Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий
кафедрой
Информатики

(подпись)
Волорова Н.А. 2020 г.

ЗАДАНИЕ

по курсовому проекту

Студенту *Кременевскому Владиславу Сергеевичу*

1. Тема работы *Мессенджер.*

2. Срок сдачи студентом законченной работы *31.05.2020.*

3. Исходные данные к работе *Операционная система MacOS. Язык программирования C++.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение. 1. Анализ предметной области. 2. Разработка приложения. 3. Методика работы с полученным приложением. Заключение. Список использованных источников. Приложения.

5. Консультант по курсовой работе *Удовин И. А.*

6. Дата выдачи задания *10.02.2020*

7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1, Введение к 03.03.2020 – 10% готовности работы;

раздел 2 к 15.05.2020г. – 70% готовности работы;

раздел 3 к 20.05.2020г. – 80% готовности работы;

Заключение, Приложение к 23.05.2020г. – 95% готовности работы; оформление пояснительной записки и графического материала к 25.05.2020г. – 100% готовности работы.

Защита курсового проекта с 31.05.2020 г. по 01.06.2020 г.

РУКОВОДИТЕЛЬ *Удовин И.А.*

Задание принял к исполнению Кременевский. В. С. 10.02.2020 г.
(дата и подпись студента)

ОГЛАВЛЕНИЕ

Введение

“Кто владеет информацией, тот владеет миром”, - Натан Ротшильд.

Современную жизнь, а тем более современный бизнес невозможно представить без информационных технологий. Мир окутан информационными потоками, имеющими разную среду для ее передачи. Сбор, анализ и хранение информации – это первостепенные задачи не только современного человека, но и общества в целом, в том числе и бизнеса.

Раньше в глобальной сети пользователи, в основном, общались через социальные сети. Теперь появились и другие возможности. Достаточно иметь под рукой ноутбук, телефон или планшет. С этих устройств можно отправлять документы, файлы и сообщения без использования платных СМС, а также совершать звонки. Это стало возможным вместе с появлением мессенджеров.

Слово «мессенджер» произошло от английского «messenger», что означает курьер. Обмен сообщениями идет мгновенно, в режиме реального времени. Сообщения отправляются собеседнику сразу после того, как отправитель закончит ввод, редактирование и нажмет на кнопку отправки. Но при этом получатель сообщения должен быть на связи, иначе сообщение вынуждено будет ждать, пока он тоже запустит свой мессенджер и обратит на него внимание.

Если у пользователя безлимитный Wi-Fi, то все сообщения, а также аудио или видеозвонки с помощью мессенджера будут бесплатными, не ограниченными по времени и по количеству.

Многие знают, что смс-ки и звонки по обычной сотовой, мобильной связи зачастую являются платными. Но отказаться от мобильной связи и перейти полностью на мессенджеры вряд ли возможно. Дело в том, что обычно для регистрации там аккаунта требуется подтвердить номер телефона с помощью смс.

К тому же, многие знакомые используют разные мессенджеры, либо вообще не пользуются этими сервисами. С такими абонентами все равно придется общаться по телефону и с помощью смс-сообщений.

1. Анализ предметной области

1.1 Теория

Что такое мессенджер?

Мессенджер – это программа (приложение) для мгновенного обмена сообщениями через интернет. В качестве сообщений мессенджеры могут использовать текст, картинки, видео, некоторые приложения поддерживают передачу файлов любого формата.

Предшественниками мессенджеров являются электронная почта и телефонные смс-ки.

Но почта работает не так быстро. Как правило, почтовые клиенты проверяют входящие письма раз в несколько минут. Для быстрого общения это неудобно, поэтому почту используют чаще для написания больших писем.

СМС по сравнению с современными онлайн мессенджерами слишком дорогое удовольствие, да и функционал сильно устарел (даже смайлики нормально не передать).

Мессенджеры используют для передачи данных интернет, а он сейчас недорогой, даже если брать во внимание мобильные тарифы.

Обмен сообщениями идет мгновенно, пользователь пишут фразу, нажимает кнопку для отправки и его собеседник читает послание спустя доли секунды.

Большинство приложений для связи не только быстро передают сообщения, но и в реальном времени показывают статус контактов, находящихся в записной книжке – вы видите, кто из них в сети и получит ваше сообщение сразу, а кого в сети нет.

1.2 Анализ лидирующих мессенджеров.

Иногда пользователи задаются вопросом — какой мессенджер лучше? На самом деле ответить на этот вопрос так просто нельзя, ведь кому-то нравится один мессенджер, второму человеку — совершенно другой. По этой причине будет приведен список самых популярных мессенджеров на сегодняшний день

Viber. Один из самых популярных мессенджеров как в мире, так и в России. Наделен массой всевозможных функций, в том числе возможностью

звонить на стационарные и мобильные смартфоны — функция называется Viber Out. Правда, эта услуга платная.

WhatsApp. Ничуть не менее, а может, даже более популярный мессенджер — WhatsApp. Простой, удобный, шустрый, позволяет чатиться с друзьями, совершать звонки, обмениваться самыми различными файлами. Есть даже функция смены номера — если вы решили поменять сим-карту.

Telegram. Мессенджер, созданный тем самым Павлом Дуровым, что когда-то основал социальную сеть ВКонтакте. Считается одним из наиболее защищенных мессенджеров в мире, а заодно — одним из наиболее популярным во многих странах.

Skype. Считается, что Viber и WhatsApp были сделаны как альтернатива мессенджеру Skype, который сегодня хоть и утратил лидирующие позиции, но все еще остается одним из наиболее популярных мессенджеров в мире. Пользуются им в Росиии. Доступен для всех популярных операционных систем.

Facebook Messenger. Еще один очень популярный мессенджер, аудитория которого еще весной 2017 года насчитывала порядка 1 миллиарда пользователей — Facebook Messenger. Приложение интегрировано с системой обмена сообщениями на основном сайте Facebook. В России приложение пока не столь популярно, однако им все равно пользуется огромное количество пользователей.

1.3 Интернет

Сеть Интернет представляет собой множество компьютеров, соединенных друг с другом кабелями, а также радиоканалами, спутниковыми каналами и т. д. Однако, как известно, одних проводов или радиоволн для передачи информации недостаточно: передающей и принимающей сторонам необходимо придерживаться ряда соглашений, позволяющих строго регламентировать передачу данных и гарантировать, что эта передача пройдет без искажений. Такой набор правил называется протоколом передачи.

Упрощенно, протокол — это набор правил, который позволяет системам, взаимодействующим в рамках сети, обмениваться данными в наиболее удобной для них форме. Разумеется, для разных целей существуют различные протоколы. Любой компьютер, подключенный к Интернету и желающий обмениваться информацией со своими “сородичами”, должен иметь некоторое уникальное имя, или IP-адрес. Вот уже 30 лет среднестатистический IP-адрес выглядит примерно так: 127.12.232.56

В любом случае IP-адрес — дефицитный ресурс, получить который не просто. Поэтому в ближайшие годы нас ожидает смена версии протокола с текущей IPv4 на новую IPv6, в которой адрес расширяется с 32 до 128 бит (шестнадцать 8-разрядных чисел). С переходом на IPv6 будет достаточно выделяемых персональных IP-адресов каждому жителю планеты и каждому производимому устройству.

И все-таки обычным людям довольно неудобно работать с IP-представлением адреса, особенно с новыми IPv6-адресами. Действительно, куда как проще запомнить символьное имя, чем набор чисел. Чтобы облегчить простым пользователям работу с Интернетом, придумали систему DNS (Domain Name System, служба имен доменов).

Итак, при использовании DNS любой компьютер в Сети может иметь не только IP-адрес, но также и символическое имя. Выглядит оно примерно так: www.google.com

То есть это набор слов (их число произвольно), опять же разделенных точкой. Каждое такое сочетание слов называется доменом N-го уровня (например, com — домен первого уровня, google.com — второго и т. д.)

Порт записывается после IP-адреса или доменного имени через двоеточие. Порт 80 является стандартным для обмена данными с Web-сервером, порт 443 — стандартным для зашифрованного SSL-соединения. В повседневной жизни при использовании адресов мы не указываем стандартные порты, браузеры и другие сетевые клиенты назначают их автоматически. Однако, если сервер использует какой-то нестандартный порт, например 8080, его придется указывать в адресе явно: http://localhost:8080.

Как только обмен “приветственными” сообщениями закончен (его еще называют “тройным рукопожатием”, потому что в общей сложности посылаются 3 таких сообщения), между Клиентом и Сервером

устанавливается логический канал связи. Программы могут использовать его, как обычный канал UNIX (это напоминает случай файла, открытого на чтение и запись одновременно). Иными словами, Клиент может передать данные Серверу, записав их с помощью системной функции в канал, а Сервер — принять их, прочитав из канала.

1.4 Типы сетевых протоколов

Рассмотрим несколько самых популярных сегодня протоколов, работающих на разных уровнях:

MAC. Уже известный вам Media Access Control (MAC) это низкоуровневый сетевой протокол. С ним, в той или иной мере приходится сталкиваться всем пользователям. Используется он для идентификации сетевых устройств.

