
# Sviluppare servizi web con WebApi e oData

**Raffaele Rialdi**


Twitter: @raffaeler

Email: malta@vevy.com

Articoli e codice: http://www.iamraf.net

Blog: http://blogs.ugidotnet.org/raffaele

Profilo MVP: https://mvp.support.microsoft.com/profile/raffaele

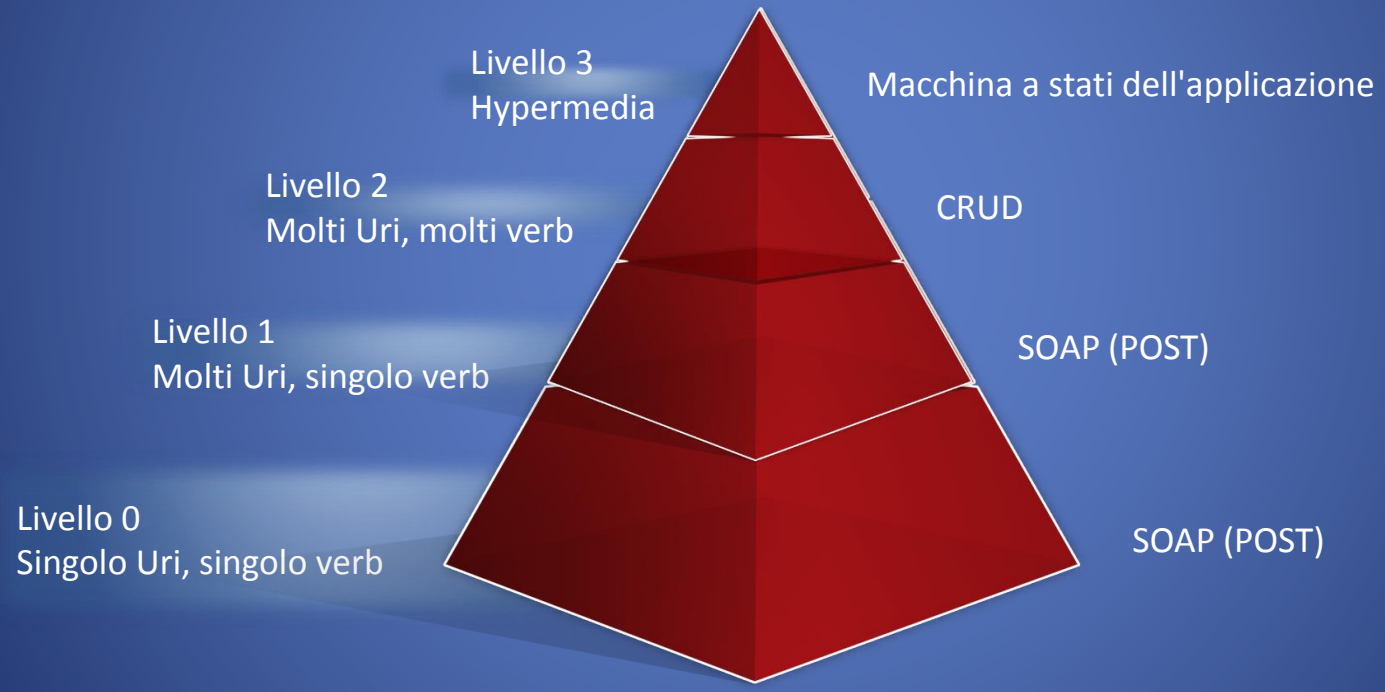
# Il Web Service «In principio era SOAP …»

- Le quattro regole SOA di WCF
  - I confini sono espliciti
    - Definiti da un contratto, il loro attraversamento ha un costo
  - I servizi sono autonomi
    - Il contratto consente l'evoluzione dell'implementazione
  - Sono pubblici solo lo schema e contratto
    - Il formato neutro XML consente di sopravvivere alle tecnologie
  - Le policy determinano capacità e requisiti del servizio

