



**Escuela Politécnica Superior de  
Alicante**

# **Diseño de aplicaciones web**

**Taller de ASP.net  
CORE III**

# Contenidos

- Modificar una BD existente con Migraciones de CodeFirst
- Añadir identificación de usuarios a un sitio web existente
  - Patrón de inicio/cierre de sesión
  - Otros patrones de identificación de usuarios
- Ejercicios
- Referencias

# Cambios en la base de datos

- Vamos a añadir una tabla nueva a nuestra BD, la tabla de los usuarios del sitio web.
  - Atributos: id, fechaalta, nombre, apellidos, email, password, rol y activo.
- Para ello usaremos las migraciones de 'codeFirst' incorporandolas al proyecto mediante una primera migración vacía.
- Añadiremos un nuevo modelo con anotaciones de base de datos para indicar la estructura que tendrá la tabla.
- Añadiremos una nueva migración con el nuevo modelo y actualizaremos la BD para crear la tabla.

# Cambios en la base de datos

- En la consola del gestor de paquetes 'Packet manager console', invocaremos el siguiente comando:

Add-Migration InitialCreate

- Esto creará la primera migración, que podemos ver en la carpeta 'Migrations', fichero: '2023....\_InitialCreate.cs'

- Antes de subirla a la BD, vamos a vaciar las funciones 'up' y 'down' para evitar que hagan cambios en la BD.
- Ahora podemos enviar la migración inicial vacia a la BD, con el comando:

Update-Database

- Ya tenemos la primera migración, con la tabla 'EFMigrationsHistory' en nuestra BD.

# Cambios en la base de datos

- Ahora añadimos un nuevo modelo: Usuario.cs, con los campos: id, fechaalta, nombre, apellidos, email, password, rol y activo.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace TallerCORE.Models
{
    [Table("usuarios", Schema = "tallerDAW")]
    public class Usuario
    {
        [Key]
        [Column("id", Order = 0)]
        public int id { get; set; }

        [Column("fechaalta", Order = 1)]
        public DateTime fechaalta { get; set; }

        [Display(Description = "Nombre del usuario",
            Name = "Nombre: ",
            Prompt = "Nombre del usuario",
            ShortName = "Nombre")]
        [Required(ErrorMessage = "Este campo es obligatorio")]
        [Column("nombre", Order = 2, TypeName = "varchar(50)")]
        public string nombre { get; set; }
    }
}
```

**Key** indica que es clave primaria.  
**Column** permite añadir información de como debe ser la columna en la tabla

# Cambios en la base de datos

- Nuevo modelo: Usuario.cs (2)

```
[Display(Description = "Apellidos del usuario",
    Name = "Apellidos",
    Prompt = "Apellidos del usuario",
    ShortName = "Apellidos")]
[Required(ErrorMessage = "Este campo es obligatorio")]
[Column("apellidos", Order = 3, TypeName = "varchar(100)")]
public string apellidos { get; set; }

[Display(Description = "Correo electrónico del usuario",
    Name = "Correo electrónico :",
    Prompt = "Correo electrónico del usuario",
    ShortName = "Correo electrónico")]
[Required(ErrorMessage = "Este campo es obligatorio")]
[Column("email", Order = 4, TypeName = "varchar(100)")]
public string email { get; set; }

[Display(Description = "Clave de acceso",
    Name = "Clave:",
    Prompt = "Clave de acceso",
    ShortName = "Clave")]
[Required(ErrorMessage = "Este campo es obligatorio")]
[Column("password", Order = 5, TypeName = "varchar(50)")]
public string password { get; set; }
```

# Cambios en la base de datos

- Modelo: Usuario.cs (3)

```
[Display(Description = "Rol del usuario",
    Name = "Rol: ",
    Prompt = "Rol del usuario",
    ShortName = "Rol")]
[Required(ErrorMessage = "Este campo es obligatorio")]
[Column("rol", Order = 6, TypeName = "varchar(25)")]
public string rol { get; set; }

[Display(Description = "Usuario activo?",
    Name = "Activo: ",
    Prompt = "Usuario activo",
    ShortName = "Activo")]
[Required(ErrorMessage = "Este campo es obligatorio")]
[Column("activo", Order = 7, TypeName = "varchar(1)")]
public string activo { get; set; }
```

# Cambios en la base de datos

- Modelo: Usuario.cs (y 4)

