МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«ПОЛЕССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

(отметка о допуске курсовой работы

(проекта) к защите)

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

(подпись научного руководителя) (Ф.И.О.)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_ г.

Дата регистрации

работы на кафедре \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка

курсовой работы (проекта) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**КУРСОВАЯ РАБОТА (ПРОЕКТ)**

**ПО ДИСЦИПЛИНЕ**: Программирование сетевых приложений

на тему: Разработка сетевого приложения «Онлайн календарь и планировщик»

Исполнитель:

студент группы № 20ИТ-3

В.А. Хвесюкович

Научный руководитель:

В.А. Клаченков, магистр э.н., Ассистент

Пинск 2022

**ОГЛАВЛЕНИЕ**

[введение 4](#_Toc121167225)

[ГЛАВА 1 ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc121167226)

[1.1 Описание предметной области 5](#_Toc121167227)

[1.2 Выбор и обоснование способа решения задачи 5](#_Toc121167228)

[ГЛАВА 2 разработкА сетевого приложения СЕРВЕРА 6](#_Toc121167229)

[2.1 Описание основных функций сервера 6](#_Toc121167230)

[2.2 Список основных переменных в приложении сервера 6](#_Toc121167231)

[2.3 Сетевая часть сервера 7](#_Toc121167232)

[2.3.1 Реализация метода RecieveMessage 7](#_Toc121167233)

[2.3.2 Реализация метода SendMessage 7](#_Toc121167234)

[ГЛАВА 3 разработкА сетевого приложения клиентА 10](#_Toc121167235)

[3.1 Описание основных функций приложения клиента 10](#_Toc121167236)

[3.2 Список основных переменных в приложении клиента 10](#_Toc121167237)

[3.3 Сетевая часть клиента 11](#_Toc121167238)

[3.3.1 Реализация методов нажатия кнопок (отправка запросов) 11](#_Toc121167239)

[3.3.2 Реализация метода RecieveMessage 12](#_Toc121167240)

[ГЛАВА 4 Инструкция по эксплуатации программного обеспечения 14](#_Toc121167241)

[4.1 Клиент 14](#_Toc121167242)

[4.2 Сервер 18](#_Toc121167243)

[ЗАКЛЮЧЕНИЕ 19](#_Toc121167244)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc121167245)

[ПРИЛОЖЕНИЕ А 21](#_Toc121167246)

[Приложение Б 25](#_Toc121167247)

[Приложение В 30](#_Toc121167248)

[Приложение Г 31](#_Toc121167249)

# реферат

Курсовая работа: с. 31, рис. 24, 2 таблицы, 7 источников, 4 приложения.

КАЛЕНДАРЬ, ПЛАНИРОВЩИК, ОНЛАЙН ПРИЛОЖЕНИЕ, ПРОГРАММА, АЛГОРИТМИЗАЦИЯ, ПРОТОКОЛ, ОРГАНАЙЗЕР.

Объектом разработки является сетевое программное средство онлайн календаря и планировщика.

Цель курсовой работы: Создание сетевого программного обеспечения для управления задачами пользователя и сетевого хранения задач пользователя.

При выполнении работы был использован теоретический материал учебных пособий и ресурсов сети Интернет.

Областью возможного практического применения являются пользователи, которые хотят упорядочить все задачи и найти баланс между работой, домом и личными интересами.

Автор работы подтверждает, что приведенный в работе материал правилен и объективно отражает состояние исследуемого процесса (разрабатываемого объекта), все заимствованные из литературных и других источников теоретические, методологические и методические положения и концепции сопровождаются ссылками на их авторов.

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

­­*(подпись студента)*

# введение

Многие люди часто сталкиваются с проблемой лишних временных затрат при организации своего времени. Будь то директор компании, которому приходиться затрачивать немало времени на постановку задач для сотрудников, назначение времени встреч и сроков проведения различных мероприятий или человек, занимающийся какой-либо частной деятельностью, для которого так же важно иметь возможность легко и быстро обозначить список необходимых задач и выделить время под необходимые мероприятия [7].

Сфера применения планировщиков задач огромна: составление планов работ, списка покупок, текущих дел и т.д. Без хорошего планировщика задач в наши дни трудно, а самое главное — он всегда должен быть под рукой, чтобы вы в любой момент могли посмотреть, что у вас запланировано.

Темой курсовой работы является «Онлайн календарь и планировщик», реализация данной темы немало важна в наше время, так как сейчас у людей очень много дел и записывать все в тетрадку как-то неудобно, поэтому данное веб-приложение будет довольно востребованным. По сравнению с другими похожими проектами можно выделить следующие отличительные черты: очень простой, понятный и не надоедливый дизайн.

Задачи курсовой работы:

* + 1. Создать приложения сервера и клиента.
    2. Реализовать передачу информации между этими приложениями.
    3. Реализовать хранение информации о клиентах на сервере.
    4. Реализовать создание, изменение и удаление заметок в приложениях сервера и клиента.

# ИССЛЕДОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

## Описание предметной области

Эффективная организация времени — актуальная проблема для любого человека, вне зависимости от его занятости. Этой дисциплине, известной также как таймменеджмент, посвящена не одна книга, регулярно проводятся многочисленные курсы, предлагающие разные методики.

Свою лепту в планирование времени вносят органайзеры. Вообще, говоря о данной сфере программного обеспечения, следует отметить, что служат они не только для составления распорядка дня, но и для хранения данных. Человеку сложно держать в голове, или же в неупорядоченном виде, всю необходимую информацию, которая должна быть под рукой.

## Выбор и обоснование способа решения задачи

Для решения данной задачи разработали программное приложение, с использованием языка программирования C# и средой программирования Visual studio. В программе использовали графический интерфейс Windows Forms для упрощенного взаимодействия с элементами программного кода [2].

C# — объектно-ориентированный, ориентированный на компоненты язык программирования. C# предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. C# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET. Благодаря этому C# подходит для создания и применения программных компонентов.

Язык C# был выбран, так как он объектно-ориентированный и изучается в рамках дисциплины ООП.

# разработкА сетевого приложения СЕРВЕРА

## Описание основных функций сервера

Сервер работает автоматически и выполняет следующие операции:

1. Получать сообщения (запросы) от всех подключенных клиентов и отправлять ответ каждому конкретному клиенту (многопоточность).
2. Осуществлять регистрацию новых клиентов.
3. Подтверждать вход уже существующих клиентов в программу.
4. Хранить все заметки, созданные пользователем.
5. После подтверждения входа клиента в систему, сервер должен отправить информацию о всех заметках клиента.
6. Осуществлять создание, изменение и удаление заметок определенных пользователей.

## Список основных переменных в приложении сервера

Таблица 1 – Список переменных приложения Сервер

|  |  |  |  |
| --- | --- | --- | --- |
| Переменная | Тип | Стандартное значение | Описание |
| users | List<User> |  | Список всех пользователей (и их записок), хранящихся на сервере |
| remoteAddress | string |  | Удаленный адрес, с которого пришел запрос клиента |
| remotePort | int | - | Удаленный порт, с которого пришел запрос клиента |
| localPort | int | 8889 | Локальный порт сервера |
| buffer | byte[65000] |  | Запрос клиента |
| response | string |  | Ответ сервера |
| recieveThread | thread | - | Поток, создается для метода принятия сообщений |
| receiver | UdpClient | - | Создается для принятия сообщений |
| sender | UdpClient |  | Создается для отправки сообщений |

## Сетевая часть сервера

Для разработки сетевой части приложения выбран протокол UDP, так как сеть соединения является открытой [6]. Основным в сетевой части разрабатываемого приложения, есть реализация многопоточной работы сервера (сервер одновременно отвечает на запросы нескольких клиентов).

Для создания приложения Сервер использовали технологию Windows forms, представленную в среде разработки Visual Studio. При загрузке консольного приложения сервера создается udpClient, и сервер ожидает запросы клиентов.

Основные операции сервер производит посредством двух методов: RecieveMessage (получение сообщений клиентов) и SendMessage (отправление ответа сервера).

Запускается метод принятия сообщений сразу при открытии программы. При создании, программе присевается определенный IP-адрес, чтобы клиент могли отправлять сообщения, а сервер - принимать.

Метод ReceiveMessage запускается отдельным потоком.

### Реализация метода RecieveMessage

Код метода находится в приложении А.

Вначале создается UdpClient для получения сообщения, а также remoteEndPoint, переменная для хранения ip клиента, чтобы ему потом отправить ответ (рисунок 2.3.1.1).



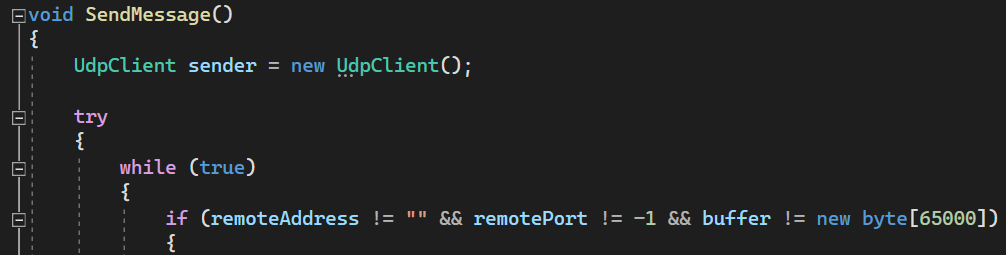
**Рисунок 2.3.1.1 – Метод RecieveMessage**

Затем запускается бесконечный цикл приема сообщений клиентов. На консоль выводятся полученные сообщения. Ip клиента сохраняется для того, чтобы сервер отправил ответ конкретному клиенту, от которого пришел запрос.

