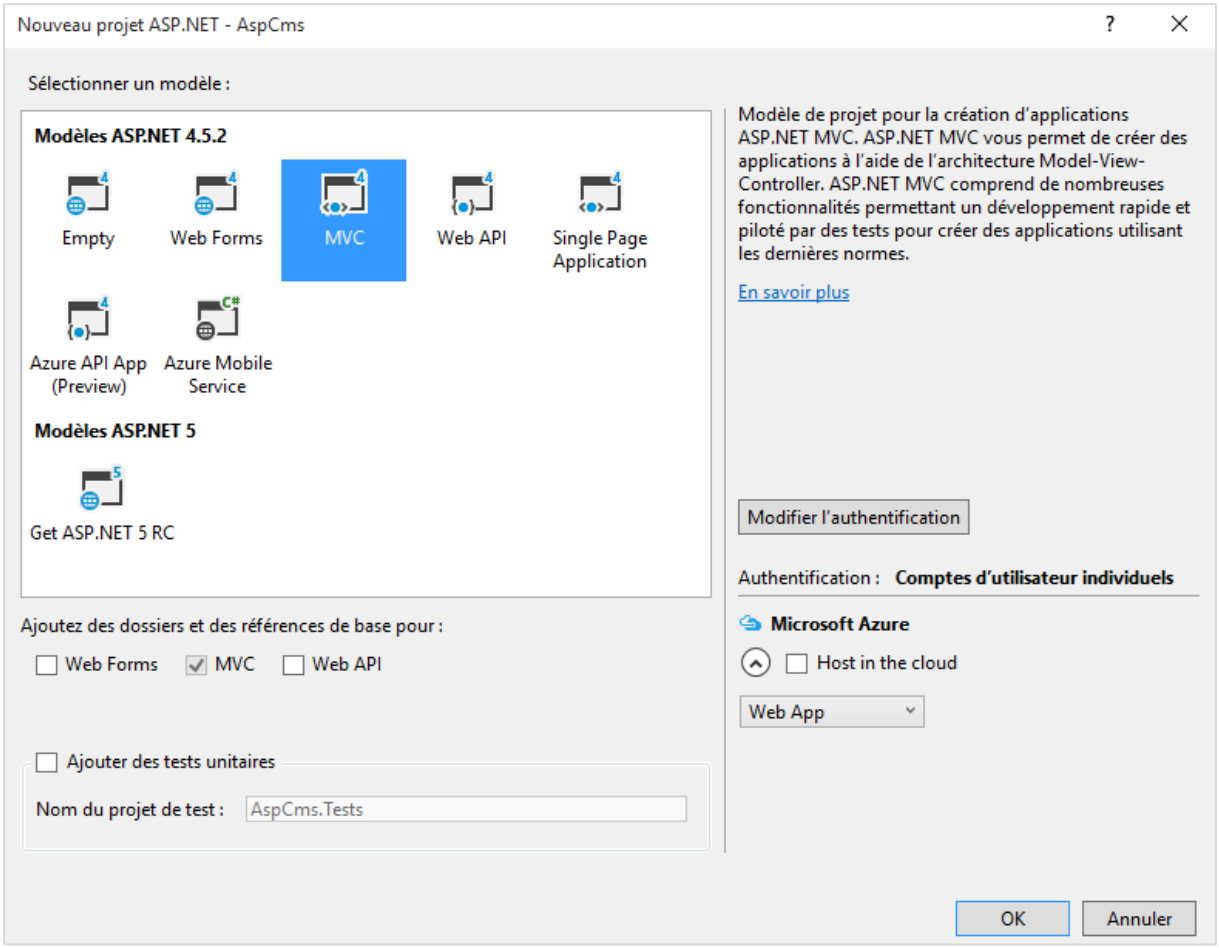
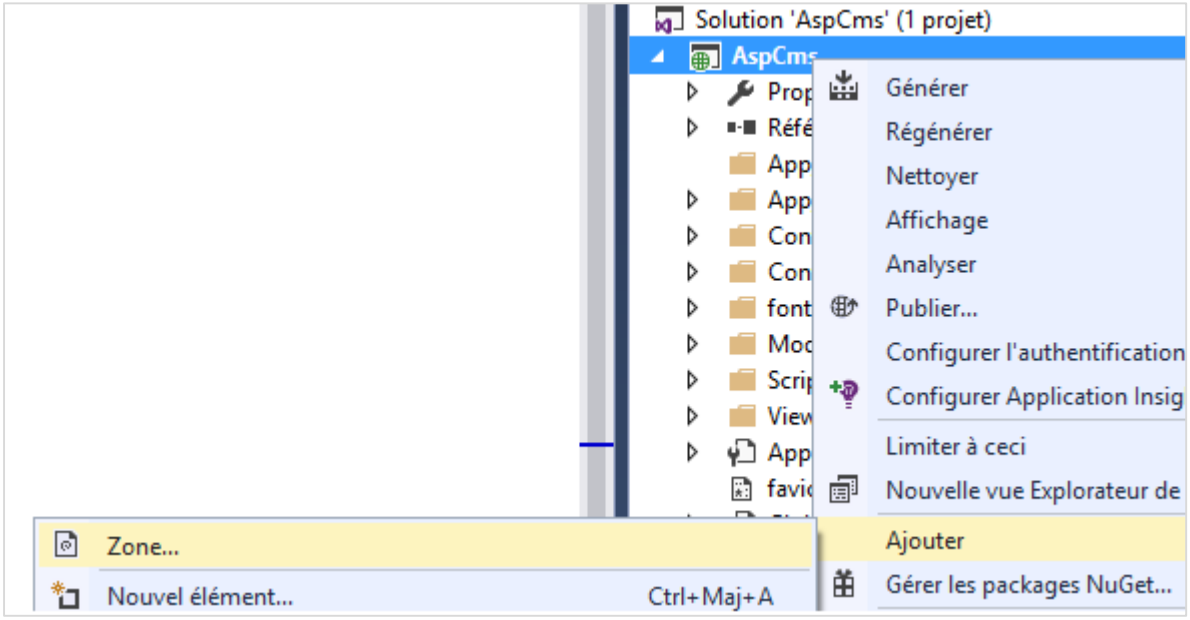
Asp.Net MVC avec Areas et ApplicationUser

