Министерство сельского хозяйства Российской Федерации

ФГБОУ ВО

«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.Т. ТРУБИЛИНА»

Кафедра системного анализа и обработки информации

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине: Объектно ориентированное программирование

на тему: «Сетевое приложение - Чат»

выполнил студент группы БИ1602 Белусов Юрий Эдуардович

Допущен к защите

Руководитель проекта Иванова Елена Александровна

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, расшифровка подписи)

Защищен\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

Члены комиссии

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата, расшифровка подписи)

Краснодар

2017 г

ФГБОУ ВО

«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.Т. ТРУБИЛИНА»

Кафедра системного анализа и обработки информации

**УТВЕРЖДАЮ:**

**Зав. кафедрой**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ЗАДАНИЕ**

**на курсовой проект**

Студенту: Белоусову Юрию Эдуардовичу

группы БИ1602 2 курса

Факультета Прикладной информатики

специальности Бизнес-информатика

(Ф.И.О., шифр)

Тема проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание задания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Объем работы:

а) пояснительная записка к проекту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_листа формата А4

б) графическая часть\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_лист формата А4

Рекомендуемая литература: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Срок выполнения проекта: с “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_ по “\_\_\_”\_\_\_\_\_\_\_\_20\_\_ г.

Срок защиты: “\_\_\_”\_\_\_\_\_\_\_\_20\_\_ г.

Дата выдачи задания: “\_\_\_”\_\_\_\_\_\_\_\_20\_\_ г.

Дата сдачи проекта на кафедру: “\_\_\_”\_\_\_\_\_\_\_20\_\_ г.

Руководительпроекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

(подпись, Ф.И.О., звание, степень)

Задание принял студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Краснодар

2017 г.

**РЕФЕРАТ**

35 стр. , 15 рис.

ЧАТ, ИНТЕРЕНТ, СЕТЬ, КЛИНЕТ, СЕРВЕР, ПРОТОКОЛЫ, АВТОРИЗАЦИЯ, СОКЕТЫ, TCP/IP, C#, C++, SFML, Visual Studio

Целью работы является разработка сетевого приложения «Чат» на арихитектуре клиент-сервер, с помощью интерфесов прикладного уровня сокетов, в среде Visual Studio, на языках программирования С++ и С#.

Объект исследования – создание работающего сервера на языке С++, умеющего считывать пакеты поступающие на доступный указанный порт с помощью сокетов и отправка данных остальным клиентам подключенных к серверу. Реализация десктопного приложения под клиент разработана на С#, с возможность авторизации на сервере с помощью хешерованных токенов.

Предмет исследования – объектно-ориентированные средства языков программирования и работа c современной сетевой модель TCP/IP. В частности с интрефесом прикладного уровня.

Также у сервера предусмутренна возможность хранения и логгирования данных от пользователей. Чтобы клинету подключиться к серверу, ему нужно пройти авторизацию. У каждого пользователя есть свой список друзей и личных сообщений.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc697801458)

[1 ПОСТАНОВКА ЗАДАЧИ 7](#_Toc2109067824)

[1.1 Cпецификация задачи 7](#_Toc584116884)

[1.2 Выбор средства программирования 7](#_Toc670557768)

[1.3 Требуемое оборудование и программное обеспечение 9](#_Toc1982686185)

[2 СВЕДЕНИЯ ИЗ ТЕОРИИ 10](#_Toc618076307)

[3 ОПИСАНИЕ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ 14](#_Toc1321594863)

[4 ОПИСАНИЕ ПРОГРАММЫ 17](#_Toc2079266927)

[1.1 Описание разработанных классов 17](#_Toc2131358612)

[1.2 Поля и методы разработанных классов 19](#_Toc499890246)

[1.3 Обработчики событий приложения 20](#_Toc1046520992)

[1.4 Описание прочих переменных и функций программы 20](#_Toc90074310)

[5 РАЗРАБОТКА ИНТЕРФЕЙСА 22](#_Toc1633913324)

[6 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ 23](#_Toc710035079)

[7 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЮ 24](#_Toc1805105328)

[ЗАКЛЮЧЕНИЕ 26](#_Toc1097137331)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc826336409)

[ПРИЛОЖЕНИЕ 28](#_Toc76713412)

# **ВВЕДЕНИЕ**

Интерент - всемирная система объединённые компьютерных сетей для хранения и передачи информации. Часто упоминается как Всемирная сеть и Глобальная сеть, а также просто Сеть. Построена на базе стека протоколов TC/IP.

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

Сокет - название программного интерфейса для обеспечения обмена данными между процессами. Сокет - абстрактный объект, представляющий конечную точку соединения.

Целью данной работы является разработка сетевого клиент - сервер приложения с возможностью авторизации, передачи данных, и хранением этих данных на стороне сервера..

Задачи данного проекта:

* Написать сервер на С++
* Написать клиент на С#
* Добавить возможность авторизации
* Реализовать передачу данных через сокеты
* Сделать систему социализации;
* Придумать свой внутренний протокол
* Сохранить масштабируемость приложения;

1. **ПОСТАНОВКА ЗАДАЧИ**
   1. **Спецификация задачи**

Основной задачей этого курсового проекта является разработка сетевого интернет приложения для обмена данными между пользователями, построенного на архитектуре клиент-сервер. Дополнительными задачами являются реализация авторизации на сервере, возможность хранить на нём данные, для передачи клиенту.

Передача информации будет происходить на программном уровне с помощью интерфейса сокетов. Процесс обмена будет происходить по принципу запрос – ответ. Для распознавания и обработки, запросов и ответов, на сервер или клиент. Мы будет использовать собственный принцип или протокол передачи пакетов через сокеты.

