Gliwice 29 czerwca 2017r.

# Systemy interaktywne i multimedialne

## Dokument projektowy

## Multiplekser kamer/system CCTV

Skład sekcji:
Adam Krawiec
Adam Leks
Grzegorz Spakowski
Dragon Tytus
Michał Slabik
Miłosz Skubis

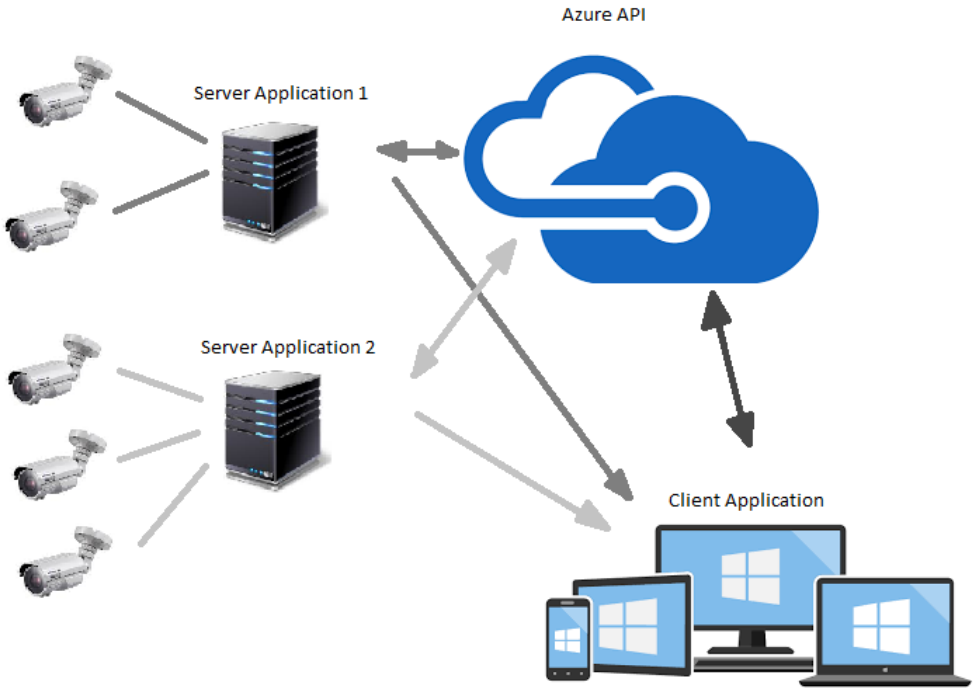
1. Opis projektu

Naszym zadaniem było stworzenie systemu do obsługi kamer, zintegrowanie go z daną platformą oraz implementacja aplikacji służącej do wyświetlania strumiena/i video. Z powodów braku dostępności skorzystaliśmy z kamer internetowych, wbudowanych w komputery oraz urządzeń tego typu podłączanych przy pomocy interfejsu USB.

Finalnie udało nam się stworzyć system oparty na trzech warstwach. Pierwszą częścią systemu stanowi w aplikacja serwer, następną aplikacja kliencka, a trzecią warstwą jest API, serwis na platformie Azure.

2. Specyfikacja zewnętrzna

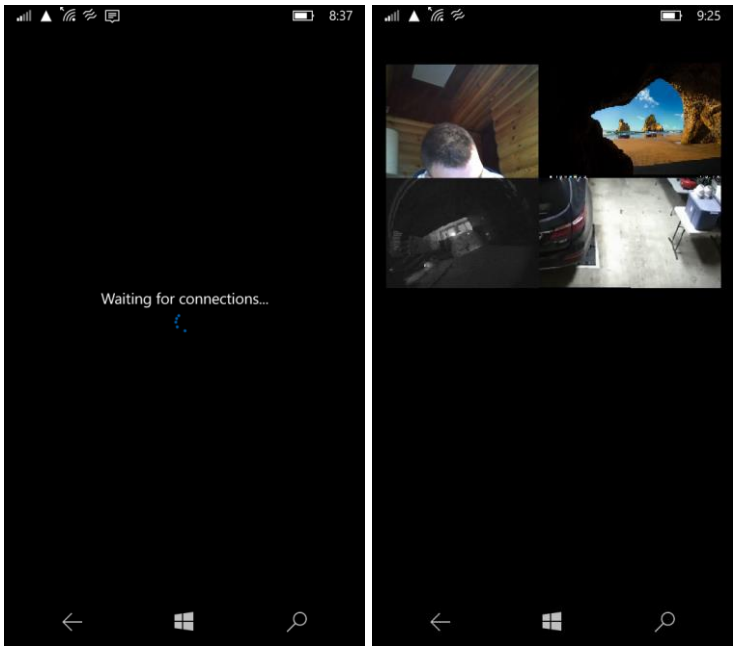
Oto krótki schemat działania systemu ułatwiający rozumienie jego działania w późniejszym opisie:




