# Raport 6

Mikołaj Zawada 259431
Tomasz Płóciennik 260404

Projekt wykonany został przy pomocy Visual Studio 2022, .NET Framework 4.8.

**Pierwsze kroki wykonania projektu:**

1. Zainstalowanie wymaganego IDE oraz .NET Framework 4.8.
   Dodatkowo trzeba doinstalować pakiet CFW

2. Uruchomić Visual Studio 2022.

3. Stworzyć nowy projekt WCF Service Application.

4. Nazwać projekt MyWebService

5. Tworzymy osobny projekt, C# Console App, nazywamy go Client

6. W projekcie Client stworzyć klasę MyData oraz ustawić wywołanie jej statycznej metody .info() jako pierwszą operację w plikach Program.cs. Z racji na to, że serwer nie uruchamia konsoli nie robimy tego dla serwera

MyData.info:

```
public class MyData
{
    1 reference
    public static void info()
    {
        Console.WriteLine("Płóciennik Tomasz 260404");
        Console.WriteLine("Zawada Mikołaj 259431");
        Console.WriteLine(DateTime.Now.ToString("dd MMMM, HH:mm:ss", new System.Globalization.CultureInfo("pl-PL")));
        Console.WriteLine(Environment.Version);
        Console.WriteLine(Environment.UserName);
        Console.WriteLine(Environment.OSVersion);

        IPAddress[] localIPs = Dns.GetHostAddresses(Dns.GetHostName());
        foreach (IPAddress addr in localIPs)
        {
            if (addr.AddressFamily == AddressFamily.InterNetwork)
            {
                Console.WriteLine(addr.ToString());
            }
        }
    }
}
```

## Implementacja serwera:

W pliku IRestService.cs:

Dodajemy wszystkie endpointy:

```
[ServiceContract]
1 reference
public interface IRestService
{
    [OperationContract]
    [WebGet(UriTemplate = "/xml/People")]
    1 reference
    List<Person> getAllXml();

    [OperationContract]
    [WebGet(UriTemplate = "/xml/People/{id}", ResponseFormat = WebMessageFormat.Xml)]
    1 reference
    Person getByIdXml(string id);

    [OperationContract]
    [WebInvoke(UriTemplate = "/xml/People", Method = "POST", RequestFormat = WebMessageFormat.Xml)]
    1 reference
```

Dodajemy również nasz nietypowy typ danych wykorzystywany w komunikacji:

```
[DataContract]
27 references
public class Person
{
    [DataMember(Order = 0)]
    21 references
    public int id { get; set; }

    [DataMember(Order = 1)]
    13 references
    public string name { get; set; }

    [DataMember(Order = 2)]
    13 references
    public int age { get; set; }

    [DataMember(Order = 3)]
    13 references
    public string email { get; set; }

}
```

W Service1.svc:

Dodajemy listę z wstępnie zainicjalizowanymi osobami oraz implementujemy wszystkie funkcje wykorzystywane przez endpointy.

```
private static List<Person> people = new List<Person>(){
    new Person() { id = 0, name = "Adam", age = 22, email = "adam@nowak.com"},
    new Person() { id = 1, name = "Jan", age = 26, email = "kowalski@onet.pl"},
    new Person() { id = 2, name = "Stefan", age = 55, email = "stef2344123@gmail.com"}
};

1 reference
public List<Person> getAllXml(){
    return people;
}

1 reference
public Person getByIdXml(string Id){
    int id = int.Parse(Id);
    int index = people.FindIndex(p => p.id == id);
    if (index == -1){
        throw new WebFaultException<string>("404: Not Found", HttpStatusCode.NotFound);
    }
    return people.ElementAt(index);
}
```

## Implementacja klienta:

W pliku Program.cs:
Wyświetlamy Mydata.info oraz tworzymy klienta

```
MyData.info();

RestClient client = new RestClient();
```

W zależności od wybranej opcji ustawiamy odpowiedni endpoint, metodę i typ zapytania, po czym wykonujemy funkcję request z pliku RestClient.cs.

```
            case 12:
                endpoint = "json/People/count";
                method = "GET";
                type = "application/json";
                break;
            default:
                EXIT = true;
                break;
        }

        client.request(endpoint, method, type);
    }
```

Plik RestClient.cs:

Funkcja pobiera endpoint, dopisuje go do adresu i wywołuje zapytanie do serwera

```
private static readonly string ADDRESS = "http://192.168.100.10:10000/Service1.svc/";

1 reference
public void request(string endpoint, string method, string type)
{
    try
    {
        HttpWebRequest req = WebRequest.Create(ADDRESS + endpoint) as HttpWebRequest;
        req.KeepAlive = false;
        req.Method = method;
        req.ContentType = type;
```

