Webbtjänster med API er

Mål med lektionen!

- Repetera kunskaperna

Vad lektionen omfattar

- Repetition av kursen.

Webbtjänster?

En webbtjänst är en tjänst som erbjuds av en elektronisk anordning till en annan elektronisk anordning, som kommunicerar med varandra via World Wide Web.

I en webbtjänst, Web-teknik såsom HTTP, som ursprungligen var avsedd för människa-till-maskin kommunikation, används för maskin-till-maskin kommunikation, närmare bestämt för att överföra maskinläsbara filformat som XML och JSON.

WCF

Windows Communication Foundation (WCF)

Är ett ramverk för att bygga serviceorienterade applikationer.

Med hjälp av WCF, kan du skicka data som asynkrona meddelanden från en service slutpunkt (endpoint) till en annan. En service slutpunkt kan vara en del av en ständigt tillgänglig tjänst hostad av IIS, eller det kan vara en tjänst i en applikation.

En slutpunkt kan vara en klient hos en service som begär data från en service slutpunkt.

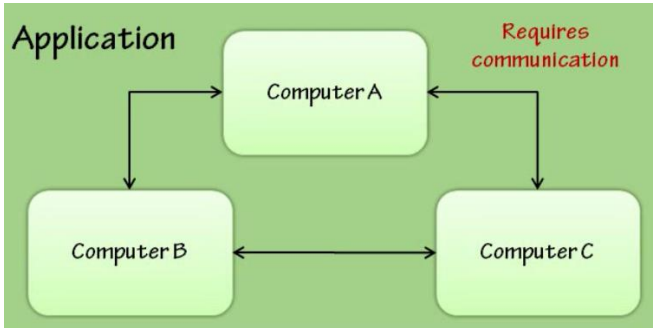
Meddelandena kan vara så enkelt som ett enda tecken eller ord skickas som XML, eller så komplicerat som en ström av binära data

WCF

När man pratar om WCF och vad WCF är så har Microsoft ändrat om benämningen till ”connected system” (tidigare var det distributed system).

Men vad är connected system:

> *Ett sk ”connected system” är ett system som är distribuerat över flera dator noder som har olika delar av ett visst program, men jobbar tillsammans och kommunicerar emellan varandra genom att skicka meddelanden.*



Service Orientation

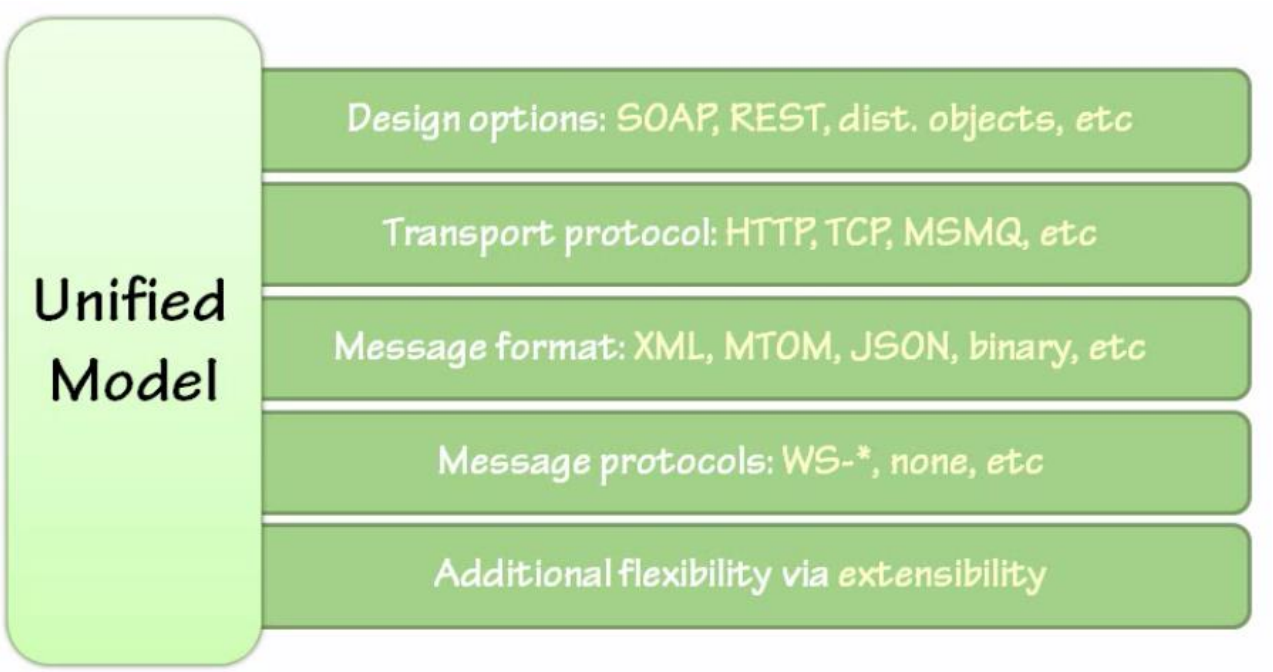
Microsoft har använt sig av både SOAP baserade och REST baserade tjänster tidigare även om de är fundamentalt olika ifrån varandra så anses de vara ”service oriented” i grund och botten.

