Usługa REST w Web API 2 (Visual Studio 2017)

Jacek Matulewski

Protokół HTTP

Protokół HTTP (od ang. *Hypertext Transfer Protocol*) kojarzy się przede wszystkim z przeglądarkami i stronami WWW („http://” na początku adresów stron WWW). I słusznie, bo HTTP to bezstanowy protokół używany do komunikacji z serwerami i pobierania z nich dokumentów (np. kod HTML strony WWW), wysyłania na serwer danych z formularzy oraz innych operacji na zasobach udostępnianych przez serwer. Słowo „bezstanowy” oznacza, że sam protokół nie ma pamięci poprzednich aktów komunikacji między serwerem, a klientem (przeglądarką), choć oczywiście zarówno serwer, jak i klient może przechowywać dane, które pozwolą na zmianę w udostępnianym dokumencie lub sposobie jego wyświetlania. Protokół zakłada scenariusz komunikacji, w którym klient wysyła do serwera zapytanie (ang. *request*), serwer je przetwarza i odsyła odpowiedź (ang. *response*). Zapytanie składa się z nagłówka oraz ciała. W nagłówku określana jest metoda, o nich za chwilę, wersję protokołu HTTP (aktualna to 1.1) oraz dokładny adres URI. Adres URI wskazuje na ścieżkę do dokumentu lub innego zasobu na serwerze tj. nie zawiera części wskazującej na domenę. Dla przykładu w adresie *http://www.serwer.domena.pl/sciezka/dokument.html*, adres URI to */sciezka/dokument.html*.

W protokole HTTP zdefiniowane jest kilka metod, spośród których najważniejsze są cztery odpowiadające operacjom CRUD, a mianowicie: GET (*read*), PUT (*create* i *update*), POST (*update* i *create*) oraz DELETE (*delete*). Najważniejsza jest oczywiście metoda GET, która pozwala przeglądarkom na pobranie dokumentu z serwera w celu jego wyświetlenia użytkownikowi.

Przykładowe zapytanie, czyli kilkuliniowy łańcuch przesyłany do serwera, może wyglądać następująco:

```
GET /sciezka/dokument.html HTTP/1.1
Host: www.serwer.domena.com
Accept: image/gif, image/jpeg, */*
Accept-Language: pl, en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

Oznacza ono, że chcemy pobrać dokument *http://www.serwer.domena.pl/sciezka/dokument.html*, w którym mogą być odwołania do rysunków w formacie GIF i JPEG, preferujemy język polski i amerykański angielski oraz dopuszczamy dwie metody kompresji. W zapytaniu może być też wysłany łańcuch identyfikujący przeglądarkę.

Na to serwer wyśle odpowiedź. Odpowiedź również zawiera nagłówek i ciało. W ciele znajdzie się kod HTML pobieranego dokumentu, a nagłówek – dodatkowe informacje o serwerze (łańcuch identyfikujący oprogramowanie) i wysyłanym dokumencie (np. jego długość, datę utworzenia i modyfikacji, itp.):

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
```

```
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body>Zawartość pliku HTML</body></html>
```

Dlaczego usługi REST

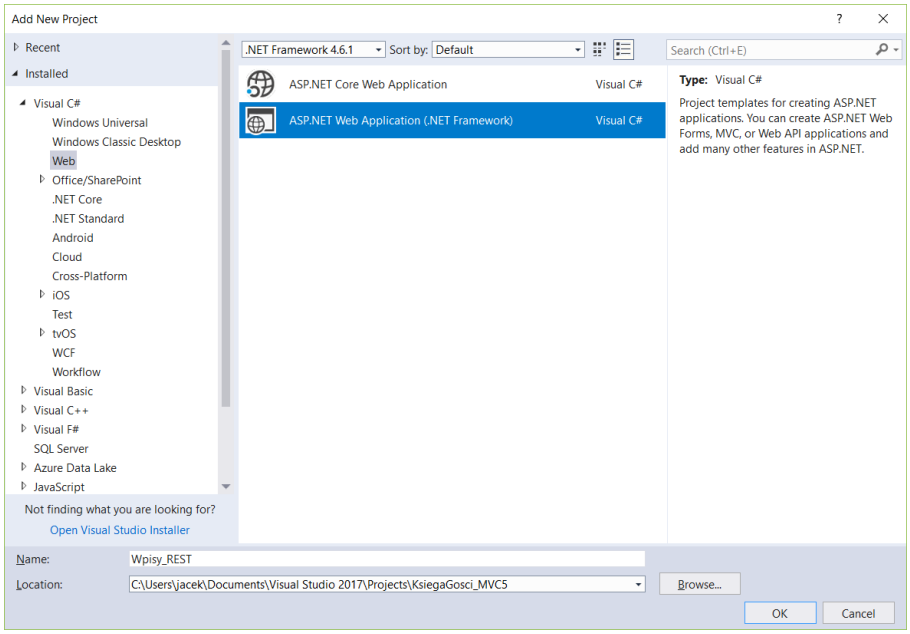
Najważniejszą odpowiedzią na to pytanie jest: możliwość wykorzystania prze każdego klienta. W odróżnieniu od usług sieciowych .NET lub usług WCF, usługi REST są powszechnym protokołem, który może być wykorzystany nie tylko w aplikacji internetowej ASP.NET, czy w aplikacji dekstopowej WPF, ale również w każdej innej technologii, która posiada wsparcie dla protokołu HTTP.

Źródło: *https://docs.microsoft.com/en-us/aspnet/web-api/*

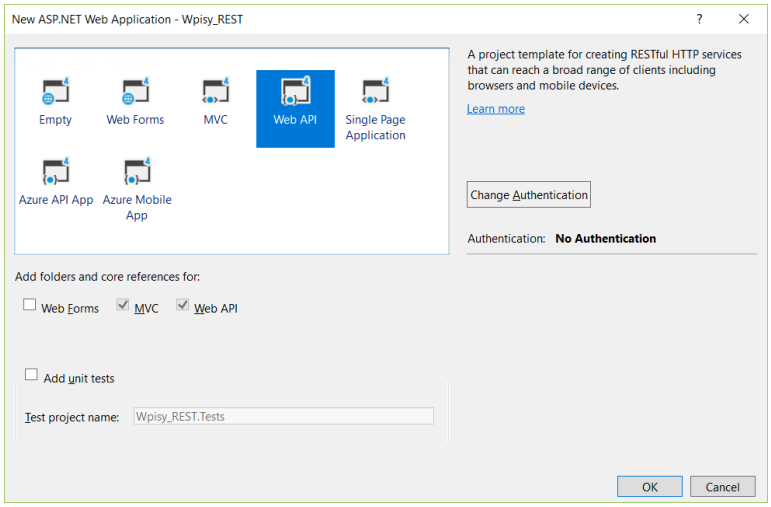
Tworzenie projektu

Do rozwiązania „KsiegaGosci_MVC5” chcemy dodać projekt usługi REST implementowanej jako Web API. Usługa będzie odpowiedzialna za operacje CRUD (nie wszystkie) na zbiorze wpisów.

