Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Системное программное обеспечение вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

PEER-TO-PEER СЕТЕВОЙ ЧАТ

БГУИР КП 1-40 02 01 105 ПЗ

Студент: группы 950501,
Деркач А. В.

Руководитель: старший
преподаватель каф. ЭВМ
Поденок Л. П.

Минск 2021

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____Б.В. Никульшин
(подпись)
«___» _____2021 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Деркач Анжелике Валерьевне*

1. Тема проекта *Peer-to-peer сетевой чат*

2. Срок сдачи студентом законченного проекта *25 мая 2021 г.*

3. Исходные данные к проекту *Язык программирования – C++*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)
Введение. 1. Обзор литературы. 2. Системное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. Заключение. Список использованных источников

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков) *1. Схема структурная. 2. Диаграмма классов*

6. Консультант по проекту *Поденок Л. П.*

7. Дата выдачи задания *20 февраля 2021 г.*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):
разделы 1,2 к 15 марта 2021 г. – 20 %;
разделы 3,4 к 15 апреля 2021 г. – 30 %;
разделы 5,6,7 к 15 мая 2021 г. – 30 %;
оформление пояснительной записки и графического материала к 25 мая 2021 г. 20 %
Защита курсового проекта с 5 июня 2021 г. по 10 июня 2021 г.

РУКОВОДИТЕЛЬ _____ *Л. П. Поденок*
(подпись)

Задание принял к исполнению _____ *А.В. Деркач*

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 4
1 ОБЗОР ЛИТЕРАТУРЫ 5
1.1 Обзор аналогов..... 5
1.2 Постановка задачи 7
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 9
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ..... 11
3.1 Класс ChatDialog 11
3.2 Класс Client..... 11
3.3 Класс Connection..... 13
3.4 Класс PeerManager..... 14
3.5 Класс Server..... 15
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ..... 16
4.1 Алгоритм ввода сообщения..... 16
4.2 Алгоритм обновления адресов..... 16
4.3 Алгоритм удаления соединения..... 17
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ..... 18
5.1 Отсутствие подключения..... 18
5.2 Пользовательский ввод..... 19
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 21
ЗАКЛЮЧЕНИЕ..... 25
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ..... 26
ПРИЛОЖЕНИЕ А Структурная схема..... 27
ПРИЛОЖЕНИЕ Б Диаграмма классов 28
ПРИЛОЖЕНИЕ В Листинг кода..... 29
ПРИЛОЖЕНИЕ Г Ведомость документов..... 47

ВВЕДЕНИЕ

Темой данного курсового проекта является «Peer-to-peer сетевой чат». Данный курсовой проект представляет собой приложение, использующее технологию peer-to-peer (равный к равному), представляющей собой оверлейную компьютерную сеть, основанную на равноправии участников. Данное приложение — это средство обмена сообщениями по компьютерной сети в режиме реального времени, а также программное обеспечение, позволяющее организовывать такое общение.

В рамках данной курсовой работы необходимо ознакомиться с технологией peer-to-peer и принципом работы сетевого чата. В процессе разработки предполагается углубить знания по языку C++ и в области объектно-ориентированного программирования, а также осуществить разработку удобного пользовательского интерфейса, используя Фреймворк Qt. В конце протестировать полученный продукт и провести эксперимент на нескольких устройствах.

Данная тема актуальна, так как на ней базируется много современных ресурсов, и она является довольно популярной среди пользователей.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Анализ существующих аналогов

Тема курсового проекта была выбрана в первую очередь для углубления знаний по языку C++ и в области объектно-ориентированного программирования, а также получения знаний в проектировании пользовательских приложений, поэтому моей целью не является разработать конкурентоспособный продукт. Тем не менее, чтобы создать корректно работающее приложение, нужно иметь представление о существующих аналогах, об их недостатках, преимуществах и реализованных внутри функциях.

1.1.1 StrongDC++

StrongDC++ — клиент для обмена файлами в пиринговых P2P сетях Direct Connect (основанны на принципе "расшаривания" ресурсов), базирующийся на исходном коде DC++. Поддерживает обмен данными по протоколам NMDC и ADC. Интерфейс StrongDC++ изображен на рисунке 1.1:

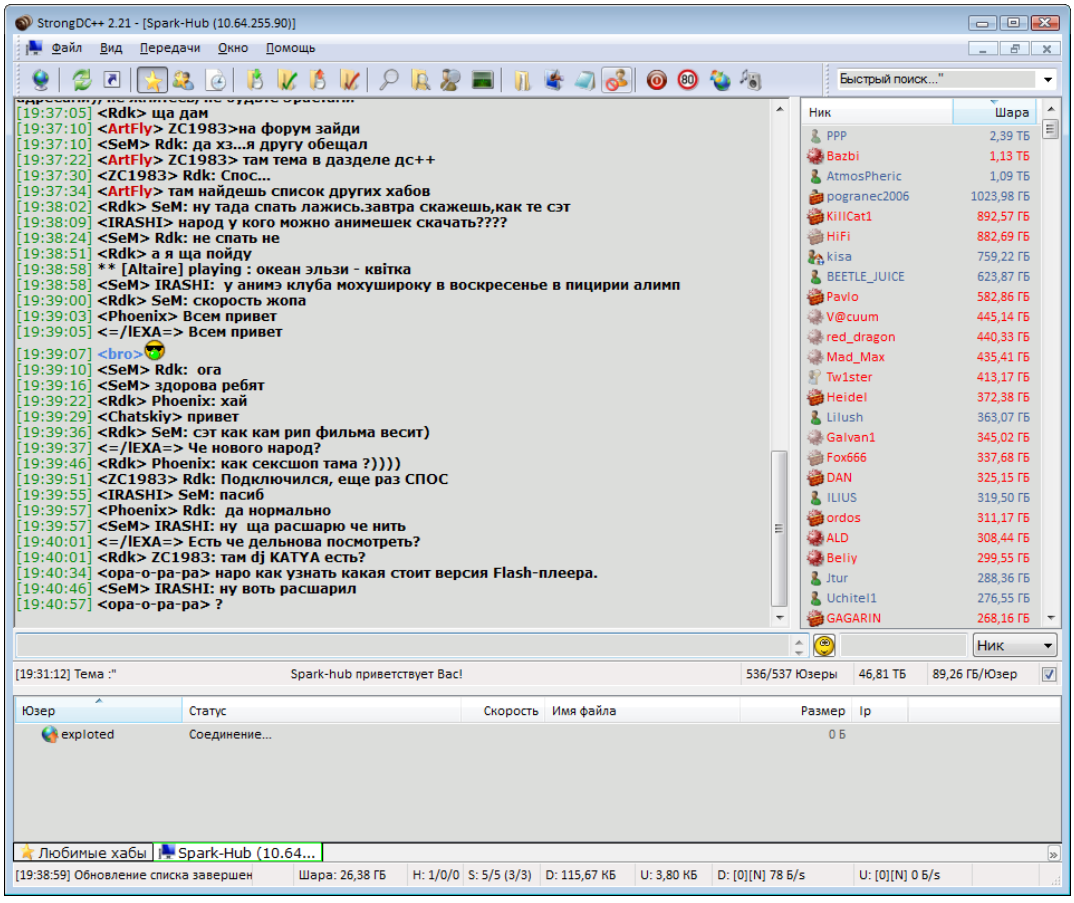

Рисунок 1.1 – Скриншот программы StrongDC++

StrongDC++ разработано компанией “BigMuscle”. Приложение написано на языке C++ и работает на Windows XP / Vista / 7. Последняя версия была выпущена 27 декабря 2010 года и уже не поддерживается.

1.1.2 Telegram

Telegram — кроссплатформенный мессенджер с функциями VoIP, позволяющий обмениваться текстовыми, голосовыми и видеосообщениями, стикерами и фотографиями, файлами многих форматов. Также можно совершать видео и аудио-звонки, организовывать конференции и многопользовательские группы и каналы.

