Memòria

Estudi de tecnologies d'accés a dades

Alumne: Daniel Aguilar Aranda

Tutor: Xavier Navarro Esteve

13/6/2011

Index

1. Resum del Projecte

1.1 Objectius del Projecte

L'objectiu del projecte ha estat realitzar un estudi de les tecnologies d'accés a dades disponibles a la plataforma .NET: model clàssic , Entity Framework i OData.

Els aspectes més rellevants que s'han considerat són:

- El seu funcionament.
- Les seves avantatges i desavantatges.
- Comparativa a nivell d'aspectes teòrics i pràctics.

Per tal d'exemplificar i demostrar l'estudi, s'ha desenvolupat una aplicació web, primerament amb l'accés a dades clàssic i després dues versions amb Entity Framework i OData, per tal de poder avaluar les dificultats , beneficis i desavantatges alhora de desenvolupar cadascuna de les versions.

Aquesta aplicació web simula un prototipus d'un portal per cercar vehicles d'ocasió orientat a facilitar el contacte entre venedors i compradors.

Les funcionalitats que s'han desenvolupat són:

1. **Manteniment de vehicles**: donar d'alta , donar de baixa i modificar un vehicle.
2. **Filtrar i Buscar vehicles segons criteris**: Marca,Model, Preu, Km,….
3. **Consulta de les característiques d'un vehicle**: a partir del llistats resultants de cerques, seleccionar un vehicle per veure les seves característiques.
4. **Fer una oferta per un vehicle**: el comprador potencial podrà fer una oferta per un vehicle i el venedor podrà acceptar-la o rebutjar-la.
5. **Gestió d'usuaris**: els usuaris venedors s'han de registrar per oferir el seus vehicles. El compradors NO cal que ho facin.

Un cop desenvolupades les versions, s'ha pogut comprovar que l'exemple escollit ha estat un encert, ja que les funcionalitats desenvolupades estan formades per operacions / tasques presents en gran part dels projectes de desenvolupament que es realitzen en el món de les TI: manteniments, consultes amb filtre i llistats.

1.2 Abast del Projecte

L'abast del projecte ha vingut determinat per l'estudi teòric i pràctic. S'han explicat les tecnologies d'accés a dades i s'ha demostrat el seu comportament mitjançant un exemple pràctic. En cap moment s'ha tingut la intenció de fer una gran aplicació ni entrar en aspectes tecnològics que queden fora de l'objectiu del projecte. Les versions entregades són prototipus que exemplifiquen la manera de resoldre l'accés a dades mitjançant una tecnologia.

Resumidament l'estudi teòric i pràctic ha comportat les tasques que és van preveure:

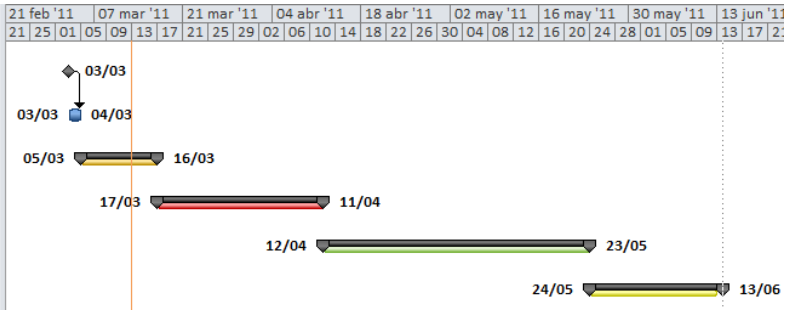
- Buscar informació sobre les tecnologies.
- Buscar una idea de projecte adient per posar-les en pràctica.
- Planificar i estimar el projecte.
- Fer un estudi teòric per comprendre el seu funcionament.
- Desenvolupar un prototipus de l'exemple triat per cada versió.
- Fer una comparativa entre les tecnologies.
- Extreure conclusions.
- Presentar el projecte en una memòria i una presentació.

Finalment, comentar que les decisions presses tant a nivell d'estimació, planificació i profunditat han vingut condicionades pel temps disponible (suficient) i la meva experiència.

1.3 Planificació

A l'inici del projecte es va preveure una planificació pel correcte desenvolupament del projecte:

Nombre de tarea	Duración	Comienzo	Fin
Inici del Projecte	0 días	jue 03/03/11	jue 03/03/11
Escollir Tema del projecte	2 días	jue 03/03/11	vie 04/03/11
Pla de Treball	**9 días**	**sáb 05/03/11**	**mié 16/03/11**
Anàlisi i Disseny	**22 días**	**jue 17/03/11**	**lun 11/04/11**
Desenvolupament	**36 días**	**mar 12/04/11**	**lun 23/05/11**
Desplegament,Memòria i Presentació	**15 días**	**mar 24/05/11**	**lun 13/06/11**

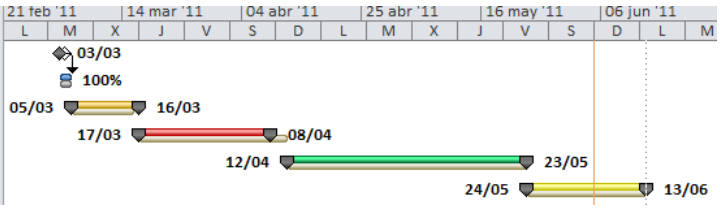


D'acord amb els terminis de lliurament de les PACS, es van agrupar les tasques en 4 fases:

- Pla de Treball
- Anàlisi i Disseny
- Desenvolupament
- Desplegament, memòria i presentació.

Un cop realitzat el projecte, s'han complert les estimacions de la duració de cada fase:

Nombre de tarea	Duración	Comienzo	Fin
Inici del Projecte	0 días	jue 03/03/11	jue 03/03/11
Escollir Tema del projecte	2 días	jue 03/03/11	vie 04/03/11
Pla de Treball	**9 días**	**sáb 05/03/11**	**mié 16/03/11**
Anàlisi i Disseny	**22 días**	**jue 17/03/11**	**vie 08/04/11**
Desenvolupament	**38 días**	**mar 12/04/11**	**lun 23/05/11**
Memòria i Presentació	**17 días**	**mar 24/05/11**	**lun 13/06/11**

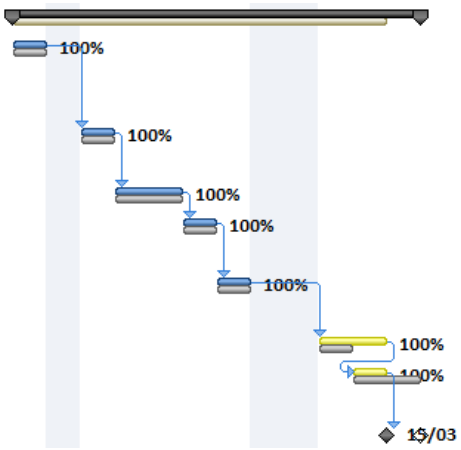


Les tasques de cada fase s'han pogut finalitzar en el temps planificat. Si s'observen els totals, s'aprecia alguna diferència en la duració i dates finals. Això es degut a que en la planificació inicial no es van tenir en compte alguns dies especials (diumenges o festius), que en la realitat si que han estat necessaris. De totes maneres ho he considerat "un fet anecdòtic", ja que la intenció sempre ha estat utilitzar tots els dies disponibles per cada entrega.

Si que hi ha hagut diferències en estimacions de les tasques que formen cada fase:

Fase Pla de Treball

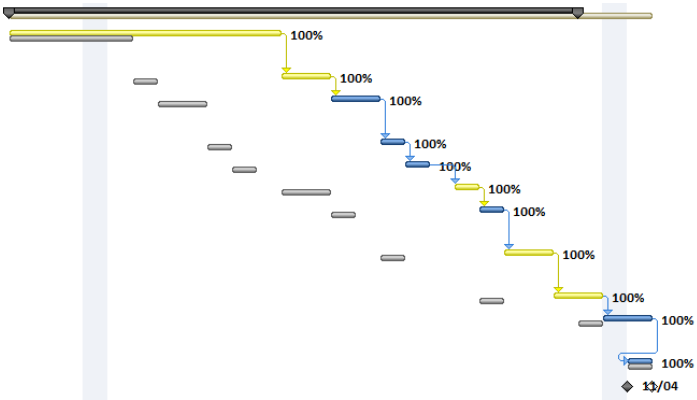
Pla de Treball	9 días	sáb 05/03/11	mié 16/03/11
Cercar informació sobre el tema del projecte "Tecnologies Accés a dades"	1 día	sáb 05/03/11	sáb 05/03/11
Cercar informació sobre les tecnologies a utilitzar	1 día	lun 07/03/11	lun 07/03/11
Definir la proposta de projecte	2 días	mar 08/03/11	mié 09/03/11
Descarregar el programari necessari	1 día	jue 10/03/11	jue 10/03/11
Instal·lació i configuració del programari	1 día	vie 11/03/11	vie 11/03/11
Planificació de tasques	2 días	lun 14/03/11	mar 15/03/11
Redacció document "Pla de Treball"	1 día	mar 15/03/11	mar 15/03/11
Entrega PAC 1	0 días	mar 15/03/11	mar 15/03/11



El motiu de l'alteració de les estimacions inicials ha estat la necessitat dedicar un dia més a la planificació de tasques i al fet que la documentació s'ha anat fent a partir d'un esborrany, en el qual s'anaven fent anotacions diàries de les tasques. Per tant, 1 dia ha estat suficient per redactar el document de la fase.

Fase Anàlisi i Disseny

Anàlisi i Disseny	22 días	jue 17/03/11	vie 08/04/11
Estudi de la tecnologies d'accés a dades: ODATA, Entity F. i Clàssic	10 días	jue 17/03/11	dom 27/03/11
Redacció Estudi	2 días	lun 28/03/11	mar 29/03/11
Anàlisi Funcional de la proposta de projecte	2 días	mié 30/03/11	jue 31/03/11
Disseny del model de dades	1 día	vie 01/04/11	vie 01/04/11
Disseny de la interfície d'usuari	1 día	sáb 02/04/11	sáb 02/04/11
Disseny tècnic del projecte	1 día	lun 04/04/11	lun 04/04/11
Disseny tècnic accés Dades "model clàssic"	1 día	mar 05/04/11	mar 05/04/11
Disseny tècnic accés Dades "Entity Framework"	2 días	mié 06/04/11	jue 07/04/11
Disseny tècnic accés Dades "OData"	2 días	vie 08/04/11	sáb 09/04/11
Primer anàlisi comparatiu de les tecnologies d'accés a dades	1 día	dom 10/04/11	lun 11/04/11
Redacció documentació	1 día	lun 11/04/11	lun 11/04/11
Entrega PAC 2	0 días	lun 11/04/11	lun 11/04/11



En aquesta fase es va preveure una duració de l'estudi teòric de 4 dies. Donada la desconeixença de les tecnologies Entity Framework i OData i considerant que l'estudi teòric, era la tasca més important d'aquesta fase, es va decidir destinar quasi el 50 % del temps.

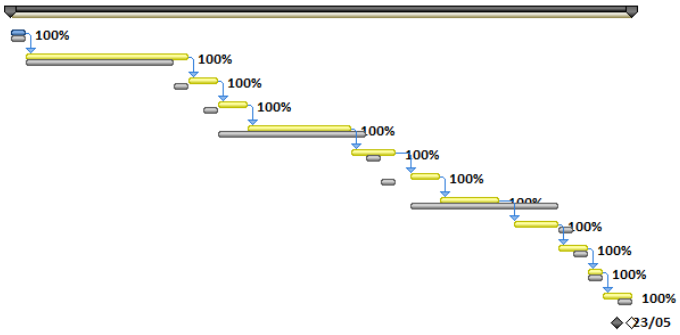
A nivell d'anàlisi i disseny es va veure que la importància es centrava en l'accés a dades en els tres models. Per tant, es va reduir el temps destinat al disseny tècnic de la lògica

de l'aplicació, en favor del disseny de l'accés amb OData i Entity Framework, ja que eren les tecnologies que més desconeixia.

A l'inici de la planificació d'aquesta fase, es va creure convenient realitzar prototipus de cadascun dels models d'accés a dades. Degut a l'augment de temps necessari per fer l'estudi teòric, i la importància de dissenyar correctament l'accés a dades de cada model, vaig decidir suprimir les tasques de prototipus així com la tasca de reconsideracions de redisseny segons els prototipus. Tota tasca de desenvolupament, es va deixar per la següent fase.

Fase Desenvolupament

Desenvolupament	38 días	mar 12/04/11	lun 23/05/11
Implementar la base de dades	1 día	mar 12/04/11	mar 12/04/11
Codificació de la versió "model clàssic"	10 días	mié 13/04/11	sáb 23/04/11
Realització joc de proves	2 días	dom 24/04/11	lun 25/04/11
Resolució errors	2 días	mar 26/04/11	mié 27/04/11
Codificació de la versió "Entity Framework"	6 días	jue 28/04/11	mié 04/05/11
Realització joc de proves	3 días	jue 05/05/11	sáb 07/05/11
Resolució errors EF	2 días	lun 09/05/11	mar 10/05/11
Codificació de la versió "OData"	4 días	mié 11/05/11	sáb 14/05/11
Realització joc de proves	3 días	lun 16/05/11	mié 18/05/11
Resolució errors OData	2 días	jue 19/05/11	vie 20/05/11
Creació Instal·lables versions	1 día	sáb 21/05/11	sáb 21/05/11
Manual Usuari i Instal·lació	1 día	dom 22/05/11	lun 23/05/11
Entrega PAC 3	0 días	lun 23/05/11	lun 23/05/11

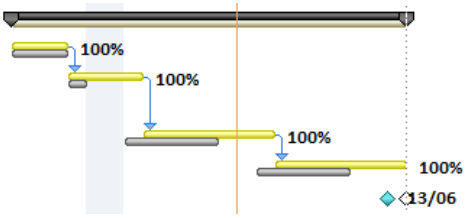


Aquesta fase és la que ha patit més variacions respecte la estimació inicial. A l'inici es va preveure que la codificació de cadascuna de les versions portaria 9 dies. En la realitat, la codificació de la versió "model clàssic" és la que ha dut més dies ja que, a part de desenvolupar el model clàssic, creava tota la lògica de l'aplicació juntament amb les pantalles, estils... La versió Entity Framework, ha vist reduït el seu temps de desenvolupament, ja que ha aprofitat la lògica creada en l'altra versió i només ha calgut implementar l'accés a dades. La versió OData, també ha patit una reducció de temps, ja que, a part d'aprofitar la lògica, ha basat part del seu desenvolupament en la versió Entity Framework, degut a que també fa l'accés a dades a partir d'un Entity Data Model.

Destacar també que les tasques de jocs de proves i correcció d'errors han necessitat més temps i que la tasca d'anàlisi comparatiu s'ha deixat per la següent fase. En el seu lloc, s'ha destinat el temps a crear els instal·lables de les versions i els manuals d'instal·lació i usuari.

Fase Memòria i Presentació

Memòria i Presentació	**17 días**	**mar 24/05/11**	**lun 13/06/11**
Comparativa i conclusions entre models	3 días	mar 24/05/11	jue 26/05/11
Redactar documentació comparativa i conclusions	2 días	vie 27/05/11	lun 30/05/11
Elaborar la memòria del PFC	6 días	mar 31/05/11	lun 06/06/11
Elaborar la presentació	6 días	mar 07/06/11	lun 13/06/11
Fi del Projecte - Lliurament	0 días	lun 13/06/11	lun 13/06/11



En aquesta fase, inicialment s'havia previst una tasca de desplegament d'un entorn de test de les versions. Aquesta tasca ha estat suprimida perquè ja es va fer en la fase anterior (joc proves). En el seu lloc, s'ha realitzat un comparativa entre els models estudiats i s'ha redactat les conclusions per poder-les incorporar a la memòria final.

Altrament, s'ha necessitat un dia més del previst per les tasques de la memòria i la presentació.

1.4 Producte Entregat

Al finalitzar cadascuna de les fases especificades en la planificació, s'ha entregat el següent:

Fase 1: Pla de treball

Document que explica i detalla els objectius del projecte, la seva planificació inicial, l'abast del projecte, els riscos i la descripció de que es vol fer a nivell pràctic.

Fase 2: Estudi teòric , Anàlisi i Disseny.

Document que explica l'estudi teòric de les tecnologies d'accés a dades i descriu l'anàlisi i disseny de l'estudi pràctic.

Fase 3: Producte

S'entrega una versió per cada tecnologia d'accés (clàssic, Entity Framework i OData) de l'aplicació d'exemple de l'estudi juntament amb un manual d'instal·lació i un manual d'usuari.

Fase 4: Memòria i Presentació

S'entrega un document que recull la memòria del projecte i un vídeo per presentar la feina realitzada amb un demostració pràctica del producte.

1.5 Tecnologia utilitzada

Per la realització de les aplicacions s'ha utilitzat les següents eines / tecnologies:

- ASP .NET 4.0
- ADO .NET amb Datasets
- Entity Framework
- OData (WCF Data Service)
- AJAX
- LINQ to Entities
- Llenguatge de programació C#
- Visual Studio 2010 Professional Edition
- Sql Server 2008
- IIS, Internet information Services.

No ha calgut utilitzar cap tecnologia o eina addicional a la previsió inicial.

1.6 Riscos del projecte

La previsió de riscos del projecte que es va estimar va ser la següent:

- La desconeixença de la tecnologies Entity Framework i OData.
- L'exemple d'aplicació triat podia no ser el més adient per mostrar el beneficis d'una tecnologia. La intenció era intentar exemplificar les operacions més comunes en el 90 % del projectes web actuals (formularis de manteniment de dades, filtres de consulta, llistats amb el resultat de la consulta i detall d'un element del llistat).

Com es va preveure, aquests riscos han estat els causants de les variacions en la estimació inicial de la planificació de tasques, si bé no han provocat cap retard important ni cap situació que hagi provocat la no finalització del projecte en el temps disponible.

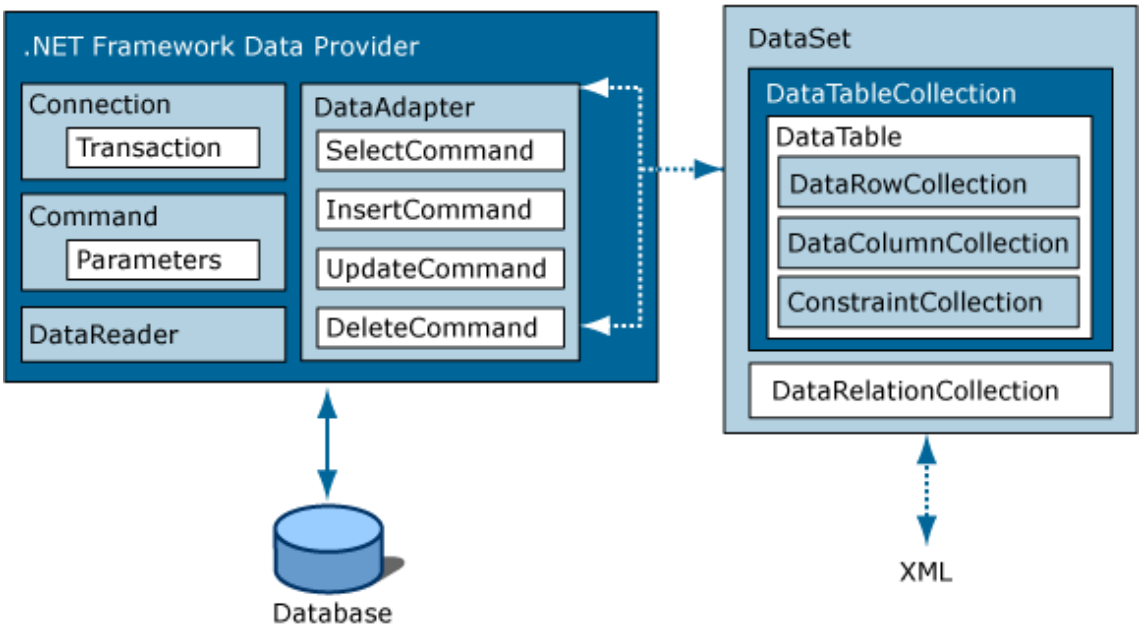
2. Estudi Teòric

2.1 ADO .NET Model Clàssic

ADO.NET és un conjunt de classes que faciliten el desenvolupament d'aplicacions que interactuen amb orígens de dades (p.e bases de dades, XML).

La seva arquitectura permet separar l'accés a les dades de la seva manipulació i ofereix proveïdors de dades per connectar amb bases de dades , executar comandes i recuperar resultats.

A continuació podem veure el seu esquema:



Com es pot observar està formada pel Proveïdor, el Dataset i l'origen de dades.

El **Proveïdor** és conjunt de components que permeten l'accés i manipulació de les dades per una base de dades concreta. Els més importants:

- **Connection**: realitza la connexió amb el servidor de dades.
- **Command**: permet realitzar operacions d'escriptura i lectura de dades.
- **DataReader**: crea un canal d'accés a les dades d'alt rendiment.
- **DataAdapter**: interfície entre el **Dataset** i l'origen de dades.

El **Dataset** és una representació de dades a memòria que proporciona una estructura relacional independent de l'origen de dades que conté. Està format per una col·lecció de taules (**DataTables**) que a la vegada contenen columnes (**DataColumn**) que defineixen l'estructura de les files (**DataRow**).

Aquest tipus d'objectes actuen com a repositoris desconnectats per mantenir un conjunt de dades amb les que es vol operar. Donada la seva versatilitat poden ser utilitzats directament (carregar informació sobre ells) o definir un estructura (**Typed Dataset**) per garantir una integritat del seu contingut.

A mida d'explicar el funcionament d'aquesta arquitectura, descriuré una operació de lectura (SELECT) contra una base de dades:

1. Escollir un proveïdor segons l'origen. Els més utilitzats:
   - **SQL Server**: orígens de dades en bases de dades SQL Server.
   - **Oracle**: orígens de dades en bases de dades Oracle.
   - **OLE DB**: orígens de dades exposats mitjançant OLE DB.
   - **ODBC**: orígens de dades exposats mitjançant ODBC.

2. Crear una connexió mitjançant l'objecte **Connection** del proveïdor i la cadena de connexió **ConnectionString.**

   La cadena de connexió proporciona a l'objecte Connection la informació necessària per realitzar la connexió a la base de dades.

3. Crear la comanda que es vol executar mitjançant l'objecte **Command** del proveïdor i els seus paràmetres.
4. Obrir la connexió
5. Executar la comanda
6. Operar amb les dades obtingudes.
7. Tancar la connexió.

Exemple del procés per una consulta de dades:

```
    String connectionString= "Data Source=.\SQLEXPRESS;Initial
    Catalog=BBDDVehicles;Integrated Security=True
    providerName="System.Data.SqlClient"

using (SqlConnection connection = new SqlConnection(connectionString))

    {

        // Creació de la comanda

        SqlCommand command = new SqlCommand();

        command.Connection = connection;

       command.CommandText = "SELECT EmployeeName FROM Employees Where
      DepartmentID=@DepartmentID";

        command.CommandType = CommandType.Text;

        // Crear el paràmetre @DepartmentID

        SqlParameter parameter = new SqlParameter();

        parameter.ParameterName = "@DepartmentID";

        parameter.SqlDbType = SqlDbType.NVarChar;

        parameter.Direction = ParameterDirection.Input;

        parameter.Value =”D90089”;

        // Afegir el paràmetre ala comanda

        command.Parameters.Add(parameter);

        // Obrir la connexió

        connection.Open();

        //Executar la comanda

        SqlDataReader reader = command.ExecuteReader();

        // Operar amb la informació obntinguda

        if (reader.HasRows)
        {
            while (reader.Read())
            {
                Console.WriteLine("Nom Treballador: {0}", reader[0]);
            }
        }
        else
        {
            Console.WriteLine("Sense resultats");
        }
        reader.Close();}
```

En aquest exemple, s'ha utilitzat un **DataReader** per executar la comanda. Aquest objecte permet anar llegint dades de l'origen per operar sense emmagatzemar-les prèviament en un objecte.

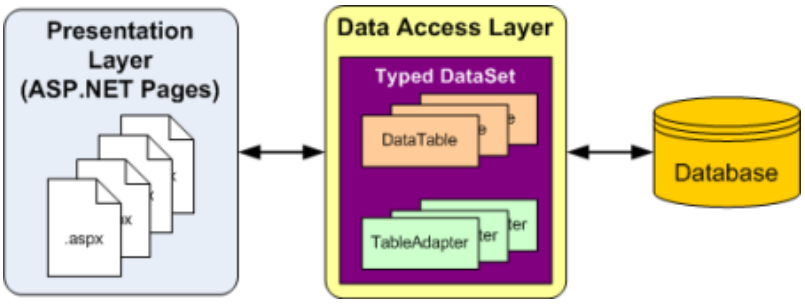
Altra manera de treballar amb el resultat de la comanda és fer-ho de manera desconnectada, és a dir, emmagatzemant prèviament els resultats de la comanda en memòria utilitzant una objecte anomenat **Dataset** el qual és independent del proveïdor i pot contenir informació de diferents orígens. Aquest procés és possible gràcies a un objecte **DataAdapter** el qual s'encarrega d'obrir la connexió, executar la comanda i emmagatzemar el resultat en un **Dataset**.

Exemple de comanda amb Dataset utilitzant un DataAdapter:

```
using (SqlConnection connection = new SqlConnection(connectionString))

{

        string queryString = "SELECT * FROM Employees";
        SqlDataAdapter adapter = new SqlDataAdapter(queryString, connection);

        Dataset dsEmployees = new Dataset();
        adapter.Fill(dsEmployees, "Employees");
}
```

Destacar que no cal crear un objecte Command. El DataAdapter rep la comanda amb un objecte string i ja s'encarrega d'executar-la (internament ja crea els objectes Command que necessiti).

Aquesta potència que ofereixen el DataAdapter i el Dataset pot ser evolucionada fins a treballar seguint la següent arquitectura:



una capa d'accés a dades fa l'enllaç entre la capa de presentació i l'origen de dades. Els Dataset amb tipus defineix l'estructura de la informació a guardar i un "TableAdapter" (objecte semblant al DataAdapter) gestiona les operacions de lectura i escriptura entre l'origen de dades i aquest Dataset. La capa de presentació utilitza el Dataset per llegir i escriure informació a l'origen de dades.

Aquest escenari és el que ofereix l'entorn de disseny de Visual Studio i en la part pràctica de l'estudi, s'utilitzarà juntament amb una capa de lògica (Business Logic Layer).

La utilització d'aquest model clàssic aporta les següents avantatges:

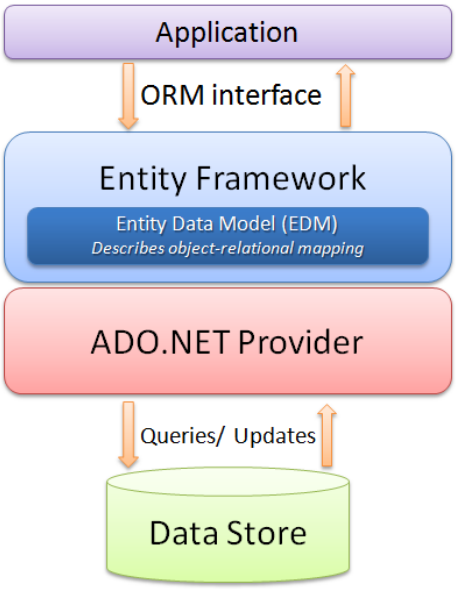
- **Interoperabilitat**: ADO.NET aprofita la flexibilitat de XML com a format de transmissió de Datasets. Qualsevol component o aplicació que pugui llegir XML podrà operar amb les dades.
- **Mantenibilitat:** la facilitat per comunicar capes mitjançant XML (Datasets) fan que afegir capes en aplicacions per distribuir i reduir la càrrega del sistema sigui un tasca viable.
- **Programabilitat:** l'ús de Datasets amb tipus simplifica l'escriptura de codi i informa d'errors en temps de compilació.
- **Escalabilitat:** ADO .NET utilitza accés desconnectat a les dades per tal de no boquejar connexions amb bases de dades i estalviar recursos (servei a gran nombre d'usuaris).

2.2 ADO .NET Entity Framework

ADO .NET Entity Framework és un conjunt de tecnologies que permet treballar amb dades utilitzant un model relacional d'objectes (Object Relational Mapping, ORM) enlloc d'un model relacional de dades, evitant la necessitat d'utilitzar sentències pròpies de llenguatges de consulta a orígens de dades (SQL).

Aquesta característica ajudar als desenvolupadors ja que no han de preocupar-se en les taules de la base de dades, columnes, tipus ...., sinó que treballen amb objectes i propietats per tractar les operacions amb dades. D'aquesta manera, poden centrar més l'atenció en la lògica de l'aplicació minimitzant el temps dedicat a tasques relacionades amb la gestió de les dades.

Per visualitzar que aporta de nou sobre el model clàssic, mostrem la seva arquitectura:

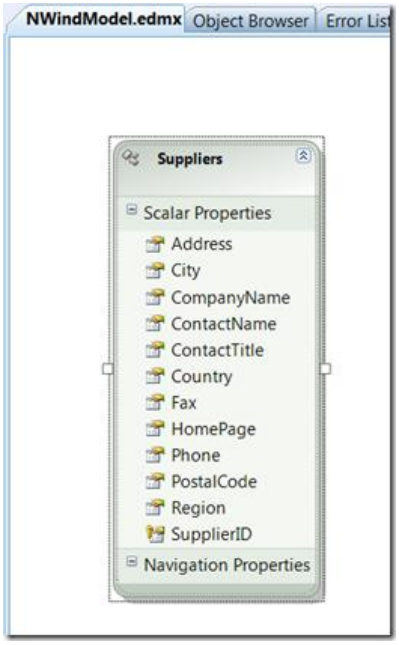


Entity Framework està construït com un capa per sobre del Proveïdor ADO .NET. Mitjançant un proveïdor específic (**EntityClient**) pot accedir a les dades. Aquest nou, EntityClient, utilitza proveïdors existents (p.e. proveïdor de SQL Server) per poder dur a terme les connexions i operacions amb els orígens de dades.

L'esquema mostra un element anomenat **Entity Data Model**, el qual descriu l'estructura de dades independentment del format amb que estiguin emmagatzemades, basant-se en el model entitat relació.

Per tant, mitjançant Entity Data Model (**EDM),** es representa el model conceptual (entitats i relacions...) en forma d'objectes creant un nivell d'abstracció superior a l'esquema de la base de dades.

Per veure aquesta arquitectura des d'un punt de vista pràctic posarem un exemple d'una operació de consulta partint del següent EDM:




```
using (NORTHWNDEntities NWcontext = new NORTHWNDEntities)
{
   Var companyInfo =  From s In Nwcontext.Suppliers Where s.SupplierID = 23
   Select s.CompanyName, s.ContactName;
}
```

Com es pot observar, s'ha realitzat una consulta contra la taula "Suppliers" de la base de dades amb les següents característiques:

- No s'ha utilitzat cap conjunt d'objectes d'un proveïdor específic.
- No s'ha utilitzat el llenguatge SQL com es feia en el model clàssic.

La capa Entity Framework s'ha encarregat de proporcionar un model conceptual de la base de dades al qual s'ha accedit mitjançant la creació d'un context (o connexió al model: *NORTHWNDEntities*) i mitjançant LINQ , s'ha fet la consulta sobre els objectes del EDM.

No ens ha calgut saber ni el proveïdor de la base de dades ni la seva sintaxi SQL específica. Aquest codi d'exemple funcionaria per tots els proveïdors de bases de dades; només caldria tenir un EDM que estigués enllaçat amb la base de dades amb la que volem treballar.

Aquest exemple justifica la principal raó d'utilitzar Entity Framework.

Resumint les característiques interessants que ens aporta:

- **Independència del servidor de dades**: el EntityClient ja s'entendrà amb el proveïdor corresponent .
- **Motor de mapeig d'objectes**: permet mantenir qualsevol esquema de dades i accepta treballar amb stored procedures.
- **Proporciona eines visuals integrades amb Visual Studio:** permet crear els models, regenerar-los ... Gestió integral del model dades.
- **Compatible amb les diferents tecnologies .NET**: ASP .NET, Windows Presentation Foundation (WPF), Windows Communication Foundation (WCF)

També les avantatges:

- **Reduir el temps de desenvolupament**: aquest framework encapsula la gestió de les dades de manera que els desenvolupadors poden dedicar-se al aspectes funcionals de l'aplicació.
- **Treballar només amb objectes**: "el desenvolupador només pensa amb objectes". Treballa a un nivell superior d'abstracció on només tenim objectes
- **Independència de codi específic al motor de base de dades**: les aplicacions són lliures de codi "hardcoded" específic per a un motor de base de dades gràcies al model d'objectes
- **Mapeig automàtic:** el mapeig del model d'objectes amb el model dades s'actualitza als canvis sense haver de modificar el codi font**.**
- **LINQ amb Intellisense:** aportar "intellisense" (ajuda de l'editor al teclejar) i validació de sintaxis en temps de compilació al llenguatge de consultes utilitzat contra el model conceptual.

2.3 Diferències teòriques Entity Framework vs Model clàssic

Quan es desenvolupa utilitzant Entity Framework , el sistema genera automàticament objectes simplificant el procés d'actualització de la base de dades. Aquesta característica proporciona la flexibilitat de realitzar canvis sobre l'esquema de dades amb total llibertat minimitzant l'impacte sobre el codi a modificar (el sistema crea un nivell d'abstracció que ajuda a aïllar l'aplicació de la base de dades). Aquesta característica no és present en el model clàssic, on tot canvi en la base de dades implica un canvi manual en els Datasets amb tipus.

Altra millora que aporta Entity Framework és la independència del motor de base de dades. Les operacions realitzades contra la base de dades, s'escriuen mitjançant LINQ o Entity SQL, les quals són traduïdes en temps d'execució pel proveïdor corresponent a la sintaxi de la base de dades.

En el model clàssic, el proveïdor d'una certa base de dades (p.e. Oracle) aporta el seus objectes per crear connexions, comandes.., alhora que requereix un sintaxi específica per fer les operacions de lectura / escriptura de dades. Aquest fet provoca que una migració, p.e Oracle a SQL Server, impliqui canviar aquestes sentències SQL (habitualment passades amb un String) així com altres objectes propis del proveïdor.

2.4 Model OData

En molts escenaris hi ha la necessitat de tenir accés a les dades d'un sistema de informació, p.e. el departament comercial ubicat a Madrid vol accedir a les dades de facturació ubicades a Barcelona o altre escenari on un client vol integrar la seva aplicació amb el nostre sistema i necessita accedir a les nostres dades. En el primer exemple, una aplicació web que permetés consultar / modificar les dades hauria de ser suficient. En el segon, un servei web que exposés el sistema de dades mitjançant un conjunt d'operacions (webmethods) per ser consumides pels processos de l'aplicació del client , solucionaria l'escenari.

Que passaria si les funcionalitats van augmentant i cap cop es necessita fer més operacions contra la base de dades ?

S'hauria d'anar modificant el servei web per oferir les noves operacions i segons la seva complexitat i diversitat , aquest sistema no seria l'adient. Sorgiria la necessitat de tenir accés a les dades d'una manera més directa i independent , com si disposéssim d'un llenguatge per parlar amb el servei web igual que ho podem fer amb les bases de dades (SQL). A més, posats a triar, el resultat hauria de seguir un estàndard de manera que fos vàlid tant per aplicacions d'escriptori, web o altres tecnologies (p.e. PHP). La resposta a aquesta problemàtica és **Open Data Protocol**, **OData**.

OData és una tecnologia d'accés a dades orientada a la compartició d'aquestes a través de Internet. Permet la creació de serveis de dades sobre HTTP mitjançant un model de dades abstracte a través d'un identificador uniforme de recursos (URI).

Objectius de OData

- Crear un model uniforme de representació de dades estructurades a través de JSON i Atom.
- Utilitzar convencions URL uniformes per la navegació, filtrat, ordre i paginació de dades.
- Crear operacions estandarditzades de les accions GET,POST,PUT i DELETE per la consulta i manipulació de dades.

Funcionament de OData

OData exposa les dades a través de **AtomPub** (RFC 2023), el qual està construït sobre **Atom** (estàndard RFC 4287). També exposa funcions anomenades **Service Operations** les quals acceptem paràmetres i retornen **Entries**.

AtomPub és un protocol del nivell d'aplicació per publicar i editar recursos web. Està basat en transferències HTTP de representacions Atom.

Atom és un format XML que descriu informació relacionada a la qual anomena ***feeds***. Aquests a la vegada estan formats per un conjunts de col·leccions anomenades ***Entries***. Cada entry és un registre estructurat que representa un conjunt de propietats (***Properties***) de dades amb una o vàries ***keys.***

Un exemple d'una entry simple que forma un feed (RFC 4287) :

```
<?xml version="1.0" encoding="utf-8"?>
  <feed xmlns="http://www.w3.org/2005/Atom">

    <title>Example Feed</title>
    <link href="http://example.org/"/>
    <updated>2003-12-13T18:30:02Z</updated>
    <author>
      <name>John Doe</name>
    </author>
    <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>

    <entry>
      <title>Atom-Powered Robots Run Amok</title>
      <link href="http://example.org/2003/12/13/atom03"/>
      <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
      <updated>2003-12-13T18:30:02Z</updated>
      <summary>Some text.</summary>
    </entry>

  </feed>
```

OData exposa els feeds en forma de documents mitjançant **Service Document** o **Service Metadata Document**.

Service Document descriu els elements de més alt nivell. L'exemple que podem trobar a http://services.odata.org/OData/OData.svc/:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<service xml:base="http://services.odata.org/OData/OData.svc/"
xmlns:atom="http://www.w3.org/2005/Atom" xmlns:app="http://www.w3.org/2007/app"
 xmlns="http://www.w3.org/2007/app">
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="Products">
      <atom:title>Products</atom:title>
    </collection>
    <collection href="Categories">
      <atom:title>Categories</atom:title>
    </collection>
    <collection href="Suppliers">
      <atom:title>Suppliers</atom:title>
    </collection>
  </workspace>
</service>
```

Service Metadata Document descriu tot el EDM (Entity Data Model) utilitzant la opció $metada. L'exemple http://services.odata.org/OData/OData.svc/$metadata que mostra la entitat Product:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
  <edmx:DataServices xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  m:DataServiceVersion="2.0">
    <Schema Namespace="ODataDemo" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns="http://schemas.microsoft.com/ado/2007/05/edm">
      <EntityType Name="Product">
        <Key>
          <PropertyRef Name="ID" />
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false" />
        <Property Name="Name" Type="Edm.String" Nullable="true" m:FC_TargetPath="SyndicationTitle"
        m:FC_ContentKind="text" m:FC_KeepInContent="false" />
        <Property Name="Description" Type="Edm.String" Nullable="true" m:FC_TargetPath="SyndicationSummary"
        m:FC_ContentKind="text" m:FC_KeepInContent="false" />
        <Property Name="ReleaseDate" Type="Edm.DateTime" Nullable="false" />
        <Property Name="DiscontinuedDate" Type="Edm.DateTime" Nullable="true" />
        <Property Name="Rating" Type="Edm.Int32" Nullable="false" />
        <Property Name="Price" Type="Edm.Decimal" Nullable="false" />
        <NavigationProperty Name="Category" Relationship="ODataDemo.Product_Category_Category_Products"
        FromRole="Product_Category" ToRole="Category_Products" />
        <NavigationProperty Name="Supplier" Relationship="ODataDemo.Product_Supplier_Supplier_Products"
        FromRole="Product_Supplier" ToRole="Supplier_Products" />
      </EntityType>
```

Aquesta descripció del EDM en XML és independent al format que s'exposi (JSON,Atom), segueix la definició d'esquemes conceptuals (CSDL) i només exigeix que es publiqui a través de HTTP.

Consum de dades a través de OData

Per exposar un EDM mitjançant OData cal seguir unes convencions respecte a la URI (Uniform Resource Indentifier) que l'identifica.

Una URI consta de tres parts:

- URI arrel del servei
- Direcció dels recursos
- Opcions de Consulta

Un exemple de URI per obtenir els dos primers productes de la categoria 1 ordenats pel nom seria:

http://services.odata.org/OData/OData.svc/Category(1)/Products?$top=2&$orderby=name

URI Arrel — Direcció Recursos — Opcions Consulta

Les consultes sobre entitats , col·leccions o propietats es poden fer mitjançant la direcció de recursos. Els elements més habituals que podem consultar són els següents:

- **Collection**: col·lecció de entries exposades pel servei. P.e.: *http://services.odata.org/OData/OData.svc/Categories*
- **KeyPredicate**: valor o expressió que identifica a una o vàries entries. P.e.: *http://services.odata.org/OData/OData.svc/Categories(1)*
- **Property**: propietat d'una entry. P.e.: *http://services.odata.org/OData/OData.svc/Categories(1)/Name*
  Resultat de la consulta:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<Name xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">Beverages</Name>
```

Aquest sistema de consultes permet utilitzar un conjunt d'opcions que permeten fer operacions habituals com ordenar els resultats, filtrar, obtenir relacions entre entitats, obtenir el valor d'una propietat... Ens els següents exemples es mostra la seva potència:

- **$links**: consultar relacions entre entitats.

  P.e. per consultar els productes de la categoria 1:

  *http://services.odata.org/OData/OData.svc/Categories(1)/**$links**/Products*

- **$value**: obtenir el valor d'una propietat

  *http://services.odata.org/OData/OData.svc/Categories(1)/Name/**$value***

- **$orderby**: ordenar el resultats.

  *http://services.odata.org/OData/OData.svc/Categories?**$orderby**=Name*

- **$filter**: filtrar el resultats.

  http://services.odata.org/OData/OData.svc/Categories?$filter=toupper(Name) eq 'Beverages'

També es pot obtenir els resultats en els formats Atom, JSON o XML mitjançant l'opció de consulta **$format:**

http://services.odata.org/OData/OData.svc/Categories?$format=json

En resum, veiem que a partir d'una URI, segons el que especifiquem en la direcció de recursos i les opcions de consulta, podem obtenir la informació desitjada. Els exemples mostrats es poden executar mitjançant jquery (javascript), amb una programació des de el costat client o mitjançant LINQ to Entities, fent referència al servei que exposa el EDM.

Consumidors de serveis OData

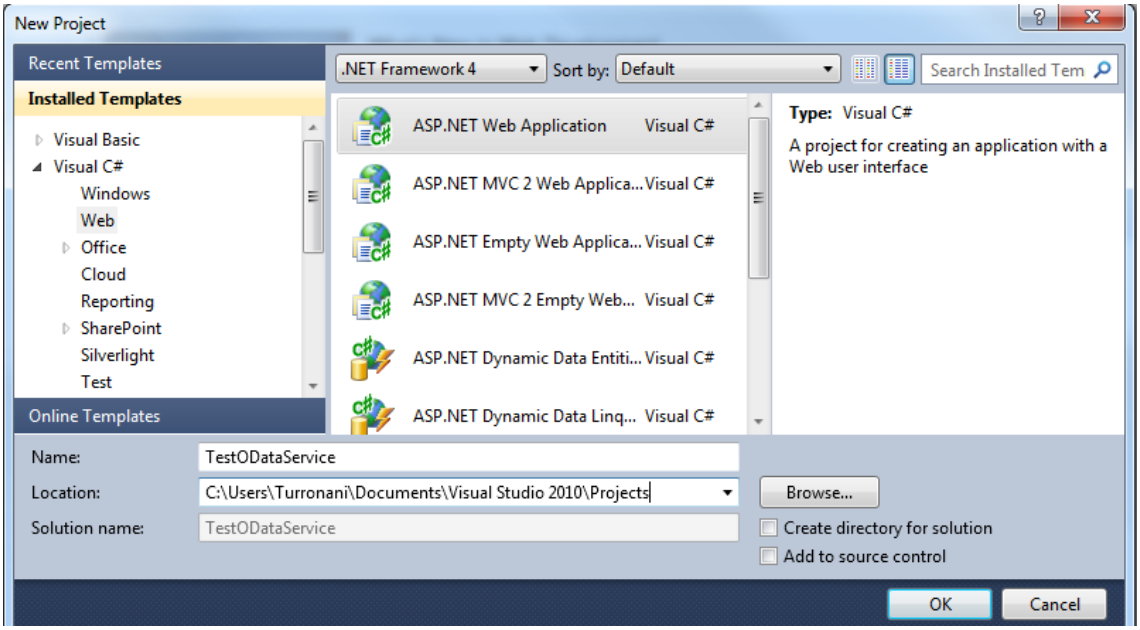
Existeix moltes tecnologies multiplataforma que permeten el consum de dades a través de OData des de el costat client. Exemples:

- **Windows Phone 7**: OData és la plataforma de serveis de dades que utilitza.
- **WCF**: mitjançant un proxy al servei de dades, es poden consumir des de aplicacions client creades amb visual studio..
- **JQuery**: utilitzant javascript, i concretament la llibreria jquery, es pot accedir a dades publicades amb OData.
- **PowerPivot amb excel 2010**: extensió gratuïta per excel que permet connectar-se a serveis OData i obtenir la seva informació.
- **Altres llibreries / llenguatges**: PHP, java, Objective C (iphone)...

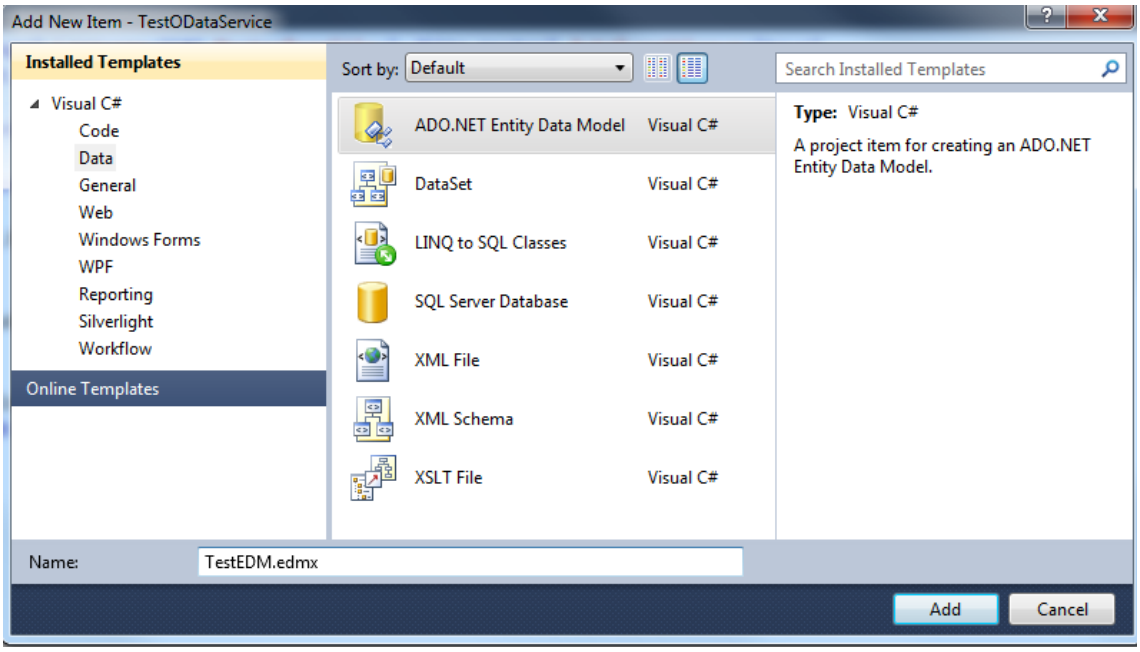
Creació d'un servei OData

Per poder fer realitat aquest tecnologia, cal crear un servei per exposar les dades que es volem oferir mitjançant OData. Un exemple de procés a seguir és el següent:

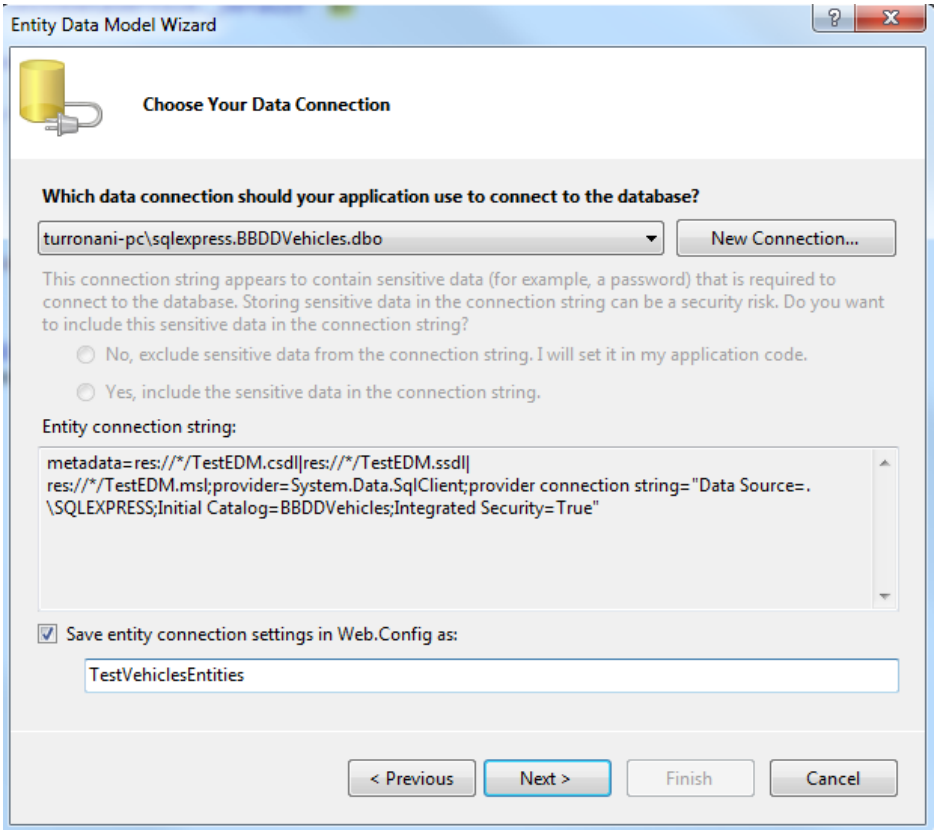
1. Crear una aplicació web per allotja el servei



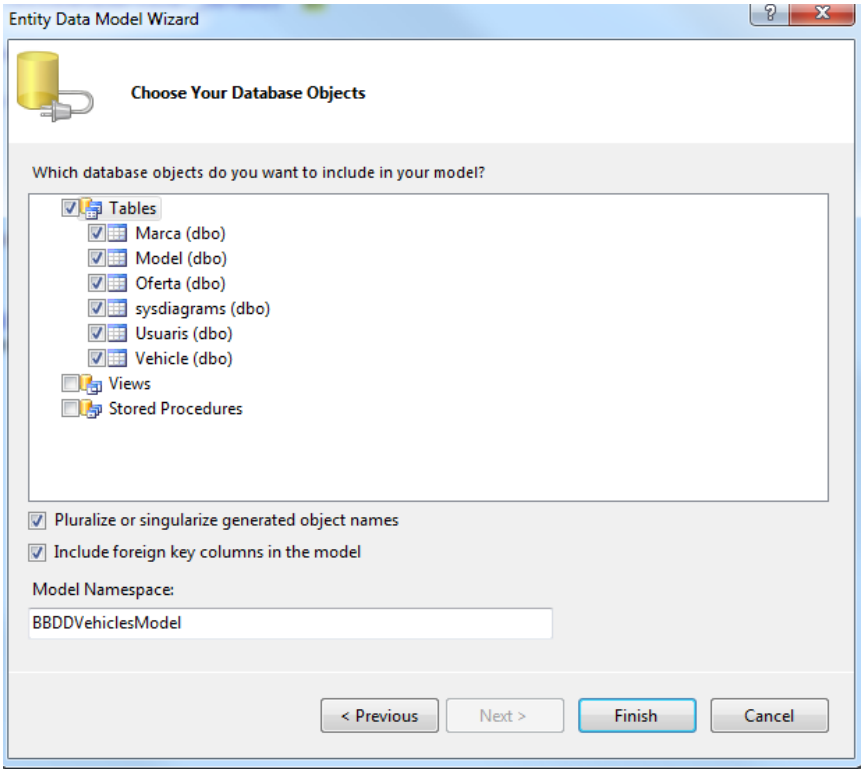
2. Afegir l'origen de dades a publicar mitjançant Entity Framework:



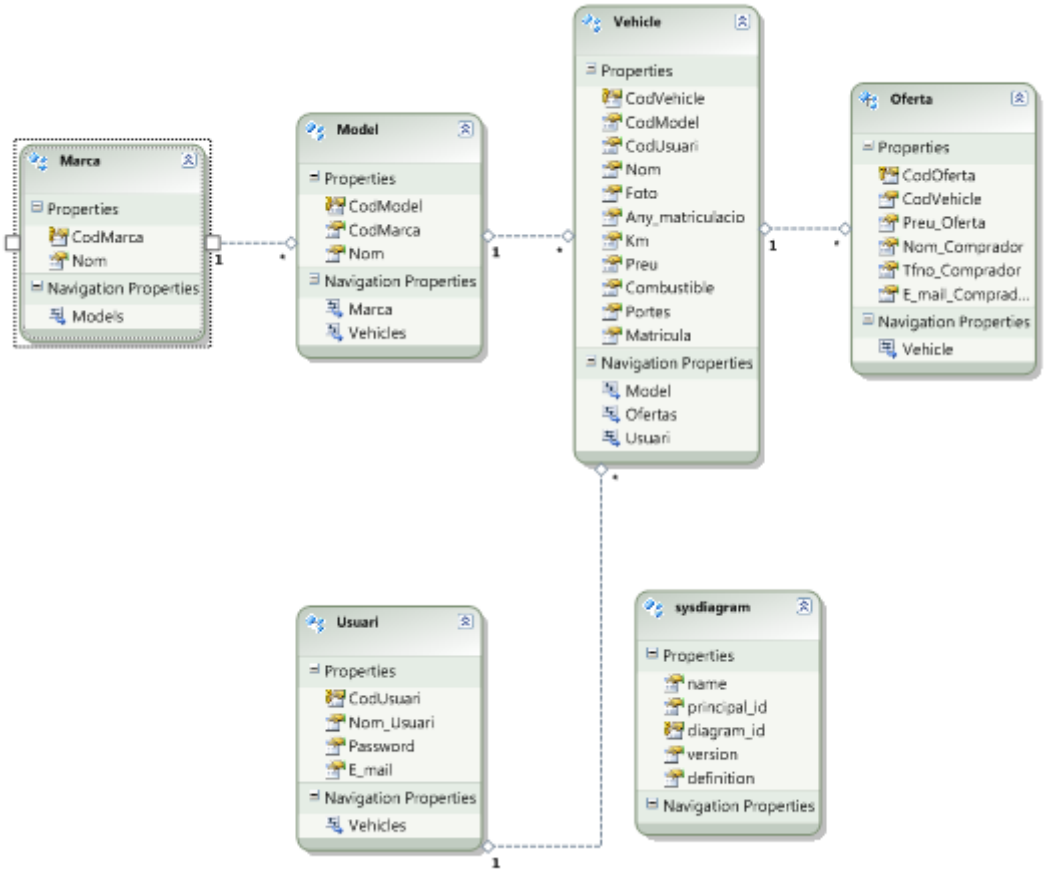
Seguir l'assistent de configuració indicat la base de dades a exposar



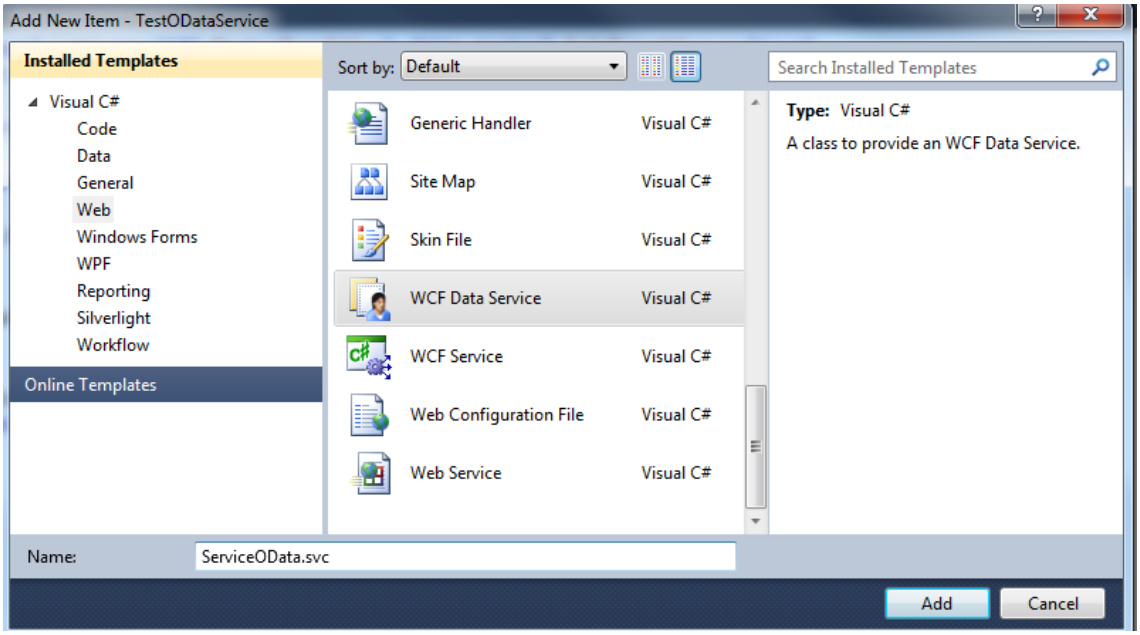
i les taules:



Aquí tenim el Entity Data Model:



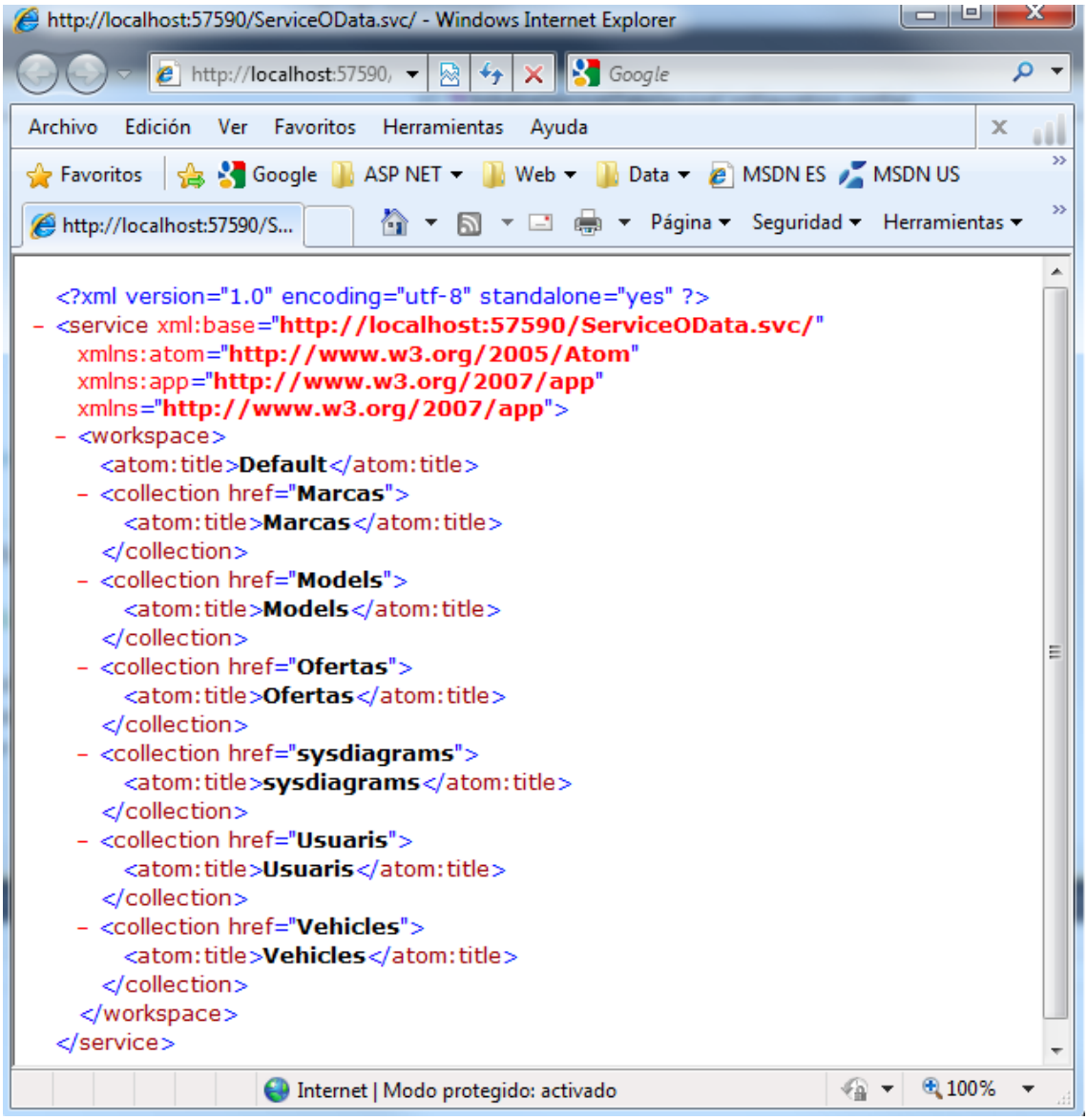
3. Publicar aquest EDM mitjançant un servei WCF Data Service:



Cal configurar el servei per indicar que es vol publicar:

```
public static void InitializeService(DataServiceConfiguration config)
{
            config.SetEntitySetAccessRule("*", EntitySetRights.All);
            config.SetServiceOperationAccessRule("*", ServiceOperationRights.All);
            config.DataServiceBehavior.MaxProtocolVersion =
DataServiceProtocolVersion.V2;
}
```

Executant el servei, comprovem que ja exposa el Service Document del EDM:



Productors de serveis OData

En el mercat existeixen aplicacions i sistemes , anomenats Productors, que poden exposar les seves dades a través de ODATA. Algunes d'aquestes són:

- **SAP NetWeaver Gateway**: és una tecnologia, basada en OData, que permet connexions a sistemes SAP.
- **SharePoint 2010:** plataforma de gestió empresarial que permet uilitizar OData per manipular la seva informació.
- **Microsoft SQL Azure:** permet publicar un servei per donar accés a vistes OData segons els permisos dels usuaris.
- **Microsoft Dynamics CRM 2011:** sistema de gestió de clients que permet consultes utilitzant OData.

També estan disponibles serveis públics que utilitzen ODATA:

Facebook Insights: servei OData per consumir dades de Facebook.

Ebay: exposa el seu catalog a través de OData.


```
  <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <service xml:base="http://ebayodata.cloudapp.net/" xmlns:atom="http://www.w3.org/2005/Atom"
    xmlns="http://www.w3.org/2007/app">
  - <workspace>
      <atom:title>Default</atom:title>
    - <collection href="Bidders">
        <atom:title>Bidders</atom:title>
      </collection>
    - <collection href="Categories">
        <atom:title>Categories</atom:title>
      </collection>
    - <collection href="CrossPromotions">
        <atom:title>CrossPromotions</atom:title>
      </collection>
    - <collection href="Deals">
        <atom:title>Deals</atom:title>
      </collection>
    - <collection href="Feedback">
        <atom:title>Feedback</atom:title>
      </collection>
    - <collection href="Items">
        <atom:title>Items</atom:title>
      </collection>
    - <collection href="Shippings">
        <atom:title>Shippings</atom:title>
      </collection>
    - <collection href="Transactions">
        <atom:title>Transactions</atom:title>
      </collection>
    - <collection href="Users">
        <atom:title>Users</atom:title>
      </collection>
    </workspace>
  </service>
```


Nerd Dinner: website per tenir cites amb gent i conversar. Publicar la seva informació amb OData:


```
  <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <service xml:base="http://www.nerddinner.com/Services/OData.svc/"
    xmlns="http://www.w3.org/2007/app">
  - <workspace>
      <atom:title>Default</atom:title>
    - <collection href="Dinners">
        <atom:title>Dinners</atom:title>
      </collection>
    - <collection href="RSVPs">
        <atom:title>RSVPs</atom:title>
      </collection>
    </workspace>
  </service>
```


Windows Live: mitjaçant un client OData es pot consultar els recursos propis de l'usuari (fotos, contactes …).

Nuget: és una extensió de Visual studio que facilita la instal·lació i actualització de eines opensource. Utilitza OData per publicar el seu catàleg d'eines.


```
  <?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <service xml:base="http://packages.nuget.org/v1/FeedService.svc/"
    xmlns="http://www.w3.org/2007/app">
  - <workspace>
      <atom:title>Default</atom:title>
    - <collection href="Packages">
        <atom:title>Packages</atom:title>
      </collection>
    - <collection href="Screenshots">
        <atom:title>Screenshots</atom:title>
      </collection>
    </workspace>
  </service>
```

3. Anàlisi i Disseny de l'aplicació

3.1 Funcionalitat de l'aplicació

Per la part pràctica de l'estudi, s'ha desenvolupat una aplicació que simula un portal per exposar vehicles d'ocasió orientat a facilitar el contacte entre venedors i compradors.

L'aplicació permet realitzar les següents funcionalitats:

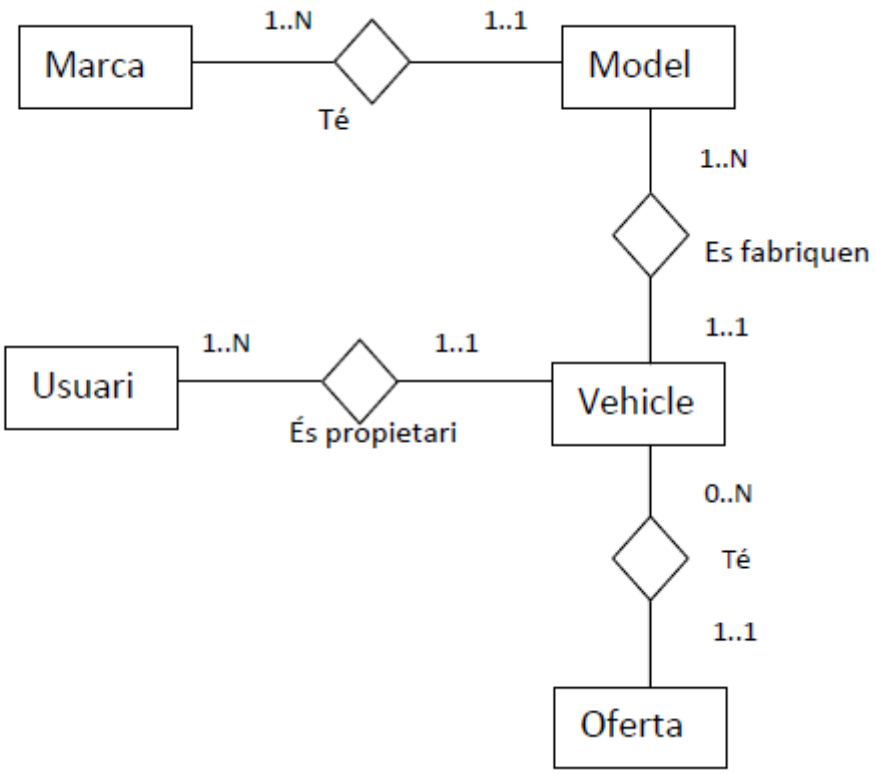
- **Gestió de vehicles**
- **Cerca de vehicles segons criteris.**
- **Consulta de les característiques d'un vehicle**.
- **Fer un oferta per un vehicle**.

En el següent diagrama de casos es mostra les funcionalitats de l'aplicació:

Nº	Funcionalitat	Dependència	Accions de l'usuari	Que mostra l'aplicació per Pantalla
1	Registre usuari	NO	Registrar-se en el sistema	Formulari per introduir les dades de l'usuari.
2	Gestió de vehicles	1	Donar d'alta un vehicle que es vol vendre.	Formulari sense dades per introduir les dades del vehicle.
			Modificar les dades d'un veh	Llistat de vehicles amb una columna "edit" que dona accés a un formulari editable amb les dades del vehicle seleccionat.
			Donar de baixa un vehicle.	Llistat de vehicles amb una columna "delete" que al seleccionar-la mostra una pregunta demanant la confirmació de la baixa del vehicle.
3	Cerca de vehicles	NO	Cercar vehicles segons uns criteris de cerca.	Llistat amb els vehicles que compleixen els criteris de cerca.
4	Característiques d'un vehicle	NO	Consultar les característiques del vehicle seleccionat.	Vista amb les característiques del vehicle seleccionat del llistat: preu, any matriculació, km....
5	Fer una Oferta	NO	Fer una oferta per un vehicle interessant.	Llistat de vehicles on cada vehicle té una columna "oferta" que donat accés a una finestra que permet introduir la oferta en €
6	Consultar Ofertes als vehicles de l'usuari	1	Consultar les ofertes que han fet pels vehicles propis	Llistat amb les ofertes fetes als vehicles de l'usuari

3.2 Model de dades

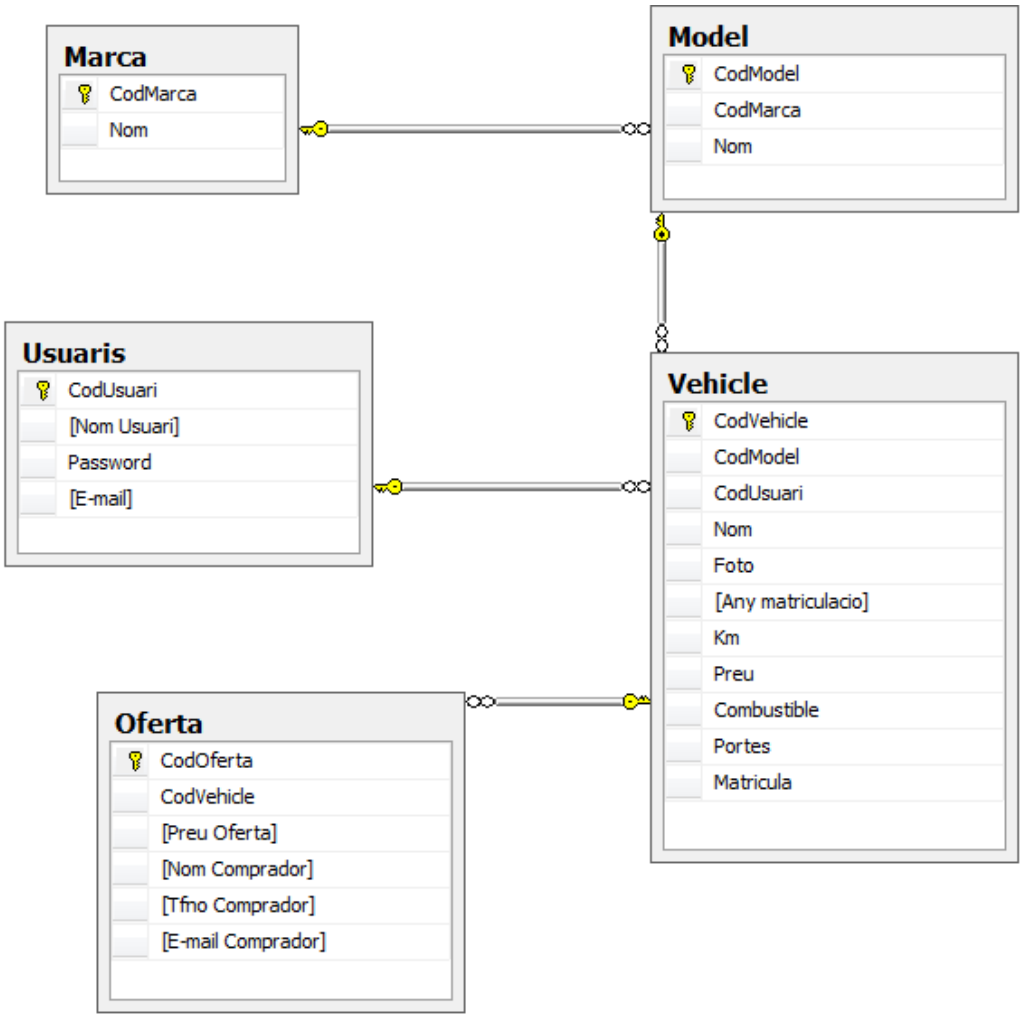
Per tal de representar la informació necessària de l'aplicació desenvolupada, s'ha creat un model entitat relació :



S'ha identificat les següents entitats:

- **Marca**: característica que diferencia un conjunt de vehicles que segueixen una mateixa línea. P.e: Seat , Renault, Opel.
- **Model**: conjunt de vehicles diferents que representen una mateixa marca. P.e: Seat Ibiza, Seat León, Seat Exeo.
- **Vehicle**: producte final que representa un determinat model d'una marca. P.e: Seat Ibiza 1.9 TDI Reference 100 cv 3p
- **Oferta**: acció on s'ofereix una quantitat per un vehicle determinat.
- **Usuari**: actor que vol vendre un o varis vehicles.

A partir del model Entitat – Relació s'ha creat l'esquema de taules següent:



En el curs del desenvolupament, va sorgir la necessitat de modificar el disseny la taula Usuaris per tal d'adaptar-la a les funcionalitats de l'inici de sessió de l'aplicació. Aquest canvis han estat el següents:

- Canviar el NIF per codUsuari per treballar amb un clau primària numèrica que fes la funció d'identitat. D'aquesta manera a cada registre d'un nou usuari, automàticament se li assigna un identificador, i per tant, no cal que introdueixi el DNI (més privacitat per l'usuari). Com a conseqüència, la taula vehicle s'ha vist afectada en el camp NIF que actuava com a clau forana cap a la taula d'usuaris. S'ha reemplaçat el camp NIF per codUsuari per poder mantenir la clau forana necessària.

- Suprimir els camps Nom Contacte i Telèfon perquè no són necessaris ja que l'usuari que es registra, és un venedor que rep ofertes i és ell qui es posa en contacte amb el possible comprador.

3.3 Interfície d'usuari

Per la visualització de l'aplicació i la seva interacció amb l'usuari s'ha creat un disseny format per un entorn comú (capçalera i menú) i una àrea de contingut que mostra la funcionalitat de la pàgina actual.

Totes les pàgines hereten aquest disseny i mostren la seva funcionalitat en l'àrea de contingut.

El menú permet la navegació per l'aplicació i té les següents opcions:

- **Iniciar Sessió**
- **Gestió de Vehicles**
- **Cerca de Vehicles**
- **Ofertes**

Pantalla Inici de Sessió

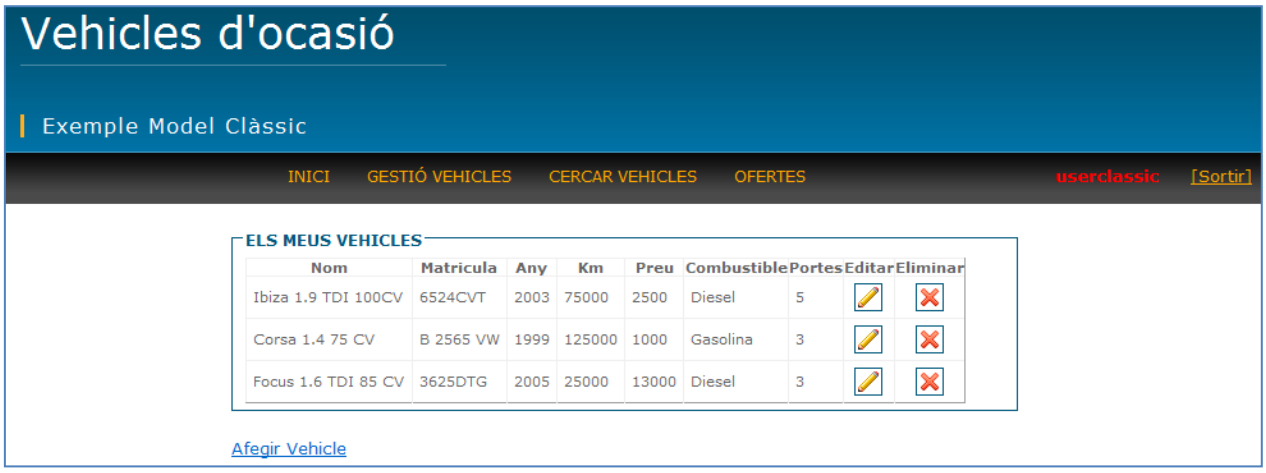
Es mostra un formulari per identificar si l'usuari és vàlid.

En cas que l'usuari necessiti registrar-se, pitjant l'enllaç "Register" accedeix a una formulari on pot introduir les seves dades.

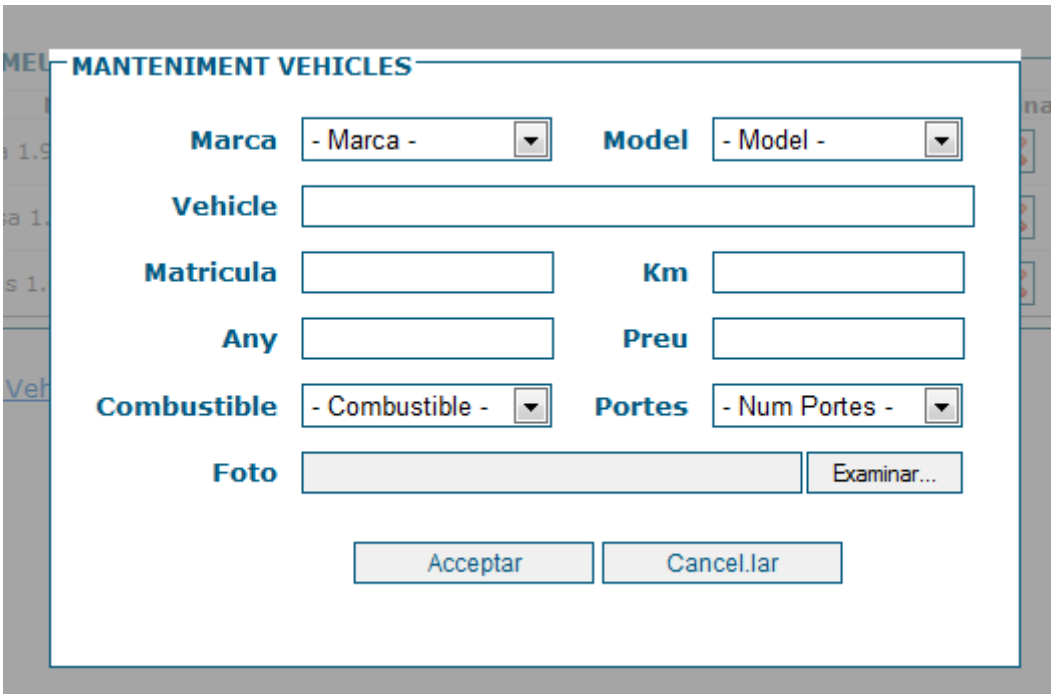
Pantalla Gestió de vehicles

En la gestió de vehicles es pot donar d'alta vehicles, modificar-los i eliminar-los.

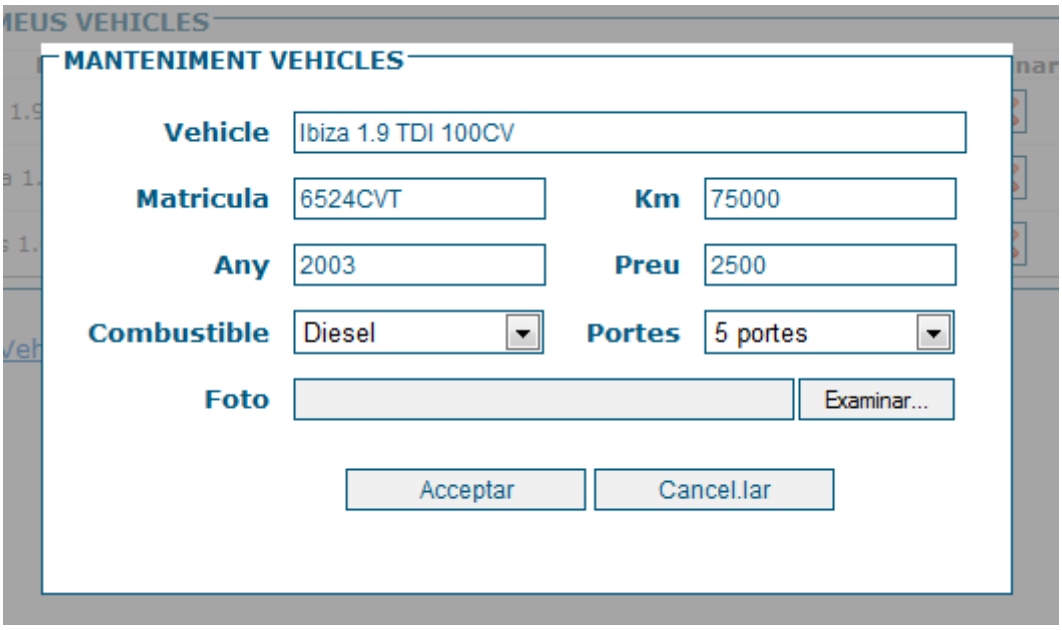
Al escollir a aquesta opció, es mostra un llistat amb els vehicles que l'usuari vol vendre.



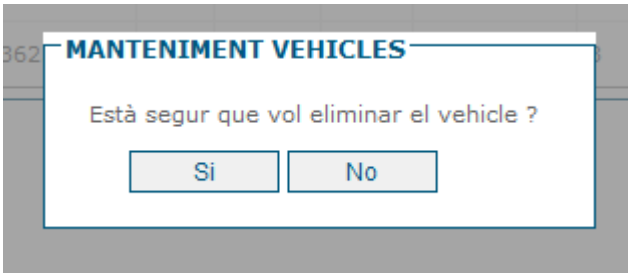
Per l'alta es mostra un formulari que permet introduir les característiques del vehicle: marca , model, preu, any matriculació.... La marca i el model s'han d'escollir de llistes desplegables.



Per la modificació, l'usuari escull un vehicle del llistat (icona llapis) i l'aplicació mostra un formulari editable amb les seves dades.



Per l'eliminació, es selecciona el vehicle a eliminar (icona X) i es mostra una confirmació de tipus "pop up" que s'haurà d'acceptar o cancel·lar.



Pantalla Cerca de Vehicles

La cerca de vehicles es presenta un filtre on es pot escollir la marca, el model , el preu (min / max), km (min / max), any matriculació, combustible i el nombre de portes.

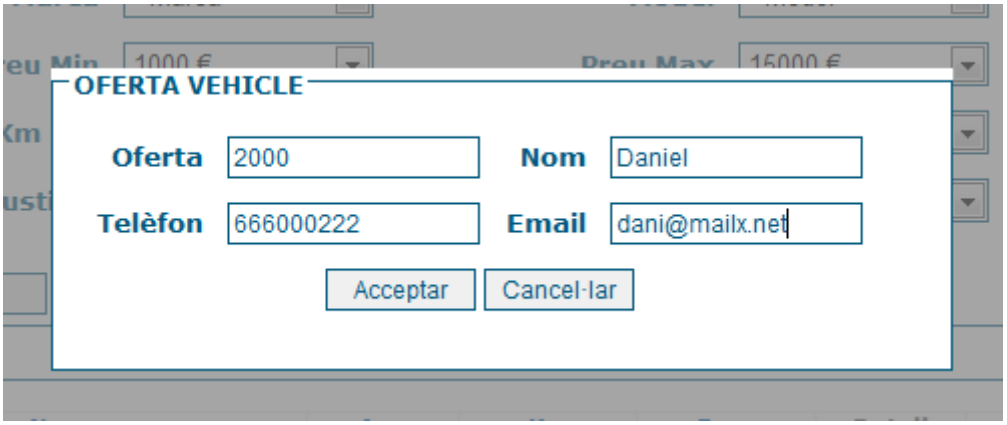
Tots aquests paràmetres són acumulatius; és a dir, tots aquells que s'informin formaran part de la consulta.

El preu i els km tindran un comportament de tipus "rang". Es cercaran els valors que estiguin dins del rang.

Un cop seleccionats els criteris de cerca en el filtre, es mostra un llistat amb els resultats amb les següents columnes: nom vehicle, foto , any matriculació,km, preu , una icona tipus "lupa" i un botó per fer una oferta.

La icona tipus “lupa” que dona accés a la fitxa del vehicle.

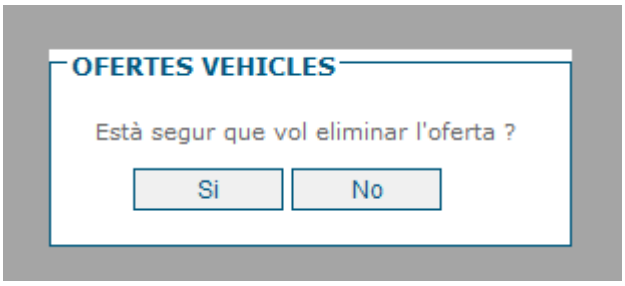
La icona oferta, que dona accés a un “pop up” on es pot introduir la quantitat a oferir.



Pantalla Ofertes

L'opció ofertes del menú, ensenya un llistat amb les ofertes que l'usuari té respecte els seus vehicles.

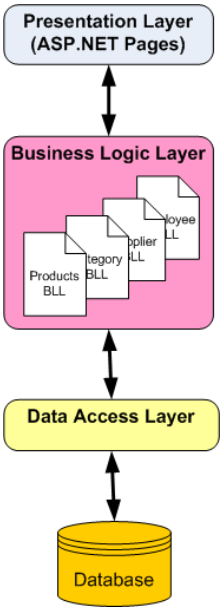
La oferta pot ser rebutjada. Al pitjar en descartar es mostra una confirmació.



4. Estudi Pràctic

4.1 Model Clàssic : descripció general.

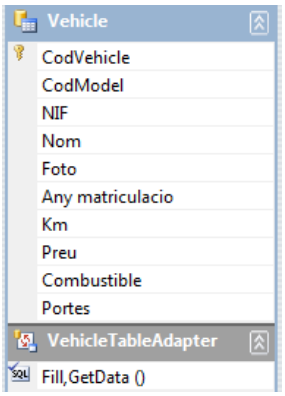
En la versió de l'aplicació utilitzant el model clàssic, s'ha definit l'accés a les dades utilitzant seguint la següent arquitectura:



S'ha creat una capa d'accés a dades (**Data Access Layer**) formada pels següents Datasets amb tipus:

- Marca
- Model
- Vehicle
- Oferta
- Usuaris

Cadascun d'aquests Datasets defineix l'estructura de la informació que conté i les operacions per poder gestionar les dades mitjançant un TableAdapter associat. P.e:



Per tal de crear la lògica de l'aplicació s'ha desenvolupat un capa de lògica (**Business Logic Layer**) que permet fer les validacions oportunes i invocar els objectes de la capa de dades per tal comunicar la capa de presentació amb les dades.

Les classes que formen aquesta capa són:

- BLMarca
- BLModel
- BLVehicle
- BLOferta
- BLUsuaris

Exemple per carregar la llista de Models des de la capa de presentació:

```
Private void carregaLlistaModels (int codMarca)
{
        BLModel oBLModel = new BLModel();
        //lògica necessària
        //Accés a la capa de dades
        ddlModels.DataSource=oBLModel.getModels(codMarca);
        ddlModels.DataTextField="nom";
        ddlModels.DataValueField="codModel";
        ddlModels.DataBind();
}
```

A partir de la classe BLModel (Business Layer) s'obté la informació per omplir la llista de models. Aquesta classe alhora utilitzarà un objecte (Data Layer) per obtenir l'accés a les dades.

```
Public ModelDataTable getModels( int codMarca)
{
        ModelTableAdapter modelAdapter=new ModelTableAdapter();
        Return modelAdapter.getModels(codMarca);
}
```

A partir de la funció getModels definida en la creació del TableAdapter es gestionaria l'accés a l'origen de dades (connexió, consulta i tancament de la connexió).

4.2 Model Clàssic : descripció específica de l'accés a dades.

L'accés a dades, per les funcionalitats de l'aplicació, s'ha estructurat de la següent manera:

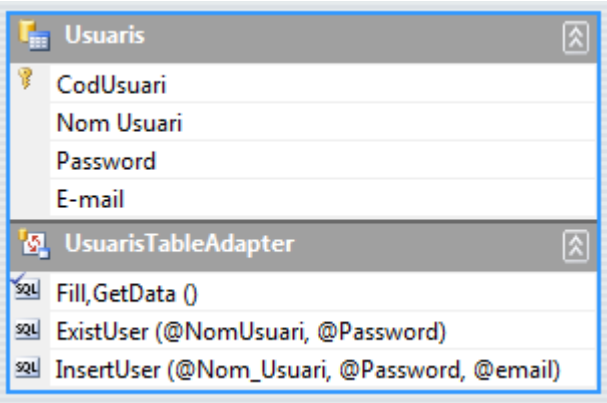
Iniciar Sessió / Registre

Capa Data Layer

S'ha creat un dataset amb tipus per representar les dades de la taula Usuaris i un TableAdapter per definir les seves operacions.

Per tal de realitzar les operacions de registrar usuaris i iniciar sessió s'han creat els mètodes:

- **InsertUser** : dona d'alta un usuari.
- **ExistUser**: comprovar si existeix un usuari.



Capa Business Layer

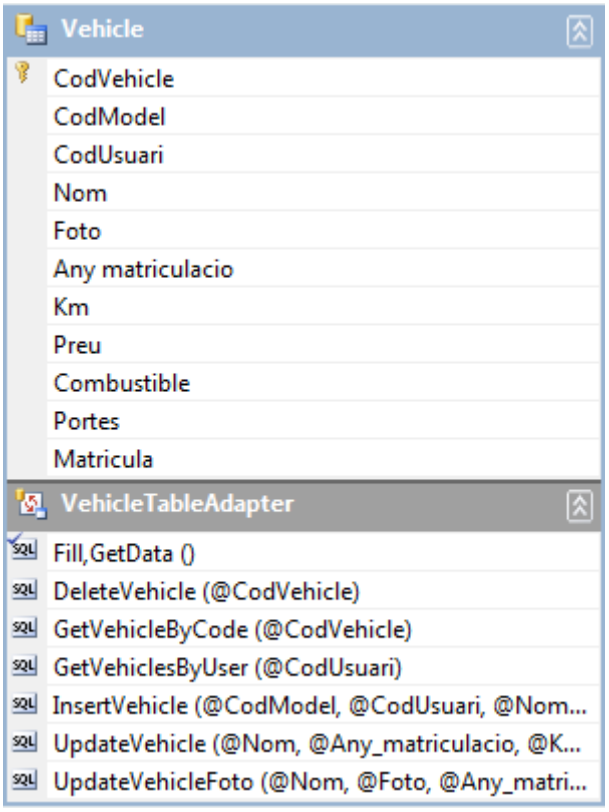
S'ha declarat una propietat global anomenada Adapter de tipus UsuarisTableAdapter que instancia l'objecte TableAdapter.

S'ha definit els mètodes:

- **AddUser**: utilitza la propietat Adapter per cridar al mètode **InsertUser** de la capa Data Layer.
- **ValidateUser**: utilitza la propietat Adapter per cridar al mètode **ExistUser** de la capa Data Layer. Si l'usuari existeix , retornar el seu codi d'usuari; en cas contrari, retorna un "0".

Operacions amb la taula vehicles: Gestió de vehicles i Cerca de vehicles

S'ha creat un Dataset amb tipus per representar les dades de la taula Vehicles i un TableAdapter per definir les seves operacions.



Gestió de Vehicles

Capa Data Layer

En el TableAdapter **VehicleTableAdapter**, s'han definit els mètodes:

- **InsertVehicle**: dona d'alta un vehicle.
- **UpdateVehicle** / **UpdateVehicleFoto**: modifica les dades d'un vehicles (segons si s'actualitza la foto o no).
- **DeleteVehicle**: dona de baixa un vehicle.
- **GetVehiclesByUser**: obté els vehicles de l'usuari actual de l'aplicació.

Capa Business Layer

S'ha declarat una propietat global anomenada Adapter de tipus **VehicleTableAdapter** que instancia l'objecte TableAdapter.

S'ha definit els mètodes:

- **AddVehicle**: utilitza la propietat Adapter per cridar al mètode **InsertVehicle** de la capa Data Layer.
- **UpdateVehicle**: utilitza la propietat Adapter per cridar al mètode **UpdateVehicle / UpdateVehicleFoto** de la capa Data Layer. És valida si l'usuari ha informat una nova foto pel vehicle o no. Per defecte, es fa una modificació de tots els camps excepte la foto.
- **DeleteVehicle**: utilitza la propietat Adapter per cridar al mètode **DeleteVehicle**.
- **GetVehiclesByUser**: utilitza la propietat Adapter per cridar al mètode **GetVehiclesByUser.**

Cerca Vehicles

Capa Data Layer

S'ha ampliat la classe parcial que defineix el TableAdapter **VehicleTableAdapter** per crear el mètode **GetVehiclesByFilter,** el qual rep per paràmetre l'expressió WHERE amb el filtre de criteris i retorna els vehicles que compleixen el filtre. Donat el caràcter dinàmic del filtre de criteris (l'usuari pot especificar 0 o més criteris a aplicar en la cerca), no s'ha pogut definir aquesta operació directament en el TableAdapter com la resta d'operacions..

Capa Business Layer

S'ha declarat una propietat global anomenada Adapter de tipus **VehicleTableAdapter** que instancia l'objecte TableAdapter.

S'ha definit el mètode:

- **GetVehiclesByFilter:** valida els paràmetres que rep per construir l'expressió WHERE i, mitjançant l'objecte Adapter, crida a la funció **GetVehiclesByFilter** passant l'expressió per paràmetre.

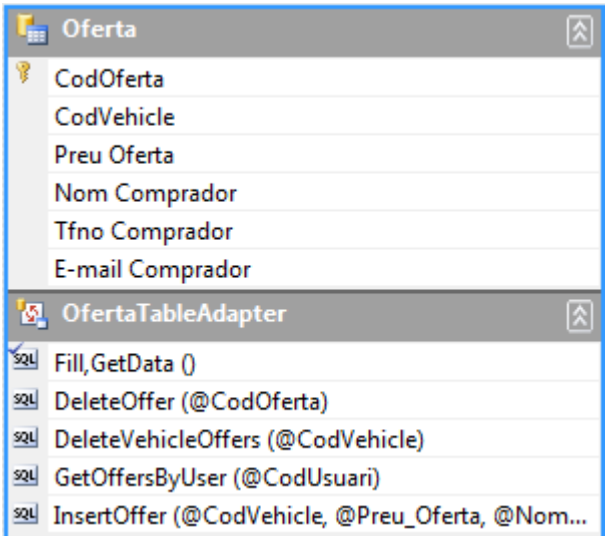
Oferta

Capa Data Layer

S'ha creat un dataset amb tipus per representar les dades de la taula Oferta i un TableAdapter per definir les seves operacions.

Per tal de realitzar les operacions relaciones amb les ofertes s'han creat els mètodes:

- **InsertOffer:** dona d'alta una oferta per un vehicle determinat.
- **DeleteOffer:** eliminar una oferta per un vehicle determinat.
- **DeleteVehicleOffers:** eliminar totes les ofertes d'un vehicle.
- **GetOffersByUser:** retornar totes les ofertes dels vehicles d'un usuari.



Capa Business Layer

S'ha declarat una propietat global anomenada Adapter de tipus **OfertaTableAdapter** que instancia l'objecte TableAdapter.

S'han definit el mètodes:

- **AddOffer**: utilitza la propietat Adapter per cridar al mètode **InsertOffer**.
- **DeleteOffer:** utilitza la propietat Adapter per cridar al mètode **DeleteOffer.**
- **DeleteVehicleOffers:** utilitza la propietat Adapter per cridar al mètode **DelelteVehicleOffers**.
- **GetOffers:** utilitza la propietat Adapter per cridar al mètode **GetOffersByUser**.

El mètode **AddOffer** és cridat des de la pantalla **Cerca Vehicles,** quan l'usuari ha fet una cerca i ha trobat un vehicle interessant.

El mètode **DeleteVehicleOffers** és cridat des de la pantalla **Gestió de Vehicles** en el moment de donar de baixa un vehicle.

Marca i Model

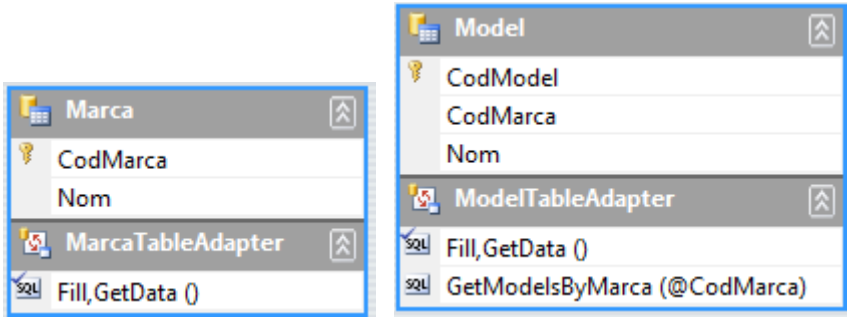
No són funcionalitats però formen part d'altres funcionalitats. En concret de les funcionalitats Cerca de Vehicles i Gestió Vehicles.

Capa Data Layer

S'ha creat els datasets amb tipus per representar les dades de les taules Marca i Model i un TableAdapter per definir les seves operacions.

Per tal de realitzar les operacions relaciones amb les ofertes s'han creat els mètodes:

- **GetData:** retorna totes les marques de la base de dades.
- **GetModelsByMarca:** retorna tots els models d'una marca.



Capa Business Layer

S'ha declarat una propietat global anomenada Adapter de tipus **MarcaTableAdapter** i **ModelTableAdapter** que instancien l'objecte TableAdapter.

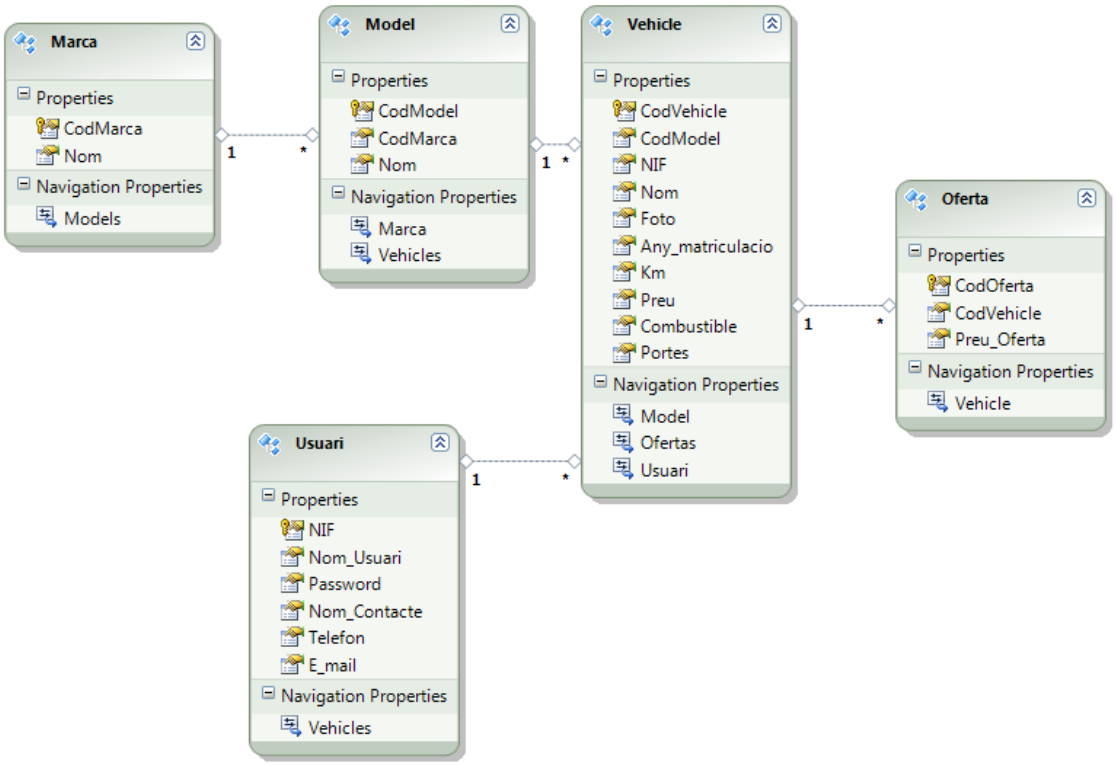
S'han definit el mètodes:

- **GetAllMarca**: a partir de la propietat Adapter, crida a la funció GetData de la capa Data Layer.
- **GetModelsByMarca**: a partir de la propietat Adapter , crida a la funció GetModelsByMarca de la capa Data Layer.

4.3 Model Entity Framework: descripció general.

En la implementació del model Entity Framework també partirem d'un model de 3 capes: presentació, lògica de negoci i accés a dades.

La capa **Data Access Layer** es definirà a partir del següent Entity Data Model:



Cada taula de la base de dades tindrà el seu mapeig en aquest EDM en forma d'entitat. Per tant, a diferència del model clàssic on teníem Datasets, ara tenim **entitats**.

La capa **Business Logic Layer** vindrà definida per una classe per entitat per tal de generar la lògica necessària i obtenir l'accés a dades a partir de les entitats.

Es crearan les següents classes:

- BLMarca
- BLModel
- BLVehicle
- BLOferta
- BLUsuaris

En el model Entity Framework donat tenim un model d'objectes (Entitats) podem utilitzar LINQ to Entities per fer les operacions contra la base de dades (concretament les entitats).

Exemple per carregar la llista de Models des de la capa de presentació:

En aquest cas, la capa de lògica podria fer el mateix que en el model clàssic:

```
Private void carregaLlistaModels (int codMarca)
{
        BLModel oBLModel = new BLModel();
        //lògica necessària
        //Accés a la capa de dades
        ddlModels.DataSource=oBLModel.getModels(codMarca);
        ddlModels.DataTextField="nom";
        ddlModels.DataValueField="codModel";
        ddlModels.DataBind();
}
```

En canvi la diferència seria en la capa d'accés a dades:

```
Public Object getModels( int codMarca)
{
        Using (var context=new BBDDVehiclesEntities)
        {
        Var models = (from m in context.Model Where m.codMarca=codMarca select
        m.CodModel,m.nom).ToList();
        Return models;
        }
}
```

Mitjançant LINQ to Entities consultem l'entitat model; el EDM que tenim ja està enllaçat amb la base de dades gràcies en Entity Framework (no cal pensar en obrir connexions, tancar-les ...).

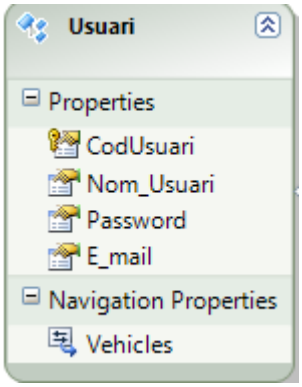
4.4 Model Entity Framework : descripció específica de l'accés a dades.

L'accés a dades, per les funcionalitats de l'aplicació, s'ha estructurat de la següent manera:

Iniciar Sessió / Registre

Capa Data Layer

En el Entity Data Model definit tenim l'entitat Usuari que permet emmagatzemar la informació dels usuaris en forma d'objecte.



Capa Business Layer

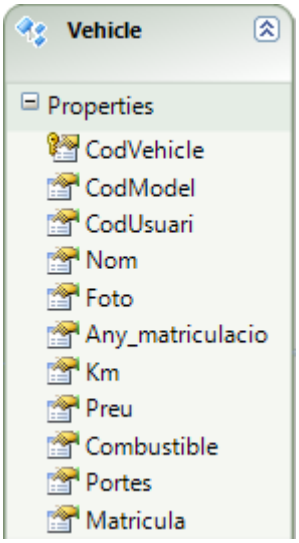
Tots els mètodes definits, creen una connexió al context del Entity Data Model definit per poder treballar amb les entitats. Les operacions es fan amb LINQ to Entities.

S'ha definit els mètodes:

- **AddUser**: crea un nou usuari declarant un objecte tipus Usuari, l'informa, l'afegeix al context i el persisteix en la base de dades.
- **ValidateUser**: verifica si existeix l'usuari. Retorna el codi d'usuari si existeix o un "0" si no existeix.

Operacions amb la taula vehicles: Gestió de vehicles i Cerca de vehicles

L'entitat Vehicle del EDM ens permet guardar la informació dels vehicles en forma d'objecte. Defineix la capa Data Layer per les operacions amb Vehicles.



Gestió de Vehicles

Capa Business Layer

Abans de realitzar qualsevol operació, tots els mètodes creen una connexió al context del Entity Data Model definit per poder treballar amb les entitats. Les operacions es fan amb LINQ to Entities.

S'ha definit els mètodes:

- **AddVehicle**: dona d'alta un vehicle instanciant un objecte de tipus Vehicle, informant les seves propietats, afegint-lo al context del EDM i guardant-lo en la base de dades.
- **UpdateVehicle**: es recupera el vehicle a editar , es modifiquen les propietats i es guarda a la base de dades.
- **DeleteVehicle**: cerca el vehicle a partir del seu codi. Si existeix, l'elimina.
- **GetVehiclesByUser**: cerca el vehicles de l'usuari a partir del codi d'Usuari.

Cerca Vehicles

Capa Business Layer

Abans de realitzar qualsevol operació, tots els mètodes creen una connexió al context del Entity Data Model definit per poder treballar amb les entitats. Les operacions es fan amb LINQ to Entities.

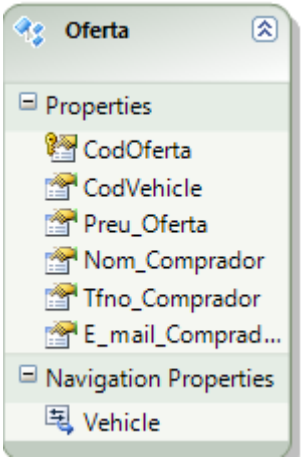
S'ha definit el mètode:

- **GetVehiclesByFilter:** recupera els vehicles de la base de dades en una llista i va validant els paràmetres que rep per anar filtrant-los sobre la llista, per obtenir els vehicles segons els criteris introduïts per l'usuari.

Oferta

Capa Data Layer

En el Entity Data Model tenim l'entitat Oferta que permet emmagatzemar la informació de les ofertes en forma d'objecte.



Capa Business Layer

Abans de realitzar qualsevol operació, tots els mètodes creen una connexió al context del Entity Data Model definit per poder treballar amb les entitats. Les operacions es fan amb LINQ to Entities.

S'han definit el mètodes:

- **AddOffer**: dona d'alta una oferta creant un objecte de tipus Vehicle, informant les seves propietats, afegint-lo al context del EDM i guardant-lo en la base de dades.
- **DeleteOffer:** cerca l'oferta a partir del seu codi. Si existeix, l'elimina.
- **DeleteVehicleOffers:** cerca totes les ofertes d'un vehicle a partir del seu codi i les elimina.
- **GetOffers:** obté totes les ofertes dels vehicles d'un usuari. Fa una "join" de les entitats Oferta i Vehicles del context a partir del codi de vehicle i ho filtra pel codi d'usuari.

El mètode **AddOffer** és cridat des de la pantalla **Cerca Vehicles,** quan l'usuari ha fet una cerca i ha trobat un vehicle interessant.

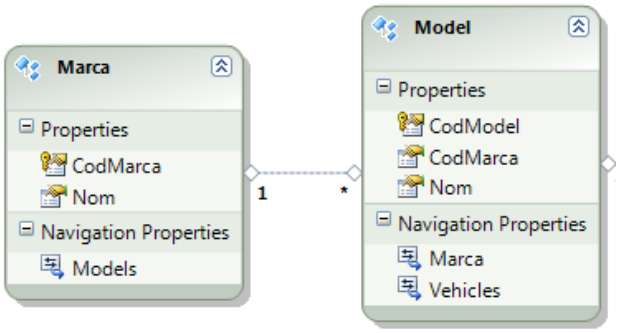
El mètode **DeleteVehicleOffers** és cridat des de la pantalla **Gestió de Vehicles** en el moment de donar de baixa un vehicle.

Marca i Model

No són funcionalitats però formen part d'altres funcionalitats. En concret de les funcionalitats Cerca de Vehicles i Gestió Vehicles.

Capa Data Layer

En el Entity Data Model tenim les entitats Marca i Model que permeten guardar la informació de les Marques i Models dels vehicles en forma d'objecte



Capa Business Layer

Abans de realitzar qualsevol operació, tots els mètodes creen una connexió al context del Entity Data Model definit per poder treballar amb les entitats. Les operacions es fan amb LINQ to Entities.

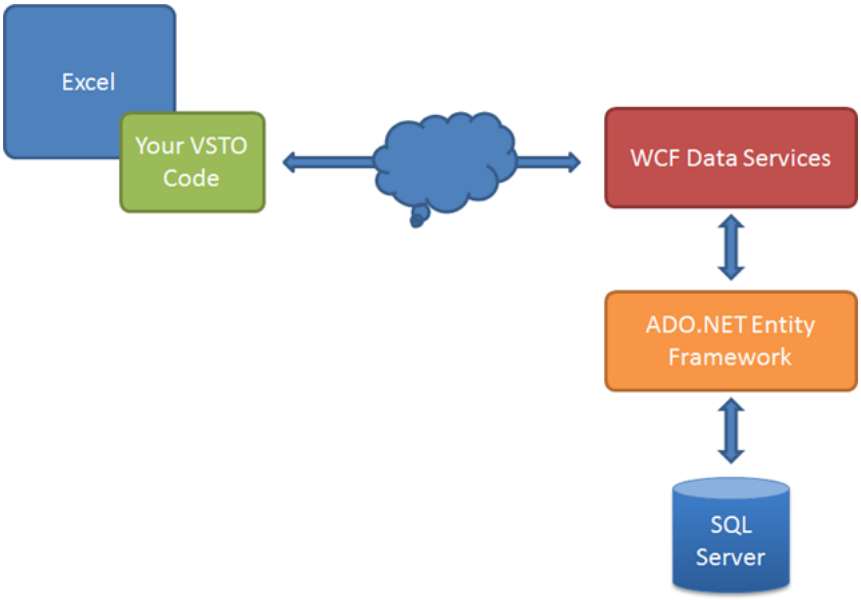
S'han definit el mètodes:

- **GetAllMarca**: obté un llistat de totes les marques de vehicles de la base de dades.
- **GetModelsByMarca**: obté un llistat amb tots els models d'una marca.

4.5 Model OData: descripció general.

La implementació del model OData es basarà en la implementació Entity Framework amb la característica que s'haurà d'exposar el EDM a través d'un servei (WCF DataSevice) per poder ser consumit a través de consultes HTTP.

Per tant, tindrem l'escenari següent:



El consum de les dades a través del servei **Proxy amb LINQ to Entities** ens permet fer el mateix que jquery a un nivell superior d'abstracció , treballant amb objectes enlloc de codi javascript des de el costat client.

Exemple LINQ to Entites

```
Private BBDDVehiclesEntities vehiclesEntities;
vehiclesEntities =new BBDDVehiclesEntities ( new
Uri(""http://localhost/ServeiExemple.svc"));
var query = (from vehicle in vehiclesEntities.vehicle where vehicle.codi=1
select vehicle.name,vehicle.preu) as DataServiceQuery<vehicle>;
// lògica de la funció
```

4.6 Model OData: descripció específica de l'accés a dades.

En aquest model, també treballem mitjançant un Entity Data Model i LINQ to Entities. Les funcionalitats ha estat definides com en el model Entity Framework amb algunes particularitats. Les principals diferències entre els models han estat:

- **Capa Data Layer : Accés a les dades**

Aquesta està formada per un EDM el qual està publicat en un servei WCF Data Service.

- **Capa Business Layer : Accés a les dades**

L'accés al Entity Data Model es fa mitjançant un proxy (referència web).

Exemple funció GetVehicleByCode de la classe BLVehicle:

```
Vehicle selectedVehicle = null;
BBDDVehiclesEntities db = null;

try
{
    db = new BBDDVehiclesEntities(new
  Uri(Properties.Settings.Default.ServiceURI));

  selectedVehicle = (from v in db.Vehicles
                     where (v.CodVehicle == CodVehicle)
                     select v).First();

  return selectedVehicle;

}
catch (Exception ex)
{
    throw ex;
}
```

Al crear l'objecte que ens dona accés al context del EDM (db), es passa en el constructor la URI del servei.

- **Capa Business Layer : Operacions no permeses.**

He comprovat que hi ha operacions permeses amb el model Entity Framework que no ho són amb el model OData. Per exemple:

- Join entre entitats:

En Entity Framework es pot directament:

```
OfertesVehiclesUsuari = (from o in db.Ofertas

                        from v in db.Vehicles

                          where  ( o.CodVehicle ==
v.CodVehicle) && (v.CodUsuari==codUsuari)

                        select o).ToList();
```

En OData s'ha de fer entre llistes :

```
          List<Oferta> OfertesVehicles = null;
   List<Vehicle> VehiclesUsuari = null;
   List<Oferta> OfertesVehiclesUsuari = null;

VehiclesUsuari = db.Vehicles.Where(v =>v.CodUsuari==  codUsuari ).ToList();
        OfertesVehicles = db.Ofertas.ToList();


           var query = from v in VehiclesUsuari
           join o in OfertesVehicles

          on v.CodVehicle equals o.CodVehicle

          select o;

        OfertesVehiclesUsuari = query.ToList();
```

- Cercar un element (p.e. una oferta):

En Entity Framework es pot utilitzat la funció First de l'entitat:

```
Oferta deleteOferta = db.Ofertas.First(v => v.CodOferta == codOferta);
```

En OData s'ha de fer una query per després aplicar First:

```
Oferta deleteOferta = (from o in db.Ofertas

          where (o.CodOferta == codOferta)

          select o).First();
```

4.7 Observacions Estudi pràctic.

Per realitzar l'estudi pràctic , s'ha dut a terme una tasca que no formava part de l'objectiu però si que era necessària per la funcionalitat de l'aplicació. Aquesta tasca ha estat crear un proveïdor d'accessos a mida per poder utilitzar la taula usuaris com a repositori per dona d'alta usuaris i validar la seva existència (fer el login).

Per defecte, visual studio 2010, al crear una aplicació web ofereix un exemple per la gestió del control d'usuaris a l'aplicació. Utilitza un proveïdor (**AspNetSqlMemberShipProvider**) el qual utilitza la base de dades **aspnetdb.mdf**. Donat que en el model de dades creat, hi ha una taula d'usuaris per gestionar l'accés a l'aplicació (entre altres coses), s'ha reemplaçat aquest proveïdor per un a mida.
La nova classe creada has estat **AccessMembershipProvider** la qual hereta de **MembershipProvider**. Només ha calgut definir els mètodes **CreateUser** i **ValidateUser**.

En el mètode **CreateUser**, s'ha utilitzat el mètode **AddUser** definit en la capa Business Layer. Un cop donat d'alta l'usuari, s'ha guardat en sessió el codi d'usuari.
En el mètode **ValidateUser,** s'ha cridar el mètode ValidateUser de la capa Business Layer. Si l'usuari existeix, s'ha guardat en sessió el seu codi.

5. Comparativa entre els diferents models d'accés a dades

A partir de l'estudi realitzat i les versions desenvolupades, he pogut analitzar les diferències entre les tecnologies. Enumerant les característiques més significatives:

1. **Creació de l'esquema de classes per l'accés a dades**
    a. **Model Clàssic:** a partir d'una connexió amb la base de dades, creació de datasets amb tipus mitjançant drag and drop. Cal definir les operacions contra la base de dades de manera manual utilitzant el llenguatge SQL.

    b. **Entity Framework:** creació del Entity Data Model de manera automàtica a partir d'una connexió amb la base de dades. No cal definir les operacions prèviament ja que es pot treballar amb tecnologies com LINQ to Entities que permeten fer-les a partir d'objectes.

    c. **ODATA:** igual que Entity Framework només que cal haver exposat el EDM mitjançant un servei.

2. **Tolerància a canvis en el model de dades.**
    a. **Model Clàssic:** cal modificar manualment els datasets afectats pels canvis i les operacions (SELECT, INSERT, DELETE, UPDATE…).
    b. **Entity Framework:** la possibilitat de regenerar el model automàticament fa que es disposi del model de classes actualitzat al moment. No s'ha de reescriure cap operació a priori. Només caldria modificar alguna sentència si hagués algun camp modificat que es veies afectat.
    c. **ODATA:** igual que Entity Framework . Republicar el servei que exposa el EDM.

3. **Tolerància a migracions del motor de base de dades**
    a. **Model Clàssic:** cal refer tot l'esquema de classes i totes les operacions per adaptar-les a les particularitats del llenguatge SQL del nou motor de base de dades.
    b. **Entity Framework:** només cal crear una connexió amb la nova base de dades i refer el EDM. El codi existent que utilitzava el EDM anterior, es pot aprofitar ja que no cal refer-lo (treball amb objectes independent del llenguatge SQL).
    c. **ODATA:** igual que EntityFramework . Republicar el servei que exposa el EDM.

4. **Escalabilitat: fer créixer el model de dades**
    a. **Model Clàssic:** qualsevol ampliació del model, implica crear els Datasets necessaris i les operacions implicades de manera manual.
    b. **Entity Framework :** regenerant el EDM ja tenim al moment les ampliacions. No cal refer operacions.
    c. **ODATA:** igual que Entity Framework . Republicar el servei que exposa el EDM.

5. **Disponibilitat del model en entorns distribuïts**
    a. **Model Clàssic:** creació de serveis web que exposin les operacions permeses per explotar el model de dades. Aplicacions d'altres plataformes poden consumir el servei però no disposen del model de dades lliurement. Dependència de la programació si el model creix (pot incidir negativament en l'escalabilitat).
    b. **Entity Framework :** depèn d'una altra tecnologia (OData) per poder exposar el EDM.
    c. **ODATA:** exposar el EDM mitjançant un servei de manera que el model de dades està disponible, com si fos una base de dades local, per aplicacions d'altres sistemes independentment de la seva tecnologia.

Després del treball realitzat, veig en Entity Framework un model condemnat a substituir el model clàssic . Només he trobat que beneficis en la seva utilització i , a més, si es disposa de la capacitat per oferir el model de dades a altres sistemes mitjançant una tecnologia com OData, clarament complementària a Entity Framework, crec que tot nou desenvolupament hauria de plantejar-se amb aquesta tecnologia i caldria avaluar la conveniència de fer l'esforç de migració d'aplicacions amb el model clàssic cap al model Entity Framework.

6. Conclusions de les tecnologies d'accés a dades

Un cop realitzat l'estudi teòric de les tecnologies, i experimentat la seva implementació en un cas pràctic, he pogut comprovar gran part de les seves avantatges i desavantatges.

El **model clàssic** aporta un conjunt d'avantatges:

- **Interoperabilitat**: ADO.NET aprofita la flexibilitat de XML com a format de transmissió de Datasets. Qualsevol component o aplicació que pugui llegir XML podrà operar amb les dades.
- **Mantenibilitat:** la facilitat per comunicar capes mitjançant XML (Datasets) fan que afegir capes en aplicacions per distribuir i reduir la càrrega del sistema sigui un tasca viable.
- **Programabilitat:** l'ús de Datasets amb tipus simplifica l'escriptura de codi i informa d'errors en temps de compilació.
- **Escalabilitat:** ADO .NET utilitza accés desconnectat a les dades per tal de no boquejar connexions amb bases de dades i estalviar recursos (servei a gran nombre d'usuaris).

Des de el punt de vista del desenvolupament, he comprovat que aquest model aporta aquest beneficis,sobretot pel fet de treballar amb el gran protagonista del model: **el dataset**.

Poder tenir objectes que representen la mateixa estructura de les taules que tenim a la bases de dades (datasets), i oferir una organització on cada element que el forma té un tipus associat (DataTable, DataRow, DataColumn ...) en forma d'objecte, facilita molt la interacció amb la font de dades.

A més a més, aprofitar la possibilitat d'associar objectes TableAdapter als datasets per poder definir les operacions a realitzar, fan que amb poques línies de codi podem definir una arquitectura de capes per accedir a l'origen de dades.

En el meu exemple, el model de dades era senzill i per tant he pogut treballar amb poques dificultats alhora d'implementar aquest model però he vist que en un projecte de major envergadura, amb un model de dades més complexa, s'hauria d'haver desenvolupat una arquitectura de classes, seguint la filosofia del model, però adaptada a les necessitats. Aquest fet implica un cost a nivell de disseny i desenvolupament.

La màxima dificultat i els inconvenients que he trobat en la utilització d'aquest model han estat els següents:

- Qualsevol canvi en la base de dades, implica un canvi manual en els datasets afectats i les operacions associades. El manteniment estructural queda afectat per aquesta característica.
- Dependència d'un llenguatge com SQL per definir les operacions. Tenim programació a nivell d'objectes i programació a nivell de base de dades. Aquest aspecte afecta negativament al manteniment i a un possible canvi del motor de base de dades (migració de tot el codi SQL).

- Operacions amb condicions dinàmiques més difícils d'implementar: les operacions definides als table dapters associats ,mitjançant un assistent, són de caire senzill i directe. És a dir, són operacions per retornar registres filtrats amb condicions definides de manera estàtica.

Quan he hagut de filtrar amb condicions variables, segons un conjunt de criteris opcionals, he hagut d'extendre la classe parcial associada al table adapter per poder indicar les condicions de forma dinàmica.

Ex: funció GetVehiclesByFilter de la classe VehicleTableAdapter

```
public VehicleDS.VehicleDataTable GetVehiclesByFilter(string WhereExp)
 {
     string strSelect = string.Empty;

     try
     {
         strSelect = this._commandCollection[0].CommandText;
         if (WhereExp != string.Empty)
               this._commandCollection[0].CommandText += " WHERE " + WhereExp;
       VehicleDS.VehicleDataTable dataTable = new
VehicleDS.VehicleDataTable();
         this.Fill(dataTable);
         return dataTable;
     }
     catch (System.Exception ex)
     {
         throw ex;
     }
     finally
     {
         this._commandCollection[0].CommandText = strSelect;
     }
 }
```

Finalment, comentar a títol personal, que aquest model és millorable però que encara té cabuda en els projectes actuals degut a gran quantitat d'aplicacions estan funcionant , que han estat desenvolupades seguint aquest model i que tenen necessitats de manteniment.

El **model Entity Framework** l'he entès com una evolució del model clàssic. Pel que he vist, segueix amb la idea de tenir una representació de l'origen de dades en forma de classes a nivell estructural però independent de l'origen de dades (motor de base de dades). Només treballem amb **objectes**.

Les avantatges que ens aporta:

- **Reduir el temps de desenvolupament**: aquest framework encapsula la gestió de les dades de manera que els desenvolupadors poden dedicar-se al aspectes funcionals de l'aplicació.
- **Treballar només amb objectes**: "el desenvolupador només pensa amb objectes". Treballa a un nivell superior d'abstracció on només tenim objectes
- **Independència de codi específic al motor de base de dades**: les aplicacions són lliures de codi "hardcoded" específic per a un motor de base de dades gràcies al model d'objectes
- **Mapeig automàtic:** el mapeig del model d'objectes amb el model dades s'actualitza als canvis sense haver de modificar el codi font**.**
- **LINQ amb Intellisense:** aportar intellisense i validació de sintaxis en temps de compilació al llenguatge de consultes utilitzat contra el model conceptual.

Dintre de les opcions que hi ha per poder treballar amb els objectes i fer les operacions, he escollit LINQ to Entities perquè he vist que s'adaptava molt a tasques directes contra entitats.

Aprofitar la potència de LINQ per consultar entitats, m'ha fet evitar nombroses línies de codi recorrent estructures amb sentències per obtenir la informació, i he estat capaç d'obtenir els resultats de manera ràpida i directe, minimitzant l'ús d'objectes temporals.

La principal dificultat que he trobat ha estat començar a utilitzar aquest model ja que estava acostumat a treballar amb el model clàssic. A més, també he hagut d'aprendre LINQ per treballar amb les entitats. Un cop superat aquest petit inconvenient, he gaudit dels beneficis del model i he estat capaç de migrar la versió del model clàssic cap a Entity framework sense grans problemes.

Sobretot vull destacar dos conceptes que m'han fascinat: independència del motor de base de dades i la auto regeneració del model davant canvis en el model de dades. L'estalvi de temps i maldecaps que ens aporten, incideixen molt positivament en el desenvolupament del projecte i en el seu resultat: el producte final.

Crec, modestament, que tot projecte que es comencés amb tecnologia .NET hauria de considerar i apostar per aquest model d'accés a dades deixant de banda el model clàssic.

El **model OData** l'he entès com un complement del model Entity Framework a nivell de disponibilitat. És a dir, Entity Framework suposa un gran avenç i OData posa aquest avenç a l'abast de tothom, independentment de qui vulgui utilitzar-ho (plataformes lliures : PHP, mòbils: Windows Phone , Aplicacions Webs ...).

Per tant, ser capaç d'oferir un origen de dades, explotable mitjançant Entity Framework, que es pugi consumir des de múltiples tecnologies, obre el món a la interconnexió entre sistemes independents que, per necessitats de negoci, col·laboren per créixer i crear un sistema d'informació més potent i complert.

Fins fa poc, aquesta interconnexió era possible gràcies als Web Services però aquest tenien un sèrie de limitacions:

- El Web Service ens indica que podem fer i com ho hem de fer.
- Noves necessitats implica reprogramar el Web Service: necessitats de manteniment, perill d'un creixement exponencial que perjudiqui l'estabilitat del servei, acords / negociacions entre les parts implicades....

OData està oferint l'origen de dades directament, com si el tinguéssim en una base de dades local al nostre computador. Certament, hi ha algunes limitacions però no és cap bogeria el que acabem d'afirmar.

Els objectius del model OData per poder oferir els orígens de dades són:

- Crear un model uniforme de representació de dades estructurades a través de JSON i Atom.
- Utilitzar convencions URL uniformes per la navegació, filtrat, ordre i paginació de dades.
- Crear operacions estandarditzades de les accions GET,POST,PUT i DELETE per la consulta i manipulació de dades.

Alhora de desenvolupar la versió OData, em va donar la impressió que la podria obtenir ràpidament agafant com a referència la versió Entity Framework. Veia que si era capaç de publicar l'origen de dades mitjançant un servei (WFC Data Service), el consum de les dades seria pràcticament igual com ho feia en la versió Entity Framework. Aquesta impressió no estava desencaminada però vaig tenir que fer algunes modificacions ja que vaig trobar operacions fetes amb LINQ to Entities que no eren permeses amb el model OData.

Destacaré dos casos:

- Obtenir un vehicle a partir del seu codi:
  - amb Entity Framework he feia a partir del mètode First de l'entitat

```
Vehicle updateVehicle = db.Vehicles.First(v => v.CodVehicle == codVehicle);
```

  - OData no permet utilitzar First de l'entitat però si el First d'una consulta

```
selectedVehicle = (from v in db.Vehicles
where (v.CodVehicle == CodVehicle)    select v).First();
```

- Fer una join entre dues entitats per obtenir totes les ofertes dels vehicles d'un usuari:
  - Amb EF

```
OfertesVehiclesUsuari =(from o in db.Ofertas

from v in db.Vehicles where (o.CodVehicle == v.CodVehicle) && (v.CodUsuari==codUsuari) select o).ToList();
```

  - Amb OData vaig tenir que guardar els resultats en llistes i després fer la join

```
VehiclesUsuari = db.Vehicles.Where (v => v.CodUsuari == codUsuari).ToList();

OfertesVehicles = db.Ofertas.ToList();

var query = from v in VehiclesUsuari
                    join o in OfertesVehicles
                    on v.CodVehicle equals o.CodVehicle
                    select o;


 OfertesVehiclesUsuari = query.ToList();
```

Per acabar les conclusions, comentar que , al igual que Entity Framework, veig OData com una tecnologia a considerar sèriament sobretot en entorns distribuïts i anar de la mà amb Entity Framework.

7. Millores del producte entregat

Per l'estudi pràctic es va decidir realitzar una aplicació web que simulava un petit portal per vendre / comprar vehicles d'ocasió. Al estudiar tres models d'accés a dades, es va creure convenient desenvolupar tres versions de la mateixa aplicació per poder comparar els models a nivell de desenvolupament . Per aquest motiu, es va donar més pes a la implementació de les funcionalitats d'accés a dades deixant en segon pla l'aspecte funcional de l'aplicació. Un cop finalitzat l'estudi , s'han detectat unes "mancances" que , d'haver disposat més temps, s'haurien implementat. No obstant, les comentaré:

1. **Sistema missatges entre comprador i venedor.**
   L'aplicació no permet fer un seguiment del contacte entre comprador – venedor.
   Permet que el comprador faci una oferta i que el venedor la visualitzi però no es registra un "feedback" de les negociacions entre comprador / venedor. Per aquest motiu, només el venedor està obligat a registrar-se en l'aplicació per poder oferir els seus vehicles.

2. **Millora en la pantalla d'ofertes.**
   El venedor consulta les ofertes que té però el llistat que veu és molt senzill. A més, caldria agrupar les ofertes per vehicle (si el venedor té més d'un vehicle a la venda) de manera que el creixement del nombre d'ofertes no faci que tinguem un llistat molt gran i de difícil consulta.

3. **Millora en el detall del vehicle**
   La pantalla de mostra el detall del vehicle informa de les característiques mínimes. Crec que es podria haver ampliat per donar-li un aire més atractiu afegint una secció de comentaris perquè l'usuari expressi el que vulgui sobre el vehicle.

4. **No permetre fer ofertes per un vehicle propi.**

   Un venedor també pot cercar vehicles, i pot trobar el seus propis vehicles. Podria arribar a fer una oferta per un vehicle seu i això seria un error funcional. Segurament, s'hauria d'haver corregit aquest aspecte en la versió entregada però al no haver creat una gestió de rols on identificar usuaris compradors i venedors, s'ha deixat de banda aquesta incidència per dedicar més temps en la resolució d'aspectes de l'accés a dades de cadascuna de les versions.

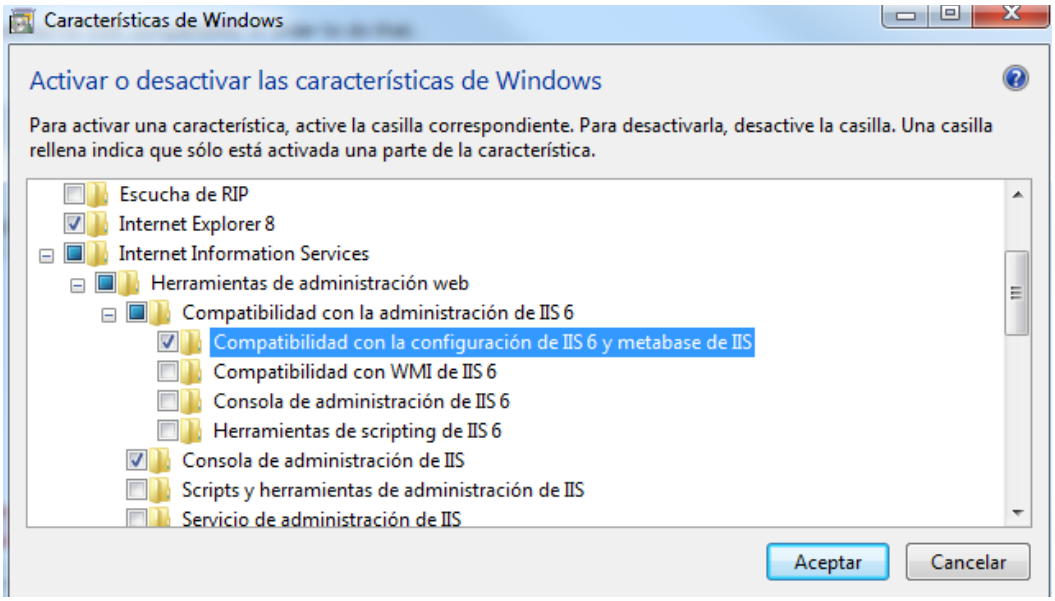
5. **Millores interfície.**

   La interfície d'usuari és millorable. S'ha entregat un entorn presentable però crec que es podria haver donat un aspecte més atractiu ja que l'objectiu del portal es vendre i cal que l'usuari experimenti una sensació el més agradable possible. Tecnologies com Silverlight haurien d'estar presents en una aplicació d'aquest tipus.

8. Instal·lació del producte

El procés d'instal·lació ha estat testejat en un entorn Windows 7 ,IIS 7.0 i .NET Framework 4.0.

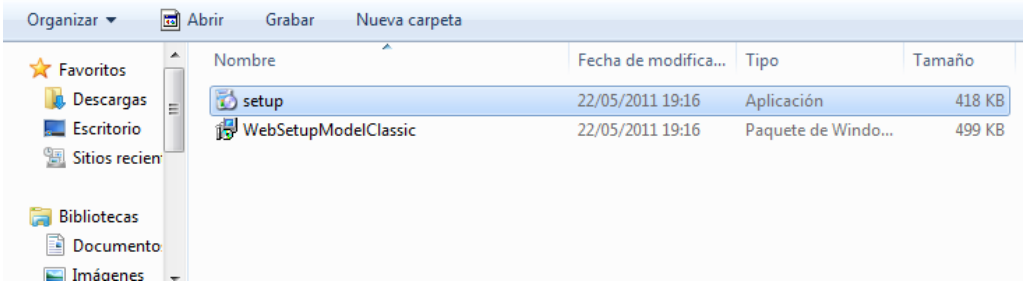
Abans de realitzar la instal·lació de les versions , cal verificar que tenim instal·lada la "compatibilidad con la configuración de IIS 6 y metabase de IIS"



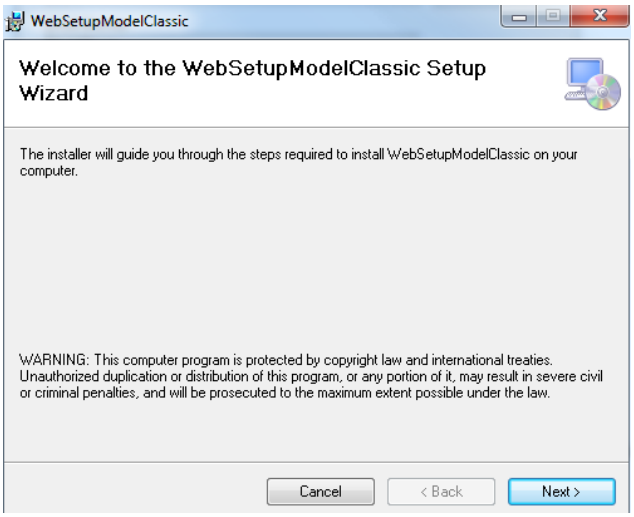
Per fer-ho cal anar al "Panel de Control → Programas → Activar / Desactivar características de Windows" i marcar la opció.

8.1 Model Clàssic

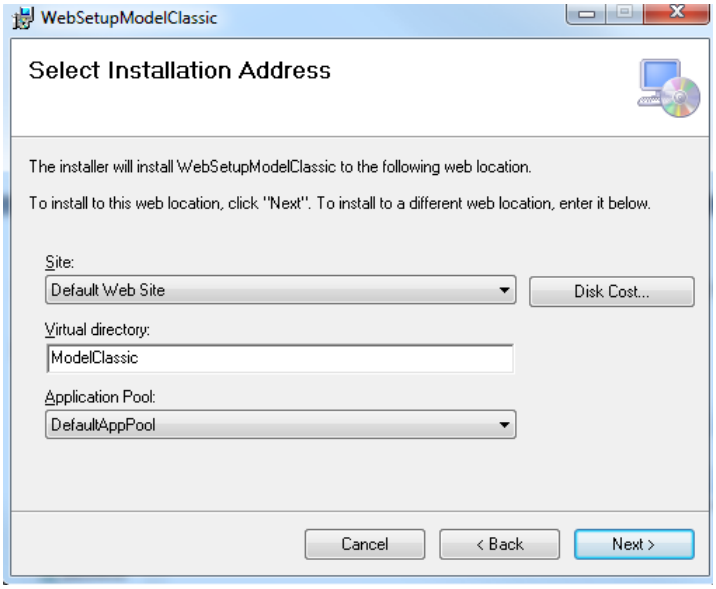
Executar el fitxer d'instal·lació (setup.exe):



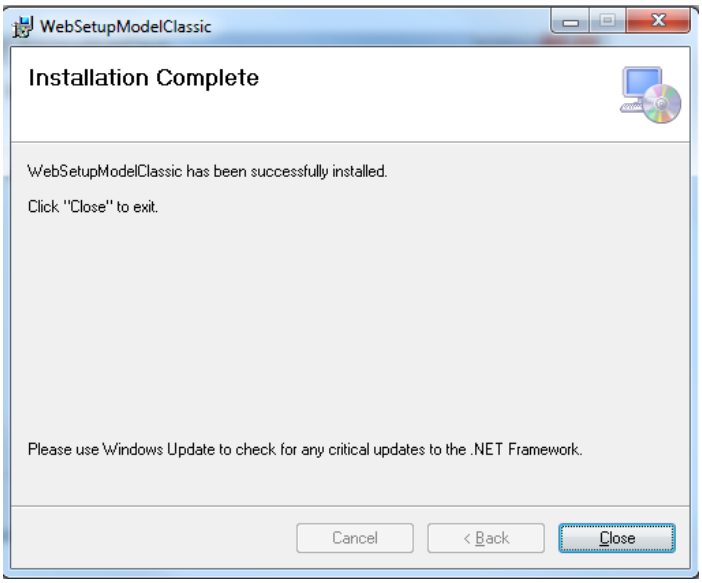
Seguir l'assistent d'instal·lació:



Indicar el nom del directori virtual : **ModelClassic** i continuar.



Esperar la finalització de l' instal·lació:

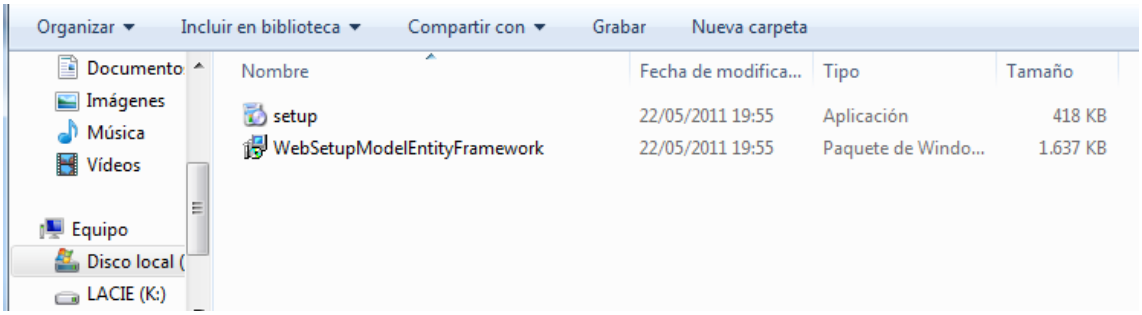


Verificar la correcta instal·lació de la versió:

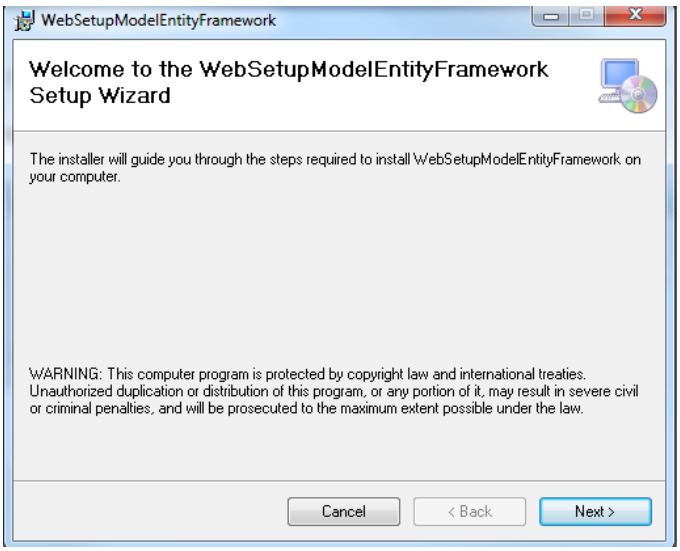
http://localhost/ModelClassic/Home.aspx

8.2 Model Entity Framework

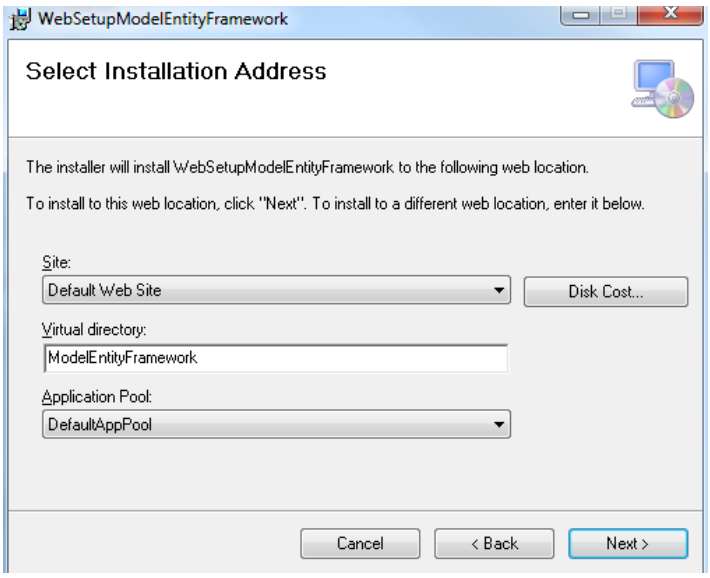
Executar el fitxer d'instal·lació (setup.exe):



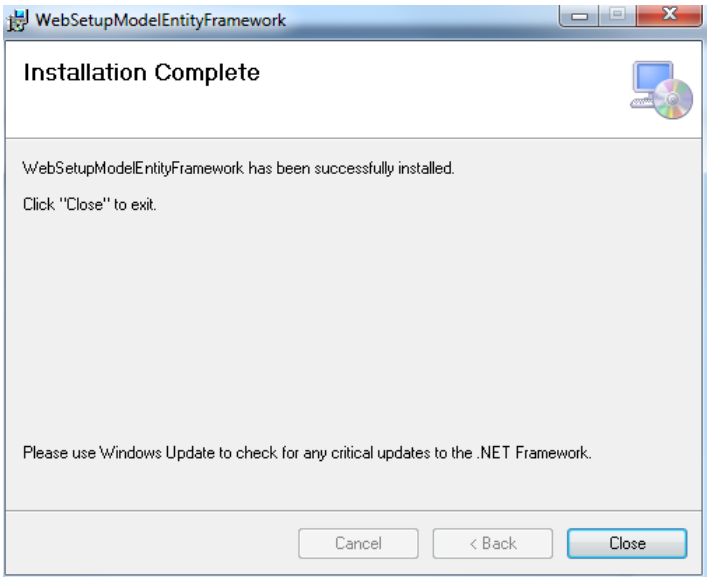
Seguir l'assistent:



Indicar el nom del directori virtual : **ModelEntityFramework** i continuar.



Esperar la finalització de l' instal·lació:

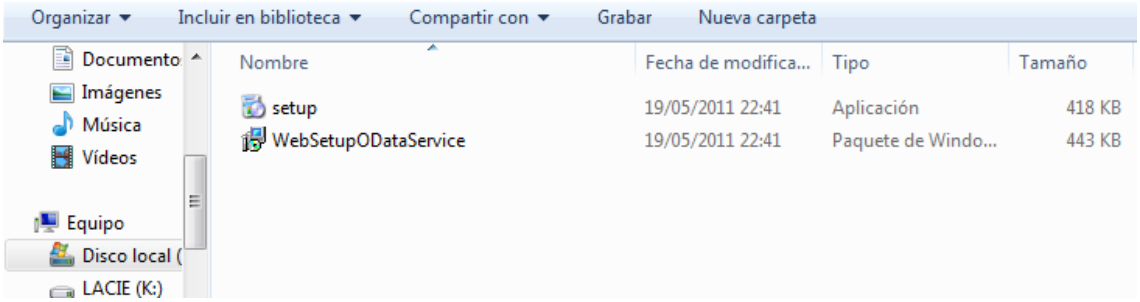


Comprovar la correcta instal·lació de la versió:

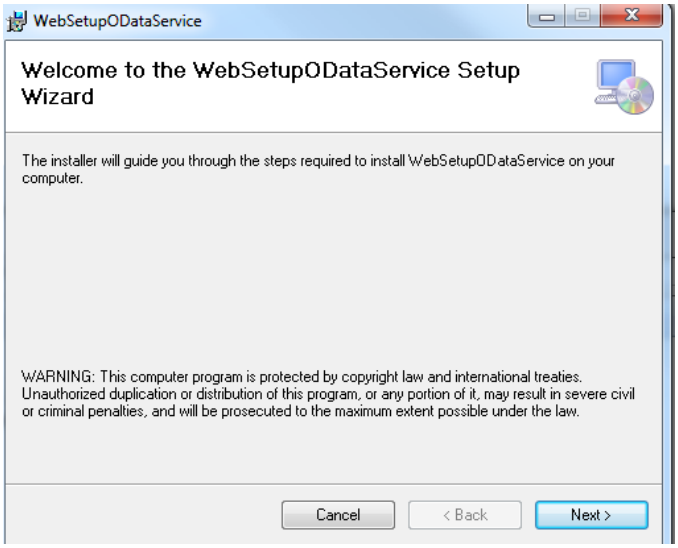
http://localhost/ModelEntityFramework/Home.aspx

8.3 Model OData - Servei

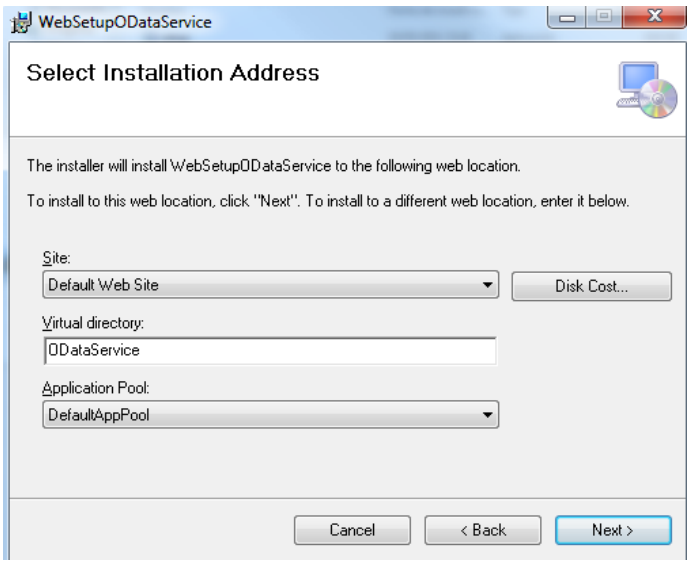
Executar el fitxer d'instal·lació (setup.exe):



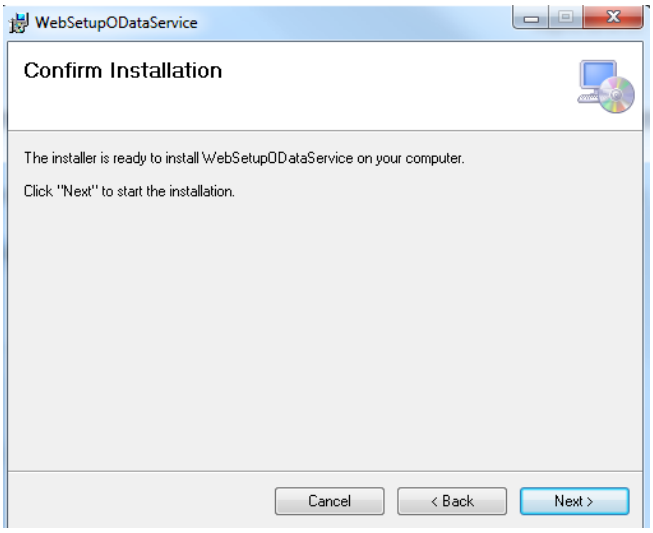
Segui l'assistent:



Indicar el nom del servei: **ODataService** i continuar.

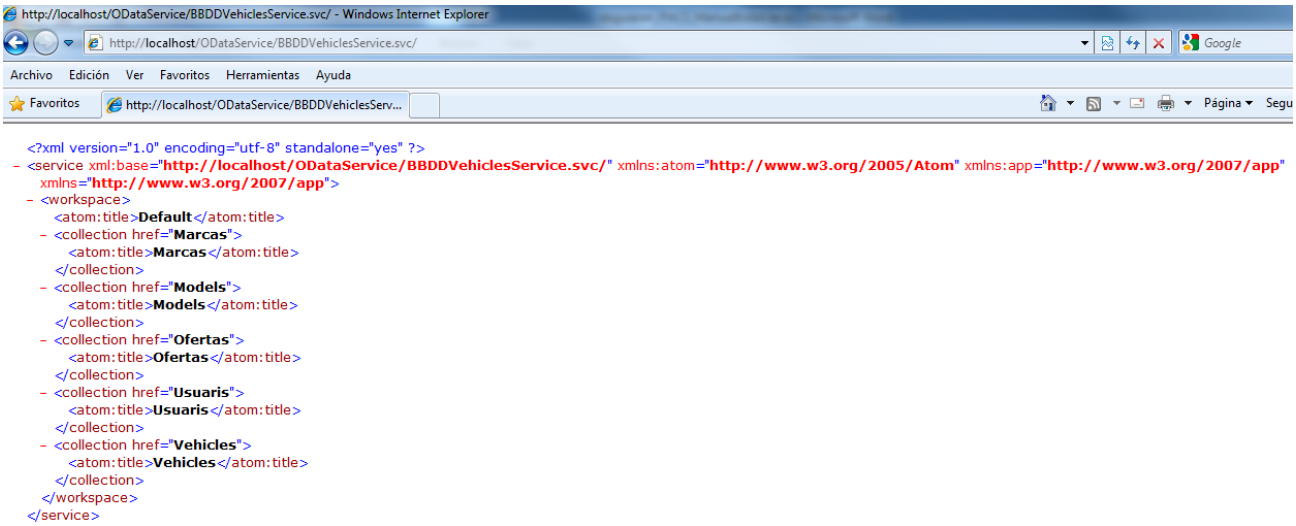


Esperar la finalització de l' instal·lació:

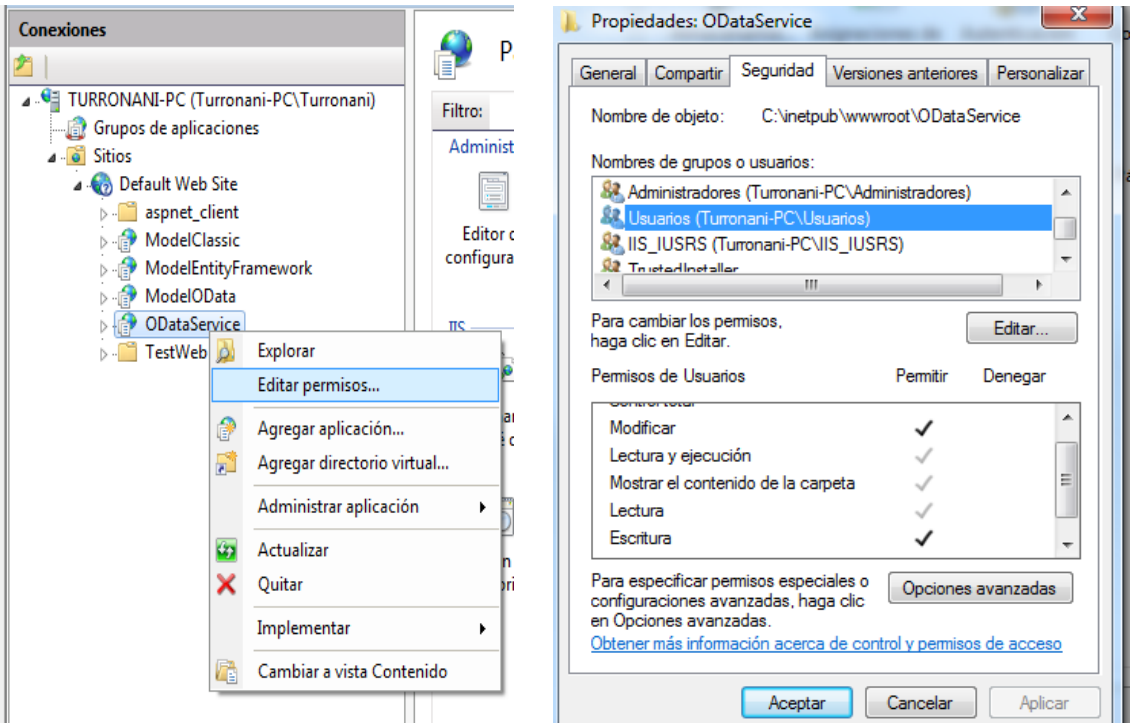


Verificar la correcta instal·lació del servei:

http://localhost/ODataService/BBDDVehiclesService.svc/

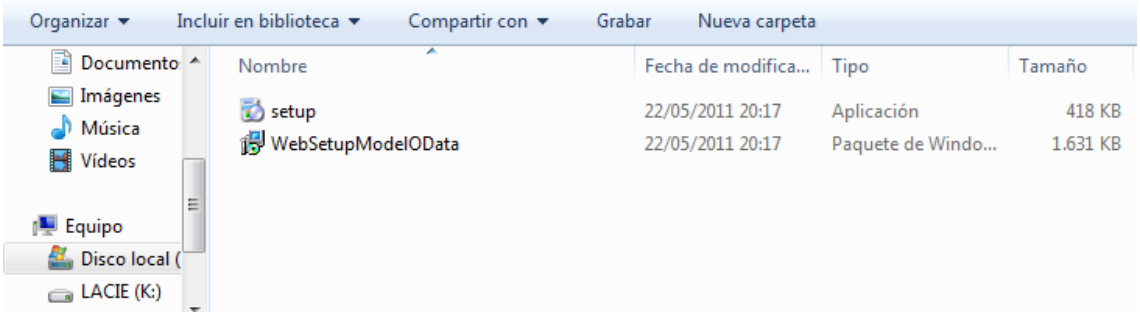


Assignar permisos, al directori virtual del servei, per modificar i escriptura a "Usuarios":

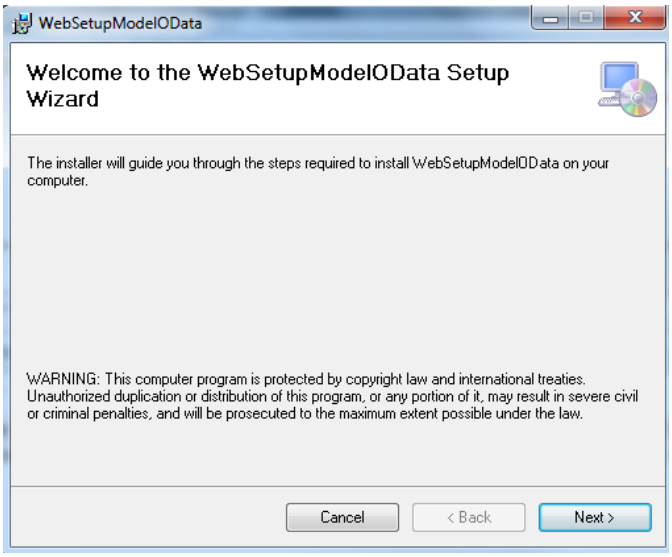


8.4 Model OData - Client

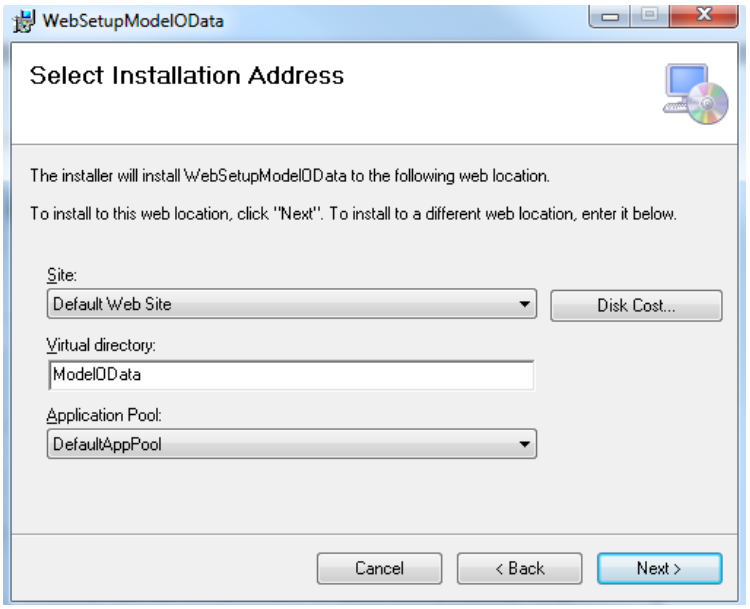
Executar el fitxer d'instal·lació (setup.exe):



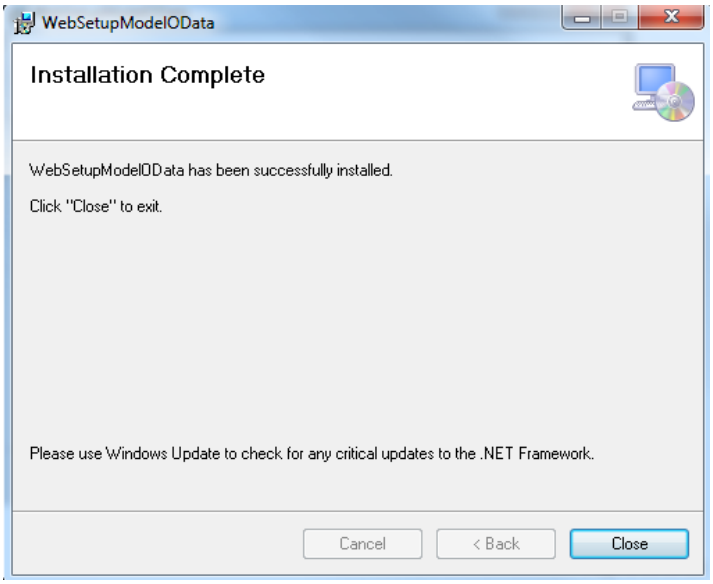
Seguir l'assistent:



Indicar el nom del directori virtual: **ModelOData** i continuar.



Esperar la finalització de l' instal·lació:



Comprovar la correcta instal·lació de la versió:

http://localhost/ModelOdata/Home.aspx

9. Bibliografia

Les fonts bibliogràfiques consultades han estat les següents:

Model Clàssic

- http://www.asp.net/data-access/tutorials
- http://www.asp.net/data-access/tutorials/creating-a-data-access-layer-cs
- http://www.asp.net/data-access/tutorials/creating-a-business-logic-layer-cs
- http://msdn.microsoft.com/en-us/sqlserver/aa937722.aspx
- http://msdn.microsoft.com/es-es/library/ss7fbaez.aspx

Entity Framework

- Unai Zorrilla Castor, ADO .NET Entity Framework 4.0 Aplicaciones y servicios centrados en datos. Ed. Krasis Press
- http://www.asp.net/entity-framework/tutorials#Getting%20Started
- http://msdn.microsoft.com/en-us/library/bb738458%28v=vs.90%29.aspx
- http://msdn.microsoft.com/es-es/data/aa937723
- http://www.scip.be/index.php?Page=ArticlesNET16#AddUpdateDelete
- http://www.scip.be/index.php?Page=ArticlesNET12#EntityFramework
- http://msdn.microsoft.com/es-es/data/aa937709

OData

- José Miguel Torres ,Whitepaper - Comenzando con OData. Ed. Krasis Press
- http://www.odata.org/
- http://msdn.microsoft.com/en-us/library/ff478141.aspx

Altres

- http://www.asp.net/ajax/ajaxcontroltoolkit/samples/
- http://asp-net-example.blogspot.com/2009/12/ajax-asyncfileupload-how-to-use.html
- http://msdn.microsoft.com/en-us/library/ms366730.aspx