### Реализация метода SendMessage

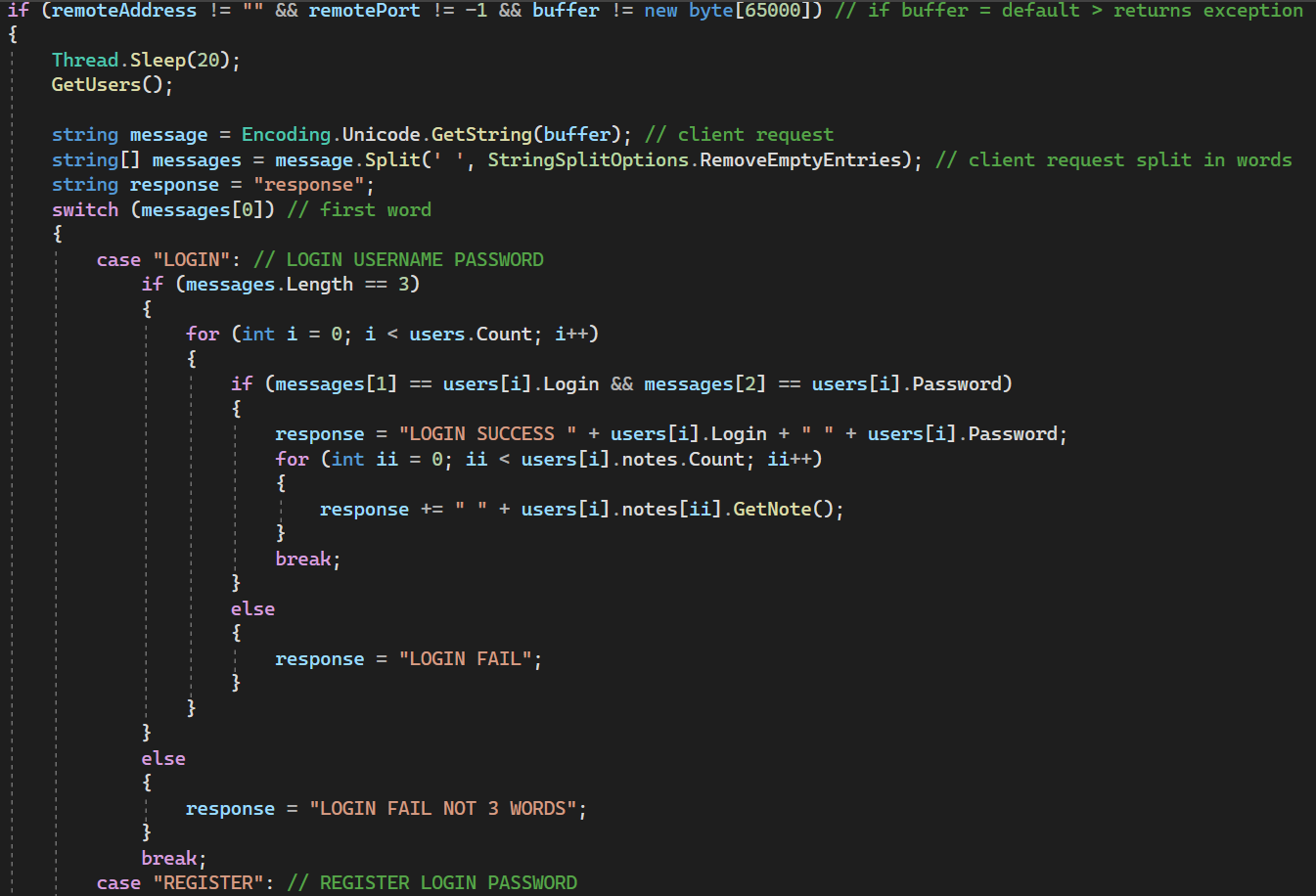
Код метода отправки сообщения находится в приложении А.

Вначале создается бесконечный цикл отправки ответа с сервера клиентам, но в нем происходит проверка, сохранен ли на сервере ip последнего клиента (рисунок 2.3.2.1).



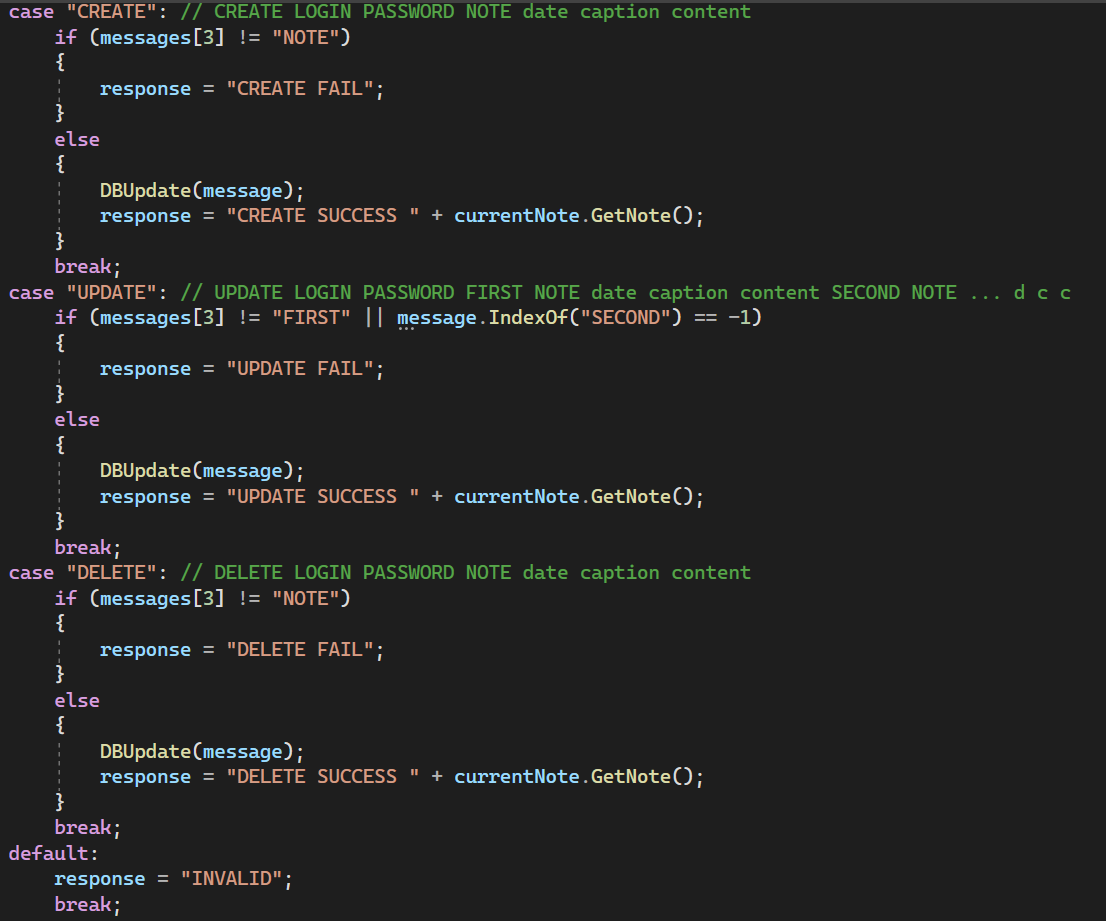
**Рисунок 2.3.2.1 – Проверка, был ли получен запрос от клиента**

Затем, если на сервер уже поступал запрос, то формируется ответ. Сервер считывает данные о клиентах из текстового файла (метод GetUsers), так как после определенных запросов клиентов, серверу придется менять данные в txt файле. После этого, сообщение клиента преобразуется в тип данных string, затем разбивается на слова, и происходит проверка первого слова в запросе клиента (рисунок 2.3.2.2).



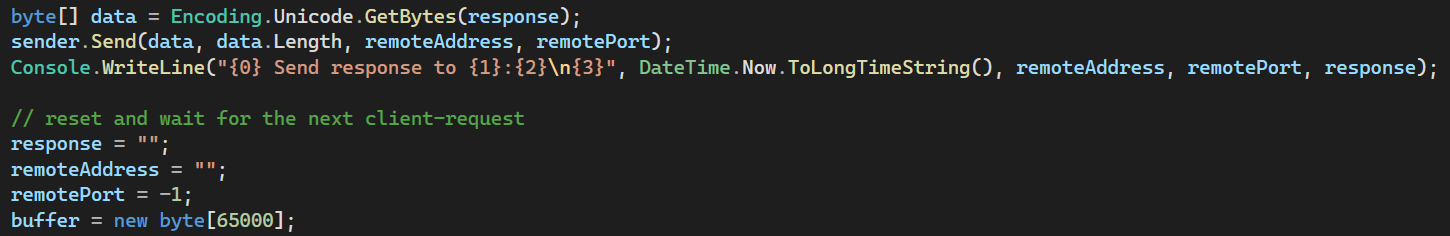
**Рисунок 2.3.2.2 – Проверка первого слова запроса клиента**

Сервер проверяет, какой запрос он получил, методом switch, в который заносится первое слово запроса. В зависимости от того, какой запрос отправил клиент, сервер сформирует разные ответы (рисунок 2.3.2.3).



**Рисунок 2.3.2.3 –Формирование ответа сервера в зависимости от запроса клиента**

После формирования ответа, сервер отправляет ответ на ip адрес клиента, от которого он получил запрос, а затем обнуляет переменные, где хранился запрос клиента и его ip (рисунок 2.3.2.4).