IP. На следующем уровне после MAC располагается IP – Internet Protocol, имеющий две основные разновидности IPv4 и IPv6. Он назначает компьютерам уникальные IP-адреса, благодаря которым устройства могут себя обнаруживать в сети. маршрутизируемый протокол сетевого уровня стека TCP/IP. Именно IP стал тем протоколом, который объединил отдельные компьютерные сети во всемирную сеть Интернет. Неотъемлемой частью протокола является *адресация* сети (см. IP-адрес). IP объединяет сегменты сети в единую сеть, обеспечивая доставку пакетов данных. IP не гарантирует надёжной доставки пакета до адресата — в частности, пакеты могут прийти не в том порядке, в котором были отправлены, продублироваться (приходят две копии одного пакета), оказаться повреждёнными (обычно повреждённые пакеты уничтожаются) или не прийти вовсе. Гарантию безошибочной доставки пакетов дают некоторые протоколы более высокого уровня — *транспортного уровня*, например, TCP, которые используют IP в качестве транспорта. Выше IP находятся такие протоколы:

ICMP (Internet control message protocol), отвечающий за обмен информацией. Не используется для передачи данных. Именно ICMP используется в известной всем команде ping.

TCP (Transmission control protocol). Этот сетевой протокол управляет передачей данных. TCP дает гарантия в том, что все переданные пакеты данных будут приняты правильно и ошибки будут полностью исключены.

UDP (user datagram protocol) похож на TCP, но работает быстрее, так как в нем данные при получении не проверяются. В некоторых случаях использование UDP бывает вполне достаточным.

HTTP. По своей распространенности в Интернет Hyper Text Transfer protocol находится на первом месте, ведь именно на его основе работают все сайты. С его помощью с локального компьютера можно открыть веб-сервис на удалённом сервере.

FTP. Как понятно из перевода названия, File Transfer Protocol служит для передачи файлов.

POP3 и SMTP. Два протокола для работы с электронной почтой. POP3 отвечает за получение почты, SMTP – за отправку. Оба протокола позволяют доставлять email, передавая его почтовому клиенту.

SSH. Протокол SSH (Secure Shell) относится к уровню приложений. Создает защищенный канал для удаленного управления другой операционной системой. Поддерживает различные алгоритмы шифрования

1.5 Отличия TCP и UDP протоколов

Следует обратить внимание на разницу между двумя транспортными протоколами.

TCP – транспортный протокол передачи данных в сетях TCP/IP, предварительно устанавливающий соединение с сетью.

UDP – транспортный протокол, передающий сообщения-датаграммы без необходимости установки соединения в IP-сети.

1. TCP гарантирует доставку пакетов данных в неизменных виде, последовательности и без потерь, UDP ничего не гарантирует.

2. TCP нумерует пакеты при передаче, а UDP нет

3. TCP работает в дуплексном режиме, в одном пакете можно отправлять информацию и подтверждать получение предыдущего пакета.

4. TCP требует заранее установленного соединения, UDP соединения не требует, у него это просто поток данных.

5. UDP обеспечивает более высокую скорость передачи данных.

6. TCP надежнее и осуществляет контроль над процессом обмена данными.

7. UDP предпочтительнее для программ, воспроизводящих потоковое видео, видеотелефонии и телефонии, сетевых игр.

8. UPD не содержит функций восстановления данных

Надежность TCP протокола достигается за счет тройного рукопожатия.

После того как хэндшейк совершен, может быть начат обмен данными. Клиент может отправить пакет данных сразу после ACK-пакета, сервер должен дождаться ACK-пакета, чтобы начать отправлять данные. Этот процесс происходит при каждом TCP-соединении и представляет серьезную сложность в плане производительности сайтов. Ведь каждое новое соединение означает некоторую сетевую задержку.

В то время как UDP протокол, не требует подтверждения ack. Просто при необходимости отправитель отсылает данные, получатель принимает, без гарантии, что все пришло в нужном порядке, а так же без гарантии того, что не произошла потеря данных. За счет этого UDP считается более быстрым протоколом передачи данных.

Основное использование TCP там, где необходима гарантия, что все данные придут в целости и в нужном порядке без потери, например при отправке сообщений от одного пользователя к другому, необходимо, чтобы все сообщения пришли в целости и сохранности. Именно поэтому было принято решение писать мессенджер на TCP сервере, который будет проверять и давать нам гарантию того, что все данные в точности были отправлены нашему получателю.

UDP протокол, напротив, используется, например, при стриминге, трансляции, в онлайн многопользовательских играх. Именно поэтому иногда можно было заметить, такую резкую прогрузку кадров, которые пришли до этого, не было гарантии что все пакеты придут в нужном порядке и не потеряются, некоторая часть пакетов приходит позже, и потом все прогружается, стараясь восстановить текущую связь.

1.6 Протокол TCP/IP

TCP/IP - основополагающее звено всего интернета. TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Наборы правил, решающих задачу по передаче данных, составляют стек протоколов передачи данных, на которых базируется Интернет. TCP/IP - потому что название происходит из двух основополагающих проколов TCP и IP, о которых было упомянуто ранее.

TCP- протокол управления передачей, IP - интернет протокол, для идентификации пользователя.

Набор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься. Эта функциональность организована в четыре слоя абстракции, которые классифицируют все

связанные протоколы в соответствии с объемом задействованных сетей. От самого низкого до самого высокого уровня — это уровень связи, содержащий методы связи для данных, которые остаются в пределах одного сегмента сети; интернет-уровень, обеспечивающий межсетевое взаимодействие между независимыми сетями; транспортный уровень, обрабатывающий связь между хостами; и прикладной уровень, который обеспечивает обмен данными между процессами для приложений.

На стеке протоколов TCP/IP построено всё взаимодействие пользователей в IP-сетях. Стек является независимым от физической среды передачи данных, благодаря чему, в частности, обеспечивается полностью прозрачное взаимодействие между проводными и беспроводными сетями.

1.7 Локальные и глобальные сети.

Глобальные сети (WAN) - любые сети, которые охватывают всю Землю. Локальные сети - сети, которые покрывают довольно небольшую площадь взаимодействия подключённых устройств ~ до 2 км.

В повседневной жизни мы чаще, конечно, уже сегодня встречается в глобальными сетями, различные сайты, стриминговые сервисы, все глобальные мессенджеры функционируют в рамках глобальных сетей. Локальные сети чаще используется для объединения организаций а общую сеть.

И тут стоит отметить очень важную деталь, касающуюся IP адресов. Все адреса протокола IPv4 делятся на «белые» - публичные, глобальные, внешние и «серые» - частные, локальные, внутренние. Проблема в том, что протокол IPv4 не был изначально рассчитан на такое большое количество пользователей интернета, и уже сегодня количество пользователей превышает допустимое максимальное количество айпи адресов IPv4.

Поэтому было принято решение разграничить айпи адреса, появилось разделение : «Белые» и «Серые». Белые непосредственно предоставляет

провайдер за отдельную плату, а серые создаются внутри локальной сети, например, при подключении нескольких компьютеров к одному роутеру. И каждому присваивается свой уникальный в рамках сети айпи адрес. И уже в рамках созданной локальной сети, компьютеры могут взаимодействовать между собой.

1.8 Выбор инструментария

Для разработки курсового проекта был выбран фреймворк для разработки программного обеспечения на языке программирования C++, который носит название **Qt**. Непосредственно разработка в Qt Creator. Отличительная особенность — использование метаобъектного компилятора — предварительной системы обработки исходного кода. Расширение возможностей обеспечивается системой плагинов, которые возможно размещать непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

C++ — компилируемый, статически типизированный язык программирования общего назначения. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности.

Git – распределенная система контроля версий, для сохранения прогресса на разных этапах разработки и возможности отката.

GitHub – веб-сервис для хостинга IT-проектов и их совместной разработки. Для хранения курсового проекта не только локально на компьютере.

1.9 Постановка задачи

Задачей курсового проекта является создание кросплатформенного мессенджера написанного на языке C++ с использованием Qt фреймворка. Который имеет базовый функционал в виде:

1. Отправки сообщений другим пользователям.
2. Поддержка нескольких пользователей
3. Подключения, отключения от сервера

Имеет дополнительный функционал в виде:

1. Приятный и удобный, а также интуитивно понятный графический интерфейс.
2. Возможность выбора хоста, на котором можно разместить сервер (Серверная часть).
3. Возможность выбора сервера, к которому может подключиться клиент.
4. Безопасное отключение от сервера при необходимости.
5. Безопасное закрытие сервера, с уведомление об этом всех подключённых клиентов.
6. Возможность смена никнейма онлайн, с уведомлением об этом всех подключённых пользователей.
7. Уведомление всех пользователей о подключении нового клиента к серверу.
8. Уведомление всех пользователей при отключении клиента от сервера.
9. Лог сервера, с выводом всех поступающих к нему сообщений, а также обработку ошибок.

