

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.22ПТ107.01.ПЗ

Специальность — 10.05.02 Информационная безопасность  
телекоммуникационных систем.

Специализация — Разработка защищенных телекоммуникационных систем

Выполнил студент: \_\_\_\_\_ Крестина С.Д.

Группа: 22ПТ1

Руководитель:

к.т.н., доцент \_\_\_\_\_ Лупанов М.Ю.

Работа защищена с оценкой \_\_\_\_\_

Дата защиты \_\_\_\_\_

УТВЕРЖДАЮ

Зав. кафедрой ИБСТ

к.т.н., доцент

Зефирова С.Л.

18 09 2023 г.

## ЗАДАНИЕ

на курсовую работу

по теме: Программная реализация сетевого сервера

1 Дисциплина: Технологии и методы программирования.

2 Студент: Крестина Софья Дмитриевна

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 одностороннем режиме, в том числе:
  - идентификацию и аутентификацию подключаемого клиента;
  - выполнение вычислений над данными, передаваемыми клиентом;
  - операция, выполняемая над данными — среднее арифметическое.

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  $SALT_n$  (при успешной идентификации)
  - 3б. сервер передает строку "ERR" и разрывает соединение(при ошибке идентификации)
  4. клиент передает  $HASH_{MD5}(SALT_n || PASSWORD)$
  - 5а. сервер передает OK 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  $SALT_n$  — 16 шестнадцатеричных цифр;
- метод обеспечения постоянного размера строки  $SALT_n$  при различных значениях числа  $SALT$  — дополнение слева цифрами «0»;
- используемая хэш-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 `double`, значения вектора;
- формат передаваемых данных — двоичный:
  - поле «результат вычислений по вектору» 8 байт, число типа `double`;
- функция обработки — вычисление среднего арифметического вектора;

#### 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 `/home/stud/bas`;
  - имя файла с журналом работы, необязательный;
    - значение по умолчанию `/home/stud/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 09 сентября 2023
- разработка диаграмм вариантов использования 18 сентября 2023  
и диаграмм классов
- разработка диаграмм последовательностей и 02 октября 2023  
диаграмм деятельности
- разработка кода программы 28 октября 2023
- разработка модульных тестов и выполнение 11 ноября 2023  
тестирования
- разработка функциональных тестов и 25 ноября 2023  
приемочное тестирование
- разработка документации 09 декабря 2023
- оформление отчета о курсовой работе 11 декабря 2023
- защита курсовой работы 23 декабря 2023

Руководитель работы

Задание получил *16.09.2023г* 2023г.

Студент

Нормоконтролер

 М.Ю. Лупанов

 С.Д. Крестина

 М.Ю. Лупанов

## Содержание

Введение.....	7
1 Проектирование.....	9
2 Разработка программы.....	11
3 Модульное тестирование.....	13
4 Функциональное тестирование.....	16
5 Документирование.....	18
Заключение.....	19
Приложение А.....	20
Приложение Б.....	25

## Введение

В современном мире сетевые серверы играют важную роль в передаче и обработке данных между клиентами и серверными системами. В связи с этим, разработка эффективных и надежных серверных программ становится необходимой задачей для многих компаний и организаций.

Целью данной курсовой работы является разработка серверной программы для клиент-серверной системы обработки данных. Для достижения этой цели поставлены следующие задачи:

- Анализ требований к программе.
- Построение UML-диаграмм вариантов использования и проектирование пользовательского интерфейса
- Построение UML-диаграмм классов и планирование модулей.
- Построение UML-диаграмм последовательностей.
- Построение UML-диаграмм деятельности.
- Разработка серверной программы.
- Разработка модульных тестов и проведение модульного тестирования.
- Разработка функциональных тестов и проведение приемочного тестирования.
- Проведение документирования кода программы с использованием Doxygen.

Курсовая работа изложена на 55 страницах печатного текста состоит из введения, 5 глав, заключения и двух приложения. Во введении обоснована актуальность темы, сформулированы цели, задачи исследования. В первой главе проведено проектирование программы, во второй главе проведена разработка кода программы, в третьей главе проведено модульное тестирование, в четвертой главе проведено функциональное тестирование программы, в пятой главе написана документация программы. В заключении подведён итог результатов исследования и сделаны выводы по проделанной работе.

## 1 Проектирование

Проектирование — процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части. Данный этап позволяет определить архитектуру будущей программы и упростить процесс разработки.

В данной главе проведено проектирование программы. Были разработаны UML диаграммы следующих типов:

- Диаграмма вариантов использования(прецедентов)
- Диаграмма классов
- Диаграмма деятельности
- Диаграмма последовательности

Диаграмма прецедентов или диаграмма вариантов использования в UML — диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

На диаграмме вариантов использования приведены возможные взаимодействия актёров и программы сервера.

Диаграмма классов — структурная диаграмма языка моделирования UML, демонстрирующая общую структуру иерархии классов системы, их коопераций, атрибутов (полей), методов, интерфейсов и взаимосвязей (отношений) между ними.

На диаграмме классов изображены классы, необходимые для реализации функций программы и их связь между друг другом.

Диаграмма деятельности — UML-диаграмма, на которой показаны действия. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов — вложенных видов деятельности и



отдельных действий , соединённых между собой потоками, которые идут от выходов одного узла ко входам другого.

На диаграмме деятельности представлена логика работы программы-сервера.

Диаграмма последовательности — UML-диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актеров (действующих лиц) информационной системы в рамках прецедента.

На диаграмме последовательности изображено взаимодействие классов программы во времени. Показан переход управления одних классов к другим и функции, с помощью которых этот переход осуществляется.

Диаграммы сделаны при помощи программы Umbrello. Диаграммы представлены в приложении А.

### **Проект интерфейса**

Спроектируем интерфейс, основанный на передаче параметров при запуске программы, для работы сервера и реализуем его с помощью функций getopt.

Интерфейс выполняет функцию получения ПКС и передачи их значений в другие функции программы. В ПКС передаются следующие параметры:

а) путь к файлу с базой клиентов б) путь к файлу с журналом работы в) порт для работы сервера

Спроектируем интерфейс с короткими параметрами.

Для параметра указания пути к файлу с базой клиентов выбрано обозначение -b. После чего вводится строка , которая является полным путем до файла с базой данных, в случае, если файл расположен в той же папке , что и код программы, достаточно ввести название файла. Имеет значение по умолчанию.

Для параметра указания пути к файлу с журналом работы выбрано обозначение -l. После чего вводится строка , которая является полным путем до файла с журналом работы, в случае, если файл расположен в той же папке , что и код программы, достаточно ввести название файла. Имеет значение по умолчанию.