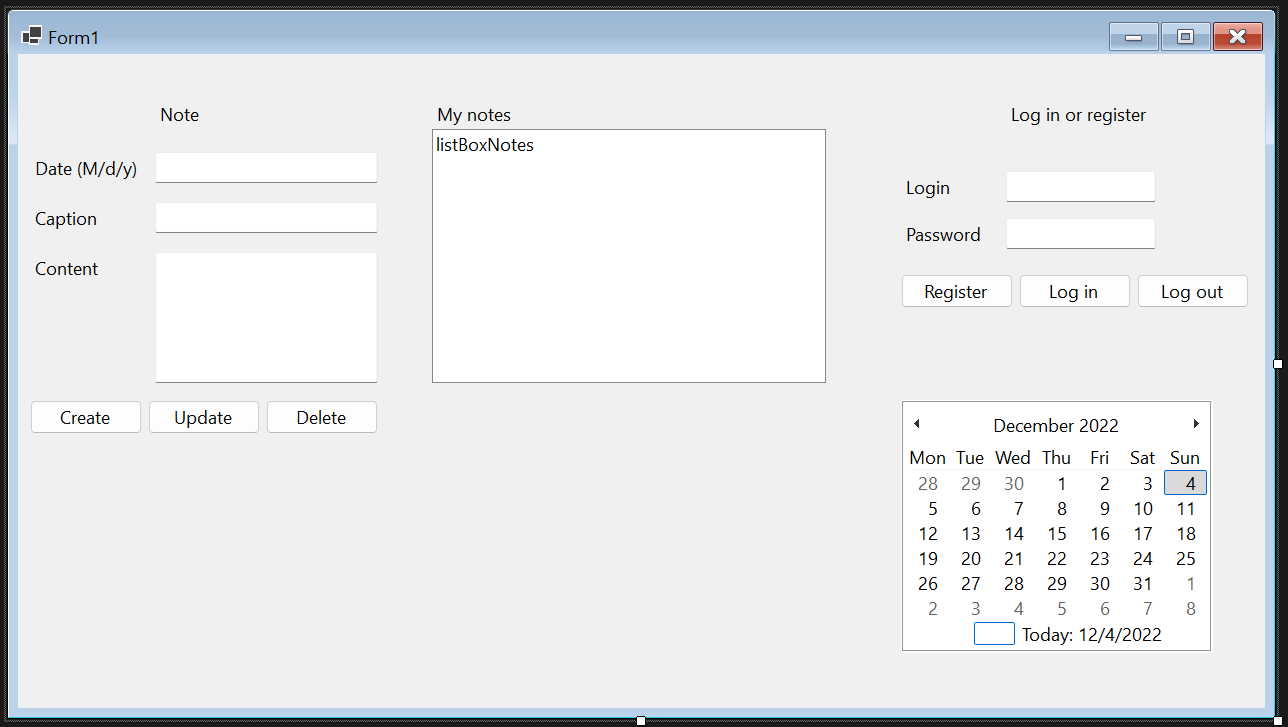
**Рисунок 2.3.2.4 –Обнуление предыдущего запроса клиента и ожидание нового**

# разработкА сетевого приложения клиентА

## Описание основных функций приложения клиента

Приложение клиента (рисунок 3.1.1) выполняет следующие функции:

1. Регистрация, вход или выход клиента.
2. Отображение всех заметок клиента.
3. Выделение жирным шрифтом дат календаря с заметками.
4. Создание новых заметок.
5. Изменение или удаление уже существующих заметок.



**Рисунок 3.1.1 –Вид приложения клиента**

## Список основных переменных в приложении клиента

Список переменных представлен в таблице 2.

Таблица 2 – список переменных приложения Клиент

|  |  |  |  |
| --- | --- | --- | --- |
| Переменная | Тип | Стандартное значение | Описание |
| remoteAddress | string |  | IP-адрес сервера |
| remotePort | int | 8889 | Порт сервера |
| localPort | int | - | Локальный порт клиента, устанавливается случайным образом |
| udpClient | UdpClient |  | Используется для получения и отправки сообщений |
| user | User | ("0", "0") | Объект класса user, используется для хранения логина, пароля и заметок |
| recieveThread | thread |  | Поток, создается для метода принятия сообщений |

## Сетевая часть клиента

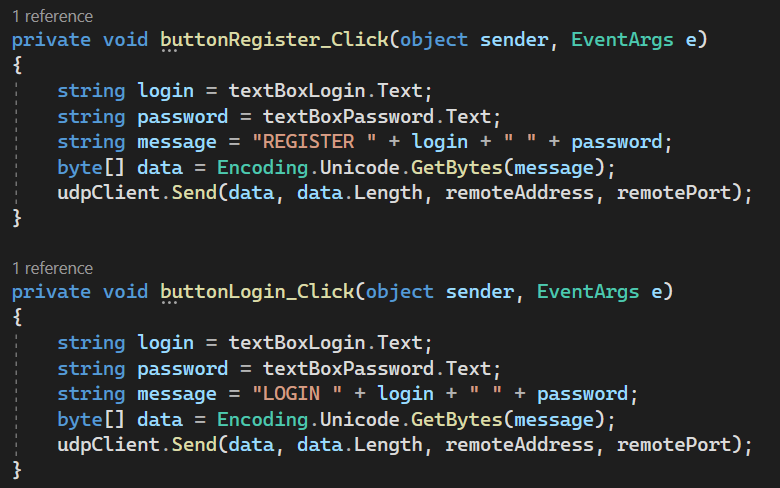
Отправка сообщений происходит посредством нажатия кнопок.

Принятие ответа сервера происходит в методе RecieveMessage.

### Реализация методов нажатия кнопок (отправка запросов)

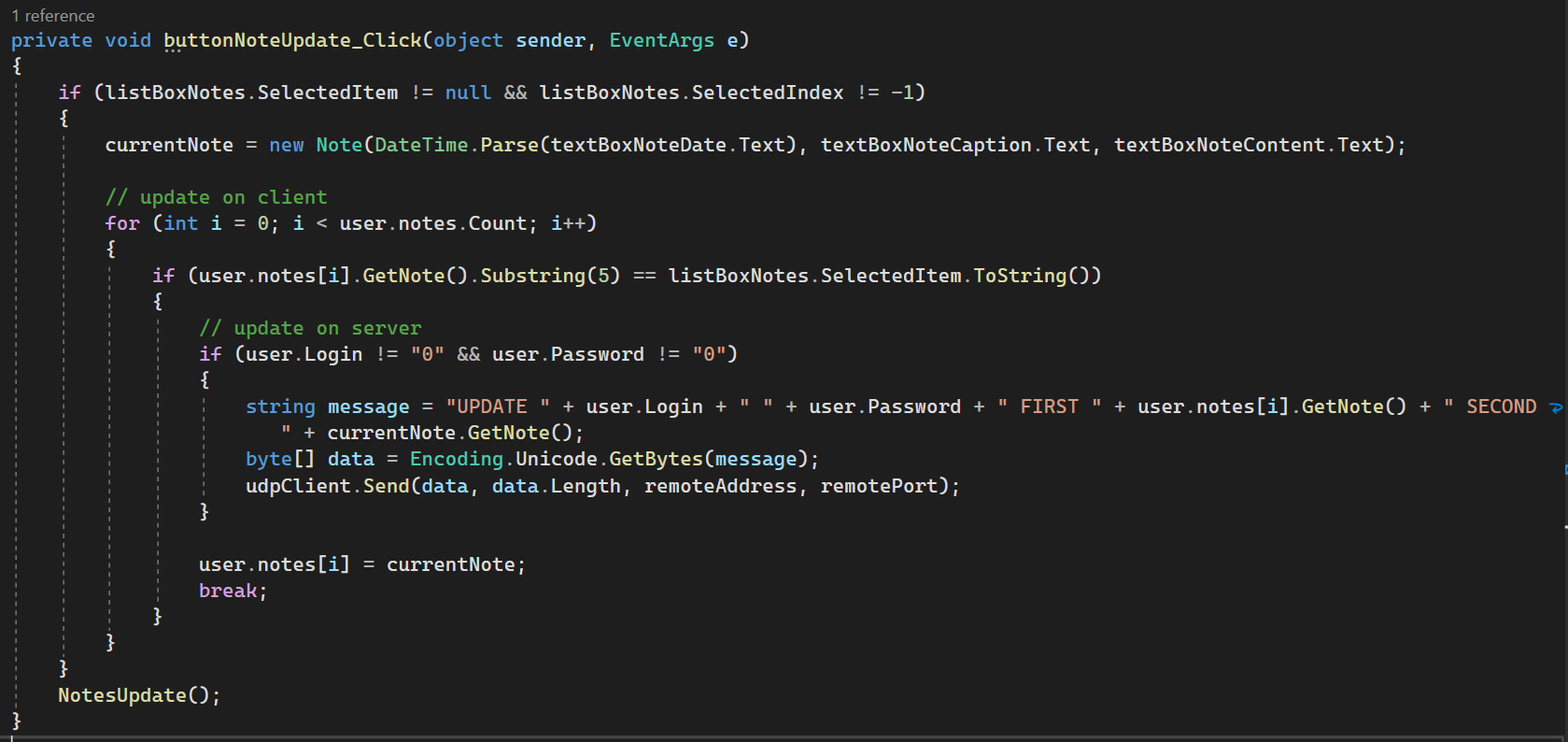
Код метода находится в приложении Б.

Клиент отправляет запросы серверу по нажатию кнопок. Вначале клиент вводит логин и пароль, затем регистрируется в приложении по нажатию кнопки “Register”, или входит в аккаунт по нажатию кнопки “Log in” (рисунок 3.3.1.1).