Если все просуммировать, то необходимо:

1. Разработать надежный рабочий сервер, для обслуживания всех клиентов.

2. Разработать интуитивно понятное клиент приложение, которое будут использовать другие пользователи, для обмена информаций онлайн.

2. Разработка проекта

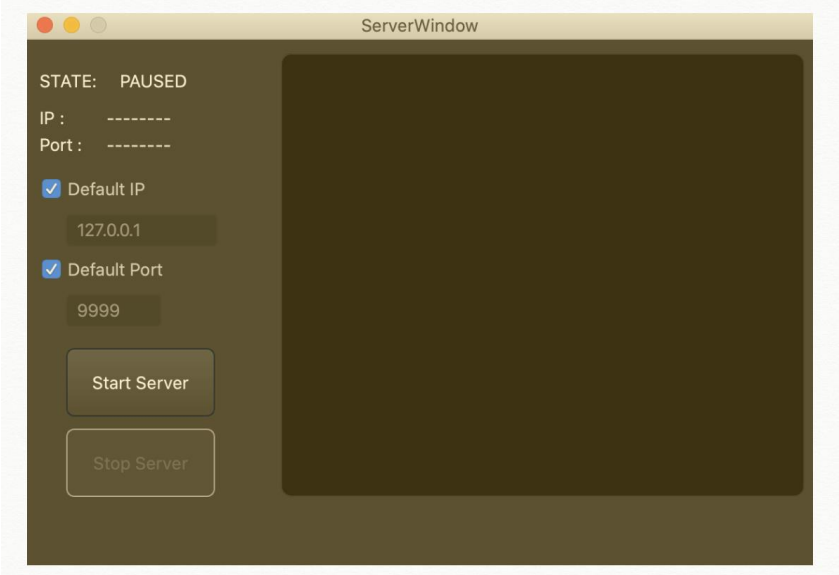
2.1 Реализация интерфейса

При создании интерфейса были приняты во внимания следующие критерии:

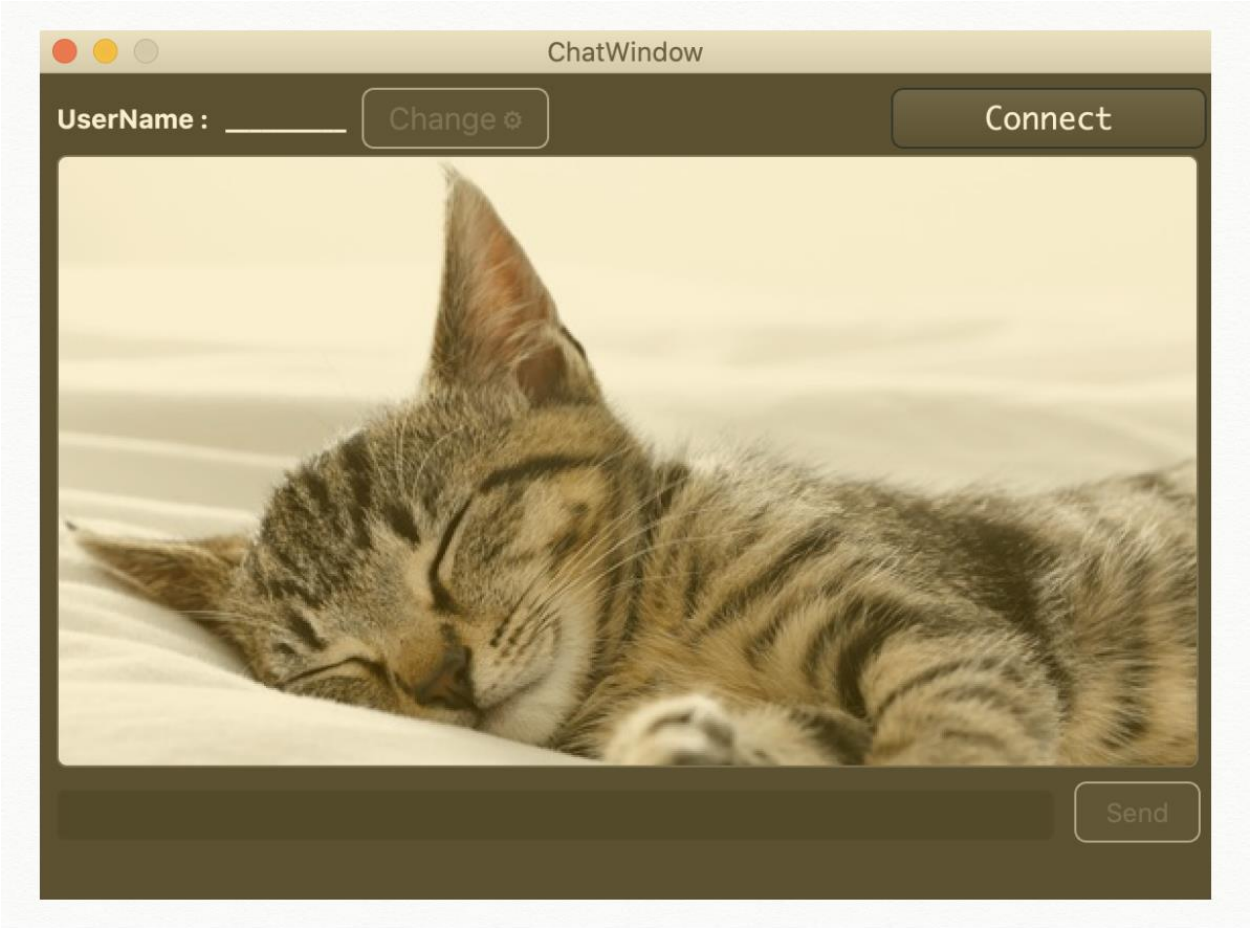
1. Интерфейс должен быть интуитивно понятен, чтобы любой пользователь смог с легкостью в нем разобраться.
2. Интерфейс должен доставлять удовольствие пользователю при работе с ним.
3. Все элемента должны быть логически взаимосвязаны. Логично в современном мессенджера оставлять рядом виджеты MessageEdit (для ввод текста сообщения) и кнопку отправки сообщения.

Пользовательский интерфейс разделяется на отдельно представленный интерфейс сервера и отдельно представленный интерфейс клиента.

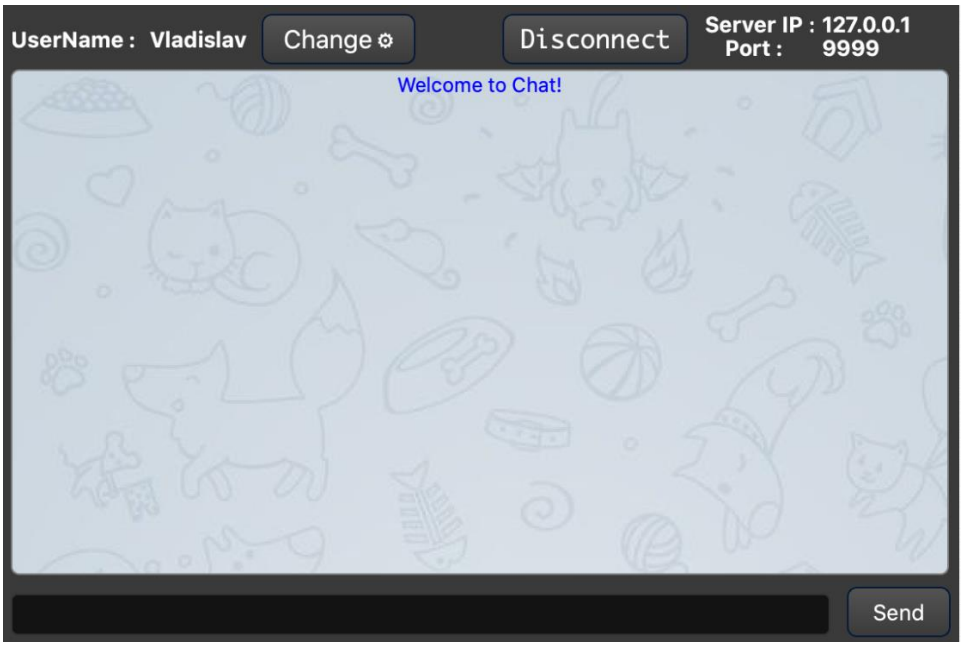
На стороне сервера, представлено одно окно, в котором, содержаться нужные кнопки для запуска сервера, а также лог сервера – отдельное окно, куда выводится вся нужная информация, которую обрабатывает сервер в процессе взаимодействия с клиентами.



На стороне клиента при запуске представлено главное окно с кнопкой connect, для подключения к серверу, и приятной заставкой



Уже после подключения к серверу, меняется картинка, добавляется окно чата, появляется возможность отправки сообщений, активируется кнопка смены имени, а также кнопка выхода из чата.



2.2 Совместимость

Так как используется для разработки фреймворк Qt, а он является кроссплатформенным решением для разработки програмного обеспечения на C++, то очевидно, что проект написанный в Qt, будет работать на всех, поддерживающих Qt, основных операционных системах, а именно:

1. Встраиваемые Linux-системы:
2. MacOS X
3. Windows

Таким образом можно сделать вывод, что написанный Мессенджер будет совместим во всеми основными десктопными ОС.

