ASP.NET MVC : gestion d'utilisateurs

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

1. Introduction
2. Le modèle
3. Le contrôleur
4. Les vues
5. Le fichier `Web.config`
6. Tester
7. Les Rôles

Gestion d'utilisateurs

Objectif

- Créer un système d'authentification
- Sécuriser l'accès à notre application web

Gestion d'utilisateurs

Étape

- Utiliser l'entité `Personne` pour la gestion des utilisateurs
- Créer les formulaires et les contrôleurs qui vont permettre à un utilisateur de s'authentifier ou de s'inscrire s'il n'a pas de compte utilisateur valide
- Interdire l'accès à certaines ressources pour les utilisateurs non authentifiés

Gestion d'utilisateurs

Avant de commencer

- Créer un nouveau projet ASP.NET `AspNetSecurity`
- Créer un contrôleur MVC vide `HomeController`

Gestion d'utilisateurs

Contenu de `HomeController`

```
namespace  AspNetSecurity.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult Both()
        {
            return View();
        }
    }
}
```

Gestion d'utilisateurs

Contenu de la vue `Index.cshtml` **définie dans** `/Views/Home`

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
        Page pour les utilisateurs ayant les rôles Admin
           ou User
    </div>
</body>
</html>
```

Gestion d'utilisateurs

Contenu de la vue `Both.cshtml` **définie dans** `/Views/Home`

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Both</title>
</head>
<body>
    <div>
        Page pour les utilisateurs ayant les rôles Admin
           et User
    </div>
</body>
</html>
```

Gestion d'utilisateurs

Objectif

Sécuriser l'accès au contrôleur `HomeController` et ses vues en fonctions des rôles attribués aux utilisateurs.

Gestion d'utilisateurs

Commençons par créer l'entité `Personne` suivante

```
namespace AspNetSecurity.Models
{
    public class Personne
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Num { get; set; }
        public string Nom { get; set; }
        [Required]
        public string Prenom { get; set; }
        public bool Sportif { get; set; }
        [Required]
        public string Password { get; set; }
    }
}
```

Gestion d'utilisateurs

Commençons par créer l'entité `Personne` suivante

```
namespace AspNetSecurity.Models
{
    public class Personne
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Num { get; set; }
        public string Nom { get; set; }
        [Required]
        public string Prenom { get; set; }
        public bool Sportif { get; set; }
        [Required]
        public string Password { get; set; }
    }
}
```

Pour les décorateurs, il faut utiliser les espaces de noms suivants :

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

Gestion d'utilisateurs

N'oublions pas de

Préparer le contexte (clic droit sur le projet et aller `Ajouter > Nouvel élément > Données > ADO.NET Entity Data Model > Modèle vide Code First`)

Gestion d'utilisateurs

Mettons à jour le contexte (la classe `Model1`)

```
public class Model1 : DbContext
{
    public Model1()
        : base("name=Model1")
    {
    }
    public virtual DbSet<Personne> Personnes { get;
       set; }
}
```

Gestion d'utilisateurs

Mettons à jour le contexte (la classe `Model1`)

```
public class Model1 : DbContext
{
    public Model1()
        : base("name=Model1")
    {
    }
    public virtual DbSet<Personne> Personnes { get;
       set; }
}
```

N'oublions pas

```
using AspNetSecurity.Models;
```

Gestion d'utilisateurs

Ensuite créons un contrôleur contenant les actions suivantes

- Une pour la connexion
- Une pour l'inscription
- Une pour la déconnexion

Gestion d'utilisateurs

`HttpContext.User` : les données sur l'utilisateur

- `HttpContext.User.Identity.Name` : retourne le nom de l'utilisateur (personne) connecté
- `HttpContext.User.Identity.IsAuthenticated` : retourne `true` si la personne s'est authentifiée

Gestion d'utilisateurs

`HttpContext.User` : les données sur l'utilisateur

- `HttpContext.User.Identity.Name` : retourne le nom de l'utilisateur (personne) connecté
- `HttpContext.User.Identity.IsAuthenticated` : retourne `true` si la personne s'est authentifiée

Pour enregistrer la personne authentifiée

- `FormsAuthentication.SetAuthCookie(chaîne,bool)` : permet de créer un cookie contenant `chaîne`. Si `bool` est à `false`, alors l'authentification n'aura qu'une durée de vie limitée à la session.
- `FormsAuthentication.SignOut()` : permet de détruire le cookie de l'authentification

Gestion d'utilisateurs

Le contrôleur `LoginController` : partie connexion (1) `GET`

```
public ActionResult Index()
{
  var Authentifie = HttpContext.User.Identity.IsAuthenticated;
  ViewData["Authentifie"] = Authentifie;
  Personne personne = null;
  if (Authentifie)
  {
     using (var db = new Model1())
     {
        personne = (from p in db.Personnes
                    where p.Nom == HttpContext.User.Identity.
                       Name
                    select p).FirstOrDefault();
     }
  }
  return View(personne);
}
```