# REST
# REpresentational State Transfer

- È uno stile architetturale
  - Un'astrazione, una modalità di interazione con un Sistema
- Usa HTTP in toto come protocollo applicativo
- Impone un design stateless
  - ogni richiesta è auto-descrittiva
  - il client è detentore dello stato delle richieste
- Impone un basso accoppiamento ("loosely-coupled")
  - il client conosce solo l'entry-point del server

# Il «maturity level» di Leonard Richardson



# Hypermedia

- Hypermedia as the engine of application state (HATEOAS)
  - Il cambio di stato avviene eseguendo le 'request' reperite nelle 'response' del server sotto forma di Hyperlink
- Le risposte del server NON sono le entità
  - Le risposte descrivono lo stato applicativo
  - Le risposte contengono le entità

# SOAP vs REST

- Contratto rigido
  - problemi di versioning
- Metadati del servizio
  - Descrive le policy

- Indicato quando …
  - basso numero di client
  - client omogenei (.net, java, …)
  - protocollo diverso da HTTP

- Hypermedia style
  - scoperta dinamica
- Requisiti di un client
  - HTTP ed entry-point

- Indicato quando …
  - alto numero di client
  - client eterogenei
  - client poco potenti (IoT)
  - portabilità del servizio su altre piattaforme

# Perché HTTP

- Permette alta scalabilità
  - È un protocollo disconnesso
- È molto veloce
  - Interamente implementato in kernel mode (HTTP.SYS)
- Non è (solo) un protocollo di trasporto ma anche un protocollo **applicativo**
  - Lo standard descrive la semantica e i codici di ritorno
- L'infrastruttura IT è la stessa delle pagine web
  - I Proxy per le politiche di Caching
  - Gli apparati di routing e load balancing
- La security ha una lunga storia alle spalle (SSL)

# Chi ha detto che http è 'lento'?

## 100 Persons con 1K blob

GetPersons(100)

| scheme | elapsed | |
|---|---|---|
| basichttp | 5,719 | |
| wshttp | 2,097 | |
| net.tcp | 1,655 | |
| net.pipe | 1,666 | |
| RIAsoap | 5,972 | |
| MVCjson | 88,848 | |
| WebAPIjson | 82,198 | * |
| WebAPIbson | 2,617 | |
| WebAPIxml | 2,571 | |

(*) standard json

GetPersons(100)

| scheme | elapsed | |
|---|---|---|
| basichttp | 5,829 | |
| wshttp | 2,052 | |
| net.tcp | 1,562 | |
| net.pipe | 1,454 | |
| RIAsoap | 5,706 | |
| MVCjson | 88,621 | |
| WebAPIjson | 3,502 | * |
| WebAPIbson | 1,762 | |
| WebAPIxml | 2,634 | |

(*) json.net

## 500 Persons senza blob

GetPersons(**500**)

| scheme | elapsed | |
|---|---|---|
| basichttp | 8,353 | |
| wshttp | 6,865 | |
| net.tcp | 5,920 | |
| net.pipe | 5,875 | |
| RIAsoap | 8,476 | |
| MVCjson | 8,537 | |
| WebAPIjson | 4,932 | * |
| WebAPIbson | 2,275 | |
| WebAPIxml | 2,266 | |

(*) standard json

GetPersons(**500**)

| scheme | elapsed | |
|---|---|---|
| basichttp | 8,387 | |
| wshttp | 6,898 | |
| net.tcp | 5,942 | |
| net.pipe | 5,843 | |
| RIAsoap | 8,563 | |
| MVCjson | 8,414 | |
| WebAPIjson | 4,248 | * |
| WebAPIbson | 4,719 | |
| WebAPIxml | 2,266 | |

