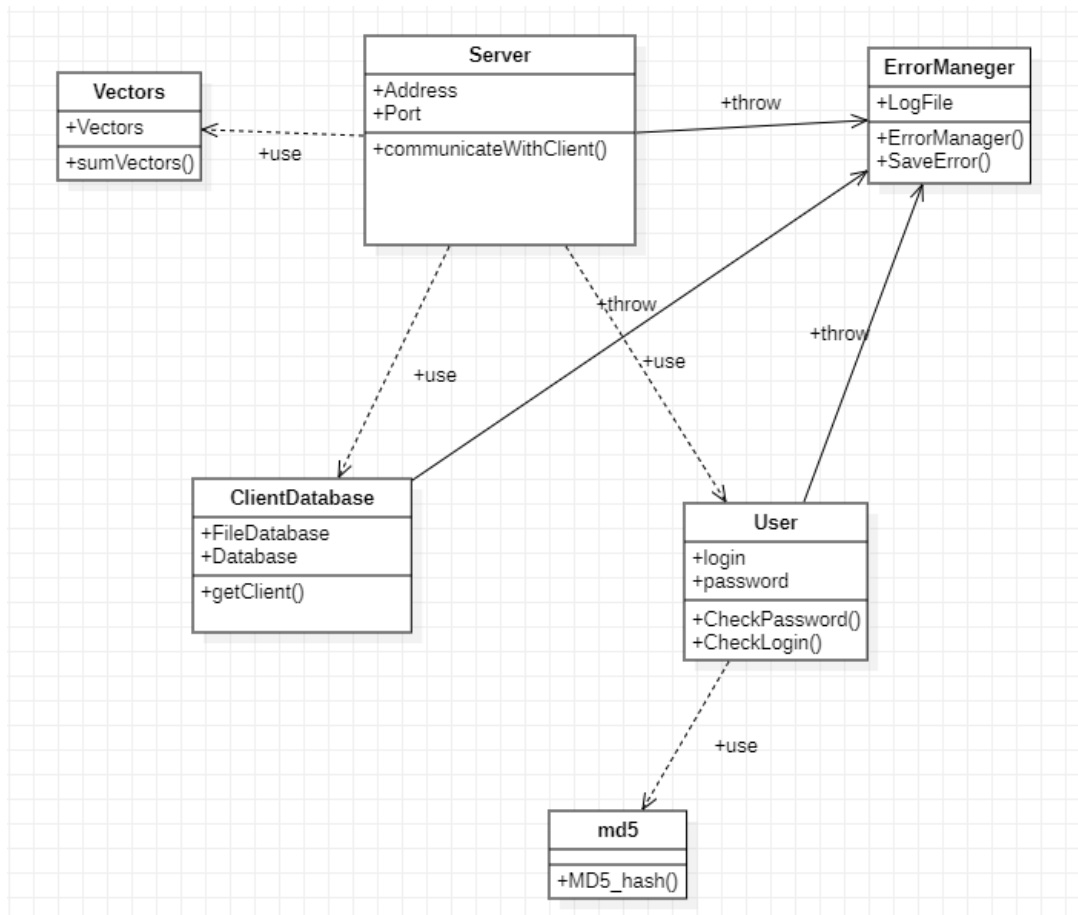


Рисунок 3 - Диаграмма классов

В UML-диаграмме есть 5 классов.

#### 4 Построение UML-диаграммы последовательностей

На данном этапе были созданы диаграммы последовательностей, которые отражают взаимодействие компонентов системы в соответствии с требованиями, изложенными в техническом задании (ТЗ). Общее количество созданных диаграмм составляет 1, а подробное изображение этой диаграммы представлено на рисунках 4.

