Upute uz prezentaciju WCF\_RazvojAlati

Contents

[Osnovne upute 3](#_Toc356221486)

[Servis: Kreiranje WCF servisne biblioteke– WCF Service Library 4](#_Toc356221487)

[Servis: testiranje servisa - WCF test client – manualno pokretanje 5](#_Toc356221488)

[Servis hosting: Self-hosting u command line aplikacija 6](#_Toc356221489)

[Servis hosting: Hosting u Windows Servisu 8](#_Toc356221490)

[Servis hosting: Hosting u Web aplikaciji na web serveru 10](#_Toc356221491)

[Servis konzument: Konzument pomoću ChannelCactory klase 12](#_Toc356221492)

[Servis konzument: Konzument pomoću proxy klase – WSDL –Visual Studio Add Service Reference 13](#_Toc356221493)

[Servis konzument: Konzument pomoću proxy klase – WSDL –SvcUtil command prompt 16](#_Toc356221494)

[Servis konfiguracija: Service Configuration Editor alat - SvcConfigEditor.exe 17](#_Toc356221495)

[Servis konfiguracija: Dijagnostika i Tracing 19](#_Toc356221496)

[Servis ugovor: Upravljenje ugovorom – nekoliko primjera iz vježbi 22](#_Toc356221497)

[Servis ugovor: Verzioniranje 24](#_Toc356221498)

[Servis i rad sa greškama: includeExceptionDetailsInFaults 25](#_Toc356221499)

[Servis i rad sa greškama: bacati FaultException - throw naše greške 26](#_Toc356221500)

[Servis i rad sa greškama: stronglytyped custom Fault tip 27](#_Toc356221501)

[Servis i Načini komunikacija: OneWay 29](#_Toc356221502)

[Servis i Načini komunikacija: Načini komunikacija: Duplex 30](#_Toc356221503)

[Servis security: podešavanje default konfiguracije na wsHttpBinding 31](#_Toc356221504)

[Servis security: Transport Security - Postavljenje SSL-a na IIS 38](#_Toc356221505)

[Servis Security: Transport Security i načini autentifikacije 43](#_Toc356221506)

[Servis security: Transport security + Anonymous autentifikacija – spajanje konzumenta na servis bez slanja credentials-a (anonymous) 44](#_Toc356221507)

[Servis security: Transport security i Windows Credentials autentifikacija 47](#_Toc356221508)

[Servis security: Transport security + Basic security - šaljemo podatke o useru – predstavljamo se kao određeni windows user 49](#_Toc356221509)

[Servis security: TransportWithMessage Security - Transport security + Message Username credentials - sami implementiramo username/password provjeru 51](#_Toc356221510)

[Servis security – razmjena ključeva 53](#_Toc356221511)

[Dodatno – Pobrojani security scenariji sa kombinacijama zaštite 56](#_Toc356221512)

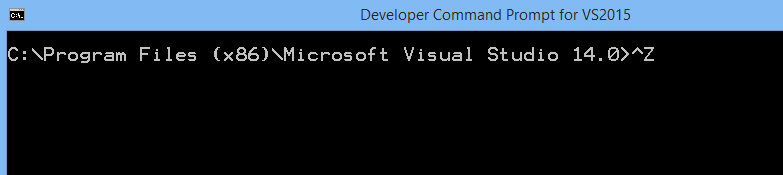
[Dodatno: Demo skripta –Web HTTP programiranje – REST na WCF 57](#_Toc356221513)

[Dodatno: Demo skripta – WCF i JSON 59](#_Toc356221514)

[Dodatno: Demo skripta - WCF Sindikacija 61](#_Toc356221515)

# Osnovne upute

* Znakovima ## označeni su placeholderi za nazive koji se mijenjaju ovisno o ciljanim imenima klasa, sučelja i aplikacija
* VS = Visual Studio
* VS Command Prompt = Developer Command Prompt for VS
  + - Otvara command prompt za podešenim systemskim i .NET pathovima tako da su nam alati dostupni iz bilo kojeg foldera

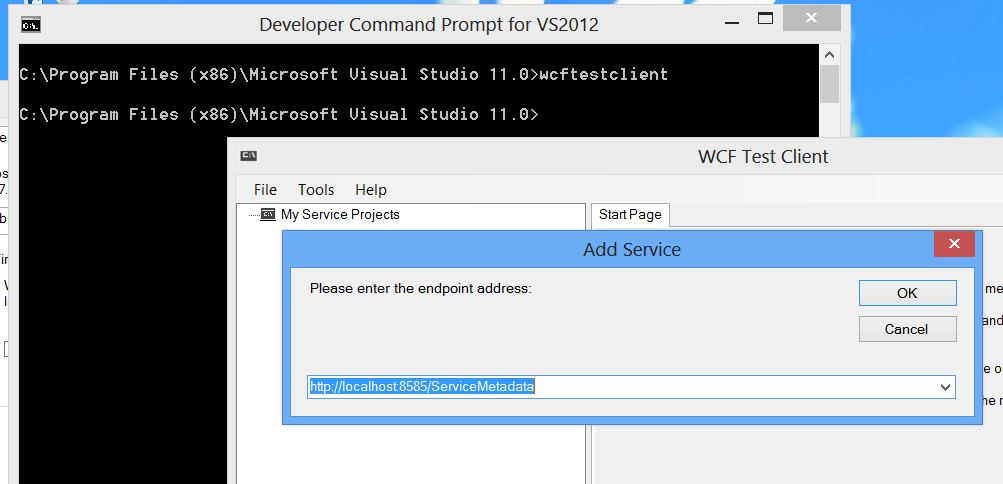


# Servis: Kreiranje WCF servisne biblioteke– WCF Service Library

1. file->new->project->c#->wcf service library – naziv #NazivServisa#
2. Kreirano je:
   * IService1.cs
   * CompositeType klasa
   * Service1.cs
   * app.config
3. Po potrebi preimenovati klase i datoteke Service => #NoviNazivServisa#
4. U sučelju/intarfece-u I#NoviNazivServisa#.cs obrisati sve „default“ metode i tipove i dodati svoje
   * [OperationContract] atribut na metode u sučelju
   * Na [ServiceContract] dodati Namespace i postaviti na imenski prostor novoga servisa npr. <http://www.imetvrtke.hr/imeaplikacije/verzija>
5. Implementirati interface u klasi #NoviNazivServisa#.cs
   * Napisati korisni kod
6. pokrenuti sa F5, pokreće se wcf test client, provjera rada biblioteke

# Servis: testiranje servisa - WCF test client – manualno pokretanje

1. WCF test client – alat za testiranje web servisa po dostupnom WSDL dokumentu
2. Otvoriti Visual Studio command prompt
3. Pokrenuti wcftestclient.exe
4. Dodati URL koji pokazuje na WSDL document
   1. File – Add Service



1. Izabrati metodu i testirati

# Servis hosting: Self-hosting u command line aplikacija

1. Kreirati command prompt aplikaciju naziva HostServisaKonzola
2. Dodati referencu na projekt servisne biblioteke - #NazivServisa# projekt
3. Dodati referencu na System.ServiceModel
   1. Add Reference System.ServiceModel
4. Dodati kod za self hosting I#NazivServisa# servisa – puna imena klase i interfejsa su iz primjera => prilagoditi

using (ServiceHost host = new ServiceHost(typeof(MoviesWS.SOAPService.PublicService), new Uri("http://127.0.0.1:8989/Movies")))

{

host.AddServiceEndpoint(typeof(MoviesWS.Contracts.IPublicService), new BasicHttpBinding(), "Public");

host.Open();

Console.WriteLine("Servis pokrenut. ENTER za zaustavljanje servisa!");

Console.ReadLine();

}

1. Pokrenuti Command Line hostani servis
2. Pokrenuti WCF Test Client iz VS command prompta – wcftestclient – dodati servis sa publiciranom adresom http://127.0.0.1:8989/Movies i testirati.
3. Otvoriti u browseru – pogledati WSDL

# Servis hosting: Hosting u Windows Servisu

1. U VS solution u kojem se nalazi servisna biblioteka dodati projekt tipa Windows Service nazvati ga HostWindowsService
2. Promijeniti ime klase u #NazivServisa#WinServis
3. Dodati reference:
   * System.ServiceModel - .NET biblioteka
   * Servisna biblioteka projekt – projekt u kojem se nalazi #NazivServisa# servis
4. U kontruktoru WinServis klase postaviti ime servisa
   * this.ServiceName = "#NazivServisa#WinServis";
5. Implementirati kod – isto kao konzolni host samo nema interakcije sa korisnikom – koristiti EventLog za zapis – piše automatski u Application Event Log

public partial class KalkulatorWinServis : ServiceBase

{

public #NazivServisa#WinServis()

{

InitializeComponent();

this.ServiceName = "#NazivServisa#WinServis";

}

public ServiceHost #NazivServisa#Host= null;

protected override void OnStart(string[] args)

{

if (#NazivServisa#Host== null)

{

try

{

//instanciranje sa tipom MojPrviServis - imeplementacija

#NazivServisa#Host = new ServiceHost(typeof(MojPrviServis.MojPrviServis));

//Instanca krajnje točke - interface, binding, adresa

//naučiti preseliti u config - pogledati demo za web server hosting

ServiceEndpoint #NazivServisa#Endpoint = #NazivServisa#Host.AddServiceEndpoint(

Typeof#(NamespaceServisa#.I#NazivServisa#),

new WSHttpBinding(),

"http://localhost:9090/#NazivServisa#Service");

//Otvaranja kanala

#NazivServisa#Host.Open();

EventLog.WriteEntry(String.Format("Otvorio sam kanal na {0} {1}",

#NazivServisa#Endpoint.Address.ToString(),

#NazivServisa#Endpoint.Binding.Name)

);

}

catch (Exception ex) {

EventLog.WriteEntry(String.Format("Došlo je do greške {0}",

ex.ToString()));

}

}

}

protected override void OnStop()

{

if (#NazivServisa#Host!= null)

{

#NazivServisa#Host.Close();

#NazivServisa#Host = null;

EventLog.WriteEntry("Zatvorio servis");

}

}

}

1. Dodati installer klasu – otvoriti #NazivServisa#WinServis.cs u design view
   * desni klik Add Installer – mogli smo i na ruke add class pa naslijediti Installer...
   * ServiceInstaller1 komponenta – desni klik Properties – podesiti ServiceName – #NazivServisa#WinServis
   * podesiti StartType – Automatic ili Manual
   * ServiceProcessINstaller1 komponenta – desni klik Properties
   * podesiti account na Local System
2. Instalacija servisa
   * Otvoriti VS command prompt
   * installutil HostWindowsServis.exe
   * provjeriti da li je instalacija uspjela
   * otvoriti Local Services panel
   * start/stop – pogledati servis
   * Startati servis
3. Testirati uz pomoć WCF test client aplikacije

# Servis hosting: Hosting u Web aplikaciji na web serveru

1. U #NazivServisa#Servis solution dodati novi project – Web – Empty ASP.NET web application naziva HostServisaIIS
2. Desni klik na projekt pa Properties
   1. u Web tabu podesiti da se izvršava na IIS-u
   2. VAŽNO: da bi VS kreirao IIS virtualni direktorij mora VS biti pokrenut As Administrator
   3. postaviti da adresa bude http://localhost:9191/HostServisaIIS
3. U web projekt dodati krajnju točku servisa:
   1. Desni klik na web projekt pa Add New Item
   2. Izabrati WCF Service predložak
   3. Dati naziv krajnjoj točki #NazivServisa#.svc
4. Zato što imamo servisnu biblioteku kreirani „višak“ datoteka brišemo
   1. Obrisati svc.cs – ne treba nam
   2. I#NazivServisa#.cs – ne trba nam
5. Dodati referencu na projekt servisne biblioteke
   1. Add Reference – iz popisa projekata odabrati #NazivServisa# projekt
6. U #NazivServisa#.svc deklaraciji maknuti codebehind atribut
7. U #NazivServisa#.svc deklaraciji promijeniti class atribut na #NamespaceServisa#. #NazivServisa#.
8. <%@ ServiceHost Language="C#" Debug="true" Service="#NamespaceServisa#. #NazivServisa# " %>
9. Otvoriti svc krajnju točku u browseru
   1. Desni klik na #NazivServisa#.svc pa View in Browser
   2. pogledati WSDL – dodati „?wsdl“ u URL
10. Opcionalno: Konfiguracija – promjena iz basicHttpBinding u wsHttpBinding
    1. Kopirati services element iz app.configa servisne biblioteke u app.config web aplikacije
    2. Ostaviti samo jedan service element za wsHttpBinding i Icalc interface