1. W oknie *Solution Explorer* zaznaczamy pozycję *Solution ‘KsiegaGosci_MVC5’ (1 project)* i prawym klawiszem myszy rozwijamy meny kontekstowe.
2. Wybieramy *Add*, *New project...*.
3. W oknie *Add New Project* wybieramy kategorię *Web*, a w niej zaznaczamy *ASP.NET Web Application (.NET Framework)*.
4. Wpisujemy nazwę projektu „Wpisy_REST” i klikamy *OK*.



5. W kolejnym oknie zaznaczamy ikonę *Web API*. Zwróćmy uwagę na domyślnie zaznaczone pola opcji *MVC* i *Web API*. Klikamy *OK*.
6. Po dłuższej chwili do rozwiązany dodanie zostanie projekt usługi Web API.



7. Do pliku *Global.asax.cs* w nowym projekcie kopiujemy metodę `DopiszDoPlikuLog`

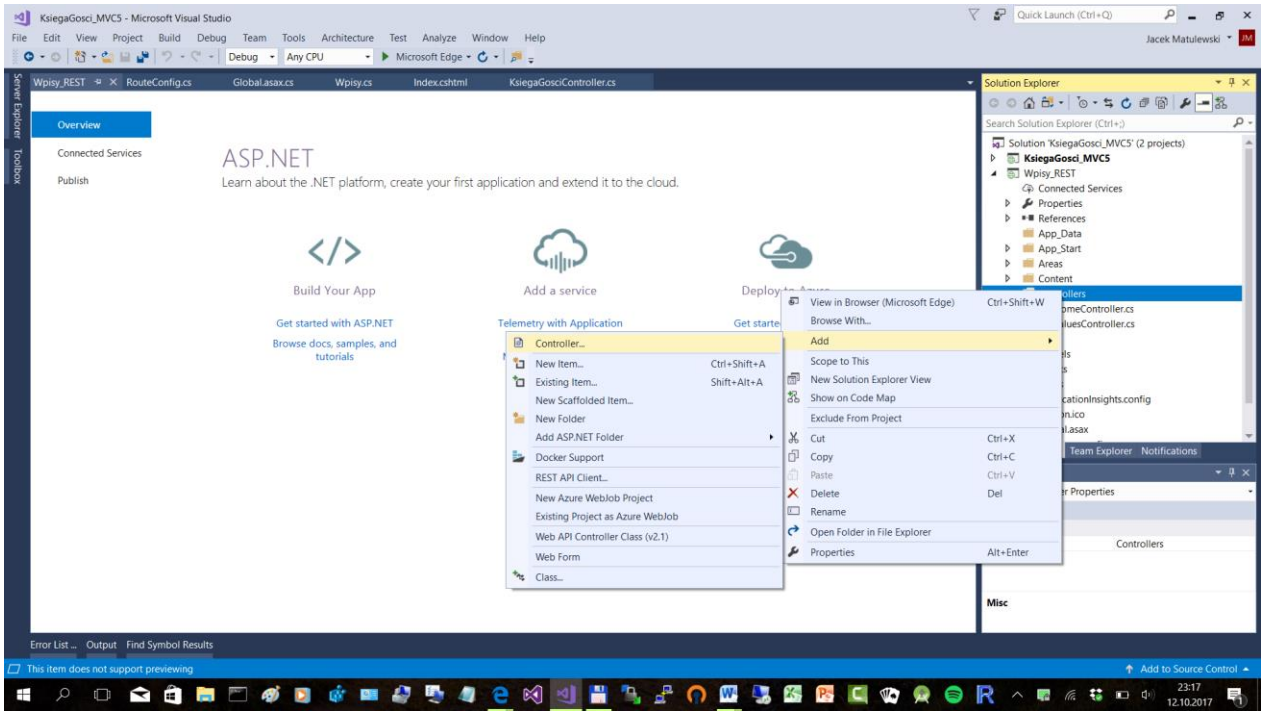
Listing 1. Plik Global.asax.cs w projekcie Wpisy_REST

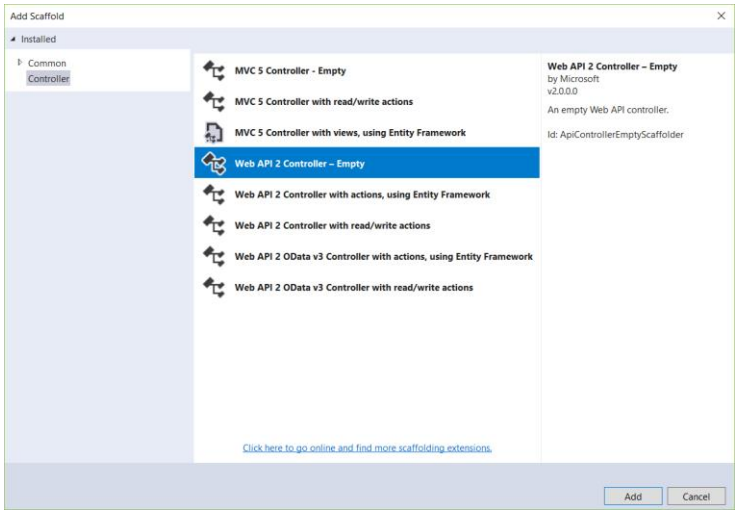
```
using System;
using System.IO;
```

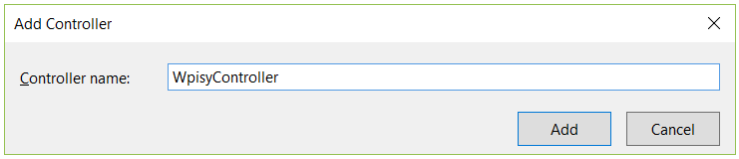
```
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace Wpisy_REST
{
    public class WebApiApplication : System.Web.HttpApplication
    {
        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();
            GlobalConfiguration.Configure(WebApiConfig.Register);
            FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
            RouteConfig.RegisterRoutes(RouteTable.Routes);
            BundleConfig.RegisterBundles(BundleTable.Bundles);
        }

        private const string nazwaPliku_Log = @"d:\KsiegaGosci.log";

        public static void DopiszDoPlikuLog(string informacja)
        {
            if (File.Exists(nazwaPliku_Log))
                File.Copy(nazwaPliku_Log, nazwaPliku_Log + ".bak", true);
            using (StreamWriter sw = new StreamWriter(nazwaPliku_Log, true))
            {
                sw.WriteLine(DateTime.Now.ToString() + ": " + informacja);
            }
        }
    }
}
```

8. Do katalogu *Models* w nowym projekcie przeciągamy plik *Wpisy.cs* z katalogu *Models* projektu *KsiegaGosci_MVC5*. Następnie klikamy go dwukrotnie, aby otworzyć go w edytorze.
    a. Zmieniamy przestrzeń nazw na `namespace Wpisy_REST.Models`
    b. Przy wywołaniach metody `DopiszDoPlikuLog` zmieniamy nazwę klasy z `MvcApplication` na `WebApiApplication` (klasa z pliku *Global.asax.cs*).
9. Kompilujemy projekt.

Kontroler – Hello World!