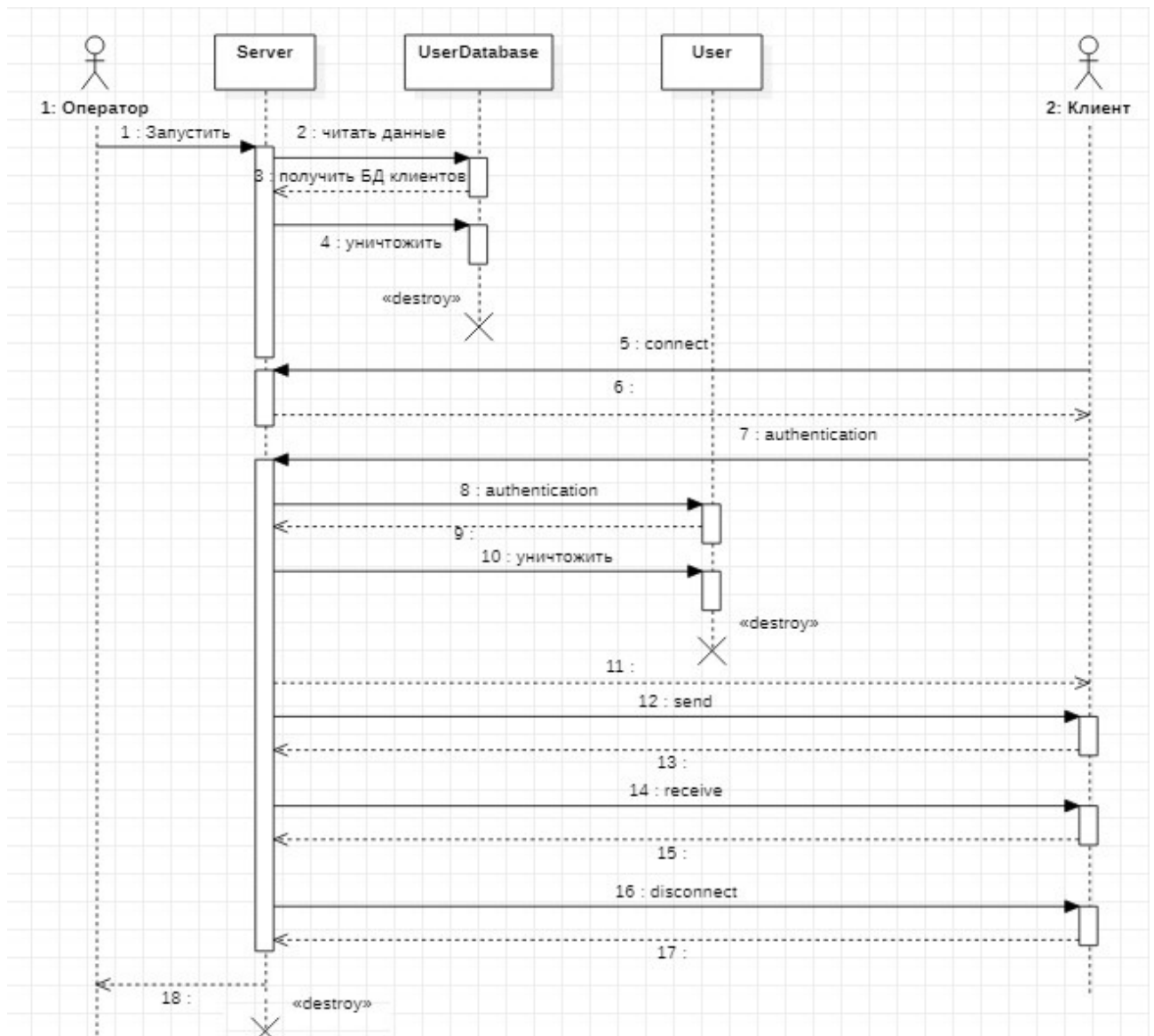


Рисунок 4 - Диаграмма последовательности

Эта диаграмма отображает взаимодействие между сервером и клиентом, а также все компоненты программы сервера и их взаимосвязь.

## 5 Построение UML-диаграммы деятельности

На данном этапе выполнено создание UML-диаграммы деятельности, которая наглядно отражает последовательность шагов и действий в рамках определенного процесса. Иллюстрация этой диаграммы представлена на рисунке 5, предоставляя четкое представление о последовательности взаимодействия компонентов системы в рамках конкретной деятельности.

