

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: «Программное обеспечение средств ВТ и
автоматизированных систем»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

БАКАЛАВРА

на тему

**ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДЛЯ
ПЕРЕДАЧИ ИНФОРМАЦИИ СЕТЕВЫМ
СПОСОБОМ**

Студент

 31.05.2022
(подпись, дата)

Немков А.М.

Руководитель

 31.05.2022
(подпись, дата)

Дубравин А.В.

Нормоконтролёр

 31.05.2022
(подпись, дата)

Бычков А.С.

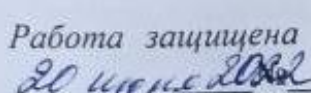
Работа допущена к защите (протокол зас. кафедры от 02.06.2022 г. № 13)

Заведующий кафедрой


(подпись)

М.А. Митрохин

Работа защищена с отметкой

 15

(протокол заседания ГЭК от

Секретарь ГЭК


(подпись)

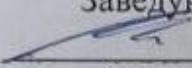
Е.И. Гурин

Пенза 2022

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«30» апреля 2022 г.

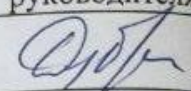
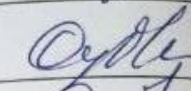
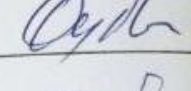
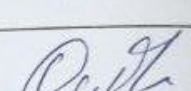
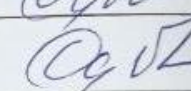
**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВАРА**

1. Студент Немков Артем Михайлович, гр.18ВВ2 факультета вычислительной техники.
2. Тема работы: «Программное обеспечение для передачи информации сетевым способом»
Тема утверждена приказом ПГУ №357/о от "26" апреля 2022 г.
3. Руководитель бакалаврской работы к.т.н., доцент каф. ВТ ПГУ Дубравин А.В.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

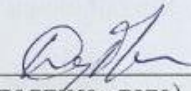
Разработать приложение «Программное обеспечение для передачи информации сетевым способом», позволяющее пользователям взаимодействовать между собой посредством обмена текстовой и звуковой информации. Приложение должно быть спроектировано с применением клиент-серверной архитектуры. Клиентская часть программы должна обладать графическим интерфейсом. Клиентская программа должна включать в себя функционал приёма данных от пользователя, их обработку и передачу на сервер, а также приём и отображение данных, полученных с сервера. Серверная часть программы должна производить обработку данных, полученных от клиентских приложений, и отправку обработанных данных клиентам.

5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметных областей: изучить аналоги существующих программ для передачи звуковой и текстовой информации, изучить способы передачи информации по сети, изучить методики проектирования и разработки клиент-серверных приложений.

6. Календарный график выполнения работы				Подпись руководителя
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	
		50	26.04.2022 – 30.04.2022	
1.	Изучение предметной области	10	01.05.2022 – 03.05.2022	
2.	Анализ требований к работе	50	04.05.2022 – 08.05.2022	
3.	Выбор технологий, применяемых при разработке.	40	09.05.2022 – 12.04.2022	
4.	Описание алгоритмов	150	13.04.2022 – 27.05.2022	
5.	Разработка приложения «Программное обеспечение для передачи информации сетевым способом»	15	28.05.2022 – 29.05.2022	
6.	Выполнение задания по БЖД	24	30.05.2022 – 31.05.2022	
7.	Оформление пояснительной записки			

Дата выдачи задания "26" апреля 2022 г.

Руководитель бакалаврской работы  26.04.22 Дубравин А.В.
(подпись, дата)

Задание к исполнению принял студент  26.04.22 Немков А.М.
(подпись, дата)

Работу к защите допустить
Декан факультета  17.06.2022 Фионова Л.Р.
(подпись, дата)

Реферат

Работа содержит 102 листа, 27 рисунков, 1 таблицу, 5 источников, 3 приложения.

ПЕРЕДАЧА ИНФОРМАЦИИ ПО СЕТИ, ОБРАБОТКА ЗВУКОВОЙ ИНФОРМАЦИИ, ЗАПИСЬ И ВОСПРОИЗВЕДЕНИЕ ЗВУКА, КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА.

Цель ВКР – разработать приложение «Программное обеспечение для передачи информации сетевым способом», позволяющее пользователям взаимодействовать между собой посредством обмена текстовой и звуковой информации.

В ходе выполнения ВКР были разработаны:

- клиентское приложение для записи звуковой и текстовой информации и её передачи на сервер, а также для приёма и вывода информации, полученной от сервера;

- серверное приложение для приёма, обработки и отправки звуковых и текстовых данных, полученных от клиентов.

В результате проделанной работы разработано клиент-серверное приложение, полностью соответствующее поставленной задаче, а именно: разработано серверное приложение, которое принимает звуковую и текстовую информацию от клиентов, обрабатывает её и отправляет клиентским приложениям, разработано клиентское приложение с графическим интерфейсом позволяющее отправлять на сервер, принимать с сервера и выводить на устройства вывода звуковую и текстовую информацию.

					ПГУ 1.090301.27.001 ПЗ		
Изм.	ЛистЛ	№ докум.№	Подпись	Дата			
Разраб.		Неликов А.М.			Лит.	Лист	Листов
Провер.		Дубравин А.В.				2	102
Н. Копр.		Бычков А.С.			ФВТ, каф. ВТ		
Утверд.		Митрохин М.А.			гр. 18ВВ2		
					Программное обеспечение для передачи информации сетевым способом		
					Пояснительная записка		

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Программное обеспечение для передачи информации сетевым способом	7
1.1 Анализ предметной области	7
1.1.1 Постановка задачи	7
1.1.2 Аналоги	8
1.1.3 Выбор архитектуры приложения	9
1.1.4 Выбор языка программирования.....	9
1.1.5 Выбор библиотеки для работы со звуком	10
1.1.6 Выбор среды разработки	11
1.1.7 Протокол взаимодействия клиента и сервера.....	11
1.2 Разработка приложения.....	18
1.2.1 Реализация серверного приложения	22
1.2.2 Реализация клиентского приложения	25
1.2.3 Тестирование	28
1.3 Анализ результатов.....	31
1.3.1 Окно серверного приложения.....	32
1.3.2 Окно клиентского приложения	37
2 Безопасность жизнедеятельности.....	44
2.1 Эргономические требования к рабочему месту.....	45
2.2 Воздействие шума на программиста.....	46
2.3 Опасность повышенного уровня напряженности электромагнитного поля.....	47
2.4 Освещение.....	50
Заключение	53

Список используемых источников.....	54
Приложение А – <i>UML</i> -диаграммы	55
Приложение Б – Листинг программы	66
Приложение В – Презентация.....	96

Перечень сокращений и обозначений

TCP/IP (Transmission Control Protocol / Internet Protocol) – сетевая модель передачи данных, представленных в цифровом виде

UML (Unified Modelling Language) – унифицированный язык моделирования

VoIP (Voice over IP) – телефонная связь по протоколу *IP*

ОС – операционная система

					<i>ПГУ 1.090301.27.001 ПЗ</i>	<i>Лист</i>
						5
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпись</i>	<i>Дата</i>		

Введение

Не так давно интернет стал частью повседневной жизни. Уже сейчас без него невозможно представить функционирование предприятий, медучреждений, государственных структур и даже образовательных учреждений. Современный интернет сейчас для многих людей является средством коммуникации между собой. Для многих он уже стал стандартным средством связи. Многие компании также стараются упростить коммуникацию между своими сотрудниками с помощью современных технологий.

Актуальность темы обусловлена нарастающей тенденцией перевода сотрудников компаний на удалённую работу. В связи с этим многие компании ищут себе корпоративные системы для общения сотрудников в сети.

Цель работы – разработка клиент-серверного приложения для голосового и текстового общения в вычислительной сети.

Необходимо разработать клиентское приложение, которое будет передавать на сервер и принимать от сервера звуковую и текстовую информацию, и серверное приложение, которое будет обрабатывать звуковые дорожки и текстовые сообщения и рассылать их клиентам.

1 Программное обеспечение для передачи информации сетевым способом

1.1 Анализ предметной области

1.1.1 Постановка задачи

Целью данной выпускной квалификационной работы является разработка клиент-серверного приложения, осуществляющего передачу текстовой и звуковой информации через вычислительную сеть.

К приложению предъявляются следующие требования:

- клиентское приложение должно принимать от пользователя текстовую или звуковую информацию и отправлять их на сервер;
- клиентское приложение должно принимать от сервера текстовую или звуковую информацию и выводить её в диалоговое поле или на устройство вывода звука соответственно;
- серверное приложение должно обеспечивать подключение нескольких клиентов;
- серверное приложение должно осуществлять приём от клиентов информации, обработку информации в соответствии с её типом и рассылку информации клиентам;
- в серверном приложении необходимо реализовать обработку двух и более звуковых потоков при одновременном разговоре клиентов;
- клиентское приложение должно иметь дружелюбный к пользователю графический интерфейс;
- в клиентском приложении необходимо реализовать возможность выбора звуковых устройств ввода и вывода;
- клиентское приложение должно иметь возможность подключения к серверу по его IP-адресу и порту;

- обмен информации между клиентской и серверной частью должен осуществляться с использованием протокола *TCP/IP*;
- операционная система *MS Windows 7/8/10/11*.

1.1.2 Аналоги

Аналогами данного проекта являются приложения для аудиоконференций, такие как *TeamSpeak* или *Discord*.

TeamSpeak – это компьютерная программа, через которую люди общаются друг с другом. Она отличается от многих популярных программ тем, что в ней люди в основном общаются группами и все их общение основано на каком-то интересе или совместной деятельности.

Например, люди, играющие в сетевые компьютерные игры, объединяются в команды и общаются между собой через Тим Спик. Внутри своей команды игроки могут договариваться о тактике игры или просто разговаривать на веселые темы, прикалываться.

В самой программе нет звонков друг другу, там есть только создание специальных каналов (или - комнат), к которым может подключиться определенное количество человек. Т.е. вы можете подключиться к каналу, на котором уже будет несколько человек, вы начнете говорить и все вас услышат. Вы также будете слышать разговор всех остальных подключенных к каналу пользователей.

Discord — это приложение для групповых аудио- и видеозвонков, осуществляемых посредством технологии *VoIP* — системы передачи голоса через интернет-протокол. *Discord* можно использовать в том числе и как обычный мессенджер: он позволяет отправлять и простые текстовые сообщения.

Мессенджер *Discord* позволяет создавать как каналы-чаты, так и отдельные серверы с несколькими каналами разной направленности, в которых администраторы серверов могут предоставлять участникам определенные роли и права доступа.

					ПГУ 1.090301.27.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

1.1.3 Выбор архитектуры приложения

Для данного проекта была выбрана архитектура «клиент-сервер». Архитектура «клиент-сервер» описывает отношения взаимодействующих программ в приложении. Серверный компонент предоставляет функцию или услугу одному, или нескольким клиентам, которые инициируют запросы на такие услуги. Серверы классифицируются по предоставляемым ими услугам. Например, веб-сервер обслуживает веб-страницы, а файловый сервер обслуживает компьютерные файлы. Общий ресурс может быть любой из программного обеспечения и электронных компонентов компьютера — сервера, от программ и данных в процессорах и запоминающих устройств. Совместное использование ресурсов сервера представляет собой услугу.

Данная архитектура является наиболее подходящей для проекта. Она позволяет создать программу клиент, множество копий которой могут одновременно подключаться к одному серверу, а также программу сервер, который занимается маршрутизацией и обработкой данных от нескольких клиентов.

1.1.4 Выбор языка программирования

Для написания программы была выбрана среда программирования *Visual Studio 2017 C#*, основанная на языке программирования C#. Данная среда выгодно отличается эффективностью и надежностью.

Для решения поставленной задачи необходимо использовать функциональную, эффективную и удобную платформу для разработки, позволяющую применять принципы объектно-ориентированного программирования. В качестве такой платформы была выбрана среда *.NET*.

Среда разработки *Visual Studio*, поставляемая вместе с *.NET*, предоставляет необходимый инструментарий для эффективного и быстрого создания приложений с графическим интерфейсом.

Появление технологии *.NET* повлекло за собой массовую реконструкцию некоторых языков программирования, стремящихся

использовать те или иные возможности платформы, такие как *C++* и *Visual Basic*. *Microsoft* решили предложить разработчикам альтернативу - язык, ориентированный специально *.NET* и создали *C#*. Сами разработчики языка описывают его, как простой, современный, объектно-ориентированный и безопасный язык программирования. Синтаксически *C#* напоминает *C++* и *Java*, что позволяет за достаточно короткое время изучить тонкости нового языка [1].

Несмотря на то, что *C#* и *.NET* предназначены в первую очередь для веб-разработки, их также активно применяют для создания приложений, которые должны устанавливаться на машине конечного пользователя, где и будет выполняться вся обработка данных. Разработку таких приложений обеспечивает библиотека *Windows Forms*, позволяющая проектировать графический интерфейс. Система, описанная в данной работе, разработана именно с помощью библиотеки *Windows Forms* [1].

Язык программирования *C#* претендует на подлинную объектную ориентированность.

Язык программирования *C#* призван реализовать компонентно-ориентированный подход к программированию, который способствует меньшей машинно-архитектурной зависимости результирующего программного кода, большей гибкости, переносимости и легкости повторного использования программ [1].

Принципиально важным отличием от предшественников являются изначальная ориентация на безопасность кода, расширенная поддержка событийно-ориентированного программирования, а также большое количество сторонних библиотек.

Язык программирования *C#* является «родным» для создания приложений в среде *Microsoft .NET*, поскольку наиболее тесно и эффективно интегрирован с ней.

1.1.5 Выбор библиотеки для работы со звуком

NAudio — это аудио инструментарий *.NET* с открытым исходным кодом, содержащий множество классов для работы с аудиофайлами и устройствами в *Windows*.

С помощью библиотеки *Naudio* можно считывать аудиопоток с микрофона, выводить звук на динамики или в наушники. Также есть возможность микшировать несколько аудиопотоков, переводить данные из звукового буфера в массив байтов и обратно, выбирать устройства ввода-вывода звука, настраивать частоту дискретизации и количество каналов записываемого звука [2].

Данная библиотека даёт все необходимые функции для работы со звуком, которые могут понадобиться при реализации данного проекта.

1.1.6 Выбор среды разработки

Microsoft Visual Studio - это версия *Visual Studio* и *.NET Framework*, которая поддерживает новые и улучшенные объекты, включает среду разработки с обновленным интерфейсом и отличается интегрированной поддержкой *Microsoft SQL Server*, позволяя создавать и развертывать проекты с применением сервера баз данных [3].

Из инструмента программиста, пишущего и отлаживающего код, *Microsoft Visual Studio*, превратилась в полноценное инструментальное средство, позволяющее автоматизировать деятельность всех членов команды, работающих над проектом.

Интерфейс *Visual Studio* традиционно выполнен в одном стиле с *MS Office*.

Имеется список задач, в который помещают информацию об ошибках и о необходимых доработках. Каждому пункту можно назначить приоритет, а после выполнения установить флажок, сообщающий о завершении указанной задачи. *Task List* поддерживает сортировку записей по тексту, по приоритету и статусу [3].

1.1.7 Протокол взаимодействия клиента и сервера

					ПГУ 1.090301.27.001 ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

Transmission Control Protocol/Internet Protocol (TCP/IP) - это промышленный стандарт стека протоколов, разработанный для глобальных сетей.

Стандарты *TCP/IP* публикуются в серии документов *RFC (Request for Comment)*. В документах *RFC* описана внутренняя работа сети *Internet*. Некоторые *RFC* описывают структуру сетевых сервисов или протоколов и их реализацию, в то время как другие обобщают условия применения. Стандарты *TCP/IP* всегда публикуются в виде документов *RFC*, но не все *RFC* определяют стандарты. [4]

Разработка стека была инициирована Министерством обороны США (*Department of Defence, DoD*) более 20 лет назад для связи экспериментальной сети *ARPAnet* с другими сателлитными сетями как набор общих протоколов для разнородной вычислительной среды. Сеть *ARPA* поддерживала разработчиков и исследователей в военных областях. В сети *ARPA* связь между двумя компьютерами осуществлялась с использованием протокола *Internet Protocol (IP)*, который и по сей день является одним из основных в стеке *TCP/IP* и фигурирует в названии стека.

Большой вклад в развитие стека *TCP/IP* внес университет Беркли, реализовав протоколы стека в своей версии ОС *UNIX*. Широкое распространение ОС *UNIX* привело и к широкому распространению протокола *IP* и других протоколов стека. На этом же стеке работает всемирная информационная сеть *Internet*, чье подразделение *Internet Engineering Task Force (IETF)* вносит основной вклад в совершенствование стандартов стека, публикуемых в форме спецификаций *RFC* [4].

Если в настоящее время стек *TCP/IP* распространен в основном в сетях с ОС *UNIX*, то реализация его в последних версиях сетевых операционных систем для персональных компьютеров (*Windows NT 3.5, NetWare 4.1, Windows 95*) является хорошей предпосылкой для быстрого роста числа установок стека *TCP/IP* [5].

					ПГУ 1.090301.27.001 ПЗ	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

Итак, лидирующая роль стека *TCP/IP* объясняется следующими его свойствами:

Это наиболее завершенный стандартный и в то же время популярный стек сетевых протоколов, имеющий многолетнюю историю.

Почти все большие сети передают основную часть своего трафика с помощью протокола *TCP/IP*.

Это метод получения доступа к сети *Internet*.

Этот стек служит основой для создания *intranet*- корпоративной сети, использующей транспортные услуги *Internet* и гипертекстовую технологию *WWW*, разработанную в *Internet*.

Все современные операционные системы поддерживают стек *TCP/IP*.

Это гибкая технология для соединения разнородных систем, как на уровне транспортных подсистем, так и на уровне прикладных сервисов.

Это устойчивая масштабируемая межплатформенная среда для приложений клиент-сервер.

Структура стека *TCP/IP* и краткая характеристика продемонстрирована на рисунке 1.1.

Так как стек *TCP/IP* был разработан до появления модели взаимодействия открытых систем *ISO/OSI*, то, хотя он также имеет многоуровневую структуру, соответствие уровней стека *TCP/IP* уровням модели *OSI* достаточно условно.

Протокол TCP/IP

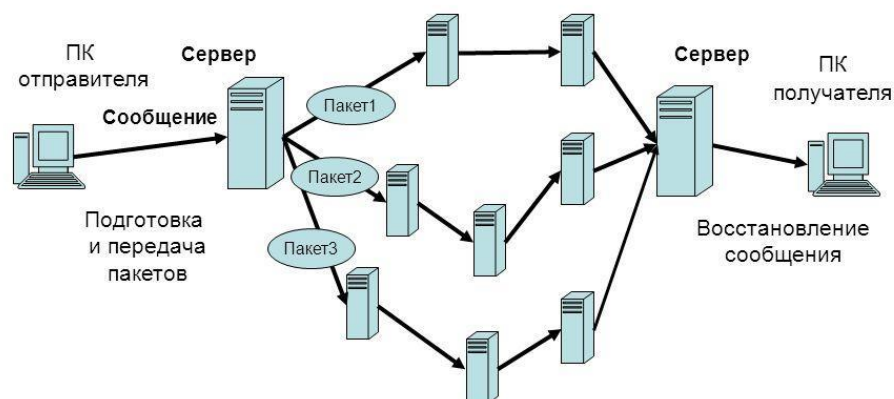


Рисунок 1.1 – Протокол TCP/IP

Структура протоколов TCP/IP. делятся на 4 уровня.

Самый нижний (уровень IV) соответствует физическому и каналному уровням модели OSI. Этот уровень в протоколах TCP/IP не регламентируется, но поддерживает все популярные стандарты физического и каналного уровня: для локальных сетей - это Ethernet, Token Ring, FDDI, Fast Ethernet, 100VG-AnyLAN, для глобальных сетей - протоколы соединений "точка-точка" SLIP и PPP, протоколы территориальных сетей с коммутацией пакетов X.25, frame relay [4].

Разработана также специальная спецификация, определяющая использование технологии ATM в качестве транспорта каналного уровня. Обычно при появлении новой технологии локальных или глобальных сетей она быстро включается в стек TCP/IP за счет разработки соответствующего RFC, определяющего метод инкапсуляции пакетов IP в ее кадры.

Следующий уровень (уровень III) - это уровень межсетевого взаимодействия, который занимается передачей пакетов с использованием

различных транспортных технологий локальных сетей, территориальных сетей, линий специальной связи и т. п.

В качестве основного протокола сетевого уровня (в терминах модели *OSI*) в стеке используется протокол *IP*, который изначально проектировался как протокол передачи пакетов в составных сетях, состоящих из большого количества локальных сетей, объединенных как локальными, так и глобальными связями. Поэтому протокол *IP* хорошо работает в сетях со сложной топологией, рационально используя наличие в них подсистем и экономно расходуя пропускную способность низкоскоростных линий связи. Протокол *IP* является дейтаграммным протоколом, то есть он не гарантирует доставку пакетов до узла назначения, но старается это сделать [4].

К уровню межсетевого взаимодействия относятся и все протоколы, связанные с составлением и модификацией таблиц маршрутизации, такие как протоколы сбора маршрутной информации *RIP* (*Routing Internet Protocol*) и *OSPF* (*Open Shortest Path First*), а также протокол межсетевых управляющих сообщений *ICMP* (*Internet Control Message Protocol*). Последний протокол предназначен для обмена информацией об ошибках между маршрутизаторами сети и узлом - источником пакета. С помощью специальных пакетов *ICMP* сообщается о невозможности доставки пакета, о превышении времени жизни или продолжительности сборки пакета из фрагментов, об аномальных величинах параметров, об изменении маршрута пересылки и типа обслуживания, о состоянии системы и т.п [4].

Следующий уровень (уровень *II*) называется основным. На этом уровне функционируют протокол управления передачей *TCP* (*Transmission Control Protocol*) и протокол дейтаграмм пользователя *UDP* (*User Datagram Protocol*). Протокол *TCP* обеспечивает надежную передачу сообщений между удаленными прикладными процессами за счет образования виртуальных соединений. Протокол *UDP* обеспечивает передачу прикладных пакетов дейтаграммным способом, как и *IP*, и выполняет только функции

связующего звена между сетевым протоколом и многочисленными прикладными процессами.

Верхний уровень (уровень *I*) называется прикладным. За долгие годы использования в сетях различных стран и организаций стек *TCP/IP* накопил большое количество протоколов и сервисов прикладного уровня. К ним относятся такие широко используемые протоколы, как протокол копирования файлов *FTP*, протокол эмуляции терминала *telnet*, почтовый протокол *SMTP*, используемый в электронной почте сети *Internet*, гипертекстовые сервисы доступа к удаленной информации, такие как *WWW* и многие другие. Остановимся несколько подробнее на некоторых из них [4].

Протокол пересылки файлов *FTP* (*File Transfer Protocol*) реализует удаленный доступ к файлу. Для того, чтобы обеспечить надежную передачу, *FTP* использует в качестве транспорта протокол с установлением соединений - *TCP*. Кроме пересылки файлов протокол *FTP* предлагает и другие услуги. Так, пользователю предоставляется возможность интерактивной работы с удаленной машиной, например, он может распечатать содержимое ее каталогов. Наконец, *FTP* выполняет аутентификацию пользователей. Прежде, чем получить доступ к файлу, в соответствии с протоколом пользователи должны сообщить свое имя и пароль. Для доступа к публичным каталогам *FTP*-архивов *Internet* парольная аутентификация не требуется, и ее обходят за счет использования для такого доступа предопределенного имени пользователя *Anonymous*.

В стеке *TCP/IP* протокол *FTP* предлагает наиболее широкий набор услуг для работы с файлами, однако он является и самым сложным для программирования. Приложения, которым не требуются все возможности *FTP*, могут использовать другой, более экономичный протокол - простейший протокол пересылки файлов *TFTP* (*Trivial File Transfer Protocol*). Этот протокол реализует только передачу файлов, причем в качестве транспорта

используется более простой, чем *TCP*, протокол без установления соединения - *UDP*.

Протокол *telnet* обеспечивает передачу потока байтов между процессами, а также между процессом и терминалом. Наиболее часто этот протокол используется для эмуляции терминала удаленного компьютера. При использовании сервиса *telnet* пользователь фактически управляет удаленным компьютером так же, как и локальный пользователь, поэтому такой вид доступа требует хорошей защиты. Поэтому серверы *telnet* всегда используют как минимум аутентификацию по паролю, а иногда и более мощные средства защиты, например, систему *Kerberos* [4].