2.3 Функциональность

Так как помимо базовой функции отправления сообщений, было решено реализовать такие функции как:

1. Возможность смены Никнейма
2. Возможность отключения от сервера в любой момент
3. Возможность получать сообщения при подключении к чату нового пользователя
4. Возможность получать сообщений при отключении пользователя от сервера / сети
5. Выбор порта и хоста для размещения сервера
6. Выбор порта и хоста при подключении клиента
7. Вывод полного лога (истории всех произведенных операций) внутри сервера
8. Обработку всех возможных ошибок

В таком случае на сервер должны будут приходить сообщения, в которых будет содержаться информация о запрошенных действиях непосредственно от пользователя или от сервера к серверу для обработки некоторых ситуаций, и в соответствии с полученной информаций сервер должен будет правильно обработать запрос и выполнить нужное действие.

Отправка данных просто в формате string, представленный как QString в Qt, является не рациональным решением, так как распаковать string является довольно трудоемкой работой, да и к тоже же тем более не рациональной.

Поэтому я решил осуществить взаимодействие клиента и сервера через формат JSON - java script object notation, который позволяет как нужно упаковать нужное сообщение и отправить его на сервер / клиент, где в последующем будет произведена обработка этого сообщения и выполнения необходимого действия.

2.4 Принцип работы

Вся информация о приведенных ниже классах будет рассмотрена в следующей главе “Реализация сервера” и “Реализация клиента”

Прежде всего, чтобы установить соединение между клиентом и сервером, мы должны разработать наш сервер, который сможет открывать выбранный нами порт и “Прослушивать” (listning) уже там все попытки подключения клиента(нашего пользователя) к этому серверу.

server->**listen**(hostAddress, port)

При попытке “постучаться” на сервер от клиента, у клиента создается Socket (Socket - абстракция, которая означает конечную точку соединения) на стороне пользователя и на сервер будет посылаться сообщение о попытке установления соединения.

void ***incomingConnection***(qintptr handle) override;

Далее на сервере создается объект класса Worker, в котором создается socket на стороне сервера, чтобы в дальнейшем сервер и пользователь могли обмениваться информацией.

Далее сервер получает информацию об имени пользователя и уведомление о попытке присоединения к серверу клиента с выбранным пользователем именем. Сервер проверяет нет ли уже пользователей с таким именем, и при отсутствии таких генерирует jsonObject { “type” :”login”, “success” : true} (при наличии уже пользователей с таким именем ключ “success” имеет значение false) после чего этот объект преобразуется в формат UTF8 и отсылается к пользователю.

Пользователь в свою очередь видит, что к нему поступило сообщение и пытается сделать его разбор.

Разбор происходит следующим образом

1. Полученная информация пытыется считаться пользователем
2. Если полученную информацию возможно преобразовать в jsonDocument -> преобразуем, иначе игнорируем (нам не интересны сообщения, с которыми мы не можем работать)
3. Проверяется содержится ли в полученном jsonDocument объект jsonObject который имеет следующий вид: {“key” : value, “key2” : value2 и т.д} -> если является объектом то принимаем его и генерируем сигнал jsonReceived(const QJsonObject &object), иначе игнорируем.

```
void ChatClient::onReadyRead()
{
    QByteArray jsonArray;
    QDataStream clientStream(clientSocket);

    for(;;){
        clientStream.startTransaction();

        clientStream >> jsonArray;

        if (clientStream.commitTransaction()){
            QJsonParseError jsonError;
            QJsonDocument jsonDoc = QJsonDocument::fromJson(jsonArray, &jsonError);

            if(jsonError.error == QJsonParseError::NoError)
                if(jsonDoc.isObject())
                    jsonReceived(jsonDoc.object());
        }
        else
            break;
    }
}
```

Далее происходит разбор полученного jsonObject В котором проверяется тип сообщения

```
const QJsonValue typeValue = docObj.value("type");

if(typeValue.toString() == "login"){ // попытка установить связь между сокетом и клиентом

    …

    const QJsonValue loginValue = docObj.value("success");

     ...

    if (loginSuccess){ // если все успешно, сообщаем об этом клиенту

        emit loggedIn();

        return;

    }

emit loginError(reasonValue.toString()); // сообщаем клиенту об ошибке
```

….

else if(typeValue.toString() == "message”){ // Если пришло сообщение

….

else if(typeValue.toString() == "newUser"){ // Если подключился новый пользователь

….

else if(typeValue.toString() == "userDisconnected"){ // Если пользователь отключился

….

else if(typeVlue.toString() == "changeName"){ // Если кто-то сменил ник, что уведомить об этом всех

….

if (typeValue.toString() == "resultNameChanged"){ // проверяем можно ли сменить ник и предпринимаем нужные действия

Если typeValue (тип сообщения) равно одному из этих значений, то воспроизводим необходимые действия со стороны клиента.

Далее, например, при попытке отправить сообщение от пользователя, на пользователе создается снова jsonObject, который содержит информацию об действии, которое необходимо совершить, в данном случает отправить сообщение:

1. Проверяется все ли было сделано верно
2. Упаковываем всю информацию в jsonObject, например:

{“type” : “message”,
“text” : <textFromUsser>,
“time” : <timeMessageSend> }

Этот объект в свою очередь переводить снова в формат UTF8 и отправляется на сокет со стороны сервера, который также проверяет, можно ли работать с этой информацией (проверка на jsonDocument и jsonObject)

Производим разбор полученной информации на сервере, почти также как на клиенте и выполняем необходимые действия.

Таким образом сервер получает информацию о каждом действии клиента, и каждой операции, которую хочет выполнить пользователь, обрабатывает ее и уже далее генерирует новый jsonObject с результатам на запрошенное действие, после чего на стороне клиента происходят определённые действия в ответ на его запрос, либо же пользователю приходит сообщение об ошибке, так как в соответствии с логикой сервера возникли какие-то неполадки.

Чтобы была возможность взаимодействия больше одного пользователя, при каждом новом подключении, создается новый объект класса Worker, который непосредственно отвечает за логику взаимодействия с пользователем, на стороне сервера. А также данный объект worker заносится в QList<Workers *> clients:

void ChatServer::***incomingConnection***(qintptr handle){

Worker *worker = new Worker(this);

….

clients.append(worker);

…. }

И чтобы отправить сообщение для всех пользователей, вызывается функция:

```
void ChatServer::Broadcast(const QJsonObject jsonObj, Worker *exclude){

    for(Worker *worker : clients){
        if(worker == exclude){
            continue;
        }

        SendJson(worker, jsonObj);
    }
}
```

```
void ChatServer::SendJson(Worker *receiver, const QJsonObject jsonObj){
    QByteArray dataArray = QJsonDocument(jsonObj).toJson();

    receiver->SendJson(jsonObj);
}
```

Непосредственная отправка:

```
void Worker::SendJson(const QJsonObject jsonObj){

    QByteArray dataArray = QJsonDocument(jsonObj).toJson();

    emit logMessage("Sending to: " + UserName() + dataArray);

    QDataStream socketStream(serverSocket);
    socketStream << dataArray;

}
```