Table des matières

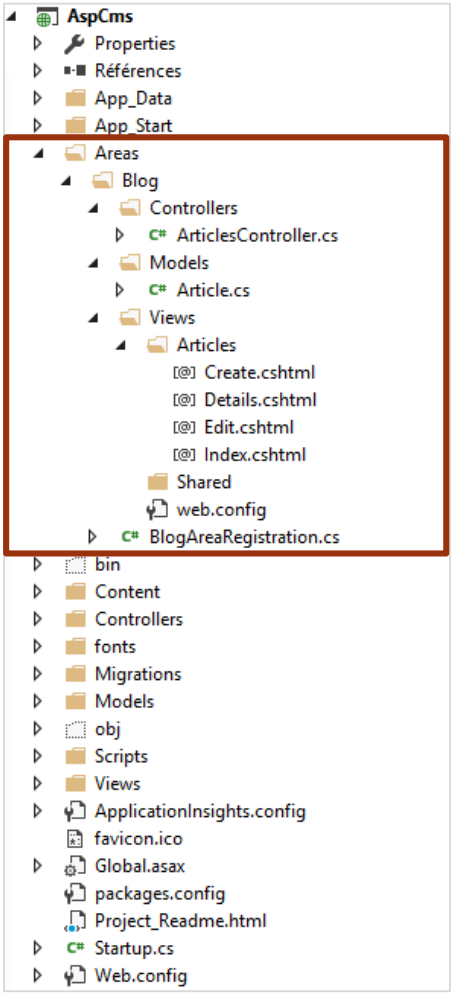
1. Créer le projet



2. Ajout d'une zone (Area)



Exemple création d'une Area « Blog »



Pour éviter les problèmes de routes, il vaut mieux conserver le nom de l'Area en début de route « AreaRegistration » de la zone.

```
using System.Web.Mvc;
namespace AspCms.Areas.Blog
{
    public class BlogAreaRegistration : AreaRegistration
    {
        public override string AreaName
        {
            get
            {
                return "Blog";
            }
        }

        public override void RegisterArea(AreaRegistrationContext context)
        {
            context.MapRoute(
                "Blog_default",
                "Blog/{controller}/{action}/{id}",
                new { controller = "Articles", action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```

Liens d'un projet avec Areas

Pour les projets avec des Areas il faudra indiquer en « **routeValues** » l'**Area** ciblée. **Soit** « **vide** » pour un contrôleur « sans Area », **soit le nom** de l'**Area**, sinon la vue ou l'action risque de ne pas être trouvée.

Exemple : lien depuis un contrôleur « sans Area » AccountController

- Libellé
- Action
- Contrôleur
- Valeurs de route, c'est ici qu'on indique le nom de l'Area ou vide
- Les attributs html (comme une classe CSS ou l'id)

```
@Html.ActionLink("S'inscrire", "Register", "Account", routeValues: new { area = "" }, htmlAttributes: new
{ id = "registerLink" })
```

Exemple lien vers une action d'un contrôleur d'une Area (Area « Blog »)

```
@Html.ActionLink("Blog", "Index", "Articles", new { area = "Blog" }, null)
```

C'est toujours possible de faire un lien direct avec la route