Men vad är **Service Oriented**?

Det hela är en slags generell design paradigm (design mönster) som fokuserar på **SoC**.

Kommunikations ramverk

Med de tidigare erfarenheterna i minnet så insåg Microsoft snabbt att en av deras viktigaste design mål skulle vara att bygga en enhetlig programmeringsmodell för all kommunikations logik.



WCF Services

Vad är en WCF service?

En WCF Service är ett program som visar/ställer ut en samling av endpoints. Varje endpoint är en portal för att kommunicera med världen (ifrån servicens ögon sett).

All kommunikation sker genom en/flera endpoints, en endpoint består av tre komponenter:

- **Address** – vart man skall skicka meddelanden.
- **Binding** – hur man skall skicka meddelanden.
- **Contract** – vad meddelanden måste innehålla.

endpoints

Din implementering av en webbtjänst definierar själva affärslogiken för det är så vi skriver koden, och våra endpoints definierar de olika kommunikations sätten som våran tjänst stödjer, det är så vi kopplar ihop allt.

Tjänster kan exponera multipla endpoints för konsumenter, vi kan ha hur många vi vill men minst en.

Endpoint: address

Endpoints är vanligtvis konfigurerad med en unik adress.

Endpoints kan ”dela” en adress om de ”delar” samma binding instans, men de måste då vara konfigurerade med olika kontrakt.

Adresser måste alltid vara i forma av en URI vilken är beroende på vilken typ av transport som vi använder för vår endpoint, och transporten är bestämd av vår bindning.

Endpoint: address

Transport	Scheme	Example
HTTP	http://	http://pluralsight.com/myservice
TCP	net.tcp://	net.tcp://pluralsight.com:8081/myservice
Pipes	net.pipe://	net.pipe://pluralsight.com/myservice
MSMQ	net.msmq://	net.msmq://pluralsight.com/private/myservice

De vanligaste transport typerna och deras protokoll schema tillsammans med exempel på hur uri'n skulle se ut.

Bindings?

