

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Информационная безопасность систем и технологий»

Курсовая работа
по дисциплине «Методы и средства программирования»
на тему «Программная реализация сетевого сервера»

ПГУ.100503.С1.О.26.КР.22ПИ208.01.ПЗ

Специальность — 10.05.03 Информационная безопасность
автоматизированных систем

Специализация — Разработка автоматизированных систем в защищённом
исполнении

Дисциплина: Методы и средства
программирования

Группа: 22ПИ2

Выполнил: Никитина М. А.

Количество баллов:

Дата сдачи:

Принял: к. т. н Н. А. Сидоров

2023

УТВЕРЖДАЮ
Зав. кафедрой ИБСТ

к.т.н., доцент

Зефилов С.Л.

20 09 2023г.

ЗАДАНИЕ

на курсовую работу

по теме: Программная реализация сетевого сервера

1 Дисциплина: Методы и методы программирования

2 Студент: Никитина Маргарита Андреевна

3 Группа: 22ПИ2

4 Исходные данные на работу:

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

4.2 Требования к программе:

4.2.1 Требования к выполняемым функциям

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

4.2.2 Требования к базе клиентов

- хранение базы клиентов в файле текстово-двоичного 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 на сервере;
 - передача числовых векторов на сервер;

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

- поле «размер вектора», 4 байта, число типа `uint32_t`;
- поле «значение вектора», последовательность полей по 8 байт, числа типа `int64_t`, значения вектора;
- форма передаваемых данных — двоичный:
 - поле «результат вычислений по вектору» 8 байт, число типа `int64_t`;
- функция обработки — среднее арифметическое элементов вектора;
 - при переполнении вверх возвращать значение $2^{63} - 1$;
 - при переполнении вниз возвращать значение -2^{63} ;

4.2.6 Требование к пользовательскому интерфейсу и обработке ошибок в процессе выполнения программы

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

4.2.7 Требования к среде функционирования программы

- программа должна функционировать в операционной системе с ядром Linux

5 Структуры 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 | «11» сентября 2023 г. |
| - разработка диаграмм вариантов использования и диаграмм классов | «25» сентября 2023 г. |
| - разработка диаграмм последовательностей и диаграмм деятельности | «9» октября 2023 г. |
| - разработка кода программы | «30» октября 2023 г. |
| - разработка модульных тестов и выполнение тестирования | «13» ноября 2023 г. |
| - разработка функциональных тестов и приемочное тестирование | «27» ноября 2023 г. |
| - разработка документации | «11» декабря 2023 г. |
| - оформление отчета о курсовой работе | «18» декабря 2023 г. |
| - защита курсовой работы | «25» декабря 2023 г. |

Руководитель работы

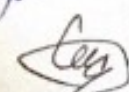
Задание получил «11» 09 2023г.

Студент

Нормоконтролер

 Н. А. Сидоров

 М. А. Никитина

 Н. А. Сидоров

Оглавление

1 Введение.....	7
2 Проектирование.....	9
3 2 Разработка программы.....	11
4 3 Модульное тестирование.....	14
5 4 Функциональное тестирование.....	18
6 5 Документирование.....	20
7 Заключение.....	21
8 Список используемых источников.....	22
9 Приложение А.....	23
10 (обязательное).....	23

Введение

В современном мире сетевые серверы играют ключевую роль в обеспечении эффективной и безопасной передачи информации. Программная реализация сетевого сервера является одной из важных задач в области разработки программного обеспечения. Эта тема является актуальной и интересной, так как сеть является базовой инфраструктурой для многих приложений и сервисов, используемых в настоящее время.

Актуальность, поставленной в работе проблемы, позволяет определить объект, предмет, цель и задачи исследования.

Цель работы: разработка серверной программы для клиент-серверной системы обработки данных на языке программирования C++.

