

ASP.NET Identity y CodeFirst.

Gonzalo Carmenado

Contenido

| | |
|--|----|
| 1. Introducción..... | 3 |
| 2. Creación del proyecto e inicialización de los componentes. | 3 |
| 3. Añadir un campo nuevo a la tabla AspNetUsers..... | 5 |
| 4. Añadir tablas al modelo. | 7 |
| 4.1. Pluralización de una tabla. | 8 |
| 5. Creación de tablas con relaciones..... | 8 |
| 5.1. Relación 1-N. | 8 |
| 5.2. Relación N-M..... | 9 |
| 5.3. Relación 1-1..... | 10 |
| 6. Referencias y otras guías..... | 10 |

1. Introducción.

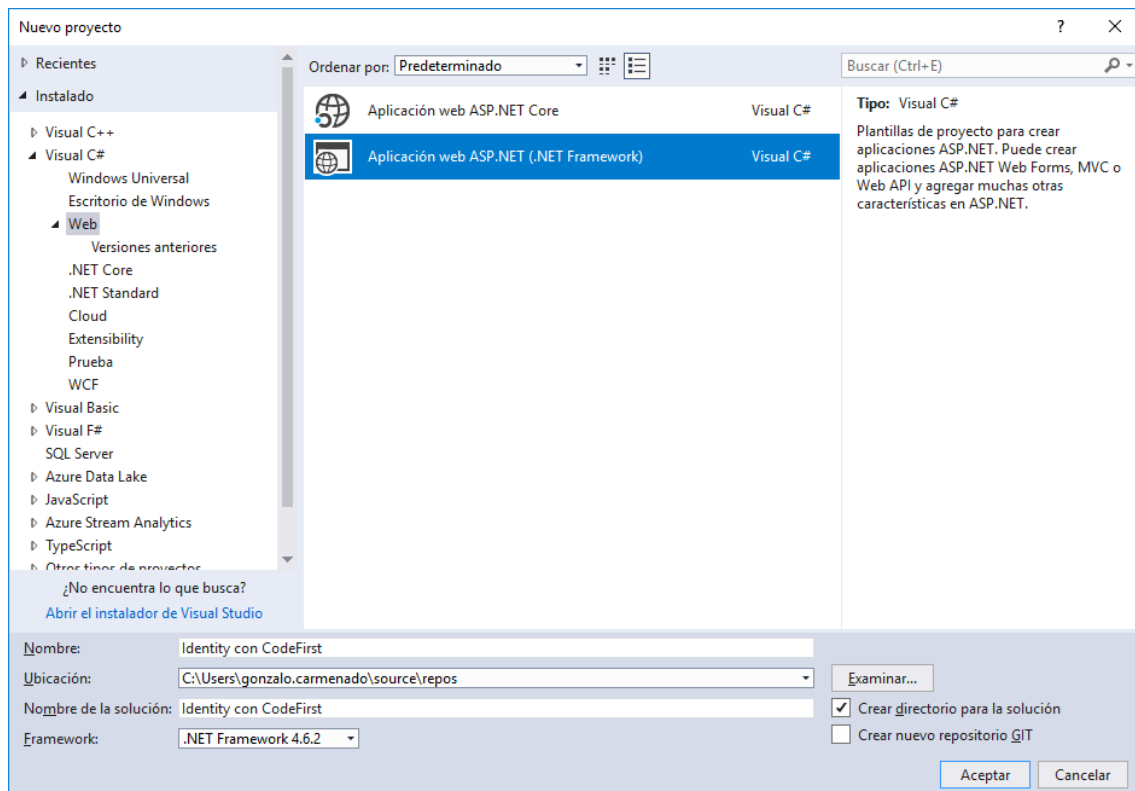
Este manual permite al usuario crear un proyecto el cual trabaje con una BBDD que se genera automáticamente mediante Code-First que usa el sistema de autenticación “Cuentas de usuario locales”

