ASP.NET Identity avec Microsoft Visual Studio 2012 et MVC5

Introduction

Si vous utilisez toujours 'Visual Studio 2012' et si vous voulez intégrer une identification dans votre site Internet vous pouvez utiliser 'ASP.Net identity' avec 'EntityFramework'. Contrairement aux versions plus récentes de 'Visual Studio', il n'existe pas de modèle de projet par défaut pour ce type de projet.

Nous allons donc montrer comment intégrer manuellement ce 'Framework'.

Je pars du principe que vous avez des bases sur l'utilisation de 'VisualStudio', de 'C#', 'ASP.NET', 'MVC' et de la programmation orienté objet.

Vous pouvez télécharger ce document sous forme de fichier '.pdf' à l'adresse suivante :
http://www.technofactory.fr/Blogdocs/ASPNetIdentityVS2012/ASP identity VS2012.pdf

> Note : la correction automatique d'orthographe modifie par fois le code source que j'ai mis dans ce document. Il est préférable d'utiliser le code source du projet exemple.

Utilisation du projet exemple

Le projet d'exemple ce trouve ici :
http://www.technofactory.fr/Blogdocs/ASPNetIdentityVS2012/aspNetIdentityExemple.zip

L'utilisation de l'exemple nécessite que 'VS2012' et 'Sql Serveur' soient installés.

Décompressez l'archive sur votre disque, chargez la solution avec 'Visual Studio' et modifiez la chaine de connexion (voir ci-dessous) dans le fichier 'Web.Config'.

Installation des prérequis

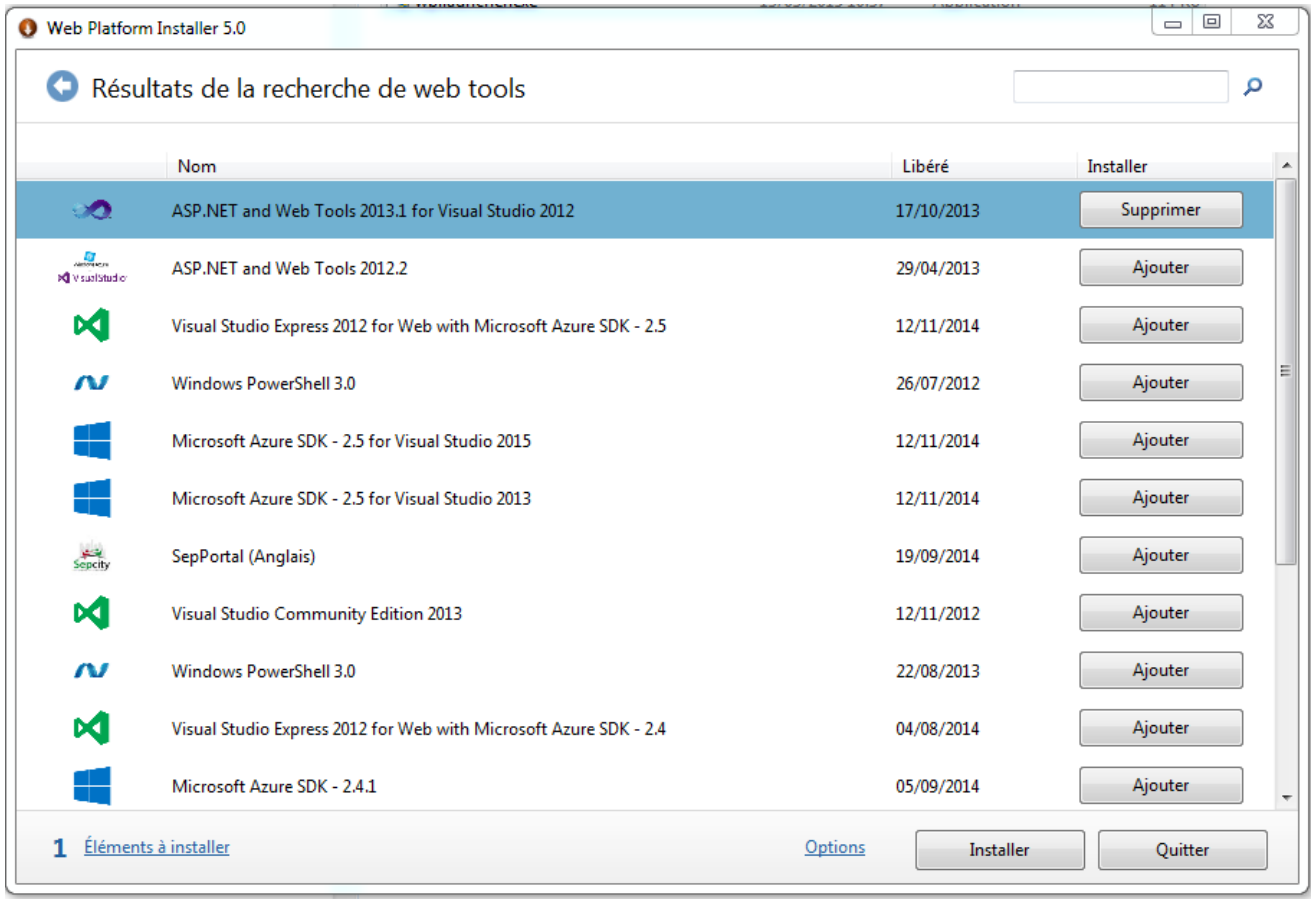
Mise à jour Visual Studio

D'abord on va commencer par créer un projet 'MVC 5' sous 'Visual Studio 2012'. Cela nécessite d'installer 'Web Tools pour VS2012'. Il s'agit d'une mise à jour de 'Visua lStudio 2012'.

Il peut être installé avec le 'Web Plateform Installer' (Web PI) que vous pouvez obtenir ici :
http://www.microsoft.com/web/downloads/platform.aspx.

> Note : d'une façon générale, il est utile d'installer le 'Web Plateform Installer – Web PI'. La page (en anglais) n'est pas très intuitive et pour lancer le téléchargement il faut cliquer sur 'Free Download'. Une fois le programme téléchargé, lancez l'installation qui se termine par une liste d'applications que vous pouvez installer.

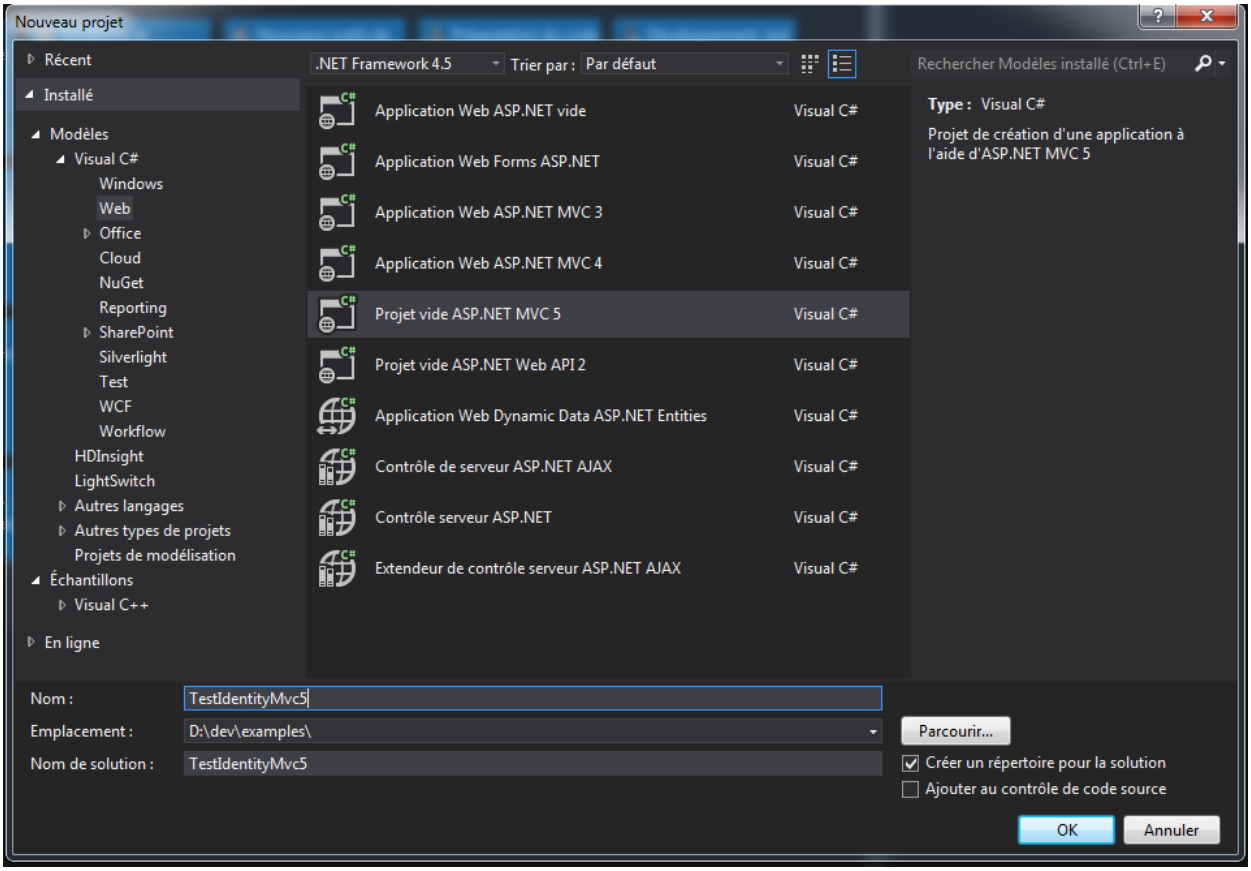
Cherchez ensuite 'Web Tools' et ajoutez 'ASP.NET and Web Tools 2013.1 for Visual Studio 2012' et '*Installer*'.