Задачи работы:

- проанализировать требования к программе;
- построить UML-диаграммы вариантов использования и проектирование пользовательского интерфейса;
- построить UML-диаграммы классов и планирование модулей;
- построить UML-диаграммы последовательностей;
- построить UML-диаграммы деятельности;
- разработать программу;
- разработать модульные тесты и провести модульное тестирование;
- разработать функциональные тесты и провести приемочное тестирование;
- провести документирование кода программы с использованием Doxygen.

Объект исследования — разработка сетевых приложений на языке C++.

Предмет исследования — серверная программа.

Практическая значимость состоит в получении практических навыков разработки сетевых приложений на языке C++.

Курсовая работа изложена на 30 страницах печатного текста состоит из введения, 5 разделов, заключения, списка используемых источников, и одного приложения. Во введении обоснована актуальность темы, сформулированы цели, задачи исследования. В первом разделе проведено проектирование программы, во втором разделе проведена разработка кода программы, в третьем разделе проведено модульное тестирование, в четвёртом разделе проведено функциональное тестирование программы, в пятом разделе написана документация программы. В заключении подведён итог результатов исследования и сделаны выводы по проделанной работе.

1 Проектирование

Проектирование представляет собой процесс определения структуры компонентов, интерфейсов и других характеристик системы или её части, что позволяет задать архитектуру будущей программы и упростить разработку.

В данном разделе проведено проектирование программы, включая разработку UML-диаграмм вариантов использования, классов, деятельности и последовательности.

Диаграмма вариантов использования в UML — графическое представление, отражающее взаимодействие между акторами (внешними сущностями, взаимодействующими с системой) и прецедентами (специфичными функциональными элементами системы).

На диаграмме вариантов использования приведены возможные взаимодействия сервера как актора.

Диаграмма классов в UML представляет собой структурную диаграмму, которая демонстрирует общую структуру иерархии классов в системе, их взаимосвязи, атрибуты, методы, интерфейсы и другие элементы. Она помогает в визуализации структуры объектно-ориентированной системы.

На диаграмме классов классы представляются в виде прямоугольников с тремя отдельными секциями: названием класса, его атрибутами и методами. Связи между классами обозначают отношения и зависимости. Эти диаграммы полезны для понимания структуры программы, а также в процессе проектирования и визуализации отношений между различными компонентами системы.

Диаграмма деятельности представляет собой графическое изображение действий и процессов в системе. Она используется для моделирования бизнес-процессов, алгоритмов и логики работы программы.

На диаграмме деятельности действия представлены в виде прямоугольников, а переходы между ними обозначают стрелки, указывающие направления выполнения.

Эта диаграмма позволяет визуализировать последовательность действий, в том числе параллельные и условные, что облегчает понимание логики работы программы или процесса.

UML-диаграмм последовательности — изображение взаимодействия между объектами в определенном промежутке времени. Она используется для моделирования последовательности вызовов методов или передачи сообщений между объектами в рамках некоторого прецедента или сценария.

На диаграмме последовательности объекты представлены вертикальными линиями, а вызовы методов или передача сообщений изображается стрелками, идущими между объектами в порядке, определенном временной шкалой. Такие диаграммы помогают визуализировать взаимодействие между различными элементами системы во времени и легко понять порядок выполнения операций.

Диаграммы сделаны при помощи сайта Aspose и приложения Umbrello и приведены в приложении А.

2 Разработка программы

В данном разделе 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 течение всего времени функционирования;
- в) обработка запросов клиентов в однопоточном режиме, в том числе:
 - 1) идентификацию и аутентификацию подключаемого клиента;
 - 2) выполнение вычислений над данными, передаваемыми клиентом;
 - 3) операция, выполняемая над данными — среднее арифметическое элементов вектора.

В ходе работы были разработаны следующие модули:

- main.cpp;
- server.cpp;
- server.h;
- auth.cpp;
- auth.h;
- errors.cpp;
- errors.h.

Модуль main.cpp — основной файл, который запускает сервер с параметрами командной строки. Он принимает порт, файл базы данных и файл журнала в качестве аргументов.

