МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра програмних систем і технологій

Дисципліна
« Об’єктно-орієнтоване конструювання програм »

Лабораторна робота № 7
Варіант №10

Виконав:	Сергійчук Дмитро Віталійович	**Перевірила**:	Зубик Людмила Володимирівна
Група	ІПЗ-21-2	Дата перевірки	
Форма навчання	денна	Оцінка	
Спеціальність	121		
2024			

Тема: Шаблони проектування. Архітектурні шаблони.

Мета: Реалізація програми, яка здатна виконувати операції паралельно. Порівняння паралельного та послідовного виконання завдань.

Завдання 1: Розробити застосунок для організації корпоративного спілкування. Передбачити використання в ньому засобів з функціями групових чатів, відео-дзвінків та конференцій.

Забезпечити такі можливості сервісу:

- проведення онлайн-зустрічей з відео-супроводом та можливістю участі до 30 користувачів;
- функцію демонстрації матеріалів під час зустрічі;
- планування зустрічей заздалегідь;
- можливість запрошувати учасників;
- опцію запису зустрічей;
- організацію загальних та приватних чатів для листування та обміну файлами.

Структура проєкту, опис та код:

Сервер:

Серверну частину було реалізовано у двох частинах:

1. Клас UdpServer, що є надбудовою над класом UdpClient. Даний клас надає основні методи для роботи сервера з протоколом Udp.

   Код:

   - Поля та властивості:

```
private CancellationTokenSource _cts;

private readonly List<UdpServerClient> _udpClients;
private readonly SemaphoreSlim _semaphore;

public UdpClient Instance { get; private set; }
public IPEndPoint IPEndPoint { get; private set; }
```

   - Метод, який читає повідомлення від клієнтів:

```
public async Task<UdpReceiveResult> ReceiveAsync(CancellationToken token)
{
    return await Instance.ReceiveAsync(token);
}
```

   - Метод, який забезпечує роботу сервера у асинхронному режимі, де для кожного клієнта є окремий потік:

```
public async void HandleRequest(UdpReceiveResult res, Action<UdpReceiveResult>
function)
{
    var udpClinent = _udpClients.Find(x => Equals(x.IPEndPoint,
res.RemoteEndPoint));
```

```
    if (udpClinent != null)
    {
        await _semaphore.WaitAsync();

        await Task.Run(async () =>
        {
            function(res);

            _semaphore.Release();
        });

        udpClinent.UpdateLastRequestTime();
    }
    else
    {
        var newClient = new UdpServerClient(res.RemoteEndPoint);
        _udpClients.Add(newClient);

        await Task.Run(() =>
        {
            function(res);
        });

        newClient.UpdateLastRequestTime();
    }
}
```

- Методи, для відправки повідомлень одному / багатьом клієнтам:

```
public async Task SendAsync(byte[] data, IPEndPoint remote)
{
    await Instance.SendAsync(data, data.Length, remote);
}

public async Task BroadcastAsync(byte[] data, IPEndPoint[] remotes, IPEndPoint
sender, bool isSendToSender = false)
{
    foreach (var remote in remotes)
    {
        if (isSendToSender || (!isSendToSender && !Equals(remote, sender)))
        {
            await Instance.SendAsync(data, data.Length, remote);
        }
    }
}
```

- Метод, який перевіряє наявність клієнтів за тим, як довго вони не відсилали повідомлень. Якщо протягом 1 хвилини не було ні одного запиту, то клієнт видаляється зі списку і його наступний запит буде оброблятися у новому потоці:

```
private async void CheckClients(CancellationToken token)
{
    while(!token.IsCancellationRequested)
    {
        await Task.Delay(60000);

        for (var i = 0; i < _udpClients.Count; i++)
        {
            var time = DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond;
            if (time - _udpClients[i].LastRequest > 60000)
            {
                _udpClients.RemoveAt(i);
                i--;
            }
        }
```

```
    }
}
```

