

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

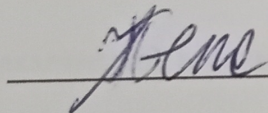
Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПОТОКОВЫЙ КЛИЕНТ-СЕРВЕРНЫЙ
АНАЛИЗАТОР ТЕКСТОВОЙ
ИНФОРМАЦИИ

Студент

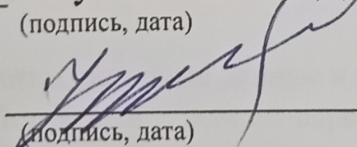


Петухов Максим Дмитриевич

(подпись, дата)

(ФИО полностью)

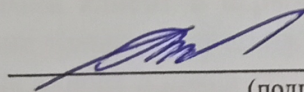
Руководитель



Слепцов Н. В.

(фамилия, инициалы)

Нормоконтролёр



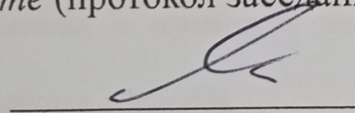
А.С. Бычков

(подпись, дата)

(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от . 2023 г. №)

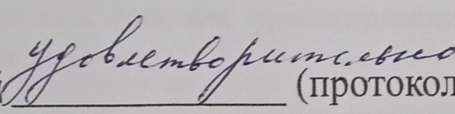
Заведующий кафедрой



М.А. Митрохин

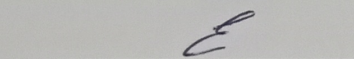
(подпись)

(фамилия, инициалы)

Работа защищена с отметкой  (протокол заседания ГЭК от

27.06.2023 № 14)

Секретарь ГЭК



Е.И. Гурин

(подпись)

(фамилия, инициалы)

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

«26» 04 2023г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Петухов Максим Дмитриевич. 19ВВВ1 факультета вычислительной техники.

2. Тема работы «Потоковый клиент-серверный анализатор текстовой информации»

Тема утверждена приказом ПГУ № 458/о от «26» апреля 2023 г. 2023 г.

3. Руководитель бакалаврской работы к.т.н., доцент Слепцов Н. В.

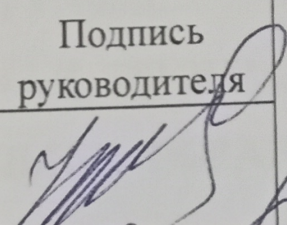
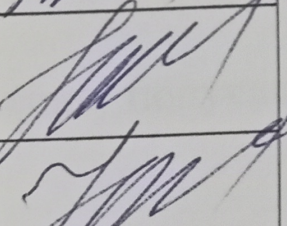
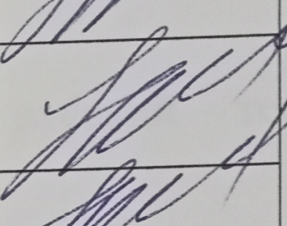
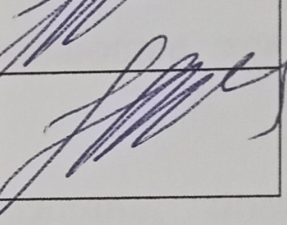
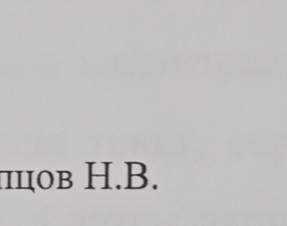
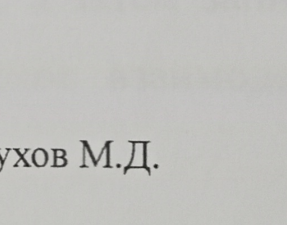
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение «Потоковый клиент-серверный анализатор текстовой информации». 1) Программа клиент в первом потоке должна получать от пользователя текст и проверять, что он состоит только из русских символов и количество слов не превышает 32. 2) Во втором потоке программа клиент должна производить транслитерацию, разбиение текста на слова, подсчет количества вхождений полученных слов и отправку результата на сервер. 3) Программа сервер должна принимать данные с клиента и подсчитывать количество символов, а затем записывать результат в файл.

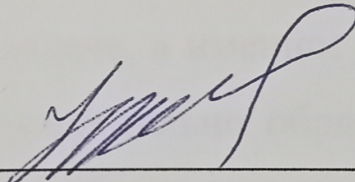
5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить аналоги существующих приложений для анализа текста, изучить способы проектирования и разработки собственного клиент-серверного приложения на языке C++.

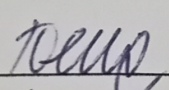
6. Календарный график выполнения работы

№п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	50	26.04.2023-30.04.2023	
2.	Анализ требований	10	01.05.2023-02.05.2023	
3.	Выбор технологий, применяемых при проектировании и разработке	50	03.05.2023-07.05.2023	
4.	Описание алгоритмов	40	08.05.2023-11.05.2023	
5.	Разработка клиент-серверного анализатора текстовой информации	150	12.05.2023-29.05.2023	
6.	Оформление пояснительной записки	24	30.05.2023-31.05.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы  Слепцов Н.В.

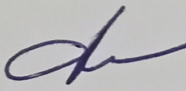
(подпись, дата)

Задание к исполнению принял студент  Петухов М.Д.

(подпись, дата)

Работу к защите допустить

Декан факультета



Фионова Л.Р.

(подпись, дата)

27.06.2023

Реферат

Пояснительная записка содержит 79 л., 32 рис., 5 ист. и 3 прил.

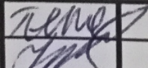
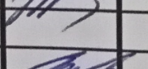
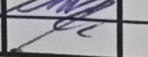
КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, МНОГОПОТОЧНАЯ ПРОГРАММА,
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ, АНАЛИЗ ПОСОМВОЛЬНО, ОБРАБОТКА
ТЕКСТА, ТРАНСЛИТЕРАЦИЯ ТЕКСТА

Цель работы – разработать клиент-серверное приложение, получающее и обрабатывающее текстовую информацию.

Объектами разработки являются:

- клиентское приложение, транслитерирующее русский текст и отправляющее результат на сервер;
- серверное приложение, принимающее и обрабатывающее текст для последующей записи в файл.

В результате проектирования: разработано приложение, полностью соответствующее поставленной задаче, а именно: разработана клиентская часть приложения, получающая и предварительно обрабатывающая текст, серверная часть, которая принимает и обрабатывает текст с клиента, а затем записывает результат обработки в файл, реализовано клиент-серверное взаимодействие программ.

					ПГУ1.090301.16.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Петухов М.Д.					
Провер.		Слепцов Н.В.					
Реценз.							
Н. Контр.		Бычков А.С.					
Утверд.		Митрохин М.А.					
					Потоковый клиент-серверный анализатор текстовой информации Пояснительная записка		
					Лит.	Лист	Листов
						2	79
					ФВТ, каф. ВТ гр. 19ВВВ1		

Содержание

Обозначения и сокращения	5
Введение	6
1. Анализ предметной области и технического задания	7
1.1 Анализ предметной области	7
1.2 Анализ аналогов	7
1.2.1 Сервис <i>text.ru</i>	7
1.2.2 Сервис <i>advego.com</i>	8
1.2.3 Сервис Главред.....	9
1.3 Постановка задачи	10
1.4 Требования к среде разработки и исполнения программы	11
1.5 Функциональные требования к ПО.....	11
1.6 Выбор инструментов разработки	12
1.6.1 Выбор языка программирования.....	12
1.6.2 Выбор среды разработки для проекта	14
1.7 Реализация программы	20
1.7.1 Клиентская часть.....	20
1.7.2 Серверная часть.....	25
1.8 Описание системы и программы с помощью диаграммы развертывания ...	29
1.9 Отладка программы	30
1.10 Описание работы программы	36
2 Охрана труда.....	41
Заключение	43
Список использованных источников	44

Приложение А	45
Приложение Б	64
Приложение Б.1 – Диаграмма вариантов использования	65
Приложение Б.2 – Диаграмма развертывания	66
Приложение Б.3 – Диаграмма последовательности	67
Приложение В.....	68

Обозначения и сокращения

В настоящей работе применяют следующие сокращения и обозначения

ПК – Персональный Компьютер

ЭВМ – Электронная Вычислительная Машина

API (Application Programming Interface) – описание способов взаимодействия одной компьютерной программы с другими

GUI (Graphical User Interface) – графический интерфейс пользователя

IDE (Integrated Development Environment) – интегрированная среда разработки.

SEO (Search Engine Optimization) – поисковая оптимизация

STD (Standard Template Library) – стандартная библиотека шаблонов

TCP (Transmission Control Protocol) – протокол передачи данных

					ПГУ1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		5

Введение

Современный мир становится всё более зависимым от технологий, и для человека основной способ коммуникации с компьютером – это текст. В связи с этим, появилось необходимость в создании программ, которые могли бы получать текст и обрабатывать его.

Компьютеры не понимают естественный язык, поэтому для взаимодействия компьютера и человека нужен посредник, программа, которая переведет для компьютера текст в понятную ему форму.

Программа, которая преобразует текст, звук, изображение, видео в понятный для компьютера вид, называется программой анализатором.

Программы анализаторы могут разрабатываться как в чистом виде, так и как часть большей программы, например для поисковиков, нейросетей, *IDE*.

Целью дипломного проекта является разработка текстового анализатора, на базе клиент серверного приложения.

Дипломный проект состоит из двух разделов: в первом разделе проведен анализ предметной области, выбран стек технологий для реализации проекта, описано проектирование и реализация программного приложения.

					ПГУ1.090301.16.001 ПЗ	Лист
						6
Изм.	Лист	№ докум	Подпись	Дата		

1. Анализ предметной области и технического задания

1.1 Анализ предметной области

Программа анализатор текста обычно является частью систем аналитики текстовой и иной информации, так, как только обработка и преобразование текста без последующего использования в последующей аналитике мало где может пригодиться. Программа анализатор текста обеспечивает преобразование текста в более удобный или понятный формат для дальнейшей обработки, какими либо программами.

1.2 Анализ аналогов

В рамках ВКР был проведен анализ различных анализаторов текстовой информации, для выявления их преимуществ и недостатков.

1.2.1 Сервис *text.ru* [1].

Графический интерфейс сервиса представлен на рисунке 1.

The screenshot displays the text.ru service interface with the following components:

- Header:** Includes a '+ Новый текст' button and a timestamp 'Время проверки уникальности: 16.06.2023 19:16 (UTC +03:00)'.
- Checkboxes:** Three colored boxes for 'Проверка уникальности' (red), 'Проверка орфографии' (green), and 'SEO-анализ текста' (orange).
- Uniqueness Check Results:** Shows 'Уникальность: 51.78%' and a comparison of two URLs with 37% and 30% similarity. A 'Подробнее' button is present.
- Spelling Check Results:** States 'В данном тексте ошибки не найдены.' with a 'Подробнее' button.
- SEO Analysis Results:** Lists statistics: 'Всего символов: 559', 'Заспамленность: 37%', 'Без пробелов: 488', 'Вода: 17%', and 'Количество слов: 71'. A 'Подробнее' button is present.
- Main Text Area:** Contains a paragraph about organic products. A legend indicates 'Подсвечено: Неуникальные фрагменты'.
- Improvement Suggestion:** A box suggests increasing uniqueness on the 'Бирже рерайтинга' with a 'Повысить уникальность' button.
- Text History:** A table titled 'Версии текста:' showing two versions: '7 минут назад' and '42 минуты назад', each with its own set of analysis statistics.
- Footer:** Includes 'Текст сохранен' and a 'Проверить уникальность' button.

Рисунок 1 – Результат обработки текста

Сервис проверяет уникальность, орфографию, *SEO* введённого текста. Также на сайте имеется возможность заказать или выполнить заказ на написание статьи.

На сайте присутствует история запросов, что позволяет просмотреть уже когда-то обработанный текст без необходимости ожидать его повторную обработку.

Так же присутствует подсветка слов по выбранной категории, что позволяет быстро обнаружить и исправить те или иные проблемы текста.

1.2.2 Сервис *advego.com* [2].

Графический интерфейс сервиса представлен на рисунках 2 и 3.

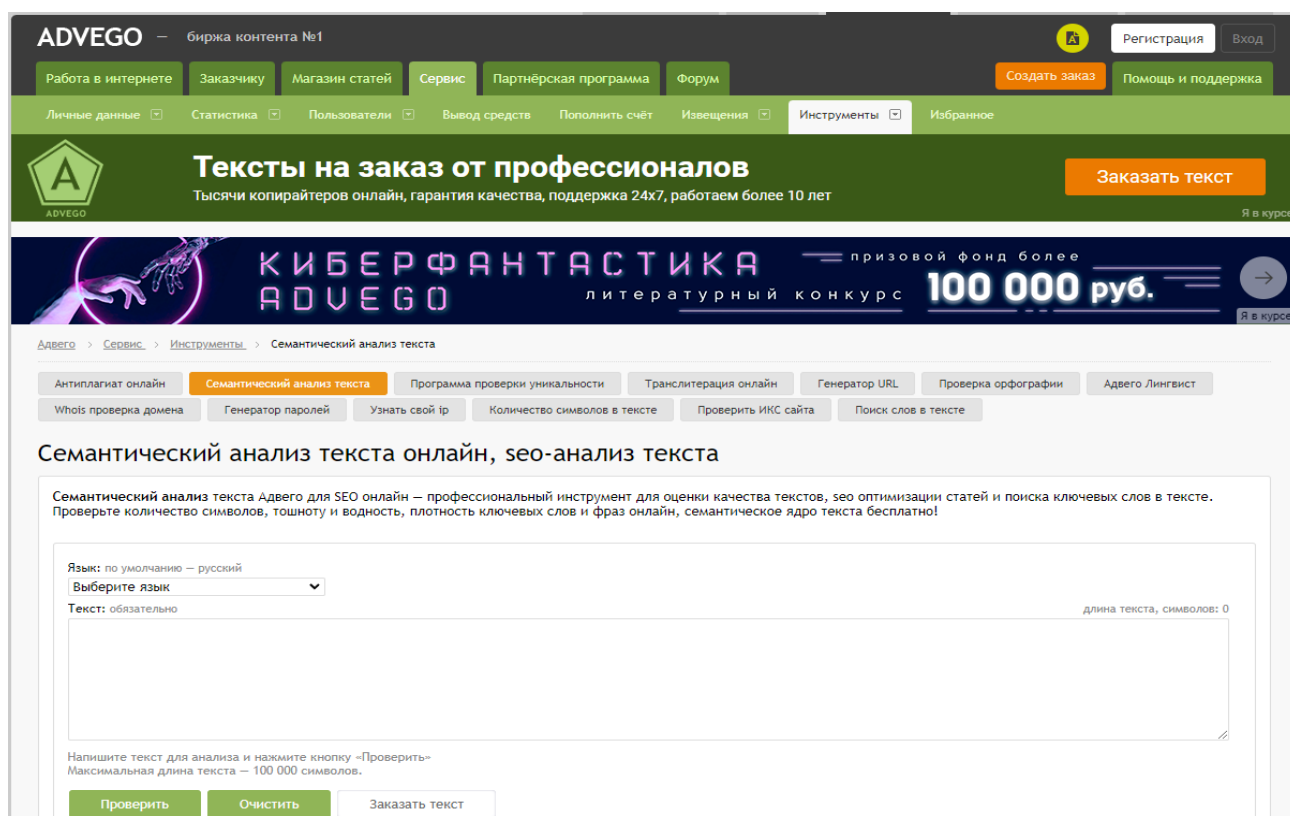


Рисунок 2 – Главная страница сайта

Статистика текста

Наименование показателя	Значение
Количество символов	561
Количество символов без пробелов	489
Количество слов	71
Количество уникальных слов	56
Количество значимых слов	19
Количество стоп-слов	22
Вода	73.2 %
Количество грамматических ошибок	0
Классическая тошнота документа	1.73
Академическая тошнота документа	12.6 %

Рисунок 3 – Результат обработки текста

Сервис имеет простой и понятный интерфейс, множество функций, среди которых антиплагиат, семантический анализ текста, транслитерация и т. д.

Сайт выполняет развернутый онлайн анализ текста, по множеству параметров: SEO, стоп-слова, количество символов, слов, уникальных слов, плотность ключевых фраз, водность, академическая и классическая тошнота и определяет семантическое ядро.

1.2.3 Сервис Главред [3].

Графический интерфейс сервиса представлен на рисунке 4.

