educación IT
CODE YOUR FUTURE

Programación Web .NET Core

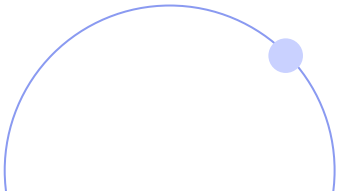
Módulo 10

Autorización en ASP.NET Core

Autorización

En muchas aplicaciones, será necesario evitar que algunos usuarios accedan a ciertos recursos, páginas, datos, o acciones específicas. Puedes manejar este acceso mediante la ***Infraestructura de autorización*** **ASP.NET Core**.

> **Práctica recomendada:** si bien es posible realizar la lógica de autorización confiando completamente en **`SignInManager`** y sus diversas funciones, los sistemas de autenticación integrados son mucho más simples y ofrecen una gran cantidad de opciones.



La autorización ASP.NET Core permite crear e implementar diferentes tipos de autorización dentro de la aplicación y utilizarlos para limitar el contenido según sea apropiado. Por ejemplo, se puede bloquear una página de la aplicación de una tienda. Mediante una autorización simple, podrías evitar que se muestren productos para adultos a usuarios más jóvenes, al establecer la autorización de ***Claims***. También, puedes permitir que solo los administradores accedan a los detalles financieros de un sitio web mediante la función de autorización.

Configuración de la autorización

La autenticación también es relativamente simple de configurar. Requiere solo un pequeño cambio en la llamada a **`ConfigureServices`**.

En lugar de llamar al método cuyo nombre es **`AddDefaultIdentity<*User*>()`** en el **`IServicesCollection`**, necesitarás llamar al parámetro **`AddIdentity<*User*, *Role*>()`**. El parámetro de usuario debe permanecer igual, sin embargo, el parámetro de rol es el rol de la clase Identity o una clase derivada de **`IdentityRole`**. La clase **`IdentityRole`** está diseñada para administrar roles, dentro de la aplicación, de manera similar a como **`IdentityUser`** gestiona usuarios y se utiliza para iniciar la clase llamada **`RoleManager`**. Sin embargo, a diferencia de **`IdentityUser`**, que frecuentemente es extendido, **`IdentityRole`** no se extiende y la clase base se usa con mayor frecuencia. Aunque los nombres **`AddDefaultIdentity`** y **`AddIdentity`** son similares, operan de manera muy diferente y si se desea utilizar la lógica ASP.NET Core MVC junto con la autorización, se deberá usar **`AddIdentity`**.

El siguiente código es un ejemplo de cómo llamar a **`AddIdentity`**:

```
services.AddIdentity<WebsiteUser, IdentityRole>()
.AddEntityFrameworkStores<AuthenticationContext>();
```

Nota: Si deseas seguir utilizando **`AddDefaultIdentity`** en tus aplicaciones, puedes canalizar una llamada al denominado **`AddRoles<*Role*>()`** después de la llamada al denominado **`AddDefaultIdentity`**. Debes tener en cuenta que esto requiere que administres la autenticación vía **ASP.NET Core RazorPages**.

Autorización simple

La forma más básica de autorización gira en torno al bloqueo de usuarios que no han iniciado sesión para acceder a determinadas páginas o realizar acciones específicas. A esto se le llama ***autorización simple*** y redirige automáticamente a la acción **`Account\Login`**, junto con una **URL de retorno** para una fácil reconexión. Permite bloquear fácilmente controladores completos o acciones individuales y evita tener que configurar redirecciones manualmente. Para implementar la autorización simple, todo lo que se necesita es agregar el atributo **`[Authorize]`**.

El atributo `[Authorize]` se puede agregar a una clase de controlador, evitando que usuarios no autorizados accedan a la clase mientras no haya iniciado sesión. Los usuarios que no hayan iniciado sesión que intenten navegar hasta el controlador serán redirigidos a la **URL *Login***. El atributo **`[Authorize]`** no se limita a la clase de controlador, también se puede agregar a acciones individuales. Esto puede ser valioso al configurar un controlador que se desea que sea de acceso público pero con acciones específicas bloqueadas a los usuarios que aún no han iniciado sesión.

> **Por ejemplo:** la navegación en una tienda web con acceso público que requiere iniciar sesión cuando se realiza una compra.