- Допоміжний клас, для зручної роботи з клієнтами:

```
class UdpServerClient
{
    public IPEndPoint IPEndPoint { get; private set; }
    public long LastRequest { get; private set; }

    public UdpServerClient(IPEndPoint iPEndPoint)
    {
        IPEndPoint = iPEndPoint;
        LastRequest = DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond;
    }

    public void UpdateLastRequestTime()
    {
        LastRequest = DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond;
    }
}
```

2. Клас Server, який виконує основні функції сервера даного застосунку. Є надбудовою над класом UdpServer, який надає базові методи. Зв'язок з клієнтами реалізовано у вигляді команд, де клієнт та сервер передають дані, зі спеціальним словом, за яким визначається подальша обробка запиту та відповідь.

   Код:

   - Поля та властивості:

```
private static object _locker = new object();
private static CancellationTokenSource _cancellationTokenSource = new
CancellationTokenSource();
public static bool IsWorking { get; private set; }

public static UdpServer Instance { get; private set; }
public static List<MeetingData> Meetings { get; private set; }
```

   - Метод, що перевіряє наявність клієнтів:

```
private static async void PingClients(CancellationToken token)
{
    try
    {
        byte[] data;

        using (var ms = new MemoryStream(8))
        using (var bw = new BinaryWriter(ms))
        {
            bw.Write("PING");
            data = ms.ToArray();
        }

        while (!token.IsCancellationRequested)
        {
            for (int i = 0; i < Meetings.Count; i++)
            {
                for (int j = 0; j < Meetings[i].Clients.Count; j++)
                {
                    if (Meetings[i].Clients[j].CheckLastPong())
                    {
                        await Instance.SendAsync(data,
Meetings[i].Clients[j].IpEndPoint);
```

```
                    }
                    else
                    {
                        var client = Meetings[i].Clients[j];
                        Meetings[i].Clients.Remove(client);
                        if (client.IsShareScreen)
                        {
                            Meetings[i].StopShare();
                            using (var response_ms = new MemoryStream(8))
                            using (var bw = new BinaryWriter(response_ms))
                            {
                                bw.Write("SHARE_END");

                                await
Instance.BroadcastAsync(response_ms.ToArray(), Meetings[i].Clients.Select(x =>
x.IpEndPoint).ToArray(), null);
                            }
                        }

                        if (Meetings[i].Clients.Count == 0)
                        {
                            FileManager.DeleteMeetingCatalog(Meetings[i].Id);
                            Meetings.RemoveAt(i);
                            i--;
                            break;
                        }
                        else
                        {
                            using (var broadcast_ms = new MemoryStream(64))
                            using (var bw = new BinaryWriter(broadcast_ms))
                            {
                                bw.Write("UPDATE_LIST");
                                bw.Write(Meetings[i].Clients.Count);

                                for (var k = 0; k < Meetings[i].Clients.Count; k++)
                                {
                                    bw.Write(Meetings[i].Clients[k].Name);

bw.Write(Meetings[i].Clients[k].IpEndPoint.Address.GetAddressBytes());
bw.Write(Meetings[i].Clients[k].IpEndPoint.Port);
                                }

                                await
Instance.BroadcastAsync(broadcast_ms.ToArray(), Meetings[i].Clients.Select(x =>
x.IpEndPoint).ToArray(), null, true);
                            }
                            j--;
                        }
                    }
                }
            }

            await Task.Delay(5000, token);
        }
    }
    catch { }
}
```