(*) json.net

# ASP.NET WEBAPI

# WebAPI

- WebAPI è un framework per costruire servizi HTTP
  - Si appoggia allo stack di asp.net
- Permette lo sviluppo di servizi REST
  - Un servizio WebApi **NON** è automaticamente REST
- È flessibile e pluggabile
  - Offre una pipeline semplice da espandere
- Usa concetti consolidati in asp.net MVC
  - asp.net routing per mappare un URL ad un controller (classe) e action (metodo della classe controller)

# Ciclo di vita di una richiesta

pipeline WebAPI

Client | Request | HTTP Server | URI routing | Execution | Response

- URL
- Header
- *Body (*)*
- Verb

/Values/1

**Controller:** ValuesController
**Action:** Get
**Parameter:** 1

- HTTP Status code
- Headers
- Body

Headers | TextView | WebForms | HexView | Auth | Cookies | Raw | JSON | XML

```
GET http://10.0.50.84:9999/api/Photos/Integrated%20Webcam HTTP/1.1
User-Agent: Fiddler
Host: 10.0.50.84:9999
accept: */*
```

Find... (press Ctrl+Enter to highlight all)

Get SyntaxView | Transformer | Headers | TextView | ImageView | HexView | WebView | Au

```
HTTP/1.1 200 OK
Content-Length: 5427
Content-Type: image/jpeg
Server: Microsoft-HTTPAPI/2.0
X-Cam: Image Captured by Raf
Date: Thu, 09 Oct 2014 11:57:18 GMT

♦♦♦♦♦JFIF♦    ♦`♦`♦♦♦♦♦C♦□---•-|□•••              □
```

["Integrated Webcam"]

# Configurazione

1. Global.asax.cs (modificato dal template)

```
public class WebApiApplication : System.Web.HttpApplication
{
    protected void Application_Start() {
        GlobalConfiguration.Configure(WebApiConfig.Register);
    }
}
```

2. WebApiConfig.cs (generato dal template)
   - Abilita due diverse possibilità di routing

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config) {
        config.MapHttpAttributeRoutes();   // abilita gli attribute

        config.Routes.MapHttpRoute(  // aggiunge una route secondo un pattern
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
}
```

# Gestire la request

- Grazie al routing vengono individuati
  - La classe del controller
  - Il nome della action (nome del metodo) da eseguire

- Il suffisso "Controller" è aggiunto per convenzione

http://localhost:1111/api/orders

```
public class OrdersController : ApiController
{
    public IHttpActionResult Get() {
        var ctx = new NorthwindContext();
        var list = ctx.Orders;
        return Ok(list);
    }
}
```

# Le tabelle di routing

- Possono essere mappati dalle tabelle di routing
  - Occhio al nome
  - O anche con l'attributo [FromBody]
- Possono essere aggiunti altri routing

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

```
// http://localhost:1111/api/orders/10248
public IHttpActionResult Get(long id)
```

```
// PUT api/Customers/5
public void Put(string id,
  [FromBody]Customer updatedCustomer)
```

```
config.Routes.MapHttpRoute(
    name: "api2",
    routeTemplate: "api/Special/{action}");
```

http://localhost:1111/api/Special/Add?x=2&y=5

```
public class SpecialController : ApiController {
    [HttpGet]
    public HttpResponseMessage Add(int x, int y) {
        var result = x + y;
        var response = this.Request.CreateResponse(
              HttpStatusCode.OK, result);
        return response;
    }
}
```

# Attributi di Routing

- Abilitato via config.MapHttpAttributeRoutes();
  - Eseguito via attribute: RoutePrefix e Route

```
[RoutePrefix("api/Demo")]
public class RafController : ApiController
```

```
[Route("raf1")]
public IHttpActionResult Get()
```

  - Oppure direttamente sulla action specificando anche i parametri

```
[Route("Orders/{id}/details")]
public IEnumerable<Order> GetDetailsByOrder(int id) { ... }
```

- Supporta numerosi formati tra cui le regex

# Action

- È possibile rinominare le action
  - [ActionName("NuovoNomeMetodo")]
- Oppure escluderle dalla selezione delle Action
  - [NonAction]
- Il parametro di ritorno può essere
  - Il type CLR desiderato
  - IHttpActionResult
  - HttpResponseMessage