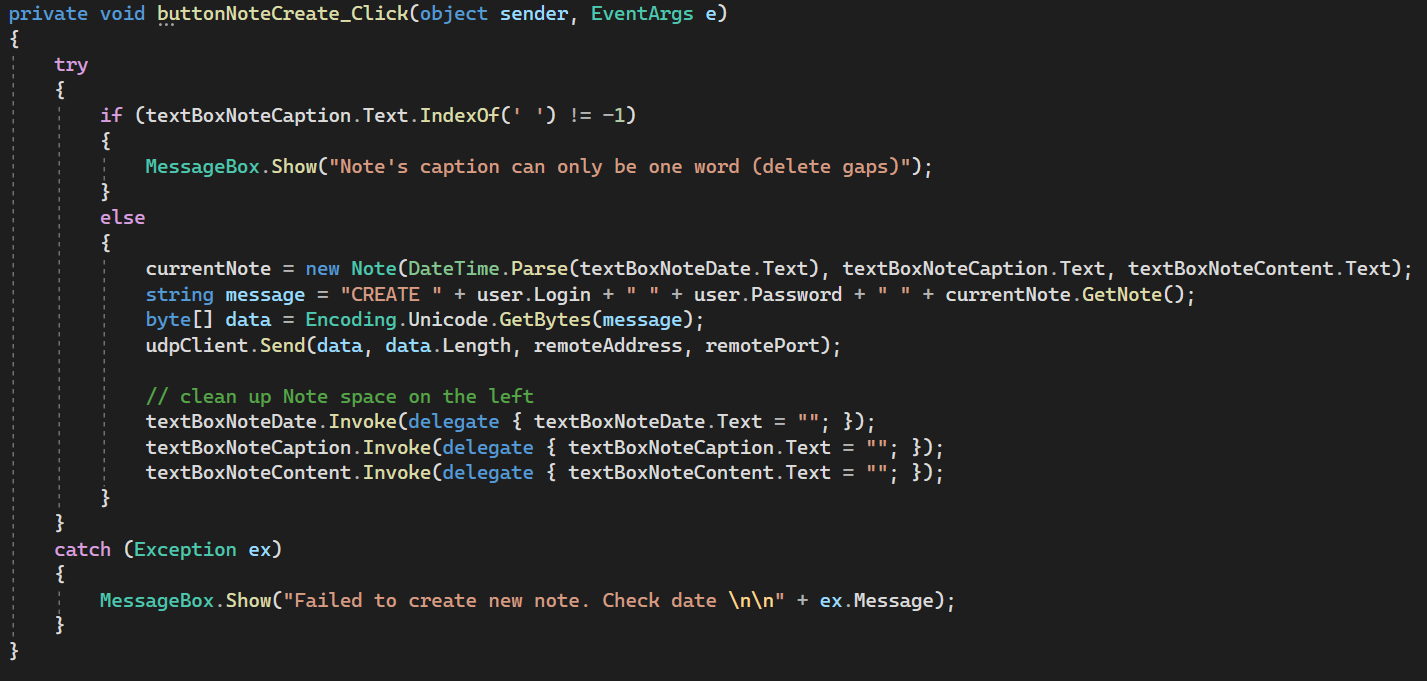
**Рисунок 3.3.1.1 – Работа кнопок Register и Log in**

После входа в аккаунт, клиент может просмотреть свои заметки и изменить или удалить уже существующие заметки (рисунок 3.3.1.2).



**Рисунок 3.3.1.2 – Работа кнопки Update (изменение заметки)**

Или создать новую заметку, записав дату, тему и описание заметки и нажав на кнопку Create (рисунок 3.3.1.3).



**Рисунок 3.3.1.3 – Работа кнопки Create (создание новой заметки)**

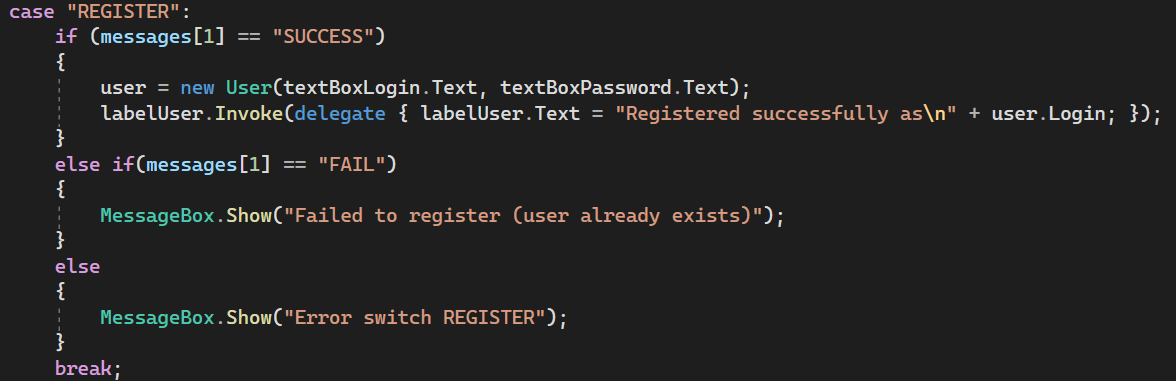
### Реализация метода RecieveMessage

Код метода получения сообщения в приложении Б.

Клиент получает сообщения от сервера только после отправки запросов (нажатия кнопок). Получение сообщений реализовано в бесконечном цикле.

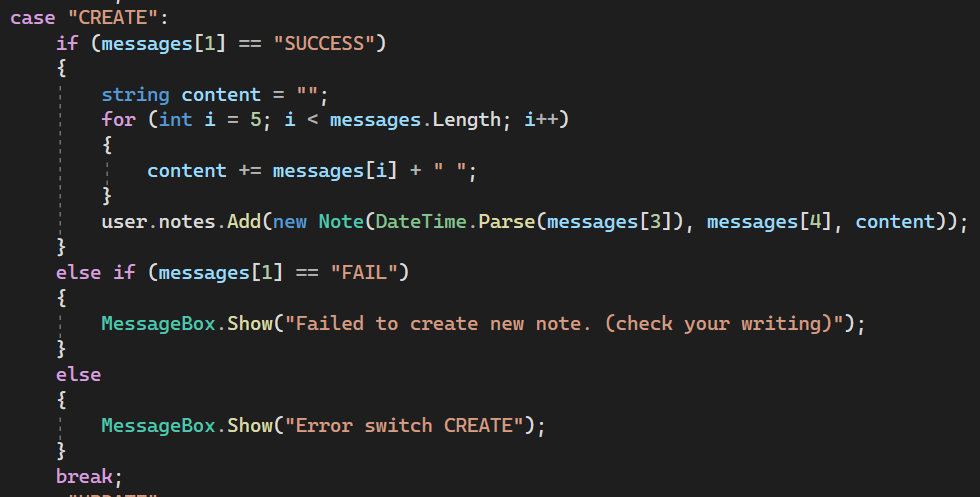
Первое слово полученного сообщения заносится в конструкцию switch, а затем, в зависимости от полученного ответа, в приложении клиента происходят изменения.

Например, при получении ответа сервера на запрос Register (регистрация), сервер либо успешно зарегистрирует пользователя, либо отправит ответ о возникшей ошибке (клиент уже зарегистрирован) (рисунок 3.3.2.1).



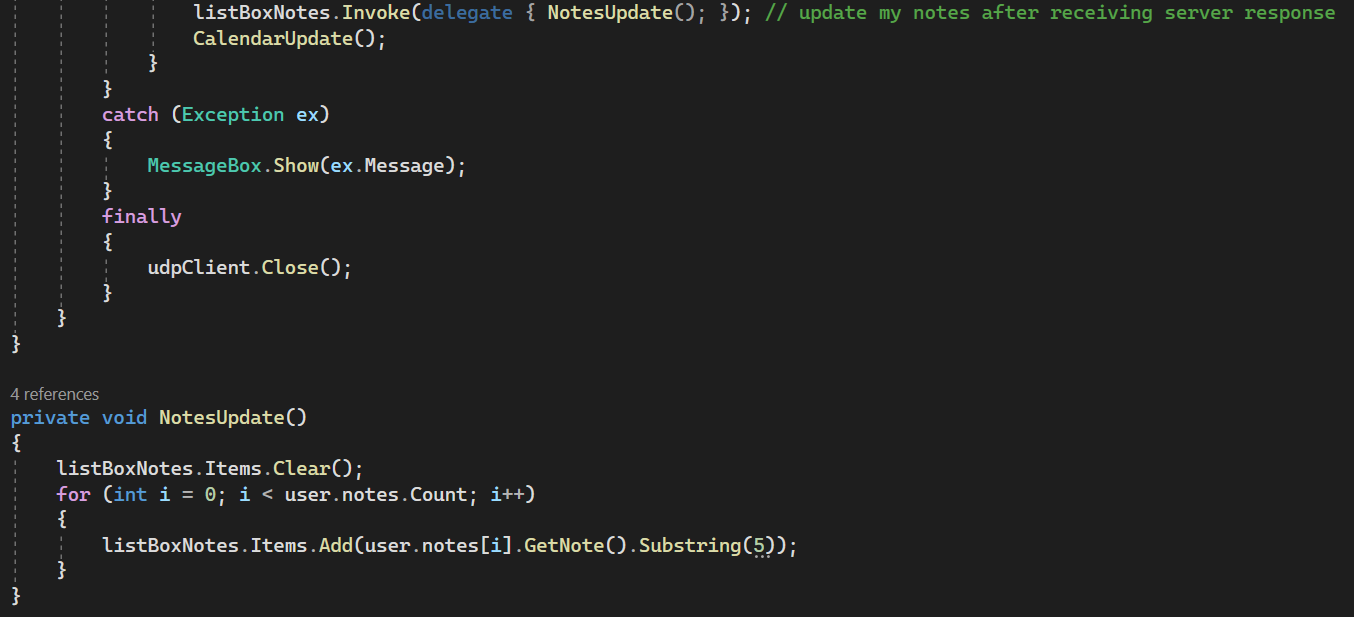
**Рисунок 3.3.2.1 – Получение ответа сервера на запрос Register (регистрация)**

При создании новой заметки, клиент может получить отказ, если, например, дата указана неверно (рисунок 3.3.2.2).



