

# Academy Week 4




**Roberto Ajolfi**

Senior Developer@icubedsrl

[roberto.ajolfi@icubed.it](mailto:roberto.ajolfi@icubed.it)



# Week 4 - Agenda

- Il protocollo HTTP
- Internet Information Services (IIS)
- SOAP / Windows Communication Foundation (WCF)
- JSON / ASP.NET Core Web API (REST)
-  Esercitazione



# I Servizi Web

Un Servizio Web è un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su una medesima rete oppure in un contesto distribuito

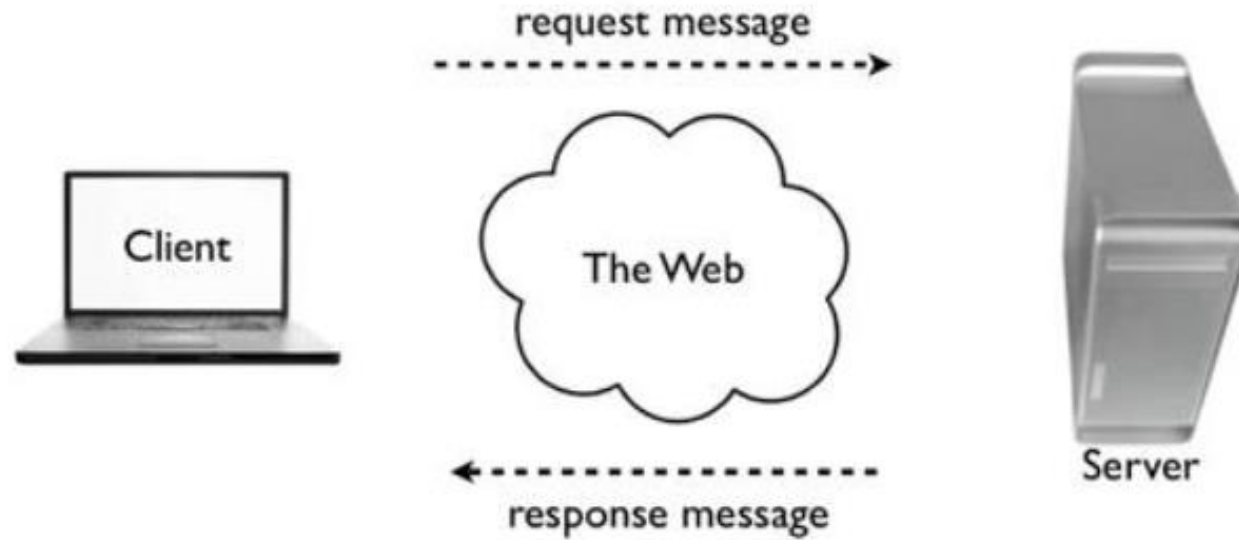
World Wide Web Consortium (W3C)

# I Servizi Web

- Si tratta di un'interfaccia attraverso la quale due dispositivi (o applicazioni) possono comunicare tra loro
- Due sono le caratteristiche fondamentali:
  - *Multipiattaforma*
  - *Condivisione*

# I Servizi Web

Quando viene utilizzato un Web service, un client invia una richiesta a un server e vi provoca un'azione. Il server invia quindi una risposta al client

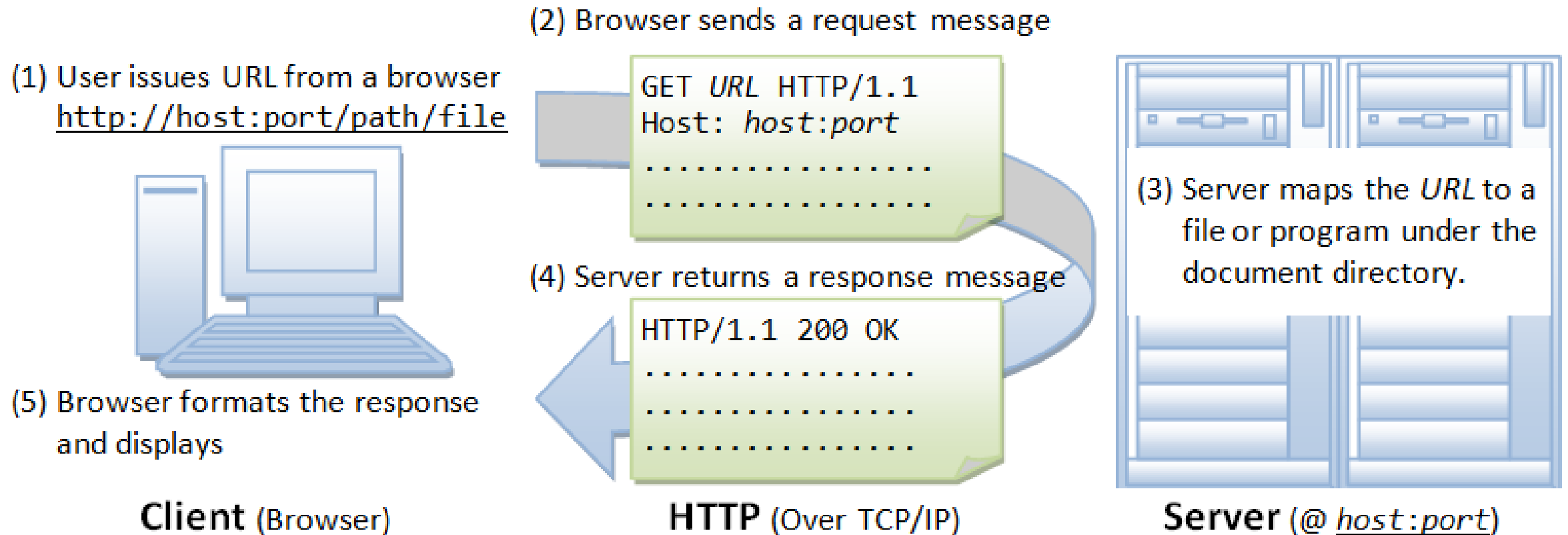


# Il protocollo HTTP

- Il protocollo HTTP (Hypertext Transfer Protocol) è un protocollo a livello di applicazione per sistemi informativi distribuiti, collaborativi e ipermediali
- È il fondamento della comunicazione dati per il World Wide Web
- Lo sviluppo di HTTP è stato avviato da **Tim Berners-Lee** al CERN nel 1989

| Year         | HTTP Version |
|--------------|--------------|
| 1991         | 0.9          |
| 1996         | 1.0          |
| 1997         | 1.1          |
| 2015         | 2.0          |
| Draft (2020) | 3.0          |

# Il protocollo HTTP



# Il protocollo HTTP

- È un protocollo Client-Server, il che significa che le richieste vengono avviate dal destinatario
- Una sessione HTTP è una sequenza di transazioni di richiesta - risposta di rete
- Un client HTTP (User Agent) avvia una richiesta stabilendo una connessione TCP (Transmission Control Protocol) a una particolare porta su un server (in genere la **porta 80**)

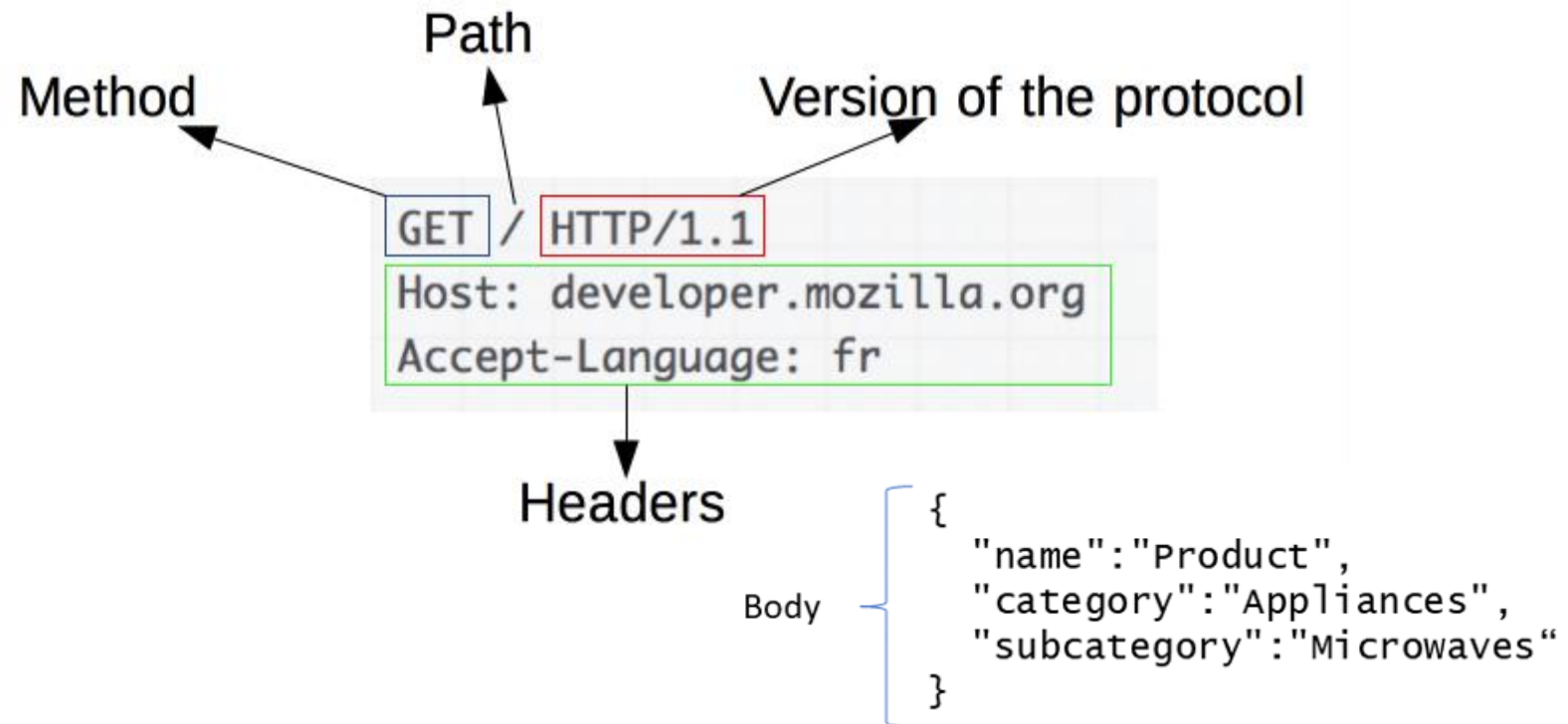


# Il protocollo HTTP

- Un server HTTP in ascolto su quella porta attende il messaggio di richiesta di un client
- Dopo aver ricevuto la richiesta, il server restituisce una riga di stato e un proprio messaggio
- Il corpo di questo messaggio è in genere
  - la risorsa richiesta
  - oppure un messaggio di errore
  - oppure altre informazioni

# Il protocollo HTTP

## Request



# Il protocollo HTTP

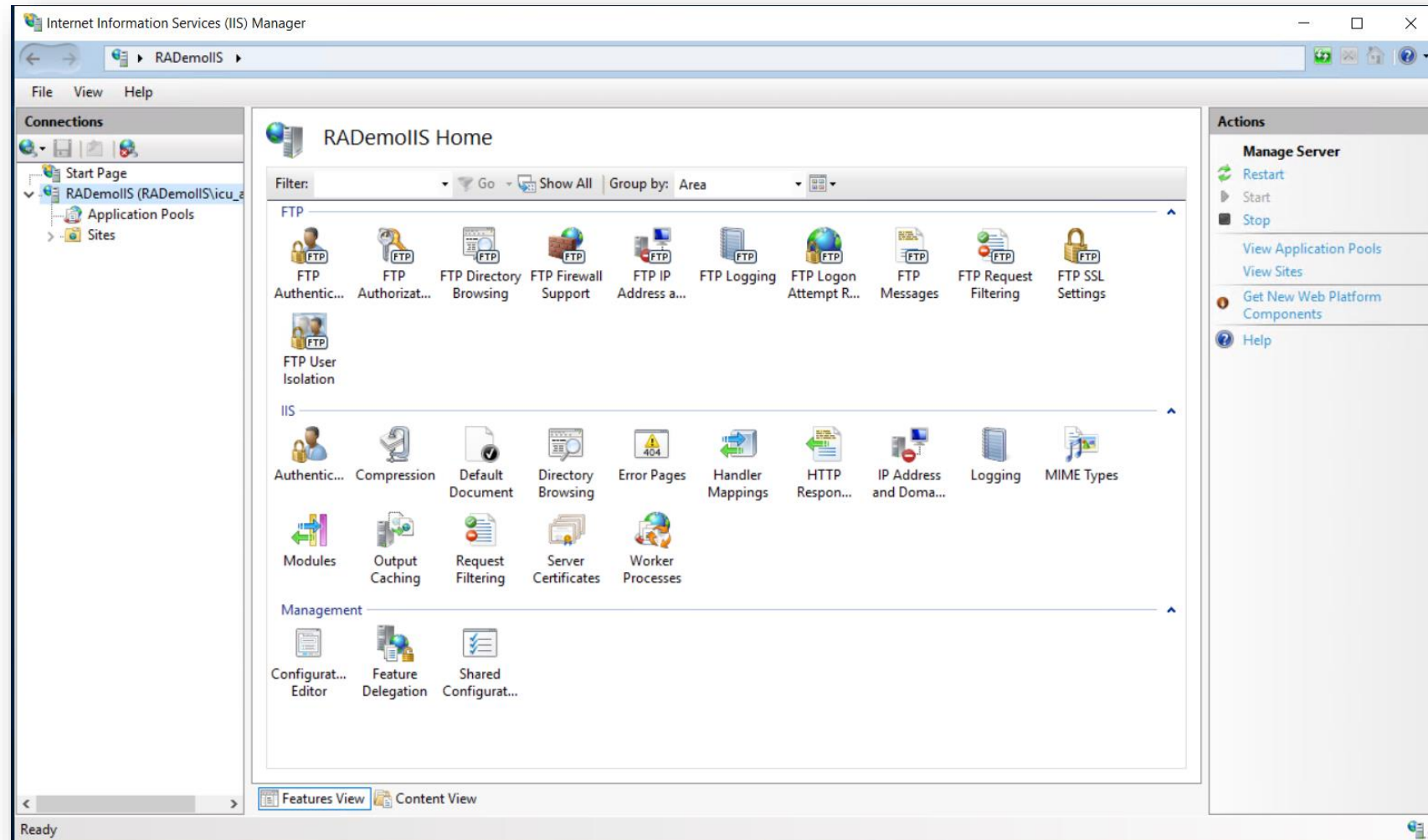
## Response

```
      HTTP      Status
      Version   Code
      {         {
HTTP/1.1 200 OK
      {
Headers { Content-Type: application/json; charset=utf-8
        { Server: Kestrel
          { X-Powered-By: ASP.NET
            { Date: Sun, 11 Feb 2018 18:34:00 GMT
              { Content-Length: 69
                {
Body { {
      { "name": "Product",
        { "category": "Appliances",
          { "subcategory": "Microwaves"
            {
          {
        {
      {
```

# Il protocollo HTTP

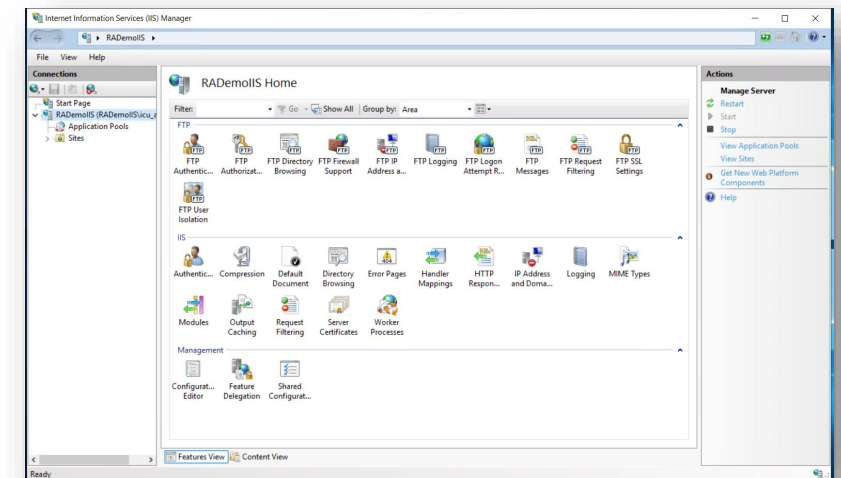
- HTTP è un protocollo **stateless**
  - Non esiste alcun collegamento tra due richieste eseguite successivamente sulla stessa connessione
  - Non è possibile conservare informazioni condivise tra due richieste
- Questo limite viene superato tramite i cookie HTTP
  - i cookie HTTP vengono aggiunti agli header dei messaggi, consentendo la condivisione degli stessi dati tra diverse richieste HTTP all'interno di una sessione