Интерфейс обеспечивает передачу программе следующей информации:

- имя файла с базой клиентов, необязательный;
 - значение по умолчанию /etc/vcalc.conf;

- имя файла с журналом работы, необязательный;
 - значение по умолчанию /var/log/vcalc.log;
- номер порта, на котором будет работать сервер, необязательный;
 - значение по умолчанию 33333;
- программа предусматривает выдачу справки о пользовательском интерфейсе
 - при запуске с параметром -h;
 - при запуске без параметров.

Модули server.cpp и server.h содержат определение и реализацию класса “Server”. Класс “Server” отвечает за обработку взаимодействия с клиентом, аутентификацию, передачу хэшей и обработку векторов.

- 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.

Модули auth.cpp и auth.h содержат код для аутентификации клиентов. Функция “authenticateClient” проверяет правильность имени пользователя и пароля, сравнивая их с данными из файла базы данных.

Модули errors.cpp и errors.h содержат определение и реализацию класса “ErrorTracker”. Класс “ErrorTracker” отвечает за отслеживание и логирование

ошибок. Когда происходит ошибка (например, ошибка аутентификации), класс “ErrorTracker” используется для записи информации об ошибке в файл логов.

База данных имеет вид «идентификатор пароль».

Подробности реализации приведены в документации, которая будет рассмотрена в разделе 5.

Код модулей расположен в репозитории на github. Ссылка на репозиторий: https://github.com/Popitka994/course_work

3 Модульное тестирование

В данном разделе проведено тестирование модулей, разработанных в предыдущем разделе.

Модуль `main.cpp` является точкой входа. Он имеет функцию «`main`», которая запускается при старте программы. В этом модуле устанавливаются значения параметров по умолчанию, обрабатываются параметры командной строки, запускается сервер, выводится справка.

Тестовый сценарий для модуля `main.cpp` представлен в таблице 1. Тесты «запуск с 1 из параметров», «запуск с 2-мя параметрами», «запуск с 3-мя параметрами» и «задан путь к файлу с базой данных, который уже существует» будут рассмотрены в разделе «Функциональное тестирование».

Таблица 1 - Тестовый сценарий `main.cpp`

Название теста	Вводимые параметры	Ожидаемый результат
Запуск без параметров		Искл.
Запуск с параметром <code>-h</code>	<code>-h</code>	
Запуск с неизвестным параметром	<code>-f</code>	
Запуск с пустым параметром	<code>--port</code>	
Запуск с 1 из параметров	<code>--port 33333</code>	Следующий шаг
Запуск с 2-мя параметрами	<code>--port 33333 --database usersdata.txt</code>	
Запуск с 3-мя параметрами	<code>--port 33333 --database usersdata.txt --log log.txt</code>	
Задан путь к файлу с базой данных, который уже существует	<code>usersdata.txt</code>	
Задан путь к файлу с базой данных, которого не существует	<code>database.txt</code>	Искл.

Результат реализации тестового сценария для `main.cpp` предоставлен на рисунке 1.


```

stud@virtdeb:~/C++Projects/CourseWorkTests$ ./my_program
Тест с отсутствием аргументов:
Использование: program_name [--port <порт>] [--database <файл_базы_данных>] [--log <файл_журнала>]
Параметры:
  --port <порт>: Порт сервера (по умолчанию: 33333)
  --database <файл_базы_данных>: Путь к файлу базы данных (по умолчанию: usersdata.txt)
  --log <файл_журнала>: Путь к файлу журнала работы (по умолчанию: log.txt)
Тест с аргументом -h:
Использование: program_name [--port <порт>] [--database <файл_базы_данных>] [--log <файл_журнала>]
Параметры:
  --port <порт>: Порт сервера (по умолчанию: 33333)
  --database <файл_базы_данных>: Путь к файлу базы данных (по умолчанию: usersdata.txt)
  --log <файл_журнала>: Путь к файлу журнала работы (по умолчанию: log.txt)
Тест с неизвестным аргументом:
Неизвестный параметр: --unknown
Тест с передачей несуществующей базы данных:
Не удалось открыть файл базы данных.
Тест с передачей пустого параметра:
Отсутствует значение для параметра: --database
Success: 5 tests passed.
Test time: 0.00 seconds.