- Частина метода, що обробляє запити клієнтів:

```
    private static async void ProcessRequest(UdpReceiveResult request)
    {
        try
        {
            var clientEP = request.RemoteEndPoint;

            using (var ms = new MemoryStream(request.Buffer))
            using (var br = new BinaryReader(ms))
```

```
{
    var method = br.ReadString();

    if (method == "CREATE")
    {
        var name = br.ReadString();

        var meeting = new MeetingData();
        Meetings.Add(meeting);

        FileManager.CreateMeetingCatalog(meeting.Id);

        meeting.AddClient(new MeetingUser(clientEP, name));

        using (var response_ms = new MemoryStream(16))
        using (var bw = new BinaryWriter(response_ms))
        {
            bw.Write("CONNECTED");
            bw.Write(meeting.Id);

            await Instance.SendAsync(response_ms.ToArray(), clientEP);
        }
    }
```

Клієнт:

Клієнтська частина реалізована у вигляді класу Client, що використовує клас UdpClient, для зв'язку з сервером. Даний клас надсилає запити до сервера та прослуховує відповіді. Також даний клас зв'язаний подіями з іншими класами, що відповідають за графіку.

Код:

- Події, поля та властивості:

```
public static event Action? Connected;
public static event Action? ClientListUpdated;
public static event Action<MeetingParticipant>? AnotherCameraStarted;
public static event Action<MeetingParticipant>? AnotherCameraUpdated;
public static event Action<MeetingParticipant>? AnotherCameraStopped;
public static event Action<MeetingParticipant>? AnotherAudioStarted;
public static event Action<MeetingParticipant>? AnotherAudioStopped;
public static event Action<bool>? ShareResultReceived;
public static event Action? AnotherShareStarted;
public static event Action<MeetingParticipant> AnotherShareUpdated;
public static event Action? AnotherShareStopped;
public static event Action<string, string, string>? MessageReceived;
public static event Action<string, long, string, string, string>? FileUploaded;
public static event Action<string, byte[]>? FilePartDownloaded;


public static readonly string ScheduleFile = ".\\.schedule.txt";

public static readonly IPEndPoint REMOTE_END_POINT;
public static UdpClient Instance { get; private set; }
public static string LocalName { get; private set; }
public static readonly IPEndPoint LocalIpEndPoint;
private static long _serverLastPong;

public static long MeetingId { get; private set; }
public static List<MeetingParticipant> Participants { get; private set; }
```

- Метод, що перевіряє наявність сервера:

```
private static async void PingRemote()
```

```
{
    try
    {
        byte[] data;

        using (var ms = new MemoryStream(8))
        using (var bw = new BinaryWriter(ms))
        {
            bw.Write("PING");
            data = ms.ToArray();
        }

        while (true)
        {
            if (DateTime.Now.Ticks / TimeSpan.TicksPerMillisecond - _serverLastPong
> 21000)
            {
                MessageBox.Show("Server is not responding", "Error",
MessageBoxButton.OK, MessageBoxImage.Error);
                Environment.Exit(0);
            }
            else
            {
                await Instance.SendAsync(data, data.Length, REMOTE_END_POINT);
            }

            await Task.Delay(4000);
        }
    }
    catch { }
}
```

- Частина метода, що обробляє повідомлення сервера:

```
private static async void ListenServer()
{
    while (true)
    {
        try
        {
            var response = await Instance.ReceiveAsync();

            using (var ms = new MemoryStream(response.Buffer))
            using (var br = new BinaryReader(ms))
            {
                var method = br.ReadString();

                if (method == "CONNECTED")
                {
                    MeetingId = br.ReadInt64();

                    Participants = new List<MeetingParticipant>() { new
MeetingParticipant(LocalName, LocalIpEndPoint) };

                    Connected?.Invoke();
                }
```

