



1. Notas

NO hagan esto nunca (consejos vendo, para mí no tengo), nunca accedan con 'sa'. Creen siempre un usuario o rol de aplicación específico para acceder a su base de datos desde la aplicación.

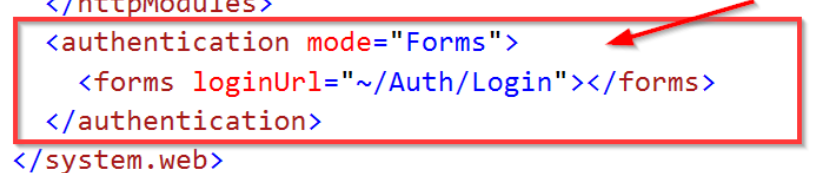
2. Puntos clave para añadir la autenticación / autorización

2.1 Configurar el proyecto para autenticación por formularios

Configuración en el fichero **web.config**.

NOTA: Cuidado hay que añadir un elemento dentro de **<system.web>** no en cualquier parte.

```
<system.web>
  <compilation debug="true" targetFramework="4.5.2">
    <httpRuntime targetFramework="4.5.2" />
    <httpModules>
      <add name="ApplicationInsightsWebTracking" type="Microsoft.ApplicationInsights.WebTracking.WebTrackingModule, Microsoft.ApplicationInsights.WebTracking, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" />
    </httpModules>
    <authentication mode="Forms">
      <forms loginUrl="~/Auth/Login"></forms>
    </authentication>
  </system.web>
```



En nuestro ejemplo al ser la URL ~/Auth/Login quiere decir que irá al método **Login** del controlador **AuthController**.

2.2 Autenticación por base de datos. Modificar el esquema de BD

Añadimos las tablas necesarias para dar soporte a la gestión de usuarios y grupos. En este ejemplo es quizá un poco más complicado porque un usuario puede pertenecer a múltiples grupos/roles.

Los cambios al esquema se encuentran al final del documento.

2.3 Añadir clases según el modelo de BD.

Se podría utilizar Entity Framework para obtener las clases a partir de las tablas, prefiero hacerlo a mano.

2.3.1 Clase Usuario

Incluye los datos del usuario, así como la colección de UsuariosGrupos dependientes.

Hemos añadido también atributos de validación, para el formulario de login (Required).



```
public class Usuario
{
    [Key]
    public int IdUsuario { get; set; }
    [Required(ErrorMessage = "Login requerido")]
    public String Login { get; set; }
    [Required(ErrorMessage = "Contraseña requerida")]
    public String Password { get; set; }

    public Usuario()
    {
        UsuarioGrupos = new HashSet<UsuarioGrupo>();
    }

    public virtual ICollection<UsuarioGrupo> UsuarioGrupos { get; set; }
}
```

2.3.2 Clase Grupo

Incluye los datos del grupo, así como la colección de UsuariosGrupos dependientes.

```
public class Grupo
{
    [Key]
    public int IdGrupo { get; set; }
    public String Nombre { get; set; }

    public Grupo()
    {
        UsuarioGrupos = new HashSet<UsuarioGrupo>();
    }

    public virtual ICollection<UsuarioGrupo> UsuarioGrupos { get; set; }
}
```



2.3.3 Clase UsuarioGrupo

Contiene, obviamente, los campos de clave ajena (FOREIGN KEY) hacia usuarios y hacia grupos.

Además define dos propiedades de navegabilidad: Usuario, Grupo, utilizando las foreign key IdUsuario e IdGrupo, respectivamente.

```
public class UsuarioGrupo
{
    [Key]
    public int IdUsuarioGrupo { get; set; }
    public int IdUsuario { get; set; }
    public int IdGrupo { get; set; }

    [ForeignKey("IdUsuario")]
    public virtual Usuario Usuario { get; set; }
    [ForeignKey("IdGrupo")]
    public virtual Grupo Grupo { get; set; }
}
```

2.4 Mapeo de las nuevas clases en nuestra clase DAL: DiscosDAL que hereda de DbContext

```
public class DiscosDAL : DbContext
{
    public DbSet<Cliente> Clientes { get; set; }
    public DbSet<Disco> Discos { get; set; }
    public DbSet<Puntuacion> Puntuaciones { get; set; }
    public DbSet<Interprete> Interpretes { get; set; }
    public DbSet<Usuario> Usuarios { get; set; }
    public DbSet<Grupo> Grupos { get; set; }
    public DbSet<UsuarioGrupo> UsuariosGrupos { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Cliente>().ToTable("Cliente");
        modelBuilder.Entity<Disco>().ToTable("Disco");
        modelBuilder.Entity<Puntuacion>().ToTable("Puntuacion");
        modelBuilder.Entity<Interprete>().ToTable("Interprete");
        modelBuilder.Entity<Usuario>().ToTable("Usuarios");
        modelBuilder.Entity<UsuarioGrupo>().ToTable("UsuariosGrupos");
        modelBuilder.Entity<Grupo>().ToTable("Grupos");

        base.OnModelCreating(modelBuilder);
    }
}
```