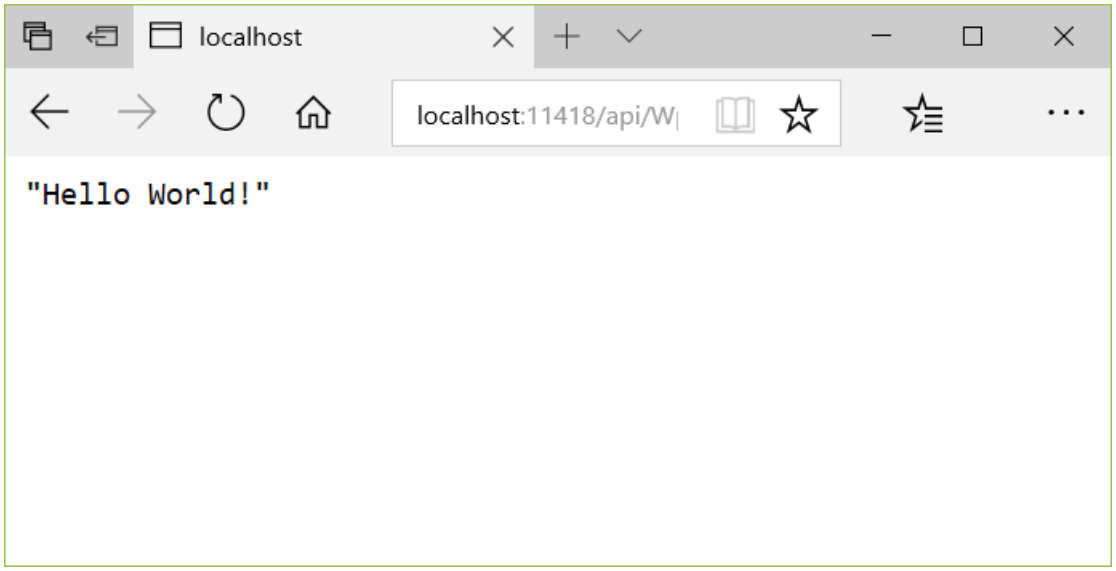


1. Dla katalogu *Controllers* w nowym projekcie rozwijamy menu kontekstowe i wybieramy *Add*, *Controller...*.
2. W oknie *Add Scaffold* zaznaczamy *Web API 2 Controller – Empty* i klikamy *Add*.
3. W kolejnym oknie *Add Controller* podajemy nazwę „WpisyController” i klikamy *Add*.
4. Przechodzimy do edycji pliku *Controllers\WpisyController.cs*. Do klasy `WpisyController` dodajemy metodę `Get` obsługującą metodę `GET` z protokołu HTTP:

```
using System.Web.Http;
```

```
namespace Wpisy_REST.Controllers
{
    public class WpisyController : ApiController
    {
        public string Get()
        {
            return "Hello World!";
        }
    }
}
```

5. Oznaczamy projekt *Wpisy_REST* jako projekt uruchamiany (*Set as StartUp Project*).
6. Uruchamiamy aplikację (*F5*). W przeglądarce wpisujemy adres analogiczny do (może różnić się numerem portu): *http://localhost:11418/api/wpisy*. Wielkość liter w adresie (np. *api* lub *Api*, *wpisy* lub *Wpisy*) nie jest ważna.



Łańcuch został zserializowany do JSON. Cokolwiek zwrócimy, zostanie zserializowane.

Kontroler – Liczba wpisów. Trasowanie (routing) oparte na atrybutach

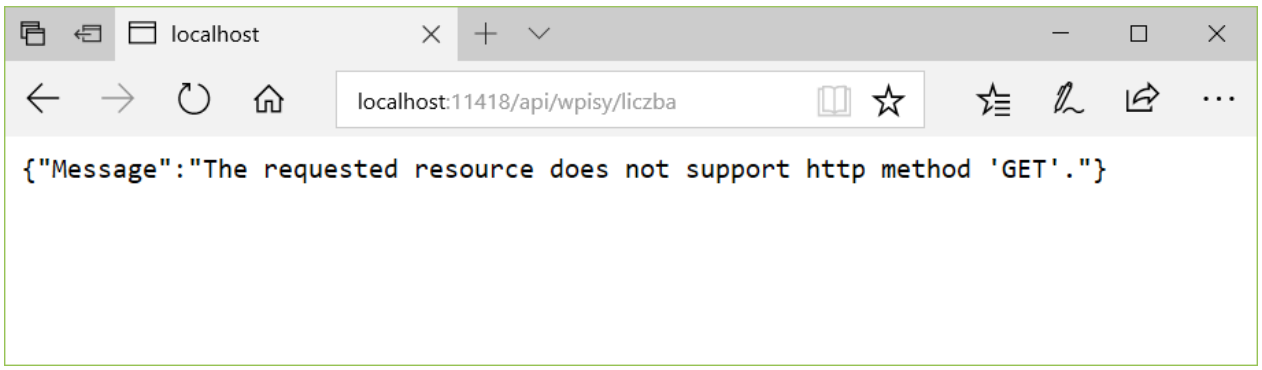
Do klasy kontrolera `WpisyController` dodajmy metodę `PobierzLiczbęWpisów`. To już druga metoda `GET` tym samym kontrolerze, a jej nazwa nie rozpoczyna się od `Get..`, więc oznaczamy ją atrybutem `HttpGet`. Ponadto musimy wskazać jej „ścieżkę dostępu” w adresie URI. Robimy to za pomocą atrybutu `Route`. Podobny atrybut wstawiamy przy klasie kontrolera.

```
using System.Web.Http;

namespace Wpisy_REST.Controllers
{
    using Models;

    [RoutePrefix("api/wpisy")]
    public class WpisyController : ApiController
```

```
    {
        public string Get()
        {
            return "Możliwe akcje GET: liczba, wszystkie, indeks";
        }

        public const string nazwaPliku_WpisyXml = @"d:\wpisy.xml";
        Wpisy wpisy = new Wpisy(nazwaPliku_WpisyXml);

        [HttpGet]
        [Route("liczba")]
        public int PobierzLiczbę()
        {
            return wpisy.Liczba;
        }
    }
}
```

Routing (trasowanie)

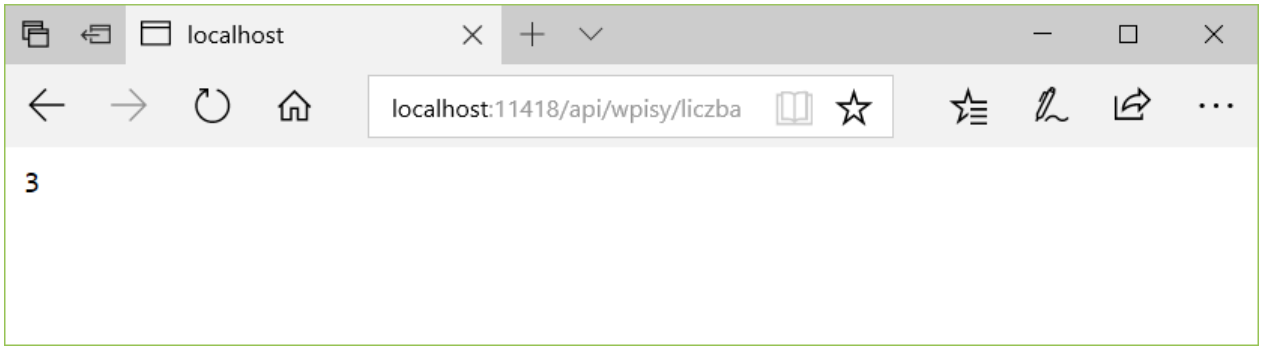
Jeżeli teraz spróbujemy uruchomić metodę wskazując adres, który podaliśmy w atrybutach metody i klasy kontrolera tj. *http://localhost:11418/api/wpisy/liczba*, to pojawi się w przeglądarce komunikat błędu.