- Для передачі будь-яких запитів використовуються відповідні методи, вони є однотипними, тому не потребують демонстрації. А для передачі даних, як кадр з вебкамери, кадр з екрану, при демонстрації, або файл, використовуються спеціальні методи, які передають дані по частинам, через обмеження в 2^16 байт при передачі даних. Приклад такого метода на основі передачі кадру з демонстрації екрана:

```
public static void SendShareFrame(Bitmap bitmap)
```

```
{
    using (var memoryStream = new MemoryStream())
    {
        bitmap.Save(memoryStream, ImageFormat.Jpeg);

        byte[] imageData = memoryStream.ToArray();

        int bufferSize = 32768;
        int chunks = (int)Math.Ceiling((double)imageData.Length / bufferSize);

        for (var i = 0; i < chunks; i++)
        {
            using (var ms2 = new MemoryStream(new byte[32900]))
            using (var bw = new BinaryWriter(ms2))
            {
                if (i == 0)
                {
                    bw.Write("SHARE_FRAME_FIRST");
                    bw.Write(imageData.Length);
                }
                else if (i + 1 == chunks)
                {
                    bw.Write("SHARE_FRAME_LAST");
                }
                else
                {
                    bw.Write("SHARE_FRAME");
                }

                bw.Write(MeetingId);
                bw.Write(i);

                int bytesToSend = Math.Min(bufferSize, imageData.Length - i *
bufferSize);
                byte[] chunkData = new byte[bytesToSend];
                Buffer.BlockCopy(imageData, i * bufferSize, chunkData, 0,
bytesToSend);

                bw.Write(bytesToSend);
                bw.Write(chunkData);

                Instance.Send(ms2.ToArray(), (int)ms2.Length, REMOTE_END_POINT);
            }
        }
    }
}
```

Ще одним з компонентів клієнтського застосунку є клас MeetingParticipant, він зберігає дані кожного учасника зустрічі та виконує функцію зберігання кадру вебкамери, кадру демонстрації:

- Поля та властивості:

```
public IPEndPoint IpEndPoint { get; private set; }
public string Name { get; private set; }
public bool IsUsingCamera { get; private set; } = false;
public bool IsUsingAudio { get; private set; } = false;

public List<(byte[], int)> Frame { get; private set; } = new List<(byte[], int)>();
private byte[] _frameBuffer;

public List<(byte[], int)> ShareFrame { get; private set; } = new List<(byte[],
int)>();
private byte[] _shareFrameBuffer;
```

- Методи, які створюють та з’єднують по частинах кадр вебкамери, для подальшого виведення на екран. Для кадру з демонстрації екрана все виглядає аналогічно:

```
public void CreateFrame(int length)
{
    Frame = new List<(byte[], int)>();
    _frameBuffer = new byte[length];
}
public void AddFrameData(byte[] array, int index)
{
    Frame.Add(new(array, index));
}
public BitmapImage GetFrame()
{
    for (var i = 0; i < Frame.Count; i++)
    {
        Buffer.BlockCopy(Frame[i].Item1, 0, _frameBuffer, Frame[i].Item2 * 32768,
Frame[i].Item1.Length);
    }

    using (MemoryStream memoryStream = new MemoryStream(_frameBuffer))
    {

        BitmapImage bitmapImage = new BitmapImage();
        bitmapImage.BeginInit();
        bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
        bitmapImage.StreamSource = memoryStream;
        bitmapImage.EndInit();

        bitmapImage.Freeze();
        return bitmapImage;
    }
}
```

Також ще є декілька класів, які необхідні для роботи застосунку:

1. AudioPlayer – клас, що відтворює звук мікрофона інших учасників зустрічі:

   Код:

```
internal class AudioPlayer
{
    private readonly WaveOutEvent _waveOut;
    private readonly MemoryStream _stream;

    public AudioPlayer()
    {
        _waveOut = new WaveOutEvent();
        _stream = new MemoryStream();
        _waveOut.Init(new RawSourceWaveStream(_stream, new WaveFormat(44100, 16, 1)));
    }

    public void Play(byte[] buffer)
    {
        _waveOut.Play();
        _stream.SetLength(0);
        _stream.Write(buffer, 0, buffer.Length);
        _stream.Position = 0;
    }

    public void Stop()
    {
        _waveOut.Stop();
    }
```

```
    public static AudioPlayer Instanse { get; set; }

    static AudioPlayer()
    {
        Instanse = new AudioPlayer();
    }
}
```

