Proyecto Angular con Breeze.

Gonzalo Carmenado Ruiz

# Contenido

## 1. Introducción.

Este manual permite al usuario crear un proyecto el cual trabaje con una API Breeze a la que atacaremos desde AngularJs.

Este tutorial esta destinado a personas con conocimientos básicos de AngularJs, HTML y c#, ya que no se explicarán conceptos como la creación de un controlador AngularJs, creación de BBDD o manejo general de Visual Estudio.

Para desarrollar este proyecto se ha utilizado Visual Studio Comunity 2017, HTML 5, AngularJs(v1.7.2), Breeze, Jquery(v3.3.1) y SQLManager. La base de datos utilizada es la BBDD que crea por defecto Visual Studio almacenada en local.

## 2. Creación del proyecto e inicialización de los componentes.

Para comenzar creamos un proyecto “Aplicación ASP.NET Web vacia”.

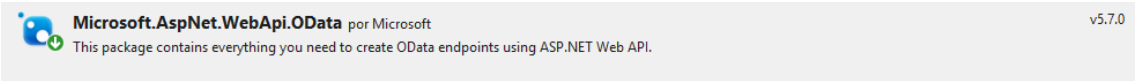
Necesitaremos instalar los siguientes paquetes desde el gestor de paquetes:

- `Vreeze Webapi2 = install-package breeze.webapi2.ef6`
- `Angularjs core = install-package angularjs.core`
- `Breeze AngularJs = install-package breeze.angular`

Paquetes opcionales para dar estilos y funcionalidades que no son propias de breeeze:

- `Toaster = install-package toastr`
- `Fontawesome = install-package fontawesome`

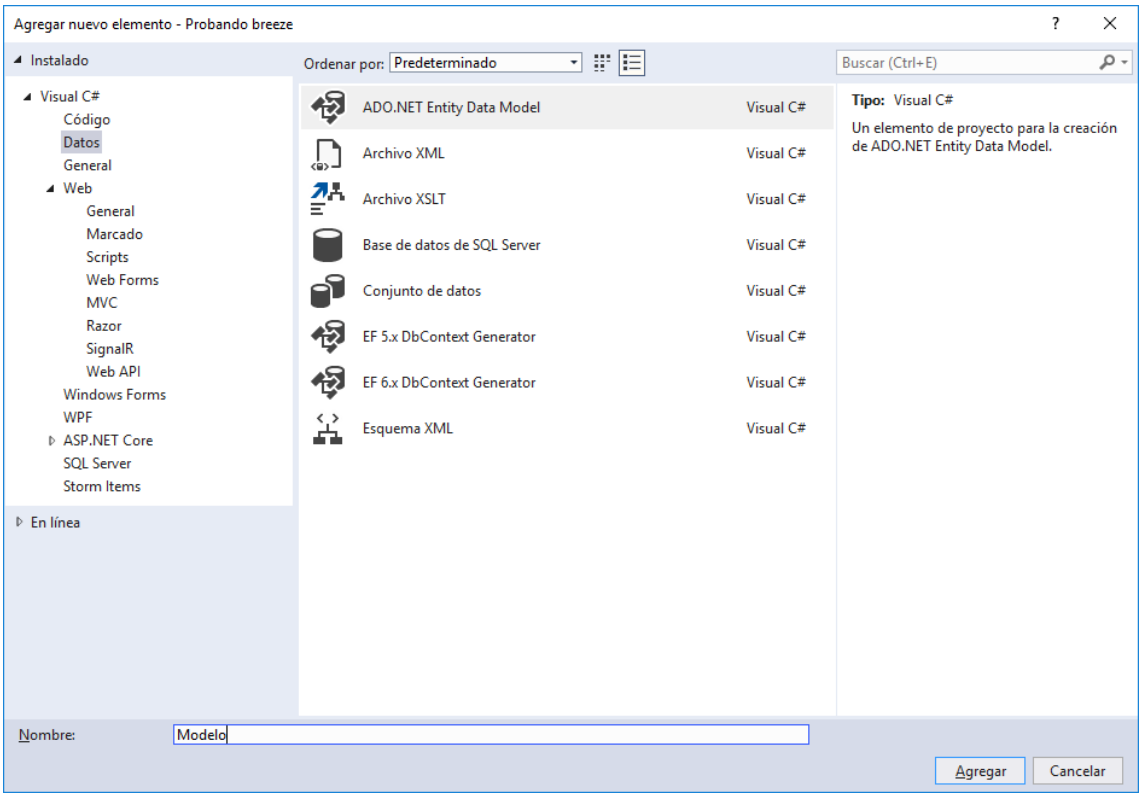
También deberemos instalar el paquete oficial de OData en nuestro proyecto.



