Windows Azure Toolkit For Devices 2/ Authentification Windows Phone

Nous allons commencer notre tour de WAT par l'authentification (au sens large) des utilisateurs.

Comme je l'ai expliqué dans la 1ère partie, WAT propose un proxy permettant un accès sécurisé aux données stockées sur Windows Azure. Ceci signifie que le proxy est indispensable à la mise en œuvre de cette seconde partie.

WAT propose 2 manières de gérer les utilisateurs :

ASP.NET Membership	Enregistrement et accès classique des utilisateurs via login et mot de passe.
Azure Access Contro (ACS)	Accès via un fournisseur d'identité tels que Facebook, Yahoo!, Active Directory, etc.

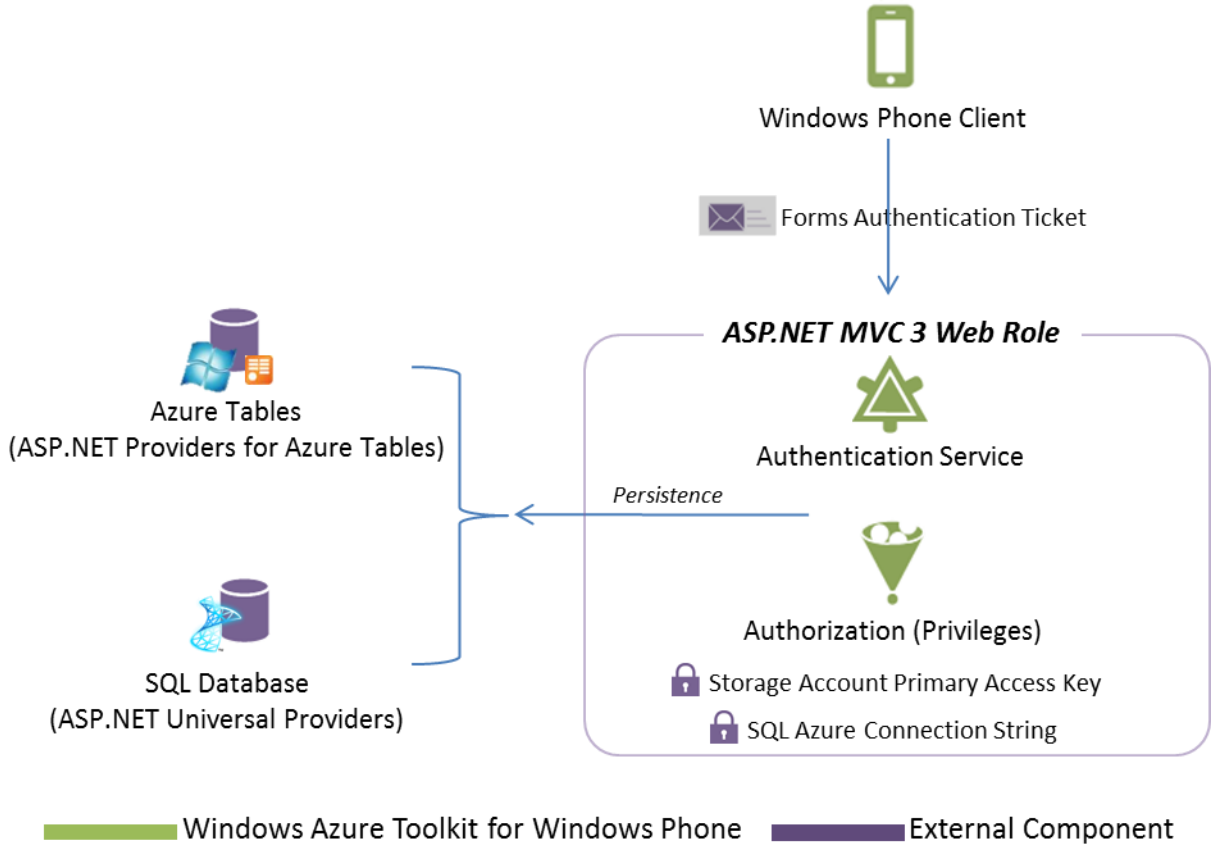
Très important : Comme nous allons utiliser l'émulation Windows Azure (vous allez faire tourner un pseudo Windows Azure sur votre machine de développement, et oui c'est la classe), il faut lancer Visual Studio avec les droits d'administrateur.

Authentification ASP.NET Membership

Je vous rappelle rapidement le principe :

Le proxy WAT se charge de la gestion des utilisateurs. De ce fait il doit les stocker. Vous avez le choix entre deux manière de les stocker dans Azure, soit en utilisant les Tables Azure, soit SQL Azure. Le proxy WAT permet de créer de nouveaux utilisateurs depuis une application mobile et bien sûr de les identifier lors de leur connexion (login / mot de passe).

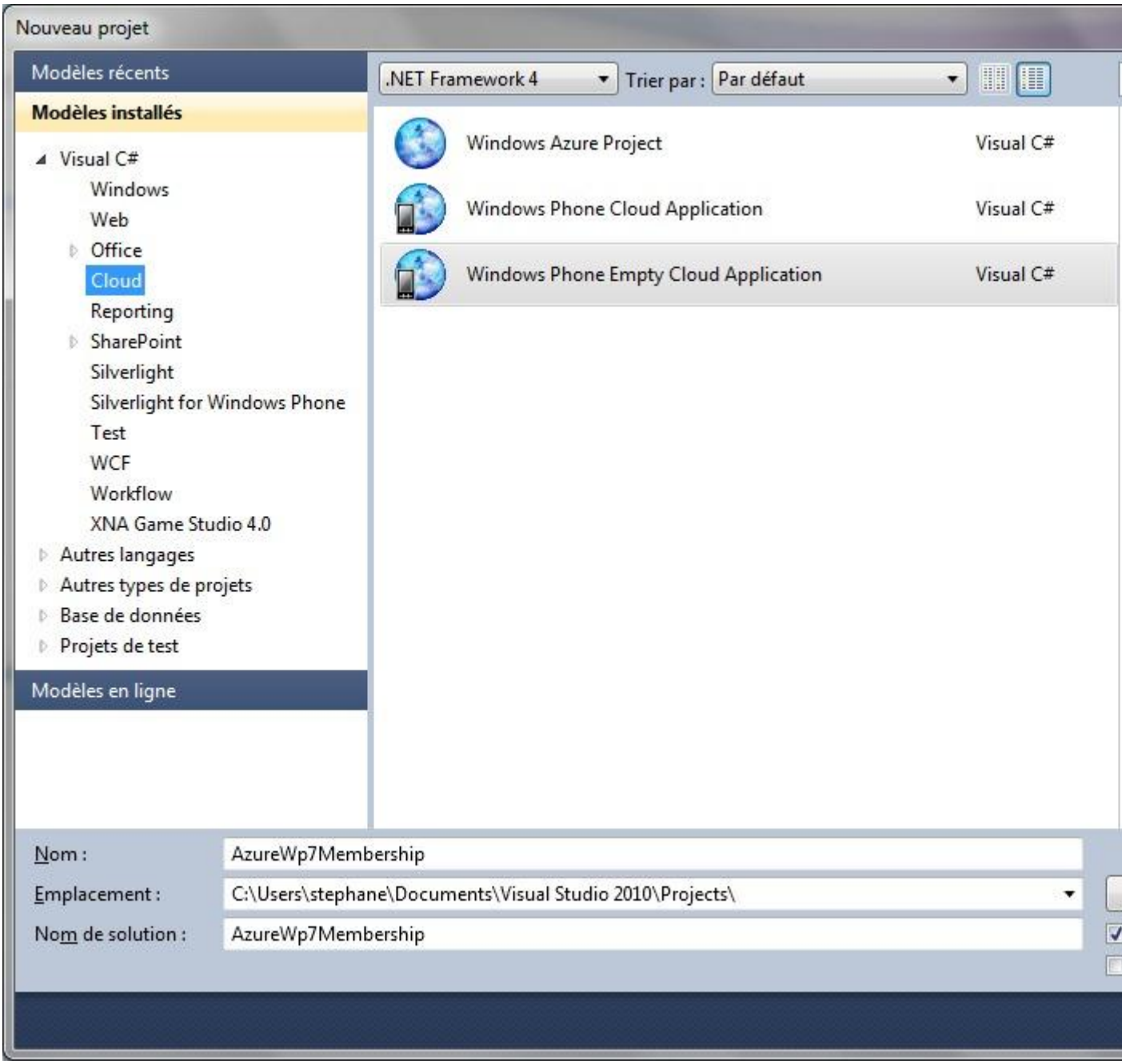
Voici un diagramme qui explique ce fonctionnement.



Pour illustrer cette possibilité nous allons créer un nouveau projet de type « Windows Phone Empty Cloud Application ».

Création du projet

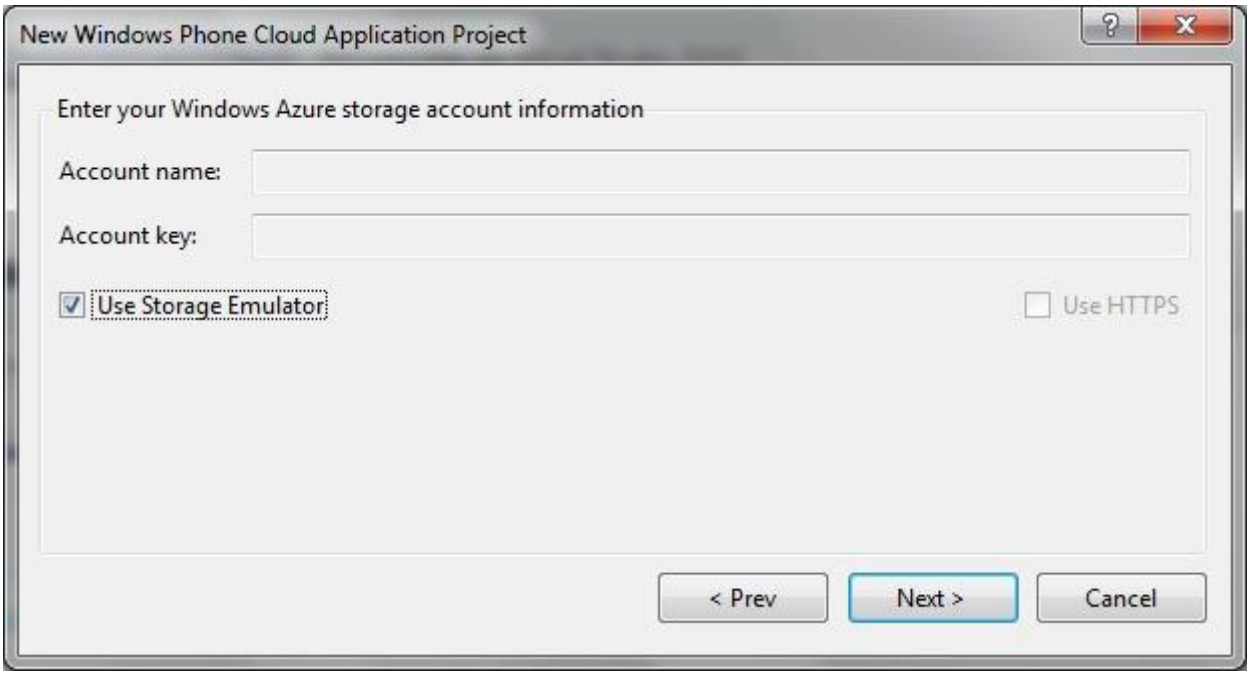
Le Template « Windows Phone Empty Cloud Application » se trouve dans la catégorie « Cloud ». Nous allons lui donner comme nom « AzureWp7Membership » :



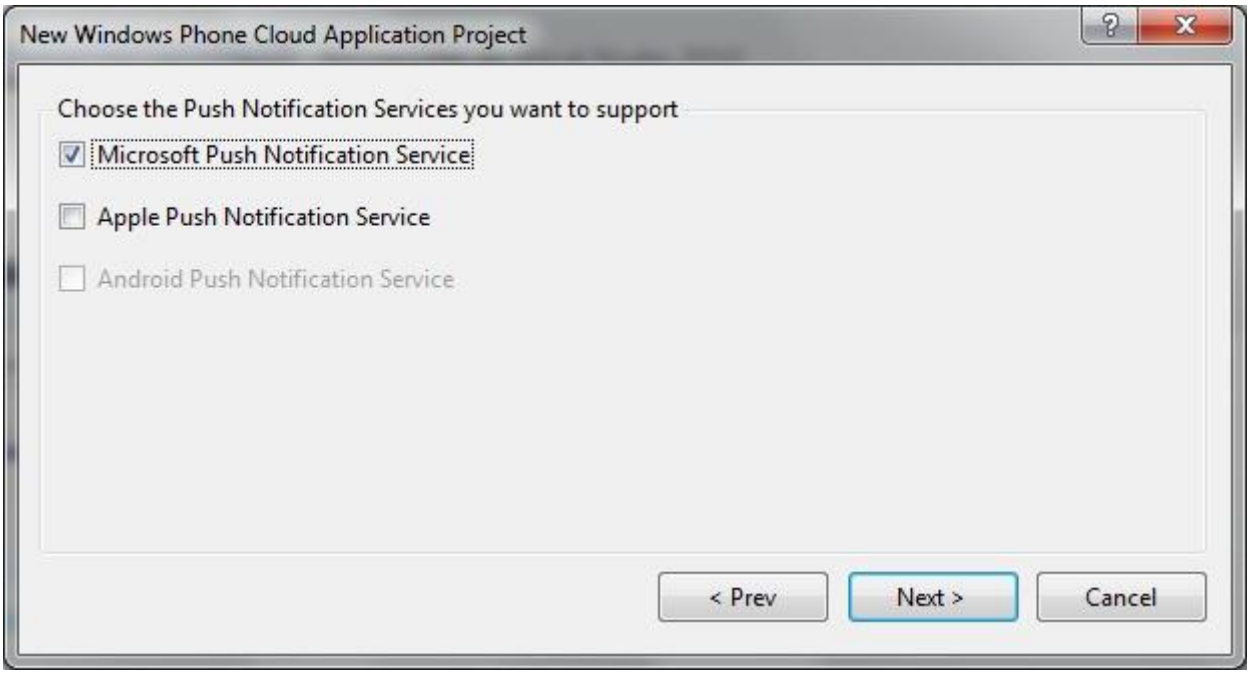
Pour créer un projet correspondant à vos besoins, Visual Studio va ensuite vous poser une série de questions :

Vous allez devoir indiquer quel type de stockage vous souhaitez utiliser sur Windows Azure. Pour notre exemple nous allons utiliser « Windows Azure Storage ».

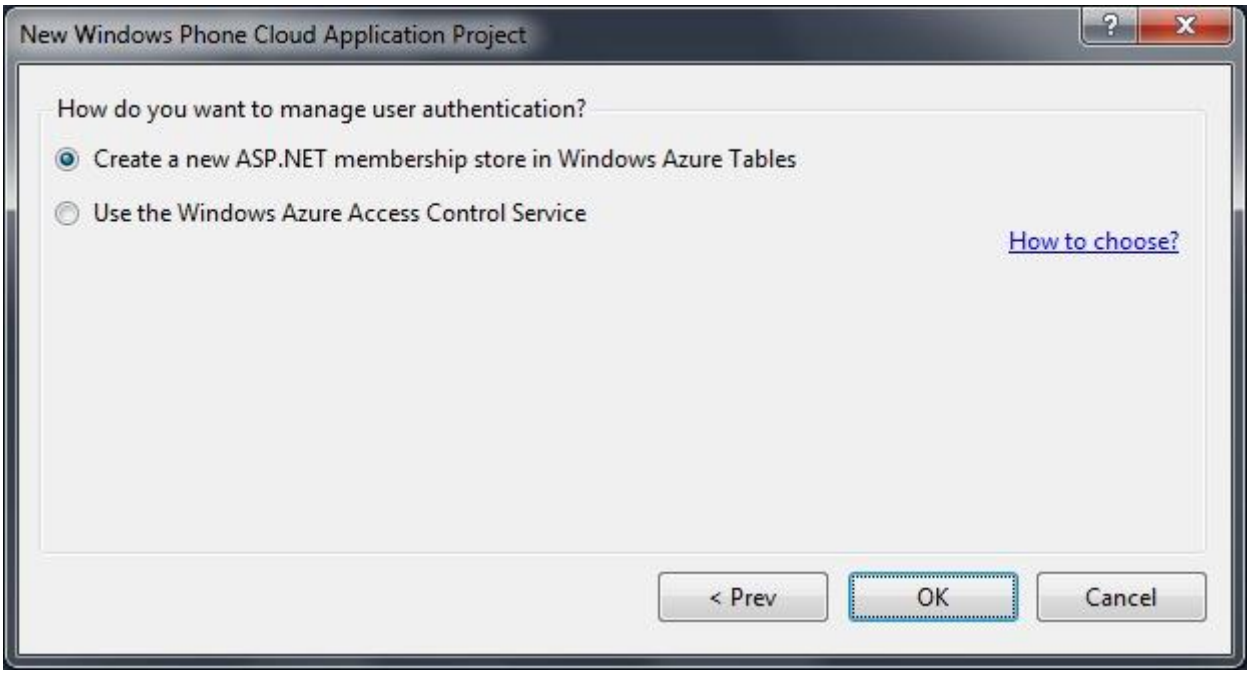
Puis vous allez devoir fournir vos identifiants Windows Azure dans le cas où vous souhaitez tester directement sur Windows Azure. Vous pouvez aussi décider d'utiliser les services de l'émulateur de stockage « Storage Emulator » pour effectuer tous vos tests sur votre propre machine de développement. C'est le choix que nous allons sélectionner pour nos tests.



Vient le tour des notifications. Par défaut Microsoft Push Notification Service est sélectionné. Vous pouvez aussi activer les notifications Apple si besoin. Les notifications Android ne sont pas accessibles pour le moment. Pour notre découverte, nous allons activer les notifications Microsoft seulement (les notifications Apple seront vues plus tard dans un tutoriel consacré aux notifications pour iOS).



Il reste maintenant à indiquer quel type de gestion des utilisateurs vous souhaitez utiliser. Nous allons sélectionner « ASP.NET Membership » qui est ma méthode que nous sommes en train de découvrir.

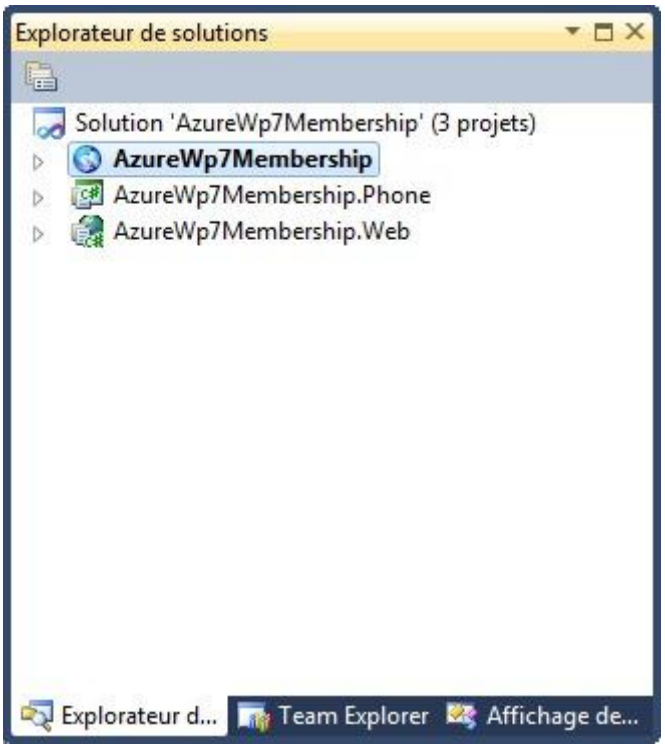


Après avoir répondu à toutes ces questions, Visual Studio prépare la solution

Elle est composée de trois projets :

AzureWp7Membership	Projet Windows Azure
AzureWp7Membership.Phone	Projet Windows Phone (votre application)

AzureWp7Membership.Web	Projet ASP.NET (le proxy WAT)

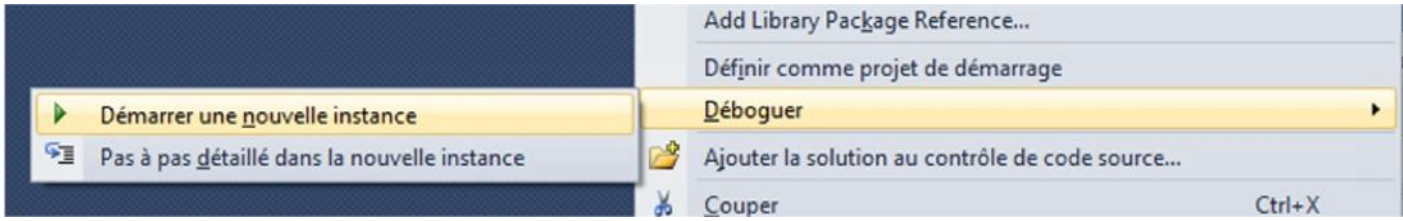


Il est important de noter que par défaut le projet web est configuré pour utiliser le port 10080 en http et 443 en https. Comme nous allons l'utiliser en mode émulation, vous devez vous assurer que votre IIS local ne fait déjà rien tourner sur ces ports.

Pour que toute la chaîne fonctionne, il faut créer et installer des certificats sur Windows Phone. Pour simplifier les choses, nous allons utiliser l'émulateur Windows Phone avec un certificat préparé pour l'occasion par WAT. Il est fait pour ne fonctionner que sur votre machine locale et ne pourra pas être utilisé pour un déploiement réel sur Windows Azure. Comme l'émulateur « oublie » tout dès qu'il est arrêté, il faudra à chaque fois installer ce certificat. Mais rassurez-vous cette opération est très simple à mettre en œuvre.