Для связи Telegram использует peer-to-peer соединение, что позволяет обмениваться информацией намного быстрее. Главное преимущество peer-to-peer – экономия трафика и улучшенное качество связи. Но есть и один минус – собеседник может узнать ваш IP адрес. На рисунке 1.2 представлен интерфейс Telegram:

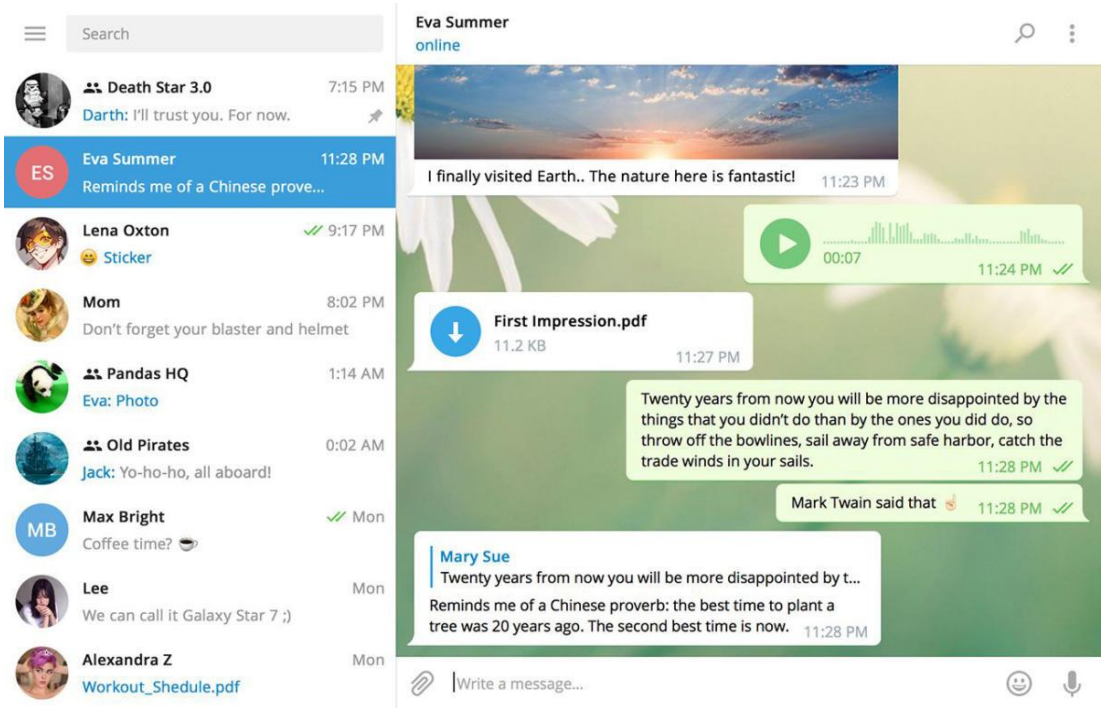

Рисунок 1.2 – Скриншот программы Telegram

Telegram разработан компанией “Telegram Messenger” и выпущен 14 августа 2013. Приложение написано на языках C++ и Java, пользовательский интерфейс создан с помощью Фреймворка Qt. Telegram поддерживается на всех популярных ОС и активно обновляется.

1.1.3 Briar

Briar — это одна из технологий децентрализованных сетей (mesh) с открытым исходным кодом, предназначенная для обеспечения надежной сети и надежной одноранговой связи без централизованных серверов и с минимальной зависимостью от внешней инфраструктуры. Соединения выполняются через Bluetooth, Wi-Fi или через Интернет через Tor, и все сообщения зашифрованы оконечным шифрованием. Соответствующий контент хранится в зашифрованном виде на устройствах участников.

Briar может отправлять сообщения в местах, где нет сотовой связи или Интернета через Bluetooth. Briar не имеет центрального сервера. Все сообщения синхронизируются напрямую между мобильными устройствами пользователей с помощью технологии peer-to-peer. Интерфейс Briar изображен на рисунке 1.3:

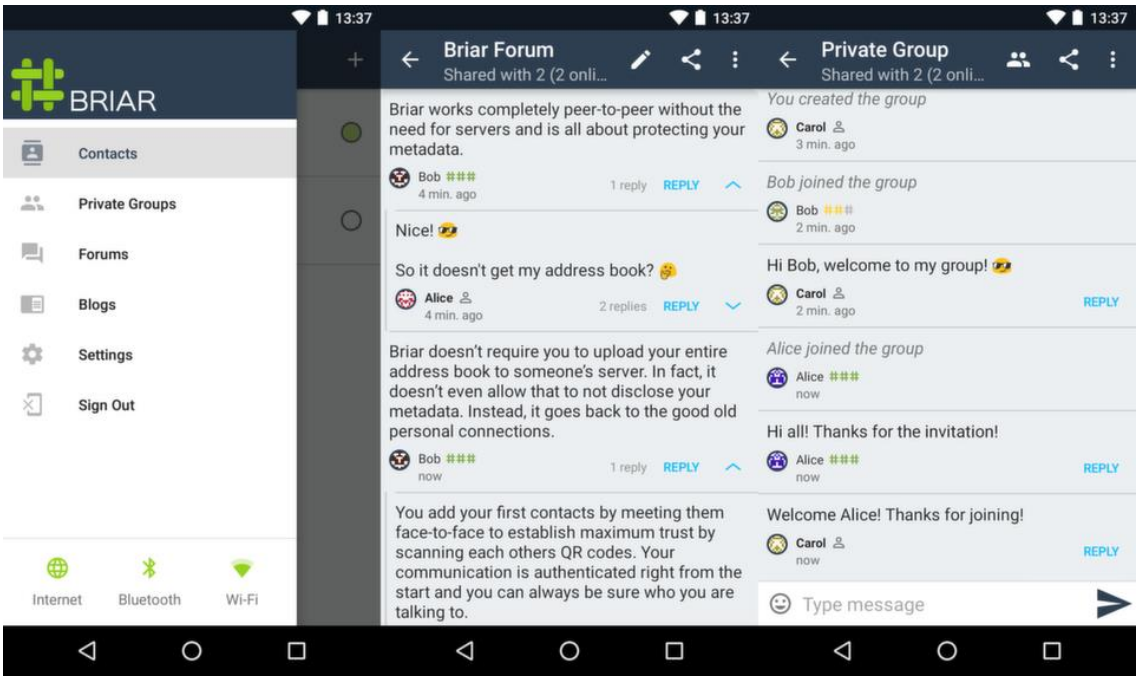

Рисунок 1.3 – Скриншот программы Briar

Briar впервые был выпущен 21 мая 2017 года. Приложение написано на языке Java и используется на ОС Android. На сегодняшний день Briar поддерживается и активно используется.

1.2 Постановка задачи

После рассмотрения аналогов можно сказать, что все они обладают большим количеством функций, которые невозможно реализовать в курсовом проекте за данный период времени. Поэтому были выбраны несколько ключевых возможностей, которые будут выполнены в рамках одного семестра:

- Программа должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню.

- В программе должна быть предусмотрена возможность хранения информации.

- Программа должна поддерживать обмен сообщениями в режиме реального времени.

- В программе должна быть реализована технология peer-to-peer.

- Должна быть реализована возможность авторизации пользователей.

В качестве языка программирования выбран C++, по причине быстрой производительности, требуемой для обработки большого количества объектов, поддержки объектно-ориентированного подхода к программированию и наличия опыта в использования данного языка. В качестве реализация графического интерфейса используется фреймворк Qt, основанный на языке C++. Qt обладает большим количеством базовых классов, которые позволяют создавать собственные классы для реализации графического интерфейса, удобной системой общения между виджетами приложения с помощью системы сигналов и слотов и хорошей документацией, позволяющей в быстрые сроки разбираться в устройстве Qt.

Данный список средств позволяет реализовать все задачи, выбранные для курсового проекта.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

После определения требований к функционалу разрабатываемого приложения его следует разбить на функциональные блоки. Такой подход упростит понимание проекта, позволит устранить проблемы в архитектуре, обеспечит гибкость и масштабируемость программного продукта в будущем путем добавления новых блоков.

2.1 Модуль авторизации

Модуль авторизации пользователя необходим для входа в приложение, это необходимо для выполнения различных действий, таких как, например, написания сообщения другим пользователям.

Модуль предоставляет доступ к функциям приложения. Его задача заключается в авторизации пользователей в приложении, определении имени и всех необходимых данных пользователя, зашедшего в приложение.

2.2 Модуль работы с приложением

Модуль работы с приложением необходим для взаимодействия пользователя с приложением и является неотъемлемой его частью. Тут осуществляется весь необходимый функционал и возможности:

- Просмотр активных клиентов
- Просмотр полученных сообщений
- Отправка сообщений
- Просмотр параметров подключения

2.3 Модуль пользовательского интерфейса

Модуль пользовательского интерфейса предназначен для взаимодействия пользователя с приложением, основываясь на представлении и визуализации данных.

Для разрабатываемого проекта создается простейший пользовательский интерфейс с помощью фреймворка Qt. Данный пользовательский интерфейс представляет собой набор пунктов меню, с которыми может взаимодействовать пользователь.

2.4 Модуль оперативного хранения данных

Модуль оперативного хранения данных необходим для организации временного хранения информационных ресурсов по мере работы с приложением. Информация о пользователе, который в данный момент работает с приложением, полученные и отправленные сообщения должны

фиксироваться для непосредственной работы.