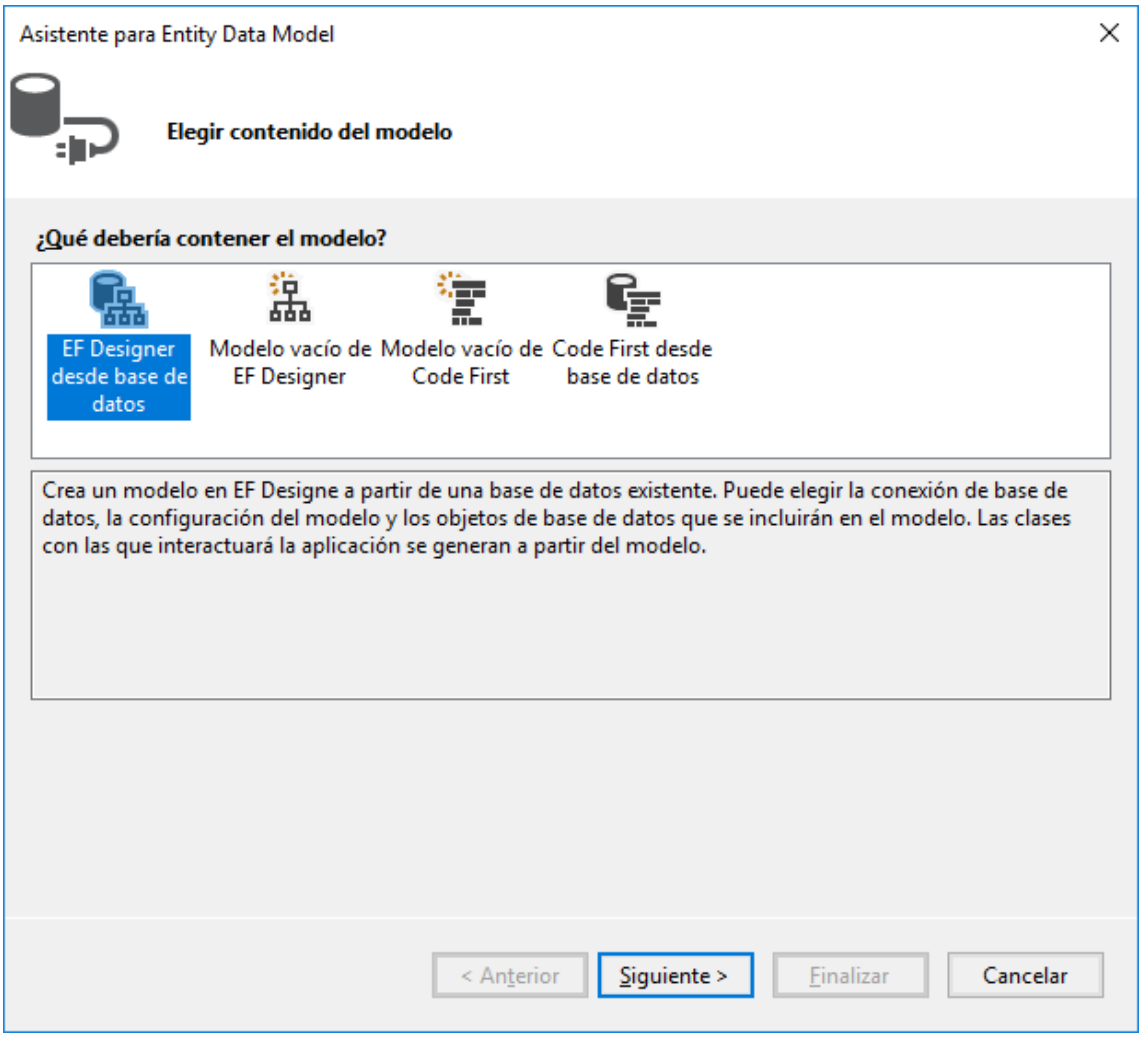

Creamos un fichero HTML y su controlador correspondiente de Angular de forma normal.