<services>

<service name="#NamespaceServisa#. #NazivServisa#">

<endpoint address="" binding="wsHttpBinding" contract="#NamespaceServisa#. I#NazivServisa#">

<identity>

<dns value="localhost" />

</identity>

</endpoint>

</service>

</services>

1. Testirati servis kroz WCF test client

# Servis konzument: Konzument pomoću ServiceHost klase

1. Dodatni novi projekt tipa command line app u VS solution gdje se nalazi #NazivServisa# projekt servisne biblioteke
2. Nazvati ga KonzumentServiceFactory
3. Dodati referencu na System.Service model
   1. Add Reference System.ServiceModel
4. Add Reference na #NazivServisa# projekt
   1. Na ovaj način se ruši SOA princip jer dijelimo servisnu klasu a ne samo WSDL ugovor ali će biti lakše raditi jer nećemo morati paziti na verzioniranje

using (ServiceHost host = new ServiceHost(typeof(MoviesWS.SOAPService.PublicService), new Uri("http://127.0.0.1:8989/Movies")))

{

host.AddServiceEndpoint(typeof(MoviesWS.Contracts.IPublicService), new BasicHttpBinding(), "Public");

host.Open();

Console.WriteLine("Servis pokrenut. ENTER za zaustavljanje servisa!");

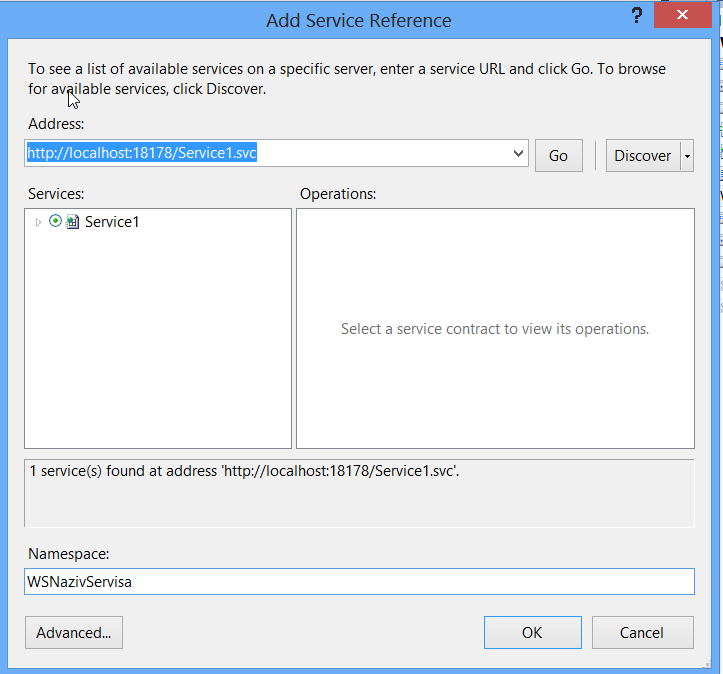
Console.ReadLine();

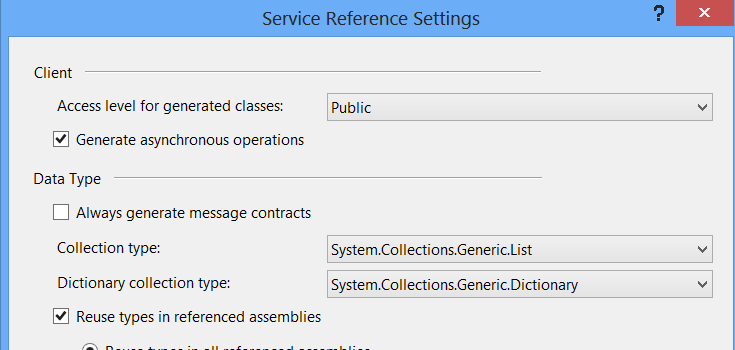
}

1. Testirati – promjene na Interface-u servisa se vide u intellisensu i debug-u
   1. nije „slabo povezano“ – nije prava SOA arhitektura
   2. ovo možemo samo ako imamo servis i interface zajedno u našem razvoju

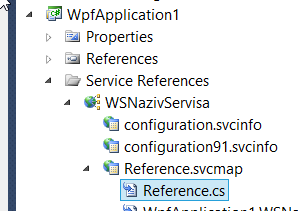
# Servis konzument: Konzument pomoću proxy klase – WSDL –Visual Studio Add Service Reference

1. Otvoriti novu instancu Visual Studio alata
   1. Može i isti Solution gdje je servisna biblioteka
   2. za primjer kreiramo novi da se naglasi razlika u odnosu na korištenje ChannelFactory klase
   3. sada dijelimo samo ugovor odnosno WSDL a ne direktno projekte
2. Kreirati novi project Windows Forms naziva KonzumentWinForms
   1. Umjesto Windows Forms može i bilo koji drugi projekt – daljnji postupak je potpuno isti
3. Kreirati takozvanu proxy klasu preko koje pristupamo servisu:
   1. Dodati servisnu referencu
      1. Add Service Reference
      2. Upisati adresu do wsdl-a <http://localhost[port].svc?imeservisa?wsdl> pa GO
      3. Ukoliko je servis dostupan i WSDL dostupan možemo vidjeti dijalog kao na slici ispod



* + 1. Postaviti vrijednost za imenski prostor Namespace = Ws#NazivServisa#
  1. Otvoriti Advanced dijalog – dolje lijevo
     1. postaviti da se kreiraju async verzije metoda
        1. Generate asynchronous operations
     2. tip podataka neka se generira kao generička lista – Generic.List kao na slici ispod:
     3. 

1. Pregled generirane proxy klase ispod servisne reference u VS solution stablu – ako se ne vidi izabrati Show All Files u zaglavlju Solution Explorera



1. Poziv servisa preko proxy klase
   1. Instanciranje proxy klase

WSNazivServisa.#ImeServisa#Client proxy = new WSNazivServisa.#ImeServisa#Client();

1. poziv metoda sinkrono

int rezultat = proxy.#ImeMetoda#(2,3);

1. prikaz rada sa asinkronim metodama

private void button1\_Click(object sender, EventArgs e)

{

proxy.AddCompleted += new EventHandler<WSNazivServisa.#ImeMetode#CompletedEventArgs>(proxy\_#ImeMetode#Completed);

proxy.#ImeMetode#Async();

}

void proxy\_#ImeMetode#Completed(object sender, WSNazivServisa.#ImeMetode#CompletedEventArgs e)

{

int r = e.Result;

textBox1.Text = r.ToString();

}

Ili Async/Await – ako je u CMD onda treba ovaj Task.Run async. Inače se metoda u kojoj je await markira kao async.

Task.Run(async () =>

{

// Do any async anything you need here without worry

var res = await proxy.GetMovieAsync(3);

Console.WriteLine(res.Name);

}).Wait();

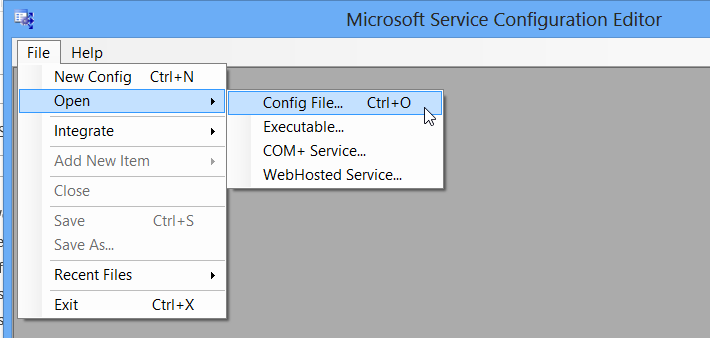
1. Promjene na interface-u servisa zahtjevaju rekreiranje proxy klase
   1. Desni klik na servisnu referencu pa Update Service Reference

# Servis konzument: Konzument pomoću proxy klase – WSDL –SvcUtil command prompt

1. Otvoriti VS Command Prompt
2. Pokrenuti svcuutil.exe
   * 1. Pregled opcija
3. kreirati proxy klasu naredbom:
   * 1. svcutil PathDoWsdl-a
     2. svcutil <http://localhost[port].svc?imeservisa?wsdl>
4. Rezultat je kreirana \*.cs i \*.output rezultat
5. imeservisa.cs => je proxy klasa koja nastaje i kada radimo Add Service Reference
6. korištenje proxy klase
   * 1. dodati imeservisa.cs u projekt gdje želimo koristiti
     2. dalje jednako kao i sa Add Service Reference

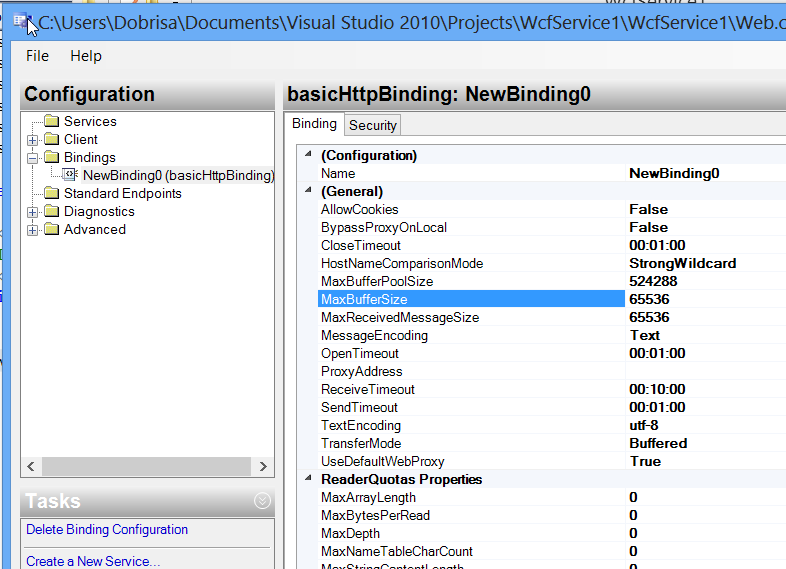
# Servis konfiguracija: Service Configuration Editor alat - SvcConfigEditor.exe

1. Za podešavanje servisne konfiguracije u web.config odnosno app.config uz manualno XML editiranje dostupan je i specijalizirani alat za tu namjenu – Service Configuration Editor
2. Ovisno o tipu projekta može biti dostupan na desni klik na web.config pa Edit Configuration
3. Manualno pokretanje
   * 1. VS Command Prompt
     2. Izvršiti SvcConfigEditor.exe
     3. Otvara se aplikacija



* + 1. Sa File -> Open -> Config File potražiti \*.config datoteku i otvoriti je

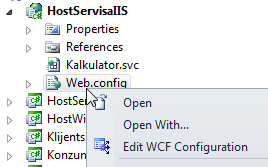
1. Primjer: za povećati maksimalnu veličinu poruka
   * 1. Otvoriti Binding node
     2. Označiti željeni binding
     3. Proučiti atribute
        1. \*MaxReceivedMessageSize\* i \*MaxBufferSize\* povečati vrijednosti



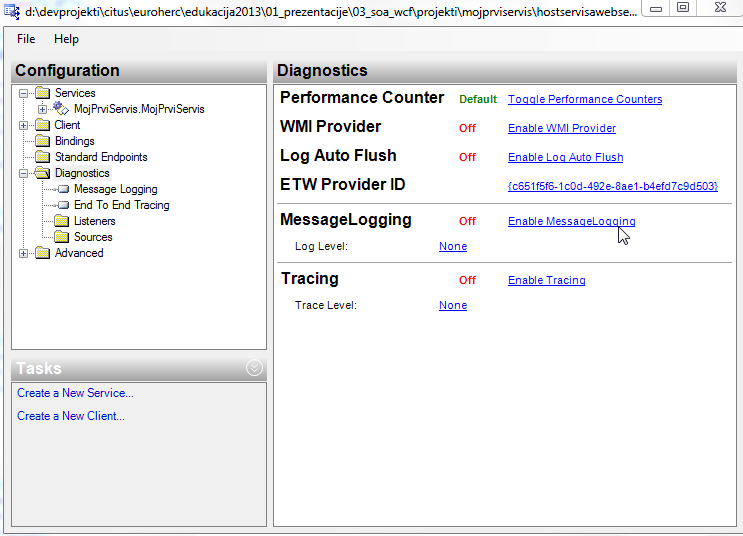
1. File -> Save
2. Pogledati u VS-u promjenu

