МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**КУРСОВОЙ ПРОЕКТ**

по дисциплине: «Компьютерные системы и сети»

на тему: «***SMTP сервер***»

Выполнил: И.А.Воробей

студент группы 10701118

Руководитель: С.В.Белова

Старший преподаватель

Минск 2020

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

по дисциплине: «Компьютерные системы и сети»

на тему: «***SMTP сервер***»

Выполнил: И.А.Воробей

студент группы 10701118

Руководитель: С.В.Белова

Старший преподаватель

Минск 2020

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc40649478)

[1 Постановка задачи 5](#_Toc40649479)

[2 Обзор состояния вопроса 6](#_Toc40649480)

[3 Описание программы 8](#_Toc40649481)

[3.1 Технологии программы 8](#_Toc40649482)

[3.2 Структурная и физическая схема 8](#_Toc40649483)

[3.3 Функционал 9](#_Toc40649484)

[4 Руководство пользователя 10](#_Toc40649485)

[5 Тестирование 19](#_Toc40649486)

[ЗАКЛЮЧНЕНИЕ 20](#_Toc40649487)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 21](#_Toc40649488)

[ПРИЛОЖЕНИЕ А 22](#_Toc40649489)

[ПРИЛОЖЕНИЕ Б 27](#_Toc40649490)

# **ВВЕДЕНИЕ**

Сегодня информация является ключевым компонентом экономической и социальной жизни. Информация приобрела статус товара и сравнялась по значимости для общества с другими материальными ресурсами.

Стремительная информатизация человеческого общества неизбежно влечет за собой внедрение во все сферы человеческой деятельности новых информационных технологий, благодаря которым происходит быстрое распространение информации. Они играют ключевую роль в формировании техноструктуры, в повышении роли образования и активно внедряются во все сферы социально-политической и культурной жизни, включая домашний быт, развлечения и досуг.

Также использование информационных технологий существенно повышает производительность труда персонала, освобождает его от рутинных операций, позволяет выполнять работу быстрее, повышает качество процессов.

Сегодня электронная почта — это такой же важный инструмент связи, как и номер телефона. Электронная почта используется для общения по работе, для регистрации в социальных сетях и мессенджерах, для официальных запросов в госорганы.

Всего в мире [3,7 миллиардов](https://blog.sendsay.ru/email-markietingh-v-tsifrakh-vpiechatliaiushchaia-spornaia-i-liubopytnaia-statistika-za-2018-ghod/) почтовых аккаунтов [на 7,7 миллиардов](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%97%D0%B5%D0%BC%D0%BB%D0%B8) населения Земли. Само собой, не у каждого жителя Земли есть электронная почта. У некоторых пользователей есть по 5 электронных ящиков. Это не считая служебных ящиков для доступа к разным сервисам. В России интернетом пользуется 93 миллиона человек, и почти у каждого есть электронная почта.

Целью курсового проекта является разработка SMTP сервера.

1 Постановка задачи

В рамках данного курсового проекта необходимо реализовать SMTP сервер.

Разрабатываемое приложение должно иметь архитектуру клиент-сервер и обеспечивать многопользовательский режим работы.

Приложение должно включать выполнение следующих функций:

* режим работы с одним клиентом;
* режим работы с множеством клиентов в разных потоках;
* отображение данных в графическом интерфейсе
* отображение всего протокола общения с клиентом;
* логирование протоколов общения.

Клиентская часть должна иметь консольный интерфейс. Серверная часть – графический интерфейс.

Вся информация о пользователях хранится на сервере в базе данных.

Для разработки приложения используется операционная система Windows 10.

Язык программирования C# 7.0.

Среда разработки Visual Studio 2019.

Взаимодействие клиента и сервера осуществляется на основе технологии сокетов.

2 Обзор состояния вопроса

SMTP ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Simple Mail Transfer Protocol — простой протокол передачи почты) — это широко используемый [сетевой протокол](https://ru.wikipedia.org/wiki/IP), предназначенный для передачи [электронной почты](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D1%87%D1%82%D0%B0) в сетях TCP/IP.

SMTP впервые был описан в [RFC 821](https://tools.ietf.org/html/rfc821) (1982 год); последнее обновление в [RFC 5321](https://tools.ietf.org/html/rfc5321) (2008) включает масштабируемое расширение — ESMTP ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA)  Extended SMTP). В настоящее время под «протоколом SMTP» как правило подразумевают и его расширения. В стеке протоколов TCP/IP протокол SMTP находится на прикладном уровне.

В то время, как электронные [почтовые серверы](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%87%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80) и другие агенты пересылки сообщений используют SMTP для отправки и получения почтовых сообщений, работающие на пользовательском уровне клиентские почтовые приложения обычно используют SMTP только для отправки сообщений на почтовый сервер для ретрансляции. Для получения сообщений клиентские приложения обычно используют либо [POP](https://ru.wikipedia.org/wiki/POP3) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Post Office Protocol — протокол почтового отделения), либо [IMAP](https://ru.wikipedia.org/wiki/IMAP) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) Internet Message Access Protocol), либо патентованные системы (такие как Microsoft [Exchange](https://ru.wikipedia.org/wiki/Microsoft_Exchange_Server) и [Lotus Notes](https://ru.wikipedia.org/wiki/IBM_Lotus_Notes)/[Domino](https://ru.wikipedia.org/wiki/Lotus_Domino)) для доступа к учётной записи своего почтового ящика на сервере.

Теоретически протокол SMTP может использовать любой транспортный протокол (TCP, UDP и др.). За SMTP стандартами закреплено два номера порта, а именно порт 25 – для передачи почты между почтовыми серверами и порт 587 – для передачи почтового сообщения от клиенту к серверу. Однако на практике почти всегда используется протокjл TCP и порт 25.

Электронное письмо состоит из трех частей - это конверт, заголовки и тело письма. Команды протокола SMTP находятся только в конверте. Таким образом данные в конверте определяют, как почта будут передаваться. Заголовки и тело письма формально не являются частью SMTP. Команды SMTP состоят из четырех символов, основные команды - это “HELO”, ”MAIL”, ”RCPT”, ”DATA”, “QUIT”. Команда “HELO” служит для установки соединения с сервером, команда “DATA” служит для того что бы сообщить принимающему серверу, что конверт закончился и дальше пойдет письмо. Сервер SMTP использует ответы состоящие из двух частей, а именно код сообщения и некоторое текстовое сообщение, поясняющее что именно произошло. Коды начинающиеся на “2” говорят, что предыдущая команда выполнена успешно (220 smtp.example.com ESMTP Postfix). Коды начинающиеся на “3” говорит о том что текущее состояние успешно, но необходимы дополнительные данные. Сообщения начинающиеся на “5” говорят о том что произошла ошибка (503 5.5.1 Error: need MAIL command). Так как текст сообщений не входит в стандарт, а для автоматической обработки используются только коды, то некоторые реализации серверов различные неформальные сообщения в ответах (221 2.0.0 So long, and thanks for all the fish).

