UNIVERSIDAD POLITECNICA DE VALENCIA

Laboratory

Lab Session 4

Persistence layer with Entity Framework

Ingeniería del Software

ETS Ingeniería Informática
DSIC – UPV

Curso 2019-2020

1. Goal

IMPORTANT: Read the whole document before starting to programm.

The goal in this session is the development of a data access layer that will offer the business logic all services required to retrieve, modify, remove or add objects to the persistence layer, but ensuring that the business logic layer does not know which persistence mechanism is being used. We are going to use Entity Framework, a persistence framework by Microsoft for the .NET, which offers object-relational mapping and is designed around the Repository + Unit of Work patterns.

Previously to this session, the student should revise the chapter 6 "Persistence Design" and the related seminars "Entity Framework" y "DAL". Additionally, you should also check the "VehicleRental" study case. You can download this complete implementation from Poliformat, which is an example to develop the current case study.

Thus, in this session you will develop the classes that will make the persistence layer of the case study EcoScooter, in a similar way to the reference example, and testing that the persistence mechanism is working properly.

2. Design of the Data Access Layer (DAL)

Transparent access to the persistence mechanism (persistence agnostic) will be achieved by introducing an **interface** to bridge the gap between the business logic layer and the persistence layer. This interface, named **IDAL** (DAL = Data Access Layer), is responsible of abstracting the data access services from the underlying persistence mechanism (i.e. SQL, XML, etc.).

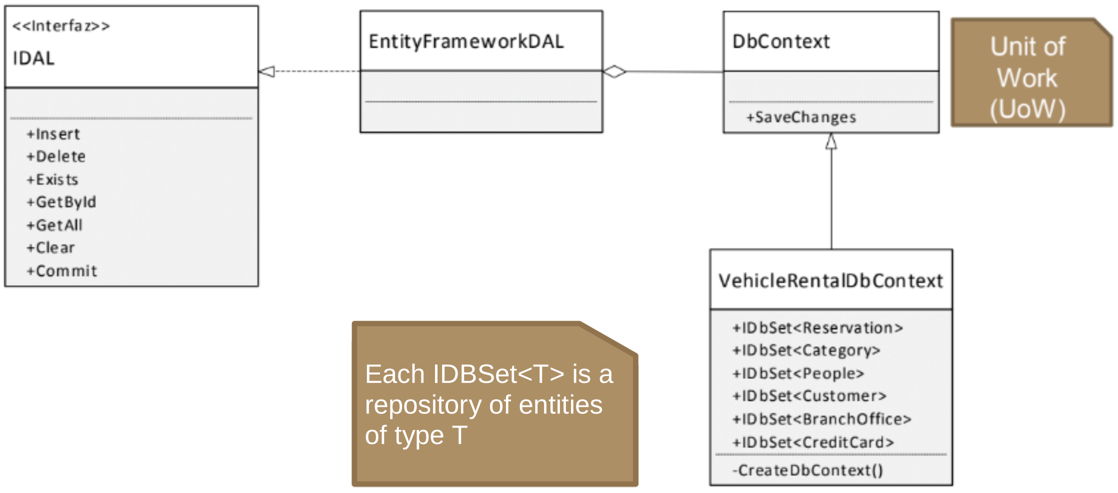

Figure 1. Architecture of the Data Access Layer

We are going to work with Entity Framework (EF) as the persistence mechanism in the laboratory Project, therefore we need a concrete implementation of IDAL for EF, which in the example project is called `EntityFrameworkDAL`.

Entity Framework is an object-relational mapping framework by Microsoft, which will allow our application work directly with Entities that are objects from the business logic, and not specific data-transfer objects. EF will automatically manage the persistence of those objects in a relational database like SQL.