# Servis konfiguracija: Dijagnostika i Tracing

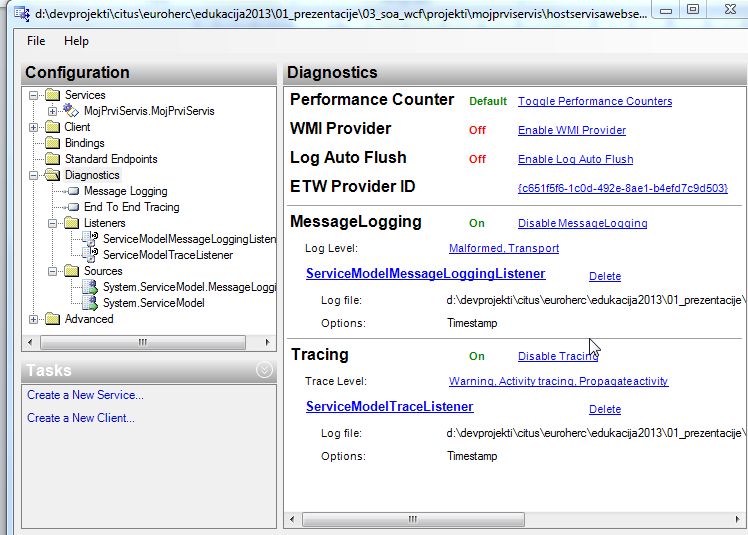
1. U projektu web aplikacije u kojoj radimo hosting servisa
2. Desni klik na web.config pa Edit WCF Configuration



1. Napomenuti - može i ovako: VS – Tools – WCF Service Configuration Editor (ili VS command prompt SvcConfigEditor) – kako smo ga već koristili za povećavanje kvote veličine poruke. Ponekad se VS zbuni pa ne ponudi Edit shortcut.
2. Bilo 3 ili 4 otvaramo web.config iz web projekta u WCF config editoru
3. Podesiti Diagnostics
   1. kliknuti na Enable MessageLogging i Enable Tracing



Nakon akcija ekran izgleda kao:



Kreirani su listeneri za prikupljanje poruka.

1. Spremiti promjene sa File –> Save
2. Pogledati web.config u VS-u
   1. vidimo dodani element diagnostics

<system.diagnostics>

<sources>

<source name="System.ServiceModel.MessageLogging" switchValue="Warning, ActivityTracing">

<listeners>

<add type="System.Diagnostics.DefaultTraceListener" name="Default">

<filter type="" />

</add>

<add name="ServiceModelMessageLoggingListener">

<filter type="" />

</add>

</listeners>

</source>

<source name="System.ServiceModel" switchValue="Warning, ActivityTracing"

propagateActivity="true">

<listeners>

<add type="System.Diagnostics.DefaultTraceListener" name="Default">

<filter type="" />

</add>

<add name="ServiceModelTraceListener">

<filter type="" />

</add>

</listeners>

</source>

</sources>

<sharedListeners>

<add initializeData="d:\devprojekti\citus\euroherc\edukacija2013\01\_prezentacije\03\_soa\_wcf\projekti\mojprviservis\hostservisawebserver\web\_messages.svclog"

type="System.Diagnostics.XmlWriterTraceListener, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"

name="ServiceModelMessageLoggingListener" traceOutputOptions="Timestamp">

<filter type="" />

</add>

<add initializeData="d:\devprojekti\citus\euroherc\edukacija2013\01\_prezentacije\03\_soa\_wcf\projekti\mojprviservis\hostservisawebserver\web\_tracelog.svclog"

type="System.Diagnostics.XmlWriterTraceListener, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"

name="ServiceModelTraceListener" traceOutputOptions="Timestamp">

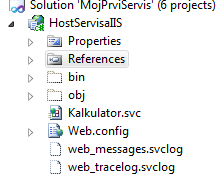
<filter type="" />

</add>

</sharedListeners>

</system.diagnostics>

1. Nakon novih poziva prema servisu koje možemo odraditi ili iz test client alata ili iz aplikacije konzumenta vidimo dvije nove datoteke dva svclog file-a



1. Otvoriti u VS – XML – log datoteke
2. Za pregled SVC log datoteka koristimo alata Svc Trace Viewer
   1. VS command prompt – naredba svctraceviewer
   2. File > Open – tražimo svclog datoteke iz web projekta
3. Otvoriti message log
4. Pregledati poruke
5. Otvoriti trace log
6. pregledati aktivnosti

# Servis ugovor: Upravljenje ugovorom – nekoliko primjera iz vježbi

1. Ime servisa i imenski prostor u ICalc.cs interface-u
   1. [ServiceContract(Name="Kalkulator", Namespace="http://www.euroherc.hr/kalkulator/ver1")]
2. Naziv metode definiran kroz atribut

[OperationContract(Name="Zbroji")]

int Add(int a, int b);

1. Nije ograničeno na Interface npr. na klasi implementacije

[ServiceBehavior(Name="MojPrviServis", Namespace="http://www.euroherc.hr/MojPrviServis/ver1")];

public class MojPrviServis : IMojPrviServis, ICalc

1. Dodati u ICalc metodu i ispod odmah klasu označenu kako DataContract

[OperationContract]

KompozitniOdgovor Echo();

}

[DataContract]

public class KompozitniOdgovor {

[DataMember]

public bool Obavljeno { get; set; }

[DataMember]

public string Poruka { get; set; }

}

1. Ukrasiti klasu KompozitniOdgovor sa MessageContract atributima

[DataContract]

[MessageContract]

public class KompozitniOdgovor {

[DataMember]

[MessageHeader]

public bool Obavljeno { get; set; }

[DataMember]

[MessageBodyMember]

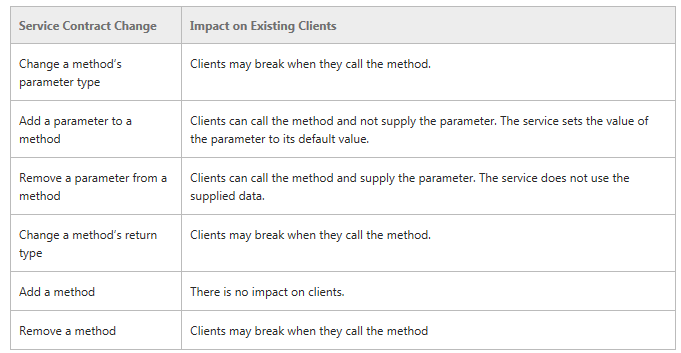
public string Poruka { get; set; }

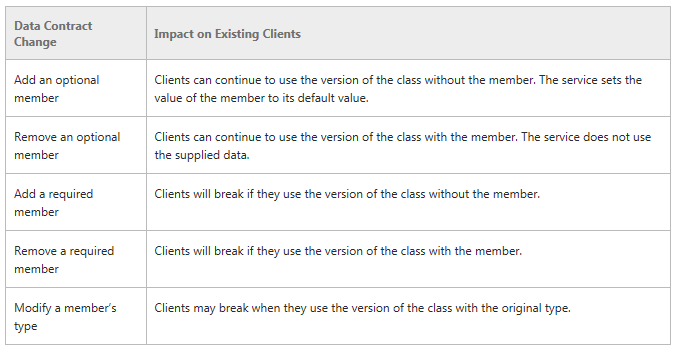
}

* 1. sa MessageHeader kažemo da Obavljeno ide u Header element SOAP poruke
  2. Sa MessageBodyMember definiramo da Poruka ostaje u Body element SOAP poruke
  3. Build pa WSDL search Obavljeno – pogledati kako se definira drugačije za smještaj u Header
  4. Zakomentirajte Message attribute da ne smetaju za kasnije

# Servis ugovor: Verzioniranje

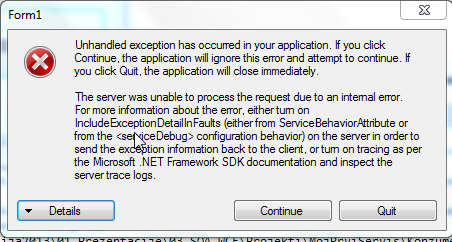
Popis sa MSDN-a – kako promjena u kodu odnosno u ugovoru utjeće na konzumenta servisa:



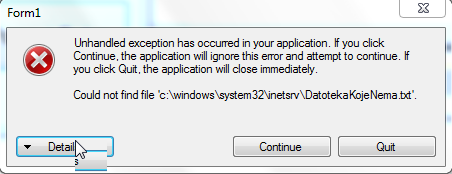


# Servis i rad sa greškama: includeExceptionDetailsInFaults

1. includeExceptionDetailInFaults
2. atribut konfiguracije kojiim možemo dopustiti servisu da šalje puni stack podataka o pogrešci konzumentu
3. ovo je za produkciju neprihvatljivo i može se koristiti samo za debug i test
4. U web.config dokumentu potražiti liniju:
   1. <serviceDebug includeExceptionDetailInFaults="false" />
   2. Ako je nema kreirati uz konfiguraciju servisa
   3. Ako je false poruka o grešci je kao na slici ispod:



* 1. Ako je na true poruka o grešci je kao na slici ispod:



# Servis i rad sa greškama: bacati FaultException - throw naše greške

1. Ispravan način prosljeđivanja grešaka sa servisa je bacanje servisne greške korištenjem FaultException klase
2. kreira se Fault SOAP element i uredno putuje do klijenta sa porukom kakvu želimo poslati a ne sistemskom koja može otkriti ono što ne želimo
3. Obavezno na false: <serviceDebug includeExceptionDetailInFaults="**false**" />
4. Primjer bacanja greške
   1. Radimo try/catch i ako ulovimo grešku radimo bacanje nove greške tipa FaultException u kojoj šaljemo poruku kakvu želimo odnosno kakvu zahtjeva naša poslovna logika

try

{

//datoteke nema - dolje kod mora puknuti

fi.OpenRead();

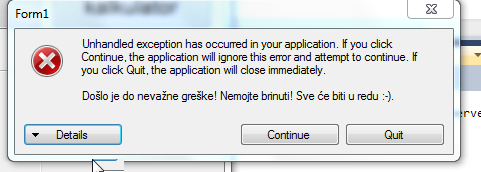
}

catch {

throw new FaultException("Došlo je do nevažne greške! Nemojte brinuti! Sve će biti u redu :-)");

}

1. Konzument vidi samo našu grešku – ne i internu sa servisa



# Servis i rad sa greškama: stronglytyped custom Fault tip

1. Ukoliko želimo dodatno postaviti našu klasu koju ćemo slati u FAULT omotaču unutar SOAP poruke to možemo odraditi kroz sljedeće korake
   1. definiramo DataContract klasu u interface definiciji servisa na #NazivServisa# projektu

/// <summary>

/// StronglyTyped Fault

/// </summary>

[DataContract]

public class MojFault

{

[DataMember]

public string Naziv { get; set; }

[DataMember]

public string Detalji { get; set; }

}

* 1. U istom interface-u označimo operaciju atributom kojim dodajemo našu klasu kao dio FaultException klase – sa FalutContact atributom

[OperationContract]

[FaultContract(typeof(MojFault))]

KompozitniOdgovor NekaMetoda();

* 1. U implementaciji metode instanciramo našu Fault klasu i mapiramo je u FaultException

try

{

//datoteke nema - dolje kod mora puknuti

fi.OpenRead();

}

catch

{

//bacamo grešku našega tipa

var mojFault = new MojFault();

mojFault.Naziv = "NEMA DATOTEKE";

mojFault.Detalji = "puno još detalja o grešci mogu proslijediti";

throw new FaultException<MojFault>(mojFault);

}

* 1. Na klijentu (PAŽNJA: mijenjali smo DataContract – obavezna nam je Update Service Reference akcija)

try

{

ZaZvatiServis.KompozitniOdgovor rezultat = proxy.NekaMeotda();

}

catch (FaultException<ZaZvatiServis.MojFault> fe)

{

MessageBox.Show(fe.Detail.Naziv);