Для параметра указания порта, на котором будет работать сервер выбрано обозначение -p. После чего вводится целое число, которое является значением порта. Имеет значение по умолчанию. Имеет ограничениеЖ порт выбирается среди значений начиная с 1023 до 65536

Для параметра вывода справки выбрано обозначение -h. Выводится справка об использовании. Имеет приоритет среди операций.

Результат проектирования был сведен в таблицу 1.

Таблица 1 - Проект командного интерфейса

<b>Параметр</b>	<b>Значение</b>	<b>Функции</b>	<b>Умолчание</b>	<b>Зависимость</b>
-b	string	Путь к файлу с базой клиентов	base.txt	Нет
-l	string	Путь у файлу с журналом работом	log.txt	Нет
-p	int от 1023 до 65536	Порт сервера	33333	Нет
-h	нет	Справка	При ошибке	Приоритет

## 2 Разработка программы

На данном этапе проведена разработка программы сервера на языке программирования C++ с применением классовой структуры.

Программа выполняет следующие функции:

- а) чтение базы пользователей при старте программы;
- б) обеспечение возможности подключения клиента в течении всего времени функционирования;
- в) обработка запросов клиентов в однопоточном режиме, в том числе:
  - идентификацию и аутентификацию подключаемого клиента;
  - выполнение вычислений на данными, передаваемыми клиентом;
  - операция, выполняемая над данными — среднее арифметическое.

В ходе работы были разработаны следующие модули:

- calculator;
- communicator;
- interface;
- logger;
- programerror;
- userbase.

Модуль calculator обеспечивает вычисления среднего арифметического по векторам, присылаемыми клиентом.

Модуль communicator обеспечивает аутентификацию клиента на сервере, так же модуль отвечает за работу с сокетами и обмен данными с клиентом. Позволяет вычислить хэш-отпечаток от строки ( SALT16 || PASSWORD) и сравнить с хэш-отпечатком, который присылает клиент, взаимодействовать с клиентом.

- разрядность случайного числа SALT — 64 бита;
- длина строки SALT16 — 16 шестнадцатеричных цифр;

- метод обеспечения постоянного размера строки SALT16 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

Модуль `interface` обеспечивает передачу информации от пользователя программе через терминал и включение остальных модулей. Интерфейс обеспечивает передачу программе следующей информации:

- 1) имя файла с базой клиентов, необязательный;
  - значение по умолчанию «DB.txt»;
- 2) имя файла с журналом работы, необязательный;
  - значение по умолчанию «log.txt»;
- 3) номер порта, на котором будет работать сервер, необязательный;
  - значение по умолчанию 33333;

Программа предусматривает выдачу справки о пользовательском интерфейсе

- при запуске с параметром `-h`;
- при передаче несуществующего параметра.

Модуль `logger` обеспечивает получение времени и запись сообщений или ошибок в журнал работы.

Модуль `programmerlog` обеспечивает обработку ошибок, присвоение статуса критичности: при `true` статус преобразуется в строку «Критическая», в случае с `false` — в «Штатная».

Модуль `userbase` обеспечивает получение пары «идентификатор:пароль» и внесение этих данных в словарь.

Подробности реализации приведены в документации, рассмотренной в главе 5.

Код модулей расположен в репозитории на [github](https://github.com).

Ссылка на репозиторий:<https://github.com/KrestinaSD/kursovichok>

### 3 Модульное тестирование

При создании курсовой, на данном этапе, были разработаны модульные тесты для проверки корректности работы программы.

Сценарии:

Проверка модуля коммуникации

- размер соли
- неправильная соль(негативный)
- пустая соль(негативный)
- неправильный пароль(негативный)
- пустой пароль(негативный)
- нормальная соль и пароль

Проверка модуля вычислений

- пустой вектор
- переполнение вверх (позитивный)
- переполнение вниз (позитивный)
- нормальный вектор (позитивный)

Проверка модуля подключения к базе данных

- пустой путь (негативный)
- неправильный путь (негативный)

Проверка модуля записи логов

- пустой путь (негативный)
- неправильный путь (негативный)
- пустое сообщение (негативный)
- генерация времени
- нормальный путь и сообщение (позитивный)

Таблица 2 - Сценарии тестирования модуля коммуникации

№	Тест	Входные данные	Ожидаемый результат	Полученный результат
1.1	SIZE_SALT	Нет	true	true
1.2	BAD_SALT	«0000»	server_error	server_error
1.3	EMPTY_SALT	«»	server_error	server_error
1.4	BAD_PASS	«badpassword»	server_error	server_error
1.5	EMPTY_PASS	«»	server_error	server_error
1.6	NORM	«password»	true	true

*Примечания:*

*«» в путь к файлу не входят*

Таблица 3 - Сценарии тестирования модуля вычисления

№	Тест	Входные данные	Ожидаемый результат	Полученный результат
2.1	EMPTY_VEC	Нет	server_error	server_error
2.2	MAX_OVER_VEC	{1.79769e+308, 1.79769e+308, 1.79769e+308}	inf	inf
2.3	MIN_OVER_VEC	{-1.79769e+308, -1.79769e+308, -1.79769e+308}	-inf	-inf
2.4	NORM_VEC	{1, 2, 3}	2	2

Таблица 4 - Сценарии тестирования модуля подключения к базе данных

№	Тест	Входные данные	Ожидаемый результат	Полученный результат
3.1	EMPTY_PATH	«»	server_error	server_error
3.2	BAD_PATH	«no/base/file.txt»	server_error	server_error

*Примечания:*

*«» в путь к файлу не входят*

Таблица 5 - Сценарии тестирования модуля записи логов

№	Тест	Входные данные	Ожидаемый результат	Полученный результат
4.1	EMPTY_PATH_FILE	«»	invalid_argument	invalid_argument
4.2	VERY_BAD_PATH	«Путь/к/лог/файлу.txt»	invalid_argument	invalid_argument
4.3	EMPTY_MESSAGE	Нет	invalid_argument	invalid_argument
4.4	VREMYA	Нет	true	true
4.5	NORM	«Test message»	0	0

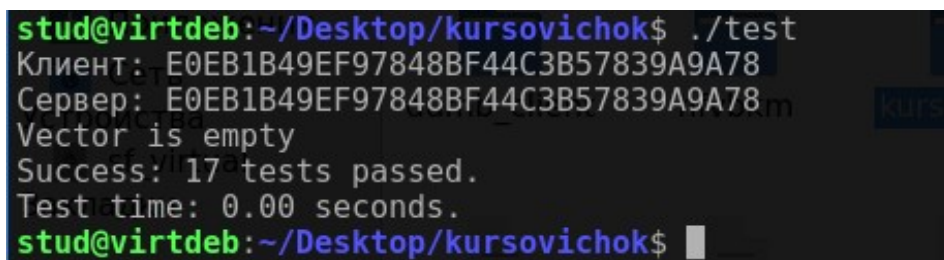
*Примечания:*

*«» в путь к файлу не входят*

Код модульных тестов, а также же этот проект представлены в репозитории:

<https://github.com/KrestinaSD/kursovichok.git>

Все тесты пройдены. Результат представлен на рисунке 1.



```

stud@virtdeb:~/Desktop/kursovichok$ ./test
Клиент: E0EB1B49EF97848BF44C3B57839A9A78
Сервер: E0EB1B49EF97848BF44C3B57839A9A78
Vector is empty
Success: 17 tests passed.
Test time: 0.00 seconds.
stud@virtdeb:~/Desktop/kursovichok$

```

Рисунок 1 Результат выполнения тестов



#### 4 Функциональное тестирование

Сценарии:

Проверка разбора ПКС

- запуск со всеми параметрами(позитивный)
- неверный путь до базы данных(негативный)
- неверный путь до журнала работы(негативный)
- некорректный порт (позитивный)
- передача несуществующего параметра(позитивный)
- запуск параметра -h (позитивный)

Проверка стабильности работы сервера:

- Запуск 1000 корректный клиентов;
- Запуск 1000 клиентов с другим типом данных

Таблица 6 - Сценарии тестирования разбора ПКС

№	Тест	Входные данные	Ожидаемый результат	Полученный результат
1.1	Все параметры	-l log.txt -b bas.txt -p 7777	Без исключений	Без исключений
1.2	Неверный путь до БД	-b ne/basa.txt	Исключение server_error	Исключение server_error
1.3	Неверный путь до ЖР	-l ne/log/log.txt	Исключение server_error	Исключение server_error
1.4	Некорректный порт	-p 88888888	Вывод справки	Вывод справки
1.5	Передача несуществующего параметра	-o	Исключение invalid option Вывод справки	Исключение invalid option Вывод справки
1.6	Вывод справки	-h	Вывод справки	Вывод справки

Таблица 7 - Сценарии тестирования стабильности работы сервера

№	Тест	Входные данные	Ожидаемый результат	Полученный результат
2.1	Клиент с типом данных double	Нет	Стабильная работа	Стабильная работа
2.2	Клиент с типом данных int32_t	Нет	Стабильная работа	Стабильная работа

```
stud@virtdeb:~/Desktop/kursovichok/code$ for i in range{1..1000}; do ./client_double; done
```

Рисунок 2 Запуск 1000 клиентов с типом данных double

```
stud@virtdeb:~/Desktop/kursovichok/code$ for i in range{1..1000}; do ./client_int32_t; done
```

Рисунок 3 Запуск 1000 клиентов с типом данных int32\_t

Данный проект представлен в репозитории:  
<https://github.com/KrestinaSD/kursovichok.git>

## 5 Документирование

На этом этапе было произведено документирование кода программы с использованием Doxygen.

Для создания автоматической документации необходимо сформировать в программах специальные блоки документации. Блоки документации в программах и документация в формате HTML находятся в репозитории GitHub: <https://github.com/KrestinaSD/kursovichok>

Документация представлена в приложении. Пример документации модуля calculator представлен на рисунке ниже.

```
communicator.cpp x calculator.h x
/**
 * @file calculator.h /home/stud/Desktop/kursovichok/code/
 * @author Крестина С.Д.
 * @version 1.0
 * @brief Заголовочный файл для модуля calculator, отвечающий за вычисления
 */

#pragma once
#include <vector>
#include <iostream>
#include <exception>
#include <limits>
#include "programmererror.h"

/**
 * @brief Класс для вычисления среднего арифметического вектора
 * @details Метод для вычисления
 */
class Average
{
private:
    double result; ///< Результат вычислений
public:
    /**
     * @brief Конструктор класса без параметров
     */
    Average(){};
    /**
     * @brief Функция вычисления среднего арифметического вектора
     * @param [in] vector<double>& arr вектор со значениями типа double
     * @return result
     */
    double average(std::vector<double>& arr);
};
```

Рисунок 4 Пример документации модуля

## Заключение

В ходе работы над курсовым проектом была осуществлена программная реализация сетевого сервера. Было построено 4 диаграммы и реализовано 6 модулей.

В первой главе проведено проектирование программы.

Во второй главе проведена разработка кода программы.

В третьей главе проведено модульное тестирование.

В четвёртой главе проведено функциональное тестирование программы

В пятой главе написана документация программы.

Во время выполнения курсовой работы было разработано и протестировано серверное приложение с использованием TCP протокола и сокетов в Linux. В результате работы были созданы диаграммы, написан код сервера и тесты для проверки его работоспособности, а также составлена техническая документация и отчёт о проделанной работе. Все компоненты программы были сохранены на GitHub.

Репозиторий GitHub: <https://github.com/KrestinaSD/kursovichok>

## Приложение А

В данном приложении приведены UML-диаграммы.