**Рисунок 3.3.2.2 – Получение ответа сервера на запрос Create (создать заметку)**

После каждого получения ответа от сервера, на стороне клиента обновляется список заметок (рисунок 3.3.2.3).

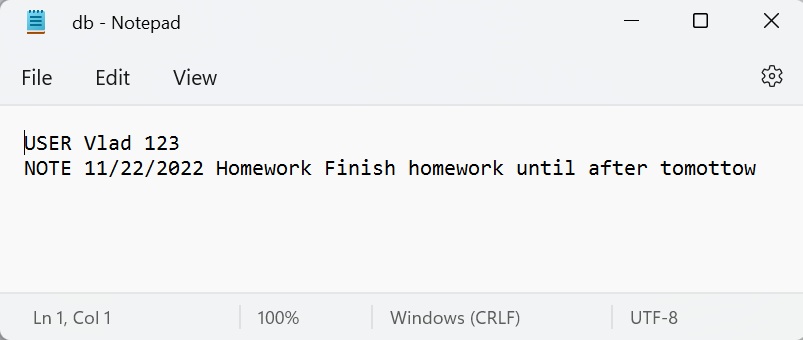


**Рисунок 3.3.2.3 – Обновление списка заметок**

# Инструкция по эксплуатации программного обеспечения

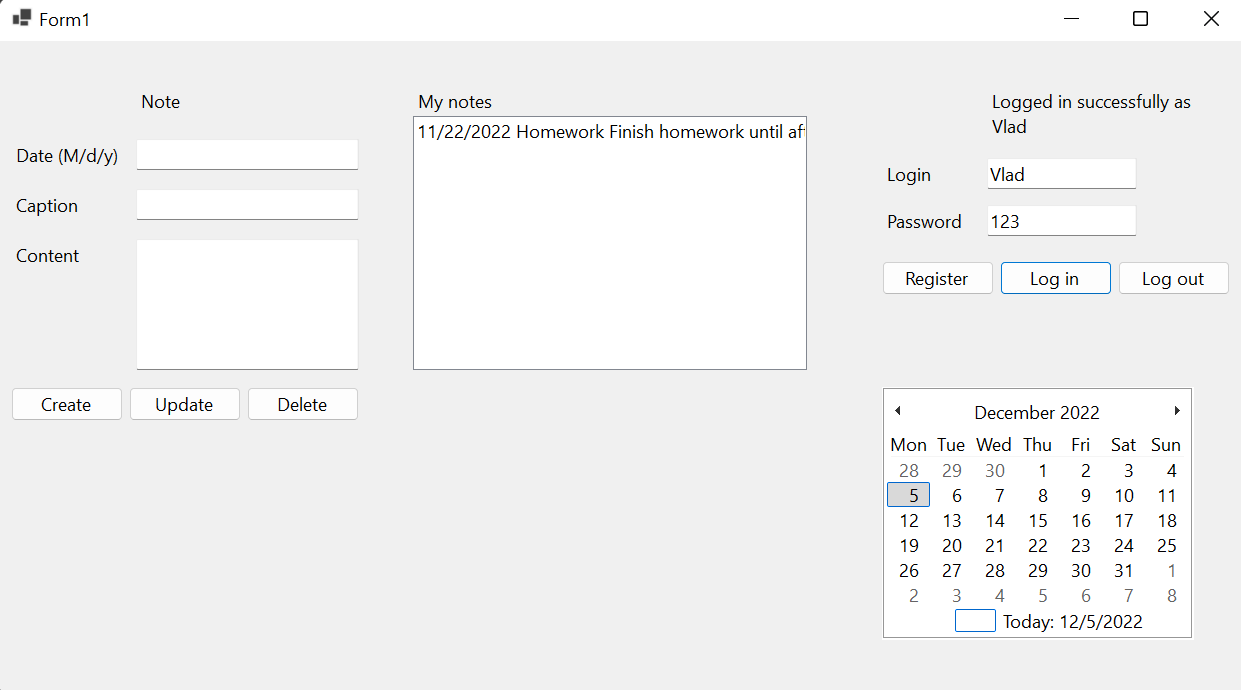
## Клиент

Рассмотрим пример взаимодействия приложений клиента с сервером. Допустим, при запуске приложений сервера и клиента, на стороне клиента, в текстовом файле, была записана информация, предоставленная на рисунке 4.1.1.



**Рисунок 4.1.1 – Текстовый файл во время запуска приложений**

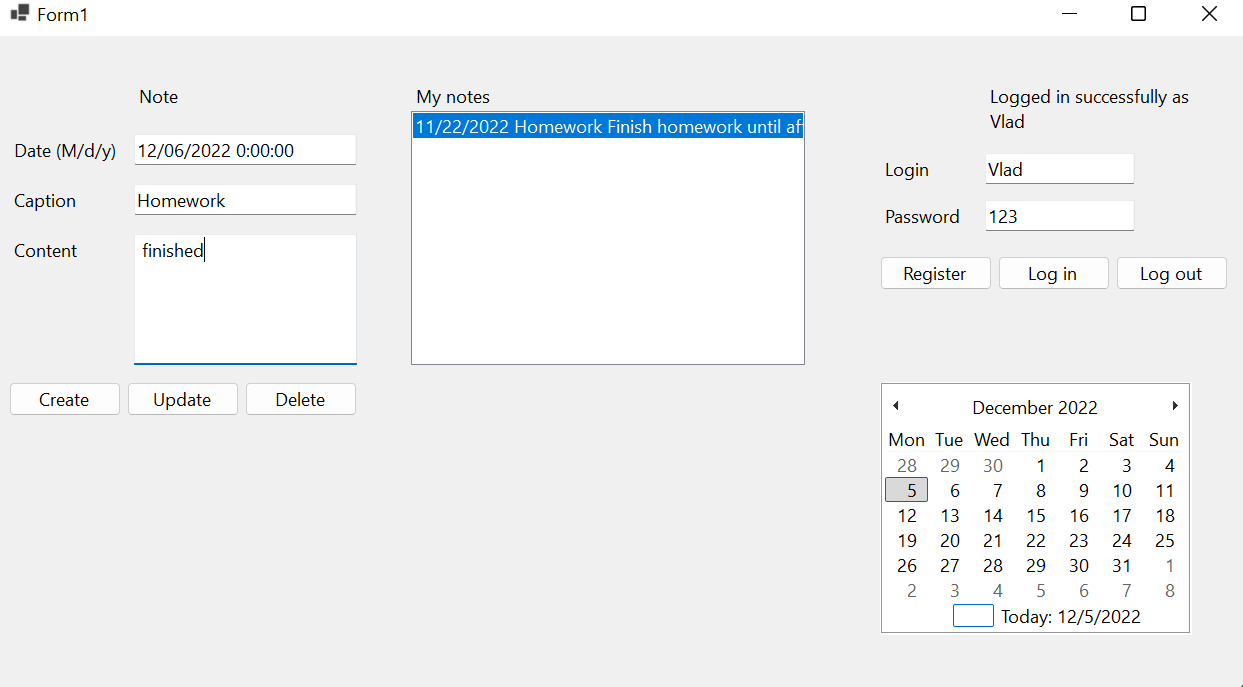
Для выполнения главной задачи данного программного обеспечения (сохранение заметок пользователя на сервере), клиенту необходимо зарегистрироваться, или войти в уже существующий аккаунт. Для этого клиенту необходимо ввести логин и пароль, и нажать на кнопку Register или на Log in (рисунок 4.1.2).



**Рисунок 4.1.2 – Авторизация клиента**

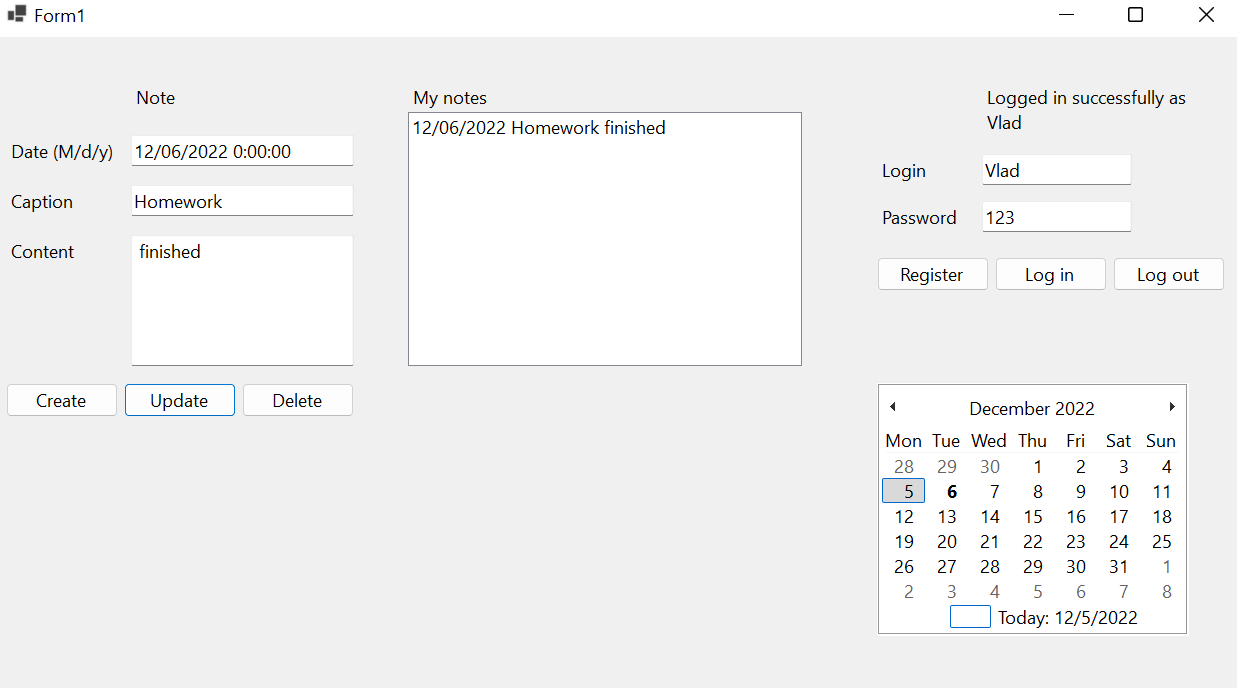
После входа в учетную запись, клиент получает информацию обо всех его заметках, которые уже хранились на сервере (рисунок 4.1.2).