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

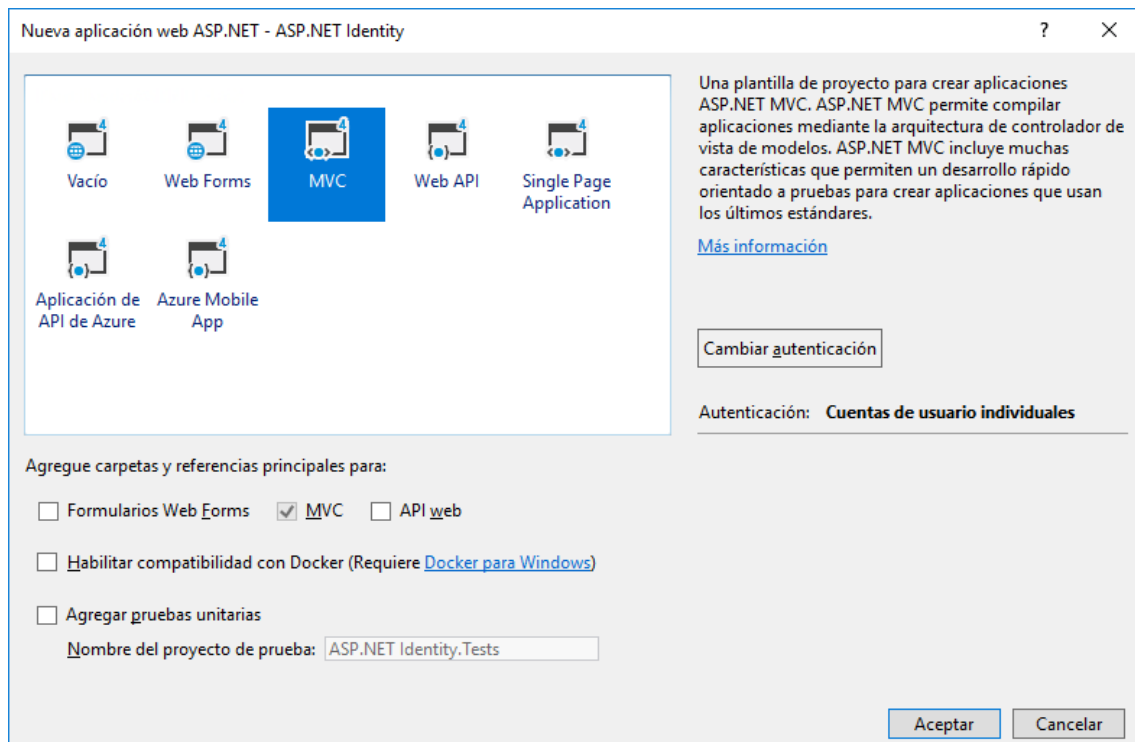
Para desarrollar este proyecto se ha utilizado Visual Studio Community 2017, HTML 5, Code-First, y Entity Framework. La base de datos utilizada es la BBDD que crea por defecto Visual Studio almacenada en local.

2. Creación del proyecto e inicialización de los componentes.

Para comenzar crearemos un proyecto ASP.NET. En este caso estamos utilizando la versión 4.6.2 de .NET Framework



La plantilla que utilizaremos será la de aplicación MVC.



Deberemos pinchar en el botón Cambiar autenticación y marcaremos la opción “Cuentas de usuario Individuales.



Con esto ya tendremos nuestro proyecto creado, ahora será necesario activar las migraciones con CodeFirst, para ello seguiremos la siguiente ruta:

-Herramientas > Administrador de paquetes NuGet > Consola del administrador de paquetes.

Una vez tenemos la consola abierta, escribimos y ejecutamos el siguiente código:

-Enable-Migrations

```

Versión de host 4.7.0.5148 de la Consola del Administrador de paquetes

Escriba 'get-help NuGet' para ver todos los comandos de NuGet disponibles.

PM> Enable-Migrations
Comprobando si el contexto indica una base de datos existente...
Se ha habilitado Migraciones de Code First para el proyecto Identity con CodeFirst.
PM>

```

Si todo se ha generado correctamente, la aplicación debería mostrar este mensaje.

Una vez hemos hecho esto, ya tendríamos creado nuestro proyecto con autenticación con cuentas de usuario individuales y CodeFirst para generar la base de datos. Gracias a esto, nos hemos ahorrado programar todas las validaciones, crear la BBDD y las tablas y sobre todo, nuestra base de datos se actualizará simplemente cambiando el modelo que tenemos en nuestra aplicación.

Pero... ¿Qué pasaría si queremos añadir campos a las tablas creadas por defecto en nuestra aplicación, o si deseamos añadir nuevas tablas a nuestro modelo de datos? A continuación, se va a mostrar como proceder en caso de querer modificar nuestro modelo de datos, bien sea añadiendo, borrando o actualizando campos.