2. ProgramManager – клас, який керує вікнами та сторінками застосунку:

```
internal class ProgramManager
 {
     public event Action<PageType> Navigated;
     public event Action<MeetingPage> ShareStarted;
     public event Action ShareFinished;

     public static ProgramManager Instance { get; private set; }

     static ProgramManager()
     {
         Instance = new ProgramManager();
     }

     public void Navigate(PageType pageType)
     {
         Navigated?.Invoke(pageType);
     }
     public void StartShare(MeetingPage page)
     {
         ShareStarted?.Invoke(page);
     }
     public void StopShare()
     {
         ShareFinished?.Invoke();
     }
 }

 enum PageType
 {
     MainPage = 0,
     MeetingPage = 1,
 }
```

3. Recorder – клас, що записує зустріч (весь екран) зустрічі за допомогою програми ffmpeg:

```
internal class Recorder
{
    private const string _ffmpegPath = @"D:\Program Files\ffmpeg\bin\ffmpeg.exe";
    private const string args = "-f dshow -i audio=\"Stereo Mix (Realtek High
Definition Audio)\" -f gdigrab -framerate 30 -draw_mouse 0 -i desktop -c:v libx264 -
preset ultrafast -tune zerolatency -pix_fmt yuv420p -c:a aac";

    private readonly static string _outputPath =
Environment.GetFolderPath(Environment.SpecialFolder.MyVideos);
    private readonly static string _tempName;

    private static Process _ffmpegProcess;

    static Recorder()
    {
        _tempName = $"{DateTime.Now.Ticks}.mp4";

        _ffmpegProcess = new Process();
        _ffmpegProcess.StartInfo.FileName = _ffmpegPath;
        _ffmpegProcess.StartInfo.Arguments = $"{args} {_outputPath}\\{_tempName}";
        _ffmpegProcess.StartInfo.UseShellExecute = false;
```