```

Рисунок 1 - Результат работы tests_main.cpp

Модуль server.cpp предоставляет собой реализацию функциональности сервера. Функции, которые выполняет этот модуль:

- инициализация сервера, включая обработку параметров;
- взаимодействие с пользователем для ввода имени пользователя и пароля;
- аутентификация с использованием базы данных (открытие файла базы данных и сравнение введенных пользователем данных с данными в файле);
- генерация salt;
- хэширование пароля с использованием salt и алгоритма MD5;
- вычисление среднего арифметического.

Тестовый сценарий для модуля server.cpp представлен в таблице 2.

Таблица 2 - Тестовый сценарий для server.cpp

Название теста	Вводимые данные	Ожидаемый результат
Верный ID	Nikitina Margosha994	Следующий шаг
Неверный ID	Hi Hacker	Искл.

Продолжение таблицы 2

Название теста	Вводимые данные	Ожидаемый результат
Обычный вектор	10 20 30 40 50	30
Нецелый результат	6 7	6.5
Отрицательные числа	-10 -20 -30 -40 -50	-30
Положительные и отрицательные числа	10 -20 30 -40	-5
Максимальные положительные значения	32767 32767 32767	32767
Максимальные отрицательные значения	-32768 -32768 -32768	-32768
Пустой вектор		Искл.

Результат реализации тестового сценария для server.cpp предоставлен на рисунке 2.

```

Тест с верной аутентификацией:
Имя и пароль совпадает с базой данных.
> [C] course_work

Тест с неверной аутентификацией:
Имя и/или пароль не совпадает с базой данных.

Передача нормального вектора:
Элементы: 10 20 30 40 50
Average: 30

Нецелый результат вычислений:
Элементы: 6 7
Average: 6.5

Отрицательные элементы вектора:
Элементы: -10 -20 -30 -40 -50
Average: -30

Отрицательные и положительные элементы вектора:
Элементы: 10 -20 30 -40
Average: -5

Максимальные положительные элементы вектора:
Элементы: 32767 32767 32767
Average: 32767

Максимальные отрицательные элементы вектора:
Элементы: -32768 -32768 -32768
Average: -32768

Пустой вектор:
Вектор пуст, возвращаем 0.
Success: 14 tests passed.
Test time: 0.00 seconds.

```

Рисунок 2 - Результат работы tests_server.cpp

Код проекта, созданного для модульного тестирования, представлен в репозитории на github.

4 Функциональное тестирование

В этом разделе было проведено комплексное тестирование программы в общем и ее модулей, которые не подлежат модульному тестированию.

Ручное тестирование, также известное как функциональное тестирование, представляет собой метод проверки программного обеспечения, при котором тестовые сценарии выполняются вручную тестировщиком, без использования автоматизированных средств. Основной целью ручного тестирования является выявление ошибок, проблем и дефектов в программном приложении.

Тестовый сценарий для функционального тестирования представлен в таблице 3.

Таблица 3 - Функциональное тестирование программы

Тест	Ожидаемый результат
Запуск с 1 из параметров	Аутентификация пользователя и передача ему вычислений по векторам
Запуск с 2-мя параметрами	
Запуск с 3-мя параметрами	
Задан путь к файлу с базой данных, который уже существует	
Проверка файла с записями об ошибках	Все исключения были записаны верно
Генерация различной соли и хэша	Программа генерирует разную соль для каждого пользователя, соответственно, хэш тоже разный

Был проведён тест запуска программы с одним, двумя и тремя параметрами. Результат тестирования представлен на рисунке 3.

