Chapitre 3
ASP.Net Identity

1. Présentation

ASP.NET Identity est un composant du Framework ASP.Net qui fournit toutes les fonctions nécessaires à la gestion des utilisateurs dans une application. Il propose un ensemble de classes de modèles d'objets et de services pour gérer la création d'un compte, l'authentification, la gestion des mots de passe, des rôles, la double authentification, etc. Il propose également un interfaçage avec Entity Framework facilitant son utilisation avec des bases de données pour la persistance des données. S'il est tout à fait possible d'utiliser ASP.NET Identity avec un autre ORM (*Object Relational Mapping*), c'est toutefois fortement déconseillé car cela serait au prix de gros efforts de réimplémentation de nombreuses classes.

ASP.NET Identity a été introduit avec la version 2.0 du .Net Framework sous le nom Identity Framework. Au fil des versions, il s'est enrichi et a changé d'appellation pour devenir ASP.NET Identity.

2. Configuration

La méthode `AddDefaultIdentity()` vue dans le chapitre Création d'une application MVC prend en argument `Action<IdentityOptions>`, soit une expression lambda qui permet de définir des propriétés de la classe `IdentityOptions`.

Les différentes options disponibles, concernant la gestion de l'identité de l'utilisateur, sont regroupées dans les catégories suivantes :

– ClaimsIdentity
– Lockout
– Password
– SignIn
– Tokens
– User

Pour chaque catégorie, les différentes options seront présentées avec leur valeur par défaut.

2.1 ClaimsIdentity

Les options de la catégorie ClaimsIdentity permettent de paramétrer les types de claims. Par défaut, le système ASP.NET Identity repose sur des claims qui ressemblent à des namespaces XML. Ce standard est adopté par Microsoft, mais des standards comme le JWT ont opté pour d'autres claims dont les types sont beaucoup plus courts.

Remarque

La notion de claim, que l'on retrouve partout dans les systèmes d'authentification, et que l'on peut traduire par revendication, désigne les propriétés que l'utilisateur revendique. Cette notion est un peu abstraite mais elle deviendra plus claire au fil des exemples. Retenez pour l'instant qu'un claim est une paire clé/valeur. Par exemple, e-mail : julien@exemple.com

Voici la liste des options et de leur valeur par défaut :

```
options.ClaimsIdentity.EmailClaimType =
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"

options.ClaimsIdentity.RoleClaimType =
"http://schemas.microsoft.com/ws/2008/06/identity/claims/role"

options.ClaimsIdentity.SecurityStampClaimType =
"AspNet.Identity.SecurityStamp"

options.ClaimsIdentity.UserIdClaimType =
"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"
```

La version JWT ressemblerait à ça :

```
options.ClaimsIdentity.EmailClaimType = "email"

options.ClaimsIdentity.RoleClaimType = "role"

options.ClaimsIdentity.SecurityStampClaimType = "security_stamp"

options.ClaimsIdentity.UserIdClaimType = "sub"
```

2.2 Lockout

Les options `Lockout` concernent le verrouillage temporaire des comptes en cas de trop nombreuses tentatives de connexion infructueuses. Ceci peut permettre d'éviter le recours à des services tiers comme Google reCAPTCHA pour se prémunir contre les robots.

Remarque

Google reCAPTCHA, bien qu'utilisé à grande échelle, est un système qui est loin d'être exempt de défauts et peut nuire à votre application. Il n'est pas rare que son temps de réponse frôle ou dépasse la seconde, qu'il soit indisponible, qu'il refuse des utilisateurs pour des raisons inexpliquées, etc.

```
options.Lockout.AllowedForNewUsers = true;
```

La propriété `AllowedForNewUsers` autorise le verrouillage des comptes pour les utilisateurs nouvellement créés.

```
options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
```

La propriété `DefaultLockoutTimeSpan` définit la durée avant que l'utilisateur ne puisse tenter de se connecter à nouveau.

```
options.Lockout.MaxFailedAccessAttempts = 5;
```

Le propriété `MaxFailedAccessAttempts` définit le nombre d'essais autorisés avant verrouillage du compte.