```
public class ResponseMessageResult : IHttpActionResult
{
    public ResponseMessageResult(HttpResponseMessage response);
    public HttpResponseMessage Response { get; }
    public virtual Task<HttpResponseMessage> ExecuteAsync(CancellationToken ct);
}
```

# Verb

- I verb sono fondamentali perché rispondono alla semantica HTTP
  - GET and HEAD devono solo leggere
    - sono soggetti a cache!
  - GET, HEAD, PUT and DELETE sono idempotenti
    - La loro ripetizione non provoca la riesecuzione del comando
  - POST Non è idempotente
    - usato per le insert ➔ ha senso inserire più volte
- Associare un verb ad una action
  - [HttpGet], [HttpPost], etc.
  - [AcceptVerbs("GET", "HEAD")]

# Errori, exceptions e HTTP

- Gli status code descrivono anche gli errori
  - Le exception lato server sono convertite in errori HTTP
- È possibile aggiungere un messaggio all'errore

```
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8
Date: Thu, 09 Aug 2012 23:27:18 GMT
Content-Length: 51

{ "Message": "Product with id = 12 not found" }
```

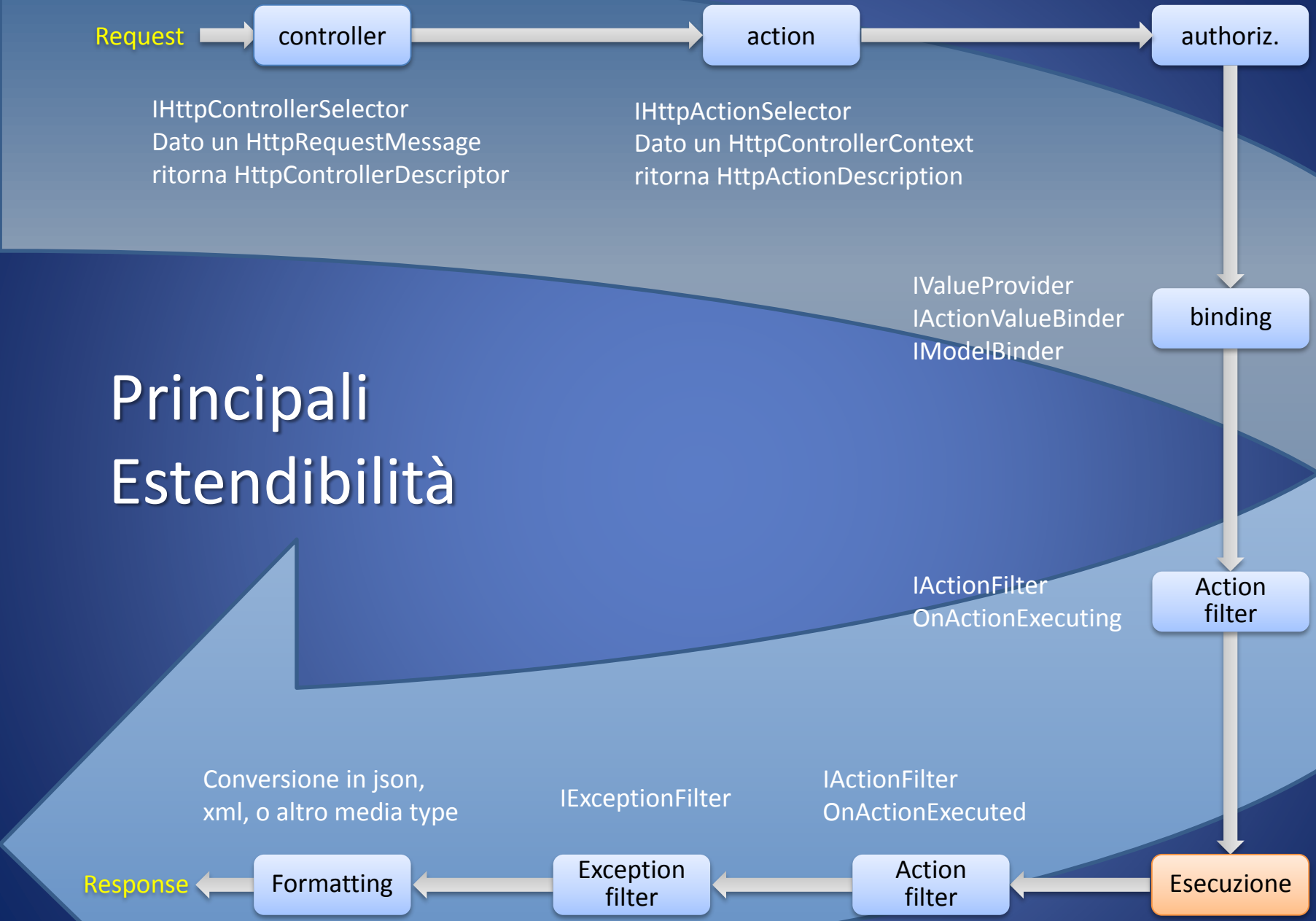
  - Usando CreateResponse + HttpError