2.6 Реализация сервера

См. приложение А. Рисунки 1, 2

2.7 Реализация клиента

См. приложение А. Рисунок 3

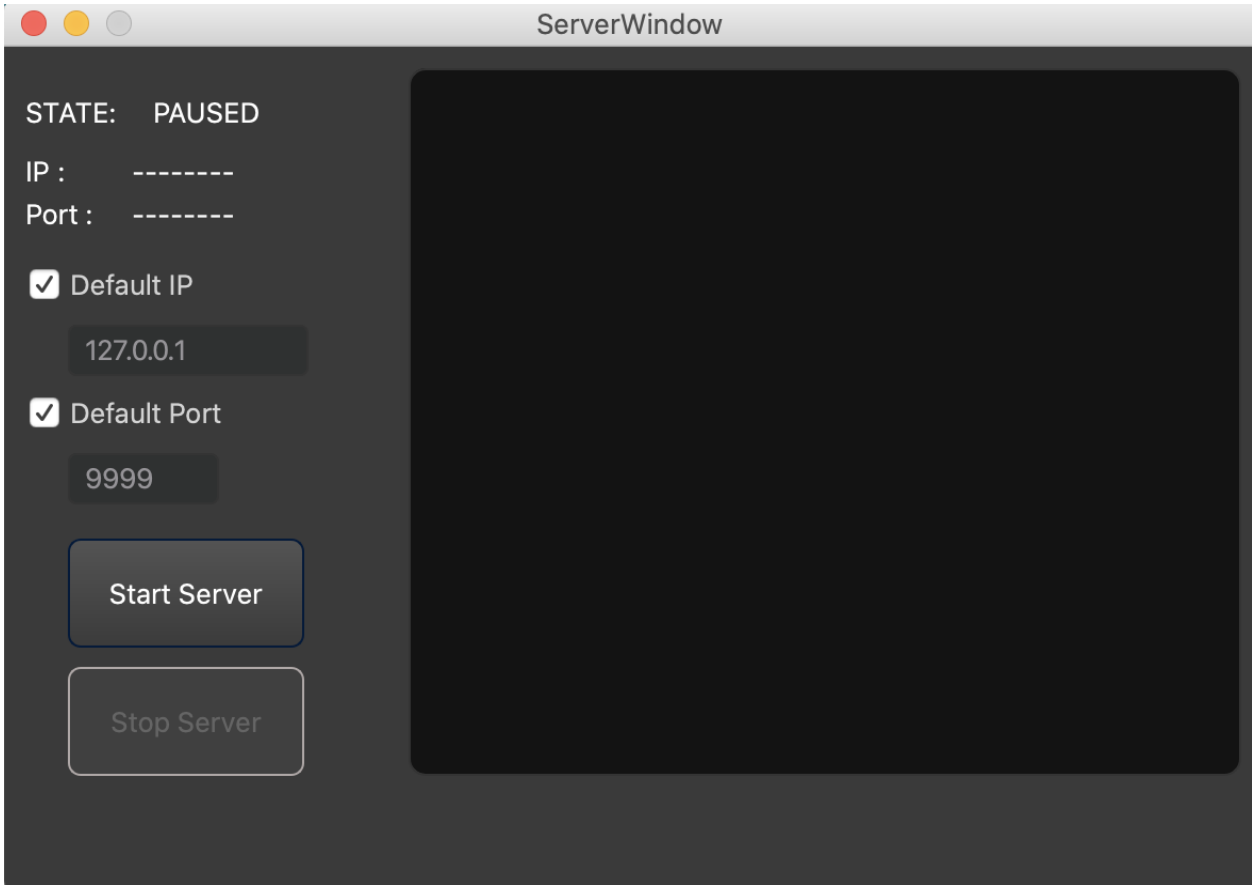
Весь исходный код проекта можно найти и просмотреть на платформе GitHub: https://github.com/kremenevskiy/Messenger

3. Методика работы с полученным приложением

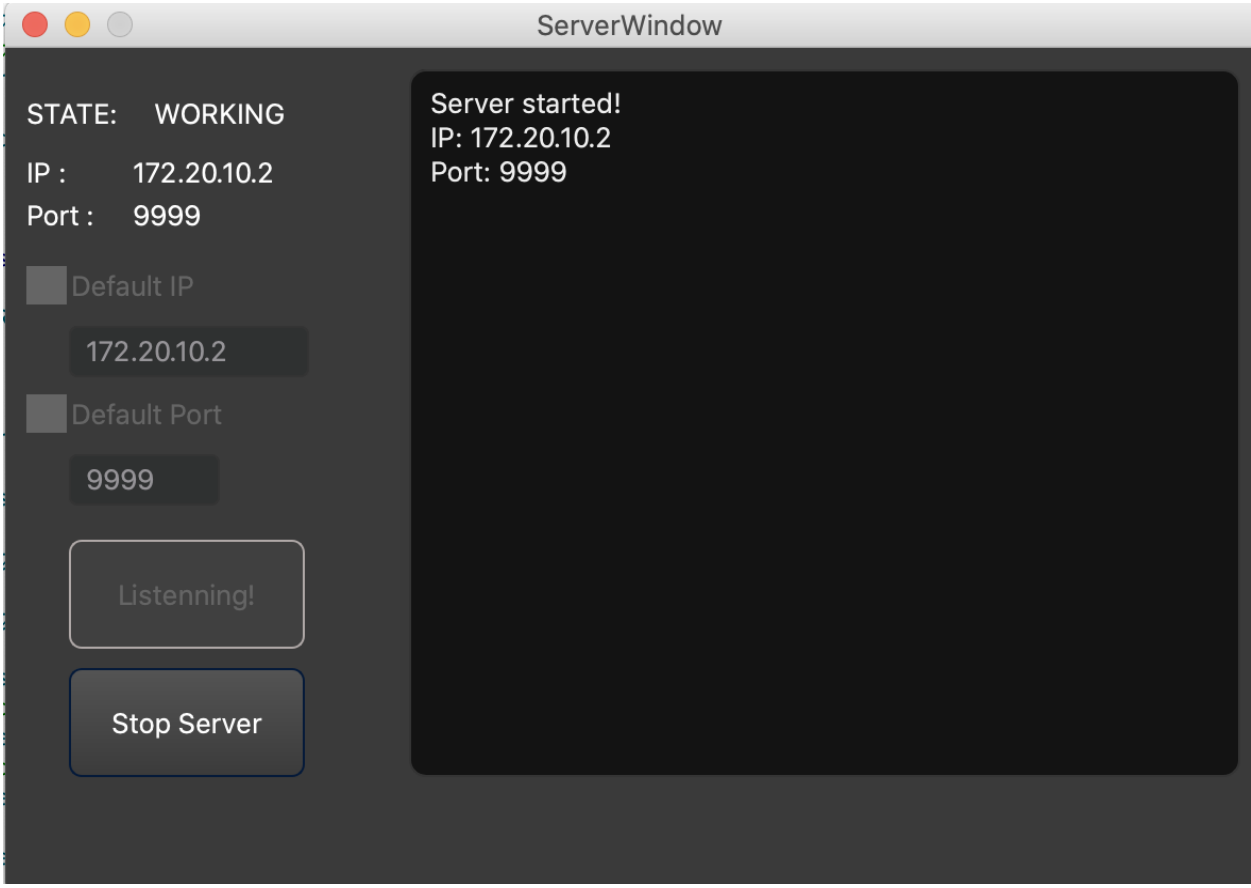
В первую очередь необходимо развернуть сервер. Для этого желательно использовать устройство, которое имеет белое айпи(см. пункт 1.7), это необходимо для того, чтобы любой пользователь, с любой точки мира, у которого есть клиент данного приложения, смог подключиться к серверу. Если же у вас нет глобального айпи, рекомендовано объединиться с нужными пользователями в одну локальную сеть (см. пункт 1.7).

Если же просто необходимо протестировать приложение, можно просто создать сервер и несколько клиентов на одном устройстве.

3.1 Запуск сервера



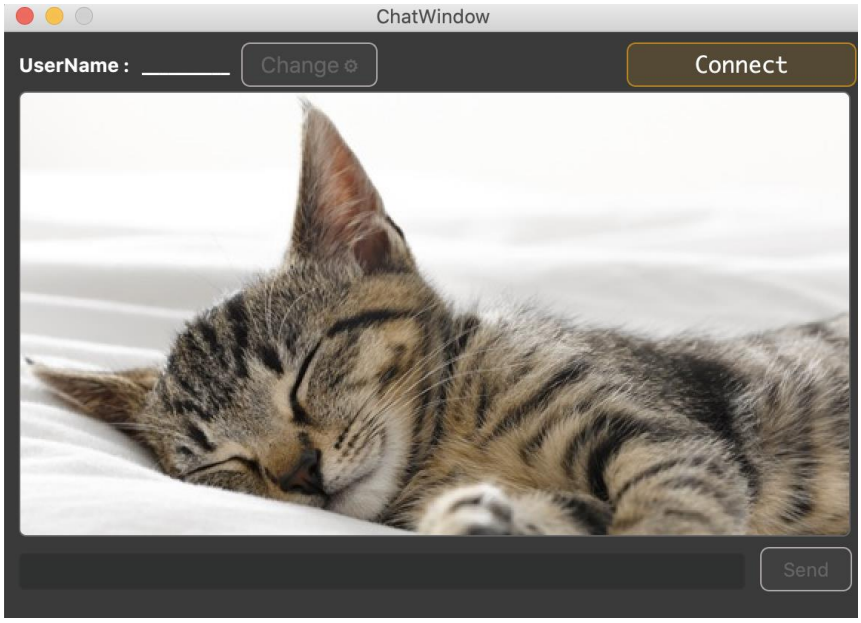
При необходимости, можно указать IP и Port, на которых необходимо развернуть сервер. Далее нужно нажать кнопку **Start Server**