Creamos un Modelo de datos:

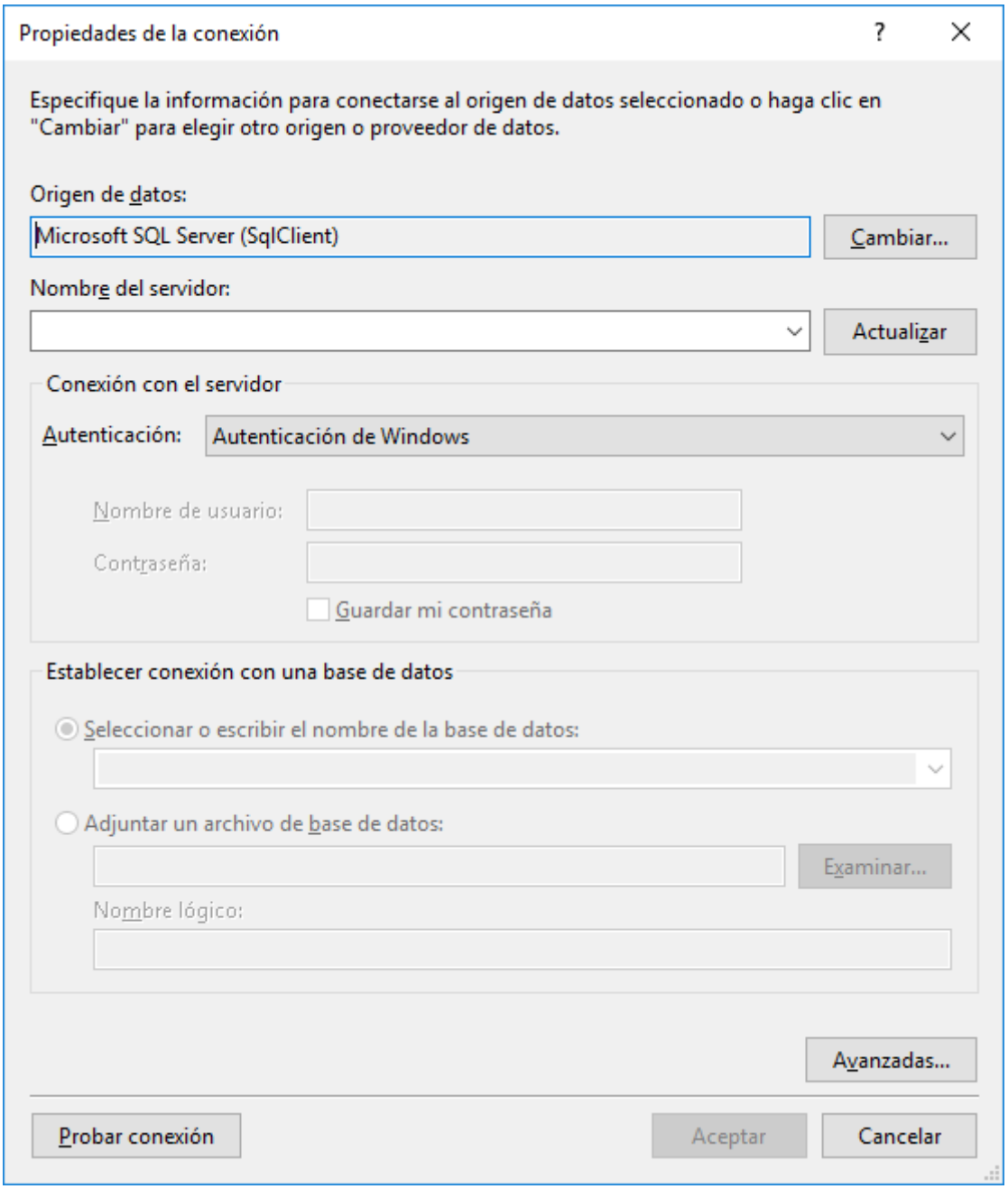
1. Elemento nuevo > Datos > ADO.NET Entity Data Model.


