

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 клиента»

ПГУ.000000.С1116.КР.21ПТ208.01.ПЗ

Специальность – 10.05.02 Информационная безопасность
телекоммуникационных систем

Специализация – Разработка защищенных телекоммуникационных систем

Выполнил студент: _____ Ковалев Д.Е.

Группа: 21ПТ1

Руководитель:

_____ Сидоров Н.А.

Работа защищена с оценкой _____

Дата защиты _____

2023

УТВЕРЖДАЮ
Зав. кафедрой ИБСТ
к.т.н., доцент

Зефирова С.Л.

28.10.2022 г.

ЗАДАНИЕ

на курсовую работу

по теме: Программная реализация сетевого клиента

1 Дисциплина: Технологии и методы программирования.

2 Студент: Ковалев Данил Евгеньевич

3 Группа: 21ПТ1.

4 Исходные данные на работу:

4.1 Цель: разработка клиентской 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 числовых векторов из файла исходных данных;

– реализация сеанса обмена данными с сервером

– сохранение результатов в файл результатов

4.2.2. Требования к входным данным

– файл исходных данных — текстовый;

– формат файла исходных данных:

– первая строка:

– число векторов, целое без знака;

– вторая и последующие строки:

– размер вектора, целое без знака;

– значения вектора, числа с плавающей точкой;

– числа в строке разделяются пробелом;

4.2.3. Требования к взаимодействию с сервером

– реализация сеанса взаимодействия с сервером 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 SALT16 (при успешной идентификации)
 - 3б. сервер передает строку "ERR" и разрывает соединение(при ошибке идентификации)
 4. клиент передает HASHMD5(SALT16 || 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 результатам работы

- файл результатов — текстовый
- формат файла данных результатов:
 - первое поле 4 байта, число типа uint64_t, количество результатов;
 - следующие поля по 8 байт, числа типа double, результаты;

4.2.5. Требования к пользовательскому интерфейсу и обработке ошибок в процессе выполнения программы

- управление программой должно осуществляться через параметры командной строки, передаваемые ей при старте;
- интерфейс должен обеспечивать передачу программе следующей информации:

- сетевой адрес сервера, обязательный;
- порт сервера, необязательный;
 - значение по умолчанию 33333;
- имя файла с исходными данными, обязательный;
- имя файла для сохранения результатов, обязательный;
- имя файла с ID и PASSWORD клиента, необязательный;
 - значение по умолчанию ~/.config/vclient.conf;
- программа не должна интерактивно взаимодействовать с пользователем;
- при возникновении ошибок, исправление которых требует вмешательства пользователя, программа должна аварийно завершаться с выдачей диагностического сообщения об ошибке;
 - диагностическое сообщение об ошибке должно содержать информацию
 - о функции, при выполнении которой произошла ошибка;
 - о значениях параметров и переменных, приведших к ошибке;
 - о типе ошибки;
- программа должна предусматривать выдачу справки о пользовательском интерфейсе
 - при запуске без параметров;
 - при запуске с параметром -h;

4.2.6. 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:

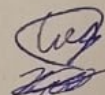
- | | |
|--|-------------------------|
| - оформление ТЗ | 09.09.22(на 2 неделе) |
| - разработка диаграмм вариантов использования
и диаграмм классов | 23.09.22(на 4 неделе) |
| - разработка диаграмм последовательностей и
диаграмм деятельности | 07.10.22(на 6 неделе) |
| - разработка кода программы | 28.10.22(на 9 неделе) |
| - разработка модульных тестов и выполнение
тестирования | 11.11.22(на 11 неделе) |
| - разработка функциональных тестов и
приемочное тестирование | 25.11.22(на 13 неделе) |
| - разработка документации | 09.12.22(на 15 неделе) |
| - оформление отчета о курсовой работе | 16.12.22(на 16 неделе) |
| - защита курсовой работы | 23.12.22 (на 17 неделе) |

Руководитель работы

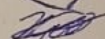
Задание получил «8» сентября 2022г.

Студент

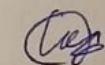
Нормоконтролер



Н.А. Сидоров



Д.Е. Ковалев



Н.А. Сидоров

Оглавление

Введение.....	7
1 Анализ требований к программе.....	9
2 Построение UML-диаграмм вариантов использования и проектирование пользовательского интерфейса.....	13
3 Построение UML-диаграммы классов и планирование модулей.....	14
4 Построение UML-диаграммы последовательностей.....	15
5 Построение UML-диаграммы деятельности.....	17
6 Разработка программы.....	18
7 Разработка модульных тестов и модульное тестирование.....	20
8 Разработка функциональных тестов и приемочное тестирование.....	24
9 Документирование кода программы с использованием Doxygen.....	27
Заключение.....	28
Список используемых источников.....	29
Приложение А.....	30
Приложение Б.....	33
Приложение В.....	53

Введение

Современный этап развития общества характеризуется возрастающей ролью Интернета и обмена информацией. Компьютерные сети, или сети передачи данных проникли во все сферы жизни человека. Появилась возможность обмениваться информацией, несмотря на расстояние. Наиболее распространенной на данный момент компьютерной сетью является сеть Интернет. Она построена на стеке протоколов TCP/IP. Наиболее важную роль играют протокол управления передачей данных и межсетевой протокол.

Как было сказано выше, протокол TCP отвечает за выполнение крайне важной задачи — управления передачей данных. Если рассматривать данный протокол с точки зрения эталонной модели OSI (определяет уровни взаимодействия систем в сетях с коммутацией пакетов, стандартные названия уровней, а также функции, которые должен выполнять каждый уровень; всего этих уровней семь: прикладной уровень, уровень представления, сеансовый уровень, транспортный уровень, сетевой уровень, канальный уровень и физический уровень), то он работает на сеансовом (обеспечивает управление взаимодействием сторон: фиксирует, какая из сторон является активной в настоящий момент, и предоставляет средства синхронизации сеанса) и транспортном уровнях (обеспечивает приложениям или верхним уровням стека передачу данных с той степенью надежности, которая им требуется).

Предметом курсовой работы являются основные принципы функционирования сетевого клиента.

На первом этапе курсовой работы проводится анализ требований к программе.

На втором этапе курсовой работы проектируются UML-диаграммы вариантов использования и проектирование пользовательского интерфейса.

На третьем этапе курсовой работы проектируются UML-диаграммы классов и планирование модулей

На четвертом этапе проектируются UML-диаграммы последовательностей

На пятом этапе курсовой работы проектируются UML-диаграммы деятельности

На шестом этапе курсовой работы разрабатывается программа-клиент.

На седьмом этапе курсовой работы разрабатываются модульные тесты и выполняется модульное тестирование.

На восьмом этапе разрабатываются функциональные тесты и выполняется приемочное тестирование

На девятом этапе курсовой работы проводится документирование кода программы с использованием Doxygen.

1 Анализ требований к программе

На данном этапе проанализированы требования к программе, которые описаны ниже:

Требования к программе:

1 Требования к выполняемым функциям:

- программа должна выполнять следующие функции:
 - чтение числовых векторов из файла исходных данных;
 - реализация сеанса обмена данными с сервером
- сохранение результатов в файл результатов

2 Требования к входным данным:

- файл исходных данных — текстовый;
- формат файла исходных данных:
 - первая строка:
 - количество последующих строк (в каждой строке — один числовой вектор), целое без знака;
 - вторая и последующие строки:
 - количество данных в строке (размер вектора), целое без знака;
 - данные (значения вектора), числа с плавающей точкой;
 - числа в строке разделяются пробелом;

3 Требования к взаимодействию с сервером:

- реализация сеанса взаимодействия с сервером 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 SALT16 (при успешной идентификации)
 - 3б. сервер передает строку "ERR" и разрывает соединение(при ошибке идентификации)
 4. клиент передает HASHMD5(SALT16 || 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 Требования к результатам работы:
- файл результатов — текстовый;
 - первая строка:

- количество результатов (число векторов), целое без знака;
- вторая и последующие строки:
- результат обработки каждого вектора, числа с плавающей точкой;

- каждый результат записан на новой строке;

5 Требования к пользовательскому интерфейсу и обработке ошибок в процессе выполнения программы:

- управление программой должно осуществляться через параметры командной

строки, передаваемые ей при старте;

- интерфейс должен обеспечивать передачу программе следующей информации:

- сетевой адрес сервера, обязательный;
- порт сервера, необязательный;
- значение по умолчанию 33333;
- имя файла с исходными данными, обязательный;
- имя файла для сохранения результатов, обязательный;
- имя файла с ID и PASSWORD клиента, необязательный;
- значение по умолчанию ~/.config/vclient.conf;

- программа не должна интерактивно взаимодействовать с пользователем;

- при возникновении ошибок, исправление которых требует вмешательства

пользователя, программа должна аварийно завершаться с выдачей

диагностического сообщения об ошибке;

- диагностическое сообщение об ошибке должно содержать информацию

- о функции, при выполнении которой произошла ошибка;

- о значениях параметров и переменных, приведших к ошибке;

- о типе ошибки;
 - программа должна предусматривать выдачу справки о пользовательском интерфейсе
 - при запуске без параметров;
 - при запуске с параметром -h;
- 6 Требования к среде функционирования программы:
- программа должна функционировать в операционной системе с ядром Linux

2 Построение UML-диаграмм вариантов использования и проектирование пользовательского интерфейса

На данном этапе курсовой была построена диаграмма вариантов использования .

Диаграмма вариантов использования показана в приложении А на рисунке А.1.