Web Platform Installer 5.0
Résultats de la recherche de web tools
Nom
Libéré
Installer
ASP.NET and Web Tools 2013.1 for Visual Studio 2012
17/10/2013
Supprimer
ASP.NET and Web Tools 2012.2
29/04/2013
Ajouter
Visual Studio Express 2012 for Web with Microsoft Azure SDK - 2.5
12/11/2014
Ajouter
Windows PowerShell 3.0
26/07/2012
Ajouter
Microsoft Azure SDK - 2.5 for Visual Studio 2015
12/11/2014
Ajouter
Microsoft Azure SDK - 2.5 for Visual Studio 2013
12/11/2014
Ajouter
SepPortal (Anglais)
19/09/2014
Ajouter
Visual Studio Community Edition 2013
12/11/2012
Ajouter
Windows PowerShell 3.0
22/08/2013
Ajouter
Visual Studio Express 2012 for Web with Microsoft Azure SDK - 2.4
04/08/2014
Ajouter
Microsoft Azure SDK - 2.4.1
05/09/2014
Ajouter
1 Éléments à installer
Options
Installer
Quitter

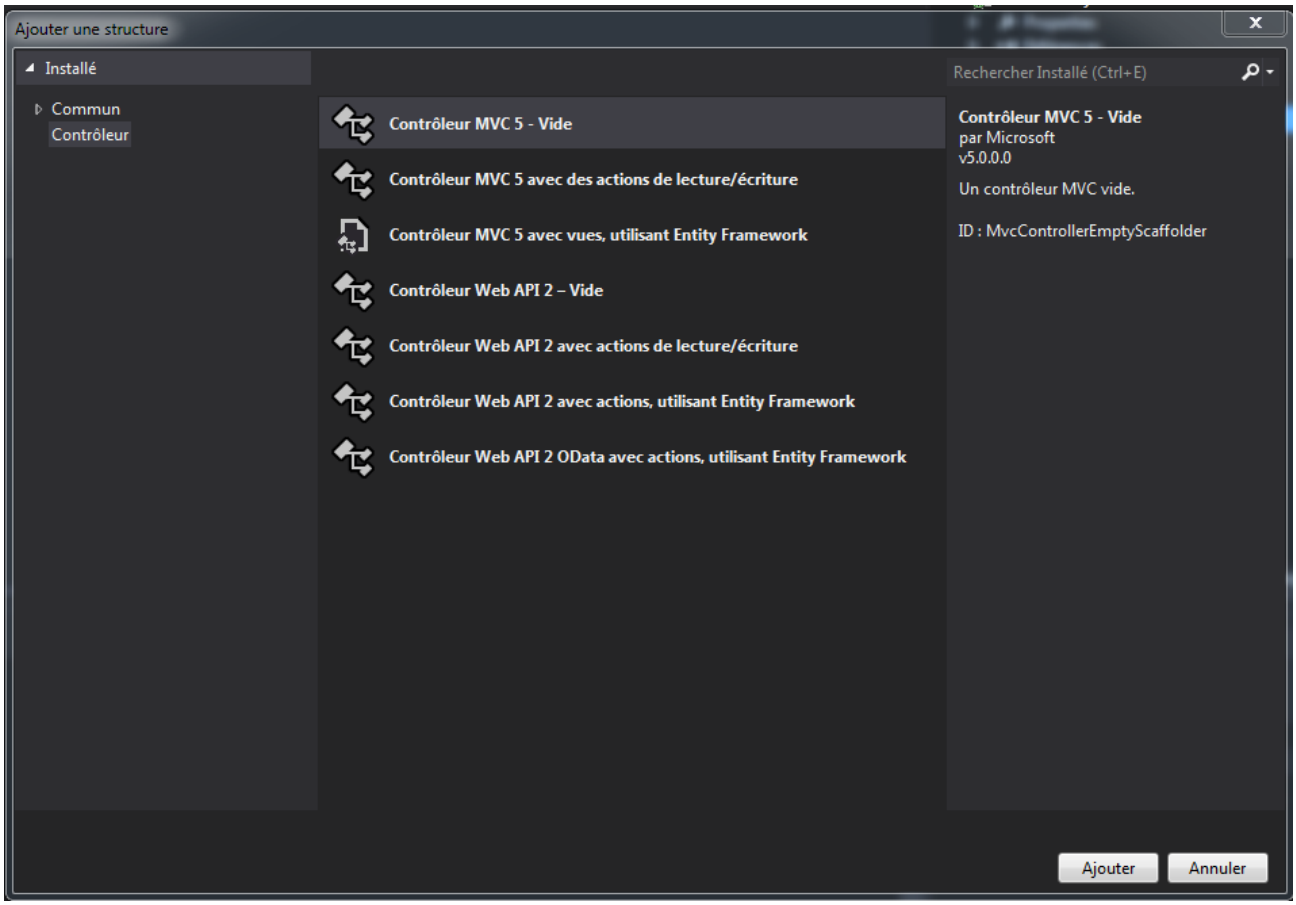
Création d'une application MVC 5 pour le test

Lancez 'Visual Studio' et créez un nouveau projet : *Fichier->Nouveau->Projet* et sélectionnez : *Projet vide ASP.NET Mvc5*.

Ici le nom du projet et de la solution sont '***TestIdentityMvc5***'.



Le projet ne contient rien, donc il faut ajouter un contrôleur avec le nom '***DefaultController'*** (clic droit sur le dossier *'Controllers -> Ajouter -> Contrôleur'*). Sélectionnez '*Contrôleur MVC 5 – Vide'*, puis *'Ajouter'*.



Le système vous demandera le nom du contrôleur : spécifiez ***'DefaultController'***, puis *'Ajouter'*.

Ajoutez une source de données statique pour avoir quelque chose à afficher sur la page: clic droit sur le dossier *Models -> Ajouter -> Classe* et créez la classe '***DataItem'***.

