educación IT
CODE YOUR FUTURE

Programación Web .NET Core

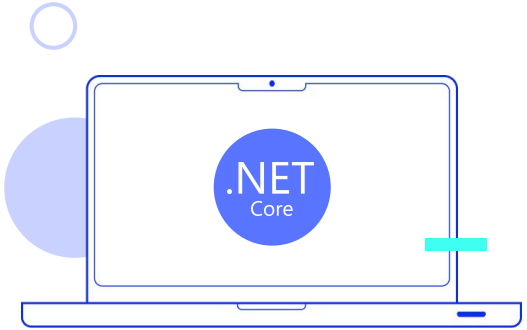
Módulo 10

educación IT
CODE YOUR FUTURE

Configuración de identidad con ASP.NET Core

Configuración de identidad con ASP.NET Core

Hasta ahora, has aprendido la configuración de extremo a extremo para **ASP.NET Core Identity**. Sin embargo, las aplicaciones que has creado utilizan la configuración predeterminada. En una aplicación real, tienes que configurar requisitos como **configuraciones de contraseña, de usuario, de sesión y más**. Muchos de estos ajustes se pueden configurar en el **`ConfigureServices`**. Como parte del método **`AddDefaultIdentity`** es capaz de recibir un parámetro **`IdentityOptions`** que se puede utilizar para configurar una variedad de ajustes adicionales.



El siguiente código es un ejemplo de cómo llamar a **`AddDefaultIdentity`** con **`IdentityOptions`**:

```
public void ConfigureServices(IServiceCollectionservices)
{
services.AddMvc();

services.AddDbContext<AuthenticationContext>(options =>
options.UseSqlite("Data Source=user.db"));

services.AddDefaultIdentity<WebsiteUser>(options =>
{
/* Configure IdentityOptionsoptionshere */
})
.AddEntityFrameworkStores<AuthenticationContext>();
}
```

Ajustes de usuario

Los ajustes de usuario se utilizan para determinar una variedad de configuraciones con respecto al registro de usuarios en el sistema. La configuración aquí se puede utilizar para requerir una dirección de correo electrónico única para cada usuario configurado o para determinar qué letras y símbolos se pueden utilizar en la aplicación. Puedes utilizar la propiedad **`User`** de **`IdentityOptions`** para configurar los ajustes relacionados con **`IdentityUser`**.

El siguiente código es un ejemplo de configuración de ajustes relacionados con **IdentityUser**:

```
services.AddDefaultIdentity<WebsiteUser>(options =>
{
options.User.RequireUniqueEmail = true;
options.User.AllowedUserNameCharacters =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 -._@+";
})
.AddEntityFrameworkStores<AuthenticationContext>();
```

En este ejemplo, puedes ver que todos los usuarios de la aplicación requerirán un valor único para la propiedad correo electrónico que establecerá, y que solo los siguientes caracteres se pueden utilizar **"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 -._@+"**en la propiedad **Username**.

Configuración de bloqueo

Una característica importante que está presente, con frecuencia, en varios sistemas que involucran autenticación es el concepto de ***bloqueo de usuarios***. **Se puede configurar que cuando la aplicación detecte varios intentos de iniciar sesión con el mismo nombre de usuario pero contraseñas no válidas, bloquee al usuario para futuros inicios de sesión durante un período de tiempo**. Esto puede evitar que fuentes maliciosas inicien sesión en el sistema mediante el uso de tácticas de fuerza bruta que involucran intentar adivinar una contraseña, aumentando enormemente el tiempo entre intentos haciéndolos inviables. Para aplicar el bloqueo, deberás asegurarte de que el parámetro **`lockoutOnFailure`** en la llamada al método **`PasswordSignInAsync`** es verdadero o se ignorará la configuración de bloqueo. Puedes utilizar la propiedad **`Lockout`** de **`IdentityOptions`** para configurar los ajustes relacionados con el bloqueo.

El siguiente código es un ejemplo de cómo habilitar el bloqueo al iniciar sesión:

```
varresult = await _signInManager.PasswordSignInAsync(model.Username, model.Password, true, true);
```

El siguiente código es un ejemplo de configuración de bloqueo en **`Identity`**:

```
services.AddDefaultIdentity<WebsiteUser>(options =>
{
options.Lockout.AllowedForNewUsers = true;
options.Lockout.MaxFailedAccessAttempts = 3;
options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(1);
})
.AddEntityFrameworkStores<AuthenticationContext>();
```

En este ejemplo, se observa que el bloqueo puede ocurrir para los nuevos usuarios que aún no han iniciado sesión correctamente. Los usuarios estarán bloqueados durante un minuto y pueden intentar iniciar sesión hasta tres veces antes de ser bloqueados.