2.5 Модуль чтения и записи данных

Модуль чтения и записи данных необходим для взаимодействия с данными, которые хранит приложение, что очень важно при работе с проектом.

Данный модуль должен отвечать за считывание информации от пользователя при авторизации и оправки сообщений, её записи в используемую область памяти для дальнейшей работы, а также считывание и запись всей другой необходимой информации по мере работы с приложением.

2.6 Модуль преобразования данных

Модуль преобразования данных необходим для преобразования данных, считываемых от пользователя или из приложения для корректной работы с ними.

Очень часто одни и те же данные используются для разных целей, например, считанное время, может сравниваться с другими данными в формате QTime или же выводиться на экран в формате QString, поэтому для корректной работы и отображения этих данных необходимо их преобразовать к нужному типу.

2.7 Модуль сети

Модуль сети предназначен для подключения пользователей друг к другу и необходим для их взаимодействия в реальном времени.

Данный модуль должен отвечать за передачу сообщений по технологии peer-to-peer между пользователями и отслеживать активность клиентов в приложении.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого приложения.

Описание классов приложения

3.1 ChatDialog

Класс наследуется от библиотечных классов QDialog и ChatDialog. Данный класс является графическим представлением программы, он представляет собой окно, в котором происходит работа в приложении.

Поля:

– `Client client` – пользователь, работающий в приложении.

– `QString myNickName` – имя пользователя.

– `QTextTableFormat tableFormat` – объект библиотечного класса, который осуществляет выравнивание текста при выводе полученных и отправленных сообщений на экран.

Методы:

– `ChatDialog` – конструктор, принимает `QWidget*` – ссылку на объект библиотечного класса, работающего с пользовательским интерфейсом. Инициализирует графическое окно работы с приложением.

– `void appendMessage` – принимает `const QString&` – имя отправителя, `const QString&` – сообщение. Добавляет сообщение на экран.

– `void returnPressed` – считывает и отправляет введенное сообщение.

– `void newParticipant` – принимает `const QString&` – имя пользователя. Добавляет пользователя в список подключенных пользователей.

– `void participantLeft` – принимает `const QString&` – имя пользователя. Удаляет пользователя из списка подключенных пользователей.

– `void on_connectButton_clicked` – срабатывает при нажатии на кнопку подключения, устанавливает соединение.

3.2 Client

Класс наследуется от библиотечного класса QObject. Данный класс является классом-сущностью, который связывает все действия пользователя с сервером.

Поля:

– `PeerManager* peerManager` – ссылка на объект класса для управления действиями с сетью.

– `Server server` – входящее соединение.

– `QMultiHash<QHostAddress, Connection*> peers` – подключенные пользователи.

– `QString userName` – имя пользователя.

Методы:

– `Client` – конструктор по умолчанию.

– `void setParameters` – принимает `QString` – имя пользователя. Устанавливает необходимые параметры.

– `void sendMessage` – принимает `const QString&` – сообщение. Отправляет сообщение.

– `bool hasConnection` – принимает `const QHostAddress` – ip пользователя, `int` – порт пользователя. Проверяет есть ли данный пользователь в списке подключенных.

– `QString getUserName` – получить имя пользователя.

– `void newMessage` – принимает `const QString&` – имя пользователя, `const QString&` – сообщение. Сигнал, который отправляется для отправления сообщения.

– `void newParticipant` – принимает `const QString&` – имя пользователя. Сигнал, который отправляется при подключении нового пользователя.

– `void participantLeft` – принимает `const QString&` – имя пользователя. Сигнал, который отправляется при отключении пользователя.

– `void newConnection` – принимает `Connection*` – новое подключение. Устанавливает новое подключение.

– `void connectionError` – принимает `SocketError` – ошибка сокета. Удаляет подключение из общего списка при возникновении ошибки.

– `void disconnected` – удаляет подключение из общего списка.

– `void readyForUse` – устанавливает необходимые параметры при подключении к сети.

– `void removeConnection` – принимает `Connection*` – подключение. Удаляет подключение из общего списка.

3.3 Connection

Класс наследуется от библиотечного класса QTcpSocket. Данный класс отвечает за установку соединения между пользователями.

Поля:

– `QCborStreamReader reader` – читатель сети.

– `QCborStreamWriter writer` - писатель в сеть.

– `QString greetingMessage` - сообщение сети

– `QString username` - имя пользователя.

– `QTimer pingTimer` - таймер задержки передачи.

– `QElapsedTimer pongTime` - таймер задержки ответа.

– `QString buffer` - буфер считывания данных.

– `ConnectionState state` - состояние подключения.

– `DataType currentDataType` - тип считываемых данных.

– `int transferTimerId` - идентификатор таймера.

– `bool isGreetingMessageSent` - статус отправки сообщения.

Методы:

– `Connection` – конструктор, принимает `QObject*` – ссылку на объект класса-сущности. Инициализирует необходимые поля.

– `Connection` – конструктор, принимает `qintptr` — дескриптор сокета, `QObject*` – ссылку на объект класса-сущности. Инициализирует необходимые поля.

– `~Connection` – деструктор по умолчанию.

– `QString getUserName` – получить имя пользователя.

– `void setGreetingMessage` – принимает `const QString&` – сообщение. Инициализирует поле класса.

– `bool sendMessage` – принимает `const QString&` – сообщение. Отправляет сообщение на сервер, в случае успеха возвращает true, иначе false.

– `void timerEvent` – принимает `QTimerEvent*` – событие таймера. Перезапускает таймер.

– `void readyForUse` – сигнал, который отправляется при подключении.

– `void newMessage` – принимает `const QString&` – имя пользователя, `const QString&` – сообщение. Сигнал, который отправляется при отправке сообщения.

– `void processReadyRead` – считывает все необходимые данные.

– `void sendPing` – отправляет задержку передачи.

– `void sendGreetingMessage` – отправляет сообщение на сервер.

– `void processGreeting` – обрабатывает сообщение для отправки на сервер.

– `void processData` – обрабатывает данные.

3.4 PeerManager

Класс наследуется от библиотечного класса QObject. Данный класс является классом-сущностью, который осуществляет управление на сервере.

Поля:

– `Client *client` - ссылка на пользователя.

– `QList<QHostAddress> broadcastAddresses` - список широковещательных адресов.

– `QList<QHostAddress> ipAddresses` - список ip адресов.

– `QUdpSocket broadcastSocket` - сокет.

– `QTimer broadcastTimer` - таймер вещания.

– `QString username` - имя пользователя.

– `int serverPort` - порт.

Методы:

– `PeerManager` – конструктор, принимает `Client*` – ссылку на объект клиента. Инициализирует поля класса.

– `void setServerPort` – принимает `int` – порт. Устанавливает серверный порт.

– `QString getUserName` – получить имя пользователя.

– `void startBroadcasting` – начинает отсчет широковещательного таймера.

– `bool isLocalHostAddress` – принимает `const QHostAddress&` – адрес. Проверяет адрес на локальность.

– `void newConnection` – принимает `Connection*` – новое подключение. Сигнал, который отправляется при новом подключении.

– `void updateAddresses` – обновляет адреса.

– `void sendBroadcastDatagram` – отправляет дейтаграмму UDP.

– `void readBroadcastDatagram` – читает дейтаграмму UDP.

3.5 Server

Класс наследуется от библиотечного класса QTcpServer. Данный класс имитирует сервер для управления входящими соединениями.

Методы:

– `Server` – конструктор, принимает `QObject*` – ссылку на объект класса-сущности. Устанавливает прослушивание сети.

– `void incomingConnection` – принимает `qintptr` – дескриптор сокета. Создает новое соединение.

– `void newConnection` – принимает `Connection*` – ссылку объекта класса соединения. Сигнал, который отправляется для подключения нового пользователя.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Метод returnPressed() класса ChatDialog.

Шаг 1. Начало.

Шаг 2. Считываем введенное сообщение с экрана.

Шаг 3. Если сообщение пустое, то переходим к шагу 4, если нет, то переходим к шагу 6.

Шаг 4. Выводим сообщение об ошибке.

Шаг 5. Выходим из метода.

Шаг 6. Если сообщение начинается с /, то переходим к шагу 7, если нет, то переходим к шагу 13.

Шаг 7. Получаем цвет текста.

Шаг 8. Если сообщение начинается с /info, то переходим к шагу 9, если нет, то переходим к шагу 11.

Шаг 9. Устанавливаем цвет текста “тёмно-зелёный”.

Шаг 10. Выводим на экран текст, соответствующий команде и переходим к шагу 15.

Шаг 11. Устанавливаем цвет текста “красный”.

Шаг 12. Выводим на экран текст об отсутствии такой команды и переходим к шагу 15.