2.5 Implementación de lógica de login

En nuestra clase AuthController añadimos los métodos de Login. Utilizamos dos métodos, una sobrecarga de Login, dado que Login sin argumentos [GET] debe mostrar el formulario de autenticación, mientras que Login (Usuario u) [POST] va a procesar el formulario anterior para realizar la validación.

```

[HttpGet]
public ActionResult Login()
{
    return View();
}

[HttpPost]
public ActionResult Login(Usuario usuario)
{
    if (ModelState.IsValid)
    {
        Usuario authUser = null;
        using (DiscosDAL discosDAL = new DiscosDAL())
        {
            string hashedPass = ShaUtils.GenerateSHA256String(usuario.Password);
            authUser = discosDAL.Usuarios.Include(usu => usu.UsuarioGrupos.Select(y => y.Grupo)).
                FirstOrDefault(u => u.Login == usuario.Login && u.Password == hashedPass);
        }

        if (authUser != null)
        {
            FormsAuthentication.SetAuthCookie(authUser.Login, false);
            Session["USUARIO"] = authUser;
            return RedirectToAction("Index", "Admin");
        }
        else
        {
            ModelState.AddModelError("CredentialError", "Usuario o contraseña incorrectos");
            return View();
        }
    }
    else
    {
        return View();
    }
}

```

La lógica es sencilla:

1. Se crea un objeto DiscosDAL para acceso a base de datos.
2. Obtenemos el valor de la password del usuario, codificada con SHA2 256, dado que en BD la password está así codificada.
3. Hacemos una consulta a BD, buscando un usuario cuyo login y password sean iguales a los introducidos (la password codificada)

Si existe el usuario, quiere decir que la autenticación es correcta.



En la consulta nos traemos además los grupos del usuario, mediante una compleja include que trae los objetos UsuarioGrupo de Usuario y, a su vez, los objetos Grupo de cada UsuarioGrupo :-
\$

```
Include(usu => usu.UsuarioGrupos.Select(y => y.Grupo))
```

4. En el caso de que hubiésemos encontrado usuario, hacemos el login
 1. Establecemos la cookie de autenticación.
 2. Nos guardamos el objeto usuario en la sesión del usuario, dado que nos hará falta en múltiples páginas (para mostrar datos del usuario, para verificar sus roles autorizados, etc.)
 3. Pedimos /Admin/Index
5. En el caso de no encontrar el usuario volvemos al formulario con los errores pertinentes.

2.6 Implementación del logout

En el mismo AuthController añadimos el método logout. Éste es sencillo:

1. Borramos la cookie de autenticación.
2. Eliminamos la sesión de usuario, para que no quede rastro de los datos y autorizaciones del usuario.
3. Pedimos /Home/Index

3. Aplicar la autenticación

Ahora simplemente debemos decidir dónde aplicar la autenticación. Vamos a utilizar un filtro, basado en el atributo **[Authorize]**

Podemos aplicar a nivel de:

- Método: Se exigirá autenticación para ese método (action method) concreto, de un controller concreto
- Controller: Se exigirá autenticación en todos los métodos (action method) de ese controller.
- Filtro global: Se exigirá autenticación para todos los métodos (action method) de cualquier controlador.

En este último caso, al menos para el login, debemos permitir acceso anónimo (sino no se podría ni “loguear”).

[AllowAnonymous]



```
public class AuthenticationController : Controller
```

3.1 Proteger el controlador Admin

En el ejemplo hemos protegido todo el controlador Admin.

```
[Authorize]
```

```
[AdminFilter]
```

```
public class AdminController : Controller
{
    // GET: Admin
    public ActionResult Index()
    {
        return View();
    }
}
```

Lo hemos protegido con [Authorize] para exigir autenticación y, además, con [AdminFilter]. Este último es un filtro propio nuestro (clase ActionFilterAttribute) que vamos a usar para exigir, además de autenticación, que el usuario en sesión tenga el rol "admin".