Протокол *SNMP* (*Simple Network Management Protocol*) используется для организации сетевого управления. Изначально протокол *SNMP* был разработан для удаленного контроля и управления маршрутизаторами *Internet*, которые традиционно часто называют также шлюзами. С ростом популярности протокол *SNMP* стали применять и для управления любым коммуникационным оборудованием - концентраторами, мостами, сетевыми адаптерами и т.д. и т.п. Проблема управления в протоколе *SNMP* разделяется на две задачи [4].

Первая задача связана с передачей информации. Протоколы передачи управляющей информации определяют процедуру взаимодействия *SNMP*-агента, работающего в управляемом оборудовании, и *SNMP*-монитора, работающего на компьютере администратора, который часто называют также консолью управления. Протоколы передачи определяют форматы сообщений, которыми обмениваются агенты и монитор.

Вторая задача связана с контролируемыми переменными, характеризующими состояние управляемого устройства. Стандарты регламентируют, какие данные должны сохраняться и накапливаться в устройствах, имена этих данных и синтаксис этих имен. В стандарте *SNMP* определена спецификация информационной базы данных управления сетью.

Эта спецификация, известная как база данных *MIB* (*Management Information Base*), определяет те элементы данных, которые управляемое устройство должно сохранять, и допустимые операции над ними.

Отчасти популярность стека протоколов *TCP/IP* объясняется возможностью его реализации на базе большого числа разнообразных каналов и протоколов канального уровня, таких как *T1* и *X.25*, *Ethernet* и линии *RS-232*. Большинство организаций использует в своих ЛВС *Ethernet* для объединения хостов и клиентских систем, а затем присоединяет эти сети с помощью *T1* к региональной сети (например, региональной магистральной сети *TCP/IP*), которая соединяет в свою очередь с сетями других организаций и другими магистральными каналами. Как правило, организации имеют одно соединение с Интернетом, но большие организации могут иметь два и более соединений.

1.2 Разработка приложения

Приложение состоит из двух частей: клиента и сервера. Вместе они образуют систему для потоковой передачи аудио с возможностью отправки текстовых сообщений. Взаимодействие клиента и сервера осуществляется через протокол *TCP/IP*. *UML*-диаграмма развёртывания показана на рисунке 1.2.

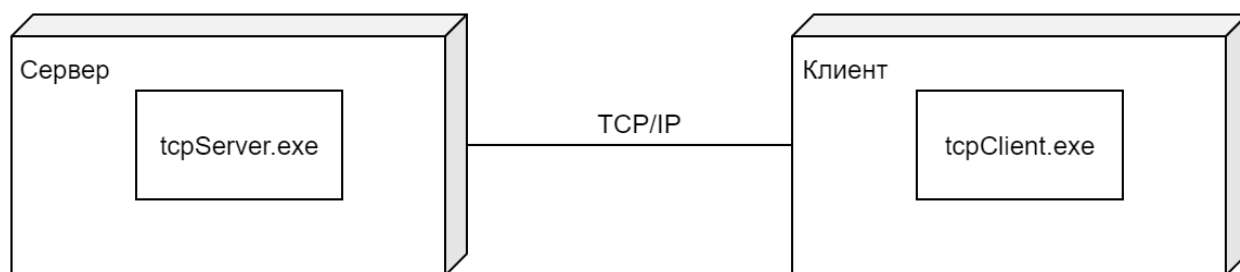


Рисунок 1.2 – Диаграмма развёртывания приложения

Пользователь работает с клиентской частью приложения, которая имеет понятный графический интерфейс. Рассмотрим работу данного приложения.

Вначале необходимо запустить серверное приложение. После запуска появляется графический интерфейс сервера, в котором есть:

- поле для ввода *IP*-адреса и поле для ввода порта;
- кнопка запуска сервера;
- поле для текстового лога сервера;
- поле ввода и кнопка отправки сообщений для всех клиентов.

Для запуска сервера необходимо ввести *IP*-адрес вычислительной машины и номер свободного порта для подключения к нему клиентов, затем нажать кнопку запустить сервер. При успешном запуске кнопка запуска изменится на кнопку остановки сервера, и сервер начнёт прослушивание подключений к порту. Для работы в глобальной сети необходимо на машине с сервером иметь статический *IP*-адрес.

Далее необходимо запустить клиент.

После запуска появляется графический интерфейс клиента, в котором есть:

- поле для ввода имени пользователя;
- поля для ввода *IP*-адреса и порта сервера;
- кнопка подключения к серверу;
- поле для получения сообщений от сервера и других пользователей;
- поле ввода и кнопка отправки текстовых сообщений всем пользователям;
- кнопка включения голосовой связи;
- поля выбора устройств ввода и вывода звука.

После запуска приложения необходимо ввести имя пользователя, *IP*-адрес и порт сервера для подключения, затем нажать кнопку

«Подключиться». Приложение отправит запрос на подключение к серверу по введённым *ip*-адресу и порту и будет ждать ответа.

При успешном подключении в лог выведется сообщение о подключении и станут доступны функции подключения к голосовому чату, а также отправка текстовых сообщений.

Более подробно это показано на *UML*-диаграмме последовательности запуска, которая представлена на рисунке 1.3.

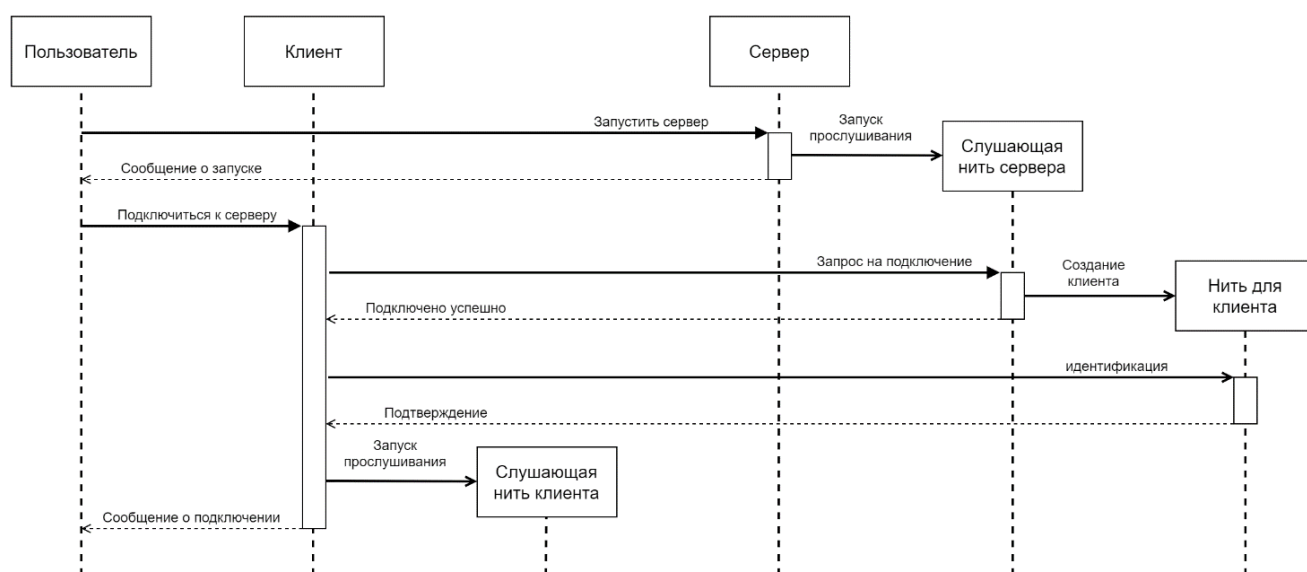


Рисунок 1.3 - Диаграмма последовательности запуска

Все *UML*-диаграммы приведены в приложении А.

Далее пользователь может присоединиться к голосовому чату. После нажатия кнопки «Говорить» и при наличии микрофона звук с микрофона пользователя будет записываться в буфер и далее передаваться на сервер. На сервере все звуковые потоки от пользователей микшируются в один и рассылаются всем клиентам. При поступлении от сервера аудиоданных и при наличии устройства вывода звука клиент начинает воспроизведение. Для остановки воспроизведения на клиенте нужно нажать кнопку заглушить.

Диаграмма последовательности отправки голосового сообщения проиллюстрирована на рисунке 1.4.

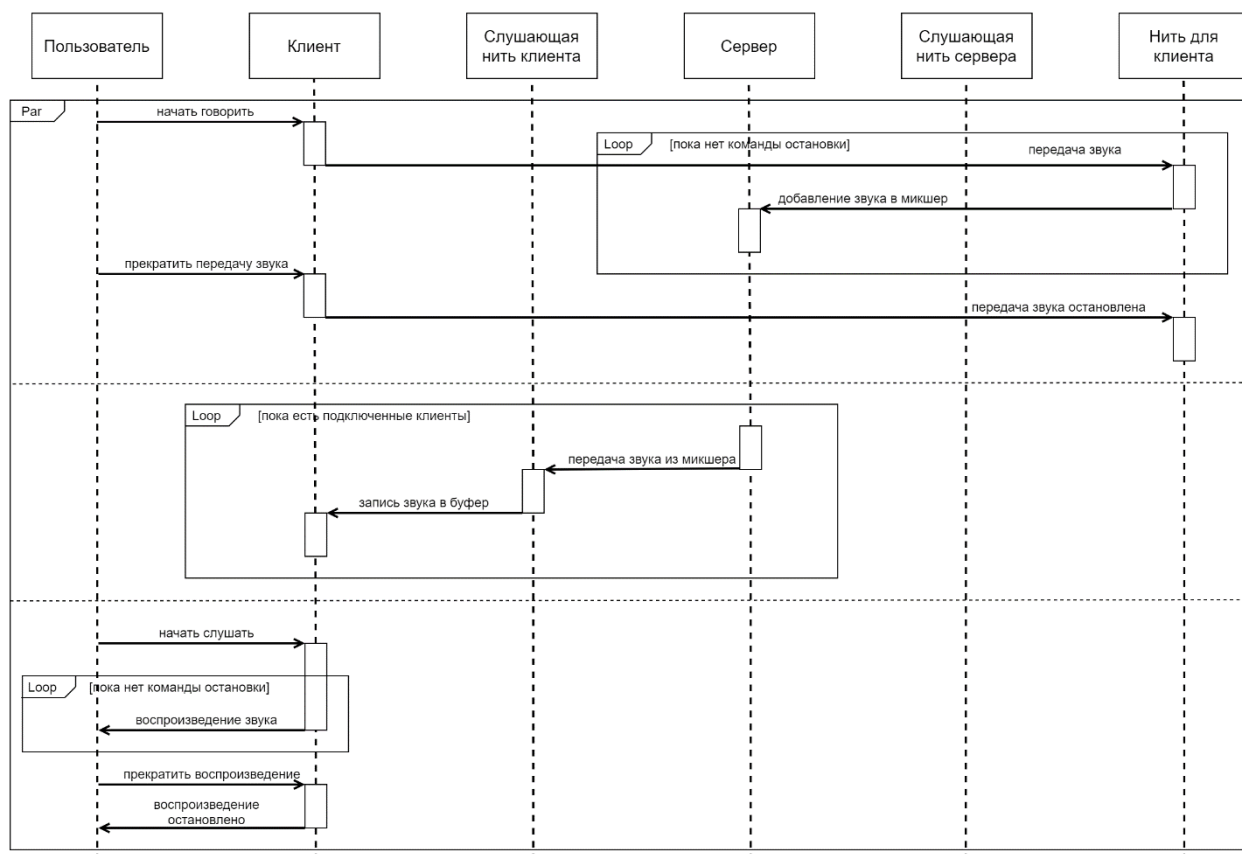


Рисунок 1.4 - Диаграмма последовательности голосового общения

Также пользователь может отправить всем клиентам текстовое сообщение. Для этого необходимо ввести текст в нижнее поле ввода и нажать кнопку отправить. Сообщение отправится на сервер и там произойдет рассылка всем пользователям. При получении текстового сообщения клиент дописывает его в лог.

Диаграмма последовательности текстового сообщения показана на рисунке 1.5.

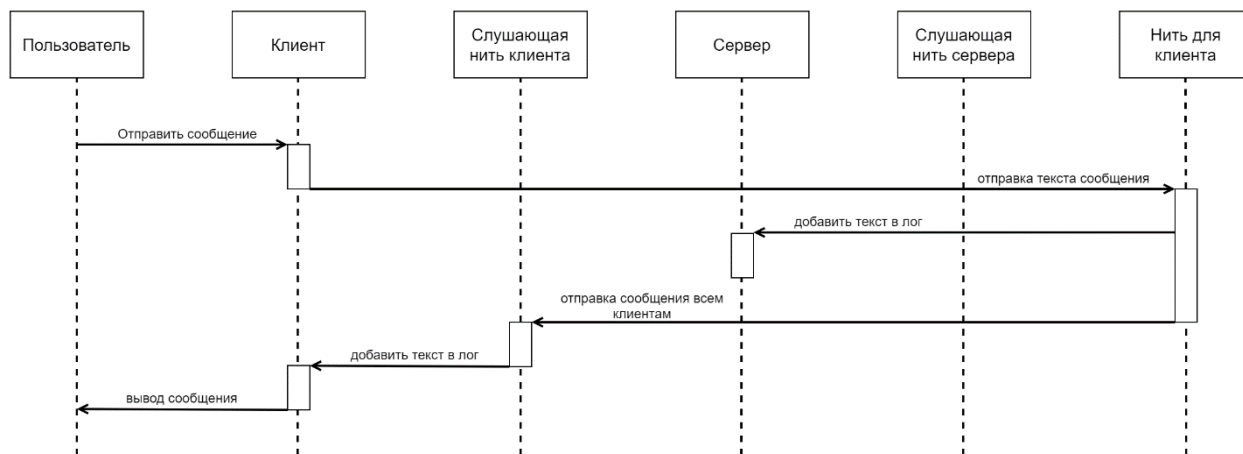


Рисунок 1.5 - Диаграмма последовательности текстового общения

1.2.1 Реализация серверного приложения

Серверное приложение в основном не требует вмешательства со стороны пользователя. Пользователь может задавать *ip*-адрес и порт, запускать и останавливать сервер, а также отправлять всем клиентам текстовые сообщения. *UML*-диаграмма деятельности сервера приведена на рисунке 1.6.

После запуска сервер начинает прослушивание входящих подключений. При подключении клиента сервер выделяет для него отдельную нить, в которой происходит прослушивание клиента на наличие входящих сообщений и их обработка в соответствии с типом сообщения.

От клиента могут приходить сообщения с несколькими маркерами:

- *#connect* – запрос на идентификацию;
- *#msg* – текстовое сообщение;
- *#audio* – запуск или остановка приёма звука.

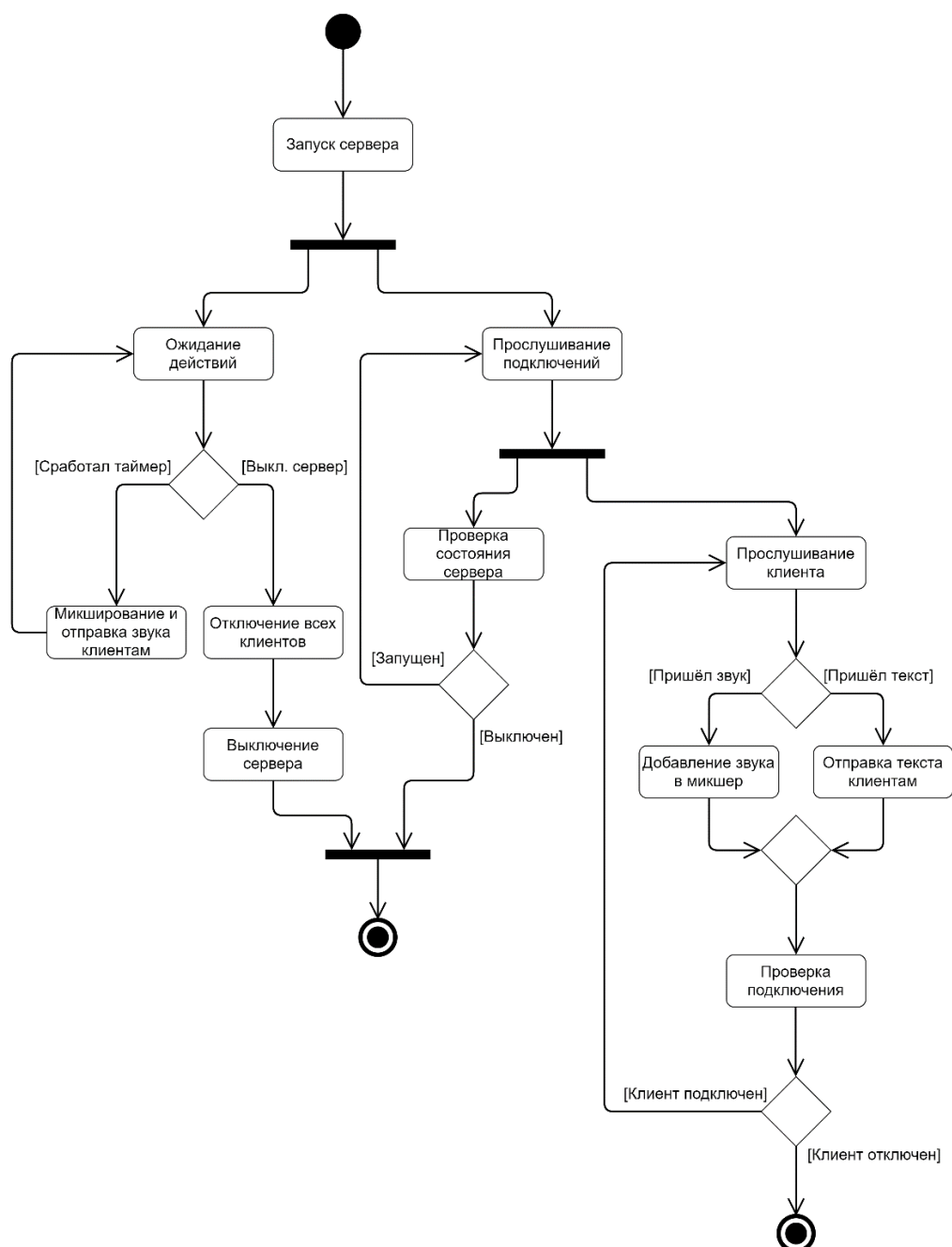


Рисунок 1.6 - Диаграмма деятельности сервера

Запрос на идентификацию приходит сразу после подключения клиента. При получении такого запроса сервер отправляет этому клиенту сообщение об успешном подключении, а всем остальным клиентам отправляет сообщение о подключении нового пользователя с именем этого пользователя. Если от клиента приходит текстовое сообщение, сервер добавляет к нему имя

пользователя, затем пересылает его всем подключенным клиентам и выводит в свой лог. Также сообщение добавляется в историю сообщений.

Сообщение с маркером *#audio* приходит с дополнительной частью, которая содержит либо команду *&start*, либо команду *&stop*. По команде *&start* сервер начинает приём массивов байтов от клиента. В этих массивах содержится преобразованная звуковая дорожка. Сервер добавляет эту дорожку в функцию микшера, которая смешивает несколько звуковых дорожек в одну. По команде *&stop* сервер прекращает приём массивов байтов от клиента и их добавление в микшер. Помимо обработки данных от клиентов, сервер с некоторой периодичностью сервер рассылает клиентам смикшированную звуковую дорожку в виде массива байтов.

Серверное приложение разрабатывается на языке программирования C#. C# поддерживает понятия класса, что позволяет повысить читаемость и наглядность исходного кода программы, а также выделить сущности предметной области. Каждая такая сущность является отдельным классом. Все они неким образом взаимодействуют между собой. Рассмотрим классы, которые понадобятся для реализации сервера [1].

На рисунке 1.7 изображена UML-диаграмма классов сервера.

Form1 – основной класс сервера. В данном классе содержится реализация графического интерфейса, обработка взаимодействий с графическим интерфейсом, а также основная логика программы.

Client – класс, отвечающий за взаимодействие с клиентом. В нём реализованы функции обработки подключения, идентификации и отключения клиента, а также функции прослушивания и обработки данных, поступающих от клиентского приложения.

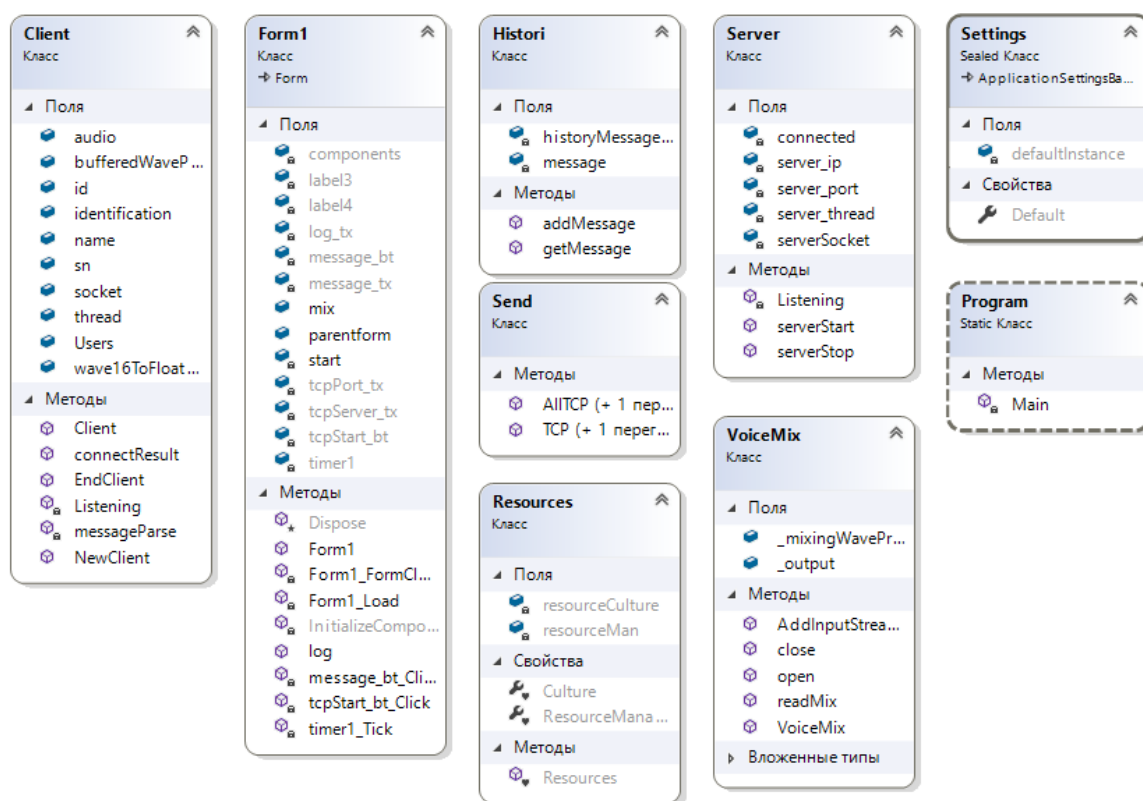


Рисунок 1.7 - Диаграмма классов сервера

History – класс, в котором реализовано сохранение истории сообщений. Содержит функции добавления сообщений в историю и получения истории сообщений.

Server – класс, отвечающий за функционирование сервера. Реализует функции запуска и остановки сервера, а также функцию прослушивания входящих подключений.

VoiceMix – класс, в котором реализовано микширование звуковых дорожек. Содержит функции добавления и удаления прослушиваемых клиентов, функцию микширования и функцию преобразования звуковой дорожки в массив байтов.

Send – класс с функциями отправки сообщений одному или всем клиентам сервера.

1.2.2 Реализация клиентского приложения

В данном проекте пользователь наиболее часто взаимодействует с клиентским приложением. В клиентском приложении пользователь может задать *ip*-адрес и порт, после чего подключиться к серверу. После подключения появляются возможности прослушивать и говорить в голосовом чате со всеми клиентами, а также отправлять всем текстовые сообщения.

Более наглядно работу клиентского приложения можно рассмотреть на *UML*-диаграмме деятельности клиента, изображенной на рисунке 1.8.

После соединения с сервером клиент по умолчанию начинает прослушивание голосового чата, если подключено устройство вывода звука. Клиент получает от сервера массивы байтов, и загружает их в буфер воспроизведения. Звук воспроизводится по мере поступления от сервера. Воспроизведение можно остановить кнопкой «Заглушить».

При нажатии кнопки «Говорить» клиент начинает запись звука с подключенного микрофона в буфер и посылает на сервер команду *#audio&start*. Эта команда активирует прослушивание клиента на сервере.