Agregar nuevo elemento - Probando breeze
Instalado
Visual C#
Código
Datos
General
Web
General
Marcado
Scripts
Web Forms
MVC
Razor
SignalR
Web API
Windows Forms
WPF
ASP.NET Core
SQL Server
Storm Items
En línea
Ordenar por: Predeterminado
Buscar (Ctrl+E)
ADO.NET Entity Data Model Visual C#
Archivo XML Visual C#
Archivo XSLT Visual C#
Base de datos de SQL Server Visual C#
Conjunto de datos Visual C#
EF 5.x DbContext Generator Visual C#
EF 6.x DbContext Generator Visual C#
Esquema XML Visual C#
Tipo: Visual C#
Un elemento de proyecto para la creación de ADO.NET Entity Data Model.
Nombre: Modelo
Agregar
Cancelar

2. Seleccionamos la opción EF Designer desde Base de Datos.

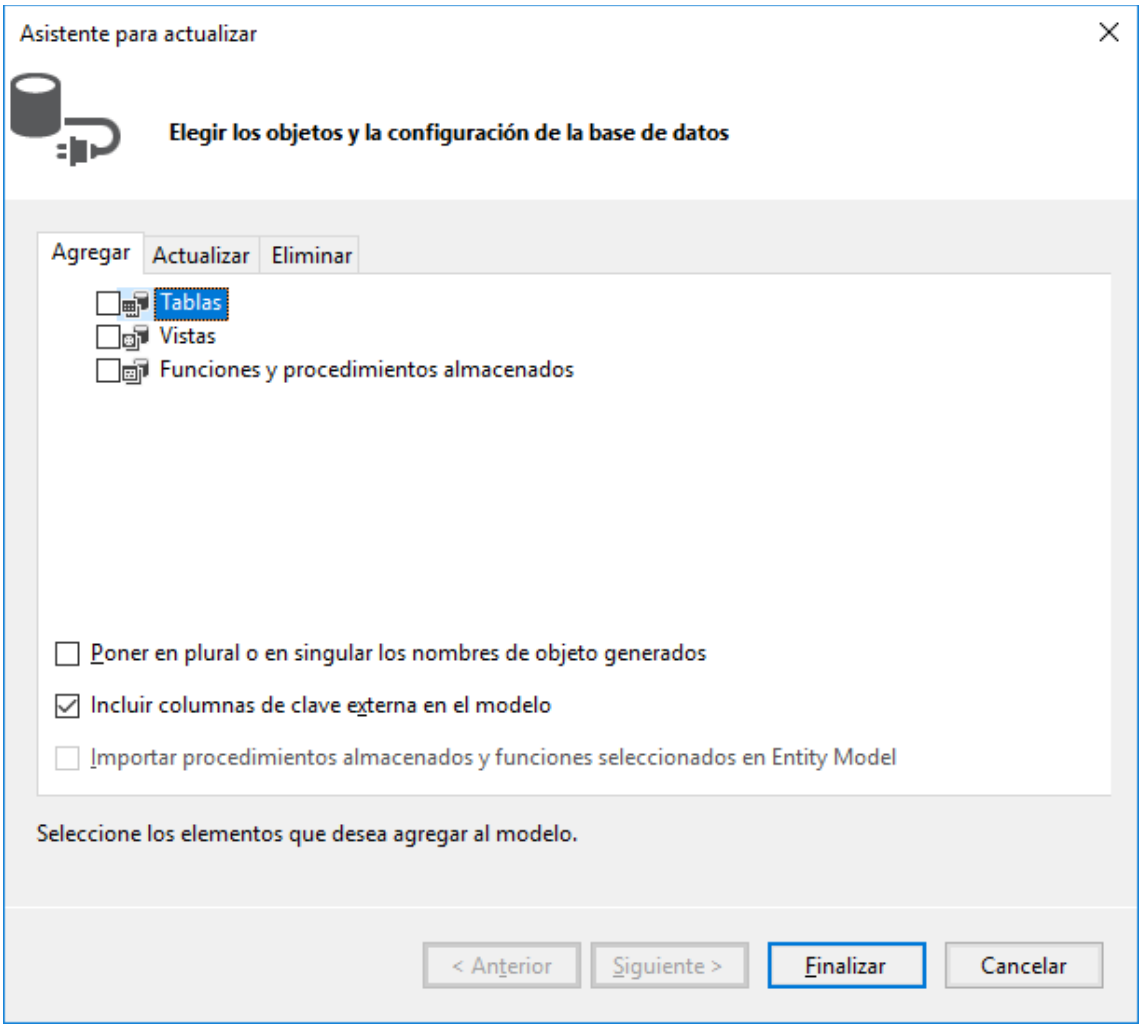


3. Pinchamos sobre nueva conexión.