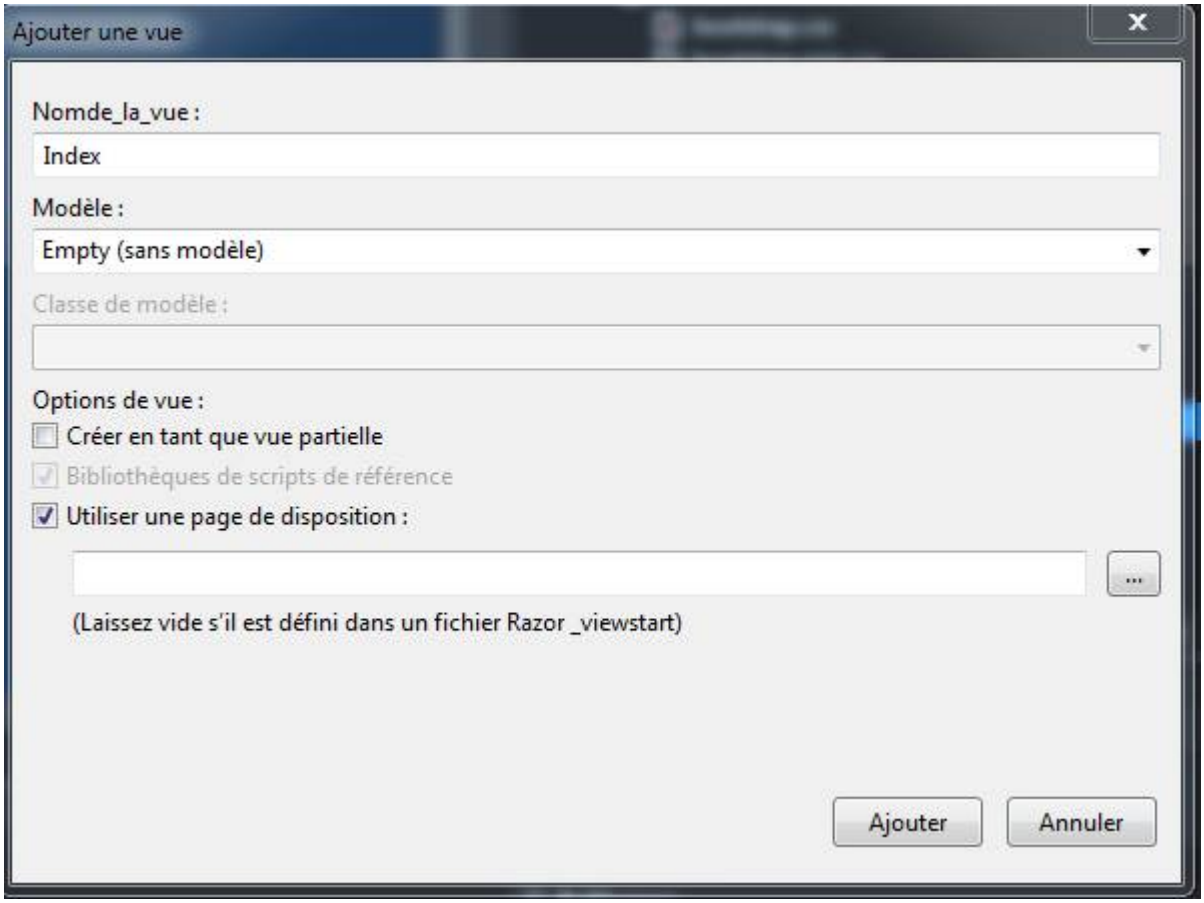
Pour l'exemple la classe du projet d'exemple ressemble à celle-ci :

```
namespace TestIdentityMvc5.Models
{
    /// <summary>
    /// Classe contient des données d'exemple
    /// </summary>
    public class DataItem
    {

        public static DataItem[] DataItems = { new DataItem("ABCD","42")
                                               ,new DataItem("EFGH","45")
                                               ,new DataItem("IJKLM","25")};

        /// <summary>
        /// Obtient le nom de la personne
        /// </summary>
        public string Name { get; private set; }

        /// <summary>
        /// Obtient l'âge de la personne
        /// </summary>
        public string Age { get; private set; }

        /// <summary>
        /// Constructeur initialise les valeurs de la classe
        /// </summary>
        public DataItem(string name, string age)
        {
            Name = name;
            Age = age;
        }
    }
}
```

Maintenant on va ajouter une vue pour afficher nos données d'exemple.

Clic droit sur le répertoire *'Views/Default'* -> *Ajouter Vue*. Nom de la vue '***Index'***, choisissez le modèle '*Empty (sans modèle)*'.

Note : le répertoire 'Default' a été créé quand vous avez créé le contrôleur.



Les fichiers suivants ont été créés : '***Index.cshtml'*** dans le dossier ***'Default', '_Layout.cshtml'*** dans le dossier '***Shared'*** and '***ViewStart.cshtml'*** directement sous '*Views*'.
Modifiez '***Index.cshtml'*** pour afficher nos données comme par exemple :

```
@model  IEnumerable<TestIdentityMvc5.Models.DataItem>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<table style="width: 100%">
    <caption>Liste de personnes</caption>
    <tr>
        <th>Name</th>
        <th>Age</th>
    </tr>
    @foreach (var dataitm in Model)
    {

        <tr>
            <td>@dataitm.Name</td>
            <td>@dataitm.Age</td>
        </tr>
    }
</table>
```

Modifiez '***Controllers/DefaultController.cs***' pour passer le modèle à la vue :

Ajoutez : `using TestIdentityMvc5.Models;` au début du fichier.

```
@using TestIdentityMvc5.Models;

…
public ActionResult Index()
{
   return View(DataItem.DataItems);
}

…
```

Et enfin, ajustez '*App_Start/ RouteConfig.cs* ' pour ajuster le contrôleur par défaut.

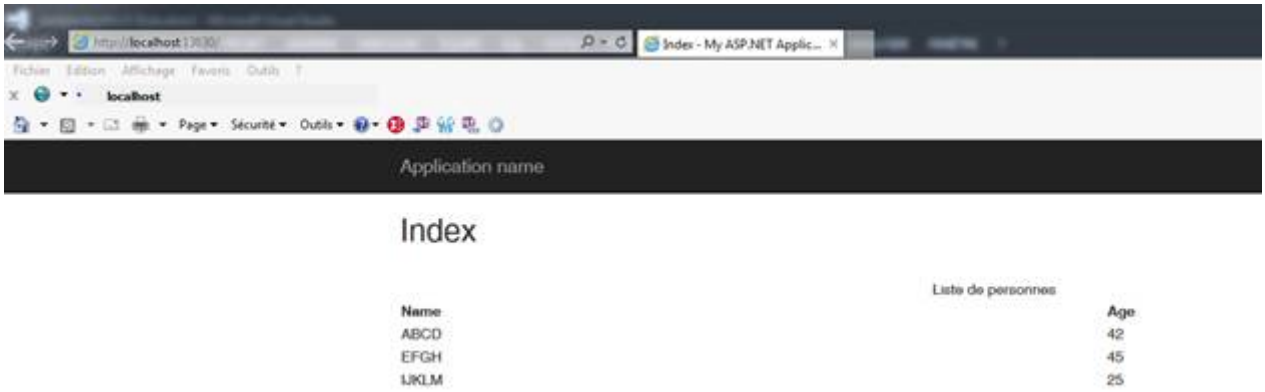
```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Default", action = "Index", id = UrlParameter.Optional }
   );
}
```

Mettez à jour l'ensemble des packages : *Projet->Gérer les packages NuGet -> Mises à jour -> Tout mettre à jour.*

Suite à la mise à jour il faut vérifier le fichier *'_Layout.cshtml'* dans le répertoire '*Shared'*.

Vérifiez que les liens vers les fichiers '*.js' (en particulier le fichier 'jquery.js' qui se trouve à la fin du fichier) pointent vers des fichiers qui se trouvent dans les répertoires ('Scripts', 'Content', …) de la solution. Sinon il peut y avoir des exceptions à l'exécution.
Malheureusement les versions changent tellement fréquemment qu'on ne peut plus indiquer les versions.

Vous pouvez maintenant essayer l'application. Vous devrez avoir un résultat similaire au suivant :



Installation des NuGetPackages

Maintenant, nous allons ajouter les 'NuGet Packages' nécessaires pour intégrer une authentification. L'authentification se base sur 'OWin'.

Il y a deux possibilités : soit par la boîte de dialogue (*Projet->Gérer les packages NuGet*) et vous cherchez les différents packages soit par ligne de commande : *Outils -> Gestionnaire de de Package NuGet -> Console du Gestionnaire de package*.

Pour installer le package la commande générale est : *install-package* <*nom du package*>. Pour spécifier un projet en particulier ajoutez : *-Project* <*le nom de votre projet*>

- 'Microsoft Owin' (recherchez 'Owin') - Version 3.0.0 ce jour-ci.

'NuGet' commande: *install-package Microsoft.Owin*

```
PM> install-package Microsoft.Owin
Tentative de résolution de la dépendance « Owin (≥ 1.0) ».
Installation de « Owin 1.0 ».
Installation de « Owin 1.0 » terminée.
Installation de « Microsoft.Owin 3.0.1 ».
Vous téléchargez Microsoft.Owin à partir de Microsoft. Le contrat de licence est disponible à l'emplacement http://www.microsoft.com/
web/webpi/eula/net_library_eula_enu.htm. Vérifiez si le package contient des dépendances susceptibles de faire l'objet de contrats de
 licence supplémentaires. Votre utilisation du package et des dépendances confirment votre acceptation de leurs contrats de licence.
Si vous n'acceptez pas les contrats de licence, supprimez les composants correspondants de votre ordinateur.
Installation de « Microsoft.Owin 3.0.1 » terminée.
Ajout de « Owin 1.0 » à TestIdentityMvc5.
Ajout réussi de « Owin 1.0 » à TestIdentityMvc5.
Ajout de « Microsoft.Owin 3.0.1 » à TestIdentityMvc5.
Ajout réussi de « Microsoft.Owin 3.0.1 » à TestIdentityMvc5.
```

- 'Microsoft ASP.NEt Identity Core' (recherchez : 'Identity.Core') .Version 2.2.0 ce jour-ci.