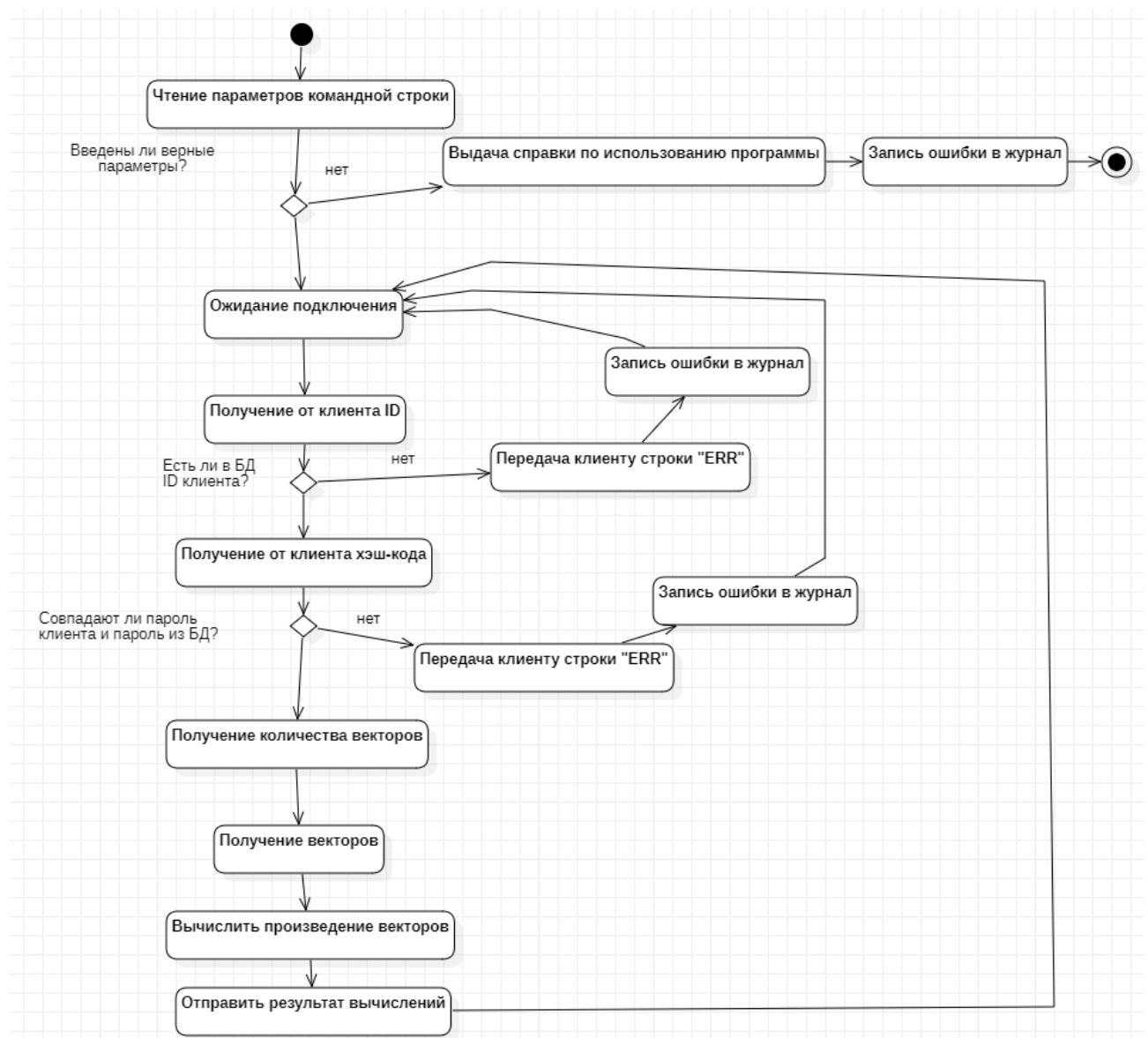


Рисунок 5 - Диаграмма деятельности



Связь между сервером и клиентом устанавливается последовательным выполнением операций, включая:

- инициализацию сеанса взаимодействия с сервером;
- процесс аутентификации клиента на сервере;
- передачу числовых векторов на сервер;
- получение результатов расчетов от сервера;
- завершение текущего сеанса.

При возникновении ошибок в процессе выполнения любой из этих операций, подробные записи о них фиксируются в соответствующем журнале.

## 6 Разработка программы

Написан код программы на языке Си++. Код представлен на [https://github.com/SigmaPro1337/Server\\_kurs.git](https://github.com/SigmaPro1337/Server_kurs.git).

В рамках разработки проекта было создано 14 файлов, каждый из которых выполняет свою уникальную функцию. Вот краткое описание каждого файла и их роли в программе:

Важные компоненты проекта включают файлы User.h, User.cpp, UserDataBase.h, UserDataBase.cpp, ErrorManager.h, ErrorManager.cpp, Server.h, и Server.cpp.

Файлы md5.cpp и md5.h реализуют алгоритм хэширования MD5, необходимый для получения хэш-кода.

Файл main.cpp содержит методы для ввода параметров командной строки от оператора.

Файл unittest.cpp предназначен для проверки модулей, из которых состоит сервер.

Файл Test.sh представляет собой скрипт для приемочного тестирования программы.

В MakeFile находятся параметры компиляции. Проект можно скомпилировать в две программы. Первая программа — сервер, а вторая программа ответственна за проверку модулей программы сервера.

Модульное тестирование осуществляется с использованием файла unittest.cpp, который проверяет работоспособность отдельных компонентов сервера.

Приемочное тестирование выполняется с использованием файла Test.sh, предназначенного для проверки программы в реальных сценариях

На изображении 6 представлен графический отчет о проверке функциональности серверной программы с использованием проверочного клиентского приложения. Все упомянутые компоненты совместно

обеспечивают исправную работу и надежность серверной программы, готовой к дальнейшему тестированию и внедрению в реальные условия эксплуатации.

## 7 Разработка модульных тестов и модульное тестирование

На текущем этапе мы создали модульные тесты для нескольких функций, занимающихся управлением файлами, обработкой данных и проверкой подлинности пользователей. Всего было разработано двенадцать тестовых сценариев:

- удачная идентификации;
- ошибка идентификации;
- удачная аутентификация;
- ошибка аутентификации;
- ошибка открытия журнала;
- ошибка открытия БД;
- ошибка открытия БД. БД пуста;
- получение БД из файла;
- ошибка в адресе сервера;
- ошибка в порту сервера;
- перемножение элементов векторов;
- переполнение вверх при перемножении элементов векторов.

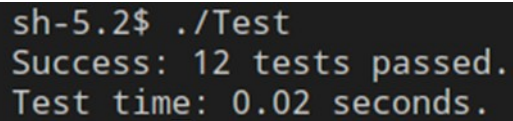
Модульное тестирование выявило несколько дефектов и неоптимальных аспектов в функционале, что способствовало повышению качества и надежности программы. Тестирование сыграло ключевую роль в улучшении проверки входных данных, исправлении ошибок и опечаток, а также обеспечении стабильной работы программы в различных сценариях использования. В таблице 1 представлены модульные тесты, снабженные подробным описанием каждого тестового сценария.