Протокол SMTP разрабатывался очень давно, и не содержит механизмов защиты данных. Например поле “MAIL FROM” ни как не проверяется и другие пользователи могут отправлять письма от вашего имени если знают ваш адрес электронной почты. В расширенной версии протокола появилась возможность использовать шифрование при помощи команды “STARTTLS”.

Обширное изменение SMTP, так же как и полная его замена, считаются непрактичными из-за огромной инсталлированной базы SMTP. Internet Mail 2000 был одним из претендентов для такой замены.

Спам функционирует благодаря различным факторам, в том числе не соответствующие стандартам реализации MTA, уязвимости в защите операционных систем (усугубляемые постоянным широкополосным подключением), что позволяет спамерам удаленно контролировать компьютер конечного пользователя и посылать с него спам.

Существует несколько предложений для побочных протоколов, помогающих работе SMTP. Исследовательская группа Anti-Spam (The Anti-Spam Research Group — ASRG) — подразделение Исследовательской группы Интернет-технологий работает над почтовой аутентификацией и другими предложениями для предоставления простой аутентификации, которая будет гибкой, легковесной и масштабируемой. Недавняя деятельность [Инженерного совета Интернета](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D0%BD%D1%8B%D0%B9_%D1%81%D0%BE%D0%B2%D0%B5%D1%82_%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82%D0%B0) (IETF) включает в себя MARID (2004), приведший к двум утверждённым IETF-экспериментам в 2005, и [DomainKeys Identified Mail](https://ru.wikipedia.org/wiki/DomainKeys_Identified_Mail" \o "DomainKeys Identified Mail) в 2006.

Пример простейшей сессии (C: — клиент, S: — серверa).

S: (ожидает соединения)

C: (Подключается к порту 25 сервера)

S:220 mail.company.tld ESMTP is glad to see you!

C:HELO

S:250 domain name should be qualified

C:MAIL FROM: <someusername@somecompany.ru>

S:250 someusername@somecompany.ru sender accepted

C:RCPT TO: <user1@company.tld>

S:250 user1@company.tld ok

C:RCPT TO: <user2@company.tld>

S:550 user2@company.tld unknown user account

C:DATA

S:354 Enter mail, end with "." on a line by itself

C:From: Some User <someusername@somecompany.ru>

C:To: User1 <user1@company.tld>

C:Subject: tema

C:Content-Type: text/plain

C:

C:Hi!

C:.

S:250 769947 message accepted for delivery

C:QUIT

S:221 mail.company.tld CommuniGate Pro SMTP closing connection

S: (закрывает соединение)

3 Описание программы

3.1 Технологии программы

Приложение реализовано в различных проектах и классах. Были задействованы технологии сокетов, C# Windows Forms, функционал языка разработки C#, архитектура клиент-сервер и т.д.

C# Windows Forms – это технология для создания графического интерфейса пользователя, применяемая для языка программирования C#.

В приложении используется архитектура клиент-сервер. Где сервер обслуживает клиентов. Сервер прослушивает порт, чтобы обнаружить входящие соединения от одного или нескольких клиентов. В связи с отсутствием аппаратного обеспечения, для демонстрации работоспособности программы выбран случай, когда и клиент и сервер расположен на одной машине. В этом случае посылаемые данные, передаются стеку, но не перемещаются в выходную очередь сетевого устройства, а закольцовываются системой и возвращаются обратно в стек, но уже в качестве принятых данных. На этапе разработки данный тип размещения клиента и сервера дает некоторые преимущества, например оценка производительности, отсутствие сетевых задержек и т.д. Перенос сервера на другую машину не повлияет на работоспособность программы.

3.2 Структурная и физическая схема

Результатом разделение и проектирования стала следующая файловая структура классов:

* Dashboard -> класс, отвечающий за работу сервера;
* SMTPServer -> класс, отвечающий за обработку клиентов в отдельных потоках;
* ClientSMTP -> класс, предназначенный для посылки запросов серверу;

UML-структура программы представлена на рисунке 1. Красным – серверная часть, Зеленым – клиентская часть.