4. Rellenamos los datos correspondientes a nuestro servidor (Cadena de conexión, usuario, contraseña…) y pulsamos sobre aceptar.

5. Nos aparecerá una ventana en la que deberemos seleccionar los elementos que queremos incorporar a nuestro proyecto.




6. En la ventana anterior se pincha sobre finalizar y ya tendremos el modelo de datos preparado.

Ahora debemos crear un controlador de la API, para lo que iremos a crear nuevo elemento > Web > Web API > Web API Controller Class(v2.1).

Borramos todo el contenido de la clase.

Encima de la clase se declara que vamos a utilizar Breeze de la siguiente manera:

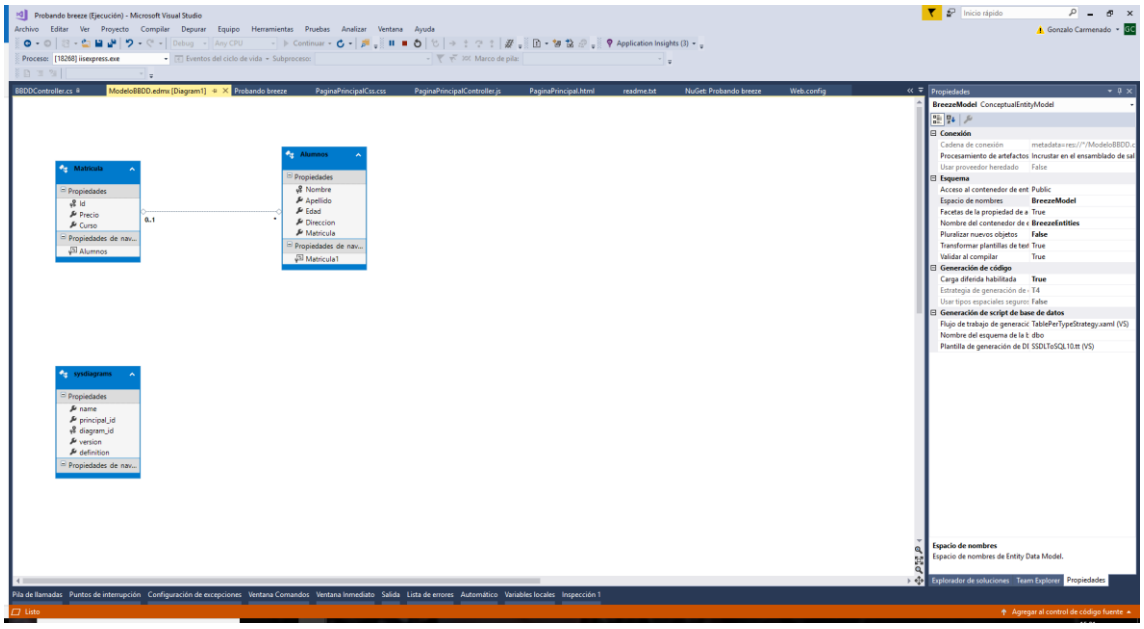
Copiamos el contenido base de la siguiente página web para tener los métodos básicos de lectura y escritura en la API. Mas tarde se podrá seguir añadiendo nuevos.