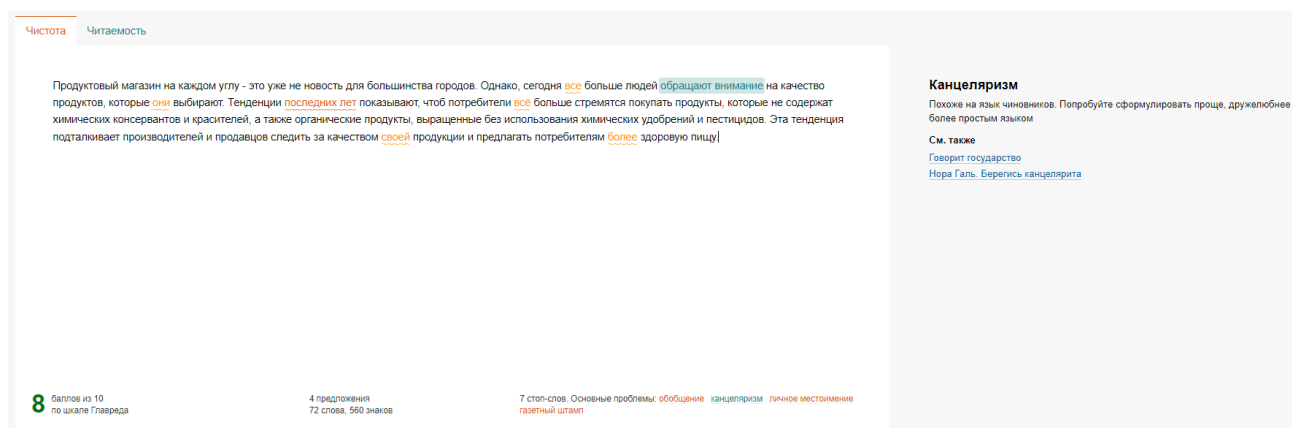


Рисунок 4 – Результат обработки текста

Сервис предоставляет услуги анализа текста на чистоту и читаемость, а также ставит оценку по этим параметрам от нуля до десяти.

Во вкладке "Чистота" приводятся рекомендации того, как стоит изменить слова или словосочетания так, чтобы текст выглядел более правдоподобно и приятно с точки зрения психологии.

Во вкладке "Читаемость" выделяются части текста и приводятся рекомендации, как изменить текст, так чтобы его проще было читать с точки зрения русского языка.

Все вышеописанные анализаторы текстовой информации являются облачными сервисами, доступными или по подписке на платной основе или имеют ограниченный функционал, расширение которого силами пользователя невозможно. Целью данной работы является реализация свободно распространяемой версии текстового анализатора с открытым исходным кодом, что позволит пользователю самостоятельно усовершенствовать его под свои специфические требования.

1.3 Постановка задачи

Потоковое клиент-серверное приложение анализатор текста должно иметь следующий функционал, который представлен ниже.

Со стороны клиента анализатор должен иметь два независимых потока.

Первый поток клиентского приложения должен уметь:

- читать и проверять текст по критериям (текст состоит только из русских букв и содержит не более 32 слов);
- если проверка пройдена, то текст будет передан второму потоку.

Второй поток клиентского приложения должен уметь:

- транслитерировать текст, полученный от первого потока;
- формировать набор данных из текста в формате (слово: кол-во этого слова в тексте);
- отправлять набор данных на сервер.

Со стороны сервера приложение должно уметь:

- проверять полученный набор данных от клиента на наличие хотя бы двух разных слов;
- если проверка пройдена, то производить преобразование данных в формат (буква-кол-во использования);

– сохранять результат в файле *Statistics.dat*, если файл уже существует, то данные из полученного текста должны дополнять его, а не создавать новый файл.

1.4 Требования к среде разработки и исполнения программы

Разработанное ПО должно работать на персональных компьютерах под управлением операционной системы *Windows*.

Аппаратной платформой для разработки и исполнения ПО является персональный компьютер на базе X86 архитектуры.

Операционная система *Windows* должна содержать:

- среда разработки: *Visual Studio*;
- язык программирования *C++*.

Используемые технологии:

- *Win32 API*;
- *C++ STLlib*.

1.5 Функциональные требования к ПО

Функциональное требование – это заявление о том, как должна вести себя система. Он определяет, что система должна делать, чтобы удовлетворить потребности или ожидания пользователя. Функциональные требования можно рассматривать как функции, которые обнаруживает пользователь.

ПО представляет собой мультипользовательский программный продукт. Пользователь должен иметь возможность ввода и отправки текстового сообщения на сервер. Ввод и отправка осуществляется из текстовой консоли приложения. Длина вводимого текста должна быть не более 2048 байт.

Сервер получает текстовое сообщение и обрабатывает его с помощью алгоритма анализа. Сервер фиксирует результат в текстовом файле.

					ПГУ1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		11

Администратор выполняет функцию контроля исполнения программы. Администратор запускает серверную часть приложения и имеет права на просмотр результата анализа.

И администратор, и пользователь имеют право завершить работу клиентской и серверной части приложения в любой момент времени.

На рисунке 5 показаны варианты взаимодействия с приложениями.

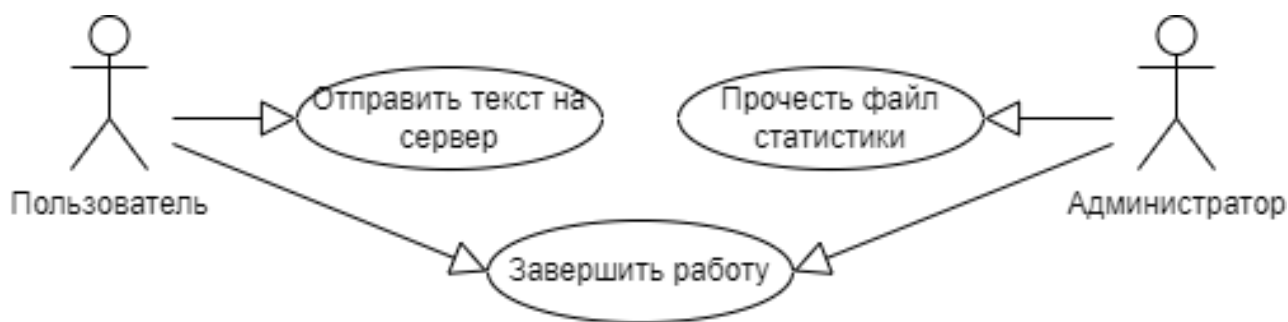


Рисунок 5 – Диаграмма вариантов использования

1.6 Выбор инструментов разработки

1.6.1 Выбор языка программирования

Для реализации проекта требуется выбрать язык программирования. Для реализации клиент-серверного приложения OS Windows используются языки высокого уровня: C++, Java, C#, Python.

Ниже приведен сравнительный анализ данных языков программирования.

Язык программирования C++.

C++ – компилируемый, статически типизированный язык программирования общего назначения.

Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В

сравнении с его предшественником – языком *C* – наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования [4].

C++ позволяет создавать приложения и программы любой сложности: от простых консольных утилит до сложных игровых движков. Также на этом языке можно программировать микроконтроллеры и системы в реальном времени. На *C++* можно написать практически все что угодно, от системных приложений до мобильных приложений. Этот язык используется для создания операционных систем, программного обеспечения разного рода и игровых движков [5].

Язык программирования *Java*.

Java – строго типизированный объектно-ориентированный язык программирования общего назначения.

Программы на *Java* транслируются в байт-код *Java*, выполняемый виртуальной машиной *Java (JVM)* – программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять *Java*-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии *Java* является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят снижение производительности.

Язык программирования *C#*.

C# – объектно-ориентированный язык программирования общего назначения. Язык разработки приложений для платформы *Microsoft .NET Framework* и *.NET Core*.

C# относится к семье языков с *C*-подобным синтаксисом, из них его синтаксис наиболее близок к *C++* и *Java*. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, переменные, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, *LINQ*, исключения, комментарии в формате *XML*.

Язык программирования *Python*.

Python – высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами. Синтаксис ядра языка минималистичен, за счёт чего на практике редко возникает необходимость обращаться к документации. Сам же язык известен как интерпретируемый и используется в том числе для написания скриптов. Недостатками языка являются зачастую более низкая скорость работы и более высокое потребление памяти написанных на нём программ по сравнению с аналогичным кодом, написанным на компилируемых языках, таких как *C* или *C++*.

Учитывая преимущества и недостатки описанных языков программирования, был выбран язык *C++*, так как является самым гибким и наименее требовательным к ресурсам ПК языком.

1.6.2 Выбор среды разработки для проекта

Для реализации проекта требуется выбрать среду разработки. Самые популярные среды разработки для языка программирования *C++*: *VisualStudio*, *Eclipse*, *QtCreator*, *NetBeans*.

Среда разработки *Eclipse*.

					ПГУ1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		14

Eclipse – свободная интегрированная среда разработки модульных кроссплатформенных приложений, которая приобрела большую популярность среди разработчиков на *Java*, однако в данной статье мы рассмотрим выпуск *Eclipse (C/C++ Development Tooling)*. Данная среда является отличным выбором, так как *Eclipse* не только обладает всеми необходимыми инструментами, но и бесплатна и работает под различными операционными системами: *Windows/Linux/macOS*.

Плюсы:

- бесплатное использование;
- автозавершение и другие возможности, которые помогают быстрее писать код;
- большой набор плагинов для расширения функциональности;
- развитое сообщество разработчиков, подробная документация;
- встроенное юнит-тестирование, оптимизация тестов;
- настраиваемый графический интерфейс.

Минусы:

- медленный запуск, потребление большого объёма памяти;
- проблемы с обратной совместимостью;
- возможны конфликты плагинов.

Среда разработки *QtCreator*.

Qt Creator – интегрированная среда разработки *C++*, доступная на *Windows, Linux* и *macOS*. Предоставляет бесплатную версию, которой можно пользоваться в течение 1 месяца. Предлагает полный набор инструментов разработчика, предназначенных для создания и развёртывания приложений.

Плюсы:

- поддерживает отладку, профилирование, автозавершение кода и рефакторинг;
- возможность компиляции проектов для разных ОС.

Минусы:

					ПГУ1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		15

- большой вес приложений;
- не всегда работает автозавершение кода;
- дорогая платная версия;
- нужна регистрация для скачивания бесплатной версии.

Среда разработки *NetBeans*.

NetBeans – бесплатная интегрированная среда разработки. Позволяет создавать приложения на языках программирования *Java*, *Python*, *PHP*, *JavaScript*, *C*, *C++* и ряде других. У этой интегрированной среды программирования *C++* есть дистрибутива для платформ *Microsoft Windows*, *Linux*, *FreeBSD*, *macOS*, *OpenSolaris* и *Solaris*, а для всех остальных платформ есть возможность собрать *NetBeans* самостоятельно из исходников.

Плюсы:

- бесплатная интегрированная среда разработки *C++*;
- кроссплатформенная поддержка;
- большой выбор плагинов;
- автозавершение кода, инструменты для рефакторинга;
- развитое сообщество разработчиков.

Минусы:

- медленный запуск;
- проблемы с собственным кэшем при сборке готовых программ;
- для установки требуется *JDK*.

В качестве среды разработки был выбран *Microsoft Visual Studio Community 2022*, так как имеет все необходимое для разработки программы на *C++*.

Плюсы:

- бесплатная *IDE*;
- поддержка разных языков программирования;
- мощный компилятор;
- тонкие настройки.

Минусы:

- высокие требования к ПК;
- занимает много места на диске.

Крайне важная вещь для *IDE* – это интерфейс. Если интерфейс прост и понятен, то он увеличивает производительность разработчика. Интерфейс *Microsoft Visual Studio 2022* изображен на рисунке 6.

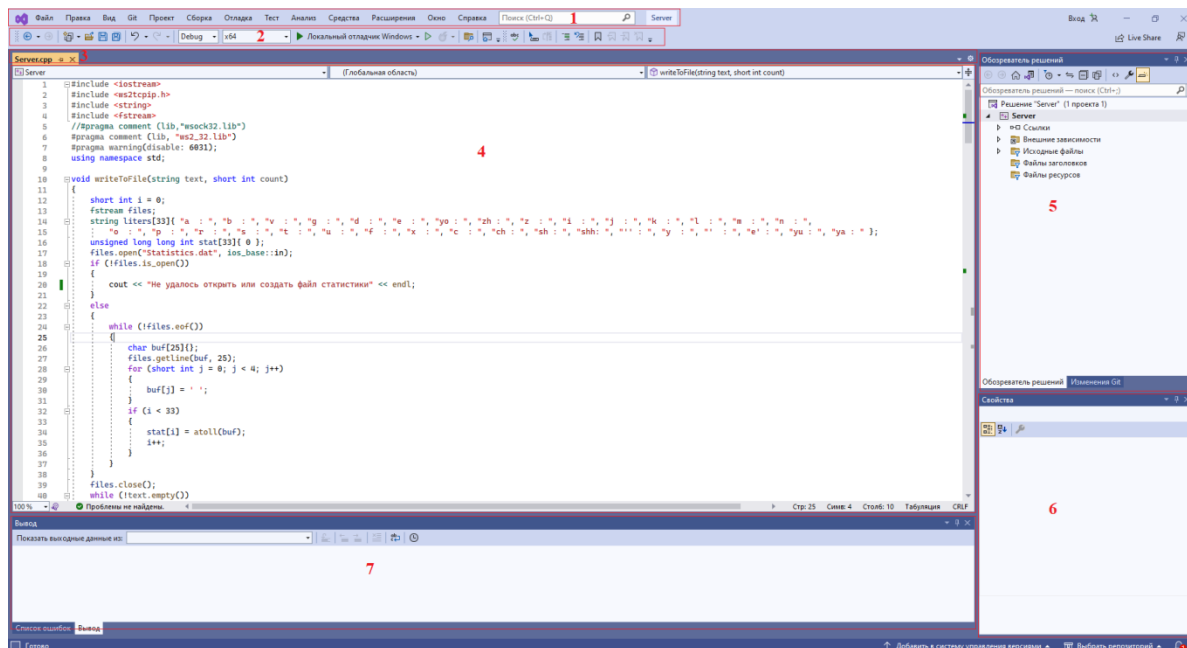


Рисунок 6 – Интерфейс *Microsoft Visual Studio 2022*

Панель меню(1) – здесь представлены различные вкладки, с помощью которых можно тонко настроить *GUI*, создать новый проект, открыть существующий проект, настроить вещи, относящиеся к отладке, компиляции, проекту.

Панель инструментов(2) – на панели инструментов находятся ярлыки для быстрого запуска компиляции, отладки, сохранения, которые можно настроить по своему желанию.

Панель навигации(3) – здесь находятся все открытые в окне редактора файлы исходного кода, заголовков, библиотек, нужна для более удобной работы с несколькими файлами одновременно.

Окно редактора(4) – это область, в которой разработчик пишет или редактирует код. Здесь имеется подсветка синтаксиса, ошибок и возможных вариантов кода.

Обозреватель решений(5) – здесь развернута вся структура проекта, все библиотеки, файлы исходного кода, заголовочные файлы, существующие внутри проекта, но не открытые в данный момент.

Панель свойств(6) – в этой части интерфейса показаны свойства конкретного файла проекта или элемента формы, если ведется разработка *GUI*.

В нижней части интерфейса(7) обычно располагается область, в которой в зависимости от настроек отслеживается, построение, сборка, значения переменных во время отладки, состояние ячеек памяти, точки останова и т. д.

Соединение клиента с сервером реализовано с помощью *Windows Sockets*.

Поскольку разные программы обычно не имеют доступа к ресурсам друг друга, то для того чтобы независимые процессы могли начать взаимодействие друг с другом требуется каким либо образом организовать передачу информации между ними.

Самым простым и понятным способом организации сетевого взаимодействия является построение канала между этими процессами.

Сокеты обеспечивают механизм связи двух компьютеров использующих *TCP*. Клиентская программа создаёт со своей стороны сокет и ожидает подключения. Когда подключение установлено серверная программа создает сокет и на своём конце и если соединение удачно, то теперь клиент и сервер могут обмениваться информацией.

При установлении связи между компьютерами выполняются следующие этапы:

- создание *WSADATA* и её заполнение с помощью функции *WSAStartup()*;
- создание структуры *SOCKET* и её инициализация с помощью функции *socket()* с серверной и клиентской части программы;
- привязка сокета сервера к паре *IP/Порт* с помощью функции *bind()*;

- сервер ожидает подключения при помощи функции *listen()*;
- клиент пытается подключиться к серверу функцией *connect()*;
- сервер подтверждает клиент серверное соединение функцией *accept()*.

После того, как соединение установлено, одна программа может передавать данные с помощью функции *send()*, а другая принимать данные с помощью функции *recv()*.

Методы *Windows Sockets*:

- *int WSAStartup (WORD wVersionRequired, WSADATA lpWSAData).*

Запускает программный интерфейс сокетов. Первым аргументом принимает версию сокетов, вторым указатель на структуру *WSADATA*;

- *SOCKET socket (int ar, int type, int protocol).* Инициализирует структуру *SOCKET*. Первым аргументом определяется семейство адресов, вторым тип сокета, третьим протокол;

- *int bind (SOCKET s, const sockaddr *addr, int namelen).* Назначает адрес сокета. Первым аргументом принимает имя сокета, вторым указатель на структуру с информацией адреса и порта, третьим размер структуры второго аргумента;

- *int listen (SOCKET s, int backlog).* Ожидает подключения со стороны клиента. Первый аргумент это принимающий сокет, второй максимальное число подключений;

- *int connect (SOCKET s, const sockaddr *name, int namelen).* Подключается к серверу. Первым аргументом принимает имя сокета, вторым структуру, содержащую *IP* и порт сервера, третьим размер структуры второго аргумента;

- *SOCKET accept (SOCKET s, sockaddr *addr, int *addrlen).* Подтверждает соединение с конкретным клиентом. Первым аргументом принимает сокет сервера, вторым пустую структуру *sockaddr* в которую будут записаны данные клиента, третьим указатель на размер второго аргумента;

– *int recv/send (SOCKET s, const char *buf, int len, int flags)*. *Send* отправляет данные, *recv* принимает. Первым аргументом идет имя подключенного сокета, вторым буфер хранящий отправляемую/принимаемую информацию, третьим размер буфера, четвертым флаги.