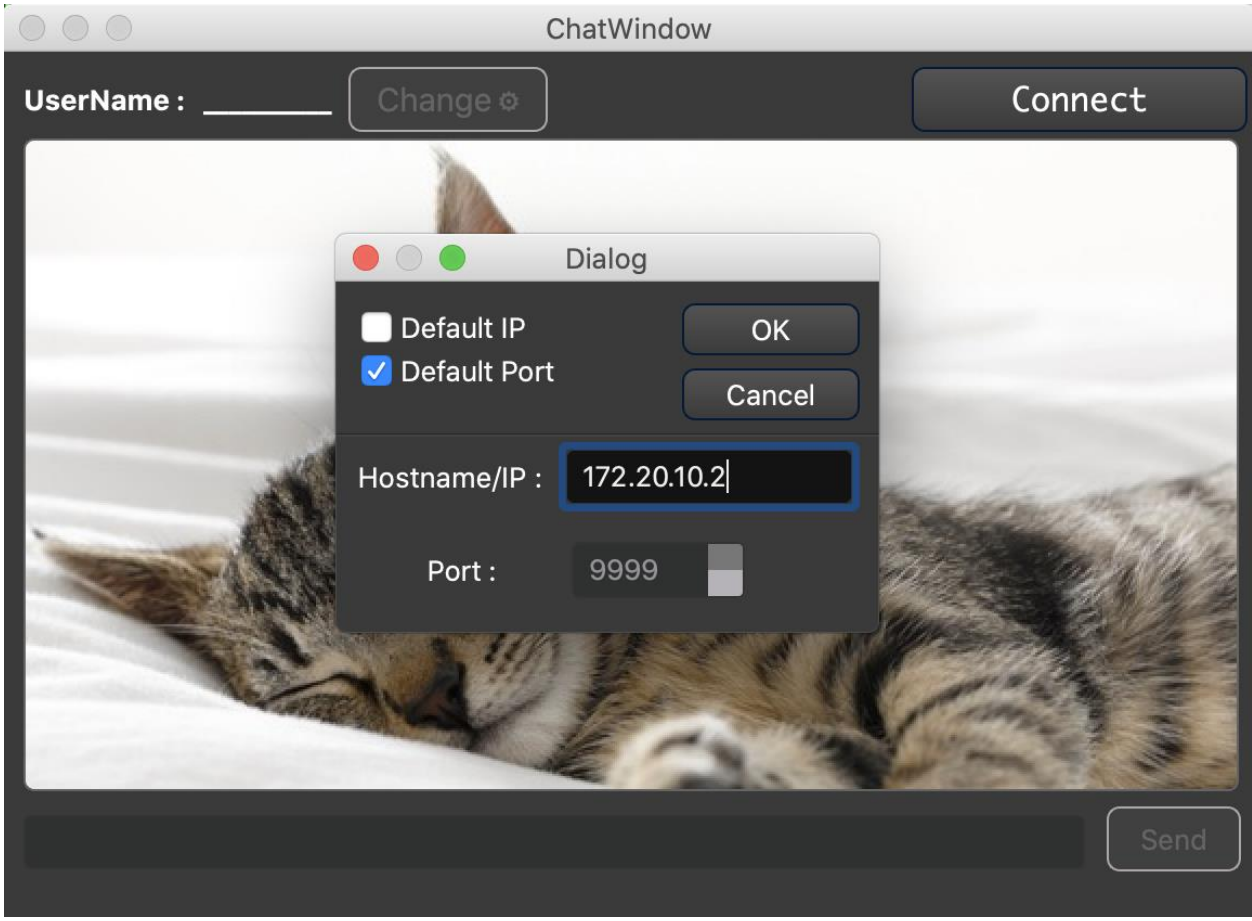
Как видно в логе: сервер успешно запустился, теперь можно подключиться к нему с помощью клиента

2.4 Подключение клиента к серверу

1. Запускаем клиент приложение

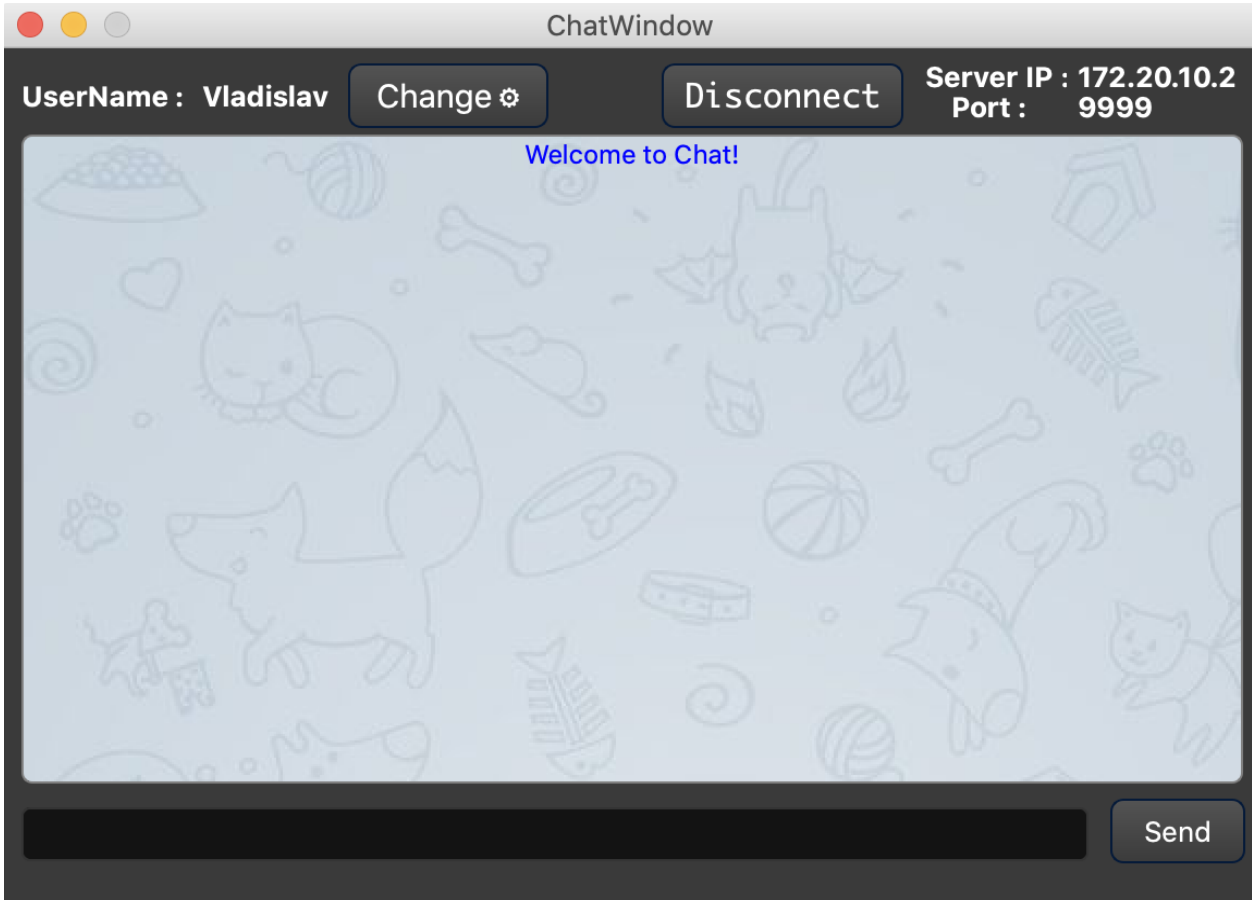


2. Выбираем сервер и порт, к которому хотим подключиться



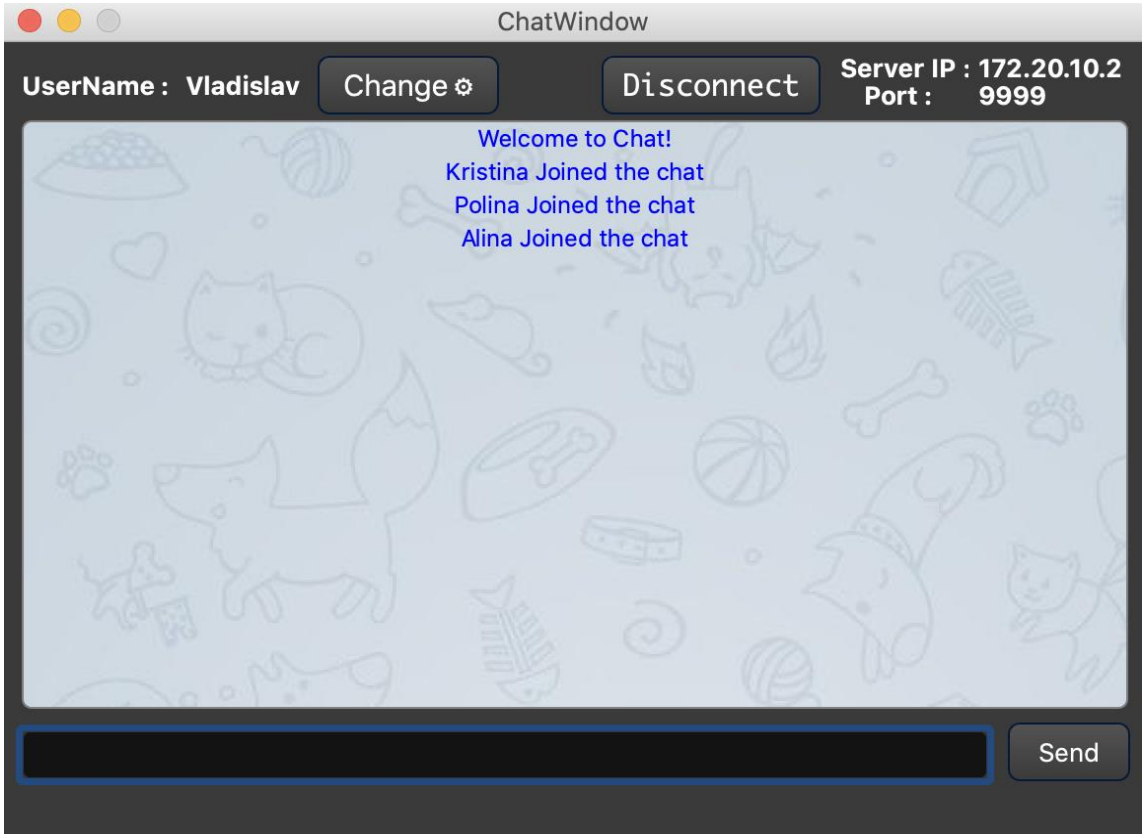
3. Вводим имя и подключаемся.