Таблица 1 — Модульное тестирование

Модуль User.h			
N	Текст	Описание	Результат
1	Удачная идентификации	Функция CheckLogin. Логин клиента и логин из БД совпадают. Функция возвращает True.	True
2	Ошибка идентификации	Функция CheckLogin. Логин клиента и логин из БД не совпадают. Функция возвращает False.	False
3	Удачная аутентификация	Функция CheckPassword. Пароль клиента и пароль из БД совпадают. Функция возвращает True.	True
4	Ошибка аутентификации	Функция CheckPassword. Пароль клиента и пароль из БД не совпадают. Функция возвращает False.	False
Модуль ErrorManager.h			
N	Текст	Описание	Результат
5	Ошибка открытия журнала	Функция ErrorManage. Ошибка в функции setLogFile. Файл для записи ошибок не может быть открыт.	Запись в журнал
Модуль UserDataBase.h			
6	Ошибка открытия БД	Функция getClientCredentials. В пути к файлу используются не допустимые символы.	Запись в журнал
7	Ошибка открытия БД	Функция getClientCredentials. Файл с предполагаемой БД пуст.	Запись в журнал
8	Получение БД из файла	Функция getClientCredentials. Получение векторов firstVector, secondVector.	Получение двух векторов
Модуль Server.h			
9	Ошибка в адресе сервера	Функция setAddress. Адрес сервера не равен «127.0.0.1».	Запись в журнал
10	Ошибка в порту сервера	Функция setPort. Порт сервера должен быть больше 1023.	Запись в журнал
11	Перемножение элементов векторов	Функция multiplyVectors.	Результат
12	Переполнение вверх при перемножении элементов векторов	Функция multiplyVectors. Переполнение вверх.	2147483647

Тестовые сценарии для разработанных модулей были описаны с применением фреймворка UnitTest++. Результаты проведенного тестирования

представлены на рисунке 6. Приложение А включает в себя тестовый сценарий, выполненный с использованием данного фреймворка.

A terminal window with a dark background and light-colored text. The text shows a command being executed and its output.

```
sh-5.2$ ./Test  
Success: 12 tests passed.  
Test time: 0.02 seconds.
```

Рисунок 6 - Модульное тестирование

## 8 Разработка функциональных тестов и приемочное тестирование

Разработаны и выполнены приемочные тесты с использованием языка программирования `bash`. В приложении Б приведен код функциональных тестов.

Создано шесть тестовых сценариев. Листинг программы функционального тестирования расположен в приложении В.

Разработанная программа успешно проходит все функционального тесты.

## RUNNING TESTS

### TEST N1

идентификатор: user, пароль: P@ssW0rd, адрес: 127.0.0.1, порт: 33333  
vector N0, vector len 2: { 1 2 }  
vector N1, vector len 2: { 3 4 }  
Ожидается число SALT. Получено число SALT.  
Ожидается код аутентификации. Получен код: OK.  
Получен результат: 2  
Получен результат: 12

### TEST N2

идентификатор: ivanov, пароль: 123456, адрес: 127.0.0.1, порт: 33333  
vector N0, vector len 2: { 11 13 }  
vector N1, vector len 3: { 31 32 334 }  
Ожидается число SALT. Получено число SALT.  
Ожидается код аутентификации. Получен код: OK.  
Получен результат: 143  
Получен результат: 331328

### TEST N3

идентификатор: ivanov2, пароль: 123456, адрес: 127.0.0.1, порт: 33333  
vector N0, vector len 2: { 11 13 }  
vector N1, vector len 3: { 31 32 334 }  
Ожидается число SALT. Получен код: ERR.

### TEST N4

идентификатор: user0, пароль: qwe123456, адрес: 127.0.0.1, порт: 33333  
vector N0, vector len 2: { 1 2 }  
vector N1, vector len 2: { 3 4 }  
Ожидается число SALT. Получено число SALT.  
Ожидается код аутентификации. Получен код: ERR.

### TEST N5

идентификатор: user, пароль: P@ssW0rd, адрес: 127.0.0.1, порт: 33334  
vector N0, vector len 2: { 1 2 }  
vector N1, vector len 2: { 3 4 }  
Error: Ошибка при подключении

### TEST N6

идентификатор: user, пароль: P@ssW0rd, адрес: 127.0.0.12, порт: 33333  
vector N0, vector len 2: { 1 2 }  
vector N1, vector len 2: { 3 4 }  
Error: Ошибка при подключении

Рисунок 7 - Функциональное тестирование



## 9 Документирование кода программы с использованием Doxygen.

В рамках данного проекта была разработана серверная программа, взаимодействующая с клиентами посредством протокола TCP. Для подробной документации проекта был выбран инструмент Doxygen.

После успешной установки и настройки Doxygen в исходный код программы были добавлены специальные комментарии, предоставляющие Doxygen необходимую информацию для автоматической генерации документации. Doxygen предоставляет возможность создавать документацию в различных форматах, таких как HTML, LaTeX, RTF, PDF и XML. Сгенерированная документация содержит полные описания всех элементов программы, включая их характеристики, параметры, возвращаемые значения и примеры использования.