Рисунок 5 Диаграмма вариантов использования





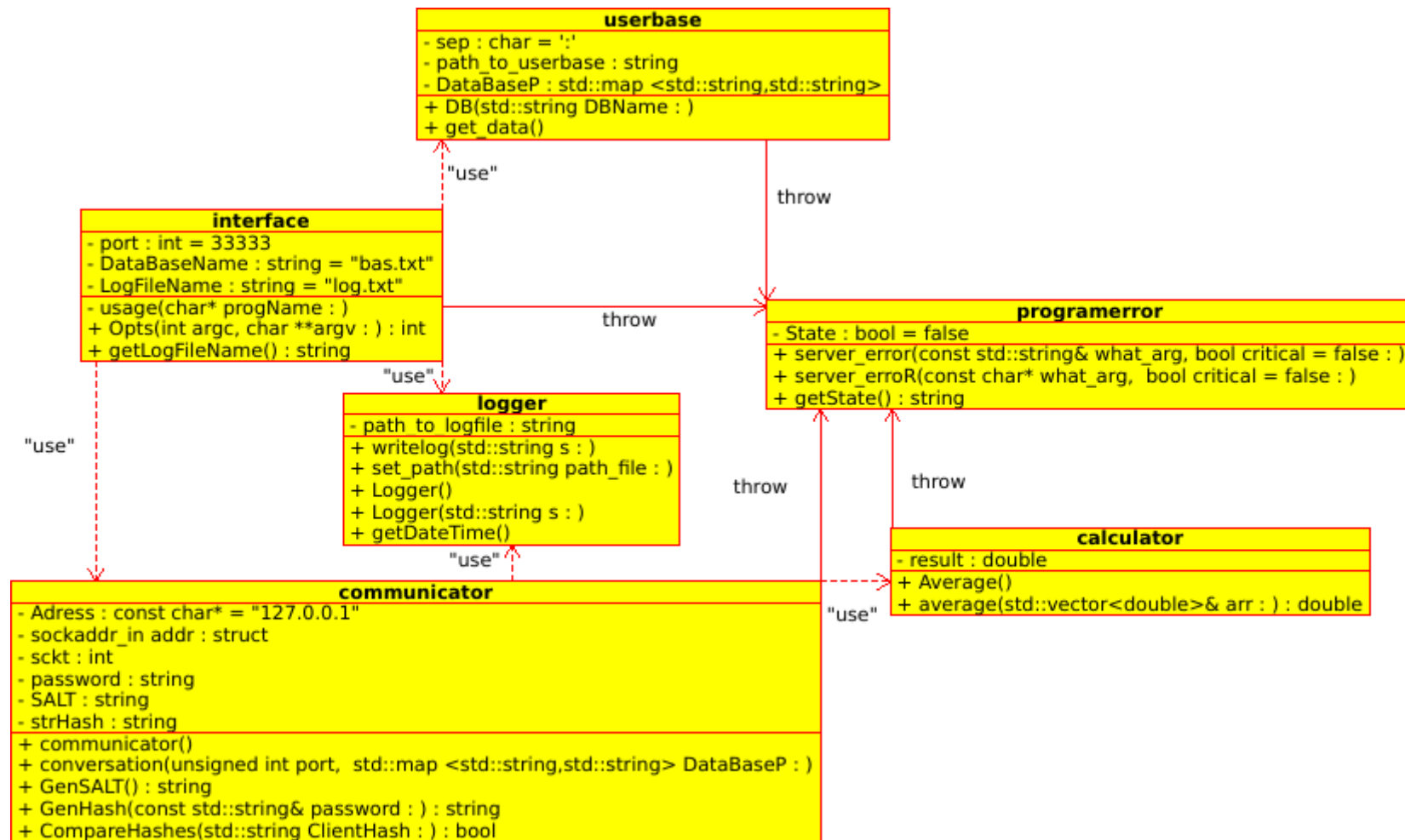


Рисунок 6 Диаграмма классов

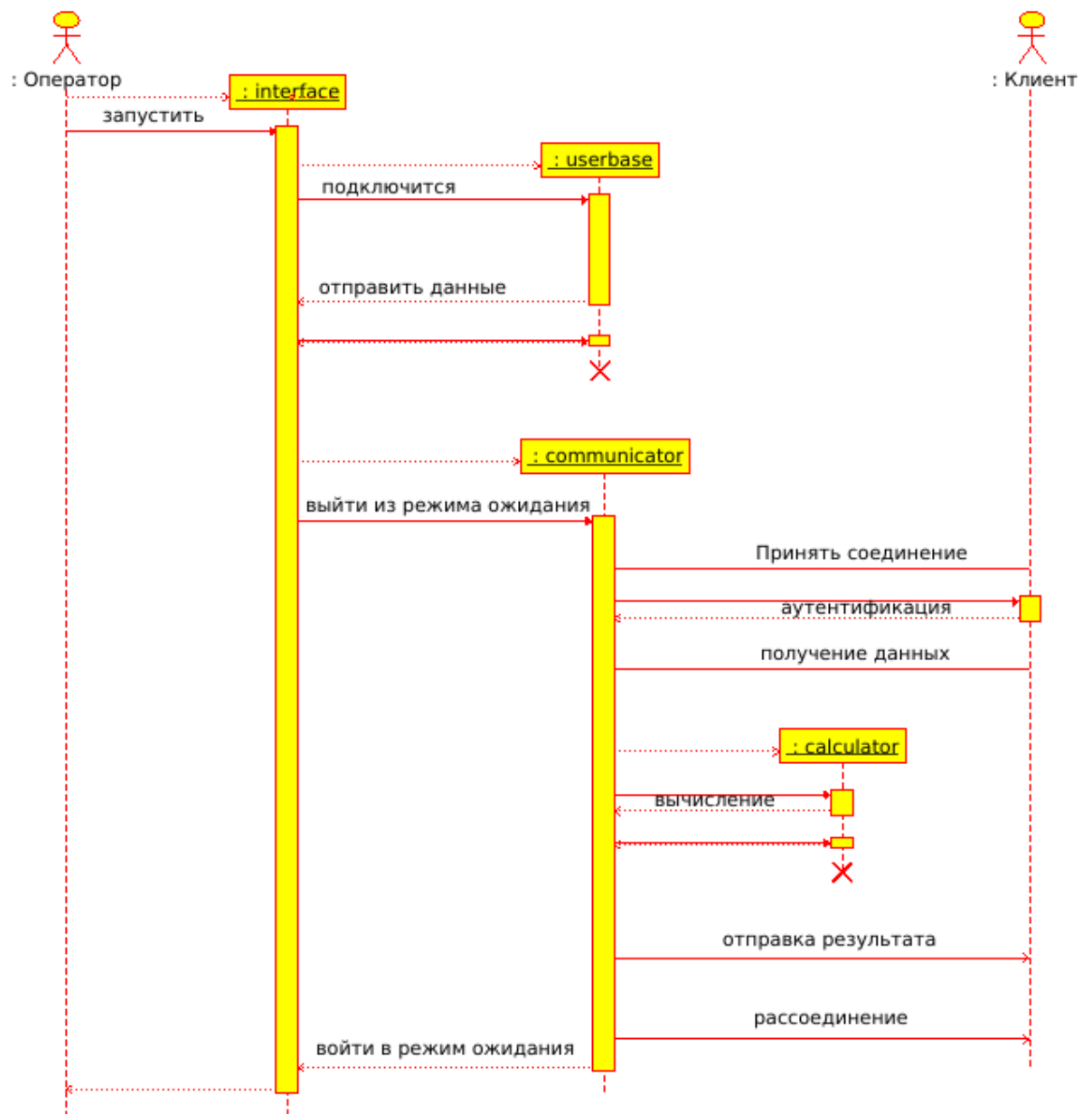


Рисунок 7      Диаграмма последовательности

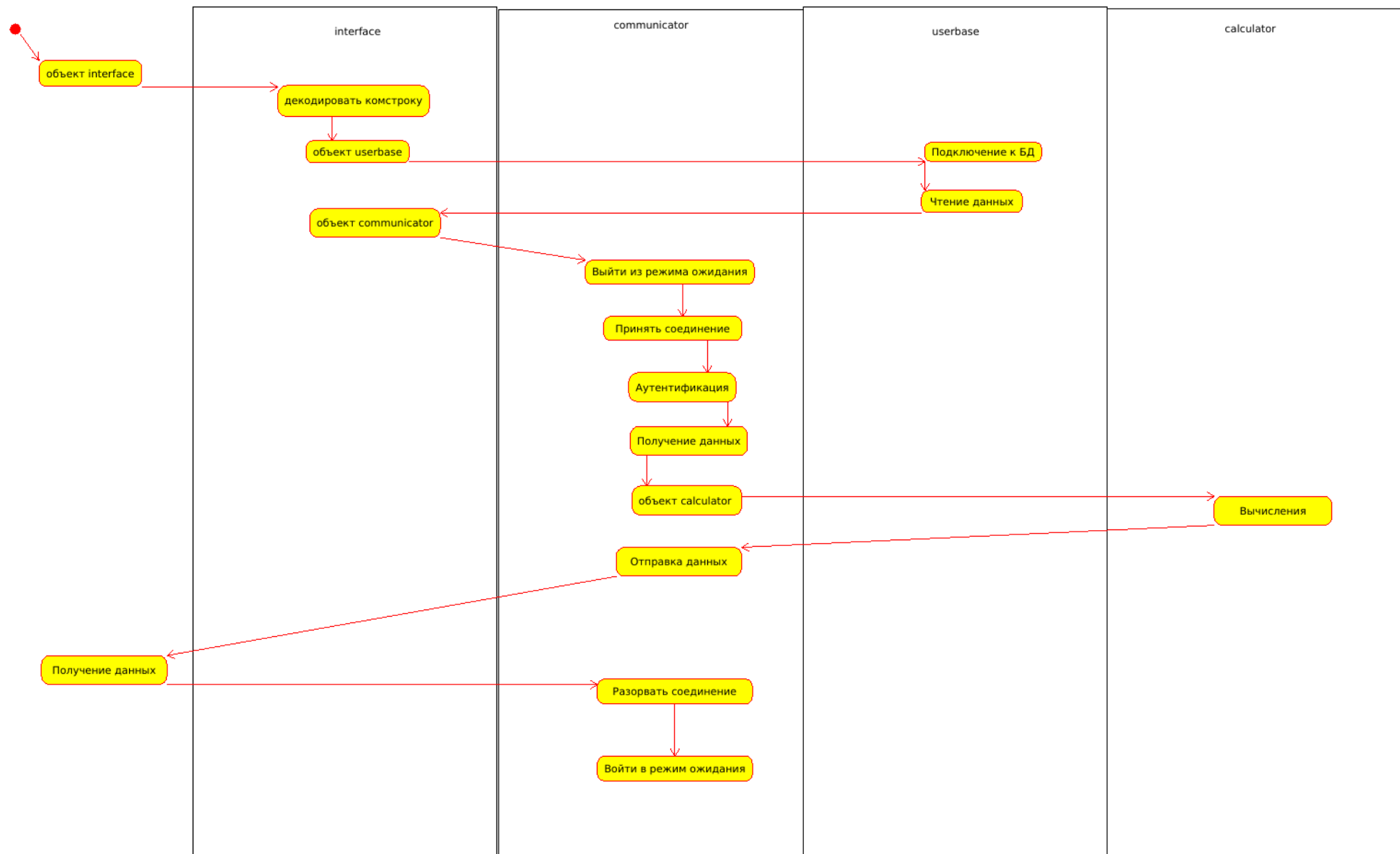


Рисунок 8      Диаграмма деятельности

## Приложение Б

В данном приложении приведена документация программы.

Программная реализация сетевого сервера. Сервер, вычисляющий  
среднее арифметическое вектора типа double.

1.0

Создано системой Doxygen 1.9.1



1 Иерархический список классов	1
1.1 Иерархия классов	1
2 Алфавитный указатель классов	3
2.1 Классы	3
3 Список файлов	5
3.1 Файлы	5
4 Классы	7
4.1 Класс Average	7
4.1.1 Подробное описание	7
4.1.2 Методы	7
4.1.2.1 average()	7
4.2 Класс communicator	8
4.2.1 Подробное описание	8
4.2.2 Методы	8
4.2.2.1 CompareHashes()	8
4.2.2.2 conversation()	9
4.2.2.3 GenHash()	9
4.2.2.4 GenSALT()	10
4.3 Класс DB	10
4.3.1 Подробное описание	10
4.3.2 Конструктор(ы)	10
4.3.2.1 DB()	10
4.3.3 Методы	11
4.3.3.1 get_data()	11
4.4 Класс interface	11
4.4.1 Подробное описание	11
4.4.2 Методы	12
4.4.2.1 getLogFileName()	12
4.4.2.2 Opts()	12
4.5 Класс Logger	12
4.5.1 Подробное описание	13
4.5.2 Конструктор(ы)	13
4.5.2.1 Logger()	13
4.5.3 Методы	13
4.5.3.1 getDateTime()	13
4.5.3.2 set_path()	14
4.5.3.3 writelog()	14
4.6 Класс server_error	14
4.6.1 Подробное описание	15
4.6.2 Конструктор(ы)	15
4.6.2.1 server_error() [1/2]	15