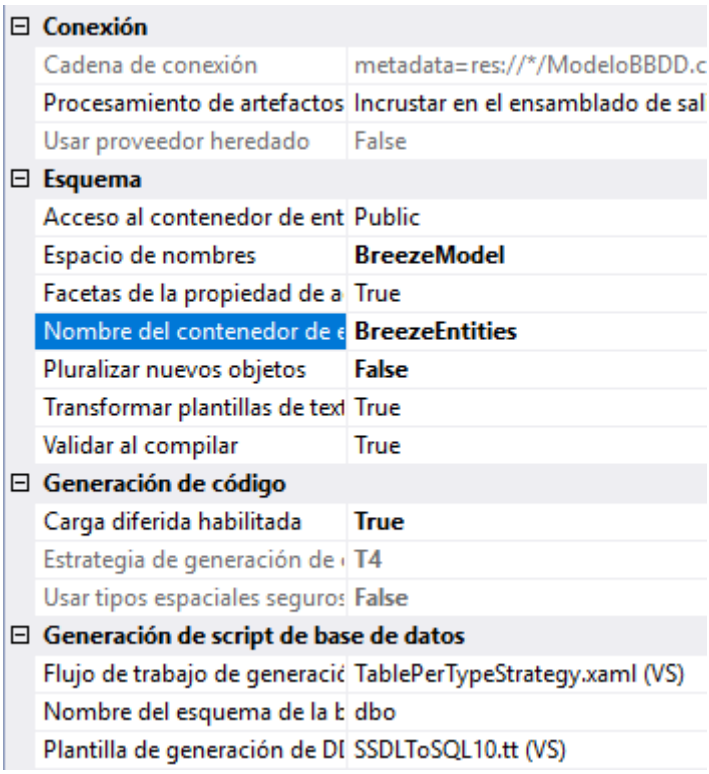
- http://breeze.github.io/doc-net/webapi-controller.html

```
1   using Breeze.ContextProvider;
2   using Breeze.ContextProvider.EF6;
3   using Breeze.WebApi2;
4   using Newtonsoft.Json.Linq;
5   using System.Linq;
6   using System.Web.Http;
7
8   namespace Probando_breeze
9   {
10      [BreezeController]
11      public class BBDDController ...
44  }
```

```
readonly EFContextProvider<BreezeEntities> _contextProvider =
                    new EFContextProvider<BreezeEntities>();
```

(*) Las palabras subrayadas en rojo indican que ahí se debe poner el contexto con el que trabajará el modelo. Para verlo iremos a nuestro modelo > doble click hasta que veamos el modelo > pulsar sobre un espacio en blanco y F4, el nombre que deberemos poner es el que coincide con el campo: Nombre del contenedor de entidades.






## 3. Creación de métodos.

Una vez terminado de hacer todas las declaraciones y crear todos los controladores, podemos comenzar a crear métodos para poder acceder a los valores de la tabla.

Vamos a comenzar creando un método Get que nos va a permitir obtener todos los datos de una de nuestras tablas.

```
//Para llamar a este método,
// ~/breeze/BBDD(Nombre del Controlador)/Matricula(Nombre del método)
              [HttpGet]
              public IQueryable<Matricula> Matricula()
              {
                     return _contextProvider.Context.Matricula;
              }
```

(*) Las palabras subrayadas en rojo hacen referencia al nombre que le hayamos dado con anterioridad a nuestro modelo.