El atributo [AllowAnonymous]

En forma alternativa, es posible que desees limitar la mayoría de las acciones de un controlador, mientras **permites una o dos excepciones**. El atributo **`[AllowAnonymous]`** habilita acceso a la acción en sí, mientras que el resto del controlador no es accesible.

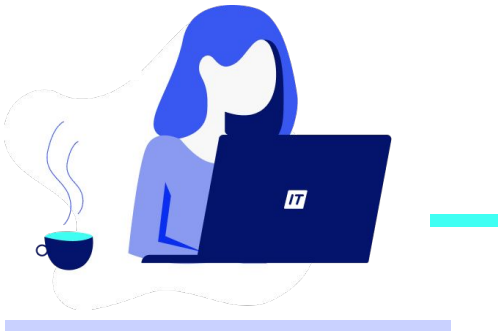
> **Por ejemplo:** una página web que permite ver datos, al tiempo que requiere iniciar sesión para realizar cambios.

El código de la siguiente pantalla es un ejemplo de **autorización simple en un controlador**.

```
[Authorize]
public class AuthorizedController : Controller
{
private readonlyUserManager<WebsiteUser> _userManager;
public AuthorizedController(UserManager<WebsiteUser>userManager)
{
_userManager = userManager;
}
public asyncTask<IActionResult>Index()
{
WebsiteUser user = await _userManager.FindByNameAsync(User.Identity.Name);
if (user != null)
{
UserDisplayViewModelmodel = new UserDisplayViewModel
{
UserHandle = user.UserHandle,
UserName = user.UserName
};
return View(model);
}
return View();
}
}
```

En este ejemplo, **el requisito de autorización está en el propio controlador**. Todas las acciones en el controlador son bloqueadas a menos que el usuario esté autenticado.

El siguiente es un código de autorización simple en una acción. Solo la acción **BuyProduct** requiere autenticación. Los usuarios pueden ver los productos libremente en la tienda, pero deben estar autenticados para comprar un producto.



```
public class ProductController : Controller
{
private IShop _shop;
public ProductController(IShop shop)
{
_shop = shop;
}
public IActionResultIndex()
{
return View(_shop.GetAllProducts());
}
public IActionResultGet(intproductId)
{
return View(_shop.GetProduct(productId));
}
[Authorize]
public IActionResultBuyProduct(intproductId)
{
_shop.PurchaseProduct(productId);
return View();
}
}
```

El siguiente código es un ejemplo del llamado **AllowAnonymous** en un controlador con autorización. En este ejemplo, se crea un controlador de libro de cocina. Todos los usuarios podrán obtener la lista de recetas, pero no podrán acceder a las páginas de recetas individuales. Solo los usuarios autenticados podrán ver recetas individuales o agregar nuevas.

```
[Authorize]
publicclassRecipeController : Controller
{
privateICookbook _cookbook;
publicRecipeController(ICookbookcookbook)
{
_cookbook = cookbook;
}
[AllowAnonymous]
publicIActionResultIndex()
{
return View(_cookbook.GetAllRecipes());
}
publicIActionResultGet(intproductId)
{
return View(_cookbook.GetRecipe(productId));
}
publicIActionResultAddRecipe(Reciperecipe)
{
_cookbook.AddRecipe(recipe);
return View();
}
}
```

Opciones de autorización

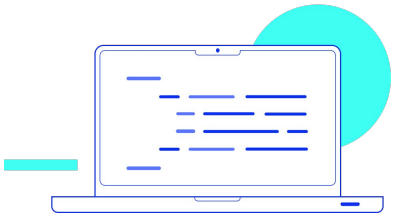
Opciones de autorización

Como hemos visto, la forma más básica de autorización implica **impedir el acceso a los usuarios que no han iniciado sesión**.

Autorización basada en roles

Una forma simple y común de autorización es la ***seguridad basada en roles***, que determina a qué pueden acceder los usuarios en función de sus roles dentro del sistema. No hay roles predeterminados definidos dentro de las aplicaciones ASP.NET Core, sin embargo, puedes crear roles que se adapten a los requisitos de tus aplicaciones. El primer paso para trabajar con roles es crearlos dentro del sistema. Esto se puede hacer utilizando el servicio **`RoleManager<* Role *>`**, que es instanciado por la llamada a **`AddIdentity`** dentro del método **`ConfigureServices`**.

El **`RoleManager`** será del tipo proporcionado para el rol en la llamada a **`AddIdentity`**, generalmente **`IdentityRole`**, y se puede inyectar en toda la aplicación, lo que permite administrar roles.