Использование Doxygen значительно облегчило процесс восприятия структуры программы, помогло разработчикам в использовании ее компонентов и внесении изменений в код. Эта документация также упрощает процессы сопровождения и обновления проекта. Таким образом, внедрение Doxygen для документирования проекта представляет важный шаг в разработке программного обеспечения, способствуя улучшению понимания кода, снижению рисков возникновения ошибок и ускорению процесса разработки. Полный текст разработанной документации предоставлен в Приложении В.

## Заключение

В ходе выполнения курсовой работы была разработана система клиент-серверного взаимодействия, предназначенная для использования сетевого протокола ТСР. Создан сервер, способный принимать подключения от клиентов, а также клиентское приложение, обеспечивающее обмен данными с сервером, отправку информации и получение результатов вычислений.

Этапы разработки включали в себя анализ требований, проектирование структуры программы, написание кода, модульное тестирование функционала, приемочное тестирование и, наконец, документирование проекта при использовании инструмента Doxygen.

Модульное тестирование выявило и устранило ошибки в функционале, что существенно повысило уровень надежности и стабильности программы. Функциональное тестирование подтвердило соответствие программы всем поставленным требованиям. Сгенерированная документация с использованием Doxygen предоставляет подробное описание всех компонентов программы, что упрощает процессы разработки, обновления и поддержки кода.

Курсовая работа предоставила возможность применить полученные знания в разработке сетевых приложений, тестировании и документировании, а также получить опыт работы с инструментами разработки и отладки. В результате была создана функциональная и надежная система, готовая к дальнейшему развитию и использованию.

Задание на курсовую работу было выполнено в полном объеме.

## Список используемых источников

- 1 Справочник по языку C++ [Электронный ресурс]: Microsoft. – URL: <https://docs.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-160>
- 2 Технология разработки программного обеспечения : учеб. пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Виснадул – URL: <https://znanium.com/catalog/document?pid=768473>
- 3 Введение в архитектуру программного обеспечения: Учебное пособие / Гагарина Л.Г., Федоров А.Р., Федоров П.А. – URL: <http://znanium.com/bookread2.php?book=542665>
- 4 Архитектура и проектирование программных систем: Монография / С.В. Назаров – URL: <http://znanium.com/bookread2.php?book=542562>
- 5 Ховард, М. 19 смертных грехов, угрожающих безопасности программ. Как не допустить типичных ошибок – URL: <https://e.lanbook.com/book/1118>
- 6 Буч, Г. Язык UML. Руководство пользователя – URL: <https://e.lanbook.com/book/1246>
- 7 Энтони, У. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ – URL: <https://e.lanbook.com/book/4813>

## ПРИЛОЖЕНИЕ А

(обязательное)

### Модульное тестирование

Модуль User.h			
N	Текст	Описание	Результат
1	Удачная идентификация	Функция CheckLogin. Логин клиента и логин из БД совпадают. Функция возвращает True.	True
2	Ошибка идентификации	Функция CheckLogin. Логин клиента и логин из БД не совпадают. Функция возвращает False.	False
3	Удачная аутентификация	Функция CheckPassword. Пароль клиента и пароль из БД совпадают. Функция возвращает True.	True
4	Ошибка аутентификации	Функция CheckPassword. Пароль клиента и пароль из БД не совпадают. Функция возвращает False.	False
Модуль ErrorManager.h			
N	Текст	Описание	Результат
5	Ошибка открытия журнала	Функция ErrorManage. Ошибка в функции setLogFile. Файл для записи ошибок не может быть открыт.	Запись в журнал
Модуль UserDataBase.h			
6	Ошибка открытия БД	Функция getClientCredentials. В пути к файлу используются не допустимые оши.	Запись в журнал
7	Ошибка открытия БД	Функция getClientCredentials. Файл с предполагаемой БД пуст.	Запись в журнал
8	Получение БД из файла	Функция getClientCredentials. Получение векторов firstVector, secondVector.	Получение двух векторов
Модуль Server.h			
9	Ошибка в адресе	Функция setAddress. Адрес сервера не	Запись в

	сервера	равен «127.0.0.1».	журнал
10	Ошибка в порту сервера	Функция setPort. Порт сервера должен быть больше 1023.	Запись в журнал
11	Перемножение элементов векторов	Функция multiplyVectors.	Результат
12	Переполнение вверх при перемножении элементов векторов	Функция multiplyVectors. Переполнение вверх.	2147483647

## ПРИЛОЖЕНИЕ Б

(обязательное)