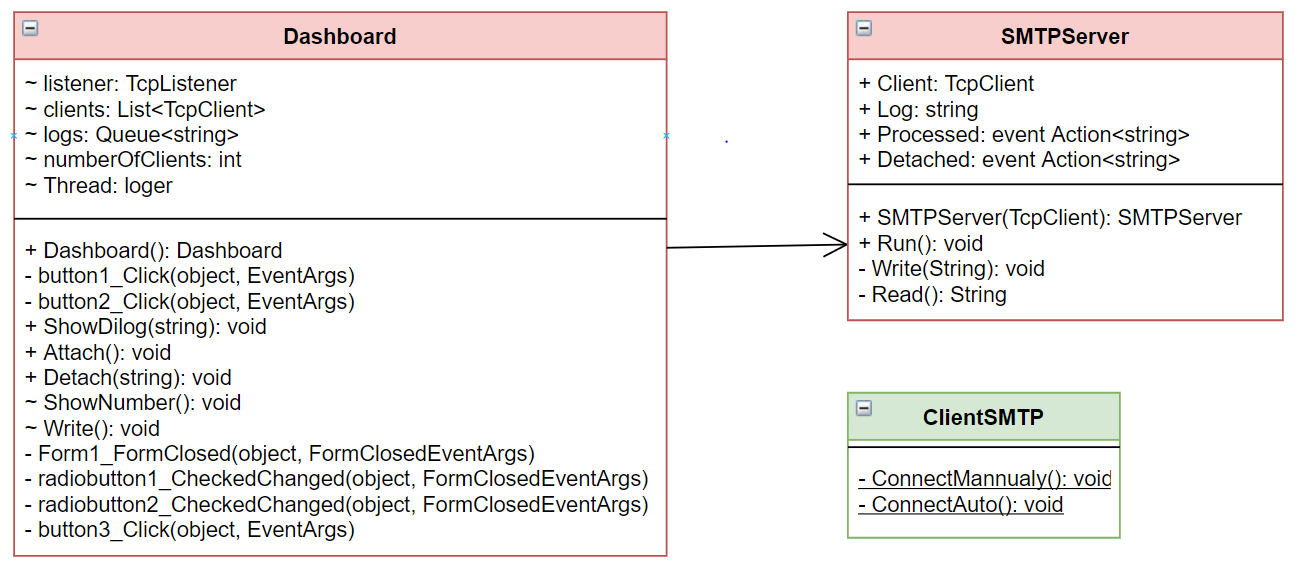


Рисунок 1 – UML-диаграмма программы

3.3 Функционал

Сервер способен принимать сообщения от клиентов, предварительно установив TCP соединение. В зависимости от содержания принятого сообщения, сервер посылает ответное сообщение. Например, если сервер видит, что сообщение начинается на “DATA” (определенная в стандарте команда), тогда он ответит “354 send the mail data end with .” Где “354” – код состояния, означающий, что все хорошо, но необходимы дополнительные данные. И оставшаяся часть сообщения – это не определенный в стандарте текст. У разных серверов это сообщение может различаться, оно не влияет на работу, но позволяет человеку понять, что происходило в тот или иной момент общения.

GUI позволяет пользователю отслеживать когда клиенты подключаются, отключаются, количество подключенных в данный момент, а так же пользователь может видеть приходящие и исходящие сообщения. Сервер может обслуживать несколько клиентов. Вся накопленные данные собираются в файл.

Код сервера располагается в двух классах: Dashboard, SMTPServer. Класс Dashboard содержит список подключившихся клиентов, объект TcpListener, сконфигурированный на порт 25, очередь на логирование диалогов а так же объект класса Thread, который осуществляет запись диалогов из очереди в файл в отдельном потоке. Когда любой из клиентов завершает общение с сервером, происходит событие “Detached” и обработчик события добавляет принятую строку в очередь на запись. Даже если пользователь закроет приложение нажав на “крестик”, останется один поток, который запишет оставшиеся в очереди строки в файл и только потом процесс приложения завершиться. Кроме того, класс Dashboard содержит множество обработчиков событий, которые происходят при работе потоков обрабатывающих клиентов.

Основным методом в классе Dashboard можно назвать асинхронный метод button1\_Click. Этот метод запускает прослушивание, а когда клиент подключился, создает новый поток для этого клиента и осуществляет подписки обработчиков на события этого самого нового потока.

Класс SMTPServer непосредственно обменивается сообщениями с клиентом. Он содержит событие Processed и событие Detached, которые вызываются при появлении сообщения в диалоге и при отключении клиента соответственно. Методы Read и Write читают данные и записывают в NetworkStream. Метод Run вызывает ранее описанные методы в зависимости от условий, сложившихся в ходе работы программы.

Для тестирования сервера было написано консольное приложение клиента. Код клиента расположен в классе ClientSMTP. В классе ClientSMTP определены два метода, а именно для автоматической отправки почтового сообщения при помощи высокоуровневого класса из .Net Framework SMTPClient и для отправки и создания сообщений вручную с использованием TcpClient. Пример работы клиента в “ручном” режиме (то есть пользователь набирает сообщения на клавиатуре и посылает их серверу, потом сервер присылает ответное сообщение) рисунок 2.

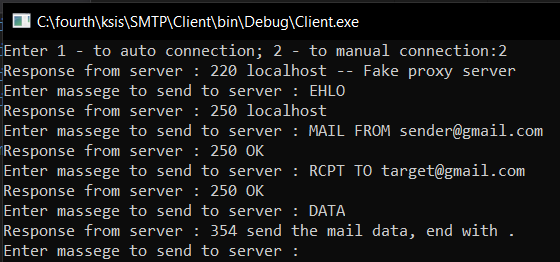


Рисунок 2 – Работа клиента в ручном режиме

4 Руководство пользователя

Руководство пользователя предназначено для облегчения освоения программы новыми пользователями, поэтому для SMTP сервера также прилагается сопутствующая документация.

Когда пользователь запускает программу, он имеет доступ только к radio group, кнопки будут недоступны (кроме “Clean”) пока пользователь не выберет один из двух режимов работы рисунок 3.

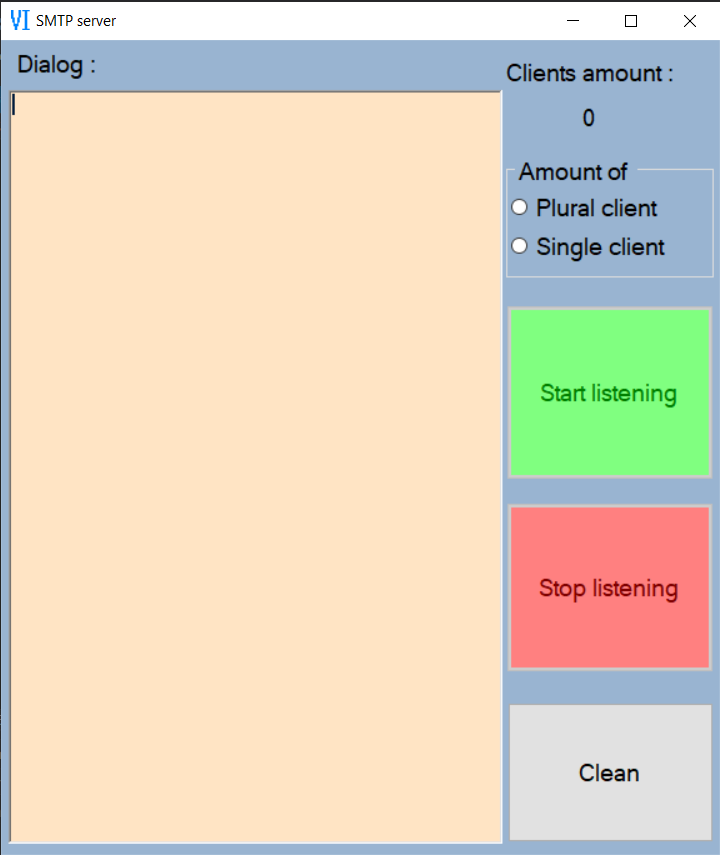


Рисунок 3 – Изначальный вид

После выбора режима работы открывается кнопка “Start listening” рисунок 4.