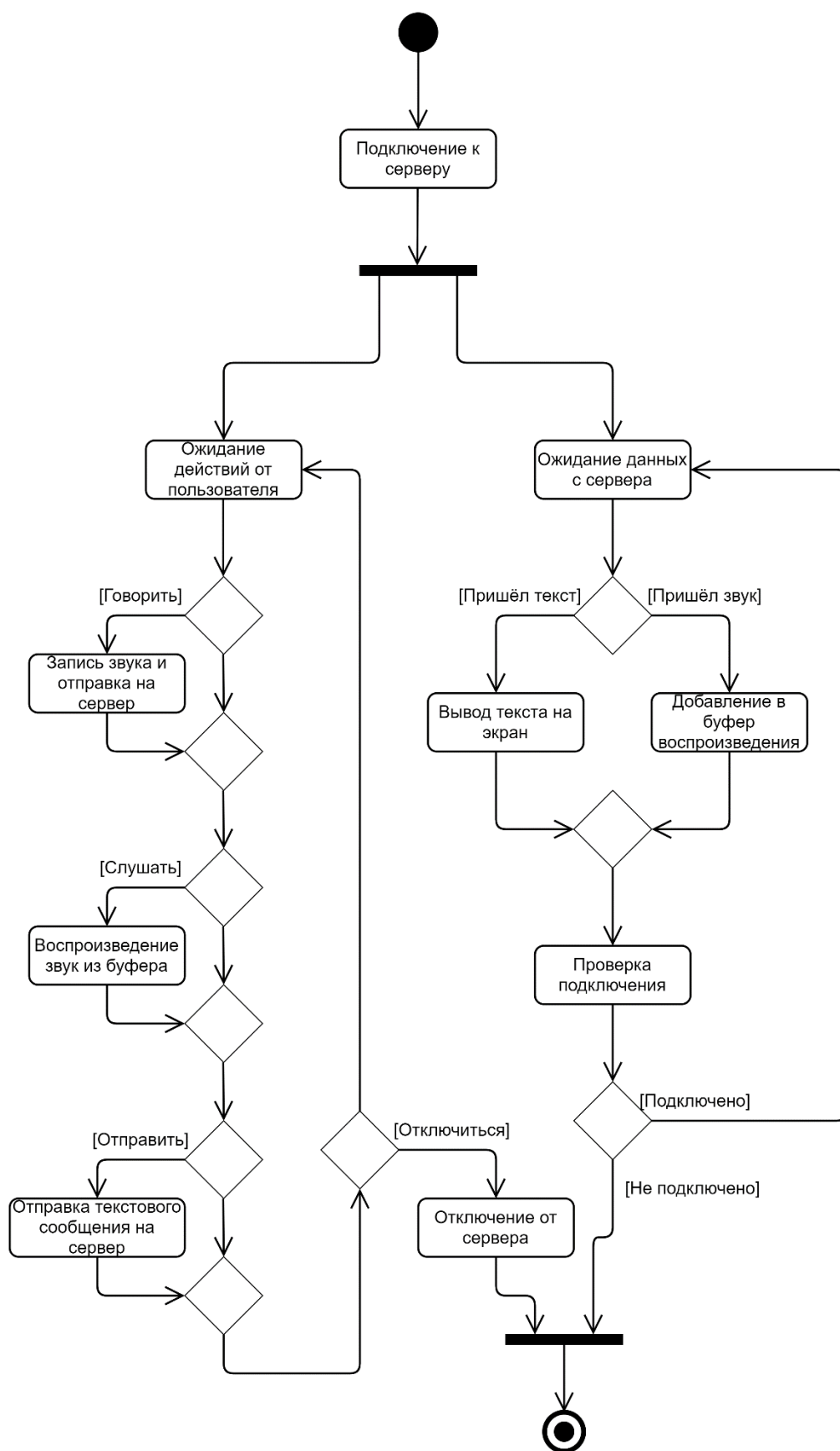


Рисунок 1.8 - Диаграмма деятельности клиента

Затем клиент периодически отправляет на сервер звуковые дорожки из буфера записи звука. Остановить запись и передачу звука на сервер можно кнопкой «Остановить». При этом клиент отправляет на сервер команду *#audio&stop* для остановки прослушивания.

У пользователя также есть возможность выбора устройства ввода и вывода звука, но сделать это можно, только когда неактивны функции записи и воспроизведения соответственно.

Кроме голосовой связи пользователь может вводить и отправлять текстовые сообщения на сервер. Отправка сообщения происходит при нажатии кнопки «Отправить». Программа считывает текст сообщения из текстового поля и добавляет в начало сообщения маркер *#msg*. Маркер добавляется для того, чтобы сервер обрабатывал это сообщение как текстовое. После окончания работы на данном сервере пользователь может отключиться от него и подключиться к другому серверу.

Для разработки клиента также был использован язык C#.

Рассмотрим классы, которые понадобились для реализации клиента.

Form1 – основной класс программы. В нём содержатся все функции взаимодействия с графическим интерфейсом основная логика программы.

Client – класс, в котором реализованы все функции взаимодействия клиента и сервера. В него входят функции подключения и отключения, прослушивания и отправки сообщений.

На рисунке 1.9 изображена UML-диаграмма классов сервера.

Полный листинг программы расположен в приложении Б.

1.2.3 Тестирование

В ходе тестирования разработанного приложения были выявлены и обработаны большинство исключительных ситуаций. Рассмотрим обработку некоторых из них.

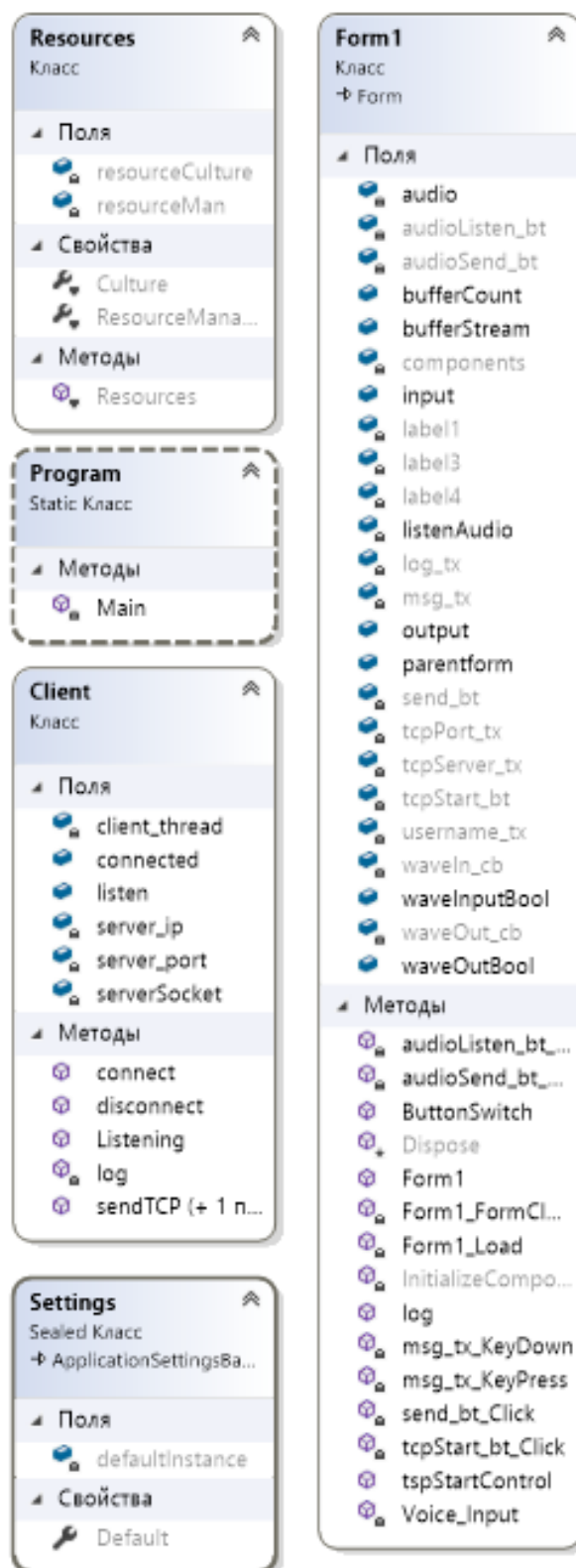


Рисунок 1.9 - Диаграмма классов клиента

Во время запуска сервера или подключения к серверу клиента пользователь может некорректно ввести *ip*-адрес сервера и его порт. Для

недопущения ошибки в случае некорректно введенных данных была добавлена проверка *ip*-адреса и порта на соответствие формату и максимальному значению. Если введенные пользователем данные не проходят проверку, выводится сообщение о том, что введены некорректные значения. Сообщение об ошибке показано на рисунке 1.10.

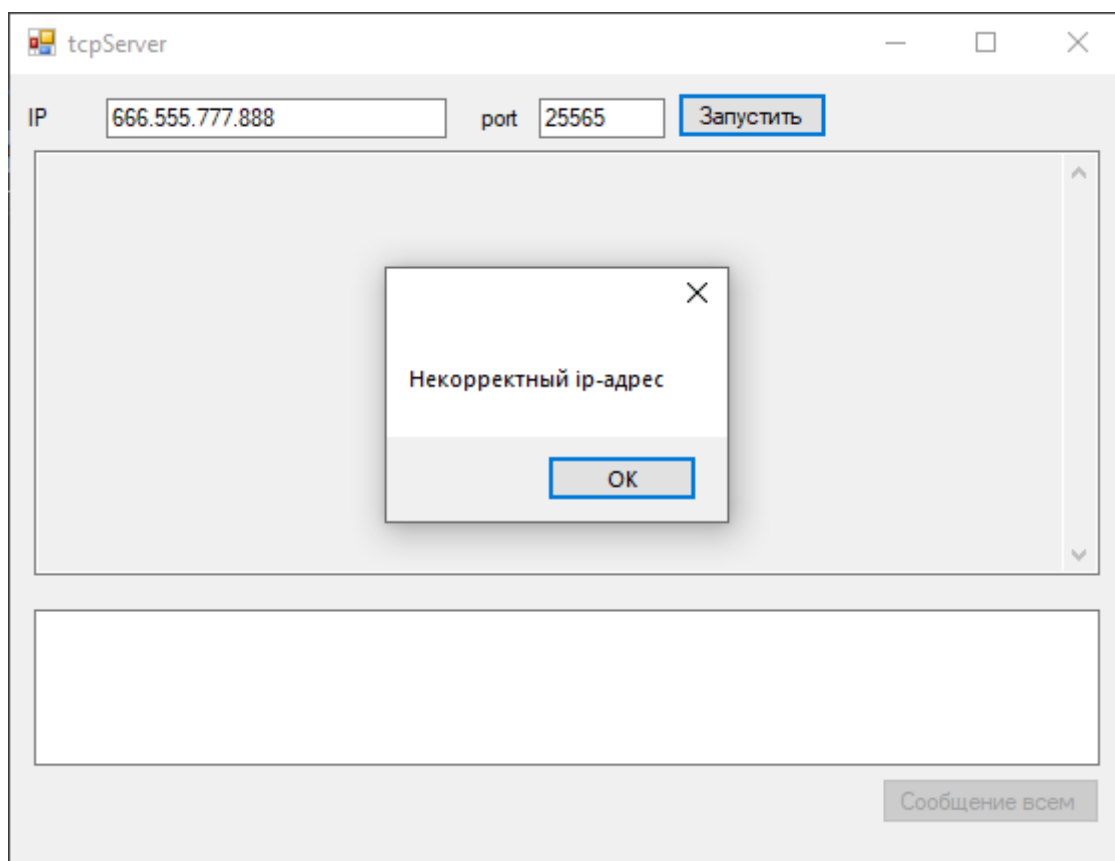


Рисунок 1.10 – Сообщение об ошибке ввода *ip*-адреса

Во время запуска прослушивания или начала разговора в голосовом чате в клиентском приложении, к компьютеру должны быть подключены устройства воспроизведения и записи звука соответственно. Если пользователь не подключил какое-либо устройство, при попытке начать общение пользователю будет выведено сообщение о том, что устройство отключено. Это показано на рисунке 1.11.

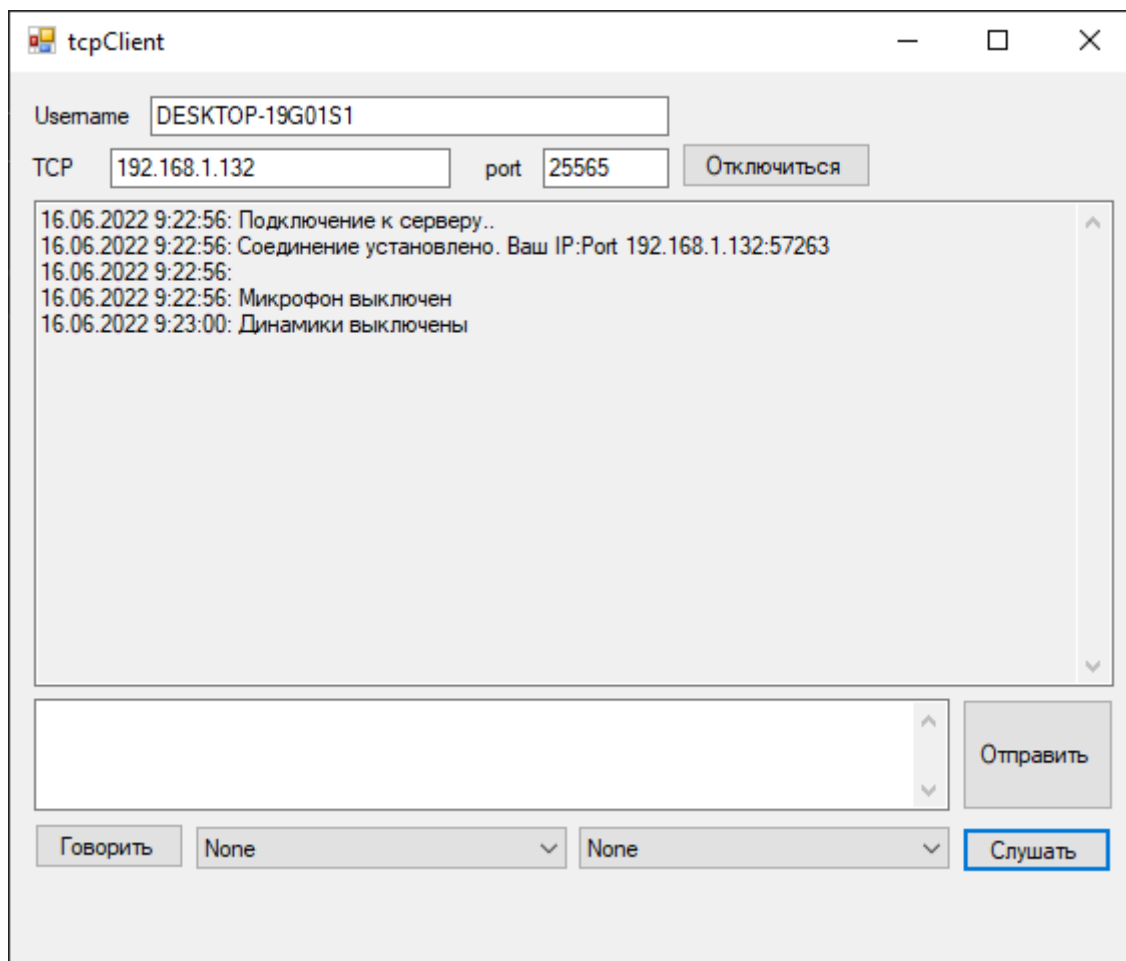


Рисунок 1.11 – Сообщение об отключенных устройствах ввода-вывода

Помимо ошибок пользовательского ввода также были обработаны основные исключительные ситуации, связанные с некорректной реакцией системы.

После тестирования и обработки исключительных ситуаций, и клиентское, и серверное приложение стали работать гораздо стабильнее и надёжнее.

1.3 Анализ результатов

Было реализовано клиентское и серверное приложения, предназначенные для голосового и текстового общения между пользователями.

1.3.1 Окно серверного приложения

Рассмотрим подробнее взаимодействие с серверным приложением. Варианты взаимодействия пользователя с серверным приложением приведены на *UML* – диаграмме вариантов использования на рисунке 1.12.

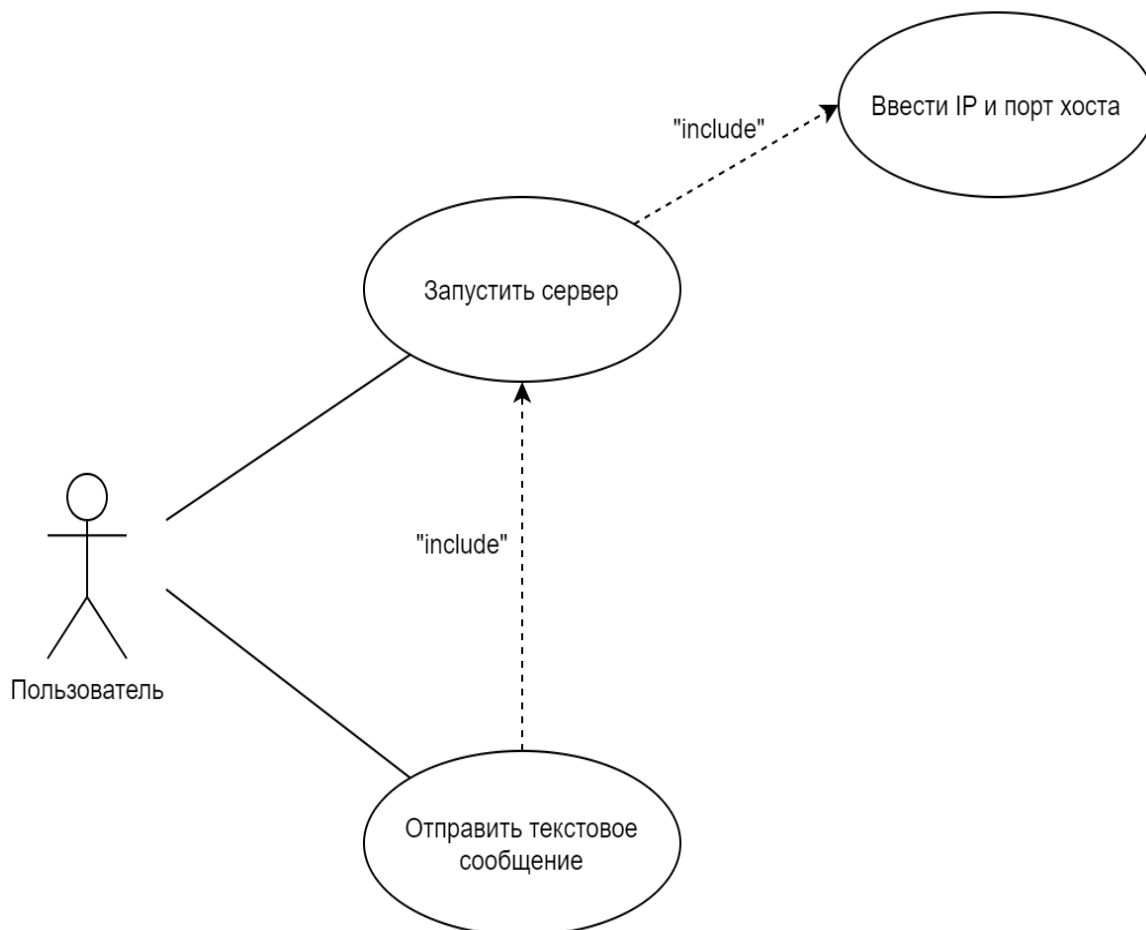


Рисунок 1.12 – Диаграмма вариантов использования сервера

После запуска приложения открывается окно, показанное на рисунке 1.13.

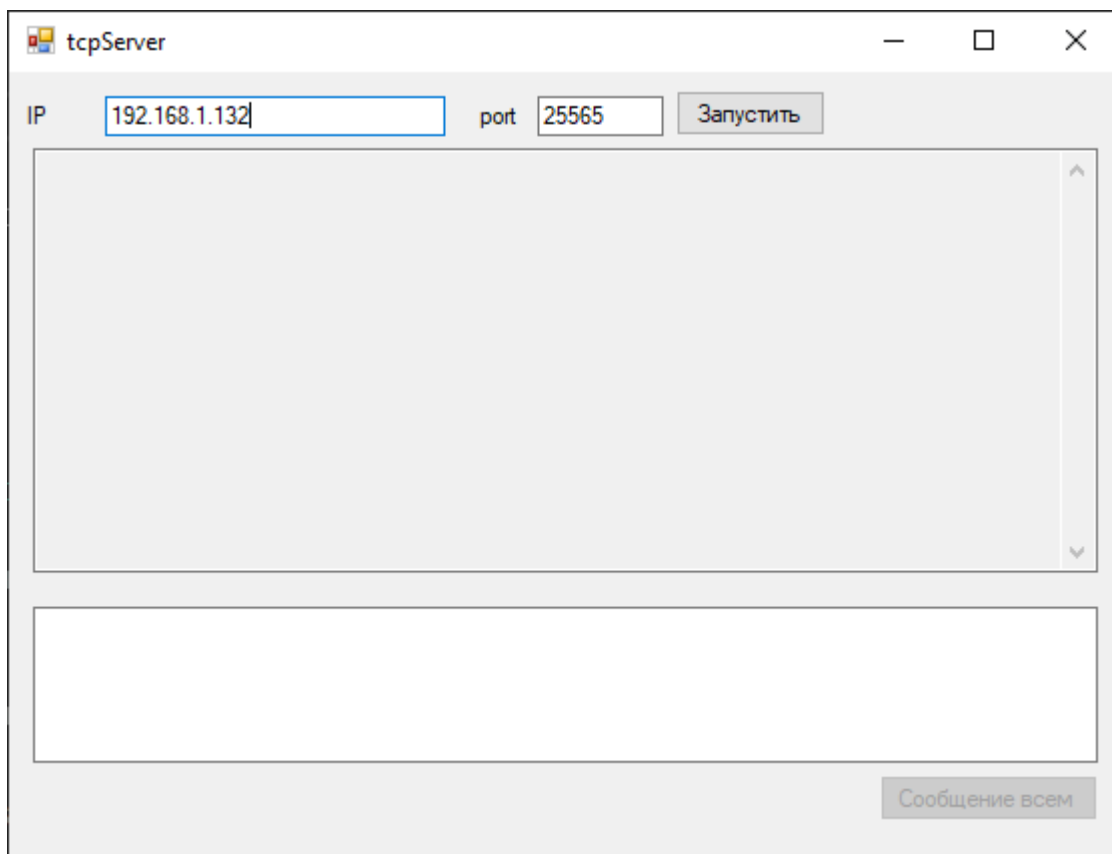


Рисунок 1.13 - Окно серверного приложения после запуска

В поле «*IP*» выводится локальный *ip*-адрес вашего ПК.

В поле «*port*» выводится номер порта, который сервер будет прослушивать для приёма входящих подключений. Значение *ip*-адреса и порта можно изменить до запуска сервера.

После нажатия кнопки «Запустить» сервер попытается создать точку подключения по введённым данным и начать её прослушивание.

При успешном запуске в лог выведется сообщение, кнопка «Запустить» поменяется на кнопку «Остановить», а также активируется кнопка «Сообщение всем».

Это показано на рисунке 1.14.

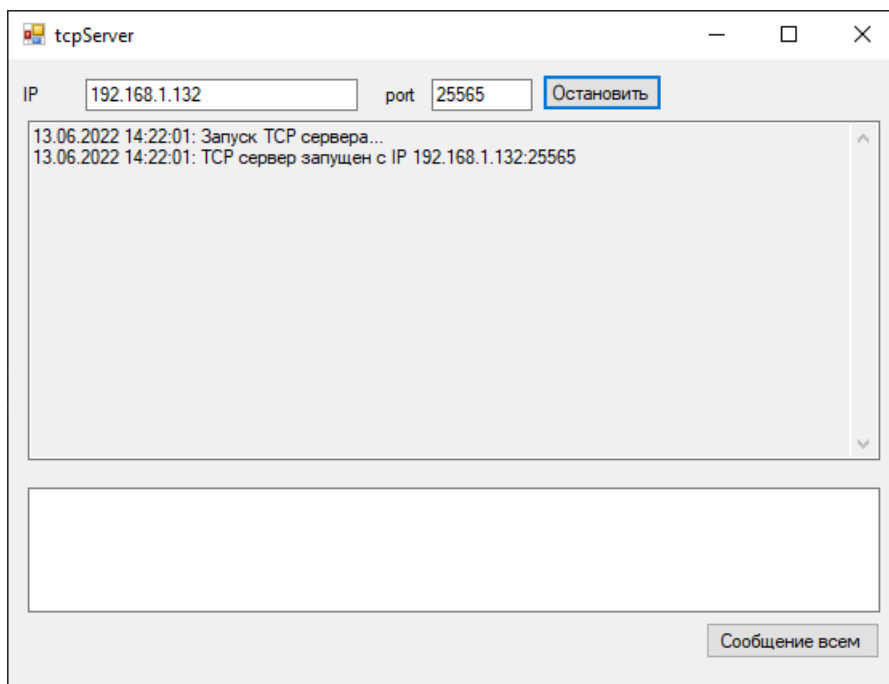


Рисунок 1.14 - Успешный запуск сервера

После успешного запуска сервер начинает принимать подключения. При подключении к серверу клиентов в лог выводится сообщение с именем, *ip*-адресом и портом клиента. Это показано на рисунке 1.15.