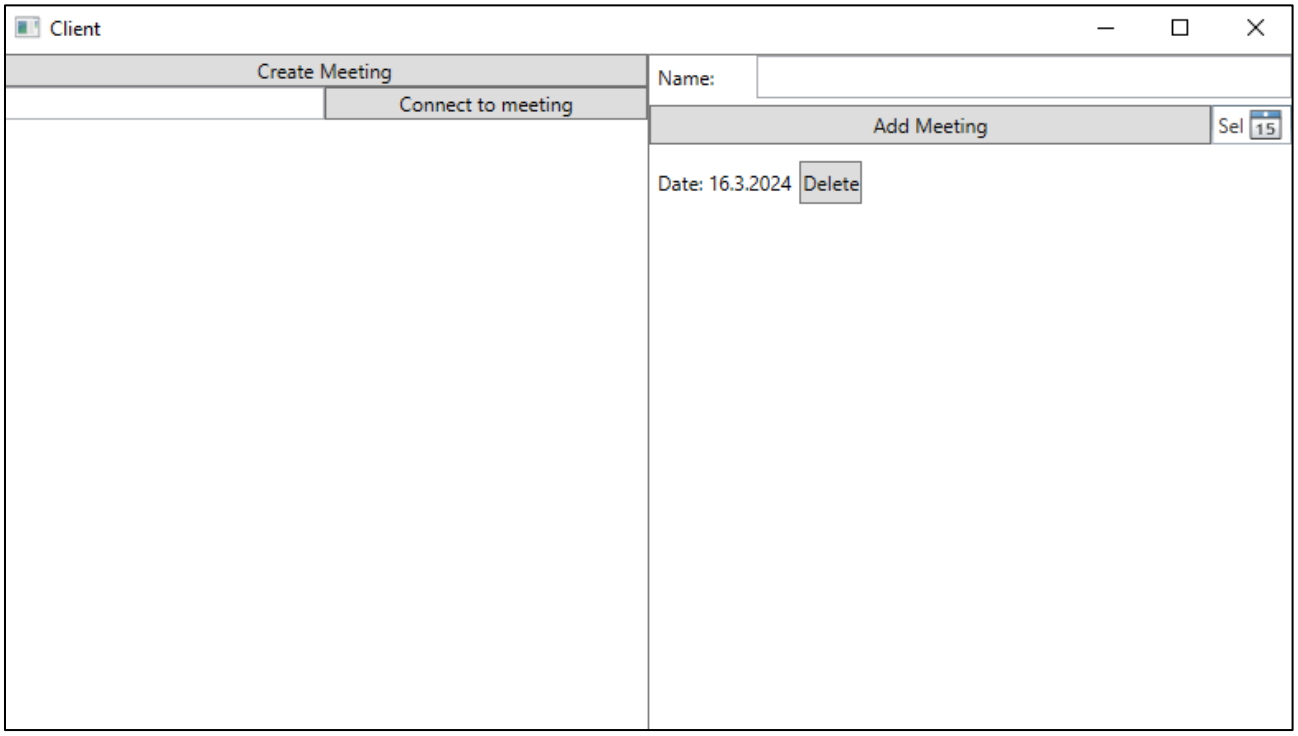
```
        _ffmpegProcess.StartInfo.RedirectStandardInput = true;
        _ffmpegProcess.StartInfo.CreateNoWindow = true;
    }

    public static void StartRecording()
    {
        _ffmpegProcess.Start();
    }

    public static void StopRecording()
    {
        using (var writer = _ffmpegProcess.StandardInput)
            if (writer.BaseStream.CanWrite)
                writer.WriteLine("q");

        _ffmpegProcess.WaitForExit();

        var source = $"{_outputPath}\\{_tempName}";

        if (File.Exists(source))
        {
            var fi = new FileInfo(source);

            var newName = $"Record_{DateTime.Now.ToString("yyyy-MM-dd HH-mm-
ss")}.mp4";
            var dest = $"{_outputPath}\\{newName}";

            fi.MoveTo(dest);
        }
    }
}
```