---

4.6.2.2 <code>server_error()</code> [2/2]	16
5 Файлы	17
5.1 Файл <code>calculator.h</code>	17
5.1.1 Подробное описание	18
5.2 Файл <code>communicator.h</code>	18
5.2.1 Подробное описание	19
5.3 Файл <code>interface.h</code>	19
5.3.1 Подробное описание	20
5.4 Файл <code>logger.h</code>	20
5.4.1 Подробное описание	21
5.5 Файл <code>programmer.h</code>	22
5.5.1 Подробное описание	23
5.6 Файл <code>userbase.h</code>	23
5.6.1 Подробное описание	24
Предметный указатель	25



# Глава 1

## Иерархический список классов

### 1.1 Иерархия классов

Иерархия классов.

Average . . . . .	7
communicator . . . . .	8
DB . . . . .	10
interface . . . . .	11
std::invalid_argument	
server_error . . . . .	14
Logger . . . . .	12



## Глава 2

# Алфавитный указатель классов

### 2.1 Классы

Классы с их кратким описанием.

<a href="#">Average</a>	Класс для вычисления среднего арифметического вектора . . . . .	7
<a href="#">communicator</a>	Класс для связи с клиентом . . . . .	8
<a href="#">DB</a>	Класс для получения данных из файла базы клиентов . . . . .	10
<a href="#">interface</a>	Класс для разбора командной строки и включения других модулей . . . . .	11
<a href="#">Logger</a>	Класс для формирования и записи логов . . . . .	12
<a href="#">server_error</a>	Класс вызова исключений . . . . .	14



## Глава 3

# Список файлов

### 3.1 Файлы

Полный список документированных файлов.