4. Далее в приложении меняется заставка, сразу же высвечивается имя пользователя, которое видно другим клиентам, а также IP и Port сервера, к которому было произведено подключение. Чат запущен! Осталось дождаться знакомых и начинать переписываться!

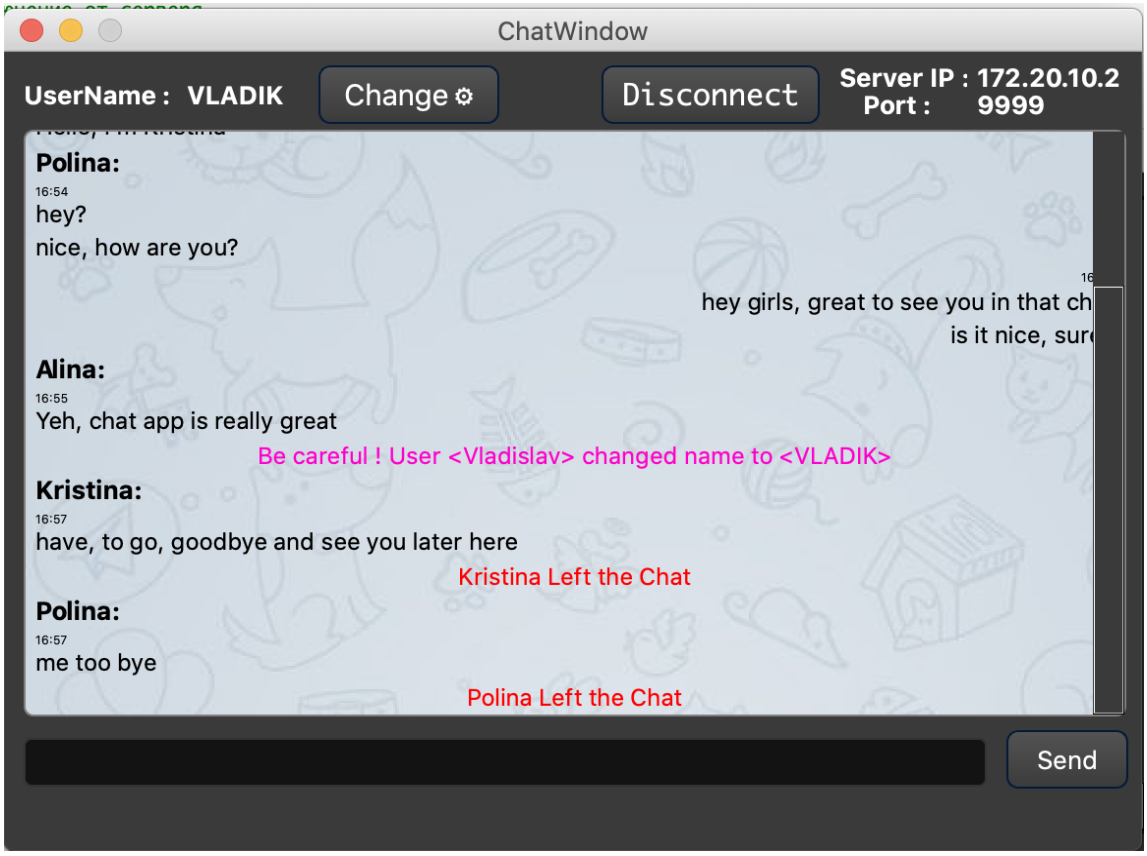


2.5 Взаимодействие внутри чата

1. Сразу же будет видно, если кто-то присоединился к чату, а также можно будет увидеть его имя пользователя на данном сервере

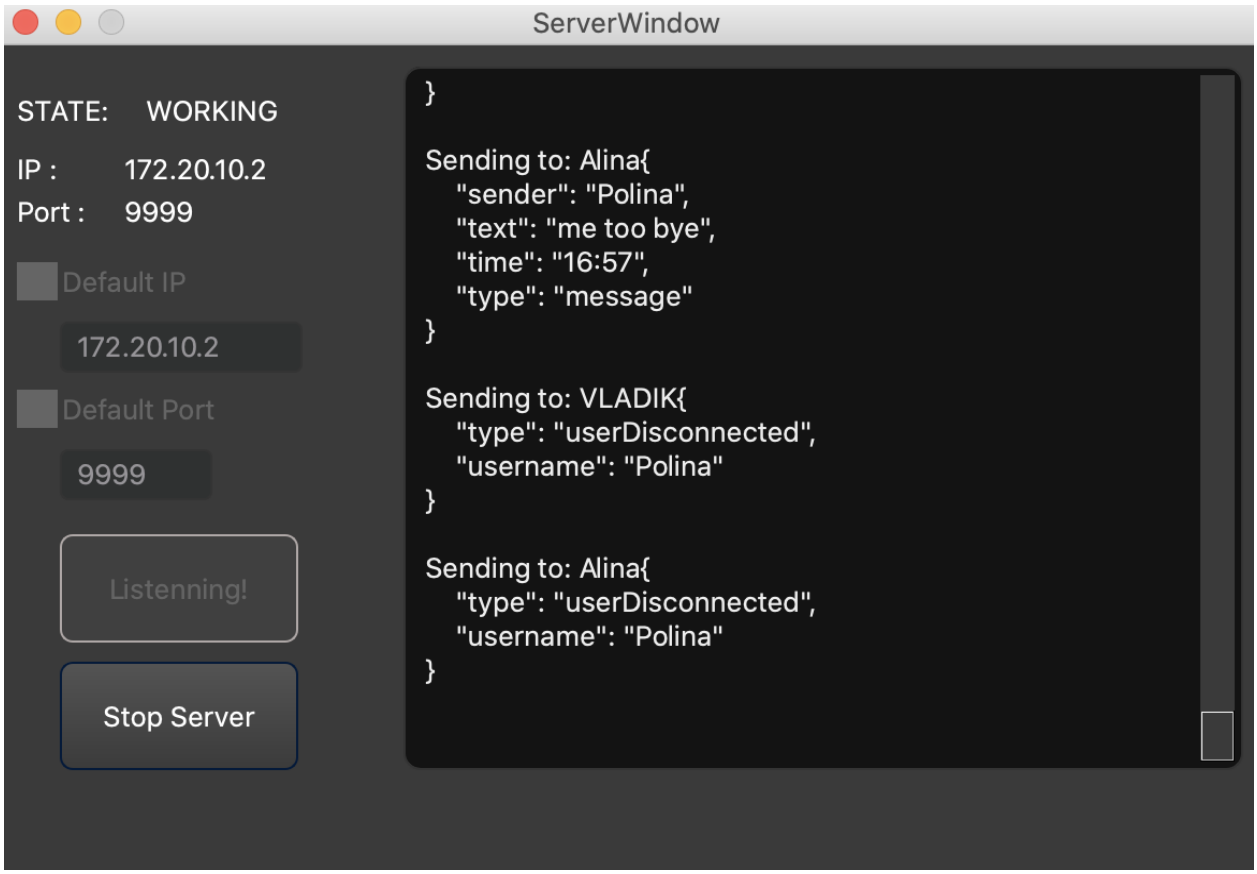


2. Как только подключились новые пользователи, можно начинать переписываться. Также в чате предусмотрена кнопка смены никнейма (Change) и кнопка покинуть чат (Disconnect), если возникнет необходимость.