На стороне сервера информация будет логироваться и храниться в отдельных файлах, для обеспечения авторизации. Система авторизации будет работать по принципу хеширования логина и пароля, для уникального ключа каждого пользователя.

* 1. **Выбор средства программирования**

В приложение, клиент-сервер, будет разработан, в среде Visual Studio, на языках программирования С++ и С#. Клиент будет реализован как консольное приложение на С++, а клиентская часть будет реализована на С#, как оконное приложение с графическим интерфейсом.

Microsoft Visual Studio - линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

C++ - компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником языком C, наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C# - объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

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

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

* 1. **Требуемое оборудование и программное обеспечение**

Для полноценной работы программы на машине должна быть установлена операционная система Windows, с программным обеспечением Microsoft Visual, и платформа. .NET Framework.

Windows — семейство проприетарных операционных систем (OC) корпорации Microsoft, ориентированных на применение графического интерфейса при управлении. Изначально Windows была всего лишь графической надстройкой-программой для операционной системы 80-х и 90-х годов MS-DOS. По состоянию на август 2014 года под управлением операционных систем семейства Windows по данным ресурса Net Applications работает около 89% персональных компьютеров. Windows работает на платформах x86, x86-64, IA-64 и ARM. Существовали также версии для DEC Alpha, MIPS, PowerPC и SPARC.

Microsoft Visual C++ (MSVC) — интегрированная среда разработки приложений на языке C++, разработанная корпорацией Microsoft и поставляемая либо как часть комплекта Microsoft Visual Studio, либо отдельно в виде бесплатного функционально ограниченного комплекта Visual C++ Express Edition. Сменила интегрированную среду разработки Microsoft QuickC.

.NET Framework — программная платформа, выпущенная компанией Microsoft в 2002 году. Основой платформы является общеязыковая среда исполнения Common Language Runtime (CLR), которая подходит для разных языков программирования. Функциональные возможности CLR доступны в любых языках программирования, использующих эту среду.

1. **СВЕДЕНИЯ ИЗ ТЕОРИИ**

**Чат** - средство обмена сообщениями по компьютерной сети в режиме реального времени, а также программное обеспечение, позволяющее организовывать такое общение.

Характерной особенностью является коммуникация именно в реальном времени или близкая к этому, что отличает чат от форумов и других «медленных» средств. То есть, если на форуме можно написать вопрос и ждать, пока кто-нибудь посчитает нужным на него ответить (в то же время можно получить и несколько ответов сразу от разных пользователей), то в чате общение происходит только с теми, кто присутствует в нём в настоящий момент, а результаты обмена сообщениями могут и не сохраняться. В последнее время чаты значительно расширили свою функциональность за счёт красивого радующего глаз дизайна, а также большого числа улучшений.

Например, появились возможности помещать одного или нескольких пользователей в игнор, сообщения которых после этого перестают быть видимыми тому, кто поместил их в игнор, при том для данной операции необязательно быть модератором или администратором чата. Такое бывает необходимо, когда пользователь не нарушает

Правила чата, но в то же время неприятен, либо пользователь по какой-то причине не банится, то есть не наказывается Администрацией чата за своё поведение. Также есть комнаты привата, где можно вести беседу с другим пользователем, при этом модерация не видит и не модерирует их общение. В некоторые чатах стало появляться несколько комнат для общения с разными Правилами.

Под словом чат обычно понимается групповое общение, хотя к ним можно отнести и обмен текстом «один на один» посредством программ мгновенного обмена сообщениями, например, XMPP, ICQ или даже SMS.

**Протокол передачи данных** - набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии программного обеспечения разнесённой в пространстве аппаратуры, соединённой тем или иным интерфейсом.

Стандартизированный протокол передачи данных также позволяет разрабатывать интерфейсы (уже на физическом уровне), не привязанные к конкретной аппаратной платформе и производителю (например, USB, Bluetooth).

Сигнальный протокол используется для управления соединением — например, установки, переадресации, разрыва связи. Примеры протоколов: RTSP, SIP. Для передачи данных используются такие протоколы как RTP.

Сетево́й протоко́л — набор правил и действий (очерёдности действий), позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть устройствами.

Разные протоколы зачастую описывают лишь разные стороны одного типа связи. Названия «протокол» и «стек протоколов» также указывают на программное обеспечение, которым реализуется протокол.

Новые протоколы для Интернета определяются IETF, а прочие протоколы — IEEE или ISO. ITU-T занимается телекоммуникационными протоколами и форматами.

Наиболее распространённой системой классификации сетевых протоколов является так называемая модель OSI, в соответствии с которой протоколы делятся на 7 уровней по своему назначению — от физического (формирование и распознавание электрических или других сигналов) до прикладного (интерфейс программирования приложений для передачи информации приложениями).

Сетевые протоколы предписывают правила работы компьютерам, которые подключены к сети. Они строятся по многоуровневому принципу. Протокол некоторого уровня определяет одно из технических правил связи.

В настоящее время для сетевых протоколов используется модель OSI (Open System Interconnection — взаимодействие открытых систем, ВОС).

Модель OSI это 7-уровневая логическая модель работы сети. Модель OSI реализуется группой протоколов и правил связи, организованных в несколько уровней:

* на физическом уровне определяются физические (механические, электрические, оптические) характеристики линий связи;
* на канальном уровне определяются правила использования физического уровня узлами сети;
* сетевой уровень отвечает за адресацию и доставку сообщений;
* транспортный уровень контролирует очередность прохождения компонентов сообщения;
* задач сеансового уровня — координация связи между двумя прикладными программами, работающими на разных рабочих станциях;
* уровнь представления служит для преобразования данных из внутреннего формата компьютера в формат передачи;
* прикладной уровень является пограничным между прикладной программой и другими уровнями — обеспечивает удобный интерфейс связи сетевых программ пользователя.

