Exemple d'applications métier avec Silverlight 3 RTM et .NET RIA Services mise à jour de juillet

Partie 15 : ASP.NET MVC

par Brad Adams Jérôme Lambert (Traduction) (Espace perso) (Blog)

Date de publication : 08 janvier 2011

Dernière mise à jour :

Cet article est une traduction d'un des articles de la série "**Business Apps Example for Silverlight 3 RTM and .NET RIA Services July Update**" de Brad Adams.

I - ASP.NET MVC

Continuons notre discussion sur Silverlight 3 et de la mise à jour de .NET RIA Services. J'ai mis à jour l'exemple provenant de ma session **Mix09 "building business applications with Silverlight 3"**. Un client a récemment demandé comment utiliser ASP.NET MVC et Silverlight avec RIA Services. Ce scénario spécifique était une application avec l'interface admin complexe en Silverlight mais utilisant ASP.NET MVC pour avoir un maximum de portée pour la partie web. Le client voulait partager ses logiques applicatives autant que possible.

Donc, pour faire ça, j'ai pensé que je mettrais à jour ma démo Mix 09 pour avoir une vue ASP.NET MVC ainsi qu'une vue Silverlight.

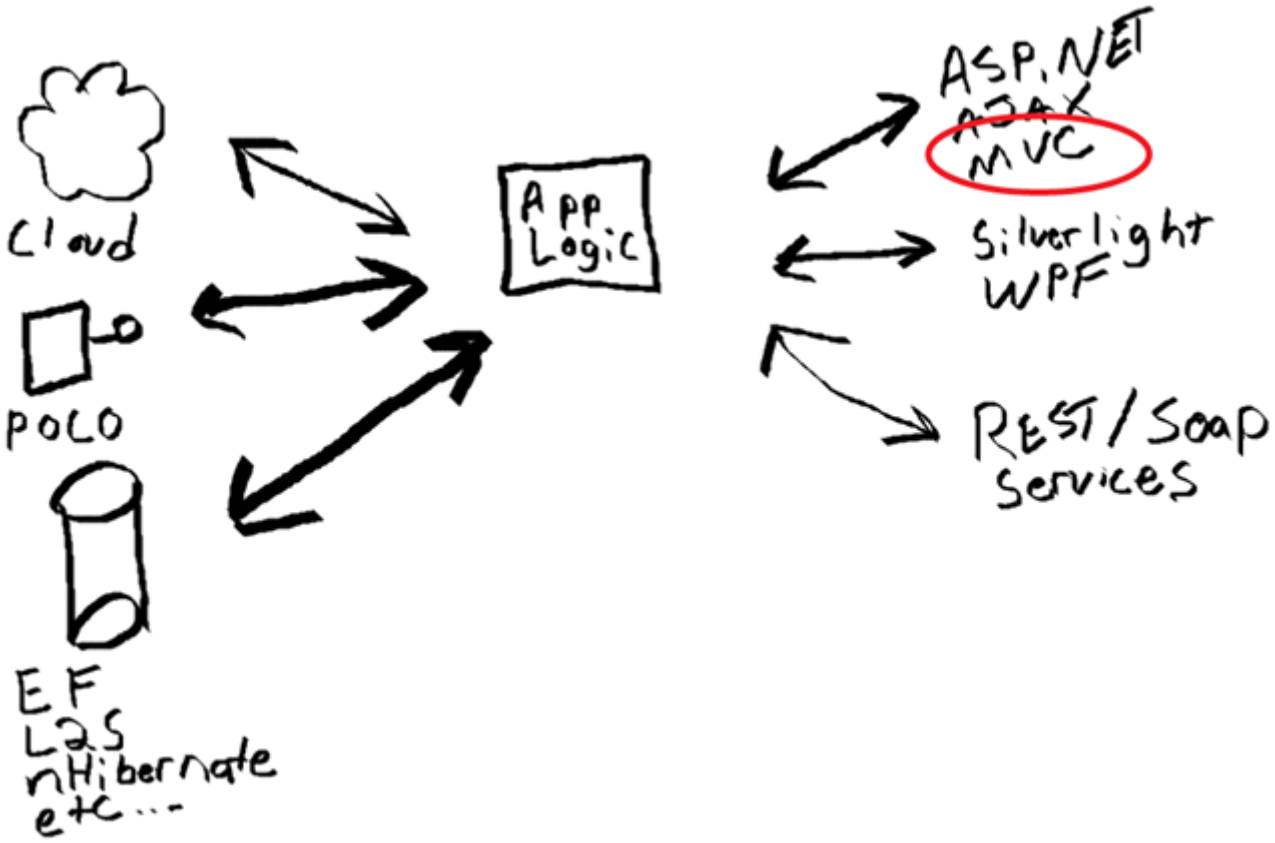
Vous pouvez regarder la **vidéo originale de la sesion complète**.