Так же в данной главе было произведено проектирование пользовательского интерфейса. Пользовательский интерфейс включает в себя 6 параметров:

- «без параметров». Если программа запущена без параметров, то она выводит справку и завершает работу;

- «h» предназначен для выдачи справки. Если программа запускается с этим параметром, то она выводит справку и завершает работу;

- «i» предназначен для передачи программе адреса сервера. Данный параметр является обязательным. Если данный параметр не введен, то программа завершается с выводом соответствующего сообщения;

- «p» предназначен для передачи программе порта сервера. Данный параметр является необязательным. Если данный параметр не введен, то программа использует значение по умолчанию «33333»;

- «e» предназначен для передачи программе файла с векторами. Данный параметр является обязательным. Если данный параметр не введен, то программа завершается с выводом соответствующего сообщения.;

- «s» предназначен для передачи программе файла для записи результата. Данный параметр является обязательным. Если данный параметр не введен, то программа завершается с выводом соответствующего сообщения.;

- «a» предназначен для ввода получения логина и пароля пользователя. Данный параметр является необязательным. Если данный параметр не введен, то программа используется значение по умолчанию «~/config/vclient.conf».

3 Построение UML-диаграммы классов и планирование модулей

На этом этапе была построена UML-диаграмма классов, а также произведено планирование модулей.

После анализа требований к программе и диаграммы вариантов использования была построена диаграмма классов. Был выделен класс для подключения клиента к серверу `Client`. Данный класс имеет несколько атрибутов и методов.

В классе есть также метод `Server(str1, str2)`. Этот метод является главной функцией для взаимодействия клиента и сервера. В случае возникновения ошибки используется класс обработки ошибок `client_error`. Параметрами данного метода являются строки `str1`, `str2`. Параметр `str1` – это адрес сервера, `str2` – порт сервера. Также этот метод предназначен для считывания из файла с логином и паролем этих параметров. В случае возникновения ошибки используется класс обработки ошибок `client_error`. Параметрами данного метода являются строки `str1`, `str2`. Параметр `str1` – это адрес сервера, `str2` – порт сервера.

Диаграмма классов показан в приложении А на рисунке А.2.

4 Построение UML-диаграммы последовательностей

На данном этапе были разработаны . Были разработаны три диаграммы последовательностей.

Первая диаграмма последовательностей показывает как взаимодействуют клиент и сервер, когда клиент отправляет неправильный логин. Клиент и сервер устанавливают соединение. Клиент отправляет логин серверу. Сервер получает логин, проверяет его. Отправляет сообщение «ERR» и завершают соединение. Диаграмма последовательностей при неверном логине показана в приложении А на рисунке А.3.

Вторая диаграмма последовательности показывает как взаимодействуют клиент и сервер, когда клиент отправляет неправильный хеш, сформированный из-за неправильного пароля. Клиент и сервер устанавливают соединение. Клиент отправляет логин серверу. Сервер получает логин, проверяет его. Сервер отправляет «SALT» клиенту. Клиент получает данное сообщение. Получает новую строку из двух строк: SALT16 || PASSWORD. Считает хэш по алгоритму MD5. После чего отправляет серверу. Сервер получает строку HASHMD5(SALT16 || PASSWORD). Проверяет ее со своим значением. Если значения не совпадают, то сервер отправляет клиенту сообщение «ERR». Диаграмма последовательностей при неверном хеше показана в приложении А на рисунке А.4.

Третья диаграмма последовательности показывает как взаимодействуют клиент и сервер, когда клиент отправляет правильный логин и пароль. Клиент и сервер устанавливают соединение. Далее взаимодействие с сервером по аналогии до проверки со своим значением. Отправляет клиенту сообщение «OK». Клиент отправляет количество векторов серверу, размер первого вектора, отправляет вектор. Получает сумму от элементов первого вектора и так далее, пока вектора не закончатся. Клиент и сервер завершают взаимодействие.

Диаграмма последовательностей при успешном сценарии показана в приложении А на рисунке А.5.

5 Построение UML-диаграммы деятельности

На этом этапе была построена UML-диаграмма деятельности. В начале пользователь вводит параметры, если параметры не верны, то пользователь получает сообщение об ошибке. Если проверка параметров прошла успешно, то происходит проверка данных указанных в файлах, не пустые ли они и правильно ли введены данные. Если возникает ошибка, то пользователь получает сообщение об ошибке. В случае успеха проверок, производится чтение всех данных из файла, выполнение всех операций и запись результатов в файл. После всех действий пользователь получает сообщения в терминале о состоянии программы. Диаграмма деятельности показана в приложении А на рисунке А.6.

6 Разработка программы

На данном этапе была разработана программа. Код программы был написан на языке C++. Код программы разбит на 5 модулей. Главным модулем программы является *main.cpp*. В данном модуле происходит взаимодействие с пользователем, получение от него данных и передача их в модуль *Client.cpp*. Код модуля *main.cpp* представлен в приложении Б код Б.3.

Модули *Client.h* и *Client.cpp* используется для взаимодействие с сервером. В модуле *Client.h* подключены библиотеки для работы программы. Также использован класс *Client* для инициализации переменных используемых в модуле *Client.cpp*. Так же в данном модуле используется класс для обработки ошибок. В модуле *Client.cpp* производится открытия файлов с логином и паролем, с векторами и для записи результата. Если данные файлы не открыты или пусты, то возбуждается исключение и выводится ошибка. Далее производится установка соединения с сервером. Клиент передает логин и получает соль в случае успешного взаимодействия, в случае ошибки получает сообщение «ERR» и сервер разрывает соединение. Если аутентификация прошла успешна, то клиент передает хеш сформированный из соли, полученной от сервера и пароля пользователя. В случаи успешной авторизации выводится сообщение «OK», при не удачной авторизации выводится сообщение «ERR» и соединение разрывается. В случае успешной авторизации, клиент отправляет количество векторов, их размер и сами вектора. И получает от сервера результат вычисления по каждому вектору и записывает его в файл для результатов. Коды модулей *Client.h* и *Client.cpp* показаны в приложении Б код Б.1 и Б.2.

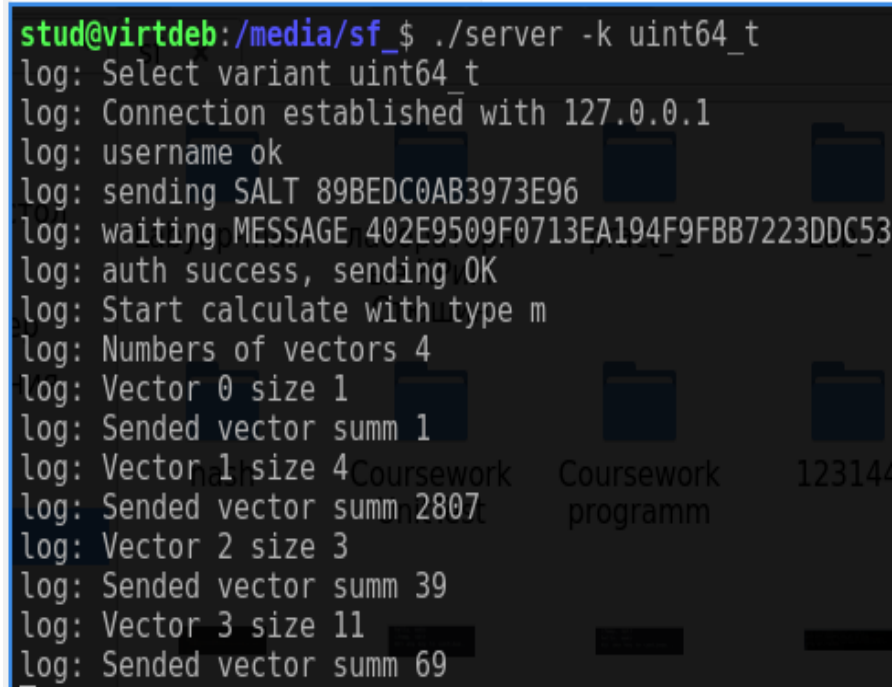
Хеширование соли и пароля вынесено в два отдельных модуля *md5.h* и *md5.cpp*. В модуле *md5.h* подключены библиотеки для хеширование. В модуле *md5.cpp* производится само хеширование полученного сообщения. Коды модулей *md5.h* и *md5.cpp* показаны в приложении Б код Б.4 и Б.5.

Для компиляции и сборки был написан Makefile. Его содержимое показано в приложении Б код Б.6

Все коды представлены в репозитории на GitHub:

<https://github.com/XxXxUWUxXxX/ClientProg>

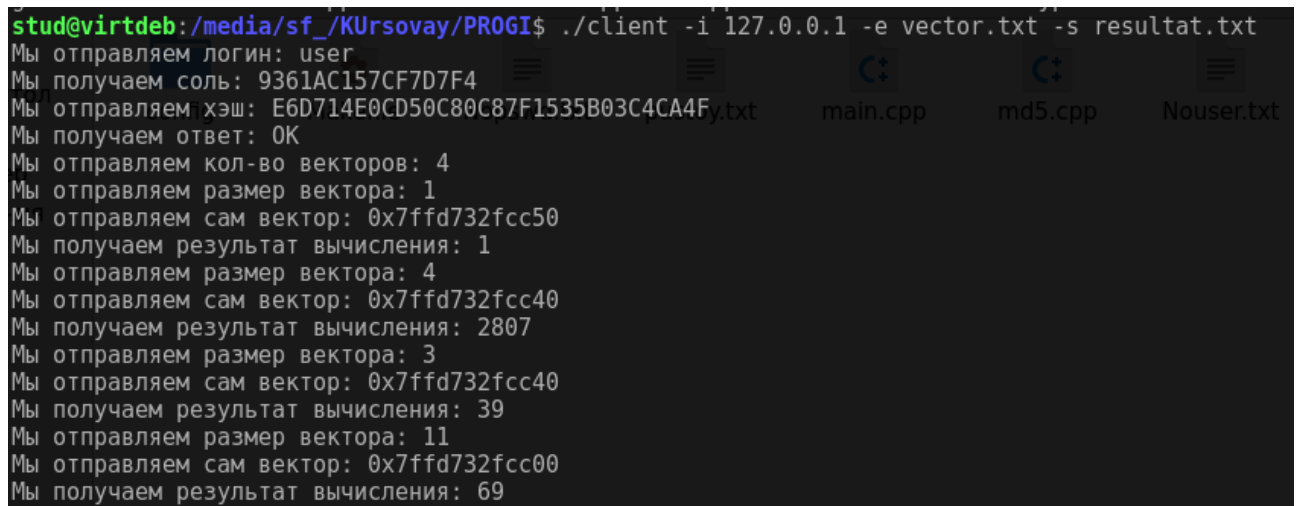
На рисунке 1 показан терминал сервера при работе.



```
stud@virtdeb:/media/sf_$ ./server -k uint64_t
log: Select variant uint64_t
log: Connection established with 127.0.0.1
log: username ok
log: sending SALT 89BEDC0AB3973E96
log: waiting MESSAGE 402E9509F0713EA194F9FBB7223DDC53
log: auth success, sending OK
log: Start calculate with type m
log: Numbers of vectors 4
log: Vector 0 size 1
log: Sended vector summ 1
log: Vector 1 size 4
log: Sended vector summ 2807
log: Vector 2 size 3
log: Sended vector summ 39
log: Vector 3 size 11
log: Sended vector summ 69
```