1.7 Реализация программы

1.7.1 Клиентская часть

secondPart – класс содержащий функции создания сокета и функции запускаемой во втором потоке программы.

Ниже представлен код класса.

```
class secondPart {
public:
    WSADATA wsd; // структура для WSStartup
    SOCKET s1;
    bool servIsWork = false;
    int startServer();
    void doMyWork(string text);
    void endWork();
};
```

Ниже приведены методы класса *secondPart*.

startServer() – создает и настраивает все, что связано с сокетом и пытается подключиться к серверу, возвращает *int*. Если возвращаемое значение равняется 1, то попытка создать соединение провалилась на этапе создания сокета, если 2, то попытка провалилась на этапе подключения к серверу, если 0, то соединение успешно установлено.

doMyWork() – получает строку, обрабатывает ее, вызывая функции *transliterate()* и *parseStringToWords()*, затем отправляет результат на серверную часть программы.

Ниже представлен код метода.

```
void secondPart::doMyWork(string text)
{
```

```

text = transliterate(text);
text = parseStringToWords(text);
if (send(s1, text.c_str(), text.size(), 0) == SOCKET_ERROR)
{
    cout << "Error: send!" << endl;
    servIsWork = false;
}
else
    cout<< "Текст отправлен на сервер" <<endl;
}

```

endWork() – завершает все операции сокетов, прерывает клиент-серверное соединение, закрывает сокет.

Ниже представлен код метода.

```

void secondPart::endWork()
{
    shutdown(s1, SD_BOTH);
    closesocket(s1);
    WSACleanup();
}

```

Ниже представлены методы клиентской части приложения.

countWords() – подсчитывает количество слов в строке, использует пробел в качестве разделителя. Возвращает *true*, если слов меньше 32 иначе возвращает *false*.

Ниже представлен код метода.

```

bool countWords(stringtext)
{
    int count = 0;
    char previous = ' ';
    for (int i = 0; i < text.length(); i++)
    {
        if (text[i] != ' ' && previous == ' ')
        {
            count++;
        }
    }
}

```

```

        }
        previous = text[i];
    }
    if (count > 32)
    {
        return false;
    }
    return true;
}

```

cleanStringFromZnaki() – очищает строку от всех символов кроме букв и пробелов.

Ниже представлен код метода.

```

string cleanStringFromZnaki(string text)
{
    char previous = ' ';
    for (int i = 0; i < text.length(); )
    {
        if (previous == text[i] && text[i] == 32 || text[i] > 32
        && !isalnum(text[i]) || text[i] >= 48 && text[i] <= 57 || text[i] <
        -64 && text[i] != -72 && text[i] != -88)
        {
            text.erase(i, 1);
        }
        else
        {
            previous = text[i];
            i++;
        }
    }
    return text;
}

```

bazarPoNashemu() – проверяет, чтобы строка состояла только из русских букв. Возвращает *true*, если строка состоит только из русских букв, иначе возвращает *false*.

Ниже представлен код метода.

```
bool bazarPoNashemu(string text)
{
    for (int i = 0; i < text.length(); i++)
    {
        if (text[i] == 32 || text[i] < 0 && text[i] >= -64 ||
text[i] == -72 || text[i] == -88)
        {

        }
        else
            return false;
    }
    return true;
}
```

transliterate() – транслитерирует строку.

Ниже представлен код метода.

```
string transliterate(string text)
{
    string trText;
    for (; !text.empty(); )
    {
        switch (text[0])
        {
            case ' ': trText.append(" "); break;
            case 'А':
            ...
            case 'Я':
            case 'я': trText.append("ya"); break;
        }
    }
}
```

```

        text.erase(0, 1);
    }
    return trText;
}

```

parseStringToWords() – преобразует строку в формат (слово:кол-во вхождений слова), с запятой в качестве разделителя.

Ниже представлен код метода.

```

string parseStringToWords(string text)
{
    short int i = 0;
    string textb;
    string words[32];
    short int cwords[32]{};
    while (!text.empty())
    {
        int pos = 0;
        string tempStr;
        bool isDup = false;
        pos = text.find(' ');
        if (pos != string::npos)
        {
            tempStr = text.substr(0, pos);
            text.erase(0, pos+1);
        }
        else
        {
            tempStr = text;
            text.clear();
        }
        for (i = 0; !words[i].empty(); i++)
        {
            if (words[i] == tempStr)
            {
                isDup = true;
            }
        }
    }
}

```

```

        break;
    }
}
if (isDup)
{
    cwords[i]++;
}
else
{
    words[i] = tempStr;
    cwords[i]++;
}
}
i = 0;
while (!words[i].empty())
{
    textb.append(words[i]);
    textb.append(":");
    textb.append(to_string(cwords[i]));
    textb.append(",");
    i++;
}
return textb;
}

```

1.7.2 Серверная часть

Методы серверной части приложения представлены ниже.

inputProc() – проверяет, чтобы в строке было, как минимум два отличающихся друг от друга слова, если условие истинно, то для каждого слова вызывается функция *writeToFile()*, иначе функция *inputProc()* прерывается.

Ниже представлен код метода.

```

void inputProc(string text)
{
    short int i = 0;

```

```

string words[32];
short int cwords[32]{};
while (!text.empty())
{
    int pos = 0;
    string tempStr;
    pos = text.find(',');
    tempStr = text.substr(0, pos);
    text.erase(0, pos + 1);
    pos = tempStr.find(':');
    words[i] = tempStr.substr(0, pos);
    cwords[i] = stoi(tempStr.substr(pos + 1,
tempStr.size()));
    i++;
}
if (i < 2)
{
    cout<< "Ошибка: Количество отличающихся друг от друга
слов должно быть не менее 2" <<endl;
    return;
}
while (i >= 0)
{
    writeToFile(words[i], cwords[i]);
    i--;
}
}

```

writeToFile() – ведет подсчет количества букв в строке и записывает их в файл *Statistics.dat*, если файл не существует, то создает, если существует, то дополняется новыми данными.

Ниже представлен код метода.

```

void writeToFile(string text, short int count)
{
    short int i = 0;

```

```

    fstream files;

    string liters[33]{ "a  : ", "b  : ", "v  : ", "g  : ", "d  : 
", "e  : ", "yo : ", "zh : ", "z   : ", "i   : ", "j   : ", "k   : ",
"l   : ", "m   : ", "n   : ",
        "o   : ", "p   : ", "r   : ", "s   : ", "t   : ", "u   : ", "f
: ", "x   : ", "c   : ", "ch  : ", "sh  : ", "shh: ", "'   : ", "y   : ",
"'    : ", "e'  : ", "yu  : ", "ya  : " };

    unsigned long long int stat[33]{ 0 };
    files.open("Statistics.dat", ios_base::in);
    if (!files.is_open())
    {
        cout<< "Не удалось открыть или создать файл статистики"
<<endl;
    }
    else
    {
        while (!files.eof())
        {
            char buf[25]{};
            files.getline(buf, 25);
            for (short int j = 0; j < 4; j++)
            {
                buf[j] = ' ';
            }
            if (i < 33)
            {
                stat[i] = atoll(buf);
                i++;
            }
        }
    }
    files.close();
    while (!text.empty())
    {
        if (text.length() > 2 && text.substr(0, 3) == "shh")

```

```

        {
            stat[26] += count;
            text.erase(0, 3);
            continue;
        }
        else
            ...
            (код определения буквы)
            ...
            stat[i] += count;
            text.erase(0, 1);
        }
        files.open("Statistics.dat", ios_base::out);
        if (!files.is_open())
        {
            cout<< "Не удалось открыть или создать файл статистики"
<<endl;

            return;
        }
        for (i = 0; i < 33; i++)
        {
            string temp = "";
            temp.append(liters[i]);
            temp.append(to_string(stat[i]));
            files.write(temp.c_str(), temp.length());
            if (i < 32)
            {
                files.write("\n", 1);
            }
        }
        files.close();
    }

```

На рисунке 7 представлена диаграмма взаимодействия слоёв приложения.

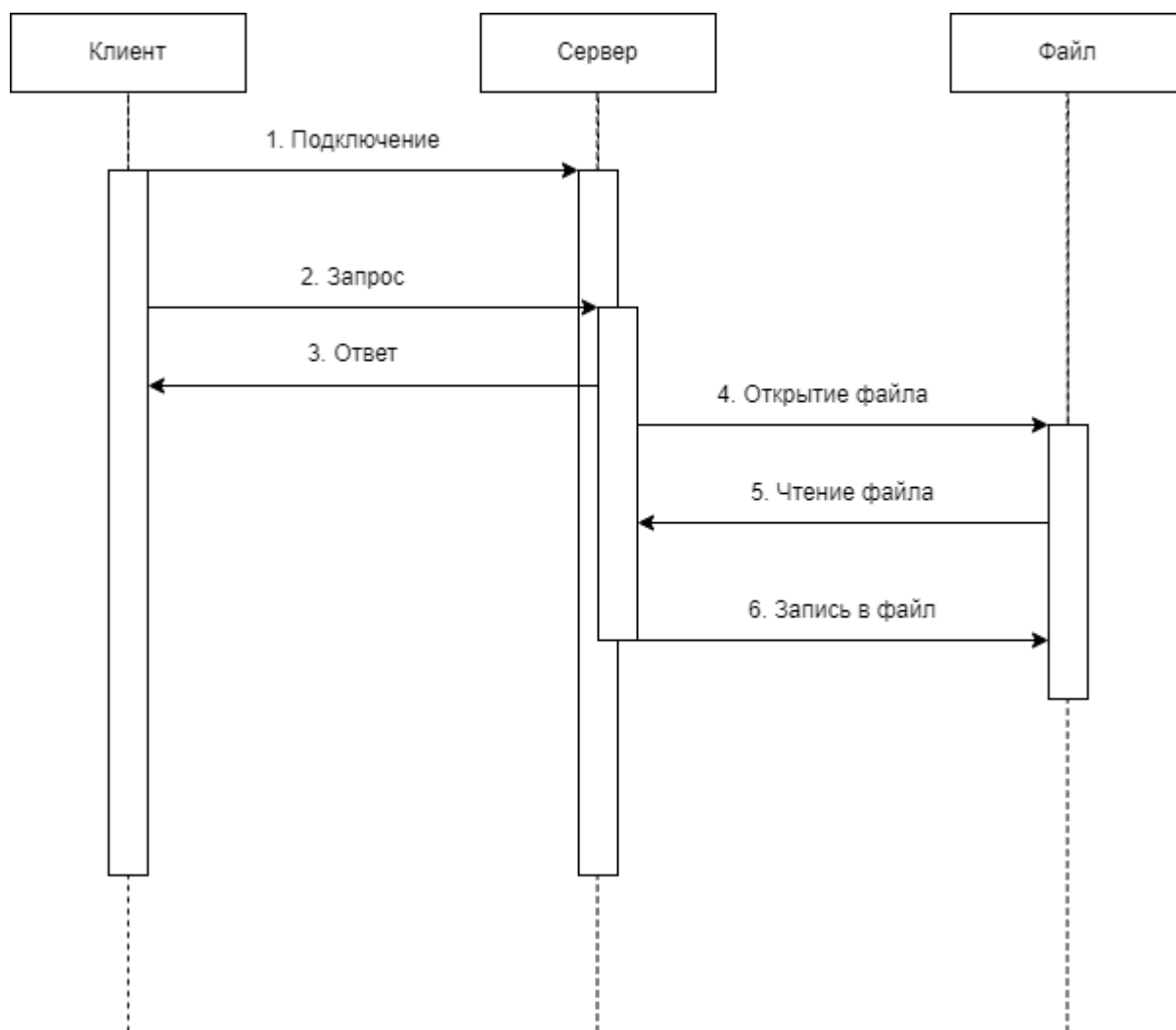


Рисунок 7 – Диаграмма взаимодействия слоёв приложения

1.8 Описание системы и программы с помощью диаграммы развертывания

Для описания архитектуры приложения была выбрана UML диаграмма развертывания, так как она может описать структуру программы. Диаграмма показывает, как компоненты системы взаимодействуют между собой в рамках сети.

Диаграмма развертывания помогает в планировании реализации программы. Диаграмма развертывания приведена на рисунке 8.

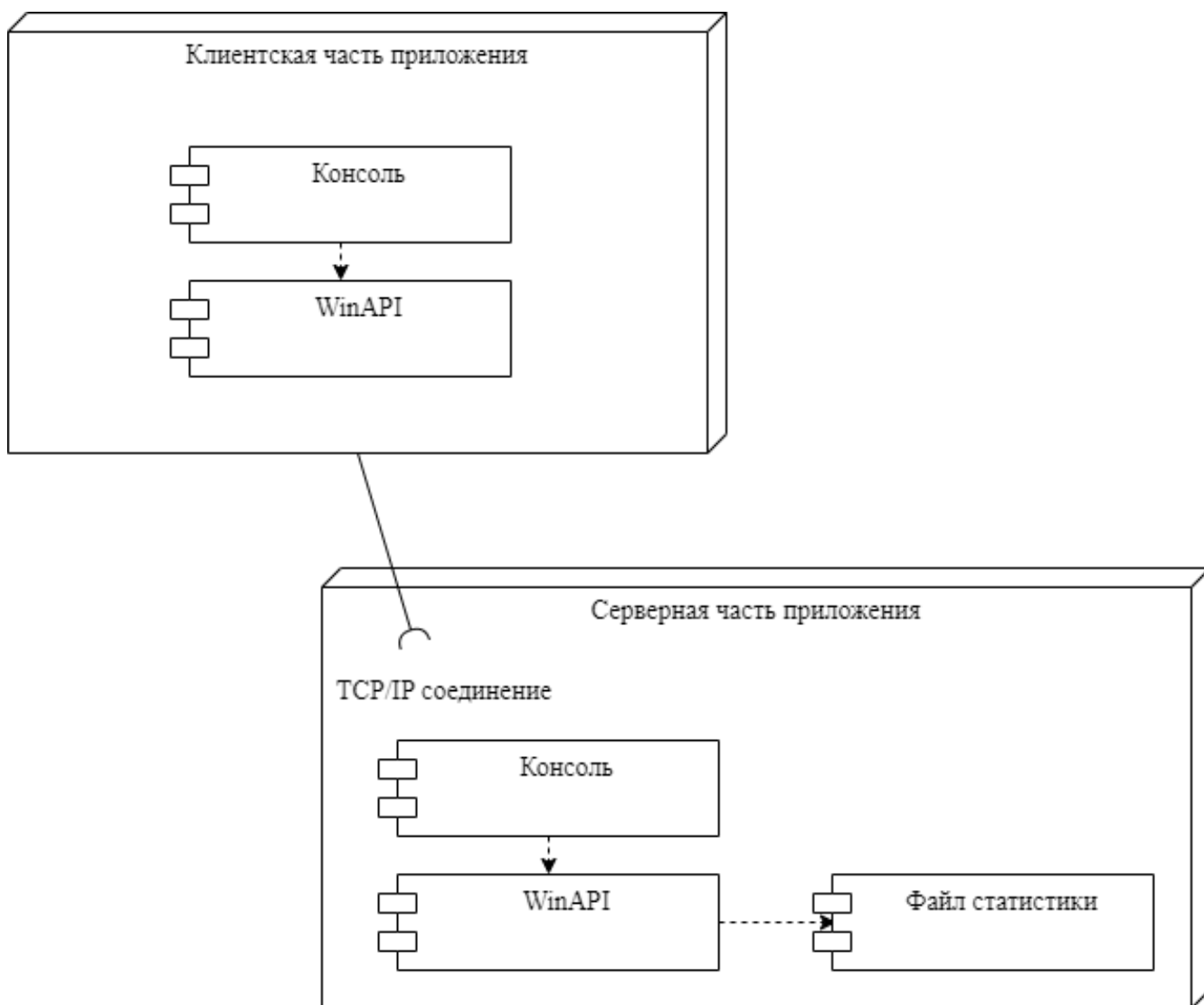


Рисунок 8 – Диаграмма развертывания

1.9 Отладка программы

Для проверки правильности работы не только с точки зрения результата, но и с точки зрения правильности действий ведущих к результату и существует отладка.

Сразу после введения строки вызывается функция, которая очищает строку от всех символов, не являющихся буквами.

На рисунках 9-23 показана отладка с сопровождающим пояснением перед рисунком.