Тест с заданным путём к файлу с базой данных, который уже существует так же представлен на рисунке 3.

Была проведена проверка файла с записями об ошибках. Результат предоставлен на рисунке 4. Файл с записями об ошибках log.txt представлен в репозитории на github.

```

stud@virtdeb:~/C++Projects/Lb1$ ./my_program --port 33333
Введите Ваше имя пользователя: ^Z
[7]+ Остановлен ./my_program --port 33333
stud@virtdeb:~/C++Projects/Lb1$ ./my_program --port 33333 --database usersdata.txt
Введите Ваше имя пользователя: ^Z
[8]+ Остановлен ./my_program --port 33333 --database usersdata.txt
stud@virtdeb:~/C++Projects/Lb1$ ./my_program --port 33333 --database usersdata.txt --log log.txt
Введите Ваше имя пользователя: ^Z

```

Рисунок 3 - Тест с параметрами

```

log.txt x
[2024-01-15 22:49:52] INFO: Сервер запущен.
[2024-01-15 22:50:00] INFO: Успешная аутентификация.
[2024-01-15 22:50:37] INFO: Сервер запущен.
[2024-01-15 22:50:42] ERROR: Ошибка аутентификации. Закрываем соединение.
[2024-01-16 23:32:42] INFO: Сервер запущен.
[2024-01-16 23:32:51] INFO: Успешная аутентификация.
[2024-01-16 23:40:19] INFO: Сервер запущен.
[2024-01-16 23:40:29] INFO: Успешная аутентификация.
[2024-01-16 23:42:57] INFO: Сервер запущен.
[2024-01-16 23:43:04] INFO: Успешная аутентификация.
[2024-01-18 16:17:13] INFO: Сервер запущен.
[2024-01-18 16:17:23] ERROR: Ошибка аутентификации. Закрываем соединение.
[2024-01-18 17:09:29] INFO: Сервер запущен.

```

Рисунок 4 - Файл log.txt

Был проведён тест на генерацию различной соли и хэша. В базе данных содержится два пользователя с одинаковым паролем. Тест визуально доказывает, что значение соли зависит от времени. Если соль каждый раз разная, то даже у пользователей с одинаковым паролем, хэш будет разным. Результат тестирования представлен на рисунке 5.

```

stud@virtdeb:~/C++Projects/Lb1$ ./my_program --port 33333
Введите Ваше имя пользователя: Kitty
Введите Ваш пароль: Hello1
OK
Аутентификация успешна. Передаем HASH.
SALT: 9B53EA762E854679
HASH: a720577588d1e0bd6c6003fe4582b1fd
Введите количество векторов: ^Z
[10]+ Остановлен ./my_program --port 33333
stud@virtdeb:~/C++Projects/Lb1$ ./my_program --port 33333
Введите Ваше имя пользователя: Hi
Введите Ваш пароль: Hello1
OK
Аутентификация успешна. Передаем HASH.
SALT: D05A3CE2D03F9F9C
HASH: 2312e3e22ed802d5e78eea60e27638ab

```

Рисунок 5 - Различность соли и хэша

Результаты тестов соответствуют ожиданиям.

5 Документирование

При формировании программы необходимо предоставить не только код, но и текстовую документацию, описывающую различные аспекты функционирования кода. Эта документация часто встраивается прямо в исходный код программы или предоставляется в комплекте с ним.

В данном разделе была создана документация программы при помощи `doxygen`.

Был создан `Doxyfile` с помощью команды `doxygen -g`. После были сделаны документирующие блоки по всему программному коду проекта. С помощью команды `doxygen` была создана сама документация.

Полная документация расположена в репозитории на `github`.

Заключение

В ходе работы над курсовым проектом была осуществлена программная реализация сетевого сервера. Было построено 4 диаграммы и реализовано 7 модулей.

В первом разделе проведено проектирование программы.

Во втором разделе проведена разработка кода программы.