Рисунок 1 — Терминал сервера

Для клиента написана отладочная трассировка показанная на рисунке 2.



```
stud@virtdeb:/media/sf_/KURsovay/PROGI$ ./client -i 127.0.0.1 -e vector.txt -s resultat.txt
Мы отправляем логин: user
Мы получаем соль: 9361AC157CF7D7F4
Мы отправляем хэш: E6D714E0CD50C80C87F1535B03C4CA4F
Мы получаем ответ: OK
Мы отправляем кол-во векторов: 4
Мы отправляем размер вектора: 1
Мы отправляем сам вектор: 0x7ffd732fcc50
Мы получаем результат вычисления: 1
Мы отправляем размер вектора: 4
Мы отправляем сам вектор: 0x7ffd732fcc40
Мы получаем результат вычисления: 2807
Мы отправляем размер вектора: 3
Мы отправляем сам вектор: 0x7ffd732fcc40
Мы получаем результат вычисления: 39
Мы отправляем размер вектора: 11
Мы отправляем сам вектор: 0x7ffd732fcc00
Мы получаем результат вычисления: 69
```

Рисунок 2 — Терминал клиента

7 Разработка модульных тестов и модульное тестирование

При создании курсовой, на данном этапе, были разработаны модульные тест для проверки корректности работы программы. Были созданы тесты для проверки введенных параметров, ошибок при взаимодействии с программой. Также был создан удачный сценарий для проверки правильности работы проверки программы. В результате этого теста произведено успешное взаимодействие с сервером с верными логином и паролем указанными в файле *./config/vclient.conf*, открыт нужный файл с векторам *vector.txt* и результат записан в файл *restltat.txt*. Файлы показаны на рисунках 3,4 и 5. Тестовые сценарии были представлены в таблице 1.

Таблица 1 — Модульное тестирование

№	Тесты	Описание	Результат
1	Удачный сценарий	Успешное подключение. В файл для результата был записана сумма векторов.	Не исключение
2	Неверно введен ip	Невозможно подключиться к серверу из-за неверно введенного ip	Исключение
3	Неверно введен port	Невозможно подключиться к серверу из-за неверного введенного порта	Исключение
4	Ошибка открытия файла с логином и паролем	Невозможно открыть файл с логином и паролем. Неверно указан путь	Исключение
5	Ошибка чтения из файла с логином и паролем	Файл с логином и паролем пуст	Исключение
6	Ошибка идентификации	Введен неверный логин	Исключение
7	Ошибка аутентификации	Введен неверный пароль	Исключение
8	Ошибка открытия файла с векторами	Невозможно открыть файл с векторами. Неверно указан путь	Исключение
9	Ошибка чтения из файла с векторами	Файл с векторами пуст	Исключение
10	Ошибка открытия файла для записи суммы	Невозможно открыть файл для записи суммы. Неверно указан путь	Исключение

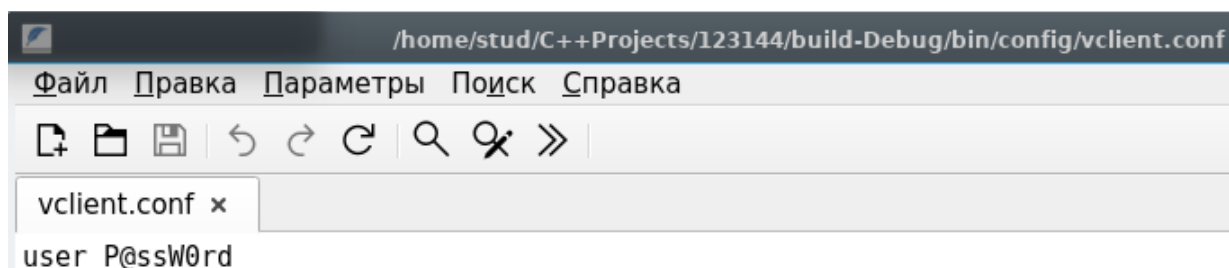


Рисунок 3 — Файл `./config/vclient.conf`

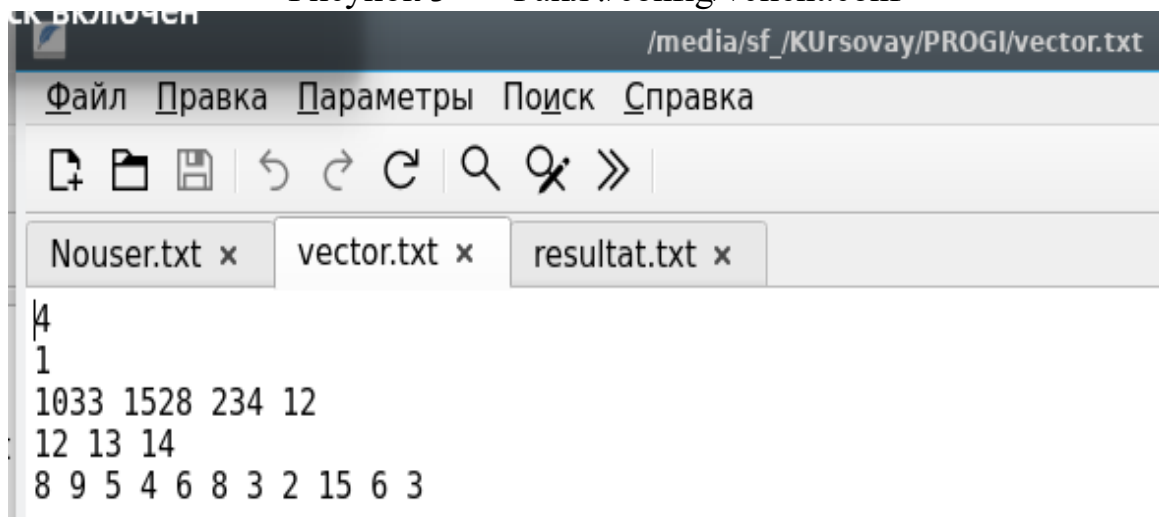


Рисунок 4 — Файл `vector.txt`

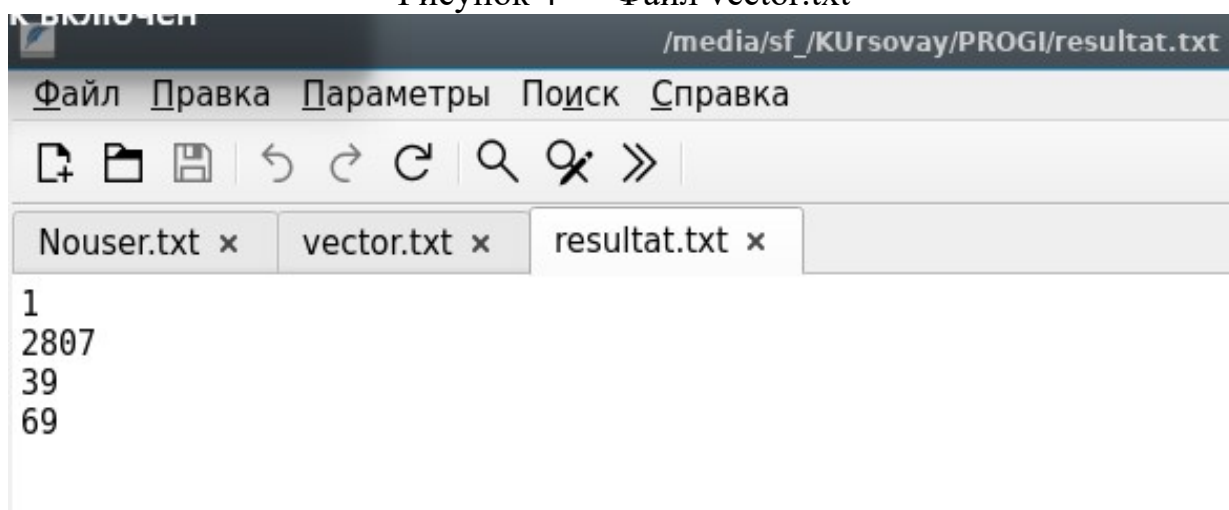


Рисунок 5 — Файл `resultat.txt`

На рисунке 6 показан файл, использованный в сценарии 5 и 9. Данный файл сделан пустым для вызова исключения.