```
var msg = string.Format("Product with id = {0} not found", id);
HttpError err = new HttpError(msg);
return Request.CreateResponse(HttpStatusCode.NotFound, err);
```

  - Usando CreateErrorResponse

```
var msg = string.Format("Product with id = {0} not found", id);
return Request.CreateErrorResponse(HttpStatusCode.NotFound, msg);
```

# Principali Estendibilità



# MediaTypeFormatter

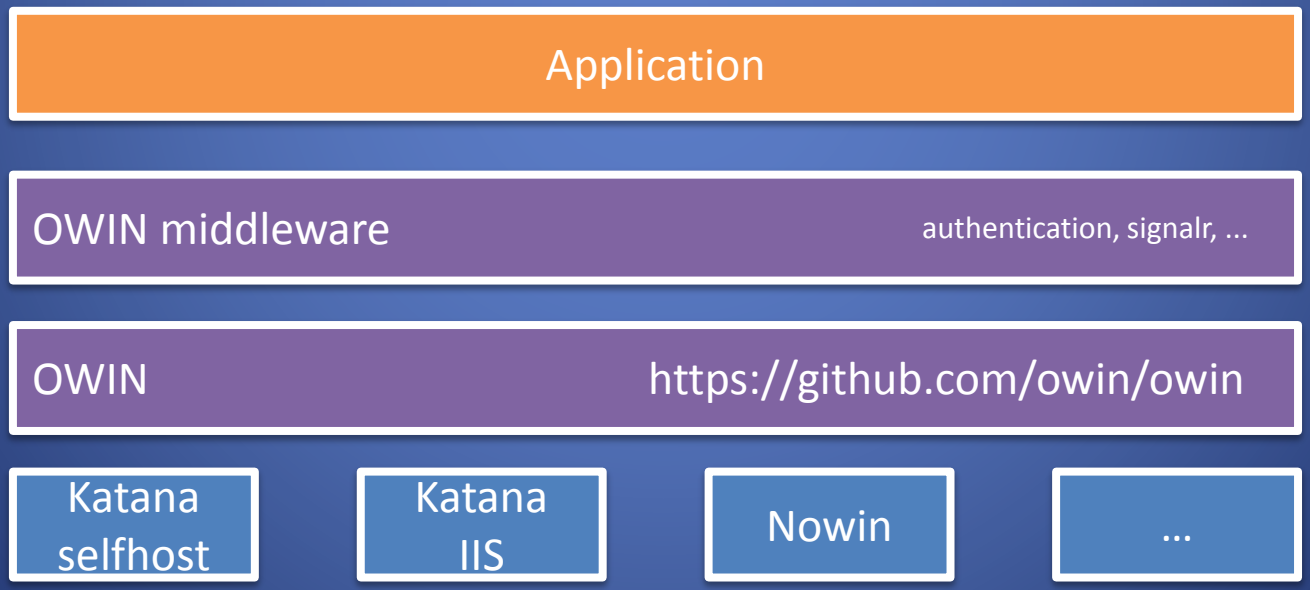
Demo time

- MediaTypeFormatter è la classe astratta per serializzare una response in qualsivoglia format
- I formatter già presenti nella pipeline:
  - Json.net e Xml formatter
  - bson e molti altri su nuget
  - Uno nostro formatter custom derivando questa classe
- La scelta avviene dietro una negoziazione standard HTTP
  - Il client manda un header "accept"
  - Il server risponde con un "content-type"

# OWIN & KATANA

# OWIN, uno standard per unificare lo stack dei component web in .NET

Consente di eseguire le applicazioni su una moltitudine di differenti host

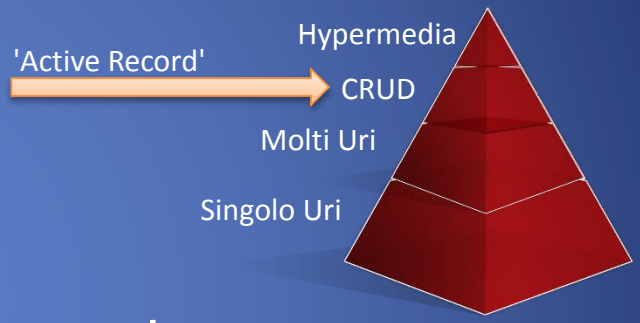


Device + Services +
+ OWIN + ...

# ODATA

# oData

- oData è un'insieme di specifiche create per esporre dati
  - Definizione della forma degli URI
  - Operazioni di tipo CRUD per agire sulle **entità** e le loro **relazioni**



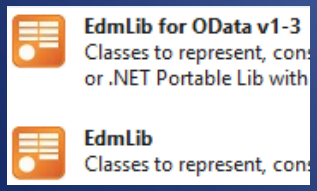

- Il formato di uscita può essere AtomPub o Json