}

1. Unutar Detail svojstva FaultException klase ugnijezdila se naša Fault klasa sa svojim svojstvima

# Servis i Načini komunikacija: OneWay

1. Tipično korištenje OneWay operacije – pisanje log-a koji ne zahtjeva čekanje odgovora
2. Primjer sa dvije sekunde pauze i pozivom iz Windows Forms aplikacije:
   1. U projektu #NazivServisa# u interface definiciji servisa stavljamo novu metodu ZapisiLog

[OperationContract]

void ZapisiLog(string poruka);

* 1. Implementirati ZapisiLog u #NazivServisa#.cs

public void ZapisiLog(string poruka)

{

Thread.Sleep(2000);

//jedna linija za dodati text u datoteku - ako ne postoji kreirati će

poruka = System.Environment.NewLine + DateTime.Now.ToString() + ": " + poruka;

System.IO.File.AppendAllText(@"C:\Users\Public\MojPrviServisLog.txt", poruka);

}

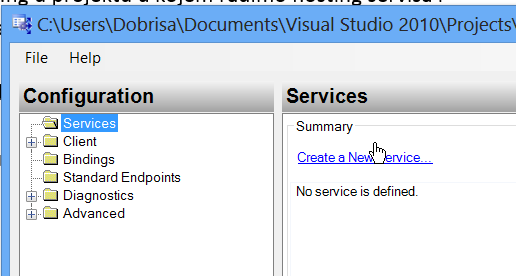
* 1. Update ServiceReference u konzumentu
  2. Kreirati Button na UI formi koji radi Piši u Log – poziv ZapisiLog operacije na servisu preko proxy klase
  3. Isprobati by default – request/reply – 2 sekunde je sučelje zablokirano.
  4. Dodati na OperationContract atribut **IsOneWay=true**
  5. Rebuild, Update Service Reference, isprobati
  6. Sučelje je slobodno čim se odradi request i dobije odgovor na komunikacijskom kanalu

# Servis i Načini komunikacija: Načini komunikacija: Duplex

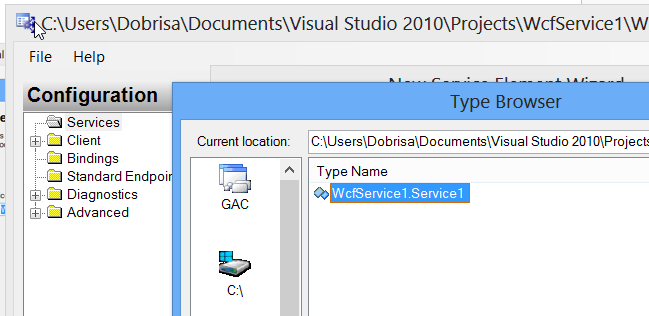
Primjer se nalazi u prilogu edukaciji u VS solution-u sa WCF primjerima

# Servis security: podešavanje default konfiguracije na wsHttpBinding

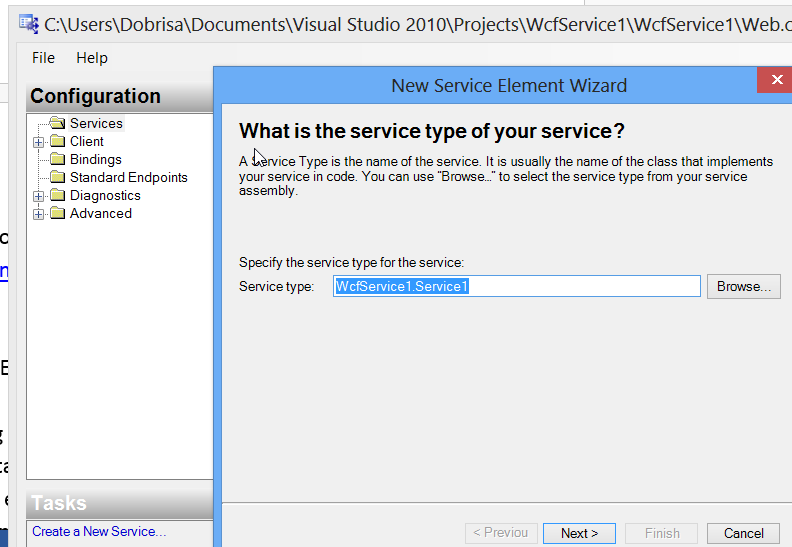
1. Na novo kreiranim WCF projektima default vrijednosti će biti postavljene na basicHttpBinding te u konfiguracijskih datotekama neće biti posebno definirani servis, krajnja točka ili binding konfiguracije
2. Da bi podešavali razne konfiguracijske postavke servisa inicijalno je dobro kreirati servis i osnovne postavke korištenjem Svc Configuration Editor alata
3. Kreiranje osnovnih postavi korak po korak:
   1. Podešavamo web.config u projektu u kojem radimo hosting servisa i gdje se nalazi naša krajnja točka \*.svc
   2. desni klik na web.config pa Edit WCF Configuration
      1. ili VS command prompt pa svcconfigeditor.exe – vidi poglavlje “Servis konfiguracija: Service Configuration Editor alat - SvcConfigEditor.exe”
   3. Create New Servis



* 1. Browse do library dll-a – tipično bin/debug pa izabrati DLL te odmah izabrati i interface

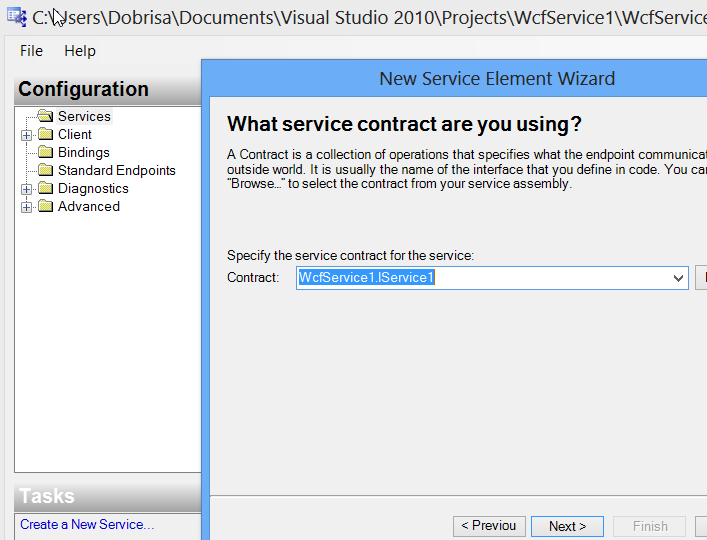


* 1. Izabrati klasu



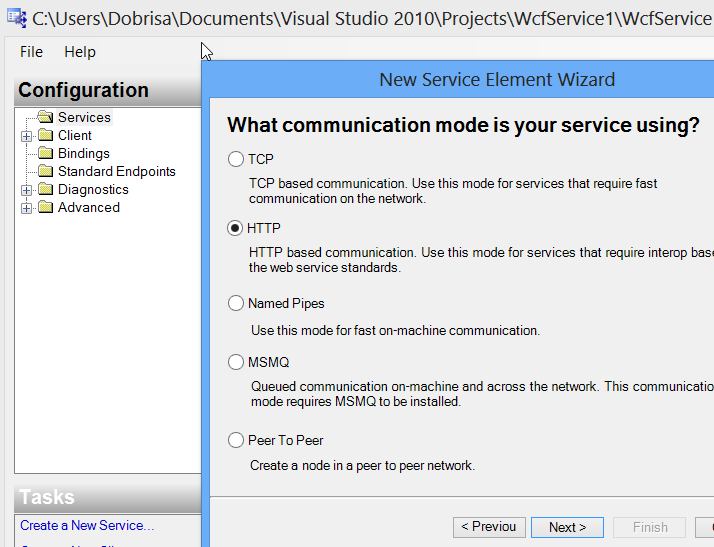
Next:

* 1. Izabrati Inteface



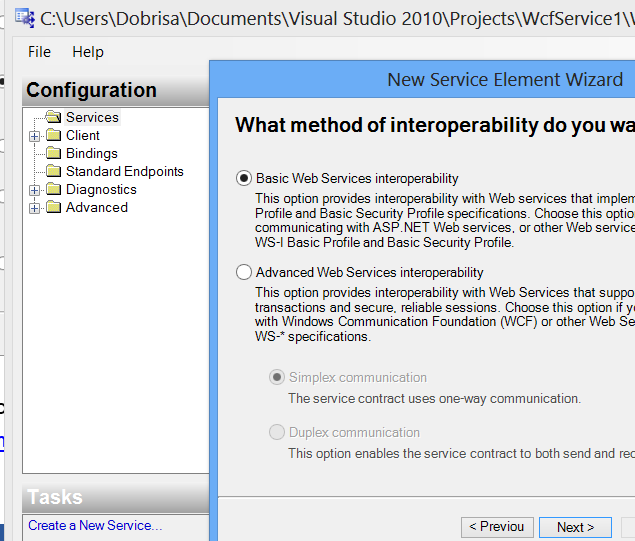
Next:

* 1. HTTP



Next:

* 1. Basic WS



Next:

* 1. Podešavamo krajnju točku - endpoint – gdje radimo hosting servisa = <http://server:port/virtualnidirektorij/krajnjatocka.svc>
  2. Finish
  3. Save
  4. Pogledati promjene u Visual Studio alatu – web.config
  5. Obavezno promijeniti multipleSiteBindingsEnabled iz true u false jer smo stavili aspolutnu putanjeu do servisa. Kada je postavljeno na true očekuje se relativna adresa do servisa

<serviceHostingEnvironment multipleSiteBindingsEnabled="true" />

* 1. Otvoriti svc u browseru – provjeriti da li je sve u redu

1. Promijeniti basicHttpBinding u wsHttpBinding – mijenjati kroz config endpointa

<endpoint address="http://localhost:18178/Service1.svc" binding="**basic**HttpBinding"

bindingConfiguration="" contract="WcfService1.IService1" />

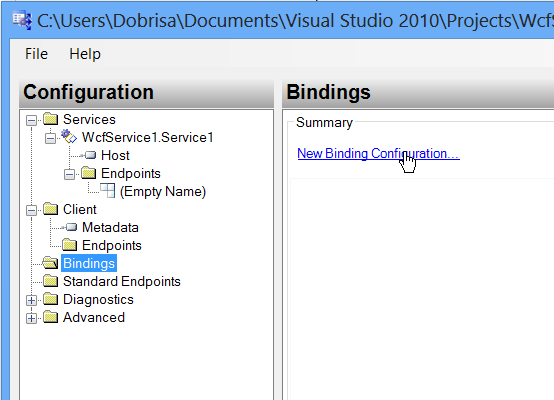
u

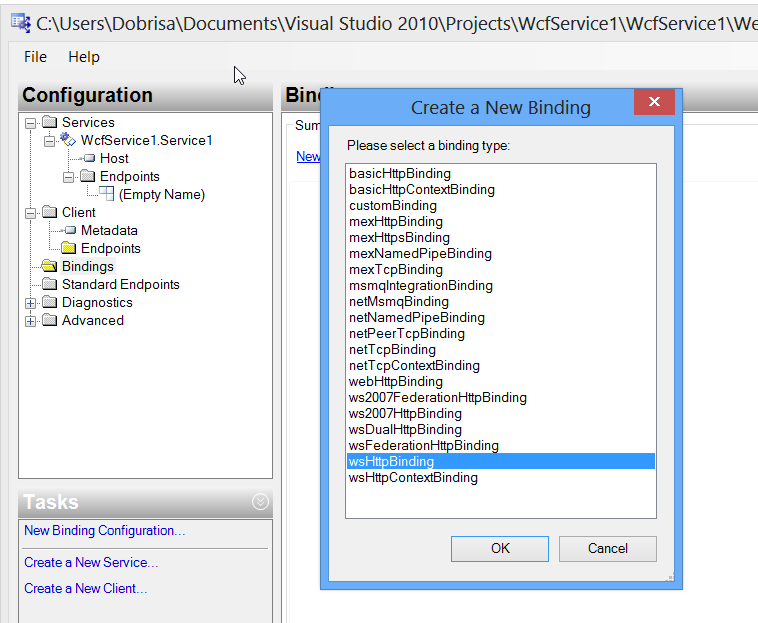
<endpoint address="http://localhost:18178/Service1.svc" binding="**ws**HttpBinding"

bindingConfiguration="" contract="WcfService1.IService1" />

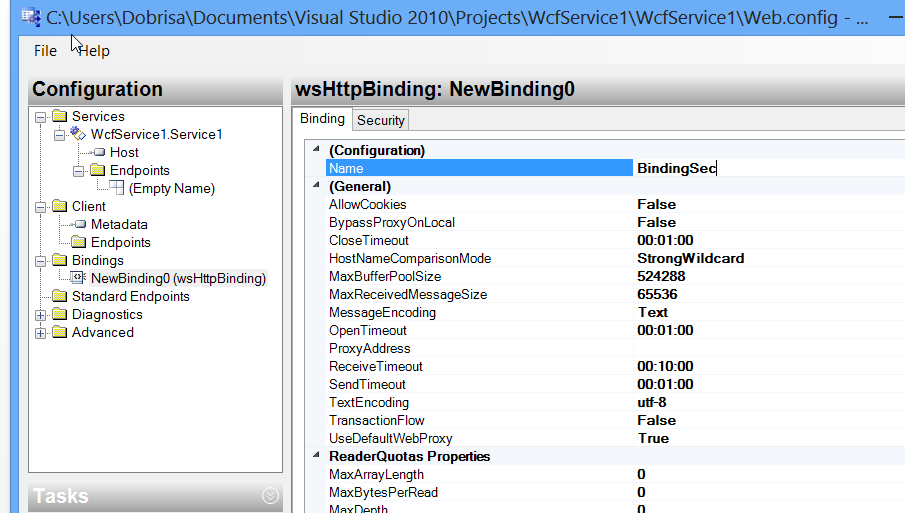
Razlika:

* + 1. basicHttpBinding = clear text
    2. wsHttpBinding enkriptira sa SPNEGO - **Simple and Protected** [**GSSAPI**](http://en.wikipedia.org/wiki/GSSAPI) **Negotiation Mechanism** (**SPNEGO**), often pronounced "spen-go", is a [GSSAPI](http://en.wikipedia.org/wiki/Generic_Security_Services_Application_Program_Interface) "pseudo mechanism" that is used to negotiate one of a number of possible real mechanisms. SPNEGO is used when a client application wants to authenticate to a remote server, but neither end is sure what authentication protocols the other supports. The pseudo-mechanism uses a protocol to determine what common GSSAPI mechanisms are available, selects one and then dispatches all further security operations to it. This can help organizations deploy new security mechanisms in a phased manner.
    3. SPNEGO – kod prvog podizanja konzumenta tri poziva – svojevrsni handshake i razmjena podataka potrebnih za eknkripciju

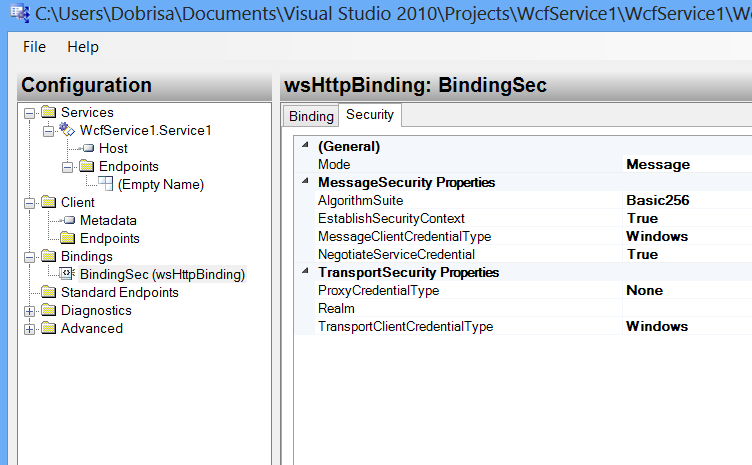
1. Kroz SvcConfigEditor dodati osnovnu definiciju za binding – zbog toga da ga možemo fino podešavati ovisno o tipu autentifikacije i autorizacije
   1. Otvoriti Web.config u svcconfigeditor alatu
   2. Odabrati Bindings u stablu pa New Biinding Configuration
   3. Izabrati wsHttpBinding sa liste ponuđenih



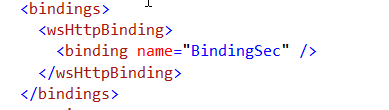
* 1. Postaviti ime za binding



* 1. Pogledati security tab – ovo su default postavke – da ništa ne mijenjamo wsHttpBinding radi Message/Windows security



* 1. Spremiti promjene sa File -> Save
  2. Pogledati web.config u VS-u
  3. Vidimo novi wsHttpBinding element u bindings elementu



* 1. Možemo ga raspisati i šire kao ispod u primjeru – ali pošto su ovo default vrijednosti rezultat je isti

<wsHttpBinding>

<binding name="BindingSec">

<security mode="Message">

<transport clientCredentialType="Windows"

proxyCredentialType="None"

realm="" />

<message clientCredentialType="Windows"

negotiateServiceCredential="true"

algorithmSuite="Default"

establishSecurityContext="true" />

</security>

</binding>

</wsHttpBinding>

1. Pridružiti binding konfiguraciju krajnjoj točki kroz konfiguraciju. Da bi krajnja točka uključila binding konfiguraciju kreiranu u koraku prije potrebno je u atribut endpoint elementa podesiti atribut bindingConfiguration te ga postaviti na ime binding kongifuracije. U našem primjeru to je BindingSec. Primjer je ispod:

<bindings>

<wsHttpBinding>

<binding name="**BindingSec**" />

</wsHttpBinding>

</bindings>

<services>

<service name="WcfService1.Service1">

<endpoint address="http://localhost:18178/Service1.svc" binding="wsHttpBinding"