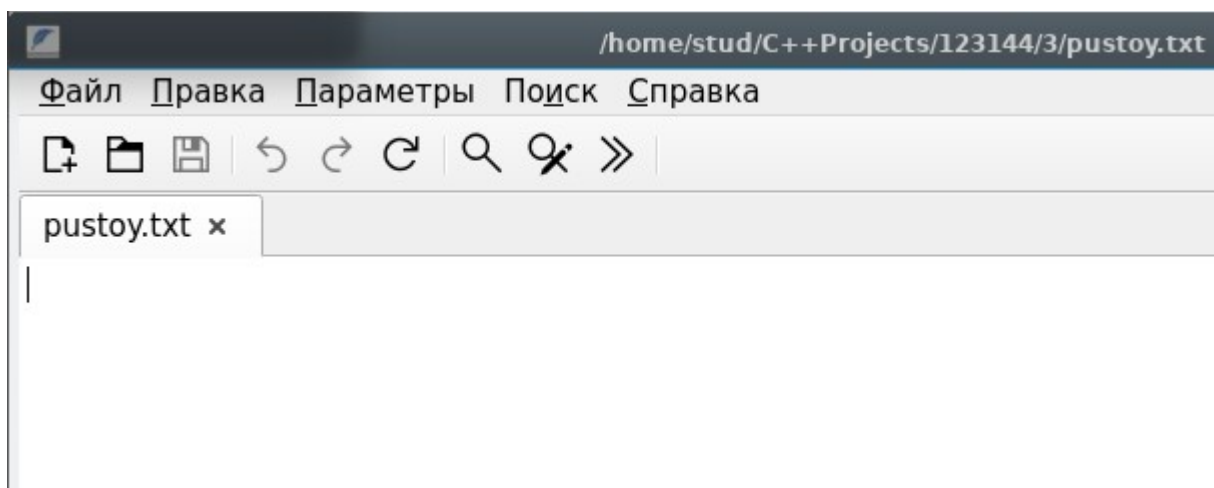


Рисунок 6 — Файл pustoy.txt

На рисунке 7 продемонстрирован файл, использованный для сценария 6, с неверным логином.

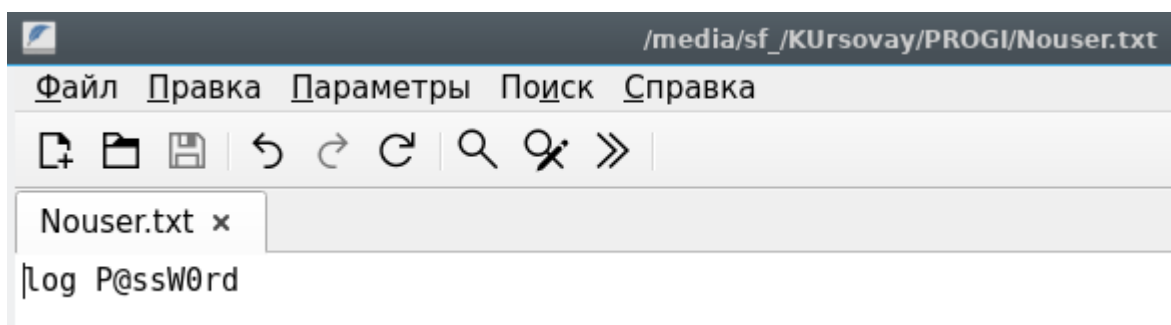


Рисунок 7 — Файл Nouser.txt

На рисунке 8 показан файл, использованный для сценария 7, с неверным паролем.

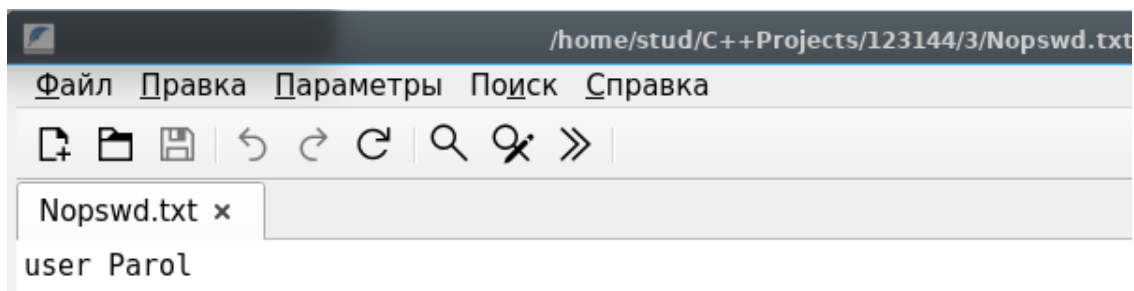


Рисунок 8 — Файл Nopswd.txt

На рисунке 9 представлен результат выполнения тестов. Все 10 тестов успешно пройдены.

```
stud@virtdeb:~/C++Projects/123144/build-Debug/bin$ ./UnitTest
Мы отправляем логин: user
Мы получаем соль: 82BDD6FF6D5C3616
Мы отправляем хэш: D065D2C9FCAA049C2A99E7618223364D
Мы получаем ответ: OK
Мы отправляем кол-во векторов: 4
Мы отправляем размер вектора: 1
Мы отправляем сам вектор: 0x7ffd3ebcfb50
Мы получаем результат вычисления: 1
Мы отправляем размер вектора: 4
Мы отправляем сам вектор: 0x7ffd3ebcfb40
Мы получаем результат вычисления: 2807
Мы отправляем размер вектора: 3
Мы отправляем сам вектор: 0x7ffd3ebcfb40
Мы получаем результат вычисления: 39
Мы отправляем размер вектора: 11
Мы отправляем сам вектор: 0x7ffd3ebcfb00
Мы получаем результат вычисления: 69
Success: 10 tests passed.
Test time: 0.00 seconds.
```

Рисунок 9 — Результат тестирования

Данные тесты нужны для проверки введенных параметров, а также ошибок при взаимодействии.

Код модульного тестирования можно посмотреть в репозитории в приложении Б код Б.7 или на GitHub:

<https://github.com/XxXxUWUxXxX/ClientProg>

8 Разработка функциональных тестов и приемочное тестирование

На данном этапе была произведена разработка функциональных тестов и было сделано приемочное тестирование. В таблице 2 представлено описание функциональных тестов и их результаты.

Таблица 2 — Функциональные тесты

№	Описание	Результат
1	Использование программы с необязательными параметрами	Не исключение
2	Использование программы без необязательных параметров	Не исключение
3	Использование программы с примером выдачи ошибки. Введен неправильный путь к файлу с векторами	Ошибка. Аварийное завершение программы
4	Использование программы без параметров	Завершение программы с выдачей сообщения
5	Использование программы с параметром -h	Завершение программы с выдачей сообщения
6	Использование программы с неизвестным параметром	Завершение программы с выдачей сообщения

На рисунке 10 показан результат 1 теста.

```
stud@virtdeb:/media/sf/_KUrsovay/PROGI$ ./client -i 127.0.0.1 -p 33333 -e vector.txt -s resultat.txt -
a ./config/vclient.conf
Мы отправляем логин: user
Мы получаем соль: F8D99EB150B9FB0F
Мы отправляем хэш: CDC3B215477151B376C07428727370CC
Мы получаем ответ: OK
Мы отправляем кол-во векторов: 4
Мы отправляем размер вектора: 1
Мы отправляем сам вектор: 0x7ffe3adba350
Мы получаем результат вычисления: 1
Мы отправляем размер вектора: 4
Мы отправляем сам вектор: 0x7ffe3adba340
Мы получаем результат вычисления: 2807
Мы отправляем размер вектора: 3
Мы отправляем сам вектор: 0x7ffe3adba340
Мы получаем результат вычисления: 39
Мы отправляем размер вектора: 11
Мы отправляем сам вектор: 0x7ffe3adba300
Мы получаем результат вычисления: 69
```

Рисунок 10 — Результат первого теста

На рисунке 11 показан результат 2 теста.

```
stud@virtdeb:/media/sf_/KURsovay/PROGI$ ./client -i 127.0.0.1 -e vector.txt -s resultat.txt
Мы отправляем логин: user
Мы получаем соль: E5261826B9927205
Мы отправляем хэш: 5073227295895502865F9F41FADBB88F
Мы получаем ответ: OK
Мы отправляем кол-во векторов: 4
Мы отправляем размер вектора: 1
Мы отправляем сам вектор: 0x7ffd9c7c3080
Мы получаем результат вычисления: 1
Мы отправляем размер вектора: 4
Мы отправляем сам вектор: 0x7ffd9c7c3070
Мы получаем результат вычисления: 2807
Мы отправляем размер вектора: 3
Мы отправляем сам вектор: 0x7ffd9c7c3070
Мы получаем результат вычисления: 39
Мы отправляем размер вектора: 11
Мы отправляем сам вектор: 0x7ffd9c7c3030
Мы получаем результат вычисления: 69
```

Рисунок 11 — Результат второго теста

На рисунке 12 показан результат 3 теста.

```
stud@virtdeb:~/C++Projects/123144/3$ ./client -i 127.0.0.1 -e v12352.txt -s resultat.txt
terminate called after throwing an instance of 'client_error'
  what(): fun:Server, param:vector_file.
Ошибка отрыва файла с векторами!
Аварийный останов
stud@virtdeb:~/C++Projects/123144/3$ █
```

Раскладк

Рисунок 12 — Результат третьего теста

На рисунке 13 показан результат 4 теста.

```
stud@virtdeb:~/C++Projects/123144/3$ ./client
!!!Программа клиента!!!
Параметры для запуска клиента:
  -h — справка
  -i — адрес сервера (обязательный)
  -p — порт сервера (необязательный — по умолчанию 33333)
  -e — имя файла с исходными данными (обязательный)
  -s — имя файла для сохранения результатов (обязательный)
  -a — имя файла с ID и PASSWORD клиента (необязательный — по умолчанию ~/.config/vclient.conf)
stud@virtdeb:~/C++Projects/123144/3$ █
```

Рисунок 13 — Результат четвертого теста

На рисунке 14 показан результат 5 теста.

```
stud@virtdeb:~/C++Projects/123144/3$ ./client -h
!!!Программа клиента!!!
Параметры для запуска клиента:
  -h — справка
  -i — адрес сервера (обязательный)
  -p — порт сервера (необязательный — по умолчанию 33333)
  -e — имя файла с исходными данными (обязательный)
  -s — имя файла для сохранения результатов (обязательный)
  -a — имя файла с ID и PASSWORD клиента (необязательный — по умолчанию ~/.config/vclient.conf)
stud@virtdeb:~/C++Projects/123144/3$ █
```

Рисунок 14 — Результат пятого теста

На рисунке 15 показан результат 6 теста.

```
stud@virtdeb:~/C++Projects/123144/3$ ./client -o
./client: invalid option -- 'o'
Параметр задан не верно или такого параметра не существует
stud@virtdeb:~/C++Projects/123144/3$ █
```

Рисунок 15 — Результат шестого теста

Тестовые сценарии предназначены для проверки программы, ее общего функционирования.