### Функциональное тестирование

N	Описание	Шаги программы-клиента	Настройки работы сервера
1	Удачное взаимодействие 1, пользователь user	<ol style="list-style-type: none"> <li>1. Установка соединения с сервером по адресу 127.0.0.1 и порту 33333</li> <li>2. Отправка идентификатора user</li> <li>3. Получение числа <math>SALT_{16}</math></li> <li>4. Отправка <math>HASH_{md5}(SALT_{16}  P@ssW0rd)</math></li> <li>5. Получение кода ОК</li> <li>6. Отправка значений: 2 2 1 2 (два вектора; длина первого вектора 2, его значение {2 1})</li> <li>7. Получение результата по первому вектору равно 2</li> <li>8. Отправка значений: 2 3 4 (длина второго вектора 2, значение {3 4})</li> <li>9. Получение результата по второму вектору равно 12</li> <li>10. Закрытие соединения</li> </ol>	Адрес: 127.0.0.1 Порт: 33333
2	Удачное взаимодействие 2, пользователь ivanov	<ol style="list-style-type: none"> <li>1. Установка соединения с сервером по адресу 127.0.0.1 и порту 33333</li> <li>2. Отправка идентификатора ivanov</li> <li>3. Получение числа <math>HASH_{md5}</math></li> <li>4. Отправка <math>HASH_{md5}(SALT_{16}  123456)</math></li> <li>5. Получение кода ОК</li> <li>6. Отправка значений: 2 2 11 13 (два вектора; длина первого вектора 2, его значение {11 13})</li> <li>7. Получение результата по первому вектору равно 143</li> <li>8. Отправка значений: 3 31 32 334 (длина второго вектора 3,</li> </ol>	Адрес: 127.0.0.1 Порт: 33333

		<p>значение {31 32 334})</p> <p>9. Получение результата по второму вектору равное 331328</p> <p>10.Закрытие соединения</p>	
3	Неудачное взаимодействие 1, неправильный логин	<p>1. Установка соединения с сервером по адресу 127.0.0.1 и порту 33333</p> <p>2. Отправка идентификатора ivanov2</p> <p>3. Получение ошибки ERR</p> <p>4. Завершение соединения</p>	<p>Адрес: 127.0.0.1</p> <p>Порт: 33333</p>
4	Неудачное взаимодействие 2, неправильный пароль	<p>1. Установка соединения с сервером по адресу 127.0.0.1 и порту 33333</p> <p>2. Отправка идентификатора user0</p> <p>3. Получение числа SALT</p> <p>4. Отправка HASHmd5 (SALT    qwe123456 )</p> <p>5. Получение ошибки ERR</p> <p>6. Завершение соединения</p>	<p>Адрес: 127.0.0.1</p> <p>Порт: 33333</p>
5	Неудачное взаимодействие 3, неправильный порт подключения	<p>1. Установка соединения с сервером по адресу 127.0.0.1 и порту 33334</p> <p>2. Ошибка соединения</p>	<p>Адрес: 127.0.0.1</p> <p>Порт: 33333</p>
6	Неудачное взаимодействие 4, неправильный адрес подключения	<p>1. Установка соединения с сервером по адресу 127.0.0.12 и порту 33333</p> <p>2. Ошибка соединения</p>	<p>Адрес: 127.0.0.1</p> <p>Порт: 33333</p>

## ПРИЛОЖЕНИЕ В

(обязательное)

Doxygen



# Руководство по эксплуатации серверной программы

## 1.0

Создано системой Doxygen 1.9.1



1 Алфавитный указатель классов	1
1.1 Классы	1
2 Список файлов	3
2.1 Файлы	3
3 Классы	5
3.1 Класс ErrorManager	5
3.1.1 Подробное описание	5
3.1.2 Методы	5
3.1.2.1 ErrorManage()	6
3.2 Класс Server	6
3.2.1 Подробное описание	6
3.2.2 Методы	7
3.2.2.1 handleClientInteraction()	7
3.2.2.2 multiplyVectors()	7
3.3 Класс User	7
3.3.1 Подробное описание	8
3.3.2 Методы	8
3.3.2.1 CheckLogin()	8
3.3.2.2 CheckPassword()	9
3.4 Класс UserDataBase	9
3.4.1 Подробное описание	9
3.4.2 Методы	9
3.4.2.1 getClientCredentials()	10
4 Файлы	11
4.1 Файл ErrorManager.cpp	11
4.2 Файл ErrorManager.h	11
4.2.1 Подробное описание	11
4.3 Файл main.cpp	12
4.4 Файл Server.cpp	12
4.5 Файл Server.h	12
4.5.1 Подробное описание	13
4.6 Файл User.cpp	13
4.7 Файл User.h	13
4.7.1 Подробное описание	14
4.8 Файл UserDataBase.cpp	14
4.9 Файл UserDataBase.h	14
4.9.1 Подробное описание	15
Предметный указатель	17



# Глава 1

## Алфавитный указатель классов

### 1.1 Классы

Классы с их кратким описанием.

<a href="#">ErrorManager</a>	Класс обработки ошибок ErrorManage . . . . .	5
<a href="#">Server</a>	Основной класс для взаимодействия клиента с сервером . . . . .	6
<a href="#">User</a>	Класс, содержащий информацию о текущем пользователе . . . . .	7
<a href="#">UserDataBase</a>	Класс, предназначенный для извлечения информации из базы данных клиентов .	9



## Глава 2

# Список файлов

### 2.1 Файлы

Полный список документированных файлов.