# Internet Information Service



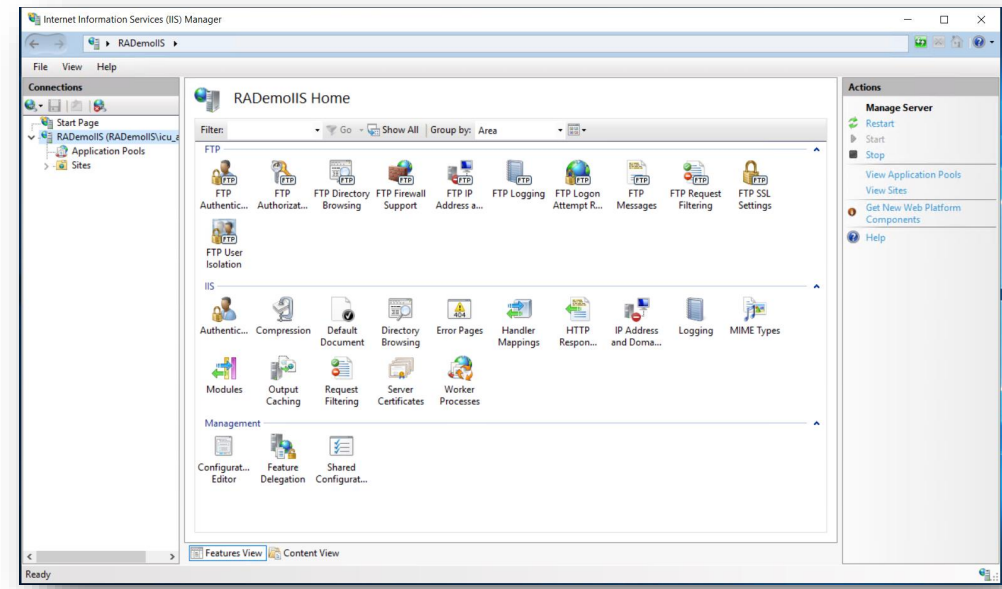
# Internet Information Service

- Internet Information Services (IIS, precedentemente Internet Information Server) è un complesso di servizi per realizzare un server Internet con i sistemi operativi Microsoft Windows
- Supporta HTTP, HTTP / 2, HTTPS, FTP, FTPS, SMTP e NNTP
- È parte integrante della famiglia Windows NT
- Non è attivo per impostazione predefinita
- La versione corrente, integrata in Windows Server 2016 e Windows 10, è la **10.0**



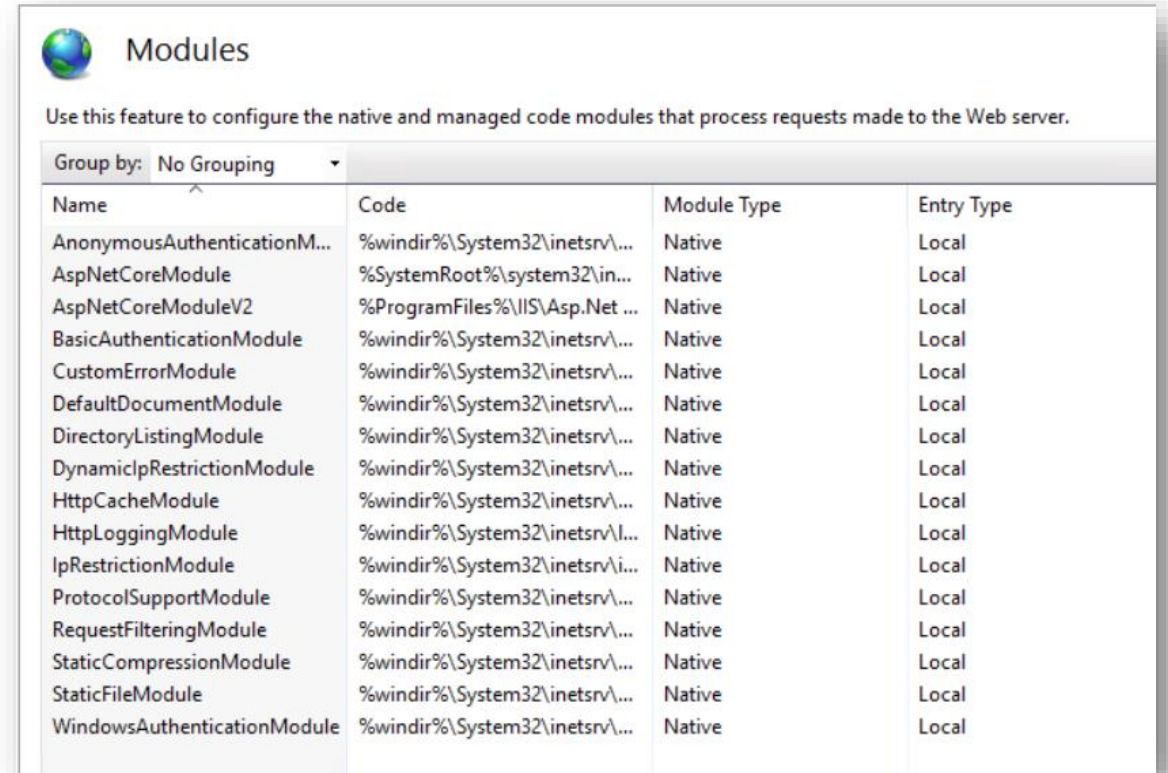
# Internet Information Service

- I dati di Netcraft a febbraio 2017 indicano che IIS aveva una "quota di mercato del primo milione di siti più trafficati" del 10,19%, rendendolo il terzo server web più popolare al mondo, dietro Apache al 41,41% e nginx al 28,34%



# Internet Information Service

- IIS non è in grado, di per sé, di eseguire elaborazioni server-side, ma ne delega l'esecuzione ai moduli ISAPI
  - Microsoft stessa fornisce una serie di moduli tra le quali quello per Active Server Pages (ASP) ed ASP.NET
  - Altri sviluppatori hanno reso disponibili i moduli per il supporto ai linguaggi PHP e Perl

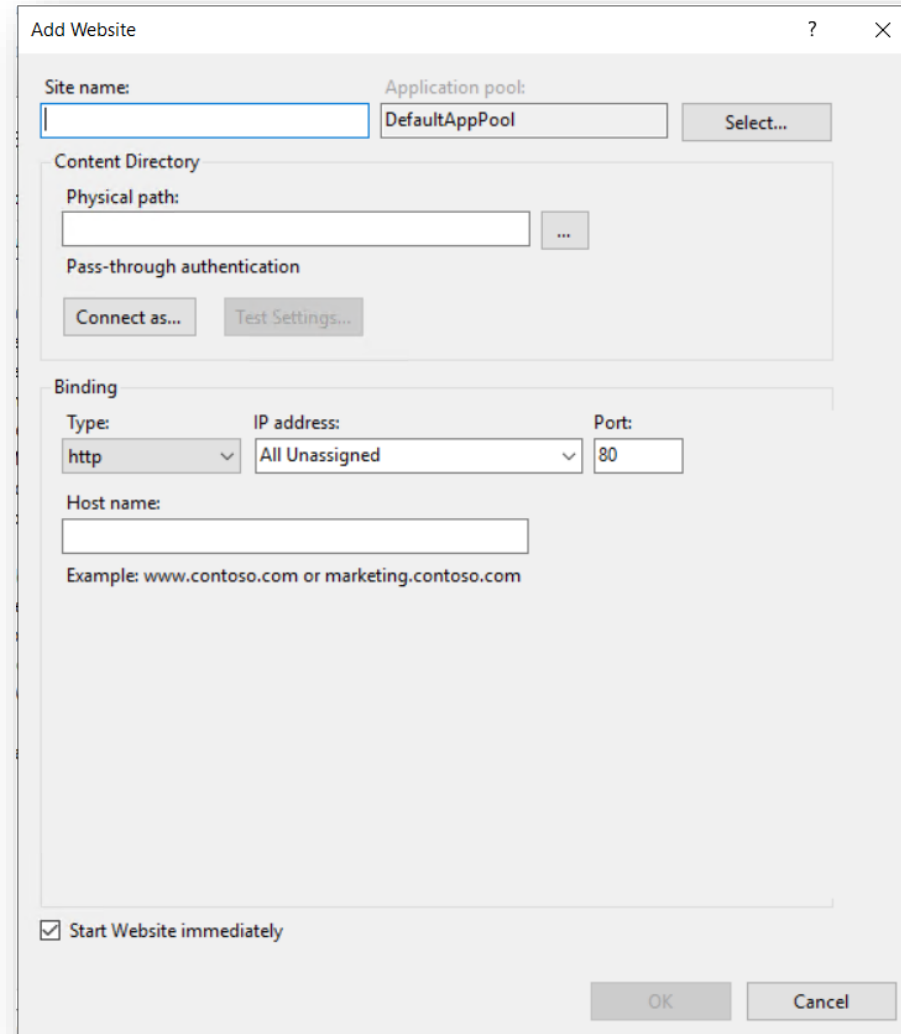
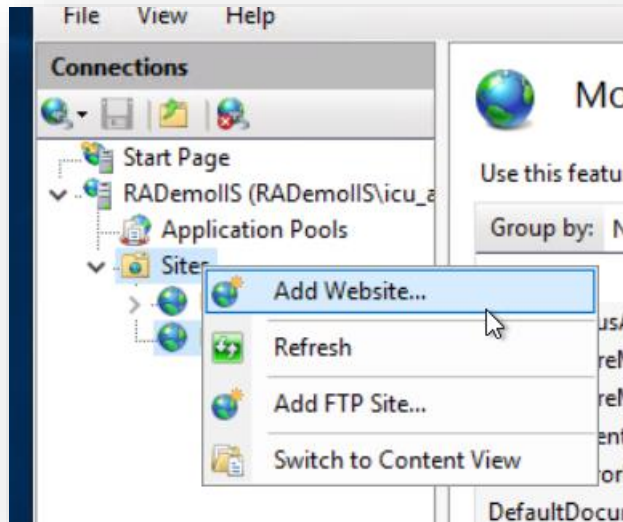


The screenshot shows the 'Modules' configuration window in IIS Manager. It features a title bar with a globe icon and the word 'Modules'. Below the title bar is a subtitle: 'Use this feature to configure the native and managed code modules that process requests made to the Web server.' A 'Group by:' dropdown menu is set to 'No Grouping'. The main area is a table with four columns: 'Name', 'Code', 'Module Type', and 'Entry Type'. The table lists 18 modules, all of which are 'Native' type and have a 'Local' entry type. The 'Code' column contains file paths, many of which are truncated in the image.

| Name                        | Code                           | Module Type | Entry Type |
|-----------------------------|--------------------------------|-------------|------------|
| AnonymousAuthenticationM... | %windir%\System32\inetsrv\...  | Native      | Local      |
| AspNetCoreModule            | %SystemRoot%\system32\in...    | Native      | Local      |
| AspNetCoreModuleV2          | %ProgramFiles%\IIS\Asp.Net ... | Native      | Local      |
| BasicAuthenticationModule   | %windir%\System32\inetsrv\...  | Native      | Local      |
| CustomErrorModule           | %windir%\System32\inetsrv\...  | Native      | Local      |
| DefaultDocumentModule       | %windir%\System32\inetsrv\...  | Native      | Local      |
| DirectoryListingModule      | %windir%\System32\inetsrv\...  | Native      | Local      |
| DynamicIpRestrictionModule  | %windir%\System32\inetsrv\...  | Native      | Local      |
| HttpCacheModule             | %windir%\System32\inetsrv\...  | Native      | Local      |
| HttpLoggingModule           | %windir%\System32\inetsrv\l... | Native      | Local      |
| IpRestrictionModule         | %windir%\System32\inetsrv\i... | Native      | Local      |
| ProtocolSupportModule       | %windir%\System32\inetsrv\...  | Native      | Local      |
| RequestFilteringModule      | %windir%\System32\inetsrv\...  | Native      | Local      |
| StaticCompressionModule     | %windir%\System32\inetsrv\...  | Native      | Local      |
| StaticFileModule            | %windir%\System32\inetsrv\...  | Native      | Local      |
| WindowsAuthenticationModule | %windir%\System32\inetsrv\...  | Native      | Local      |



# Internet Information Service

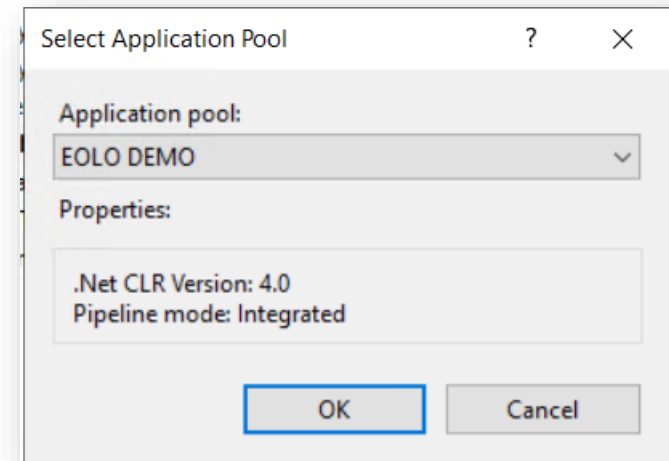


The 'Add Website' dialog box is shown, used for configuring a new website in IIS. It contains the following fields and options:

- Site name:** A text input field.
- Application pool:** A dropdown menu currently showing 'DefaultAppPool', with a 'Select...' button to the right.
- Content Directory:**
  - Physical path:** A text input field with a browse button ('...').
  - Pass-through authentication:** A checkbox.
  - Connect as...** and **Test Settings...** buttons.
- Binding:**
  - Type:** A dropdown menu set to 'http'.
  - IP address:** A dropdown menu set to 'All Unassigned'.
  - Port:** A text input field set to '80'.
  - Host name:** A text input field.
  - Example:** 'www.contoso.com or marketing.contoso.com'.
- Start Website immediately:** A checked checkbox.
- Buttons:** 'OK' and 'Cancel' buttons at the bottom right.