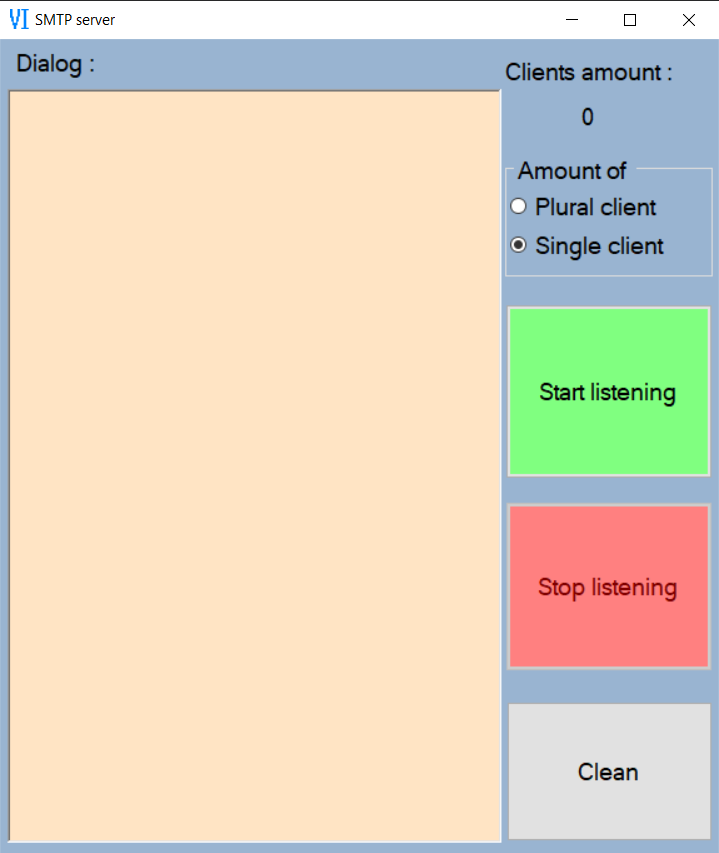


Рисунок 4 – Зеленая кнопка открыта

После нажатия зеленой кнопки, сервер начинает слушать подключения, открывается красная кнопка и закрывается radio group рисунок 5.

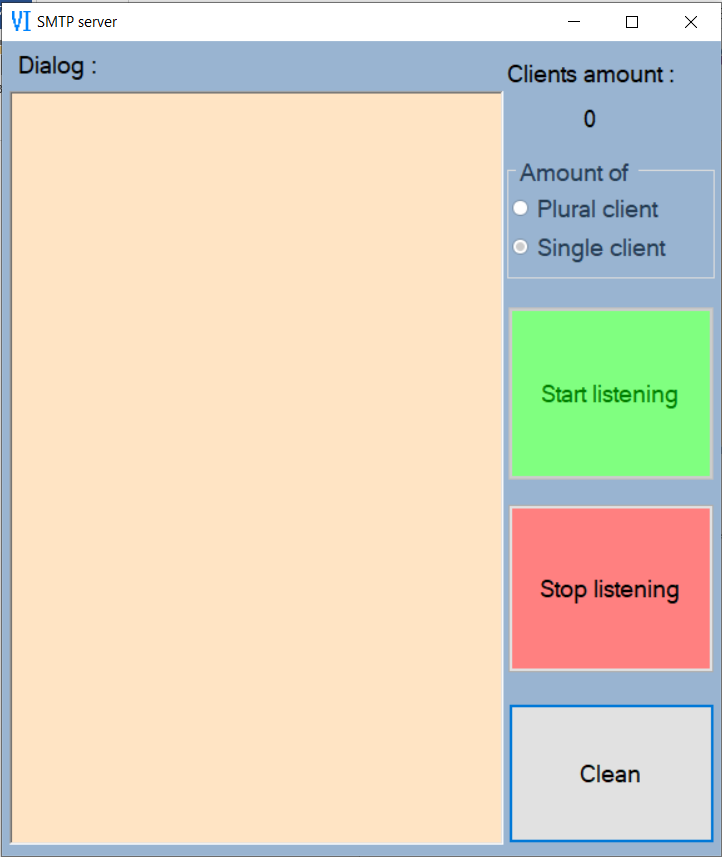


Рисунок 5 – Прослушивание запущено

После подключения клиента происходит обмен сообщениями и весь протокол общения отображается в окне рисунок 6. Счетчик остается нулевым так как клиент был обработан и отключен.

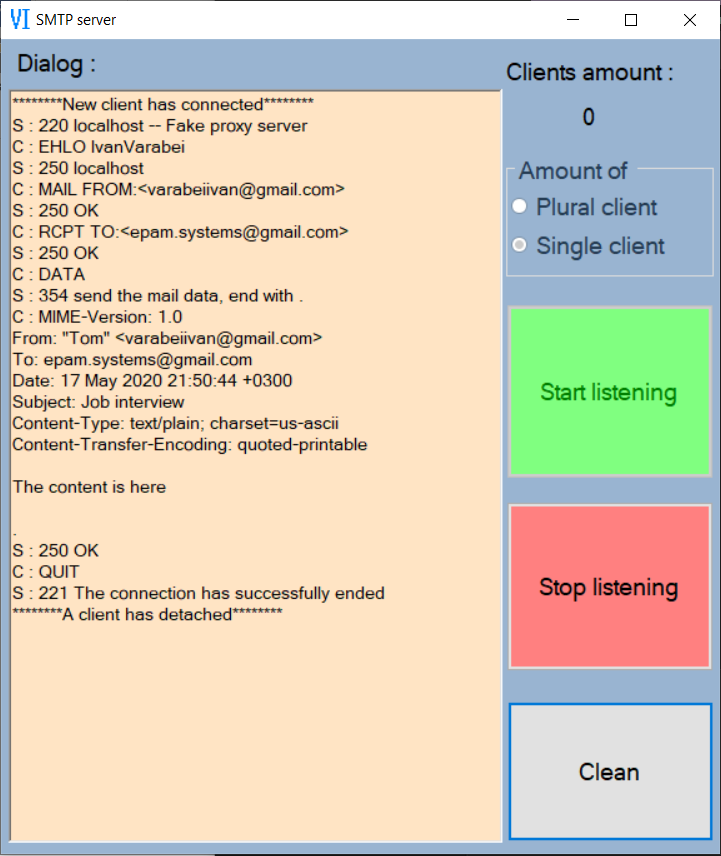


Рисунок 6 – Работа сервера в режиме одного клиента

После нажатия кнопки “Stop listening” открывается зеленая кнопка и radio group, а красная кнопка закрывается рисунок 7.