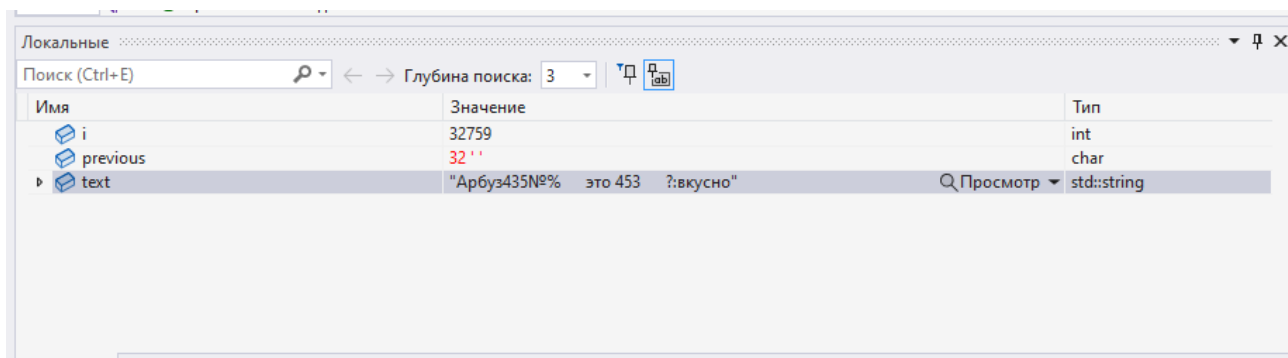


Рисунок 9 – Начало функции *cleanStringFromZnaki()*

В процессе функция *cleanStringFromZnaki()* проверяет каждый символ и удаляет его, если это не буква, или несколько пробелов находятся рядом.

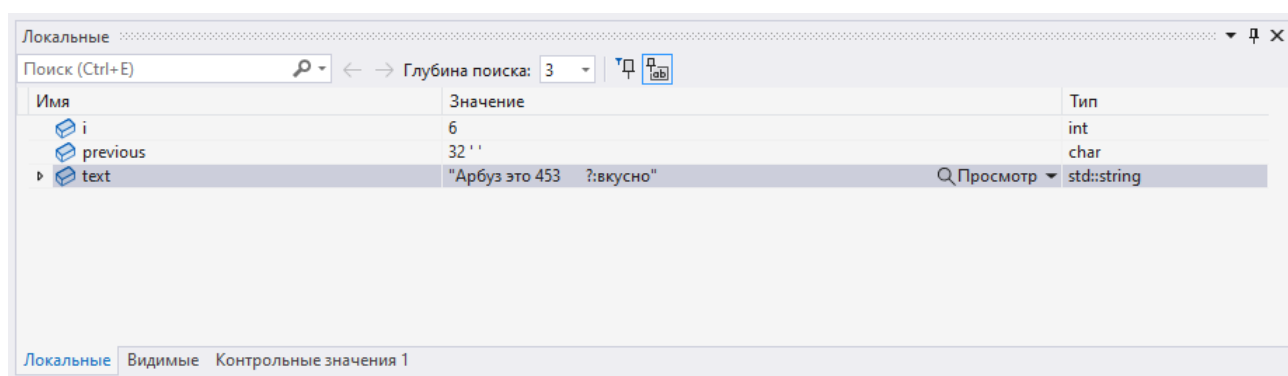


Рисунок 10 – Функция *cleanStringFromZnaki()* в процессе

По завершении функции очистки строки должны остаться только буквы и не должно быть более одного пробела подряд.

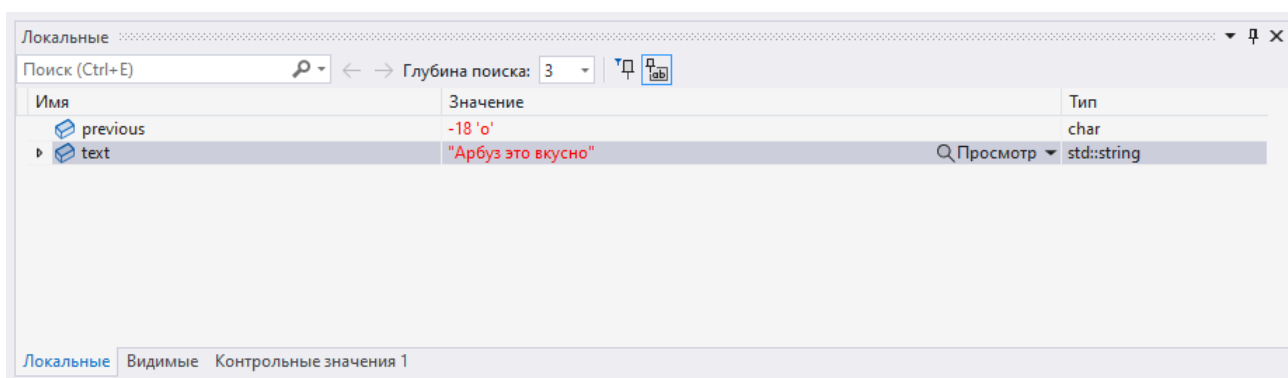


Рисунок 11 – Функция *cleanStringFromZnaki()* по завершении

После функции очистки строки идет функция, что считает количество слов в строке.

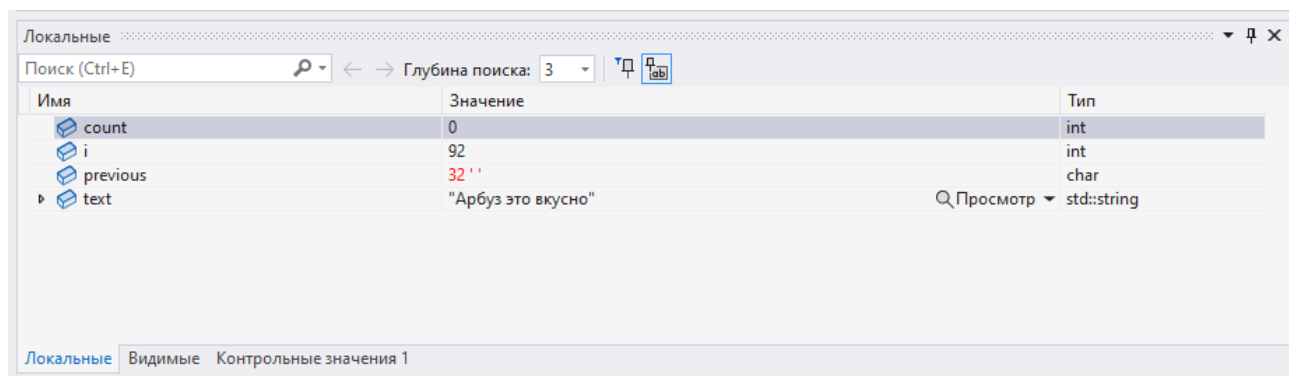


Рисунок 12 – Начало функции подсчёта слов

Функция считает количество пробелов прибавляя в самом начале 1 к результату.

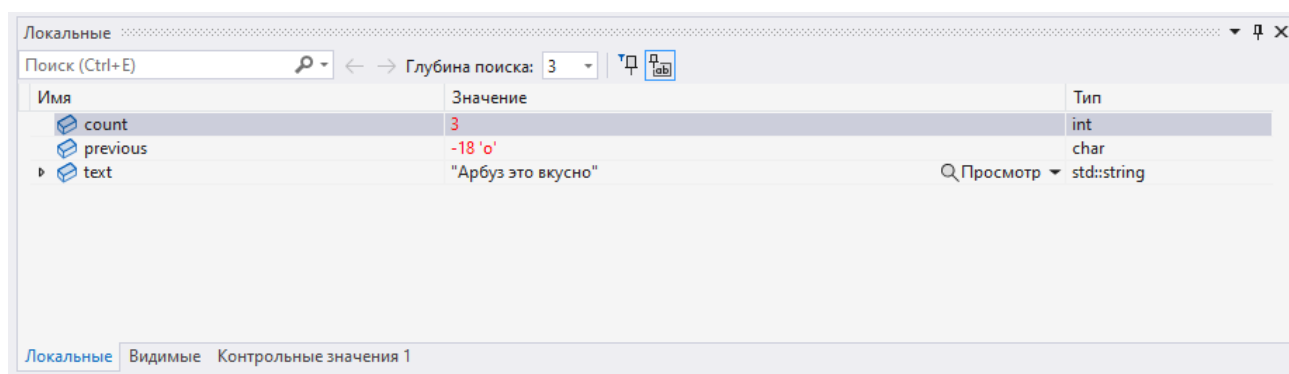


Рисунок 13 – Функция подсчета слов по завершении

Далее следует функция транслитерации текста, которая проверяет букву, а затем записывает ее транслитерированный вид во временную строку, удаляя символ из изначальной строки.

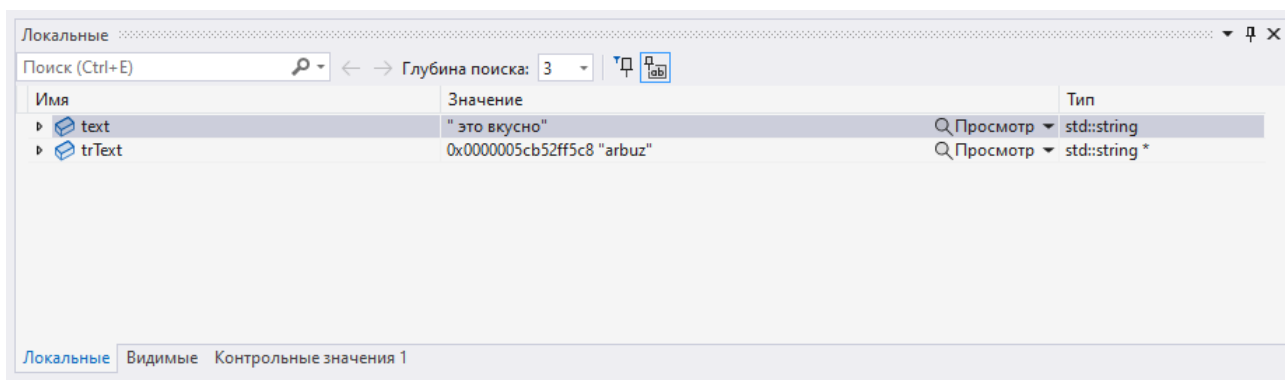


Рисунок 14 – Функция транслитерации в процессе

В качестве результата функцией будет возвращена транслитерированная строка.

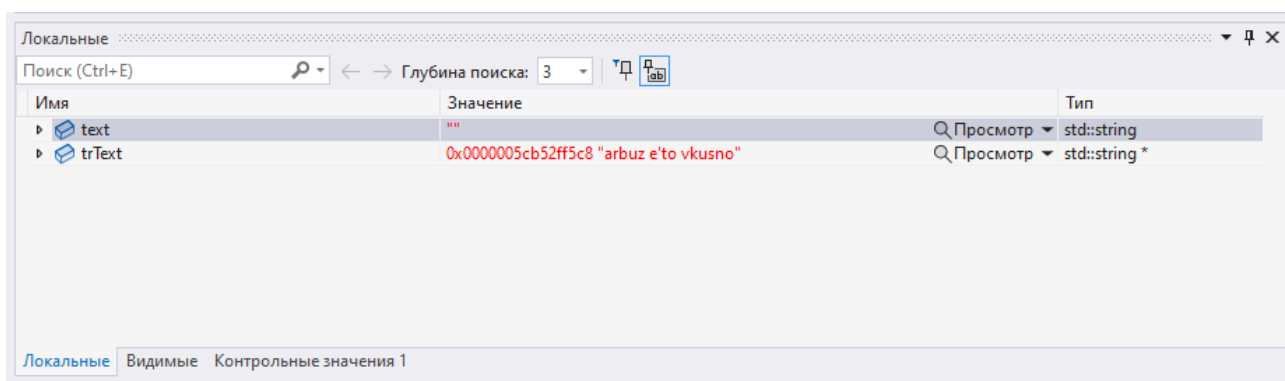


Рисунок 15 – Завершение функции транслитерации

Следующая функция преобразовывает транслитерированную строку в массив слов.

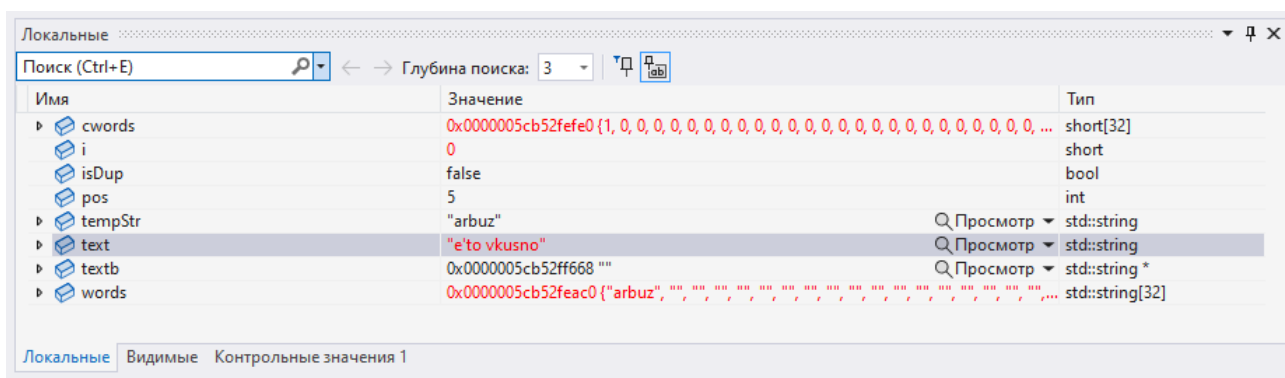


Рисунок 16 – Разбиение строки на слова

Строка разбивается на два массива, в `words[]` записываются слова, в `cwords[]` количество этих слов.

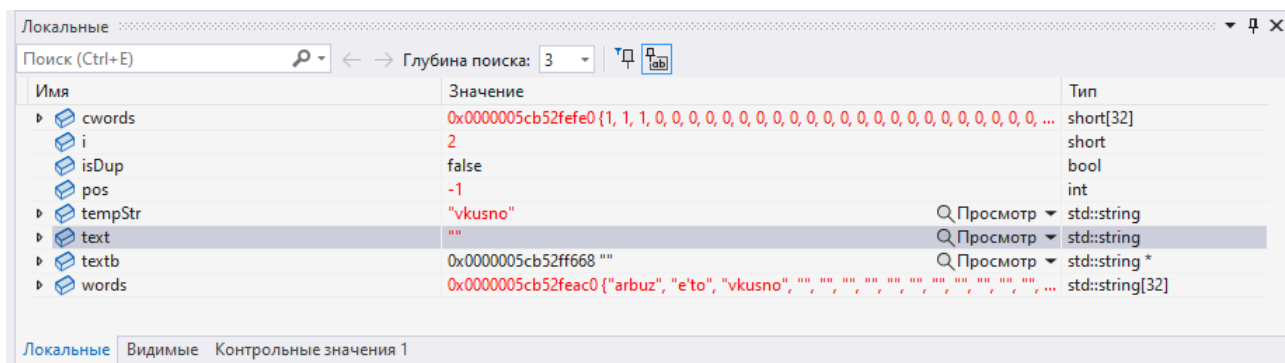


Рисунок 17 – Результат разбиения строки

Далее функция использует полученные массивы для того, чтобы создать строку в формате (Слово:кол-во слов) используя в качестве разделителя запятую.

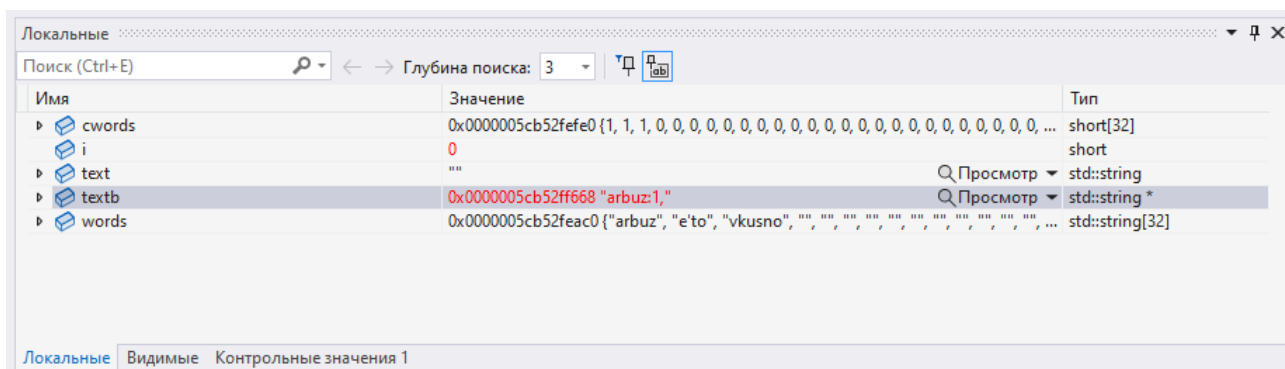


Рисунок 18 – Создание строки

Когда строка будет завершена, то она будет отправлена на сервер

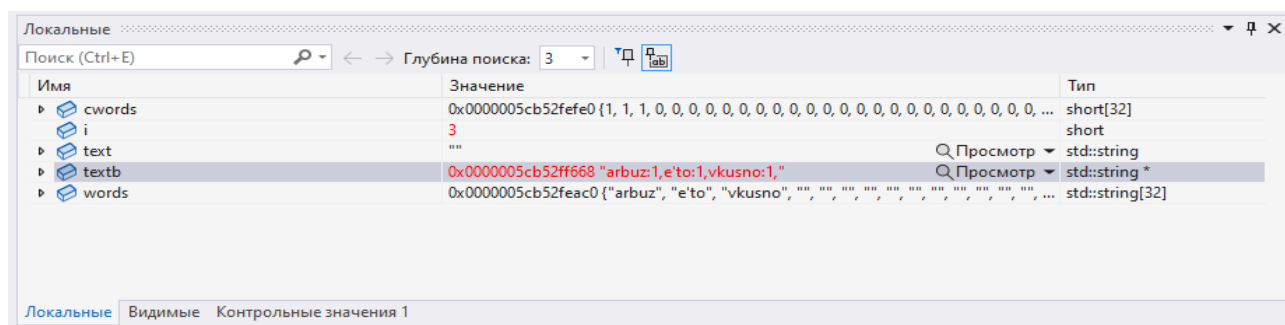


Рисунок 19 – Итоговая строка

Серверная часть приложения после получения строки разбивает её на слова и считает количество разных слов. Если слов менее 2, то сервер не обрабатывает строку.