Commande: *install-package Microsoft.Aspnet.Identity.Core*

```
PM> install-package Microsoft.Aspnet.Identity.Core
Installation de « Microsoft.AspNet.Identity.Core 2.2.0 ».
Vous téléchargez Microsoft.AspNet.Identity.Core à partir de Microsoft. Le contrat de licence est disponible à l'emplacement http://
www.microsoft.com/web/webpi/eula/net_library_eula_ENU.htm. Vérifiez si le package contient des dépendances susceptibles de faire
l'objet de contrats de licence supplémentaires. Votre utilisation du package et des dépendances confirment votre acceptation de leurs
 contrats de licence. Si vous n'acceptez pas les contrats de licence, supprimez les composants correspondants de votre ordinateur.
Installation de « Microsoft.AspNet.Identity.Core 2.2.0 » terminée.
Ajout de « Microsoft.AspNet.Identity.Core 2.2.0 » à TestIdentityMvc5.
Ajout réussi de « Microsoft.AspNet.Identity.Core 2.2.0 » à TestIdentityMvc5.
```

- 'Microsoft Asp.net Identity Owin' package (recherchez 'Identity.owin')

Commande: install-package Microsoft.AspNet.Identity.Owin

```
PM> install-package Microsoft.AspNet.Identity.Owin
Tentative de résolution de la dépendance « Microsoft.AspNet.Identity.Core (≥ 2.2.0) ».
Tentative de résolution de la dépendance « Microsoft.Owin.Security (≥ 2.1.0) ».
Tentative de résolution de la dépendance « Owin (≥ 1.0) ».
Tentative de résolution de la dépendance « Microsoft.Owin (≥ 2.1.0) ».
Tentative de résolution de la dépendance « Microsoft.Owin.Security.Cookies (≥ 2.1.0) ».
Tentative de résolution de la dépendance « Microsoft.Owin.Security.OAuth (≥ 2.1.0) ».
Tentative de résolution de la dépendance « Newtonsoft.Json (≥ 4.5.11) ».
Installation de « Microsoft.Owin.Security 2.1.0 ».
Vous téléchargez Microsoft.Owin.Security à partir de Microsoft. Le contrat de licence est disponible à l'emplacement http://
www.microsoft.com/web/webpi/eula/aspnetcomponent_rtw_enu.htm. Vérifiez si le package contient des dépendances susceptibles de faire
l'objet de contrats de licence supplémentaires. Votre utilisation du package et des dépendances confirment votre acceptation de leurs
 contrats de licence. Si vous n'acceptez pas les contrats de licence, supprimez les composants correspondants de votre ordinateur.
Installation de « Microsoft.Owin.Security 2.1.0 » terminée.
Installation de « Microsoft.Owin.Security.Cookies 2.1.0 ».
Vous téléchargez Microsoft.Owin.Security.Cookies à partir de Microsoft. Le contrat de licence est disponible à l'emplacement http://
www.microsoft.com/web/webpi/eula/aspnetcomponent_rtw_enu.htm. Vérifiez si le package contient des dépendances susceptibles de faire
l'objet de contrats de licence supplémentaires. Votre utilisation du package et des dépendances confirment votre acceptation de leurs
 contrats de licence. Si vous n'acceptez pas les contrats de licence, supprimez les composants correspondants de votre ordinateur.
Installation de « Microsoft.Owin.Security.Cookies 2.1.0 » terminée.
Installation de « Newtonsoft.Json 4.5.11 ».
Installation de « Newtonsoft.Json 4.5.11 » terminée.
Installation de « Microsoft.Owin.Security.OAuth 2.1.0 ».
Vous téléchargez Microsoft.Owin.Security.OAuth à partir de Microsoft. Le contrat de licence est disponible à l'emplacement http://
www.microsoft.com/web/webpi/eula/aspnetcomponent_rtw_enu.htm. Vérifiez si le package contient des dépendances susceptibles de faire
l'objet de contrats de licence supplémentaires. Votre utilisation du package et des dépendances confirment votre acceptation de leurs
 contrats de licence. Si vous n'acceptez pas les contrats de licence, supprimez les composants correspondants de votre ordinateur.
Installation de « Microsoft.Owin.Security.OAuth 2.1.0 » terminée.
Installation de « Microsoft.AspNet.Identity.Owin 2.2.0 ».
Vous téléchargez Microsoft.AspNet.Identity.Owin à partir de Microsoft. Le contrat de licence est disponible à l'emplacement http://
www.microsoft.com/web/webpi/eula/net_library_eula_ENU.htm. Vérifiez si le package contient des dépendances susceptibles de faire
l'objet de contrats de licence supplémentaires. Votre utilisation du package et des dépendances confirment votre acceptation de leurs
 contrats de licence. Si vous n'acceptez pas les contrats de licence, supprimez les composants correspondants de votre ordinateur.
Installation de « Microsoft.AspNet.Identity.Owin 2.2.0 » terminée.
Ajout de « Microsoft.Owin.Security 2.1.0 » à TestIdentityMvc5.
Ajout réussi de « Microsoft.Owin.Security 2.1.0 » à TestIdentityMvc5.
Ajout de « Microsoft.Owin.Security.Cookies 2.1.0 » à TestIdentityMvc5.
Ajout réussi de « Microsoft.Owin.Security.Cookies 2.1.0 » à TestIdentityMvc5.
Ajout de « Newtonsoft.Json 4.5.11 » à TestIdentityMvc5.
Ajout réussi de « Newtonsoft.Json 4.5.11 » à TestIdentityMvc5.
Ajout de « Microsoft.Owin.Security.OAuth 2.1.0 » à TestIdentityMvc5.
Ajout réussi de « Microsoft.Owin.Security.OAuth 2.1.0 » à TestIdentityMvc5.
Ajout de « Microsoft.AspNet.Identity.Owin 2.2.0 » à TestIdentityMvc5.
Ajout réussi de « Microsoft.AspNet.Identity.Owin 2.2.0 » à TestIdentityMvc5.
```

- 'Identity EntityFramework'

Commande*: install-package Microsoft.AspNet.Identity.EntityFramework*

```
PM>  install-package Microsoft.AspNet.Identity.EntityFramework
Tentative de résolution de la dépendance « Microsoft.AspNet.Identity.Core (≥ 2.2.0) ».
Tentative de résolution de la dépendance « EntityFramework (≥ 6.1.0) ».
Installation de « EntityFramework 6.1.0 ».
Vous téléchargez EntityFramework à partir de Microsoft. Le contrat de licence est disponible à l'emplacement http://go.microsoft.com/
fwlink/?LinkID=320539. Vérifiez si le package contient des dépendances susceptibles de faire l'objet de contrats de licence
supplémentaires. Votre utilisation du package et des dépendances confirment votre acceptation de leurs contrats de licence. Si vous
n'acceptez pas les contrats de licence, supprimez les composants correspondants de votre ordinateur.
Installation de « EntityFramework 6.1.0 » terminée.
Installation de « Microsoft.AspNet.Identity.EntityFramework 2.2.0 ».
Vous téléchargez Microsoft.AspNet.Identity.EntityFramework à partir de Microsoft. Le contrat de licence est disponible à
l'emplacement http://www.microsoft.com/web/webpi/eula/net_library_eula_ENU.htm. Vérifiez si le package contient des dépendances
susceptibles de faire l'objet de contrats de licence supplémentaires. Votre utilisation du package et des dépendances confirment
votre acceptation de leurs contrats de licence. Si vous n'acceptez pas les contrats de licence, supprimez les composants
correspondants de votre ordinateur.
Installation de « Microsoft.AspNet.Identity.EntityFramework 2.2.0 » terminée.
Ajout de « EntityFramework 6.1.0 » à TestIdentityMvc5.
Ajout réussi de « EntityFramework 6.1.0 » à TestIdentityMvc5.

Type 'get-help EntityFramework' to see all available Entity Framework commands.
Ajout de « Microsoft.AspNet.Identity.EntityFramework 2.2.0 » à TestIdentityMvc5.
Ajout réussi de « Microsoft.AspNet.Identity.EntityFramework 2.2.0 » à TestIdentityMvc5.
```