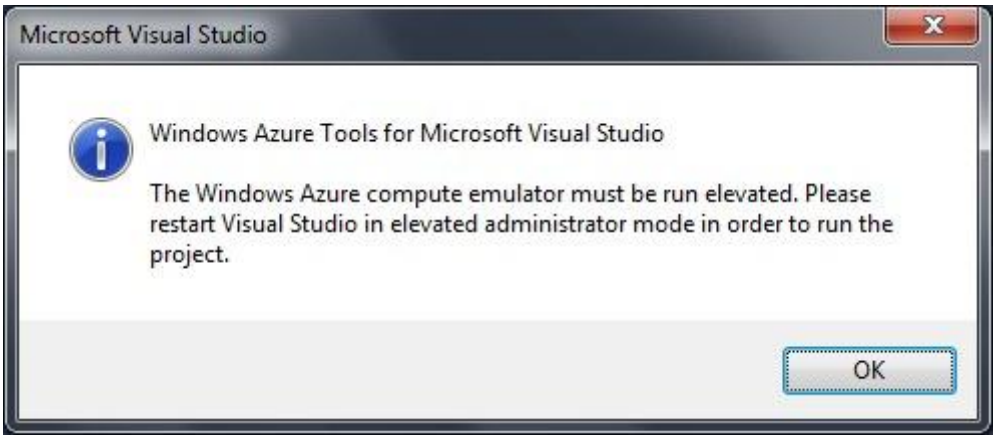
Lancement du proxy WAT

Nous allons maintenant lancer le proxy WAT. La méthode est très simple, il suffit de lancer le projet Windows Azure. Faites un clic droit sur le projet Windows Azure, choisissez « Déboguer » puis « Démarrer une nouvelle instance » (si vous faites F5 directement, vous allez aussi lancer l'émulateur Windows Phone, donc une chose à la fois ça ira déjà bien ☺).



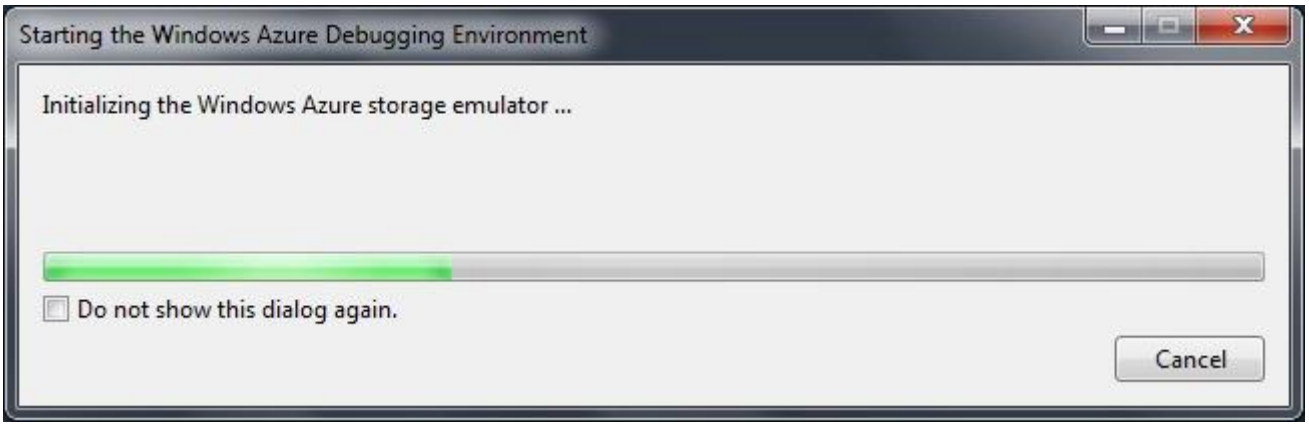
Après quelques instants de compilation, l'émulateur de stockage Windows Azure se lance.

Si ce message d'erreur s'affiche :

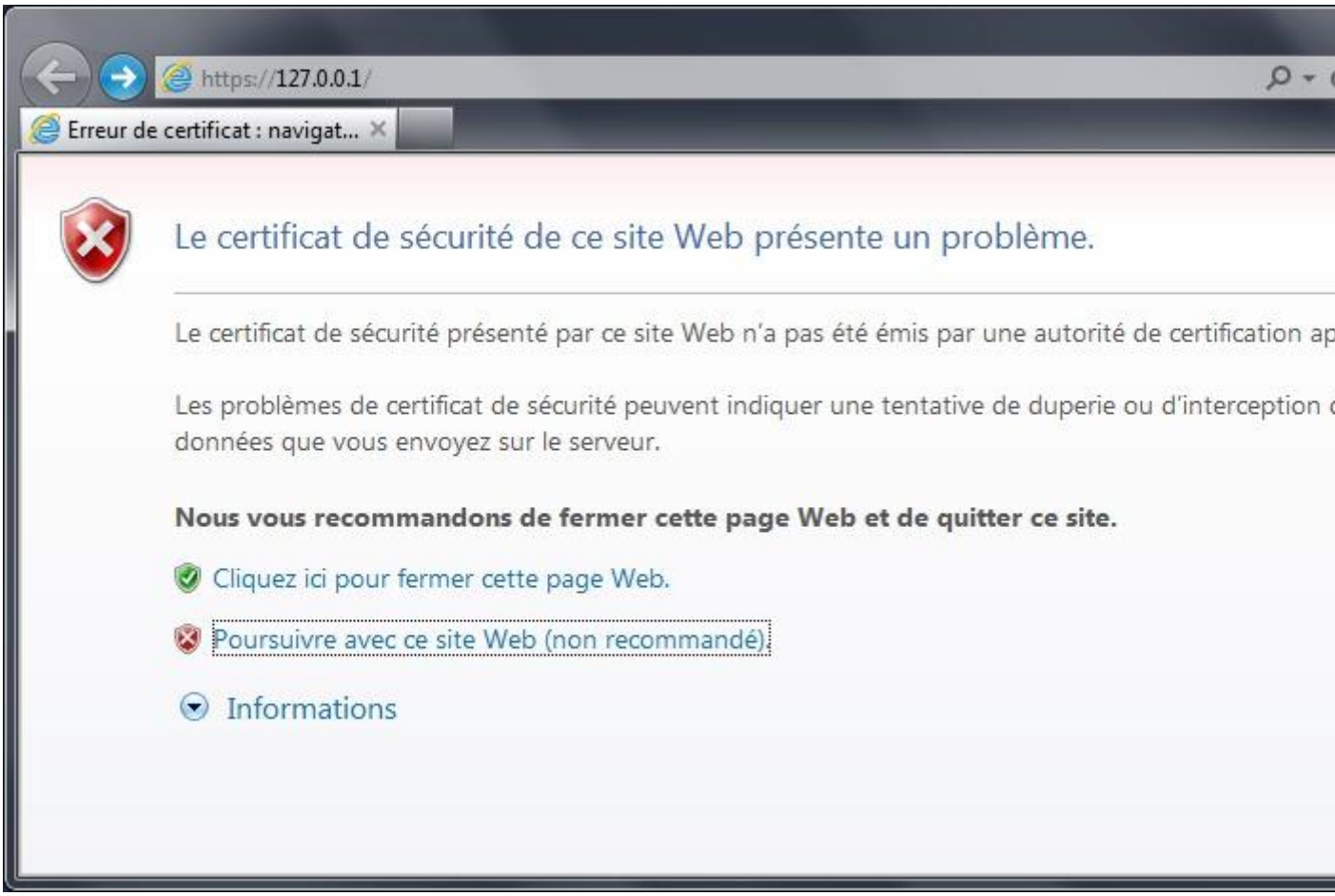


C'est que vous n'avez pas lancé Visual Studio avec les droits d'administrateur. Sans cette élévation des droits, toute la partie émulation de Windows Azure ne peut fonctionner. Dans ce cas, fermez tout, relancez Visual Studio en mode administrateur, ré-ouvrez le projet que nous venons de créer et relancer comme expliqué un peu plus haut.

Si tout se passe bien vous allez voir cette fenêtre apparaitre pendant l'initialisation du « Windows Azure Storage Emulator »

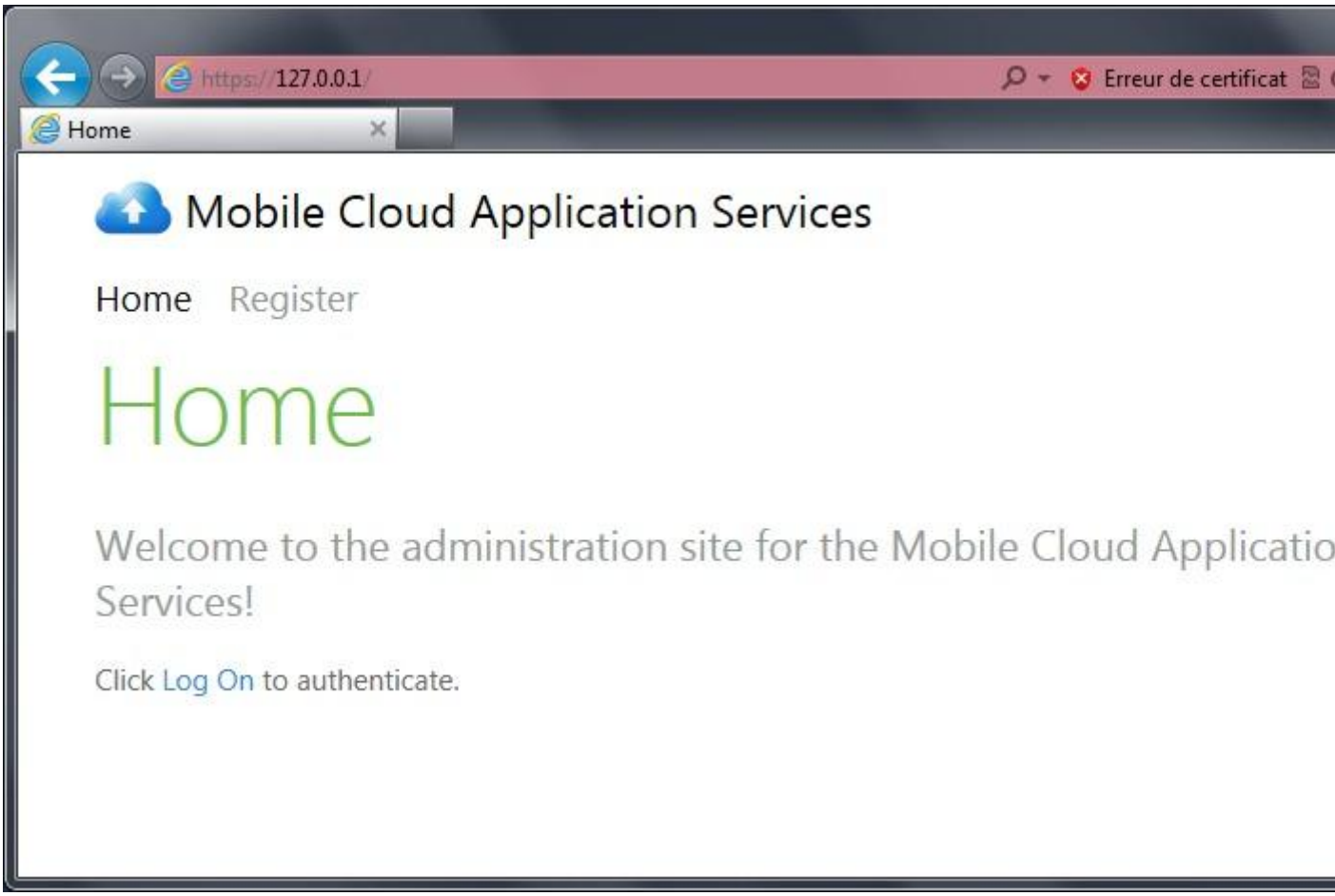


Il faut être patient et laisser faire. Au bout d'un petit moment vous allez voir apparaître dans votre browser Internet cette page :

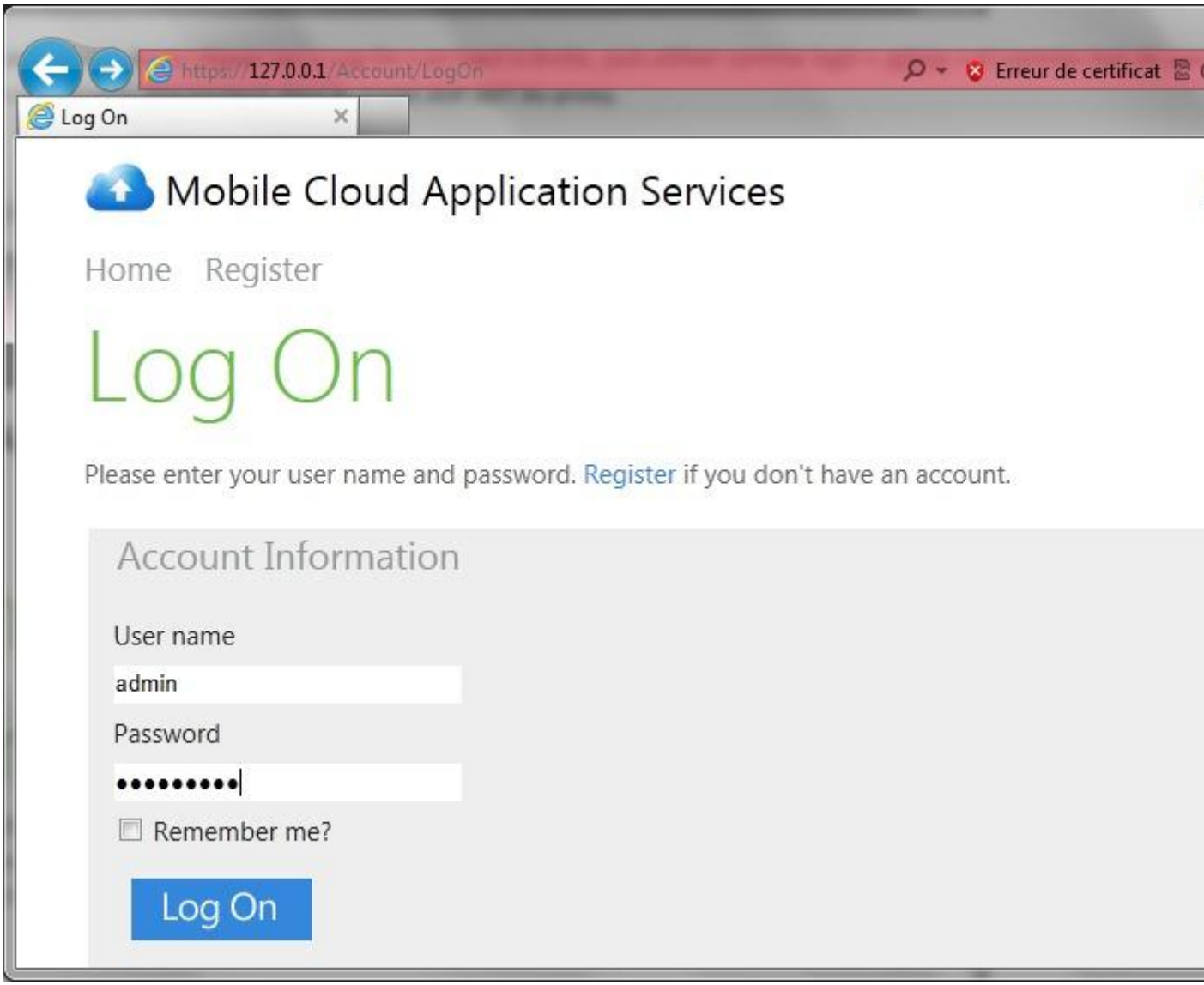


Comme nous utilisons un certificat auto signé cet avertissement est normal. Il suffit d'accepter de poursuivre avec ce site web.

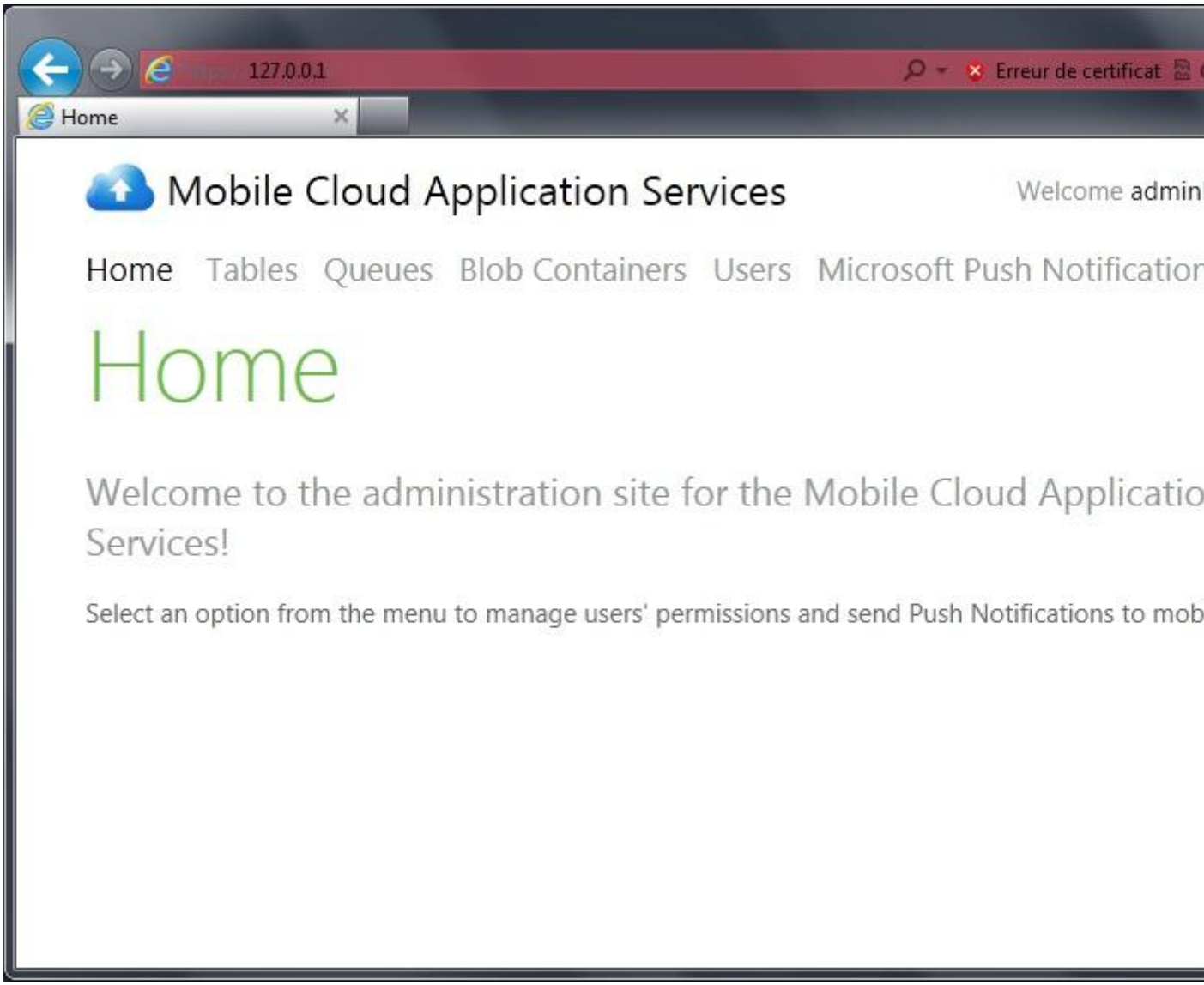
Finalement, on obtient la page principale du proxy WAT (home) :



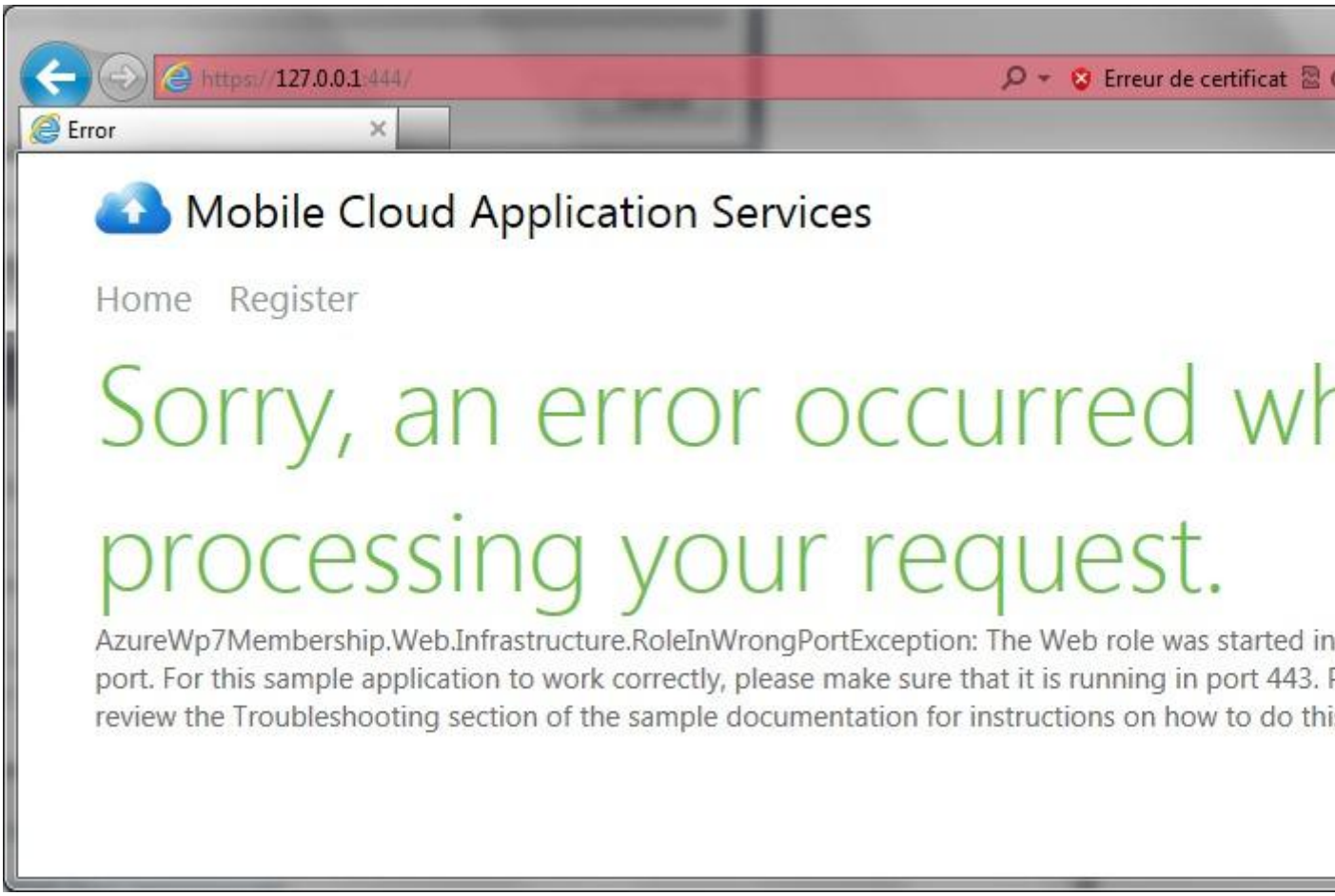
Pour accéder aux fonctions de l'administrateur, vous devez cliquer sur « Log On », puis utiliser comme login « admin » et comme mot de passe « Passw0rd ! » (avec un zéro à la place du O). Il est possible (et bien sûr hautement recommandé) de modifier ces identifiants directement dans le projet ASP.NET du proxy le jour du déploiement vers Windows Azure.



Une fois entré, vous obtenez cette page :



Si à la place de la page home du proxy vous obtenez celle-ci :



C'est que le port 443 n'est pas disponible sur votre machine. Dans ce cas le proxy se lance sur un autre port (typiquement le 444) et plus rien ne fonctionne.

En ce qui me concerne, j'ai eu ce problème avec Skype qui tournait en tâche fond. Une fois Skype arrêté tout est rentré dans l'ordre.

Vous pouvez maintenant stopper l'exécution du proxy car nous allons créer notre projet Windows Phone.

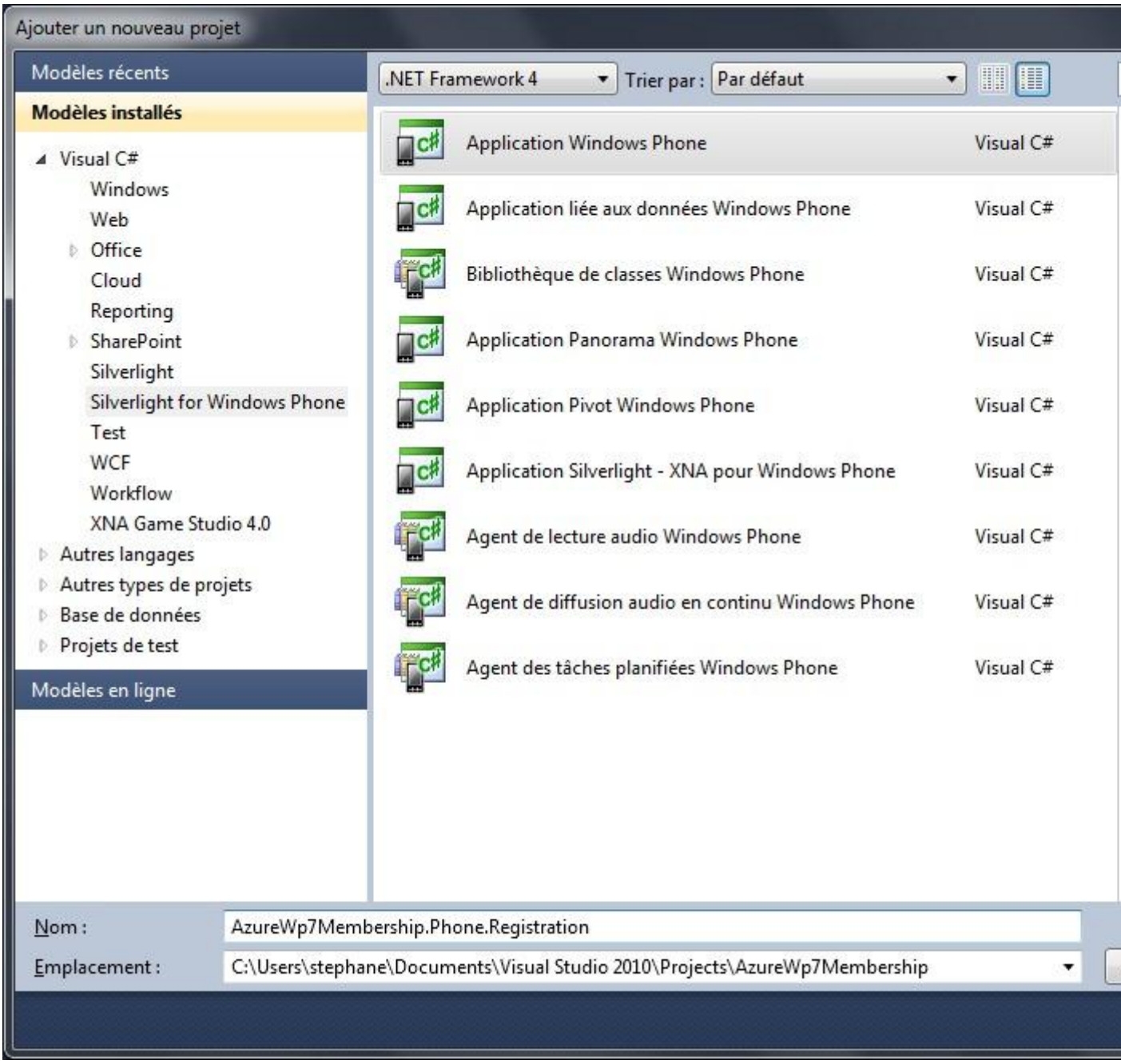
Création du projet Windows Phone

Lorsque nous avons créé notre nouveau projet avec le Template « Windows Phone Empty Cloud Application », Visual Studio nous a automatiquement créé 3 projets, dont un de type Windows Phone. Ce projet est entièrement paramétré pour utiliser les services de WAT sans se soucier de rien (ou presque).

Plutôt que d'utiliser ce projet pré-paramétré, nous allons en créer de nouveaux et les paramétrer nous-même. Ainsi nous allons créer pour cette découverte de l'identification via ASP.NET Membership 2 projets Windows Phone séparés. Le premier aura pour tâche l'enregistrement d'un nouvel utilisateur, et le second, l'identification (via login et mot de passe) d'un utilisateur existant. Le code sera bien plus simple et clair à comprendre en procédant de cette manière.

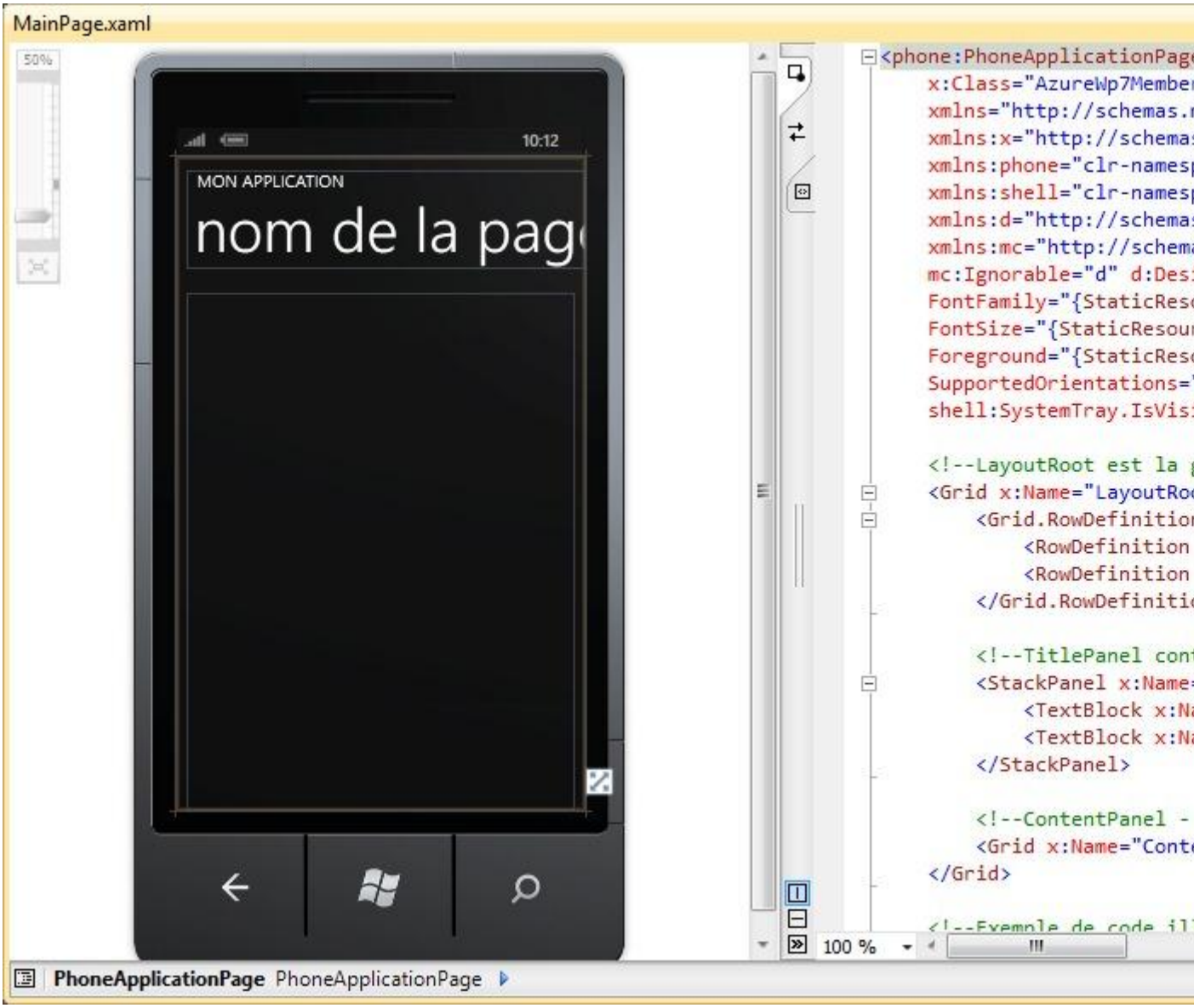
Création du projet d'enregistrement d'un nouvel utilisateur

Allons-y ! Ajoutons le projet « AzureWp7Membership.Phone.Registration » à notre solution déjà existante. C'est un projet de type « Silverlight pour Windows Phone » :



Nous allons ensuite sélectionner comme cible le SDK « Windows Phone OS 7.1 » (qui correspond à la version 7.5 du système d'exploitation, merci au marketing Microsoft au passage) :

Après quelques courts instants, nous nous retrouvons avec un projet classique Windows Phone possédant une page principale « MainPage » :



Un projet Windows Phone devant utiliser les services du proxy WAT pour l'identification ASP.NET Membership doit référencer une assembly particulière qui se trouve dans le dossier « Libs » de la solution et qui porte le nom de « WindowsPhoneCloud.StorageClient.dll». Donc c'est le moment d'ajouter cette référence au projet.

Composition d'une page de saisie des informations de l'utilisateur

Maintenant nous allons composer une interface qui permettra de renseigner les champs nécessaires à la création d'un nouvel utilisateur (C'est déjà tout prêt dans les sources accompagnant ce tutoriel pour les moins courageux).

Nous allons donc ajouter les champs suivants :

Champ	**Propriété Name**
Textbox de saisie du nom de l'utilisateur	TxtUserName
Textbox de saisie de l'adresse mail	TxtMail
PasswordBox de saisie du mot de passe	TxtPassword
PasswordBox de saisie de la confirmation du mot de passe	TxtPasswordBis
Button d'enregistrement	BouRegister
Button d'installation du certificat	BouInstallerCetificat

(Vous avez sans doute remarqué mon savant mélange d'anglais et de français dans les noms des contrôles, c'est tout moi ça !).

Pour arriver en gros à ce résultat :

Comme vous devez vous en douter en regardant cet écran, pour créer un nouvel utilisateur il faut fournir 4 informations au proxy WAT :

1. Le nom de l'utilisateur (son login)
2. Son adresse mail
3. Son mot de passe
4. La confirmation du mot de passe (c'est WAT qui vérifie que les deux sont identiques)

Installation du certificat

Mais avant toute chose, il faut être en mesure d'installer le certificat sur l'émulateur Windows Phone. N'oubliez pas qu'à chaque redémarrage de l'émulateur vous serez obligé de l'installer. Rassurez-vous cette opération n'est nécessaire qu'une seule fois sur une machine réelle.

Nous allons donc ajouter le code suivant à l'événement « Click » du bouton « BouInstallerCertificat » :

```
private void BouInstallerCertificat_Click(object sender, RoutedEventArgs e)
{
    var wTask = new WebBrowserTask();
```

```
    wTask.Uri = new Uri("http://127.0.0.1:10080/127.0.0.1.cer", UriKind.Absolute);
    wTask.Show();
}
```

Note : Vous devez ajouter la directive « Using Microsoft.Phone.Tasks » pour que ce code fonctionne.

AuthenticationClient, la classe centrale pour la gestion des utilisateurs

WAT fournit la classe « AuthenticationClient » qui se trouve dans l'espace de nom « Microsoft.Samples.WindowsPhoneCloud.StorageClient.Credentials ». Elle permet d'effectuer toute la gestion des utilisateurs, c'est-à-dire l'enregistrement, la connexion et la vérification de session.

La fonction login, si elle valide la connexion, renvoie un token de session qui sera utilisé par toutes les opérations possibles avec WAT.

La classe « AuthenticationClient » fournit les méthodes suivantes :

Register	Permet de créer un nouvel utilisateur
Login	Permet de connecter un utilisateur et de récupérer son token de session
Validate	Permet de vérifier que le token est toujous valide

Enregistrement du nouvel utilisateur

Pour être en mesure d'enregistrer un nouvel utilisateur, nous avons besoin d'instancier un objet de type « AuthenticationClient » qui se trouve dans l'espace de nom « Microsoft.Samples.WindowsPhoneCloud.StorageClient.Credentials ». Il vous faut donc ajouter la directive « using Microsoft.Samples.WindowsPhoneCloud.StorageClient.Credentials » et créer une propriété que nous appellerons « AuthClient » :

```
        private AuthenticationClient _authClient;

        private AuthenticationClient AuthClient
        {
            get
            {
                if (_authClient == null)
                {
                    _authClient = new
AuthenticationClient("https://127.0.0.1/AuthenticationService");
                }

                return _authClient;
            }
        }
```

Vous remarquerez dans le code qu'il faut donner l'adresse du service d'authentification du proxy WAT lors de l'instanciation de l'objet d'authentification. Cette adresse se termine toujours par « /AuthenticationService », le début étant conditionné par l'adresse réelle du proxy WAT. Par exemple pour un proxy installé sur Azure sous le nom de « monproxy.cloudapp.net » l'adresse sera « https://monproxy.cloudapp.net/AuthenticationService ».

Nous pouvons maintenant placer le code en réponse à l'événement « Click » du bouton « BouRegister ». Mais avant cela quelques explications s'imposent :

La fonction d'enregistrement n'est pas bloquante, et c'est une très bonne chose. Elle utilise donc des méthodes de rappel (callback) pour exprimer son contentement quand un nouvel utilisateur a été accepté par le proxy WAT, ou son mécontentement dans le cas contraire.

Pour simplifier les choses, nos méthodes de rappel vont simplement afficher un message, et comme elles seront exécutés sur un autre thread que celui de l'interface, nous allons devoir nous créer une petite fonction permettant tout de même un tel affichage :

```
        private void ShowMessageOnMainThread(string wMessage)
        {
            Deployment.Current.Dispatcher.BeginInvoke(
                () =>
                {
                    MessageBox.Show(wMessage);
                });
        }
```

Nous pouvons maintenant créer nos deux délégués, l'un pour le cas où l'enregistrement se passe bien (RegisterOK), et l'autre pour le cas où l'enregistrement est refusé pour quelque raison que ce soit (RegisterKO) :

```
private void RegisterOK(RegistrationSuccessEventArgs e)
{
    ShowMessageOnMainThread("Register OK");
}

private void RegisterKO(RegistrationExceptionEventArgs e)
{
    string wMessage = "Registrer KO !";

    if (e.Exception != null)
    {
        wMessage += string.Format("\nException = {0}", e.Exception.Message);
    }

     wMessage += "\nAvez-vous installé le certificat ?";

    ShowMessageOnMainThread(wMessage);
}
```

Dans le cas où l'enregistrement du nouvel utilisateur se passe mal, nous affichons la raison. Comme souvent le problème vient du certificat qui n'a pas été installé, la fin du message fait un petit rappel sur le sujet.

Nous avons tout maintenant pour effectuer l'enregistrement à proprement parler. Nous allons donc répondre à l'événement « Click » de « BouRegister » de la manière suivante :

```
private void BouRegister_Click(object sender, RoutedEventArgs e)
{
    this.AuthClient.Register(TxtUserName.Text,
        TxtMail.Text,
        TxtPassword.Password,
        TxtPasswordBis.Password,
```

```
        RegisterOK, RegisterKO);
}
```

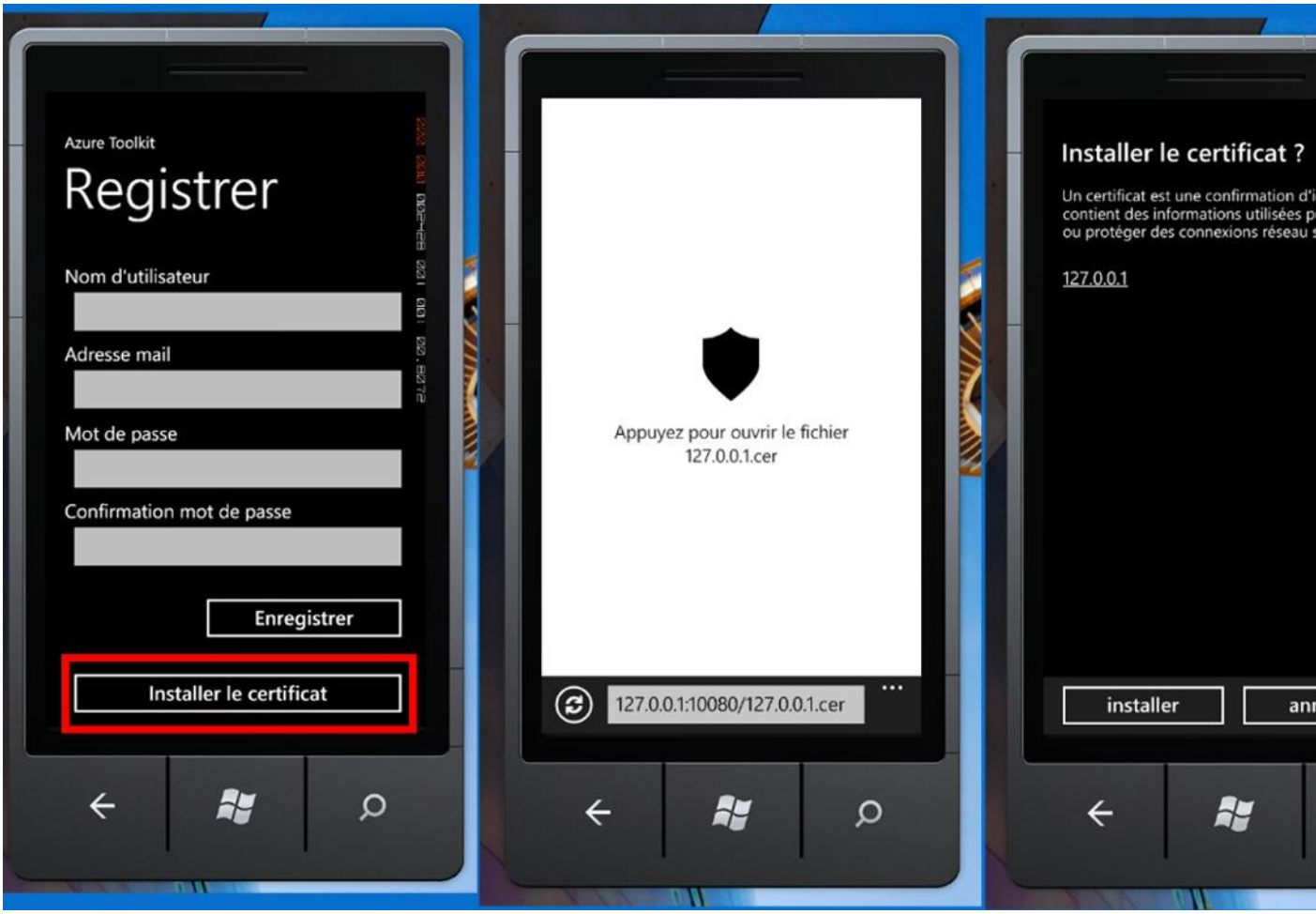
Les 4 informations (nom d'utilisateur, mail, mot de passe et confirmation du mot de passe) sont fournies à la méthode « Registrer » de l'objet « AuthenticationClient » que nous avons géré tout à l'heure via la propriété « AuthClient ». Les méthodes de rappel de réussite et d'échec sont aussi fournies, et c'est tout. Simple non ?

Tester notre super programme

Il ne nous reste plus qu'à tester notre super programme.

Dans un premier temps, lancez le proxy WAT (comme expliqué plus haut), puis effectuez un clic droit sur le projet « AzureWp7Membership.Phone.Registration » et utilisez dans le menu « Deboguer > Démarrer une nouvelle instance » pour lancer sur l'émulateur Windows Phone notre programme d'enregistrement de nouveaux utilisateurs (vous devez veiller à ce que l'application Windows Phone se lance bien sur l'émulateur).

Une fois le programme lancé, vous devez installer le certificat en cliquant sur le bouton prévu à cet effet. Le code que nous avons tapé tout à l'heure lance le browser Internet sur l'adresse du certificat et permet de l'installer :



Une fois le certificat correctement installé on peut saisir les informations d'un nouvel utilisateur :

L'utilisateur est identifié par son nom et aussi par son mail. Il est donc impossible d'enregistrer 2 utilisateurs différents ayant la même adresse mail.

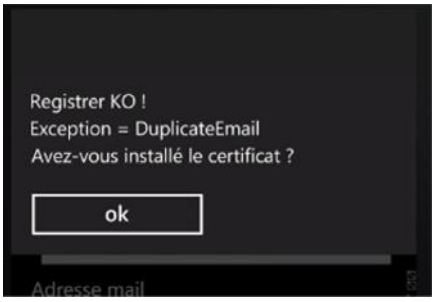
Dans cet exemple nous allons enregistrer l'utilisateur suivant :

Login ou nom	user1
Mail	user1@mail.com
Mot de passe	toto

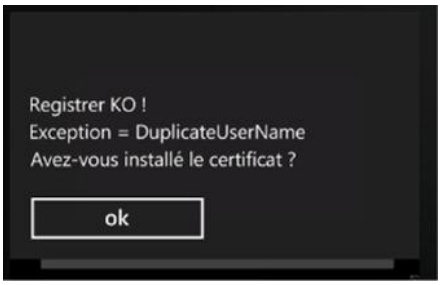
Après quelques secondes nous obtenons l'écran suivant :

Note : Comme l'appel est asynchrone, il faudrait idéalement bloquer les zones de saisie, les boutons et afficher une barre de progression en attendant la réponse.

Maintenant, si on essaye à nouveau d'enregistrer un utilisateur avec les mêmes valeurs, on essuie un refus catégorique (heureusement me direz-vous). Et ce qui le gêne le plus c'est que le mail est identique.

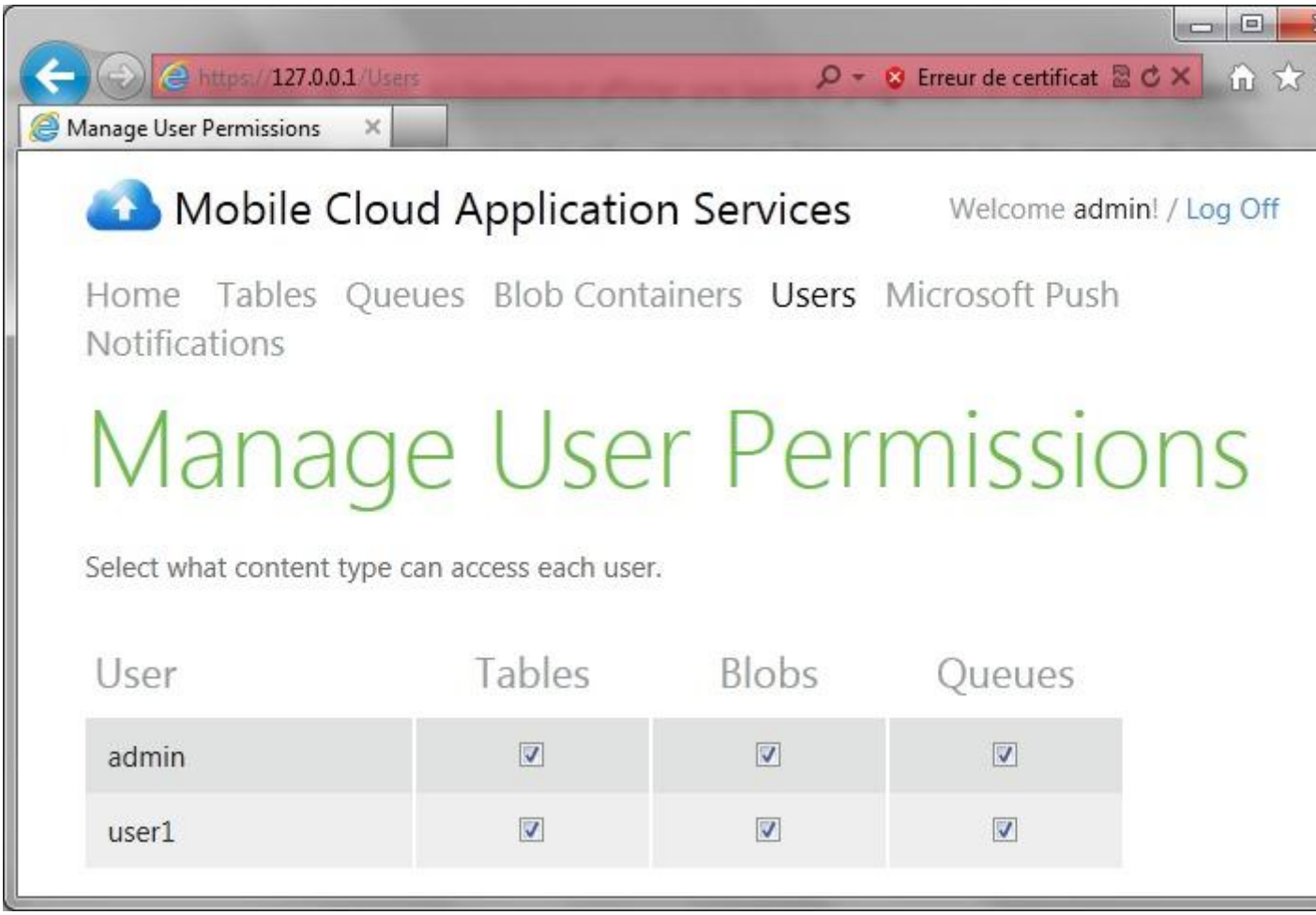


Et si on change le mail sans rien changer d'autre ?



Et bien c'est la duplication du login qui ne lui plait pas.

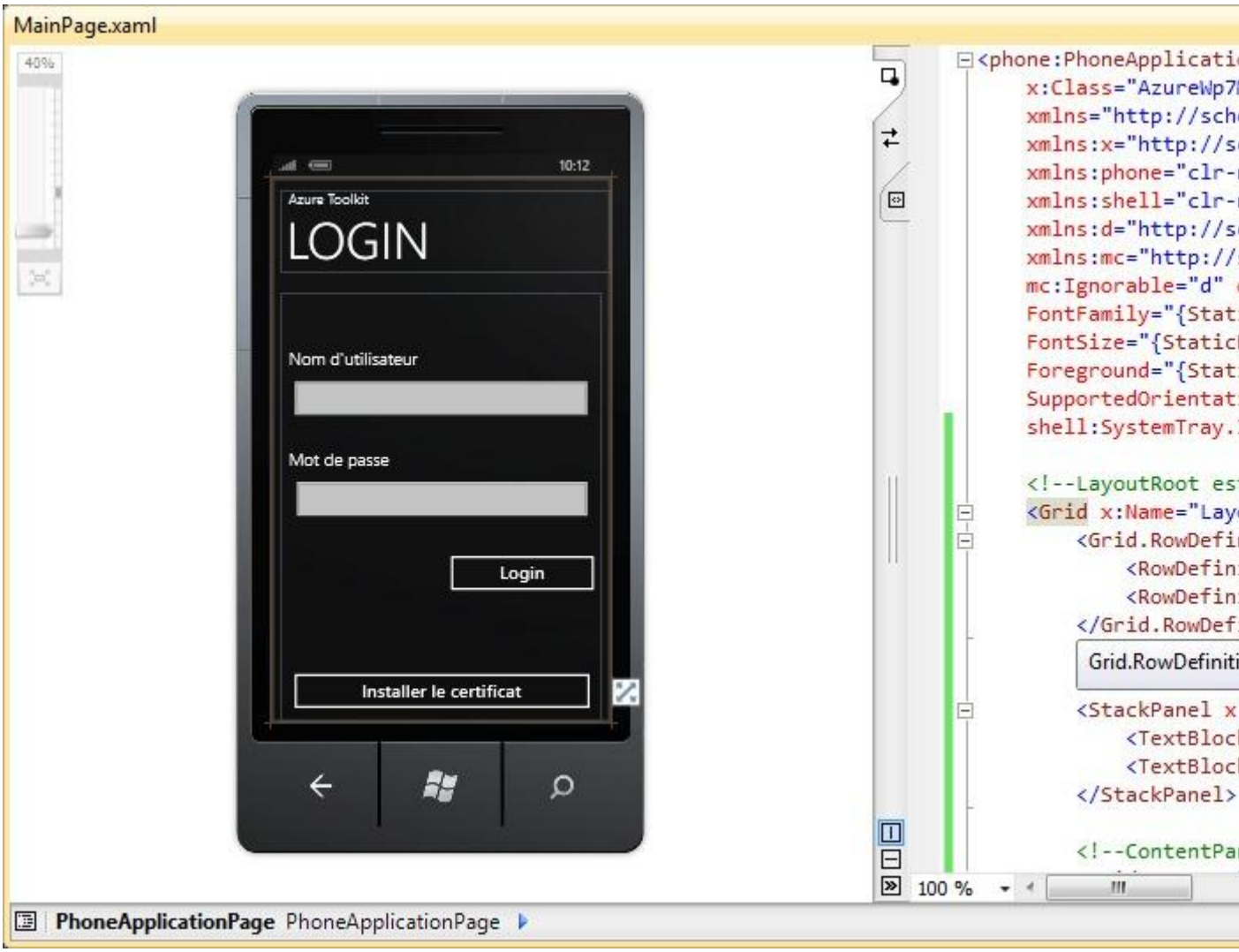
Si vous vous loguez en tant qu'administrateur sur le proxy WAT, vous pourrez constater que l'utilisateur « user1 » est bien présent :



Après l'enregistrement d'un nouvel utilisateur, nous allons maintenant passer à son identification.

Création du programme de login

Comme pour le programme d'enregistrement, nous allons créer un projet Windows Phone pour illustrer l'opération d'identification d'un utilisateur déjà existant. Nous allons donc ajouter un nouveau projet « AzureWp7Membership.Phone.Login » du même type que celui ajouté pour l'enregistrement (Template Silverlight for Windows Phone) et cibler « Windows Phone OS 7.1 ». Une fois créé, nous lui ajoutons la référence vers « WindowsPhoneCloud.StorageClient » et modifions l'interface de « MainPage » pour arriver en gros à ce type d'écran :



Cet écran est composé des éléments suivants :

Champ	**Propriété Name**
Textbox de saisie du nom de l'utilisateur	TxtUserName
PasswordBox de saisie du mot de passe	TxtPassword
Button de login	BouRegistrer
Button d'installation du certificat	BouInstallerCetificat

Comme pour l'enregistrement, il est nécessaire pour le login de disposer un objet de type « AuthenticationClient » (ne pas oublier d'ajouter le using qui va bien comme dans le premier projet). Nous ajoutons donc la propriété « AuthClient » de la même manière. Même chose pour la fonction « ShowMessageOnMainThread » pour afficher le résultat simplement. Le bouton d'installation du certificat doit reprendre le même code aussi car le certificat est aussi nécessaire pour l'opération de login.

Comme l'opération de login est elle aussi asynchrone, la même logique est utilisée. Elle est basée sur l'utilisation d'une méthode de rappel en cas de login réussi, et d'une autre en cas de login échoué. Quand l'opération de login se passe bien, il faut récupérer le token d'accès fourni par le délégué. Ce token sera ensuite utilisé pour d'autres opérations avec le proxy. Le token étant une chaîne de caractères, une simple variable de type « string » fera l'affaire. Dans cet exemple elle est déclarée comme suit :

```
private AuthenticationClient _authClient;
```

Et les méthodes de rappel comme ceci :

```
private void LoginOK(AuthenticationSuccessEventArgs e)
{
    // Le login s'est bien passé
    // On mémorise le token d'accès

    _authToken = e.AuthenticationToken;

    // On affiche le message ok

    ShowMessageOnMainThread("Login ok");
}

private void LoginKO(AuthenticationExceptionEventArgs e)
{
    // Le login s'est mal passé
    // On affiche le message KO

    string wMessage = "Login KO !";

    if (e.WrongCredentials)
    {
        wMessage += "Login et/ou mot de passe incorrects !";
    }
    else if (e.Exception != null)
    {
        wMessage += string.Format("\nException = {0}", e.Exception.Message);
    }

    wMessage += "\nAvez-vous installé le certificat ?";

    ShowMessageOnMainThread(wMessage);
}
```

Le délégué « LoginKO », appelé en cas de problème, possède un argument de type « AuthenticationExceptionEventArgs » qui permet de savoir ce qui ne va pas. Soit le problème vient des informations transmises, et la propriété « WrongCredentials » sera vraie, soit le problème vient de la communication avec WAT, et la propriété « Exception » portera l'exception générée et la propriété « IsError » sera vraie.

Il ne reste plus qu'à répondre à l'événement « Click » du bouton de login en utilisant la méthode « Login » de notre propriété « AuthClient » :

```
private void BouLogin_Click(object sender, RoutedEventArgs e)
{
```

```
    this.AuthClient.Login(TxtUserName.Text, TxtPassword.Password, LoginOK, LoginKO);
}
```

C'est aussi simple à mettre en œuvre que l'enregistrement, vous ne trouvez pas ?

Maintenant on peut lancer les tests en suivant la même procédure que pour l'enregistrement, sauf que là vous allez démarrer le projet « AzureWp7Membership.Phone.Login ».

Une fois le programme Windows Phone lancé, il ne vous reste plus qu'à installer le certificat (sauf si vous n'avez pas coupé l'émulateur depuis la dernière fois), saisir un login et un mot de passe valide et cliquer sur le bouton « Login ».

Si tout se passe bien, vous aurez un message « Login OK », sinon un message « Login KO » avec le texte de l'erreur liée.

Vérification de la validité de la session

Lors de l'opération de login réussie, un token est retourné, il correspond à la session en cours. Ce token a une durée de vie limitée. Vous pouvez le stocker (c'est une chaîne de caractères) et le réutiliser d'une connexion à une autre jusqu'à ce que sa limite de validité soit atteinte.

Pour vérifier que le token est toujours valide, il suffit d'utiliser la méthode « Validate » qui prend en arguments le token à vérifier et comme pour les autres une méthode de rappel si la validation est bonne et une autre dans le cas contraire. C'est aussi simple que ça.

Conclusion

Il est très simple de mettre en œuvre l'identification par ASP.Membership avec WAT. Pour simplifier au maximum, il suffit d'ajouter la référence à « WindowsPhoneCloud.StorageClient.dll », et utiliser les méthodes « Register » pour créer un nouveau compte et « Login » pour connecter un utilisateur existant. Dans chaque situation c'est un objet de type « AuthenticationClient » qui fournit ces méthodes. Quand la méthode « Login » réussit, un token est fourni permettant ensuite d'accéder aux services de stockage proposés par Windows Azure.

Nous allons maintenant parler de l'authentification via « Windows Azure Access Control Service » ou ACS pour les intimes.

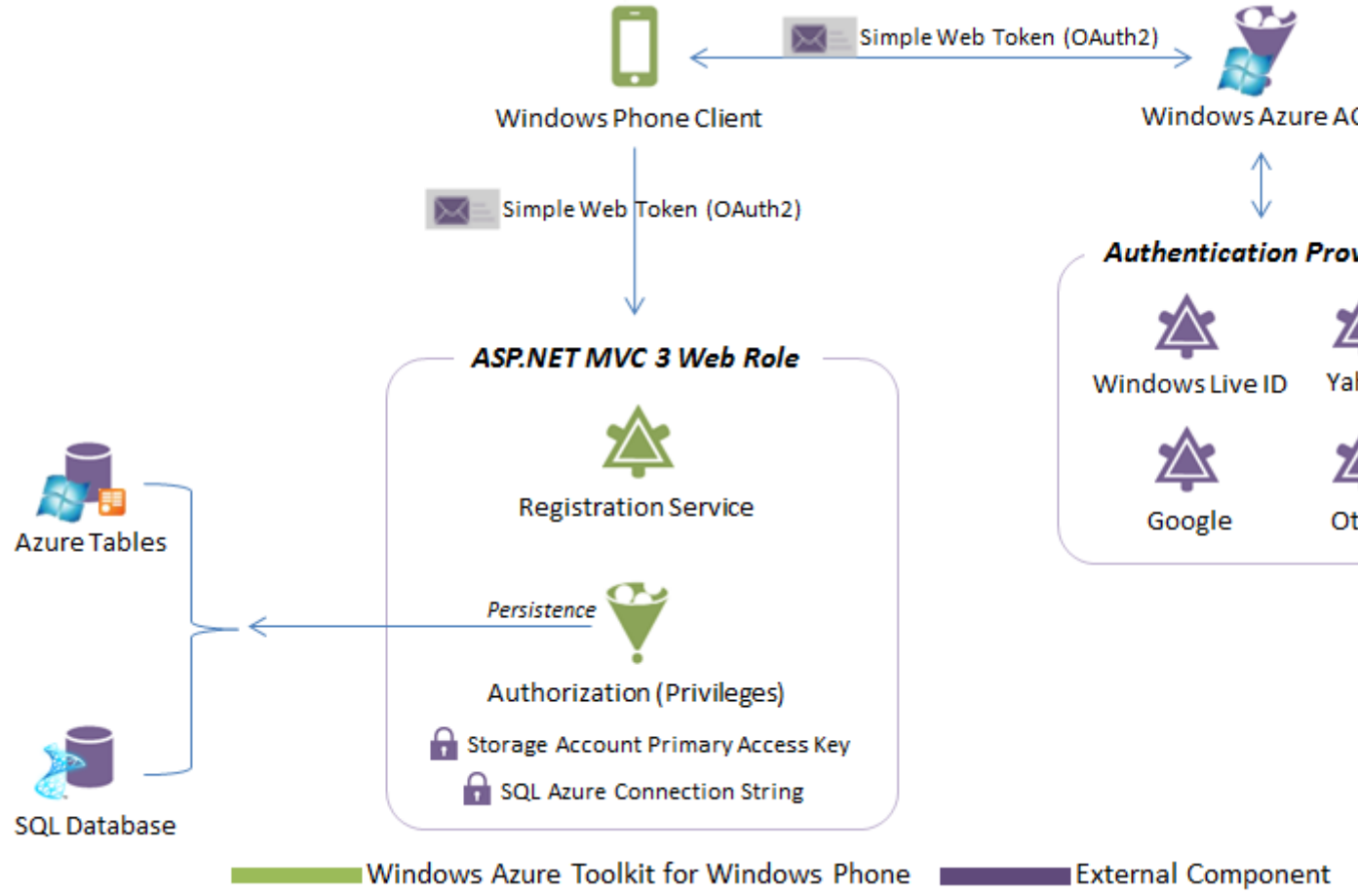
Authentification ACS

Je vous rappelle rapidement le principe :

Windows Azure Access Control Service (ACS) permet à vos applications d'externaliser l'authentification des utilisateurs, ce qui permet de s'inscrire et se connecter en réutilisant les comptes déjà existants chez des fournisseurs d'identité tels que Windows Live ID, Yahoo, Google, Facebook, et même Active Directory.

Plus en détails, l'application souhaitant utiliser ACS récupère la liste des fournisseurs d'identité disponibles auprès de l'espace de nom configuré ACS (nous allons voir ensuite comment créer un tel espace de nom). L'utilisateur peut donc sélectionner le type de compte qu'il souhaite utiliser pour s'identifier. Ensuite, le token fourni en retour (si l'identification se passe bien) est utilisé pour accéder aux services de stockages fournis par Windows Azure via le proxy WAT.

Voici un diagramme qui explique ce fonctionnement.

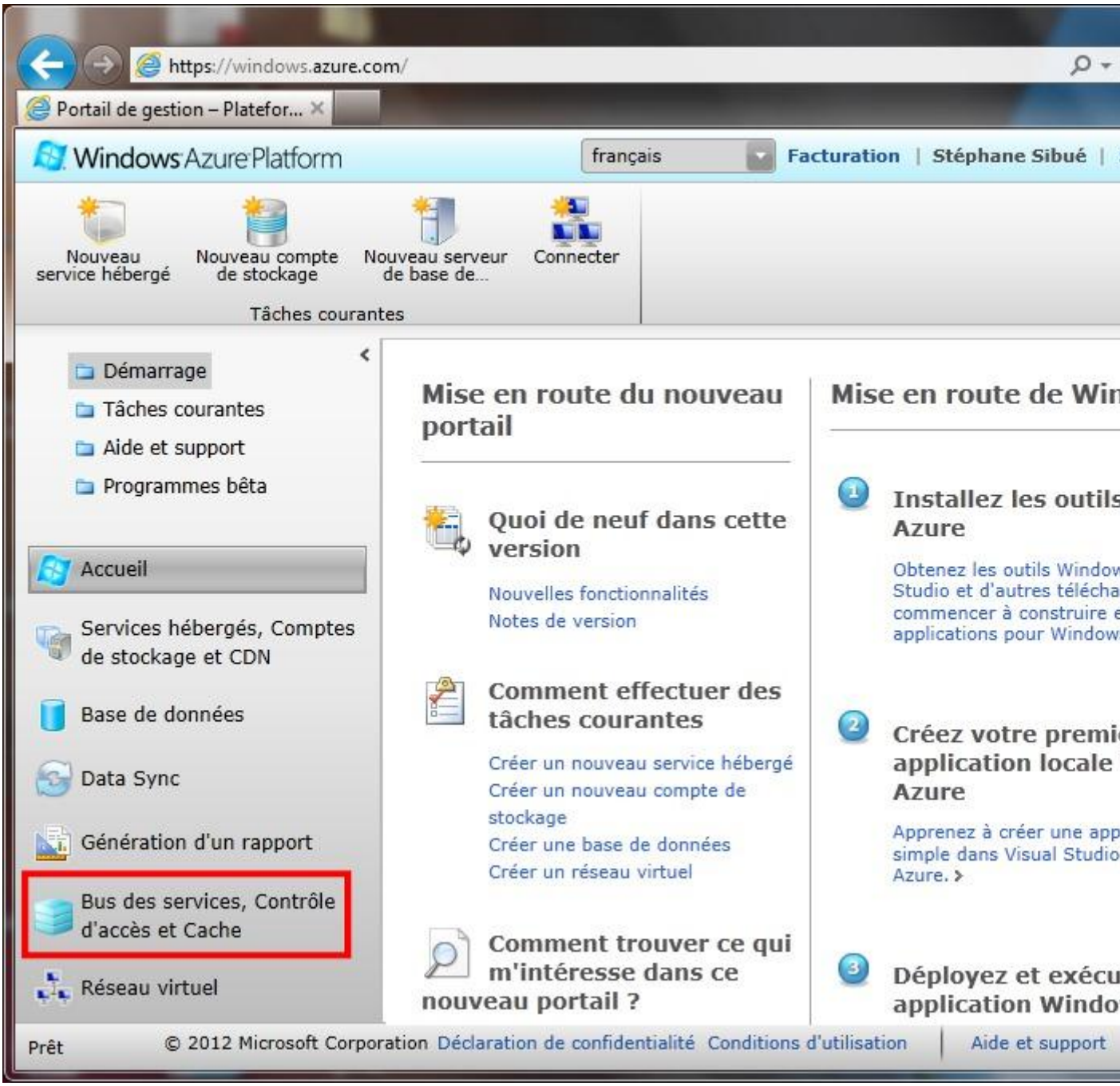


Comment créer un espace de noms ACS ?

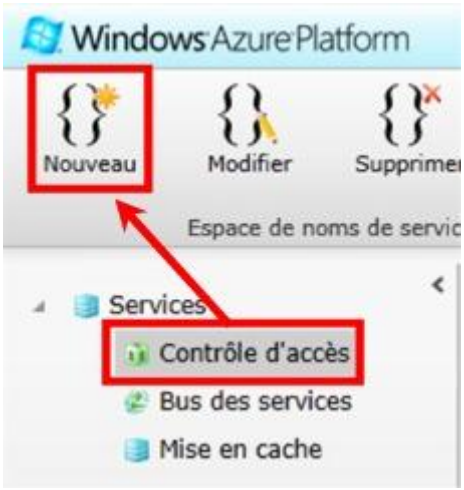
Vous devez disposer d'un compte Windows Azure pour profiter des services de l'ACS. Si vous n'en avez pas vous pouvez profiter de certaines offres d'essai pour découvrir Windows Azure et en profiter pour créer votre nom de domaine ACS.

Vous devez vous rendre sur http://windows.azure.com pour accéder au portail de gestion Windows Azure. Connectez-vous avec votre identifiant Windows Live ID (tiens, une bon exemple d'utilisation d'un fournisseur d'identité).

Une fois connecté vous devez sélectionner l'option « Bus des services, Contrôle d'accès et Cache »



Puis vous devez sélectionner l'option « Contrôle d'accès » et enfin « Nouveau »

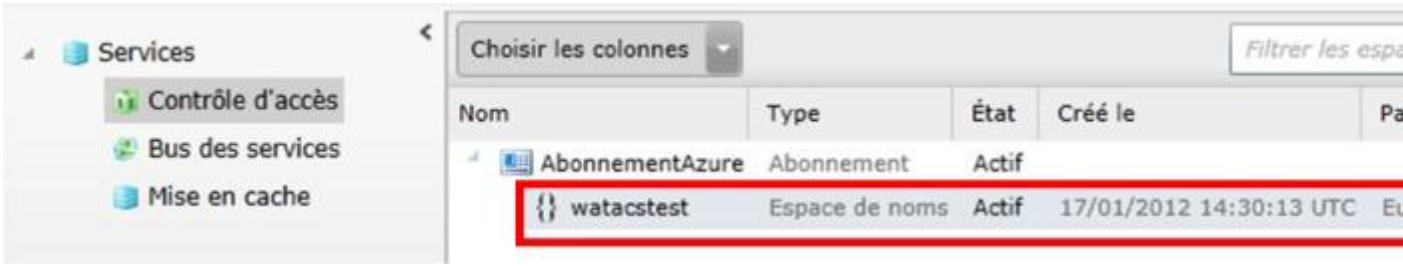


Là vous devez donner plusieurs informations :

Information	Valeur
Espace de noms	watacstest
Pays/Region	Europe (Nord)

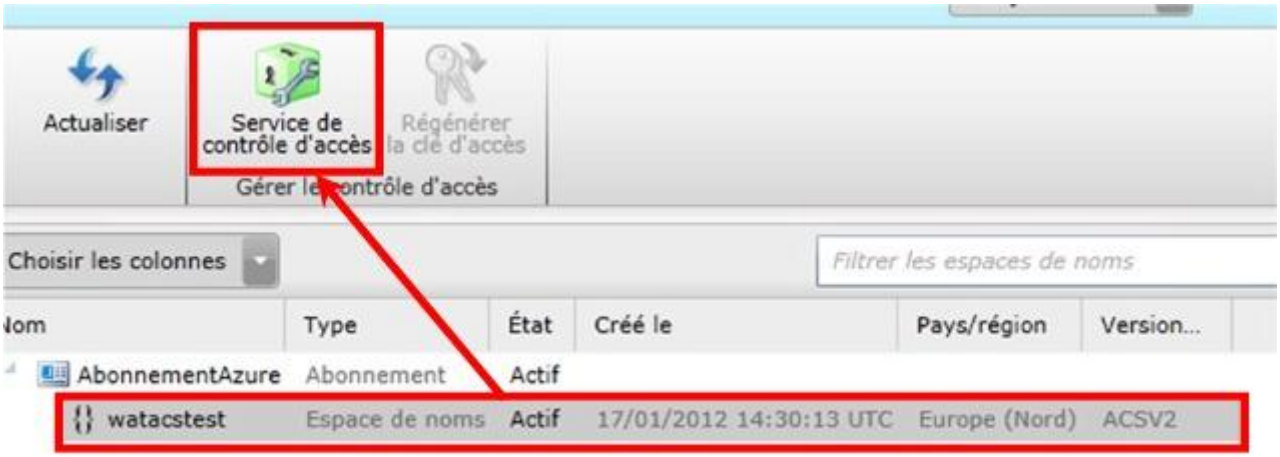
Nous allons ainsi créer l'espace de noms ACS « watacstest » que nous pourrons utiliser pour effectuer les identifications ACS depuis nos applications mobile.

Cliquez maintenant sur « Créer un espace de noms ». Un fois créé, le nouvel espace de nom apparaît dans la liste :

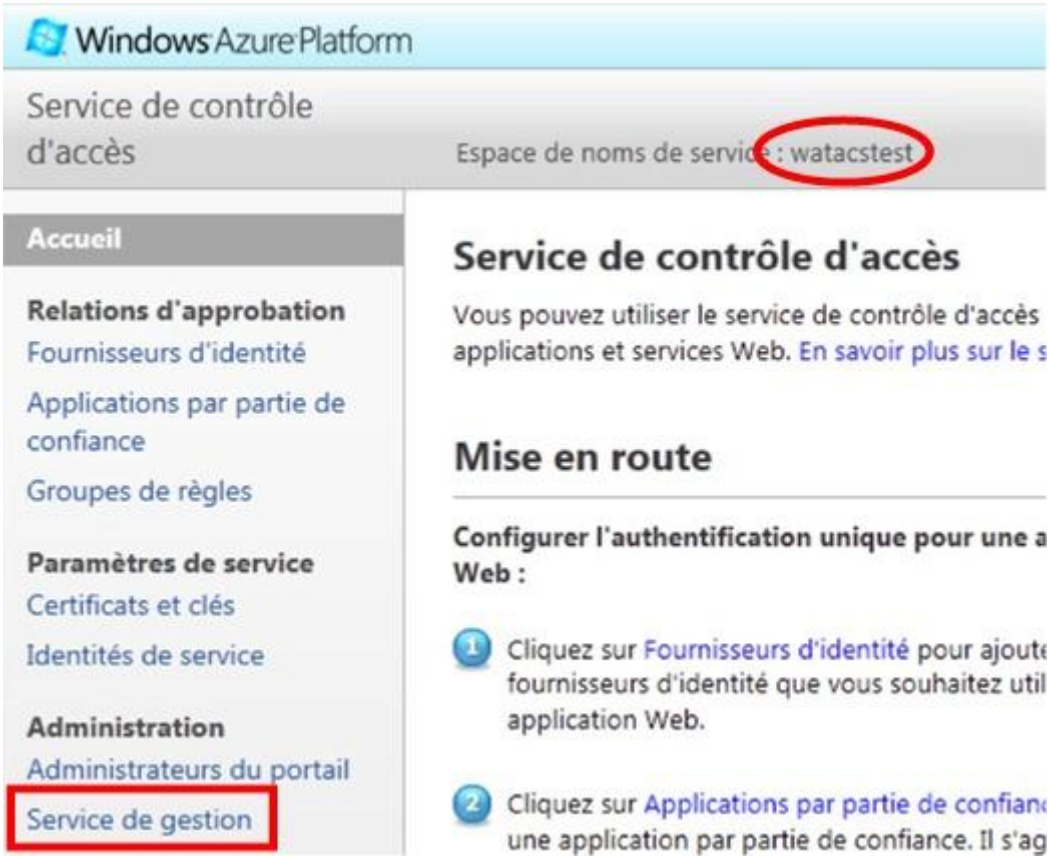


Nous avons fait la moitié du chemin. Nous devons maintenant récupérer la clé de gestion pour cet espace de noms car nous devrons la fournir au moment de créer le projet avec identification ACS.

Sélectionnez l'espace de noms que vous venez de créer et cliquez ensuite sur le bouton «Service de contrôle d'accès »



Vous allez vous retrouver sur la page de gestion du contrôle d'accès pour l'espace de noms que vous venez de créer. Il vous suffit maintenant de sélectionner « Service de gestion »



Sur la page suivante sélectionnez « Management Client »

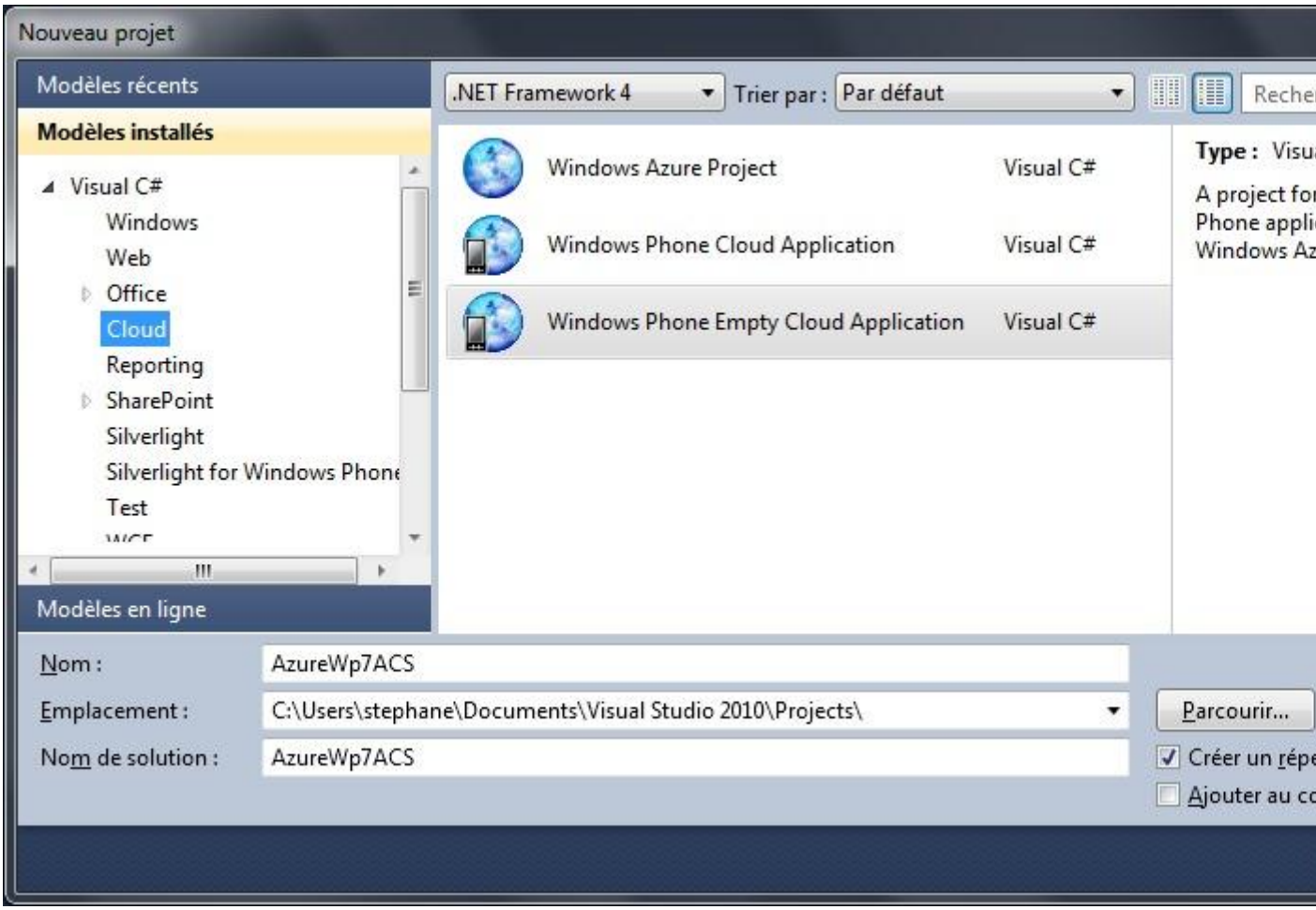
Puis sur « Clé symétrique »

Nous arrivons alors sur la page qui contient la fameuse clé dont nous avons besoin pour continuer notre aventure. La clé se trouve là. Vous devez la copier temporairement dans le bloc-notes par exemple.

Nous avons maintenant tout ce qu'il faut pour créer notre projet WAT avec identification ACS.

Création du projet

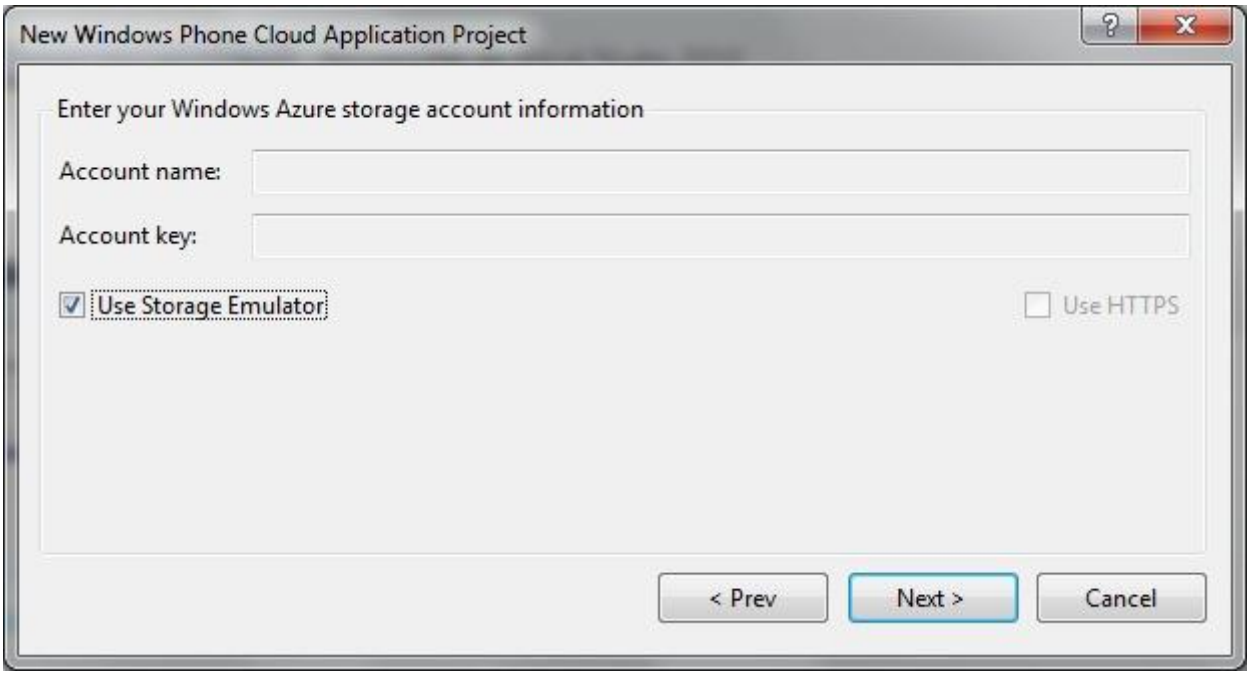
Le Template « Windows Phone Empty Cloud Application » se trouve dans la catégorie « Cloud ».
Nous allons lui donner comme nom « AzureWp7ACS » :



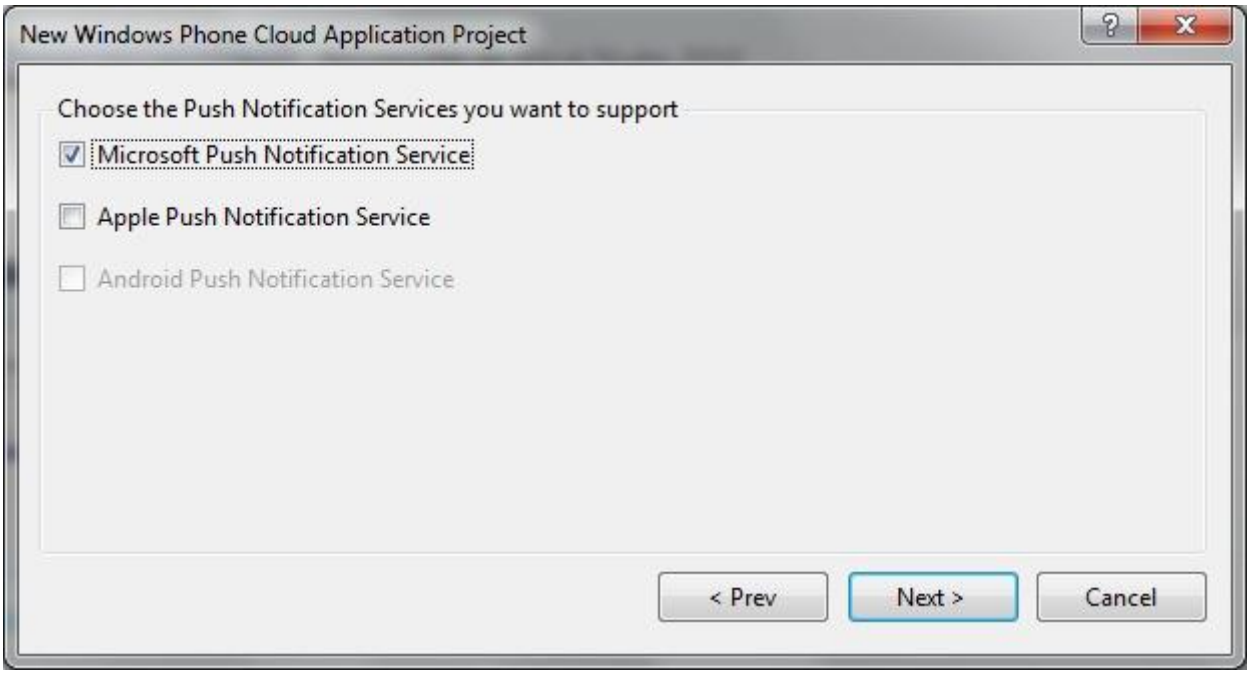
Pour créer un projet correspondant à vos besoins, Visual Studio va ensuite vous poser une série de questions :

Vous allez devoir indiquer quel type de stockage vous souhaitez utiliser sur Windows Azure. Pour notre exemple nous allons utiliser « Windows Azure Storage ».

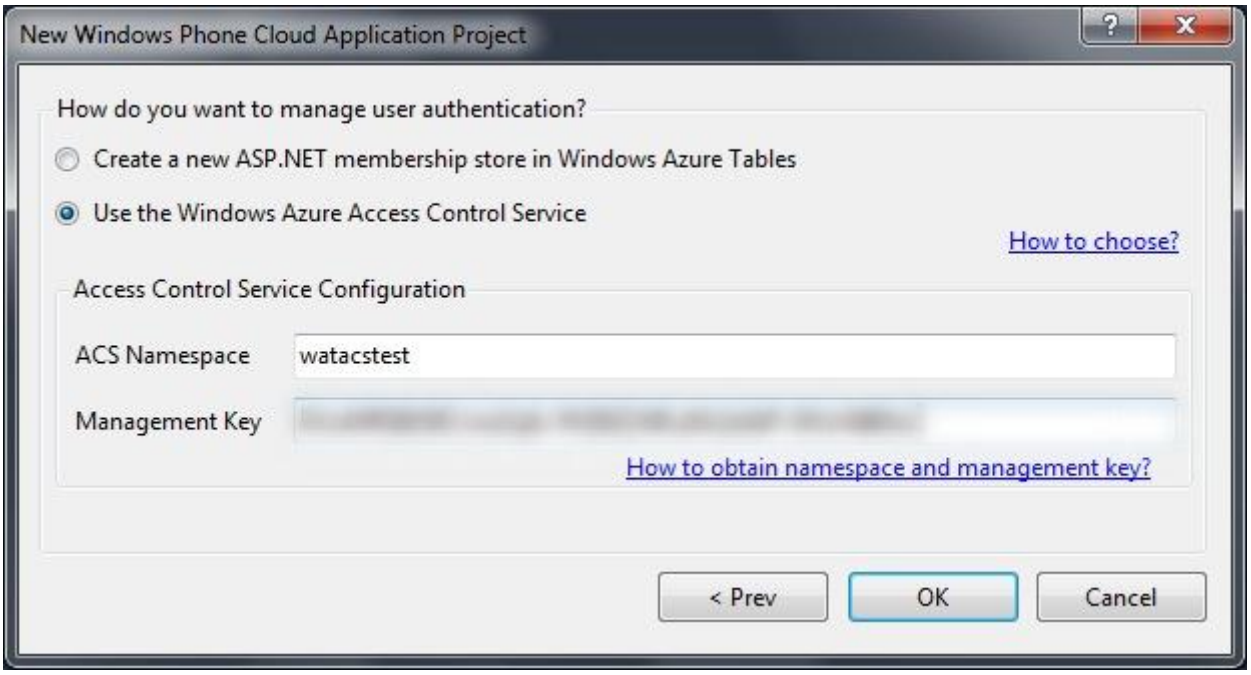
Puis vous allez devoir fournir vos identifiants Windows Azure dans le cas où vous souhaitez tester directement sur Windows Azure. Vous pouvez aussi décider d'utiliser les services de l'émulateur de stockage « Storage Emulator » pour effectuer tous vos tests sur votre propre machine de développement. C'est le choix que nous allons sélectionner pour nos tests.



Vient le tour des notifications. Par défaut Microsoft Push Notification Service est sélectionné. Vous pouvez aussi activer les notifications Apple si besoin. Les notifications Android ne sont pas accessibles pour le moment. Pour notre découverte, nous allons activer les notifications Microsoft seulement (les notifications Apple seront vues plus tard dans un tutoriel consacré aux notifications pour iOS).



Il reste maintenant à indiquer quel type de gestion des utilisateurs vous souhaitez utiliser. Nous allons sélectionner « ACS » et taper l'espace de noms ACS et la clé de gestion que nous venons de récupérer.

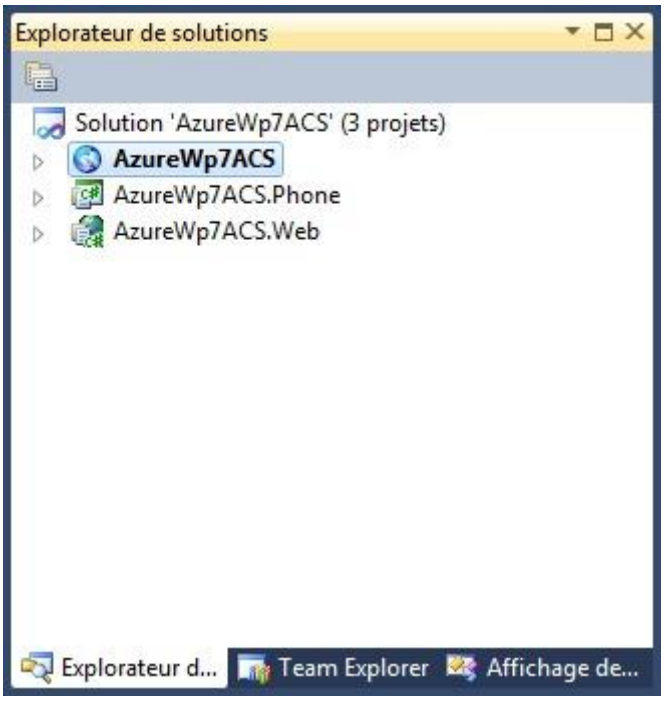


Après avoir répondu à toutes ces questions, Visual Studio prépare la solution

Elle est composée de trois projets :

AzureWp7ACS	Projet Windows Azure
AzureWp7ACS.Phone	Projet Windows Phone (votre application)

AzureWp7ACS.Web	Projet ASP.NET (le proxy WAT)



Il est important de noter que par défaut le projet web est configuré pour utiliser le port 10080 en http et 443 en https. Comme nous allons l'utiliser en mode émulation, vous devez vous assurer que votre IIS local ne fait déjà rien tourner sur ces ports.

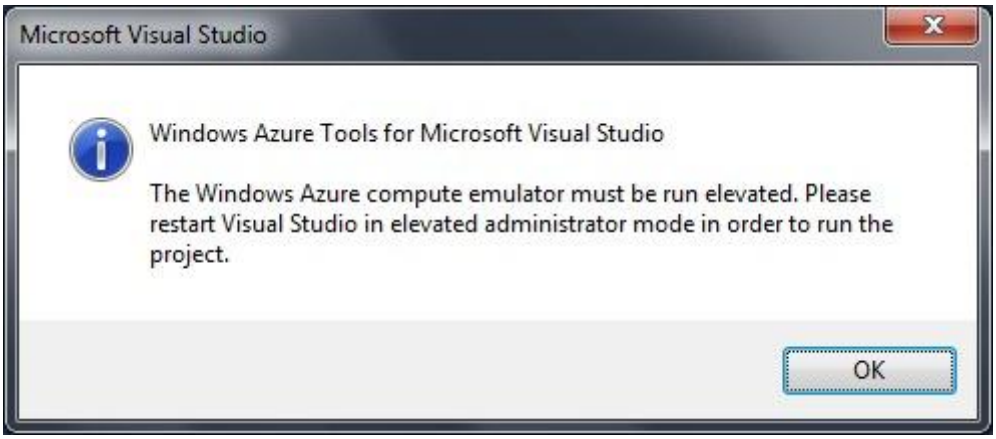
Pour que toute la chaîne fonctionne, il faut créer et installer des certificats sur Windows Phone. Pour simplifier les choses, nous allons utiliser l'émulateur Windows Phone avec un certificat préparé pour l'occasion par WAT. Il est fait pour ne fonctionner que sur votre machine locale et ne pourra pas être utilisé pour un déploiement réel sur Windows Azure. Comme l'émulateur « oublie » tout dès qu'il est arrêté, il faudra à chaque fois installer ce certificat. Mais rassurez-vous cette opération est très simple à mettre en œuvre.

Lancement du proxy WAT

Nous allons maintenant lancer le proxy WAT. La méthode est très simple, il suffit de lancer le projet Windows Azure. Faites un clic droit sur le projet Windows Azure, choisissez « Déboguer » puis « Démarrer une nouvelle instance » (si vous faites F5 directement, vous allez aussi lancer l'émulateur Windows Phone, donc une chose à la fois ça ira déjà bien ☺).

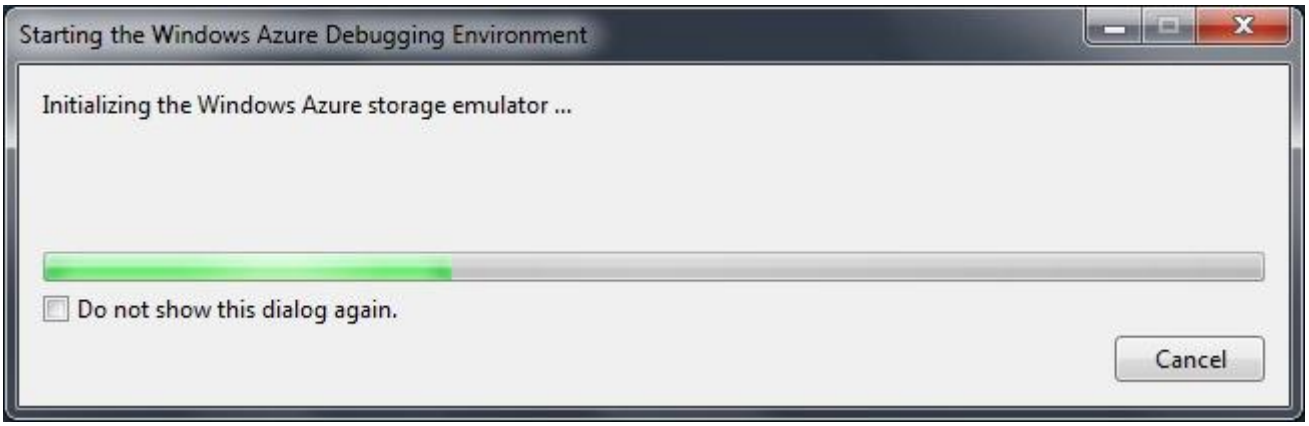
Après quelques instants de compilation, l'émulateur de stockage Windows Azure se lance.

Si ce message d'erreur s'affiche :

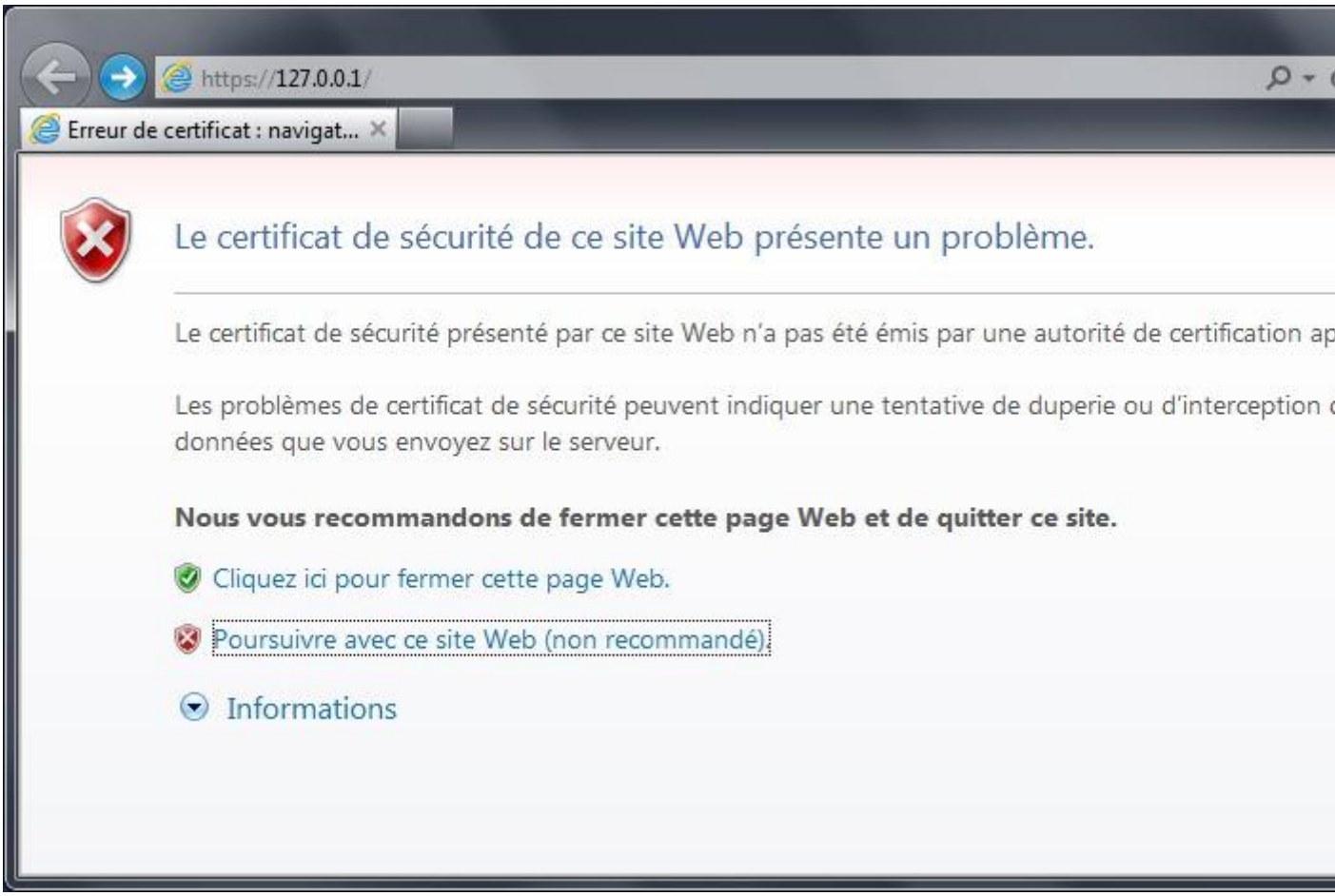


C'est que vous n'avez pas lancé Visual Studio avec les droits d'administrateur. Sans cette élévation des droits, toute la partie émulation de Windows Azure ne peut fonctionner. Dans ce cas, fermez tout, relancez Visual Studio en mode administrateur, ré-ouvrez le projet que nous venons de créer et relancer comme expliqué un peu plus haut.

Si tout se passe bien vous allez voir cette fenêtre apparaitre pendant l'initialisation du « Windows Azure Storage Emulator »

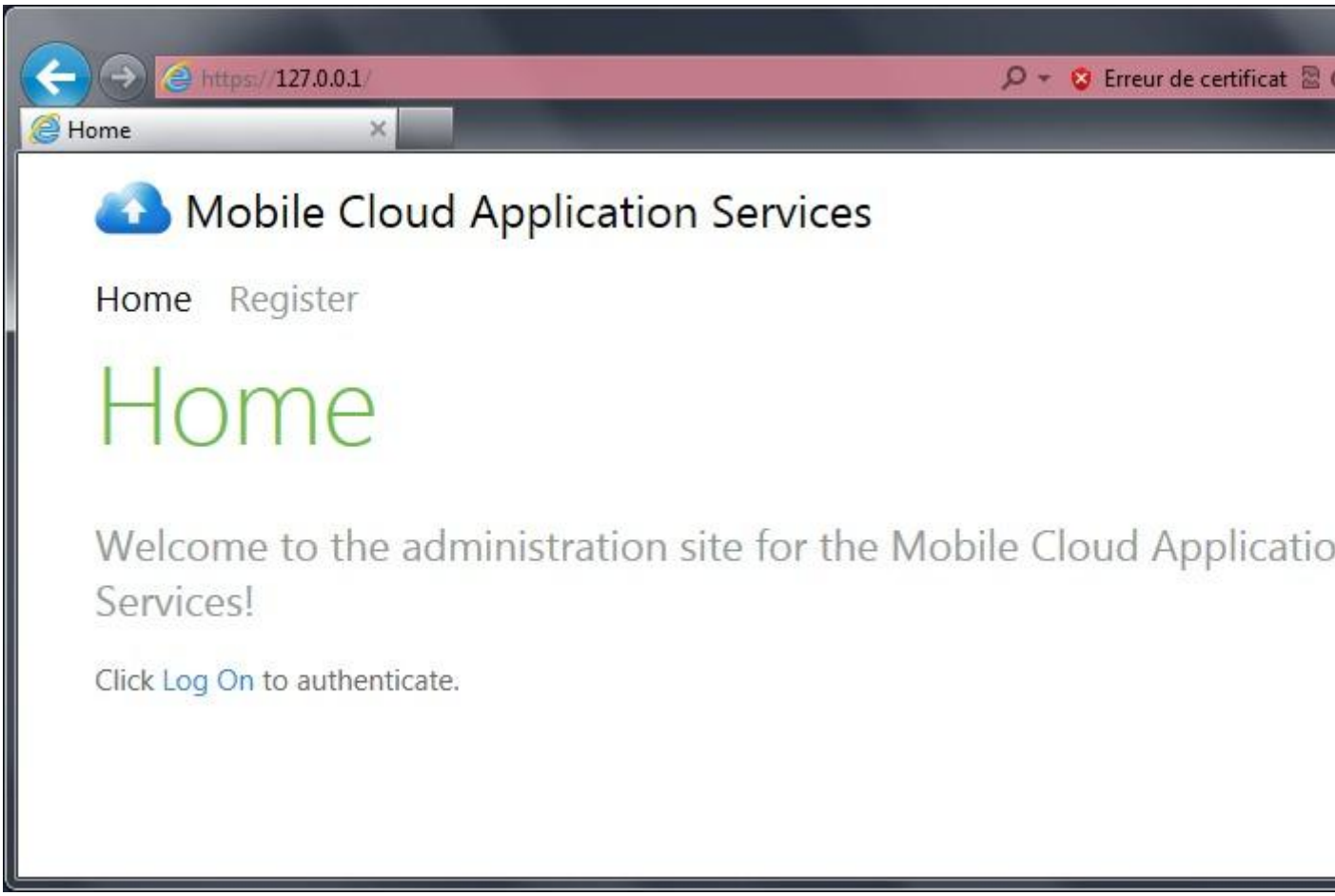


Il faut être patient et laisser faire. Au bout d'un petit moment vous allez voir apparaître dans votre browser Internet cette page :

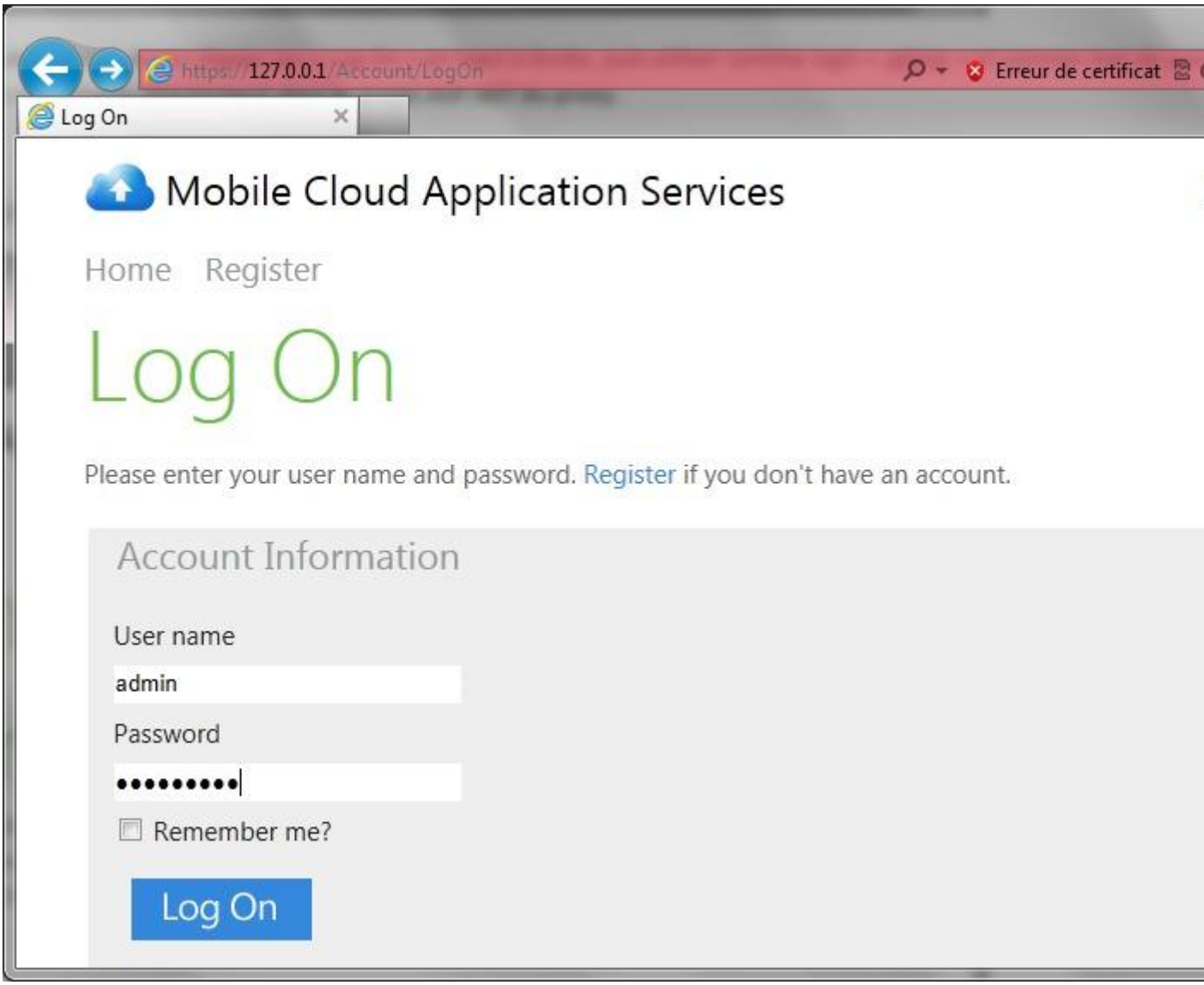


Comme nous utilisons un certificat auto signé cet avertissement est normal. Il suffit d'accepter de poursuivre avec ce site web.

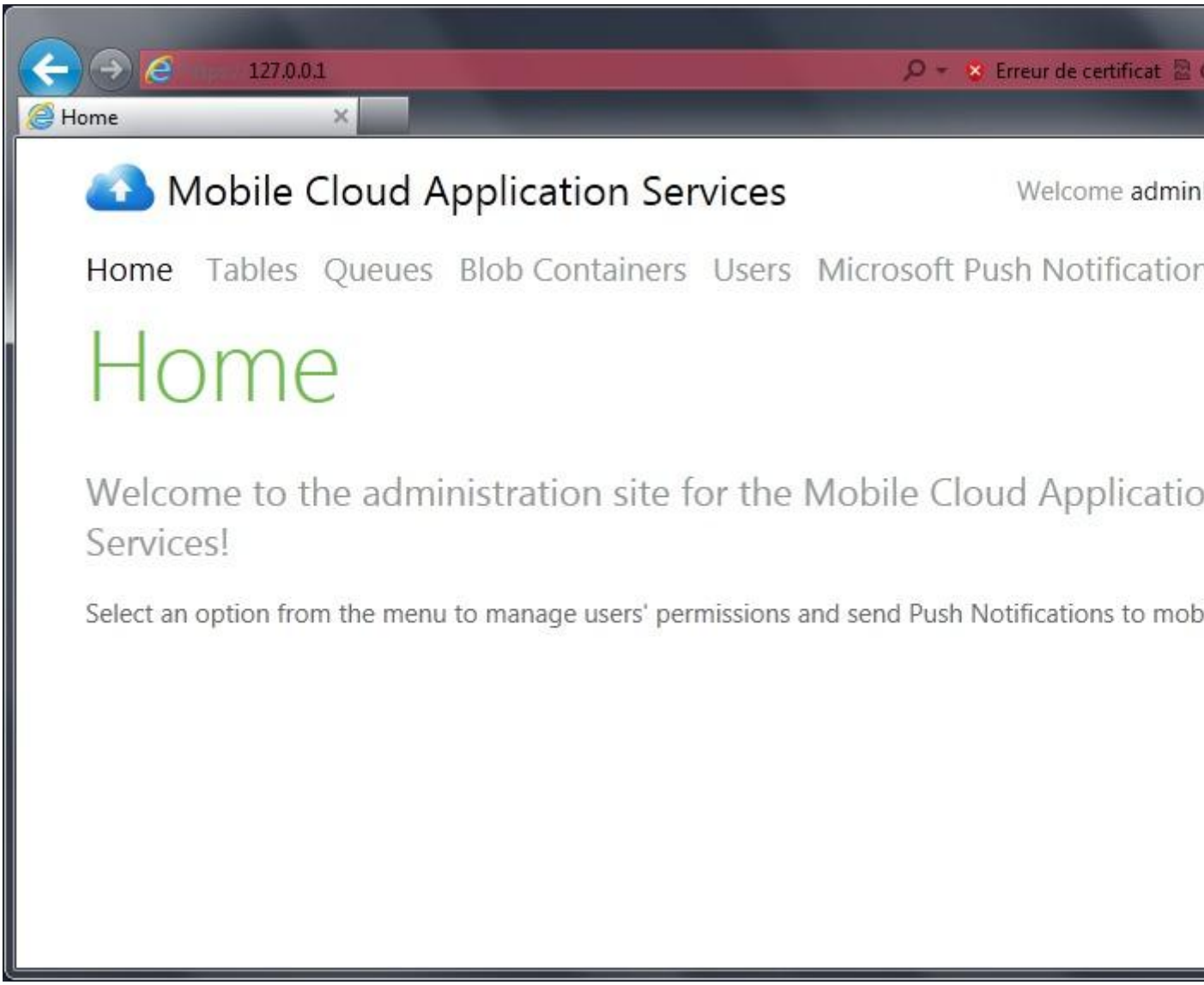
Finalement, on obtient la page principale du proxy WAT (home) :



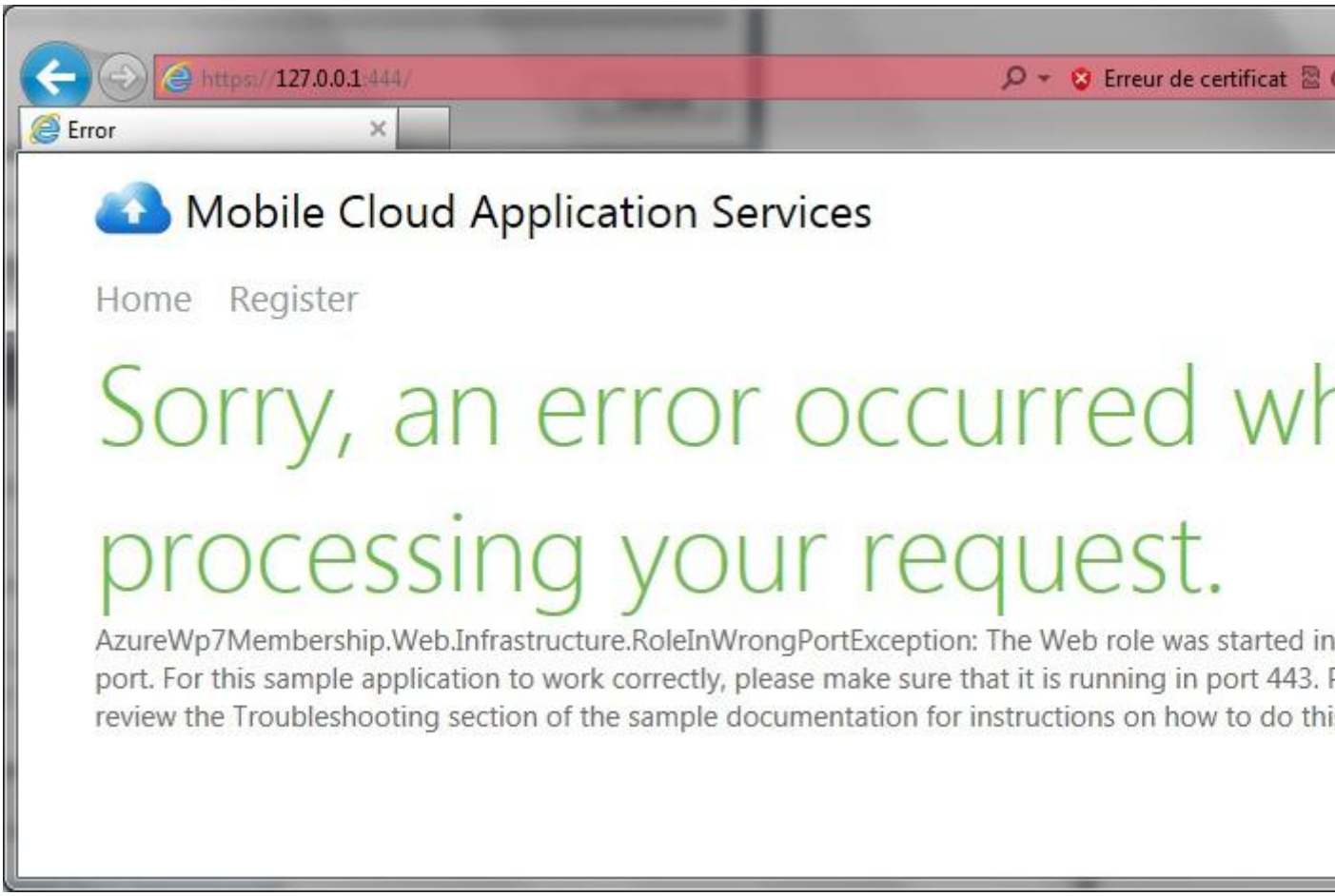
Pour accéder aux fonctions de l'administrateur, vous devez cliquer sur « Log On », puis utiliser comme login « admin » et comme mot de passe « Passw0rd ! » (avec un zéro à la place du O). Il est possible (et bien sûr hautement recommandé) de modifier ces identifiants directement dans le projet ASP.NET du proxy le jour du déploiement vers Windows Azure.



Une fois entré, vous obtenez cette page :



Si à la place de la page home du proxy vous obtenez celle-ci :



C'est que le port 443 n'est pas disponible sur votre machine. Dans ce cas le proxy se lance sur un autre port (typiquement le 444) et plus rien ne fonctionne.

En ce qui me concerne, j'ai eu ce problème avec Skype qui tournait en tâche fond. Une fois Skype arrêté tout est rentré dans l'ordre.

Vous pouvez maintenant stopper l'exécution du proxy car nous allons créer notre projet Windows Phone.

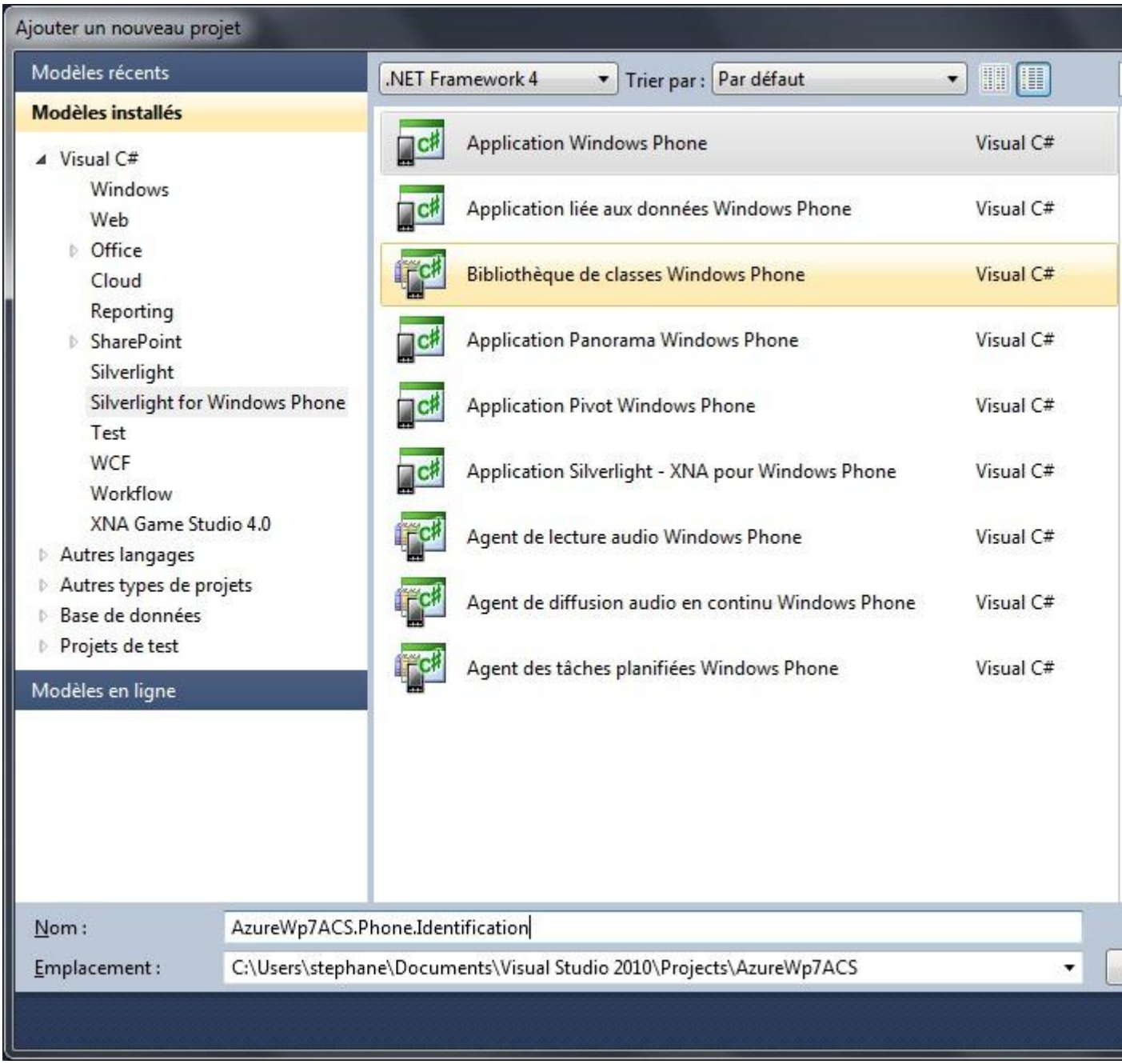
Création du projet Windows Phone

Lorsque nous avons créé notre nouveau projet avec le Template « Windows Phone Empty Cloud Application », Visual Studio nous a automatiquement créé 3 projets, dont un de type Windows Phone. Ce projet est entièrement paramétré pour utiliser les services de WAT sans se soucier de rien (ou presque).

Plutôt que d'utiliser ce projet pré-paramétré, nous allons en créer un nouveau et le paramétrer nous-même comme des grands !

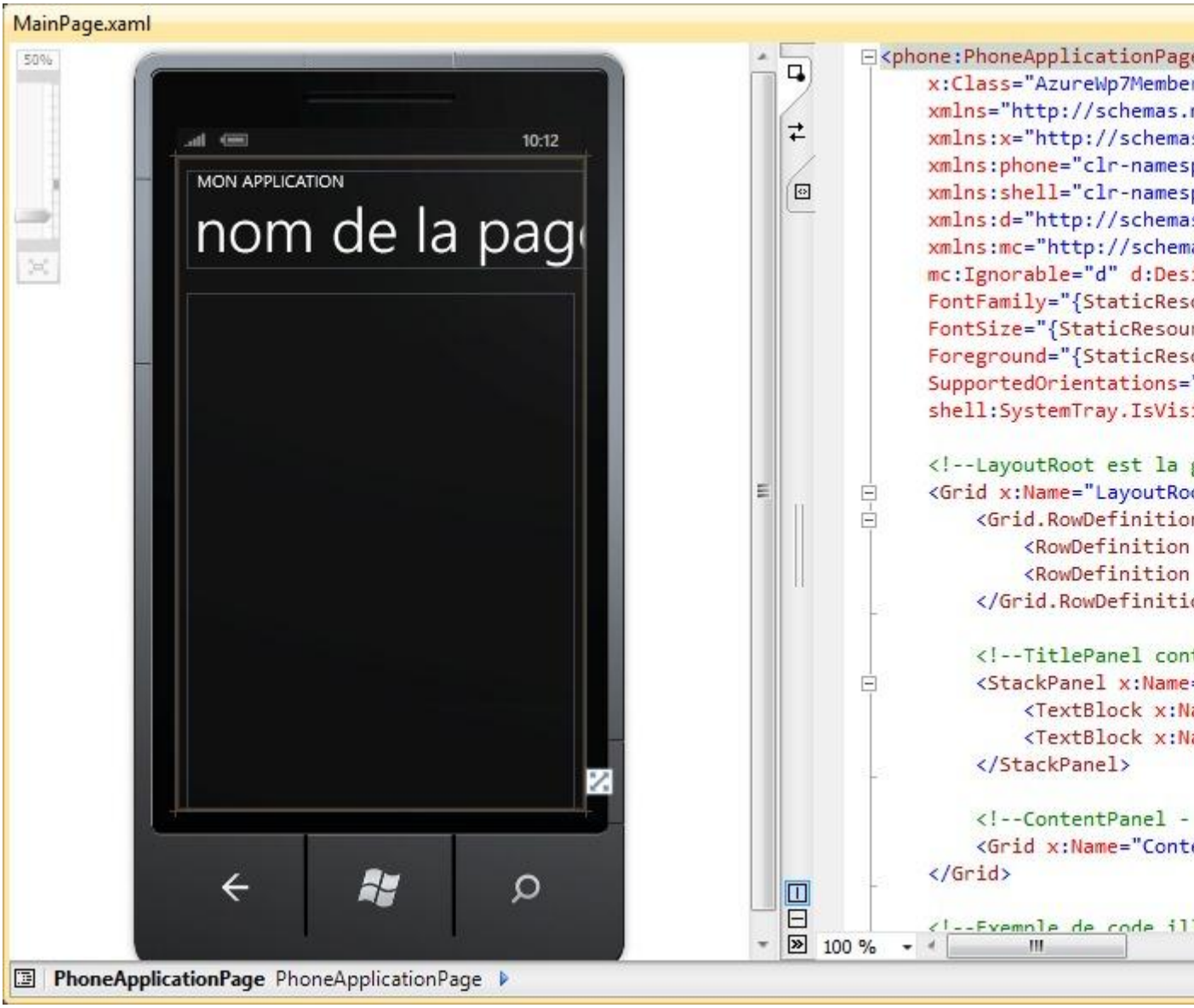
La logique d'enregistrement est un peu différente entre ASP.NET Membership et ACS. Avec ACS l'enregistrement se passe un peu comme un login qui fonctionne au niveau du fournisseur d'identité mais qui échoue au niveau du proxy WAT (normal l'utilisateur n'y est pas encore référencé).

Le mieux est de faire les choses dans l'ordre, donc commençons par ajouter à notre solution notre nouveau projet « AzureWp7ACS.Phone.Identification » de type « Silverlight pour Windows Phone » :



Nous allons ensuite sélectionner comme cible le SDK « Windows Phone OS 7.1 » (qui correspond à la version 7.5 du système d'exploitation) :

Après quelques courts instants, nous nous retrouvons avec un projet classique Windows Phone possédant une page principale « MainPage » :



Nous devons maintenant ajouter deux nouvelles références à notre projet. Ces assemblies se trouvent dans le dossier « Libs » de la solution et font partie intégrante de WAT :

1. WindowsPhoneCloud.StorageClient.dll
2. SL.Phone.Federation.dll

WAT nous mâche complètement le travail pour la gestion via ACS. En effet, c'est lui qui va se charger des tâches suivantes :

1. Lister les fournisseurs d'identités disponibles et permettre à l'utilisateur d'en sélectionner un
2. Contacter le fournisseur d'identité afin qu'il affiche son panneau d'identification
3. Affichage du panneau d'identification du fournisseur
4. Gérer tous les échanges avec le fournisseur d'identité (dans la pratique ce n'est pas simple du tout)

Toutes ces tâches sont effectuées par un contrôle utilisateur fourni par WAT qui se trouve dans l'assembly « SL.Phone.Federation.dll » dont nous avons ajouté la référence. Le nom de ce contrôle est « AccessControlServiceSignIn »

Pour résumer, voici ce que nous allons devoir faire pour mettre en place l'identification ASC :

1. Créer une page de login (simple page Windows Phone)
2. Ajouter le contrôle « AccessControlServiceSignIn »
3. Le paramétrer
4. S'abonner à son événement « RequestSecurityTokenResponseCompleted » qui retourne le verdict du fournisseur d'identité
5. Invoquer sa méthode « GetSecurityToken » pour lancer l'identification
6. Répondre à l'événement « RequestSecurityTokenResponseCompleted »
7. Vérifier que l'utilisateur est déjà enregistré dans le proxy WAT
8. Si pas encore enregistré, procéder à son enregistrement
9. Récupérer le token de session pour la suite des opérations sur Windows Azure

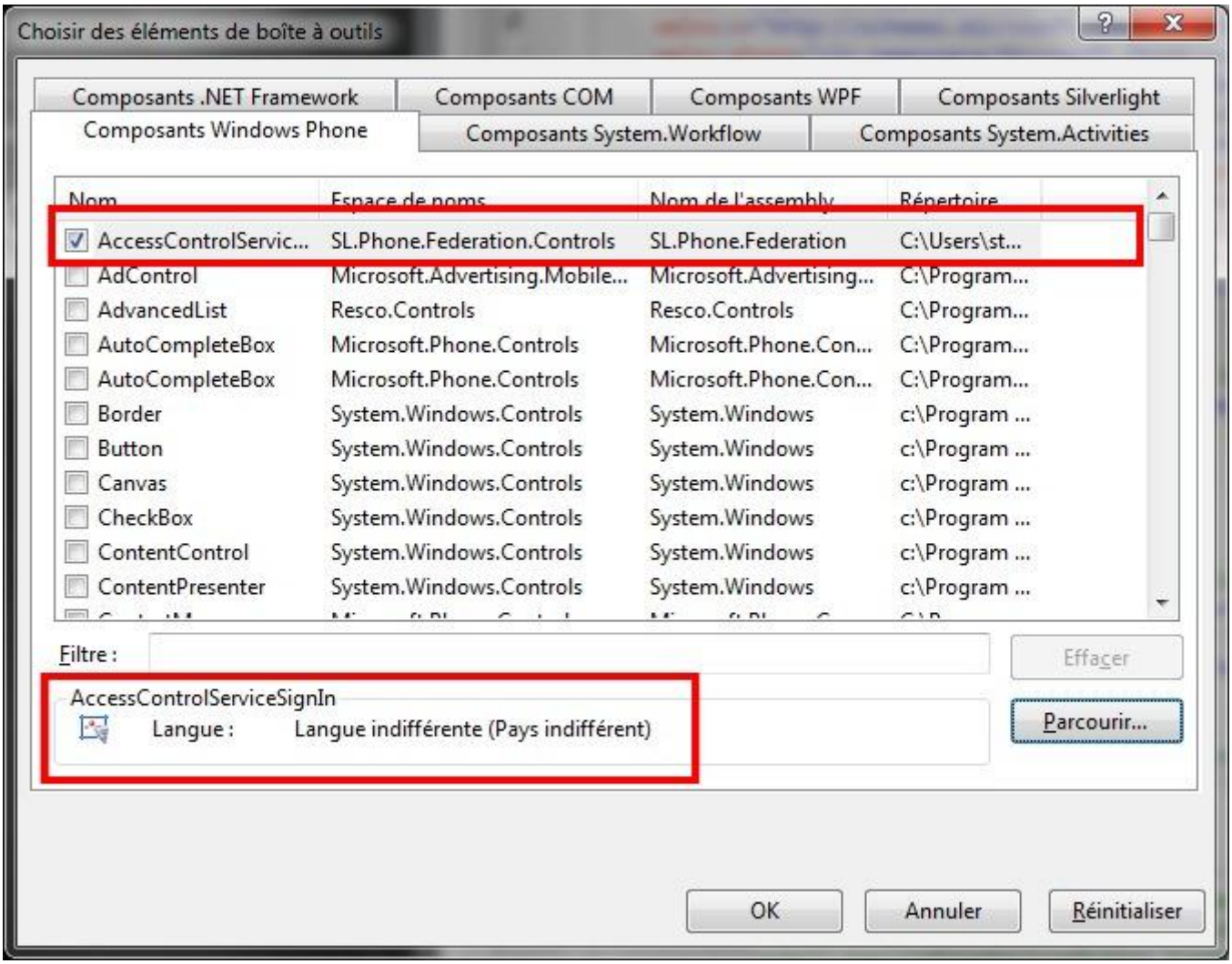
Et c'est tout. Simple vous ne trouvez pas ?

Ajouter le contrôle AccessControlServiceSignIn

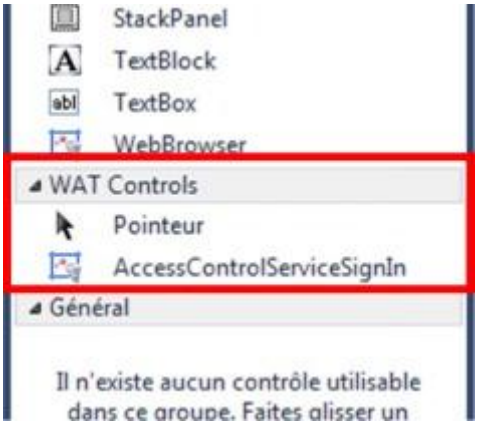
A mon sens, le plus simple est d'ajouter ce contrôle à la boîte à outils de Visual Studio puis de l'ajouter à notre page. Une autre manière serait de l'ajouter directement en tapant le code XAML… Perso j'ajoute le contrôle à la boîte à outils car c'est très simple à réaliser :

On crée tout d'abord un nouvel onglet dans la boîte à outils qu'on va appeler « WAT Controls » (un clic droit dans la boîte à outils puis « Ajouter un onglet »).

Ensuite dans cet onglet, un autre clic droit et « Choisir les éléments ». Là il faut faire un coup de « Parcourir » et sélectionner le fichier « SL.Phone.Federation.dll » qui se trouve dans le dossier « Libs » de la solution.



Avec cette petite opération, la boîte à outils héberge maintenant le contrôle dont nous avons besoin :



Il suffit maintenant de placer ce contrôle sur la « MainPage » de notre projet, la place n'a pas d'importance car nous n'allons pas nous attarder sur le côté esthétique de notre page d'identification.
Vous remarquez qu'une fois ajouté à la page, le contrôle affiche en permanence une barre de progression. C'est normal. On va en profiter aussi pour donner comme titre à notre page « ACS » et donner au contrôle que nous venons d'ajouter le nom « SignInControl »

Par défaut dans un projet Windows Phone 7, toutes les autorisations sont validées. Mais au moment de publier votre chef d'œuvre, vous devrez enlever celles qui ne servent à rien. Comme le contrôle « AccessControlServiceSignIn » utilise un web browser control, il faut donc le déclarer au niveau du manifest de l'application « WMAppManifest.xml » :

```
<Capability Name="ID_CAP_WEBBROWSERCOMPONENT"/>
```

Taper un peu de code

Si l'opération d'identification réussit, le control d'accès nous retourne un token emballé dans un objet de type « RequestSecurityTokenResponseStore » qui se trouve dans l'espace de nom « SL.Phone.Federation.Utilities ».

Pour simplifier l'écriture, nous allons ajouter deux directives using à notre code :

```
using SL.Phone.Federation.Utilities;
using SL.Phone.Federation.Controls;
```

Nous allons déclarer une variable privée dans laquelle nous récupérerons le fameux emballage à token :

```
private RequestSecurityTokenResponseStore pTokenResponseStore = new
RequestSecurityTokenResponseStore();
```

Maintenant nous pouvons paramétrer notre contrôle d'accès. Nous devons lui fournir 3 informations pour qu'il fonctionne correctement :

Propriété	Usage
Realm	Le nom de l'application demandeuse (doit commencer par les lettres uri :)
ServiceNamespace	Le nom de votre domaine ACS
RequestSecurityTokenResponseStore	L'objet qui recevra le token d'accès

Vous pouvez paramétrer le contrôle depuis le code XAML ou via le code behind C#. Nous allons passer par le C# et placer le code d'initialisation dans le constructeur de notre page. Il faut aussi s'abonner à l'événement « RequestSeurityTokenResponseCompleted » et créer la méthode de rappel liée.

Lorsque le contrôle a terminé ses petites affaires, que tout soit ok ou que tout aie foiré lamentablement, il déclenche la méthode de rappel liée à l'événement « RequestSecurityTokenResponseCompleted ». Le paramètre « e » de cet événement porte soit le token tant convoité (tout s'est bien passé), soit des informations sur le problème rencontré. Si tout s'est bien passé, le token a été automatiquement stocké dans la variable « pTokenResponseStore » que nous avons utilisée pour le paramétrage du contrôle.

Maintenant que le décor est complètement planté, il ne reste plus qu'à demander à SingInControl de démarrer son job en invoquant sa méthode « GetSecurityToken ». Dans notre exemple nous allons appeler cette méthode depuis « OnNavigatedTo ».

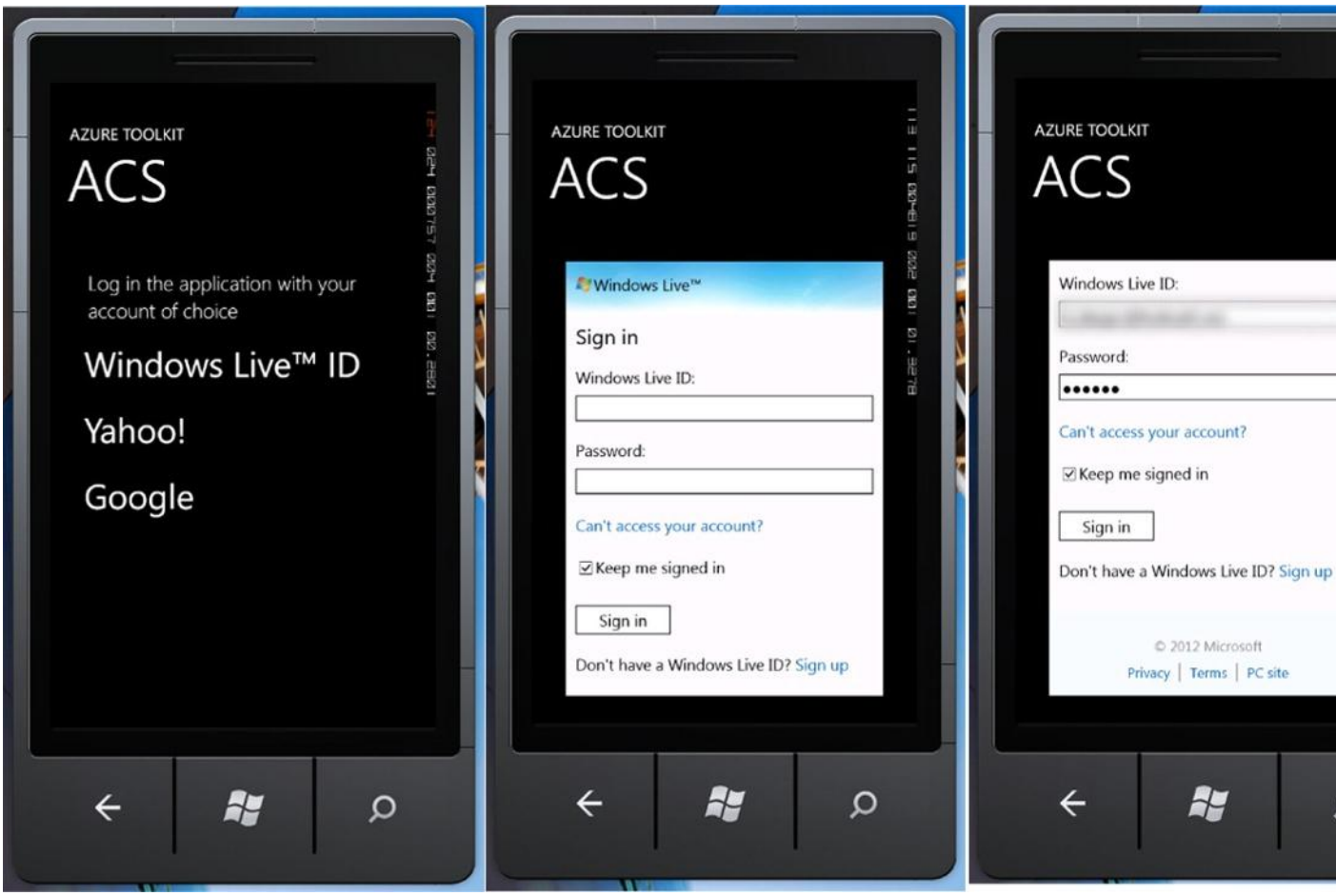
Voici le code complet de l'exemple :

```
private RequestSecurityTokenResponseStore pTokenResponseStore = new
RequestSecurityTokenResponseStore();

public MainPage()
{
    InitializeComponent();

    SignInControl.Realm = "uri:azurewp7acs";
    SignInControl.ServiceNamespace = "watacstest";
    SignInControl.RequestSecurityTokenResponseStore = pTokenResponseStore;

    SignInControl.RequestSecurityTokenResponseCompleted += new
EventHandler<RequestSecurityTokenResponseCompletedEventArgs>(OnRequestSecurityTokenRes
ponseCompleted);
}

void OnRequestSecurityTokenResponseCompleted(object sender,
RequestSecurityTokenResponseCompletedEventArgs e)
{
    if (e.Error != null)
    {
        // Problème
    }
    else
    {
        // Tout va bien
        // Le token est maintenant disponible dans pTokenResponseStore
    }
}
```

```
protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    SignInControl.GetSecurityToken();
}
```

On lance pour vérifier que tout se passe bien

Maintenant nous pouvons lancer notre application de test. Elle commence par nous lister les fournisseurs d'identité qu'elle prend en charge. Moi j'ai choisi « Windows Live ID ». Ensuite, c'est le panneau d'identification du fournisseur d'identité qui s'affiche, car c'est lui qui prend à ce moment-là le relai. S'il valide votre identité, le token de sécurité sera fourni à l'application et l'événement « RequestSecurityTokenResponseCompleted » indiquera que tout va bien (pas d'erreur), sinon l'erreur rencontrée sera remontée via la propriété « Error » du paramètre « e ».



On dialogue avec le proxy WAT

Maintenant que nous avons mis en place toute l'architecture pour profiter de l'ACS, il nous reste à vérifier auprès du proxy WAT que l'utilisateur identifié fait déjà partie ou non de la maison. En gros, déterminer s'il se connecte pour la première fois ou s'il est déjà connu des services de police ☺

Pour effectuer cette vérification nous allons utiliser les services de la classe « RegistrationClient » qui se trouve dans l'espace de nom

« Windows.Samples.WindowsPhoneCloud.StorageClient.Credentials ». Pour simplifier l'écriture du code nous allons en profiter pour ajouter la directive suivante :

```
using Microsoft.Samples.WindowsPhoneCloud.StorageClient.Credentials;
```

La classe « RegistrationClient » possède une méthode « CheckUserRegistration » qui vérifie auprès du proxy WAT si l'utilisateur fait partie ou non des utilisateurs déjà enregistrés. Lors de la création de l'objet de type « RegistrationClient » qui nous permettra de faire cette vérification, nous allons lui passer en paramètre l'adresse du service en charge de cette tâche dans le proxy WAT et les informations sur l'utilisateur.

L'adresse du service d'enregistrement se termine toujours par « /RegistrationService », le début étant conditionné par l'adresse réelle du proxy WAT. Par exemple pour un proxy installé sur Azure sous le nom de « monproxy.cloudapp.net » l'adresse sera « https://monproxy.cloudapp.net/RegistrationService ».

Les informations sur l'utilisateur nous sont fournies directement par le fournisseur d'identité, il va bien falloir faire avec car à ce stade on ne connait rien de plus sur l'utilisateur. Nous allons passer ces informations via un objet de type « StorageCredentialsSwtToken » dans lequel nous allons stocker le token de sécurité retourné par le fournisseur d'identité ainsi que l'expiration de celui-ci.

Nous allons aussi créer 3 méthodes de rappel pour les trois réponses possibles qui sont :

1. L'utilisateur fait bien partie de la maison (il est déjà enregistré)
2. L'utilisateur ne fait pas encore partie de la maison (il n'est pas encore enregistré)
3. Un problème s'est produit pendant la vérification

Voici le prototype de ces 3 callbacks :

```
void OnUserIsRegistered(RegistrationSuccessEventArgs e)
{
    // Le client est déjà enregistré dans le proxy WAT
}

void OnUserIsNotRegistered(RegistrationSuccessEventArgs e)
{
    // Le client n'est pas encore enregistré dans le proxy WAT
}

void OnRegistrationError(RegistrationExceptionEventArgs e)
{
    // Un problème est survenu pendant la vérification
    // Il faut remonter l'info à l'utilisateur
}
```

Puis la création de l'objet porteur des informations sur l'utilisateur :

```
long wExpire = 0;

if (pTokenResponseStore.SecurityToken != null)
{
    wExpire = pTokenResponseStore.RequestSecurityTokenResponse.expires;
}
```

```
var wStorageCredentials = new
StorageCredentialsSwtToken(pTokenResponseStore.SecurityToken, wExpire);
```

Nous allons stocker l'objet « RegistrationClient » dans une variable privée de la page, car nous allons par la suite le réutiliser à un autre endroit. Donc voici sa déclaration :

```
private RegistrationClient pRegistrationClient = null;
```

Et pour finir le lancement de la vérification en elle-même :

```
string wRegistrationService = "https://127.0.0.1/RegistrationService";

pRegistrationClient = new RegistrationClient(wRegistrationService,
wStorageCredentials);

pRegistrationClient.CheckUserRegistration(OnUserIsRegistered, OnUserIsNotRegistered,
OnRegistrationError);
```

Voilà, avec tout ça on peut maintenant savoir quel est le statut de l'utilisateur qui vient de s'identifier via ACS. Reste maintenant à réagir correctement au fait que l'utilisateur n'est pas encore enregistré dans le proxy WAT. C'est ce que nous allons voir maintenant.

Installation du certificat

Pour que toute la chaîne fonctionne, il faut créer et installer des certificats sur Windows Phone. Pour simplifier les choses, nous allons utiliser l'émulateur Windows Phone avec un certificat préparé pour l'occasion par WAT. Il est fait pour ne fonctionner que sur votre machine locale et ne pourra pas être utilisé pour un déploiement réel sur Windows Azure. Comme l'émulateur « oublie » tout dès qu'il est arrêté, il faudra à chaque fois installer ce certificat. Mais rassurez-vous cette opération est très simple à mettre en œuvre. Il vous suffit d'ajouter un bouton sur la page qui exécute le code suivant (ne pas oublier la directive using « Microsoft.Phone.Tasks ») :

```
var wTask = new WebBrowserTask();
wTask.Uri = new Uri("http://127.0.0.1:10080/127.0.0.1.cer", UriKind.Absolute);
wTask.Show();
```

Ensuite, au moment de choisir quel fournisseur d'identité vous allez utiliser, vous pourrez effectuer l'installation du certificat.

Enregistrement d'un nouvel utilisateur

Dans le cas où l'utilisateur ne fait pas encore partie des utilisateurs enregistrés, il faut l'enregistrer explicitement auprès du proxy WAT. Dans une application de la vraie vie, il conviendrait d'afficher une page permettant de saisir le nom et l'adresse email de ce nouvel utilisateur. Dans le cas de cette découverte nous allons enregistrer le nouvel utilisateur avec un nom et une adresse email codés en dur et ce pour simplifier les choses.

L'enregistrement à proprement parler s'effectue par l'intermédiaire du l'objet de type « RegistrationClient » que nous avons utilisé pour effectuer la vérification précédente. Cet objet est correctement paramétré et « connaît » l'utilisateur que nous voulons enregistrer.

Comme pour les autres fonctions vues précédemment, l'enregistrement est asynchrone, ce qui signifie que nous allons là encore devoir créer des méthodes de rappel au nombre de deux. L'une pour le cas où l'enregistrement se déroule normalement, l'autre pour le cas où l'enregistrement n'est pas possible ou qu'un problème survient. Nous allons donc commencer par déclarer nos deux méthodes de rappel que nous allons appeler respectivement « OnRegistrationOK » et « OnRegistrationKO » :

```
void OnRegistrationOK(RegistrationSuccessEventArgs e)
{
    // L'utilisateur a été correctement enregistré
}

void OnRegistrationKO(RegistrationExceptionEventArgs e)
{
    // Un problème est survenu pendant l'enregistrement
}
```

Une fois nos méthodes de rappel déclarées, nous pouvons lancer l'enregistrement du nouvel utilisateur à qui nous allons donner comme nom « Louis Dupont » (et oui c'est très original me direz-vous) et comme adresse email « louis.dupont@wat.com » :

```
string wNom = "Louis Dupont";
string wEmail = "louis.dupont@wat.com";

pRegistrationClient.Register(wNom, wEmail, OnRegistrationOK, OnRegistrationKO);
```

Placez des points d'arrêt sur les méthodes de rappel « OnRegistrationOK » et « OnRegistrationKO » pour voir lors de vos essais quel résultat remonte.

Voici le code complet à inclure dans la page de login :

```
private RequestSecurityTokenResponseStore pTokenResponseStore = new
RequestSecurityTokenResponseStore();
private RegistrationClient pRegistrationClient = null;

public MainPage()
{
    InitializeComponent();

    SignInControl.Realm = "uri:azurewp7acs";
    SignInControl.ServiceNamespace = "watacstest";
    SignInControl.RequestSecurityTokenResponseStore = pTokenResponseStore;

    SignInControl.RequestSecurityTokenResponseCompleted += new
EventHandler<RequestSecurityTokenResponseCompletedEventArgs>(OnRequestSecurityTokenRes
ponseCompleted);
}

void OnRequestSecurityTokenResponseCompleted(object sender,
RequestSecurityTokenResponseCompletedEventArgs e)
{
    if (e.Error != null)
    {
        // Problème
    }
    else
    {
```

```
        // Tout va bien
        // Le token est maintenant disponible dans pTokenResponseStore

        // On vérifie la registration du user lié

        long wExpire = 0;

        if (pTokenResponseStore.SecurityToken != null)
        {
            wExpire = pTokenResponseStore.RequestSecurityTokenResponse.expires;
        }

        var wStorageCredentials = new
StorageCredentialsSwtToken(pTokenResponseStore.SecurityToken, wExpire);

        string wRegistrationService = "https://127.0.0.1/RegistrationService";

        pRegistrationClient = new RegistrationClient(wRegistrationService,
wStorageCredentials);

        pRegistrationClient.CheckUserRegistration(OnUserIsRegistered,
OnUserIsNotRegistered, OnRegistrationError);
    }
}

protected override void OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    SignInControl.GetSecurityToken();
}

void OnUserIsRegistered(RegistrationSuccessEventArgs e)
{
    // L’utilisateur est déjà enregistré dans le proxy WAT
}

void OnUserIsNotRegistered(RegistrationSuccessEventArgs e)
{
    // L’utilisateur n'est pas encore enregistré dans le proxy WAT

    // On va l'enregistrer maintenant en donnant directement
    // Le nom et l'adresse email
    // Normalement on devrait passer par une page qui permet à l'utilisateur
    // de saisir lui-même son nom et son email

    string wNom = "Louis Dupont";
    string wEmail = "louis.dupont@wat.com";

    pRegistrationClient.Register(wNom, wEmail, OnRegistrationOK, OnRegistrationKO);
}

void OnRegistrationError(RegistrationExceptionEventArgs e)
{
    // Un problème est survenu pendant la vérification
}

void OnRegistrationOK(RegistrationSuccessEventArgs e)
{
    // L'utilisateur a été correctement enregistré
}
```

```
void OnRegistrationKO(RegistrationExceptionEventArgs e)
{
    // Un problème est survenu pendant l'enregistrement
}

private void BouInstallerCertificat_Click(object sender, RoutedEventArgs e)
{
    var wTask = new WebBrowserTask();
    wTask.Uri = new Uri("http://127.0.0.1:10080/127.0.0.1.cer", UriKind.Absolute);
    wTask.Show();
}
```

N'oubliez pas d'installer le certificat (via le bouton que nous avons ajouté dernièrement) afin d'être certain que les erreurs ne viennent pas de ce problème-là.

Vous pouvez dès à présent essayer d'utiliser vos identifiants Windows Live ID, Yahoo ou Google pour vous enregistrer dans le proxy WAT puis le coup suivant pour vous identifier à lui.

Conclusion

Il est moins simple de mettre en œuvre l'identification ACS que l'identification par ASP.Membership, mais la tâche est loin d'être insurmontable. WAT fait tout le travail complexe et pénible pour nous, comme d'habitude. L'utilisation du contrôle « AccessControlServiceSignIn » permet très simplement d'effectuer les opérations de sélection du fournisseur d'identité et la saisie des identifiants chez ce fournisseur. Les fonctions liées à l'enregistrement d'un nouvel utilisateur sont très simple d'emploi et l'usage systématique des méthodes de rappel permet de découper le code de manière claire.

Quel que soit la méthode utilisée (ACS ou ASP.NET Membership), le but est de récupérer le très convoité token de sécurité qui ouvre les portes de Windows Azure à vos applications mobiles.