```
<a href="/Blog/Articles/Index">Blog</a>
```

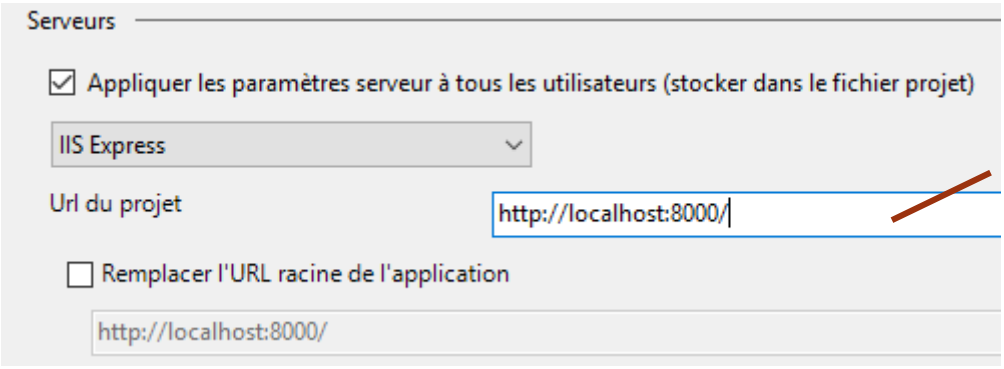
Exemple complet « _loginPartial »

Pour le formulaire permettant de se déconnecter on indique l'Area

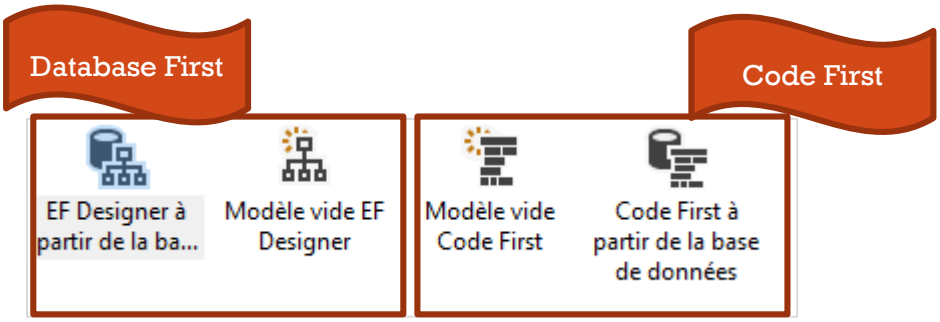
```
@using Microsoft.AspNet.Identity
@if (Request.IsAuthenticated)
{
    using (Html.BeginForm("LogOff", "Account" , new { area = "" }, FormMethod.Post, new { id =
"logoutForm", @class = "navbar-right" }))
    {
    @Html.AntiForgeryToken()

    <ul class="nav navbar-nav navbar-right">
        <li>
            @Html.ActionLink("Bonjour " + User.Identity.GetUserName() + "!", "Index", "Manage",
routeValues: new { area = "" }, htmlAttributes: new { title = "Manage" })
        </li>
        <li><a href="javascript:document.getElementById('logoutForm').submit()">Se déconnecter</a></li>
    </ul>
    }
}
else
{
    <ul class="nav navbar-nav navbar-right">
        <li>@Html.ActionLink("S'inscrire", "Register", "Account", routeValues: new { area = "" },
htmlAttributes: new { id = "registerLink" })</li>
        <li>@Html.ActionLink("Se connecter", "Login", "Account", routeValues: new { area = "" },
htmlAttributes: new { id = "loginLink" })</li>
    </ul>
}
```

3. Changer l'url du projet

Dans les propriétés du projet, onglet « Web »



4. Générations des modèles Entity Framework



- « *Database First* » : Génération des modèles à partir d'une base de données, avec **designer**.
  Il faudra permettre de **créer les tables liées à l'authentification** soit en ajoutant un utilisateur, soit avec la console du gestionnaire de package. « Enable-Migrations » puis « Add-Migration nom_fichier » et enfin « Update-Database »
- « *Code First* » : **Pas de designer**.
  - Il faut utiliser les **Data annotations** et l'**API Fluent** pour définir les contraintes de la table et des colonnes
  - ensuite créer un **fichier de migration** « Enable-Migrations » puis « Add-Migration nom_fichier »
  - enfin **mettre à jour la base** « Update-Database –verbose »

  C'est assez compliqué de définir toutes les contraintes en Code First, une solution peut être de générer les modèles en Code First à partir de la base.

a. Modèle avec clé étrangère sur l'utilisateur (ApplicationUser) en Code First

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

namespace AspCms.Areas.Blog.Models
{
    public class Article
    {
        public Article()
        {
            Created = DateTime.Now;
        }

        [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        [Required]
        public string Title { get; set; }
        [Required]
        [Column(TypeName = "text")]
        public string Content { get; set; }

        public DateTime Created { get; set; }

        [MaxLength(128), ForeignKey("User")]
        public virtual string UserId { get; set; }

        public virtual ApplicationUser User { get; set; }
    }
}
```

Foreign key

Ajout à « **ApplicationDbContext** » (fichier « IdentityModels » du dossier « Models »)

Important : si on crée un autre DbContext ça va poser des problèmes, demander à définir des clés et créer des tables inutiles en base de données.

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
        public ApplicationDbContext()
            : base("DefaultConnection", throwIfV1Schema: false)
        {
        }

        public static ApplicationDbContext Create()
        {
            return new ApplicationDbContext();
        }

        public DbSet<Article> Articles { get; set; }

}
```

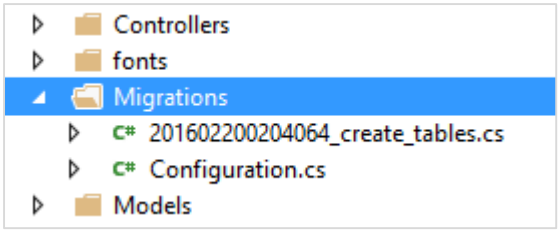
La chaine de connexion utilisée

Ensuite … depuis la **console du gestionnaire de package**

```
Enable-Migrations
```

Puis **création d'un fichier de migration** (nommé ici « create_tables »). Celui-ci est ajouté dans le **dossier** « **migrations** » du projet.