```
public Usuario(int i=0, DateTime? f=null, string n="", string a="", string e="", string p="",
    string r="USER", string ac="S") {
    id = i;
    fechaalta = f!=null?f.Value:DateTime.Now;
    nombre = n;
    apellidos = a;
    email = e;
    password = p;
    rol = r;
    activo = ac;
}

public Usuario(int id, DateTime fechaalta, string nombre, string apellidos,
    string email, string password, string rol, string activo) {
    this.id = id;
    this.fechaalta = fechaalta;
    this.nombre= nombre;
    this.apellidos = apellidos;
    this.email = email;
    this.password = password;
    this.rol = rol;
    this.activo = activo;
}
```

Añadimos dos constructores, el primero con argumentos con valores por defecto. El segundo nos lo pide CodeFirst para crear la migración

# Cambios en la base de datos

- Debemos modificar el fichero con nuestro DbContext, añadiendo la nueva entidad:

```
public BDTallerCore(DbContextOptions<BDTallerCore> options)
    : base(options)
{
}

public virtual DbSet<Categoria> Categorias { get; set; }
public virtual DbSet<Producto> Productos { get; set; }
public virtual DbSet<Usuario> Usuarios { get; set; }
```

- A continuación, añadimos una nueva migración con: Add-Migration Usuarios
- Esto generará un nuevo fichero en la carpeta 'Migrations', llamado: '2023...\_Usuarios.cs'.

# Cambios en la base de datos

- Finalmente, actualizamos la BD con Update-Database
- Y comprobamos que ha creado la nueva tabla 'Usuarios' en el servidor de BD.

+ [grid] dbo._EFMigrationsHistory
+ [grid] dbo.Categorias
+ [grid] dbo.Productos
- [grid] tallerDAW.usuarios
[folder] Columns
id (PK, int, not null)
fechaalta (datetime2(7), not null)
nombre (varchar(50), not null)
apellidos (varchar(100), not null)
email (varchar(100), not null)
password (varchar(50), not null)
rol (varchar(25), not null)
activo (varchar(1), not null)

# Configurar el proyecto

- Para añadir mecanismos de autenticación de usuarios mediante cookies y variables de sesión debemos modificar 'Program.cs', añadiendo un sistema de autenticación:

```
// Add services to the container.  
builder.Services.AddRazorPages();  
builder.Services.AddDbContext<TallerCORE.Models.BDTallerCore>();  
  
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)  
    .AddCookie(options => {  
        options.ExpireTimeSpan = TimeSpan.FromMinutes(60);  
        options SlidingExpiration = true;  
        options.LoginPath = "/Account/Login";  
        options.AccessDeniedPath = "/Account/Login";  
    });  
  
var app = builder.Build();  
    app.UseRouting();  
  
    app.UseAuthentication();  
    app.UseAuthorization();  
  
    app.MapRazorPages();  
  
    app.Run();
```

## Opciones:

- Sesión de 60 minutos
- Caducidad de sesión 'deslizante'
- Rutas de inicio de sesión y de acceso denegado.

Añadir 'UseAuthentication' antes de 'UseAuthorization'

# Configurar el proyecto

- Ahora, vamos a poner 'tras el login' las dos opciones del CRUD de productos que no deben tener acceso anónimo: Nuevo producto y eliminar producto.
- Para ello, añadiremos el atributo 'authorize' a las páginas 'Nuevo' y 'Borrar', concretamente modificaremos los ficheros 'Nuevo.cshtml.cs' y 'borrar.cshtml.cs' de la carpeta 'Catalogo':

```
namespace TallerCORE
{
    [Authorize]
    public class NuevoModel : PageModel
    {
        private readonly TallerCORE.Models.BDTallerCore _context;

        public NuevoModel(TallerCORE.Models.BDTallerCore context)
        {
            _context = context;
        }
    }
}
```

```
namespace TallerCORE
{
    [Authorize]
    public class BorrarModel : PageModel
    {
        private readonly TallerCORE.Models.BDTallerCore _context;

        public BorrarModel(TallerCORE.Models.BDTallerCore context)
        {
            _context = context;
        }
    }
}
```

# Patrones de identificación de usuario

- Registro de usuarios
- Captchas
- Inicio de sesión (login)
- Cierre de sesión (logout)
- Logout automático
- Mi perfil (ver y modificar)
  - Cambio de contraseña
- Recuperación de contraseña

# Inicio de sesión (login)

- Vamos a implementar el patrón de inicio de sesión.
- Primero crearemos una carpeta nueva dentro de 'Pages' que llamaremos 'Account', y en ella añadiremos una nueva página 'Login', en blanco.