Схему данной модели представлена на рисунке 1.

Другая модель — стек протоколов TCP/IP — содержит 4 уровня:

* канальный уровень (link layer),
* сетевой уровень (Internet layer),
* транспортный уровень (transport layer),
* прикладной уровень (application layer).

Схему TCP/IP модели представлена на рисунке 2.

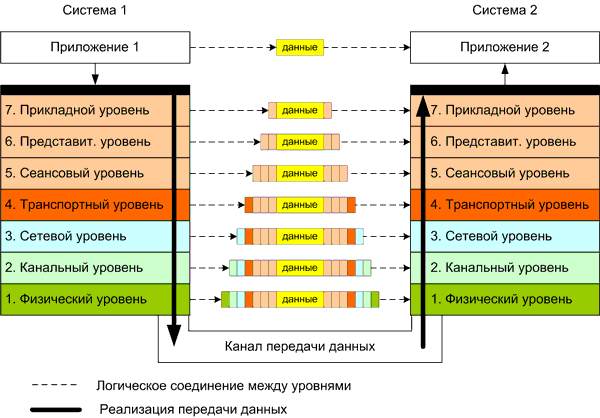


Рисунок 1 – Модель OSI

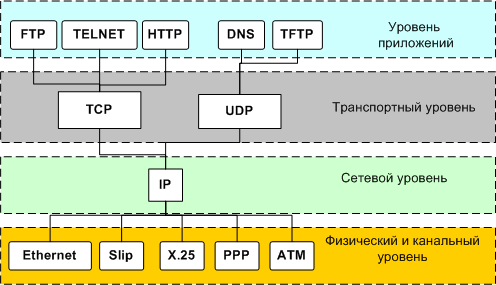


Рисунок 2 – Модель TCP/IP

1. **ОПИСАНИЕ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ**

«Клиент — сервер» (англ. client–server) — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных) или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине). Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Пример работы архитектуры можно увидеть на рисунке 3.

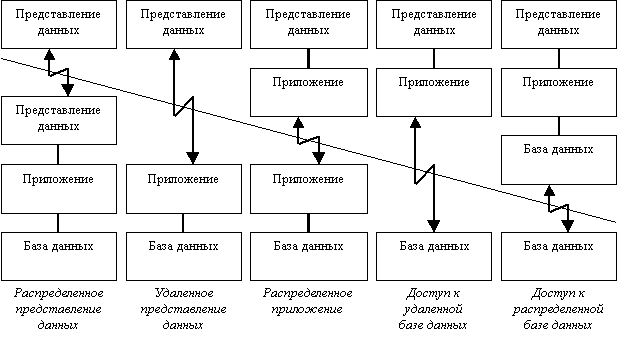


Рисунок 3 - Модели взаимодействия клиент-сервер

Со́кет - название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет - абстрактный объект, представляющий конечную точку соединения.

Следует различать клиентские и серверные сокеты. Клиентские сокеты грубо можно сравнить с конечными аппаратами телефонной сети, а серверные — с коммутаторами. Клиентское приложение (например, браузер) использует только клиентские сокеты, а серверное (например, веб-сервер, которому браузер посылает запросы) — как клиентские, так и серверные сокеты.

Как известно, для взаимодействия между машинами с помощью стека протоколов TCP/IP используются адреса и порты. Первое на текущий момент представляет собой 32-битный адрес (для протокола IPv4, 128-битный для IPv6), наиболее часто его представляют в символьной форме mmm.nnn.ppp.qqq (адрес, разбитый на четыре поля, разделённых точками, по одному байту в поле).

Второе — это номер порта в диапазоне от 0 до 65535 (для протокола TCP). Эта пара и есть сокет («гнездо», соответствующее адресу и порту).

В процессе обмена, как правило, используется два сокета — сокет отправителя и сокет получателя. Например, при обращении к серверу на HTTP-порт сокет будет выглядеть так: 194.106.118.30:80, а ответ будет поступать на mmm.nnn.ppp.qqq: xxxxx.

Каждый процесс может создать «слушающий» сокет (серверный сокет) и привязать его к какому-нибудь порту операционной системы (в UNIX непривилегированные процессы не могут использовать порты меньше 1024).

Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить тайм-аут для операции и т. д.

Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются INET-адрес и UNIX-адрес. Если привязать сокет к UNIX-адресу, то будет создан специальный файл (файл сокета) по заданному пути, через который смогут сообщаться любые локальные процессы путём чтения/записи из него (см. Доменный сокет Unix). Сокеты типа INET доступны из сети и требуют выделения номера порта.

Обычно клиент явно «подсоединяется» к слушателю, после чего любое чтение или запись через его файловый дескриптор будут передавать данные между ним и сервером. Алгоритм работы сокетов отображен на рисунке 4.



Рисунок 4 – Алгоритм работы сокетов

1. **ОПИСАНИЕ ПРОГРАММЫ**
   1. **Описание разработанных классов**

Основная работа подключений и обработка запросов и ответов на стороне сервера реализована в исходном файле Main.cpp. Помимо это на сервере разработанный такие классы как: DTO – слой программы обспечиваюший доступ к данным которые зранятся на сервере, так же у него есть список пользовтелей подключенных к клиентуб и различные методы для обработки и данных; Logger – представляет из себя утилиту, для записи всей деятильности сервера в файл, а так же он логирует все запросы и ответы от клиента; Menu – представляет из себя обычное консольное меню для настройки сервера перед его запуском; Utill – набор статических функций для парсинга и буфферизации данных. Диаграммы классов представлена на рисунке 5.

