Réalisation d'un chat en Silverlight 1.1

par Benjamin Roux (Retour aux articles)

Date de publication : 25/10/2007

Dernière mise à jour :

Dans cet article, vous verrez comment réaliser un chat en Silverlight 1.1.

1 - Introduction

Mon précédent article sur Silverlight ( **voir l'article**) étant assez théorique, j'ai décidé de vous montrez un peu de pratique dans cet article avec la réalisation d'un chat en Silverlight 1.1.

2 - Préparation du terrain

2-1 - Fonctionnement

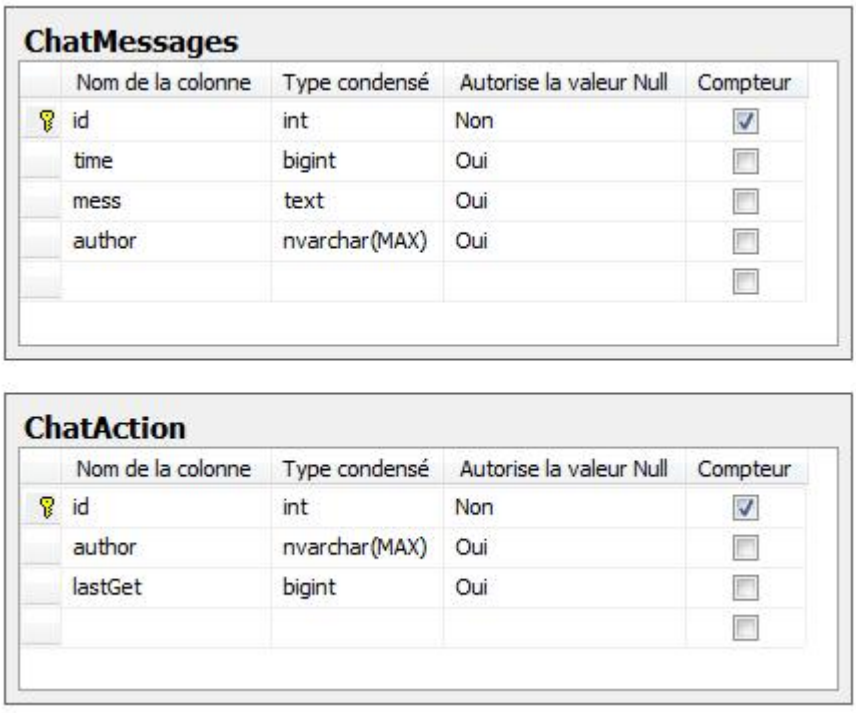
Le fonctionnement du chat est on ne peut plus simple. Tout est basé sur l'utilisation d'une base données, qui contiendra tous les messages avec l'heure à laquelle ils ont été postés, ainsi que l'heure à laquelle les utilisateurs ont récupéré les messages.
Le programme client, lui, effectuera un poll tous les x secondes et récupérera tous les messages depuis son dernier Get (ou depuis sa connexion pour la première récupération).
Les messages seront quant à eux supprimés au bout de 10 minutes, ainsi que les utilisateurs n'ayant pas donné signe de vie depuis 10 minutes (pas de récupération de messages).

2-2 - Création du projet

Pour commencer, nous allons créer une solution vide, à laquelle nous allons ajouter un projet de type Site Web.

2-3 - Création de la base de données

Voici le schéma de la base de données qui sera utilisée pour le chat.



Base de données

Rien de bien compliqué, seulement deux tables.

ChatMessages

- id : ID du message, primary key et compteur
- time : heure du message ; stockée dans un bigint
- mess : le message du message
- author : l'auteur du message

ChatAction

- id : ID de l'action, primary key et compteur
- author : auteur de l'action
- lastGet : temps du dernier get (récupération des messages) de l'auteur

Cette BDD est le minimum requis, vous pourrez bien évidemment rajouter des tables si vous le souhaitez.

Voilà donc notre base de données.

2-4 - Création de la classe pour l'accès aux données

Dans cette partie nous allons créer la classe permettant d'accéder aux données de la BDD. Nous allons appeler cette classe *MyChatMessages*

MyChatMessages
```
public class MyChatMessages
{
}
```

Attaquons nous maintenant à l'accès aux données proprement dit.
Tout d'abord occupons nous de la connexion à la base de données, pour cela, comme toujours, nous allons utiliser un *SqlConnection*.

MyChatMessages
```
public class MyChatMessages
{
    string cs = ConfigurationManager.ConnectionStrings["SkyChatConnectionString"].ConnectionString;
    SqlConnection mConnection = new SqlConnection(cs);
}
```

Je vous laisse le soin de créer votre chaîne de connexion.

Nous allons maintenant créer toute une suite de méthodes permettant d'ajouter des messages, de les récupérer, des les supprimer, de mettre à jour les actions d'un utilisateur...

Commençons par la première méthode qui sera appelée lorsqu'un utilisateur se connectera : j'ai nommé *HelloPeople*.

HelloPeople
```
public bool HelloPeople(string user)
{
    if (UserExists(user) == 1)
    {
        return false;
    }

    string timestamp = DateTime.Now.Ticks.ToString();

    if (mConnection.State == ConnectionState.Closed)
    {
        mConnection.Open();
    }

    SqlCommand command = new SqlCommand();
    command.Connection = mConnection;
    command.CommandText = "INSERT INTO ChatAction (author, lastGet) VALUES (@user, @timestamp)";
    command.Parameters.AddWithValue("@user", user);
    command.Parameters.AddWithValue("@timestamp", timestamp);
    command.ExecuteNonQuery();

    mConnection.Close();
```

HelloPeople

```
    return true;
}
```

Cette méthode vérifie tout d'abord si un utilisateur avec le même pseudo n'existe pas déjà. Si ce n'est pas le cas, on récupère le timestamp de l'heure actuelle, on ouvre ensuite la connexion. Puis on insère dans la BDD la première action de notre utilisateur à savoir sa connexion.

Voyons du coup le corps de la méthode *UserExists*.

UserExists

```
private int UserExists(string user)
{
    if (mConnection.State == ConnectionState.Closed)
    {
        mConnection.Open();
    }

    SqlCommand command = new SqlCommand();
    command.Connection = mConnection;
    command.CommandText = "SELECT COUNT(*) FROM ChatAction WHERE author=@user";
    command.Parameters.AddWithValue("@user", user);

    int res = int.Parse(command.ExecuteScalar().ToString());

    mConnection.Close();

    return res;
}
```

On regarde simplement si le pseudo **user** est déjà en action sur le chat.

Créons maintenant la méthode permettant d'ajouter un message à la base de données.

AddMessage

```
public bool AddMessage(string user, string message)
{
    if (UserExists(user) == 0)
    {
        return false;
    }

    if (mConnection.State == ConnectionState.Closed)
    {
        mConnection.Open();
    }

    SqlCommand command = new SqlCommand();
    command.Connection = mConnection;

    string timestamp = DateTime.Now.Ticks.ToString();


 command.CommandText = "INSERT INTO ChatMessages (time, mess, author) VALUES (@timestamp, @message, @user)";
    command.Parameters.AddWithValue("@timestamp", timestamp);
    command.Parameters.AddWithValue("@message", message);
    command.Parameters.AddWithValue("@user", user);

    command.ExecuteNonQuery();

    mConnection.Close();

    return true;
}
```

Le code n'est pas commenté, mais n'est pas difficile à comprendre.
On ajoute à la table ***ChatMessages*** une nouvelle ligne avec le message, l'auteur du message, mais surtout l'heure du message.

Voici maintenant les deux méthodes pour mettre à jour la dernière action (Get) d'un utilisateur et celle pour récupérer l'heure de cette fameuse dernière action.

UpdateGetForUser
```
private void UpdateGetForUser(string author, string timestamp)
{
    if (mConnection.State == ConnectionState.Closed)
    {
        mConnection.Open();
    }

    SqlCommand command = new SqlCommand();
    command.Connection = mConnection;

    command.CommandText = "UPDATE ChatAction SET lastGet=@timestamp WHERE author=@author";
    command.Parameters.AddWithValue("@timestamp", timestamp);
    command.Parameters.AddWithValue("@author", author);
    command.ExecuteNonQuery();
}
```

Toujours rien de compliqué, on met à jour l'heure du dernier Get selon le nom de l'utilisateur.

Maintenant nous allons ajouter une nouvelle méthode, pour récupérer le timestamp du dernier Get d'un utilisateur.

GetLastGetFromUser
```
private string GetLastGetFromUser(string user)
{
    if (mConnection.State == ConnectionState.Closed)
    {
        mConnection.Open();
    }

    SqlCommand command = new SqlCommand();
    command.Connection = mConnection;

    command.CommandText = "SELECT lastGet FROM ChatAction WHERE author=@user";
    command.Parameters.AddWithValue("@user", user);

    return command.ExecuteScalar().ToString();
}
```

Un simple SELECT dans la table.

Maintenant voici la méthode la plus importante pour un chat, celle pour récupérer les messages. Comme je l'ai déjà dit, nous allons récupérer tous les messages depuis le dernier Get de l'utilisateur.

GetMessages
```
public string GetMessages(string user)
{
    if (UserExists(user) == 0)
    {
        return string.Empty;
    }

    if (mConnection.State == ConnectionState.Closed)
    {
        mConnection.Open();
    }

    string timestamp = DateTime.Now.Ticks.ToString();
```

GetMessages
```
    SqlCommand command = new SqlCommand();
    command.Connection = mConnection;

    string lastGet = GetLastGetFromUser(user);

    command.CommandText = "SELECT author, mess FROM ChatMessages WHERE time > @lastGet";
    command.Parameters.AddWithValue("@lastGet", lastGet);

    DbDataReader reader = command.ExecuteReader();

    string messages = null;

    try
    {
        while (reader.Read())
        {
            messages += string.Format("<{0}> : {1}¤", reader.GetString(0), reader.GetString(1));
        }
        reader.Close();
    }
    catch
    {
    }

    UpdateGetForUser(user, timestamp);

    mConnection.Close();

    return messages;
}
```

On récupère les derniers messages via le SELECT puis on construit notre string à renvoyer. On sépare simplement chaque message par le caractère ¤ qui n'est, vous serez d'accord avec moi, jamais utilisé.
On pourrait bien évidemment effectuer un contrôle au cas où le pseudo ou le message contiendrait le caractère ¤.
On fait ensuite une mise à jour du dernier Get de l'utilisateur.

Voilà nous pouvons à présent ajouter et récupérer des messages, mais n'oublions pas la méthode pour les supprimer.

ClearMessagesAndUsers
```
public void ClearMessagesAndUsers()
{
    if (mConnection.State == ConnectionState.Closed)
    {
        mConnection.Open();
    }

    SqlCommand command = new SqlCommand();
    command.Connection = mConnection;

    command.CommandText = "DELETE FROM ChatMessages WHERE time < @time";
    command.Parameters.AddWithValue("@time", DateTime.Now.AddMinutes(-10).Ticks.ToString());
    command.ExecuteNonQuery();
    command.CommandText = "DELETE FROM ChatAction WHERE lastGet < @time";
    command.Parameters.Clear();
    command.Parameters.AddWithValue("@time", DateTime.Now.AddMinutes(-10).Ticks.ToString());
    command.ExecuteNonQuery();

    mConnection.Close();
}
```

Comme je l'ai expliqué, on supprime tous les messages âgés de plus de 10 minutes et tous les utilisateurs n'ayant pas donné signe de vie depuis 10 minutes aussi.

Voilà nous avons terminé notre classe pour l'accès aux données.
Passons à présent au Web Service qui sera chargé de faire le lien entre le client Silverlight et cette classe.

2-5 - Création du Web Service

Le Web Service sera donc chargé de faire le lien entre le client en Silverlight et la classe d'accès aux données.
Nous allons donc le doter de trois méthodes : ***NewUser***, ***SendMessage*** et ***GetMessages***.
Ajoutons donc un fichier de type Web Service à notre projet Web.

Créons maintenant nos méthodes.

NewUser
```
[WebMethod]
public bool NewUser(string user)
{
    MyChatMessages message = new MyChatMessages();

    message.ClearMessagesAndUsers();

    return message.HelloPeople(user);
}
```

Rien de difficile on appelle juste les méthodes de notre classe précédemment créée. On supprime aussi les anciens messages et les anciens utilisateurs. Ce sont donc les clients qui, à leur insu, sont chargés de cette tâche.

SendMessage
```
[WebMethod]
public bool SendMessage(string author, string message)
{
    MyChatMessages myMessage = new MyChatMessages();

    bool res = myMessage.AddMessage(author, message);
    myMessage.ClearMessagesAndUsers();

    return res;
}
```

Ici, nous appelons non seulement la méthode AddMessage, mais aussi la méthode pour supprimer les messages et les utilisateurs.

GetMessages
```
[WebMethod]
public string GetMessages(string user)
{
    MyChatMessages message = new MyChatMessages();

    string messages = message.GetMessages(user);
    message.ClearMessagesAndUsers();

    return messages;
}
```

Nous nettoyons aussi les messages et les utilisateurs dans cette méthode.

3 - Création du client Silverlight

Nous allons maintenant passer à la création de notre client Silverlight.
Ajoutons donc un projet de type Silverlight à notre solution, et ajoutons aussi une Web Reference vers notre Web Service développé auparavant.
N'oubliez pas non plus d'ajouter un lien Silverlight, de votre projet Web vers le projet Silverlight. Je rappelle la procédure : Clic droit sur le projet Web, puis Add Silverlight Link...

Pour plus d'informations sur cette méthode je vous renvoie à mon précédent article (**Silverlight et les Web Services**).

Le design de l'application ne sera, en revanche, pas développé en Silverlight, j'utiliserai simplement des contrôles HTML basiques pour afficher et saisir les messages.

Commençons par ajouter le timer qui pollera toutes les x secondes notre serveur pour récupérer les nouveaux messages.

```
<Canvas.Resources>
 <Storyboard x:Name="timer" Duration="00:00:3" />
</Canvas.Resources>
```

Ici nous créons un storyboard qui durera 3 secondes...
Allons voir du côté de notre code behind.

Dans la méthode ***Page_Loaded***, abonnons nous à l'événement ***Completed*** de notre timer, et démarrons le.

```
timer.Completed += new EventHandler(timer_Completed);
timer.Begin();
```

Créons maintenant la méthode ***timer_Completed***.

timer_Completed
```
void timer_Completed(object sender, EventArgs e)
{
    IAsyncResult iar = mService.BeginGetMessages(mHidden.GetAttribute("value"), new
 AsyncCallback(onGetMessageResponse), null);
}
```

On appelle de façon asynchrone la méthode ***GetMessages*** qui nous permettra de récupérer tous les messages. La méthode ***onGetMessageResponse*** sera appelée quand notre résultat sera prêt.
Voyons cette méthode.

onGetMessageResponse
```
public void onGetMessageResponse(IAsyncResult iar)
{
    string messages = mService.EndGetMessages(iar);

    if (messages != null)
    {
        foreach (string s in messages.Split(new char[] { '¤' }))
        {
            if (s != string.Empty)
            {
                mAllMessages.SetAttribute("value", mAllMessages.GetAttribute("value") + s + "\r\n");
            }
        }
    }

    timer.Begin();
}
```

On récupère nos messages, qui je vous le rappelle sont sous la forme d'une string unique. On découpe donc notre string selon notre délimiteur ¤ et si le message n'est pas vide nous l'ajoutons à ***mAllMessages***, qui est simplement un textarea HTML.
En effet le design de notre chat est des plus basiques, il est composé de 4 élément HMTL. Un textarea nommé **messages**, une textbox nommée **message**, un bouton nommé **send** et un champ hidden contenant le pseudo. Nous créerons notre page HTML par la suite.
Enfin, n'oublions pas de redémarrer notre timer.

Maintenant, nous allons nous occuper de nos fameux éléments HTML pour ajouter entre autres, des événements dessus.

Ajoutons tout d'abord quelques membres à notre classe Page.

```
public partial class Page : Canvas
{
    private HtmlDocument mDocument;
    private HtmlElement mAllMessages;
    private HtmlElement mSend;
    private HtmlElement mMessage;
    private HtmlElement mHidden;
    ...
}
```

Nous avons bien nos 4 éléments HTML.

Dans la méthode *Page_Loaded*, nous allons faire comme vu dans mon précédent article.

Page_Loaded
```
public void Page_Loaded(object o, EventArgs e)
{
    // Required to initialize variables
    InitializeComponent();

    mDocument = HtmlPage.Document;

    timer.Completed += new EventHandler(timer_Completed);

    mHidden = mDocument.GetElementByID("HiddenField1");

    mAllMessages = mDocument.GetElementByID("messages");
    mSend = mDocument.GetElementByID("send");
    mMessage = mDocument.GetElementByID("message");

    mService = new SilverChat.WebServiceChat.WebServiceChat();

    if (mService.NewUser(mHidden.GetAttribute("value")) == false)
    {
        mAllMessages.SetAttribute("value", "Pseudo déjà  utilisé !");
        return;
    }

    mSend.AttachEvent("onclick", new EventHandler<HtmlEventArgs>(sendButton_Clicked));

    timer.Begin();
}
```

Comme on peut le voir, nous récupérons nos différents éléments HTML via la méthode *GetElementById*, et nous ajoutons également un événement sur le clic sur notre bouton.
Vous pouvez également voir que l'on instancie notre Web Service (n'oubliez pas de rajouter ce membre à votre classe) et que nous appelons sa méthode *NewUser* avec la valeur de notre champ hidden, qui je vous le rappelle, contient le pseudo de notre utilisateur. On vérifie le retour de cette méthode et on affiche un message au cas où l'utilisateur existe déjà.

Voyons voir l'événement clic de notre bouton.

sendButton_Clicked
```
private void sendButton_Clicked(object sender, EventArgs e)
{
    string message = mMessage.GetAttribute("value");

    if (message == string.Empty)
    {
        return;
    }

    mService.SendMessage(mHidden.GetAttribute("value"), message);
    IAsyncResult iar = mService.BeginGetMessages(mHidden.GetAttribute("value"), new
 AsyncCallback(onGetMessageResponse), null);
    mMessage.SetAttribute("value", string.Empty);
}
```

Rien de bien compliqué, on récupère la valeur du champ ***message***, si le champ n'est pas vide on envoie le message et on récupère les nouveaux messages par la même occasion. On vide également la textbox.

4 - Création de l'interface du chat en HTML

Finissons avec quelque chose de facile, à savoir la création de la page HTML. J'ai décidé d'utiliser le composant ASP.NET *HiddenField* pour le champ hidden.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Chat Silverlight</title>
    <script type="text/javascript" src="Silverlight.js"></script>
    <script type="text/javascript" src="TestPage.html.js"></script>

    <style type="text/css">
        .silverlightHost { width: 0px; height: 0px; }
        #messages
        {
            height: 375px;
            width: 548px;
            background: #FFFFFF;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:HiddenField ID="HiddenField1" runat="server" />
        <div id="SilverlightControlHost" class="silverlightHost" >
            <script type="text/javascript">
                createSilverlight();
            </script>
        </div>
        <div>
            <textarea id="messages" readonly="readonly"></textarea>
        </div>
        <div>
            <input id="message" type="text" />
            <input id="send" type="button" value="Envoyer" />
        </div>
    </div>
    </form>
</body>
</html>
```

Un peu de code behind pour le remplissage du HiddenField.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        string pseudo = Request["pseudo"];
        if (pseudo == null)
        {
            Random rand = new Random();
            pseudo = "Anonyme" + rand.Next();
        }
        HiddenField1.Value = pseudo;
    }
}
```

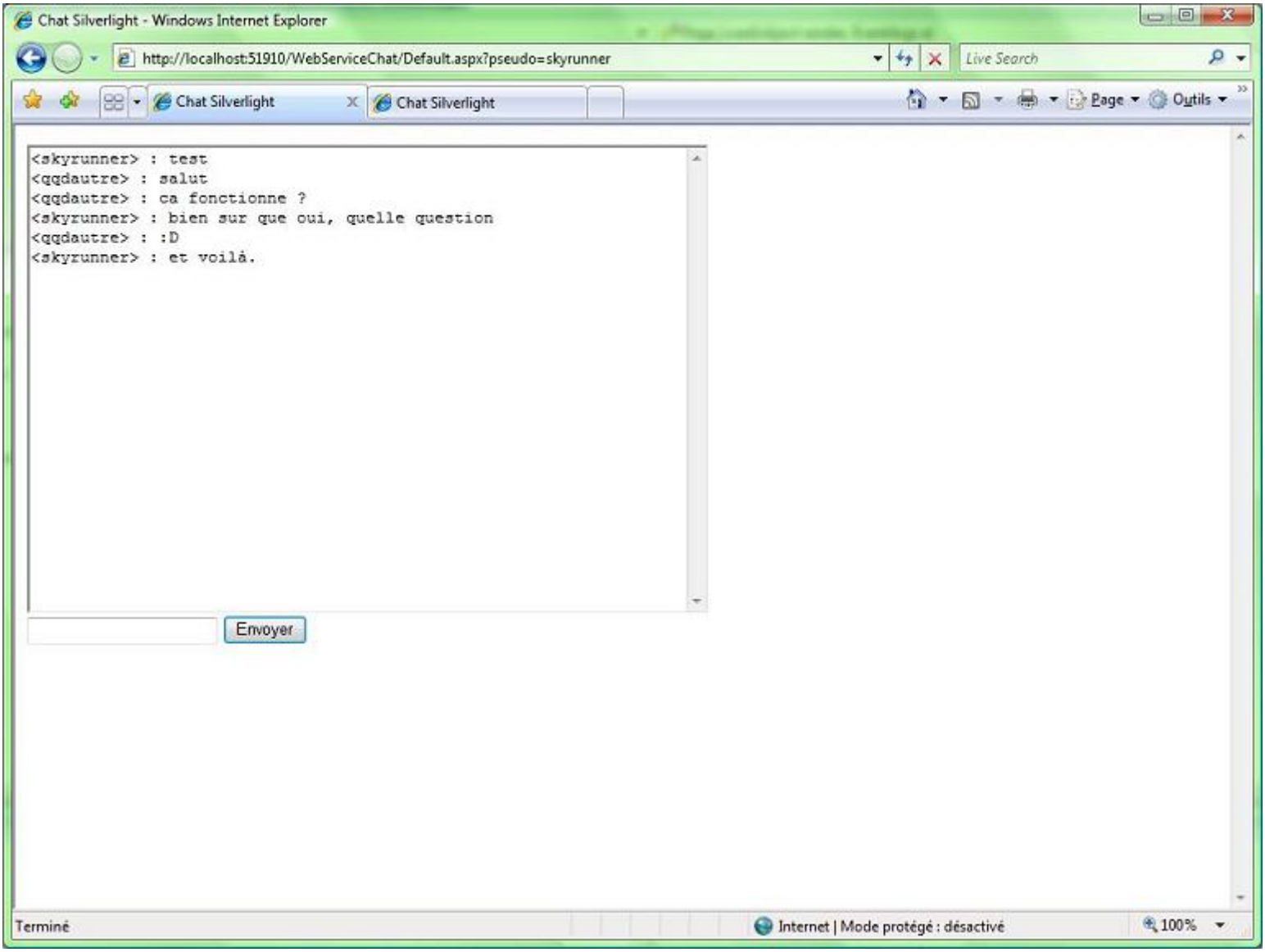
Comme vous pouvez le voir, pour remplir le champ pseudo vous devrez appeler votre page avec un ?pseudo=xxx.

Si aucun pseudo n'est spécifié on en crée un aléatoirement : "Anonyme" plus un nombre aléatoire.

Et voilà notre page HTML est prête, nous sommes donc prêts à tester.

5 - Test

Voilà vous pouvez tester votre magnifique chat réalisé en Silverlight avec les choses vues avec mon précédent article (**voir l'article**).



Chat Silverlight

6 - Conclusion

Vous venez donc de réaliser votre première vraie application à l'aide de Silverlight. Vous commencez donc sûrement à voir l'étendue des possibilités que cette technologie nous offre. Malgré sa jeunesse, un nombre de choses assez puissantes sont d'ores et déjà disponibles.
Il est bien entendu dommage de ne pas avoir profité de Silverlight pour réaliser une interface toute belle pour notre chat, mais rien ne vous empêche de le faire.

7 - Remerciements

Merci à ma petite pomme dauphine (**LineLe**) pour sa relecture et ses corrections orthographiques.