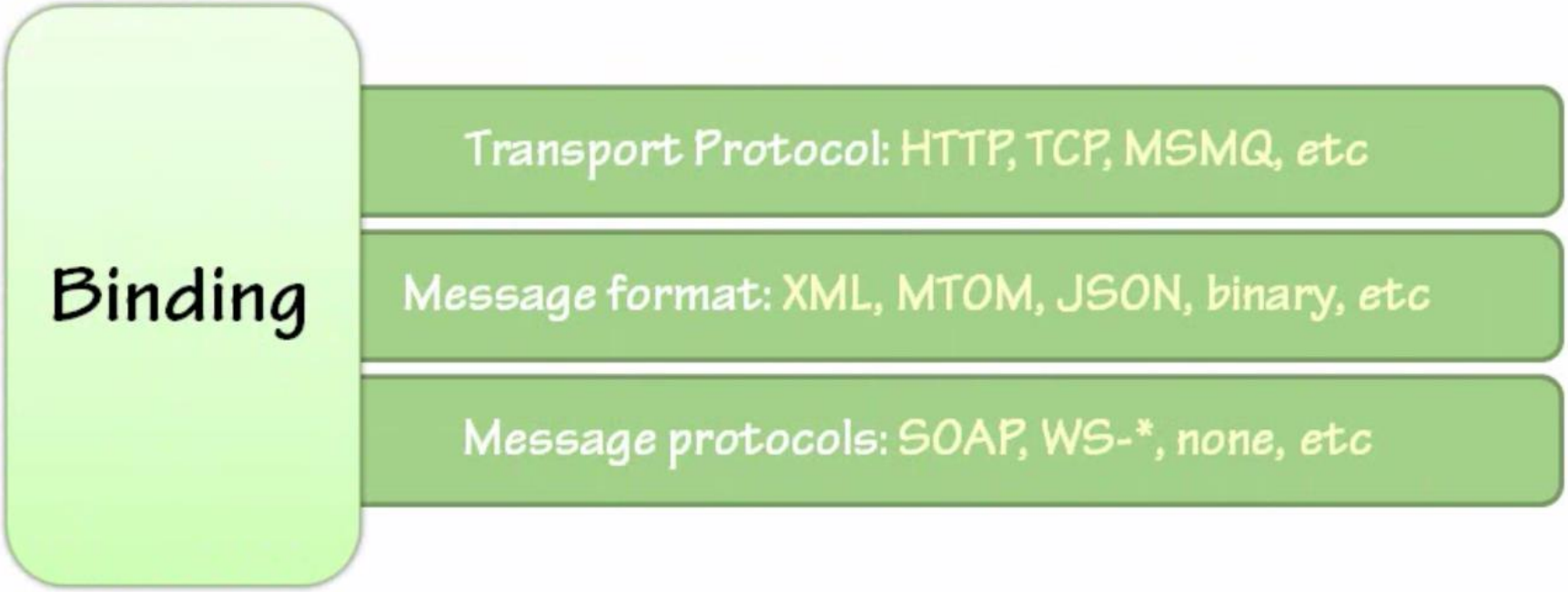
En bindning är egentligen ett ritning för hur WCF kommer att bygga den underliggande kommunikations kanalen.

WCF kommer med en hel hög med inbyggda bindningar för några av de vanligaste kommunikations scenariona.

Binding Name	Communication Scenario
WebHttpBinding	Interoperable RESTful communication via HTTP
BasicHttpBinding	Interoperable SOAP communication via HTTP, offering only the "basic" protocols conforming to WS-I Basic Profile
WSHttpBinding	Interoperable SOAP communication via HTTP, offering the full range of SOAP + WS-* protocols
NetTcpBinding	Cross-machine WCF communication via TCP
NetPeerTcpBinding	Cross-machine WCF communication via P2P
NetNamedPipesBinding	Same-machine WCF communication via IPC
NetMsmqBinding	Disconnected/asynchronous WCF communication via MSMQ

Endpoint: binding

En binding är som ett recept som specificerar tre olika kommunikations detaljer.



Contracts

I WCF så är alla tjänster exponerade som kontrakt.

Kontrakt är ett plattforms neutralt sätt för att beskriva vad tjänsten gör. I huvudsak är det fyra olika typer av kontrakt inom WCF som är tillgängliga/används.

- Service Contract
- Data Contract
- Message Contract
- Fault Contract

Service Contract

Service kontraktet beskriver operationen/metoden som tjänsten erbjuder. En tjänst kan ha mer än ett kontrakt men minst ett kontrakt. Detta kontrakt definieras som en slags opt-in modell med attribut.

Vi använder [ServiceContract] och [OperationContract] attributen och vi använder en interface definition för att modellera dessa saker eftersom det passar bra in i det vi vill uträtta.

```
[ServiceContract()]
    public interface ISimpleCalculator
    {
        [OperationContract()]
        int Add(int num1, int num2);
    }
```

En länk till properties för den intresserade.

Data Contract

För WCF så finns det ett flertal serilizers men vi ska titta/använda default varianten som heter DataContractSerializer.