Creación de roles

Para crear un rol, deberás llamar al método **`CreateAsync (* role *)`** en el administrador de roles. El parámetro role espera un rol del mismo tipo que está configurado por el método **`AddIdentity`**. Crea una instancia del rol básico, todo lo que necesitas es proporcionar un nombre de rol. Puedes, en cualquier momento, comprobar si existe un rol determinado llamando al denominado **`RoleExistsAsync(*role name*)`**. Este acepta un nombre de cadena y comprueba si este rol específico ya se ha creado.

Nota: Generalmente, puedes crear roles en cualquier punto de la aplicación, pero se recomienda que lo hagas como parte de la instanciación de la base de datos, o en el método Configure. Si tienes la intención de crearlos en el método Configure, asegúrate de que la base de datos exista antes de crear roles.

El siguiente código es un ejemplo de un método en la clase de **`startup`** para configurar roles:

```
public async void CreateRoles(RoleManager<IdentityRole>roleManager)
{
string[] roleNames = { "Administrator", "Manager", "User" };
foreach (varroleName in roleNames)
{
boolroleExists = awaitroleManager.RoleExistsAsync(roleName);
if (!roleExists)
{
IdentityRole role = new IdentityRole();
role.Name = roleName;
awaitroleManager.CreateAsync(role);
}
}
}
```

En este ejemplo, se crean tres roles en la aplicación. Antes de crear cada rol, si ya existe uno, por ejemplo, si la base de datos se creó en un momento anterior, no intentar recrear el rol. Esto también puede permitirte agregar roles adicionales en el futuro. Se crearán entonces los roles de **`"Administrador"`**, **`"Gerente"`** y **`"Usuario"`**.

Asignar roles a los usuarios

Puedes otorgar uno o más roles a un usuario. Esto se puede hacer en el momento del registro o mediante diferentes procesos. Por ejemplo, es posible que quieras ofrecer a todos los usuarios que se registren en un rol **`"Usuario"`**, pero otorgues solo privilegios de administrador a otro usuario. Para otorgar un rol a un usuario, deberás llamar al método **`AddToRoleAsync(*user*, *role name*)`**. Este método espera un usuario del tipo definido por el administrador de usuarios, así como una cadena con el nombre del rol que es utilizado para conectar al usuario a un rol existente. Si el rol no se ha creado, el usuario no será asignado a él.

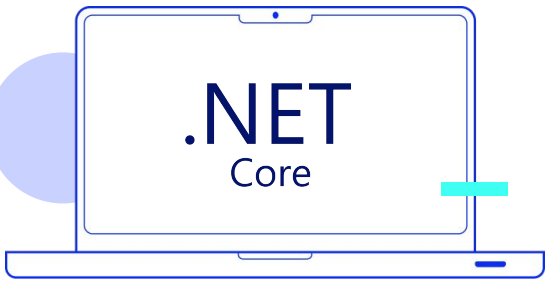
El código de la próxima pantalla es un ejemplo de asignación de roles a un usuario.

```
[HttpPost]
oublic asyncTask<IActionResult>Register(RegisterViewModelmodel)
{
if (ModelState.IsValid)
{
WebsiteUser user = new WebsiteUser
{
UserHandle = model.UserHandle,
UserName = model.Username,
Email = model.Email
};
var result = await _userManager.CreateAsync(user, model.Password);
if (result.Succeeded)
{
var addedUser = await _userManager.FindByNameAsync(model.Username);
await _userManager.AddToRoleAsync(addedUser, "Administrator");
await _userManager.AddToRoleAsync(addedUser, "User");
return awaitLogin(model);
}
}
return View();
}
```

En este ejemplo, puede ver que se crea un nuevo usuario y se asigna tanto al **"Usuario"** como Roles de **"administrador"**.

Autorizar con roles

Para autorizar con un rol, puedes usar el atributo **`Authorize`**. Debes proporcionar un ***parámetro de rol* para el atributo**, de esta manera se determina qué rol o roles pueden acceder a la clase o al método decorado por ese atributo. **Este parámetro es siempre una *cadena* y debe coincidir con el nombre de uno de los roles que se han creado**. Si más de un rol puede acceder a la clase o al método, se pueden separar todos los valores de rol válidos por **una coma**.