```
Add-Migration create_tables
```



Classiquement la méthode « up » sert à mettre à jour la base. Et la méthode « down » pour faire un rollback.

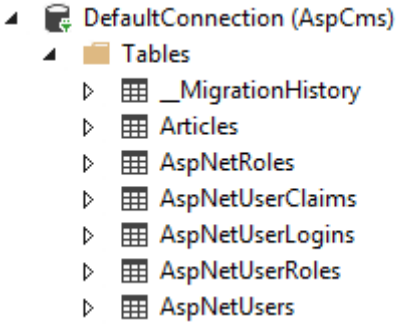
Exemple pour la table à créer, automatiquement la **clé étrangère** sera ajoutée **sur la bonne table** « **AspNetUsers** »

```
CreateTable(
          "dbo.Articles",
          c => new
              {
                  Id = c.Int(nullable: false, identity: true),
                  Title = c.String(nullable: false),
                  Content = c.String(nullable: false, unicode: false, storeType: "text"),
                  UserId = c.String(nullable: false, maxLength: 128),
              })
          .PrimaryKey(t => t.Id)
          .ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)
          .Index(t => t.UserId);
      // etc.
```

Mettre à jour la base de données

```
Update-Database
```

Les tables sont crées



Pour faire un **rollback**

```
Update-Database -TargetMigration:0
```

b. Colonne avec date par défaut à « now »

Exemple on ajoute une colonne « Created »

```
public class Article
{
        public Article()
        {
            Created = DateTime.Now;
        }
        // etc.
        public DateTime Created { get; set; }
}
```

Ajouter « manuellement » dans le fichier de migration (avant d'avoir effectué la mise à jour de la base) en dessous la création de la table « Articles »

```
Sql("ALTER TABLE dbo.Articles ADD CONSTRAINT default_created DEFAULT GETDATE() FOR
Created");
```

5. Sécurité

a. Ajout de champs à l'utilisateur

Model dans le fichier « IdentityModels »

```
public class ApplicationUser : IdentityUser
{
        public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser>
manager)
        {
            var userIdentity = await manager.CreateIdentityAsync(this,
DefaultAuthenticationTypes.ApplicationCookie);
            return userIdentity;
        }
        public string FirstName { get; set; }

        public string LastName { get; set; }
}
```

ViewModel dans le fichier « AccountViewModels » (utilisé par la vue)

```
public class RegisterViewModel
{
         // etc.

        [Display(Name = "Prénom")]
        public string FirstName { get; set; }

        [Display(Name = "Nom")]
        public string LastName { get; set; }
}
```

Contrôleur « **AccountController** »

```
        // POST: /Account/Register
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Register(RegisterViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = new ApplicationUser {
                    FirstName = model.FirstName,
                    LastName = model.LastName,
                    UserName = model.Email,
                    Email = model.Email
                };
                var result = await UserManager.CreateAsync(user, model.Password);
                if (result.Succeeded)
                {
                    string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

                    var callbackUrl = Url.Action(
                        "ConfirmEmail",
                        "Account",
                        new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

                    await UserManager.SendEmailAsync(
                        user.Id,
                        "Confirmez votre compte", "Confirmez votre compte en cliquant <a href=\"" +
callbackUrl + "\">ici</a>");

                    return View("CheckYourEmail");
                }
                AddErrors(result);
            }
            return View(model);
        }
```

Dans la Vue « Register.cshtml » on ajoute les nouveaux champs

```
@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-
horizontal", role = "form" }))
{
    @Html.AntiForgeryToken()
    <h4>Créer un nouveau compte.</h4>
    <hr />

    @Html.ValidationSummary("", new { @class = "text-danger" })

    <div class="form-group">
        @Html.LabelFor(m => m.FirstName, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.FirstName, new { @class = "form-control" })
        </div>
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.LastName, new { @class = "col-md-2 control-label" })
        <div class="col-md-10">
            @Html.TextBoxFor(m => m.LastName, new { @class = "form-control" })
        </div>
    </div>
  //autres champs
```

Créer un **fichier de migration** pour ajouter les colonnes FirstName et LastName à la table « AspNetUsers »

```
Add-Migration Add_FirstName_Last_Name_AspNetUsers
```

Fichier généré

```
namespace AspCms.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class Add_FirstName_Last_Name_AspNetUsers : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.AspNetUsers", "FirstName", c => c.String());
            AddColumn("dbo.AspNetUsers", "LastName", c => c.String());
        }

        public override void Down()
        {
            DropColumn("dbo.AspNetUsers", "LastName");
            DropColumn("dbo.AspNetUsers", "FirstName");
        }
    }
}
```

Puis **Mise à jour de la base**

```
Update-Database
```

b. Validation de compte utilisateur

On demande à l'utilisateur de valider son inscription en cliquant sur le lien d'un email.