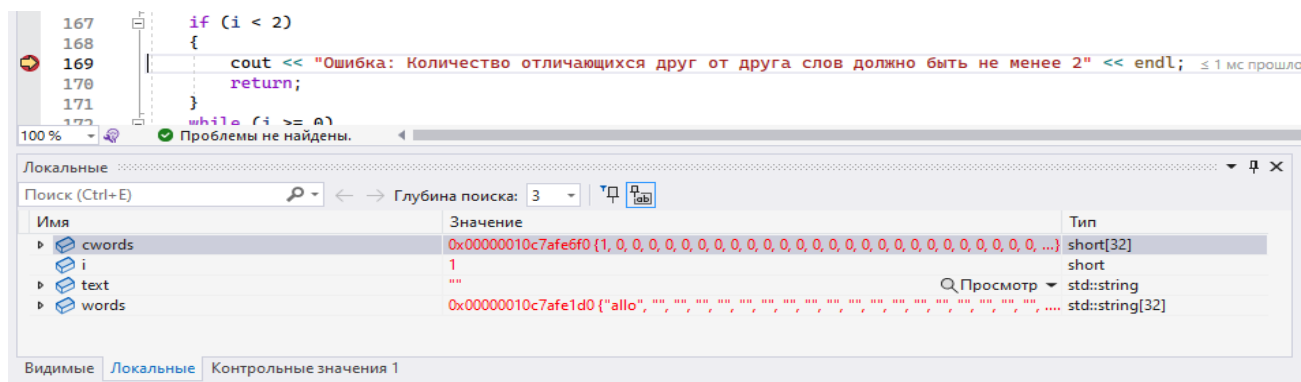


Рисунок 20 – Проверка сервера на количество слов в строке

Если же слов более 1, то для каждого слова вызывается функция записи в файл. Переменная *text* хранит слово, переменная *count* хранит количество этих слов в изначальном тексте.

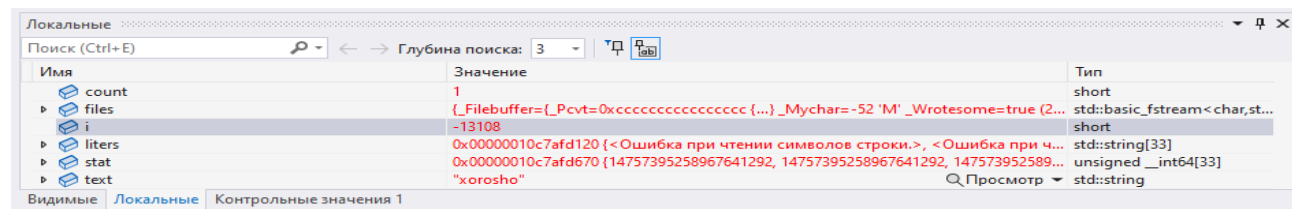


Рисунок 21 – Начало функции записи в файл

Если файл статистики существовал, то информация из него будет записана в массив *stat[]*.

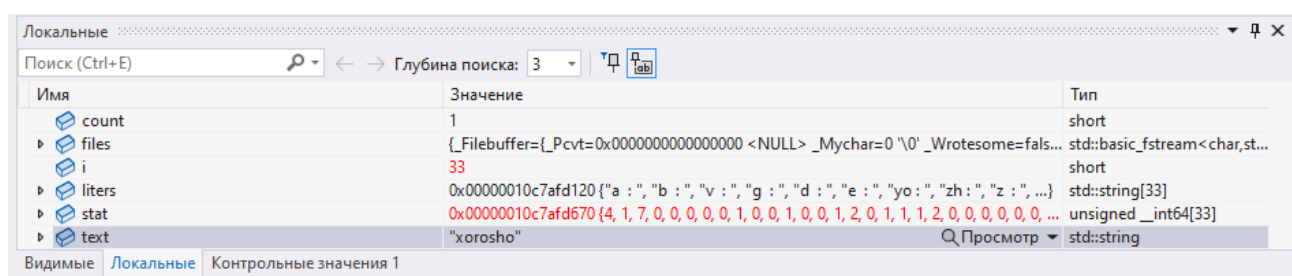


Рисунок 22 – Чтение файла

Затем массив `stat[]` будет дополнен новой информацией.

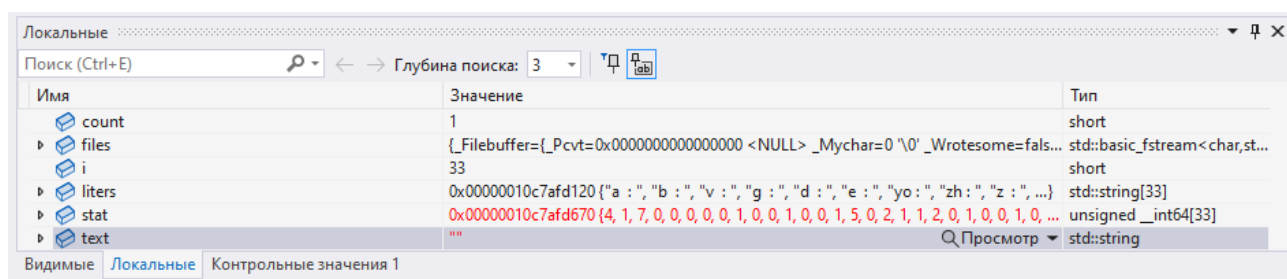


Рисунок 23 – Конец функции записи в файл.

1.10 Описание работы программы

На рисунках 24-32 показана работа программы с сопровождающим пояснением перед рисунком.

Сразу после запуска программы сервер ожидает подключения.

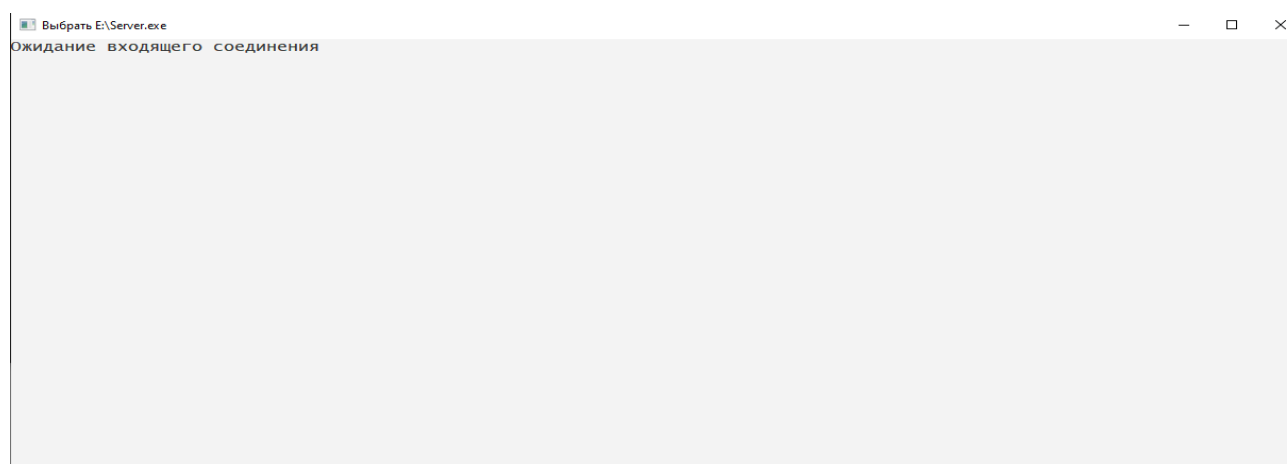


Рисунок 24 – Ожидание подключения (серверная часть)

После запуска программы клиент, ожидает ввода *IP* адреса.

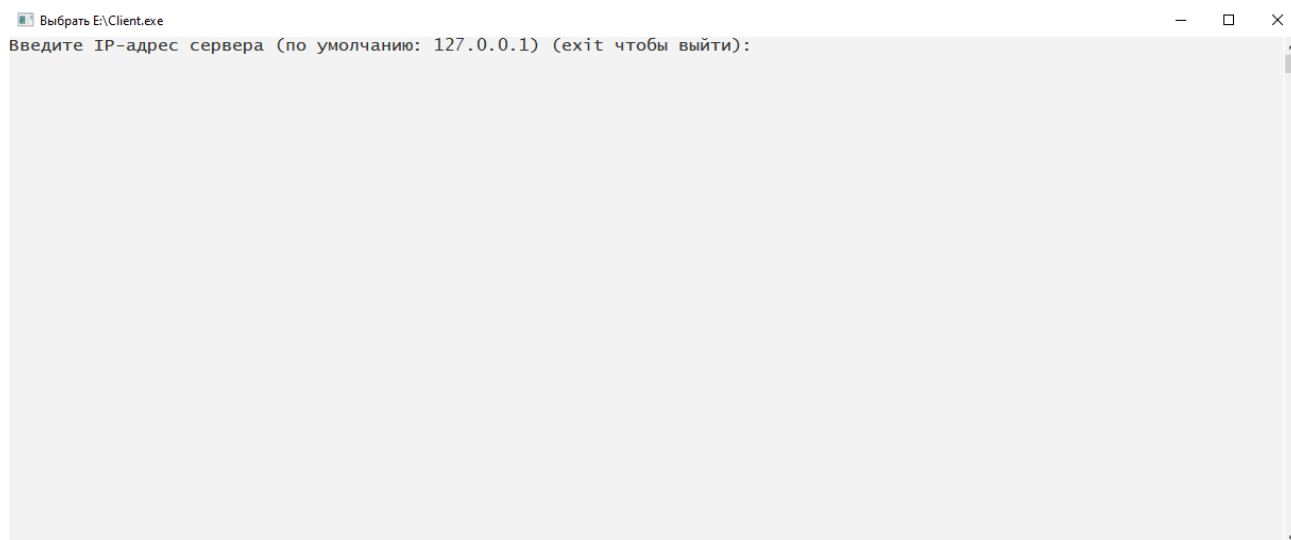


Рисунок 25 – Ожидание подключение (клиентская часть)

Если не удастся подключиться, то программа сообщит об этом и попросит снова ввести *IP*-адрес.

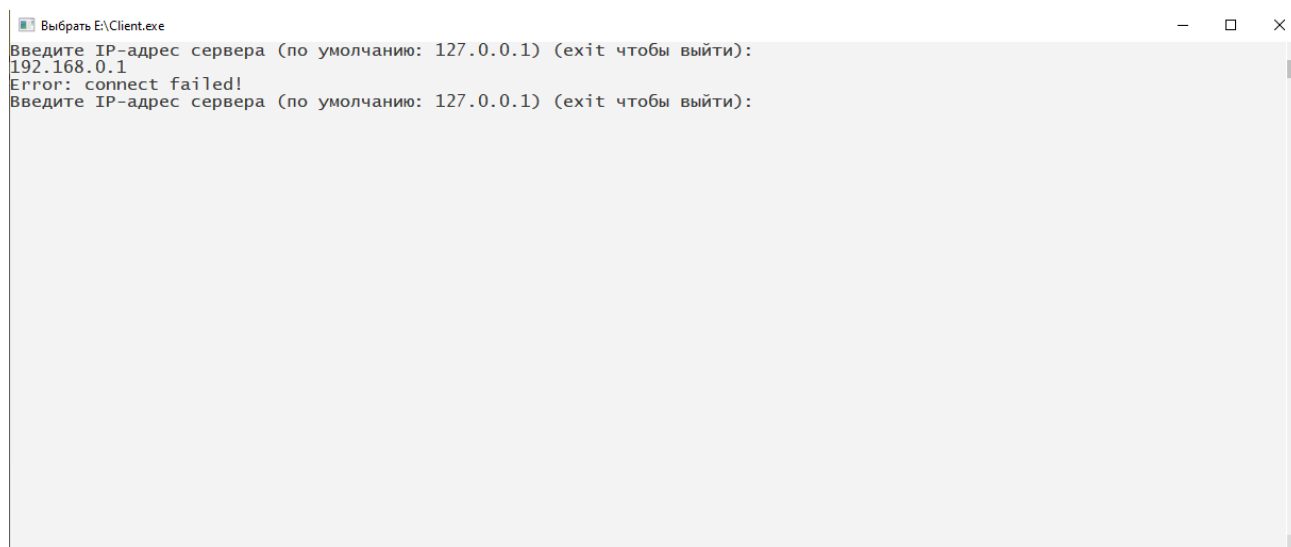


Рисунок 26 – Ошибка подключения.

При удачном подключении клиент и сервер напишут об этом.

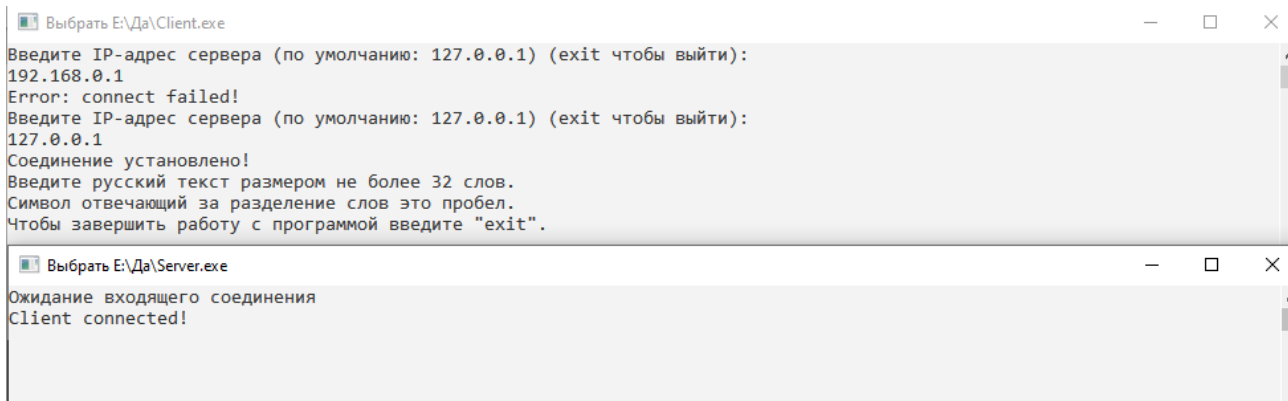


Рисунок 27 – Подключение успешно

Клиент будет сообщать об ошибках, если текст будет некорректен.

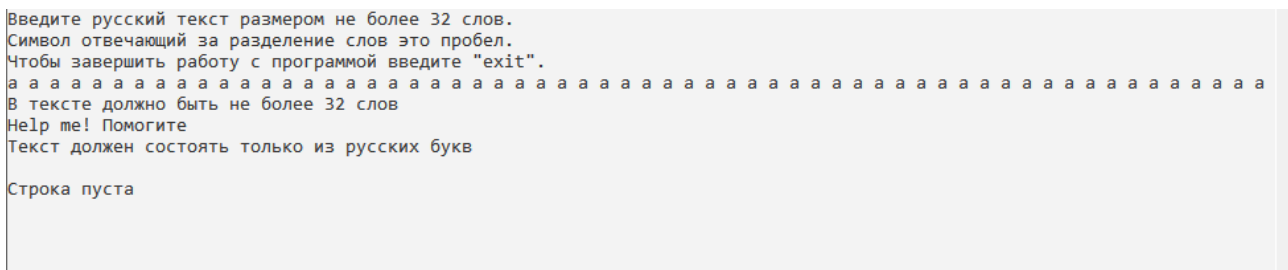


Рисунок 28 – Сообщения об ошибках (клиентская часть)

Если текст соответствует требованиям, то он будет отправлен на сервер.

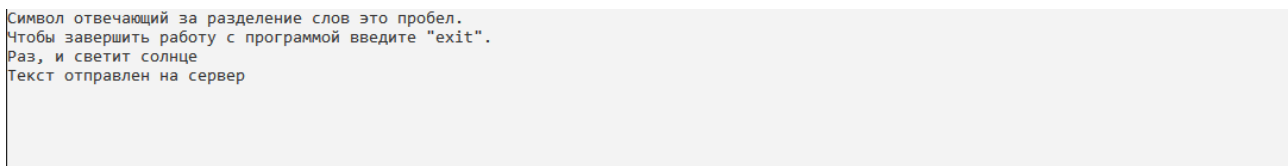


Рисунок 29 – Текст успешно отправлен на сервер

Если файла *Statistics.dat* не существовало, то он будет создан.