3. Añadir un campo nuevo a la tabla `AspNetUsers`.

Lo primero es añadir el campo deseado al modelo que queremos utilizar. En este caso he utilizado el modelo "AccountViewModel", concretamente la clase encargada del registro de usuarios (RegisterViewModel).

```

[Required]
[Display(Name = "Apellido")]
public string Apellido { get; set; }

```

Ahora debemos añadir este campo a la clase encargada de generar y utilizar el modelo de la BBDD. En este caso la clase el controlador es "IdentityModel.cs" y la clase "ApplicationUser".

```

public string Apellido { get; set; }

```

La clase deberá quedar similar a esta:

```

public class ApplicationUser : IdentityUser
{
    public string Apellido { get; set; }
    public async Task<ClaimsIdentity>GenerateUserIdentityAsync
        (UserManager<ApplicationUser> manager){
        var userIdentity = await manager.CreateIdentityAsync(this,
            DefaultAuthenticationTypes.ApplicationCookie);
        return userIdentity;
    }
}

```

Llegados a este punto, ya hemos añadido al modelo MVC y al controlador de nuestra BBDD el campo Apellido, pero aun no hemos informado a CodeFirst que existe un cambio en nuestro modelo y que tiene que modificar la BBDD. Para ello, deberemos actualizar la lista de migraciones mediante el siguiente código, siendo Apellido el nombre que vamos a dar a esta migración/actualización de la BBDD:

```

- Add-Migration "Apellido"

```

```
PM> Add-Migration "Apellido"
Aplicando técnica scaffolding a la migración 'Apellido'.
El código del diseñador de este archivo de migración incluye una instantánea del modelo actual de Code First. Esta instantánea se usa para calcular los cambios que se
producirán en el modelo al aplicar la técnica scaffolding a la siguiente migración. Si realiza otros cambios en el modelo y desea incluirlos en esta migración, deberá
ejecutar de nuevo 'Add-Migration Apellido' para volver a aplicar la técnica scaffolding.
PM>
```

Si todo ha ido bien, la consola mostrará el siguiente código y se habrá creado un nuevo fichero con el nombre del cambio creado dentro de la carpeta “Migrations”.

```

└─ Migrations
  └─ C# 201808131059406_Apellido.cs
  └─ C# Configuration.cs

```

Si se accede al fichero Configuration, se puede ver una sentencia como esta:

```
AutomaticMigrationsEnabled = false;
```

Esta variable nos permite decidir si CodeFirst va a actuar de forma automática o si por el contrario será de forma manual. Tenerla en “true” puede ser mas cómodo inicialmente, ya que cada vez que se ejecute y se detecte un cambio, de forma automática actualizará la base de datos. Como contrapartida no podremos hacer las actualizaciones manualmente, lo que implica que no podremos guardar diferentes versiones y ordenarlas como queramos.

Para terminar, solo necesitamos hacer efectivo el cambio que acabamos de ordenar a nuestra aplicación. Para ello utilizaremos el comando:

- Update-Database

Si todo ha ido bien, la consola nos mostrará el siguiente código.

```
PM> Update-Database
Especifique la marca '-Verbose' para ver las instrucciones SQL que se están aplicando a la base de datos de destino.
Aplicando migraciones explícitas: [201808131059406_Apellido].
Aplicando migración explícita: 201808131059406_Apellido.
Ejecutando el método Seed.
PM>
```

A partir de ahora, cada vez que la aplicación genere la BBDD, lo hará con esta modificación.

Por último, nos queda modificar la vista y la llamada a la BBDD para poder ver los cambios que hemos realizado en el modelo. En este caso, en la vista de registro (“Register.cshtml”) añadiremos un nuevo campo llamado Apellidos.

```

<div class="form-group">
    @Html.LabelFor(m => m.Apellido, new { @class = "col-md-2 control-label" })
    <div class="col-md-10">
        @Html.TextBoxFor(m => m.Apellido, new { @class = "form-control" })
    </div>
</div>