4. ScreenShot – клас, що робить знімок екрану для демонстарції:

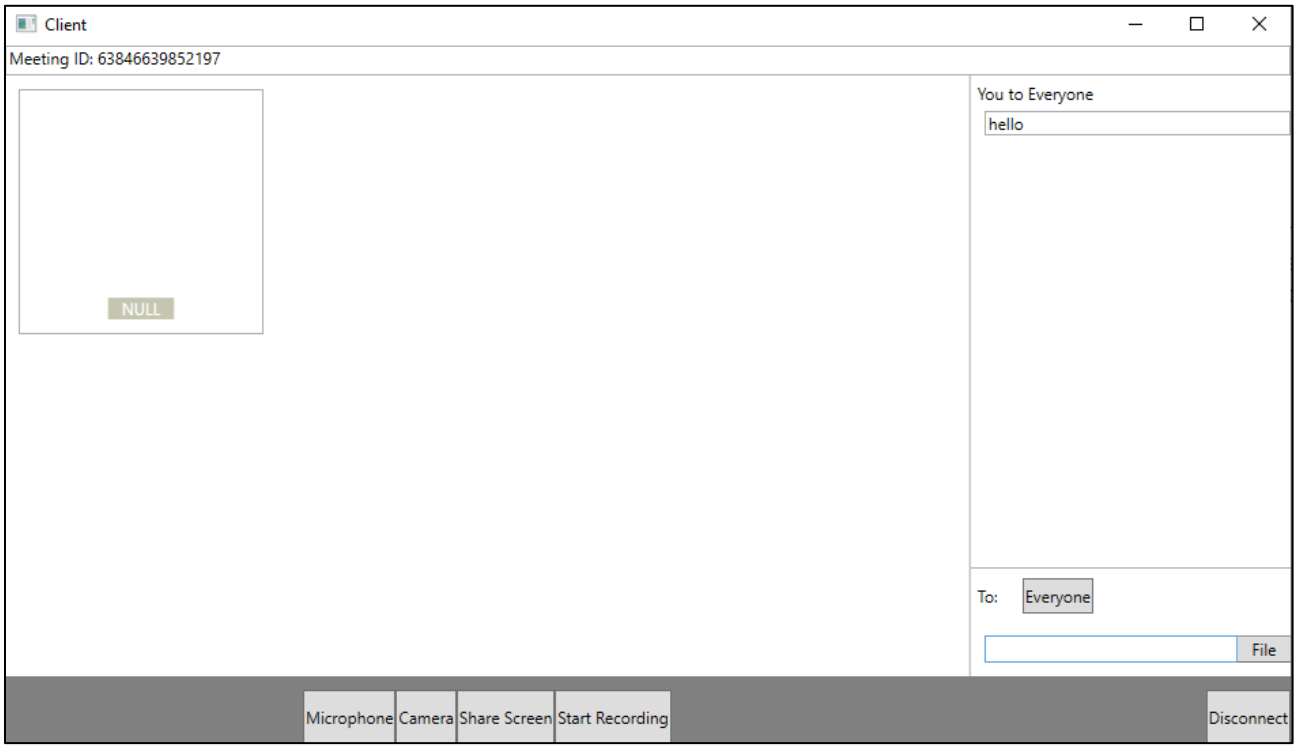
```
internal class ScreenShot
{
    public static Bitmap GetFullScreen(int width, int height)
    {
        var bmp = new Bitmap(width, height, PixelFormat.Format32bppArgb);

        using (var g = Graphics.FromImage(bmp))
        {
            g.CopyFromScreen(Point.Empty, Point.Empty, new Size(width, height));

            var rBmp = new Bitmap(1280, 720);

            using (var g2 = Graphics.FromImage(rBmp))
            {
                g2.DrawImage(bmp, new Rectangle(0, 0, 1280, 720));
            }

            return rBmp;
        }
    }
}
```

Останнім компонентом клієнтського застосунку є графічна частина. Вона має 3 частини:

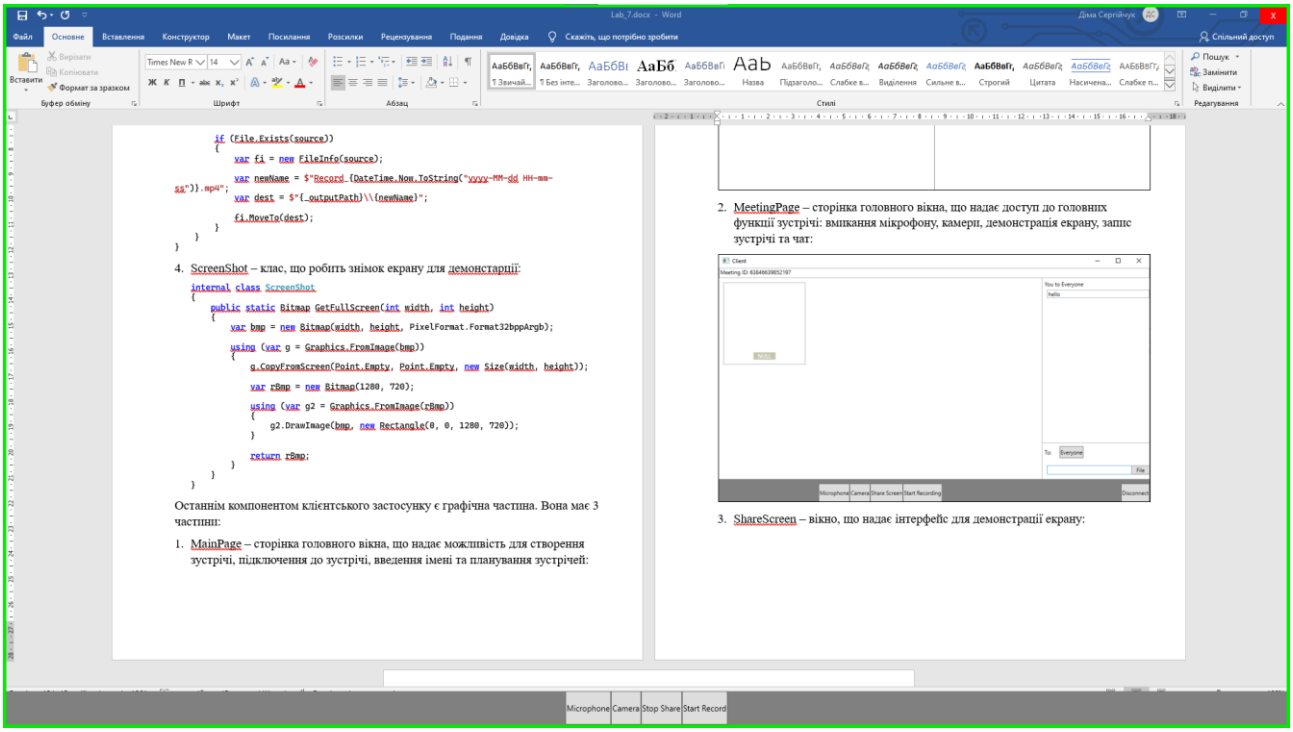
1. MainPage – сторінка головного вікна, що надає можливість для створення зустрічі, підключення до зустрічі, введення імені та планування зустрічей:



2. MeetingPage – сторінка головного вікна, що надає доступ до головних функції зустрічі: вмикання мікрофону, камери, демонстрація екрану, запис зустрічі та чат:



3. ShareScreen – вікно, що надає інтерфейс для демонстрації екрану:



Приклад демонстрації екрану на даному документі

Висновки:

Для створення даного проєкту було використано мову програмування C# та Windows Presentation Foundation (WPF). Проєкт реалізовано у двох застосунках: сервер та клієнт. Було реалізовано всі пункти завдання.

Для зв’язку між клієнтами та сервером було обрано протокол UDP, який надає можливість швидкої передачі, великих за обсягом, даних. Через відсутність двостороннього зв’язку у Udp, було додано відповідні методи для клієнта та сервера, які перевіряють наявність зв’язку. Для підвищення швидкодії роботи сервера, було реалізовано асинхронну обробку запитів для окремих клієнтів.

Роботу з мікрофоном було здійснено за допомогою бібліотеки NAudio. Для отримання зображення з вебкамери використовувалася бібліотека AForge.NET.

Запис зустрічі виконаний у вигляді запису екрану за допомогою програми ffmpeg. Таким чином на відео може потрапити матеріал, що не відноситься до зустрічі, але таке рішення було обрано, як найкраще. Спроби реалізувати запис, лише захопленням вікна зустрічі виявилися неефективними, через відсутність нормальних бібліотек для цього та через повільний запис кожного кадру.

Можливість запрошення учасників реалізовано за допомогою унікальних id кожної зустрічі, які кожен з учасників може поширити іншим людям.

Функція демонстрації матеріалу реалізована шляхом зміни вікна на «рамку» та передачу іншим учасникам знімок екрану.

Планування зустрічей заздалегідь реалізовано у вигляді списку з датою, коли даний користувач, має провести зустріч. Додавання та видалення записів у списку проводиться користувачем.

Організація загального та приватного чату реалізовано таким чином, що клієнт обирає кому відсилати повідомлення – всім, чи конкретній людині (за іменем). Розділення на приватні чати та загальний – не існує, всі повідомлення знаходяться в одному списку, але мають надпис, який вказує від кого і кому прийшло повідомлення. Тому приватні повідомлення показуються разом з публічними, але вони присутні лише у того учасника, якому вони були адресовані.