Użyliśmy trasowania opartego na atrybutach. Alternatywnym jest trasowanie oparte na szablonach określanych w konfiguracji aplikacji. Odpowiednie odwzorowania znajdują się w metodzie `WebApiConfig.Register` wywoływanej z metody `WebApiApplication.Application_Start` przy uruchomieniu usługi. Przykład pokazany jest na poniższym listingu, w którym dodane jest polecenie odwzorowania ścieżki URI: *api/[nazwa kontrolera]/[nazwa akcji]*. W naszym przypadku chodzi o ścieżkę *api/wpisy/liczba*.

```
using System.Web.Http;

namespace Wpisy_REST
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API configuration and services
```

```
            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );

            config.Routes.MapHttpRoute("DefaultApi1", "api/{controller}/{action}",
                                       new { id = RouteParameter.Optional });
        }
    }
}
```



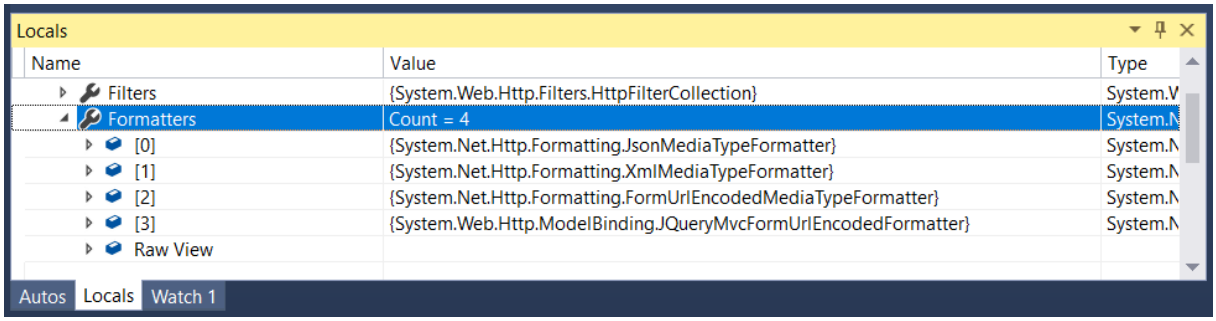
Ćwiczenie – Kontroler – Pobieranie całej kolekcji

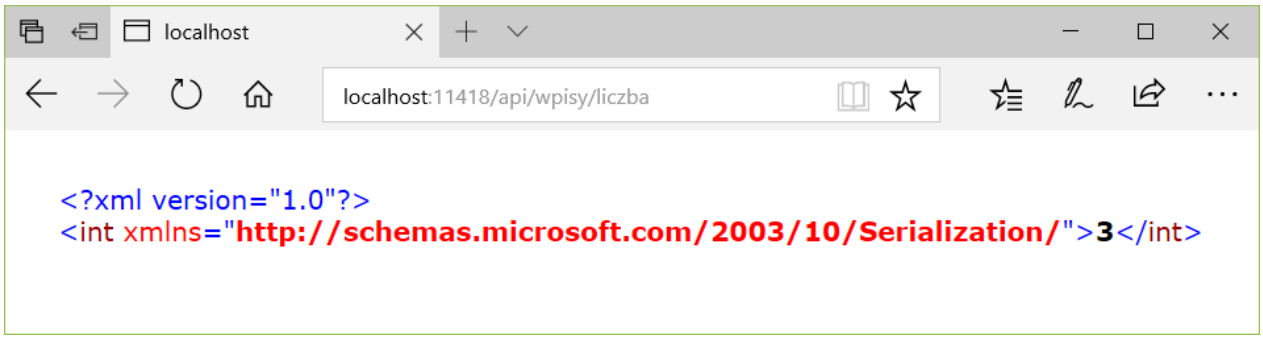
Dodaj do kontrolera metodę `PobierzWszystkie`, która zwróci tablicę wszystkich wpisów.

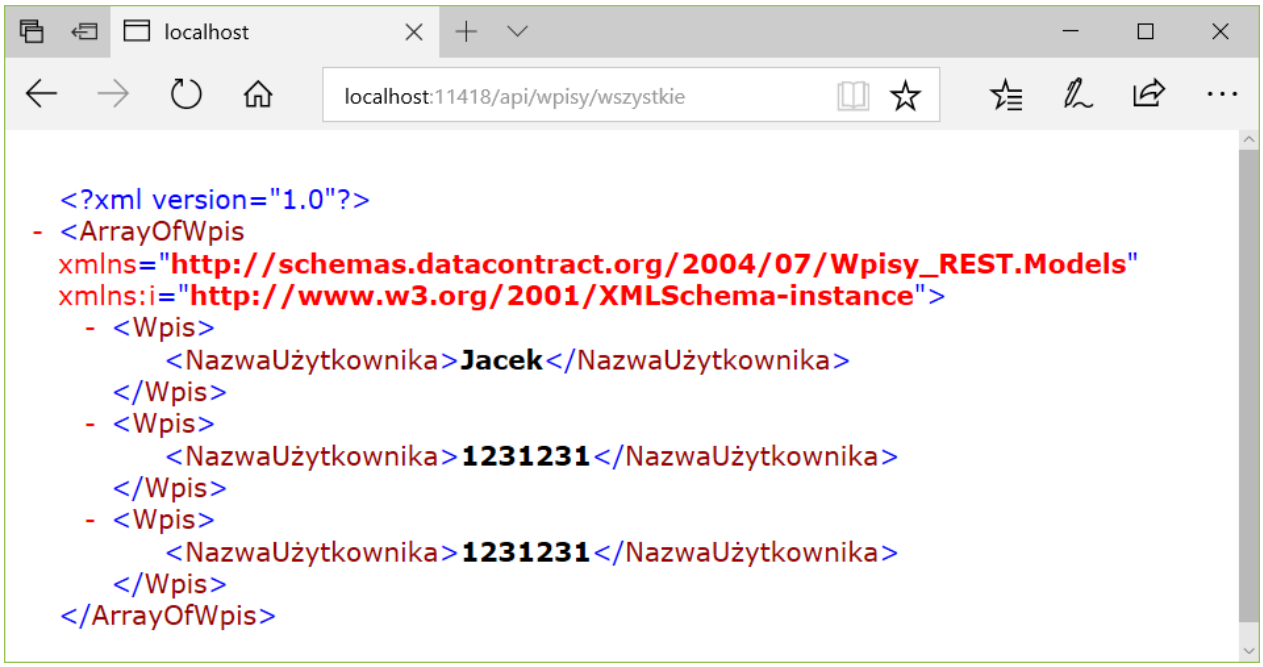
```
[HttpGet]
[Route("wszystkie")]
public Wpis[] PobierzWszystkie()
{
    //bez LINQ, bo Wpisy nie implementują IEnumerable
    int liczbaWpisów = wpisy.Liczba;
    Wpis[] tablicaWpisów = new Wpis[liczbaWpisów];
    for (int i = 0; i < liczbaWpisów; ++i) tablicaWpisów[i] = wpisy[i];
    return tablicaWpisów;
}
```