Para comprobar si esto está funcionando, puedes utilizar un cliente API Rest y hacer una llamada al método creado anteriormente mediante la URL que aparece en el comentario superior. Recuerda que “~” es la ruta completa de tu servidor.

# 4. Llamadas a API Breeze desde AngularJs.

## 4.1. Configuración inicial.

Como se ha dicho al inicio de este documento, se considera que el lector de este documento tiene conocimientos básicos en HTML 5 y AngularJs, por lo que no se explicará como se generan los ficheros ni la estructura básica del mismo.

Lo primero será añadir todos los paquetes descargados inicialmente en el orden apropiado. Para ello dejo un ejemplo de cómo debería quedar.

```
<!--Angular y Jquery-->
<script src="Scripts/angular.js"></script>
<script src="Scripts/q.js"></script>
<script src="Scripts/jquery-3.3.1.js"></script>

<!--Breeze-->
<script src="Scripts/breeze.min.js"></script>
<script src="Scripts/breeze.directives.js"></script>
<script src="Scripts/breeze.bridge.angular.js"></script>

<!--Otros ficheros-->
<script src="PaginaPrincipalController.js"></script>
```

Una vez se tiene creado el documento HTML se creará un controlador de AngularJs normal, al cual le inyectaremos el complemento: “breeze.angular”.

```
angular.module('breezeApp', ['breeze.angular'])
    .controller('breezeCtr', breezeCtr)
    .run();
```

Ahora debemos comenzar a trabajar con Breeze, por lo que lo primero será crear un gestor que indicará a nuestro programa la ubicación del controlador Breeze al que vamos a acceder. Para ello crearemos una sentencia como la siguiente:

```
vm.manager = new breeze.EntityManager('breeze/BBDD');
```

(*) La parte entrecomillada es la ruta en la que se encuentra nuestro controlador Breeze (Ejemplo: breeze/nombreControlador). Si has seguido este tutorial, esa ruta será la misma que aparece en el ejemplo de cómo crear un método (en este caso sin el nombre del método).

## 4.2. Llamada GET.

A continuación creamos una función para ejecutar el código (No es necesario crear una función en sí, es solo por mantener el programa organizado):

```
function consultarDatosAlumnos() {
	var consulta = breeze.EntityQuery.from('Alumnos').expand('Matricula1');
		vm.manager.executeQuery(consulta)
			.then(function (promise) {
				vm.respuesta = promise.results;
			});
	}
```

(*) Se crea una variable consulta que será la que ejecutaremos posteriormente. “Alumnos” es la referencia al nombre de la tabla a la que queremos acceder, “Matricula1” es el nombre que

le hemos dado a la variable de navegabilidad para Breeze, es decir, la clave foránea que utilizaremos para obtener la matricula asociada a cada uno de los alumnos. Esto puede anidarse de forma ilimitada, obteniendo una estructura de datos descendente.

### 4.3. Llamada Delete.

Ahora vamos a realizar una llamada a nuestra API para borrar uno de los registros. Para ello crearemos otra función, la cual recibirá por parámetros el objeto que queremos borrar. Después, marcaremos ese objeto para borrado con la función "`.entityAspect.setDeleted()`". Es necesario recordar que esto no va a borrar el registro, para que se haga efectivo tendremos que llamar a la función creada anteriormente que aplica los cambios en la BBDD. Ejemplo:

```
function borrarAlumnos(alumno) {
            alumno.entityAspect.setDeleted();
            guardarCambios();
      }
```

### 3.4. Llamada Create.

Para crear un nuevo registro en la BBDD crearemos otra función. Esta función se encargará de crear los JSON con los datos que queremos (Nombre, Apellidos…) y de crear las entidades a las que se asociaran esos JSON.

```
function anadirAlumnos() {
      var matriculaId = generadorGUID();
      var alumnoId = generadorGUID();
      var matriculaInsertar = { Id: matriculaId, Precio: vm.precioCurso, Curso:
      vm.nombreCurso, FormStatus: 'new' };
      vm.manager.createEntity('Matricula', matriculaInsertar);
      var alumno = { Id: alumnoId, Nombre: vm.nombreAlumno, Apellido:
      vm.apellidoAlumno, Edad: vm.edadAlumno, Direccion: vm.direccionAlumno,
      Matricula: matriculaId, FormStatus: 'new' }
      vm.manager.createEntity('Alumnos', alumno);
      guardarCambios();
      }
```

(*) El texto subrayado en verde llama a un método que permite crear GUID en JavaScript.