```

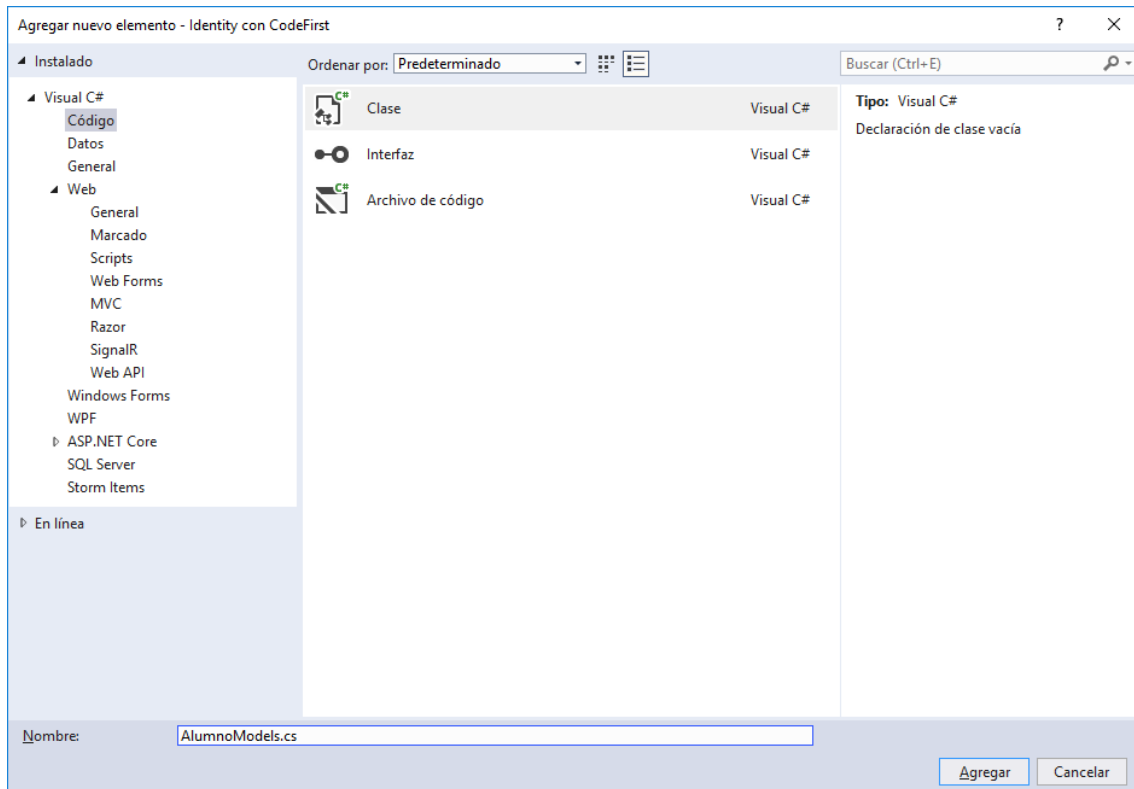
Para añadir este campo en la BBDD cada vez que rellenemos su Textbox, será necesario añadirlo en el controlador correspondiente. En este caso es en el controlador “AccountController.cs” en el método POST llamado “Register”.

```
var user = new ApplicationUser {UserName = model.Email, Email = model.Email,
Apellido = model.Apellido };
```

Una vez hemos realizado estos cambios, la próxima vez que ejecutemos nuestra aplicación recompondrá la BBDD y la adaptará a nuestro modelo.

4. Añadir tablas al modelo.

Si queremos añadir tablas al modelo de nuestra BBDD, será necesario crear una clase que posea las columnas (con sus limitaciones, tipos...) en nuestro proyecto.



```
namespace Identity_con_CodeFirst.Models
{
    public class AlumnoModels
    {
        public Guid Id { get; set; }
        public string Nombre { get; set; }
        public string Apellido { get; set; }
        public int Edad { get; set; }
    }
}
```

Una vez tenemos nuestra clase, es necesario declararlo en el contexto de la BBDD para que CodeFirst sea consciente de que hay cambios que hacer. Para ello vamos al archivo "IdentityModel.cs" y añadimos el siguiente texto dentro del contexto:

```
public DbSet<AlumnoModels> Alumno { get; set; }
```

Esto será necesario hacerlo cada vez que añadamos una tabla nueva.

Al final la clase "ApplicationDbContext" deberá quedarte parecido a esta:

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext()
        : base("DefaultConnection", throwIfV1Schema: false)
    {
    }

    public DbSet<AlumnoModels> Alumno { get; set; }
    public static ApplicationDbContext Create()
    {
    }
}
```

```

        return new ApplicationDbContext();
    }
}

```

Ahora ya solo tenemos que actualizar la BBDD mediante el comando “Update-Database” y se crearán la tabla nueva.

4.1. Pluralización de una tabla.

Cuando creéis vuestra tabla, es muy probable que en algunos casos se cambie el nombre de esta, ya que Entity Framework considera que si está en singular, no cumple las convenciones establecidas de diseño en una BBDD. Si queremos que el nombre de nuestra tabla esté en singular y que no se modifique, es necesario declararlo de manera explícita en el constructor que crea nuestro modelo de la BBDD. Para ello iremos a nuestro “IdentityModel” y dentro de nuestra clase “ApplicationDbContext” copiaremos el siguiente código:

```

protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Conventions.
        Remove<PluralizingTableNameConvention>();
    base.OnModelCreating(modelBuilder);
}

```

(*) El texto subrayado en rojo nos permite anular la pluralización en el nombre de las tablas.

(*) El texto en verde dice a c# que esta sobrescritura del método OnModelCreating va a utilizar todo lo que tiene la original, y que además va a añadir lo que pongamos aquí. Si no ponemos esto, el modelo de Identity Framework fallará.

5. Creación de tablas con relaciones.

En este apartado se va a mostrar como hacer tablas que estén relacionadas entre ellas.

5.1. Relación 1-N.

Para crear una relación vamos a necesitar la creación de otra tabla, que en este caso será Matricula.