Допустим, клиент хочет изменить заметку. Для этого он должен выбрать заметку, которую он хочет изменить, и, слева листа с заметками, изменить содержание заметки и нажать на кнопку “Update” (рисунок 4.1.3).

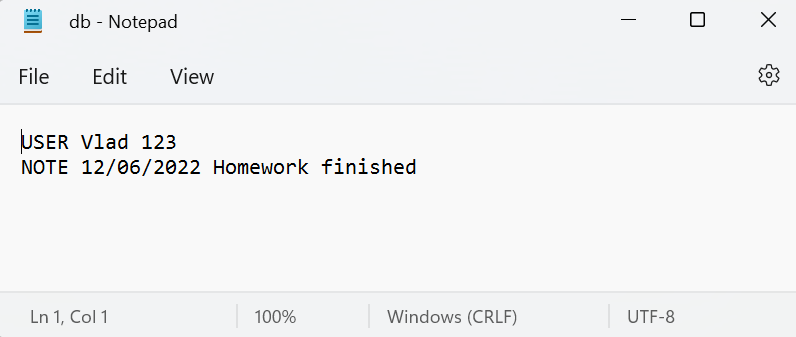


**Рисунок 4.1.3 – Процесс изменения заметки пользователем**

После нажатия на кнопку “Update”, заметка изменится на стороне клиента (рисунок 4.1.4), а также и на стороне сервера (рисунок 4.1.5). Также, на стороне клиента, дата заметки станет выделена жирным шрифтом.

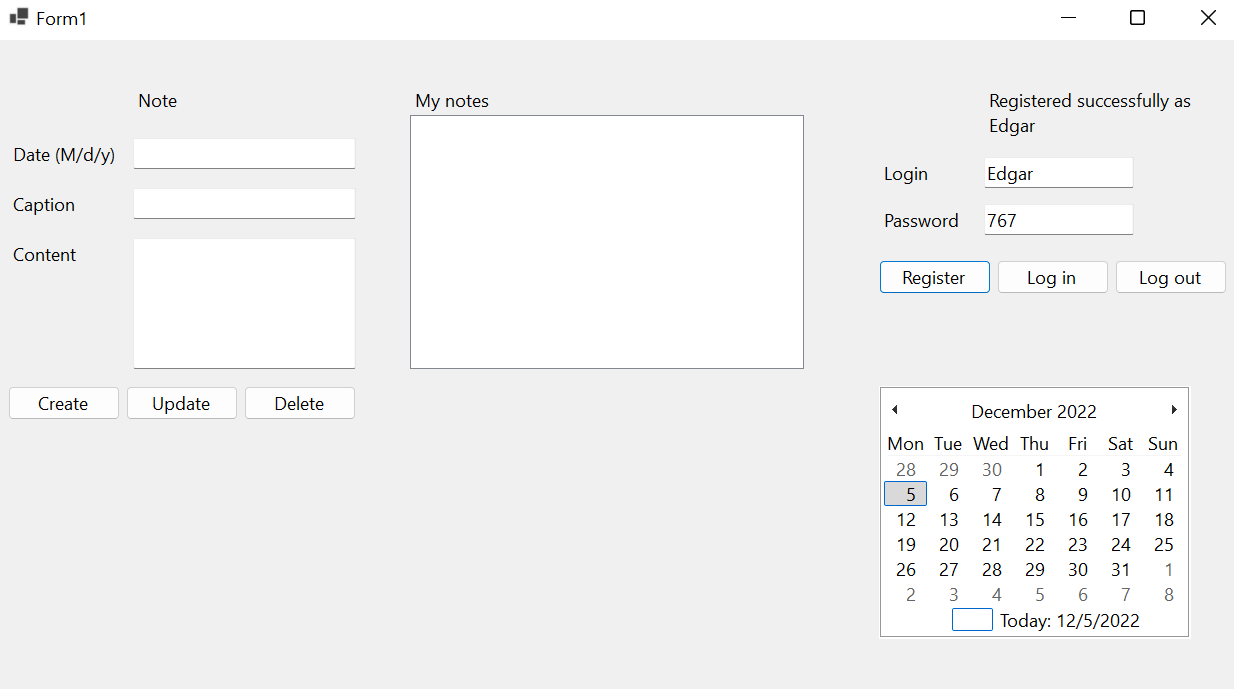


**Рисунок 4.1.4 – Заметка пользователя была изменена на стороне клиента**

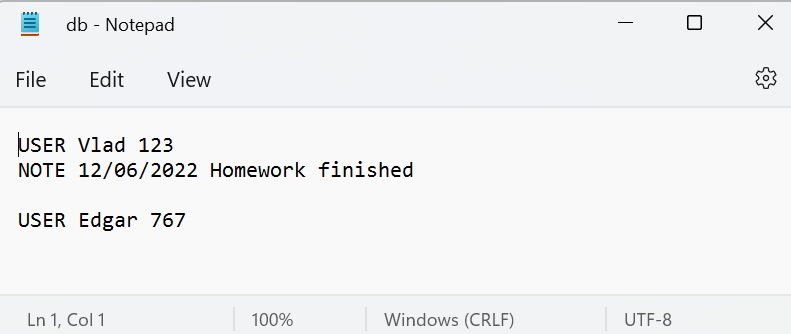


**Рисунок 4.1.5 – Текстовый файл после изменения заметки**

Допустим, после изменения заметки пользователя Vlad, к серверу подключился новый клиент, и он же решил зарегистрироваться как новый пользователь (реализация многопоточности) (рисунок 4.1.6-7).

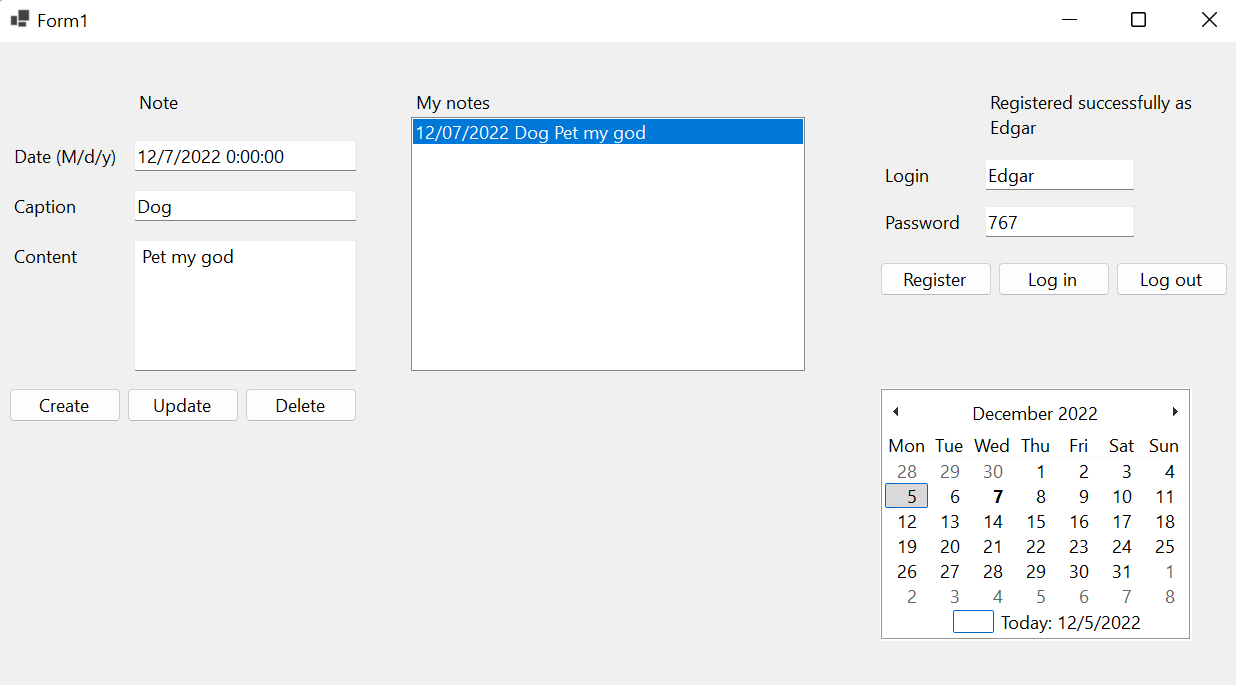


**Рисунок 4.1.6 – Регистрация нового клиента**



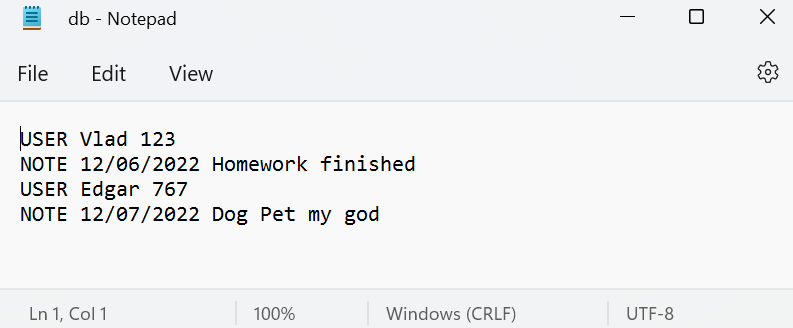
**Рисунок 4.1.7 – Текстовый файл после регистрации нового клиента**

Допустим, новый клиент хочет создать заметку. Для этого ему необходимо внести данные в поля, которые находятся слева листа с заметками, и затем нажать на кнопку “Create” (рисунок 4.1.8).