<a href="#">calculator.h</a>	Заголовочный файл для модуля calculator, отвечающий за вычисления . . . . .	17
<a href="#">communicator.h</a>	Заголовочный файл для модуля communacator, отвечающий за связь с клиентом	18
<a href="#">interface.h</a>	Заголовочный файл для модуля interface, отвечающий за разбор ПКС, и включение других модулей . . . . .	19
<a href="#">logger.h</a>	Заголовочный файл для модуля logger, отвечающий за запись логов . . . . .	20
<a href="#">programmererror.h</a>	Заголовочный файл для модуля <a href="#">server_error</a> , отвечающий за ошибки . . . . .	22
<a href="#">userbase.h</a>	Заголовочный файл для модуля userbase, отвечающий за получение данных из базы клиентов . . . . .	23



## Глава 4

# Классы

### 4.1 Класс Average

Класс для вычисления среднего арифметического вектора

```
#include <calculator.h>
```

Открытые члены

- [Average](#) ()  
Конструктор класса без параметров
- double [average](#) (std::vector< double > &arr)  
Функция вычисления среднего арифметического вектора

#### 4.1.1 Подробное описание

Класс для вычисления среднего арифметического вектора

Метод для вычисления

#### 4.1.2 Методы

##### 4.1.2.1 average()

```
double Average::average (  
    std::vector< double > & arr )
```

Функция вычисления среднего арифметического вектора

Аргументы

in	vector<double>&	arr вектор со значениями типа double
----	-----------------	--------------------------------------

Возвращает

result

Объявления и описания членов классов находятся в файлах:

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

## 4.2 Класс communicator

Класс для связи с клиентом

```
#include <communicator.h>
```

Открытые члены

- [communicator](#) ()  
Конструктор класса без параметров
- void [conversation](#) (unsigned int port, std::map< std::string, std::string > DataBaseP)  
Функция "разговора" с клиентом.
- std::string [GenSALT](#) ()  
Функция генерации соли
- std::string [GenHash](#) (const std::string &password)  
Генерация хэша из пароля и соли.
- bool [CompareHashes](#) (std::string ClientHash)  
Сравнение хэшей клиента и сервера.
- void getpass (std::string pass)
- void setSALT (const std::string &salt)

### 4.2.1 Подробное описание

Класс для связи с клиентом

Методы: установка бинда, установка сервера в режим ожидания соединения принятие соединения  
получение данных от клиента отправка данных клиенту "разговор" с клиентом

### 4.2.2 Методы

#### 4.2.2.1 CompareHashes()

```
bool communicator::CompareHashes (
    std::string ClientHash )
```

Сравнение хэшей клиента и сервера.

Данный метод сравнивает хэш, полученный от клиента, с хэшем, сгенерированным на сервере. Если хэши не совпадают, то генерируется исключение [server\\_error](#) с сообщением "Invalid Hash". Затем выводится на экран хэш клиента и сервера.\*



## Аргументы

in	ClientHash	Хэш, полученный от клиента.
----	------------	-----------------------------

## Возвращает

true, если хэши совпадают, иначе false.

## Исключения

<code>server_error</code> , если	хэши не совпадают.
----------------------------------	--------------------

## 4.2.2.2 conversation()

```
void communicator::conversation (
    unsigned int port,
    std::map< std::string, std::string > DataBaseP )
```

Функция "разговора" с клиентом.

## Аргументы

in	unsigned	int port,
in	map	<string,string> DataBaseP

В этом методе происходит все взаимодействие с клиентом. Ничего не возвращает.

## 4.2.2.3 GenHash()

```
std::string communicator::GenHash (
    const std::string & password )
```

Генерация хэша из пароля и соли.

Данный метод использует библиотеку Crypto++ для генерации хэша MD5 из пароля и соли. Затем хэш преобразуется в шестнадцатеричную строку.

## Аргументы

in	password	Пароль для хэширования.
----	----------	-------------------------

## Возвращает

Сгенерированный хэш в шестнадцатеричной строке.

#### 4.2.2.4 GenSALT()

```
std::string communicator::GenSALT ( )
```

Функция генерации соли

В этом методе происходит генерация соли.

Возвращает

SALT

Объявления и описания членов классов находятся в файлах:

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

### 4.3 Класс DB

Класс для получения данных из файла базы клиентов

```
#include <userbase.h>
```

Открытые члены

- [DB](#) (std::string DBName)  
Конструктор класса
- std::map< std::string, std::string > [get\\_data](#) ()  
Метод для получения данных из словаря

#### 4.3.1 Подробное описание

Класс для получения данных из файла базы клиентов

Методы: проверка ID

#### 4.3.2 Конструктор(ы)

##### 4.3.2.1 DB()

```
DB::DB (
    std::string DBName )
```

Конструктор класса

## Аргументы

in	DBName	- путь до файла с данными клиентов
----	--------	------------------------------------

В этом конструкторе происходит открытие файла, получение данных, заполнение словаря, закрытие словаря

## 4.3.3 Методы

## 4.3.3.1 get\_data()

```
std::map<std::string, std::string> DB::get_data ( ) [inline]
```

Метод для получения данных из словаря

Возвращает

DataBaseP Словарь с парами идентификатор:пароль

Объявления и описания членов классов находятся в файлах:

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

## 4.4 Класс interface

Класс для разбора командной строки и включения других модулей

```
#include <interface.h>
```

## Открытые члены

- int [Opts](#) (int argc, char \*\*argv)  
Функция разбора ПКС и включение модулей
- std::string [getLogFileName](#) ()  
Функция получения пути до журнала работы

## 4.4.1 Подробное описание

Класс для разбора командной строки и включения других модулей

Методы: разбор ПКС проверка файлов получение пути до файла с логам

## 4.4.2 Методы

### 4.4.2.1 getLogFileName()

```
std::string interface::getLogFileName ( ) [inline]
```

Функция получения пути до журнала работы

В этом методе возвращается путь до файла с журналом работы

Возвращает

LogFileName Возвращает путь к файлу для записи логов

### 4.4.2.2 Opts()

```
int interface::Opts (
    int argc,
    char ** argv )
```

Функция разбора ПКС и включение модулей

Аргументы

in	int	argc,
in	char	**argv

В этом методе происходит разбор ПКС и включение модулей

Возвращает

1 or 0

Объявления и описания членов классов находятся в файлах:

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

## 4.5 Класс Logger

Класс для формирования и записи логов

```
#include <logger.h>
```

## Открытые члены

- `int writelog (std::string s)`  
Функция записи лога
- `void set_path (std::string path_file)`  
Функция получение пути до файла журнала работы
- `Logger ()`  
Конструктор класса без параметров
- `Logger (std::string s)`  
Конструктор класса

## Открытые статические члены

- `static std::string getDateTime ()`  
Функция получения времени

### 4.5.1 Подробное описание

Класс для формирования и записи логов

Методы: запись лога, получение пути до файла журнала работы получение даты и времени

### 4.5.2 Конструктор(ы)

#### 4.5.2.1 Logger()

```
Logger::Logger (  
    std::string s ) [inline]
```

Конструктор класса

Аргументы

in	s	- путь до файла с логами
----	---	--------------------------

### 4.5.3 Методы

#### 4.5.3.1 getDateTime()

```
std::string Logger::getTime ( ) [static]
```

Функция получения времени

В этом методе происходит получение времени и вывод его в нужном формате

Возвращает

`string(buf)`

#### 4.5.3.2 `set_path()`

```
void Logger::set_path (  
    std::string path_file )
```

Функция получение пути до файла журнала работы

В этом методе происходит получение пути до файла журнала работы. Ничего не возвращает.

#### 4.5.3.3 `writelog()`

```
int Logger::writelog (  
    std::string s )
```

Функция записи лога

В этом методе происходит открытие файла, запись, закрытие файла

Возвращает

1 or 0

Объявления и описания членов классов находятся в файлах:

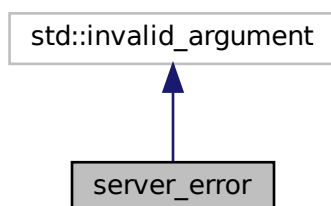
- [logger.h](#)
- [logger.cpp](#)

## 4.6 Класс `server_error`

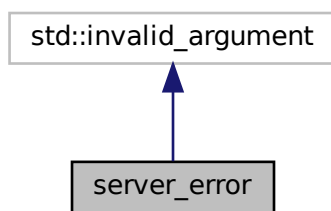
Класс вызова исключений

```
#include <programmererror.h>
```

Граф наследования:`server_error`:



Граф связей класса `server_error`:



## Открытые члены

- `server_error` (`const std::string &what_arg, bool critical=false`)  
Конструктор ошибок с строкой в качестве параметра
- `server_error` (`const char *what_arg, bool critical=false`)  
Конструктор ошибок с си-строкой в качестве параметра
- `std::string getState () const`  
Возвращает статус критичности ошибки

### 4.6.1 Подробное описание

Класс вызова исключений

Класс `server_error`

### 4.6.2 Конструктор(ы)

#### 4.6.2.1 `server_error()` [1/2]

```

server_error::server_error (
    const std::string & what_arg,
    bool critical = false )    [inline], [explicit]
  
```

Конструктор ошибок с строкой в качестве параметра

Аргументы

in	what_arg, тип	ошибки,
in	const	std::string, critical, критическая ошибка - true, штатная - false, bool

#### 4.6.2.2 server\_error() [2/2]

```
server_error::server_error (  
    const char * what_arg,  
    bool critical = false )    [inline], [explicit]
```

Конструктор ошибок с си-строкой в качестве параметра

Аргументы

in	what_arg, тип	ошибки,
in	const	char*, critical, критическая ошибка - true, штатная - false, bool

Объявления и описания членов класса находятся в файле:

- [programmerror.h](#)



## Глава 5

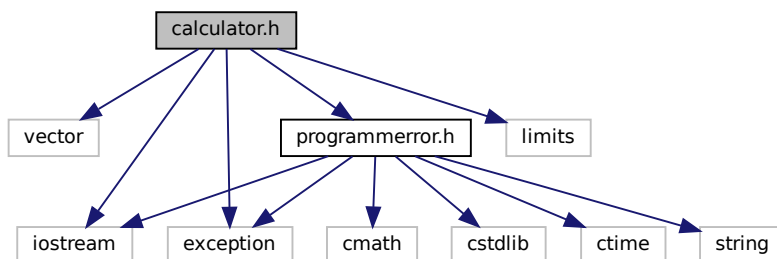
# Файлы

### 5.1 Файл calculator.h

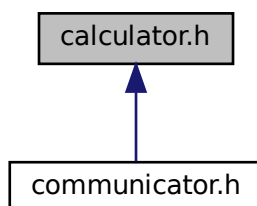
Заголовочный файл для модуля calculator, отвечающий за вычисления

```
#include <vector>
#include <iostream>
#include <exception>
#include <limits>
#include "programmerror.h"
```

Граф включаемых заголовочных файлов для calculator.h:



Граф файлов, в которые включается этот файл:



## Классы

- class [Average](#)

Класс для вычисления среднего арифметического вектора

### 5.1.1 Подробное описание

Заголовочный файл для модуля calculator, отвечающий за вычисления

Автор

Крестина С.Д.

Версия

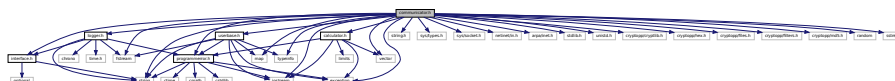
1.0

## 5.2 Файл communicator.h

Заголовочный файл для модуля communicacator, отвечающий за связь с клиентом

```
#include <iostream>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <vector>
#include <string>
#include <map>
#include <cryptopp/cryptlib.h>
#include <cryptopp/hex.h>
#include <cryptopp/files.h>
#include <cryptopp/filters.h>
#include <cryptopp/md5.h>
#include <fstream>
#include <random>
#include <sstream>
#include <exception>
#include <typeinfo>
#include "programmerror.h"
#include "logger.h"
#include "calculator.h"
#include "userbase.h"
#include "interface.h"
```

Граф включаемых заголовочных файлов для communicator.h:



## Классы

- class `communicator`  
Класс для связи с клиентом

## Макросы

- `#define CRYPTOPP_ENABLE_NAMESPACE_WEAK 1`

### 5.2.1 Подробное описание

Заголовочный файл для модуля `communicator`, отвечающий за связь с клиентом

Автор

Крестина С.Д.