В третьем разделе проведено модульное тестирование.

В четвёртом разделе проведено функциональное тестирование программы

В пятом разделе написана документация программы.

В заключении подведён итог результатов исследования и сделаны выводы по проделанной работе.

Список используемых источников

- 1 Прата, С. Язык программирования C++. Лекции и упражнения / С. Прата. - СПб.: Питер, 2017. - 560 с.
- 2 Дьюхерст, С. Многозадачное программирование на C++ / С. Дьюхерст. - СПб.: БХВ-Петербург, 2018. - 416 с.
- 3 Липпман, С. Б., Лажойе, Дж., Му, Б. Э., Исак, Дж. Г. Язык программирования C++. Базовый курс / С. Б. Липпман, Дж. Лажойе, Б. Э. Му, Дж. Г. Исак. - М.: Вильямс, 2019. - 976 с.

Приложение А

(обязательное)

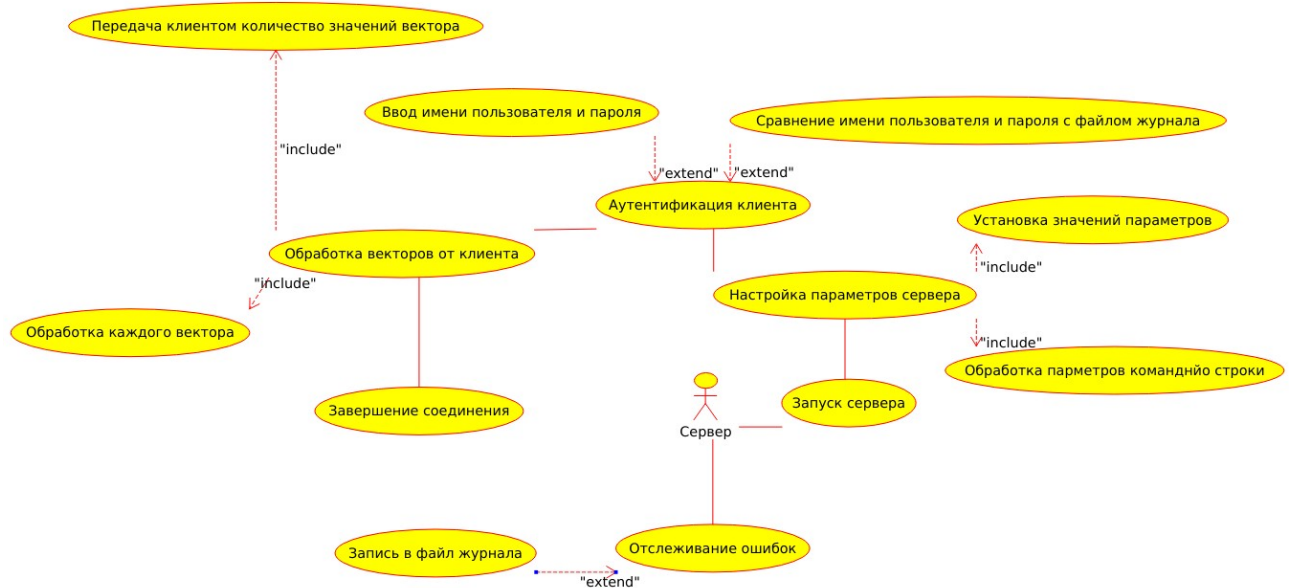


Рисунок А.1 — UML-диаграмма вариантов использования

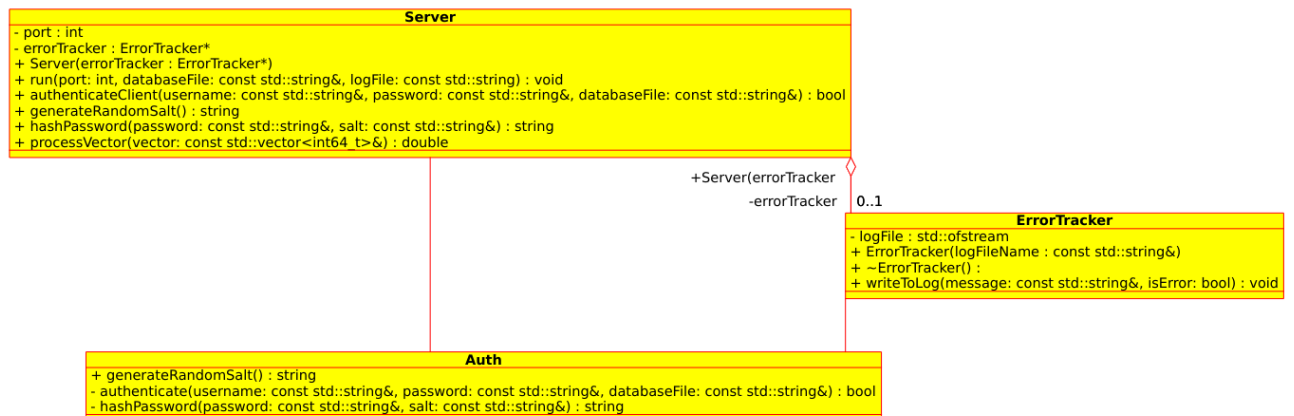


Рисунок А.2 — UML-диаграмма классов

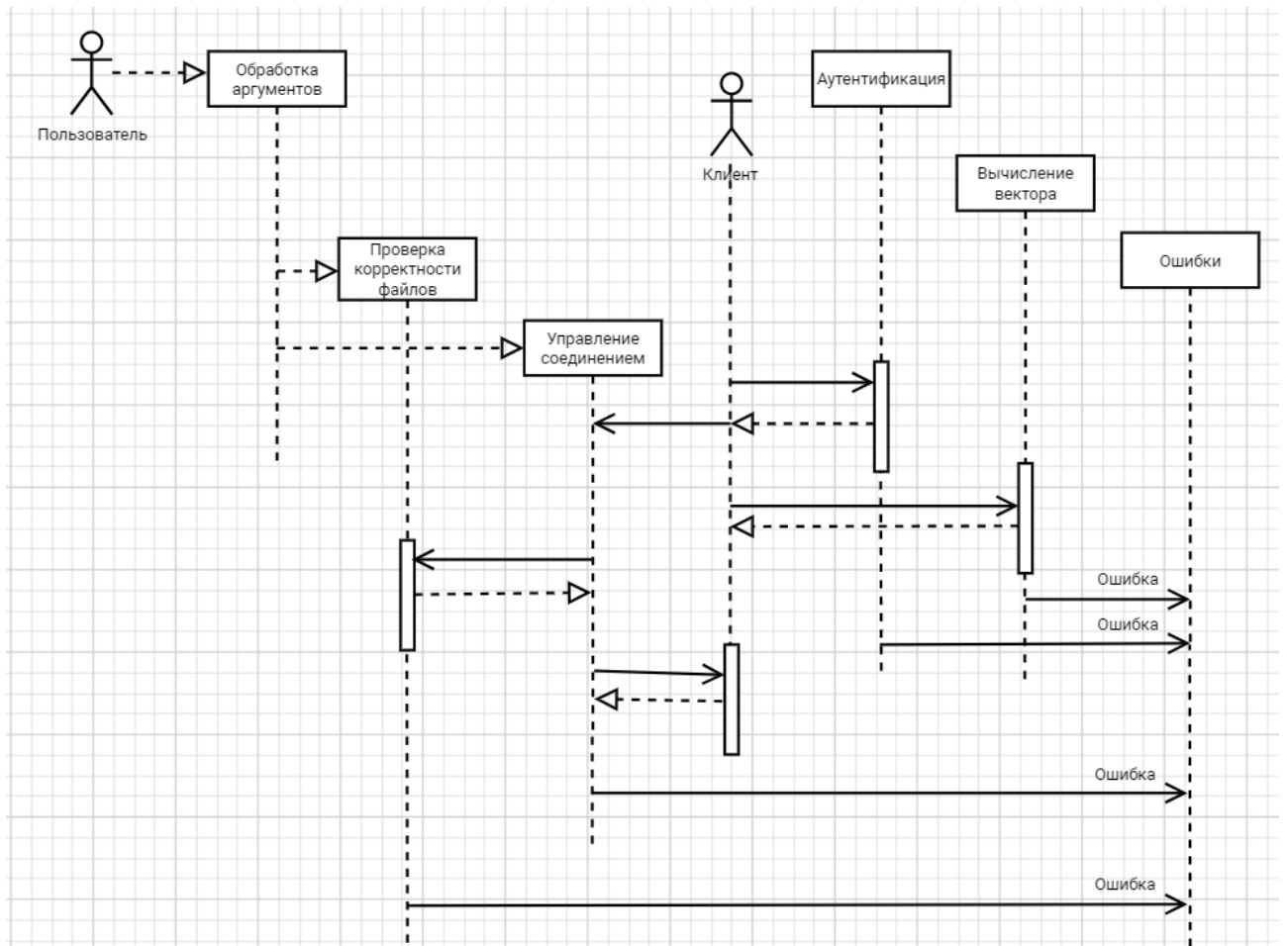


Рисунок А.3 — UML-диаграмма последовательности

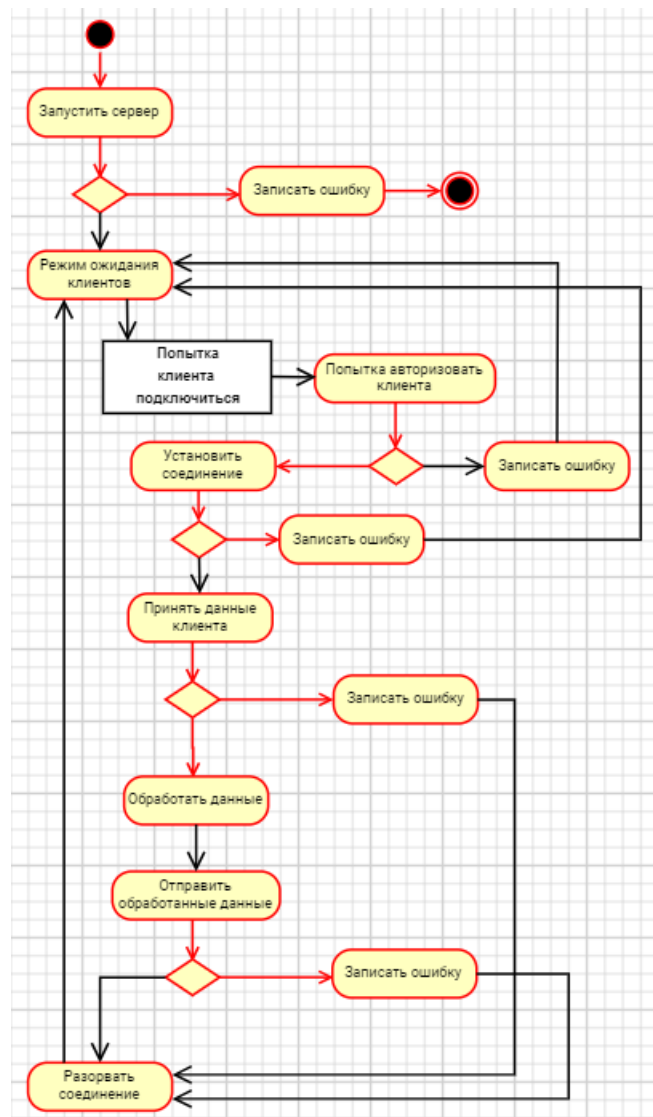


Рисунок А.4 — UML-диаграмма деятельности