El siguiente código es un ejemplo de **autorización de funciones**:

```
[Authorize(Roles = "Administrator")]
publicasyncTask<IActionResult>DeleteUser(stringusername)
{
varuser = await _userManager.FindByNameAsync(username);
if (user != null)
{
await _userManager.DeleteAsync(user);
}
return View();
}
```

En este caso, solo los usuarios **"Administrador"** podrán eliminar usuarios en la aplicación.

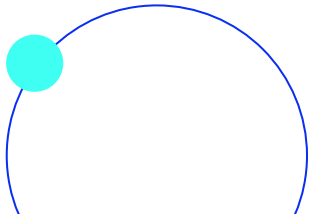
El siguiente código es un ejemplo de la **sintaxis para dar acceso a múltiples roles**:

```
[Authorize(Roles = "Administrator, Manager")]
```

Autorización basada en claims

Otro método común de autorización en ASP.NET Core es el sistema de notificaciones. **Un *Claim* es un par valor-clave que define al usuario, en vez de los permisos del usuario**. Los ejemplos de Claims incluyen correo electrónico o dirección, e incluso es posible agregar claims personalizados.

Las notificaciones se pueden declarar tal cual en una aplicación ASP.NET Core MVC con solo usar una clave y un valor, pero opcionalmente, también se puede optar por **agregar un *Emisor* a los Claims**. Esto denota de dónde proviene el Claim y puede ser utilizado para identificar la fuente de los datos. Puede resultar útil cuando tienes una aplicación que se comunica con otras aplicaciones o proveedores y deseas verificar que los datos estén certificados por una fuente confiable.



La autorización basada en Claims se utiliza para determinar si el usuario tiene un tipo específico de Claim. Esto puede ser de utilidad para cuando se necesite que el usuario ingrese detalles específicos antes de acceder a páginas específicas.

> **Por ejemplo:** puede solicitar que el usuario proporcione una dirección de correo electrónico para permitir que se registre en un boletín o puede requerir que el usuario ingrese una dirección postal para permitir el acceso a un botón de envío.

Crear un claim

Se puede hacer con la clase **`Claim`**. El constructor, **`Claim(*claimtype*, *claimvalue*)`**, recibe un tipo de Claim, que es una cadena que indica la clave del Claim y un valor que es el valor de la Claim. También se puede utilizar la enumeración **`ClaimTypes`** como primer parámetro para proporcionar una gran variedad de afirmaciones comunes. Después de haber creado un Claim, se usa el método llamado **`AddClaimAsync(*user*, *claim*)`** en el **`UserManager`** para agregar el Claim a un usuario específico. El usuario es del mismo tipo que el utilizado por el **`UserManager`**.

El código de la siguiente pantalla es un ejemplo de cómo agregar un **`Claim`** a un nuevo usuario.

```
[HttpPost]
public asyncTask<IActionResult>Register(RegisterViewModelmodel)
{
if (ModelState.IsValid)
{
WebsiteUser user = new WebsiteUser
{
UserHandle = model.UserHandle,
UserName = model.Username,
Email = model.Email
};
var result = await _userManager.CreateAsync(user, model.Password);
if (result.Succeeded)
{
var addedUser = await _userManager.FindByNameAsync(model.Username);
if (!string.IsNullOrWhiteSpace(model.Email))
{
Claimclaim = new Claim(ClaimTypes.Email, model.Email);
await _userManager.AddClaimAsync(addedUser, claim);
}
return awaitLogin(model);
}
}
return View();
}
```

Si el usuario agrega su dirección de **correo electrónico** durante el registro, se realizará una `Claim` de tipo de correo electrónico creado con el mismo valor que el correo electrónico del usuario.

Creación de una política de reclamaciones

Para autorizar reclamos, deberás crear una *política de autorización* simple en tu solicitud. La política de autorización es un conjunto de **reglas que se validan cada vez que se invoca la política**. Por ejemplo, para requerir si existe un Claim, deberás agregar una nueva política que verifique la presencia del Claim en el usuario.

Para crear una política, deberás agregar una llamada a **`AddAuthorization(*authorization optionsaction*)`** en el parámetro denominado **`IServiceCollection`** en la clase **`Startup`**.