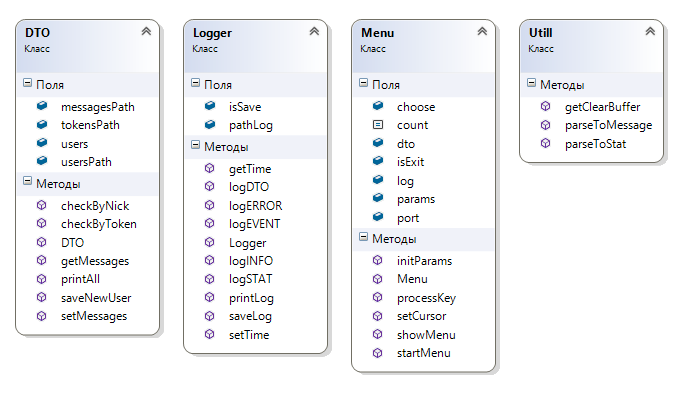
****

Рисунок – 5 UML Диаграмма классов на сервере

Оснавная логика на клиенте состоит из таких классов как: Connector – разработан для обработки ответов и отправки запросов со стороны клиента; User – класс описывающий пользователя; Message – класс описывающий сообщения. Диаграммы классов представлена на рисунке 6.

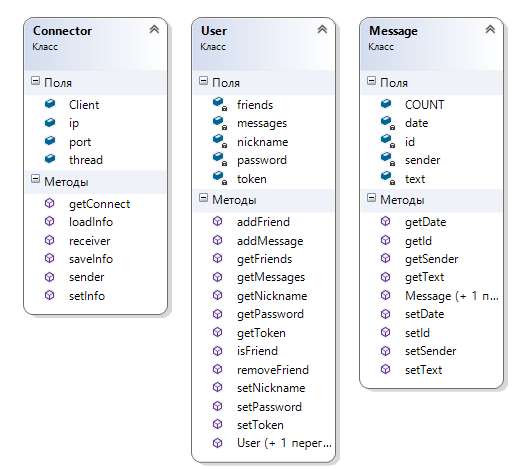
****

Рисунок – 6 UML Диаграмма классов на клиенте

Для оконного интерфейса программы мы сделали 5 классов Form под кажое окно. 1.Основное окно; 2.Окно авторизации, 3.Окно личного пользователя, 4.Окно настроек, 5.Окно дополнительной информации. Диаграммы классов представлена на рисунке 7.

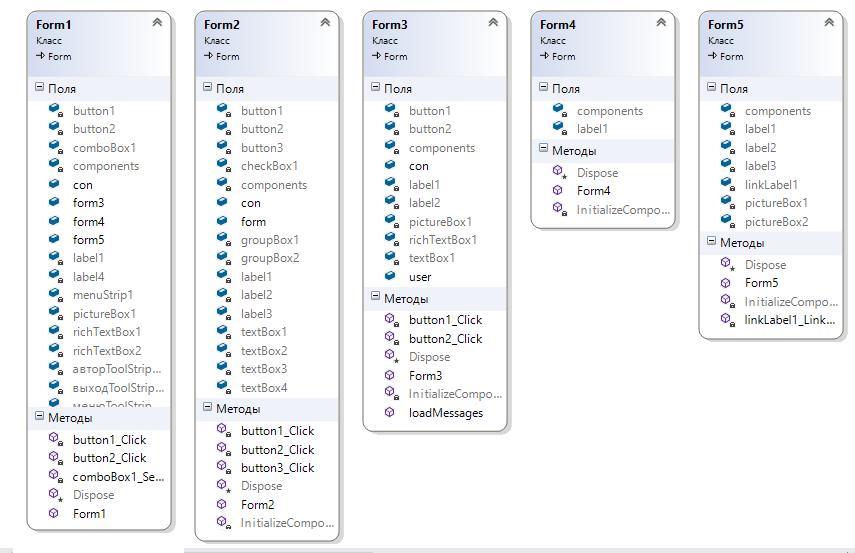
****

Рисунок – 7 UML Диаграмма классов интерфейса

Дополнительными классами на стороне клинета являются: Mapper класс для парсинга обьектов в строковый тип данных и наоборот; Utill – класс содержащий строковые константы описывающие тип передачи данных а так же разные статические функции; Programm – класс запуска программы Диаграммы классов представлена на рисунке 8.

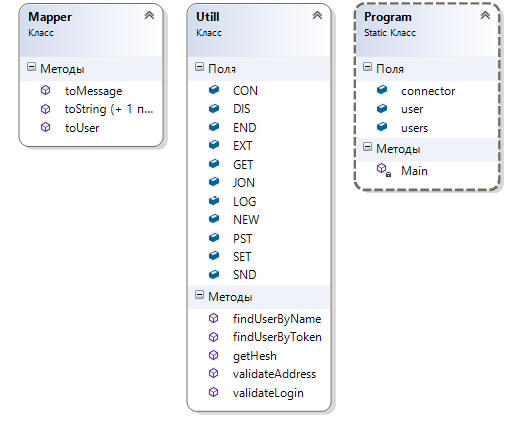
****

Рисунок – 8 UML Диаграмма дополнительных классов

* 1. **Поля и методы разработанных классов  
     Поля и методы класса User**
* private string nickname; - имя пользователя
* private string password; - пароль пользователя
* private string token; - уникальный токен пользователя
* private List<Message> messages = new List<Message>(); - список личных сообщений
* private List<User> friends = new List<User>(); - список друзей пользователя

Для обеспечения инкапсуляции все поля пользователя приватные, для каждого поля есть свой геттер и сеттер.

**Поля и методы класса Message**

* public static int COUNT = 0; - колличество сообщений
* private int id; - номер сообщения
* private string text; - содержание
* private DateTime date; - дата отправки
* private string sender; - токен отправителя

Для обеспечения инкапсуляции все поля пользователя приватные, для каждого поля есть свой геттер и сеттер.