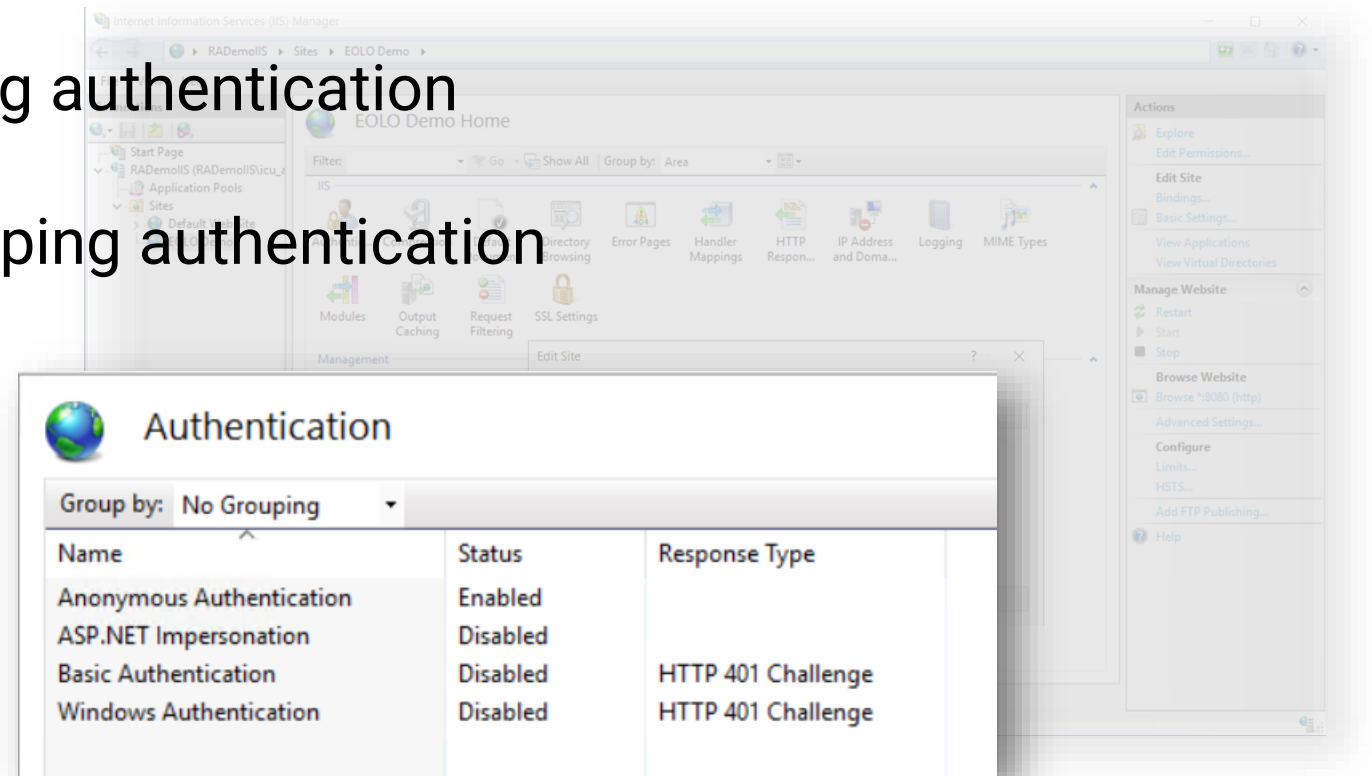
# Internet Information Service

- Gli Application Pool consentono di isolare le applicazioni l'una dall'altra, anche se sono in esecuzione sullo stesso server
  - In questo modo, se si verifica un errore in un'app, non verranno disattivate altre applicazioni
  - Inoltre, i pool di applicazioni consentono di separare diverse app che richiedono diversi livelli di sicurezza



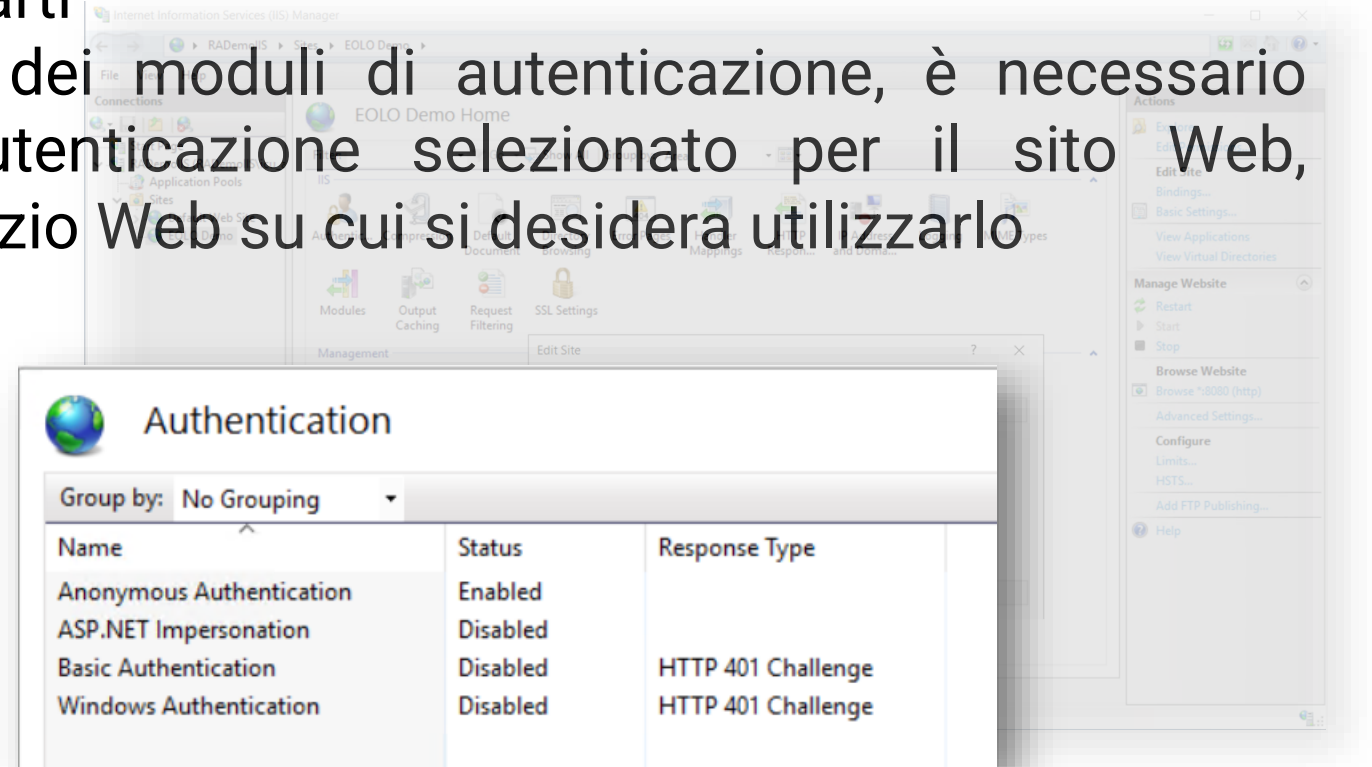
# Internet Information Service

- IIS 7 supporta
  - Anonymous authentication
  - Basic authentication
  - Client Certificate Mapping authentication
  - Digest authentication
  - IIS Client Certificate Mapping authentication
  - Windows authentication



# Internet Information Service

- Modalità di autenticazione aggiuntive possono essere fornite da moduli di autenticazione di terze parti
- Dopo aver installato uno dei moduli di autenticazione, è necessario abilitare il modulo di autenticazione selezionato per il sito Web, l'applicazione Web o il servizio Web su cui si desidera utilizzarlo

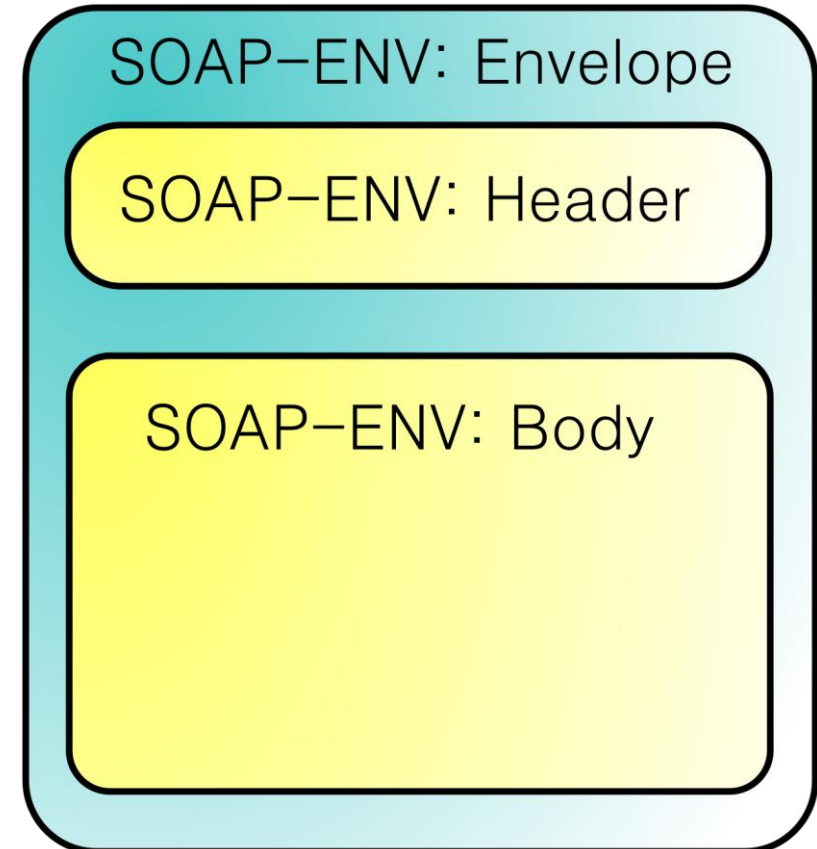


# SOAP

**SOAP** (acronimo di *Simple Object Access Protocol*) è un protocollo per lo scambio di messaggi tra componenti software.

SOAP può operare su differenti protocolli di rete, ma HTTP è il più comunemente utilizzato.

La parola "oggetto" sta ad indicare che l'uso del protocollo dovrebbe effettuarsi secondo il paradigma della programmazione orientata agli oggetti.



# SOAP

SOAP si basa su **XML** e la sua struttura segue una struttura **header-body**, analoga a quella dell'HTML.

L'**header** contiene metadati come quelli che riguardano l'instradamento, la sicurezza, le transazioni ...

Il **body** trasporta il contenuto informativo, talora viene detto carico utile (payload).

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetails xmlns="http://magazzino.example.com/ws">
      <productId>827635</productId>
    </getProductDetails>
  </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetailsResponse xmlns="http://magazzino.example.com/ws">
      <getProductDetailsResult>
        <productName>Toptimate, set da 3 pezzi</productName>
        <productId>827635</productId>
        <description>Set di valigie; 3 pezzi; poliestere; nero.</description>
        <price>96.50</price>
        <inStock>true</inStock>
      </getProductDetailsResult>
    </getProductDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

Il Payload deve seguire uno schema definito dal linguaggio **XML Schema**.

# WSDL

Il Web Services Description Language è un linguaggio formale in formato XML per la creazione di servizi web e la loro interazione.

```
<xs:element name="request">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="header" />
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string" />
        </xs:simpleContent>
      </xs:complexType>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<!-- Abstract Types -->

<interface name="RESTfulInterface">
  <fault name="ClientError" element="tns:response"/>
  <fault name="ServerError" element="tns:response"/>
  <fault name="Redirection" element="tns:response"/>
  <operation name="Get" pattern="http://www.w3.org/ns/wsd1/in-out">
    <input messageLabel="GetMsg" element="tns:request"/>
    <output messageLabel="SuccessfulMsg" element="tns:response"/>
  </operation>
  <operation name="Post" pattern="http://www.w3.org/ns/wsd1/in-out">
    <input messageLabel="PostMsg" element="tns:request"/>
    <output messageLabel="SuccessfulMsg" element="tns:response"/>
  </operation>
  <operation name="Put" pattern="http://www.w3.org/ns/wsd1/in-out">
    <input messageLabel="PutMsg" element="tns:request"/>
    <output messageLabel="SuccessfulMsg" element="tns:response"/>
  </operation>
  <operation name="Delete" pattern="http://www.w3.org/ns/wsd1/in-out">
    <input messageLabel="DeleteMsg" element="tns:request"/>
    <output messageLabel="SuccessfulMsg" element="tns:response"/>
  </operation>
</interface>

<binding name="RESTfulInterfaceSoapBinding" interface="tns:RESTfulInterface"
  type="http://www.w3.org/ns/wsd1/soap"
  wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
  wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
  <operation ref="tns:Get" />
  <operation ref="tns:Post" />
  <operation ref="tns:Put" />
  <operation ref="tns:Delete" />
</binding>

<!-- Web Service offering endpoints for both the bindings-->
<service name="RESTfulService" interface="tns:RESTfulInterface">
  <endpoint name="RESTfulServiceRestEndpoint"
    binding="tns:RESTfulInterfaceHttpBinding"
    address="http://www.example.com/rest/" />
  <endpoint name="RESTfulServiceSoapEndpoint"
    binding="tns:RESTfulInterfaceSoapBinding"
    address="http://www.example.com/soap/" />
</service>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<description xmlns="http://www.w3.org/ns/wsd1"
  xmlns:tns="http://www.example.com/wsd120sample"
  xmlns:xs="http://www.w3.org/ns/wsd1/http"
  xmlns:wsoap="http://www.w3.org/ns/wsd1/soap"
  targetNamespace="http://www.example.com/wsd120sample">
```

# WCF



Windows Communication Foundation (WCF) è un framework per la creazione di applicazioni connesse e orientate ai servizi (SOA).

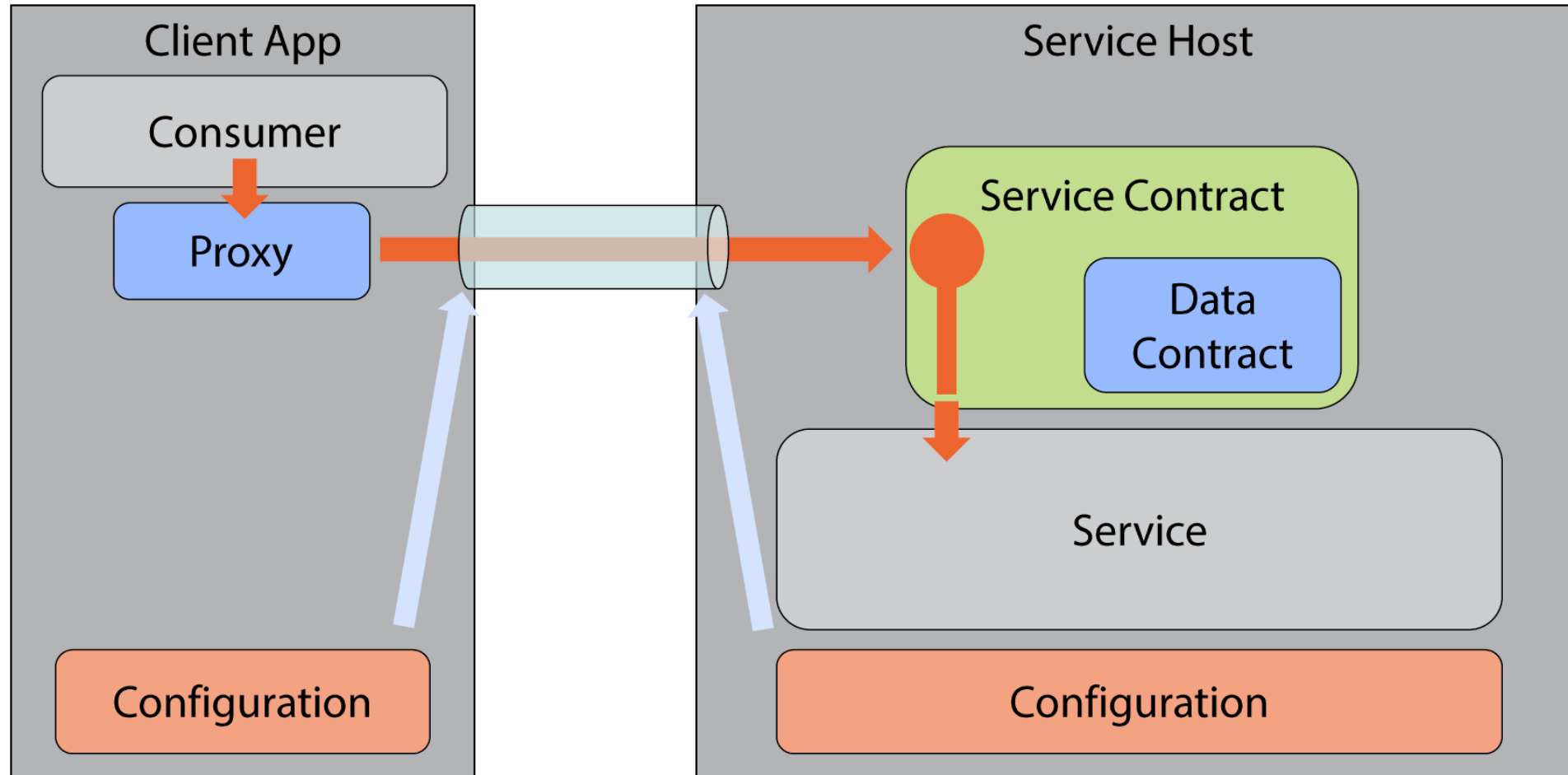
- Implementa i moderni standard di settore per l'interoperabilità dei servizi Web
- Supporta la pubblicazione dei metadati utilizzando WSDL, XML Schema e WS-Policy
- Secure-by-Default
- Estensibile



# WCF



Windows  
Communication  
Foundation



# ABC



Alla base di WCF c'è l'**ABC**:

- **Address**: dove il servizio viene pubblicato
- **Binding**: come il servizio viene pubblicato
- **Contract**: cosa pubblica il servizio

L'insieme di queste informazioni costituisce un **Endpoint**

# Address



L'indirizzo di un servizio è composto da un identificatore della tipologia ed un url

Es. "*http://MySite/MyService*"

Gli identificatori sono:

- **http(s)://** - WebService
- **net.tcp://** - TCP/IP
- **net.pipe://** - Named Pipes
- **net.msmq://** - Microsoft Message Queue
- **net.p2p://** - Peer To Peer

# Binding



Il binding è il meccanismo di comunicazione con cui il servizio viene pubblicato ed è completamente customizzabile:

- *BasicHttpBinding*
- *WSHttpBinding*
- *WSDualHttpBinding*
- *WSFederationBinding*
- *TcpBinding*
- *MsmqBinding*
- *MsmqIntegrationBinding*
- *NamedPipeBinding*

# Binding



Windows  
Communication  
Foundation

```
<system.serviceModel>
  <services>
    <service name="MyService">
      <host>
        <baseAddresses>
          <add baseAddress="net.tcp://localhost:8001"/>
        </baseAddresses>
      </host>
      <endpoint address="Service" binding="basicHttpBinding" contract="IService" />
    </service>
  </services>
  <bindings>
    <basicHttpBinding>
      <binding name="DuplexBinding" >
        <reliableSession enabled="true" />
        <security mode="None" />
      </binding>
    </basicHttpBinding>
  </bindings>
</system.serviceModel>
```

# Contract

Il **Contract** rappresenta i metodi che un servizio espone all'esterno:

- Definiti in un'interfaccia .NET
- Implementati nel servizio che la espone

```
[ServiceContract]
public interface IService
{
    [OperationContract]
    string Method(int parameter);
}

public class MyService : IService
{
    public string Method(int parameter)
    {
        return parameter.ToString();
    }
}
```

# Data Contract



Un contratto dati è un accordo formale tra un servizio e un cliente che descrive in modo astratto i dati da scambiare.