Шаг 13. Отправляем сообщение на сервер.

Шаг 14. Выводим сообщение на экран.

Шаг 15. Очищаем введенный текст.

Шаг 16. Конец.

4.2 Метод updateAddresses() класса PeerManager.

Шаг 1. Начало.

Шаг 2. Очищаем список широковещательных адресов.

Шаг 3. Очищаем список ip адресов.

Шаг 4. Получаем список всех сетевых интерфейсов хоста.

Шаг 5. Проходим по всему списку интерфейсов и выполняем шаги 6-11, после чего переходим к шагу 12.

Шаг 6. Получаем список всех сетевых IP-адресов сетевого интерфейса.

Шаг 7. Проходим по всему списку адресов и выполняем шаги 8-11, после чего переходим к шагу 5.

Шаг 8. Получаем широковещательный адрес.

Шаг 9. Если широковещательный адрес не пустой и IP-адрес не равен локальному хосту, то переходим к шагу 10, если нет, то переходим к шагу 7.

Шаг 10. Добавляем в список широковещательных адресов полученный широковещательный адрес.

Шаг 11. Добавляем в список IP-адресов полученный IP-адрес.

Шаг 12. Конец.

4.3 Метод removeConnection(Connection *connection) класса Client.

Шаг 1. Начало.

Шаг 2. Если список адресов содержит адрес соединения, которое требуется удалить, то переходим к шагу 3, если нет, то переходим к шагу 7.

Шаг 3. Удаляем соединения из списка адресов.

Шаг 4. Получаем имя пользователя, соединение которого потеряно.

Шаг 5. Если имя пользователя не пустое, то переходим к шагу 6, если нет, то переходим к шагу 7.

Шаг 6. Отправляем сигнал классу ChatDialog для того, чтобы удалить пользователя из списка на экране.

Шаг 7. Удаляем ссылку на полученное соединение.

Шаг 8. Конец.

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование — это наблюдение за работой приложения в разных искусственно созданных ситуациях. Данные, полученные в ходе тестирования, важны при планировании последующей стратегии развития приложения. Это своего рода диагностика, которая влияет на многие дальнейшие действия.

Для тестирования разработанного приложения были сымитированы ошибки, чтобы проверить, как приложение на них реагирует.

5.1 Тестирование работы приложения при отсутствии подключения.

Данное приложение использует работу по сети. На практике не всегда удается корректно установить соединение по разным причинам, из-за чего приложение будет работать некорректно или ломаться. Для решения этой проблемы в коде есть проверки на результаты работы при подключении пользователей. Данное решение дает возможность пользоваться приложением без возникновения ошибок.

Если подключения нет, то, при входе в приложение, пользователь подключится, но не будет видеть других пользователей (рисунок 5.1.1). При появлении сети соединение появится.

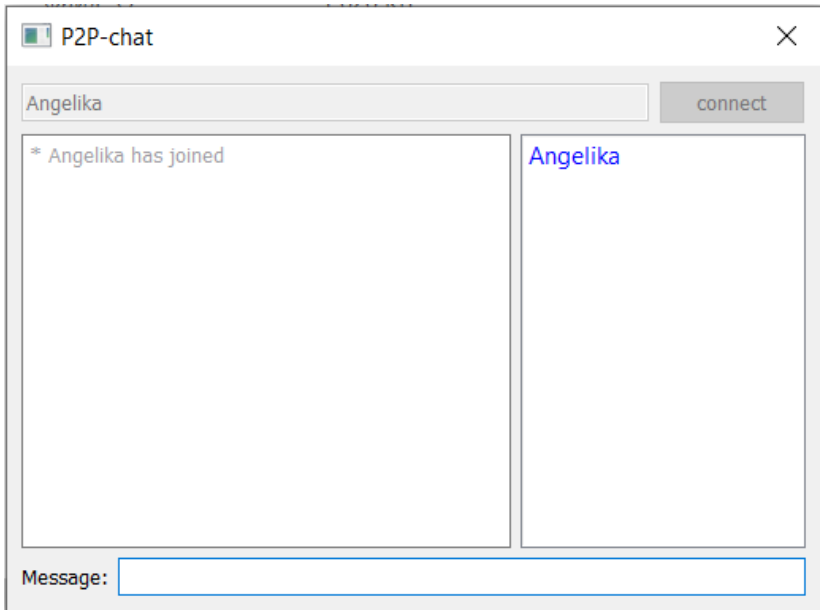

Рисунок 5.1.1 – Ошибка подключения.

Подключение к сети может пропасть и во время работы приложения, если данная ситуация происходит, то, пользователь, который потерял соединение, отключится от приложения и в поле сообщений высветится соответствующее письмо, которое увидят все пользователи (рисунок 5.1.2).

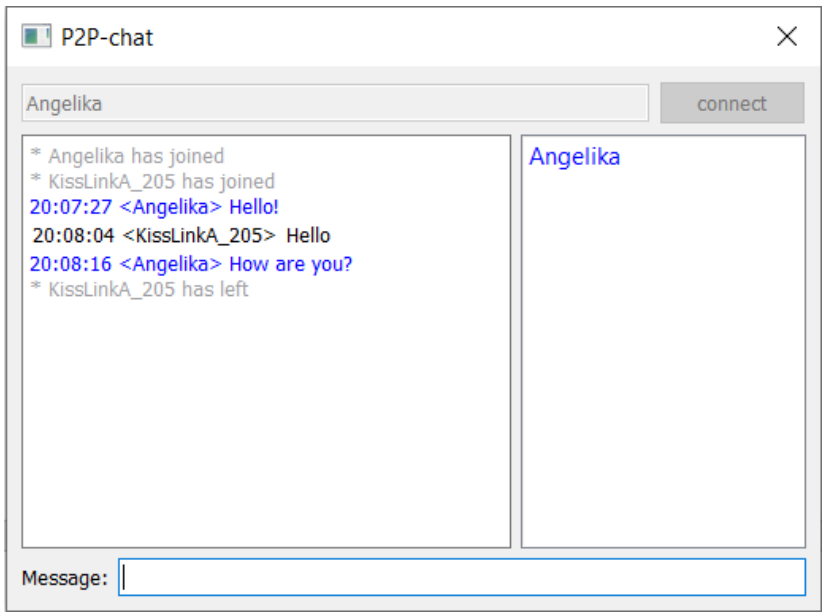

Рисунок 5.1.2 – Прерванное подключение.

5.2 Тестирование корректности пользовательского ввода.

Каждое приложение, в котором есть ввод данных должно проверять эти данные на валидацию. В данном приложении проверяется корректность ввода имени и сообщения. Если введенная информация пустая, то на экране появится сообщение об ошибке и пользователю будет необходимо повторить ввод. Ниже приведены скриншоты ошибок:

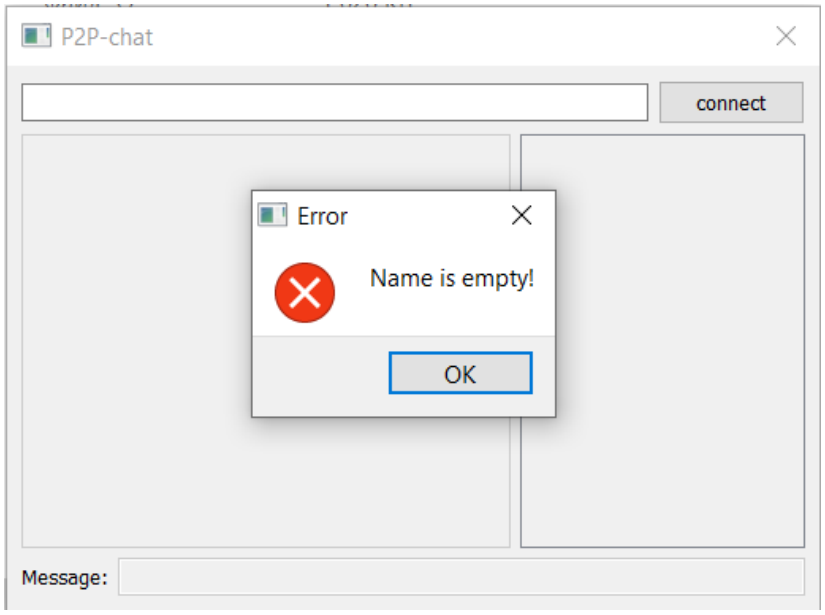

Рисунок 5.2.1 – Ошибка в вводе имени.