- 'IIS ASP.NET Pipeline'

Commande: install*-package Microsoft.Owin.Host.SystemWeb*

```
PM>  install-package Microsoft.Owin.Host.SystemWeb
Tentative de résolution de la dépendance « Owin (≥ 1.0) ».
Tentative de résolution de la dépendance « Microsoft.Owin (≥ 3.0.1) ».
Installation de « Microsoft.Owin.Host.SystemWeb 3.0.1 ».
Vous téléchargez Microsoft.Owin.Host.SystemWeb à partir de Microsoft. Le contrat de licence est disponible à l'emplacement http://
www.microsoft.com/web/webpi/eula/net_library_eula_enu.htm. Vérifiez si le package contient des dépendances susceptibles de faire
l'objet de contrats de licence supplémentaires. Votre utilisation du package et des dépendances confirment votre acceptation de leurs
 contrats de licence. Si vous n'acceptez pas les contrats de licence, supprimez les composants correspondants de votre ordinateur.
Installation de « Microsoft.Owin.Host.SystemWeb 3.0.1 » terminée.
Ajout de « Microsoft.Owin.Host.SystemWeb 3.0.1 » à TestIdentityMvc5.
Ajout réussi de « Microsoft.Owin.Host.SystemWeb 3.0.1 » à TestIdentityMvc5.
```

Remettez à jour les 'NuGet packages' comme précédemment (un redémarrage de 'Visual Studio' peut être nécessaire).

Création du code de démarrage

Ajoutez directement sous la racine du projet le fichier "***OwinStartup.cs***" avec le contenu suivant:

```
using Microsoft.Owin;
using Owin;

[assembly: OwinStartup(typeof(TestIdentityMvc5.OwinStartup))]
namespace TestIdentityMvc5
{
    public partial class OwinStartup
    {
        /// <summary>
        /// Méthode appelée par le Framework Owin
        /// </summary>
        public void Configuration(IAppBuilder app)
        {
            ConfigureAuth(app);
        }


    }
}
```

Au démarrage, 'Owin' cherche la méthode 'Configuration'. L'attribut *'OwinStartup'* spécifie la class dans laquelle se trouve cette méthode. Il faut le nom complet dans le type. Plus d'informations ici : http://www.asp.net/aspnet/overview/owin-and-katana/owin-startup-class-detection

Ensuite il faut définir la méthode '***ConfigureAuth'***.

Pour cela, créez un fichier '***OwinStartup.Auth.cs'*** dans le répertoire '*App_Start'*. Le but est de respecter la structure des applications MVC. D'ailleurs le modèle de données avec VS2013 génère le même type de code.

Attention : il faut ajuster le *'namespace'* dans la classe du nouveau fichier. Il doit être le même que dans le fichier ***'OwinStartup.cs'***, parce que la classe est partielle. Autrement, notre méthode '***ConfigureAuth'*** ne sera pas résolue par le compilateur.

Voilà le contenu de notre fichier '***OwinStartup.Auth.cs'***.

```
using System;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Owin;
using TestIdentityMvc5.Models.Account;
using TestIdentityMvc5.Models.Db;


namespace TestIdentityMvc5
{
  public partial class OwinStartup
  {
      private void ConfigureAuth(IAppBuilder app)
      {
        // Configuration du contexte de la base de données,
        // de la gestion de l'utilisateur  et de la gestion
        // du login(connexion)
        app.CreatePerOwinContext(AppDbContext.Create)
           .CreatePerOwinContext<AppUserManager>(AppUserManager.Create)
           .CreatePerOwinContext<AppSignInManager>(AppSignInManager.Create);

        // Autorize l'application à utiliser un  cookie pour stocker les informations
        // de l'utilisateur connecté.
        // Configure le cookie de connexion
        app.UseCookieAuthentication(new CookieAuthenticationOptions
        {
            AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
            LoginPath = new PathString("/User/Login"),
            LogoutPath = new PathString("/User/Logout"),
            Provider = new CookieAuthenticationProvider
            {
              // Permet à l'application de valider le tampon (stamp) de sécurité du cookie
              // Indispensable pour des logins externe ou un changement de mot de passe
               OnValidateIdentity = SecurityStampValidator
                                    .OnValidateIdentity<AppUserManager, AppUser>(
                                            validateInterval: TimeSpan.FromMinutes(30),
                                            regenerateIdentity: (manager, user) =>
                                                user.GenerateUserIdentityAsync(manager))
            }
         });

      }
  }
}
```

Les méthodes de type '***App…***' n'existent pas encore, l'éditeur ('intellisense') affiche donc des erreurs.

L'explication des fonctions :

'CreatePerOwinContext' : est une fonction d'extension. Elle permet d'enregistrer une fonction d'initialisation dans le contexte 'Owin'. Les fonctions enregistrées sont invoquées à chaque début de requête qui arrive au serveur IIS. **'CreatePerOwinContext'** est définie dans le 'NuGet package Microsoft.AspNet.Identity.Owin'.

'UseCookieAuthentication' : Configure le site de façon à utiliser des 'cookies' pour la connexion.

En réalité, il charge un middleware d'authentification utilisant des 'cookies'.
Les options pour le 'cookie' sont passées en paramètre : il s'agit d'une instance de la classe ***'CookieAuthenticationOptions'*** qui dérive de la classe ***'AuthenticationOptions'***.

Les options les plus importantes :

- ***'AuthenticationType'*** : est de type string. La chaine de caractères dépend du middleware d'authentification qu'on veut identifier. Il existe des valeurs par défaut, mais on peut choisir à sa guise. Ceci étant dit, ce choix influence le nom du cookie.
- ***'LoginPath'***: Change le résultat HTTP 401 (Accès refusé) en HTTP 302 (redirection) avec le chemin indiqué. Si cette propriété est vide alors il n'y pas de redirection.
- ***'LogoutPath'***: Url de redirection si l'utilisateur se déconnecte
- ***'Provider'***: prend une instance de l'interface ***'ICookieAuthenticationProvider'***. Les méthodes membres sont appelées par le middleware pour influencer le process d'authentification. Microsoft fournit la classe '***CookieAuthenticationProvider'*** qui implémente cette interface. Cette classe permet de définir des 'callback' (fonctions) appelées par le middleware, notamment ***'OnValidateIdentity'*** pour valider une authentification. Ce qui est important si on utilise des logins externes.
- ***'SecurityStampValidator':*** contient la classe `static` ***'OnValidateIdentity',*** utilisée comme validateur dans la propriété ***'Provider'***. ***'OnValidateIdentity'*** vérifie si le temps de vie du 'cookie' a expiré (pour cela la propriété ***'Systemclock'*** doit être renseignée). Dans ce cas elle récupère l'utilisateur et compare le tampon (stamp) de sécurité du cookie avec l'utilisateur. S'il s'agit bien du même 'cookie' alors elle appelle la callback passée par le paramètre ***'regenerateIdentity'*** de la méthode ***'OnValidateIdentity'*** pour régénérer une preuve par réclamation (claim).

A la place de '***UseCookieAuthentication'*** on peut utiliser d'autres méthodes d'extensions :

- '***UseTwitterAuthentication'***,
- '***UseFacebookAuthentication'***,
- etc.

'UseCors' : Cette méthode permet d'installer une extension pour des appels sur plusieurs domaines (cross domaine). Elle nécessite au passage de configurer IIS correctement.

Création des classes de base

D'abord on va créer les répertoires dans le projet 'VisualStudio'. Les classes seront séparées selon leurs objectifs.

Créez une structure de répertoires similaires à la structure suivante dans le projet (clic droit sur le dossier en question,puis *Ajouter -> Nouveau dossier*).

- Dossier ***Db*** dans le répertoire Model
- Dossier ***Account*** dans le répertoire Model
- Dossier ***Account*** dans le répertoire Controllers
- Dossier ***Connection*** dans le répertoire Controllers
- Dossier ***Account*** dans le répertoire Views

Si un des dossiers *Model, App_code,* etc. n'existe pas, faites un *clic droit sur le projet -> Ajouter -> Ajouter le dossier ASP.Net -> et choisissez le dossier* à ajouter.

Il s'agit juste d'une suggestion bien entendu.

Ensuite, il faut créer des classes qui manquent dans la méthode '***ConfigureAuth'***.