Vous pouvez voir la **série complète ici**.

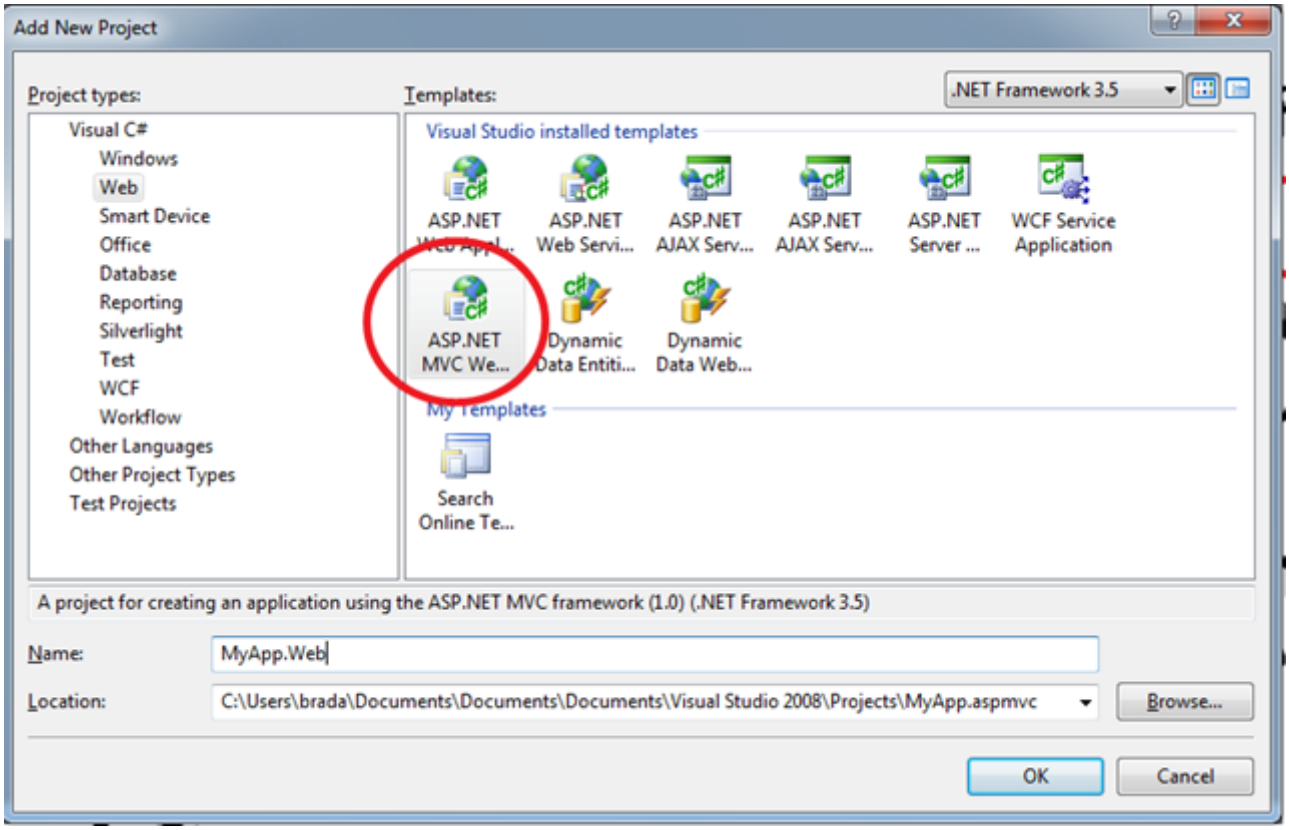
1. **VS2008 SP1**
2. **Silverlight 3 RTM**
3. **.NET RIA Services July '09 Preview** <--- N'est en réalité pas nécessaire pour cette démo ! Mais c'est néanmoins bien de le savoir ;-)
4. **ASP.NET MVC 1.0**

Téléchargez ensuite l'ensemble des fichiers des démos (**lien sur le site original** - **lien sur developpez.com**).