2.3 Password

Les options `Password` permettent de définir les règles qui régissent les mots de passe des utilisateurs.

```
options.Password.RequireDigit = true;
```

La propriété `RequireDigit` indique que le mot de passe doit contenir au moins un chiffre.

```
options.Password.RequiredLength = 6;
```

La propriété `RequiredLength` indique que le mot de passe doit avoir une longueur minimale.

```
options.Password.RequiredUniqueChars = 1;
```

La propriété `RequiredUniqueChars` définit le nombre de caractères uniques que le mot de passe doit contenir.

```
options.Password.RequireLowercase = true;
```

La propriété `RequireLowercase` indique que le mot de passe doit contenir au moins un caractère en minuscule.

```
options.Password.RequireNonAlphanumeric = true;
```

La propriété `RequireNonAlphanumeric` indique que le mot de passe doit contenir au moins un caractère spécial comme @#! etc.

```
options.Password.RequireUppercase = true;
```

La propriété `RequireUppercase` indique que le mot de passe doit contenir au moins une majuscule.

2.4 SignIn

Les options `SignIn` contiennent les règles relatives à la connexion.

```
options.SignIn.RequireConfirmedAccount = false;
```

La propriété `RequireConfirmedAccount` indique qu'un compte confirmé est nécessaire pour se connecter. Un compte confirmé est un compte dont l'e-mail OU le numéro de téléphone ont été confirmés.

```
options.SignIn.RequireConfirmedEmail = false;
```

La propriété `RequireConfirmedEmail` indique qu'une adresse e-mail confirmée est nécessaire pour se connecter. La confirmation de l'e-mail se fait via l'envoi d'un lien de confirmation sur l'adresse e-mail renseignée.

```
options.SignIn.RequireConfirmedPhoneNumber = false;
```

La propriété `RequireConfirmedPhoneNumber` indique qu'un numéro de téléphone confirmé est nécessaire pour se connecter. La confirmation du numéro de téléphone se fait via l'envoi d'un code au numéro de téléphone renseigné.

2.5 Token

Les options Tokens contiennent les propriétés relatives aux différents jetons de sécurité utilisés par l'application pour l'authentification à double facteur, la récupération de mot de passe, la validation d'e-mail, etc.

```
options.Tokens.AuthenticatorIssuer =
"Microsoft.AspNetCore.Identity.UI";
```

La propriété `AuthenticatorIssuer` définit le nom de l'émetteur des tokens de sécurité.

```
options.Tokens.AuthenticatorTokenProvider = "Authenticator";
```

La propriété `AuthenticatorTokenProvider` définit le nom du provider utilisé pour valider l'authentification à double facteur avec une application d'authentification type Microsoft Authenticator. Cela correspond à la classe `AuthenticatorTokenProvider`.

```
options.Tokens.ChangeEmailTokenProvider = "Default";
```

La propriété `ChangeEmailTokenProvider` définit le nom du provider utilisé pour générer le token de changement d'adresse e-mail. Cela correspond à la classe `DataProtectorTokenProvider`.

```
options.Tokens.ChangePhoneNumberTokenProvider = "Phone";
```

La propriété `ChangePhoneNumberTokenProvider` définit le nom du provider utilisé pour générer le token de changement de numéro de téléphone. Cela correspond à la classe `PhoneNumberTokenProvider`.

```
options.Tokens.EmailConfirmationTokenProvider = "Default";
```

La propriété `EmailConfirmationTokenProvider` définit le nom du provider utilisé pour générer le token de confirmation d'adresse e-mail. Cela correspond à la classe `DataProtectorTokenProvider`.

```
options.Tokens.PasswordResetTokenProvider = "Default";
```

La propriété `PasswordResetTokenProvider` définit le nom du provider utilisé pour générer le token de réinitialisation de mot de passe. Cela correspond à la classe `DataProtectorTokenProvider`.

```
options.Tokens.ProviderMap = = new Dictionary<string,
TokenProviderDescriptor>();
```

La propriété `ProviderMap` définit un dictionnaire contenant la liste des providers ciblés par leur nom.