```

public class MatriculaModels
{
    public Guid Id { get; set; }
    public string Curso { get; set; }
    public int Precio { get; set; }
}

```

Una vez tengamos la clase tenemos que declararla en el modelo “IdentityModel”.

```

public DbSet<MatriculaModels> Matricula { get; set; }

```

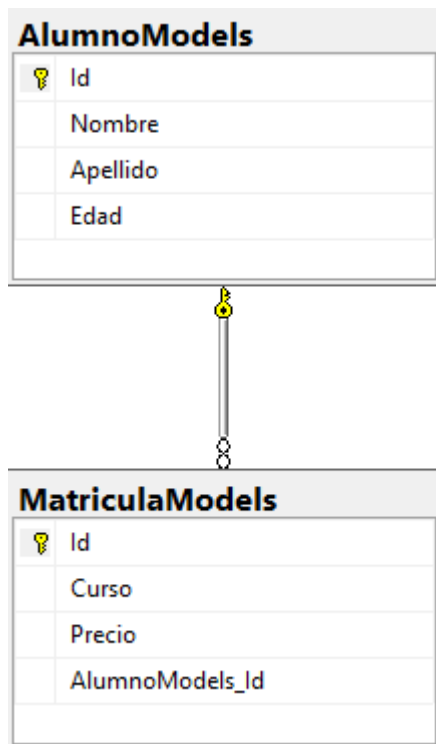
Ahora debemos modificar nuestra tabla Alumnos para que incluya una colección de matrículas:

```

public class AlumnoModels
{
    public Guid Id { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public int Edad { get; set; }
    public ICollection<MatriculaModels> IdMatricula { get; set; }
}

```

Ahora solo debemos actualizar la base de datos con el comando “Update-Database” y se crearán las tablas con su correspondiente relación:



5.2. Relación N-M.

Para crear una relación vamos a necesitar la creación de otra tabla, que en este caso será Matricula.

```

public class CursoModels
{
    public Guid Id { get; set; }
    public string Nombre { get; set; }
    public ICollection<AlumnoModels> Alumnos { get; set; }
}
  
```

Una vez tengamos la clase tenemos que declararla en el modelo "IdentityModel".

```










public DbSet< CursoModels> Curso { get; set; }
  
```

Ahora debemos modificar nuestra tabla Alumnos para que incluya una colección de cursos:

```

public class AlumnoModels
{
    public Guid Id { get; set; }
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public int Edad { get; set; }
    public virtual ICollection<Matricula> Matriculas { get; set; }
    public virtual ICollection<Curso> Cursos { get; set; }}
  
```

Ahora solo debemos actualizar la base de datos con el comando "Update-Database" y se crearán las tablas con su correspondiente relación. Recuerda que al ser una relación N-M, en la base de datos se generará una tabla intermedia:

- +  dbo.AlumnoModels
- +  dbo.AspNetRoles
- +  dbo.AspNetUserClaims
- +  dbo.AspNetUserLogins
- +  dbo.AspNetUserRoles
- +  dbo.AspNetUsers
- +  dbo.CursoAlumnoModels
- +  dbo.Cursoes
- +  dbo.MatriculaModels

5.3. Relación 1-1.

Para crear una relación 1-1 solo es necesario añadir el campo correspondiente a la otra tabla:

```
public class Usuario
{
    public Guid Id { get; set; }
    public string Nombre { get; set; }
    public Guid AlumnoId { get; set; }
    public Alumno Alumno { get; set; }
}
```

Las relaciones 1-1 no son muy frecuentes, aun así, es importante saber cómo declarar una.

6. Referencias y otras guías.

- <http://panicoenlaxbox.blogspot.com/2013/06/relaciones-en-code-first.html>
- <https://docs.microsoft.com/es-es/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-a-more-complex-data-model-for-an-asp-net-mvc-application>
- <https://blogs.msdn.microsoft.com/webdev/2013/10/16/customizing-profile-information-in-asp-net-identity-in-vs-2013-templates/>
- <https://docs.microsoft.com/es-es/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application>