- Esiste già un ecosistema di client che usano questo protocollo
  - Tra questi MS Office Excel e PowerQuery (ex PowerPivot)
- Per creare servizi che espongono oData:
  - Microsoft ha deprecato i WCF Data Services
  - Sta maturando lo stack oData per le WebAPI

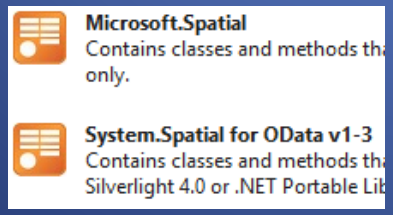
# Le version di oData

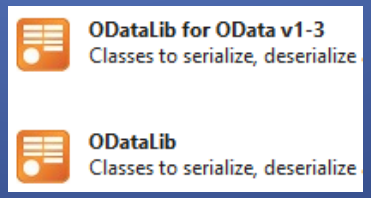
- oData v3 è un protocollo creato da Microsoft le cui specifiche sono rilasciate pubbliche
  - http://odata.org
- oData v4 è stato ceduto a OASIS che lo ha standardizzato
  - https://www.oasis-open.org/committees/odata/charter.php
- Differenze:
  - La v4 introduce breaking-changes rispetto alla v3
  - La v4 semplifica molti costrutti
  - La v3 è ancora la più diffusa tra i client esistenti
  - La v3 e v4 possono co-esistere (namespace differenti)

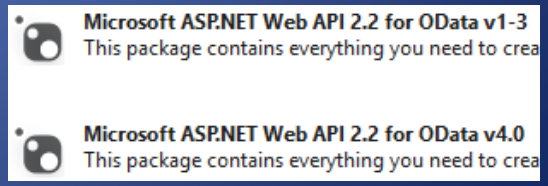
# Crazy Versioning

- Namespace versioning
  - Microsoft.OData.Core → ODataLib v4.0
  - Microsoft.Data.OData → ODataLib v3.0
  - System.Web.OData → OData v4.0
  - System.Web.Http.OData → OData v3.0
- A partire dalla v4 …
  - Extension methods spostati in System.Web.OData.Extensions
  - MapODataRoute → MapODataServiceRoute
  - QueryableAttribute → EnableQueryAttribute
- Se mischiate, l'errore più probabile è un HTTP 406
- Su nuget i package si distinguono facilmente:



# Endpoint dei metadati

- Totale antitesi con la filosofia Hypermedia
- Schema in format XML che descrive