Dans Model\Account ajoutez la classe 'AppUser'

Ajoutez le code suivant :

```
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;

namespace TestIdentityMvc5.Models.Account
{
    /// <summary>
    /// Cette classe représente l'utilisateur et implémente l'interface  IUser. Cette
    /// classe dérive de IdentityUser qui est une implémentation de
    /// l'interface IUser par EntityFramework (EF)
    /// Si on n'utilise pas EF  il faut la définir soi-même.
    /// La classe IdentityUser est définie dans le NuGet Package
    /// Microsoft.AspNet.Identity.EntityFramework
    /// </summary>
    public class AppUser : IdentityUser
    {
        public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<AppUser> manager)
        {
           // AuthenticationType doit être le même que celui défini dans les options
            //du startup CookieAuthenticationOptions.AuthenticationType
            var userIdentity = await manager.CreateIdentityAsync(this
                                                , DefaultAuthenticationTypes.ApplicationCookie);

            return userIdentity;
        }
    }
}
```

'AppUser' : Implémente '***IUser'*** interface via ***'IdentityUser'***. Si on veut stocker d'avantages d'informations sur l'utilisateur, il faut ajouter les propriétés correspondantes à cette classe.

'IdentityUser' : est l'implémentation de l'interface '***IUser'*** fournie par 'l'EntityFrameWork' . Cette interface contient également des références sur des rôles, des logins et des réclamations (claims). Cette classe utilise respectivement les classes : '***IdentityUserRole'*** (qui implémente '***TRole'***), '***IdentityUserLogin'*** (qui implémente '***TLogin'***), '***IdentityUserClaim'*** (qui implemente '***TClaim'***).

Dans Model\Account ajoutez la classe 'AppUserManager'

Avec le code suivant :

```
using System;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin;
using TestIdentityMvc5.Models.Db;

namespace TestIdentityMvc5.Models.Account
{
    /// <summary>
    /// Cette classe personnalise la classe UserManager et fournit la méthode d'instanciation
    /// appelée à chaque requête.
    /// </summary>
    public class AppUserManager : UserManager<AppUser>
    {
        public AppUserManager(IUserStore<AppUser> store)
            : base(store)
        {
        }

        /// <summary>
        /// Créer une nouvelle instance de cette classe.
        /// Cette méthode est passée pendant le démarrage du
        /// site au context OWIN qui l'appelle à chaque requête.
        /// Pour l'essentielle, elle configure le Usermanager
        /// </summary>
        /// <param name="options"></param>
        /// <param name="context"></param>
        /// <returns></returns>
        public static AppUserManager Create(  IdentityFactoryOptions<AppUserManager> options
                                            , IOwinContext context)
        {
            var manager = new AppUserManager(
                                    new UserStore<AppUser>(context.Get<AppDbContext>()))
                {
                    // Configuration des verrouillages des utilisateurs
                    UserLockoutEnabledByDefault = true,
                    DefaultAccountLockoutTimeSpan = TimeSpan.FromMinutes(5),
                    MaxFailedAccessAttemptsBeforeLockout = 5
                };

            // Configuration de la validation des utilisateurs
            manager.UserValidator = new UserValidator<AppUser>(manager)
                {
                    AllowOnlyAlphanumericUserNames = false,
                    RequireUniqueEmail = true
                };

            // Configuration de la validation des mots de passe
            manager.PasswordValidator = new PasswordValidator
                {
                    RequiredLength = 6,
                    RequireNonLetterOrDigit = true,
                    RequireDigit = true,
                    RequireLowercase = true,
                    RequireUppercase = true,
                };
```

```
            // Affectation d'une méthode qui permet de générer des tokens,
            // c'est à dire d'encrypter des données statiques ou venant d'un stream
            // Cette méthode est nécessaire pour confirmer ou changer un mot de passe
            var dataProtectionProvider = options.DataProtectionProvider;
            if (dataProtectionProvider != null)
            {
                manager.UserTokenProvider =
                    new DataProtectorTokenProvider<AppUser>(
                            dataProtectionProvider.Create("ASP.NET Identity"));
            }
            return manager;
        }

    }
}
```

La classe ***'AppUserManager'*** dérive de la classe '***UserManger'*** qui fournit les fonctionnalités de base pour la gestion des utilisateurs à savoir des fonctionnalités 'CRUD' de base (Ajouter, supprimer, etc).

Pour cet exemple, uniquement la méthode ***'Create'*** est enregistrée dans le contexte 'Owin' pendant le démarrage du projet (pour rappel : '***OwinStartup.Auth.cs'***).

Pour l'essentielle, elle positionne des paramètres pour l'authentification d'un utilisateur.

La propriété ***'UserTokenProvider'*** définit l'instance à utiliser pour protéger les données. Elle est utilisée pour générer et valider des 'token' pendant le changement d'un mot de passe ou dans le cas d'une confirmation en deux étapes (par email ou SMS)

L'instance de la classe ***'IdentityFactoryOptions',*** passée en paramètre, contient la propriété ***'DataProtectionProvider'*** qui implémente les 'Api' de 'Owin Data Protection'

Dans Model\Account ajoutez la classe 'AppSignInManager'

```
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin;
using Microsoft.Owin.Security;

namespace TestIdentityMvc5.Models.Account
{

   /// <summary>
   /// La classe 'AppSignInManager'  prend en charge la configuration de l'authentification.
   /// </summary>
   public class AppSignInManager : SignInManager<AppUser, string>
   {
      public AppSignInManager(AppUserManager userManager, IAuthenticationManager authenticationManager)
         : base(userManager, authenticationManager)
      {
      }

      /// <summary>
      /// Créer une preuve de réclamation (claim)
      /// </summary>
      /// <param name="user">Utilisateur pour le quel il faut générer une preuve</param>
      /// <returns>Nouvelle preuve</returns>
      public override Task<ClaimsIdentity> CreateUserIdentityAsync(AppUser user)
      {
         return user.GenerateUserIdentityAsync((AppUserManager)UserManager);
      }

      /// <summary>
      /// Création d'une nouvelle instance de AppSignInManager
      /// </summary>
      public static AppSignInManager Create(IdentityFactoryOptions<AppSignInManager> options, IOwinContext context)
      {
         return new AppSignInManager(context.GetUserManager<AppUserManager>(), context.Authentication);
      }
   }
}
```

Dans Model\Db ajoutez la classe 'ApplicationDbContext'

```
using Microsoft.AspNet.Identity.EntityFramework;
using TestIdentityMvc5.Models.Account;

namespace TestIdentityMvc5.Models.Db
{
    /// <summary>
    /// Cette classe représente l'interface avec EntityFramework
    /// Elle prend notamment en charge  la création de la base de données si elle n'existe pas
    /// </summary>
    public class AppDbContext : IdentityDbContext<AppUser>
    {
        /// <summary>
        /// Construit une nouvelle instance de la classe AppDbContext
        /// </summary>
        public AppDbContext()
            : base("SqlServerConnection", throwIfV1Schema: false)
        {
        }

        /// <summary>
        /// Construit une nouvelle instance de la classe AppDbContext
        /// </summary>
        /// <returns>La nouvelle instance AppDbContext</returns>
        public static AppDbContext Create()
        {
            return new AppDbContext();
        }
    }
 }
```

La classe ***'AppDbContext'*** dérive de ***'IdentityDbContext',*** implémentée dans 'Asp.net Identity EF'. Elle gère l'accès à la base de données dans un 'code-first' modèle.

Le nom ***'SqlServerConnection'*** représente la chaine de connexion. Il doit être ajouté au projet par le biais du fichier `'web.config'`. La chaine est à adapter en fonction de la base de données à utiliser.

Pour plus d'informations vous pouvez regarder : http://msdn.microsoft.com/en-us/library/jj653752(v=vs.110).aspx

La syntaxe de la chaine de connexion est la suivante :

```
 <configuration>
<configSections>    <!—Obligatoirement premier balise après la balise <configuration> sinon erreur
d'exécution-->
            ….
      </configSections>

 <connectionStrings>
     <add name="SqlServerConnection"
          connectionString="Data Source=<<mettez ici le nom du
serveur>>;Initial Catalog=<<mettez ici le nom de votre base de
données>>;Integrated Security=True"
          providerName="System.Data.SqlClient"/>
  </connectionStrings>

...

</configuration>

Attention : il faut supprimer tous les retour-chariots de la
chaîne de caractères « connectionstring ».
```

Note : si 'intellisense' de 'Visualstudio' pose problème, retapez la chaîne entièrement. Idem si vous avez une erreur d'exécution dans le fichier web.config.