2.6 Stores

Les options `Stores` définissent les règles liées au stockage des utilisateurs dans une base de données.

```
options.Stores.MaxLengthForKeys = 450;
```

La propriété `MaxLengthForKeys` définit la longueur maximale utilisée pour les clés primaires dans le magasin d'utilisateurs, et par extension dans la base de données. Par défaut, ce paramètre peut être ramené à 36 car les clés d'identité sont des `Guid` convertis en `string`. La méthode `AddDefaultIdentity()` définit ce paramètre à 128. Ce paramètre n'est valable que pour les clés primaires sous forme de `string`. Nous verrons dans le chapitre ASP.Net Identity x Entity framework que celles-ci peuvent être surchargées en `int`, `long`, `Guid`, ou tout autre type qui peut être accepté par une base de données.

```
options.Stores.ProtectPersonalData = true;
```

La propriété `ProtectPersonalData` définit la protection des données personnelles de l'utilisateur afin d'être en conformité avec le RGPD. Si la valeur est à `true`, une implémentation de `IPersonalDataProtector` est requise. De plus, un décorateur `[ProtectedPersonalData]` doit être apposé sur la propriété protégée.

2.7 User

Les options `User` regroupent les options concernant l'utilisateur qui désire se connecter.

```
options.User.AllowedUserNameCharacters =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-._@+";
```

La propriété `AllowedUserNameCharacters` liste des caractères utilisables dans le nom d'utilisateur. Permet d'étendre ou de restreindre la liste des caractères utilisables. Cela peut être utile si le nom d'utilisateur est autre chose qu'un e-mail, et que l'on désire autoriser des caractères accentués dans celui-ci par exemple.

```
options.User.RequireUniqueEmail = false;
```

La propriété `RequireUniqueEmail` indique si les utilisateurs doivent avoir un e-mail unique. Ce paramètre est intéressant si le nom d'utilisateur utilisé est autre chose que l'e-mail, comme un numéro de téléphone ou un pseudonyme. Le champ username dispose, de fait, d'un contrôle d'unicité de par l'index en base de données. Dans le cas où l'e-mail est utilisé comme username, l'unicité de l'e-mail est assurée. Dans le cas contraire, deux comptes utilisateur différents pourraient utiliser le même e-mail. Cette option permet donc de spécifier si l'on désire ou non permettre l'utilisation du même e-mail pour plusieurs comptes utilisateur.

3. Rôles

La méthode `AddDefaultIdentity<T>()` ajoute tout le nécessaire pour une gestion basique des utilisateurs mais ne propose pas la gestion des rôles. Les rôles vont vous permettre de définir les droits d'accès des utilisateurs. On retrouve souvent les grands archétypes tels que "Utilisateur", "Administrateur", "Support", etc., mais vous pouvez définir vos propres rôles en fonction de vos besoins. Vous pourrez affecter autant de rôles que vous le désirez à vos utilisateurs afin de gérer les droits d'accès aussi finement que souhaité.

Pour ajouter la gestion des rôles, il faut légèrement modifier le code précédent :

```
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
options.SignIn.RequireConfirmedAccount = true)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

La classe `IdentityRole`, tout comme la classe `IdentityUser`, peut être héritée pour étendre le système.

Exemple

Je désire que mon système d'authentification utilise ASP.Net Identity avec une base de données SQL Server qui sera gérée via Entity Framework.

Pour mes utilisateurs, je désire les règles suivantes :

– Je veux que l'utilisateur confirme son adresse e-mail pour pouvoir se connecter.
– Je veux que le mot de passe de l'utilisateur contienne au moins une majuscule, une minuscule, un caractère spécial et un chiffre avec au minimum dix-huit caractères de longueur.
– Enfin, je veux que les utilisateurs aient des adresses e-mail uniques afin d'éviter les doublons.

Dans ce cas, les options seront définies ainsi, toutes les autres étant laissées à leurs valeurs par défaut.

```
var connectionString =
builder.Configuration.GetConnectionString("DefaultConnection") ??
throw new InvalidOperationException("Connection string
'DefaultConnection' not found.");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddDefaultIdentity<IdentityUser>(options =>
{
  options.SignIn.RequireConfirmedEmail = true;
  options.Password.RequireUppercase = true;
  options.Password.RequireLowercase = true;
  options.Password.RequireNonAlphanumeric = true;
  options.Password.RequireDigit = true;
  options.Password.RequiredLength = 18;
  options.User.RequireUniqueEmail = true;
})
.AddEntityFrameworkStores<ApplicationDbContext>();
```