9 Документирование кода программы с использованием Doxygen

На этом этапе было произведено документирование кода программы с использованием Doxygen

Для создания документации была выполнена команда *doxygen -g* в каталоге, где находятся исходные тексты программы и файл конфигурации Doxyfile.

Далее в конфигурационный файл внесены ряд изменений:

PROJECT_NAME изменен на имя проекта.

OUTPUT_LANGUAGE изменен на русский язык путем установки значения Russian.

После было произведено документирование самого кода. Затем была прописана команда *doxygen* и были созданы два каталога html и latex.

Для просмотра документации в формате html просмотра необходимо открыть в каталоге html, файл index.html в браузере.

Для создания документация в формате PDF, в каталоге latex с документацией в формате TeX выполнена команда *make*. После выполнения этой команды создан файл документации в формате PDF refman.pdf

Документация можно просмотреть в репозитории на GitHub:

<https://github.com/>

Заключение

В результате выполнения курсовой работы были разработаны протокол прикладного уровня, алгоритмы программ, была реализована программа-клиент. Было проведено тестирование разработанной программы, был разработан пользовательский интерфейс, задокументирована программа, разработан и сформирован отчет о курсовой работе. Отчет был проверен на антиплагиат на сайте: <https://users.antiplagiat.ru/cabinet>. Результат проверки представлен в приложении В рисунок В.1.

В результате выполненной работы были получены знания по разработке приложений с сетевым взаимодействием и выработаны практические умения по разработке и реализации сетевого приложения с архитектурой клиент-сервер.

Задание на курсовую работу было выполнено в полном объеме.

Список используемых источников

1. Сайт: moodle.pnzgu.ru, статья «Распоряжение по требованиям к оформлению КП и ТЗ на кафедре ИБСТ_2018» [Электронный ресурс]. URL: http://moodle.pnzgu.ru/pluginfile.php/1536030/mod_resource/content/1/Распоряжение%20по%20требованиям%20к%20оформлению%20КП%20и%20ТЗ%20на%20кафедре%20ИБСТ_2018_02.pdf
2. Сайт: moodle.pnzgu.ru, статья «Положение о курсовом проектировании» [Электронный ресурс]. URL: http://moodle.pnzgu.ru/pluginfile.php/1535938/mod_resource/content/1/doc_291tu.pdf
3. Сайт: kursach37.com, статья «Рекомендация по написанию введения к курсовой работе» [Электронный ресурс]. URL: <https://kursach37.com/kak-pisat-vvedenie-k-kursovoy-rabote/>
4. Сайт: nauchniestati.ru, статья «Рекомендация по написанию заключение к курсовой работе» [Электронный ресурс]. URL: <https://nauchniestati.ru/blog/zakljuchenie-k-kursovoj-rabote/>
5. Сайт: znanium.com, статья «Технология разработки программного обеспечения : учеб. пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Виснадул» [Электронный ресурс]. URL: <https://znanium.com/catalog/document?pid=768473>
6. Сайт: znanium.com, статья «Введение в архитектуру программного обеспечения: Учебное пособие / Гагарина Л.Г., Федоров А.Р., Федоров П.А.» [Электронный ресурс]. URL: <https://znanium.com/catalog/document?pid=542665>
7. Сайт: znanium.com, статья «Архитектура и проектирование программных систем: Монография / С.В. Назаров.» [Электронный ресурс]. URL: <https://znanium.com/catalog/document?pid=542562>
8. Сайт: e.lanbook.com, статья «Буч, Г. Язык UML. Руководство пользователя» [Электронный ресурс]. URL: <https://e.lanbook.com/book/1246>