bindingConfiguration="**BindingSec**" contract="WcfService1.IService1" />

</service>

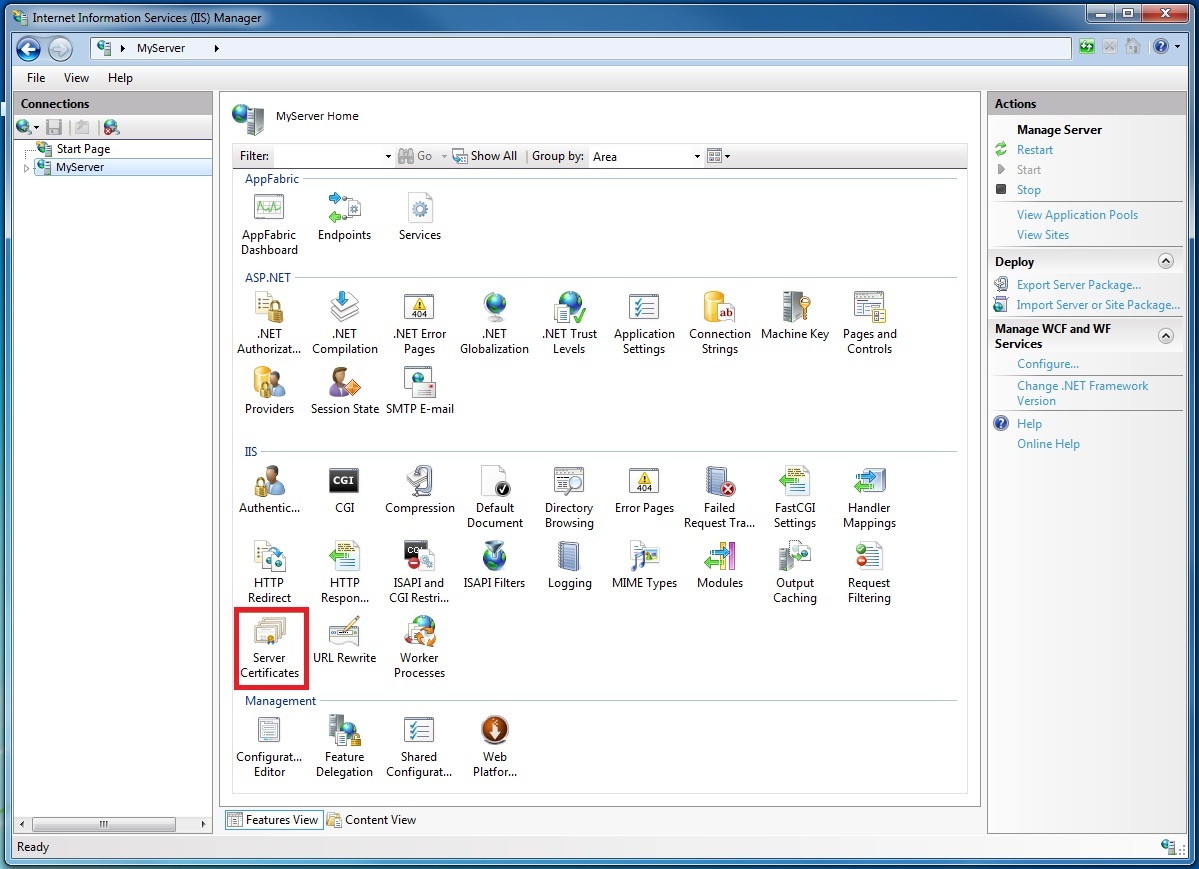
</services>

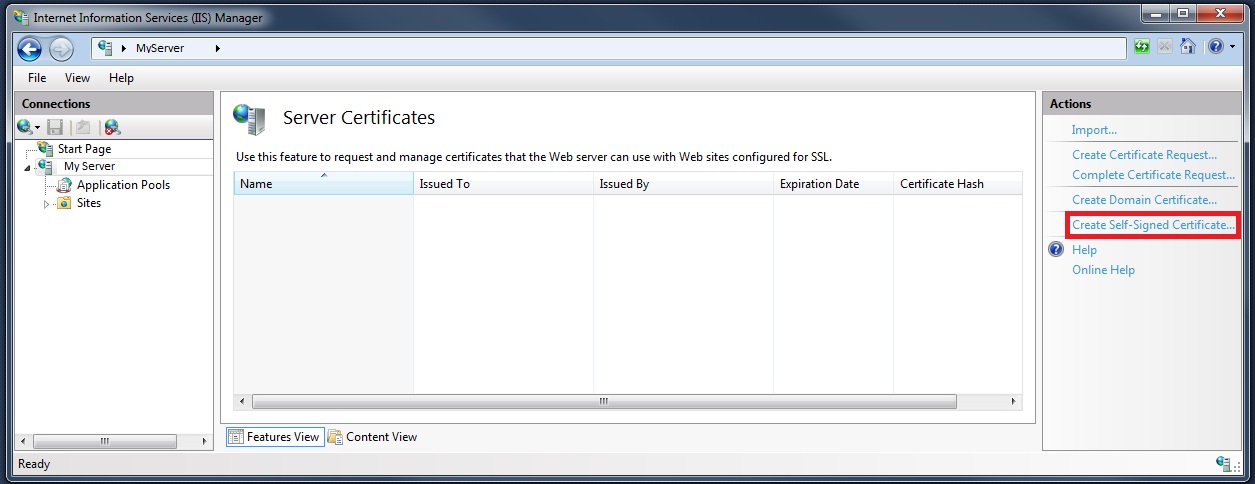
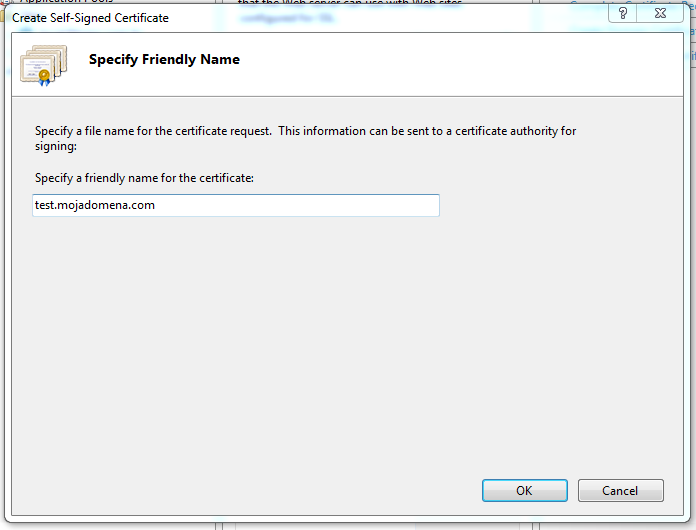
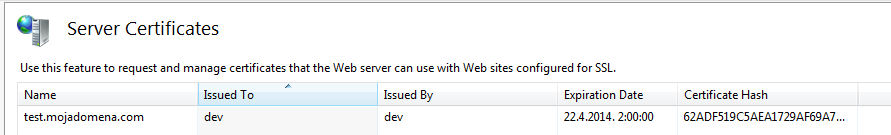
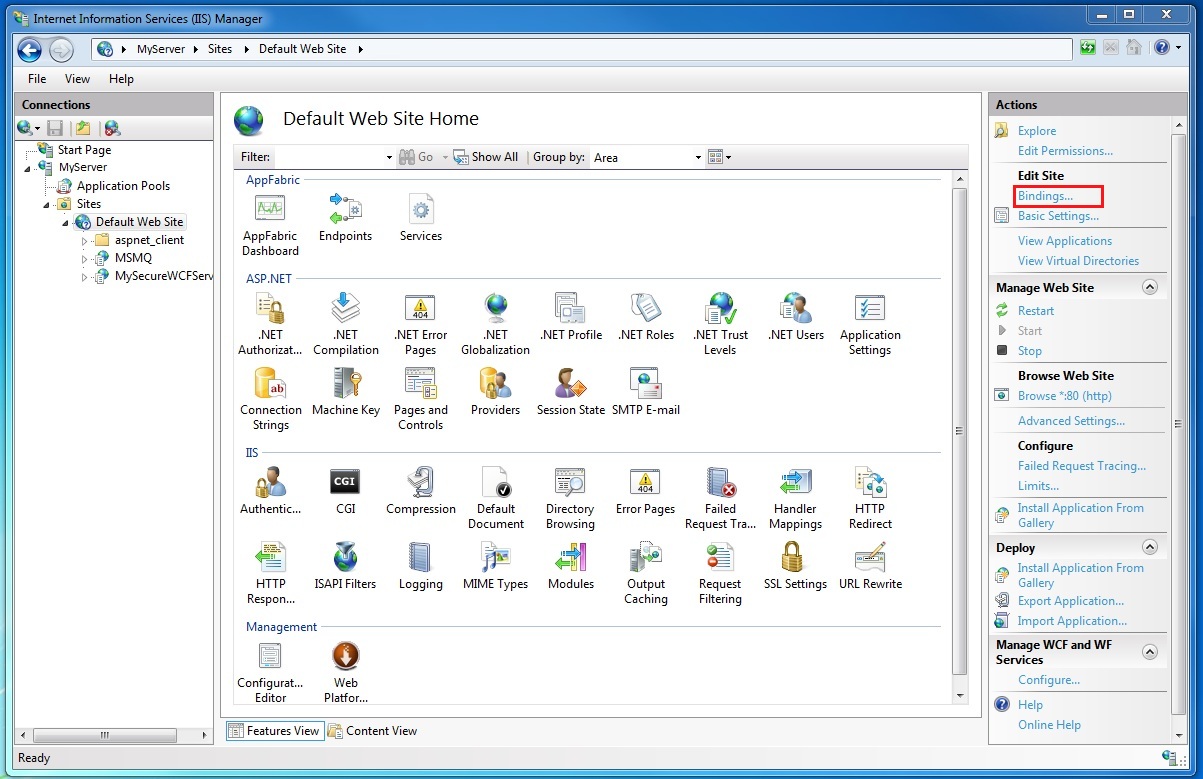
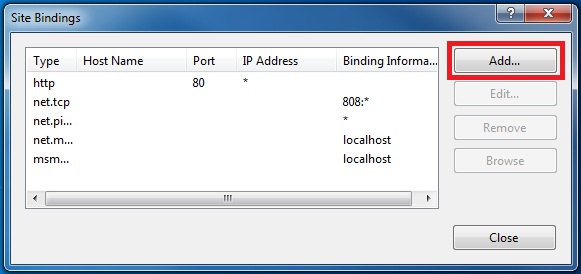
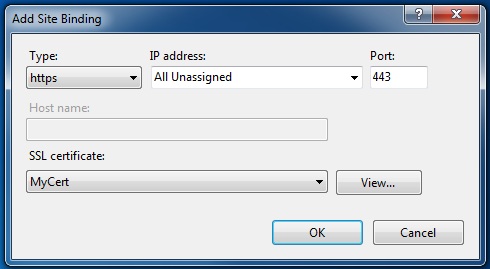
# Servis security: Transport Security - Postavljenje SSL-a na IIS

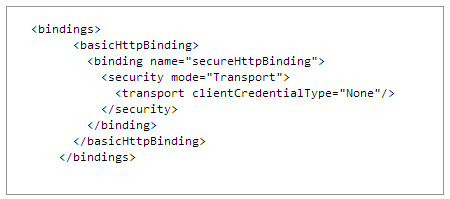
Takozvani SelfSigned certifikat služi za razvoj te može poslužiti u Intranet okolini. Za poslovne aplikacije moguće je izdati vlastiti certifikat unutar tvrtke te podijeliti javni ključ partnerima. Za javni web potrebno je nabaviti valjani global authority certifikat i postaviti ga na IIS. Thawte, GlobalSign ili drugi – nije predmet ovog dokumenta.

Za kreiranje testnog ssl certifikata koristite slijedeće korake:

1. Otvorite „Internet information services manager“
2. Odabrati server sa lijeve strane (najčešće će to biti naziv vašeg računala), te kliknuti na „Server Certificates“



1. Odabrati opciju „Create Self-signed Certificate…“
2. Upisati naziv web stranice (domene) za koju želimo kreirati certifikat   
   
3. Certifikat je kreiran i prikazan unutar „Server certificates“  
   
4. Odabrati „web site“ za koji želimo omogućiti SSL pristup i kliknuti na „Bindings“ unutar „Actions“ prozora sa desne strane  
   
5. Na „Site Bindings“ prozoru kliknuti na „Add“  
   
6. Odabrati „https“ te upravo kreirani ssl certifikat iz liste certifikata  
   

Kada je web site osiguran SSL-om, potrebno je u konfiguraciji servisa (u web.config) podesiti/kreiratibinding koji koristi SSL.

*<security mode=“Transport“* označava da koristimo SSL, dok *clientCredentialType* označava koji tip autorizacije korisnika imamo.

Napomena: Na nivou virtualnog direktorija/aplikacije a za selfsigned SSL imamo u certifikatu zapis IssuedTo =# imemašine# . Mehanizmi zaštite su postavljeni tako da se certifikatu vjeruje jedino ako mu pristupamo sa <http://imemašine>... Gdje ime mašine odgovara IssuedTo zapisu. Dakle <http://localhost>... će rezultirati SSL greškama.

# Servis Security: Transport Security i načini autentifikacije

Unutar WCF konfiguracije preporuka je koristiti neke od slijedećih „modova“ autentikacije:

1. **None/Anonymous** - ne koristi se autentikacija, tj. korisnik je anoniman.
2. **Basic** – Basic autentikacija na IIS-u gdje se korisnički podaci šalju „plain-text“ unutar poziva web servisa.   
   WCF dopušta korištenje Basic autorizacije samo ako se upiti šalju preko SSL-a te je su na taj način korisnički podaci zaštićeni.

Kod basic autorizacije klijent servisa (klijentska aplikacija) šalje korisničke podatke (username/password) direktno iz koda:  
*proxy.ClientCredentials.UserName.UserName = "nekiuser";  
proxy.ClientCredentials.UserName.Password = "123456";*

Tako poslani korisnik mora postojati unutar kao windows user (ili na active directoryju) na računalu gdje je WCF servis.

1. **Certificate** – Opcija u kojoj unutar poziva web servisa moramo poslati i klijentski certifikat.  
   Unutar IIS-a se vrši mapiranje certifikata na windows (ili active directory) korisnički račun te unutar koda možemo provjeriti koji korisnik je pozvao servis pozivanjem:  
   *OperationContext.Current.ServiceSecurityContext.WindowsIdentity*

Korisnička aplikacija certifikat šalje na slijedeći način:  
*proxy.ClientCredentials.ClientCertificate.SetCertificate(…);*

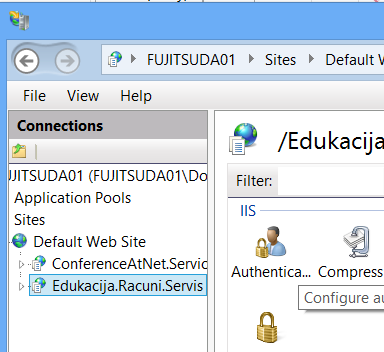
1. **Windows –** koristi se najčešće unutar windows active directory okruženja gdje se klijentska aplikacija „autenticira“ kao korisnik koji je pokrenuo aplikaciju. Prednost windows autorizacije je da nije potrebna nikakva dodatna konfiguracija (ili programiranje servisa) osim podešavanja *clientCredentialType* unutar \*.config datoteke aplikacije, te uključivanje

# Servis security: Transport security + Anonymous autentifikacija – spajanje konzumenta na servis bez slanja credentials-a (anonymous)

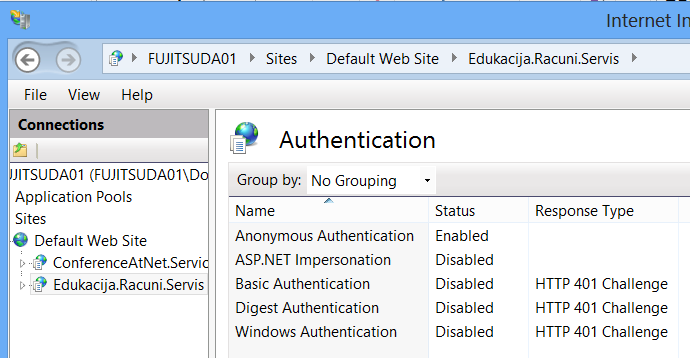
1. Želimo Servis zaštiti na transportnom nivou SSL-om i ne raditi autentifikaciju na nivou poruka
2. Kreirati self-signed certifikat po poglavlju „Servis security – Transport Security - Postavljenje SSL-a na IIS“.

Napomena: Na nivou virtualnog direktorija/aplikacije a za selfsigned SSL imamo u certifikatu zapis IssuedTo =# imemašine# . Mehanizmi zaštite su postavljeni tako da se certifikatu vjeruje jedino ako mu pristupamo sa <http://imemašine>... Gdje ime mašine odgovara IssuedTo zapisu. Dakle <http://localhost>... će rezultirati SSL greškama.

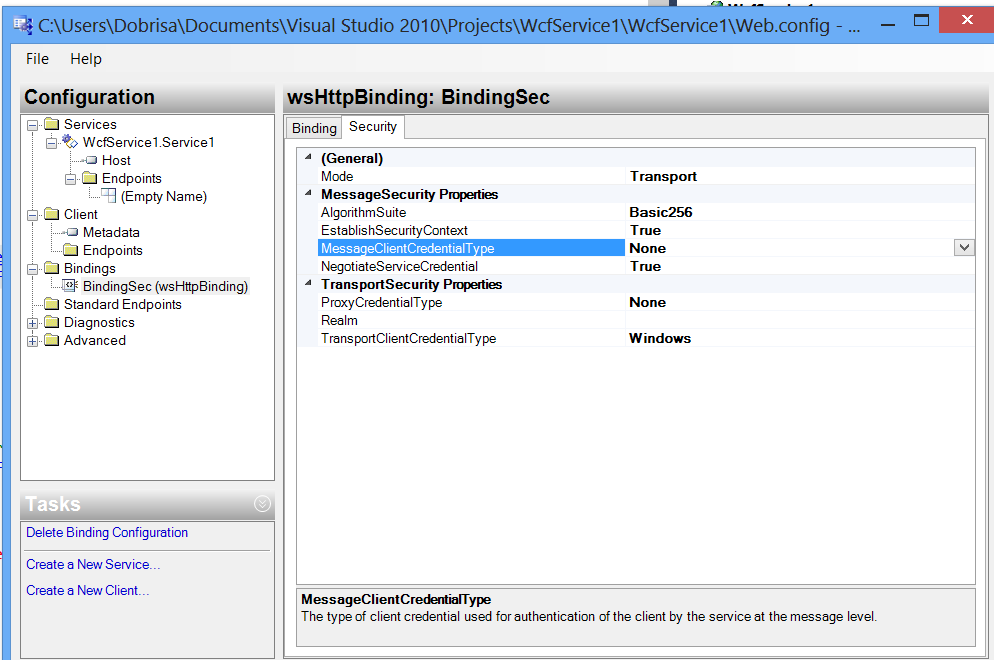
1. Kada je krajnja točka servisa dostupna putem SSL protokola potrebno je napraviti promjene:
   1. Na IIS-u:
      1. na nivou web site-a ili virtualnog foldera gdje se izvršava servis otvoriti Authentication dijalog