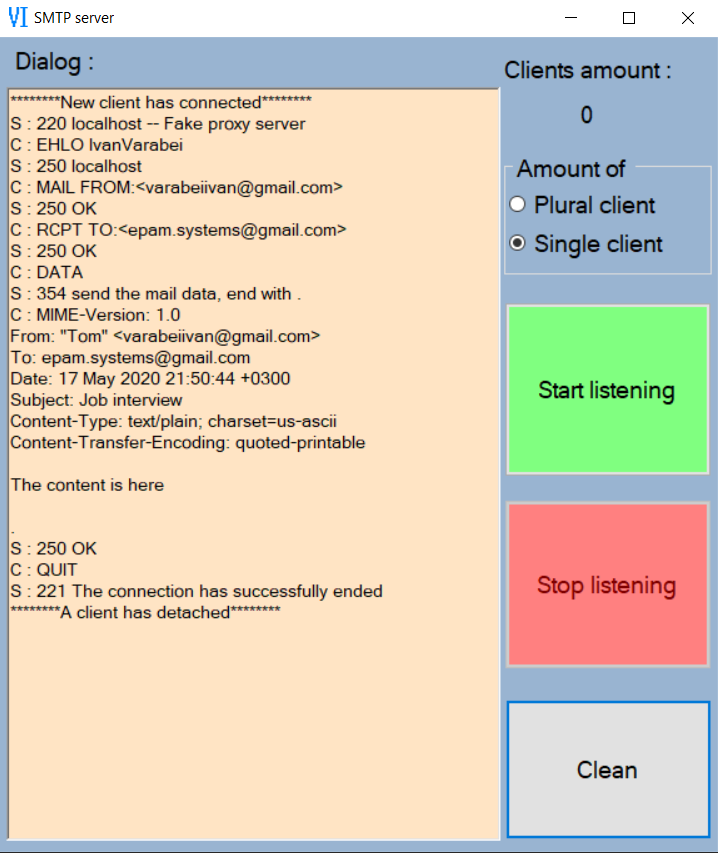


Рисунок 7 – Остановка прослушивания

Далее клиент может выбрать режим множества клиентов, при этом на стороне клиента необходимо выбрать ручной ввод, что бы увидеть что к серверу подключено несколько клиентов. В режиме множества клиентов диалоги выводятся только в файл, в интерфейсе приложения отображаются только сообщения о подключении и отключении клиентов. Счетчик количества подключенных клиентов отображает работает рисунок 8.

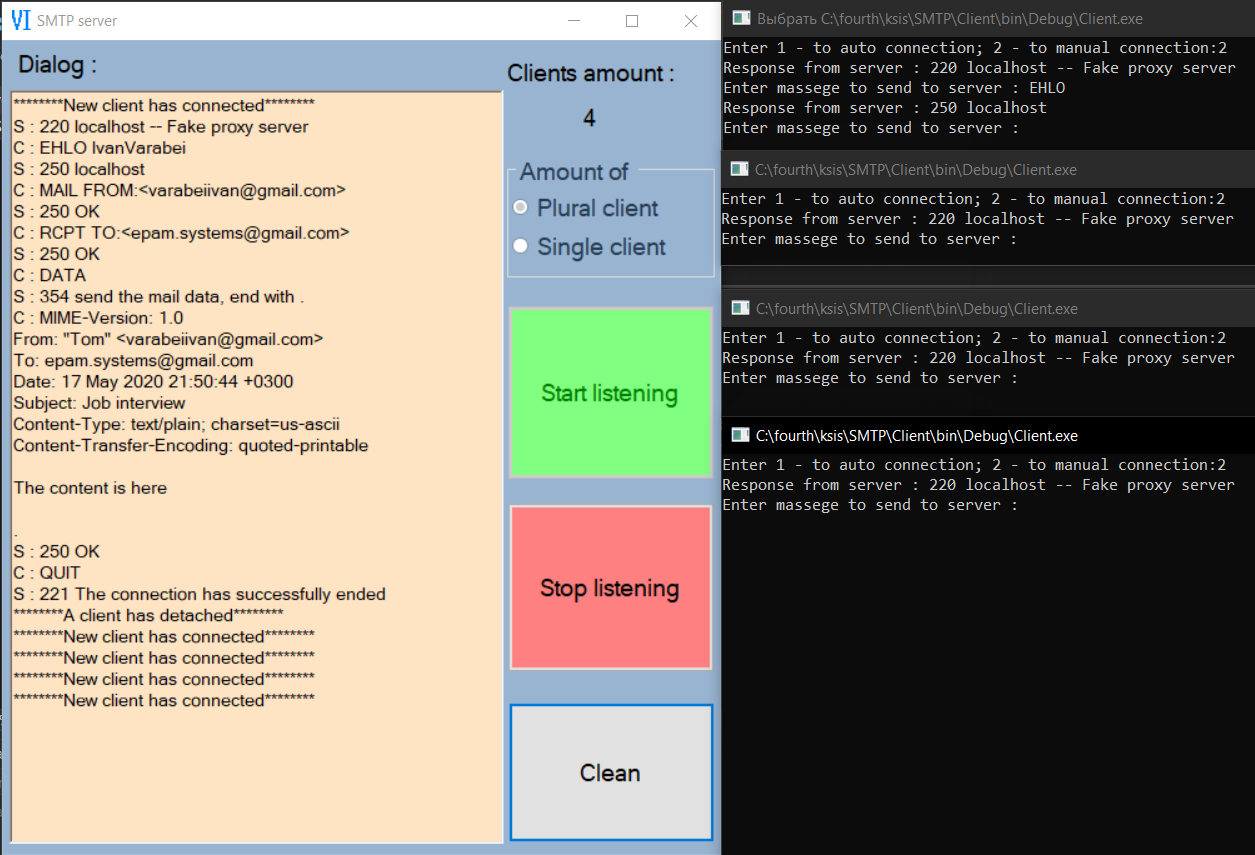


Рисунок 8 – Режим множества клиентов

Два клиента отключились, это отображено в окне вывода и на счетчике рисунок 9.