Data Access with EF is designed around the Repository + Unit of Work patterns, as explained in the theory lessons (Chapter 6 about Persistence). The Unit of Work (UoW) is implemented by the `DBContextISW` class, which extends the class DbContext of Entity Framework. **`DBContextISW`** `provides the method` ***RemoveAllData()*****,** which allows delete all the Information stored in the database**.**

In the example project, the extended class is named `VehicleRentalDbContext.` In this class, you can observe the following features:

- Extends from `DbContext`

- Has several properties of type `IDbSet<Type>` where `Type` is one of the classes in the domain model that have to be persisted (these are called entities). Each IDbSet is a repository with the typical methods to access, add or remove objects). For example, in `VehicleRentalDbContext,` we find the following:

```
public IDbSet<BranchOffice> BranchOffices { get; set; }
public IDbSet<Reservation> Reservations { get; set; }
public IDbSet<Category> Categories { get; set; }
public IDbSet<Person> People { get; set; }
public IDbSet<Customer> Customers { get; set; }
public IDbSet<CreditCard> CreditCards { get; set; }
```

- Has a constructor that passes the base constructor the name of the database connection string ("VehicleRentalDBConnection"), which is defined in configuration file `App.config`. We can also include some configuration directives in the constructor, like the caching policy (see seminar on Entity Framework)

The IDAL interface is an adapter combining the functionality of the repositories (Insert, Delete, GetXXX, Exists) and the functionality of the UoW (Commit method). If we look at the example project `VehicleRental`, we would find the following methods in IDAL:

```
void Insert<T>(T entity) where T : class;
void Delete<T>(T entity) where T : class;
IEnumerable<T> GetAll<T>() where T : class;
T GetById<T>(IComparable id) where T : class;
bool Exists<T>(IComparable id) where T : class;
void Clear<T>() where T : class;
void Commit();
```

Those are the standard methods already commented, plus an additional method (Clear) to remove all data from the database. Note that those methods use *genericity*, highly reducing the amount of code to implement. In other words, instead of implementing the following individual operations:

```
void InsertOffice(BranchOffice o);
void InsertReservation(Reservation r);
void InsertCategory(Category c);
...
```

The genericity mechanism allows us to define a single:

```
void Insert<T>(T entity) where T : class;
```

This method will be instantiated at runtime depending of the type of `T`.

It is important to remember that after completing a transaction (one or more operations that imply a change in the data), the method Commit should be called to actually persist the changes. Although there is not a specific method to update an object, **if objects have been internally modified, it is also needed to execute the method *Commit***

Task: The team will have to implement the class inhering from DbContextISW, analogously to how VehicleRentalDbContext class extends from `DbContext`, but for the actual problem. This class will declare an IDbSet property for every entity in the current domain model. As a suggestion, name that class `EcoScooterDbContext`, and their namespace as `EcoScooter.Persistence.` The team will also have to add to the project the code of `DbContextISW`, `IDAL` y `EntityFrameworkDAL` without any change (because they are generic). The only thing to change is the namespace, which should be `EcoScooter.Persistence.` Thus, first you download from PoliformaT the file EntityFrameWorkImp.zip (in `English/Laboratory/Lab4-Data Access Layer`).

The implemented classes should be placed into the folder Persistence in your EcoScooterClassLibrary project. In case you have any doubt, you can observe the example project using the solution explorer.

3. Using Data Annotations

The automatically generated code of the previous session (in the folder Persistence/entities) should be properly modified using the necessary ***Data Annotations*** to provide some information to the Entity Framework. Concretely, you will add the following annotations to the properly attributes (as explained in the seminar 4):

```
[Key]
[InverseProperty("XXX")]
```

4. Initial configuration for the solution (Team Leader).

Framework you must add a certain package to your class library project to work with Entity Framework. After connecting and downloading the latest version of the solution in Visual Studio, the team leader should select the option `Herramientas > Administrador de paquetes NuGet > Administrar paquetes NuGet para la solución,` and in the text field called "Examinar" type Entity Framework. This package will show first on the list (see Figure 2) and you will have to install it into the class library project created in sessions 2 and 3 (**EcoScooterClassLibrary**, which contains the folders `BusinessLogic` and `Persistence`). Check in the `Explorador de Soluciones` that the references of your project Entity Framework is included. Once this is done, the team leader will commit and push the changes.

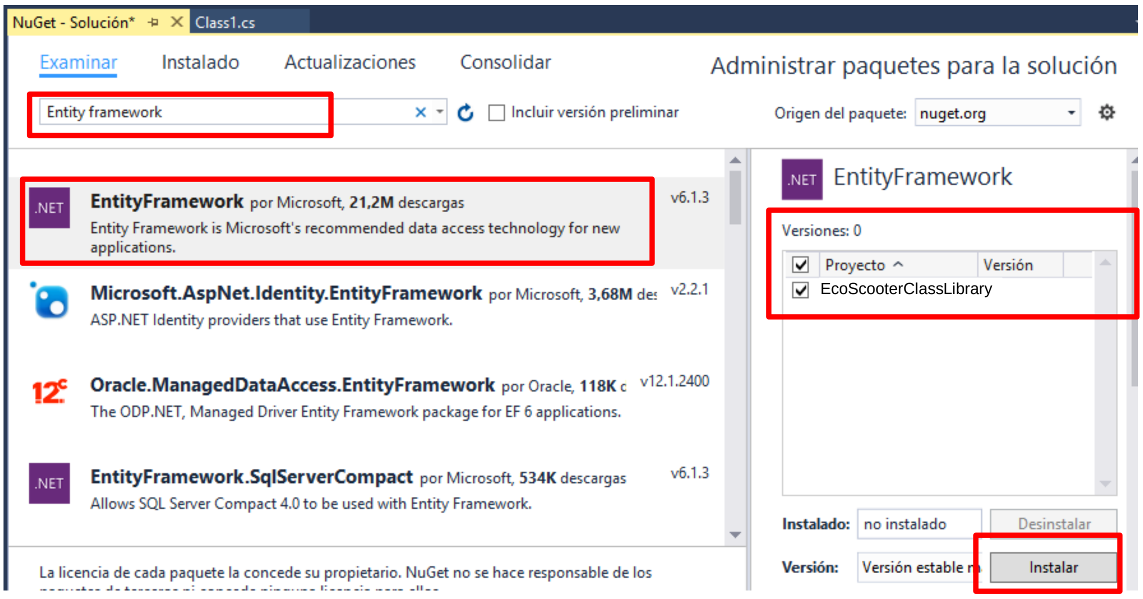
NuGet - Solución* Class1.cs
Examinar Instalado Actualizaciones Consolidar
Administrar paquetes para la solución
Entity framework
Incluir versión preliminar
Origen del paquete: nuget.org
EntityFramework por Microsoft, 21,2M descargas v6.1.3
Entity Framework is Microsoft's recommended data access technology for new applications.
Microsoft.AspNet.Identity.EntityFramework por Microsoft, 3,68M des v2.2.1
ASP.NET Identity providers that use Entity Framework.
Oracle.ManagedDataAccess.EntityFramework por Oracle, 118K c v12.1.2400
The ODP.NET, Managed Driver Entity Framework package for EF 6 applications.
EntityFramework.SqlServerCompact por Microsoft, 534K descargas v6.1.3
Allows SQL Server Compact 4.0 to be used with Entity Framework.
La licencia de cada paquete la concede su propietario. NuGet no se hace responsable de los
EntityFramework
Versiones: 0
Proyecto Versión
EcoScooterClassLibrary
Instalado: no instalado Desinstalar
Versión: Versión estable Instalar

Figure 2. Adding the Entity Framework package to the class library project

5. Updating the Constructors

In the previous session, you defined the necessary constructors to create the objects of the business logic. Following the instructions, you defined a default constructor and a constructor with arguments.

The default constructor is used by the Entity Framework when it must materialize an object in memory after retrieving the information from the tables in the database. Therefore you do not have to initialize the attributes of the object (EF already does), but you do have to initialize the attributes of type collection.

The other constructor with arguments is the one that should be used in the code when it is necessary to create an object. As you already know, EF gives values to the ID attributes of numerical type automatically in the moment in which the object is persisted for the first time. Therefore, it is not necessary to give value to these attributes in the object's constructor since EF will change the value in when it persists for the first time. Thus, **ELIMINATE FROM THE CONSTRUCTORS THE ARGUMENTS WHOSE OBJECTIVE IS TO INITIALIZE THE ATTRIBUTE ID OF THE OBJECT AS WELL AS THE ALLOCATION.**

6. Testing the persistence layer

Once the development of the data access layer is completed, commit and synchronize (push) your solution with a commentary like “Persistence layer completed (beta)” or similar.

Now is time to test the persistence layer. To accomplish that, we could create a console application named Lab4 inside the Testing folder, in a similar way we did in the first lab session. In that project we should create an instance of the `EntityFrameworkDAL`, which in turn will require an instance of EcoScooterDbContext. When the EcoScooterDbContext is instantiated for the first time, EF will create the database. Afterwards, we should create some business objects and persist them using the services offered by the DAL.

To work, that console project should meet the following requirements:

- Include a reference to the class library project (EcoScooterClassLibrary): expand the project in the solution view and open the context menu on `Referencias > Agregar referencia …` then, inside the projects section, select the class library project.
- Include the Entity Framework using the NuGet packages manager.
- Add a data base connection string to the configuration file`App.config`. If using SQLServer as database engine and “EcoScooterDB” as the name of the database, then the connection string will result as :

```
<connectionStrings>
    <clear />
    <add name="EcoScooterDBConnection"
connectionString="Server=(localdb)\mssqllocaldb;Database=EcoScooterDB;Trusted_Connection=Tr
ue;MultipleActiveResultSets=true" providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

Nevertheless, to facilitate the work and ensure the database is created properly, a console test project is already provided, with some tests already implemented. Furthermore, some data examples are provided to populate the database. Although these tests could be done using a unit test engine like MSTest or NUnit, at this point we will simply use a console application (testing will be addressed later).

Add this project (DBTest) to the solution, and open the file named Program.cs. You will find an empty method named `createSampleDB` (IDAL dal) that should be completed with the code required to create business objects and persist them in the database. If you set this project as the Initial project (context menu and choose “Establecer como Proyecto de inicio”), you could execute it and check whether the database has been created properly.

```
static void createSampleDB(IDAL dal)
{
    // Remove all data from DB
    dal.RemoveAllData();


    // Populate the database with the data described in lab 4 bulletin

}
```

7. Tools for inspecting a database

It is possible from Visual Studio to connect to a DB to explore its structure and contents. To connect to an existing local DB (local file) perform the following steps:

- Select `Herramientas > Conectar con la Base de Datos`

- Select as data source: "`Archivo de base de datos de Microsoft SQL Server (SqlClient)`"
- Select the file with extension ".mdf" created by your application. Notice that the file should be in the path: `C:\Users\userName\ EcoScooterDB.mdf` (it depends on the app.config file)

Once the connection has been established, the tables may be explored using the server explorer that appears in Visual Studio. If you double-click on any table, you will see its structure. To see the data stored on a table, you should click on the table using the mouse right-button and selecting the option `Mostrar datos tabla.`

In a similar way, you can select the option `Nueva consulta,` which allows you to write SQL sentences for your database. For example, you can easily visualize all the register of a table by writing and executing (green play button in the created SQLQuery) the following sentence:

"`Select * From TableName`"

IMPORTANT: While doing these activities, commit and synchronize your changes whenever you deem appropriate and add descriptive comments about the changes introduced.