<a href="#">ErrorManager.cpp</a>	11
<a href="#">ErrorManager.h</a>	
Заголовочный файл для модуля <a href="#">ErrorManager</a>	11
<a href="#">main.cpp</a>	12
<a href="#">Server.cpp</a>	12
<a href="#">Server.h</a>	
Заголовочный файл для модуля <a href="#">Server</a>	12
<a href="#">User.cpp</a>	13
<a href="#">User.h</a>	
Заголовочный файл для модуля <a href="#">User</a>	13
<a href="#">UserDataBase.cpp</a>	14
<a href="#">UserDataBase.h</a>	
Заголовочный файл для модуля <a href="#">UserDataBase</a>	14





## Глава 3

# Классы

### 3.1 Класс ErrorManager

Класс обработки ошибок ErrorManage.

```
#include <ErrorManager.h>
```

Открытые члены

- string [getlogFile](#) ()  
Геттер для атрибута logFile.
- void [setlogFile](#) (string logfile)  
Сеттер для атрибута logFile.
- void [ErrorManage](#) (string info)  
Функция для обработки ошибок. Аварийно завершает программу
- void [SaveError](#) (string flag, string info, int type1)  
Функция записи ошибок в журнал записи ошибок

#### 3.1.1 Подробное описание

Класс обработки ошибок ErrorManage.

Менеджер вывода сообщений об ошибках

Аргументы

logFile	Путь к журналу ошибок
---------	-----------------------

#### 3.1.2 Методы

### 3.1.2.1 ErrorManage()

```
void ErrorManager::ErrorManage (
    string info )
```

Функция для обработки ошибок. Аварийно завершает программу

Аргументы

info	Описывает подробную информацию об ошибке
------	--

Объявления и описания членов классов находятся в файлах:

- [ErrorManager.h](#)
- [ErrorManager.cpp](#)

## 3.2 Класс Server

Основной класс для взаимодействия клиента с сервером

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

Открытые члены

- int [handeClientInteraction](#) (string db, string lfile)  
Основная функция для обслуживания клиентов
- uint32\_t [multiplyVectors](#) (const std::vector< uint32\_t > &array)  
Функция перемножения элементов вектора
- void [setAddress](#) (string address1)  
Сеттер для атрибута serverAddress.
- string [getAddress](#) ()  
Геттер для атрибута serverAddress.
- void [setPort](#) (string port1)  
Сеттер для атрибута serverAddress.
- int [getPort](#) ()  
Геттер для атрибута serverAddress.

### 3.2.1 Подробное описание

Основной класс для взаимодействия клиента с сервером

Аргументы

serverAddress	адрес сервера
serverPort	порт сервера
errorManager	Экземпляр класса <a href="#">ErrorManager</a> , ответственного за обработку ошибок.

### 3.2.2 Методы

#### 3.2.2.1 handleClientInteraction()

```
int Server::handleClientInteraction (
    string db,
    string logfile )
```

Основная функция для обслуживания клиентов

Аргументы

db	хранит в себе путь к файлу с БД клиентов
log	хранит в себе путь к журналу с записями об ошибках

#### 3.2.2.2 multiplyVectors()

```
uint32_t Server::multiplyVectors (
    const std::vector< uint32_t > & array )
```

Функция перемножения элементов вектора

Возвращает

возвращает произведение. При переполнении вверх должна возвращать  $2^{31}-1$ , а при переполнении вниз -  $-2^{32}$ .

Объявления и описания членов классов находятся в файлах:

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

## 3.3 Класс User

Класс, содержащий информацию о текущем пользователе

```
#include <User.h>
```

## Открытые члены

- bool [CheckLogin](#) (vector< string > Db\_login)  
Функция проверки логина пользователя, который подключен
- bool [CheckPassword](#) (vector< string > Db\_password, vector< string > Db\_login, string SALT)  
Функция проверки пароля подключенного клиента
- string [getLogin](#) ()  
Геттер для атрибута login.
- void [setLogin](#) (string login1)  
Сеттер для атрибута login.
- string [getPassword](#) ()  
Геттер для атрибута password.
- void [setPassword](#) (string password1)  
Сеттер для атрибута password.

### 3.3.1 Подробное описание

Класс, содержащий информацию о текущем пользователе

Аргументы

login	логин клиента, который подключен
password	пароль клиента, который подключен

### 3.3.2 Методы

#### 3.3.2.1 CheckLogin()

```
bool User::CheckLogin (  
    vector< string > Db_login )
```

Функция проверки логина пользователя, который подключен

Возвращает

Возвращает true, если логин подключенного клиента обнаружен в базе данных; в противном случае возвращает false

### 3.3.2.2 CheckPassword()

```
bool User::CheckPassword (
    vector< string > Db_password,
    vector< string > Db_login,
    string SALT )
```

Функция проверки пароля подключенного клиента

Возвращает

Возвращает true, если пароль клиента, подключенного в данный момент, соответствует паролю из базы данных; в противном случае возвращает false

Объявления и описания членов классов находятся в файлах:

- [User.h](#)
- [User.cpp](#)

## 3.4 Класс UserDataBase

Класс, предназначенный для извлечения информации из базы данных клиентов.

```
#include <UserDataBase.h>
```

Открытые члены

- `pair< vector< string >, vector< string > > getClientCredentials ()`  
Функция для получения БД клиентов
- `string getClientDataBase ()`  
Геттер для атрибута db.
- `void setClientDataBase (string db)`  
Сеттер для атрибута db.

### 3.4.1 Подробное описание

Класс, предназначенный для извлечения информации из базы данных клиентов.