Версия

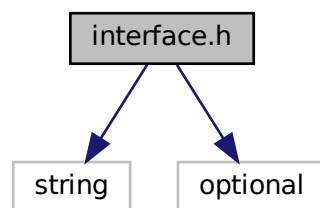
1.0

## 5.3 Файл interface.h

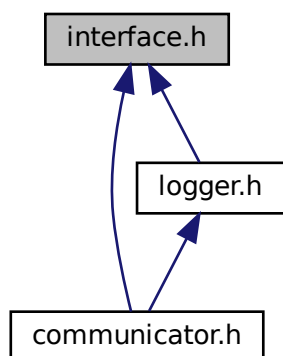
Заголовочный файл для модуля `interface`, отвечающий за разбор ПКС, и включение других модулей

```
#include <string>
#include <optional>
```

Граф включаемых заголовочных файлов для `interface.h`:



Граф файлов, в которые включается этот файл:



## Классы

- class `interface`

Класс для разбора командной строки и включения других модулей

### 5.3.1 Подробное описание

Заголовочный файл для модуля `interface`, отвечающий за разбор ПКС, и включение других модулей

Автор

Крестина С.Д.

Версия

1.0

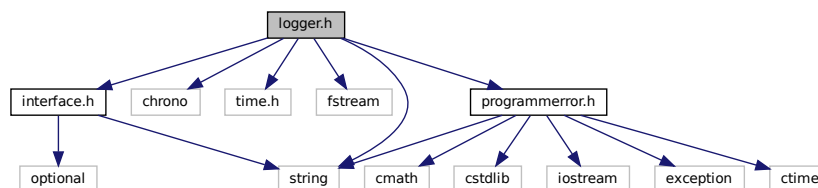
## 5.4 Файл `logger.h`

Заголовочный файл для модуля `logger`, отвечающий за запись логов

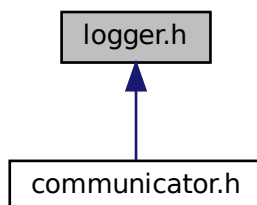
```
#include <string>
#include <chrono>
#include <time.h>
#include <fstream>
#include "interface.h"
```

```
#include "programmerror.h"
```

Граф включаемых заголовочных файлов для logger.h:



Граф файлов, в которые включается этот файл:



## Классы

- class [Logger](#)

Класс для формирования и записи логов

### 5.4.1 Подробное описание

Заголовочный файл для модуля logger, отвечающий за запись логов

Автор

Крестина С.Д.

Версия

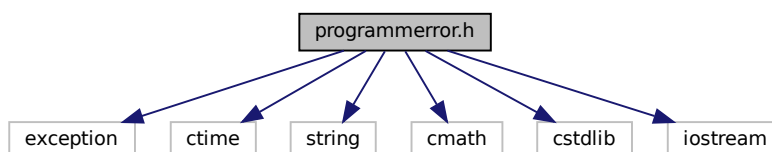
1.0

## 5.5 Файл programerror.h

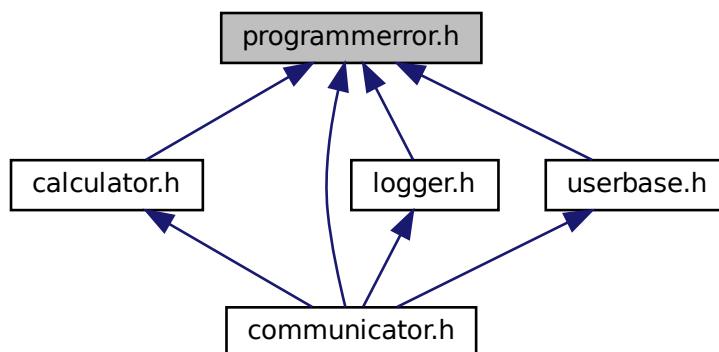
Заголовочный файл для модуля `server_error`, отвечающий за ошибки

```
#include <exception>
#include <ctime>
#include <string>
#include <cmath>
#include <cstdlib>
#include <iostream>
```

Граф включаемых заголовочных файлов для `programerror.h`:



Граф файлов, в которые включается этот файл:



### Классы

- class `server_error`

Класс вызова исключений

### 5.5.1 Подробное описание

Заголовочный файл для модуля `server_error`, отвечающий за ошибки

Автор

Крестина С.Д.

Версия

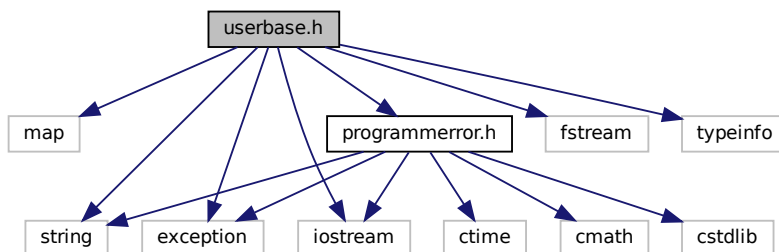
1.0

## 5.6 Файл userbase.h

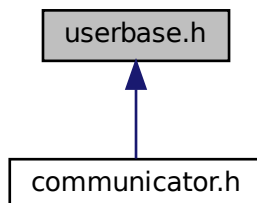
Заголовочный файл для модуля `userbase`, отвечающий за получение данных из базы клиентов

```
#include <map>
#include <string>
#include <fstream>
#include <exception>
#include <typeinfo>
#include <iostream>
#include "programmerror.h"
```

Граф включаемых заголовочных файлов для `userbase.h`:



Граф файлов, в которые включается этот файл:



## Классы

- class [DB](#)

Класс для получения данных из файла базы клиентов

### 5.6.1 Подробное описание

Заголовочный файл для модуля userbase, отвечающий за получение данных из базы клиентов

Автор

Крестина С.Д.

Версия

1.0



# Предметный указатель

- Average, [7](#)
  - average, [7](#)
- average
  - Average, [7](#)
- calculator.h, [17](#)
- communicator, [8](#)
  - CompareHashes, [8](#)
  - conversation, [9](#)
  - GenHash, [9](#)
  - GenSALT, [9](#)
- communicator.h, [18](#)
- CompareHashes
  - communicator, [8](#)
- conversation
  - communicator, [9](#)
- DB, [10](#)
  - DB, [10](#)
  - get\_data, [11](#)
- GenHash
  - communicator, [9](#)
- GenSALT
  - communicator, [9](#)
- get\_data
  - DB, [11](#)
- getDateTime
  - Logger, [13](#)
- getLogFileName
  - interface, [12](#)
- interface, [11](#)
  - getLogFileName, [12](#)
  - Opts, [12](#)
- interface.h, [19](#)
- Logger, [12](#)
  - getDateTime, [13](#)
  - Logger, [13](#)
  - set\_path, [14](#)
  - writelog, [14](#)
- logger.h, [20](#)
- Opts
  - interface, [12](#)
- programmerror.h, [22](#)
- server\_error, [14](#)
  - server\_error, [15](#), [16](#)
- set\_path
  - Logger, [14](#)
- userbase.h, [23](#)
- writelog
  - Logger, [14](#)