```
        // POST: /Account/Register
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Register(RegisterViewModel model)
        {
            if (ModelState.IsValid)
            {
                var user = new ApplicationUser {
                    FirstName = model.FirstName,
                    LastName = model.LastName,
                    UserName = model.Email,
                    Email = model.Email
                };
                var result = await UserManager.CreateAsync(user, model.Password);
                if (result.Succeeded)
                {
                    string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

                    var callbackUrl = Url.Action(
                        "ConfirmEmail",
                        "Account",
                        new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

                    await UserManager.SendEmailAsync(
                        user.Id,
                        "Confirmez votre compte", "Confirmez votre compte en cliquant <a href=\"" +
callbackUrl + "\">ici</a>");

                    return View("CheckYourEmail");
                }
                AddErrors(result);
            }
            return View(model);
        }
```

URL avec code

Envoi de l'email

On redirige vers une vue informant l'utilisateur de vérifier ses emails

Dans « **IdentityConfig** », renseigner les **informations** afin de pouvoir envoyer des **emails**

```
    public class EmailService : IIdentityMessageService
    {
        public Task SendAsync(IdentityMessage message)
        {
            const string userName = "<your email>";
            const string from = "<your email>";
            const string password = "<your password>";
            const int port = 465;

            var smtpClient = new SmtpClient("<your client smtp>",port);
            smtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;
            smtpClient.UseDefaultCredentials = false;
            //smtpClient.EnableSsl = true;
            smtpClient.Credentials = new NetworkCredential(userName, password);

            var mailMessage = new MailMessage(from, message.Destination, message.Subject,
message.Body);
            mailMessage.IsBodyHtml = true;
            return smtpClient.SendMailAsync(mailMessage);
        }
    }
```

Le port n'est pas toujours indiqué

Pour les démonstrations on peut faire plus court

```
        public class EmailService : IIdentityMessageService
        {
            public Task SendAsync(IdentityMessage message)
            {
                const string from = "<your email>";
                var smtpClient = new SmtpClient("<Your smtp client>");
                var mailMessage = new MailMessage(from, message.Destination,
message.Subject, message.Body);
                mailMessage.IsBodyHtml = true;
                return smtpClient.SendMailAsync(mailMessage);
            }
        }
```

On crée une **vue** nommée « **CheckYourEmail** » dans dans dossier « Views/Account ». Exemple

```
@{
    ViewBag.Title = "Compte créé";
}
<h2>@ViewBag.Title</h2>
<p>Un email de confirmation vient de vous être envoyé.</p>
```

c. Tester si l'utilisateur a validé son email

Lorsque l'utilisateur essaie de se connecter, on vérifie s'il a validé son email. Si ce n'est pas le cas on le redirige vers une vue l'informant qu'il doit valider son email et permettant de renvoyer un email de confirmation.

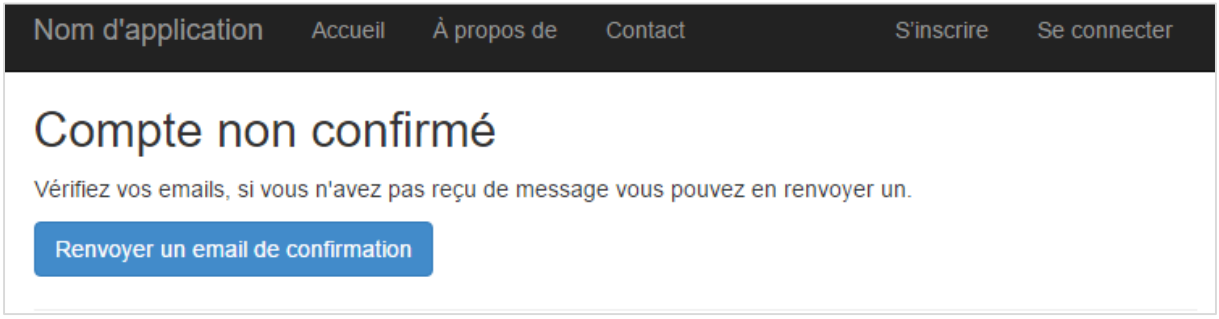
```
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }
            var user = await UserManager.FindByEmailAsync(model.Email);
            if (user != null)
            {
                if(!await UserManager.IsEmailConfirmedAsync(user.Id))
                {
                    ViewBag.Email = user.Email;
                    return View("UnconfirmedAccount");
                }
            }
            var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password,
model.RememberMe, shouldLockout: false);
            switch (result)
            {
                case SignInStatus.Success:
                    return RedirectToLocal(returnUrl);
                case SignInStatus.LockedOut:
                    return View("Lockout");
                case SignInStatus.RequiresVerification:
                    return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe =
model.RememberMe });
                case SignInStatus.Failure:
                default:
                    ModelState.AddModelError("", "Tentative de connexion non valide.");
                    return View(model);
            }
        }

        [AllowAnonymous]
        public async Task<ActionResult> RegenerateEmailConfirmation(string email)
        {
            var user = await UserManager.FindByEmailAsync(email);
            if (user != null)
            {
                string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

                var callbackUrl = Url.Action(
                    "ConfirmEmail",
                    "Account",
                    new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

                await UserManager.SendEmailAsync(
                    user.Id,
                    "Confirmez votre compte", "Confirmez votre inscription en cliquant <a href=\"" +
callbackUrl + "\">ici</a>");
            }
            return View("UnconfirmedAccount");
        }
```

Vérification que l'utilisteur a validé son email

Action appelée pour regénérer un email de confirmation

Ajout d'une **Vue** « **UnconfirmedAccount** » dans le dossier « Views/Account »