```
@page
@model TallerCORE.Pages.Account.LoginModel
 @{
    ViewData["Title"] = "Inicio de sesión";
}
@if (ViewData["error"]!=null && ViewData["error"].ToString()!="")
{
    <div class="alert alert-danger">Error: @ViewData["error"].ToString()</div>
}
<form asp-page=".~/Login" method="post" class="form">
    <div class="form-group row">
        <label for="usu" class="form-label col-md-1">Usuario:</label>
        <div class="col-md-3">
            <input type="text" name="usu" id="usu" class="form-control" required />
        </div>
    </div><div class="form-group row">
        <label for="pwd" class="form-label col-md-1">Clave:</label>
        <div class="col-md-3">
            <input type="password" name="pwd" id="pwd" class="form-control" />
        </div>
    </div>
    <div class="form-group row">
        <div class="offset-md-3">
            <input type="hidden" name="returnURL" value="@ViewData["returnURL"]" />
            <input type="submit" class="btn btn-primary" value="Acceder" />
        </div>
    </div>
</form>
```

En ella diseñaremos un formulario de login. Añadiremos una caja tipo alert en caso de recibir un error por medio del ViewData. Enviaremos como un hidden el valor de 'returnURL'

# Inicio de sesión (login)

- El code behind de la página de login:

```
namespace TallerCORE.Pages.Account
{
    public class LoginModel : PageModel
    {
        private readonly TallerCORE.Models.BDTallerCore _context;

        public LoginModel(TallerCORE.Models.BDTallerCore context)
        {
            _context = context;
        }

        public void OnGet(string returnUrl = "/Index")
        {
            ViewData["returnURL"] = returnUrl;
        }

        public async Task<IActionResult> OnPostAsync(string usu, string pwd, string returnUrl = "/Index")
        {
            if (!ModelState.IsValid)
            {
                ViewData["error"] = "Error en los datos enviados";
                return Page();
            }

            ViewData["returnURL"] = returnUrl;
        }
    }
}
```

Declaramos el DBContext.  
Lo 'inyectamos' en el constructor.  
El OnGet obtiene la URL de retorno y la envia a la vista.  
El OnPost, recibe los datos del formulario de login.

# Inicio de sesión (login)

- El code behind de la página de login: (2)

```
if (ValidateLogin(usu, pwd)) {
    var identity = new ClaimsIdentity(CookieAuthenticationDefaults.AuthenticationScheme,
        ClaimTypes.Name, ClaimTypes.Role);
    identity.AddClaim(new Claim(ClaimTypes.NameIdentifier, usu));
    identity.AddClaim(new Claim(ClaimTypes.Name, usu));
    identity.AddClaim(new Claim(ClaimTypes.Role, "Usuario"));

    var principal = new ClaimsPrincipal(identity);
    var authProperties = new AuthenticationProperties {
        AllowRefresh = true,
        ExpiresUtc = DateTimeOffset.Now.AddDays(1),
        IsPersistent = true,
    };

    await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme,
        new ClaimsPrincipal(principal), authProperties);

    if (Url.IsLocalUrl(returnURL)) {
        return RedirectToPage(returnURL);
    } else {
        return Redirect("/Index");
    }
} else {
    ViewData["error"] = "Error, credenciales inválidas";
}
return Page();
```

Validamos el login y autentificamos el usuario para al final volver a la return URL.  
Si la validación no funciona, volvemos al formulario de login con un mensaje de error.

# Inicio de sesión (login)

- El code behind de la página de login: función de validación del usuario:

```
private bool ValidateLogin(string usu, string pwd)
{
    bool res = false;
    if (!string.IsNullOrEmpty(usu) && !string.IsNullOrEmpty(pwd))
    {
        Usuario? user = _context.Usuarios.Where(u => u.activo=="S" && u.email == usu && u.password == pwd).FirstOrDefault();
        if (user != null)
        {
            HttpContext.Session.SetString("usuario", user.email);
            HttpContext.Session.SetString("nombreUsuario", user.nombre + " " + user.apellidos);
            HttpContext.Session.SetString("rol", user.rol);
            res = true;
        }
    }
    return res;
}
```

Accedemos a la BD y comprobamos que sea el mismo usuario. Si es así, creamos unas variables de sesión con información del mismo. Para trabajar con variables de sesión debemos modificar el 'Program.cs':

```
builder.Services.AddSession();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    // The default HSTS value is 30 days
    app.UseHsts();
}

app.UseSession();
app.UseHttpsRedirection();
app.UseStaticFiles();
```

# Adaptar el proyecto

- Vamos a crear una vista parcial adyacente al menú que muestre información sobre el inicio de sesión, de forma que:
  - Si el usuario NO ha iniciado sesión, mostrará un acceso al formulario de login: Acceder.
  - Si el usuario SI ha iniciado su sesión, mostraremos un saludo: 'Hola, nombreUsuario' y un botón para cerrar sesión: Salir.
- Además, vamos a ocultar el botón de borrar un artículo en la tabla del catálogo de forma que solo esté visible si el usuario ha iniciado una sesión y es de rol 'ADMIN'.

# Adaptar el proyecto

- Creamos una nueva vista de Razor vacia dentro de 'Shared', que llamaremos '\_userInfo.cshtml':