Den kräver att vi explicit kommenterar/noterar vilka typer exakt som vi vill inkludera i meddelandet.

Det finns två attribut som ni kommer att använda en hel del:

- DataContract
- DataMember

```
[DataContract]
public class Eval
{
    [DataMember]
    public string Submitter;
    [DataMember]
    public DateTime TimeSent;
    [DataMember]
    public string Comments;
}
```

Message Contract

WCF använder SOAP meddelanden för att kommunicera, mestadels så kan vi koncentrera oss på DataContract och serialisering av datan osv eftersom WCF normalt tar hand om meddelandet. Men i vissa kritiska lägen så kan det vara bra att kunna ha kontroll över meddelande formatet och i dessa fall har vi MessageContract.

```
[MessageContract]
public class Eval
{
    [MessageHeader]
    public string submitter;
    [MessageBodyMember]
    public DateTime TimeSent;
    [MessageBodyMember]
    public string comments;
}

[ServiceContract]
public interface IEvalService
{
    [OperationContract]
    void SubmitEval(Eval eval);
    [OperationContract]
    List<Eval> GetEval();
}
```

Fault Contract

I vanliga applikationer hanterar vi fel genom try-catch block, i WCF så kan vi skicka denna informationen till klienten om vi använder oss av FaultContract.

```
[DataContract()]
public class CustomException
{
    [DataMember()]
    public string Title;
    [DataMember()]
    public string ExceptionMessage;
    [DataMember()]
    public string InnerException;
    [DataMember()]
    public string StackTrace;
}
```

```
[ServiceContract()]
public interface ISimpleCalculator
{
    [OperationContract()]
    [FaultContract(typeof(CustomException))]
    int Add(int num1, int num2);
}
```

```
public int Add(int num1, int num2)
{
    //Do something
    CustomException ex = new CustomException();
    ex.Title = "Error Funtion:Add()";
    ex.ExceptionMessage = "Error occur while doing add function.";
    ex.InnerException = "Inner exception message from serice";
    ex.StackTrace = "Stack Trace message from service.";
    throw new FaultException(ex,"Reason: Testing the Fault contract") ;
```

```
  Console.WriteLine("Sum of two numbers... 5+5 =" + proxy.Add(5, 5
  Console.ReadLine();
}
catch (FaultException<MyCalculatorService.CustomException> ex)
 {
   //Process the Exception
 }
```

Implementera tjänsten

Nu har vi våra kontrakt på plats, och vi kan då börja bygga våran tjänst.

Det vi skall tänka på är att vi bör ärva ifrån i alla fall minst ett ServiceContract (för ärver vi inget så vad är det för tjänst vi erbjuder då?). Implementera alla de metoder som du behöver för att bygga Business logiken.

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]
public class EvalService : IEvalService
{
    List<Eval> evals = new List<Eval>();
    public void SubmitEval(Eval eval)
    {
        evals.Add(eval);
    }