Note : La création de la chaîne de connexion peut s'avérer épineux, n'hésitez pas à retaper la chaîne entièrement ou à défaut de la créer en utilisant la section 'Paramètres' des propriétés du projet (ce qui va générer un fichier 'Settings.setting' qui a son tour génère une entrée dans le fichier 'Web.Config' . Vérifiez dans tous les cas la chaîne générée : le nom ('SqlServerConnection' dans cette exemple) est strict et ne supporte pas la notion avec des points générés par le fichier 'Settings'.

A ce point la mécanique de l'authentification est intégrée. Le projet devrait compiler sans erreur et même s'exécuter mais bien entendu sans aucune restriction d'accès.

Création des contrôleurs et des vues

Nous commençons par les classes pour enregistrer un nouvel utilisateur.
Premièrement nous allons créer une classe de base avec les traitements partagés.

Ajoutez la classe ***'AccountBaseClass'*** dans le répertoire *'Controllers\Account'*.
On peut paramétrer les classes si on prévoit d'avoir plusieurs 'UserManager'.

```
using System.Web;
using System.Web.Mvc;

using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using TestIdentityMvc5.Models.Account;


namespace TestIdentityMvc5.Controllers.Account
{
   /// <summary>
   /// Classe de base contenant les traitements génériques, nécessaires à la gestion de
   /// la connexion des utilisateurs.
   /// </summary>
   public class AccountBaseClass : Controller
   {
      /// <summary>
      /// Construit une nouvelle instance des la classe AccountBaseClass
      /// </summary>
      public AccountBaseClass()
      {

      }

      #region Les propriétés nécessaires à la connexion

      private AppUserManager _AppUserManager;
     /// <summary>
      /// Obtient ou positionne le gestionnaire d'utilisateurs
      /// </summary>
      public AppUserManager AppUserManager
      {
         get {   return _AppUserManager ?? HttpContext.GetOwinContext().GetUserManager<AppUserManager>();  }
         private set          { _AppUserManager = value;}
      }

      private AppSignInManager _AppSignInManager;
      /// <summary>
      /// Obtient ou positionne  le gestionnaire de connexion
      /// </summary>
      public AppSignInManager AppSignInManager
      {
         get             {       return _AppSignInManager ?? HttpContext.GetOwinContext().Get<AppSignInManager>();    }
         private set { _AppSignInManager = value; }
      }

      #endregion

      #region Fonctions utilitaires
      /// <summary>
      /// Obtient ou positionne le gestionnaire d’authentification
      /// </summary>
      protected IAuthenticationManager AuthenticationManager
      {
         get      {    return HttpContext.GetOwinContext().Authentication;       }
      }
```

```
    /// <summary>
    /// Utilitaire pour gérer les erreurs
    /// </summary>
    protected void AddErrors(IdentityResult result)
    {
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError("", error);
        }
    }
    /// <summary>
    /// Redirige vers une url donnée
    /// </summary>
    protected ActionResult RedirectToLocal(string returnUrl)
    {
        if (Url.IsLocalUrl(returnUrl))
        {
            return Redirect(returnUrl);
        }
        return RedirectToAction("Index", "Default");
    }

    #endregion
}
```

Ensuite, dans le même répertoire ajoutez la classe ***'ChallengeResult'*** :

```
using System.Web;
using System.Web.Mvc;
using Microsoft.Owin.Security;

namespace TestIdentityMvc5.Controllers.Account
{
    internal class ChallengeResult : HttpUnauthorizedResult
    {
        // Utilisé pour la protection contre les attaques de type XSRF
        private const string XsrfKey = "XsrfId";

        public string LoginProvider { get; set; }
        public string RedirectUri { get; set; }
        public string UserId { get; set; }

        public ChallengeResult(string provider, string redirectUri)
            : this(provider, redirectUri, null)
        {
        }

        public ChallengeResult(string provider, string redirectUri, string userId)
        {
            LoginProvider = provider;
            RedirectUri = redirectUri;
            UserId = userId;
        }


        public override void ExecuteResult(ControllerContext context)
        {
            var properties = new AuthenticationProperties { RedirectUri = RedirectUri };
            if (UserId != null)
            {
                properties.Dictionary[XsrfKey] = UserId;
            }
            context.HttpContext
               .GetOwinContext()
               .Authentication.Challenge(properties, LoginProvider);
        }
    }

}
```

Pour les contrôleurs, j'ai choisi de deviser les contrôleurs en deux : La classe ***'ConnectionConttroller'*** contient le code pour la connexion d'un utilisateur existant, l'enregistrement d'un nouvel utilisateur et la déconnexion.
La classe ***'AccountController'*** contient les autres fonctionnalités liés aux comptes et disponibles pour les utilisateurs comme un mot de passe oublié ou la confirmation du mot de passe par email.
Cependant le code de la classe ***'AccountController'*** n'a pas été implémenté pour cet exemple.

Ajoutez la classe ***'ConnectionController'*** dans le répertoire *'Controllers\Account'* créé auparavant.

```
using System.Threading.Tasks;
using System.Web.Mvc;
using Microsoft.AspNet.Identity.Owin;
using TestIdentityMvc5.Models.Account;

namespace TestIdentityMvc5.Controllers.Account
{
    public class ConnectionController : AccountBaseClass
    {
        #region Register Controller : enrergistrement d'un nouvel utilisateur

        //
        // GET: /Connection/Register
        [AllowAnonymous]
        public ActionResult Register()
        {
            return View();
        }

        //
        // POST: /Connection/Register
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Register(RegisterViewModel model)
        {
            if (ModelState.IsValid)
            {
                // Créer un nouvel utilisateur avec les informations
                // fournies dans le formulaire d'enregistrement
                var user = new AppUser { UserName = model.Email, Email = model.Email };
                // Créer l'utilisateur. Ce qui va ajouter l'utilisateur dans la base de données
                var result = await AppUserManager.CreateAsync(user, model.Password);
                if (result.Succeeded)
                {
                    await AppSignInManager.SignInAsync(user
                                                       , isPersistent: false
                                                       , rememberBrowser: false);
                    return RedirectToAction("Index", "Default");
                }
                AddErrors(result);
            }

            // si on arrive ici il y avait un problème auparavant,
            // on réaffiche la page de connexion
            return View(model);
        }

        #endregion

        #region  Login Controller : connexion d'un utilisateur existant
        //
        // GET: /Connection/Login
        [AllowAnonymous]
        public ActionResult Login(string returnUrl)
        {
            ViewBag.ReturnUrl = returnUrl;
            return View();
        }
```

```
        //
        // POST: /Connection/Login
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            // Mettre shouldLockout à true pour activer le comptage de tentatives
            // de connexions et le verrouillage des comptes utilisateurs
            var result = await AppSignInManager
                        .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.Failure:
                default:
                    ModelState.AddModelError("", "La connexion a échouée");
                    return View(model);
            }
        }

        #endregion

        #region Logoff - Déconnexion
        //
        // POST: /Account/LogOff
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult LogOff()
        {
            AuthenticationManager.SignOut();
            return RedirectToAction("Index", "Default");
        }

        #endregion
    }
}
```

Ajoutez les modèles de données

Dans le répertoire *'Models\ Account'* ajoutez les classes suivantes:

- ***LoginViewModel.cs***
- ***RegisterViewModel.cs***

  Ces classes servent à gérer les informations de connexions. Si vous voulez utiliser d'autres informations d'indentification, il faut modifier ces classes aussi.