* 1. **Обработчики событий приложения**

При запуске и авторизации пользователя, в программе запускается второй дополнительный поток, который ждет ответы от сервера, пока запущена программа или работает сервер.

* 1. **Описание прочих переменных и функций программы**

Функция getHesh класса Utill хеширует переданный логин и пароль, и возвращет уникальный токен.

public static string getHesh(string log, string pas)

{

string res = "";

string hash = log.GetHashCode().ToString();

for (int i = 0; i < 4; i++)

{

res += hash[i];

if (i % 2 == 0)

if (i < pas.Length)

res += pas[i];

else

res += "@";

else

if (i < log.Length)

res += log[i];

else

res += "@";

}

return res;

}  
Статические поля класс Utill для обозначения типа передачи данных на сервер и обратно

public static string LOG = "LOG";

public static string CON = "CON";

public static string DIS = "DIS";

public static string PST = "PST";

public static string NEW = "NEW";

public static string SND = "SND";

public static string GET = "GET";

public static string SET = "SET";

public static string JON = "JON";

public static string EXT = "EXT";

public static string END = ";;;";

LOG – попытка пройти авторизацию

CON – ответ от сервера о успешной авторизации

DIS - ответ от сервера о провальной авторизации

NEW – создание нового пользователя

SND – отправка сообщения в общий чат

GET – загрузка личных сообщений

SET – установка личных сообщений

JON – подсоединение к чату

EXT – отключение от чата

END – конец сообщение в буфере

1. **РАЗРАБОТКА ИНТЕРФЕЙСА**

Интерфейс на стороне клиента разработан в редакторе Forms в среде разработки Visual Sttudio. Пример можно увидеть на рисунке 9.

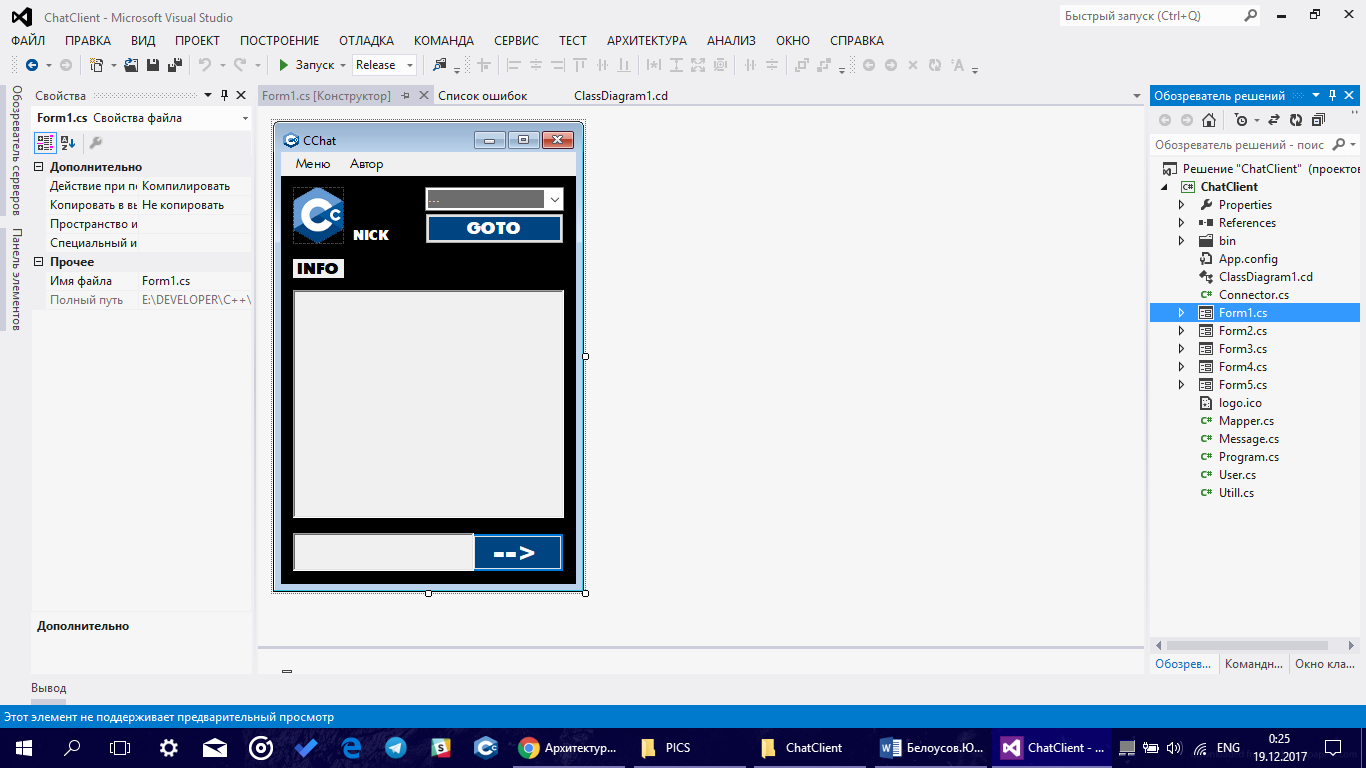


Рисунок 9 – Разработка интрефейса

Для серверной части программы был создан консольный интрефейс с интерактивным меню. Смотрите на рисунок – 10.