Konfiguracja – Zmiana JSON na XML

Dane przesyłane z odpowiedzią HTTP (ang. *HTTP Response*) są domyślnie zapisywane w formacie JSON. Taki jest pierwszy element w kolekcji `config.Formatters`. Aby domyślnie użyć formatowania XML, wystarczy zmienić kolejność formaterów. Można to zrobić w metodzie `WebApiConfig.Register`. Pliki XML są większe, za to łatwiejsze do czytania.

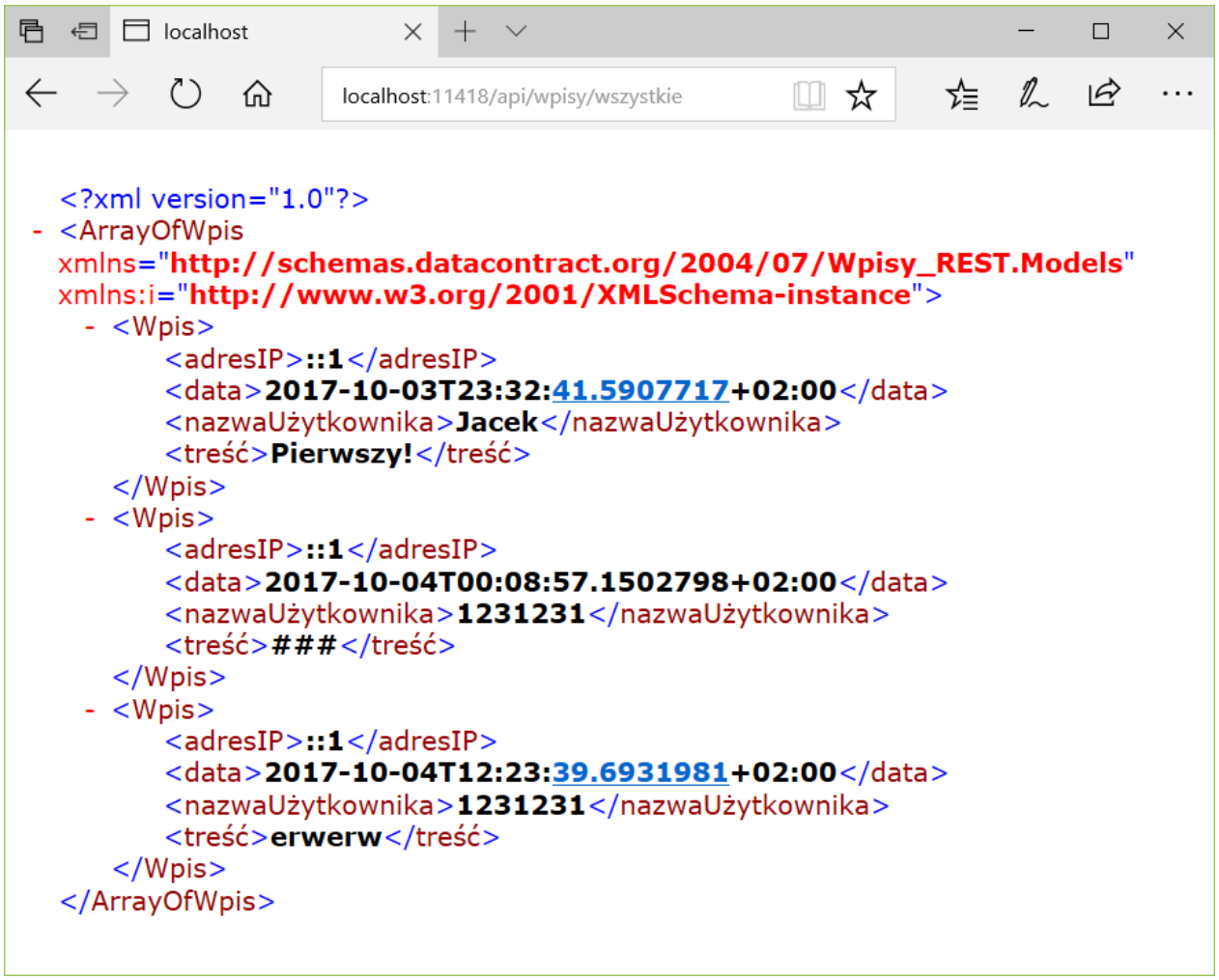






Z drugiego rysunku widać, że klasa Wpis nie jest w pełni serializowana do pliku XML – część publicznych elementów nie jest widoczna. Aby wymusić pełną serializację, dodajmy do jej definicji atrybyt `Serializable`. Wówczas zapisywane będą wartości wszystkich prywatnych pól zdefiniowanych w klasie `Wpis`.

```
namespace Wpisy_REST.Models
{
    //Struktura opisująca elementarny typ danych (klasa encji)
    [Serializable]
    public struct Wpis
    {
        private string nazwaUżytkownika;
        private string treść;
        private DateTime data;
        private string adresIP;
        ...
```