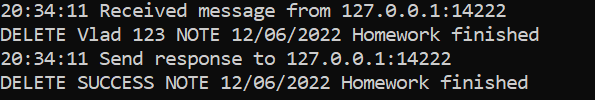
**Рисунок 4.1.8 – Создание новой заметки пользователем**

На сервере в текстовом файле заметки пользователей подписываются после пользователей (рисунок 4.1.9).

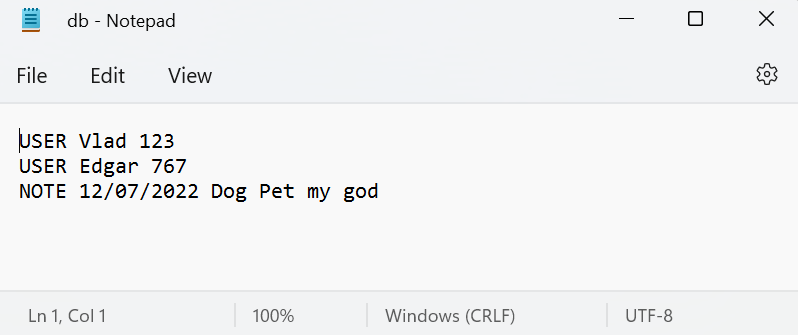


**Рисунок 4.1.9 – Текстовый файл после создания заметки**

Для того, чтобы удалить заметку, пользователю необходимо выбрать заметку для удаления и нажать на кнопку “Delete”. На стороне клиента заметка удалится, а на сервере появится запись об удалении заметки (рисунок 4.1.10-11).



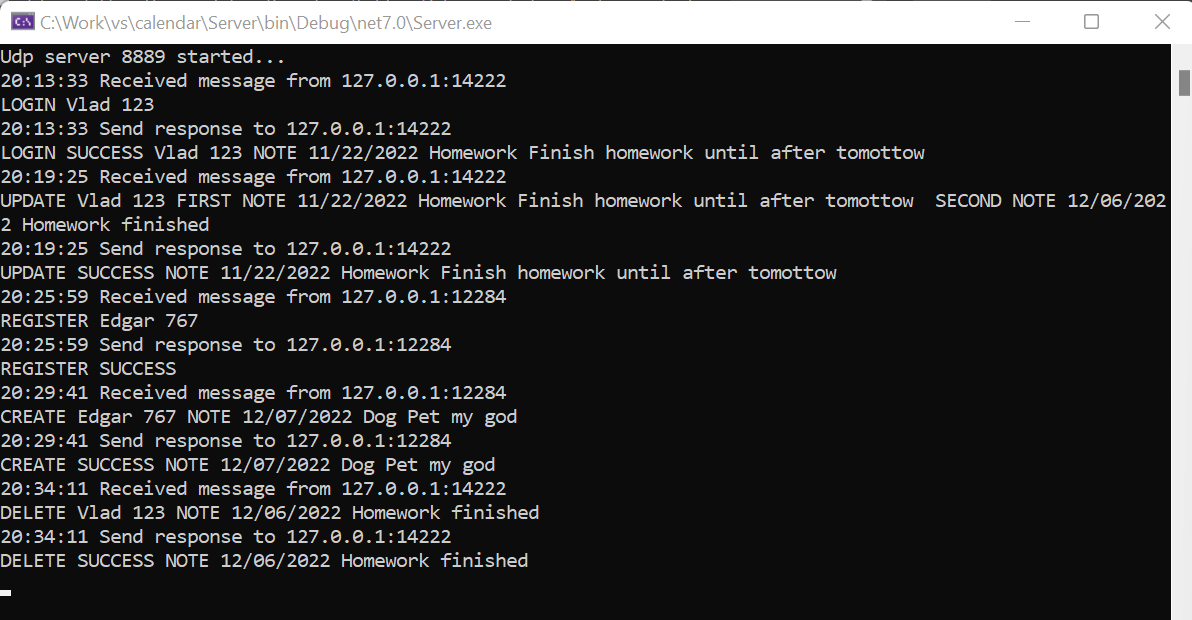
**Рисунок 4.1.10 – Удаление заметки**



**Рисунок 4.1.11 – Текстовый файл после удаления заметки Homework**

## Сервер

Сервер работает самостоятельно, отправляя ответы на запросы клиентов. Во время работы сервера, на консоль выводится информация о полученных запросах клиентов и об отправленных ответах (рисунок 4.2.1).



**Рисунок 4.2.1 – Работа сервера**

Также, во время работы, сервер, когда получает запрос от пользователя, сохраняет его ip адрес, и, после формирования ответа, отправляет ответ на тот же ip адрес. На рисунке 4.2.1 показана работа сервера с двумя пользователями одновременно.

# ЗАКЛЮЧЕНИЕ

По результатам выполнения курсовой работы можно сделать следующие основные выводы:

* Проведён анализ предметной области;
* Подробно изучены возможности C# и Windows Forms для решения поставленной задачи.
* Были описаны функции приложений сервера и клиента.
* Подробно расписана реализация сетевой части в обоих программах.
* Была написана подробная инструкция по эксплуатации для клиентов и диспетчера.
* При помощи языка программирования C# и Windows Forms создана программа, хранящая заметки пользователей на сервере.

Для реализации сетевой части использовали протокол UDP, представленный в среде разработки Visual Studio классом UdpClient.

Основная цель курсового проекта: Разработка сетевого приложения «Онлайн календарь и планировщик», была достигнута.

Можно выделить что разработанное приложение немного уступает в функциональности своим аналогам, но, возможно, будет дорабатываться в будущем.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Полное руководство по языку программирования С# 10.0 и платформе .NET 5 [Электронный ресурс] - Режим доступа: <https://metanit.com/sharp/tutorial/> – Дата доступа: 10.11.2022.
2. Руководство по программированию в Windows Forms [Электронный ресурс] – Режим доступа: <https://metanit.com/sharp/windowsforms/> – Дата доступа: 01.11.2022
3. Павловская, Т.А. C#. Программирование на языке высокого уровня: учебник для вузов / Т.А. Павловская. – СПб.: Питер, 2010.
4. Язык Си-Шарп [Электронный ресурс] – Режим доступа: <http://mycsharp.ru>. – Дата доступа: 04.11.2022
5. Методы (Руководство по программированию на C#) [электронный ресурс] – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/classes-and-structs/methods> - Дата доступа: 03.11.2022
6. Сетевые протоколы [Электронный ресурс] – Режим доступа: <https://selectel.ru/blog/network-protocols/> - Дата доступа: 07.11.2022
7. Путеводитель по органайзерам [Электронный ресурс] – Режим доступа: <https://www.ixbt.com/soft/organizers.shtml> - Дата доступа: 07.11.2022

# ПРИЛОЖЕНИЕ А

**Метод RecieveMessage приложения Сервер**

void ReceiveMessage()

{

UdpClient receiver = new UdpClient(localPort);

IPEndPoint remoteEndPoint = null;

try

{

Console.WriteLine("Udp server " + localPort.ToString() + " started...");

while (true)

{

byte[] data = receiver.Receive(ref remoteEndPoint);

remoteAddress = remoteEndPoint.Address.ToString();

remotePort = remoteEndPoint.Port;

string message = Encoding.Unicode.GetString(data);

Console.WriteLine("{0} Received message from {1}:{2}\n{3}", DateTime.Now.ToLongTimeString(), remoteAddress, remotePort, message);

buffer = data;

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

receiver.Close();

Console.WriteLine("\nreceiver closed");

}

}

**Метод SendMessage приложения Сервер**

void SendMessage()

{

UdpClient sender = new UdpClient();

try

{

while (true)

{

if (remoteAddress != "" && remotePort != -1 && buffer != new byte[65000]) // if buffer = default > returns exception

{

Thread.Sleep(20);

GetUsers();

string message = Encoding.Unicode.GetString(buffer); // client request

string[] messages = message.Split(' ', StringSplitOptions.RemoveEmptyEntries); // client request split in words

string response = "response";

switch (messages[0]) // first word

{

case "LOGIN": // LOGIN USERNAME PASSWORD

if (messages.Length == 3)

{

for (int i = 0; i < users.Count; i++)

{

if (messages[1] == users[i].Login && messages[2] == users[i].Password)

{

response = "LOGIN SUCCESS " + users[i].Login + " " + users[i].Password;

for (int ii = 0; ii < users[i].notes.Count; ii++)

{

response += " " + users[i].notes[ii].GetNote();

}

break;

}

else

{

response = "LOGIN FAIL";

}

}

}

else

{

response = "LOGIN FAIL NOT 3 WORDS";

}

break;

case "REGISTER": // REGISTER LOGIN PASSWORD

if (messages.Length == 3)

{

for (int i = 0; i < users.Count; i++)