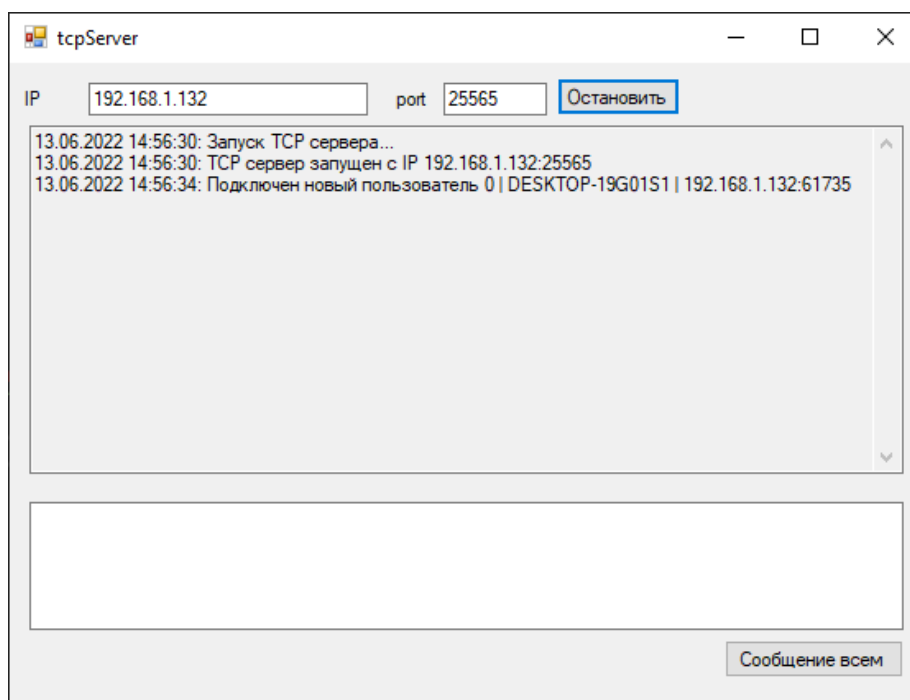


Рисунок 1.15 - Подключение клиента

С сервера можно отправлять текстовые сообщения всем клиентам. Для этого нужно ввести сообщение в поле ввода и нажать кнопку «Сообщение всем» (рисунок 1.16).

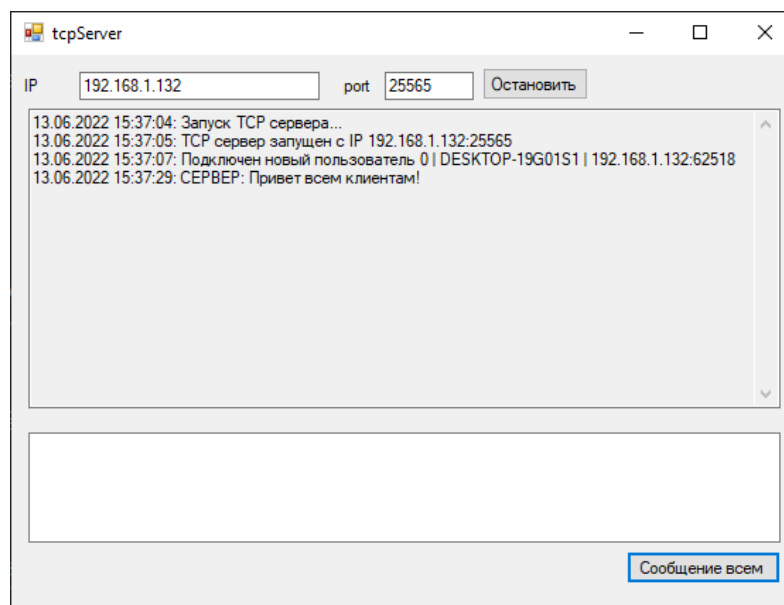


Рисунок 1.16 - Сообщение всем клиентам

При запуске и остановке голосового чата пользователем на сервер приходит соответствующее сообщение. Оно показано на рисунке 1.17.

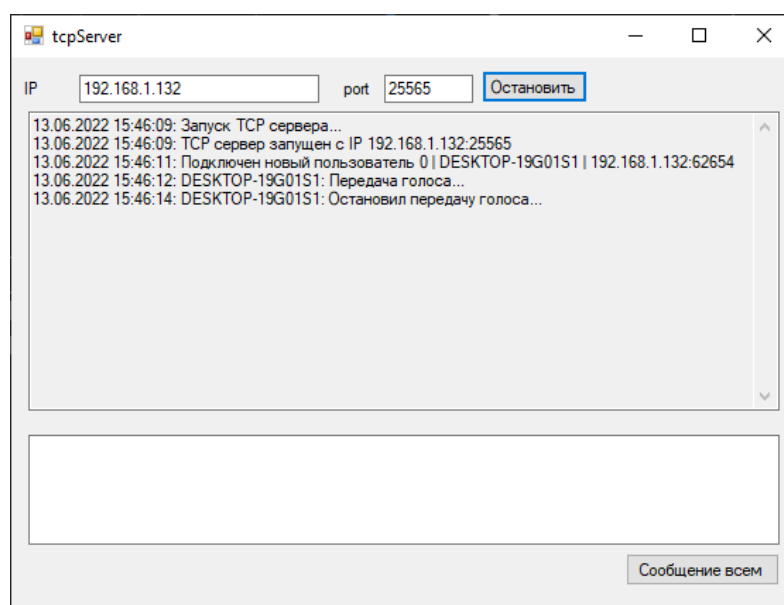


Рисунок 1.17 - Запуск и остановка голосового чата пользователем

Все сообщения от клиентов при прохождении через сервер также записываются в лог (рисунок 1.18).

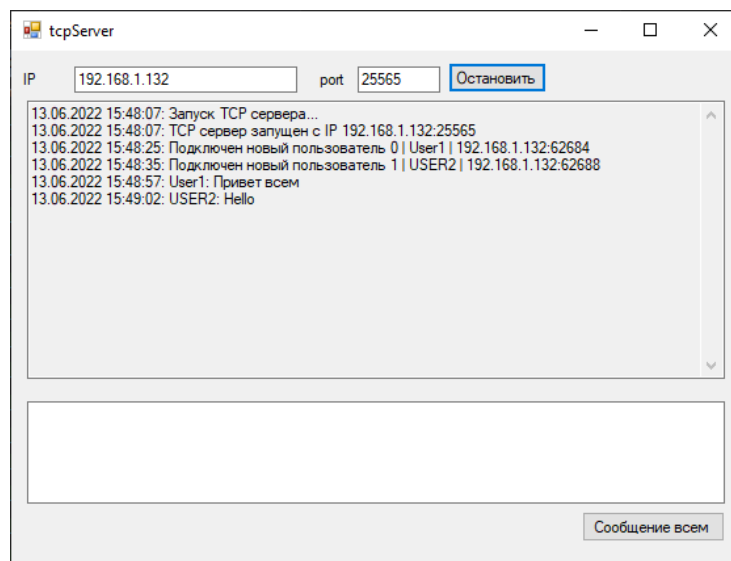


Рисунок 1.18 - Получение сообщений от пользователей

После окончания работы сервер можно остановить. Для этого нужно нажать на кнопку «Остановить». При этом сервер отключит всех клиентов, оповестив их об остановке, и выведет в лог соответствующее сообщение (рисунок 1.19).

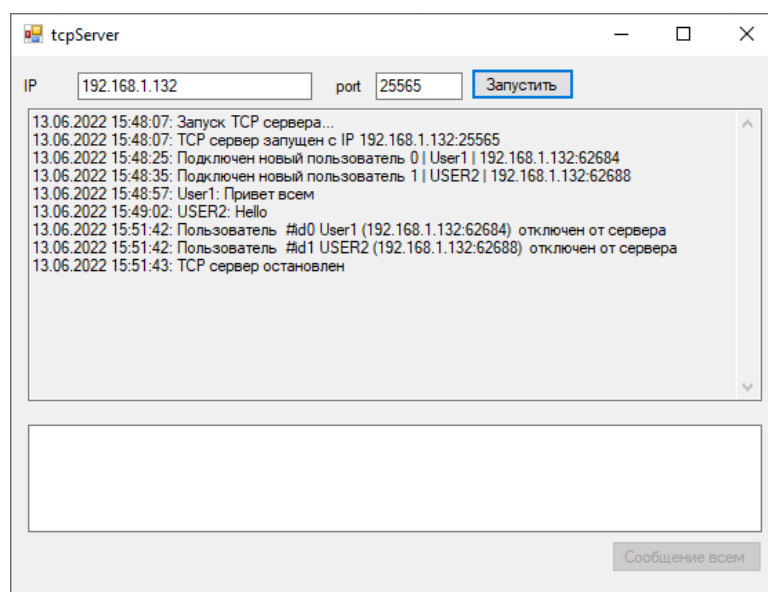


Рисунок 1.19 - Выключение сервера

1.3.2 Окно клиентского приложения

Рассмотрим работу с окном клиентского приложения.

Варианты взаимодействия пользователя с клиентским приложением приведены на *UML* – диаграмме вариантов использования на рисунке 1.20.

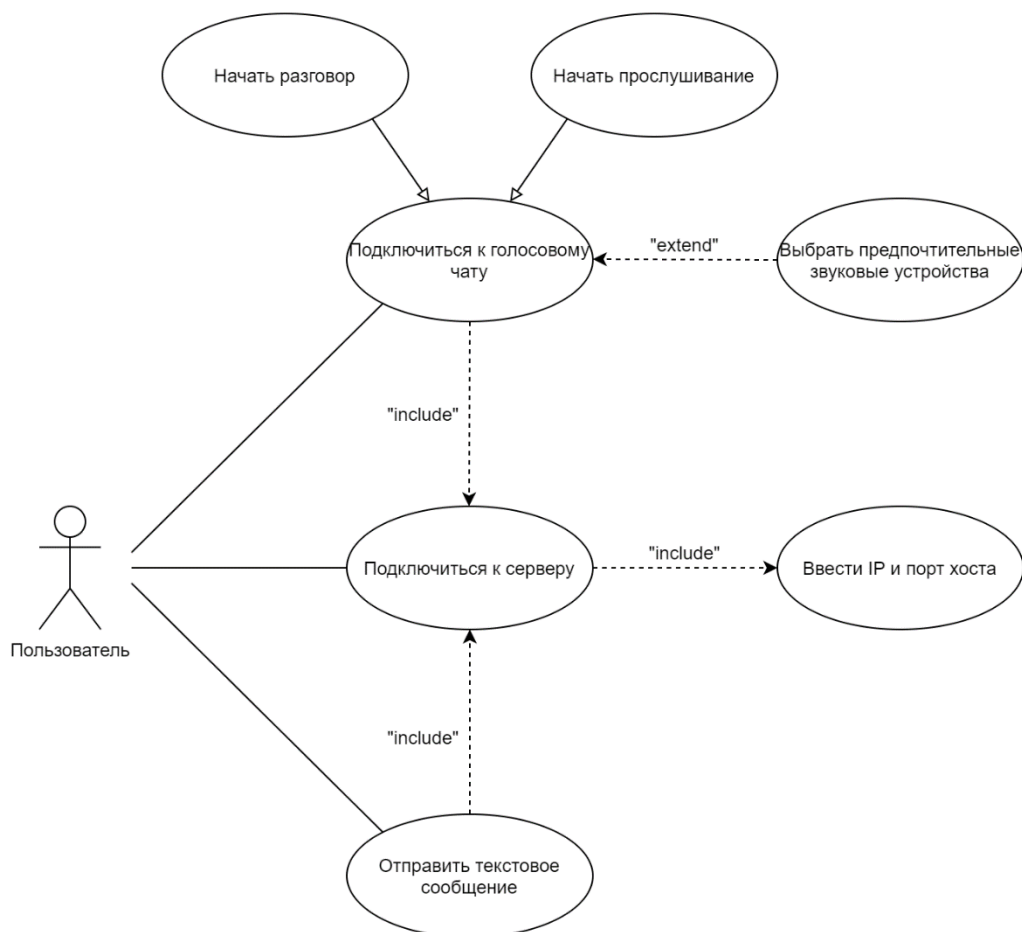


Рисунок 1.20 – Диаграмма вариантов использования клиента

После запуска окно приложения имеет вид, показанный на рисунке 1.21.

После запуска нужно ввести *ip*-адрес и порт сервера и нажать кнопку подключения.

После подключения в лог выведется сообщение от сервера и станут активными кнопки «Отправить» и «Говорить».

Кнопка «Слушать» станет активна и изменит надпись на «Заглушить», так как при подключении к серверу сразу начнётся прослушивание голосового чата.

Увидеть это можно на рисунке 1.22.

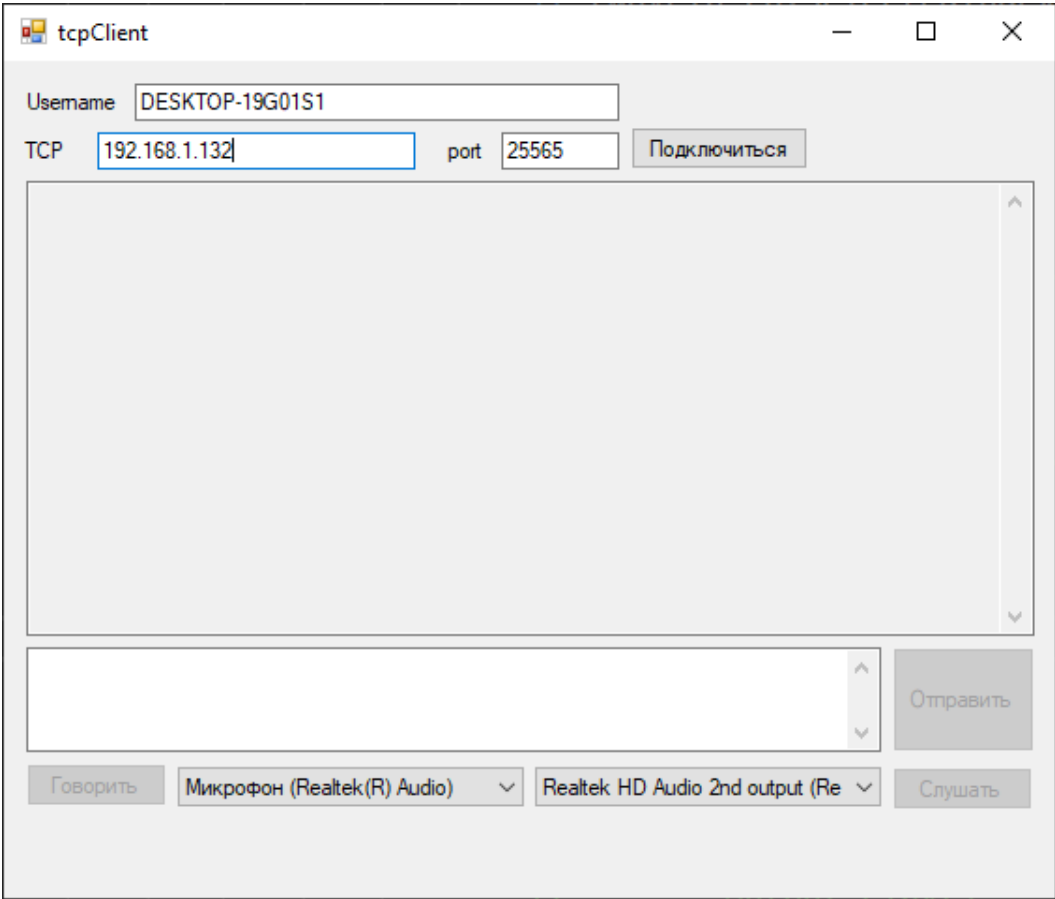


Рисунок 1.21 - Окно клиентского приложения после запуска

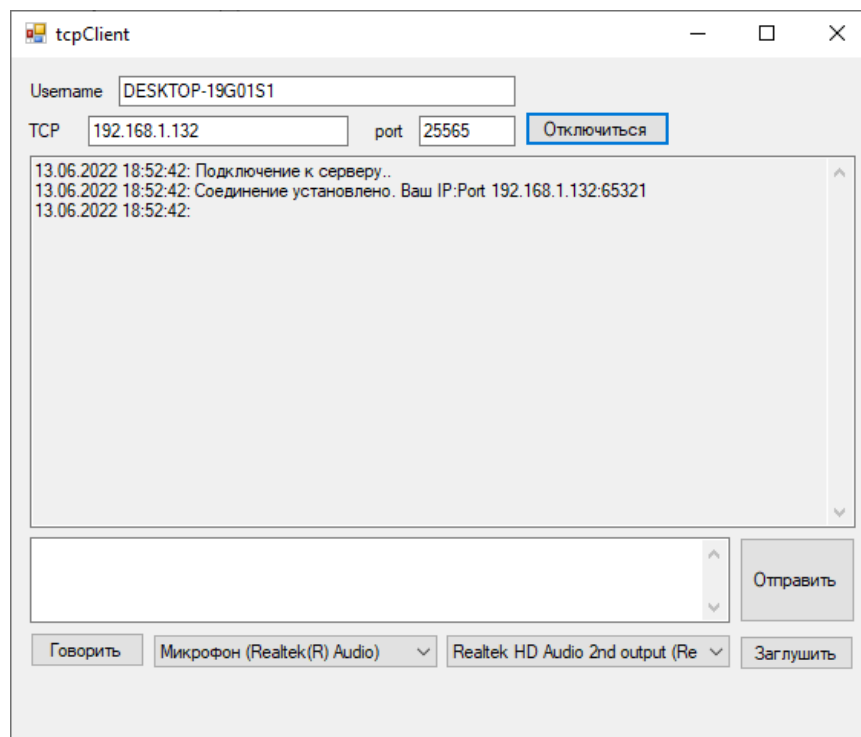


Рисунок 1.22 - Окно клиента после подключения к серверу

Перед тем, как начинать говорить в голосовой чат, можно выбрать предпочтительные устройства ввода звука.

Также можно выбрать устройство воспроизведения, но для этого сначала надо нажать кнопку заглушить, затем выбрать другое устройство. При выборе будут показаны все подключенные к вашему устройству микрофоны и динамики/наушники.

Данная возможность показана на рисунках 1.23 и 1.24.

Теперь вы можете начать общение в голосовом чате. При нажатии кнопки «Говорить» сообщение о том, что вы начали голосовой чат, отправится всем пользователям.

Затем звук с вашего устройства записи будет добавляться в общий звуковой поток голосового чата.

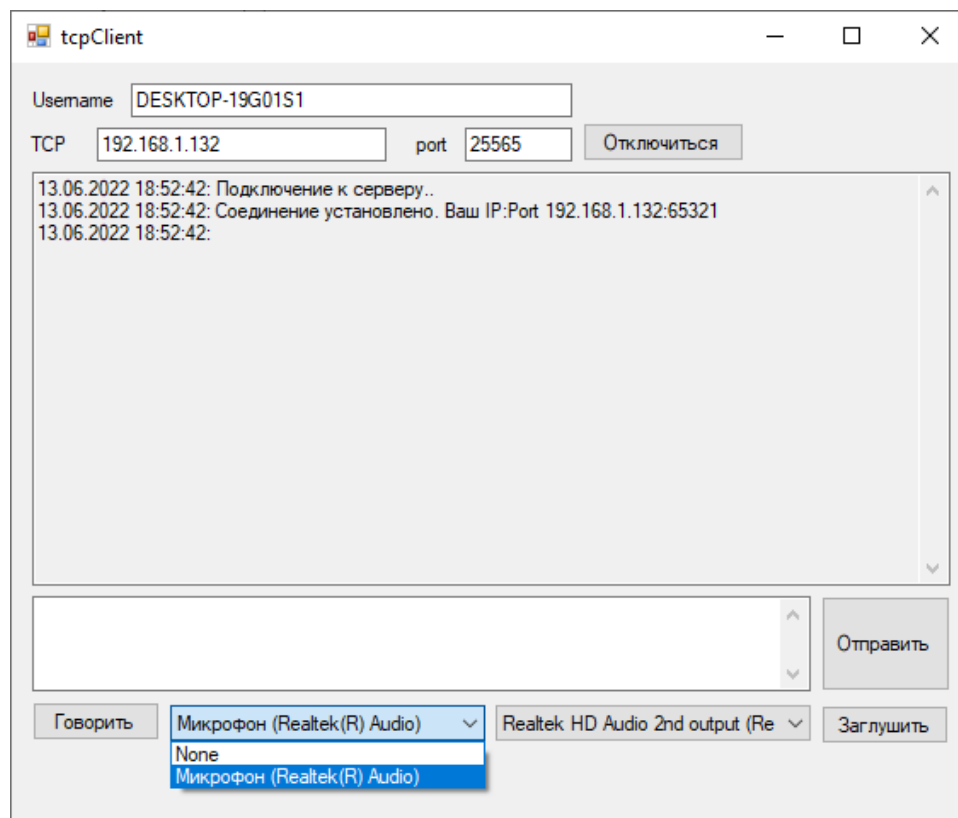


Рисунок 1.23 - Выбор микрофона

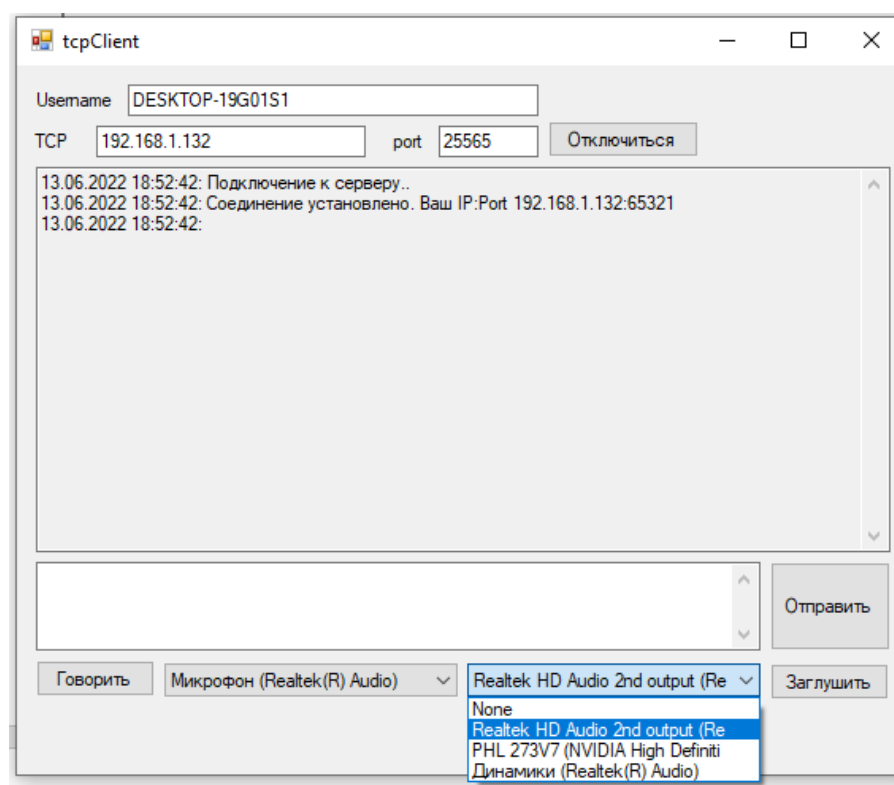


Рисунок 1.24 - Выбор устройства воспроизведения

При окончании общения также будет отправлено сообщение о прекращении вами участия в голосовом чате (рисунок 1.25).