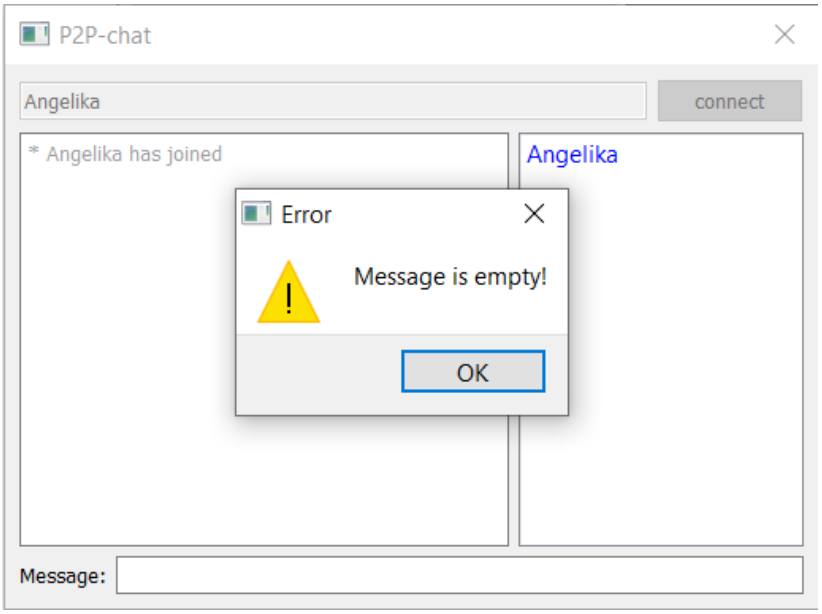

Рисунок 5.2.2 – Ошибка в вводе сообщения.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска программы необходимо открыть файл «P2P-chat.exe». После этого откроется окно программы (рисунок 6.1).

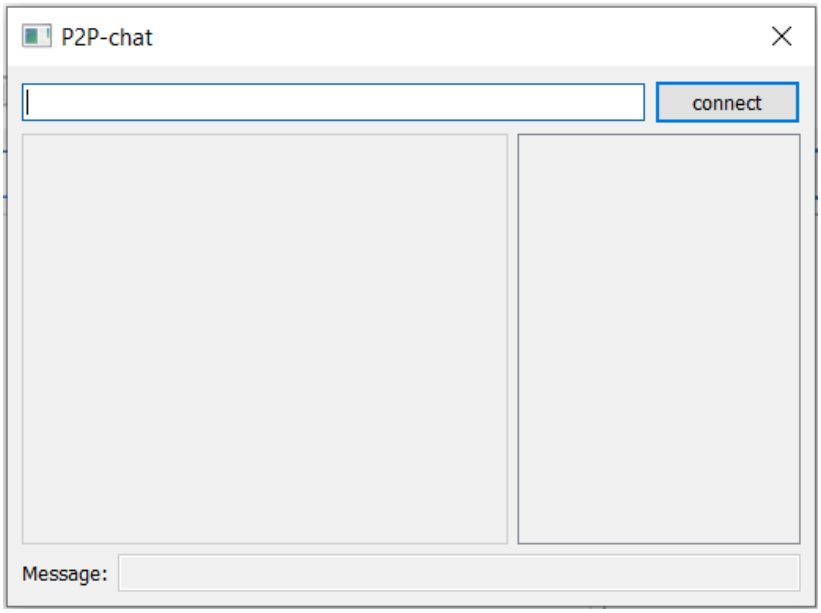

Рисунок 6.1 – Окно программы при запуске.

Для того, чтобы начать пользоваться приложением, необходимо ввести имя пользователя и нажать на кнопку “connect”, после чего появится рабочее окно приложения (рисунок 6.2).

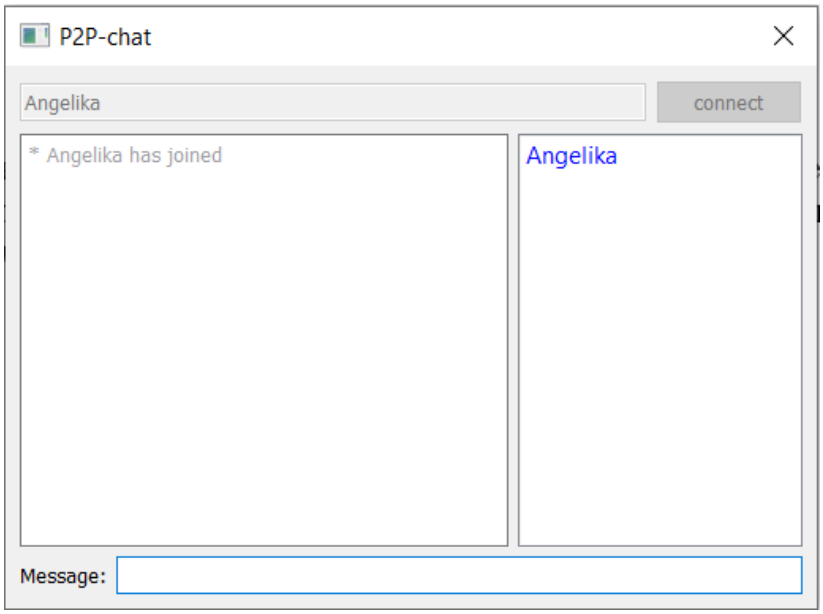

Рисунок 6.2 – Окно программы после подключения.

В рабочем окне приложения можно увидеть список подключенных пользователей (рисунок 6.3), а также принятые и отправленные сообщения рисунок 6.4).

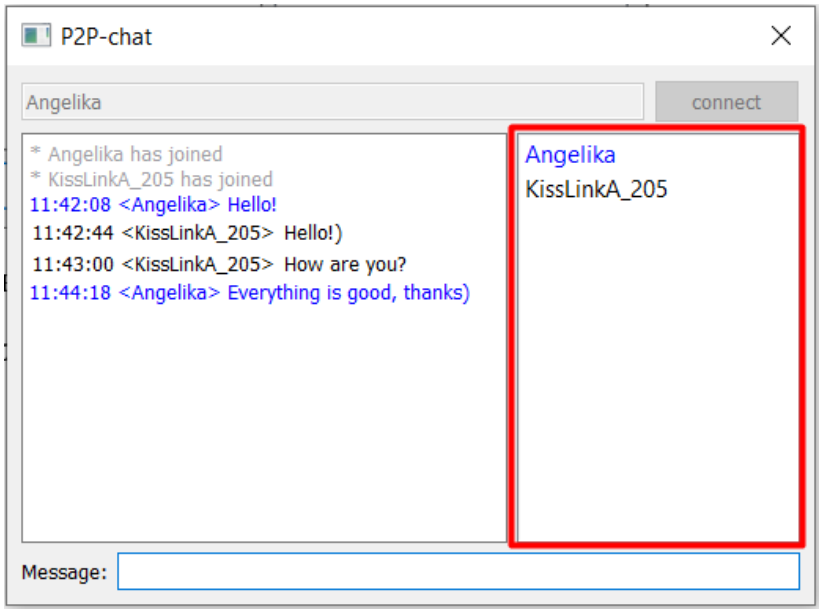

Рисунок 6.3 – Список подключенных пользователей.

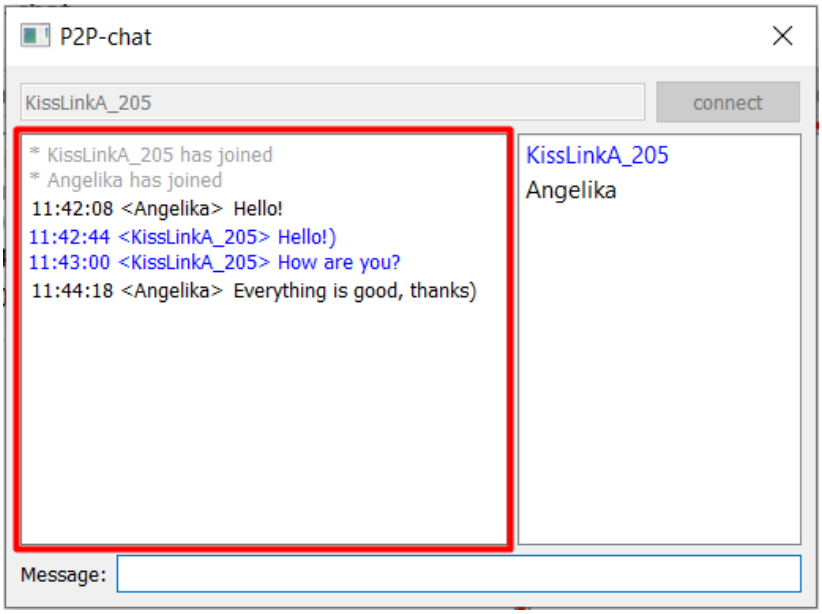

Рисунок 6.4 – Принятые и отправленные сообщения.