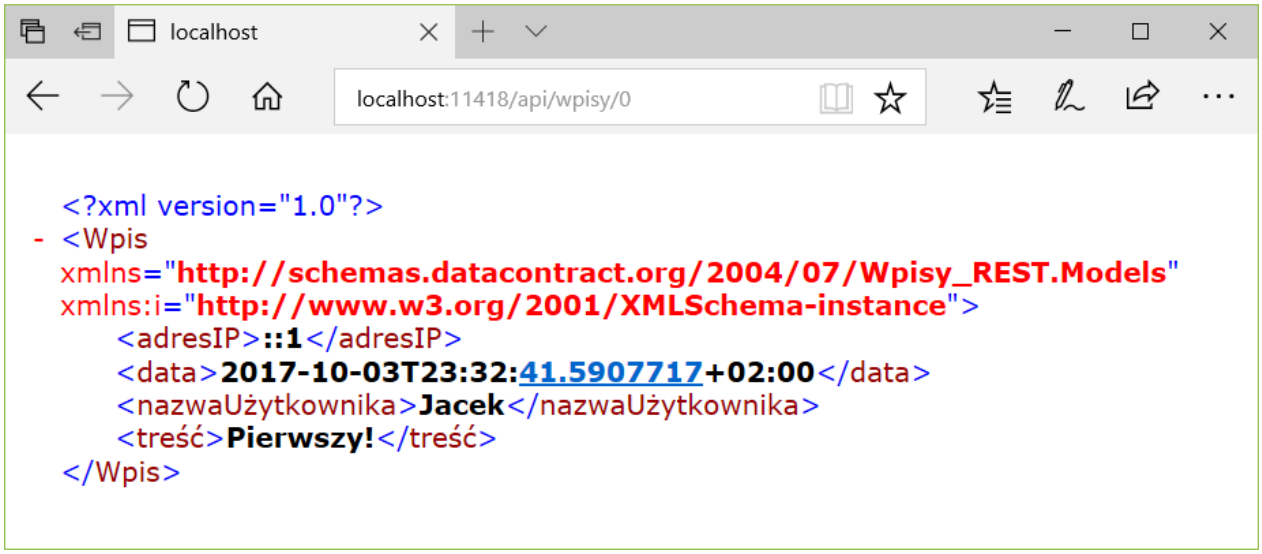


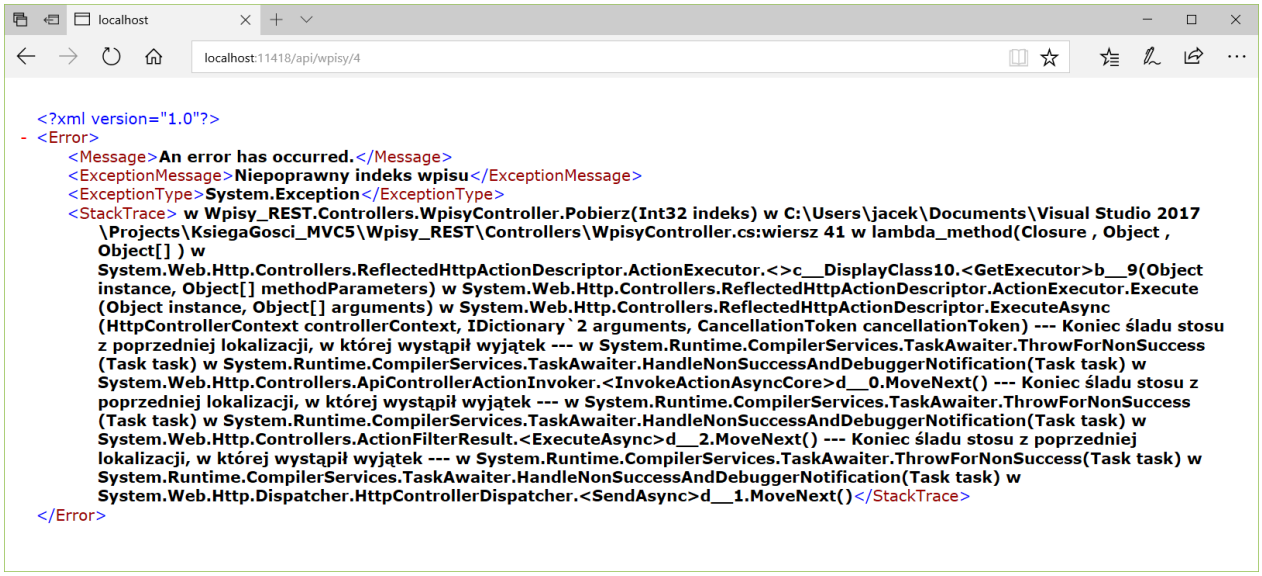
Przekazywanie argumentów do akcji GET kontrolera

Kolejnym krokiem będzie pobranie konkretnego wpisu o podanym indeksie. Dla takiej akcji ustalimy dwa trasowania: *api/wpisy/[numer]* oraz *api/wpisy/indeks/[numer]*.

```
[HttpGet]
[Route("{indeks}")]
[Route("indeks/{indeks}")]
public Wpis Pobierz(int indeks)
{
    if (indeks < 0 || indeks >= wpisy.Liczba)
        throw new System.Exception("Niepoprawny indeks wpisu");
    return wpisy[indeks];
}
```

Nazwa w nawiasach {} w atrybucie `Route` odpowiada argumentowi metody, którą atrybut ozdabia. Możemy podać tam liczby odpowiadające indeksom poszczególnych wpisów.





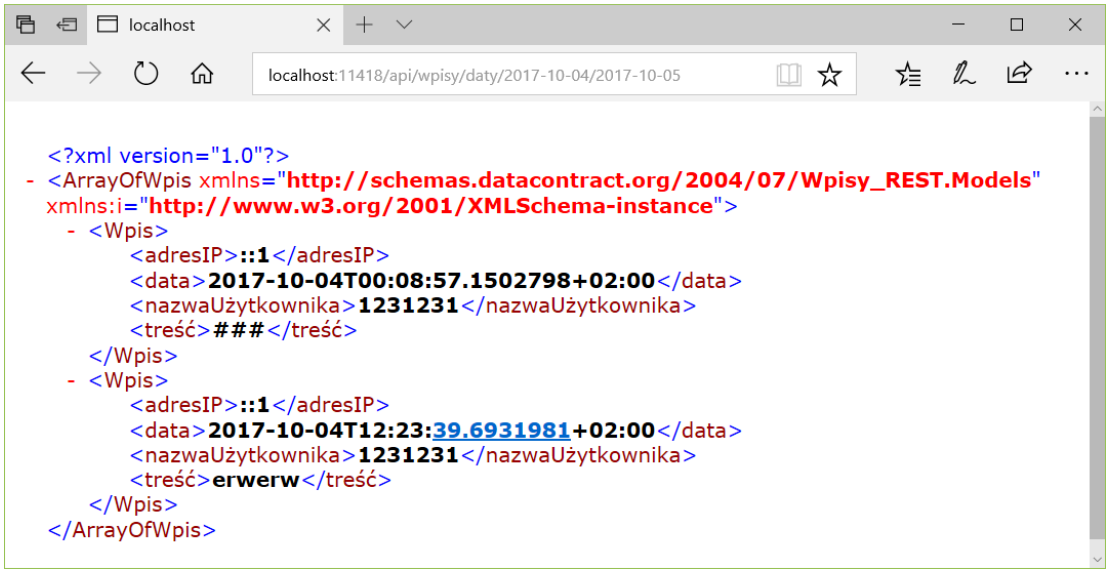
W razie podania indeksu, któremu nie odpowiada żaden element, zgłaszamy wyjątek. Zserializowany obiekt wyjątku zostanie przesłany do klienta (rysunek). Alternatywnym rozwiązaniem byłoby zwracanie wartości z brzegów kolekcji.

```
[HttpGet]
[Route("{indeks}")]
[Route("indeks/{indeks}")]
public Wpis Pobierz(int indeks)
{
    if (indeks < 0) indeks = 0;
    if (indeks >= wpisy.Liczba) indeks = wpisy.Liczba - 1;
    return wpisy[indeks];
}
```