  - Entity (type system)
  - EntitySet (collection di entity)
  - Action (funzioni che cambiano lo stato dei dati)
    - Da non confondersi con le Action di WebApi
  - Function (funzioni readonly)
  - Complex type (entity di appoggio a funzioni e action)
- Permette l'auto-generazione di client

# La CRUD di oData

- GET
  - Esegue la request di una collection, un singolo element o una proprietà
  - risponde con il body e 200 (OK)
- POST
  - Inserisce una singola entità
  - risponde con 204 (no content)
- PUT
  - modifica una singola entità
  - risponde con 204 (no content)
- MERGE / PATCH
  - modifica parziale di una singola entità
  - risponde con 204 (no content)
- DELETE
  - cancellazione di una singola entità
  - risponde con 200 (OK)

# Filtrare i dati ($filter)

- La QueryString definisce la query
  - http://services.odata.org/Northwind/Northwind.svc/Customers?$filter=substringof('Alfreds', CompanyName) eq true
  - numerosi operatori applicabili alla query
- Le query vengono tradotte in Expression Tree di Linq
  - Il metodo si limita a ritornare un IQueryable<T>
  - L'expression finale verrà tradotta in query SQL dall'ORM
- Vantaggi principali:
  - Query dinamiche
  - Si evitano problem di SQL Injection
  - Paginazione dei dati evitando così di stressare il DB

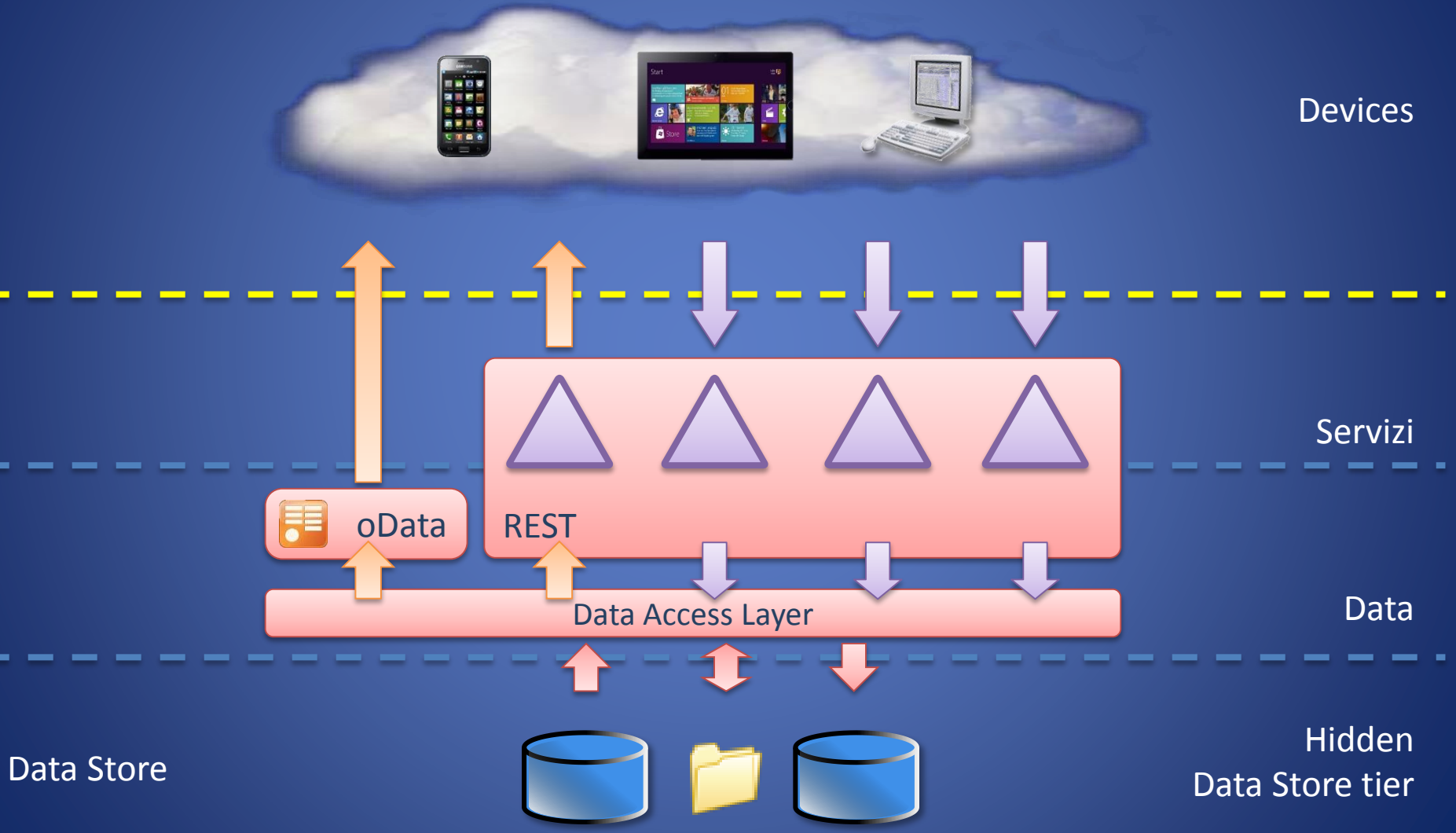
# oData $filter cheat sheet

- espressioni