El código es el siguiente:



```
public class AdminFilter : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        Usuario u = null;

        if (filterContext.HttpContext.Session["USUARIO"] is Usuario)
            u = filterContext.HttpContext.Session["USUARIO"] as Usuario;

        if (u == null || !u.UsuarioGrupos.Any(g => g.Grupo.Nombre == "admin"))
        {
            filterContext.Result = new ViewResult()
            {
                ViewName = "AuthError"
            };
        }
    }
}
```

Busca en sesión un objeto de tipo Usuario (el que habíamos metido en el Login con clave "USUARIO").

En el caso de que no se encuentre el Usuario o que éste no tenga el grupo "admin", se retorna una vista de error. Esta vista es compartida (está en Shared con el nombre "AuthError")

En caso contrario no se hace nada y se continua la ejecución.

4. Cambios en la interfaz de usuarios

En el layout por defecto hemos añadido un botón de Login/Logout en función de si hay usuario autenticado.

En el caso de que exista usuario autenticado (recuerden la cookie que habíamos establecido en Login) se mostrará el nombre del usuario (el nombre de la cookie) y el botón de logout.

En caso de que no haya usuario autenticado se mostrará el botón de registro (no implementado), así como el de Login (que nos llevará a /Auth/Login)

```
<ul class="nav navbar-nav navbar-right">
    @if (User.Identity.IsAuthenticated)
    {
        <p class="navbar-text">Bienvenido @User.Identity.Name</p>
        <li><a href="@Url.Action("Logout", "Auth")"><span class="glyphicon glyphicon-log-out"></span> Logut</a></li>
    }
    else
    {
        <li><a href="#"><span class="glyphicon glyphicon-user"></span> Sign Up</a></li>
        <li><a href="@Url.Action("Login", "Auth")"><span class="glyphicon glyphicon-log-in"></span> Login</a></li>
    }
</ul>
```



5. Script. Cambios en BD

```
USE [Discos]

GO

CREATE TABLE [dbo].[Grupos] (
    [IdGrupo] [int] IDENTITY(1,1) NOT NULL,
    [Nombre] [varchar](50) NOT NULL,
    CONSTRAINT [PK_Grupos] PRIMARY KEY CLUSTERED ([IdGrupo] ASC)
);

GO

CREATE TABLE [dbo].[Usuarios] (
    [IdUsuario] [int] IDENTITY(1,1) NOT NULL,
    [Login] [nvarchar](20) NOT NULL,
    [Password] [varchar](64) NOT NULL,
    CONSTRAINT [PK_Usuarios] PRIMARY KEY CLUSTERED ([IdUsuario] ASC)
);

GO

CREATE TABLE [dbo].[UsuariosGrupos] (
    [IdUsuarioGrupo] [int] IDENTITY(1,1) NOT NULL,
    [IdUsuario] [int] NOT NULL,
    [IdGrupo] [int] NOT NULL,
    CONSTRAINT [PK_UsuariosGrupos] PRIMARY KEY CLUSTERED ([IdUsuarioGrupo] ASC)
);

GO
```




```
ALTER TABLE [dbo].[Usuarios] ADD CONSTRAINT [AK_Usuarios_Login] UNIQUE ([Login]);
```

```
GO
```

```
ALTER TABLE [dbo].[Grupos] ADD CONSTRAINT [AK_Grupos_Nombre] UNIQUE ([Nombre]);
```

```
GO
```

```
ALTER TABLE [dbo].[UsuariosGrupos] ADD CONSTRAINT [FK_UsuariosGrupos_Grupos]  
FOREIGN KEY([IdGrupo]) REFERENCES [dbo].[Grupos] ([IdGrupo])
```

```
GO
```

```
ALTER TABLE [dbo].[UsuariosGrupos] ADD CONSTRAINT [FK_UsuariosGrupos_Usuarios]  
FOREIGN KEY([IdUsuario]) REFERENCES [dbo].[Usuarios] ([IdUsuario])
```

```
GO
```

```
ALTER TABLE [dbo].[UsuariosGrupos] ADD CONSTRAINT [AK_UsuariosGrupos_UsuGrup]  
UNIQUE ([IdUsuario],[IdGrupo])
```

```
GO
```



```
USE Discos;
```

```
GO
```

```
INSERT INTO Usuarios (Login, Password) VALUES
```

```
    ('admin', '723ac1bf7bdf89a24880ed8450dc4107889d19bf9e1ec91eeec5fa6a171ddb1f'),
```

```
    ('customer',
```

```
'723ac1bf7bdf89a24880ed8450dc4107889d19bf9e1ec91eeec5fa6a171ddb1f');
```

```
INSERT INTO Grupos (Nombre) VALUES
```

```
    ('admin'), ('customer');
```

```
INSERT INTO UsuariosGrupos (IdUsuario, IdGrupo) VALUES
```

```
    (1, 1), (2, 2);
```