* + 1. Anonymous postaviti na Enabled sve ostale na Disabled



* 1. Na servisu:
     1. Omogućiti pristup do metapodataka putem SSL-a odnosno https-a. Postaviti serviceMetadata atribut httpsGetEnabled na true
        1. <serviceMetadata **httpsGetEnabled**="true""/>
     2. Binding definiran u poglavlju o default podešavanju podesimo na Transport security + MessageClientCredentialsType na None
        1. Kroz config editor:



Ili kroz XML:

<bindings>

<wsHttpBinding>

<binding name="BindingSec">

<security mode="Transport">

<message clientCredentialType="None" />

</security>

</binding>

</wsHttpBinding>

</bindings>

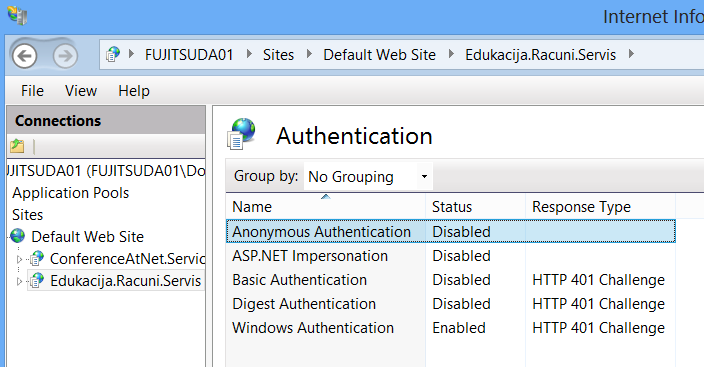
* 1. Na konzumentu:
     1. Update servisne reference
     2. Za sigurni rezultat najbolje je u potpunosti obrisati ili zakomentirati system.ServiceModel područje u app.config datoteci te tada napraviti Configure Service Reference i promijenimo http u https te ponovno rekreirati proxy klasu
     3. Ako se nije promijenio app.config – jer isti binding mora sjesti i u app.config – onda možemo delete servisne reference pa dodati ponovno
     4. U app.config-u se trebaju vidjeti odgovarajuće promjene

# Servis security: Transport security i Windows Credentials autentifikacija

1. Želimo Servis zaštiti na transportnom nivou SSL-om i autentifikacija sa Windows Credentials kontekstom
2. Kreirati self-signed certifikat po poglavlju „Servis security – Transport Security - Postavljenje SSL-a na IIS“.

Napomena: Na nivou virtualnog direktorija/aplikacije a za selfsigned SSL imamo u certifikatu zapis IssuedTo =# imemašine# . Mehanizmi zaštite su postavljeni tako da se certifikatu vjeruje jedino ako mu pristupamo sa <http://imemašine>... Gdje ime mašine odgovara IssuedTo zapisu. Dakle <http://localhost>... će rezultirati SSL greškama.

1. Na IIS-u:
   1. na nivou web site-a ili virtualnog foldera gdje se izvršava servis otvoriti Authentication dijalog
      1. Windows – Enable
      2. Svi drugi – Disable



* 1. Test – zahtjev za <https://krajnjatockaservisa> u browseru će pokazati Windows Autentifikacijski dijalog

1. Na servisu:
   1. Na servisnoj strani postavljemo clientCredentialType na Windows – bilo je None

<security mode="Transport">

<transport clientCredentialType="Windows" />

</security>

1. Na konzumentu u app.config radimo istu promjenu

<security mode="Transport">

<transport clientCredentialType="Windows" />

</security>

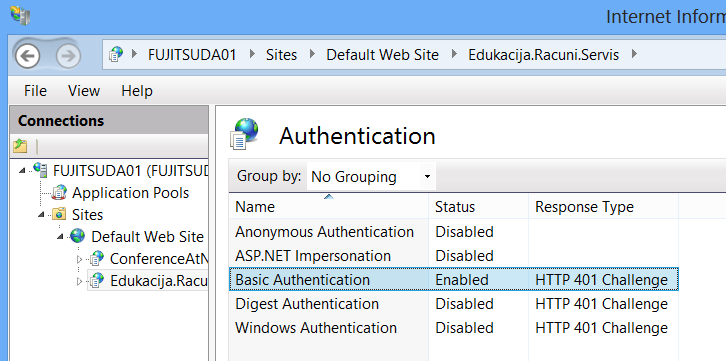
Ako tražimo user context možemo vidjeti da je učitan Windows user aplikacije koja zove servis.

# Servis security: Transport security + Basic security - šaljemo podatke o useru – predstavljamo se kao određeni windows user

1. Želimo Servis zaštiti na transportnom nivou SSL-om i autentifikacija sa Windows korisnikom ali uz opciju da kroz kod možemo odrediti koji korisnik zove servis
2. Kreirati self-signed certifikat po poglavlju „Servis security – Transport Security - Postavljenje SSL-a na IIS“.

Napomena: Na nivou virtualnog direktorija/aplikacije a za selfsigned SSL imamo u certifikatu zapis IssuedTo =# imemašine# . Mehanizmi zaštite su postavljeni tako da se certifikatu vjeruje jedino ako mu pristupamo sa <http://imemašine>... Gdje ime mašine odgovara IssuedTo zapisu. Dakle <http://localhost>... će rezultirati SSL greškama.

1. Na IIS-u:
   1. Na IIS-u pod virtualnim folderom otvaramo Authentication
      1. Basic – Enable
      2. Svi drugi – Disable



* 1. otvoriti https://krajnjatockaservisa.svc i provjeriti dostupnost – nije dostupno jer se očekuje username/password

1. Na servisnoj strani postavljemo clientCredentialType na Basic

<security mode="Transport">

<transport clientCredentialType="Basic" />

</security>

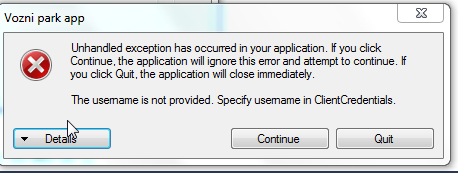
1. Na konzumentu u app.config radimo istu promjenu

<security mode="Transport">

<transport clientCredentialType="Basic" />

</security>

1. Isprobamo aplikaciju – servis očekuje da kroz kod odredimo Windows usera kojim se predstavljamo



1. U kodu upisujemo username i password

proxy.ClientCredentials.UserName.UserName = "nekiuser";

proxy.ClientCredentials.UserName.Password = "123456";

1. Isprobati sa userom koji postoji na mašini
2. Ako ne postoji kreirajte Usera testservis sa passwordom 123456. Start, Desni klik na Computer, pa Manage, pa Users and Groups

# Servis security: TransportWithMessage Security - Transport security + Message Username credentials - sami implementiramo username/password provjeru

1. Na IIS-u moramo dozvoliti anonymous jer sada mi preuzimamo provjeru
   1. Na IIS-u pod virtualnim folderom otvaramo Authentication
      1. Anonymous – Enable
      2. svi drugi - disable
2. U Library projektu dodajemo reference na System.Identity
3. Nasljeđujemo klasu UserNamePasswordValidator – using System.Identity.Selectors

i radimo implementaciju:

public class #NazivServisa#UserNameValidator : UserNamePasswordValidator

{

// metoda za validaciju

//kreirati logiklu prema bazi

public override void Validate(string userName, string password)

{

if (null == userName || null == password)

{

throw new FaultException("Niste poslali podatke");

}

//logika za test sa zadnim username/password parom

if (!(userName == "user1" && password == "123456"))

{

// vrati grešku konzumnetu

throw new FaultException("Nije dobar username ili password");

}

}

}

1. Konfiguracija servisa – prebacujemo na Message – Username credentials i zato što imamo SSL ostavljamo i transport – to je jedna security oznaka

<security mode="TransportWithMessageCredential">

<transport clientCredentialType="None" />

<message clientCredentialType="UserName"/>

</security>

1. Podešavamo Behaviour na servisu, podešavamo servisne credentials-e na userNameAuthentication, Custom i podešavamo svoju klasu, namespace. Moramo definirati i koji se certifikat koristi na server – po tutorialima da – ali ovdje ne moramo nužno.

<behaviour>...

....

<serviceCredentials>

<userNameAuthentication userNamePasswordValidationMode="Custom"

customUserNamePasswordValidatorType="#NamespaceServisa#.#NazivServisa#UserNameValidator, #NazivServisa#Library"/>

<serviceCertificate storeLocation="LocalMachine" storeName="Root" x509FindType="FindBySerialNumber" findValue="576f5b93cf4d999f4713eaaf5ea4e4ca" />

</serviceCredentials>

....

</behaviour>

1. Definiramo servisni certifikat – self signed certifikat koji smo kreirali za site – traži se po serijskom broju jer nam se svi selfsigned zovu isto

<serviceCertificate storeLocation="LocalMachine" storeName="Root" x509FindType="FindBySerialNumber" findValue="576f5b93cf4d999f4713eaaf5ea4e4ca" />

</serviceCredentials>

1. Na konzumentu trebamo napraviti update servisne reference – najbolje je u app.configu sve zakomentirati tako da dovučemo novu definiciju.
2. U kodu šaljemo podatke koje provjeravamo u custom metodi

proxy.ClientCredentials.UserName.UserName = "user1"; proxy.ClientCredentials.UserName.Password = "123456";

1. UserName credentials type može i kao Message security samo ako se promjeni binding konfiguracija.

# Servis security – razmjena ključeva

Pogledati kao referencu:

<http://robbincremers.me/2011/12/27/wcf-transport-security-and-client-certificate-authentication-with-self-signed-certificates/>

<http://robbincremers.me/2011/12/29/wcf-message-security-and-client-certificate-authentication-with-self-signed-certificates/>

1. Nastavno na Transport Security upute
2. Podešavanje servisa
   1. Binding za message + certificate security definiramo:

<binding name="MessageCertificateBinding">

<security mode="Message">

<message clientCredentialType="Certificate" negotiateServiceCredential="true" />

</security>

</binding>

* 1. U definiciji endpointa mijenjamo pokazivač na ovaj binding

<endpoint address="https://krajnjatockaservisa.svc"

binding="wsHttpBinding" bindingConfiguration="**MessageCertificateBinding**"

contract="#NamespaceServisa#.I#NazivServisa#" />

* 1. Definiramo u Behavoiur elementu predstavljenje certifikatom i očekivanje client certifikata

<behaviour>...

....