Le contrôleur `LoginController` : partie connexion (1) `POST`

```
[HttpPost]
public ActionResult Index(Personne personne, string returnUrl) {
  Personne perso = null;
  var Authentifie = HttpContext.User.Identity.IsAuthenticated;
  ViewData["Authentifie"] = Authentifie;
  if (ModelState.IsValid) {
    using (var db = new Model1())
    {
      perso = (from p in db.Personnes
               where p.Prenom.Equals(personne.Prenom) && p.Password.
                  Equals(personne.Password)
               select p).FirstOrDefault();
      if (perso != null) {
         FormsAuthentication.SetAuthCookie(perso.Nom.ToString(),false);
         Authentifie = HttpContext.User.Identity.IsAuthenticated;
         if (!string.IsNullOrWhiteSpace(returnUrl) && Url.IsLocalUrl(
            returnUrl))
            return Redirect(returnUrl);
         return Redirect("/");
      }
    }
  }
  return View(personne);
}
```

Gestion d'utilisateurs

Le contrôleur `LoginController` : deuxième partie inscription

```
public ActionResult Inscription()
{
    return View();
}
[HttpPost]
public ActionResult Inscription(Personne personne)
{
  if (ModelState.IsValid)
  {
      using (var db = new Model1())
      {
          db.Personnes.Add(personne);
          db.SaveChanges();
      }
      FormsAuthentication.SetAuthCookie(personne.Nom.ToString(),
          false);
      return Redirect("/");
   }
   return View(personne);
```

Gestion d'utilisateurs

Le contrôleur `LoginController` : troisième partie déconnexion

```
public ActionResult Deconnexion()
{
  FormsAuthentication.SignOut();
  return Redirect("/");
}
```

La vue d'authentification `Index`

```
@{  bool test = (bool)@ViewData["Authentifie"]; }
@if (test)
{
  <h3> Utilisateur déjà connecté avec le login @Model.Prenom </h3>
  @Html.ActionLink("Cliquer pour vous déconnecter ?", "Deconnexion")
}
else
{
  <h3>Connexion :</h3>
  using (Html.BeginForm())
  {
    <div>
      @Html.LabelFor(m => m.Prenom)
      @Html.TextBoxFor(m => m.Prenom)
      @Html.ValidationMessageFor(m => m.Prenom)
    </div>
    <div>
      @Html.LabelFor(m => m.Password)
      @Html.PasswordFor(m => m.Password)
      @Html.ValidationMessageFor(m => m.Password)
    </div>
      <input type="submit" value="Connexion" /><br />
      @Html.ActionLink("Inscription", "Inscription")
  }
}
```

Gestion d'utilisateurs

La vue d'inscription `Inscription`

```
<h3>Inscription </h3>
@using (Html.BeginForm())
{
    <div>
        @Html.LabelFor(m => m.Prenom)
        @Html.TextBoxFor(m => m.Prenom)
        @Html.ValidationMessageFor(m => m.Prenom)
    </div>
    <div>
        @Html.LabelFor(m => m.Nom)
        @Html.TextBoxFor(m => m.Nom)
        @Html.ValidationMessageFor(m => m.Nom)
    </div>
    <div>
        @Html.LabelFor(m => m.Password)
        @Html.PasswordFor(m => m.Password)
        @Html.ValidationMessageFor(m => m.Password)
    </div>
    <input type="submit" value="Inscription" />
}
```

Gestion d'utilisateurs

Ajouter le code suivant dans la section `<system.web>...</system.web>` **dans** `Web.config` **(à ne pas confondre avec** `web.config` **de** `Views`**)**

```
<authentication mode="Forms">
  <forms loginUrl="~/Login/Index" />
</authentication>
```

Gestion d'utilisateurs

Pour tester, utiliser les décorateurs suivants :

- `[Authorize]` : indique que la ressource (action, contrôleur) n'est accessible qu'après authentification
- `[AllowAnonymous]` : rend une action, d'un contrôleur accessible seulement après authentification, accessible sans authentification.

Gestion d'utilisateurs

Pour tester, utiliser les décorateurs suivants :

- `[Authorize]` : indique que la ressource (action, contrôleur) n'est accessible qu'après authentification
- `[AllowAnonymous]` : rend une action, d'un contrôleur accessible seulement après authentification, accessible sans authentification.