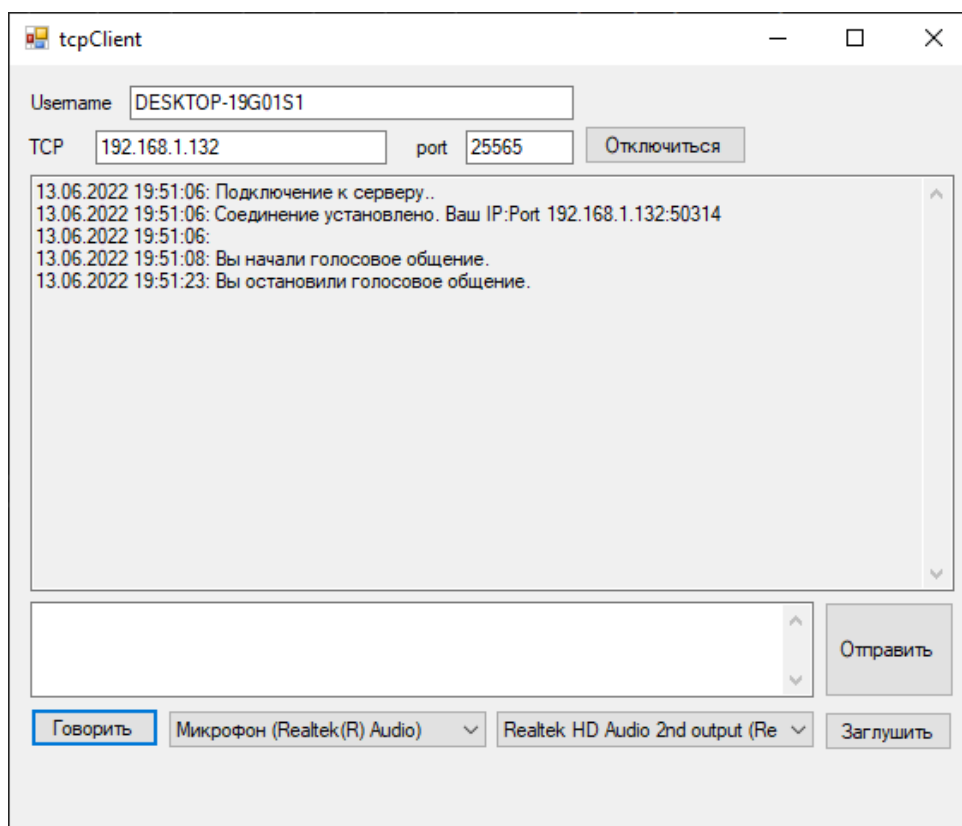


Рисунок 1.25 - Запуск и остановка голосового общения

При этом в приложении предусмотрена возможность отправки текстового сообщения каждому пользователю даже во время голосового общения.

Эта показано на рисунке 1.26.

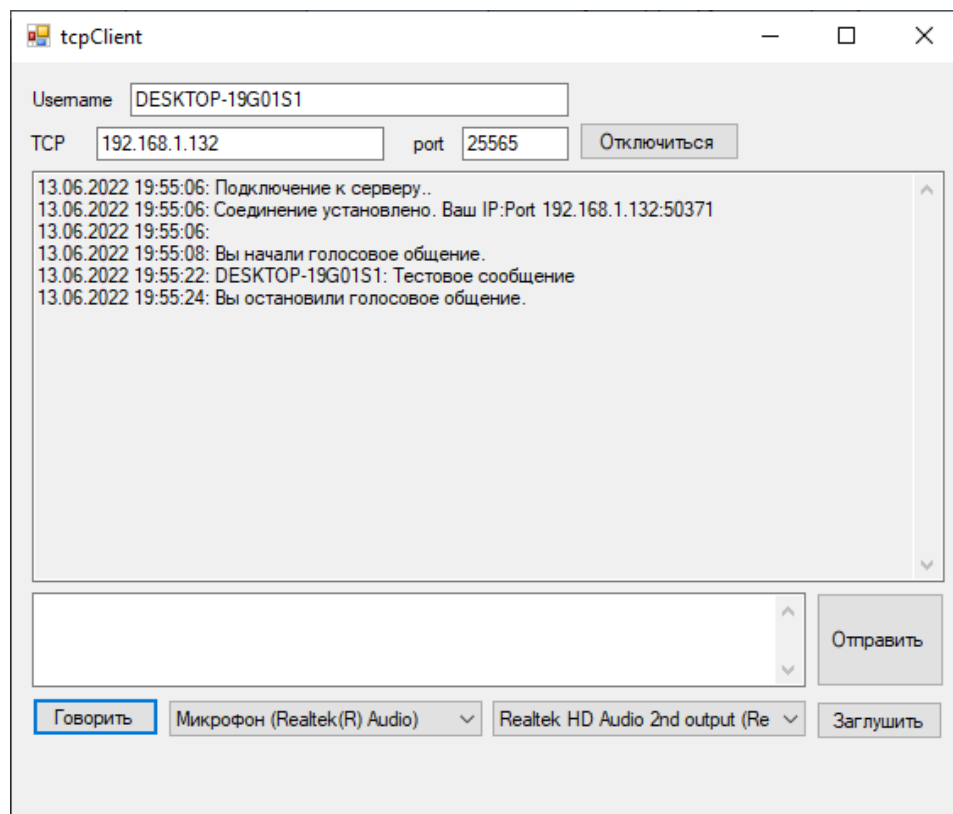


Рисунок 1.26 - Отправка текстового сообщения

После окончания общения на этом сервере можно подключиться к другому серверу. Для этого сначала необходимо нажать кнопку «Отключиться», ввести данные другого сервера, а затем подключиться к нему.

При отключении также будет написано, почему произошло отключение.

Действие программы после отключения показано на рисунке 1.27.

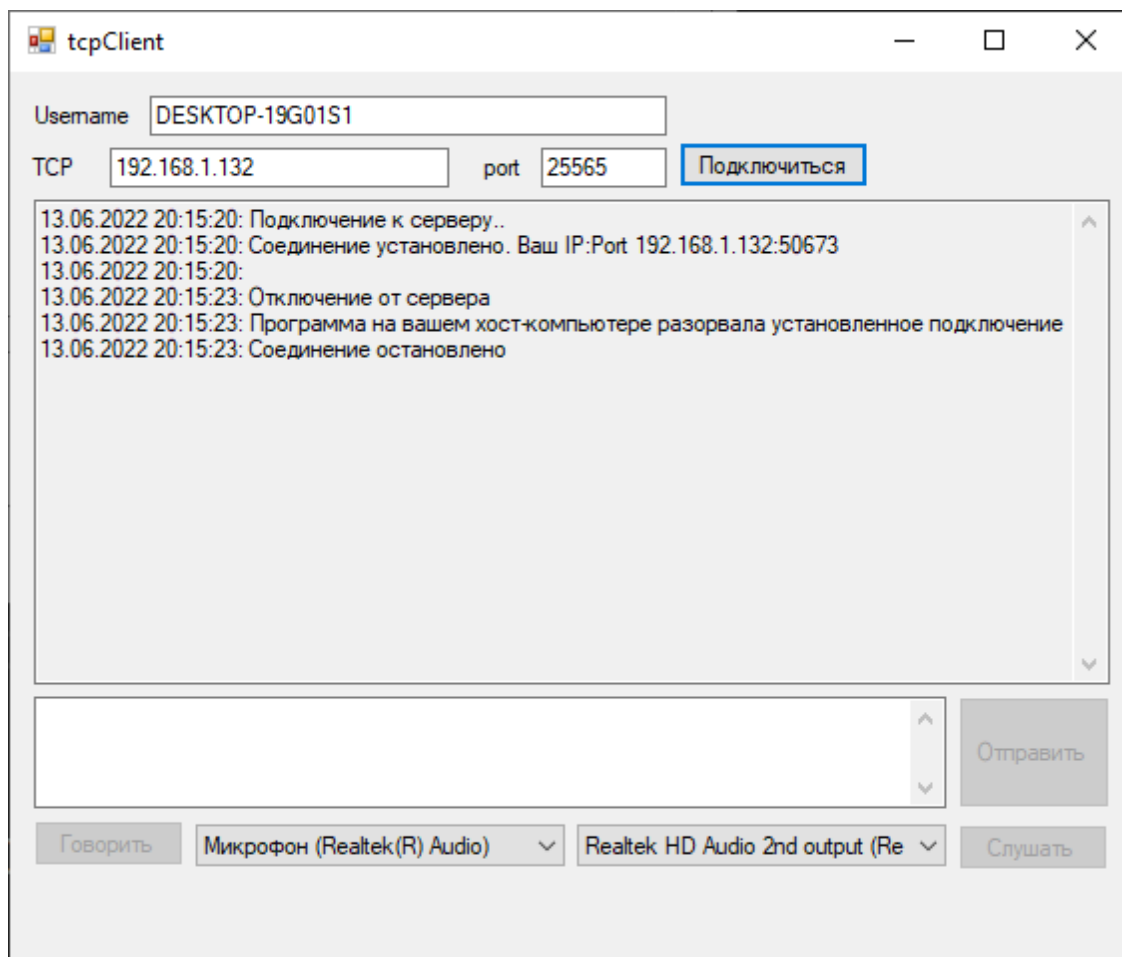


Рисунок 1.27 - Отключение от сервера

В ходе тестирования и опытной эксплуатации установлено что программное обеспечение решает все возложенные на него задачи: позволяет пользователям общаться посредством передачи голосовой и текстовой информации. Приложение корректно работает как в локальной, так и в глобальной вычислительной сети, решая поставленные ему задачи.

2 Безопасность жизнедеятельности

Работа программистов характеризуется высоким умственным и зрительным напряжением, низкой физической активностью, фиксированной позой и однообразными задачами. Кроме того, необходимо учитывать внешние факторы, такие как освещение, вентиляция, уровень шума, температура и воздействие электромагнитного излучения. Эти факторы влияют на самочувствие работников, их производительность и точность результатов труда.

В соответствии с Законом РСФСР "О гигиене и эпидемиологической охране человека" должен осуществляться контроль за соблюдением гигиенических норм и санитарно-гигиенических, лечебно-профилактических мероприятий по предупреждению профессиональных заболеваний.

Прежде всего, работники должны осознавать возможные негативные последствия несоблюдения безопасных условий труда и, по возможности, организовывать свою работу таким образом, чтобы свести эти последствия к минимуму.

При работе с компьютерами разработчики могут подвергаться воздействию *POD* и *ERP* эффектов, таких как повышенный уровень шума, повышенная температура окружающей среды, недостаточное или неадекватное естественное освещение, плохое освещение рабочей зоны, плохой микроклимат и состав воздуха в рабочей зоне, воздействие тока, статического электричества и электромагнитных полей.

Нельзя забывать и о влиянии психофизиологических факторов, таких как умственная перегрузка, перегрузка зрительного и слухового анализаторов, монотонность работы и эмоциональная перегрузка.

Разработчики работают менее гибко, что может негативно сказаться на опорно-двигательном аппарате. Неадекватная эргономика и ошибки в

проектировании рабочего места увеличивают нагрузку на систему. Давайте рассмотрим этот вопрос подробнее.

2.1 Эргономические требования к рабочему месту

Для обеспечения надлежащих условий труда гигиенические требования предусматривают, что производственное помещение каждого работника должно иметь минимальный объем 15 кубических метров, площадь пола не менее 4,5 квадратных метров и высоту потолка не менее 3,3 метра. В соответствии со строительными нормами пол в этой зоне должен быть выполнен из негорючих материалов и отвечать гигиеническим, техническим и эксплуатационным требованиям.

Планировка рабочего места играет важную роль, поскольку она должна отвечать требованиям комфорта, экономии энергии и времени, рационального использования пространства и легкого доступа к ИТ-оборудованию, а также требованиям охраны здоровья и безопасности. Антропометрические данные учитываются при проектировании рабочего места. Конструкция рабочей зоны основана на окружности мужской руки 40 см. Проксимальная зона - это область, охватываемая рукой, когда локоть прижат к телу, а дистальная зона - это область вытянутой руки. Это обеспечивает равномерную нагрузку на все группы мышц и исключает ненужные и бесполезные движения. Не сидите на расстоянии более 60-70 см от экрана (примерно на расстоянии вытянутой руки) - Избегайте использования сильных источников света за экраном (окна, искусственное освещение и т.д.), при этом сильные источники света не должны создавать блики на экране.

Все регулируемые опоры разработаны таким образом, чтобы обеспечить наиболее удобное положение перед компьютером.

Компьютерные компоненты, которые непосредственно контактируют с программистом, также играют важную роль в его работе. Эти компоненты

включают, в частности, устройства ввода и вывода, такие как мониторы, клавиатуры и мыши.

После тщательного изучения информации об электромагнитном воздействии ЭЛТ-дисплеев на человека был сделан вывод, что для стандартных компьютеров следует использовать плоские ЖК-дисплеи. Основными преимуществами по сравнению с ЭЛТ-дисплеями являются:

- он не производит сильного излучения, характерного для ЭЛТ-дисплеев;
- плоский экран без искажения изображения;
- отсутствие мерцания, как у ЭЛТ-дисплеев;
- ЖК-мониторы занимают меньше места в офисе;
- расширенная настройка яркости, контрастности, насыщенности, световой температуры и цветового баланса.

2.2 Воздействие шума на программиста

Источниками шума могут быть системы вентиляции и кондиционирования воздуха или компьютерная периферия (планшеты, принтеры и т.д.). Длительное воздействие этих звуков негативно сказывается на эмоциональном состоянии работников.

Согласно ГОСТ 12.1.003-76, эквивалентный уровень звука ССБТ не должен превышать 50 дБА. Для достижения такого уровня шума рекомендуется использовать звукопоглощающие настенные покрытия.

Могут быть рекомендованы следующие меры по снижению шума:

- покрытие потолков и стен звукопоглощающим материалом (снижает уровень шума на 6-8 дБ);
- защита рабочего места (перегородки, мембраны);
- размещайте наименее шумное оборудование в компьютерных залах;
- рациональная планировка помещения.

Защита от шума должна выполняться в соответствии с ГОСТ 12.1.003-76, а звукоизоляция здания должна соответствовать требованиям раздела СНиП "Защита от шума. Стандарты проектирования".

2.3 Опасность повышенного уровня напряженности электромагнитного поля

Электромагнитные поля, характеризующиеся силой электрических и магнитных полей, являются наиболее вредными для человеческого организма.

ПЭВМ являются источниками таких излучений как:

- мягкого рентгеновского;
- ультрафиолетового 200-400 нм;
- видимого 400-700 нм,
- ближнего инфракрасного нм;
- радиочастотного 3 кГц-30 МГц;
- электростатических полей.

Ультрафиолетовое излучение полезно в небольших количествах, но в больших дозах оно вызывает воспаление кожи, головные боли и боль в глазах. Инфракрасное излучение вызывает перегрев тканей человека (особенно хрусталика глаза), что повышает температуру тела. Напряженность электростатического поля не должна превышать 20 кВ/м. Электростатический потенциал на поверхности не должен превышать 500 В. Если поле очень сильное, следует сократить время работы за компьютером, делать 15-минутный перерыв в течение полутора часов и, конечно, носить защитную одежду.

Сила электромагнитного поля может быть опасной. На расстоянии 5-10 см от монитора и экрана уровень электрической составляющей может достигать 140 В/м, что значительно превышает значения, разрешенные

стандартом СанПиН.2 Максимально допустимые значения характеристик ЭМП приведены в таблице 1.1.

Таблица 1.1 - Предельно допустимые значения характеристик ЭМП

Наименование параметров	Допустимое значение
Напряженность электромагнитного поля по электрической составляющей на расстоянии 50 см от поверхности видеомонитора	10 В/м
Напряженность электромагнитного поля по магнитной составляющей на расстоянии 50 см от поверхности видеомонитора	0,3 А/м
Напряженность электростатического поля не должно превышать (для взрослых пользователей)	20 кВ/м
Напряженность электромагнитного поля на расстоянии 50 см вокруг ВДТ по электрической составляющей должна быть не более: - в диапазоне частот 5 Гц - 2 кГц; - в диапазоне частот кГц.	25 В/м 2,5 В/м
Плотность магнитного потока должна быть не более: - в диапазоне частот 5 Гц – 2 кГц; - в диапазоне частот 2 – 400 кГц.	250 нТл 25 нТл
Поверхностный электростатический потенциал не должен превышать	500 В

Чтобы предотвратить внедрение небезопасных методов, все дисплеи должны проверяться на соответствие нормам безопасности (например, международным стандартам *MRP 2*, *TCO 99*).

Поскольку работа программиста относится к категории Б - творческая работа в диалоге с компьютером - по типу задания и ко II категории по нагрузке (СанПиН 2.2.542-96), я предлагаю сократить время работы с компьютером за счет введения перерывов, общая продолжительность которых должна составлять 50 минут в течение восьмичасового рабочего дня.

В рабочей зоне содержится большое количество легковоспламеняющихся материалов. Согласно нормам пожарной безопасности, такие здания относятся к классу "В" (СНиП - 2 - М2 - 72).

Пожарная безопасность гарантирована:

- система противопожарной защиты;
- система противопожарной защиты;
- система противопожарной защиты.

Система противопожарной защиты:

- пожарная безопасность;
- противопожарная система;
- электрическая система;
- противопожарная система.

Для предотвращения пожаров необходимо использовать оборудование, соответствующее правилам пожарной безопасности.

Все помещения в отсеке должны иметь степень огнестойкости 1-2 (СН-245-71). Облицовка и поверхности должны быть огнестойкими. Для облицовки обычно используются кирпич, железобетон, стекло и т.д. Использование дерева должно быть ограничено, а если оно используется, то должно быть пропитано огнезащитным составом.

Комнаты, коридоры и рабочие зоны должны быть освобождены от всех видов предметов: бумаги, оборудования и т.д. Следует избегать использования макулатуры от печатных машин. Вся макулатура и другие

излишки бумаги должны быть своевременно удалены из печатного оборудования.

В отделе запрещается использовать огнетушители, содержащие воду, пену или сухой химический порошок.

Углекислотные огнетушители должны использоваться для тушения небольших локализованных пожаров и располагаться в легкодоступных местах.

Основными средствами пожаротушения являются переносные огнетушители и обычное углекислотное оборудование ОУ-2 и ОУ-5. Преимущество углекислотных огнетушителей в том, что они высокоэффективны и не повреждают электронное оборудование. Кроме того, углекислый газ не проводит электричество, что важно для тушения пожаров в помещениях, где установлено электрооборудование. В помещениях с компьютерным оборудованием должен быть один углекислотный огнетушитель на 40-50 м², но не менее двух огнетушителей на помещение (ГОСТ - 12.1 - 004 - 76).

Средством обнаружения и оповещения при пожаре являются датчики, которые устанавливаются в вытяжных воздуховодах, в подпольном пространстве, в хранилищах носителей информации и других помещениях.

2.4 Освещение

Работа программиста проходит в помещении, поэтому освещение в этих комнатах должно быть близким к оптимальным условиям для видимости солнечного света. Требования к освещению компьютерных залов следующие: для высокоточной визуальной работы общее освещение должно составлять 300 лк, а комбинированное освещение - 750 лк; для работы средней точности соответствующие требования составляют 200 лк и 300 лк. Кроме того, все поле зрения должно быть достаточно равномерно освещено - это основное гигиеническое требование. Другими словами, яркость

					ПГУ 1.090301.27.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

помещения и яркость экрана компьютера должны быть примерно одинаковыми, так как яркий свет в периферийном поле зрения значительно увеличивает напряжение глаз и тем самым приводит к усталости.

Согласно СанПиН, в компьютерном классе должно быть как естественное, так и искусственное освещение. В этом случае дополнительное искусственное освещение используется не только в темную часть суток, но и в течение дня. Естественное освещение должно поступать в основном через проемы, выходящие на север и северо-восток, а коэффициент естественного освещения (КЕО) должен составлять не менее 1,2% в районах с ровным снежным покровом и 1,5% в других районах. Искусственное освещение в компьютерных залах должно иметь единую схему общего освещения. Источниками искусственного света обычно являются люминесцентные лампы ЛБ или ДРЛ, подключенные попарно к светильникам, которые должны быть равномерно распределены по рабочим поверхностям.

Общее освещение должно состоять из постоянных или прерывистых светильников по краям рабочих мест, параллельно полю зрения пользователя, а компьютеры должны располагаться рядами. Если компьютеры расположены по кругу, ряд светильников должен располагаться над столом, около его переднего края и обращен к пользователю.

Блики от рабочих поверхностей можно уменьшить, выбрав соответствующие типы освещения и расположив рабочие места относительно источников естественного и искусственного света таким образом, чтобы блики на экране не превышали 40 кд/кВ.м, а яркость вариаций отраженного света не превышала 200 кд/кВ.м.

Для того чтобы блики от общего освещения не отражались на экранах, с обеих сторон экрана в направлении просмотра должны быть установлены антибликовые сетки, специальные экранные фильтры, защитные кожухи или источники света.

					ПГУ 1.090301.27.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

Поскольку дисплей также является источником света, постоянное считывание информации утомляет глаза, особенно если яркость дисплея установлена слишком высокой. Мерцание, вызванное низкой частотой обновления экрана, также неприятно для глаз. Для уменьшения мерцания экрана рекомендуется частота кадров не менее 75 Гц для ЭЛТ и частота кадров не менее 60 Гц для ЖК-дисплеев.

					ПГУ 1.090301.27.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения выпускной квалификационной работы было разработано клиент-серверное приложение для голосового и текстового общения в сети интернет.

Были получены навыки программирования на языке *C#* и навыки работы с *.NET Framework*. Получены знания в области программной обработки звуковой информации. Также были получены навыки работы с протоколом *TCP/IP* и освоены методы передачи информации по сети.

В ходе выполнения выпускной квалификационной работы были разработаны клиентское приложение для записи и передачи звука на сервер и получения и воспроизведения звука с сервера. Также было разработано серверное приложение для получения звуковой информации от клиентов, её обработки и рассылки.

В дальнейшем программу можно усовершенствовать, добавив возможность записи и отправки звука самой системы, возможность регистрации и авторизации пользователей на сервере, возможность обмена файлами. Также можно усовершенствовать обработку звука на сервере для удаления собственного голоса каждого пользователя.

Список используемых источников

1. Полное руководство по *C#* и *.NET Framework*. [Электронный ресурс] / Режим доступа: <https://metanit.com/sharp/> свободный. (дата обращения 17.03.2022).
2. Документация к библиотеке *Naudio*. [Электронный ресурс] / Режим доступа: <https://github.com/naudio/NAudio> свободный (дата обращения 23.03.2022).
3. Техническая документация Майкрософт. [Электронный ресурс] / Режим доступа: <https://docs.microsoft.com/ru-ru/> свободный (дата обращения 12.03.2022).
4. Стек протоколов *TCP/IP*. [Электронный ресурс] / Режим доступа: https://www.opennet.ru/docs/RUS/ip_network/ свободный (дата обращения 9.04.2022).
5. Джеффри Рихтер. *Windows для профессионалов*. - 4-е изд. Русская редакция, 2008. - 720 с.