При подключении и отключении пользователей на экране выводится соответствующее сообщение и список подключенных пользователей обновляется (рисунок 6.5).

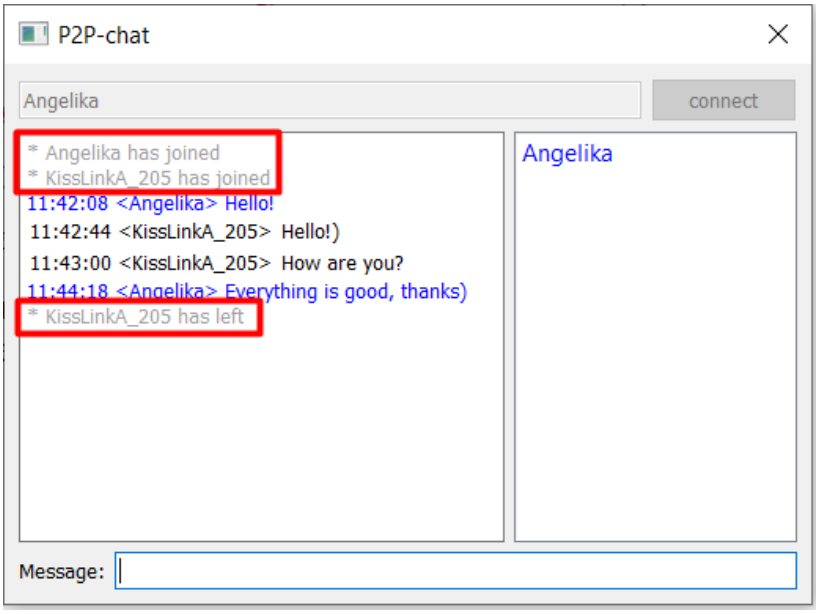

Рисунок 6.5 – Информация при подключении и отключении пользователей.

Для того, чтобы написать сообщение в чат, необходимо набрать необходимый текст в поле ввода “Message” и нажать enter.

Отправленные сообщения и личное имя обозначены синим цветом, принятые от других пользователей сообщения и их имена помечены черным цветом. Это создано для лучшей читабельности и удобства пользователя. Также при отправке и принятии сообщения можно увидеть точное время, когда оно было отправлено (информация находится перед сообщением).

Также в приложении присутствует такое понятие как “команды” (рисунок 6.6). Для того, чтобы написать команду, необходимо начать свое сообщение с символа “/”, если данной команды нет в приложении, то выведется ошибка, если данная команда присутствует, то она выполнится.

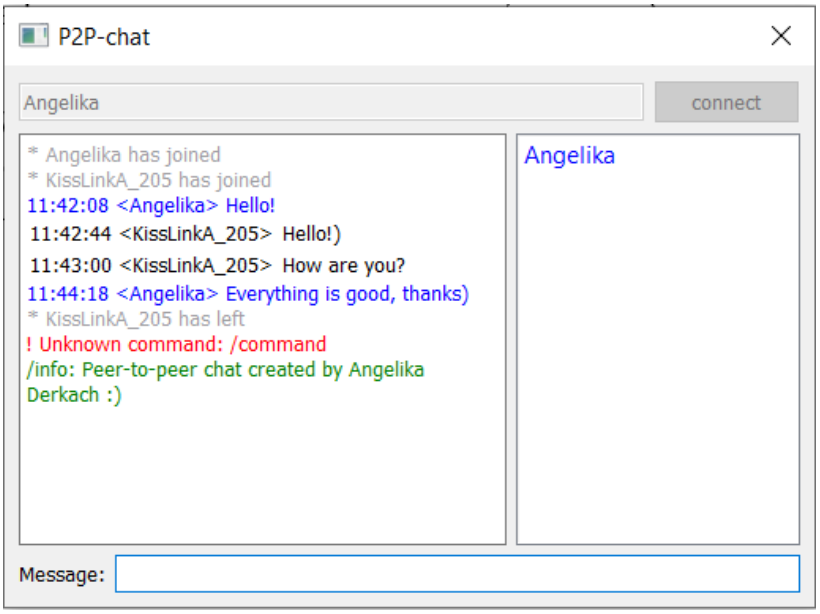

Рисунок 6.6 – Использование команд.

ЗАКЛЮЧЕНИЕ

В результате работы над данным курсовым проектом было разработано работоспособное приложение со своим набором функций и графическим интерфейсом. Данный курсовой проект был разработан в соответствии с поставленными задачами, весь функционал был реализован в полном объеме.

Для создания программного продукта была подробно исследована технология peer-to-peer и принцип работы сетевого чата. В ходе разработки были углублены знания языка программирования C++ и в области объектно-ориентированного программирования, а также получен опыт работы с Фреймворком Qt и с графическим интерфейсом.

Работа была разделена на такие этапы, как анализ существующих аналогов, литературных источников, постановка требований к проектируемому программному продукту, системное и функциональное проектирование, конструирование программного продукта, разработка программных модулей и тестирование проекта. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее приложение.

В дальнейшем планируется усовершенствование текущего функционала приложения, путем улучшения графического интерфейса, добавления новых функций и модулей, а также добавления возможности работы под разными системами.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

[1] Лафоре, Р. Объектно-ориентированное программирование в C++, 4-е издание / Р. Лафоре – СПб.: Питер, 2004.

[2] Документация Qt [Электронный ресурс]. – The Qt Company Ltd. 2019. – Режим доступа: https://doc.qt.io/

[3] Вычислительные машины, системы и сети: дипломное проектирование (методическое пособие) [Электронный ресурс]. – Минск БГУИР 2019. – Режим доступа: https://www.bsuir.by/m/12_100229_1_136308.pdf

ПРИЛОЖЕНИЕ А

(обязательное)

Схема структурная

ПРИЛОЖЕНИЕ Б

(обязательное)

Диаграмма классов

ПРИЛОЖЕНИЕ В

(обязательное)

Листинг кода

//chatdialog.h

```
#ifndef CHATDIALOG_H
#define CHATDIALOG_H

#include "ui_chatdialog.h"
#include "client.h"

class ChatDialog : public QDialog, private Ui::ChatDialog
{
    Q_OBJECT

public:
    ChatDialog(QWidget *parent = nullptr);

public slots:
    void appendMessage(const QString &from, const QString
&message);

private slots:
    void returnPressed();
    void newParticipant(const QString &nick);
    void participantLeft(const QString &nick);
    void on_connectButton_clicked();

private:
    Client client;
    QString myNickName;
    QTextTableFormat tableFormat;
};

#endif
```

//chatdialog.cpp

```
#include <QtWidgets>

#include "chatdialog.h"

ChatDialog::ChatDialog(QWidget *parent)
    : QDialog(parent)
{
    setupUi(this);
```

```
    this->setWindowFlags(this->windowFlags() &
~Qt::WindowContextHelpButtonHint);

    textEdit->setFocusPolicy(Qt::NoFocus);

    inputMessage->setDisabled(true);
    textEdit->setDisabled(true);
    listWidget->setDisabled(true);

    connect(inputMessage, &QLineEdit::returnPressed,
            this, &ChatDialog::returnPressed);
    connect(&client, &Client::newMessage,
            this, &ChatDialog::appendMessage);
    connect(&client, &Client::newParticipant,
            this, &ChatDialog::newParticipant);
    connect(&client, &Client::participantLeft,
            this, &ChatDialog::participantLeft);

    tableFormat.setBorder(0);
}

void ChatDialog::on_connectButton_clicked()
{
    if (nickName->text().isEmpty()) {
        QMessageBox messageBox;
        messageBox.critical(0, "Error", "Name is empty!");
        messageBox.setFixedSize(500, 250);
        return;
    }

    client.setParameters(nickName->text());
    myNickName = nickName->text();
    newParticipant(myNickName);

    inputMessage->setDisabled(false);
    textEdit->setDisabled(false);
    listWidget->setDisabled(false);
    nickName->setDisabled(true);
    connectButton->setDisabled(true);

    inputMessage->setFocusPolicy(Qt::StrongFocus);
    listWidget->item(0)->setForeground(Qt::blue);
}

void ChatDialog::appendMessage(const QString &from, const
QString &message)
{
    if (from.isEmpty() || message.isEmpty())
        return;

    QTextCursor cursor(textEdit->textCursor());
    cursor.movePosition(QTextCursor::End);
```

```
    if (from == nickName->text()) {

        textEdit->setTextColor(Qt::blue);
        textEdit-
>append(QTime::currentTime().toString("HH:mm:ss") + " <" + from
+ "> " + message);

    } else {
        QTextTable *table = cursor.insertTable(1, 2,
tableFormat);
        table->cellAt(0,
0).firstCursorPosition().insertText(QTime::currentTime().toStrin
g("HH:mm:ss") + " <" + from + "> ");
        table->cellAt(0,
1).firstCursorPosition().insertText(message);
    }
    QScrollBar *bar = textEdit->verticalScrollBar();
    bar->setValue(bar->maximum());
}

void ChatDialog::returnPressed()
{
    QString text = inputMessage->text();
    if (text.isEmpty()) {
        QMessageBox messageBox;
        messageBox.warning(0, "Error", "Message is empty!");
        messageBox.setFixedSize(500, 250);
        return;
    }

    if (text.startsWith(QChar('/'))) {
        QColor color = textEdit->textColor();
        if(text == "/info") {
            textEdit->setTextColor(Qt::darkGreen);
            textEdit->append(tr("/info: Peer-to-peer chat
created by Angelika Derkach :)"));
        } else {
            textEdit->setTextColor(Qt::red);
            textEdit->append(tr("! Unknown command: %1")
                        .arg(text.left(text.indexOf(' '))));
            textEdit->setTextColor(color);
        }
    } else {
        client.sendMessage(text);
        appendMessage(myNickName, text);
    }

    inputMessage->clear();
}

void ChatDialog::newParticipant(const QString &nick)
{
```

```
    if (nick.isEmpty())
        return;

    QColor color = textEdit->textColor();
    textEdit->setTextColor(Qt::gray);
    textEdit->append(tr("* %1 has joined").arg(nick));
    textEdit->setTextColor(color);
    listWidget->addItem(nick);
}

void ChatDialog::participantLeft(const QString &nick)
{
    if (nick.isEmpty())
        return;

    QList<QListWidgetItem *> items = listWidget->findItems(nick,

Qt::MatchExactly);
    if (items.isEmpty())
        return;

    delete items.at(0);
    QColor color = textEdit->textColor();
    textEdit->setTextColor(Qt::gray);
    textEdit->append(tr("* %1 has left").arg(nick));
    textEdit->setTextColor(color);
}
```