Aplikacją strumieniującą obraz video jest program zaimplementowany w technologii WPF dla systemów Windows. Działanie programu polega na zebraniu dostępnych urządzeń video oraz utworzenie odpowiednich połączeń z serwerem, aby wysyłać obraz. Interfejs użytkownika jest minimalny ze względu na brak potrzeby angażowania użytkownika w działanie serwera. Gdy włączony zostanie proces programu następuje wysłanie serii zapytań do serwera o dostępne dane, niezbędne do połączenia. Gdy program otrzyma dane może strumieniować video dla każdej kamery, która jest doń podłączona.

Aplikacja Client została zaimplementowana jako program platformy Windows Universal. Ze względu na upór członków sekcji decyzja ta wpłynęła negatywnie na czas poszukiwań dostępnych i łatwych w implementacji rozwiązań służących strumieniowaniu usług video za względu na niską popularność technologii oraz niewielkie wsparcie ze strony Microsoft.

Video App to nazwa aplikacji na urządzeniu z systemem Windows 10 stanowiąca program kliencki. Należy włączyć aplikację i czekać aż sam pojawi się obraz z kamer, które strumieniuje Server. Aplikacja przy włączeniu wysyła do serwisu listę dostępnych portów oraz dane IP. Na poniższych zrzutach widać aplikację po włączeniu oraz po nadejściu 4 połączeń.




Aby uruchomić aplikację serwer należy wejść w folder z plikami programu oraz uruchomić plik o nazwie 'CamTest.exe'.
By uruchomić aplikację klient należy zainstalować program poprzez Visual Studio w trybie debugowania lub zainstalować lokalnie plik na urządzeniu uprzednio włączając 'tryb developerski'.

## 3. Specyfikacja wewnętrzna

- Server po uruchomieniu korzysta z bibliotek AForge, RestSharp.
  Na początku zbiera listę urządzeń video podłączonych do komputera, następnie wysyła zapytania przy pomocy RestClient w celu otrzymania danych połączenia takich jak porty oraz IP.
  Następnie następuje połączenie oraz rozpoczęcie strumieniowania obrazów przy pomocy Socketów.

Metoda odpowaidająca za pobranie danych do połączenia:
`public async Task<ConnectionData> GetConnectionData()`
Jeżeli zaptanie zwróciło brak danych połączenie nie zostaje ustanowione.

Metoda dopowaidająca za wysyłanie obrazu:

```
private void VideoDeviceOnNewFrame(object sender, NewFrameEventArgs eventArgs)
{
    try
    {
        var stream = new MemoryStream();
        eventArgs.Frame.SaveJpeg(stream, 50);
        var imageBytes = stream.GetBuffer();
        Application.Current.Dispatcher.Invoke(() =>
        {
            ImageControl.Source = Helpers.ByteToImageSource(imageBytes);
        });
        if (!_ifConnected) return;
        _frameInBytes = imageBytes;
        Socket.Send(new List<ArraySegment<byte>>
        {
            new ArraySegment<byte>(
                BitConverter.GetBytes(_frameInBytes.Length))
        });
        Socket.Send(new List<ArraySegment<byte>>
        {
            new ArraySegment<byte>(_frameInBytes)
        });
    }
    catch(Exception ex)
    {
        Debug.WriteLine("Error on sending frame" + ex);

    }
}
```

Najpierw bitmapa otrzymana z API kamery zostaje skompresowana do formatu Jpeg oraz zmniejszona do potrzebnych rozmiarów przy pomocy metod pomocniczych i rozszerzeń klasy ImageSource.
Następnie przy użyciu głównego wątku UI zostaje także wyświetlony dany obraz. Kolejno zostaje wywołana metoda odpowiadająca za wysłanie konwersje danych oraz wysłanie ich przy użyciu protokołu komunikacyjnego.