Приложение А

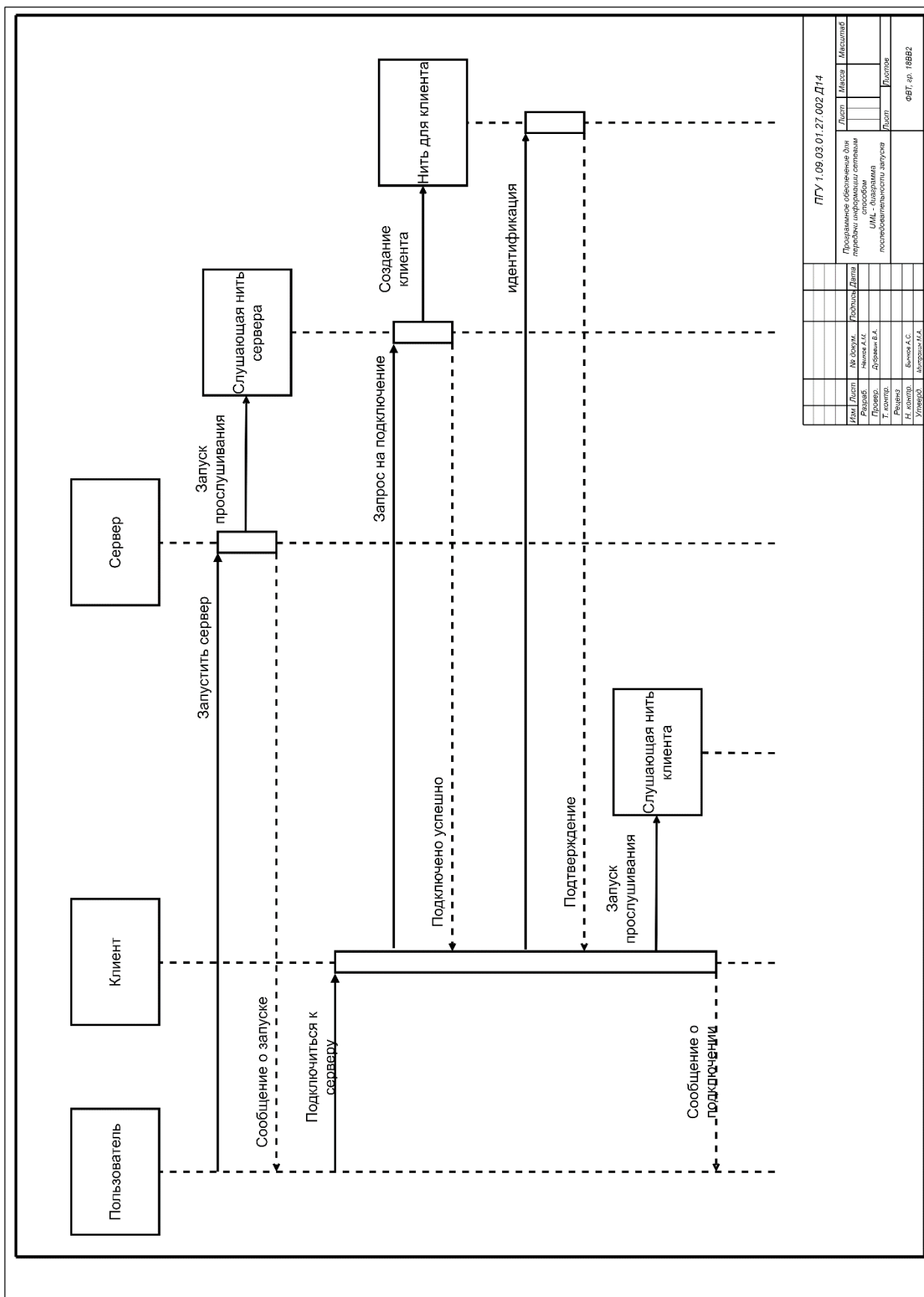
(Обязательное)

Программное обеспечение для передачи информации сетевым способом

UML – диаграммы

					ПГУ 1.090301.27.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – Диаграмма последовательности запуска



ПУ 1.09.03.01.27.002 Д14									
Имя	Лист	№ докум.	Подпись	Дата	Программное обеспечение для первичной обработки данных ИМ - Авария				
Рисовал	Человек А.И.	Проверил	Давыдов В.А.		последовательности запуска				
Т. контр.		Рисовал			Листов				
Н. контр.		Внес А.С.			Листов				
Утвердил		Исполнил М.А.			ФЕГ, стр. 18882				

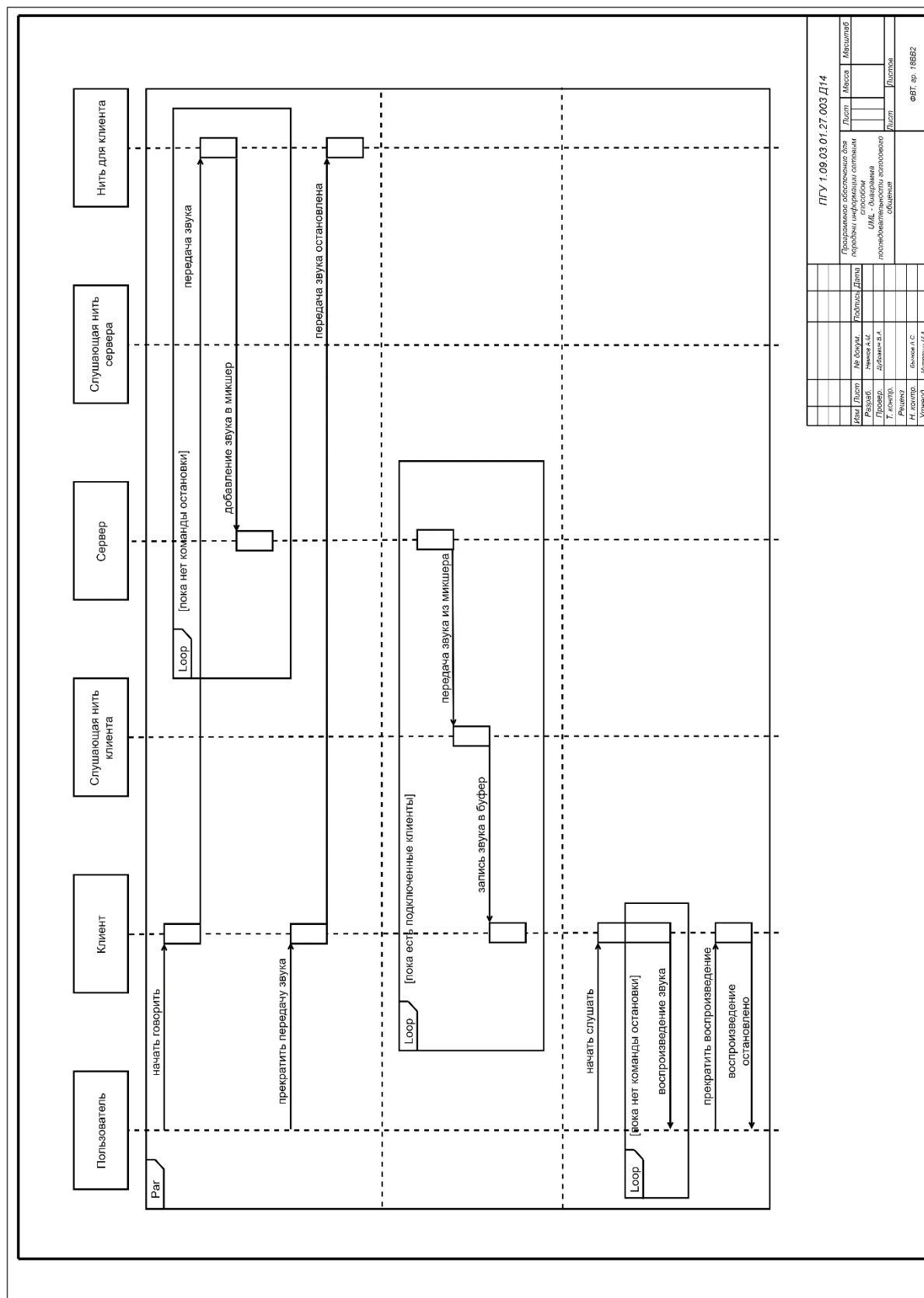
Изм.	Лист	№ докум.	Подпись	Дата

ПУ 1.090301.27.001 ПЗ

Лист

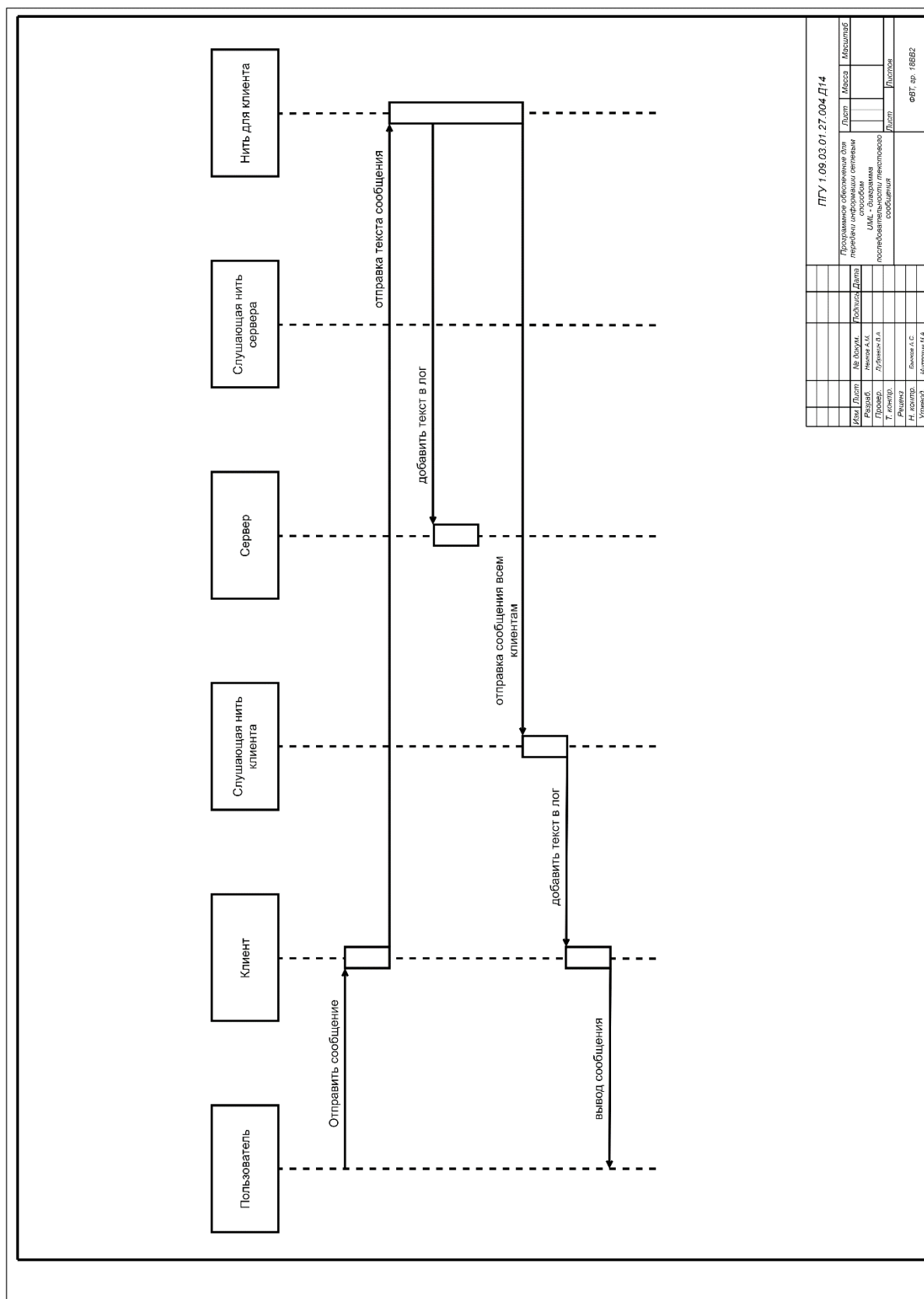
56

Приложение А.2 – Диаграмма последовательности голосового общения

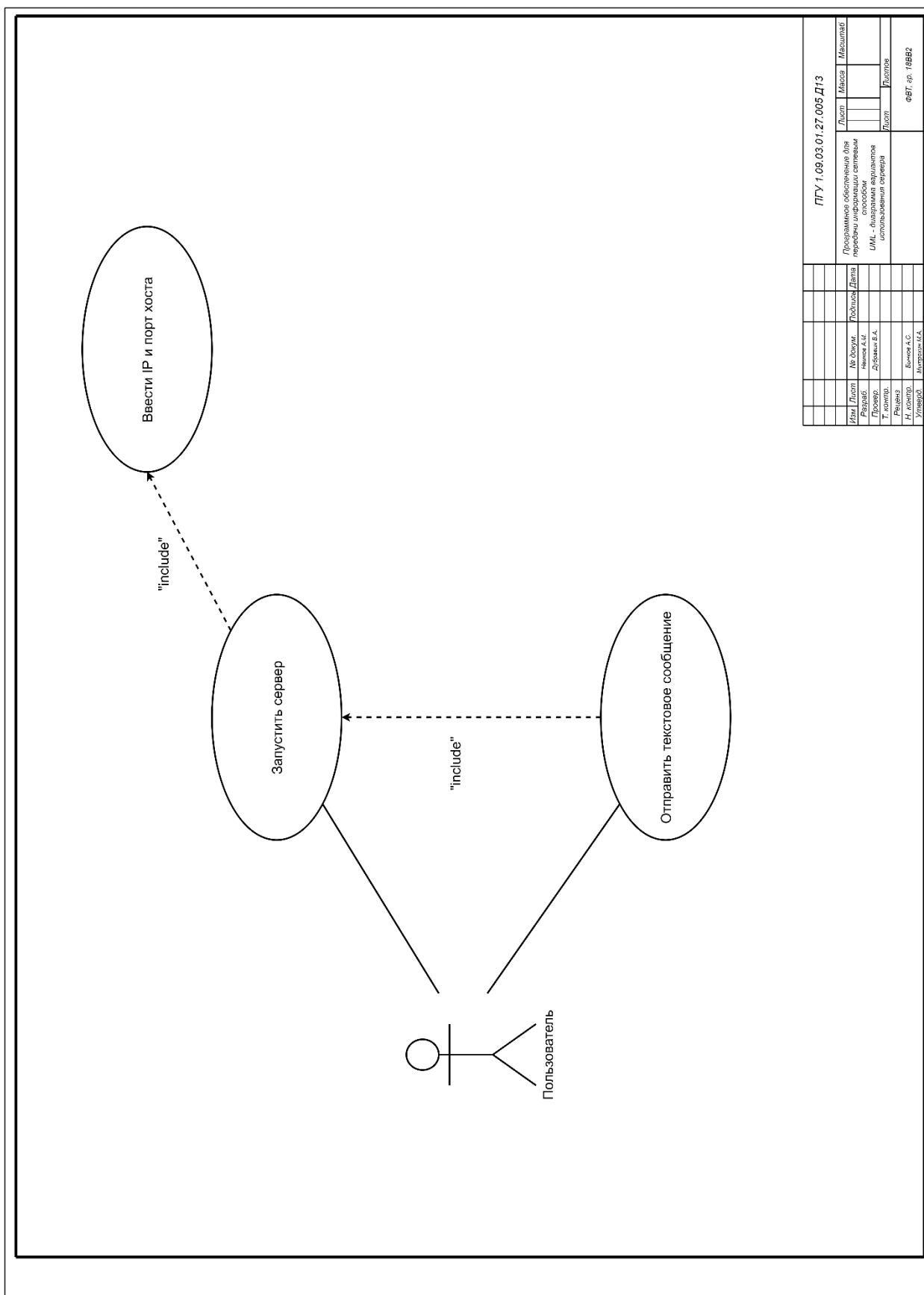


ПУУ 1.09.03.01.27.003 Д14									
Программное обеспечение для передачи информации сетевым способом ИМЭ пользовательности аппаратуры общения									
Изм.	Лист	№ докум.	Титул	Дата	Лист	Масштаб			
Разработ.	Начисл.	Лист	Лист	Лист	Лист	Лист			
Провер.	Лист	Лист	Лист	Лист	Лист	Лист			
Т. экстр.	Лист	Лист	Лист	Лист	Лист	Лист			
Рисунки	Лист	Лист	Лист	Лист	Лист	Лист			
И. экстр.	Лист	Лист	Лист	Лист	Лист	Лист			
Лист	Лист	Лист	Лист	Лист	Лист	Лист	ФЭТ, зр. 18952		

Приложение А.3 – Диаграмма последовательности текстового сообщения



Приложение А.4 – Диаграмма вариантов использования сервера



ПГУ 1.09.03.01.27.005 Д13									
Мод.	Лист	М. докум.	Получен	Дата	Программное обеспечение для передачи информации сетевым УИМ - диаграмма вариантов использования сервера				
Разработчик	Исполнитель	Содержание в.д.	Содержание в.д.	Содержание в.д.	Листов				
Проверка	Т. контро.	Внесение в.д.	Внесение в.д.	Внесение в.д.	Листов				
Рецензия	И. контро.	Внесение в.д.	Внесение в.д.	Внесение в.д.	Листов				
Утверждение	И. контро.	Внесение в.д.	Внесение в.д.	Внесение в.д.	Листов				

Ф.ВТ. стр. 188В2

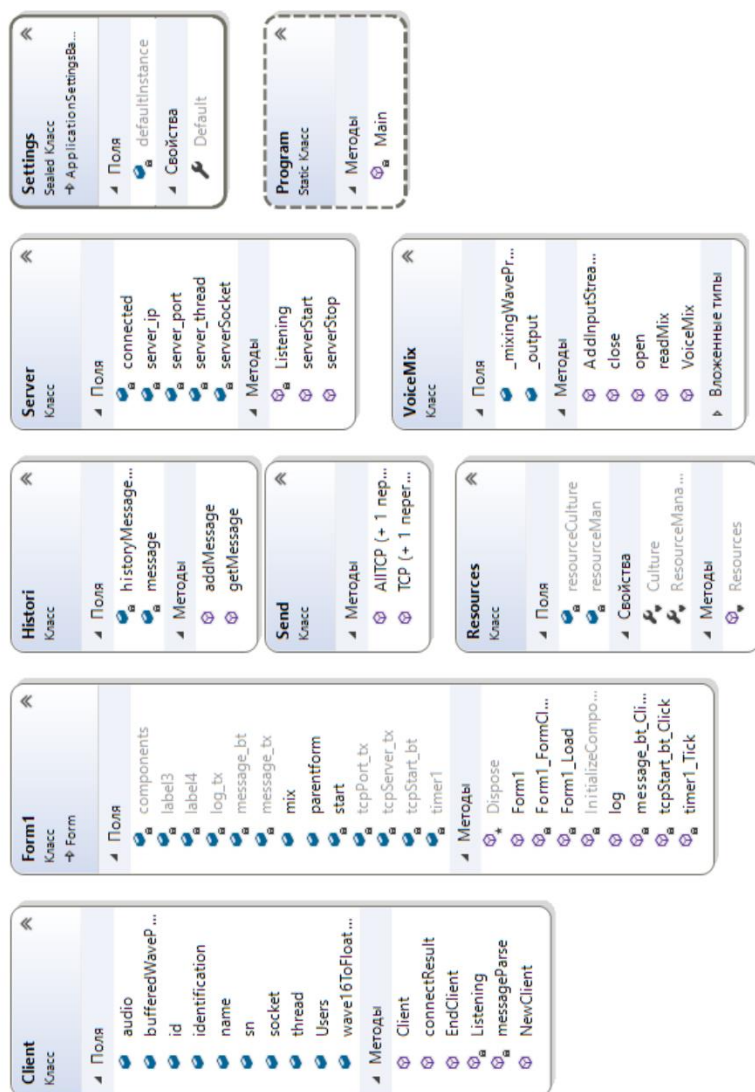
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.27.001 ПЗ

Лист

59

Приложение А.5 – Диаграмма классов сервера



ПГУ 1.09.03.01.27.006 Д5									
Изм.	Лист	№ докум.	Подпись	Дата	Лист	Масштаб	Программное обеспечение для параллельной информации Диаграмма классов сервера		
Разработ.	Провер.	Начальн. к.м.	Добавлен В.А.		Листов				
Т. контр.	Рисован	Н. контр.	Должен А.С.						
Утвержден			Александр К.А.				087 ар. 18862		

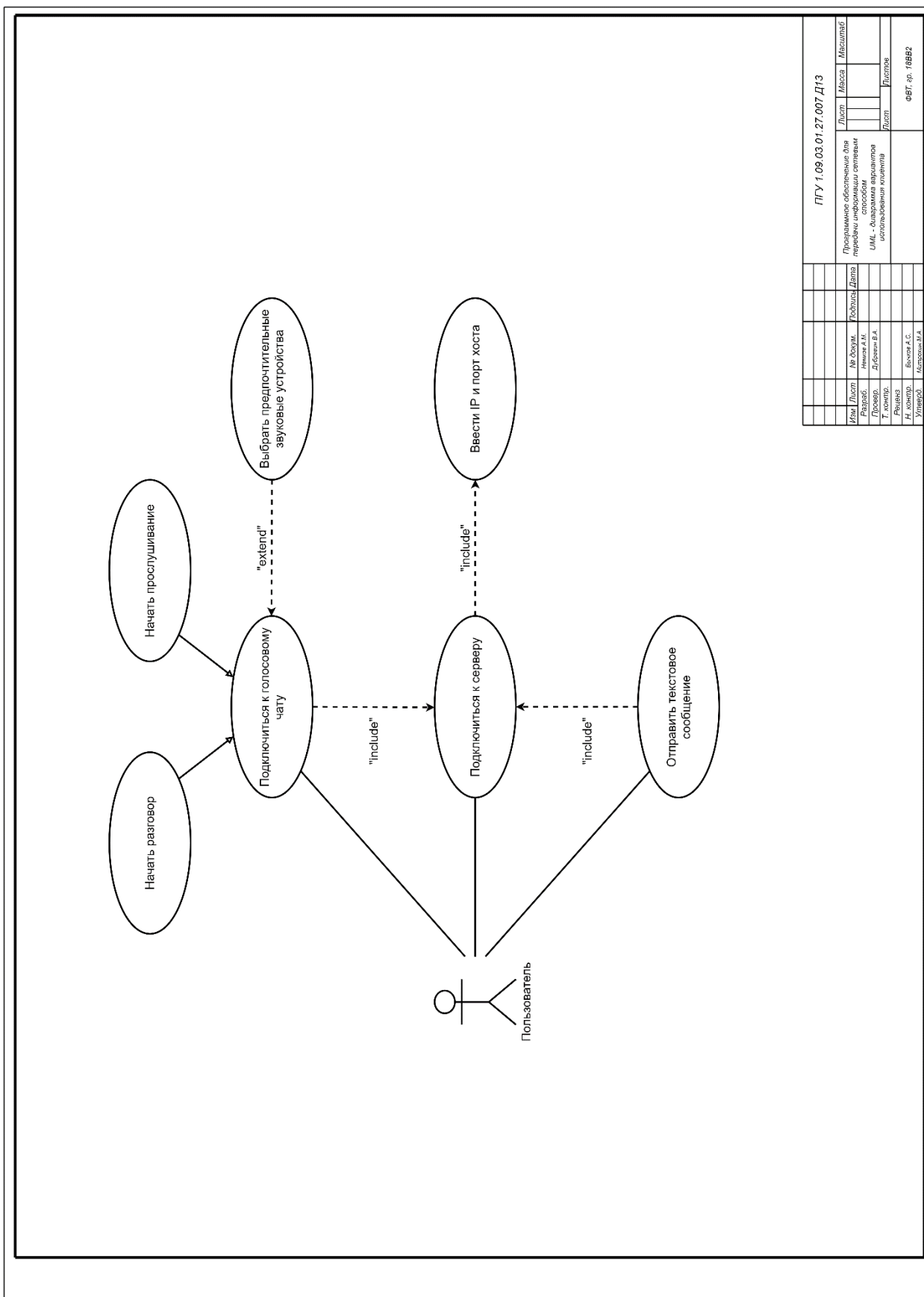
Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ 1.090301.27.001 ПЗ

Лист

60

Приложение А.6 – Диаграмма вариантов использования клиента

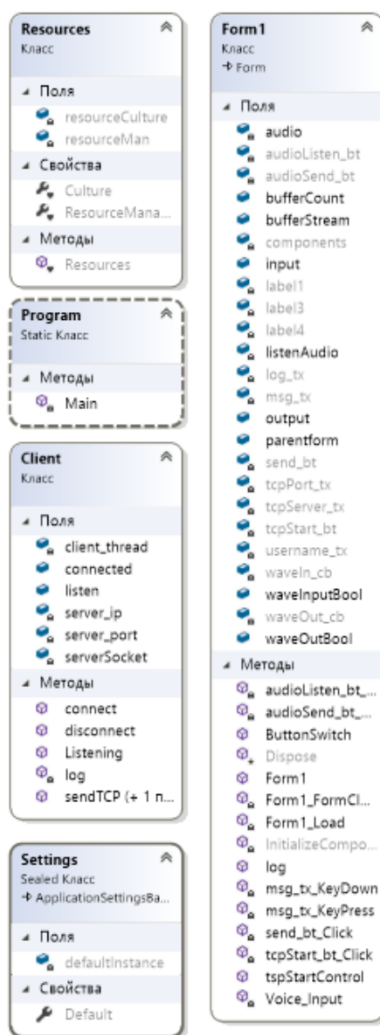


ПГУ 1.09.03.01.27.007 Д13									
Имя		Фамилия	Имя	Фамилия	Программное обеспечение для передачи информации сетевым УИМ - Адаптивная архитектура использования клиента				
Родное		Имя	Фамилия	Имя	Листов				
Полное		Имя	Фамилия	Имя	Листов				
Т. номер		Имя	Фамилия	Имя	Листов				
Результат		Имя	Фамилия	Имя	Листов				
Н. номер		Имя	Фамилия	Имя	Листов				
Утвердил		Имя	Фамилия	Имя	Листов				

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.27.001 ПЗ

Приложение А.7 – Диаграмма классов клиента



					ПГУ 1.09.03.01.27.008 Д7				
Изм.	Лист	№ докум.	Подпись	Дата	Программное обеспечение для обработки информации сетевым способом Диаграмма классов клиента				
Разраб.	Нейков А.И.								
Провер.	Дубравин В.А.								
Т. контр.									
Реценз.									
Н. контр.	Бачков А.С.								
Утверд.	Митрошкин И.А.								
					Лист 1 из 1				
					ФВТ, гр. 188В2				

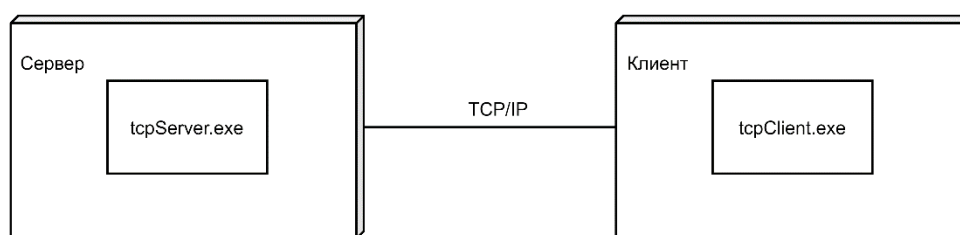
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.27.001 ПЗ

Лист

62

Приложение А.8 – Диаграмма развертывания



						ПГУ 1.09.03.01.27.009 Д19					
Изм.	Лист	№ докум.	Подпись	Дата		Программное обеспечение для передачи информации сетевым способом UML - диаграмма развертывания	Лист	Масса	Масштаб		
Разраб.		Нейков А.М.									
Провер.		Дубровин Б.А.									
Т. контр.											
Реценз.							Лист			Листов	
Н. контр.		Бычков А.С.									
Утверд.		Митрохин И.А.									