Configuraciones de contraseña

Las **contraseñas seguras** son un requisito en los entornos de autenticación modernos y es el deber del desarrollador asignar un requisito de contraseña que sea apropiado para el entorno específico. Se puede usar la contraseña predeterminada o, a veces, encontrarás que necesitas personalizarla más. Para la **configuración y la complejidad de la contraseña**, puedes acceder a la propiedad Contraseña del **`IdentityOptions`**.

El siguiente código es un ejemplo de configuración de contraseña en **Identity**:

```
services.AddDefaultIdentity<WebsiteUser>(options =>
{
options.Password.RequiredLength = 10;
options.Password.RequiredUniqueChars = 3;
options.Password.RequireDigit = true;
options.Password.RequireNonAlphanumeric = true;
options.Password.RequireUppercase = true;
options.Password.RequireLowercase = false;
})
.AddEntityFrameworkStores<AuthenticationContext>();
```

En este ejemplo, se observa que las contraseñas para esta aplicación tendrán al menos 10 caracteres, requieren al menos tres caracteres diferentes para ser utilizados, un carácter numérico así como, al menos, un símbolo.

Finalmente, se requerirá, al menos, un carácter en mayúscula. Los caracteres en minúscula no serán requeridos y pueden omitirse.

Registrarse

En las aplicaciones modernas, se puede requerir que los usuarios proporcionen **detalles adicionales** que le permitan acceder a ellos **para verificar su identidad**. Esto puede ser útil para notificarles de posibles intentos de comprometer sus cuentas. La propiedad **`SignIn`** de **`IdentityOptions`** se puede utilizar para solicitar al usuario que confirme su correo electrónico o número de teléfono en la aplicación. No será posible iniciar sesión hasta que se confirme correctamente.

> **Nota:** La configuración de la confirmación del número de teléfono y el correo electrónico requiere trabajo adicional y no se tratará en este curso.

El siguiente código es un ejemplo de la configuración de inicio de sesión en **`Identity`**:

```
services.AddDefaultIdentity<WebsiteUser>(options =>
{
options.SignIn.RequireConfirmedEmail = true;
options.SignIn.RequireConfirmedPhoneNumber = false;
})
.AddEntityFrameworkStores<AuthenticationContext>();
```

En este ejemplo, se observa que la aplicación requiere que los usuarios confirmen el registro por correo electrónico. No es necesario confirmar los números de teléfono.

Configuración de cookies

Un tipo adicional de configuración que afecta a la **identidad** es la configuración de las ***cookies***. Con el llamado al **`ConfigureApplicationCookie`** en **`ConfigureServices`**, puedes utilizar un parámetro **`CookieAuthenticationOptions`** para establecer varias propiedades para las cookies en su aplicación. Aquí se pueden establecer varias configuraciones, como el nombre de la cookie y el vencimiento, que se aplicarán a las cookies utilizadas por la aplicación.

Nota: Se debe tener en cuenta que esta configuración afecta a las cookies de la aplicación en su conjunto y no solo a la identidad. Al usarla, se debe planificar su consecuencia.

El siguiente código es un ejemplo de configuración de cookies:

```
public void ConfigureServices(IServiceCollectionservices)
{
services.AddMvc();
services.AddDbContext<AuthenticationContext>(options =>
options.UseSqlite("Data Source=user.db"));
services.AddDefaultIdentity<WebsiteUser>(options =>
{
})

.AddEntityFrameworkStores<AuthenticationContext>();
services.ConfigureApplicationCookie(options =>
{
options.Cookie.Name = "AuthenticationCookie";
options.ExpireTimeSpan = TimeSpan.FromMinutes(30);
options.SlidingExpiration = false;
});
}
```

Se observa que las cookies se configuran por separado de **Identity** y no forman parte de ella. Las cookies están configuradas para usar el nombre **"AuthenticationCookie"** y durarán hasta 30 minutos desde el momento de la creación. No se renovarán.

educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!