Statistics.dat		
1	a	: 1
2	b	: 0
3	v	: 1
4	g	: 0
5	d	: 0
6	e	: 2
7	yo	: 0
8	zh	: 0
9	z	: 1
10	i	: 2
11	j	: 0
12	k	: 0
13	l	: 1
14	m	: 0
15	n	: 1
16	o	: 1
17	p	: 0
18	r	: 1
19	s	: 2
20	t	: 2
21	u	: 0
22	f	: 0
23	x	: 0
24	c	: 1
25	ch	: 0
26	sh	: 0
27	shh	: 0
28	'	: 0
29	y	: 0
30	'	: 0
31	e'	: 0
32	yu	: 0
33	ya	: 0

Рисунок 30 – Файл *Statistics.dat*

Клиент будет сообщать о каждом удачно отправленном тексте, а сервер будет сообщать, когда он получил текст и когда закончил обработку.

```

Обработка запроса завершена
Получен запрос от клиента
Обработка запроса завершена
Получен запрос от клиента
Обработка запроса завершена
Получен запрос от клиента
Обработка запроса завершена

```

```

Выбрать E:\Da\Client.exe
Введите IP-адрес сервера (по умолчанию: 127.0.0.1) (exit чтобы выйти):

Соединение установлено!
Введите русский текст размером не более 32 слов.
Символ отвечающий за разделение слов это пробел.
Чтобы завершить работу с программой введите "exit".
Раз, и светит солнце
Текст отправлен на сервер
Два, и светит солнце
Текст отправлен на сервер
Три, небесный свет приходит в дали
Текст отправлен на сервер
Так, и светит солнце
Текст отправлен на сервер

```

Рисунок 31 – Множество запросов клиента

Если файл *Statistics.dat* уже существовал, то он будет дополнен новыми данными.

```

Statistics.dat
1  a : 4
2  b : 1
3  v : 6
4  g : 0
5  d : 3
6  e : 9
7  yo : 0
8  zh : 0
9  z : 1
10 i : 10
11 j : 1
12 k : 1
13 l : 4
14 m : 0
15 n : 5
16 o : 4
17 p : 1
18 r : 3
19 s : 8
20 t : 10
21 u : 0
22 f : 0
23 x : 1
24 c : 3
25 ch : 0
26 sh : 0
27 shh : 0
28 ' : 0
29 y : 1
30 ' : 0
31 e' : 0
32 yu : 0
33 ya : 0

```

Рисунок 32– Файл *Statistics.dat* после нескольких запросов.

2 Охрана труда

Появление современных компьютеров произвело техническую революцию во всех сферах жизни человека.

Главный недостаток современных компьютеров заключается в их негативном влиянии на здоровье человека. Для минимизации негативного влияния компьютера были разработаны требования по безопасности, при работе с ПЭВМ.

В состав данных требований входят следующие пункты:

- ПЭВМ должны соответствовать требованиям настоящих санитарных правил, и каждый их тип подлежит санитарноэпидемиологической экспертизе с оценкой в испытательных лабораториях, аккредитованных в установленном порядке;
- допустимые уровни звукового давления и уровней звука, создаваемых ПЭВМ, не должны превышать установленных норм;
- временные допустимые уровни электромагнитных полей, создаваемых ПЭВМ, не должны превышать установленных норм;
- концентрации вредных веществ, выделяемых ПЭВМ в воздух помещений, не должны превышать предельно допустимых концентраций, установленных для атмосферного воздуха;
- мощность экспозиционной дозы мягкого рентгеновского излучения в любой точке на расстоянии 0,05 м от экрана и корпуса ВДТ (на электроннолучевой трубке) при любых положениях регулировочных устройств не должна превышать 1 мкЗв/час (100 мкР/час);
- документация на проектирование, изготовление и эксплуатацию ПЭВМ не должна противоречить требованиям настоящих Санитарных правил.

Также необходимо чтобы используемое ПЭВМ, поддерживало возможность поворота корпуса и его фиксации в установленном положении. Расцветка ПЭВМ должна состоять из спокойных цветов, не отвлекающих

					ПГУ1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		41

внимание. При организации условий труда также следует учитывать, требования к освещённости рабочего места сотрудника:

- помещения, предназначенные для размещения рабочих мест пользователей персональных компьютеров, должны иметь естественное и искусственное освещение;

- естественное освещение помещений, предназначенных для размещения рабочих мест, должно осуществляться через световые проемы, ориентированные преимущественно на север и северо-восток;

- в случае иной ориентации световых проемов, необходимо предусматривать эффективные средства регулирования интенсивности естественного освещения;

- оконные проемы в помещениях, предназначенных для использования компьютеров, необходимо оборудовать устройствами регулирования интенсивности естественного освещения типа жалюзи, занавесей, внешних козырьков и др. Уровень шума в рабочем помещении, также регулируется санитарными нормами. Зачастую главным источником шума в рабочем помещении является офисное оборудование. Для снижения подобного шума необходимо предусмотреть специализированные амортизирующие прокладки, предусмотренные нормативными документами. Уровень шума на рабочих местах не должен превышать 50 дБА.

Соблюдение описанных ранее мер является обязательным требованием для любого рабочего места, так как оно позволяет снизить уровень негативного влияния на здоровье человека при работе с ПЭВМ.

Заключение

Результатом выпускной квалификационной работы является узкоспециализированное консольное клиент-серверное приложение, производящее анализ текстовой информации. Приложение подходит для слабых и не имеющих доступ к интернету машин.

За время работы с данным проектом были изучены новые методы и способы создания приложений. Значительно улучшен навык программирования на языке C++ в среде разработки *Visual Studio*.

					ПГУ1.090301.16.001 ПЗ	Лист
						43
Изм.	Лист	№ докум	Подпись	Дата		

Список использованных источников

1. text.ru: [Электронный ресурс]. URL: <https://text.ru/> (дата обращения: 26.04.2023)
2. advego.com: [Электронный ресурс]. URL: <https://advego.com/> (дата обращения: 26.04.2023)
3. glvrd.ru: [Электронный ресурс]. URL: <https://glvrd.ru/> (дата обращения: 26.04.2023)
4. wikipedia.org: [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/> (дата обращения: 26.04.2023)
5. skillfactory.ru: [Электронный ресурс]. URL: <https://blog.skillfactory.ru/> (дата обращения: 26.04.2023)

Приложение А

(обязательное)

Потоковый клиент-серверный анализатор текстовой информации

Листинг программы

					ПГУ1.090301.16.001 ПЗ	Лист
						45
Изм.	Лист	№ докум	Подпись	Дата		

Client.cpp

```
#include <iostream>
#include <ws2tcpip.h>
#include <string>
#include <thread>

#pragma comment (lib, "ws2_32.lib")
#pragma warning(disable: 6031);
using namespace std;

string cleanStringFromZnaki(string text)
{
    char previous = ' ';
    for (int i = 0; i < text.length(); i)
    {
        if (previous == text[i] && text[i] == 32 || text[i] > 32
        && !isalnum(text[i]) || text[i] >= 48 && text[i] <= 57 || text[i] <
        -64 && text[i] != -72 && text[i] != -88)
        {
            text.erase(i, 1);
        }
        else
        {
            previous = text[i];
            i++;
        }
    }
    return text;
}

bool bazarPoNashemu(string text)
{
    for (int i = 0; i < text.length(); i++)
    {
```

```

        if (text[i] == 32 || text[i] < 0 && text[i] >= -64 ||
text[i] == -72 || text[i] == -88)
        {

        }
        else
            return false;
    }
    return true;
}

bool countWords(string text)
{
    int count = 0;
    char previous = ' ';
    for (int i = 0; i < text.length(); i++)
    {
        if (text[i] != ' ' && previous == ' ')
        {
            count++;
        }
        previous = text[i];
    }
    if (count > 32)
    {
        return false;
    }
    return true;
}

string transliterate(string text)
{
    string trText;
    for (; !text.empty(); )
    {

```

```

switch (text[0])
{
case ' ': trText.append(" "); break;
case 'А':
case 'а': trText.append("a"); break;
case 'Б':
case 'б': trText.append("b"); break;
case 'В':
case 'в': trText.append("v"); break;
case 'Г':
case 'г': trText.append("g"); break;
case 'Д':
case 'д': trText.append("d"); break;
case 'Е':
case 'е': trText.append("e"); break;
case 'Ё':
case 'ё': trText.append("yo"); break;
case 'Ж':
case 'ж': trText.append("zh"); break;
case 'З':
case 'з': trText.append("z"); break;
case 'И':
case 'и': trText.append("i"); break;
case 'Й':
case 'й': trText.append("j"); break;
case 'К':
case 'к': trText.append("k"); break;
case 'Л':
case 'л': trText.append("l"); break;
case 'М':
case 'м': trText.append("m"); break;
case 'Н':
case 'н': trText.append("n"); break;
case 'О':
case 'о': trText.append("o"); break;

```

```

case 'П':
case 'п': trText.append("p"); break;
case 'Р':
case 'р': trText.append("r"); break;
case 'С':
case 'с': trText.append("s"); break;
case 'Т':
case 'т': trText.append("t"); break;
case 'У':
case 'у': trText.append("u"); break;
case 'Ф':
case 'ф': trText.append("f"); break;
case 'Х':
case 'х': trText.append("x"); break;
case 'Ц':
case 'ц': trText.append("c"); break;
case 'Ч':
case 'ч': trText.append("ch"); break;
case 'Ш':
case 'ш': trText.append("sh"); break;
case 'Щ':
case 'щ': trText.append("shh"); break;
case 'Ъ':
case 'ъ': trText.append("'"); break;
case 'Ы':
case 'ы': trText.append("y"); break;
case 'Ь':
case 'ь': trText.append(""); break;
case 'Э':
case 'э': trText.append("e"); break;
case 'Ю':
case 'ю': trText.append("yu"); break;
case 'Я':
case 'я': trText.append("ya"); break;
}

```

```

        text.erase(0, 1);
    }
    return trText;
}

string parseStringToWords(string text)
{
    short int i = 0;
    string textb;
    string words[32];
    short int cwords[32]{};
    while (!text.empty())
    {
        int pos = 0;
        string tempStr;
        bool isDup = false;
        pos = text.find(' ');
        if (pos != string::npos)
        {
            tempStr = text.substr(0, pos);
            text.erase(0, pos+1);
        }
        else
        {
            tempStr = text;
            text.clear();
        }
        for (i = 0; !words[i].empty(); i++)
        {
            if (words[i] == tempStr)
            {
                isDup = true;
                break;
            }
        }
    }
}

```

```

        if (isDup)
        {
            cwords[i]++;
        }
        else
        {
            words[i] = tempStr;
            cwords[i]++;
        }
    }
    i = 0;
    while (!words[i].empty())
    {
        textb.append(words[i]);
        textb.append(":");
        textb.append(to_string(cwords[i]));
        textb.append(",");
        i++;
    }
    return textb;
}

class secondPart {
public:
    WSADATA wsd; // структура для WSStartup
    SOCKET s1;
    bool servIsWork = false;
    int startServer();
    void doMyWork(string text);
    void endWork();
};

int secondPart::startServer()
{
    sockaddr_insa; // структура данных гнезда

```

```

in_addr addr; // адрес сервера
int iRet;
string ip_address;
unsigned short port = 3030; // порт сервера
if (WSAStartup(MAKEWORD(2, 0), &wsd) != 0) //
инициализировать библиотеку WinSock
{
    cout << "Error: WSAStartup!" << endl;
    return 1;
}
s1 = socket(AF_INET, SOCK_STREAM, 0);
if (s1 == INVALID_SOCKET)
{
    cout << "Error: socket!" << endl;
    return 1;
}
cout<< "Введите IP-адрес сервера (по умолчанию: 127.0.0.1)
(exit чтобы выйти): " << endl;
getline(cin, ip_address);
if (ip_address == "exit")
    return 1;
if (ip_address.empty())
    ip_address = "127.0.0.1";
cin.clear();
fflush(stdin);
iRet = inet_pton(AF_INET, ip_address.c_str(), &addr); //
преобразовать в 32-битное число
if (iRet == 0) // при ошибке
{
    cout << "Error: address!" << endl;
    return 2;
}
sa.sin_family = AF_INET; // заполнить структуру данных гнезда
sa.sin_port = htons(port);
sa.sin_addr.s_addr = addr.S_un.S_addr;

```

```

        if (connect(s1, (sockaddr*)&sa, sizeof(sa)) == SOCKET_ERROR)
// установить соединение
    {
        cout << "Error: connect failed!" << endl;
        return 2;
    }
    else
    {
        cout<< "Соединениеустановлено!" <<endl;
        servIsWork = true;
        return 0;
    }
}

void secondPart::doMyWork(string text)
{
    text = transliterate(text);
    text = parseStringToWords(text);
    if (send(s1, text.c_str(), text.size(), 0) == SOCKET_ERROR)
    {
        cout << "Error: send!" << endl;
        servIsWork = false;
    }
    else
        cout<< "Текст отправлен на сервер" <<endl;
}

void secondPart::endWork()
{
    shutdown(s1, SD_BOTH);
    closesocket(s1);
    WSACleanup();
}

int main()

```

```

{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    string text;
    secondPart f;
    while (true)
    {
        if (int sret = f.startServer())
        {
            if (sret == 1)
            {
                break;
            }
            if (sret == 2)
            {
                continue;
            }
        }
        cout<< "Введите русский текст размером не более 32
слов." <<endl;
        cout<< "Символ отвечающий за разделение слов это
пробел." <<endl;
        cout<< "Чтобы завершить работу с программой введите
\"exit\").\" << endl;
        while (f.servIsWork)
        {
            if (text == "exit")
            {
                break;
            }
            getline(cin, text);
            if (countWords(text))
            {
                text = cleanStringFromZnaki(text);
                if (bazarPoNashemu(text))

```

```

        {
            if (!text.empty())
            {
                thread my_thread([&]() {
                    f.doMyWork(text);
                });
                my_thread.join();
                if (!f.servIsWork)
                {
                    break;
                }
            }
            else
            {
                cout << "Строка пуста" << endl;
            }
        }
        else
        {
            cout<< "Текст должен состоять только из
русских букв" <<endl;
        }
    }
    else
    {
        cout<< "В тексте должно быть не более 32 слов"
<<endl;
    }
}

f.endWork();
cout<< "Работа программы завершена" <<endl;
cin.ignore();
return 0;
}

```

Server.cpp

```
#include <iostream>
#include <ws2tcpip.h>
#include <string>
#include <fstream>

#pragma comment (lib, "ws2_32.lib")
#pragma warning(disable: 6031);
using namespace std;

void writeToFile(string text, short int count)
{
    short int i = 0;
    fstream files;
    string liters[33]{ "a  :", "b  :", "v  :", "g  :", "d  :",
    "e  :", "yo :", "zh :", "z   :", "i   :", "j   :", "k   :",
    "l   :", "m   :", "n   :",
    "o   :", "p   :", "r   :", "s   :", "t   :", "u   :", "f
    :", "x   :", "c   :", "ch  :", "sh  :", "shh:", "'", "y   :",
    "'   :", "e'  :", "yu  :", "ya  : " };
    unsigned long long int stat[33]{ 0 };
    files.open("Statistics.dat", ios_base::in);
    if (!files.is_open())
    {
        cout<< "Не удалось открыть или создать файл статистики"
<<endl;
    }
    else
    {
        while (!files.eof())
        {
            char buf[25]{};
            files.getline(buf, 25);
            for (short int j = 0; j < 4; j++)
            {
```

```

        buf[j] = ' ';
    }
    if (i < 33)
    {
        stat[i] = atoll(buf);
        i++;
    }
}

files.close();
while (!text.empty())
{
    if (text.length() > 2 && text.substr(0, 3) == "shh")
    {
        stat[26] += count;
        text.erase(0, 3);
        continue;
    }
    else
    {
        if (text.length() > 1)
        {
            if (text.substr(0, 2) == "yo")
            {
                stat[6] += count;
                text.erase(0, 2);
                continue;
            }
            else if (text.substr(0, 2) == "zh")
            {
                stat[7] += count;
                text.erase(0, 2);
                continue;
            }
            else if (text.substr(0, 2) == "ch")
            {

```

```

        stat[24] += count;
        text.erase(0, 2);
        continue;
    }
    else if (text.substr(0, 2) == "sh")
    {
        stat[25] += count;
        text.erase(0, 2);
        continue;
    }
    else if (text.substr(0, 2) == "'")
    {
        stat[27] += count;
        text.erase(0, 2);
        continue;
    }
    else if (text.substr(0, 2) == "e'")
    {
        stat[30] += count;
        text.erase(0, 2);
        continue;
    }
    else if (text.substr(0, 2) == "yu")
    {
        stat[31] += count;
        text.erase(0, 2);
        continue;
    }
    else if (text.substr(0, 2) == "ya")
    {
        stat[32] += count;
        text.erase(0, 2);
        continue;
    }
}

```

```

switch (text[0])
{
case 'a': i = 0; break;
case 'b': i = 1; break;
case 'v': i = 2; break;
case 'g': i = 3; break;
case 'd': i = 4; break;
case 'e': i = 5; break;
case 'z': i = 8; break;
case 'i': i = 9; break;
case 'j': i = 10; break;
case 'k': i = 11; break;
case 'l': i = 12; break;
case 'm': i = 13; break;
case 'n': i = 14; break;
case 'o': i = 15; break;
case 'p': i = 16; break;
case 'r': i = 17; break;
case 's': i = 18; break;
case 't': i = 19; break;
case 'u': i = 20; break;
case 'f': i = 21; break;
case 'x': i = 22; break;
case 'c': i = 23; break;
case 'y': i = 28; break;
case '\': i = 29; break;
}

stat[i] += count;
text.erase(0, 1);
}

files.open("Statistics.dat", ios_base::out);
if (!files.is_open())
{
    cout<< "Не удалось открыть или создать файл статистики"
<<endl;