È possibile creare in modo esplicito un contratto dati utilizzando gli attributi `DataContract` e `DataMember`.

```
[DataContract]
public class PurchaseOrder
{
    private int poId_value;

    [DataMember]
    public int PurchaseOrderId
    {
        get { return poId_value; }
        set { poId_value = value; }
    }
}
```

# Tipologie istanze



Esistono 3 modi con cui si può istanziare il servizio che gestisce la richiesta in entrata:

- **Per-Call:** ad ogni chiamata viene creata una nuova istanza del servizio
- **Per-Session:** viene creata un'istanza del servizio per ogni client
- **Singleton:** viene creata una sola istanza del servizio che gestisce tutte le richieste

Il tutto configurabile via Behavior oppure tramite l'attributo `[ServiceBehaviour]`



# Per-Call



Windows  
Communication  
Foundation

- Ad ogni chiamata viene creata una nuova istanza
- Occupa meno risorse sul server
- Offre maggior scalabilità
- Si adatta perfettamente alle transazioni
- **Contro:** bisogna gestire lo stato a mano

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]  
class Service : IService  
{  
}
```

# Per-Session



- È la configurazione di default
- Viene creata un'istanza per ogni accoppiata proxy-endpoint
- Occupa risorse (a volte dispendiose) sul server
- Non offre scalabilità
- Ostico per quanto riguarda le transazioni
- La sessione termina quando il proxy viene chiuso o la comunicazione va in timeout

# Per-Session



- Attivabile solo con alcuni binding e loro configurazione
- **Pro:** Lo stato viene mantenuto automaticamente
- Va impostato il *SessionMode* per rafforzare il binding

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]  
class Service : IService  
{  
}
```

# SessionMode



- Quando si utilizza la modalita Per-Session, per collegare un proxy ad un'istanza si deve avere una sessione.
- Può essere a livello di trasporto (*NetTcpBinding*) o a livello applicativo (*WSHttpBinding* con sicurezza o *ReliableMessaging* abilitati)

# SessionMode



Il *SessionMode* specifica se la sessione deve essere presente o meno

- *Allowed* (Default): permette che ci sia una sessione
- *Required*: obbliga la presenza di una sessione
- *NotAllowed*: non permette la presenza di una

```
[ServiceContract(SessionMode = SessionMode.Allowed)]  
interface IService  
{  
}
```

# Singleton

- Viene creata una sola istanza che gestisce tutte le chiamate
- Occupa risorse (a volte dispendiose) sul server
- NON OFFRE ASSOLUTAMENTE SCALABILITA'
- Non parliamo poi di transazioni
- Non necessità di una sessione con il client
- **Pro:** Utile per centralizzare richieste ad una risorsa univoca. Ad esempio un log

```
[ServiceBehavior(InstanceContextMode = ServiceBehavior(InstanceContextMode.Single))]  
class Service : IService  
{  
}
```

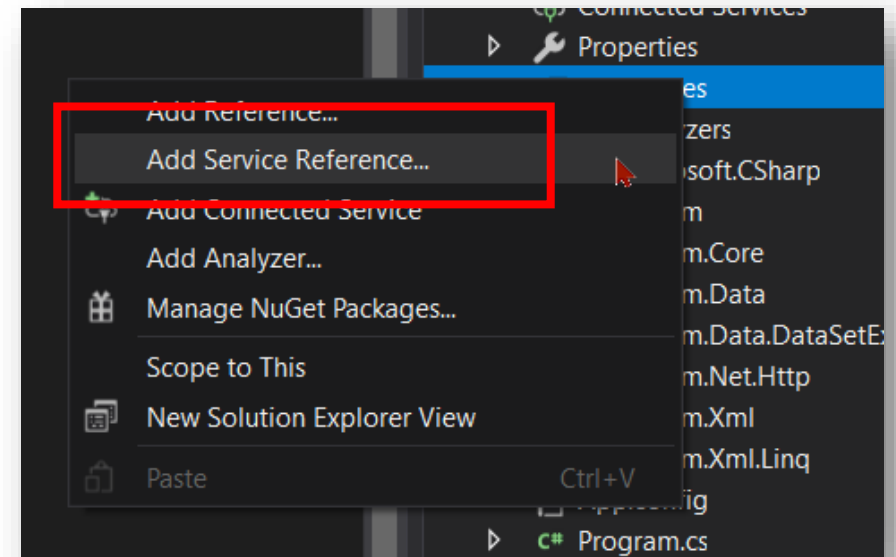
# WCF – Proxy Client



Per utilizzare un servizio WCF da una applicazione occorre definire una classe che funga da client (proxy class)

In generale esistono tool a linea di comando (CLI) che interpretano il WSDL

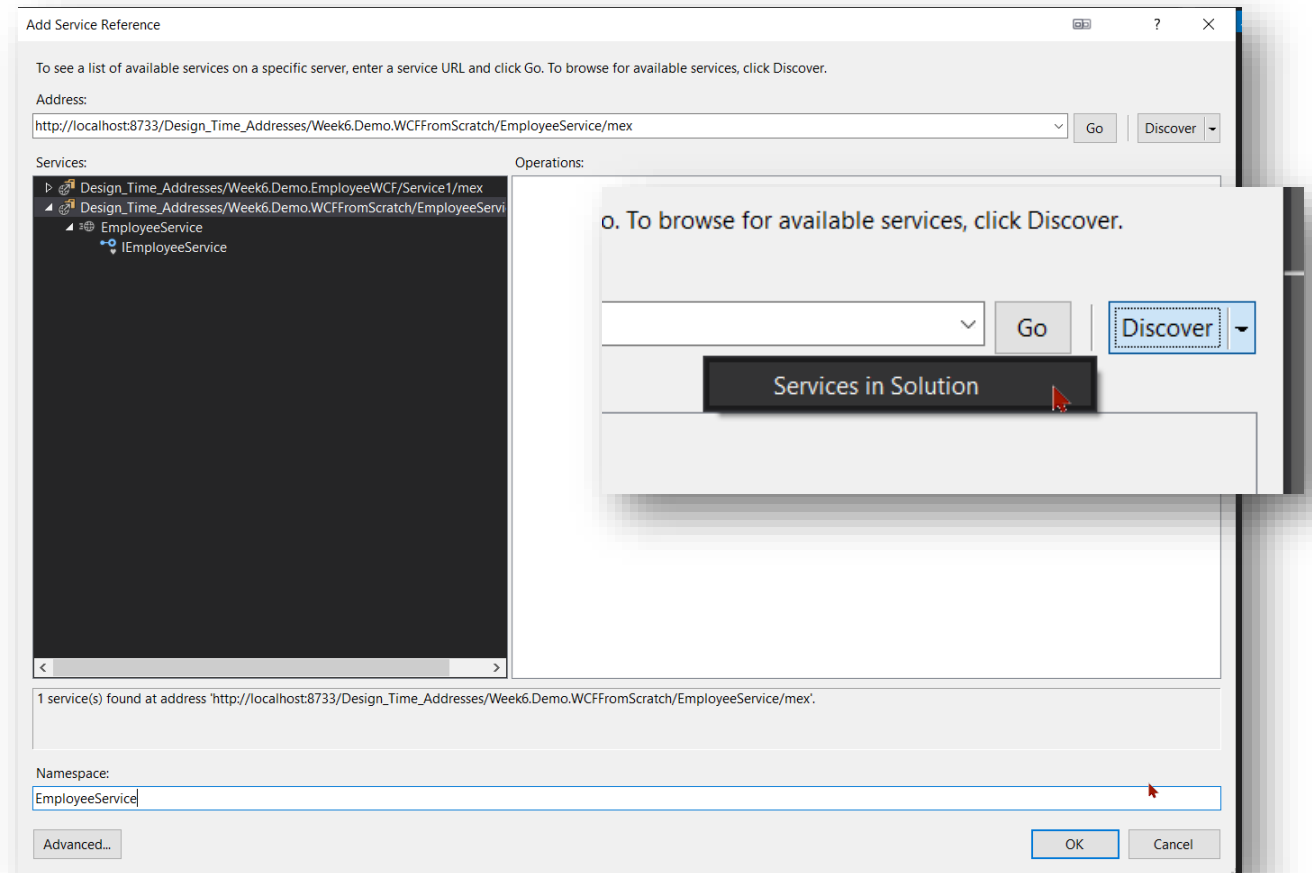
Per le applicazioni .NET è possibile (direttamente da VS) aggiungere una **Service Reference**.



# WCF – Proxy Client

Il tool permette di

- Esplorare la Solution alla ricerca dei servizi WCF
- Specificare l'URL (noto di un WCF)
- Generare un insieme di classi che costituiscono il Proxy Client
  - È possibile intervenire sul codice generato tramite le impostazioni avanzate

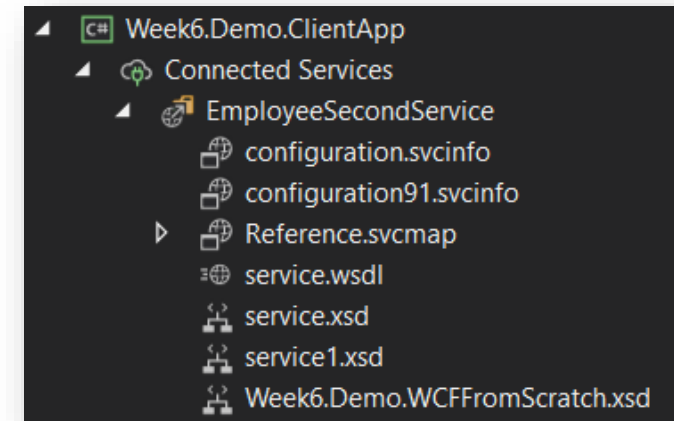




# WCF – Proxy Client



Il Proxy Client è visibile nel progetto sotto la voce Connected Services



```
EmployeeServiceClient emp = new EmployeeServiceClient();  
var employee = emp.GetEmployeeById(value);
```

Il Proxy Client è semplicemente una classe istanziabile nel codice della applicazione

# Esercitazione 1

Realizzare un servizio WCF per la gestione di una biblioteca

- Realizzare un modello dati che comprende una entità **Libro**
  - *ISBN* (string), *Titolo* (string), *Sommario* (string), *Autore* (string)
- Realizzare un layer dati con Entity Framework per la gestione delle operazioni di CRUD sull'entità **Libro** (creare un nuovo database *Biblioteca*)
  - Strutturare la soluzione in progetti in modo che sia possibile riusare il layer dati
- Realizzare un servizio WCF con un contratto che permetta di utilizzare le suddette operazioni di CRUD
- Verificare tramite il WCF Test Client il corretto funzionamento del servizio

WCF => .NET Framework 4.8  
EF => Entity Framework Core 3.1.15  
Class Libraries => .NET Standard 2.0



# WCF – Hosting



Per poter essere utilizzato, un servizio deve essere ospitato (*hosting*) all'interno di un ambiente di runtime che lo crea e ne controlla il contesto e la durata.

I servizi di WCF sono progettati per essere eseguiti in qualsiasi processo Windows che supporti il codice managed.

In particolare si possono sfruttare il **self-hosting** e l'**IIS hosting**.

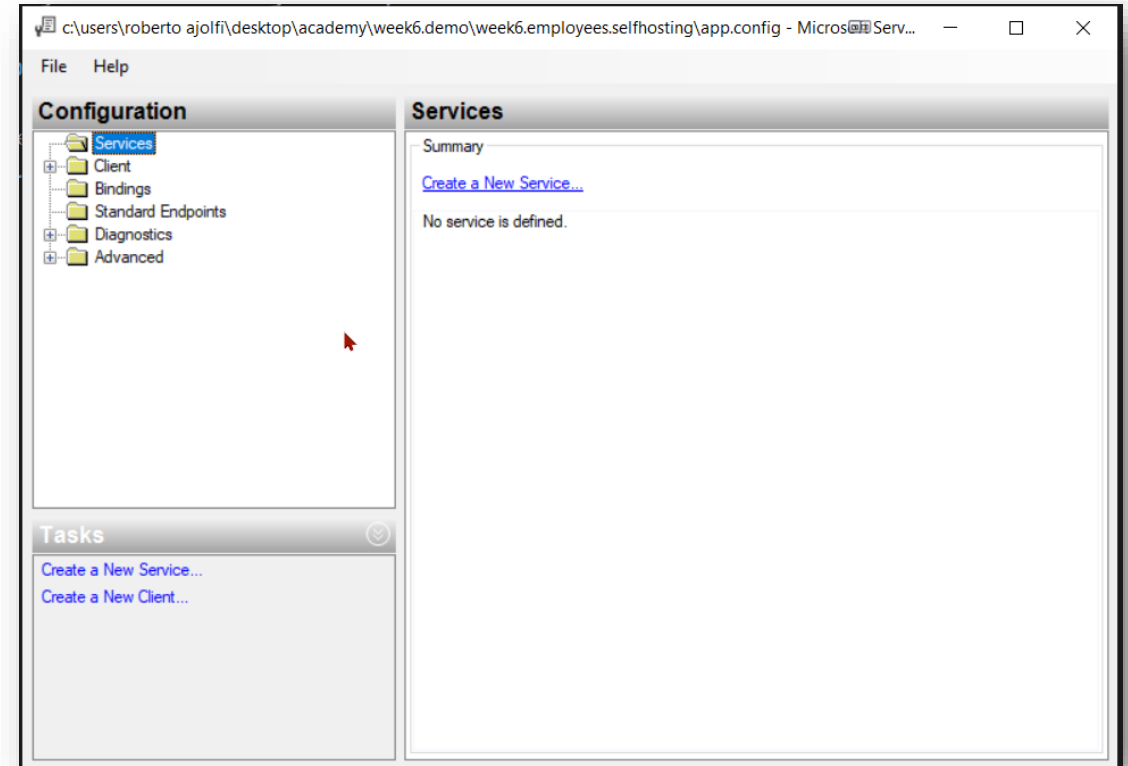
# WCF – Self Hosting



Hosting in un assembly .NET

- Console
- Windows Service (meglio in PRD)

Si può utilizzare il **WCF Service Configuration Editor** per la configurazione completa del servizio