Приложение А

(обязательное) UML-диаграммы

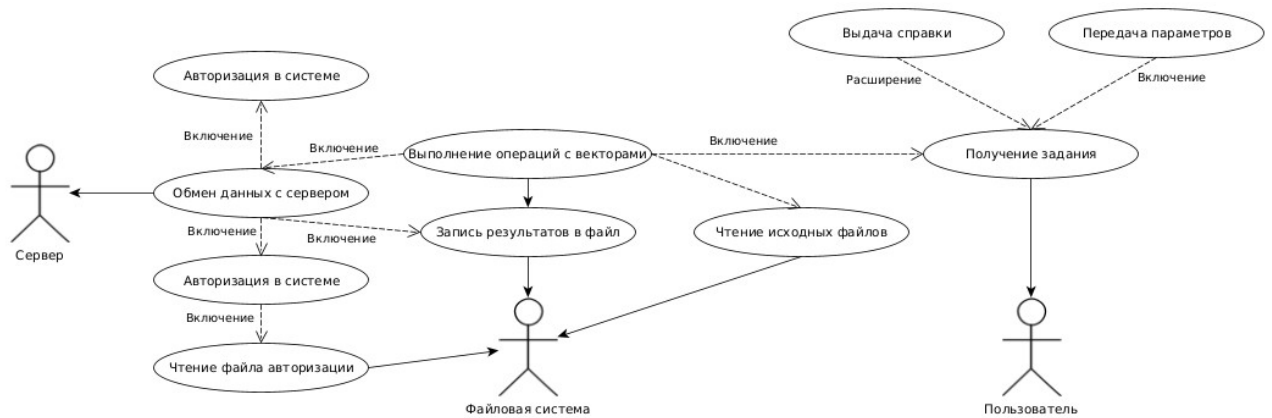


Рисунок А.1 — UML-диаграмма вариантов использования

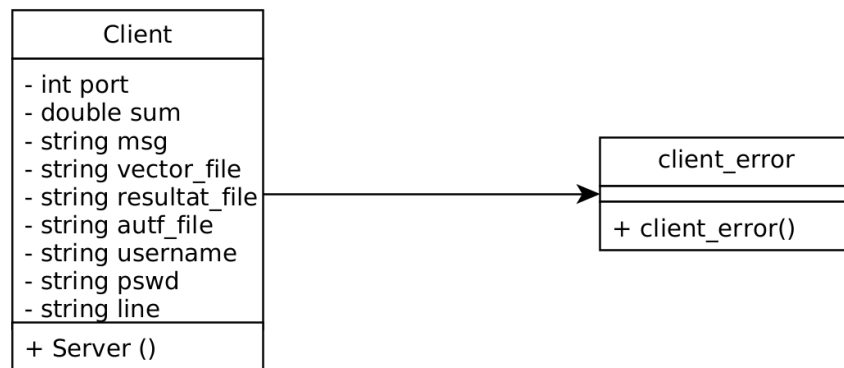


Рисунок А.2 — UML-диаграмма классов

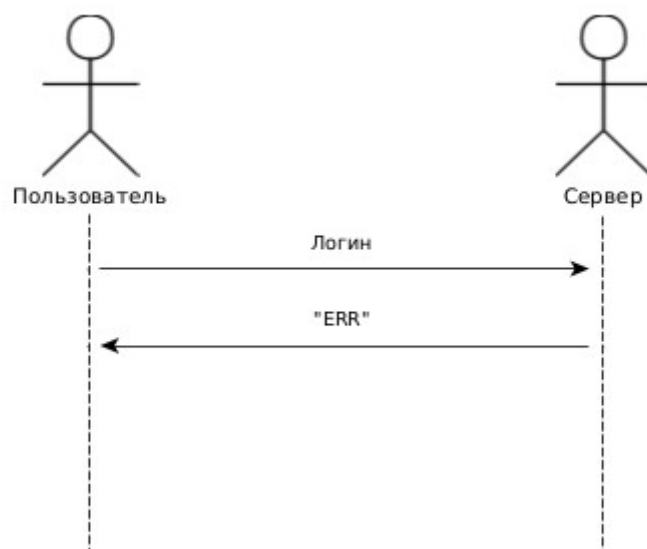


Рисунок А.3 — UML-диаграмма последовательности при неверном логине

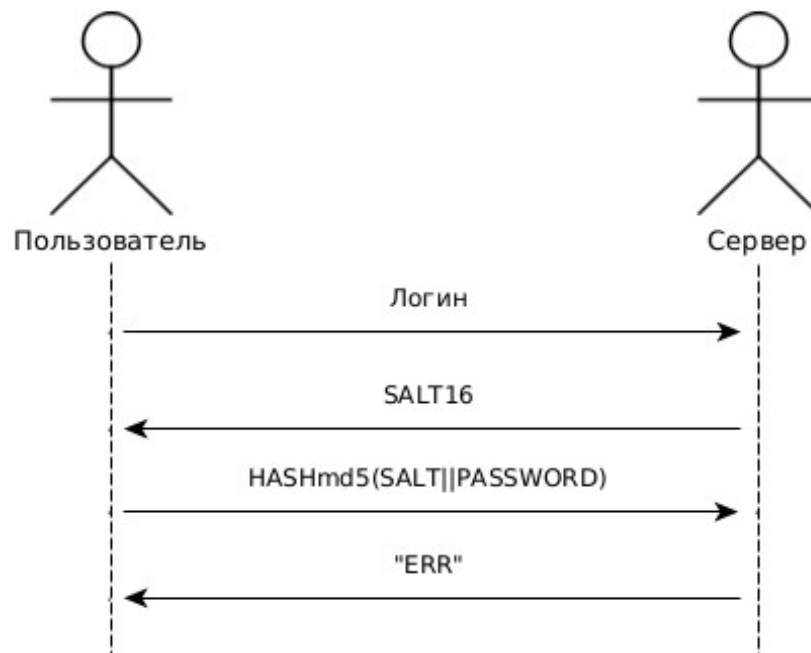


Рисунок А.4 — UML-диаграмма последовательности при неверном хеше

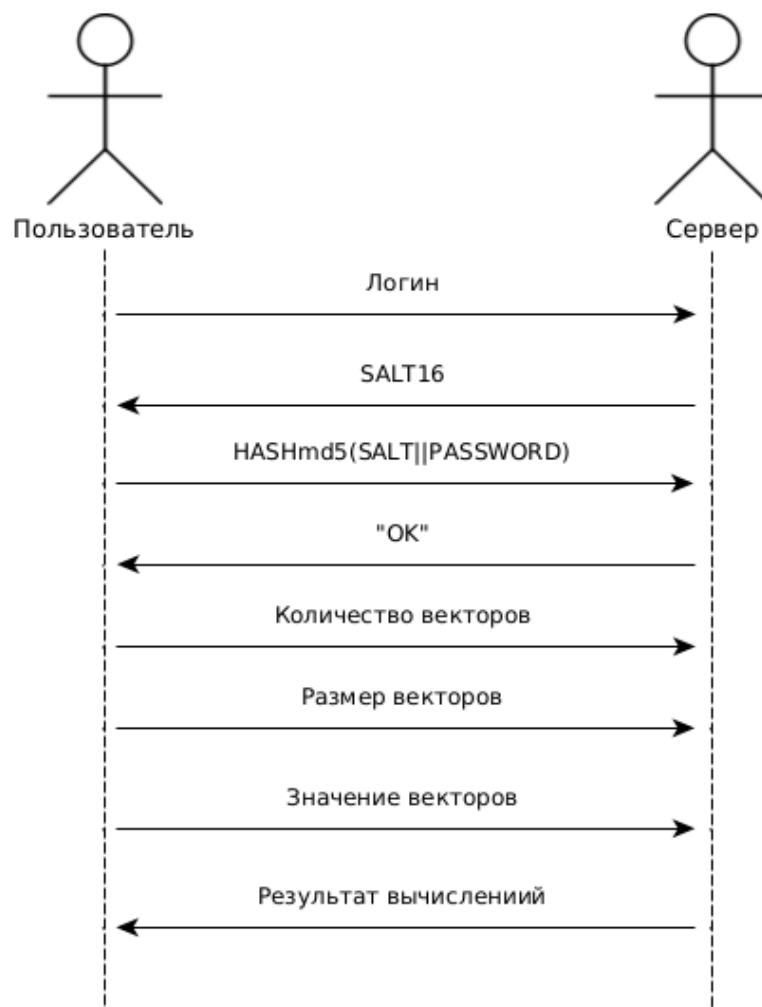


Рисунок А.5 — UML-диаграмма последовательности при удачном сценарии

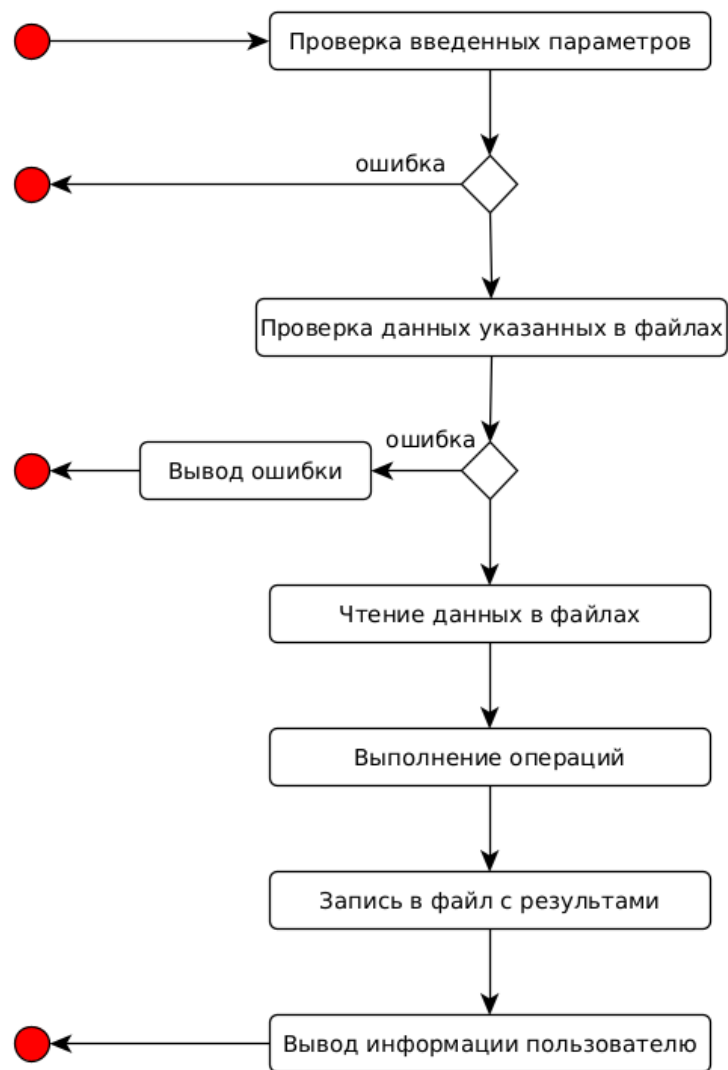


Рисунок А.6 — UML-диаграмма деятельности

Приложение Б

(обязательное)

Коды программ

Код Б. 1 — Client.cpp

```
/**
 * @file Client.cpp
 * @brief Файл взаимодействие с сервером
 */

#include "md5.h"
#include "Client.h"

/**
 * @brief Взаимодействие с сервером
 * @param str1 адрес сервера
 * @param str2 порт сервера
 * @throw client_error класс вызываемый при возникновении ошибки
 */

int Client::Server(string str1, string str2)
{
    //Проверка на введенный порт, если порт не введен выводится, то
    //вводится значение по умолчанию
    if(str2 == "") {
        str2 = "33333";
    }

    //Проверка на введенный файл аунтификации, если файл аунтификации
    //не введен выводится, то вводится значение по умолчанию
    if(auth_file == "") {
        auth_file = "./config/vclient.conf";
    }

    // Открытие файла для аутентификация
    ifstream fautf(auth_file); //fautf объект класса ifstream связан с
    //файлом auth_file

    //2ошибки
    if(!fautf.is_open()) {
```

```

        throw client_error(string("fun:Server, param:autf_file.\nОшибка
открытия файла для аутентификация!"));
    }

    if(fautf.peek() == EOF) {
        fautf.close();

        throw client_error(string("fun:Server, param:autf_file.\nФайл для
аутентификация пуст!"));
    }

    //Считывание логина и пароля
    getline(fautf, line); //Считывает строку из файла в line
    int k = line.find(" "); //Подсчет символов до пробела
    username = line.substr(0, k); //Возвращает подстроку данной строки
начиная с начала строки и до кол-ва сиволов до пробела k
    pswd = line.erase(0, k+1); //Удаляет логин который считан выше вместе
с пробелом и заносит остаток строки в пароль

    // Открытие файла для чтения векторов
    ifstream fvector(vector_file); //fautf объект класса ifstream связан
с файлом vector_file

    //2 ошибки
    if(!fvector.is_open()) {
        fvector.close();

        throw client_error(string("fun:Server, param:vector_file.\nОшибка
открытия файла с векторами!"));
    }

    if(fvector.peek() == EOF) {
        fvector.close();

        throw client_error(string("fun:Server, param:vector_file.\nФайл с
векторами пуст!"));
    }

    // Открытие файла для записи суммы
    ofstream fresultat(resultat_file); //fresultat объект класса ofstream
связан с файлом resultat_file

    //Ошибка
    if(!fresultat.is_open()) {
        fresultat.close();

```

```

        throw client_error(string("fun:Server, param:resultat_file.\n
Ошибка открытия файла для результатов!"));
    }

    //буфер для адреса
    char buf[255];

    try {
        //буфер для адреса
        strcpy(buf, str1.c_str()); //Функция strcpy() используется для
копирования содержимого str1 в буфер, c_str формирует массив и возвращает
указатель на него
    } catch (...) {
        throw client_error(std::string("fun:Server, param:buf.\nНе
возможно получить адрес!"));
    }

    try {
        //Порт
        port = stoi(str2); // stoi из стринг в инт
    } catch (...) {
        throw client_error(string("fun:Server, param:port.\nНе возможно
получить порт!"));
    }

    // структура с адресом нашей программы (клиента)
    sockaddr_in * selfAddr = new (sockaddr_in);
    selfAddr->sin_family = AF_INET; // интернет протокол IPv4
    selfAddr->sin_port = 0; // любой порт на усмотрение ОС
    selfAddr->sin_addr.s_addr = 0; // адрес

    // структура с адресом "на той стороне" (сервера)
    sockaddr_in * remoteAddr = new (sockaddr_in);
    remoteAddr->sin_family = AF_INET; // интернет протокол IPv4
    remoteAddr->sin_port = htons(port); //Порт
    remoteAddr->sin_addr.s_addr = inet_addr(buf); // адрес

    // создаём сокет
    int mySocket = socket(AF_INET, SOCK_STREAM, 0); //tcp протокол