```

```

        return;
    }
    for (i = 0; i < 33; i++)
    {
        string temp = "";
        temp.append(liters[i]);
        temp.append(to_string(stat[i]));
        files.write(temp.c_str(), temp.length());
        if (i < 32)
        {
            files.write("\n", 1);
        }
    }
    files.close();
}

void inputProc(string text)
{
    short int i = 0;
    string words[32];
    short int cwords[32]{};
    while (!text.empty())
    {
        int pos = 0;
        string tempStr;
        pos = text.find(',');
        tempStr = text.substr(0, pos);
        text.erase(0, pos + 1);
        pos = tempStr.find(':');
        words[i] = tempStr.substr(0, pos);
        cwords[i] = stoi(tempStr.substr(pos + 1,
tempStr.size()));
        i++;
    }
    if (i < 2)

```

```

    {
        cout<< "Ошибка: Количество отличающихся друг от друга
слов должно быть не менее 2" <<endl;
        return;
    }
    while (i >= 0)
    {
        writeToFile(words[i], cwords[i]);
        i--;
    }
}

int main()
{
    SetConsoleOutputCP(1251);
    SetConsoleCP(1251);
    SOCKET server, client;
    WSADATA wsd;
    string text;
    unsigned short port = 3030;
    string cmd;
    sockaddr_in sa, ca;
    chartextBuf[2048] = { 0 }; //Создать буфер для данных
    while (true)
    {
        if ((WSAStartup(MAKEWORD(2, 0), &wsd)) != 0)
        {
            cout << "Error: WSAStartup!" << endl;
            break;
        }
        if ((server = socket(AF_INET, SOCK_STREAM, 0)) ==
INVALID_SOCKET)
        {
            cout << "Error: socket!" << endl;
            break;

```

```

    }
    sa.sin_family = AF_INET; // заполнение структуры данных
    sa.sin_port = htons(port);
    sa.sin_addr.s_addr = htonl(INADDR_ANY);
    if (bind(server, (sockaddr*)&sa, sizeof(sa)) ==
SOCKET_ERROR)
    {
        cout << "Error: bind!" << endl;
        break;
    }
    if (listen(server, 1) == SOCKET_ERROR)
    {
        cout << "Error: listen!" << endl;
        break;
    }
    cout<< "Ожидание входящего соединения" <<endl;
    intclientAddrSize = sizeof(ca); //Инициализировать адрес
клиента
    if ((client = accept(server, (SOCKADDR*)&ca,
&clientAddrSize)) == INVALID_SOCKET) {
        cout << "Error: accept!" << endl;
        break;
    }
    cout << "Client connected!" << endl;
    while (true)
    {
        if (recv(client, textBuf, sizeof(textBuf), 0) ==
SOCKET_ERROR) {
            cout << "recv function failed with error " <<
WSAGetLastError() << endl;
            break;
        }
        if (client == INVALID_SOCKET) { //Если клиент
отсоединился
            cout << "Client Disconnected." << endl;

```

```

        break;
    }
    text = textBuf;
    if (text.empty())
    {
        cout<< "Клиент разорвал соединение" <<endl;
        break;
    }
    memset(textBuf, 0, sizeof(textBuf)); //Очистить
буфер

    cout<< "Получен запрос от клиента" <<endl;
    inputProc(text);
    cout<< "Обработка запроса завершена" <<endl;
}
break;
}
shutdown(server, SD_BOTH);
shutdown(client, SD_BOTH);
closesocket(client);
closesocket(server);
WSACleanup();
cout<< "Работа программы завершена" <<endl;
cin.ignore();
return 0;
}

```

Приложение Б

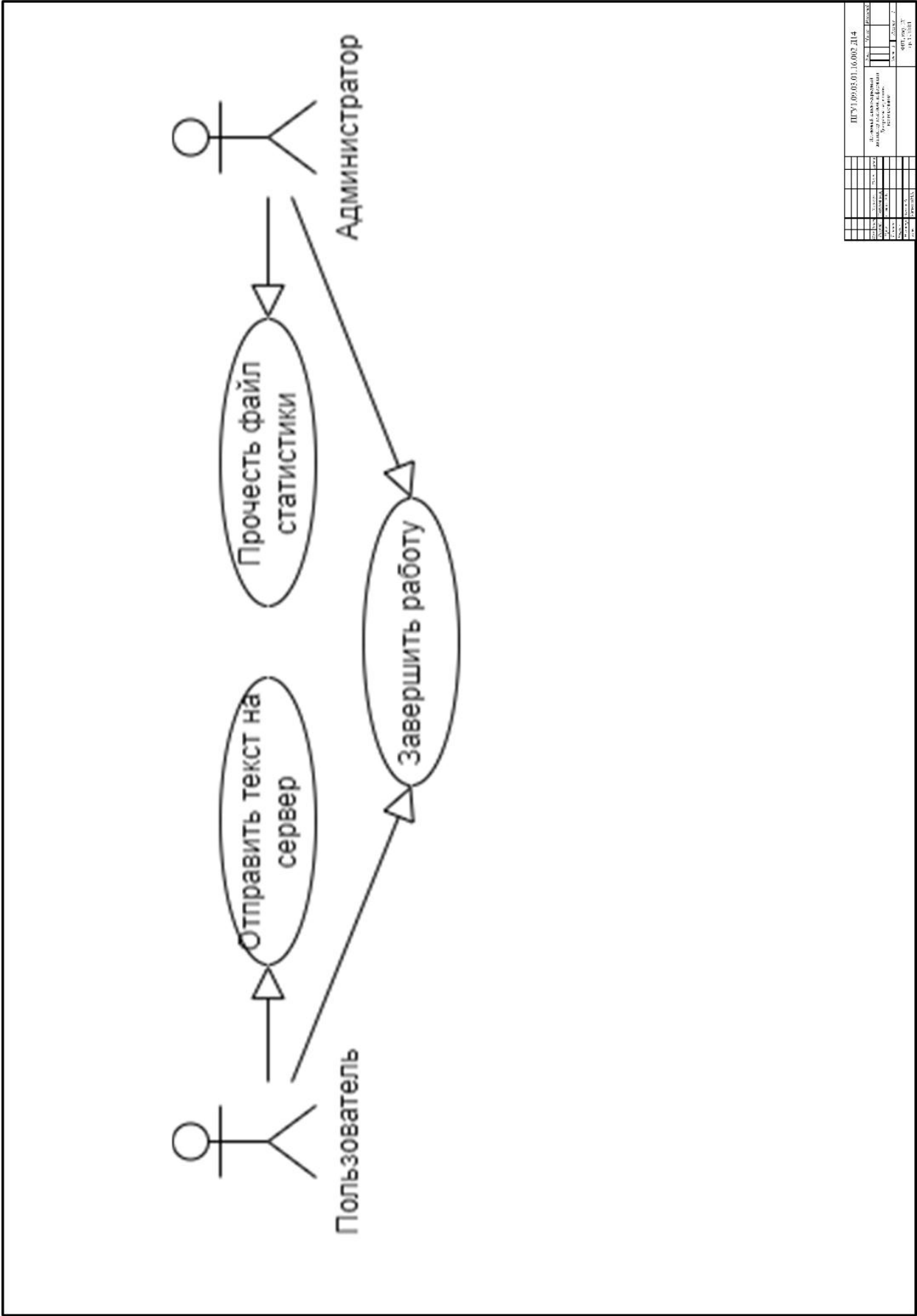
(обязательное)

Потоковый клиент-серверный анализатор текстовой информации

UML-диаграммы

					ПГУ1.090301.16.001 ПЗ	Лист
						64
Изм.	Лист	№ докум	Подпись	Дата		

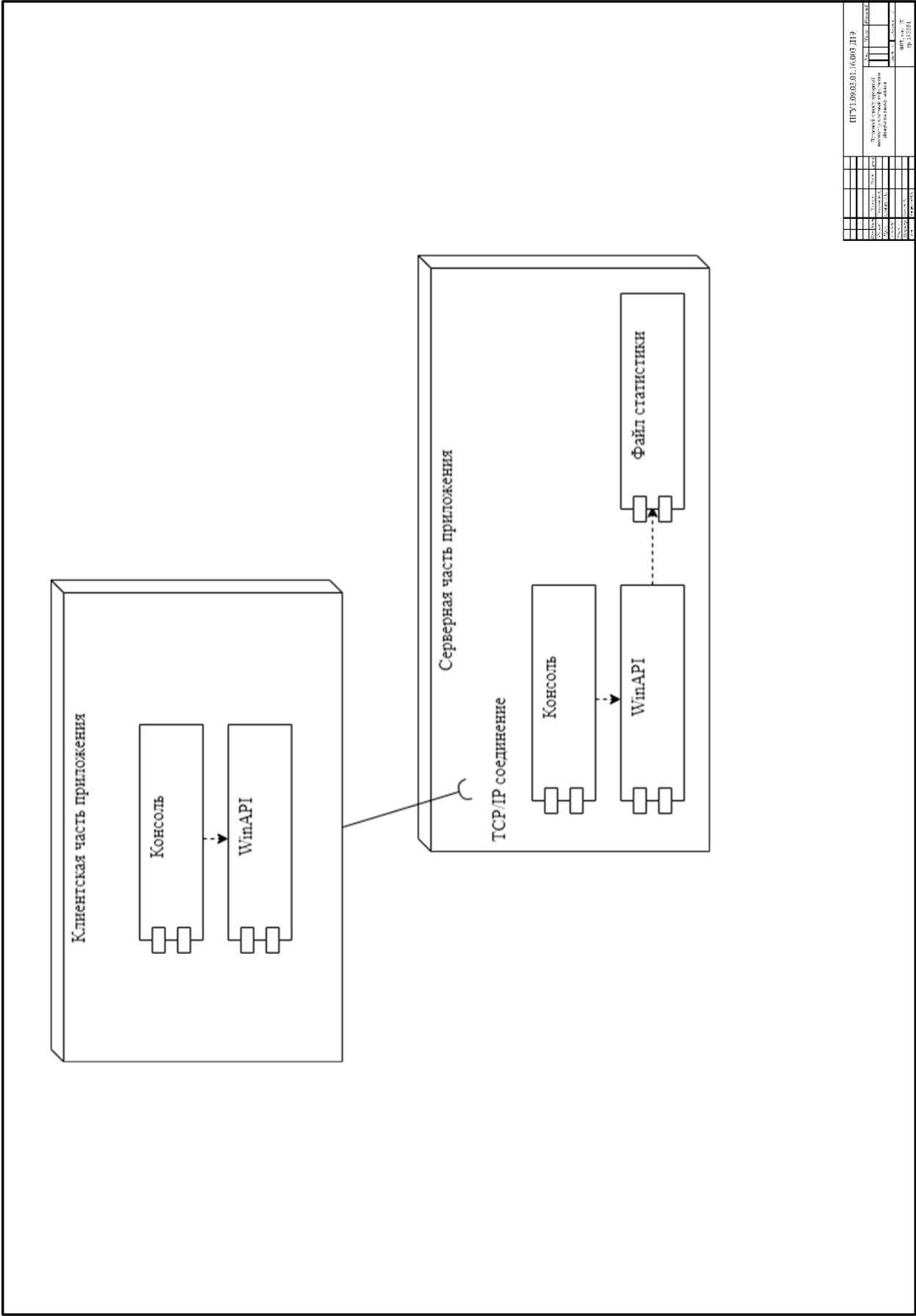
Приложение Б.1 – Диаграмма вариантов использования



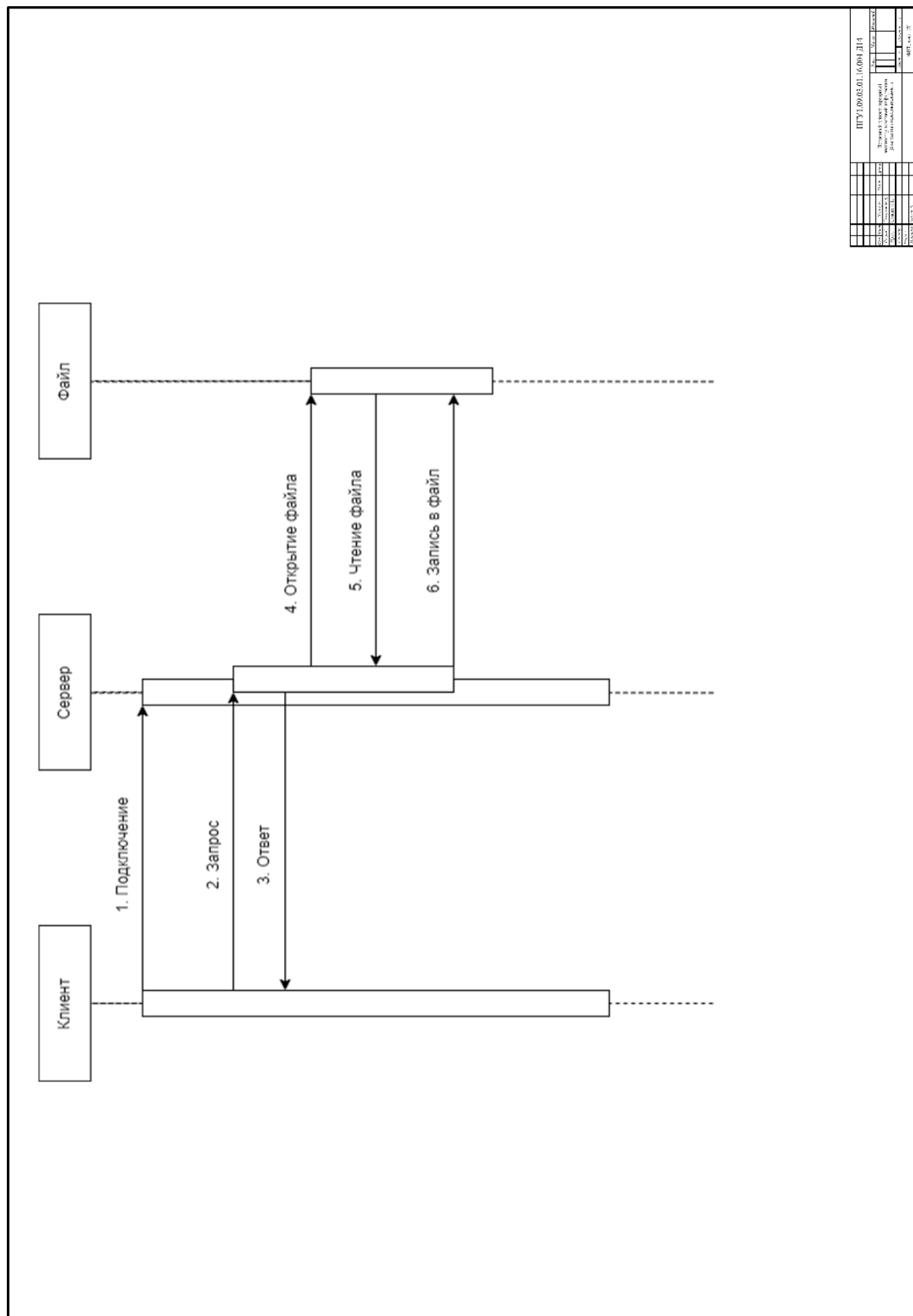
ПГУ1.09.03.01.16.002 Л14			
Изм.	Лист	№ докум	Подпись
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7
8	8	8	8
9	9	9	9
10	10	10	10
11	11	11	11
12	12	12	12
13	13	13	13
14	14	14	14
15	15	15	15
16	16	16	16
17	17	17	17
18	18	18	18
19	19	19	19
20	20	20	20
21	21	21	21
22	22	22	22
23	23	23	23
24	24	24	24
25	25	25	25
26	26	26	26
27	27	27	27
28	28	28	28
29	29	29	29
30	30	30	30
31	31	31	31
32	32	32	32
33	33	33	33
34	34	34	34
35	35	35	35
36	36	36	36
37	37	37	37
38	38	38	38
39	39	39	39
40	40	40	40
41	41	41	41
42	42	42	42
43	43	43	43
44	44	44	44
45	45	45	45
46	46	46	46
47	47	47	47
48	48	48	48
49	49	49	49
50	50	50	50
51	51	51	51
52	52	52	52
53	53	53	53
54	54	54	54
55	55	55	55
56	56	56	56
57	57	57	57
58	58	58	58
59	59	59	59
60	60	60	60
61	61	61	61
62	62	62	62
63	63	63	63
64	64	64	64
65	65	65	65
66	66	66	66
67	67	67	67
68	68	68	68
69	69	69	69
70	70	70	70
71	71	71	71
72	72	72	72
73	73	73	73
74	74	74	74
75	75	75	75
76	76	76	76
77	77	77	77
78	78	78	78
79	79	79	79
80	80	80	80
81	81	81	81
82	82	82	82
83	83	83	83
84	84	84	84
85	85	85	85
86	86	86	86
87	87	87	87
88	88	88	88
89	89	89	89
90	90	90	90
91	91	91	91
92	92	92	92
93	93	93	93
94	94	94	94
95	95	95	95
96	96	96	96
97	97	97	97
98	98	98	98
99	99	99	99
100	100	100	100

Изм.	Лист	№ докум	Подпись	Дата

Приложение Б.2 – Диаграмма разворачивания



Приложение Б.3 – Диаграмма последовательности



Изм.	Лист	№ докум	Подпись	Дата

ПГУ1.090301.16.001 ПЗ

Лист

67

Приложение В

(обязательное)

Потоковый клиент-серверный анализатор текстовой информации

Презентация

					ПГУ1.090301.16.001 ПЗ	Лист
						68
Изм.	Лист	№ докум	Подпись	Дата		

Потоковый клиент-серверный анализатор текстовой информации

Автор работы:

студент гр. 19ВВВ1 Петухов Максим Дмитриевич

Научный руководитель:

к.т.н., доцент Слепцов Николай Владимирович

Пенза, 2023

Изм.	Лист	№ докум	Подпись	Дата

ПГУ1.090301.16.001 ПЗ

Лист

69

Цель работы и объекты

Цель работы - разработка текстового анализатора, на базе клиент серверного приложения.