Ćwiczenie - Kontroler –
Pobranie wpisów z podanego okresu

Do kontrolera dodaj dwie metody dwuargumentowe pozwalające na podanie zakresów indeksów lub zakresu dat wpisów.

```
[HttpGet]
[Route("indeksy/{indeksOd}/{indeksDo}")]
public Wpis[] Pobierz(int indeksOd, int indeksDo)
{
    if (indeksOd < 0 || indeksOd >= wpisy.Liczba)
        throw new System.Exception("Niepoprawny indeks wpisu 'od'");
    if (indeksDo < 0 || indeksDo >= wpisy.Liczba)
        throw new System.Exception("Niepoprawny indeks wpisu 'do'");
    int liczbaWpisów = indeksDo - indeksOd + 1;
    Wpis[] tablicaWpisów = new Wpis[liczbaWpisów];
    for (int i = indeksOd; i < indeksDo; ++i) tablicaWpisów[i - indeksOd] = wpisy[i];
    return tablicaWpisów;
}

[HttpGet]
[Route("daty/{dataOd}/{dataDo}")]
public Wpis[] Pobierz(DateTime dataOd, DateTime dataDo)
{
    List<Wpis> listaWpisów = new List<Wpis>();
    for (int i = 0; i < wpisy.Liczba; ++i)
    {
        Wpis wpis = wpisy[i];
        if (wpis.Data > dataOd && wpis.Data < dataDo)
            listaWpisów.Add(wpis);
    }
    return listaWpisów.ToArray();
}
```

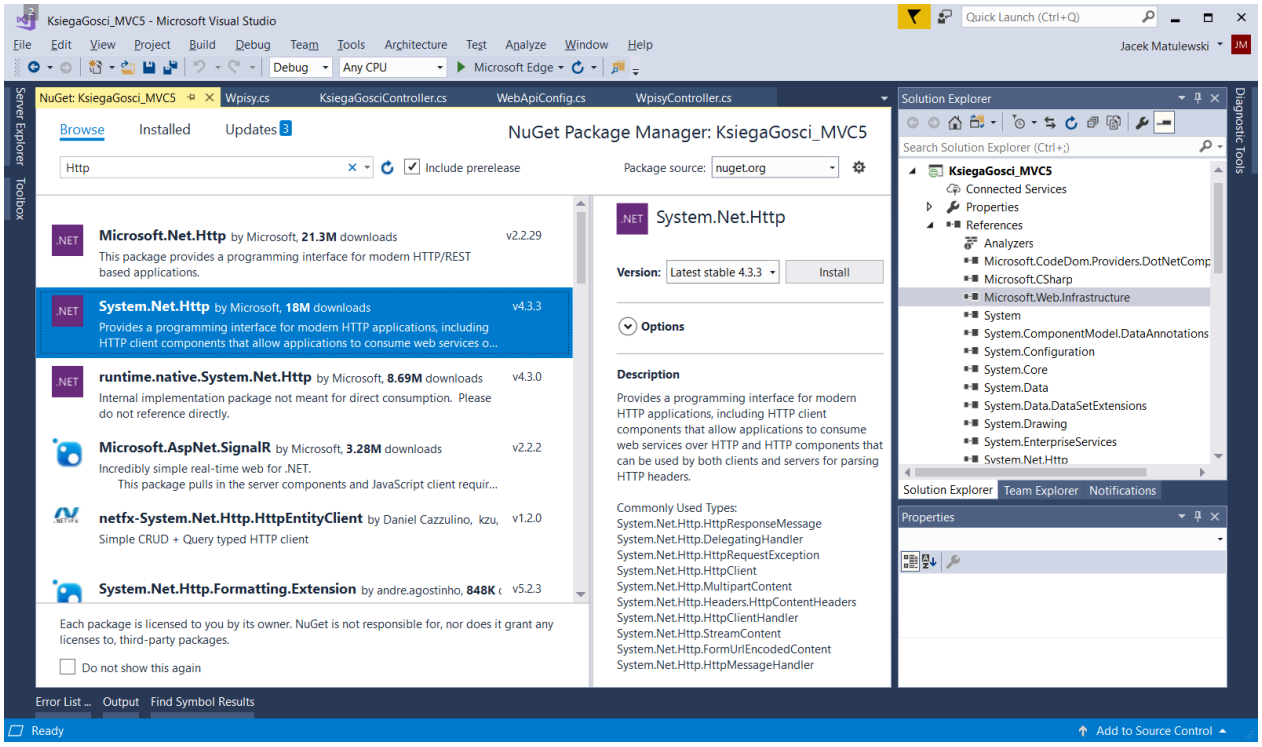


Aplikacja ASP.NET MVC
Użycie metod GET

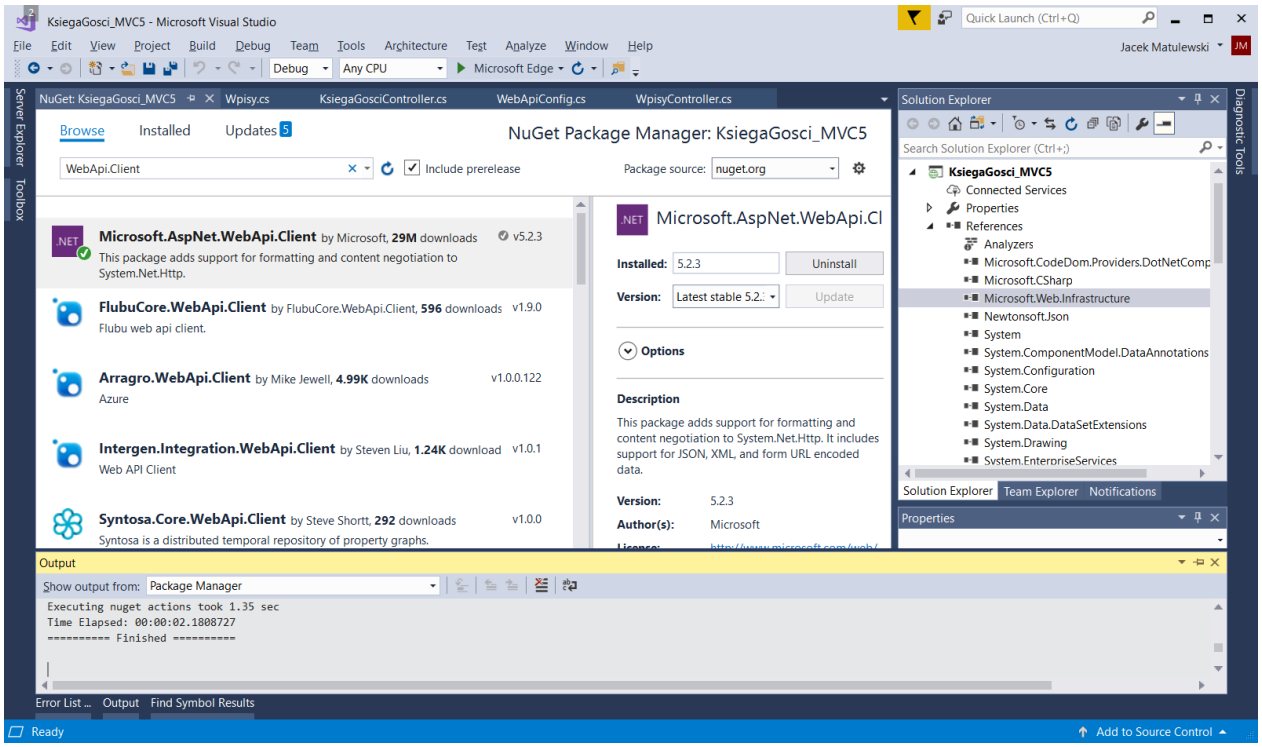
Przełączmy z powrotem aplikację KsiegaGosci_MVC5 jako aplikację domyślną w rozwiązaniu.