Respectivement les codes sources de ces deux classes à titre d'exemple:

```
using System.ComponentModel.DataAnnotations;

namespace TestIdentityMvc5.Models.Account
{

    public class LoginViewModel
    {
        [Required]
        [Display(Name = "Email")]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Mot de passe")]
        public string Password { get; set; }

        [Display(Name = "Se rappeler de moi?")]
        public bool RememberMe { get; set; }
    }

}
```

RegisterViewModel.cs :

```
using System.ComponentModel.DataAnnotations;
namespace TestIdentityMvc5.Models.Account
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "Le mot de passe {0} doit avoir au minimum {2} characters.",
MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Mot de passe")]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirmez votre mot de passe")]
        [Compare("Password", ErrorMessage = "Les mots de passe sont différents")]
        public string ConfirmPassword { get; set; }
    }
}
```

Ajoutez maintenant les vues correspondantes

Dans le répertoire *'Views\Connection'* ajoutez les fichiers Razor suivants :

- ***Login.cshtml***
- ***Register.cshtml***

'***Login.cshtml***' reçoit le code pour saisir les informations d'authentification existantes:

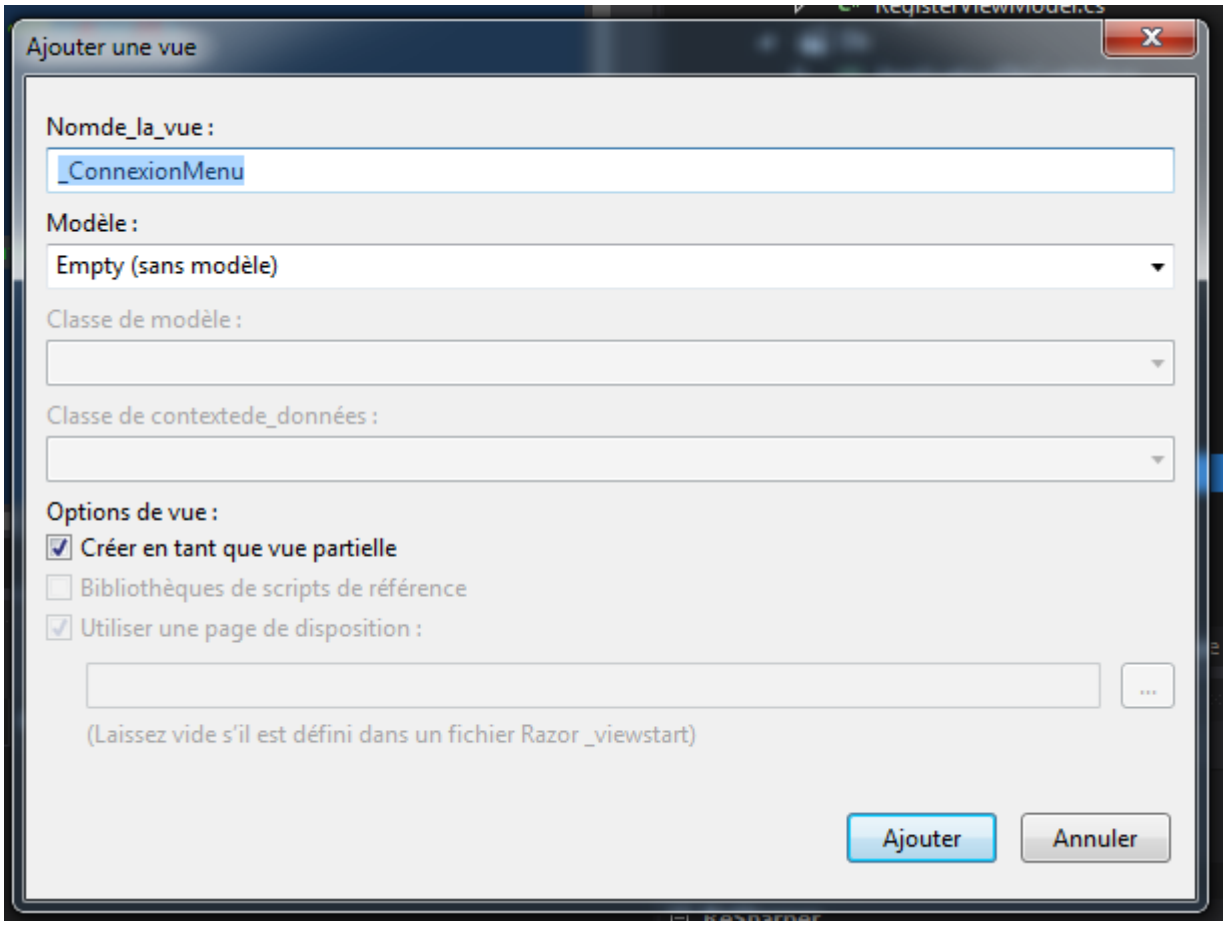
```
@model TestIdentityMvc5.Models.Account.LoginViewModel

@if (!Request.IsAuthenticated)
{

   <div class="login_page float-right" style="width:auto;">

        <div class="col-md-8">

            @using (Html.BeginForm("Login"
                                   , "Connection"
                                   , new { ReturnUrl = ViewBag.ReturnUrl }
                                   , FormMethod.Post
                                   , new { @class = "form-horizontal", role = "form" }))
            {
                @Html.AntiForgeryToken()
                @Html.ValidationSummary(true, "", new { @class = "text-danger" })
                <div class="row">
                    <div class="col-md-2">@Html.LabelFor(m => m.Email
                                                , new { @class = "control-label" })</div>
                    <div class="col-md-2"> @Html.TextBoxFor(m => m.Email
                                                , new { @class = "form-control" })</div>
                </div>
                <div class="row">
                    <div class="col-md-2">
                      @Html.LabelFor(m => m.Password, new { @class = "control-label" })
                    </div>
                    <div class="col-md-2"> @Html.PasswordFor(m => m.Password
                                                            , new { @class = "form-control"
                                                            , @title = "Entrez votre mot de pa
                                                            sse." })
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-1">
                        @Html.CheckBoxFor(m => m.RememberMe)
                        @Html.LabelFor(m => m.RememberMe)
                    </div>
                </div>
                <div class="row">
                    <div class="col-md-2">
                         <input type="submit" value="Se connecter" class="btn btn-default" />
                      </div>
                </div>



            }
        </div>
        <div class="clear block">
            @Html.ValidationMessageFor(m => m.Email
                                       , ""
                                       , new { @class = "text-danger"
                                       , @title = "Entrez votre adresse e-
                                       mail avec la quelle vous êtes enregistrée." })
            @Html.ValidationMessageFor(m => m.Password, "", new { @class = "text-danger" })
        </div>


    </div>
}
```

'Register.cshtml' reçoit le code pour créer un nouveau compte local :

```
@model TestIdentityMvc5.Models.Account.RegisterViewModel


@{
    ViewBag.Title = "Créer un nouveau compte";
}

<h2>@ViewBag.Title.</h2>
<div>
    @using (Html.BeginForm("Register", "Connection", FormMethod.Post, new { @class = "form-
horizontal", role = "form" }))
    {
        @Html.AntiForgeryToken()

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

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

            <div class="col-md-10">
                 @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
                @Html.PasswordFor(m => m.Password, new { @class = "form-control" })
            </div>
        </div>
        <div class="form-group">

            <div class="col-md-10">
                @Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })
                @Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" class="btn btn-default" value="Register" />
            </div>
        </div>
    }
</div>
```

Pour finaliser il faut adapter le menu pour donner accès aux formulaires de connexion et d'enregistrement.

Créez donc un fichier ***'_ConnexionMenu.cshtml'*** (créez une vue mais cochez la case 'Créer en tant que vue partielle') dans le répertoire '*Views\Shared*'.



Mettez un contenu équivalent au suivant :

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

    <ul class="list-unstyled">
        <li>
            @Html.ActionLink("Bonjour " + User.Identity.GetUserName() + "!"
                            , "Index"
                            , "Manage"
                            , routeValues: null, htmlAttributes: new { title = "Manage" })
        </li>
        <li>
            <a href="javascript:document.getElementById('logoutForm').submit()">
              Déconnexion
            </a>
        </li>

    </ul>
    }
}
else
{
    <ul class="nav">
        <li>@Html.ActionLink("Créer un compte"
                            , "Register"
                            , "Connection"
                            , routeValues: null
                            , htmlAttributes: new { id = "loginLink" })
    </li>
    </ul>
}
```

Ajoutez ensuite la ligne '***@Html.Partial("_ConnexionMenu")***' à votre menu ou votre page d'accueil.

Note : Dans ce même répertoire ce trouve le fichier '*_layout.cshtml*' qui contient le menu.

Le fichier '_Layout.cshtml' de cet exemple :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - ASP.NET Identity VS2012</title>

    <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
      <link href="~/Content/bootstrap.css" rel="stylesheet" />
    <script src="~/Scripts/jquery-2.1.3.min.js"></script>
    <script src="~/Scripts/modernizr-2.8.3.js"></script>
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">

                @Html.ActionLink("ASP.NET Identity VS 2012", "Index", "Home", null, new { @class =
"navbar-brand" })
                @Html.Partial("_ConnexionMenu")
            </div>

        </div>
    </div>

    <div class="container body-content">
        @RenderBody()
        <hr />
        <footer>
            <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
        </footer>
    </div>


</body>
</html>
```

Licence