```
@{
    ViewBag.Title = "Compte non confirmé";
}

<h2>@ViewBag.Title</h2>
<p>Vérifiez vos emails, si vous n'avez pas reçu de message vous pouvez en renvoyer un.</p>
@using (Html.BeginForm("RegenerateEmailConfirmation", "Account", FormMethod.Post))
{
    @Html.AntiForgeryToken()

    @Html.Hidden("email", (string)ViewBag.Email)
    <input type="submit" value="Re-envoyer un email de confirmation" class="btn btn-primary" />
}
```



d. Connexion avec les réseaux sociaux

Google

Créer un projet

- Activer l'API Google +
- Créer des identifiants : choisir « ID Client OAuth » puis indiquer
  - En origine : « http://localhost:8000 » par exemple (changer le port si besoin)
  - En redirection « http://localhost:8000/signin-google »

Facebook

Créer une application, la rendre publique (App Review), etc.

Ensuite récupérer les identifiants et les rentre dans le fichier « Startup.OAuth » du dossier « App_Start »

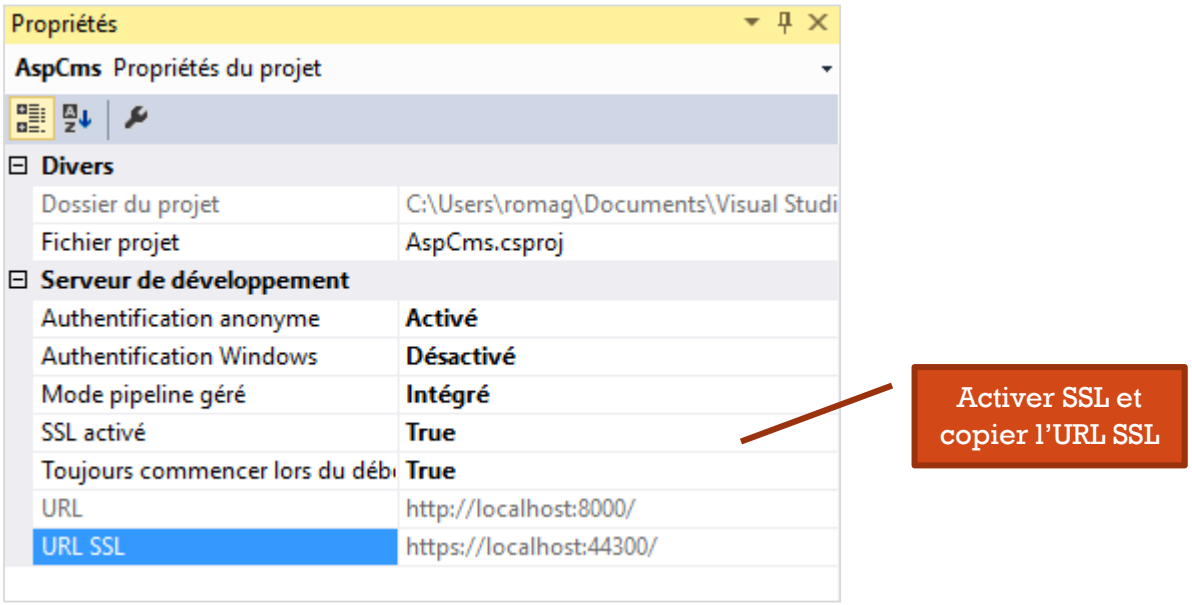
e. Protéger une route