```



```

        if(mySocket == -1) {
            close(mySocket);

            throw client_error(string("fun:Server, param:mySocket.\nОшибка
создания сокета!"));
        }

        //связываем сокет с адрессом
        int rc = bind(mySocket, (const sockaddr *) selfAddr,
sizeof(sockaddr_in));
        if (rc == -1) {
            close(mySocket);

            throw client_error(string("fun:Server, param:selfAddr.\nОшибка
привязки сокета с локальным адресом!"));
        }

        //устанавливаем соединение
        rc = connect(mySocket, (const sockaddr*) remoteAddr,
sizeof(sockaddr_in));
        if (rc == -1) {
            close(mySocket);

            throw client_error(string("fun:Server, param:remoteAddr.\nОшибка
подключения сокета к удаленному серверу!"));
        }

        // буфер для передачи и приема данных
        char *buffer = new char[4096];

        strcpy(buffer, username.c_str()); //Функция strcpy() используется для
копирования содержимого username в буфер, c_str формирует массив и
возвращает указатель на него

        int msgLen = strlen(buffer); //вычисляем длину строки

        // передаём сообщение из буфера
        rc = send(mySocket, buffer, msgLen, 0);
        if (rc == -1) {
            close(mySocket);

            throw client_error(string("fun:Server, param:buffer.\nОшибка
оправки логина!"));
        }

        cout << "Мы отправляем логин: " << buffer << endl;

```

```

// принимаем ответ в буффер
rc = recv(mySocket, buffer, 4096, 0);
if (rc == -1) {
    close(mySocket);
    throw client_error(string("fun:Server, param:buffer.\nОшибка
получения ответа!"));
}
string s1 = string(buffer);
buffer[rc] = '\0'; // конец принятой строки
cout << "Мы получаем соль: " << buffer << endl; // вывод полученного
сообщения от сервера

// Вычисление хэша-кода от SALT+PASSWORD
string hsh = s1 + pswd;
msg = MD5_hash(hsh);

// Отправка хэша от SALT+PASSWORD
strcpy(buffer, msg.c_str());
rc = send(mySocket, buffer, msg.length(), 0);
if (rc == -1) {
    close(mySocket);
    throw client_error(string("fun:Server, param:msg.\nОшибка оправки
хэша!"));
}
cout << "Мы отправляем хэш: " << buffer << endl;

// Получене ответа об аутентификации
rc = recv(mySocket, buffer, sizeof(buffer), 0);
buffer[rc] = '\0'; // конец принятой строки
if (rc == -1) {
    close(mySocket);
    throw client_error(string("fun:Server, param:buffer.\nОшибка
получения ответа об аунтефикации!"));
}
cout << "Мы получаем ответ: " << buffer << endl; // вывод полученного
сообщения от сервера

```

```

// Получение количества векторов
getline(fvector, line);
int len = stoi(line); //из строки в инт

// Отправка количества векторов
rc = send(mySocket, &len, sizeof(len), 0);
if (rc == -1) {
    close(mySocket);
    throw client_error(string("fun:Server, param:buffer.\nОшибка
оправки кол-ва векторов!"));
}

cout << "Мы отправляем кол-во векторов: " << len << endl; // вывод
полученного сообщения от сервера

//Векторы
for(int l = 0; l < len; l++) {
    getline(fvector, line);

    // Получение длины вектора
    int r=0;
    int strl = line.length();
    for (int k=0; k<strl; k++)
        if (line[k]==' ') r++; //поиск пробелов и подсчет их
    int size = r+1; //+1 чтобы получить кол-во значений

    // Получение значений вектора
    uint64_t array[size] = {0};
    for(int j = 0; j < size; j++) {
        string a;
        int i = line.find(" "); //Подсчет символов до пробела
        a = line.substr(0, i); //Возвращает подстроку данной строки
        начиная с начала строки и до кол-ва символов до пробела k
        line = line.erase(0, i+1); //Удаляет значение которое считано
        выше вместе с пробелом и заносит остаток строки в line, так продолжаешь
        до конца строки
        array[j] = stod(a); //из строки в дابل и передаем массивом
    }
}

```

```

        // Отправка размера векторов
        rc = send(mySocket, &size, sizeof(size), 0);
        if (rc == -1) {
            close(mySocket);
            throw client_error(string("fun:Server, param:buffer.\nОшибка
оправки размера векторов!"));
        }

        cout << "Мы отправляем размер вектора: " << size << endl; //
вывод полученного сообщения от сервера

        // Отправка векторов
        rc = send(mySocket, &array, sizeof(array), 0);
        if (rc == -1) {
            close(mySocket);
            throw client_error(string("fun:Server, param:buffer.\nОшибка
оправки самих векторов!"));
        }

        cout << "Мы отправляем сам вектор: " << array << endl; // вывод
полученного сообщения от сервера

        // Получене ответа в виде суммы вектора
        rc = recv(mySocket, &sum, sizeof(sum), 0);
        if (rc == -1) {
            close(mySocket);
            throw client_error(string("fun:Server, param:buffer.\nОшибка
получения ответа в виде суммы!"));
        }

        cout << "Мы получаем результат вычисления: " << sum << endl; //
вывод полученного сообщения от сервера

        fresultat << sum << endl; //запись результата в файл
    }

    // закроем сокет
    close(mySocket);
    delete selfAddr;
    delete remoteAddr;
    delete[] buffer;
    return 0;
}

```

Код Б. 2 — Client.h

```
/**
 * @file Client.h
 * @author Kovalev
 * @version 1.0
 * @date 6.04.2023
 * @copyright ИБСТ ПГУ
 * @brief Заголовочный файл для модуля Client
 */
#include <iostream>
#include <fstream>
#include <unistd.h> //close()
#include <arpa/inet.h>

using namespace std;

/**
 * @brief Класс для подключения к серверу
 * @param port порт
 * @param sum сумма
 * @param msg отправляемый хэш
 * @param vector_file имя файла для считывания векторов
 * @param resultat_file имя файла для записи суммы векторов
 * @param autf_file имя файла, в котором хранятся логины и пароли
  клиентов
 * @param username логин клиента
 * @param line строка из файла
 */

class Client
{
public:
    int Server(string str1, string str2);
    int port; //порт
    uint64_t sum; //сумма
    string msg; //отправляемый хэш
```

```

    string vector_file; //файл векторов
    string resultat_file; //файл результатов
    string autf_file; //файл аунтификации
    string username; //логин
    string pswd; //пароль
    string line; //строка из файла
};

/** @brief Класс обработки ошибок client_error
 * @details Класс выводящий сообщения об ошибках
 */

class client_error: public invalid_argument
{
public:
    explicit client_error (const string& what_arg):
        invalid_argument(what_arg) {}
    explicit client_error (const char* what_arg):
        invalid_argument(what_arg) {}
};

```

Код Б. 3 — main.cpp

```
/**
 * @file main.cpp
 * @brief Главный модуль программы для получения параметров от
 пользователя
 * @param opt переменная для работы с параметрами командной строки
 * @param optarg переменная для получения параметров командной строки
 */

#include "Client.h"

using namespace std;

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

    Client Podclychenie;    //класс для передачи айпи и порта в main

    //Параметры не заданы, выводится справка
    if(argc == 1) {
        cout << "!!!Программа клиента!!!" << endl;
        cout << "Параметры для запуска клиента:" << endl;
        cout << "\t-h - справка" << endl;
        cout << "\t-i - адрес сервера (обязательный)" << endl;
        cout << "\t-p - порт сервера (необязательный - по умолчанию
33333)" << endl;
        cout << "\t-e - имя файла с исходными данными (обязательный)" <<
endl;
        cout << "\t-s - имя файла для сохранения результатов
(обязательный)" << endl;
        cout << "\t-a - имя файла с ID и PASSWORD клиента (необязательный
- по умолчанию ~/.config/vclient.conf)" << endl;
        return 0;
    }

    string str1;//айпи
    string str2;//порт
```

```

//Функция getopt последовательно перебирает переданные параметры в
программу

//optarg переменная для получения параметров командной строки
//opt переменная для работы с параметрами командной строки
int opt;
while ((opt=getopt (argc,argv,"hi:p:e:s:a:"))!=-1) {

    switch(opt) {

        //Кейс справки
        case 'h':
            cout << "!!!Программа клиента!!!" << endl;
            cout << "Параметры для запуска клиента:" << endl;
            cout << "\t-h – справка" << endl;
            cout << "\t-i – адрес сервера (обязательный)" << endl;
            cout << "\t-p – порт сервера (необязательный – по умолчанию
33333)" << endl;
            cout << "\t-e – имя файла с исходными данными (обязательный)"
<< endl;
            cout << "\t-s – имя файла для сохранения результатов
(обязательный)" << endl;
            cout << "\t-a – имя файла с ID и PASSWORD клиента
(необязательный – по умолчанию ~/config/vclient.conf)" << endl;
            return 0;

        //Кейс айпишника
        case 'i':
            str1 = argv[2];

            break;

        //Кейс порта
        case 'p':
            str2 = string(optarg);
            break;
    }
}

```



```

//Кейс файла с векторами
case 'e':
    Podclychenie.vector_file = string(optarg);
    break;

//Кейс файла с результатов
case 's':
    Podclychenie.resultat_file = string(optarg);
    break;

//Кейс файла с логином и паролем
case 'a':
    Podclychenie.autf_file = string(optarg);
    break;

//Кейс если параметры не заданы
case '?':
    cout << "Праметр задан не верно или такого праметра не
существует" << endl;
    return 0;
};

};

Podclychenie.Server(str1, str2);
return 0;
};