W przypadku PUT i POST wysyłamy dane:

```
                case "POST":
                    byte[] buforPost = Encoding.UTF8.GetBytes(sendData);
                    req.ContentLength = buforPost.Length;
                    Stream postData = req.GetRequestStream();
                    postData.Write(buforPost, 0, buforPost.Length);
                    postData.Close();
                    break;
                case "PUT":
                    byte[] buforPut = Encoding.UTF8.GetBytes(sendData);
                    req.ContentLength = buforPut.Length;
                    Stream putData = req.GetRequestStream();
                    putData.Write(buforPut, 0, buforPut.Length);
                    putData.Close();
                    break;
```

Odbieramy odpowiedź i zamykamy streamy:

```
HttpWebResponse resp = req.GetResponse() as HttpWebResponse;
Encoding enc = Encoding.GetEncoding("UTF-8");
StreamReader responseStream = new StreamReader(resp.GetResponseStream(), enc);
string responseString = responseStream.ReadToEnd();
responseStream.Close();
resp.Close();
```

Umożliwienie konfiguracji 2 stanowiskowej:
Na serwerze trzeba odnaleźć plik ustawień serwera i zezwolić na dostęp z innych domen, w pliku applicationhost.config wyszukujemy naszą stronę z odpowiednim portem i stroną (domyślnie localhost)

C: > Studia > RSI-Lab > Proj6 > MyWebService > .vs > MyWebService > config > applicationhost.config

```
172             <site name="MyWebService(1)" id="3">
173                 <application path="/" applicationPool="Clr4IntegratedAppPool">
174                     <virtualDirectory path="/" physicalPath="C:\Studia\RSI-Lab\Proj6\MyWebService\MyWebService" />
175                 </application>
176                 <bindings>
177                     <binding protocol="http" bindingInformation="*:10000:localhost" />
178                 </bindings>
179             </site>
```

Usuwamy localhost, w bindingInformation, umożliwia to podłączenie z dowolnej domeny:

```
C: > Studia > RSI-Lab > Proj6 > MyWebService > .vs > MyWebService > config > applicationhost.config
172             <site name="MyWebService(1)" id="3">
173                 <application path="/" applicationPool="Clr4IntegratedAppPool">
174                     <virtualDirectory path="/" physicalPath="C:\Studia\RSI-Lab\Proj6\MyWebService\MyWebService" />
175                 </application>
176                 <bindings>
177                     <binding protocol="http" bindingInformation="*:10000:" />
178                 </bindings>
179             </site>
```

W kliencie sprawa jest prostsza, wystarczy ustawienie odpowiedniego IP i portu, tak samo jak w konfiguracji 1 maszynowej (tu ustawiamy tylko adres do stałej):

```
public class RestClient
{
    private static readonly string ADDRESS = "http://192.168.100.10:10000/Service1.svc/";
```

Taka konfiguracja pozwala nam na uruchomienie serwera na jednym komputerze a klienta na drugim.

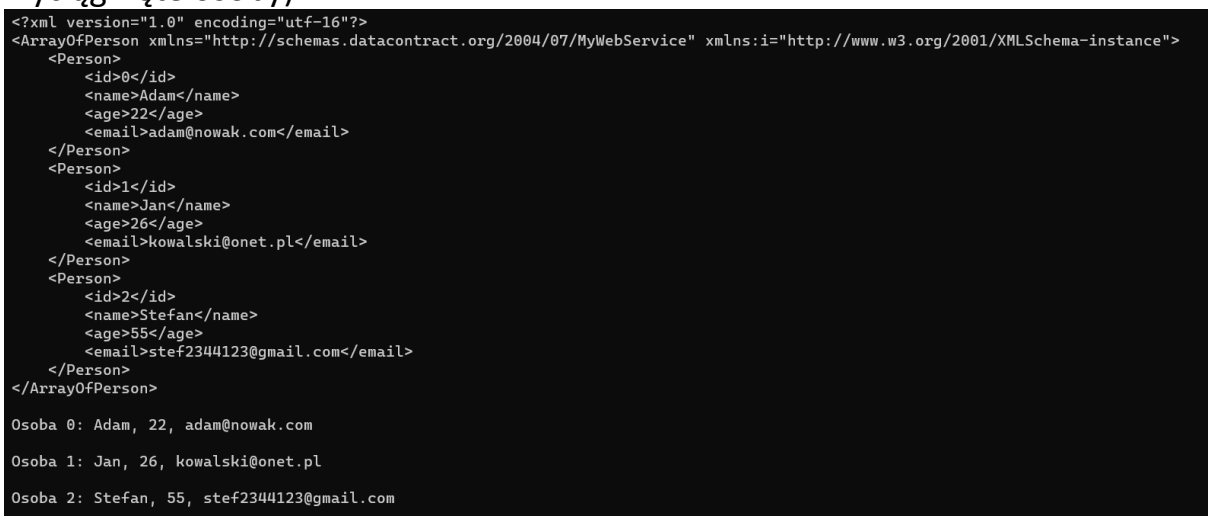
## Działanie aplikacji:

Menu główne:

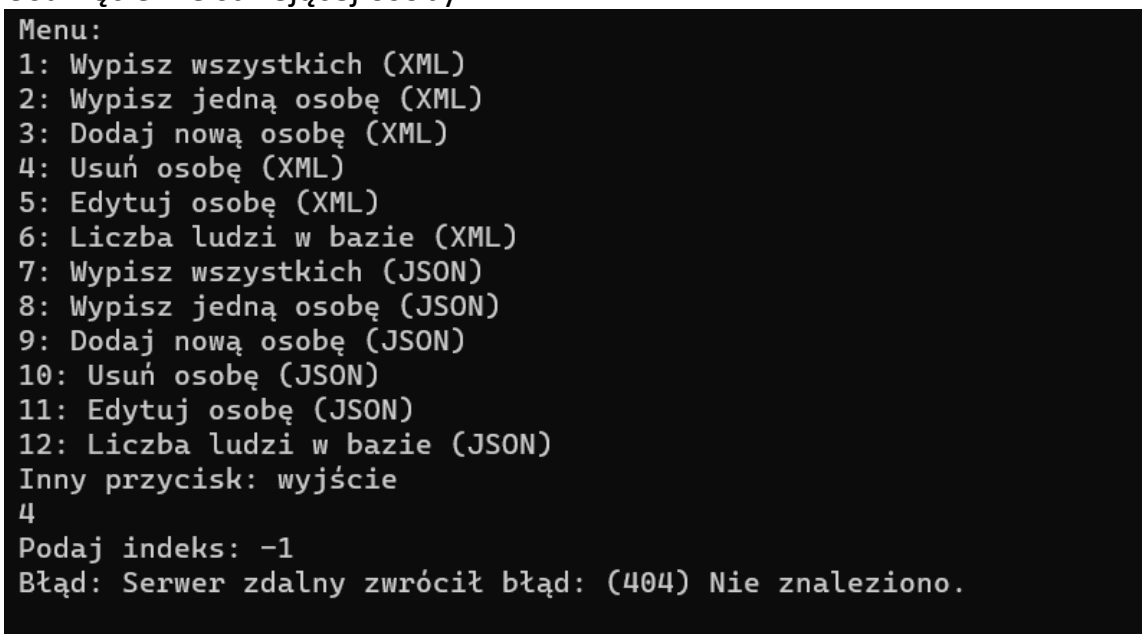
```
Płóciennik Tomasz 260404
Zawada Mikołaj 259431
10 maja, 22:56:52
4.0.30319.42000
tomik
Microsoft Windows NT 6.2.9200.0
192.168.100.46
172.24.208.1
192.168.56.1
192.168.154.1
192.168.248.1


Menu:
1: Wypisz wszystkich (XML)
2: Wypisz jedną osobę (XML)
3: Dodaj nową osobę (XML)
4: Usuń osobę (XML)
5: Edytuj osobę (XML)
6: Liczba ludzi w bazie (XML)
7: Wypisz wszystkich (JSON)
8: Wypisz jedną osobę (JSON)
9: Dodaj nową osobę (JSON)
10: Usuń osobę (JSON)
11: Edytuj osobę (JSON)
12: Liczba ludzi w bazie (JSON)
Inny przycisk: wyjście
```

Wypisanie wszystkich osób (na górze niezdekodowany XML a na dole wyciągnięte osoby):




Usunięcie nieistniejącej osoby:




Edycja osoby:


Edycja nieistniejącej osoby:

```
Podaj indeks: -1
Podaj imię: Nieistniejący
Podaj wiek: 11
Podaj email: nieistniejący@wp.pl
Wysłane dane: <Person xmlns="http://schemas.datacontract.org/2004/07/MyWeb
tniejący</name><age>11</age><email>nieistniejący@wp.pl</email></Person>
Błąd: Serwer zdalny zwrócił błąd: (404) Nie znaleziono.
```

Usunięcie osoby:

```
10
Podaj indeks: 1
"Removed person: 1, Edytowany, 22, edytowany@wp.pl"
```

Wypisanie 1 osoby:

```
Podaj indeks: 2
<?xml version="1.0" encoding="utf-16"?>
<Person xmlns="http://schemas.datacontract.org/2004/07/MyWebService" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
    <id>2</id>
    <name>Stefan</name>
    <age>55</age>
    <email>stef2344123@gmail.com</email>
</Person>

Osoba 2: Stefan, 55, stef2344123@gmail.com
```

W przypadku niedostępności serwera:

```
1
Błąd: Nie można połączyć się z serwerem zdalnym
```