Instalacja bibliotek System.Net.Http i Microsoft.AspNet.WebApi.Client

Z menu kontekstowego *KsiegaGosci_MVC5\References* wybierz *Manage NuGet Packages...*. Kliknij przycisk *Install* z prawej strony okna. Pojawi się okno *Preview* zawierające listę bibliotek zależnych, które będą również zainstalowane. Klikamy *OK*, a w następnym oknie *I Accept*.



Postępując analogicznie zainstaluj bibliotekę *Microsoft.WebApi.Client*.



Pobranie wpisów z usługi Web API

W aplikacji ASP.NET zmodyfikujemy klasę `Wpisy` z pliku *Models\Wpisy.cs* w taki sposób, żeby zamiast czytania danych z pliku XML, dane pobierane były z usługi Web API. Wobec tego zastąpmy metodę `Wpisy.CzytajZPlikuXml` metodę `Wpisy.CzytajZUsługiRest`:

```
const string url = "http://localhost:11418/";

private bool czytajZUsługiRest()
{
    const string uri = "api/wpisy/wszystkie";
    const string xmlNamespace =
        "http://schemas.datacontract.org/2004/07/Wpisy_REST.Models";
    List<Wpis> odczytaneWpisy = new List<Wpis>();

    try
    {
        using (HttpClient klient = new HttpClient())
        {
            klient.BaseAddress = new Uri(url);
            klient.DefaultRequestHeaders.Clear();
            klient.DefaultRequestHeaders.Accept.Add(
                new MediaTypeWithQualityHeaderValue("application/xml"));

            HttpResponseMessage odpowiedź = klient.GetAsync(uri).Result;
            if (odpowiedź.IsSuccessStatusCode)
            {
                string daneOdpowiedzi = odpowiedź.Content.ReadAsStringAsync().Result;
```

```
                XDocument xml = XDocument.Load(new
System.IO.StringReader(daneOdpowiedzi));
                IEnumerable<XElement> elementy = xml.Root.Elements();

                for (int i = 0; i < elementy.Count(); ++i)
                {
                    XElement element = elementy.ElementAt(i);
                    string nazwaUżytkownika = element.Element(
                        XName.Get("nazwaUżytkownika", xmlNamespace)).Value;
                    string treść = element.Element(
                        XName.Get("treść", xmlNamespace)).Value;
                    DateTime data = DateTime.Parse(element.Element(
                        XName.Get("data", xmlNamespace)).Value);
                    string adresIP = element.Element(
                        XName.Get("adresIP", xmlNamespace)).Value;
                    Wpis wpis = new Wpis(nazwaUżytkownika, treść, data, adresIP);
                    odczytaneWpisy.Add(wpis);
                }
            }
        }

        wpisy.Clear();
        wpisy.AddRange(odczytaneWpisy);

        return true;
    }
    catch
    {
        return false;
    }
}
```

Metoda oczekuje danych w formacie XML i parsuje je nie korzystając z serializacji.

W konsekwencji zmieniamy konstruktor klasy `Wpisy`:

```
string nazwaPlikuXml;

public Wpisy(string nazwaPlikuXml)
{
    this.nazwaPlikuXml = nazwaPlikuXml;
    CzytajZPlikuXml(nazwaPlikuXml);
    czytajZUsługiRest();
}
```

Zmienia się wobec tego także jego wywołanie w klasie kontrolera `KsiegaGosciController`:

```
public class KsiegaGosciController : Controller
{
    // GET: KsiegaGosci
    public ActionResult Index()
    {
        ViewData.Add("wpisy", wpisy);
        return View(new Wpis());
    }

    public const string nazwaPliku_WpisyXml = @"d:\wpisy.xml";
    Wpisy wpisy = new Wpisy(nazwaPliku_WpisyXml);
```

Pobranie liczby wpisów

```
private int pobierzLiczbęWpisówZUsługiRest()
{
    const string uri = "api/wpisy/liczba";

    using (HttpClient klient = new HttpClient())
    {
        klient.BaseAddress = new Uri(url);
        HttpResponseMessage odpowiedź = klient.GetAsync(uri).Result;
        if (odpowiedź.IsSuccessStatusCode)
        {
            string daneOdpowiedzi = odpowiedź.Content.ReadAsStringAsync().Result;
            //int liczbaWpisów = (int)new XmlSerializer(typeof(int)).Deserialize(new
System.IO.StringReader(daneOdpowiedzi));
            XDocument xml = XDocument.Load(new System.IO.StringReader(daneOdpowiedzi));
            string wartość = xml.Root.Value;
            int liczbaWpisów = int.Parse(wartość);
            return liczbaWpisów;
        }
        else return -1;
    }
}

public int Liczba
{
    get
    {
        return wpisy.Count;
        return pobierzLiczbęWpisówZUsługiRest();
    }
}
```

Usługa Web API REST
Kontroler – Polecenie POST

Do tej pory tworzyliśmy jedynie metody obsługujące słowo `GET` z protokołu HTTP. Teraz chcielibyśmy przesłać dane na serwer. Wobec tego użyjemy słowa `POST` (`PUT` domyślnie jest blokowane przez IIS Express). Zdefiniujemy wobec tego metodę dodającą wpis do pliku przechowywanego na serwerze.

Do klasy Wpisy_REST.Controllers. WpisyController dodajemy:

```
[HttpPost]
[Route("dodaj")]
public void Dodaj(Wpis wpis)
{
    wpisy.Dodaj(wpis);
}
```

Aplikacja ASP.NET MVC5
Wysłanie zapytania POST

W aplikacji zmodyfikujemy klasę modelu Wpisy tak, żeby zapisywać nowy wpis do pliku XML, przesyłał go do usługi:

```
private bool wyślijWpisDoUsługiRest(Wpis wpis)
{
    const string url = "http://localhost:11418/";
    const string uri = "api/wpisy/dodaj";

    using (HttpClient klient = new HttpClient())
    {
        klient.BaseAddress = new Uri(url);
        HttpResponseMessage odpowiedź = klient.PostAsJsonAsync<Wpis>(uri, wpis).Result;
        return odpowiedź.IsSuccessStatusCode;
    }
}

public void Dodaj(Wpis nowyWpis)
{
    wpisy.Add(nowyWpis); //dane lokalne (bufor)
    if (ZapiszDoPlikuPoKażdymDodaniuWpisu) ZapiszDoPlikuXml();
    wyślijWpisDoUsługiRest(nowyWpis); //dane w usłudze (trwały zapis)
}
```

Zwróćmy uwagę, że nadal aktualizujemy dane w lokalnej liście wpisów (pełni rolę bufora zmniejszającego użycie połączenia sieciowego z usługą REST), ale nie zapisujemy ich już na lokalnym dysku. Dane trwale przechowywane są tylko przez usługę.

Jeżeli w ogóle chcemy zupełnie pozbyć się lokalnej listy wpisów (bufora), należy zmodyfikować już tylko indekser klasy `Wpisy`.