Avec l'attribut « Authorize » au-dessus du contrôleur ou seulement pour certaines actions.

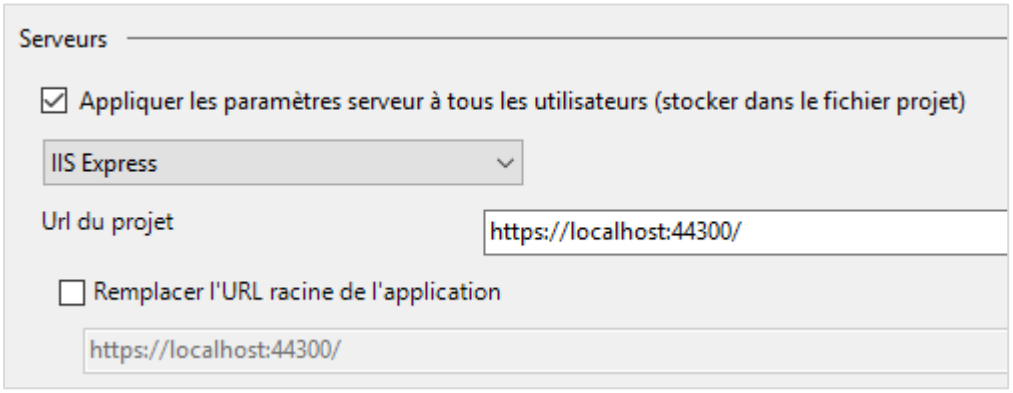
L'utilisateur est redirigé vers la page de connexion.

f. Activer SSL

Dans le panneau propriété du projet, **activer SSL**



… Coller l'**URL SSL** dans les propriétés du projet, onglet « Web »

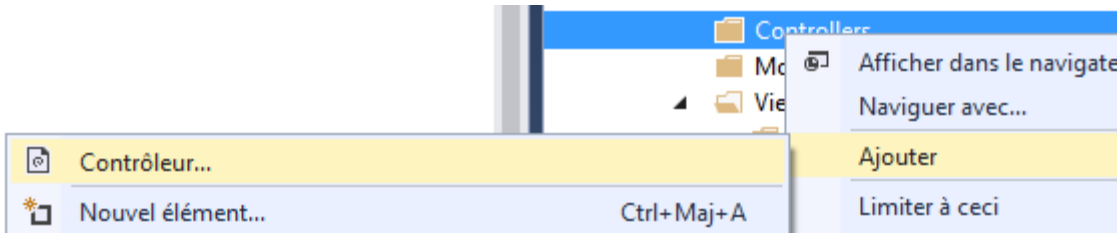


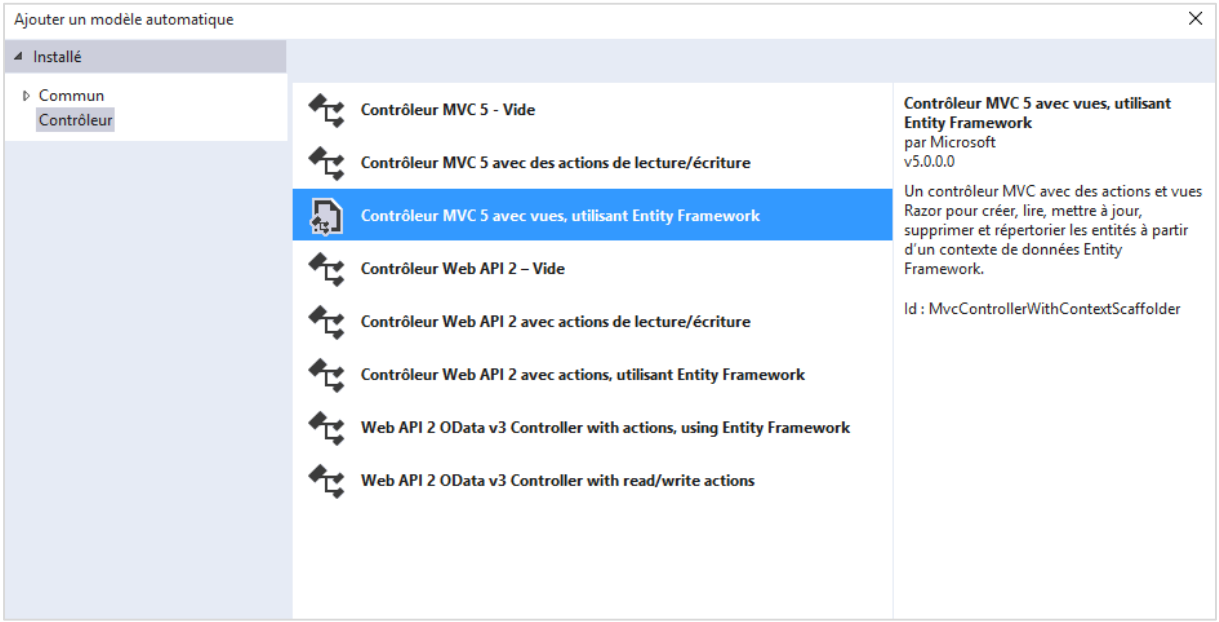
Ajouter les **attributs** « Authorize » et « RequireHttps » dans **FilterConfig**

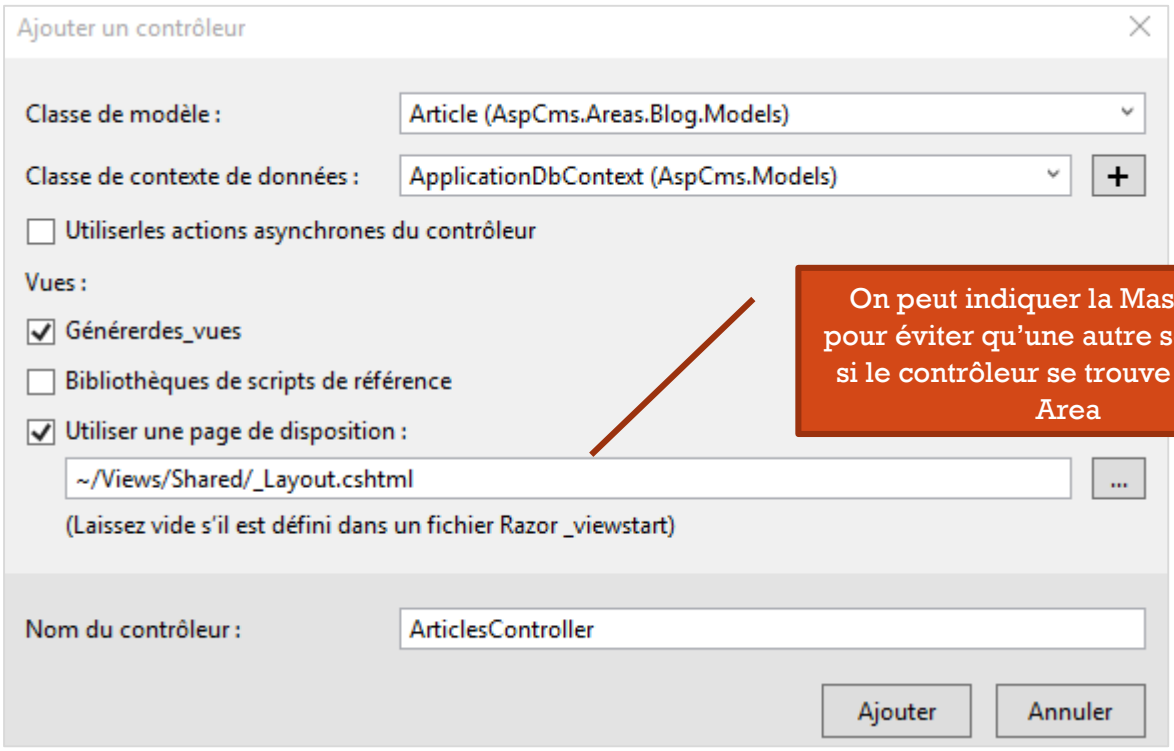
```
public class FilterConfig
{
    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
    {
        filters.Add(new HandleErrorAttribute());
        filters.Add(new AuthorizeAttribute());
        filters.Add(new RequireHttpsAttribute());
    }
}
```

6. Ajout de contrôleur

Ajout des contrôleurs pour les modèles créés. On peut utiliser un **contrôleur MVC** utilisant « **Entity Framework** » pour gagner du temps.







Le contrôleur hérite de « Controller » et utilise le DbContext