(*) El texto en rojo nos permite crear el JSON que vamos en guardar en la BBDD, simplemente asignamos valores a las distintas celdas como se hace de forma habitual en JavaScript. El último campo("`FormStatus: 'new'`") nos permite decir que el JSON que estamos introduciendo es un nuevo registro en nuestra tabla.

(*) El texto en azul nos permite crear la entidad en el modelo y asignarle los valores correspondientes. En este punto es donde se validarán los formatos de cada uno de los campos, por que suele ser zona de errores y es conveniente tener en cuenta que, si esto te da errores, es posible que sea el modelo lo que está fallando. El primer campo de la función hace referencia a la tabla que vamos a usar de modelo, por lo tanto, solo deberemos poner su nombre. El segundo campo es el JSON que vamos a pasar a dicha tabla.

Para terminar, llamamos a la función que nos permite guardar los cambios en nuestra BBDD.

Llegados a este punto, es conveniente que ejecutes tu código para mostrarte un error muy común cuando se crea un nuevo registro en una tabla.

Si ejecutas una llamada para crear un nuevo registro y no has realizado ninguna llamada previa a la BBDD desde ese controlador te saltará la siguiente excepción cuando ejecutes las líneas:

```
- vm.manager.createEntity('Matricula', matriculaInsertar);
- vm.manager.createEntity('Alumnos', alumno);
```

```
▶Error: Unable to locate a 'Type' by the name: 'Matricula'. Be sure to execute a query or call fetchMetadata first.
    at u.c._getEntityType (breeze.min.js:3)
    at X.et.createEntity (breeze.min.js:5)
    at ChildScope.anadirAlumnos (PaginaPrincipalController.js:41)
    at fn (eval at compile (angular.js:15869), <anonymous>:4:159)
    at callback (angular.js:28101)
    at ChildScope.$eval (angular.js:18838)
    at ChildScope.$apply (angular.js:18937)
    at HTMLButtonElement.<anonymous> (angular.js:28106)
    at defaultHandlerWrapper (angular.js:3795)
    at HTMLButtonElement.eventHandler (angular.js:3783)
```

Esta excepción es producida por que al no haber hecho ninguna llamada previa a la API de Breeze, actualmente no existe un modelo guardado sobre la tabla a la que estamos intentando acceder, por lo que el modelo no va a coincidir. Entonces, ¿Cómo solucionamos este problema?, tenemos 2 opciones:

1- Ejecutar alguna llamada Get en el controlador antes de hacer ninguna creación en la API de Breeze. Ejemplo: obtener los alumnos.
2- En el caso de que no tengamos la necesidad de hacer dicha llamada, por convención, se hace una llamada a la función que hemos creado con anterioridad para obtener los Metadatos de la conexión. Esta llamada se ejecutará al inicio del controlador para asegurarnos que siempre se hará antes de intentar crear un nuevo registro. Es muy importante recordar que esto debe hacerse cada vez que se inicie un controlador nuevo. Ejemplo:

```
function llamadaMetadatos() {
    var consulta = breeze.EntityQuery.from('Metadata');
    vm.manager.executeQuery(consulta)
            .then(function (promise) {
            });
}
```

### 4.4. Llamada Update.

Para modificar un registro de nuestra BBDD utilizaremos la siguiente función:

```
function modificarAlumno(alumno) {
        vm.manager.addEntity(alumno);
        guardarCambios();
}
```

(*) El texto en rojo nos permite añadir una entidad a la lista de cambios de nuestro manager. Nosotros solo tenemos que pasarle el registro con los cambios realizados, en este caso alumno.