ФВТ, гр. 18ВВ2

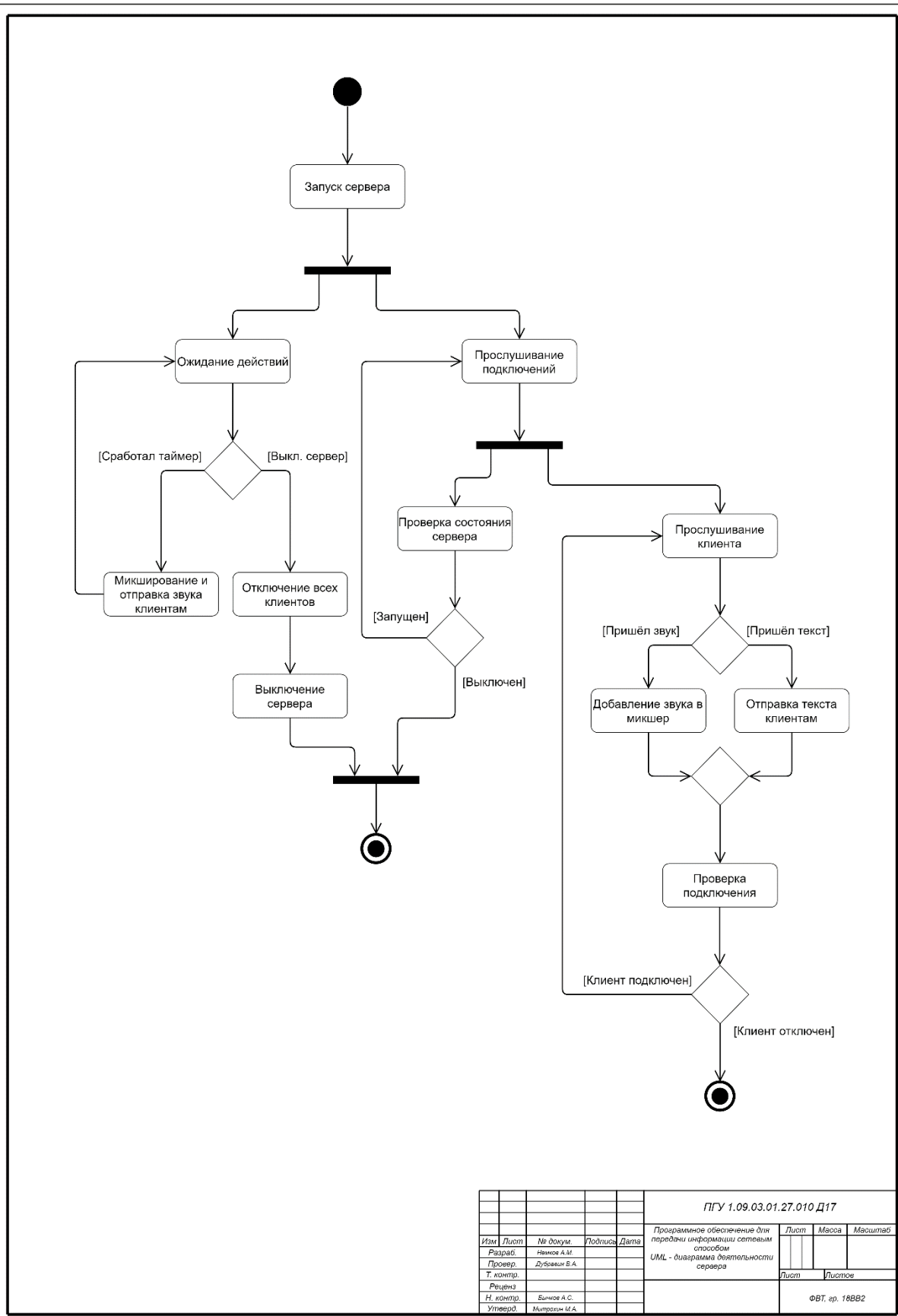
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.27.001 ПЗ

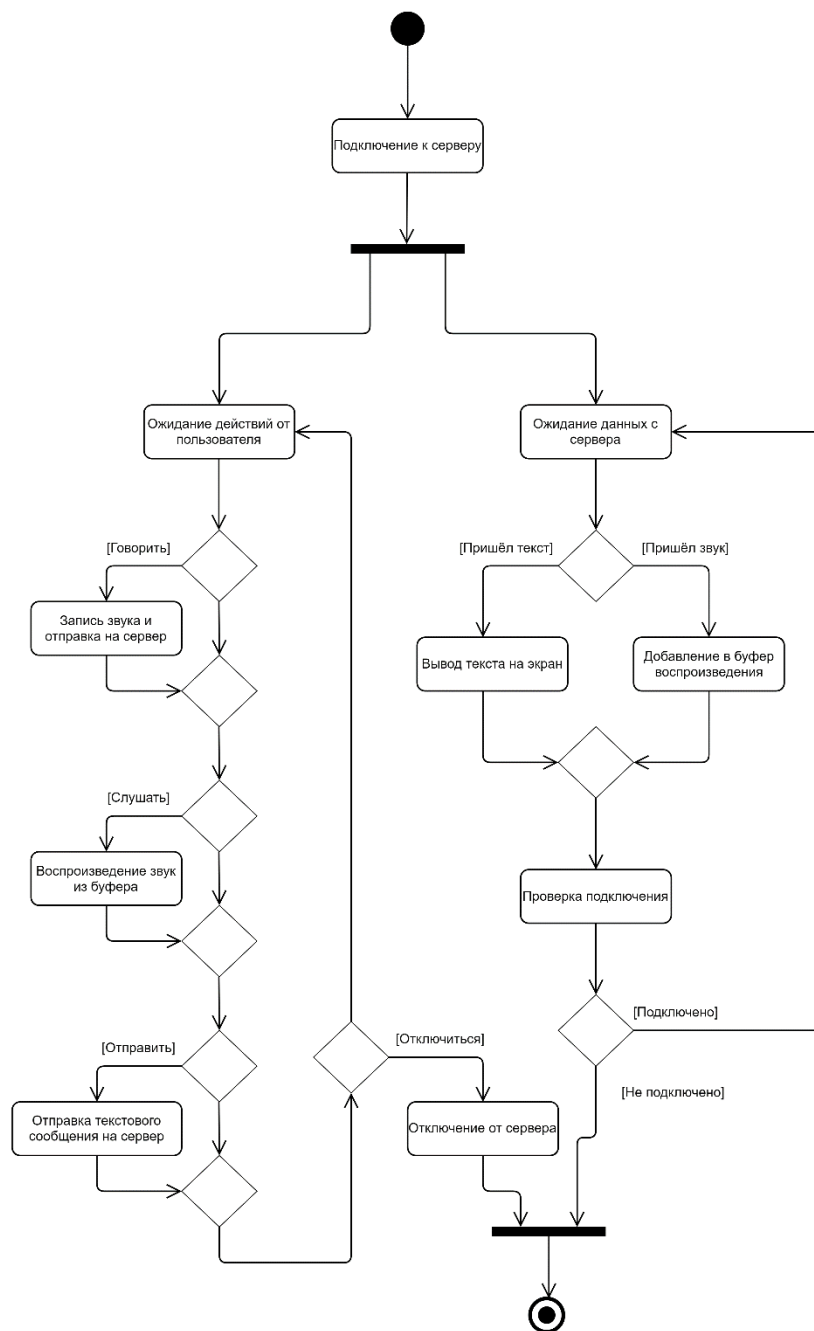
Лист

63

Приложение А.9 – Диаграмма деятельности сервера



Приложение А.10 – Диаграмма деятельности клиента



					ПГУ 1.09.03.01.27.011 Д17			
Изм.	Лист	№ докум.	Подпись	Дата	Программное обеспечение для передачи информации сетевым способом UML - диаграмма деятельности клиента	Лист	Масса	Масштаб
Разраб.		Немков А.И.						
Провер		Дубравин Б.А.						
Т. контр.						Лист	Листов	
Реценз								
Н. контр.		Бычков А.С.			ФВТ, гр. 18ВВ2			
Утверд.		Митрошкин И.А.						

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(Обязательное)

Программное обеспечение для передачи информации сетевым способом

Листинг программы

					ПГУ 1.090301.27.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

Файл Client.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using NAudio.Wave;
using System.IO;

namespace tcpClient
{
    class Client
    {
        #region Переменные
        public static bool connected = false;
        public static bool listen = true;
        //IP сервера
        private static string server_ip = "";
        //Порт сервера
        private static int server_port = 0;
        //сокет для приема (протокол TCP)
        private static Socket serverSocket = new
Socket(IPAddress.Any.AddressFamily, SocketType.Stream,
ProtocolType.Tcp);
        //Прослушивающий поток для клиента
        private static Thread client_thread;
        #endregion

        #region Соединение с сервером
        /// <summary>
        /// Соединение с сервером, true в случае успешного
соединения, false в случае отказа соединения
        /// </summary>
        public static bool connect(string host, int port, string
username, string sn)
        {
            try
            {
                if (!connected)
                {
                    server_ip = host; //Присвоить IP серверу
                    server_port = port; //Присвоить порт серверу
                    serverSocket = new
Socket(IPAddress.Parse(server_ip).AddressFamily,
SocketType.Stream, ProtocolType.Tcp);
                    IPEndPoint ipEndPoint = new
IPEndPoint(IPAddress.Parse(server_ip), server_port);
```

```

        try { serverSocket.Connect(ipEndPoint); }
catch { } //Соединение с сервером

        #region Соединение установлено
        if (serverSocket.Connected) //Соединение
установлено
        {
            //Отправить запрос на авторизацию
            sendTCP("#connect&" + username + "&" +
sn);

            //Получаем ответ
            byte[] bytes = new byte[1024];
            int bytesRec =
serverSocket.Receive(bytes);
            string data =
Encoding.Unicode.GetString(bytes, 0, bytesRec);
            #region Авторизация пройдена
            //Авторизация пройдена
            if (data.Contains("#true"))
            {
                connected = true;

                client_thread = new
Thread(Listening); //Присвоить функцию прослушивания сообщений
серверу

                client_thread.IsBackground = true;
                client_thread.Start(); //Запустить
поток прослушки

                string answer = data.Split('&')[1];
//Подробности соединения
                log(answer); //Отобразить
подробности в логе

                return true;
            }
            #endregion
            #region Сервер вернул отказ в
авторизации
            //Сервер вернул отказ в авторизации
            else if (data.Contains("#false"))
            {
                string answer = data.Split('&')[1];
//Подробности соединения
                log(answer); //Отобразить
подробности в логе

                disconnect();
                return false;
            }
            #endregion
            #region Сервер вернул неизвестный ответ

```

```

//Авторизация не пройдена из за
неизвестного ответа
else { disconnect(); return false; }
#endregion
}
#endregion
#region Соединение не установлено
if (!serverSocket.Connected)//Соединение
установлено
{
    log("Не удастся установить соединение,
сервер недоступен ");
    return false;
}
#endregion
}
return false;
}
catch (Exception ex) { log("Неизвестная ошибка " +
ex); return false; }
}
#endregion

#region Остановить соединение с сервером
/// <summary>
/// Разорвать соединение с сервером, true в случае
успешного отключения, false в случае ошибки
/// </summary>
public static bool disconnect()
{
    try
    {
        if (connected)
        {
            connected = false;
            log("Отключение от сервера");
            Form1.parentform.ButtonSwitch(false);
            if (Form1.input != null) {
Form1.input.StopRecording(); } //Остановить передачу голоса
serverSocket.Close();

Form1.parentform.tspStartControl("Подключиться");

            return true;
        }
        return false;
    }
    catch { log("Сервер недоступен"); return false; }
}
#endregion

```

```

#region Прослушивание входящих подключений
static public void Listening()
{
    Dictionary<string, WaveOut> WaveOut = new
Dictionary<string, WaveOut>();
    Dictionary<string, BufferedWaveProvider> bufferAudio
= new Dictionary<string, BufferedWaveProvider>();

    while (connected == true)
    {
        try
        {
            byte[] buffer = new byte[8000];
            int bytesRec = serverSocket.Receive(buffer);
            Array.Resize(ref buffer, bytesRec);
            string data =
Encoding.Unicode.GetString(buffer, 0, bytesRec);
            #region Новое текстовое сообщение
            if (data.Contains("#msg"))
            {
                string message = data.Remove(0, 5);

//Сообщение

                log(message);
            }
            #endregion

            if (Form1.waveOutBool &&
!data.Contains("#msg"))
            {
                // Вывод количества данных в буфере

                for (int i = 0; i < 1; i++)
                {

Form1.bufferStream[i].AddSamples(buffer, 0, buffer.Length);

                    break;

                }

            }

        }
        catch (Exception ex) { log(ex.Message);
disconnect(); }
    }
    log("Соединение остановлено");
}

```

```

        #endregion

        #region Отправить TCP сообщение
        /// <summary>
        /// TCP отправка текстового сообщения в кодировке
Unicode
        /// </summary>
        public static bool sendTCP(string text)
        {
            try
            {
                byte[] buffer = Encoding.Unicode.GetBytes(text);
                int bytesSent = serverSocket.Send(buffer);
                return true;
            }
            catch { log("sendTCP Связь с сервером
преервалась..."); return false; }
        }
        /// <summary>
        /// TCP отправка байтового сообщения
        /// </summary>
        public static bool sendTCP(byte[] bytes)
        {
            try
            {
                int bytesSent = serverSocket.Send(bytes);
                return true;
            }
            catch { log("sendTCP Связь с сервером
преервалась..."); return false; }
        }
        #endregion
        #region Добавить сообщение в лог
        static void log(string text)
        {
            Form1.parentform.log(text);
        }
        #endregion
    }
}

```

Файл Form1.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;

```

```

using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using NAudio.Wave;
using System.Text.RegularExpressions;

namespace tcpClient
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            CheckForIllegalCrossThreadCalls = false; //Доступ к
компонентам не из потока родителя
            parentform = this; //Получить копию контролов формы
        }
        #region Действия при загрузке формы
        private void Form1_Load(object sender, EventArgs e)
        {

            // Получение имени компьютера.
            String host = System.Net.Dns.GetHostName();
            // Получение ip-адреса.
            try
            {
                System.Net.IPAddress ip =
System.Net.Dns.GetHostByName(host).AddressList[1];
                tcpServer_tx.Text = ip.ToString();
            }
            catch (Exception)
            {

            }
            // Показ адреса в label'e.

            username_tx.Text = host;

            #region Список устройств для записи
            int waveInDevices = WaveIn.DeviceCount;
            for (int waveInDevice = 0; waveInDevice <
waveInDevices; waveInDevice++)
            {
                WaveInCapabilities deviceInfo =
WaveIn.GetCapabilities(waveInDevice);
                waveIn_cb.Items.Add(deviceInfo.ProductName);
            }
        }
    }
}

```

```

        }
        if (waveIn_cb.Items.Count > 1) {
waveIn_cb.SelectedIndex = 1; }
        else waveIn_cb.SelectedIndex = 0;
        #endregion
        #region Список устройств для воспроизведения
        int waveOutDevices = WaveOut.DeviceCount;
        for (int waveOutDevice = 0; waveOutDevice <
waveOutDevices; waveOutDevice++)
        {
            WaveOutCapabilities deviceInfo =
WaveOut.GetCapabilities(waveOutDevice);
            waveOut_cb.Items.Add(deviceInfo.ProductName);
        }
        if (waveOut_cb.Items.Count > 1) {
waveOut_cb.SelectedIndex = 1; }
        else waveOut_cb.SelectedIndex = 0;
        #endregion

    }
    #endregion
    #region Действия при закрытие формы
    private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
    {
        Client.disconnect(); //Закрыть соединение
    }
    #endregion
    #region События визуализации
    #region textBox message
    private void msg_tx_KeyDown(object sender, KeyEventArgs
e)
    {
        #region Переход на новую строку
        if (e.Alt && e.KeyCode == Keys.Enter)
        {
            TextBox tx = (TextBox)sender;

            int index = tx.SelectionStart;
            tx.Text = tx.Text.Substring(0,
tx.SelectionStart) + Environment.NewLine +
tx.Text.Substring(tx.SelectionStart);
            tx.SelectionStart = index + 2;
        }
        #endregion
        #region Выделить весь текст
        if (e.Control && e.KeyCode == Keys.A)
        {
            TextBox tx = (TextBox)sender;

            tx.SelectAll();

```

```

        }
        #endregion
    }

    private void msg_tx_KeyPress(object sender,
KeyPressEventArgs e)
    {
        #region Отправить сообщение
        if ((Keys)e.KeyChar == Keys.Enter)
        {
            send_bt_Click(null, null);
            e.Handled = true;
        }
        #endregion
    }
    #endregion
    #endregion

    #region Переменные
    public static Form1 parentform = null; //Копия контролов
формы
    public static bool waveInputBool = true;
//Включение/Отключение микрофона
    public static bool waveOutBool = true;
//Включение/Отключение динамиков
    #endregion
    #region Системные события
    #region Старт/Стоп
    private void tcpStart_bt_Click(object sender, EventArgs
e)
    {
        if (!Client.connected)
        {
            Regex portRegex = new Regex(@"\d{1,5}");
            if (portRegex.IsMatch(tcpPort_tx.Text))
            {
                if (IPAddress.TryParse(tcpServer_tx.Text,
out IPAddress testIp))
                {
                    if (Convert.ToInt32(tcpPort_tx.Text) <=
65535)
                    {
                        log("Подключение к серверу..");
                        if
(Client.connect(tcpServer_tx.Text,
Convert.ToInt16(tcpPort_tx.Text), username_tx.Text, " "))
                        {
                            tcpStart_bt.Text =
"Отключиться";
                            ButtonSwitch(true);

```

```

                                if (waveIn_cb.SelectedIndex ==
0) { log("Микрофон выключен"); waveInputBool = false; }
                                if (waveOut_cb.SelectedIndex ==
0) { log("Динамики выключены"); waveOutBool = false; }

                                if (waveOutBool)
                                {

                                bufferCount = 1;
                                WaveFormat waveFormat = new
WaveFormat(8000, 16, 1);
                                output = new
WaveOut[bufferCount];
                                bufferStream = new
BufferedWaveProvider[bufferCount];

                                for (int i = 0; i <
bufferCount; i++)
                                {
                                output[i] = new
                                output[i].DeviceNumber =
waveOut_cb.SelectedIndex - 1;
                                bufferStream[i] = new
BufferedWaveProvider(waveFormat);
                                bufferStream[i].BufferLength = 32000;
                                bufferStream[i].DiscardOnBufferOverflow = true;
                                output[i].Init(bufferStream[i]);

                                output[i].Play();
                                listenAudio = true;
                                audioListen_bt.Text =
"Заглушить";
                                }
                                }
                                }
                                else MessageBox.Show("Некорректный
порт");
                                }
                                else MessageBox.Show("Некорректный ip-
адрес");
                                }
                                else MessageBox.Show("Некорректный порт");
                                }
                                else
                                {

```

```

        if (Client.disconnect()) {
            tcpStart_bt.Text = "Подключиться";
            ButtonSwitch(false);
            if (output != null)
            {

                audioListen_bt.Text = "Слушать";
                audioSend_bt.Text = "Говорить";

                if (listenAudio)
                {
                    for (int i = 0; i < bufferCount;
i++)
                    {
                        output[i].Stop();
                        output[i].Dispose();
                        output[i] = null;

                    }
                }
                listenAudio = false;
                audio = false;
            }

        }

    }
}
#endregion
#region Отправить текстовое сообщение
private void send_bt_Click(object sender, EventArgs e)
{
    if (!Client.connected) { log("Вы не подключены к
серверу"); return; }
    if(msg_tx.Text.Trim().Length < 1) { log("Введите
сообщение"); return; }
    if (Client.sendTCP("#msg&" + msg_tx.Text)) {
msg_tx.Text = ""; }
}
#endregion
#endregion

#region Функции
public void tspStartControl(string text)
{
    tcpStart_bt.Text = text;
}
#region Добавить сообщение в лог
public void log(string text)
{
    try
    {

```

```

        this.Invoke(new Action(() => {
log_tx.AppendText(DateTime.Now + ": " + text +
Environment.NewLine); }));
    }
    catch { }
}

#endregion

public void ButtonSwitch(bool status)
{
    audioSend_bt.Enabled = status;
    audioListen_bt.Enabled = status;
    send_bt.Enabled = status;
}

#endregion

//поток для нашей речи
public static WaveIn input;
//поток для речи собеседника
public static WaveOut[] output;
//буфферный поток для передачи через сеть
public static BufferedWaveProvider[] bufferStream;
public static int bufferCount;
#region Обработка нашего голоса
private void Voice_Input(object sender, WaveInEventArgs
e)
{
    try
    {
        Client.sendTCP(e.Buffer);
    }
    catch (Exception ex)
    {
    }
}
#endregion

bool audio = false;
bool listenAudio = false;
private void audioSend_bt_Click(object sender, EventArgs
e)
{
    if (waveIn_cb.SelectedIndex == 0) { log("Микрофон
ВЫКЛЮЧЕН"); waveInputBool = false; }
    else waveInputBool = true;
    if (waveInputBool)

```

```

        {
            if (!Client.connected) { log("Вы не подключены к
серверу"); return; }
            if (!audio)
            {
                audio = true;
                audioSend_bt.Text = "Остановить";
                Client.sendTCP("#audio&start");
                log("Вы начали голосовое общение.");
                //создаем поток для записи нашей речи
                input = new WaveIn();
                //определяем его формат - частота
дискретизации 8000 Гц, ширина сэмпла - 16 бит, 1 канал - моно
                input.WaveFormat = new WaveFormat(8000, 16,
1);

                //добавляем код обработки нашего голоса,
поступающего на микрофон
                input.DataAvailable += Voice_Input;
                //Устройство для записи
                input.DeviceNumber = waveIn_cb.SelectedIndex
-1;

                try
                {
                    input.StartRecording();
                }
                catch(Exception ex)
                {
                    MessageBox.Show(ex.Message);
                }
            }
            else
            {
                audio = false;
                audioSend_bt.Text = "Говорить";
                Client.sendTCP("#audio&stop");
                log("Вы остановили голосовое общение.");
                input.StopRecording();

                if (input != null)
                {
                    input.Dispose();
                    input = null;
                }
            }
        }
    }
}

```

```

        private void audioListen_bt_Click(object sender,
EventArgs e)
        {
            if (!listenAudio)
            {
                if (waveOut_cb.SelectedIndex == 0) {
log("Динамики выключены"); waveOutBool = false; }
                else waveOutBool = true;
                if (waveOutBool)
                {
                    bufferCount = 1;

                    output = new WaveOut[bufferCount];

                    for (int i = 0; i < bufferCount; i++)
                    {
                        output[i] = new WaveOut();
                        output[i].DeviceNumber =
waveOut_cb.SelectedIndex - 1;
                        bufferStream[i].ClearBuffer();

                        output[i].Init(bufferStream[i]);
                        output[i].Play();
                        audioListen_bt.Text = "Заглушить";
                    }
                    listenAudio = true;
                }
            }
            else
            {
                audioListen_bt.Text = "Слушать";
                for (int i = 0; i < bufferCount; i++)
                {
                    output[i].Stop();
                    output[i].Dispose();
                    output[i] = null;
                }
                listenAudio = false;
            }
        }
    }
}