```
public class ArticlesController : Controller
{
        private ApplicationDbContext db = new ApplicationDbContext();
```

Entity Framework

Liste

Include permet de récupérer l'objet correspondant de la clé étrangère

```
// GET: Blog/Articles
public ActionResult Index()
{
      var articles = db.Articles.Include(a => a.User);
      return View(articles.ToList());
}
```

Un élément

Utilisation de la méthode find

```
        // GET: Blog/Articles/Details/5
        public ActionResult Details(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Article article = db.Articles.Find(id);
            if (article == null)
            {
                return HttpNotFound();
            }
            return View(article);
        }
```

Ajout

Article bindé aux élements du formulaire

```
 // GET: Blog/Articles/Create
[Authorize]
 public ActionResult Create()
 {
     ViewBag.UserId = User.Identity.GetUserId();
     return View();
 }

 // POST: Blog/Articles/Create
 [Authorize]
 [HttpPost]
 [ValidateAntiForgeryToken]
 public ActionResult Create([Bind(Include = "Id,Title,Content,UserId")] Article article)
 {
     if (ModelState.IsValid)
     {
         db.Articles.Add(article);
         db.SaveChanges();
          return RedirectToAction("Details", new { id = article.Id });
     }
     ViewBag.UserId = User.Identity.GetUserId();
     return View(article);
 }
```

Modification

```
// GET: Blog/Articles/Edit/5
[Authorize]
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Article article = db.Articles.Find(id);
    if (article == null)
    {
        return HttpNotFound();
    }
    return View(article);
}

// POST: Blog/Articles/Edit/5
[Authorize]
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include = "Id,Title,Content")] Article article)
{
    if (ModelState.IsValid)
    {
        db.Entry(article).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(article);
}
```

On marque l'élément comme modifié

Suppression

```
// GET: Blog/Articles/Delete/5
[Authorize]
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Article article = db.Articles.Find(id);
    if (article == null)
    {
        return HttpNotFound();
    }
    db.Articles.Remove(article);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

Le ViewBag

Le ViewBag permet de passer des données à la vue.

7. Vues

a. Master Page

C'est « _layout.cshtml » du dossier « Views/Shared »

Si on veut utiliser la Master Page de base **dans les Areas** il faut **l'indiquer**, sinon cela ira chercher la Master Page définie dans le dossier Shared de cette Area

```
@model AspCms.Areas.Blog.Models.Article

@{
    ViewBag.Title = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
```

b. Formulaires

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Article</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })

        @Html.Hidden("UserId")

        <div class="form-group">
            @Html.LabelFor(model => model.Title, htmlAttributes: new { @class = "control-label col-md-
2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Title, new { htmlAttributes = new { @class = "form-
control" } })
                @Html.ValidationMessageFor(model => model.Title, "", new { @class = "text-danger" })
            </div>
        </div>

@*etc.*@

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}
```

Ouverture du formulaire

CSRF Token

Liste complète des erreurs de validation

Camp hidden

Label

Erreur pour le champ

Envoi

Autres signatures de form

- Action
- Contrôleur
- Valeurs de Routes
- Méthode d'envoi du formulaire
- Attributs HTML

```
using (Html.BeginForm("LogOff", "Account" , new { area = "" }, FormMethod.Post, new { id =
"logoutForm", @class = "navbar-right" }))
{
```