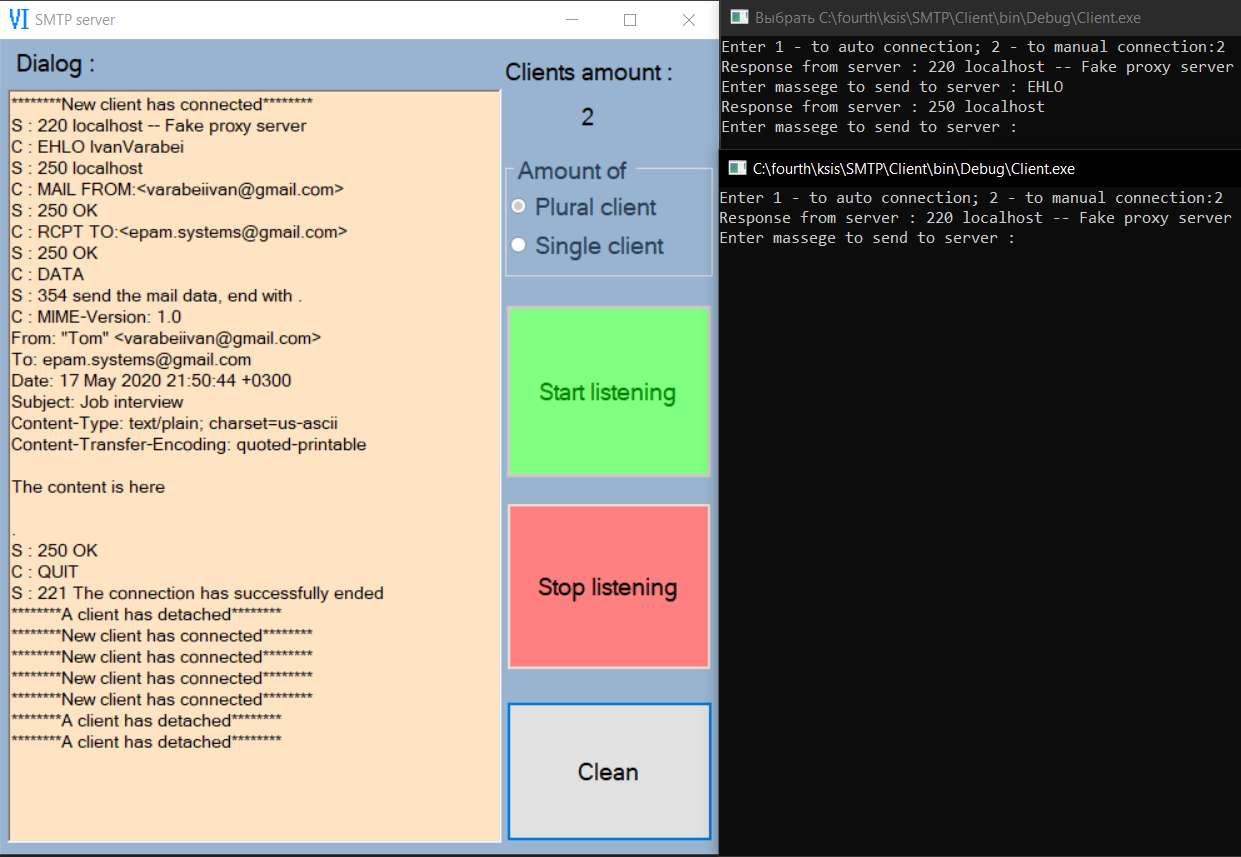


Рисунок 9 – Отключение клиентов

Кнопка “Clean” очищает окно рисунок 10.

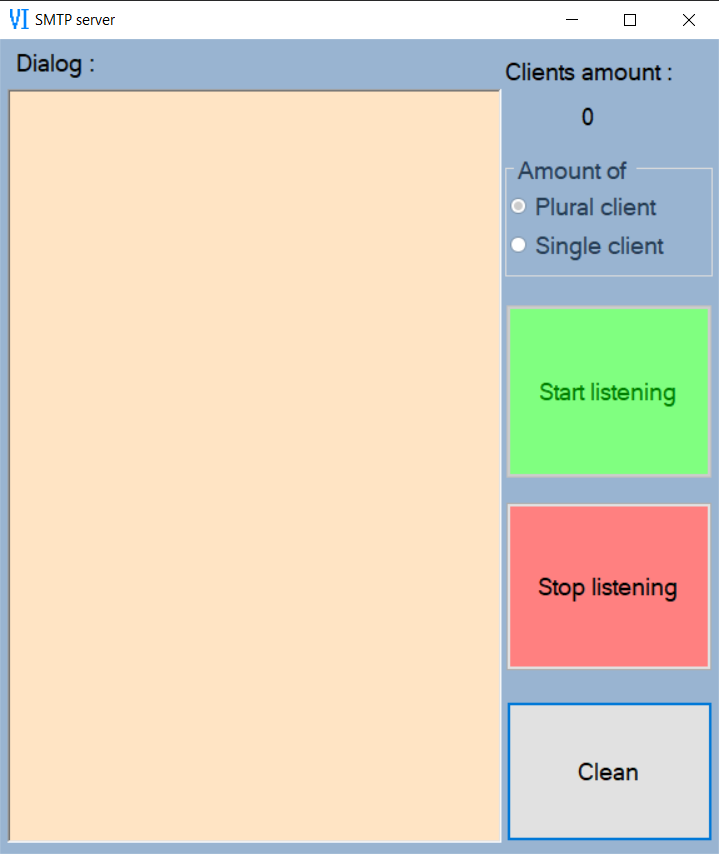


Рисунок 10 – Очистка окна после отключения клиентов

Все диалоги были записаны в файл рисунок 11.

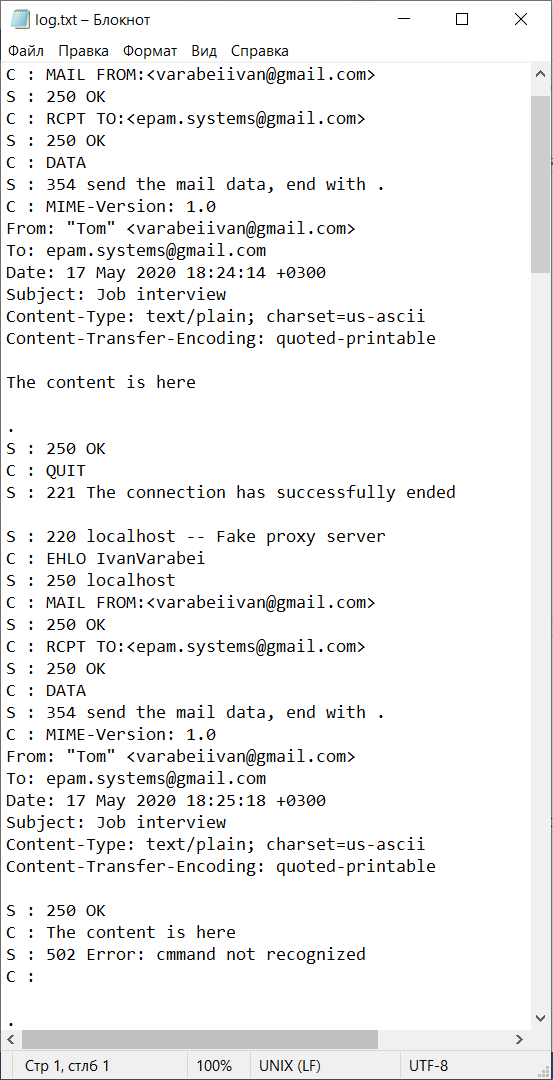


Рисунок 11 – Содержимое файла

5 Тестирование

Что бы избежать некорректной работы программы у пользователей, выявить баги или другие проблемные места программы, было проведено тестирование.