Аргументы

clientDataBase	Путь к файлу базы данных клиентов
errorManager	Экземпляр класса <a href="#">ErrorManager</a> , ответственного за обработку ошибок.

### 3.4.2 Методы

#### 3.4.2.1 getClientCredentials()

```
pair< vector< string >, vector< string > > UserDataBase::getClientCredentials ( )
```

Функция для получения БД клиентов

Возвращает

Возвращает векторы логинов и паролей соответственно firstVector, secondVector

Объявления и описания членов классов находятся в файлах:

- [UserDataBase.h](#)
- [UserDataBase.cpp](#)

# Глава 4

## Файлы

### 4.1 Файл `ErrorManager.cpp`

```
#include "ErrorManager.h"
```

Граф включаемых заголовочных файлов для `ErrorManager.cpp`:

### 4.2 Файл `ErrorManager.h`

Заголовочный файл для модуля [ErrorManager](#).

```
#include <string>
#include <chrono>
#include <ctime>
#include <iostream>
#include <fstream>
#include <iomanip>
```

Граф включаемых заголовочных файлов для `ErrorManager.h`: Граф файлов, в которые включается этот файл:

#### Классы

- class [ErrorManager](#)  
Класс обработки ошибок `ErrorManager`.

#### 4.2.1 Подробное описание

Заголовочный файл для модуля [ErrorManager](#).

Автор

Черный М.В.

Версия

1.0

Дата

31.10.2023

Авторство

ИБСТ ПГУ

### 4.3 Файл main.cpp

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

Граф включаемых заголовочных файлов для main.cpp:

#### Функции

- void [help](#) ()  
Функция для получения инструкции по использованию программы
- int [main](#) (int argc, char \*argv[])  
Функция для извлечения параметров командной строки от оператора

### 4.4 Файл Server.cpp

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

Граф включаемых заголовочных файлов для Server.cpp:

#### Функции

- string [generateSalt](#) ()  
Функция для генерации строки SALT16.

### 4.5 Файл Server.h

Заголовочный файл для модуля [Server](#).

```
#include <sys/types.h>
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fstream>
#include <vector>
#include <random>
#include <string>
#include <locale>
#include <codecvt>
#include "../md5/md5.h"
#include "UserDataBase.h"
#include "User.h"
```

Граф включаемых заголовочных файлов для Server.h: Граф файлов, в которые включается этот файл:



## Классы

- class [Server](#)

Основной класс для взаимодействия клиента с сервером

### 4.5.1 Подробное описание

Заголовочный файл для модуля [Server](#).

Автор

Черный М.В.

Версия

1.0

Дата

31.10.2023

Авторство

ИБСТ ПГУ

## 4.6 Файл User.cpp

```
#include "User.h"
```

Граф включаемых заголовочных файлов для User.cpp:

## 4.7 Файл User.h

Заголовочный файл для модуля [User](#).

```
#include <string>
#include <vector>
#include <iostream>
#include "../md5/md5.h"
```

Граф включаемых заголовочных файлов для User.h: Граф файлов, в которые включается этот файл:

## Классы

- class [User](#)

Класс, содержащий информацию о текущем пользователе

### 4.7.1 Подробное описание

Заголовочный файл для модуля [User](#).

Автор

Черный М.В.

Версия

1.0

Дата

31.10.2023

Авторство

ИБСТ ПГУ

## 4.8 Файл UserDataBase.cpp

```
#include "UserDataBase.h"
```

Граф включаемых заголовочных файлов для UserDataBase.cpp:

## 4.9 Файл UserDataBase.h

Заголовочный файл для модуля [UserDataBase](#).

```
#include <string>
#include <vector>
#include <map>
#include <fstream>
#include <sstream>
#include <iostream>
#include "ErrorManager.h"
```

Граф включаемых заголовочных файлов для UserDataBase.h: Граф файлов, в которые включается этот файл:

Классы

- class [UserDataBase](#)

Класс, предназначенный для извлечения информации из базы данных клиентов.

### 4.9.1 Подробное описание

Заголовочный файл для модуля [UserDataBase](#).

Автор

Черный М.В.

Версия

1.0

Дата

31.10.2023

Авторство

ИБСТ ПГУ



# Предметный указатель

- CheckLogin
  - User, [8](#)
- CheckPassword
  - User, [8](#)
- ErrorManage
  - ErrorManager, [5](#)
- ErrorManager, [5](#)
  - ErrorManage, [5](#)
- ErrorManager.cpp, [11](#)
- ErrorManager.h, [11](#)
- getClientCredentials
  - UserDataBase, [9](#)
- handeClientInteraction
  - Server, [7](#)
- main.cpp, [12](#)
- multiplyVectors
  - Server, [7](#)
- Server, [6](#)
  - handeClientInteraction, [7](#)
  - multiplyVectors, [7](#)
- Server.cpp, [12](#)
- Server.h, [12](#)
- User, [7](#)
  - CheckLogin, [8](#)
  - CheckPassword, [8](#)
- User.cpp, [13](#)
- User.h, [13](#)
- UserDataBase, [9](#)
  - getClientCredentials, [9](#)
- UserDataBase.cpp, [14](#)
- UserDataBase.h, [14](#)