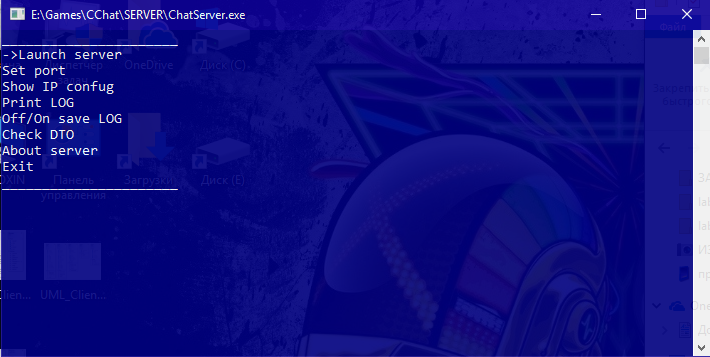


Рисунок 10 – Консольное меню

1. **РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ**

Вот что мы имеем при запуске клиентской и серверной части приложения. Результат на рисунке 11.

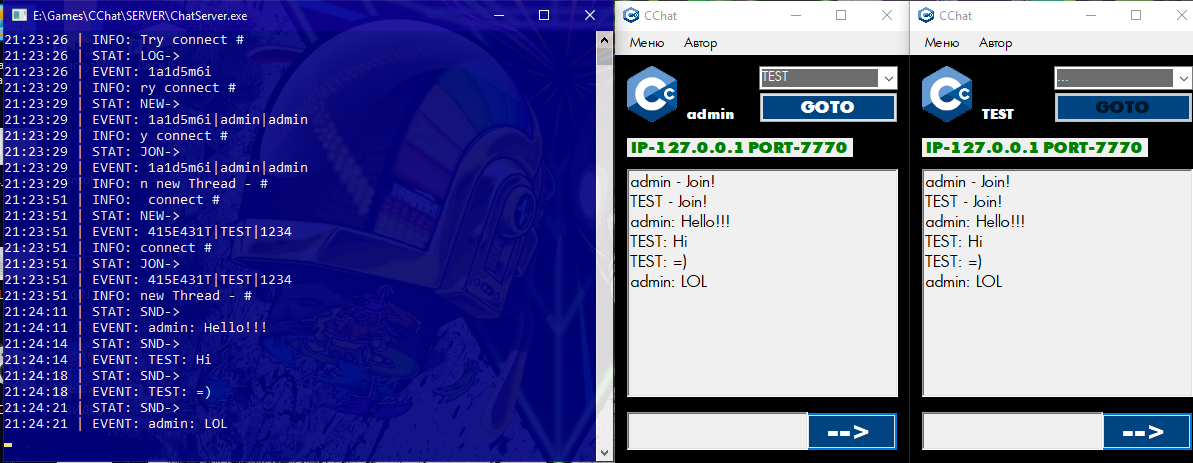


Рисунок 11 – Работа программы

Так же можно увидеть что на стороне сервера логируются различные данные. Внимание на рисунок 12.

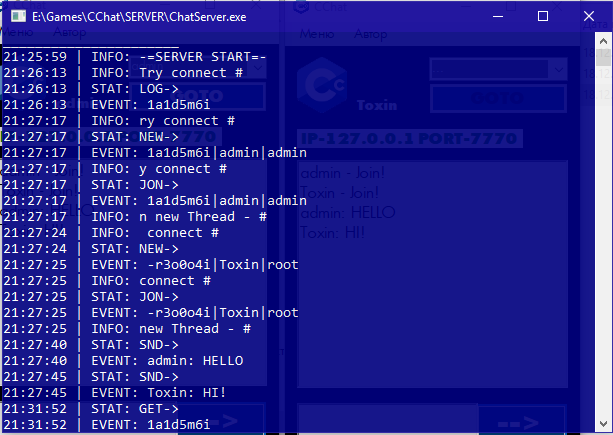


Рисунок 12 – Логирование данных

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЮ**

**Окнов для авторизации в приложение.** Представлено на риснуке 13.

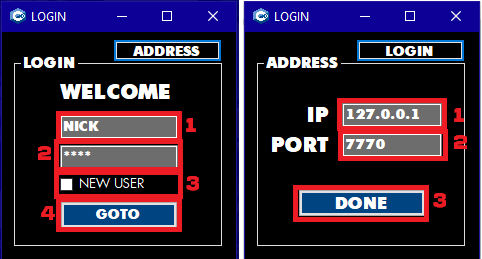
****

Рисунок 13 – Окно авторизации

**Login**:

1. Поле для имени
2. Поле для пароля
3. Установка для нового пользователя
4. Отправка запроса на авторизацию

**Address**:

1. Ввод IP
2. Ввод порта
3. Сохранение данных

**Личное окно пользователя.** Представлено на рисунке 14.



Рисунок 14 – Личное окно пользователя

1. Имя пользователя
2. Аватарка пользователя
3. Кнопка для добавления в друзья
4. Количество сообщений на стене
5. Стена с личными сообщениями
6. Поле для ввода сообщения
7. Поле для отправки сообщения

**Главное окно приложения .** Представлено на рисунке 14.

****

Рисунок 15 – Главное окно

1. Меню окна
2. Аватарка пользователя
3. Список пользователи
4. Переход к стене пользователя
5. Адрес и порт сервера
6. Чат приложения
7. Поле для ввода сообщения
8. Поле для отправки сообщения

**ЗАКЛЮЧЕНИЕ**

Итак, основная идея архитектуры «клиент-сервер» состоит в разделении сетевого приложения на несколько компонентов, каждый из которых реализует специфический набор сервисов. Компоненты такого приложения могут выполняться на разных компьютерах, выполняя серверные и/или клиентские функции. Это позволяет повысить надежность, безопасность и производительность сетевых приложений и сети в целом.

Программа с клиент - серверной архитектурой взаимодействует с пользователе через главное меню программы, где пользователю предлагается войти в систему или зарегистрироваться.