Объектом проекта является клиент-серверное приложение, выполняющее следующие функции:

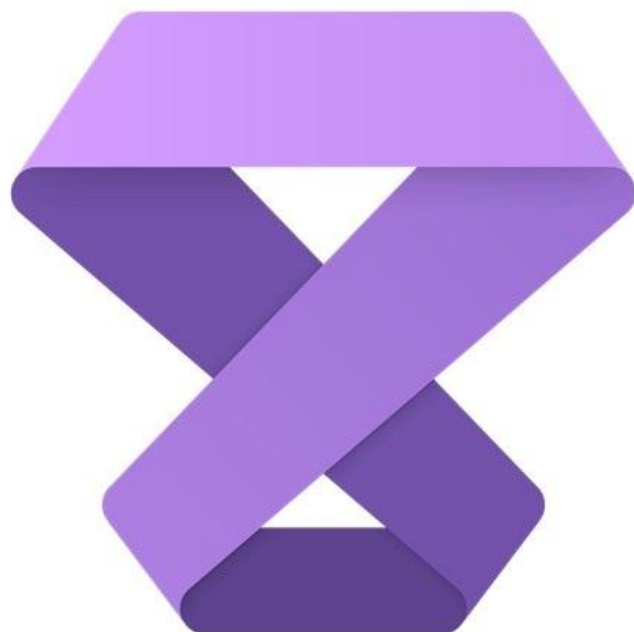
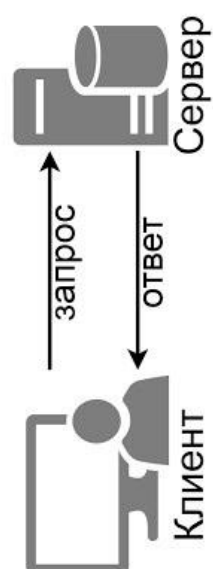
- Получение текста от пользователя;
- Анализ и преобразование текста;
- Передача данных от программы-клиента программе серверу посредством API;
- Запись обработанного текста в файл статистики.

Зачем нужны анализаторы текста?

Программные анализаторы текста существуют и используются в двух вариантах:

- 1) В чистом виде:
только обработка и преобразование текста без последующего использования в последующей аналитике.
- 2) Как часть систем аналитики текстовой и иной информации:
 - В нейросетях;
 - В системах антиплагиата;
 - В поисковиках и т. д.

Используемые технологии



Архитектура приложения

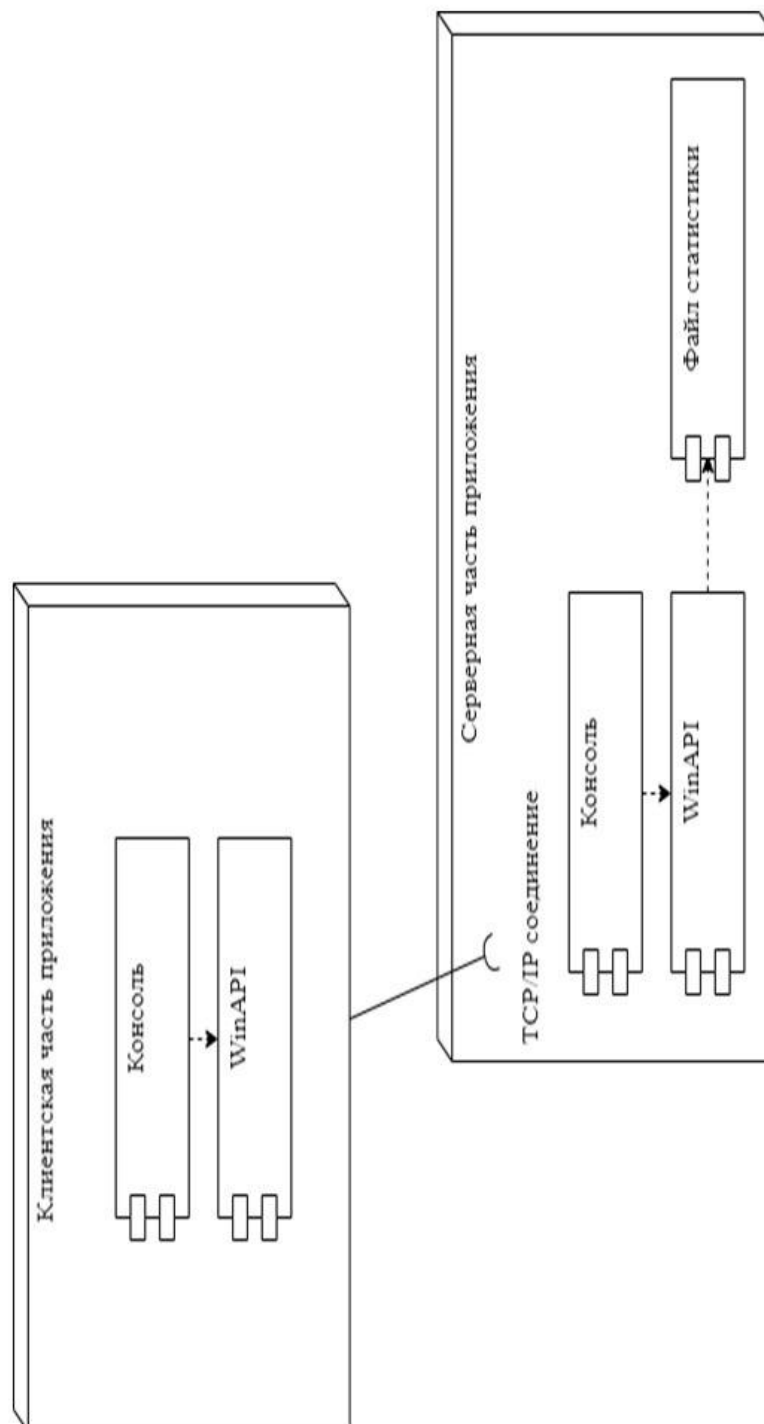


Диаграмма взаимодействия слоёв приложения

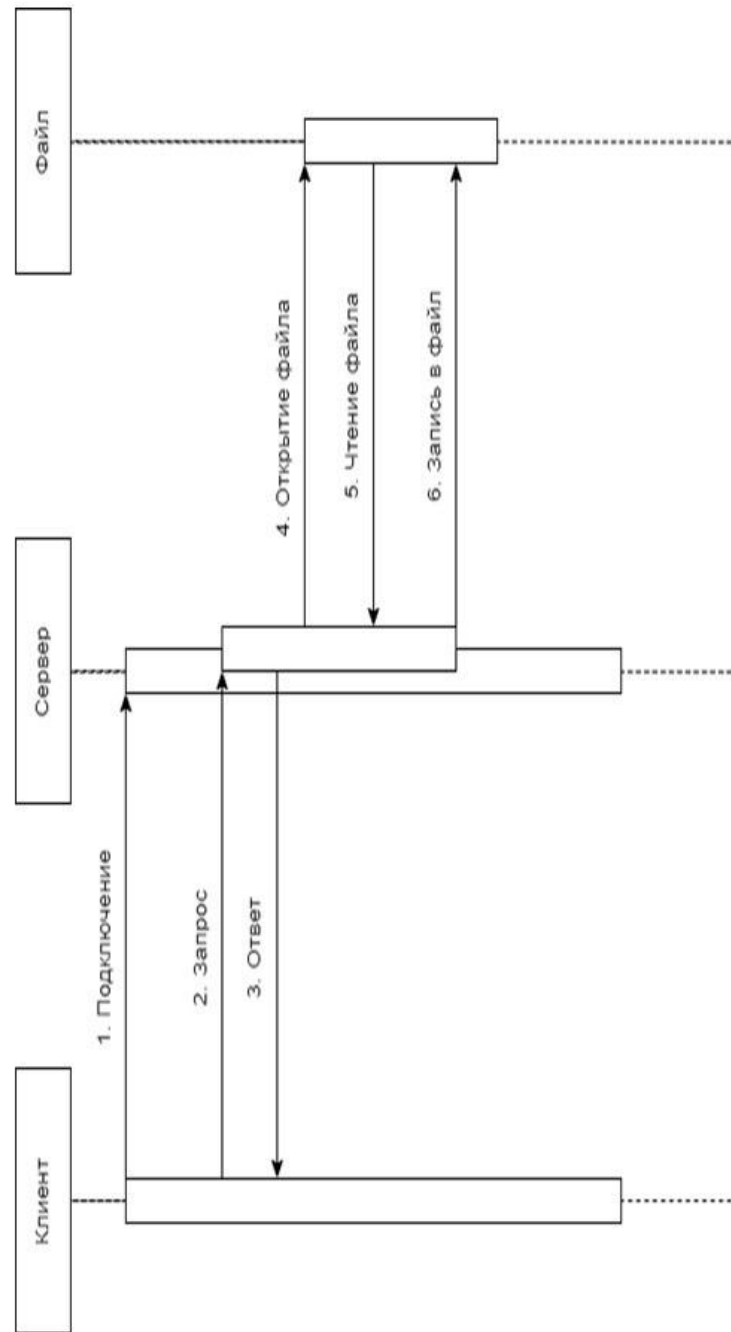
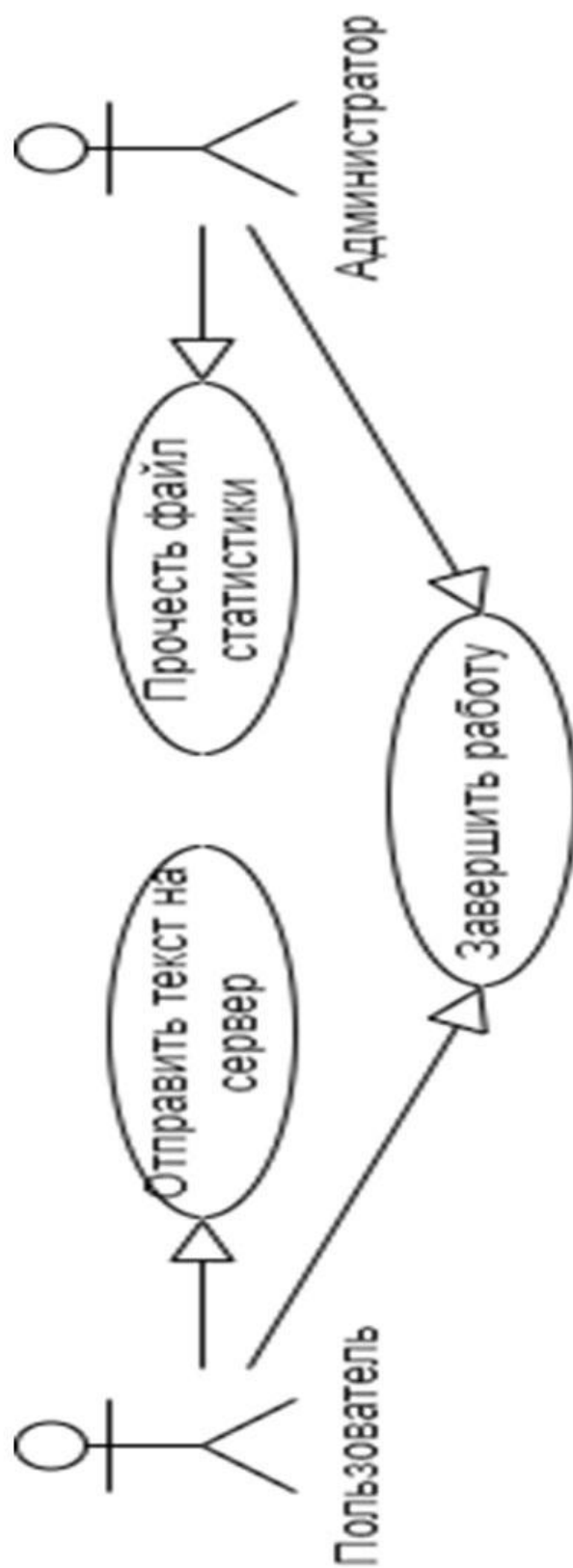
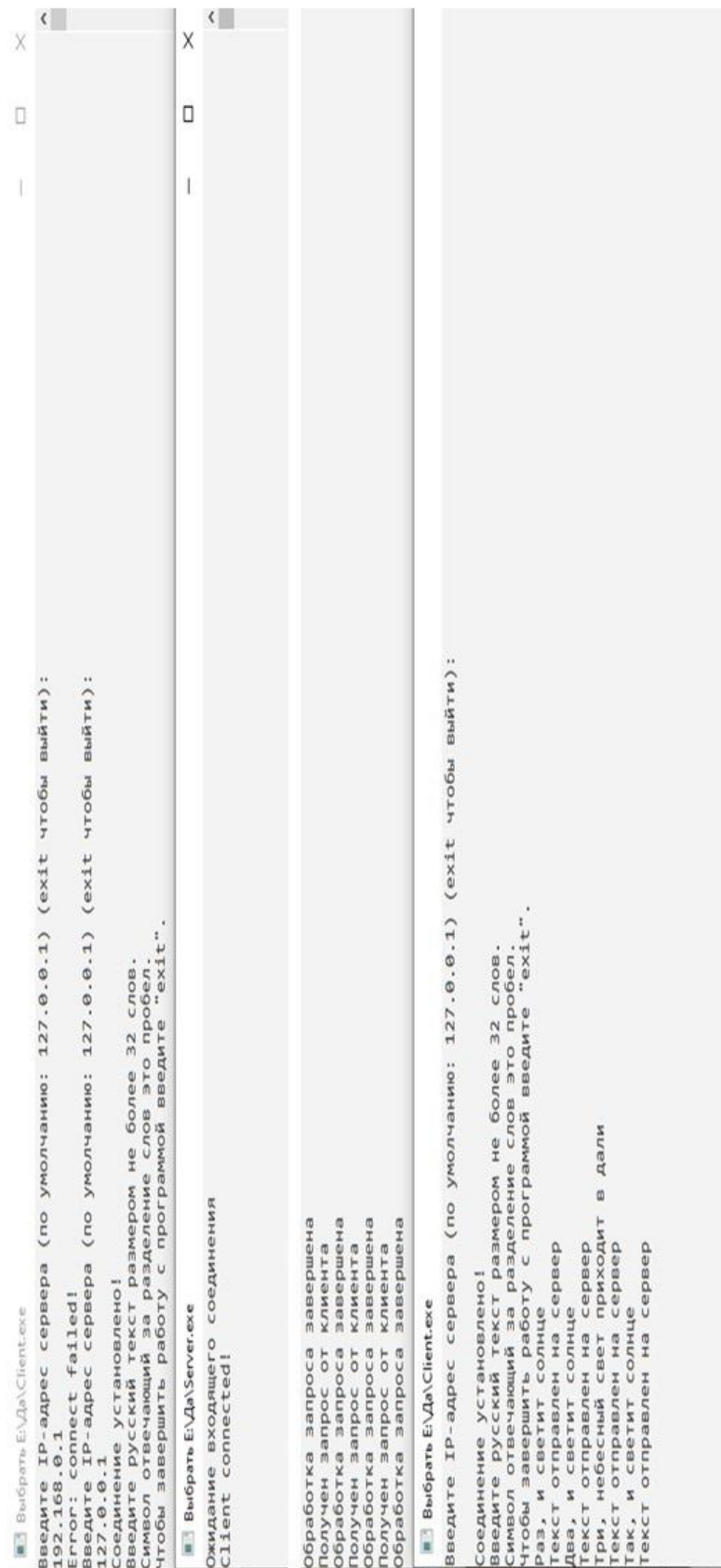


Диаграмма вариантов использования



Скриншоты работы приложения



Результаты работы

- Разработано приложение в соответствии с техническим заданием;
- Проведено функциональное тестирование приложения;
- Намечены шаги по модернизации приложения;

					ПГУ1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		78

Спасибо за внимание!

					ПГУ1.090301.16.001 ПЗ	Лист
Изм.	Лист	№ докум	Подпись	Дата		79