    public List<Eval> GetEval()
    {
        return evals;
    }
}
```

Hosting WCF Services

self-hosting
IIS hosting
WAS hosting

self-hosting

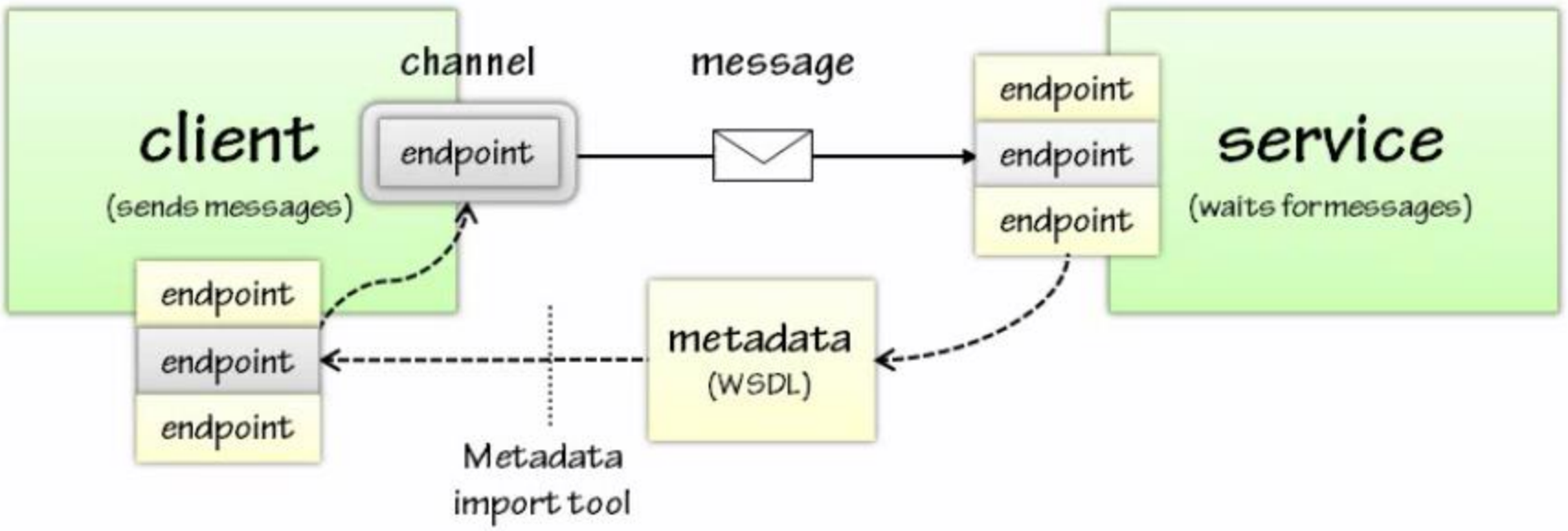
```
class Program {
    static void Main(string[] args) {
        ServiceHost host =
            new ServiceHost(typeof(InvoiceService));
        ... // configure the host before opening