# WCF – IIS Hosting



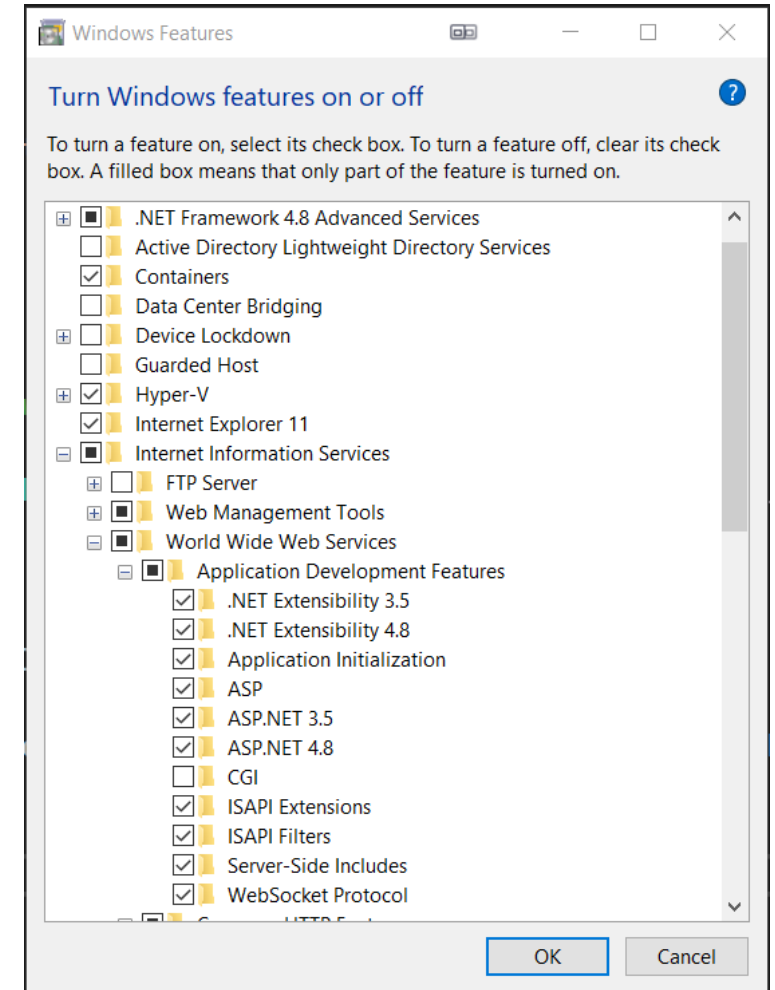
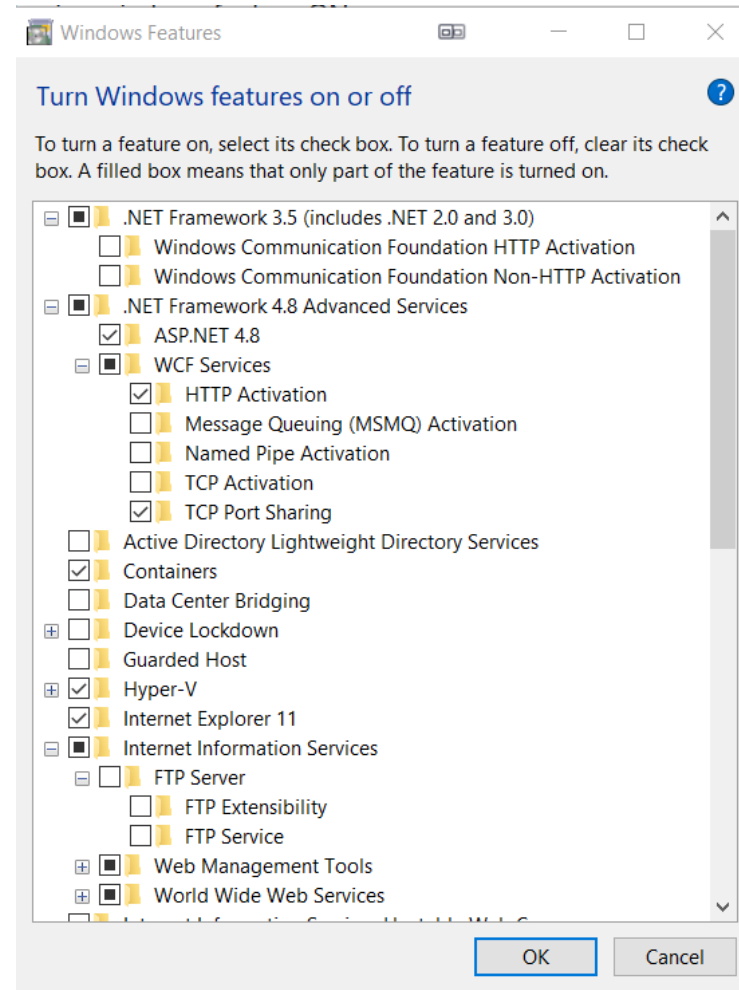
- Creare un progetto WCF Service
- In questo progetto referenziare la WCF Library in cui è stato implementato il servizio
- Modificare il file `Service.svc`

```
<%@ ServiceHost Language="C#" Debug="true" Service="Demo.Wcf.EmployeeService" %>
```

- Modificare l'`App.Config`
- Deploy su una installazione di IIS, all'interno di un Web Site

# WCF – IIS Hosting

- Installare IIS da Control Panel (vedi Windows Features)



# WCF – IIS Hosting



- Dare permessi Full Control a IUSR e IIS\_IUSR sulla cartella del progetto
- Creare sito in IIS che punta alla cartella del progetto

# Gestione errori



.NET utilizza le eccezioni per evidenziare il fatto che c'è stata un'eccezione. Per default queste causano la chiusura forzata del processo ma non è così per WCF.

Le eccezioni causano solo la chiusura del canale con il client a seconda del tipo di eccezione. L'unica cosa che il client può fare dopo un'eccezione è chiudere il proxy anche se l'eccezione è stata gestita.



# Gestione errori



Le eccezioni sono specifiche per la tecnologia .NET e quindi non interoperabili per natura.

Il problema si risolve con i Soap Faults che sono uno standard WS-\* e quindi interoperabili.

In .NET un Soap Fault viene gestito tramite le classi `FaultException` e `FaultException<T>`.

# FaultException<T>



`FaultException<T>` deriva da `FaultException` e serve esclusivamente a specializzarla in determinati casi.

Il parametro `T` deve contenere una classe che sia marcata con l'attributo `Serializable` o `DataContract` e non è affatto obbligatorio che questa derivi da `Exception`

`FaultException` viene invece utilizzata quando non si tipizza l'errore.

Quando il client riceve una `FaultException` il canale non viene chiuso e quindi si può continuare ad utilizzare il proxy.

# Fault Contracts



Si può stipulare un contratto secondo cui alcune eccezioni vengono inviate al client.

Una `ArgumentNullException` può essere gestita tra client e server, una `SqlException` non ha invece alcun senso.

Un contratto tra client e server si può stipulare tramite l'attributo **FaultContract**. Il client può così gestire le eccezioni in maniera diversa.

```
[ServiceContract]
interface IService
{
    [FaultContract(typeof(ArgumentNullException))]
    void Method(string parameter);
}
```

# Debugging eccezioni



In produzione, un client dovrebbe ricevere le minime informazioni relative ad un errore.

In fase di debugging e testing è invece comodo ricevere tutto il flusso di informazioni da servizio. Questo può essere ottenuto tramite un behavior.

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="Service" behaviorConfiguration="DebugEnabled" >
        <!-- ... -->
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="DebugEnabled">
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

# Esercitazione 2

Estendere il servizio WCF di Esercitazione 1

- Aggiungere al modello l'entità **Prestito**
  - *Id* (int), *IdLibro* (int?, string?), *Utente*(stringa), *DataPrestito* (date), *DataReso* (date, nullable)
- Realizzare un layer dati con Entity Framework per la gestione delle operazioni di CRUD sull'entità **Prestito**
- Modificare il servizio WCF con un contratto che permetta di
  - Gestire una operazione di *Prestito Libro*
  - Gestire una operazione di *Resa Libro*
- Utilizzare la modalità di self-hosting per l'esecuzione del servizio WCF
- Gestire un caso di errore tramite SOAP Faults



# Sicurezza



WCF è stato costruito tenendo la sicurezza al primo posto. É per questo che si sente spesso dire che WCF è “**Secure-by-default**”.

Come ogni applicazione, anche un servizio WCF conosce due fasi di sicurezza

- Autenticazione
- Autorizzazione

In più supporta a costo quasi zero le feature “classiche” che sono integrità, confidenzialità, logging e auditing.

# Integrità e confidenzialità



L'integrità e la confidenzialità dei messaggi scambiati può essere garantita in due modi:

- Trasporto: si sfrutta la protezione del canale (HTTPS, SSL over TCP)
- Messaggio: i dati vengono criptati con un certificato digitale

# Autenticazione



Lato server, l'autenticazione del client può essere fatta in diversi modi:

- **None:** non c'è autenticazione ed il servizio è liberamente utilizzabile da chiunque
- **Windows Authentication:** il client invia i le credenziali al server sotto forma di account Windows
- **Username e Password:** l'utente invia username e password che sono sfruttate per un'autenticazione custom
- **Certificato X509:** il client si identifica con un certificato digitale
- **Token:** Questo meccanismo sfrutta la Federation Security o Windows CardSpace.



# Autorizzazione



I meccanismi di autorizzazione non cambiano da quelli che sono già presenti nel Framework.

Si può utilizzare sia la Declarative Security, piazzando attributi sui metodi che richiedono particolari autorizzazioni, che la Imperative Security, sfruttando il metodo **IsInRole** dell'interfaccia **IPrincipal**.

# Security by Scenario



Esistono cinque principali scenari di sicurezza quando si sviluppa un servizio

- Intranet
- Internet
- B2B
- Client anonimi
- Nessuna autenticazione

# Scenario Intranet



- I Binding da utilizzare sono
  - *NetTcpBinding*
  - *NetNamedPipeBinding*
  - *NetMsmqBinding*
- L'integrità e la confidenzialità sono gestite a livello di trasporto
- L'autenticazione viene tipicamente gestita sfruttando gli account di dominio Windows

# Scenario Internet



- I Binding da utilizzare sono
  - *WsHttpBinding*
  - *WsDualHttpBinding*
- L'integrità e la confidenzialità sono gestite a livello di trasporto e di messaggio
- L'autenticazione viene tipicamente gestita sfruttando un database (ASP.NET Provider, Windows Accounts, Custom)

# Scenario B2B



- I Binding da utilizzare sono
  - *BasicHttpBinding*
  - *WSHttpBinding*
  - *WSDualHttpBinding*
- L'integrità e la confidenzialità sono gestite a livello di trasporto tramite certificato
- L'autenticazione viene tipicamente gestita attraverso la lettura del certificato digitale

# Scenario Client Anonimi



- Si può utilizzare qualunque binding
- L'integrità e la confidenzialità sono gestite a livello di trasporto tramite certificato
- **Non c'è autenticazione**

# Scenario Nessuna Sicurezza



Windows  
Communication  
Foundation

- Si può utilizzare qualunque binding
- Nessuna integrità e confidenzialità
- **Non c'è autenticazione**

# Entity Framework

ORM di Microsoft basato sul .NET Framework

Insieme di tecnologie ADO.NET per lo sviluppo software

Definisce un modello di astrazione dei dati

Traduce il nostro codice in query comprensibili dal DBMS

Disaccoppiamento tra applicazione e dati

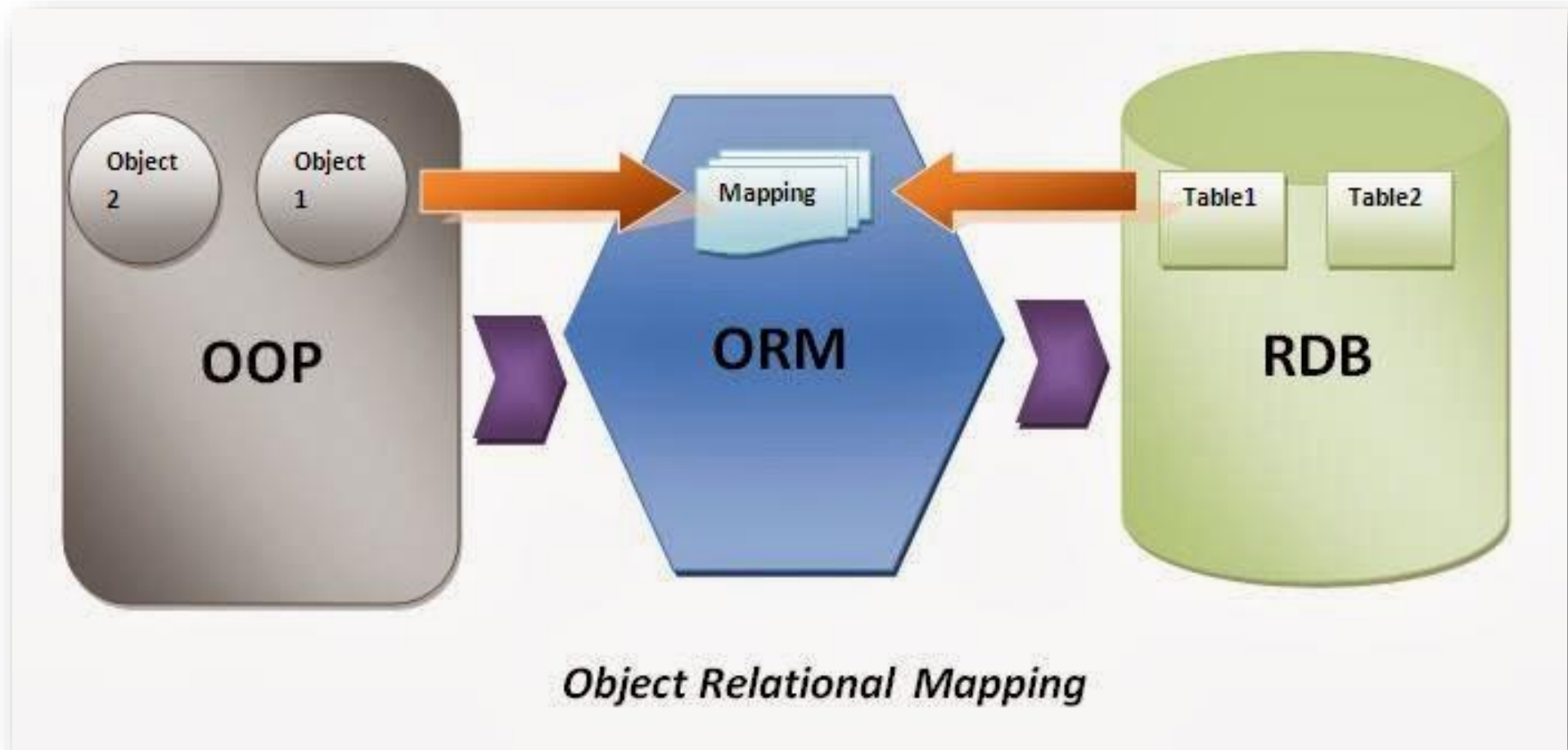
- Posso mantenere la stessa rappresentazione anche se cambia il modello fisico (es. da SQL Server ad Oracle)

Open source

- <https://github.com/aspnet/EntityFramework>



# Cos'è un ORM?



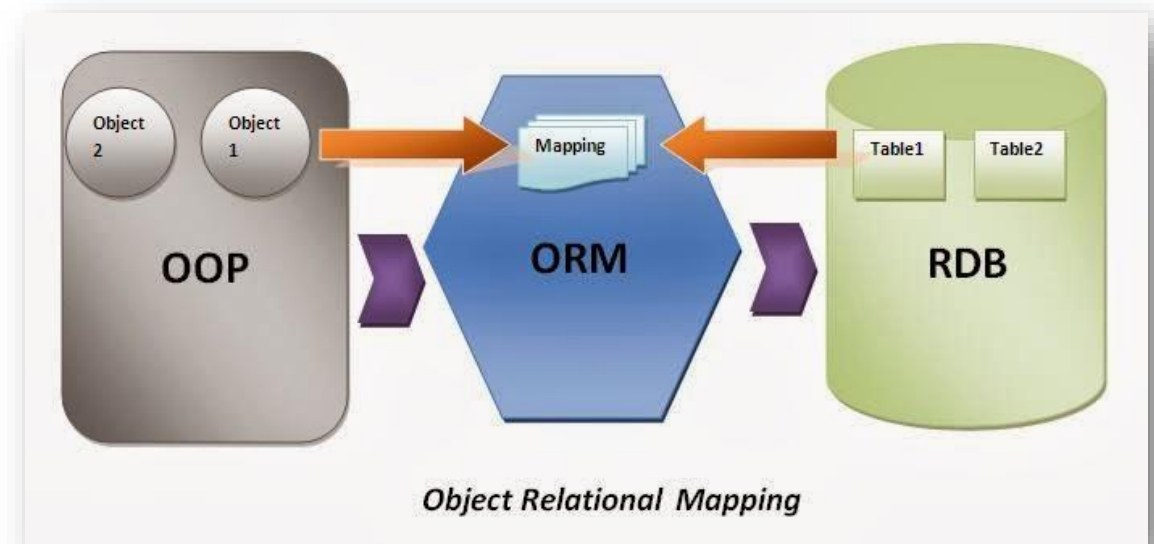
# Cos'è un ORM?

È una tecnica per convertire dati da type system incompatibili

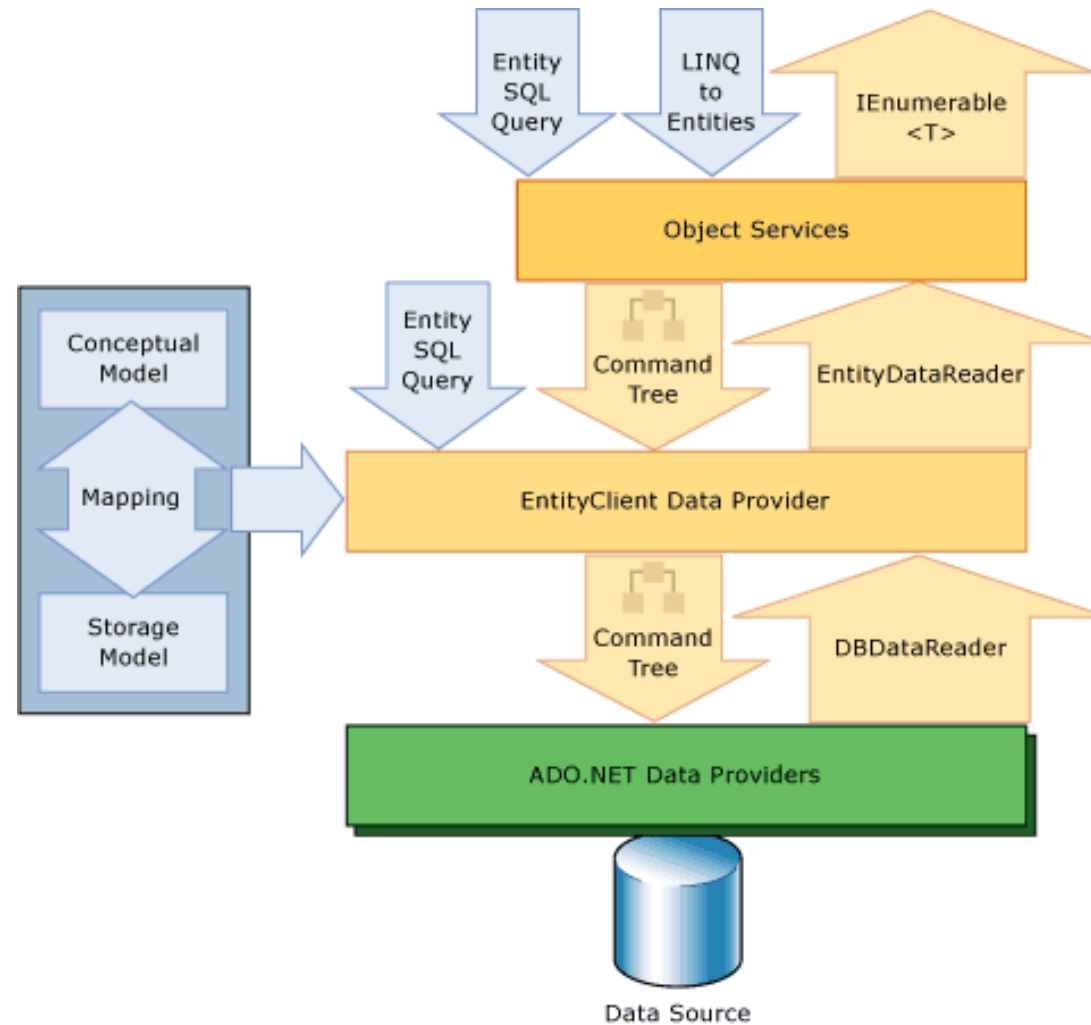
Da database ad object-oriented

## 3 caratteristiche fondamentali

- **Mapping**
  - Definisce come il database si «incastra» negli oggetti e viceversa
- **Fetching**
  - Sa come recuperare i dati dal database e materializzare i rispettivi oggetti
- **Persistenza del grafo**
  - Sa come salvare le modifiche agli oggetti, generando le query SQL corrispondenti



# Come funziona



# Diversi approcci

## Database-First

- Il modello viene importato da un DB esistente
- Se modifico il database posso (quasi) sempre aggiornare il modello

## Model-First

- Il modello del database viene creato dal designer di Visual Studio
- L'implementazione fisica è basata sul modello generato
- Non favorisce il riutilizzo del codice né la separazione tra contesto ed entità
- Poichè il modello definisce il DB, eventuali sue modifiche verranno perse

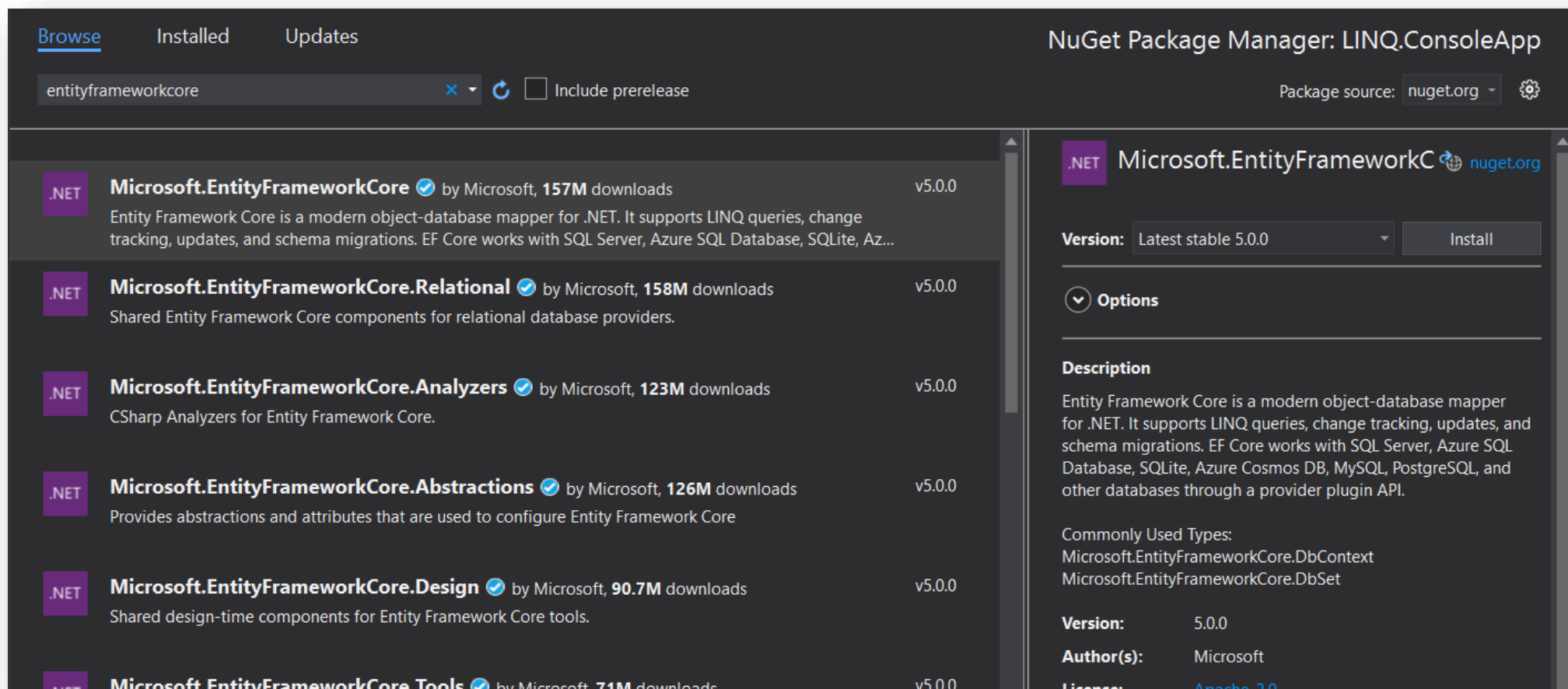
## Code-First

- Il modello viene creato dal nostro codice
- L'implementazione fisica è basata sul nostro codice

# Perché Code-First?

- Focus sul domain design
- C# potrebbe risultarci più familiare delle query di SQL
  - E sarebbe l'unico linguaggio da apprendere
- Possiamo mettere facilmente sotto source control il nostro database (niente script SQL solo codice C#)
- Evitiamo la mole di codice auto generato da EDMX
- Se scegliamo di sviluppare in .NET Core, l'EDMX non è supportato

# Configurazione di EF



The screenshot displays the NuGet Package Manager interface for a project named 'LINQ.ConsoleApp'. The search bar contains 'entityframeworkcore'. The results list several packages by Microsoft, all at version 5.0.0. The details pane on the right shows the 'Microsoft.EntityFrameworkCore' package, with the 'Latest stable 5.0.0' version selected and an 'Install' button. The description states it is a modern object-database mapper for .NET. Commonly used types listed are 'Microsoft.EntityFrameworkCore.DbContext' and 'Microsoft.EntityFrameworkCore.DbSet'. The license is 'Apache 2.0'.

| Package Name                               | Author    | Downloads | Version |
|--|-----------|-----------|---------|
| Microsoft.EntityFrameworkCore              | Microsoft | 157M      | v5.0.0  |
| Microsoft.EntityFrameworkCore.Relational   | Microsoft | 158M      | v5.0.0  |
| Microsoft.EntityFrameworkCore.Analyzers    | Microsoft | 123M      | v5.0.0  |
| Microsoft.EntityFrameworkCore.Abstractions | Microsoft | 126M      | v5.0.0  |
| Microsoft.EntityFrameworkCore.Design       | Microsoft | 90.7M     | v5.0.0  |
| Microsoft.EntityFrameworkCore.Tools        | Microsoft | 74M       | v5.0.0  |

**Microsoft.EntityFrameworkCore** by Microsoft, 157M downloads  
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Az...

**Microsoft.EntityFrameworkCore.Relational** by Microsoft, 158M downloads  
Shared Entity Framework Core components for relational database providers.

**Microsoft.EntityFrameworkCore.Analyzers** by Microsoft, 123M downloads  
CSharp Analyzers for Entity Framework Core.

**Microsoft.EntityFrameworkCore.Abstractions** by Microsoft, 126M downloads  
Provides abstractions and attributes that are used to configure Entity Framework Core

**Microsoft.EntityFrameworkCore.Design** by Microsoft, 90.7M downloads  
Shared design-time components for Entity Framework Core tools.

**Microsoft.EntityFrameworkCore.Tools** by Microsoft, 74M downloads

**Microsoft.EntityFrameworkCore** by Microsoft, 157M downloads

**Version:** Latest stable 5.0.0 **Install**

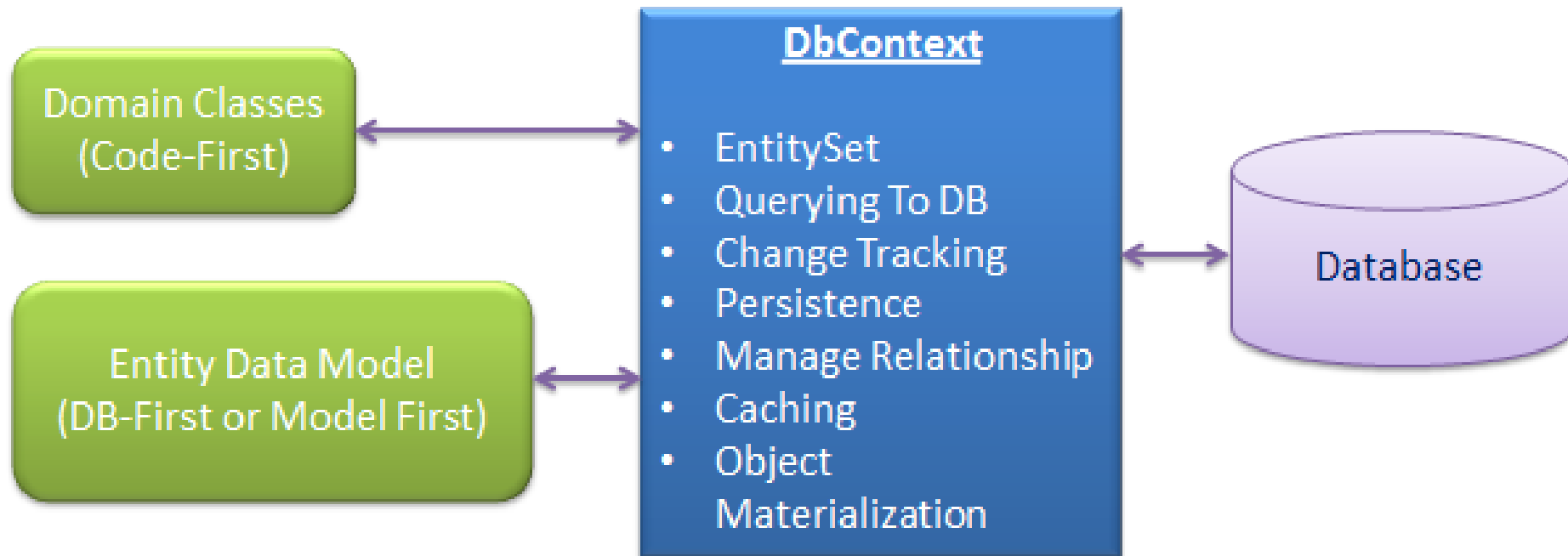
**Options**

**Description**  
Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.

**Commonly Used Types:**  
Microsoft.EntityFrameworkCore.DbContext  
Microsoft.EntityFrameworkCore.DbSet

**Version:** 5.0.0  
**Author(s):** Microsoft  
**License:** Apache 2.0

# II DbContext



## II DbContext

```
public class Context : DbContext
{
    public DbSet<Student> Students { get; set; }

    public Context() : base() { }

    public Context() : base("MyContext") { }
}
```



## II DbContext

```
public class Context : DbContext
{
    public Context(
        DbContextOptions<TicketingContext> options
    ) : base(options) { }
}
```

```
{
    // ...
    "ConnectionStrings": {
        "TicketingDb": "Server=tcp:democrito.database.windows.net,1433;Initial Catalog=Ticketing;
            Persist Security Info=False;User ID=sa;Password=xxxxxxx;MultipleActiveResultSets=False;
            Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"
    }
}
```

# Il DbSet

E' una classe che rappresenta le entity

Serve per fare operazioni CRUD

E' definito come **DbSet<TEntity>**

I metodi più utilizzati sono:

- **Add, Remove, Find, SqlQuery**

```
public DbSet<Student> Students { get; set; }
```

# Type Discovery

Nel *DbContext*:

```
public DbSet<Student> Students { get; set; }
```

Definizione della classe *Student*:

```
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime DateOfBirth { get; set; }

    public Teacher Teacher { get; set; }
}
```

# Primary Key

```
public class Student
{
    public int StudentID { get; set; }

    public string StudentName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public Teacher Teacher { get; set; }
}
```

Convenzione sul nome della chiave primaria:

- *Id*
- *<NomeClasse>ID*

```
public class Student
{
    public int MyPrimaryKey { get; set; }

    public string StudentName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public Teacher Teacher { get; set; }
}
```

Non usa la convenzione di code-first  
Genera una *ModelValidation Exception*  
se non gestita con le *DataAnnotations*

# Navigation Properties

Cos'è la proprietà Teacher?

```
public class Student
{
    // ...
    public Teacher Teacher { get; set; }
}
```

È una **Navigation Property**.

È la rappresentazione in EF di una Relazione tra due entità.

# Foreign Key

```
public class Student
{
    public int StudentID { get; set; }

    public string StudentName { get; set; }
    public DateTime DateOfBirth { get; set; }

    public int CourseId { get; set; }
    public Course Course { get; set; }
}
```

```
public class Course
{
    public int CourseId { get; set; }

    public string CourseName { get; set; }
    public Teacher Teacher { get; set; }
    public ICollection<Student> Students { get; set; }
}
```

La *ForeignKey* viene generata automaticamente da code-first ogni volta che viene individuata una navigation property

Sempre bene rispettare le stesse convenzioni della *PrimaryKey*

# DataAnnotations e Fluent API

Le DataAnnotations sono attributi che servono a specificare il comportamento per fare l'override delle convenzioni di code-first

Possono influenzare le singole proprietà

- Namespace *System.ComponentModel.DataAnnotations*
- *Key, Required, MaxLength...*

Possono influenzare lo schema del database

- Namespace *System.ComponentModel.DataAnnotations.Schema*
- *Table, Column, NotMapped...*

Le DataAnnotations sono limitate. Per il set completo bisogna andare di Fluent API

# DataAnnotations: Key

Override della convenzione sulla *PrimaryKey*

Viene applicato alle proprietà di una classe

```
[Key]  
public int MyPrimaryKey { get; set; }
```



# DataAnnotations: Required

Indica al database che quella colonna non può essere NULL  
In ASP.NET MVC viene usato anche per la validazione

```
[Required]  
public string Name { get; set; }
```

# DataAnnotations: MaxLenght, MinLenght

Possono essere applicati a stringhe o array

Possono essere usati in coppia

*EntityValidationError* se non rispettati durante una update

```
[MaxLenght(50), MinLenght(8)]  
public string Name { get; set; }
```

# DataAnnotations: Table

Rappresenta l'override del nome della tabella

Può essere solo applicato ad una classe e non alle proprietà

Si può anche inserire uno schema differente

```
[Table("Studente", Schema = "MySchema")]  
public class Student { /* ... */ }
```

# DataAnnotations: Column

Rappresenta l'override del nome della proprietà

Può essere applicato solo ad una proprietà

Si può usare in combinata con *Order* e *TypeName*

```
[Column("Nome", Order = 5, TypeName = "varchar")]  
public string Name { get; set; }
```

# DataAnnotations: ForeignKey

Rappresenta l'override della convenzione sulla chiave esterna

Viene applicato solo alle proprietà di una classe

```
public class Student
{
    public int StudentId { get; set; }
    public int CourseId { get; set; }

    [ForeignKey("CourseId")]
    public Course Course { get; set; }
}

public class Course
{
    public int CourseId { get; set; }
    public ICollection<Student> Students { get; set; }
}
```

# DataAnnotations: NotMapped

Ignora il mapping per proprietà che hanno getter e setter impostati  
Viene usato per non creare colonne nel database

```
[NotMapped]  
public string Name { get; set; }
```

# Fluent API

Sono una alternativa completa alle DataAnnotations  
Si definiscono dentro l'override di *OnModelCreating*

Tre tipologie di mapping supportate:

- **Model:** Schema e convenzioni
- **Entity:** Ereditarietà
- **Property:** chiavi primarie/esterne, colonne e altri attributi

# Model e Entity Mapping

Configurazione dello schema per tutto il database

Configurazione dello schema per singola tabella

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.HasDefaultSchema("Admin");

    //Map entity to table
    modelBuilder.Entity<Student>().ToTable("StudentInfo");
    modelBuilder.Entity<Student>().ToTable("StandardInfo", "anotherSchema");
}
```



# Property Mapping

Configurazione della chiave primaria

```
modelBuilder.Entity<Student>().HasKey<int>(s => s.StudentId);
```

Configurazione di altre proprietà

```
modelBuilder.Entity<Student>().Property(p => p.Age)
    .HasColumnName("Eta")
    .HasColumnOrder(3)
    .HasColumnType("datetime")
    .IsRequired();
```

# Fluent API Configurations

Tutte le configurazioni sono fatte via Fluent API

- Problema: troppo codice dentro *OnModelCreating*, diventa ingestibile!
- Soluzione: organizziamo le configurazioni

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.ApplyConfigurations<Student>(new StudentConfiguration());
    modelBuilder.ApplyConfigurations<Course>(new CourseConfiguration());
}
```

# Fluent API Configurations

```
public class StudentConfiguration : IEntityTypeConfiguration<Student>
{
    public void Configure(EntityTypeBuilder<Student> builder)
    {
        builder.ToTable("StudentInfo");

        builder.HasKey<int>(s => s.StudentId);

        builder.Property(p => p.Age)
            .HasColumnName("Eta")
            .HasColumnOrder(3)
            .HasColumnType("datetime")
            .IsRequired();
    }
}
```

# Aggiungere dati

Utile sapere:

- Il pattern *IDisposable*
- *Async/await*

```
using (var ctx = new Context())  
{  
    var person = new Person("Matteo", "Tumiati");  
    ctx.People.Add(person);  
    await ctx.SaveChangesAsync();  
}
```

# Fare query sui dati

Utile conoscere:

- Il pattern *IDisposable*
- *Linq*
- I dati che si vogliono ottenere ☺

```
using (var ctx = new Context())  
{  
    ctx.Students.Where(x => x.Name.StartsWith("A"))  
                .OrderBy(x => x.Age)  
                .ToList();  
}
```

# CUD (Create / Update / Delete)

## Inserimento

```
using(var ctx = new MyContext())
{
    var prd = new Product()
    {
        ProductCode = "PR0001"
    };

    ctx.Products.Add(prd);

    ctx.SaveChanges();
}
```

# CUD (Create / Update / Delete)

## Update / Delete

```
using(var ctx = new MyContext())
{
    var prd = ctx.Products.FirstOrDefault<Product>();

    // UPDATE
    prd?.ProductCode = "PR0002";
    ctx.SaveChanges();

    // DELETE
    if(prd != null)
        ctx.Products.Remove(prd);
    ctx.SaveChanges();
}
```

# Migrations

Sono il meccanismo che consente, utilizzando l'approccio code-first, l'aggiornamento del database a fronte di modifiche al modello.

Esistono 2 tipi di migrazioni

- *Automatiche*: poco invasive
- *Manuali o code-based*: richiedono un intervento specifico sul database



# Migrazioni code-based

Sono utili quando vogliamo più controllo sulle modifiche automatiche. Servono due comandi dalla Package Manager Console:

- **Add-Migrations «Migration name»**
  - Crea una nuova classe con tutte le modifiche rispetto allo stato precedente del db
- **Update-Database**
  - Aggiorna il database sulla base del modello

Si può anche fare rollback di una modifica:

- **Update-Database -Migration:"Migration name"**

# Migrazioni code-based

Per ogni Migration, viene creato un nuovo file per ogni migrazione

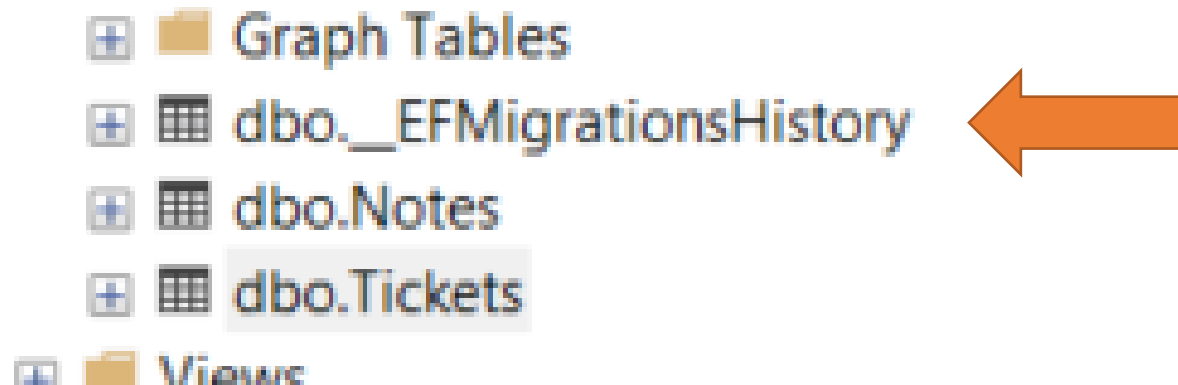
- `TimeStamp + NomeMigrazione.cs`
- Il file contiene una classe che eredita da **Migration**
- Contiene due metodi **Up** e **Down** per l'aggiornamento del database

```
public partial class FirstMigration : DbMigration
{
    public override void Up()
    {
        CreateTable("dbo.Students", c => new {
            StudentId = c.Int(nullable: false, identity: true),
            Name = c.String(),
            Age = c.Int(nullable: false)})
            .PrimaryKey(t => t.StudentId);
    }

    public override void Down()
    {
        DropTable("dbo.Students");
    }
}
```

# Migrazioni

Se andiamo a vedere il nostro database...



Viene aggiunta una tabella al nostro database per mantenere lo storico delle Migration applicate.

# JSON

In informatica, nell'ambito della programmazione web

**JSON** (acronimo di *JavaScript Object Notation*)

è un formato adatto all'interscambio di dati fra applicazioni client/server.

È basato sul linguaggio JavaScript Standard ECMA-262 3<sup>a</sup> edizione (dicembre 1999), ma ne è indipendente.

```
{  
  "name": "Mario",  
  "surname": "Rossi",  
  "active": true,  
  "favoriteNumber": 42,  
  "birthday": {  
    "day": 1,  
    "month": 1,  
    "year": 2000  
  },  
  "languages": [ "it", "en" ]  
}
```

# JSON

La semplicità di JSON ne ha decretato un rapido utilizzo specialmente nella programmazione in AJAX.

Il suo uso tramite JavaScript è particolarmente semplice, infatti l'interprete è in grado di eseguirne il parsing tramite la funzione `JSON.parse()`.

Questo lo ha reso velocemente molto popolare a causa della diffusione della programmazione in JavaScript nel mondo del Web.

```
{  
  "name": "Mario",  
  "surname": "Rossi",  
  "active": true,  
  "favoriteNumber": 42,  
  "birthday": {  
    "day": 1,  
    "month": 1,  
    "year": 2000  
  },  
  "languages": [ "it", "en" ]  
}
```

# JSON

I tipi di dati supportati da questo formato sono:

- booleani (true e false);
- interi, numero in virgola mobile;
- stringhe racchiuse da doppi apici ("");
- array (sequenze ordinate di valori, separati da virgole e racchiusi in parentesi quadre []);
- array associativi (sequenze coppie chiave-valore separate da virgole racchiuse in parentesi graffe);

```
{  
  "name": "Mario",  
  "surname": "Rossi",  
  "active": true,  
  "favoriteNumber": 42,  
  "birthday": {  
    "day": 1,  
    "month": 1,  
    "year": 2000  
  },  
  "languages": [ "it", "en" ]  
}
```

Un messaggio HTTP che contenga nel body dati in formato JSON dovrebbe avere un header

**Content-Type: application/json**

# Cos'è REST: la definizione



**{ REST:API }**

- E' l'acronimo di "Representational State Transfer", ed è definito come "stile architetturale software" che definisce una serie di vincoli per creare servizi web.
- Fornisce interoperabilità tra sistemi complessi di computer su Internet.
- Un "RESTful web service" permette di richiedere, accedere e manipolare risorse remote semplicemente usando rappresentazioni "testuali" delle stesse.
- Per la manipolazione si utilizzano indirizzi "uniformi" e un elenco predefinito di operazioni "stateless"

# Accesso ai dati (REST Web Service)

- Per accedere ai dati tramite un servizio REST bastano
- Un URL

`https://servername/api/resourcenam`

- L'utilizzo degli HTTP Verbs

| HTTP Verbs | CRUD Operations            |
|------------|----------------------------|
| GET        | READ (one or many records) |
| POST       | CREATE                     |
| PUT        | UPDATE                     |
| DELETE     | DELETE                     |



# Accesso ai dati (REST Web Service)

- Esempi di chiamate

| HTTP Verbs                     | URL   | Body             | Returns (*)             |
|--------------------------------|---|------------------|-------------------------|
| GET<br>(list of resources)     | <code>https://servername/api/resourcename</code>    | -                | JSON array of resources |
| GET<br>(single resource by ID) | <code>https://servername/api/resourcename/ID</code> | -                | Single resource as JSON |
| POST                           | <code>https://servername/api/resourcename</code>    | Resource as JSON | -                       |
| PUT                            | <code>https://servername/api/resourcename/ID</code> | Resource as JSON | -                       |
| DELETE                         | <code>https://servername/api/resourcename/ID</code> | -                | -                       |

(\*) errors in case of issues

# Accesso ai dati (REST Web Service)

- L'autenticazione viene gestita in genere tramite l'aggiunta di un Authentication Header alla chiamata HTTP
  - BASIC Authentication
    - `Authorization: Basic YWxhZGRpbjpvGVuc2VzYW11`
    - `aladdin:$apr1$ZjTqBB3f$IF9gdYAGlMrs2fuINjHsz.`  
`user2:$apr1$004r.y2H$/vEkesPhVinBByJUKXitA/`
  - JWT (Token based)
    - `Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 ....`
  - ...
- **HTTPS ! Always !!!**

# Pragmatic REST oppure REST “Strict”

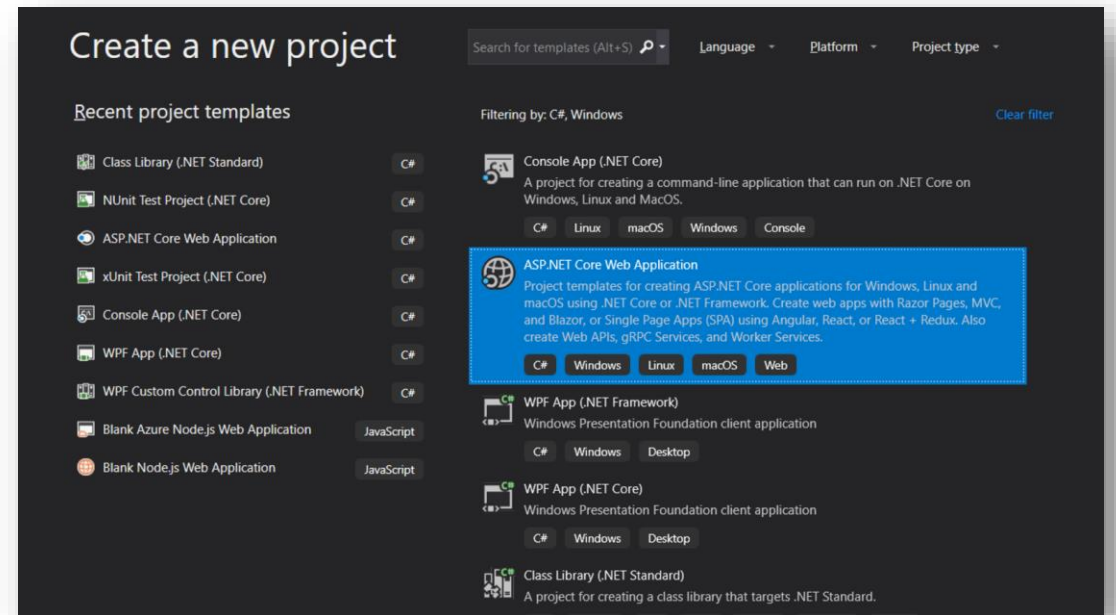
Come per ogni cosa nella vita, ci sono almeno due modi per fare le cose...e non è detto che uno dei due deve essere per forza quello sbagliato:

- REST “Strict” è un modo di sviluppare servizi REST (quindi API) seguendo i dettami di REpresentational State Transfer, che definisce che il modello dati deve essere sempre accessibile e gestibile dall'esterno
- Pragmatic REST è un approccio allo sviluppo di servizi REST che tende a centralizzare la logica di business sul servizio stesso invece che distribuirla sui client







# Cos'è ASP.NET Core?

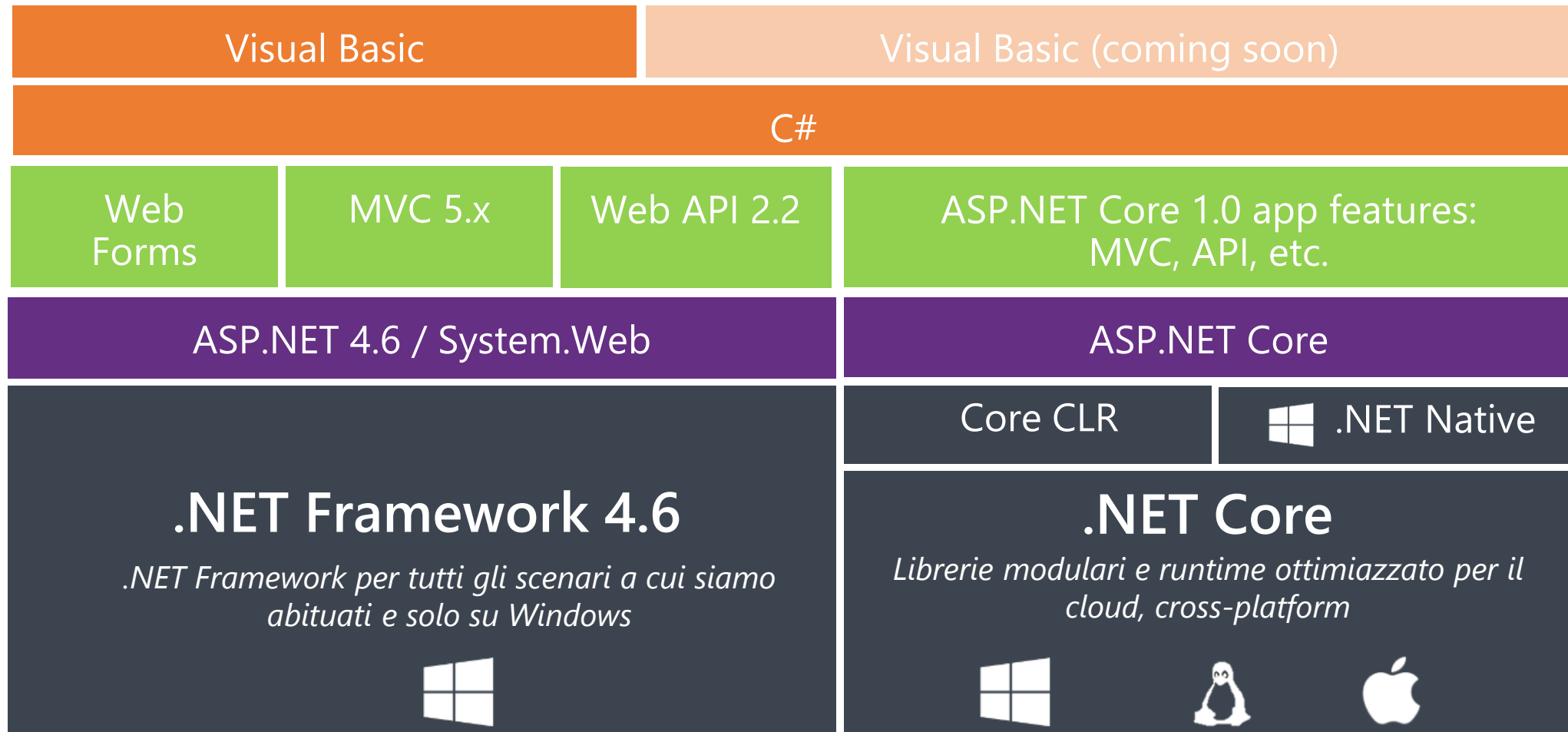
- Una rivoluzionaria versione di ASP.NET
- Basato su .NET Core
- Un taglio netto rispetto al passato
- Completamente riscritta
- Cross-platform
- Born in the cloud
- Performante



# ASP.NET 4.x e ASP.NET Core

| ASP.NET 4.x  | ASP.NET Core  |
|--|---|
| .NET Framework 4.x  | .NET Core    |
| .NET framework libraries   | .NET core libraries   |
| Compilatori e runtime<br>(.NET Compiler Platform: Roslyn, C#, VB, F#, RyuJIT, SIMD)                    |   |

# ASP.NET 4.x e ASP.NET Core



# ASP.NET Core 101

ASP.NET Core = runtime + ASP.NET MVC + ASP.NET Web API

Usando ASP.NET Core finiamo inevitabilmente ad usare ASP.NET MVC

Addio *web.config*

- Configurazione basata su codice/JSON

Modulare e componentizzabile

Gira su IIS o Self-hosted

- Kestrel

Dependency Injection già integrata

L'app web è di fatto una app console

- Vedi *Startup.cs* dentro il progetto

# Dov'è il web.config?

Esiste solo per compatibilità con IIS

Sostituito da un file chiamato Startup.cs

I parametri di configurazione sono in file JSON

- Con supporto degli environment

Dichiarazione in codice vs parsing a runtime di un file XML



# Startup.cs

Classe che definisce l'entry point

Può stare in un assembly esterno, basta specificare la chiave *Hosting:Application*

Può a propria volta accettare DI nel costruttore

Gestisce la *Configuration* attraverso un *ConfigurationBuilder*

Accende le opzioni di ASP.NET Core

- Es: file statici, MVC, etc

# Semplificazione del progetto con .NET Core 2

Aggiunta di un nuovo metapackage che include tutti i pacchetti di uso comune

- Microsoft.AspNetCore.All  
<https://www.nuget.org/packages/Microsoft.AspNetCore.All>
- Microsoft.AspNetCore.App  
<https://www.nuget.org/packages/Microsoft.AspNetCore.App> (2.1)
- 

Resta possibile referenziare i singoli pacchetti a mano

Runtime Store

- Contiene tutti gli asset di runtime, precompilati, che non vengono scaricati da NuGet – usa il nuovo metapackage

WebHostBuild di default

Tutto già preconfigurato per gli scenari più diffusi

# ASP.NET Core Web API

- Un modello per rappresentare servizi RESTful
- Ricorda da vicino l'architettura di ASP.NET Core MVC
- Ogni controller è un servizio
- Dentro un controller ci sono delle action, con un meccanismo di mapping su base REST
- Vengono usati i verb HTTP
  - GET, POST, PUT, DELETE
- La risposta è frutto di *content negotiation*
  - XML, JSON, OData

# Sviluppo di servizi con ASP.NET Core

ASP.NET Core utilizza *Startup.cs* per la configurazione

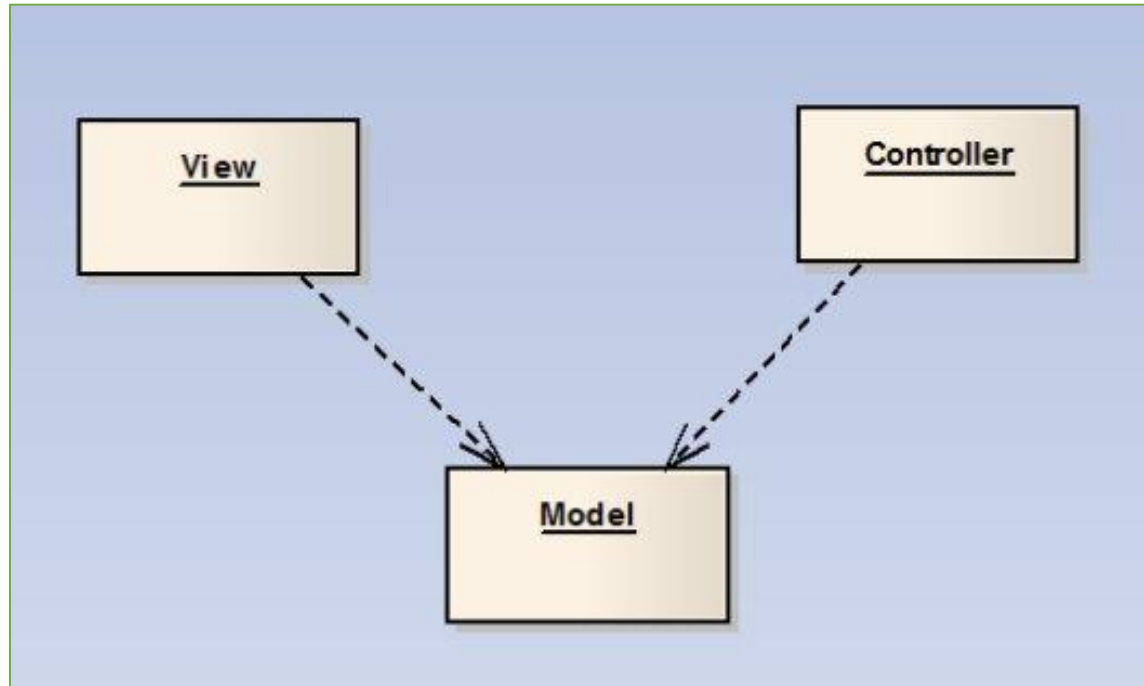
I controller WebApi diventano normali controller

- Non sono più presenti *ApiController*, *System.Web.Http* e  *IHttpActionResult*

Utilizzo di *UseMvc()* per la configurazione del routing basato su attributi/rotte

```
[Route("api/[controller]")]  
  
// GET api/values/5  
[HttpGet("{id}")]
```

# MVC in pillole

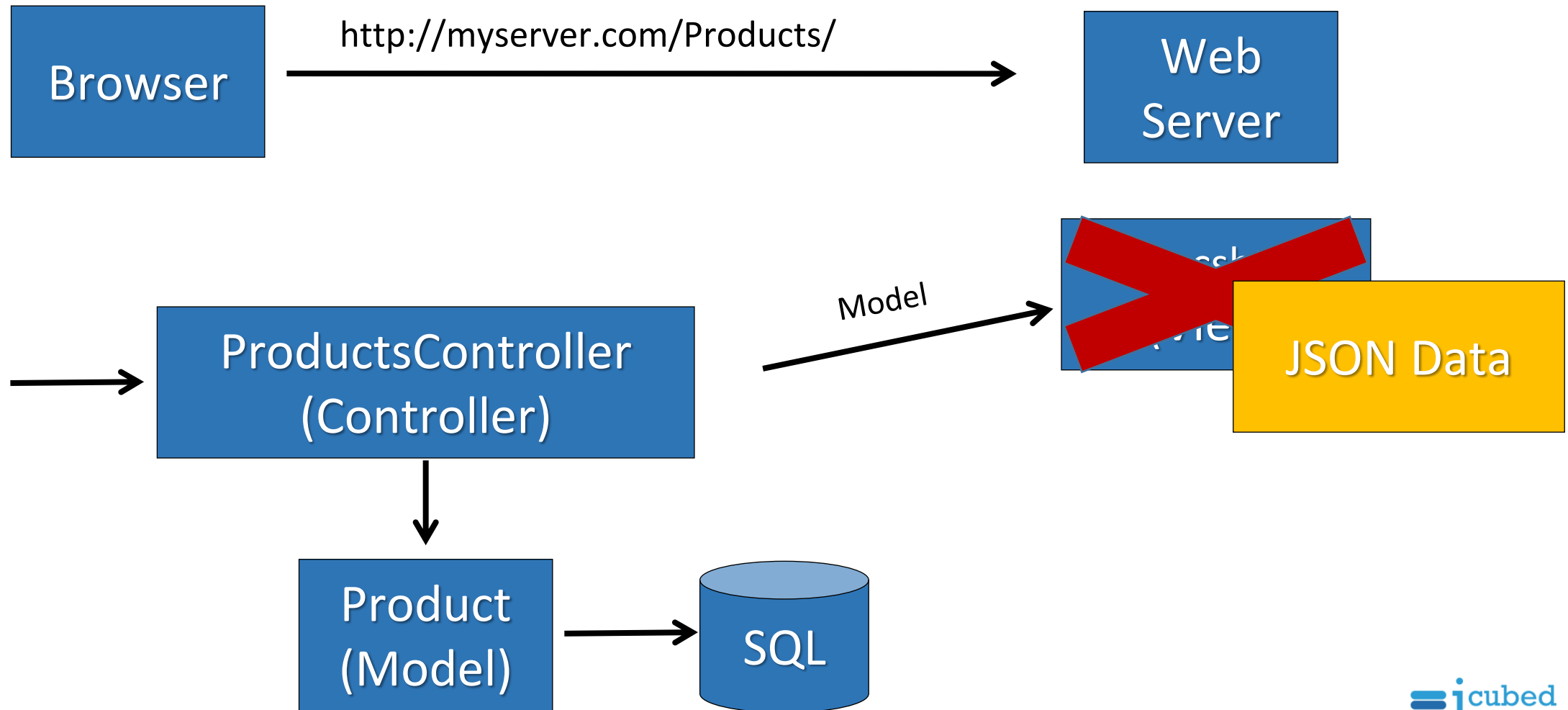


Il **Model** ha il compito di contenere i dati da visualizzare ed i metodi che ne permettono l'accesso al nostro engine di persistenza dati

La **View** ha il compito di visualizzare i dati da mostrare nella nostra User Interface

Il **Controller** è il vero cuore, si occupa delle iterazioni con l'utente invocando i metodi presenti nel Model e cambiando l'output della nostra interfaccia tramite la View

# La chiamata di una pagina con MVC



# Restituire dati dal controller

```
public ActionResult BookView()
{
    return Ok(new BookViewModel
    {
        Title = "ASP.NET Core 2 Guida completa per lo sviluppatore",
        Author = "Daniele Bochicchio et al",
        Category = "ASP.NET",
        Price = 19.99,
        ReleaseDate = DateTime.Today.AddDays(-40)
    });
}
```

# Restituire dati dal controller

```
public ActionResult BookView()
{
    return OK();                // HTTP Status 200
    return NotFound();          // HTTP Status 404
    return BadRequest();        // HTTP Status 400
    return CreatedAtAction();    // HTTP Status 201
    return RedirectToAction();
}
```



# Esercitazione 3

Realizzare un servizio REST con ASP.NET Core Web API che implementi le funzionalità di Esercitazione 1 e 2

- Definire i controller per la gestione delle operazioni di CRUD su Libri e Prestiti
- Ovviamente va utilizzato il data layer in EF Core



# Torniamo su Startup.cs

È l'entry point della configurazione

Rappresenta tutto quello che deve essere necessario al runtime per funzionare

Carica servizi, middleware

Configura l'IoC

Gestisce gli environment

# Il metodo ConfigureServices

Per configurare ASP.NET

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}
```

# Il metodo Configure

Gestisce come ASP.NET Core risponderà alle richieste

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env, ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseStaticFiles();

    app.UseIdentity();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

# HttpClient

Accedere alle API dal client



# Accedere alle API - HttpClient

Per effettuare chiamate dal vostro client, potete utilizzare la classed .NET Core **HttpClient**

```
HttpClient client = new HttpClient();
```

# Accedere alle API - HttpClient

Serve costruire una richiesta HTTP ...

```
HttpClient client = new HttpClient();

HttpRequestMessage httpRequest = new HttpRequestMessage
{
    Method = HttpMethod.Get, // HttpMethod.Post, ...
    RequestUri = fullUri
};
```

... specificando URL e HTTP Verb

# Accedere alle API - HttpClient

Tramite la proprietà **Content** posso specificare il body della request ...

```
HttpClient client = new HttpClient();  
  
HttpRequestMessage httpRequest = ...  
  
Gift newGift = new Gift();  
string json = JsonConvert.SerializeObject(newGift);  
httpRequest.Content = new StringContent(json,  
    Encoding.UTF8, "application/json");
```

... lo devo specificare serializzandolo come JSON.



# Accedere alle API - HttpClient

Tramite il metodo **SendAsync()** vado ad invocare la mia API ...

```
HttpClient client = new HttpClient();

HttpRequestMessage httpRequest = //...
string json = //...
httpRequest.Content = //...

var response = await client.SendAsync(httpRequest);

if (response.IsSuccessStatusCode)
{
    // ...
}
```

... l'esito lo leggo tramite la proprietà **IsSuccessStatusCode** della risposta

# Accedere alle API - HttpClient

Posso accedere al body della Response, che sarà in JSON ...

```
HttpClient client = new HttpClient();

HttpRequestMessage httpRequest = //...
string json = //...
httpRequest.Content = //...
var response = //...

if (response.IsSuccessStatusCode)
{
    var jsonResponse = await response.Content.ReadAsStringAsync();
    var result = JsonConvert.DeserializeObject<GiftContract>(jsonResponse);
}
```

... e quindi devo de-serializzarlo  
in un oggetto C#

# Domande?



Ricordate il feedback!



# © 2021 iCubed Srl



La diffusione di questo materiale per scopi differenti da quelli per cui se ne è venuti in possesso è vietata.

iCubed s.r.l.

Piazza Duca D'Aosta, 12 20124 MILANO

Phone: +39 02 57501057

P.IVA 07284390965