//client.h

```
#ifndef CLIENT_H
#define CLIENT_H

#include <QAbstractSocket>
#include <QHash>
#include <QHostAddress>

#include "server.h"

class PeerManager;

class Client : public QObject
{
    Q_OBJECT

public:
    Client();
    void sendMessage(const QString &message);
    bool hasConnection(const QHostAddress &senderIp, int
senderPort = -1) const;
    void setParameters(QString username);
```

```
    QString getUserName();

signals:
    void newMessage(const QString &from, const QString
&message);
    void newParticipant(const QString &nick);
    void participantLeft(const QString &nick);

private slots:
    void newConnection(Connection *connection);
    void connectionError(QAbstractSocket::SocketError
socketError);
    void disconnected();
    void readyForUse();

private:
    void removeConnection(Connection *connection);

    QString userName;
    PeerManager *peerManager;
    Server server;
    QMultiHash<QHostAddress, Connection *> peers;
};

#endif
```

//client.cpp

```
#include <QtNetwork>

#include "client.h"
#include "connection.h"
#include "peermanager.h"

Client::Client(){
}

void Client::setParameters(QString username) {
    this->userName = username;
    peerManager = new PeerManager(this);
    peerManager->setServerPort(server.serverPort());
    peerManager->startBroadcasting();

    connect(peerManager, &PeerManager::newConnection,
            this, &Client::newConnection);
    connect(&server, &Server::newConnection,
            this, &Client::newConnection);
}

void Client::sendMessage(const QString &message)
{
```

```
    if (message.isEmpty())
        return;

    for (Connection *connection : qAsConst(peers))
        connection->sendMessage(message);
}

bool Client::hasConnection(const QHostAddress &senderIp, int
senderPort) const
{
    if (senderPort == -1)
        return peers.contains(senderIp);

    if (!peers.contains(senderIp))
        return false;

    const QList<Connection *> connections =
peers.values(senderIp);
    for (const Connection *connection : connections) {
        if (connection->peerPort() == senderPort)
            return true;
    }

    return false;
}

void Client::newConnection(Connection *connection)
{
    connection->setGreetingMessage(peerManager->getUserName());

    connect(connection, &Connection::errorOccurred, this,
&Client::connectionError);
    connect(connection, &Connection::disconnected, this,
&Client::disconnected);
    connect(connection, &Connection::readyForUse, this,
&Client::readyForUse);
}

void Client::readyForUse()
{
    Connection *connection = qobject_cast<Connection
*>(sender());

    if (!connection || hasConnection(connection->peerAddress(),
                                     connection->peerPort()))
        return;

    connect(connection,  &Connection::newMessage,
            this, &Client::newMessage);

    peers.insert(connection->peerAddress(), connection);
    QString nick = connection->getUserName();
```

```
    if (!nick.isEmpty())
        emit newParticipant(nick);
}

void Client::disconnected()
{
    if (Connection *connection = qobject_cast<Connection
*>(sender()))
        removeConnection(connection);
}

void Client::connectionError(QAbstractSocket::SocketError /*
socketError */)
{
    if (Connection *connection = qobject_cast<Connection
*>(sender()))
        removeConnection(connection);
}

void Client::removeConnection(Connection *connection)
{
    if (peers.contains(connection->peerAddress())) {
        peers.remove(connection->peerAddress());
        QString nick = connection->getUserName();
        if (!nick.isEmpty())
            emit participantLeft(nick);
    }
    connection->deleteLater();
}

QString Client::getUserName() {
    return userName;
}
```

//connection.h

```
#ifndef CONNECTION_H
#define CONNECTION_H

#include <QCborStreamReader>
#include <QCborStreamWriter>
#include <QElapsedTimer>
#include <QHostAddress>
#include <QString>
#include <QTcpSocket>
#include <QTimer>

static const int MaxBufferSize = 1024000;

class Connection : public QTcpSocket
{
```

```
    Q_OBJECT

public:
    enum ConnectionState {
        WaitingForGreeting,
        ReadingGreeting,
        ReadyForUse
    };
    enum DataType {
        PlainText,
        Ping,
        Pong,
        Greeting,
        Undefined
    };

    Connection(QObject *parent = nullptr);
    Connection(qintptr socketDescriptor, QObject *parent =
nullptr);
    ~Connection();

    QString getUserName() const;
    void setGreetingMessage(const QString &message);
    bool sendMessage(const QString &message);

signals:
    void readyForUse();
    void newMessage(const QString &from, const QString
&message);

protected:
    void timerEvent(QTimerEvent *timerEvent) override;

private slots:
    void processReadyRead();
    void sendPing();
    void sendGreetingMessage();

private:
    void processGreeting();
    void processData();

    QCborStreamReader reader;
    QCborStreamWriter writer;
    QString greetingMessage;
    QString username;
    QTimer pingTimer;
    QElapsedTimer pongTime;
    QString buffer;
    ConnectionState state;
    DataType currentDataType;
    int transferTimerId;
```

```
    bool isGreetingMessageSent;
};

#endif
```