        try {
            host.Open();
            Console.ReadLine();
            host.Close();
        }
        catch (Exception e) {
            Console.WriteLine(e);
            host.Abort();
        }
    }
}
```

Configuring Services

Det finns två sätt för oss att konfigurera vår tjänst när/om vi behöver det.

Antingen explicit i program koden eller deklarativt i din konfigurations fil. Saker som vi kan konfigurera är endpoints och behaviors, sedan är det så att kan du göra det i kod så kan du göra det i konfigurations filen i alla fall när det gäller konfigurera tjänsten.

Hur klient och service kommunikation fungerar



Dynamiska endpoints

Genom att använda MEX så tillåts vi att hämta metadata programmatiskt. WCF tillhandahåller bla två klasser som gör detta .

En som heter MetadataResolver och en som heter MetadataExchangeClient. Båda gör i huvudsak samma sak, de går upp till tjänsten vid körning med hjälp av MEX protokollet och ber om metadatan. Tjänsten skickar då metadatan i svaret som sedan dessa två klasserna använder informationen i någon form av objektmodell.

Dynamiska endpoints

Istället kan klienten alltid dynamiskt hämta metadatan vid körning.

Det fungerar ungefär som så att när klienten startar så ber den om metadata, laddar ner detta, väljer en endpoint och sedan kommunicerar med tjänsten.

För om tjänsten skulle behöva göra ändringar över tiden, så kommer klienterna/klienten inte gå sönder eftersom nästa gång den startar så hämtar den ny information.

Vad vi får av WSDL

- Den ger en genererad kod fil som innehåller kontrakt definitionerna.
- Den ger en genererad konfigurations fil som vi kan använda som app.config fil, denna innehåller de endpoints som vi behöver.

Programmera kanaler

1. Skapa och konfigurera en ChannelFactory
2. Skapa en faktisk instans av en kanal
3. Gör metodanrop genom kanalen
4. Stäng eller anropa Abort

Undvika använda Channelfactory

Alla kanaler implementerar IClientChannel där vi har bla Close och Abort, men vi måste casta objektet till IClientChannel för att kunna använda dem/anropa dem.

När vi genererar kod så genereras det även en proxyklass som gör det hela lite lättare för oss. Det genereras en sådan här för varje ServiceContract typ som vi har.

Namnet byggs alltid upp enligt följande:

ServiceContract namn (minus I) + Client.

Exceptions

Där är i huvudsak två typer av exceptions som vi behöver vara beredda på/medvetna om.

Den ena är **CommunicationException** som är en slags basklass för exceptions. Vilket gör att vi kan göra en ”Gona catch em all” och fånga exceptions av denna typ. Men det finns ett annat exception som du nog gärna vill fånga och det är FaultException.

TimeOutException denna typ ärver inte av CommunicationException, och skickas när vi överskrider Timeout värdet.

Varför SOAP?

REST har ingen standard för meddelandesystem och förväntar sig att klienterna hanterar kommunikationsfel genom att försöka igen.

SOAP har ”succesful/retry” logik inbyggd och ger end-to-end tillförlitlighet även genom SOAP mellanhänder.

REST är fundamentalt olika ifrån SOAP

SOAP definierar en transport-neutral modell, fokuserat på definiera customiserade tjänste-kontrakt med customiserade operationer och du kan anropa dessa operationer över en mängd olika transporter genom att använda olika sätt att koda för meddelanden.

REST å andra sidan definierar mer en transport specifik modell. REST binder dig inte till HTTP, men i realitet, HTTP är det enda protokollet som praktiskt används idag för att bygga RESTfulla arkitekturer. Så dessa två går hand i hand.

Förstå REST

- Definiera resurser med unikt id eller URI
- Enhetliga service kontrakt
- GET, PUT, POST, DELETE och HEAD
- HTML, XML, RSS eller JSON

Vad SOAP betonar

SOAP har betoningen på verb eller handlingar/händelser, medans **REST** har betoningen på adjektiv eller resurser.

När du definierar en SOAP tjänst så är ditt fokus på att definiera service kontraktet och uppsättningen med service operationer som vi skall exponera genom tjänsten.

Vikten av GET metoden

- Ca 90% av dagens webb trafik använder GET.
- Idempotenta GET requests.
- POST PUT och DELETE, ger inte samma garantier som GET gör.
- REST & GET.
- SOAP anropen tunnlar genom ett POST request.

Vikten av webformaten

- Räckvidden på tjänsten begränsas av formatets stöd.
- XML vanligaste formatet.
  - RSS
  - Atom
- JSON

WebGetAttribute

- Nya attribut för att mappa HTTP request.
- WebGetAttribute, mappar HTTPGet mot en viss metod.
- UriTemplate.
- Vill vi använda XML eller JSON för meddelande?

WebInvokeAttribute

- WebInvokeAttribute
- Optimerar GET separat.
- POST är default.
- Man antar att den sista parametern ska deserialiseras.

Hypermedia & REST?

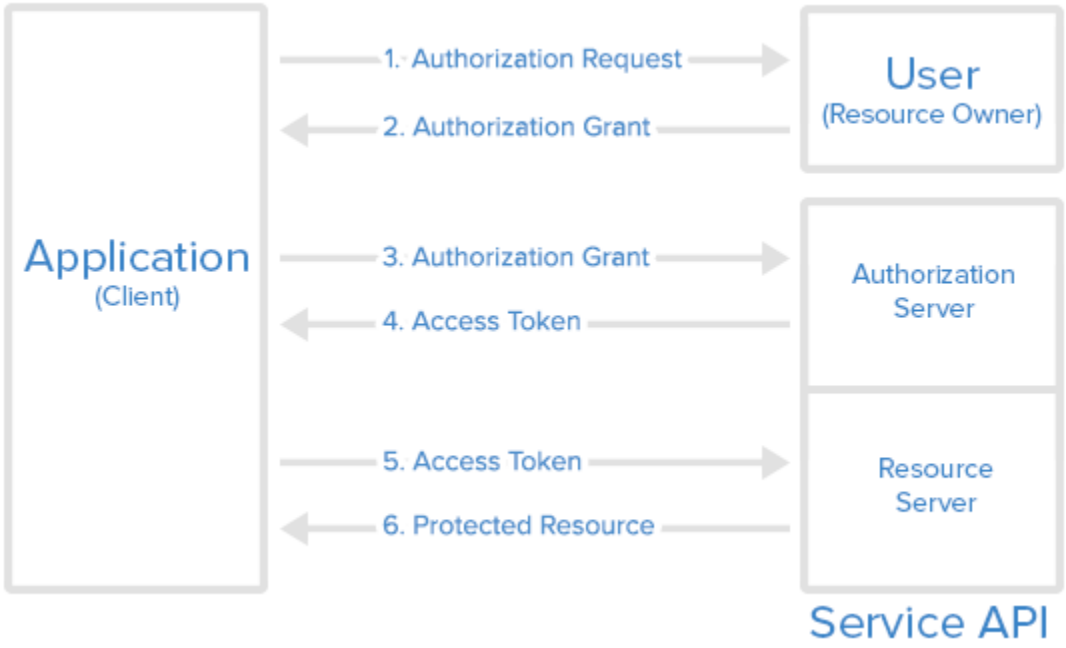
- Flickr API’et.
- Fixerat API, får ej ändras?
- Enkelt kunna gå mellan innehåll.
- Enhetligt gränssnitt.

4 roller

Oauth definierar 4 roller:

- Resurs ägare.
- Klient.
- Resurs server.
- Autentiserings server.

Protokoll flödet



Fördelar med token-baserad autentisering

- **Skalbarhet för servrar**
- **Lös koppling**
- **Mobilvänlig**

OpenID Connect

OpenID Connect, publicerad första gången 2014, är inte den första standarden för IdP, utan enligt en del det bästa när det gäller användbarhet och enkelhet, efter att ha tagit lärdomar ifrån tidigare insatser som SAML och OpenID 1.0 och 2.0.

Vad är OpenID Connect’s formel för framgång?

- Lätt att konsumera identitetstoken.
- Baserat på OAuth 2.0-protokollet.
- Enkelhet

ID token objekt

ID-token-påståenden, eller claims, är förpackade i ett enkla JSON-objekt:

```
{
  "sub"       : "alice",
  "iss"       : "https://openid.c2id.com",
  "aud"       : "client-12345",
  "nonce"     : "n-0S6_WzA2Mj",
  "auth_time" : 1311280969,
  "acr"       : "c2id.loa.hisec",
  "iat"       : 1311280970,
  "exp"       : 1311281970,
}
```

```
eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzcyI6ICJodHRw
Oi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5NzYxMDAxIiw
KICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZfV3pBMk1qIi
wKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5NzAKfQ.ggW8hZ
1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6qJp6IcmD3HP9
9Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJNqeGpe-gccM
g4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7TpdQyHE5lcMiKP
XfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoSK5hoDalrcvR
YLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4XUVrWOLrL10
nx7RkKU8NXNHq-rvKMzqg
```

ID-tokenhuvudet, fodrar JSON och signaturen kodas in i en bas 64-URL-säker sträng för enkelt skicka runt, till exempel som URL-parameter.

Lite enkelt skillnaden mellan Oauth-OpenID

I viss utsträckning kan auktorisering missbrukas i någon pseudo-autentisering på grund av att om enhet A erhåller från B en åtkomstnyckel via OAuth och visar den till servern S, kan servern S härleda att B verifierade A innan denne beviljades åtkomstnyckeln. Så vissa använder OAuth där de ska använda OpenID.

Jag tycker att denna pseudo-autentisering är mer förvirrande än vad något annat.

OWASP 10

OWASP topp tio är ett kraftfullt medvetenhetsdokument för webbapplikations säkerhet. Det representerar en bred enighet om de mest kritiska säkerhetsriskerna för webbapplikationer. Projektmedlemmar inkluderar en mängd säkerhetsexperter från hela världen som har delat med sig av sin kompetens för att producera denna lista. Vi uppmanar alla företag att anta detta medvetenhetsdokument inom sin organisation och starta processen för att se till att deras webbapplikationer minimerar riskerna.

Vad är OWASP topp tio?

OWASP topp tio ger:

- En lista över 10 mest kritiska säkerhetsapplikationer för webbapplikationer.
- För varje risk det ger:
  - En beskrivning.
  - Exempel attacker.
  - Vägledning om hur man undviker.
  - Referenser till OWASP och andra relaterade resurser.