Protokół wysyłania nie jest nad wyraz skomplikowany, najpierw zostaje wysłana liczba bajtów do odczytania, a następnie segment bajtów zawierających obraz. Jedna aplikacja serwer może strumieniować wiele kamer w jednym czasie.

- API odpowiadające za udostępnianie danych do połączenia działa w ramach usługi na platformie Azure. Interfejs posiada kilka prostych zapytań. Najważniejszymi z nich jest odpytanie o dostępne porty oraz adres IP oraz potwierdzenie zajęcia portu przez dane urządzenie.

Oto lista metod, które obsługuje API:

- `public async Task<bool> PostIpData(string ip)`
- `public async Task<ConnectionData> GetConnectionData()`
- `public async Task<bool> UpdatePortStatus(string port, string status)`

- Aplikacja Klient – Zaimpslementowana dla platformy Windows Universal. Można ją uruchomić na urządzeniu mobilnym z systemem Windows 10 Mobile oraz na tablecie, bądź PC z systemem Windows 10. Przy uruchomieniu wysyła do serwisu, używając metody POST PostIpData komunikując dostępnoś oraz uruchomienie procesu nasłuchiwania. Ze względów organizacyjnych aplikacja pozwala na jednoczesne odbieranie strumieni sześciu kamer. Dla zwięzłości rozwiązania użyto klasy StreamSocketListener, oraz DataReader, jako pomocnika przy czytaniu danych z Socketa.
  Podczas nadejścia połączenia wykonana zostaje metoda OnConnected – przypisany zostaje w niej obiekt do czytania bajtów z Socketu oraz aktualizuje się w niej status dostępnych połączeń w serwisie.

  Oto kod w C# odpowiadający za pętle odczytywania obrazów z Socketu:

```
while (true)
{
    try
    {
        await _dataReader.LoadAsync(4);
        _dataReader.ReadBytes(frameSizeInBytes);

        var frameLengthInt =
            BitConverter.ToUInt32(frameSizeInBytes, 0);
        Debug.WriteLine(frameLengthInt.ToString());

        var frame = new byte[frameLengthInt];

        await _dataReader.LoadAsync(frameLengthInt);
        _dataReader.ReadBytes(frame);

        Debug.WriteLine(frameLengthInt + "   " + frame.Length);
        await
            CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(
                CoreDispatcherPriority.Normal,async() =>
            {
                ImageSource = await Utlis.ConvertBytesToBitmapImage(frame);
            }
        );
    }
```

  Najpierw następuje odczytanie 4 bajtów stanowiących rozmiar ramki video w Int32. Potem odczytane zostaje tyle bajtów ile zawierała liczba skonwertowana przy użyciu systemowej klasy BitConverter.

Następnie aplikacja korzysta z klasy Utils jako narzędzia do konwersji tablicy bajtów do obrazu. Używając głównego wątku UI obraz jest wyświetlany na ekranie głównym aplikacji przy użyciu klasy ImageSource i mechanizmu obserwowania.

## 4. Wnioski

Podczas implementacji projektu zapoznaliśmy się z technologią WPF oraz Universal Windows. Pomocna nam była wiedza o kompresji obrazu oraz teoretycznie zapoznaliśmy się ze standardem h.264, którego użycie ograniczyła nas technologia. Wbrew założeniom nie udało nam się stworzyć strumienia odtwarzalnego przy użyciu programu typu VLC, jednakże nauczyliśmy się komunikacji przy użyciu Socketów tworząc prosty system wymiany danych w postaci segmentów bajtów. Niezwykle użyteczna okazała się implementacja serwisu do przesyłania danych o dostępnych połączeniach. Mimo tego, że dla naszego systemu użycie serwisu wydaję się być przerostem formy nad treścią, rozbudowując system o konta użytkowników rozszerzając jego możliwości do choćby prostego, domowego systemu monitoringu mógłby się okazać jeszcze bardziej użyteczny.