<serviceCredentials>

<serviceCertificate storeLocation="LocalMachine" storeName="Root" x509FindType="FindBySerialNumber" findValue="576f5b93cf4d999f4713eaaf5ea4e4ca" />

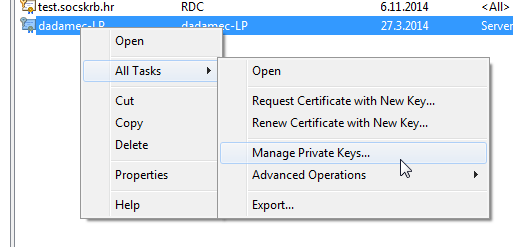
<clientCertificate>

<authentication certificateValidationMode="PeerTrust" trustedStoreLocation="LocalMachine"/>

</clientCertificate>

</serviceCredentials>

* 1. Otvoriti SVC u browser – provjera da li ima/nema grešaka u configu
  2. Tipična greška: Keyset doesnot exists – zbog toga što process nema prava na čitanje privatnog ključa
     1. U Certifikatima – desni klik pa Manage private Keys – dodati NetworkService – prava na čitanje
     2. Može zahtjevati restart IIS-a jer smog a već bili tražili



1. Podešavanje konzumenta
   1. Zakomentirati system.ServiceModel element
   2. Update servisne reference – da dobijemo novi config
   3. Rezultat je :

<system.serviceModel>

<bindings>

<wsHttpBinding>

<binding name="WSHttpBinding\_#ImeServisa#">

<security>

<message clientCredentialType="Certificate" />

</security>

</binding>

</wsHttpBinding>

</bindings>

<client>

<endpoint address="http://krajnjatockaservisa.svc"

binding="wsHttpBinding" bindingConfiguration="WSHttpBinding\_#ImeServisa#"

contract="#ImeServisa#" name="WSHttpBinding\_#ImeServisa#">

<identity>

<certificate encodedValue="" />

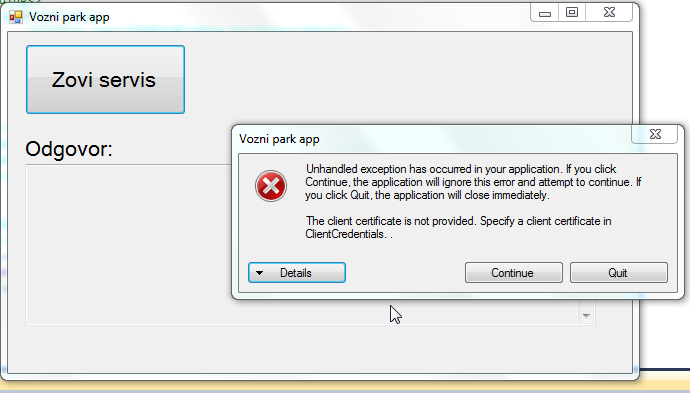
</identity>

</endpoint>

</client>

</system.serviceModel>

* 1. Identity => Public key serverskog certifikata
  2. Ako sada probamo aplikaciju dobijemo poruku da nismo posatvili klijentski certifikat



* 1. Kreiramo klijentski certifikat
     1. koristimo IIS za self signed – najbrže je
     2. ili makecert iz cmd prompta

makecert –pe –n “CN=vozniparkclient” –sr localmachine –ss my

makecert -pe -n "CN=VozniParkKlijent" -sr localmachine -ss my

* 1. Access Denied – IIS – restart MMC-a i IIS-a
  2. Pogledamo Serial Number

<behaviors>

<endpointBehaviors>

<behavior name="ClientCertificateVozniPark">

<clientCredentials>

<clientCertificate storeLocation="LocalMachine" storeName="My" x509FindType="FindBySerialNumber" findValue="‎‎1c61dbd13c0f0e9e416ac7d39735ae8b"/>

</clientCredentials>

</behavior>

</endpointBehaviors>

</behaviors>

* 1. na endpoint pridjelimo novo definirani behavior sa behaviourConfiguration atributom
  2. test
  3. ako je formatting exception – onda je copy paste iz certifikata donio lose znakove
  4. Klijentski certifikat mora biti na serveru TrustedPeople (jer imamo PeerTrust) – inače je greška da caller nije prošao provjeru
  5. Testiramo

# Dodatno – Pobrojani security scenariji sa kombinacijama zaštite

<http://msdn.microsoft.com/en-us/library/ms730301.aspx>

# Dodatno: Demo skripta –Web HTTP programiranje – REST na WCF

Napomena: nisu detaljne upute već mali demo skript za izvođenje uz predavača

1. Radimo na solution-u WCF\_SyndRESTJSON/before/TehnickiPregled
2. Omogućiti uz SOAP i čistu HTTP komunikaciju
3. Probati aplikaciju, servis postaviti na IIS ako nije, osvježiti servisne reference, isprobati
4. Operaciju GetPregledi želimo postaviti na način da može poslati podatke kao XML na HTTP GET zahtjev – atributom WebGet

[OperationContract]

[WebGet]

List<Model.Pregled> GetPregledi();

1. U konfiguraciji trebamo postaviti dva endpointa – za SOAP i za WebGet
   1. otvaramo scvconfigeditor
   2. u njemu otvaramo web config
   3. dodajemo servis, biramo DAL dll i interface
   4. dva endpointa
      1. jedan httpBinding
      2. jedan webHttpBinding
   5. dodajemo jedan endpointconfiguration

<endpointBehaviors>

<behavior name="WebEB">

<webHttp/>

</behavior>

</endpointBehaviors>

* 1. Pridružimo ga na web binding sa behaviourConfiguration atributom

behaviorConfiguration="WebEB"

* 1. Podesimo endpoints ovisno o multipleSiteBindings – ako je true onda relativnu adresu u binding inače apsolutnu
  2. za soap ostavimo kako je a za web dodamo “web” prefix na adresu

<service name="DAL.PreglediAkcije">

<endpoint address="http://localhost/TehnickiPregledServis2/akcije.svc" binding="basicHttpBinding" bindingConfiguration=""

name="BasicBinding" contract="DAL.IPreglediAkcije" />

<endpoint address="http://localhost/TehnickiPregledServis2/akcije.svc/web" binding="webHttpBinding" contract="DAL.IPreglediAkcije" bindingConfiguration=""

name="WebEndpoint" behaviorConfiguration="WebEB" />

</service>

1. Ako je multiplesitebindings true onda relativne adrese:

<service name="DAL.PreglediAkcije">

<endpoint address="" binding="basicHttpBinding" bindingConfiguration=""

name="BasicBinding" contract="DAL.IPreglediAkcije" />

<endpoint address="web" binding="webHttpBinding" contract="DAL.IPreglediAkcije" bindingConfiguration=""

name="WebEndpoint" behaviorConfiguration="WebEB" />

</service>

1. Pristupamo metodi sa HTTP GET zahtjevom kroz browser

<http://localhost/TehnickiPregledServis2/akcije.svc/web/GetPregledi>

i gledamo rezultat XML

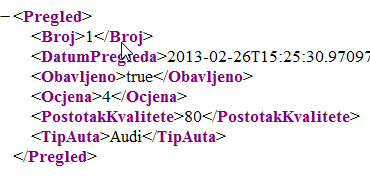
1. Probamo neku koju nismo označili sa WebGet
   1. method not allowed ili endpoint not found
2. Metoda GetPrgledByBroj prima ulazni parameter i nju moramo dodatno označiti ako je želimo koristiti kroz WebGet
   1. Stavljemo WebGet
   2. I dodatno definirano UriTemplate

[WebGet(UriTemplate="GetPregledByBroj?broj={broj}")]

* 1. koristimo query string niz za primiti parameter
  2. šaljemo ga u URL-u iza znaka ? kao name/value par
  3. probamo

http://localhost/TehnickiPregledServis2/akcije.svc/web/GetPregledByBroj?broj=1

* 1. vidimo rezultat



1. Ovo su osnove HTTP REST akcija – biti će razrađene dalje tijekom edukacije dodatno kroz sindikaciju i kroz OData podršku
2. Dodatno – JSON format - konfiguriramo da se ne vraća XML već JSON
   1. pogledati kroz Fiddler

Eventualni helper sa problemima kod mapiranja adresa – na implementaciji:

//[ServiceBehavior(AddressFilterMode = AddressFilterMode.Any)]

# Dodatno: Demo skripta – WCF i JSON

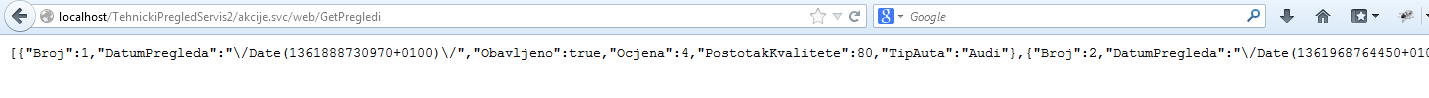
Napomena: nisu detaljne upute već mali demo skript za izvođenje uz predavača

1. Nastavno na demo prije
2. Publiciranje podataka putem JSON formata
   1. Podešavamo atribut na operaciji GetPregledi – mijenjamo ResponseFormat

[OperationContract]

[WebGet(ResponseFormat=WebMessageFormat.Json)]

List<Model.Pregled> GetPregledi();

* 1. Probamo u browseru <http://localhost/TehnickiPregledServis2/akcije.svc/web/GetPregledi>
  2. Rezultat je JSON object prikazan kao na primjer: 

1. Korištenje servisa sa response-om formatiranim kao JSON
   1. Dodajemo HTML stranicu u projket gdje se nalazi servis – HTMLtest.htm
   2. Sa adrese https://developers.google.com/speed/libraries/devguide#jquery dodati script tag i uključiti jQuery u stranicu – objasniti malo CDN princip

<script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>

* 1. Kreirati HTML sučelje, button i div za rezultat

<input type="button" value="Zovi servis" onclick="GetPregledi();" />

<div id="divRezultat"></div>

* 1. Kreirati JavaScript funkciju koja koristi jQuery AJAX poziv na naš servis, prima podatke, formatira ih i pokaže u divu divRezultata.

<script language="javascript" type="text/javascript">

function GetPregledi() {

var res = "";

$.ajax({

type: "GET",

url: "http://localhost/TehnickiPregledServis2/akcije.svc/web/GetPregledi",

data: '',

processData: false,

contentType: "application/json; charset=utf-8",

dataType: "json",

success: function (data) {

//alert(data[0].Broj);

//alert(data.length);

for (var i = 0; i < data.length; i++) {

//alert(res);

res += data[i].Broj + " " + data[i].TipAuta + " " + data[i].Ocjena + "<br/>";

}

}

});

var divR = document.getElementById("divRezultat");

divR.innerHTML = res;

}

</script>

* 1. Pogledati u Fiddleru ili Firebug-u, json ajax detalji

# Dodatno: Demo skripta - WCF Sindikacija

Napomena: nisu detaljne upute već mali demo skript za izvođenje uz predavača

1. Definiramo servisni ugovor za sindikaciju – u Interface-u
   1. using System.ServiceModel.Syndication;
   2. Definiramo operaciju GetPreglediAsRSS sa WebGet atributom . povratni tip je SyndicationFeedFormater – tip u koji je zamotan RSS i ATOM način formatiranja tako da možemo odlučiti kasnije

[WebGet]

SyndicationFeedFormatter GetPreglediAsRSS();

* 1. U ugovoru moramo dodati dodatni atribut kojime servisu kažemo da želimo uključiti tip u ugovor

[ServiceKnownType(typeof(Atom10FeedFormatter))]

[ServiceKnownType(typeof(Rss20FeedFormatter))]

1. Implementiramo metodu . kreiramo feed i iteme i na kraju odlučujemo o format koji se vraća ATOM ili RSS - Rss20FeedFormatter ili Atom10FeedFormatter

public SyndicationFeedFormatter GetPreglediAsRSS()

{

string format = "atom";

SyndicationFeed feed = new SyndicationFeed("Tehnički Pregledi iz Aplikacije A", "Feed sa računima za partnere", new Uri("http://localhost/TehnickiPregledServis2/akcije.svc/web/GetBlog"));

feed.Authors.Add(new SyndicationPerson("pregledi@tehnicki.hr"));

feed.Categories.Add(new SyndicationCategory("TehničkiPregledi"));

feed.Description = new TextSyndicationContent("Svi dostupni pregledi");

List<SyndicationItem> items = new List<SyndicationItem>();

Model.Pregled pregled = new Model.Pregled();

//rekurzija po direktoriju gdje su datoteke

DirectoryInfo dirinfo = new DirectoryInfo(@"C:\Users\Public\");

foreach (var item in dirinfo.GetFiles())

{

//samo naše datoteke sa serijaliziranim pregledima

if (item.Name.StartsWith("pregled"))

{

pregled = new Model.Pregled();

//TODO: try-catch

System.Xml.Serialization.XmlSerializer reader = new System.Xml.Serialization.XmlSerializer(pregled.GetType()); //"reader" je naš naziv za XML serializer

System.IO.StreamReader file = new StreamReader(item.FullName);

//konkretno pročitamo iz datoteke i učitamo u objekt

pregled = (Model.Pregled)reader.Deserialize(file);

SyndicationItem syndicationitem = new SyndicationItem(

pregled.TipAuta.ToString(),

pregled.Ocjena.ToString(),

new Uri("http://localhost/Pregledi/" + pregled.Broj.ToString()),

pregled.Broj.ToString(),

DateTime.Now);

items.Add(syndicationitem);

file.Close();

}

}

feed.Items = items;

if (format == "rss")

return new Rss20FeedFormatter(feed);

else if (format == "atom")

return new Atom10FeedFormatter(feed);

else return null;

}

1. Testiramo u browser

http://localhost/TehnickiPregledServis2/akcije.svc/web/GetPreglediAsRSS

1. View – Source
2. Pa zamijenimo format ATOM i RSS pa pogledamo opet i ponovno pogledamo source
3. Testiramo sa vlastitom aplikacijom RSS Reader napravljenom kroz zadaće