```
@using Microsoft.AspNetCore.Http  
@inject IHttpContextAccessor HttpContextAccessor  
  
    <div class="float-end">  
        @if (User.Identity.IsAuthenticated)  
        {  
            <form asp-page="/Account/Logout" method="post">  
                <span>Hola, @HttpContextAccessor.HttpContext.Session.GetString("nombreUsuario")</span>  
                <input type="submit" class="btn btn-sm btn-warning" value="Salir" />  
            </form>  
        } else {  
            <a class="btn btn-sm btn-info" asp-area="" asp-page="/Account/Login">Acceder</a>  
        }  
    </div>
```

Necesitamos usar 'HttpContextAccessor' para poder acceder a contenidos de la sesión.

Para hacer el cierre de sesión 'levantaremos' un formulario que envie una petición 'post'.

# Adaptar el proyecto

- Debemos añadir el servicio HttpContextAccesor en el fichero 'Program.cs':

```
builder.Services.AddSingleton  
< IHttpContextAccessor, HttpContextAccessor>();
```

- Añadimos al final de la vista parcial '\_menu.cshtml' de la carpeta 'shared', una llamada a la nueva vista parcial:

```
<partial name="_userinfo.cshtml" />
```

# Adaptar el proyecto

- Finalmente, en la vista del catálogo, donde mostramos el botón de eliminar un artículo:

```
<td class="text-center">
    @Html.DisplayFor(modelItem => item.CategoríaNavigation.Nombre)
</td>
<td>
    <div class="btn-group">
        <!--<a asp-page=".Edit" asp-route-id="@item.Id">Edit</a-->
        <a asp-page=".DetalleProducto" asp-route-id="@item.Id" class="btn btn-info">Detalles</a>
        @if (User.Identity.IsAuthenticated &&
            HttpContextAccessor.HttpContext.Session.GetString("rolUsuario")=="ADMIN")
        {
            <a asp-page=".Borrar" asp-route-id="@item.Id" class="btn btn-danger">Eliminar</a>
        }
    </div>
</td>
```

Si el usuario esta 'autenticado' y su rol es 'ADMIN', mostramos el botón de borrar.

- Debemos injectar el objeto `HttpContextAccessor`

```
@using Microsoft.AspNetCore.Http
@inject IHttpContextAccessor HttpContextAccessor
```

# Patrón de logout

- El enlace para cerrar la sesión accederá a una nueva página llamada 'Logout' que crearemos dentro de la carpeta 'Account'.
- Esta página únicamente contendrá código tal que:

```
namespace TallerCORE.Pages.Account
{
    public class LogoutModel : PageModel
    {
        public void OnGet()
        {
            HttpContext.Session.Clear();
            HttpContext.SignOutAsync();
            Redirect("/Index");
        }

        public async Task<IActionResult> OnPostAsync()
        {
            HttpContext.Session.Clear();
            await HttpContext.SignOutAsync();
            return RedirectToPage("/Index");
        }
    }
}
```

Capturamos tanto el 'get' como las llamadas por 'post'.

Limpiamos la sesión y todo su contenido.  
Eliminamos la identificación del usuario actual y volvemos a la página inicial del sitio web.

# Ejercicios

1. Implementa el patrón de **mi perfil** y el de **cambio de contraseña**. Para ello, **codifica las contraseñas** (hash) al crear/modificar el usuario y al iniciar la sesión.
2. Implementa el patrón de **registro de usuario** (rol 'CLIENTE' y activo='N') de forma que el usuario administrador tenga una **opción para ver los usuarios del sistema y activar o desactivarlos**.
3. Extra: Implementa el patrón de **recuperar mi contraseña**.

# Referencias

- Migraciones con code first sobre una BD existente:
  - <https://learn.microsoft.com/es-es/ef/ef6/modeling/code-first/migrations/existing-database>
- Data annotations para generar objetos de BD:
  - <https://www.entityframeworktutorial.net/code-first/dataannotation-in-code-first.aspx>
- Migraciones:
  - <https://learn.microsoft.com/es-es/ef/ef6/modeling/code-first/migrations/>

# Referencias

- Herramientas de identificación de usuarios:
  - <https://learn.microsoft.com/en-us/aspnet/core/security/authentication/?view=aspnetcore-6.0>
- Ejemplos de código de Securizar sitios web con ASP.net CORE:
  - <https://github.com/dotnet/aspnetcore/tree/main/src/Security/samples>