Для тестирования было написано консольное приложение клиента, которое позволяет отправлять сообщения на сервер.

Были протестированы функции обмена сообщениями в обоих режимах работы сервера, а так же функции логирования данных. Кроме того было протестировано приложение клиента как в режиме автоматической сборки отправки почтового сообщения так и в режиме ручного управления клиентом.

Также после исправления ошибок, выявленных во время тестирования, полученная программа готова к своему релизу.

Через некоторое время тестирования, все «тестировщики» дали положительный отзыв о работоспособности программы, что говорит о качестве полученного софта.

ЗАКЛЮЧНЕНИЕ

Во время работы над курсовым проектом были закреплены знания, полученные по курсу «Компьютерные системы и сети»», изучена технология сокетов, классы .NET FRAMEWORK для работы с сокетами и стеком протоколов TCP/IP.

В ходе курсового проектирования создан многопоточный SMTP сервер. Разработанный программный продукт предназначен для работы с несколькими клиентами и сохранения истории диалогов.

Программа представляет собой клиент-серверное многопользовательское приложение на основе потоковых асинхронных сокетов, в котором реализованы следующие функции:

* режим работы с одним клиентом;
* режим работы с множеством клиентов в разных потоках;
* отображение данных в графическом интерфейсе
* отображение всего протокола общения с клиентом;
* логирование протоколов общения.

Использование разработанного программного обеспечения автоматизирует процесс работы почты.

В ходе тестирования системы исправлены выявленные ошибки. Программа выполняет все поставленные перед ней задачи и функции. Пояснительная записка содержит подробное руководство пользователя.

Разработанный программный продукт может быть использован на практике в процессе обучения.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Метанит. [Электронный ресурс] / Сетевое программирование в C# и .Net. – Режим доступа: <https://metanit.com/sharp/net/>.– Дата доступа 10.05.2020, свободный – Загл. с экрана. – Яз.рус.
2. Столингс, В. Компьютерные сети, протоколы и технологии Интернета / В. Столлингс. – БХВ-Петербург, 2005. - 817 с.
3. Таненбаум, Э. Компьютерные сети / Э. Таненбаум. - 4-е изд. – СПб.: Питер, 2004.– 848с.
4. .NET. Сетевое программирование / В. Кумар [и др.]. – М.: Лори, 2007.- 400 с.
5. ВикипедиЯ. Свободная энциклопедия. [Электронный ресурс] / Сетевая модель OSI. – Режим доступа: https://ru.wikipedia.org/wiki/Сетевая модель\_OSI. - Дата доступа 27.07.2019, свободный – Загл. с экрана. – Яз. рус.

ПРИЛОЖЕНИЕ А

Листинг ПО

**ServerSMTP.Server.cs**

using System;

using System.Net.Sockets;

using System.Text;

namespace ServerSMTP

{

class SMTPServer

{

public TcpClient Client { get; set; }

public string Log { get; private set; } = "";

public event Action<string> Processed = (w) => { };//UpperCase event

public event Action<string> Detached = (s) => { };//UpperCase event

public SMTPServer(TcpClient client)

{

Client = client;

}

public void Run()

{

Write("220 localhost -- Fake proxy server");

string strMessage = String.Empty;

while (true)

{

try

{

strMessage = Read();

}

catch

{

Detached(Log);

break;

}

if (strMessage.Length > 0)

{

if (strMessage.StartsWith("QUIT"))

{

Write("221 The connection has successfully ended");

Client.Close();

Detached(Log);

break;

}

else if (strMessage.StartsWith("EHLO") || strMessage.StartsWith("HELO"))

{

Write("250 localhost");

}

else if(strMessage.StartsWith("RCPT TO"))

{

Write("250 OK");

}

else if(strMessage.StartsWith("MAIL FROM"))

{

Write("250 OK");

}

else if (strMessage.StartsWith("HELP"))

{

Write("EHLO\nHELO\nQUIT\nRCPT TO\nMAIL FROM\nDATA\nHELP");

}

else if(strMessage.StartsWith("DATA"))

{

Write("354 send the mail data, end with .");

strMessage = Read();

Write("250 OK");

}

else

{

Write("502 Error: cmmand not recognized");

}

}

}

}

private void Write(String strMessage)

{

NetworkStream clientStream = Client.GetStream();

ASCIIEncoding encoder = new ASCIIEncoding();

byte[] buffer = encoder.GetBytes(strMessage + "\r\n");

clientStream.Write(buffer, 0, buffer.Length);

clientStream.Flush();

Log += "S : " + strMessage + "\n";

Processed("S : " + strMessage + "\n");

}

private String Read()

{

byte[] messageBytes = new byte[8192];

int bytesRead = 0;

NetworkStream clientStream = Client.GetStream();

ASCIIEncoding encoder = new ASCIIEncoding();

bytesRead = clientStream.Read(messageBytes, 0, 8192);

string strMessage = encoder.GetString(messageBytes, 0, bytesRead);

if (strMessage.EndsWith("\n"))

{

Processed("C : " + strMessage);

Log += "C : " + strMessage;

}

else

{

Processed("C : " + strMessage + "\n");

Log += "C : " + strMessage + "\n";

}

return strMessage;

}

}

}

**ServerSMTP.Dashboard.cs**

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

using System.Threading;

using System.Windows.Forms;

using System.IO;

namespace ServerSMTP

{

public partial class Dashboard : Form

{

TcpListener listener = new TcpListener(new IPEndPoint(IPAddress.Any, 25));

List<TcpClient> clients = new List<TcpClient>();