```

Файл Client.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using NAudio.Wave;

namespace tcpServer
{
    public class Client
    {
        #region Аудио настройки

        #endregion

        #region Создание пользователя
        public int id; //Автоматически генерируемый id
        public string sn; //SN пользователя
        public string name; //Имя пользователя
        public bool identification; //Идентификация пользователя
        public bool audio; //Говорит ли в данный момент
        пользователь
        public BufferedWaveProvider bufferedWaveProvider;
        //Буфер пользователя
        public Wave16ToFloatProvider wave16ToFloatProvider;
        //Преобразование буфера пользователя
        public Socket socket; //Сокет пользователя
        public Thread thread; //Поток пользователя

        #region Создание нового клиента
        public Client(Socket socketNew)
        {
            //формируем id пользователя
            id = Users.Count;
            //Присвоить новому пользователю Sокет
            socket = socketNew;
            //Идентификация по умолчанию не пройдена
            identification = false;
            //По умолчанию пользователь не говорит
            audio = false;
            //Инициализируем буфер аудиопотока
            bufferedWaveProvider = new BufferedWaveProvider(new
            WaveFormat(8000, 16, 1)) { ReadFully = false }; //При
            подключении клиента
        }
    }
}
```

```

        wave16ToFloatProvider = new
Wave16ToFloatProvider(bufferedWaveProvider); //При подключении
клиента
        Form1.mix.open(bufferedWaveProvider,
wave16ToFloatProvider);
        //Создать для нового пользователя отдельный поток
для прослушивания его сообщений
        thread = new System.Threading.Thread(delegate () {
Listening(id); });
        //Установить поток в фоновый режим
        thread.IsBackground = true;
        //Запустить поток
        thread.Start();
    }
    #endregion
    #region Прослушка порта пользователя (пришедшие от
пользователя сообщения)
    private void Listening(int id = 0)
    {
        while (socket.Connected == true)
        {
            try
            {
                //промежуточный буфер
                byte[] buffer = new byte[565655];

                //получено данных
                int bytesRec = socket.Receive(buffer);
                //Обрезать массив от пустых данных
                Array.Resize(ref buffer, bytesRec);

                //Преобразовать в текст
                string data =
Encoding.Unicode.GetString(buffer, 0, bytesRec);
                //Парсить сообщения от пользователя

                messageParse(id, data, buffer);

            }
            catch { EndClient(this); }
        }
    }
    #endregion
    #region Парсер команд пришедших от пользователя
    private void messageParse(int id, string data, byte[]
buffer)
    {
        #region Идентификация пользователя
        if (data.Contains("#connect"))
        {

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        try
        {
            name = data.Split('&')[1]; //Присвоить имя
пользователя
            sn = data.Split('&')[2]; //Присвоить SN
пользователя
            identification = true; //Идентификация
пройдена
            connectResult(id, true);
        } catch { connectResult(id, false); }
        return;
    }
    else if(!identification) { connectResult(id, false);
}
#endregion

#region Новое текстовое сообщение
if (data.Contains("#msg"))
{
    string message = data.Remove(0, 5);
//Сообщение

    Form1.log(name + ": " + message);
    Histori.AddMessage(name + ": " + message);
//Сохранить сообщение
    message = name + ": " + message;
    Send.AllTCP("#msg&" + message); //Отправить
текстовое сообщение всем подключенным пользователям
    return;
}
#endregion
#region Новое аудио сообщение
if (data.Contains("#audio"))
{
    string command = data.Remove(0, 7);
//Команда

    if(command == "start") { audio = true;
Form1.log(name + ": Передача голоса..."); }
    if(command == "stop") { audio = false;
Form1.log(name + ": Остановил передачу голоса..."); }

    return;
}

    if (audio) {

Form1.mix.AddInputStreamToMix(bufferedWaveProvider, buffer);

```

```

        }
        #endregion

    }
    #endregion
    #endregion

    #region Пользователи
    public static List<Client> Users = new List<Client>();
    //Список подключенных пользователей
    #region Подключение нового пользователя
    /// <summary>
    /// Создание нового пользователя
    /// </summary>
    public static void NewClient(Socket socket)
    {
        try
        {
            Client newClient = new Client(socket);
            Users.Add(newClient);
        }
        catch (Exception exp) { Form1.log("Ошибка
подключения нового пользователя. Подробнее: " + exp.Message); }
    }

    #endregion
    #region Идентификация пользователя (вернуть ответ
пользователю)
    public static void connectResult(int id, bool
identification)
    {
        try
        {
            int index = Users.FindIndex(s =>
string.Equals(s.id.ToString(), id.ToString(),
StringComparison.CurrentCultureIgnoreCase));
            #region Идентификация пройдена
            if (identification)
            {
                Send.TCP(Users[index].socket,
"#true&Соединение установлено. Ваш IP:Port " +
Users[index].socket.RemoteEndPoint); //Отправить ответ об
успешном соединении
                Form1.log("Подключен новый пользователь " +
Users[index].id + " | " + Users[index].name + " | " +
Users[index].socket.RemoteEndPoint);
                Send.TCP(Users[index].socket, "#msg&" +
Histori.getMessage()); //Отобразить историю сообщений
            }
        }
    }
    #endregion

```

```

    }
    #endregion
    #region Идентификация не пройдена
    else
    {
        Send.TCP(Users[index].socket,
"#false&Идентификация не пройдена."); //Отправить ответ об
успешном соединении
        Form1.log("Идентификация пользователя не
пройдена " + Users[index].id + " | " + Users[index].name + " | "
+ Users[index].socket.RemoteEndPoint);
        EndClient(Users[index]);
    }
    #endregion
}
catch (Exception exp) { Form1.log("Ошибка
подключения нового пользователя. Подробнее: " + exp.Message); }
}
#endregion
#region Отключение пользователя
/// <summary>
/// Отключение пользователя
/// </summary>
public static void EndClient(Client user)
{
    try
    {
        if (Users.Contains(user))
        {
            string id = user.id.ToString();
            string username = user.name.ToString();
            string socket =
user.socket.RemoteEndPoint.ToString();

            //Инициализируем буфер аудиопотока
            Form1.mix.close(user.bufferedWaveProvider,
user.wave16ToFloatProvider);
            user.socket.Close();
            Users.Remove(user);

            Form1.log("Пользователь " + " #id" + id + "
" + username + " (" + socket + ") " + " отключен от сервера");
        }
    }
    catch (Exception exp)
    {
        Form1.log("Ошибка отключения пользователя.
Подробнее: " + exp.Message);
    }
}
}

```

```
#endregion
#endregion
```

```
}
}
```

Файл Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Text.RegularExpressions;

namespace tcpServer
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            CheckForIllegalCrossThreadCalls = false; //Доступ к
компонентам не из потока родителя
            parentform = this; //Получить копию контролов формы
        }
        #region Действия при открытие формы
        private void Form1_Load(object sender, EventArgs e)
        {
            // Получение имени компьютера.
            String host = System.Net.Dns.GetHostName();
            // Получение ip-адреса.
            System.Net.IPAddress ip =
System.Net.Dns.GetHostByName(host).AddressList[1];
            //System.Net.IPAddress[] ip =
System.Net.Dns.GetHostAddresses(host);
            // Показ адреса в label'e.
            tcpServer_tx.Text = ip.ToString();

        }
        #endregion
        #region Действия при закрытие формы
        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
```

```

        {
            Server.serverStop();
        }
    #endregion

    #region Переменные
    public static Form1 parentform = null; //Копия контролов
    формы
    public static VoiceMix mix = new VoiceMix();
    #endregion
    #region Старт/Стоп сервера
    bool start = false;
    private void tcpStart_bt_Click(object sender, EventArgs
e)
    {
        if (!start)
        {

            Regex portRegex = new Regex(@"\d{1,5}");

            if (portRegex.IsMatch(tcpPort_tx.Text))
            {
                if (IPAddress.TryParse(tcpServer_tx.Text, out
IPAddress testIp))
                {
                    if (Convert.ToInt32(tcpPort_tx.Text) <=
65535)
                    {
                        if
(Server.serverStart(tcpServer_tx.Text,
Convert.ToUInt16(tcpPort_tx.Text)))
                        {
                            tcpStart_bt.Text =
"Остановить";

                            timer1.Start();
                            message_bt.Enabled = true;
                            start = true;
                        }
                    }
                    else MessageBox.Show("Некорректный
порт");
                }
                else MessageBox.Show("Некорректный ip-
адрес");
            }
            else MessageBox.Show("Некорректный порт");
        }
        else
        {
            if (Server.serverStop())

```

```

        {
            tcpStart_bt.Text = "Запустить";
timer1.Stop();
            message_bt.Enabled = false;
            start = false;
        }
    }
}
#endregion

#region Добавить сообщение в лог
public static void log(string text)
{
    try
    {
        parentform.Invoke(new Action(() => {
parentform.log_tx.AppendText(DateTime.Now + ": " + text +
Environment.NewLine); }));
    }
    catch { }
}

#endregion

#region Отправить всем подключенным пользователям
сообщение
private void message_bt_Click(object sender, EventArgs
e)
{
    if (Send.AllTCP("#msg&" + "СЕРВЕР: " +
message_tx.Text)) { log("СЕРВЕР: " + message_tx.Text);
message_tx.Text = ""; };
}
#endregion

private void timer1_Tick(object sender, EventArgs e)
{
    byte[] bytes = mix.readMix();
    if (bytes.Length != 0)
    {
        Send.AllTCP(bytes); //Отправляем данные взятые
из буфера микшера
    }
}
}
}

```

Файл History.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace tcpServer
{
    class Histori
    {
        static int historyMessageMax = 20;
        private static List<string> message = new
List<string>();
        public static void addMessage(string text)
        {
            if(message.Count > historyMessageMax) {
message.RemoveAt(0); }
            message.Add(text);
        }

        public static string getMessage()
        {
            string allMessage = "";
            foreach(string str in message)
            {
                allMessage += str + Environment.NewLine;
            }
            return allMessage;
        }
    }
}
```

Файл Send.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Sockets;
using System.Threading;
using System.Net;

namespace tcpServer
{
    class Send
    {
        #region Отправить TCP сообщение
        /// <summary>
        /// TCP отправка текстового сообщения

```

```

    /// </summary>
    public static bool TCP(Socket socket, string text)
    {
        try
        {
            if (socket.Connected)
            {
                byte[] bytes =
Encoding.Unicode.GetBytes(text);
                socket.Send(bytes);

                return true;
            } else { return false; }
        }
        catch { return false; }
    }
    /// <summary>
    /// TCP отправка байтового сообщения
    /// </summary>
    public static bool TCP(Socket socket, byte[] bytes)
    {
        try
        {
            if (socket.Connected)
            {
                socket.Send(bytes);

                return true;
            }
            else { return false; }
        }
        catch { return false; }
    }
    /// <summary>
    /// TCP отправка текстового сообщения группе сокетов,
передайте необязательный параметр
    /// Socket ignore на который сообщения отправляться не
будут
    /// </summary>
    public static bool AllTCP(string text, Socket ignore =
null)
    {
        try
        {
            List<Socket> sockets = new List<Socket>();
            foreach (Client client in Client.Users)
            {
                if(ignore != null && ignore !=
client.socket) { sockets.Add(client.socket); }
                else if(ignore == null) {
sockets.Add(client.socket); }

```

```

    }

    byte[] bytes = Encoding.Unicode.GetBytes(text);
    foreach (Socket socket in sockets)
    {
        int bytesSent = socket.Send(bytes);
    }
    return true;
}
catch { return false; }
}
/// <summary>
/// TCP отправка байтового сообщения группе сокетов,
передайте необязательный параметр
будут
/// Socket ignore на который сообщения отправляться не
будут
/// </summary>
public static bool AllTCP(byte[] bytes, Socket ignore =
null)
{
    try
    {
        List<Socket> sockets = new List<Socket>();
        foreach (Client client in Client.Users)
        {
            if (ignore != null && ignore !=
client.socket) { sockets.Add(client.socket); }
            else if (ignore == null) {
sockets.Add(client.socket); }
        }

        foreach (Socket socket in sockets)
        {
            int bytesSent = socket.Send(bytes);
        }
        return true;
    }
    catch { return false; }
}
}
#endregion
}
}

```

Файл Server.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net.Sockets;

```

```

using System.Threading;
using System.Net;

namespace tcpServer
{
    class Server
    {
        #region Работа сервера
        //Запущен ли сервер
        private static bool connected;
        private static int threadStatus;
        //IP сервера
        private static string server_ip = "";
        //Порт сервера
        private static int server_port = 0;
        //сокет для приема (протокол TCP)
        static Socket serverSocket;
        //Прослушивающий поток для сервера
        private static Thread server_thread;

        #region Запуск сервера
        /// <summary>
        /// Запуск TCP-сервера, в host подаётся ip-адрес
        сервера, в port - подается порт сервера.
        /// В случае успеха вернется true, в случае ошибки или
        если сервер уже запущен вернется false.
        /// </summary>
        public static bool serverStart(string host, int port =
0)
        {
            try
            {
                if (!connected)
                {
                    connected = true;
                    threadStatus = 0;
                    Form1.log("Запуск TCP сервера...");
                    server_ip = host; //Присвоить IP серверу
                    server_port = port; //Присвоить порт серверу
                    //serverSocket = new
Socket(IPAddress.Parse(server_ip).AddressFamily,
SocketType.Stream, ProtocolType.Tcp);
                    serverSocket = new
Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);

                    server_thread = new Thread(Listening);
                    //Присвоить функцию прослушивания сообщений серверу
                    server_thread.IsBackground = true;
                    server_thread.Start(); //Запустить поток
прослушки

```

```

        while (true)
        {
            if (threadStatus == 1)
                return true;
            if (threadStatus == 2)
                return false;
            Thread.Sleep(10);
        }

    }
    return false;
}
catch { return false; }
}
#endregion
#region Остановка сервера
/// <summary>
/// Остановка TCP-сервера, в случае успеха вернется
true, в случае ошибки или если сервер не запущен вернется false.
/// </summary>
public static bool serverStop()
{
    try
    {
        if (connected)
        {
            if (Client.Users.Count >= 1)
            {
                foreach (Client user in
Client.Users.ToArray())
                {
                    Client.EndClient(user);
                }
            }
            connected = false;
            serverSocket.Close();
            serverSocket.Dispose();
            return true;
        }
        return true;
    }
    catch { return false; }
}
#endregion
#region Прослушивание порта (входящих подключений)
/// <summary>
/// Прослушивает порт на все начальные подключения
/// </summary>
private static void Listening()
{

```

```

        IPEndPoint ipEndPoint = new
IPEndPoint(IPAddress.Parse(server_ip), server_port);
        try
        {
            serverSocket.Bind(ipEndPoint);
        }
        catch (Exception)
        {
            connected = false;
            threadStatus = 2;
            Form1.log("Сервер не запущен. Возможно введён
неправильный ip или порт.");
            return;
        }
        threadStatus = 1;
        serverSocket.Listen(1000);
        Form1.log("TCP сервер запущен с IP " + ipEndPoint);

        while (connected == true)
        {
            try
            {
                Socket user = serverSocket.Accept(); //Новое
подключение
                Client.NewClient(user); //Создать нового
пользователя
            }
            catch (Exception exp) { }
        }
        Form1.log("TCP сервер остановлен");
    }
    #endregion
    #endregion
}
}

```

Файл VoiceMix.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NAudio.Wave;
using System.Threading;

namespace tcpServer
{
    public class VoiceMix
    {
        #region Переменные

```

```

        public static MixingWaveProvider32
        _mixingWaveProvider32;
        public static WaveFloatTo16Provider _output;

        #endregion
        public VoiceMix()
        {
            _mixingWaveProvider32 = new
MixingWaveProvider32(new[] { new
SimpleWaveProvider(WaveFormat.CreateIeeeFloatWaveFormat(8000,
1)), }); //При старте сервера
            _output = new
WaveFloatTo16Provider(_mixingWaveProvider32); //При старте
сервера
        }

        //Добавляем новый клиент
        public void open(BufferedWaveProvider
bufferedWaveProvider, Wave16ToFloatProvider
wave16ToFloatProvider)
        {

            _mixingWaveProvider32.AddInputStream(wave16ToFloatProvider);
//При подключении клиента
        }
        //Удаляем указанный клиент
        public void close(BufferedWaveProvider
bufferedWaveProvider, Wave16ToFloatProvider
wave16ToFloatProvider)
        {

            _mixingWaveProvider32.RemoveInputStream(wave16ToFloatProvider);
//При отключении клиента
        }

        //Миксуем аудиоданные
        public void AddInputStreamToMix(BufferedWaveProvider
bufferedWaveProvider, byte[] buffer)
        {

            bufferedWaveProvider.AddSamples(buffer, 0,
buffer.Length);
        }

        //Читаем данные
        public byte[] readMix()
        {
            byte[] bytes = new byte[8000];
            Thread.Sleep(0);
            int x = _output.Read(bytes, 0, bytes.Length);
            Array.Resize(ref bytes, x);
        }

```

```

        return bytes;
    }

    class SimpleWaveProvider : IWaveProvider
    {
        public SimpleWaveProvider(WaveFormat waveFormat)
        {
            WaveFormat = waveFormat;
        }

        public int Read(byte[] buffer, int offset, int
count)
        {
            return 0;
        }

        public WaveFormat WaveFormat { get; }
    }
}

```

					<i>ПГУ 1.090301.27.001 ПЗ</i>	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(Обязательное)

Программное обеспечение для передачи информации сетевым способом

Презентация

					ПГУ 1.090301.27.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпись	Дата		

Программное обеспечение для передачи информации сетевым способом

Автор работы:

студент гр. 18ВВ2 Немков Артем Михайлович

Научный руководитель:

к.т.н., доцент кафедры ВТ Дубравин Алексей Викторович

Цель работы

Цель работы – разработать клиент-серверное приложение для обмена звуковыми и текстовыми данными через вычислительную сеть.

Приложение должно выполнять следующие функции:

- Запись звука на клиенте и передача на сервер
- Обработка звука на сервере и отправка клиентам
- Воспроизведение звука, полученного с сервера

Используемые технологии



Последовательность работы приложения

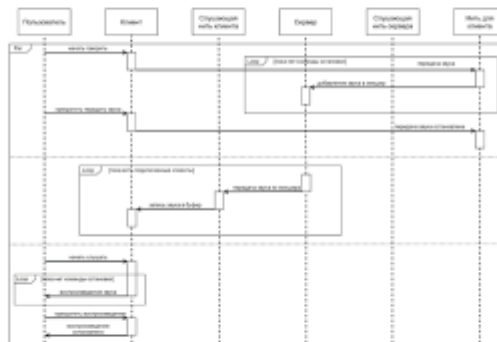


Диаграмма деятельности клиента

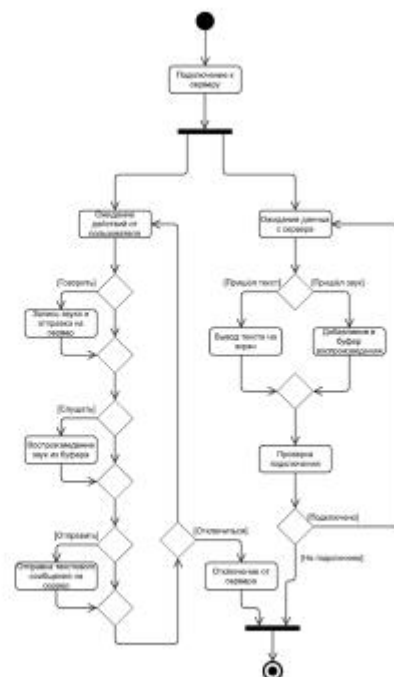
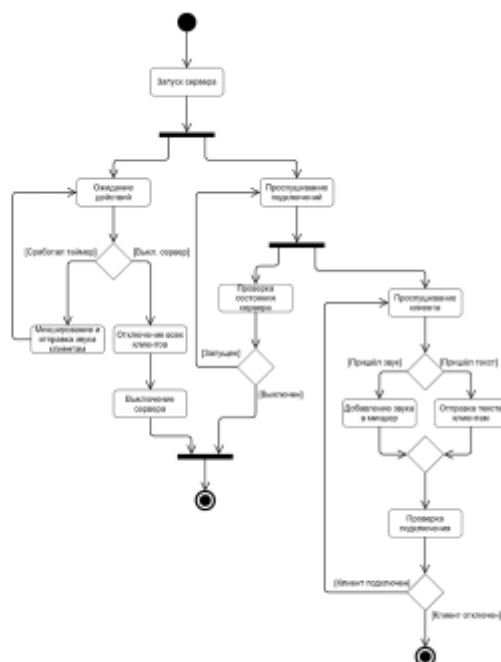


Диаграмма
деятельности
сервера



Графический интерфейс приложения

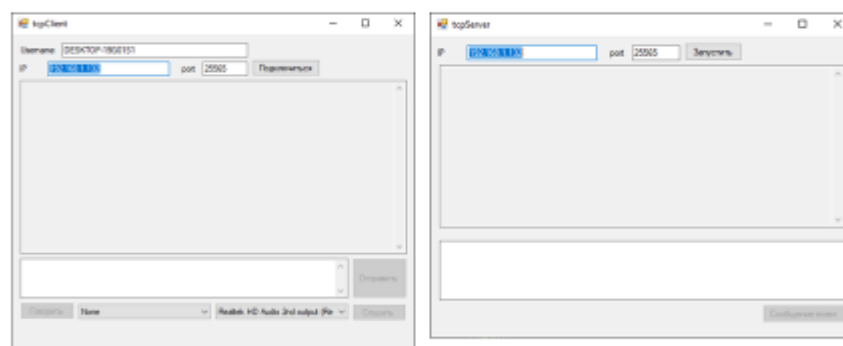


Диаграмма вариантов использования сервера

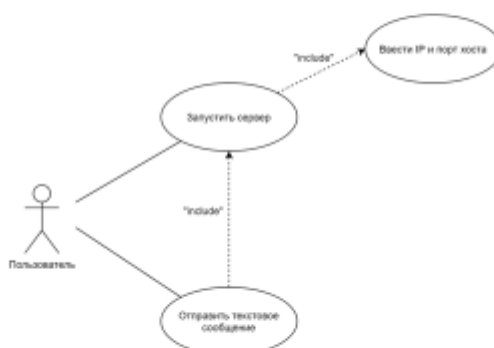
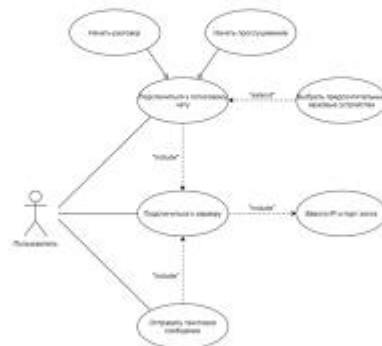
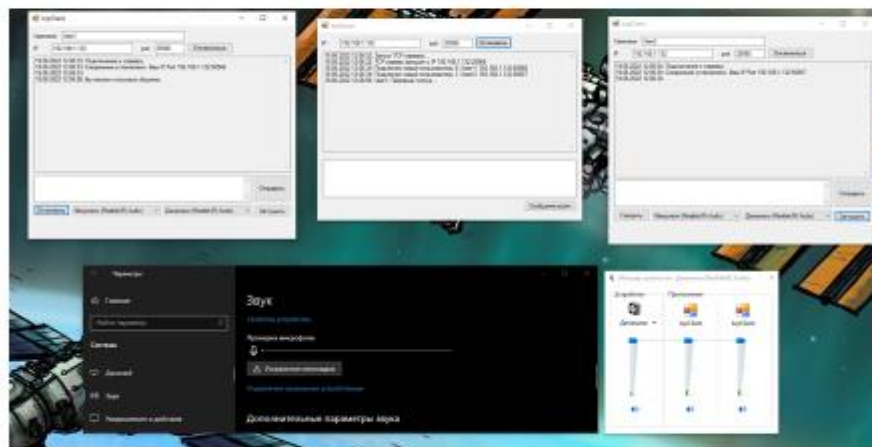


Диаграмма вариантов использования клиента



Работа программы



Спасибо за внимание!

					ПГУ 1.090301.27.001 ПЗ	Лист
						102
Изм.	Лист	№ докум.	Подпись	Дата		