Регистрация - это сохранения данных о пользователе, таких как имя, фамилия, номер группы, логин, пароль втаблице для хранения личных данных о студентах. Вход в систему - ввод логина и пароля.Данные о пользователе хранятся в файле, куда и будут сохраняться..

Были поставлены и выполнены следующие задачи:

* Написать сервер на С++
* Написать клиент на С#
* Добавить возможность авторизации
* Реализовать передачу данных через сокеты
* Сделать систему социализации;
* Придумать свой внутренний протокол
* Сохранить масштабируемость приложения;

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Айвор Хортон. Visual C++ 2010. Полный курс. : Пер. с англ. – М. : Издательский дом «Вильямс», 2012. – 624 с.
2. Литвиненко Н.А. технология программирования на С++. Win32 API-приложения. – СПб. : БХВ-Петербург, 2013. – 288 ил.
3. Сидорина Т.Л. Самоучитель Microsoft Visual Studio C++ и MFC. – СПб.: БХВ-Петербург, 2013. – 848 ил.
4. Глушаков С.В. Программирование на С++ /, Т.В. Дуравкина. – изд. 2-е, доп. и переработ. – М. : АСТ, 2015. – 685 с.
5. Шилдт Г. C++ базовый курс. : Пер. с англ. – М. : Издательский дом «Вильямс», 2012. – 624 с.
6. Павловская Т. А. C/C++. Структурное программирование : Практикум / Т. А. Павловская, Ю.А.Щупак. – СПб. : Питер, 2011. – 240 с.
7. Подбельский В.В. Язык С++: Учебное пособие. – 5-е изд. -М. : Финансы и статистика, 2015. -560 с.
8. Лафоре Р ОБЪЕКТНО - ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ В C++ / – СПб. : Питер, 2016. – 240 с.
9. Программное обеспечение компьютерных сетей: Учебное пособие / О. В. Исаченко. — М.: ИНФРА-М, 2015. — 117 с.

**ПРИЛОЖЕНИЕ**

**ЛИСТИНГ ПРОГРАММЫ**

#pragma comment(lib, "Ws2\_32.lib")

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <iostream>

#include <string>

#include "DTO.h"

#include "Logger.h"

#include "Utill.h"

#include "Connector.h"

#include "Menu.h"

using namespace std;

const string LOG = "LOG";

const string CON = "CON";

const string DIS = "DIS";

const string NEW = "NEW";

const string SND = "SND";

const string GET = "GET";

const string SET = "SET";

const string JON = "JON";

const string EXT = "EXT";

const string END = ";;;";

DTO dto;

Utill util;

Logger logger;

Menu menu(logger, dto);

SOCKET Connect;

SOCKET\* Connections;

SOCKET Listen;

int clientsCount = 0;

void handlerGET(char\* buffer, int ID)

{

string message = Utill::parseToMessage(buffer);

string res = "GET" + message;

logger.logEVENT(message);

res += dto.getMessages(message);

res += ";;;";

const char \*tmp = res.c\_str();

send(Connections[ID], tmp, strlen(tmp), NULL);

}

void handlerSET(char\* buffer, int ID)

{

string message = Utill::parseToMessage(buffer);

logger.logEVENT(message);

dto.setMessages(message);

}

void handlerSND(char\* buffer, int ID)

{

string message = Utill::parseToMessage(buffer);

logger.logEVENT(message);

for (int i = 0; i <= clientsCount; i++)

if (Connections[i] != NULL)

send(Connections[i], buffer, strlen(buffer), NULL);

}

void handlerEXT(char\* buffer, int ID)

{

string message = Utill::parseToMessage(buffer);

logger.logEVENT(message);

Connections[ID] = NULL;

dto.users.remove(message);

}

void handlerJON(char\* buffer, int ID)

{

string message = Utill::parseToMessage(buffer);

logger.logEVENT(message);

for (auto it = dto.users.begin(); it != dto.users.end(); it++)

{

string user = JON + \*it + END;

const char \*tmp = user.c\_str();

send(Connections[ID], tmp, strlen(tmp), NULL);

}

dto.users.push\_back(message);

for (int i = 0; i <= clientsCount; i++)

if (Connections[i] != NULL)

send(Connections[i], buffer, strlen(buffer), NULL);

}

void handlerNEW(char\* buffer, int ID)

{

char\* answer = "";

string message = Utill::parseToMessage(buffer);

string token = message.substr(0, 8);

logger.logEVENT(message);

if (!dto.checkByNick(message) && !dto.checkByToken(token))

{

dto.saveNewUser(token, message);

answer = "CON;;;";

}

else

{

answer = "DIS;;;";

}

send(Connections[ID], answer, strlen(answer), NULL);

}

void handlerLOG(char\* buffer, int ID)

{

char\* answer = "";

string message = Utill::parseToMessage(buffer);

logger.logEVENT(message);

if (dto.checkByToken(message))

answer = "CON;;;";

else

answer = "DIS;;;";

send(Connections[ID], answer, strlen(answer), NULL);

}

void crateConnect(string port)

{

WSAData data;

WORD version = MAKEWORD(2,2);

int res = WSAStartup(version, &data);

if (res != 0) return;

struct addrinfo hints;

struct addrinfo \* result;

Connections = (SOCKET\*) calloc(64, sizeof(SOCKET));

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_flags = AI\_PASSIVE;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

const char\* tmp = port.c\_str();

getaddrinfo(NULL, tmp, &hints, &result);

Listen = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

bind(Listen, result->ai\_addr, result->ai\_addrlen);

listen(Listen, SOMAXCONN);

freeaddrinfo(result);

}

void eventListener(int ID)

{

while(true)

{

char\* buffer = Utill::getClearBuffer();

if (recv(Connections[ID], buffer, 1024, NULL))