Queue<string> logs = new Queue<string>();

int numberOfClients;

Thread loger;

public Dashboard()

{

InitializeComponent();

loger = new Thread(new ThreadStart(Write));

loger.Start();

}

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

{

listener.Start();

button1.Enabled = groupBox1.Enabled = false;

button2.Enabled = true;

await Task.Run(() =>

{

while (true)

{

try

{

TcpClient client = listener.AcceptTcpClient();

clients.Add(client);

SMTPServer handler = new SMTPServer(client);

handler.Detached += (s) => { numberOfClients--; ShowNumber(); };

handler.Detached += Detach;

handler.Detached += (text) => logs.Enqueue(text);

if (radioButton2.Checked == true)

handler.Processed += ShowDilog;

Thread thread = new Thread(new ThreadStart(handler.Run));

thread.Start();

Attach();

numberOfClients++;

ShowNumber();

if (radioButton2.Checked == true)

{

listener.Stop();

break;

}

}

catch

{

return;

}

}

});

}

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

{

clients.ForEach((s) => s.Close());

button2.Enabled = false;

groupBox1.Enabled = button1.Enabled = true;

listener.Stop();

ShowNumber();

}

public void ShowDilog(string s)

{

Action action = () =>

{

richTextBox1.AppendText(s);

richTextBox1.ScrollToCaret();

};

this.InvokeEx(action);

}

public void Attach()

{

Action action = () =>

{

richTextBox1.AppendText("\*\*\*\*\*\*\*\*New client has connected\*\*\*\*\*\*\*\*\n");

richTextBox1.ScrollToCaret();

};

this.InvokeEx(action);

}

public void Detach(string s)

{

Action action = () =>

{

richTextBox1.AppendText("\*\*\*\*\*\*\*\*A client has detached\*\*\*\*\*\*\*\*\n");

richTextBox1.ScrollToCaret();

};

this.InvokeEx(action);

}

void ShowNumber()

{

Action action = () => label1.Text = "" + numberOfClients;

this.InvokeEx(action);

}

void Write()

{

while (true)

{

if (logs.Count > 0)

{

string writePath = @"C:\fourth\ksis\SMTP\log.txt";

try

{

using (StreamWriter sw = new StreamWriter(writePath, true, Encoding.Default))

{

sw.WriteLine(logs.Dequeue());

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

else

{

Thread.Sleep(50);

}

}

}

private void Form1\_FormClosed(object sender, FormClosedEventArgs e)

{

while (logs.Count > 0) { }

loger.Abort();

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e) => button1.Enabled = true;

private void radioButton2\_CheckedChanged(object sender, EventArgs e) => button1.Enabled = true;

private void button3\_Click(object sender, EventArgs e) => richTextBox1.Clear();

}

}

**ClientSMTP.ClientSMTP.cs**

using System;

using System.Net.Mail;

using System.Net.Sockets;

using System.Text;

namespace Client

{

class ClientSMTP

{

static void Main(string[] args)

{

Console.Write("Enter 1 - to auto connection; 2 - to manual connection:");

if (Console.ReadLine() == "2")

ConnectManually();

else

ConnectAuto();

Console.WriteLine("Connection ended.");

Console.Read();

}

private static void ConnectManually()

{

try

{

TcpClient client = new TcpClient();

client.Connect("localhost", 25);

byte[] data = new byte[256];

StringBuilder response = new StringBuilder();

NetworkStream stream = client.GetStream();

try

{

while (true)

{

do

{

int bytes = stream.Read(data, 0, data.Length);

response.Append(Encoding.UTF8.GetString(data, 0, bytes));

}

while (stream.DataAvailable); // пока данные есть в потоке

Console.Write("Response from server : " + response.ToString());

response = new StringBuilder();

Console.Write("Enter massege to send to server : ");

string mesToServ = Console.ReadLine();

data = Encoding.UTF8.GetBytes(mesToServ);

stream.Write(data, 0, data.Length);

}

}

catch

{

stream.Close();

client.Close();

}

}

catch (SocketException e)

{

Console.WriteLine("SocketException: {0}", e);

}

catch (Exception e)

{

Console.WriteLine("Exception: {0}", e.Message);

}

Console.WriteLine("Request ended.");

}

private static void ConnectAuto()

{

// отправитель - устанавливаем адрес и отображаемое в письме имя

MailAddress from = new MailAddress("varabeiivan@gmail.com", "Tom");

// кому отправляем

MailAddress to = new MailAddress("epam.systems@gmail.com");

// создаем объект сообщения

MailMessage m = new MailMessage(from, to);

// тема письма

m.Subject = "Job interview";

// текст письма

m.Body = "The content is here";

SmtpClient smtp = new SmtpClient("localhost");

smtp.Send(m);//Handles all messages in the protocol

Console.WriteLine("Request has been sent.");

smtp.Dispose();//sends a Quit message

}

}

}

ПРИЛОЖЕНИЕ Б

Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 28  **UML-диаграмма проекта** | | | | | | | | | | |
|  |  |  |  |  | ДП-1070111809-2020-01 | | | | | |
|  |  |  |  |  |
| Изм | Лист | № документа | Подпись | Дата |
| Разраб. | | Воробей |  |  | Программное обеспечение для организации курсового проекта | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | У |  | 1 | 3 |
| Консульт. | | Белова |  |  | 1-40 01 01  БНТУ, г. Минск | | | | |
| Н.контр | | Белова |  |  |
|  | |  |  |  |
| 29  **Основной функционал работы сервера** | | | | | | | | | | |
|  |  |  |  |  | ДП-1070111809-2020-02 | | | | | |
|  |  |  |  |  |
| Изм | Лист | № документа | Подпись | Дата |
| Разраб. | | Воробей |  |  | Интерфейс сетевого чата. | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | У |  | 2 | 3 |
| Консульт. | | Белова |  |  | 1-40 01 01  БНТУ, г. Минск | | | | |
| Н.контр | | Белова |  |  |
|  | |  |  |  |
| 30  **Наглядная работа клиента** | | | | | | | | | | |
|  |  |  |  |  | ДП-1070111809-2020-03 | | | | | |
|  |  |  |  |  |
| Изм | Лист | № документа | Подпись | Дата |
| Разраб. | | Воробей |  |  | Сетевой чат. Сервер. | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | У |  | 3 | 3 |
| Консульт. | | Белова |  |  | 1-40 01 01  БНТУ, г. Минск | | | | |
| Н.контр | | Белова |  |  |
|  | |  |  |  |