On peut aussi autoriser l'accès à une ressource à certains utilisateurs

- utilisation de rôles : attribuer un ou plusieurs rôles pour chaque utilisateur
- Définir les rôles autorisés d'accès à une ressource : `[Authorize(Roles = "Admin")]`

Gestion d'utilisateurs

Créons une entité `Role`

```
public class Role
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }
    [Required]
    public string Title { get; set; }
    public virtual List<Personne> Personnes { get; set; }
}
```

Gestion d'utilisateurs

Modifions l'entité `Personne`

```
public class Personne
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Num { get; set; }
    public string Nom { get; set; }
    [Required]
    public string Prenom { get; set; }
    public bool Sportif { get; set; }
    [Required]
    public string Password { get; set; }
    public virtual List<Role> Roles { get; set; }
}
```

Gestion d'utilisateurs

N'oublions pas de

- Mettre à jour la classe contexte `Model1`
- Régénérer la base de données (faire la migration)

Gestion d'utilisateurs

N'oublions pas de

- Mettre à jour la classe contexte `Model1`
- Régénérer la base de données (faire la migration)

Les commandes pour la migration à exécuter dans la `Console du Gestionnaire de package`

- `Enable-Migrations`
- `Add-Migration add_tables_role_and_personnerole`
- `Update-database`

Gestion d'utilisateurs

Préparons notre fournisseur de rôles : créons une classe `CustomRoleProvider` **dans** `Configurations` **qui implémente** `RoleProvider`

```
public class CustomRoleProvider : RoleProvider
{

}
```

Plusieurs méthodes à implémenter : celles qui nous intéressent `GetAllRoles`, `GetRolesForUser` et `IsUserInRole`

Gestion d'utilisateurs

Implémentons la méthode `GetAllRoles`

```
public override string[] GetAllRoles()
{
  using (var db = new Model1())
  {
    return db.Roles.Select(r => r.Title).ToArray();
  }
}
```

Gestion d'utilisateurs

Implémentons la méthode `GetRolesForUser`

```
public override string[] GetRolesForUser(string nom)
{
  using (var db = new Model1())
  {
     var personne = db.Personnes.SingleOrDefault(u
        => u.Nom == nom);
     if (personne == null)
        return new string[] { };
     return personne.Roles == null ? new string[] {
        } :      personne.Roles.Select(u => u.Title).
        ToArray();
  }
}
```

Gestion d'utilisateurs

Implémentons la méthode `IsUserInRole`

```
public override bool IsUserInRole(string nom, string
   title)
{
  using (var db = new Model1())
  {
     var personne = db.Personnes.SingleOrDefault(u => u.
        Nom == nom);
     if (personne == null)
       return false;
     var role = (from r in db.Roles
                 where r.Title.Equals(title) && r.
                    Personnes.Any(u => u.Nom == nom)
                 select r).First();
     return role != null;
  }
}
```

Gestion d'utilisateurs

Déclarons le provider dans `Web.config` **(à ne pas confondre avec** `web.config` **de** `Views`**) dans la section** `<system.web>`

```
<system.web>
  ...
  <roleManager enabled="true" defaultProvider="CustomRoleProvider">
    <providers>
      <clear/>
      <add name="CustomRoleProvider" type="AspNetSecurity.
        Configurations.CustomRoleProvider"/>
    </providers>
  </roleManager>
```

Gestion d'utilisateurs

Déclarons le provider dans `Web.config` **(à ne pas confondre avec** `web.config` **de** `Views`**) dans la section** `<system.web>`

```
<system.web>
  ...
  <roleManager enabled="true" defaultProvider="CustomRoleProvider">
    <providers>
      <clear/>
      <add name="CustomRoleProvider" type="AspNetSecurity.
        Configurations.CustomRoleProvider"/>
    </providers>
  </roleManager>
```

`AspNetSecurity` est le nom du projet

Gestion d'utilisateurs

Pour tester

- Créer les rôles `Admin` et `User` dans la base de données (via `Explorateur d'objets SQL Server`) et attribuer ces rôles aux différents utilisateurs
- Utiliser les décorateurs suivants `[Authorize(Roles = "Admin")]` pour limiter l'accès aux ressources selon le rôle défini

Gestion d'utilisateurs

Autoriser plusieurs rôles

- `[Authorize(Roles = "Admin,User")]`
  autoriser les utilisateurs ayant le rôle `Admin` ou le rôle `User`.
- `[Authorize(Roles = "Admin")]`
  `[Authorize(Roles = "User")]`
  autoriser les utilisateurs ayant à la fois le rôle `Admin` et le rôle `User`.

Utilisons les décorateurs de rôles dans `HomeController`

```
namespace AspNetSecurity.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        [Authorize(Roles = "Admin,User")]
        public ActionResult Index()
        {
            return View();
        }

        [Authorize(Roles = "Admin")]
        [Authorize(Roles = "User")]
        public ActionResult Both()
        {
            return View();
        }
    }
}
```