```

Код Б. 4 — md5.cpp

```
/**
 * @file md5.cpp
 * @brief Файл для хэширования md5
 */

#include "md5.h"

using namespace CryptoPP;

/**
 * @brief Получение хэша по алгоритму md5
 * @param hsh сообщения получаемое для хэширования
 * @return Полученный хэш
 */

string MD5_hash(string hsh)
{
    string massange; // определяем переменную для хранения хэша

    Weak1::MD5 hashmd4; // Определяем переменную для типа хэша из
    библиотеки CryptoPP

    StringSource(hsh, true, // источник-стока
                 new HashFilter(hashmd4, // фильтр-"хеширователь"
                                new HexEncoder( // кодировщик в
строку цифр
                                new StringSink(massange)))); //
строка-приемник

    // выводим сообщение и его хэш
    return massange;
}
```

Код Б. 5 — md5.h

```
/**
 * @file md5.h
 * @author Kovalev
 * @version 1.0
 * @date 06.04.2023
 * @copyright ИБСТ ПГУ
 * @brief Заголовочный файл для модуля md5
 */

#include <cryptopp/hex.h> // HexEncoder
#define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1
#include <cryptopp/md5.h> // md5

using namespace std;

string MD5_hash(string hsh);
```

Код Б. 6 — Makefile

```
target = client
sources = main.cpp Client.h Client.cpp md5.cpp md5.h
CC = g++
CFlags = -Wall
LDLIBS = -lcrypto++
OPT = -Ofast
all:build
build:
    $(CC) $(CFlags) $(OPT) $(sources) -o $(target) $(LDLIBS)
dbg:
    $(CC) -g $(sources) -o $(target)DBG
```

Код Б. 7 — UnitTest.cpp

```
#include <UnitTest++/UnitTest++.h>
#include "Client.h"

using namespace std;

/*
    Для макроса TEST_FIXTURE можно объявить специальный класс, в
    конструкторе которого
    будут выполняться действия, предваряющие тест, а в деструкторе —
    завершающие.
*/

struct Cons_fix {
    Client * p;
    Cons_fix()
    {
        p = new Client();
    }
    ~Cons_fix()
    {
        delete p;
    }
};

SUITE(Server) //Макрос. FIXTURE при одинаковых аргументах
{
    TEST_FIXTURE(Cons_fix, NormalTest) {
        //1 Удачный сценарий
        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/vector.txt";
        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/resultat.txt";
        p->autf_file =
"/home/stud/C++Projects/123144/build-Debug/bin/config/vclient.conf";
        p->Server("127.0.0.1", "33333");
        CHECK(true);
    }
}
```

```

    }

    TEST_FIXTURE(Cons_fix, ErrIp) {
        //2 Подключение к серверу. Введен не верный адрес
        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/vector.txt";
        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/resultat.txt";
        p->autf_file =
"/home/stud/C++Projects/123144/build-Debug/bin/config/vclient.conf";
        CHECK_THROW(p->Server("2215024", "33333"), client_error);
    }

    TEST_FIXTURE(Cons_fix, ErrPort) {
        //3 Подключение к серверу. Введен не верный порт
        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/vector.txt";
        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/resultat.txt";
        p->autf_file =
"/home/stud/C++Projects/123144/build-Debug/bin/config/vclient.conf";
        CHECK_THROW(p->Server("127.0.0.1", "3445"), client_error);
    }

    TEST_FIXTURE(Cons_fix, ErrPutyFileLogParol) {
        //4 Ошибка открытия файла с логинами и паролями
        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/vector.txt";
        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/resultat.txt";
        p->autf_file =
"/home/stud/C++Projects/123144/3/12235/config/vclient.conf";
        CHECK_THROW(p->Server("127.0.0.1", "33333"), client_error);
    }

    TEST_FIXTURE(Cons_fix, ErrPustoyFileLogParol) {
        //5 Ошибка чтения из файла с логинами и паролями

```

```

        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/vector.txt";

        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/resultat.txt";

        p->autf_file =
"/home/stud/C++Projects/123144/build-Debug/bin/pustoy.txt";

        CHECK_THROW(p->Server("127.0.0.1", "33333"), client_error);
    }

    TEST_FIXTURE(Cons_fix, ErrLogin) {
        //6 Ошибка идентификации. Введен не правильный логин

        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/my.txt";

        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/resultat.txt";

        p->autf_file =
"/home/stud/C++Projects/123144/build-Debug/bin/Nouser.conf";

        CHECK_THROW(p->Server("127.0.0.1", "33333"), client_error);
    }

    TEST_FIXTURE(Cons_fix, ErrParol) {
        //7 Ошибка аутентификации. Введен не правильный пароль

        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/vector.txt";

        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/resultat.txt";

        p->autf_file =
"/home/stud/C++Projects/123144/build-Debug/bin/NoPswd.conf";

        CHECK_THROW(p->Server("127.0.0.1", "33333"), client_error);
    }

    TEST_FIXTURE(Cons_fix, ErrPutyFileVectors) {
        //8 Ошибка открытия файла с векторами

        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/12432.txt";

        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/resultat.txt";

        p->autf_file =
"/home/stud/C++Projects/123144/build-Debug/bin/config/vclient.conf";

        CHECK_THROW(p->Server("127.0.0.1", "33333"), client_error);
    }

```

```

    }

    TEST_FIXTURE(Cons_fix, ErrPustoyFileVectors){
        //9 Ошибка чтения из файла с векторами
        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/pustoy.txt";
        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/resultat.txt";
        p->autf_file =
"/home/stud/C++Projects/123144/build-Debug/bin/vclient.conf";
        CHECK_THROW(p->Server("127.0.0.1", "33333"), client_error);
    }

    TEST_FIXTURE(Cons_fix, ErrPutyFileResultat) {
        //10 Ошибка открытия файла для записи суммы
        p->vector_file =
"/home/stud/C++Projects/123144/build-Debug/bin/vector.txt";
        p->resultat_file =
"/home/stud/C++Projects/123144/build-Debug/bin/142235/resultat.txt";
        p->autf_file =
"/home/stud/C++Projects/123144/build-Debug/bin/config/vclient.conf";
        CHECK_THROW(p->Server("127.0.0.1", "33333"), client_error);
    }
}

int main(int argc, char **argv)
{
    return UnitTest::RunAllTests();
}

```


Приложение В

(обязательное)

Проверка на антиплагиат

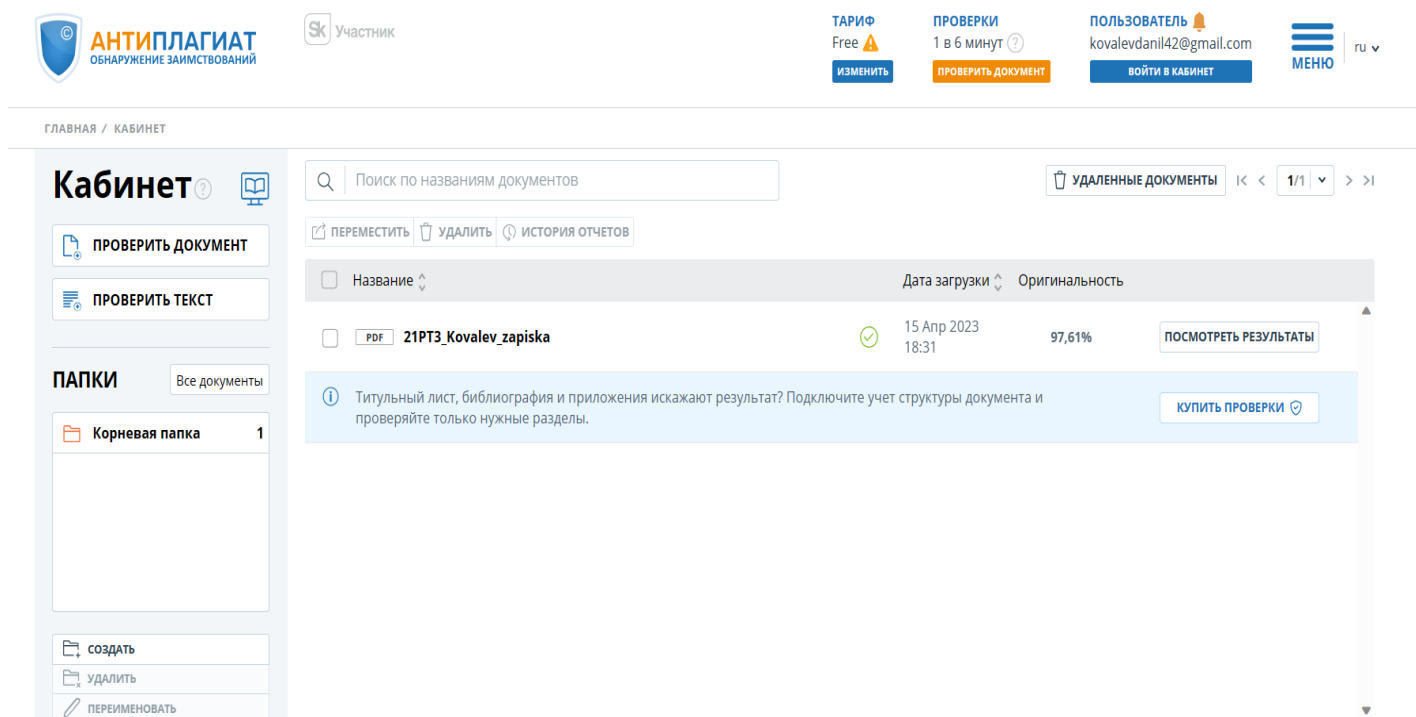


Рисунок В.1 — Проверка на антиплагиат