{

if (users[i].Login == messages[1])

{

response = "REGISTER FAIL";

break;

}

}

if (response != "REGISTER FAIL")

{

File.AppendAllText(path, "\nUSER " + messages[1] + " " + messages[2]);

response = "REGISTER SUCCESS";

}

}

else

{

response = "REGISTER FAIL INCORRECT REQUEST";

}

break;

case "CREATE": // CREATE LOGIN PASSWORD NOTE date caption content

if (messages[3] != "NOTE")

{

response = "CREATE FAIL";

}

else

{

DBUpdate(message);

response = "CREATE SUCCESS " + currentNote.GetNote();

}

break;

case "UPDATE": // UPDATE LOGIN PASSWORD FIRST NOTE date caption content SECOND NOTE ... d c c

if (messages[3] != "FIRST" || message.IndexOf("SECOND") == -1)

{

response = "UPDATE FAIL";

}

else

{

DBUpdate(message);

response = "UPDATE SUCCESS " + currentNote.GetNote();

}

break;

case "DELETE": // DELETE LOGIN PASSWORD NOTE date caption content

if (messages[3] != "NOTE")

{

response = "DELETE FAIL";

}

else

{

DBUpdate(message);

response = "DELETE SUCCESS " + currentNote.GetNote();

}

break;

default:

response = "INVALID";

break;

}

byte[] data = Encoding.Unicode.GetBytes(response);

sender.Send(data, data.Length, remoteAddress, remotePort);

Console.WriteLine("{0} Send response to {1}:{2}\n{3}", DateTime.Now.ToLongTimeString(), remoteAddress, remotePort, response);

// reset and wait for the next client-request

response = "";

remoteAddress = "";

remotePort = -1;

buffer = new byte[65000];

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

sender.Close();

Console.WriteLine("\nsender closed");

}

}

# Приложение Б

**Метод RecieveMessage приложения Клиент**

void ReceiveMessage()

{

IPEndPoint remoteEndPoint = null;

try

{

while (true)

{

Thread.Sleep(20);

byte[] data = udpClient.Receive(ref remoteEndPoint);

string message = Encoding.Unicode.GetString(data); // message received from server

string[] messages = message.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries); // message split into words

// MessageBox.Show("Received message:\n" + message); // check server resoponse

switch (messages[0]) // first word

{

case "LOGIN":

if (user.Login != "0" && user.Password != "0")

{

MessageBox.Show("You are already logged in");

}

else if (messages[1] == "SUCCESS")

{

user = new User(messages[2], messages[3]);

labelUser.Invoke(delegate { labelUser.Text = "Logged in successfully as\n" + user.Login; });

for (int i = 0; i < messages.Length; i++)

{

if (messages[i] == "NOTE")

{

string content = "";

for (int ii = i + 3; ii < messages.Length; ii++)

{

if (messages[ii] == "NOTE")

{

break;

}

else

{

content += messages[ii];

content += " ";

}

}

user.notes.Add(new Note(DateTime.Parse(messages[i + 1]), messages[i + 2], content));

}

}

}

else if (message.Split(' ', StringSplitOptions.RemoveEmptyEntries)[1] == "FAIL")

{

MessageBox.Show("LOGIN FAIL");

}

else

{

MessageBox.Show("Error switch LOGIN");

}

break;

case "REGISTER":

if (messages[1] == "SUCCESS")

{

user = new User(textBoxLogin.Text, textBoxPassword.Text);

labelUser.Invoke(delegate { labelUser.Text = "Registered successfully as\n" + user.Login; });

}

else if (messages[1] == "FAIL")

{

MessageBox.Show("Failed to register (user already exists)");

}

else

{

MessageBox.Show("Error switch REGISTER");

}

break;

case "CREATE":

if (messages[1] == "SUCCESS")

{

string content = "";

for (int i = 5; i < messages.Length; i++)

{

content += messages[i] + " ";

}

user.notes.Add(new Note(DateTime.Parse(messages[3]), messages[4], content));

}

else if (messages[1] == "FAIL")

{

MessageBox.Show("Failed to create new note. (check your writing)");

}

else

{

MessageBox.Show("Error switch CREATE");

}

break;

case "UPDATE":

break;

case "DELETE":

break;

case "INVALID":

MessageBox.Show("Error incorrect request to server");

break;

default:

MessageBox.Show("Error switch DEFAULT");

break;

}

listBoxNotes.Invoke(delegate { NotesUpdate(); }); // update my notes after receiving server response

CalendarUpdate();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

finally

{

udpClient.Close();

}

}

**Метод нажатия кнопки “Log in” приложения Клиент**

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

{

string login = textBoxLogin.Text;

string password = textBoxPassword.Text;

string message = "LOGIN " + login + " " + password;

byte[] data = Encoding.Unicode.GetBytes(message);

udpClient.Send(data, data.Length, remoteAddress, remotePort);

}

**Метод нажатия кнопки “Register” приложения Клиент**

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

{

string login = textBoxLogin.Text;

string password = textBoxPassword.Text;

string message = "REGISTER " + login + " " + password;

byte[] data = Encoding.Unicode.GetBytes(message);

udpClient.Send(data, data.Length, remoteAddress, remotePort);

}

**Метод нажатия кнопки “Log out” приложения Клиент**

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

{

if (user.Login != "0" && user.Password != "0")

{

user = new User("0", "0");

NotesUpdate();

labelUser.Invoke(delegate { labelUser.Text = "Log in or register"; });

textBoxLogin.Invoke(delegate { textBoxLogin.Text = ""; });

textBoxPassword.Invoke(delegate { textBoxPassword.Text = ""; });

// clean up Note space on the left

textBoxNoteDate.Invoke(delegate { textBoxNoteDate.Text = ""; });

textBoxNoteCaption.Invoke(delegate { textBoxNoteCaption.Text = ""; });

textBoxNoteContent.Invoke(delegate { textBoxNoteContent.Text = ""; });

}

}

**Метод нажатия кнопки “Create” приложения Клиент**

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

{

try

{

if (textBoxNoteCaption.Text.IndexOf(' ') != -1)

{

MessageBox.Show("Note's caption can only be one word (delete gaps)");

}

else

{

currentNote = new Note(DateTime.Parse(textBoxNoteDate.Text), textBoxNoteCaption.Text, textBoxNoteContent.Text);

string message = "CREATE " + user.Login + " " + user.Password + " " + currentNote.GetNote();

byte[] data = Encoding.Unicode.GetBytes(message);

udpClient.Send(data, data.Length, remoteAddress, remotePort);

// clean up Note space on the left

textBoxNoteDate.Invoke(delegate { textBoxNoteDate.Text = ""; });

textBoxNoteCaption.Invoke(delegate { textBoxNoteCaption.Text = ""; });

textBoxNoteContent.Invoke(delegate { textBoxNoteContent.Text = ""; });

}

}

catch (Exception ex)

{

MessageBox.Show("Failed to create new note. Check date \n\n" + ex.Message);

}

}

**Метод нажатия кнопки “Update” приложения Клиент**

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

{

if (listBoxNotes.SelectedItem != null && listBoxNotes.SelectedIndex != -1)

{

currentNote = new Note(DateTime.Parse(textBoxNoteDate.Text), textBoxNoteCaption.Text, textBoxNoteContent.Text);

// update on client

for (int i = 0; i < user.notes.Count; i++)

{

if (user.notes[i].GetNote().Substring(5) == listBoxNotes.SelectedItem.ToString())

{

// update on server

if (user.Login != "0" && user.Password != "0")

{

string message = "UPDATE " + user.Login + " " + user.Password + " FIRST " + user.notes[i].GetNote() + " SECOND " + currentNote.GetNote();

byte[] data = Encoding.Unicode.GetBytes(message);

udpClient.Send(data, data.Length, remoteAddress, remotePort);

}

user.notes[i] = currentNote;

break;

}

}

}

NotesUpdate();

}

**Метод нажатия кнопки “Delete” приложения Клиент**

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

{

if (listBoxNotes.SelectedItem != null && listBoxNotes.SelectedIndex != -1)

{

// delete on server

string message = "DELETE " + user.Login + " " + user.Password + " " + currentNote.GetNote();

byte[] data = Encoding.Unicode.GetBytes(message);

udpClient.Send(data, data.Length, remoteAddress, remotePort);

// delete on client

for (int i = 0; i < user.notes.Count; i++)

{

if (currentNote == user.notes[i])

{

user.notes.RemoveAt(i);

break;

}

}

textBoxNoteDate.Invoke(delegate { textBoxNoteDate.Text = ""; });

textBoxNoteCaption.Invoke(delegate { textBoxNoteCaption.Text = ""; });

textBoxNoteContent.Invoke(delegate { textBoxNoteContent.Text = ""; });

}

NotesUpdate();

}

# Приложение В

**Класс User (пользователь)**

internal class User

{

public User(string log, string pass)

{

login = log;

password = pass;

}

string login;

public string Login

{

get { return login; }

set { login = value; }

}

string password;

public string Password

{

get { return password; }

set { password = value; }

}

public List<Note> notes = new List<Note>() { };

public string GetUser()

{

return "USER " + login + " " + password;

}

}

# Приложение Г

**Класс Note (заметка)**

internal class Note

{

public Note(DateTime da, string capt, string cont)

{

date = da;

caption = capt;

content = cont;

}

DateTime date;

public DateTime Date

{

get { return date; }

set { date = value; }

}

string caption;

public string Caption

{

get { return caption; }

set { caption = value; }

}

string content;

public string Content

{

get { return content; }

set { content = value; }

}

public string GetNote()

{

return "NOTE " + date.ToString("MM/dd/yyyy") + " " + caption + " " + content;

}

}