{

string message(buffer);

string stat = message.substr(0, 3);

logger.logSTAT(stat + "-> ");

if (stat == SND)

{

handlerSND(buffer, ID);

}

else if (stat == GET)

{

handlerGET(buffer, ID);

}

else if (stat == SET)

{

handlerSET(buffer, ID);

}

else if (stat == EXT)

{

handlerEXT(buffer, ID);

return;

}

}

Sleep(100);

}

}

void startServer()

{

logger.setTime();

logger.logINFO("-=SERVER START=-");

while (true)

{

char\* buffer = Utill::getClearBuffer();

if (Connect = accept(Listen, NULL, NULL))

{

Connections[clientsCount] = Connect;

clientsCount++;

int ID = clientsCount - 1;

logger.logINFO("Try connect # " + ID);

if (recv(Connections[ID], buffer, 1024, NULL))

{

string stat = Utill::parseToStat(buffer);

logger.logSTAT(stat + "-> ");

if (stat == LOG)

{

handlerLOG(buffer, ID);

}

else if (stat == NEW)

{

handlerNEW(buffer, ID);

}

else if (stat == JON)

{

handlerJON(buffer, ID);

logger.logINFO("Run new Thread - #" + ID);

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)eventListener, (LPVOID)(ID), NULL, NULL);

}

}

}

Sleep(100);

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

while(!menu.isExit)

menu.startMenu();

crateConnect(menu.port);

startServer();

return 1;

}

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.IO;

using System.Threading;

namespace ChatClient

{

public partial class Form1 : Form

{

public Connector con;

public Form3 form3;

public Form4 form4;

public Form5 form5;

public Form1()

{

InitializeComponent();

con = Program.connector;

con.getConnect();

con.sender(Utill.JON + Mapper.toString(Program.user));

con.thread = new Thread(delegate() { startReceive(); });

con.thread.Start();

button1.Focus();

button2.Enabled = false;

label1.Text = Program.user.getNickname();

label4.ForeColor = Color.Green;

label4.Text = "IP-" + con.ip.ToString() + " PORT-" + con.port;

}

public void startReceive()

{

while (true)

{

try

{

string message = con.receiver();

string type = message.Substring(0, 3);

message = message.Substring(3);

this.Invoke((MethodInvoker)delegate()

{

if (type == Utill.SND)

{

richTextBox1.AppendText(message + "\n");

richTextBox1.SelectionStart = richTextBox1.Text.Length;

richTextBox1.ScrollToCaret();

}

else if (type == Utill.JON)

{

User newUser = Mapper.toUser(message);

Program.users.Add(newUser);

comboBox1.Items.Add(newUser.getNickname());

richTextBox1.AppendText(newUser.getNickname() + " - Join!" + "\n");

}

else if (type == Utill.GET)

{

string token = message.Substring(0, 8);

message = message.Substring(8);

string[] messages = message.Split('|');

User recv = Utill.findUserByToken(token);

for (int i = 1; i < messages.Length; i++)

{

recv.addMessage(new Message(messages[i], token));

}

if (form3 != null)

form3.loadMessages(recv.getMessages());

}

});

}

catch (Exception ex)

{

Console.WriteLine(ex);

showError();

return;

}

}

}

private void button1\_Click(object sender, EventArgs e)

{

if (richTextBox2.Text != "")

{

richTextBox2.Text = richTextBox2.Text.Trim();

string message = Program.user.getNickname() + ": " + richTextBox2.Text + Utill.END;

con.sender(Utill.SND + message);

richTextBox2.Clear();

}

}

private void button2\_Click(object sender, EventArgs e)

{

string name = comboBox1.SelectedItem.ToString();

if (name != null && name != "")

{

User user = Utill.findUserByName(name);

form3 = new Form3(user);

form3.Show();

}

}

private void pictureBox1\_Click(object sender, EventArgs e)

{

form3 = new Form3(Program.user);

form3.Show();

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

button2.Enabled = true;

}

private void авторToolStripMenuItem\_Click(object sender, EventArgs e)

{

form5 = new Form5();

form5.Show();

}

private void выходToolStripMenuItem\_Click(object sender, EventArgs e)

{

con.sender(Utill.EXT + Mapper.toString(Program.user));

if (con.thread != null)

con.thread.Abort();

Application.Exit();

}

private void настройкиToolStripMenuItem\_Click(object sender, EventArgs e)

{

form4 = new Form4();

form4.Show();

}

private void showError()

{

label4.ForeColor = Color.Red;

label4.Text = "DISCONNECT";

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

выходToolStripMenuItem\_Click(sender, e);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ChatClient

{

public class User

{

private string nickname;

private string password;

private string token;

private List<Message> messages = new List<Message>();

private List<User> friends = new List<User>();

public User() { }

public User(string nickname, string password)

{

this.nickname = nickname;

this.password = password;

this.token = Utill.getHesh(nickname, password);

}

public string getNickname()

{

return this.nickname;

}

public string getPassword()

{

return this.password;

}

public string getToken()

{

return this.token;

}

public List<Message> getMessages()

{

return this.messages;

}

public List<User> getFriends()

{

return this.friends;

}

public void setToken(string token)

{

this.token = token;

}

public void setNickname(string nickname)

{

this.nickname = nickname;

}

public void setPassword(string password)

{

this.password = password;

}

public bool isFriend(User user)

{

return friends.Contains(user);

}

public void addMessage(Message message)

{

foreach (Message itr in messages)

{

if (itr.getText() == message.getText())

{

return;

}

}

messages.Add(message);

}

public void addFriend(User user)

{

if (!isFriend(user) && user != null)

{

friends.Add(user);

}

}

public void removeFriend(User user)

{

if (!isFriend(user) && user != null)

{

friends.Remove(user);

}

}

}

}