  - [eq, ne], [lt, le], [gt, ge]
  - $filter=name eq 'raf'
  - or, and, not
- date
  - day, month, year
  - hour, minute, second
- matematica
  - add, sub, mul, div, mod (+, -, *, /, %)
  - $filter=price add 1 lt 10
    - (price+1) <10
  - round(price)
  - floor(price)
  - ceiling(price)
- Date
  - date, time, totaloffsetminutes, now
  - maxdatetime, mindatetime, totalseconds
  - year, month, day
  - hour, minute, second, fractionalseconds
- stringhe
  - substringof('Raf', name) eq true
  - startswith(name, 'Raf') eq true
  - endswith(name, 'ele') eq true
  - indexof(name, 'Raf') eq 0
  - length(name) gt 0
  - replace(name, 'R', 'r') eq \"raf\"
  - substring(name, 5) eq \"ele\"
  - substring(name, 2, 2) eq \"ff\"
  - tolower(name) eq \"raffaele\"
  - toupper(name) eq \"RAFFAELE\"
  - trim(name) \"raf\"
  - concat('X', 'Raf') eq \"XRaf\"
  - contains(name,'ff')
- Tipi
  - isof
  - cast
- Geo
  - geo.distance, geo.intersects, geo.length
- Lambda
  - any, all

# Non solo filtri

| Operazione | Operatore | Esempio |
|---|---|---|
| **Count di un EntitySet (collection)** | $count | |
| **Ordinamento** | $orderby asc/desc | $orderby=name%desc |
| **Paginazione** | $top $skip | |
| **Formato del payload** | $format | |
| **Proiezione** | $select | |
| **Espansione grafo** | $expand | |

# oData e SOA




Nota: ogni triangolo e' un set di servizi di una applicazione che usa gli stessi dati

# Domande ?