//connection.cpp

```
#include "connection.h"

#include <QtNetwork>

static const int TransferTimeout = 30 * 1000;
static const int PongTimeout = 60 * 1000;
static const int PingInterval = 5 * 1000;

Connection::Connection(QObject *parent)
    : QTcpSocket(parent), writer(this)
{
    greetingMessage = tr("undefined");
    username = tr("unknown");
    state = WaitingForGreeting;
    currentDataType = Undefined;
    transferTimerId = -1;
    isGreetingMessageSent = false;
    pingTimer.setInterval(PingInterval);

    connect(this, &QTcpSocket::readyRead, this,
            &Connection::processReadyRead);
    connect(this, &QTcpSocket::disconnected,
            &pingTimer, &QTimer::stop);
    connect(&pingTimer, &QTimer::timeout,
            this, &Connection::sendPing);
    connect(this, &QTcpSocket::connected,
            this, &Connection::sendGreetingMessage);
}

Connection::Connection(qintptr socketDescriptor, QObject
*parent)
    : Connection(parent)
{
    setSocketDescriptor(socketDescriptor);
    reader.setDevice(this);
}

Connection::~Connection()
{
    if (isGreetingMessageSent) {
        writer.endArray();
        waitForBytesWritten(2000);
    }
}
```

```
QString Connection::getUserName() const
{
    return username;
}

void Connection::setGreetingMessage(const QString &message)
{
    greetingMessage = message;
}

bool Connection::sendMessage(const QString &message)
{
    if (message.isEmpty())
        return false;

    writer.startMap(1);
    writer.append(PlainText);
    writer.append(message);
    writer.endMap();
    return true;
}

void Connection::timerEvent(QTimerEvent *timerEvent)
{
    if (timerEvent->timerId() == transferTimerId) {
        abort();
        killTimer(transferTimerId);
        transferTimerId = -1;
    }
}

void Connection::processReadyRead()
{
    reader.reparse();
    while (reader.lastError() == QCborError::NoError) {
        if (state == WaitingForGreeting) {
            if (!reader.isArray())
                break;                  // protocol error

            reader.enterContainer();
            state = ReadingGreeting;
        } else if (reader.containerDepth() == 1) {
            if (!reader.hasNext()) {
                reader.leaveContainer();
                disconnectFromHost();
                return;
            }

            if (!reader.isMap() || !reader.isLengthKnown() ||
reader.length() != 1)
                break;                  // protocol error
```

```
            reader.enterContainer();
        } else if (currentDataType == Undefined) {
            if (!reader.isInteger())
                break;                  // protocol error
            currentDataType = DataType(reader.toInteger());
            reader.next();
        } else {
            if (reader.isString()) {
                auto r = reader.readString();
                buffer += r.data;
                if (r.status != QCborStreamReader::EndOfString)
                    continue;
            } else if (reader.isNull()) {
                reader.next();
            } else {
                break;                   // protocol error
            }

            reader.leaveContainer();
            if (transferTimerId != -1) {
                killTimer(transferTimerId);
                transferTimerId = -1;
            }

            if (state == ReadingGreeting) {
                if (currentDataType != Greeting)
                    break;              // protocol error
                processGreeting();
            } else {
                processData();
            }
        }
    }

    if (reader.lastError() != QCborError::EndOfFile)
        abort();       // parse error

    if (transferTimerId != -1 && reader.containerDepth() > 1)
        transferTimerId = startTimer(TransferTimeout);
}

void Connection::sendPing()
{
    if (pongTime.elapsed() > PongTimeout) {
        abort();
        return;
    }

    writer.startMap(1);
    writer.append(Ping);
    writer.append(nullptr);
    writer.endMap();
```

```
}

void Connection::sendGreetingMessage()
{
    writer.startArray();

    writer.startMap(1);
    writer.append(Greeting);
    writer.append(greetingMessage);
    writer.endMap();
    isGreetingMessageSent = true;

    if (!reader.device())
        reader.setDevice(this);
}

void Connection::processGreeting()
{
    username = buffer;
    currentDataType = Undefined;
    buffer.clear();

    if (!isValid()) {
        abort();
        return;
    }

    if (!isGreetingMessageSent)
        sendGreetingMessage();

    pingTimer.start();
    pongTime.start();
    state = ReadyForUse;
    emit readyForUse();
}

void Connection::processData()
{
    switch (currentDataType) {
    case PlainText:
        emit newMessage(username, buffer);
        break;
    case Ping:
        writer.startMap(1);
        writer.append(Pong);
        writer.append(nullptr);
        writer.endMap();
        break;
    case Pong:
        pongTime.restart();
        break;
    default:
```

```
            break;
        }

        currentDataType = Undefined;
        buffer.clear();
    }
```

//peermanager.h

```
#ifndef PEERMANAGER_H
#define PEERMANAGER_H

#include <QByteArray>
#include <QList>
#include <QObject>
#include <QTimer>
#include <QUdpSocket>

class Client;
class Connection;

class PeerManager : public QObject
{
    Q_OBJECT

public:
    PeerManager(Client *client);

    void setServerPort(int port);
    QString getUserName() const;
    void startBroadcasting();
    bool isLocalHostAddress(const QHostAddress &address) const;

signals:
    void newConnection(Connection *connection);

private slots:
    void sendBroadcastDatagram();
    void readBroadcastDatagram();

private:
    void updateAddresses();

    Client *client;
    QList<QHostAddress> broadcastAddresses;
    QList<QHostAddress> ipAddresses;
    QUdpSocket broadcastSocket;
    QTimer broadcastTimer;
    QString username;
    int serverPort;
};
```

```
#endif
```

//peermanager.cpp

```
#include <QtNetwork>

#include "client.h"
#include "connection.h"
#include "peermanager.h"

static const qint32 BroadcastInterval = 2000;
static const unsigned broadcastPort = 45000;

PeerManager::PeerManager(Client *client)
    : QObject(client)
{
    this->client = client;

    username = client->getUserName();

    if (username.isEmpty())
        username = "unknown";

    updateAddresses();
    serverPort = 0;

    broadcastSocket.bind(QHostAddress::Any, broadcastPort,
QUdpSocket::ShareAddress
                         | QUdpSocket::ReuseAddressHint);
    connect(&broadcastSocket, &QUdpSocket::readyRead,
            this, &PeerManager::readBroadcastDatagram);

    broadcastTimer.setInterval(BroadcastInterval);
    connect(&broadcastTimer, &QTimer::timeout,
            this, &PeerManager::sendBroadcastDatagram);
}

void PeerManager::setServerPort(int port)
{
    serverPort = port;
}

QString PeerManager::getUserName() const
{
    return username;
}

void PeerManager::startBroadcasting()
{
```

```
    broadcastTimer.start();
}

bool PeerManager::isLocalHostAddress(const QHostAddress
&address) const
{
    for (const QHostAddress &localAddress : ipAddresses) {
        if (address.isEqual(localAddress))
            return true;
    }
    return false;
}

void PeerManager::sendBroadcastDatagram()
{
    QByteArray datagram;
    {
        QCborStreamWriter writer(&datagram);
        writer.startArray(2);
        writer.append(username);
        writer.append(serverPort);
        writer.endArray();
    }

    bool validBroadcastAddresses = true;
    for (const QHostAddress &address :
qAsConst(broadcastAddresses)) {
        if (broadcastSocket.writeDatagram(datagram, address,
                                          broadcastPort) == -1)
            validBroadcastAddresses = false;
    }

    if (!validBroadcastAddresses)
        updateAddresses();
}

void PeerManager::readBroadcastDatagram()
{
    while (broadcastSocket.hasPendingDatagrams()) {
        QHostAddress senderIp;
        quint16 senderPort;
        QByteArray datagram;
        datagram.resize(broadcastSocket.pendingDatagramSize());
        if (broadcastSocket.readDatagram(datagram.data(),
datagram.size(),
                                         &senderIp, &senderPort)
== -1)
            continue;

        int senderServerPort;
        {
            QCborStreamReader reader(datagram);
```

```
            if (reader.lastError() != QCborError::NoError ||
!reader.isArray())
                continue;
            if (!reader.isLengthKnown() || reader.length() != 2)
                continue;

            reader.enterContainer();
            if (reader.lastError() != QCborError::NoError ||
!reader.isString())
                continue;
            while (reader.readString().status ==
QCborStreamReader::Ok) {
            }

            if (reader.lastError() != QCborError::NoError ||
!reader.isUnsignedInteger())
                continue;
            senderServerPort = reader.toInteger();
        }

        if (isLocalHostAddress(senderIp) && senderServerPort ==
serverPort)
            continue;

        if (!client->hasConnection(senderIp)) {
            Connection *connection = new Connection(this);
            emit newConnection(connection);
            connection->connectToHost(senderIp,
senderServerPort);
        }
    }
}

void PeerManager::updateAddresses()
{
    broadcastAddresses.clear();
    ipAddresses.clear();
    const QList<QNetworkInterface> interfaces =
QNetworkInterface::allInterfaces();
    for (const QNetworkInterface &interface : interfaces) {
        const QList<QNetworkAddressEntry> entries =
interface.addressEntries();
        for (const QNetworkAddressEntry &entry : entries) {
            QHostAddress broadcastAddress = entry.broadcast();
            if (broadcastAddress != QHostAddress::Null &&
entry.ip() != QHostAddress::LocalHost) {
                broadcastAddresses << broadcastAddress;
                ipAddresses << entry.ip();
            }
        }
    }
}
```

//server.h

```
#ifndef SERVER_H
#define SERVER_H

#include <QTcpServer>

class Connection;

class Server : public QTcpServer
{
    Q_OBJECT

public:
    Server(QObject *parent = nullptr);

signals:
    void newConnection(Connection *connection);

protected:
    void incomingConnection(qintptr socketDescriptor) override;
};

#endif
```

//server.cpp

```
#include <QtNetwork>

#include "connection.h"
#include "server.h"

Server::Server(QObject *parent)
    : QTcpServer(parent)
{
    listen(QHostAddress::Any);
}

void Server::incomingConnection(qintptr socketDescriptor)
{
    Connection *connection = new Connection(socketDescriptor,
this);
    emit newConnection(connection);
}
```

//main.cpp

```
#include <QApplication>
#include <QtCore/QSettings>
```

```
#include "chatdialog.h"

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

    ChatDialog dialog;
    dialog.show();
    return app.exec();
}
```

ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов