# MVC 5 con ADO.NET

## ADO.NET

ADO.NET es un conjunto de componentes del software que pueden ser usados por los programadores para acceder a datos y a servicios de datos.

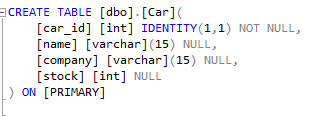
ADO.NET consiste en dos partes primarias:

* Data provider: Estas clases proporcionan el acceso a una fuente de datos, como Microsoft SQL Server y Oracle. Son los datos necesarios para generar la cadena de conexión.
* DataSets: Grupo de clases que describen una simple base de datos relacional en memoria. Estructuras que contendrán el resultado de las consultas que haremos a las tablas.

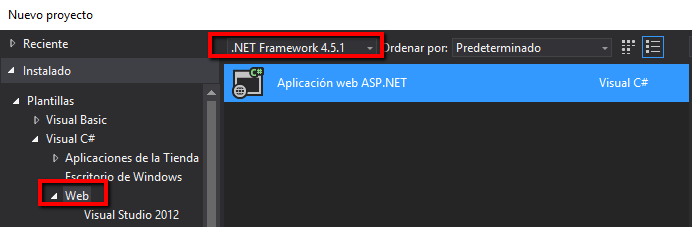
Durante la descripción del proyecto se explicarán estos puntos.

## Descripción del proyecto

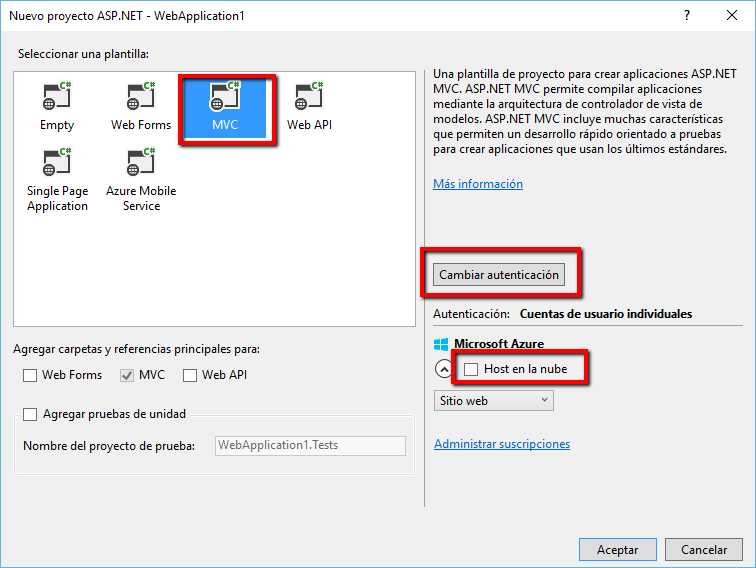
En el proyecto veremos un CRUD usando ADO.NET. Primero creamos la tabla Car en SQL Server



Crearemos una “Aplicación web ASP .NET” usando “.NET Framework 4.5.1”.

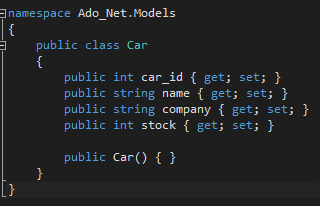


En la opción “Cambiar autenticación” seleccionamos “Sin autenticación”, así lo manejaremos en este proyecto. Igualmente no seleccionamos la opción “Host en la nube”. Y seleccionamos el proyecto MVC.



Creamos una clase para poder manejar la entidad Car. En la carpeta Models ubicaremos todas las entidades.





Previo a realizar el CRUD de la entidad Car, vamos a necesitar una conexión con la base de datos, por lo cual en la carpeta Funciones existirán dos clases: F\_Car.cs y F\_SQL.cs



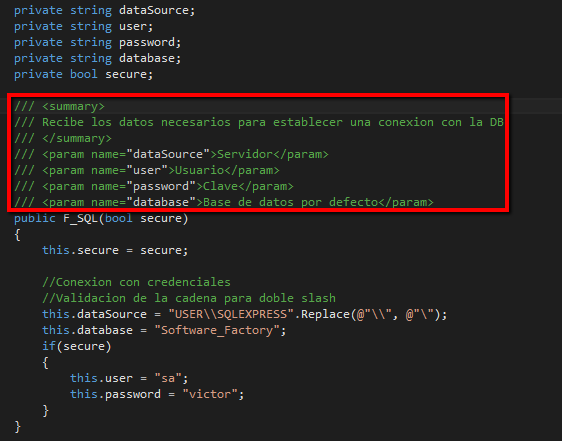
### F\_SQL.cs

Definiremos la lógica para conectarnos a una base de datos mediante la autenticación de Windows o la autenticación de SQL Server.

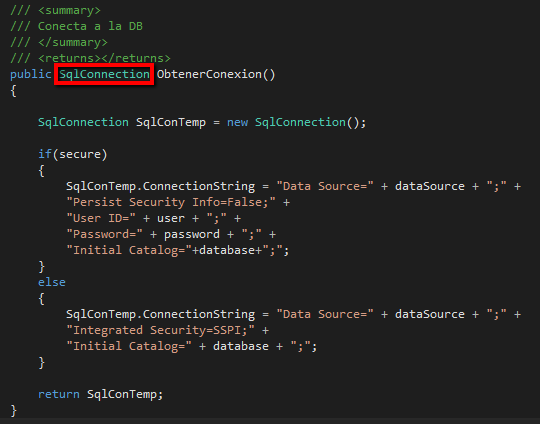
Necesitamos las siguientes importaciones:



Primero definimos los atributos básicos para conectarnos (señalados en la imagen). Para crear la conexión se recibe un bool que valida si se realizara una conexión mediante la autenticación de Windows o la autenticación de SQL Server por lo cual en el caso de ser la segunda opción asigna las credenciales. Esto varía lo que es la cadena de conexión, que se genera a continuación.



Haciendo referencia a los **Data provider**, para conectar con la base de datos se usa la clase SqlConnection y se asigna los datos a su atributo ConnectionString.

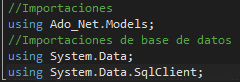


### F\_Car.cs

Usará F\_SQL.cs para poder interactuar con la base de datos y realizar las operaciones necesarias para realizar el CRUD de la entidad Car.

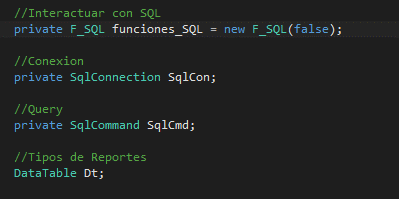
#### Método List

Realizamos las siguientes importaciones para poder usar la clase Car ubicada en Models y para poder usar los **Data provider** y **DataSets**



Se declaran clases que nos ayudarán a interactuar con la base de datos y también para almacenar los datos obtenidos

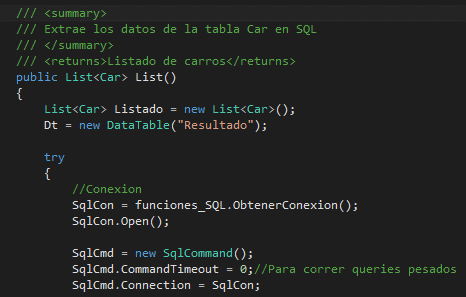
Por el lado de **Data provider** se usará SqlConnection para conectar a SQL Server y SqlCommand para poder realizar las consultas y operaciones en SQL Server. Por el lado de **DataSets** se usara la clase DataTable para capturar la data resultante.



Para el método List se tiene el siguiente código

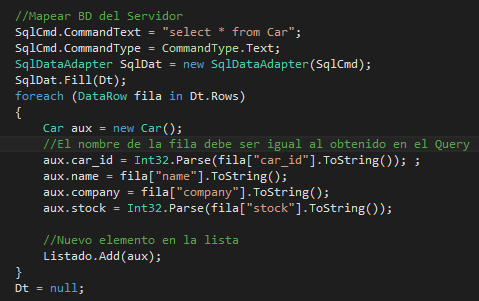
Se crea la cadena de conexión y se abre esta conexión con el método Open() a continuación se prepara el comando a ejecutar, que en este caso es una consulta a la tabla Car, previamente se asigna el atributo CommandTimeout con el valor de 0 para que en caso la consulta tarde por la gran cantidad de datos, el proceso no se detenga.

Luego al comando le asignamos la cadena de conexión mediante el atributo Connection.

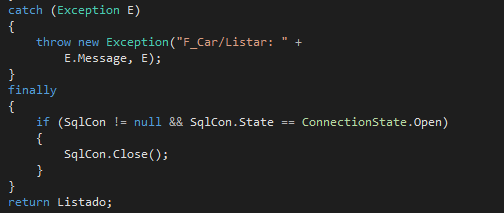


A continuación indicamos el query a ejecutar y el tipo de comando que se esta asignando. Con la clase SqlDataAdapter que sirve como "Puente" utilizado para transferir data entre una fuente de datos y un objeto DataSet le asignamos el comando y con el método Fill almacenamos los datos en el DataTable previamente definido como **Dt**.

Iteramos los registros de este DataTable que representa un DataRow y vamos almacenando cada registro en el listado de carros. Luego liberamos el DataTable dándole el valor de null.

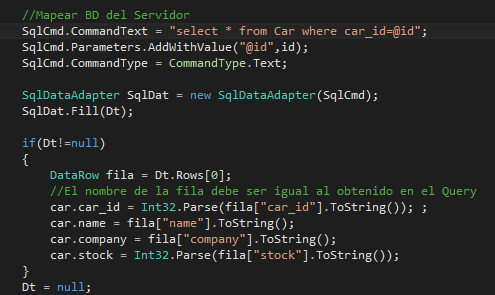


Todo esto se ejecuta dentro de un try catch para capturas las excepciones. Finalmente se cierra la conexión y se retorna el listado.



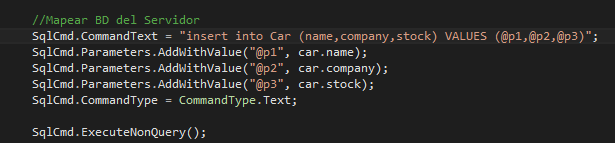
#### Método Find

En este caso el comando asignado debe recibir como parámetro el id del registro a buscar. Mediante el atributo Parameters.AddWithValue del comando se indica el id. Y se captura y retorna el registro.



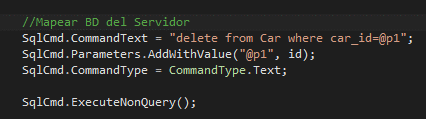
#### Método Insert

Se añaden los parámetros del registro a insertar y esta vez el comando se ejecuta con el método ExecuteNonQuery() que ejecuta una instrucción de Transact-SQL.



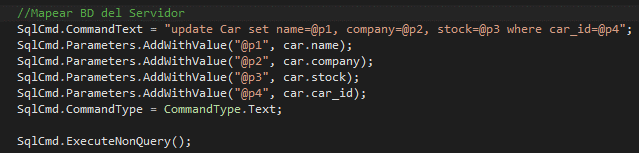
#### Método Delete

Se añaden los parámetros del registro a eliminar. Y se ejecuta la instrucción.



#### Método Update

Se añaden los parámetros del registro a editar. Y se ejecuta la instrucción.



#### Detalle de las vistas

Cuando se retorna a una vista si esta usa un modelo se especifica en la parte superior con la variable @model y se especifica el tipo de dato se ha retornado, por ejemplo para el listado de autos sería

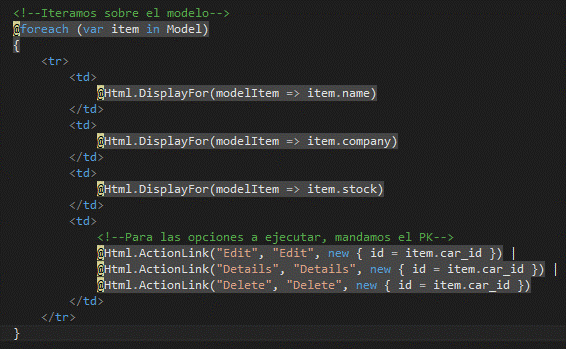


También se dejan enlaces para interactuar con las demás acciones del controlador. Estos son los @Html.ActionLink.

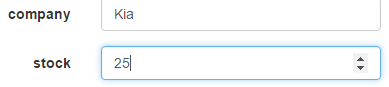


Por ejemplo en la vista que lista los carros iteramos y mostramos los datos. Además dejamos habilitadas acciones como Edit, Details y Delete que necesita el id para poder ejecutarse por lo cual en el @Html.ActionLink reciben este atributo como parámetro.

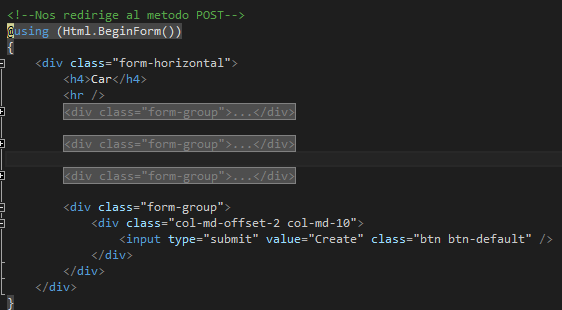
@Html.DisplayFor devuelve el formato HTML que coincide con el tipo de propiedad.



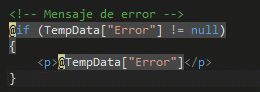
@Html.EditorFor devuelve un elemento input HTML que coincide con el tipo de propiedad, por ejemplo para el atributo stock del tipo int el input es distinto y se asigna automáticamente.



Html.BeginForm() engloba un formulario para los datos de la clase



@TempData["…"] obtiene datos temporales que se van a pasar a la vista



@Html.HiddenFor: Devuelve un elemento oculto de HTML. Evita que sea modificado el valor.



## 

## Información útil

* Información sobre Ado.Net (<https://es.wikipedia.org/wiki/ADO.NET>)