Las opciones de autorización

La acción acepta un único parámetro llamado **`AuthorizationOptions`**, que te permite llamar a la política **`AddPolicy(*policyname*, *policy action*)`** para agregar una nueva política. El nombre de la política proporcionado a **`AddPolicy`** determina cómo se referirá a ella cuando la llames desde el atributo **`Authorize`**, mientras que la acción de la política es el código que se ejecutará siempre que sea validado. La acción recibirá un parámetro **`Authorization PolicyBuilder`** que se usa para configurar los requisitos. La política puede agregar requisitos de claim usando **`Require Claim(*claimname*,*validvalues*)`**, la propiedad del objeto **`AuthorizationPolicyBuilder`**.

El siguiente código es un ejemplo de cómo agregar una política basada en **claims**:

```
public void ConfigureServices(IServiceCollectionservices)
{
services.AddMvc();
services.AddDbContext<AuthenticationContext>(options =>
options.UseSqlite("Data Source=user.db"));
services.AddIdentity<WebsiteUser, IdentityRole>()
.AddEntityFrameworkStores<AuthenticationContext>();
services.AddAuthorization(options =>
{
options.AddPolicy("RequireEmail", policy =>
policy.RequireClaim(ClaimTypes.Email));
});
}
```

Se agrega una política de autorización denominada **"RequireEmail"**. Se autorizará a todos los usuarios que tengan un claim de **"Correo electrónico"**, mientras se bloquea a los usuarios que no lo tengan.

Los nombres se agregaron a la función **RequireClaim** como un segundo parámetro, son los valores de claim especificados que estarían autorizados.

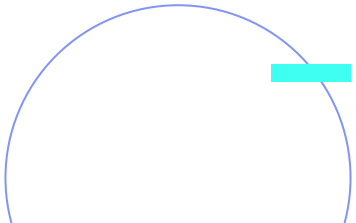
Creación de una política de reclamaciones

Después de configurar la lógica de autorización, puedes **usar el atributo `Authorize` para hacer cumplir el requisito claim**. Al proporcionar un parámetro de política al atributo, puedes asignarle un nombre de política y todas las llamadas al método o controlador que están decoradas por el atributo requerirán que el claim sea hecho.

El siguiente es un ejemplo de autorización basada en un claim:

```
[Authorize(Policy = "RequireEmail")]
publicI ActionResultIndex()
{
return View();
}
```

Se observa que la política **`RequireEmail`** se aplica en esta acción. Solo los usuarios con una reclamación por correo electrónico podrán acceder a este método.



Autorización personalizada basada en las políticas

Además de los métodos anteriores, también puedes **crear tus propias políticas dinámicas**. Estos son considerablemente más complejos de crear pero, a cambio, tenemos la **flexibilidad** que crear cualquier política que se deba exigir.

Las **políticas personalizadas** pueden recibir parámetros para validar, pero a cambio deben ser configurados desde cero.

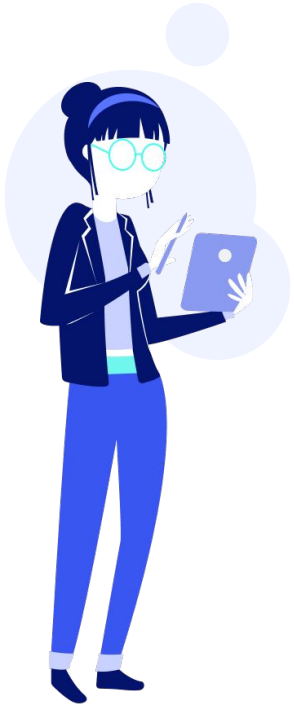
Uso de varios métodos de autorización

Puedes combinar varios métodos de autorización diferentes, agregando **múltiples atributos** llamados **`Authorize`**. Siempre que haya más de un atributo `Authorize` presente en un controlador o acción, solo será accesible si se cumplen todos los requisitos de autorización.

El siguiente es un ejemplo de varios atributos de autorización:

```
[Authorize(Policy = "RequireEmail")]
[Authorize(Roles = "Administrator")]
public IActionResultIndex()
{
return View();
}
```

En este ejemplo, el usuario solo podrá acceder a esta acción si tiene el rol de **`"Administrador"`** y cumple con la política **`"RequireEmail"`**.



Práctica recomendada: cada vez que un usuario autenticado intenta acceder a un recurso, si no tiene permisos de acceso, la aplicación ASP.NET Core MVC redirigirá la solicitud al **`"Account\AccessDenied"`**. Debes agregar una vista de controlador y un método para manejar esta ruta, y actualizar a los usuarios que no cumplen con los criterios para ver la página. Si no haces esto, los usuarios encontrarán un error del navegador.

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!