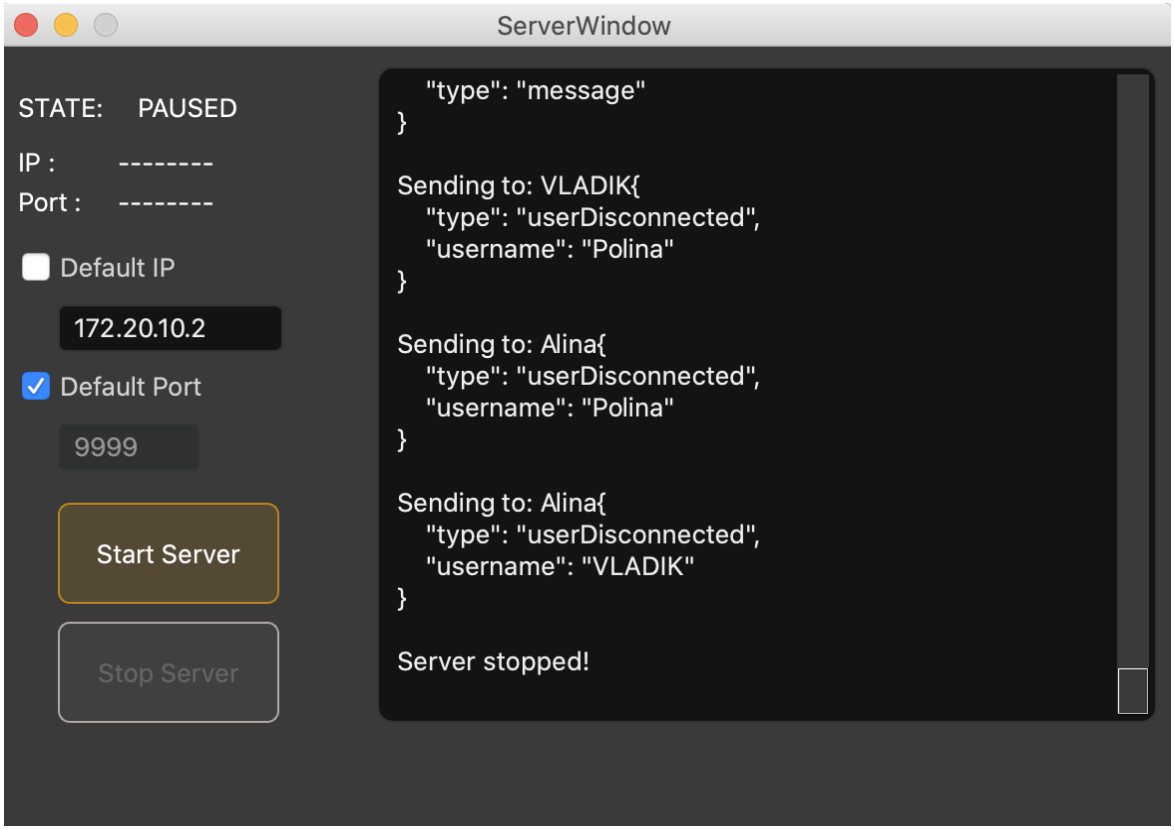


2.6 Администрирование сервера

Если возникнет необходимость, у создателя сервера будет способ проверить всю историю сообщений, а также любых действий, которые были произведены пользователями (смена имени, выход из чата, все сообщения) Для этого предусмотрен специальный лог, в котором отображается вся информация которую обрабатывает сервер.



Также есть возможность прекратить сеанс и остановить сервер, нажав на кнопку Stop Server, в таком случае все пользователи будут отключены от сервера и обязательно придет об этом уведомление.


ServerWindow
STATE: PAUSED
IP : --------
Port : --------
Default IP
172.20.10.2
Default Port
9999
Start Server
Stop Server
"type": "message"
}
Sending to: VLADIK{
"type": "userDisconnected",
"username": "Polina"
}
Sending to: Alina{
"type": "userDisconnected",
"username": "Polina"
}
Sending to: Alina{
"type": "userDisconnected",
"username": "VLADIK"
}
Server stopped!

Заключение

В результате выполнения курсовой работы было разработан мессенджер, который имеет весь базовый, а также дополнительный функционал. В ходе разработки данного программного средства были получены знания по программированию пользовательских интерфейсов в **Qt** — кроссплатформенная библиотека разработки GUI на C++.

Также были получены новые знания по устройству мессенджеров и работы общей **модели взаимодействия клиент-сервер**, которая позволяет разделять функционал и вычислительную нагрузку между клиентскими приложениями (заказчиками услуг) и серверными приложениями (поставщиками услуг).

В ходе работы были изучено много информации про взаимодействие пользователей в интернете, про работу и взаимодействие сокетов, а также TCP сервера. Были получены новые знания о стеке протоколов TCP/IP, на котором базируется большая часть интернета. В добавок ко всему ознакомлены с работой и передачей данных в json формате.

Анализируя приложение можно прийти к следующим выводам:

Qt Crator – отличный инструмент для разработки кросплатформенных приложений. На нём можно быстро и эффективно создавать интерфейсы приложений и получать при этом почти нативную производительность. Хороший подтверждающий пример - Desktop версия одного из самых популярных мессенджеров сегодня, а именно Telegram.

Разработанное программное средство представляет собой законченный продукт, готовый к использованию. Однако при желании функционал программы можно расширить, добавив новые функции, например сохранение и хранение данных переписки в базе данных, а также вход в приложение используя логин и пароль.

Список используемых источников

1. Wikipedia [эдекронный ресурс]: https://ru.wikipedia.org
2. Qt Documentation [эдекронный ресурс]: https://doc.qt.io
3. Youtube [эдекронный ресурс]: https://www.youtube.com

Приложение А. Реализация клиента и сервера

```
// Основной класс управления сервером
class ChatServer : public QTcpServer
{
    Q_OBJECT

public:
    explicit ChatServer(QObject *parent = nullptr);

    // остановка сервера
    void StopServer();

signals:
    // сигнал при отправке сообщения в лог
    void logMessage(const QString &msg);

public slots:
    // набор действий если пользователь отключился
    void UserDisconnected(Worker *sender);

private slots:
    // Если к нам пришло сообещение в формате josn, обрабатыаем его
    void JsonReceived(Worker *sender, const QJsonObject jsonObj);
    // Трансляция нашего jsonObject на всех пользователей
    void Broadcast(const QJsonObject jsonObj, Worker *exclude);

protected:
    // Обработка входящих подключений
    void incomingConnection(qintptr handle) override;

private:
    // Если пришел json от залогиненного пользователя
    void JsonFromLoggedIn(Worker *sender, const QJsonObject jsonObj);
    // Если пришел   json от нового пользователя
    void JsonFromLoggedOut(Worker *sender, const QJsonObject jsonObj);
    // Передаем jsonObject получателю
    void SendJson(Worker *receiver, const QJsonObject jsonObj);

    // Храним список подключенных пользователей
    QList <Worker *> clients;
    //QThreadPool *threadPool;

};
```

Рис. 1. Реализация основного класса ChatServer, который отвечает за работу сервера

```
// Класс на стороне сервера для взаимодейсвтия с пользователем
class Worker : public QObject
{
    Q_OBJECT
public:
    explicit Worker(QObject *parent = nullptr);

    // устанавливаем дискриптор для клиента
    bool setSocketDescriptor(int handle);
    // Функции для установления имени клиента
    void SetUserName(QString userName);
    // Получение имени
    QString UserName();
    // Отправка сообщения на клиентский сокет
    void SendJson(const QJsonObject jsonObj);


signals:
    // Для отправки сообщения в лог
    void logMessage(const QString &msg);
    // При отключении пользователя
    void disconnectedFromServer();
    // Транслируем сообщение в класс ChatServer, чтобы там его обработать
    void jsonReceived(Worker *sender, const QJsonObject jsonObj);

public slots:
    // Закрваем соединение на сокете
    void DisconnectFromServer();
    // Если у нас имеются новые данные, считываем их, проверяем на json
    void onReadyRead();

private:

    QString userName;
    // Храним сокет
    QTcpSocket *serverSocket;
    int socketDescriptor;

};
```

Рис. 2. Реализация класса Worker, который отвечает за взаимодействие с клиентом на стороне сервера

```
class ChatClient : public QObject
{
    Q_OBJECT
public:
    explicit ChatClient(QObject *parent = nullptr);
    // подключение к серверу
    void connectToServer(const QHostAddress &hostAddress, quint16 port);
    // Отключение от сервера
    void disconnectFromServer();
    // Отпраялем сообщение серверу, чтобы проверить уникальность имени
    void login(const QString &userName);
    // Отсылем на сервер сообщение с запросом поменять ник
    void changeName(const QString &oldName, const QString &newName);
    // Отправка сообщения на серверную часть
    void sendMessage(const QString &text, const QString &time);
    // Если полученные данные являются josn + обработка их
    void jsonReceived(const QJsonObject &docObj);

    QString userName;

public slots:
    // Есои на сокете есть данныые, которые возможно прочитать
    void onReadyRead();

signals:
    // Если мы успе
    void connectedToHost();
    // Если соедение с серверной частью потеряно
    void disconnected();
    // Если сервер разрешил подлючиться к серверу
    void loggedIn();
    // Если возникла ошибка при подлючении к серверу
    void loginError(const QString &reason);
    // Если мы получили в json сообещние, которое успешно распарсили
    void messageReceived(QString sender, QString message, QString time);
    // Уведомление, если какой-то пользователь подключился
    void userJoined(const QString userName);
    // Уведомление, если какой-то пользователь отключился
    void userLeft(const QString userName);
    // Если какой-то пользователь сменил ник уведомление
    void userChangedName(const QString &oldName, const QString &newName);
    // Если сервер дал ответ о возможности смены ника
    void changedName(const QString &newName);
    // Если сервер прислал ошибку
    void errorOccured(const QString &reason);


private:
    QTcpSocket *clientSocket;
    bool connectedOnce;
};
```

Рис. 3. Реализация класса ChatClient – главный класс клиента