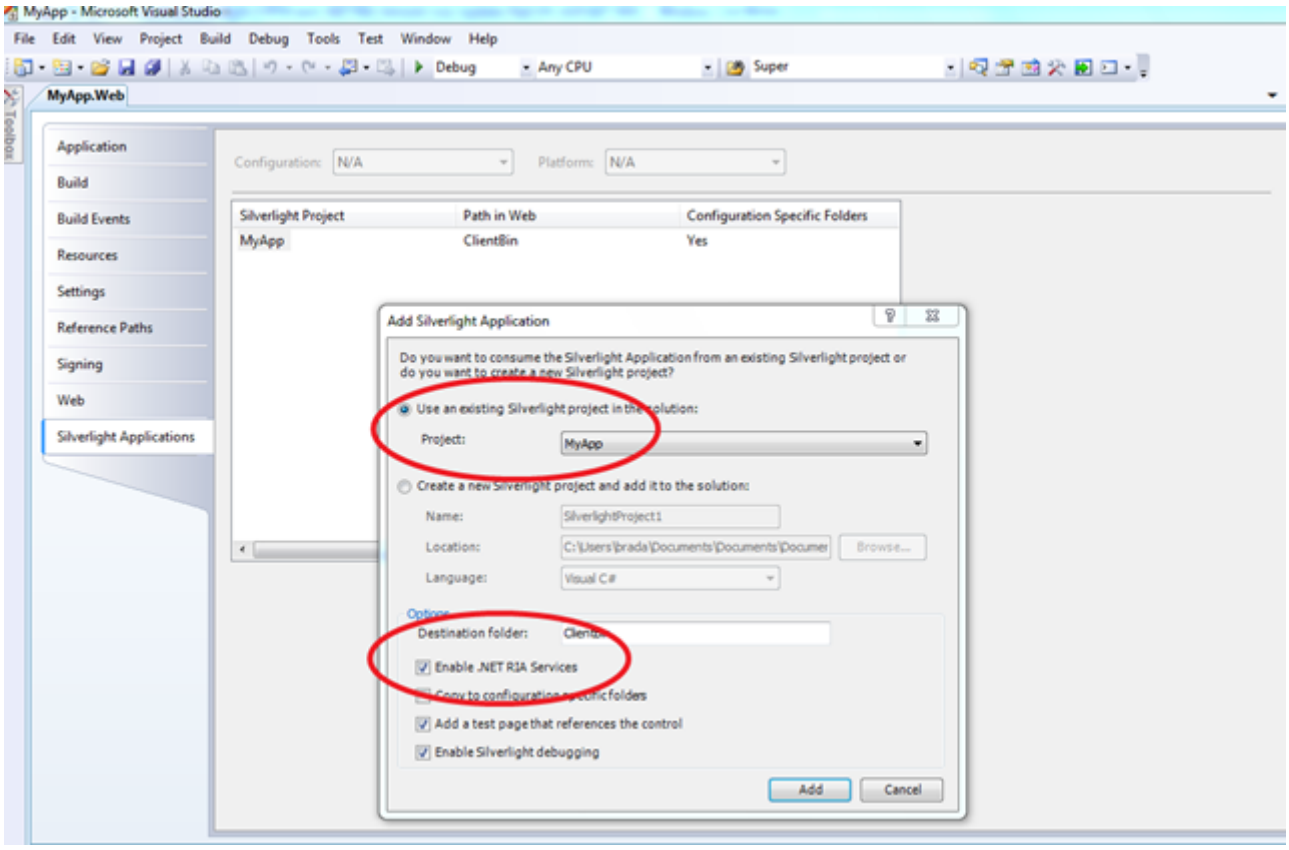
L'architecture à laquelle nous allons nous intéresser aujourd'hui concerne le développement de ASP.NET MVC reposant sur la logique applicative RIA Services. Comme vous le verrez, c'est facile à faire et cela partage tout le support de validation au niveau de l'interface.



Pour commencer, j'ai pris l'application d'origine et j'ai effacé le projet MyApp.Web et ajouté un nouveau projet qui est de type ASP.NET MVC. Vous pouvez aussi facilement ajouter un autre projet web à la même solution.



Après, j'ai associé l'application Silverlight avec ce projet web. Assurez-vous d'également activer le lien RIA Services. C'est ce qui contrôle la génération de code dans le client Silverlight.



Après, j'ai copié l'authentification liée au DomainServices du projet web.

Après, j'ai ajouté mon fichier northwind.mdf au répertoire App_Data, construit mon modèle Entity Framework et finalement j'ai ajouté mon service de domaine et je l'ai mis à jour exactement que de la même manière que celle décrite dans la partie 2. Le seule ajustement que nous avons besoin de faire est de rendre virtuel chaque méthode dans le DomainService.. Cela n'a pas d'effet réel sur le client Silverlight, mais il permet de construire un proxy pour le client MVC.

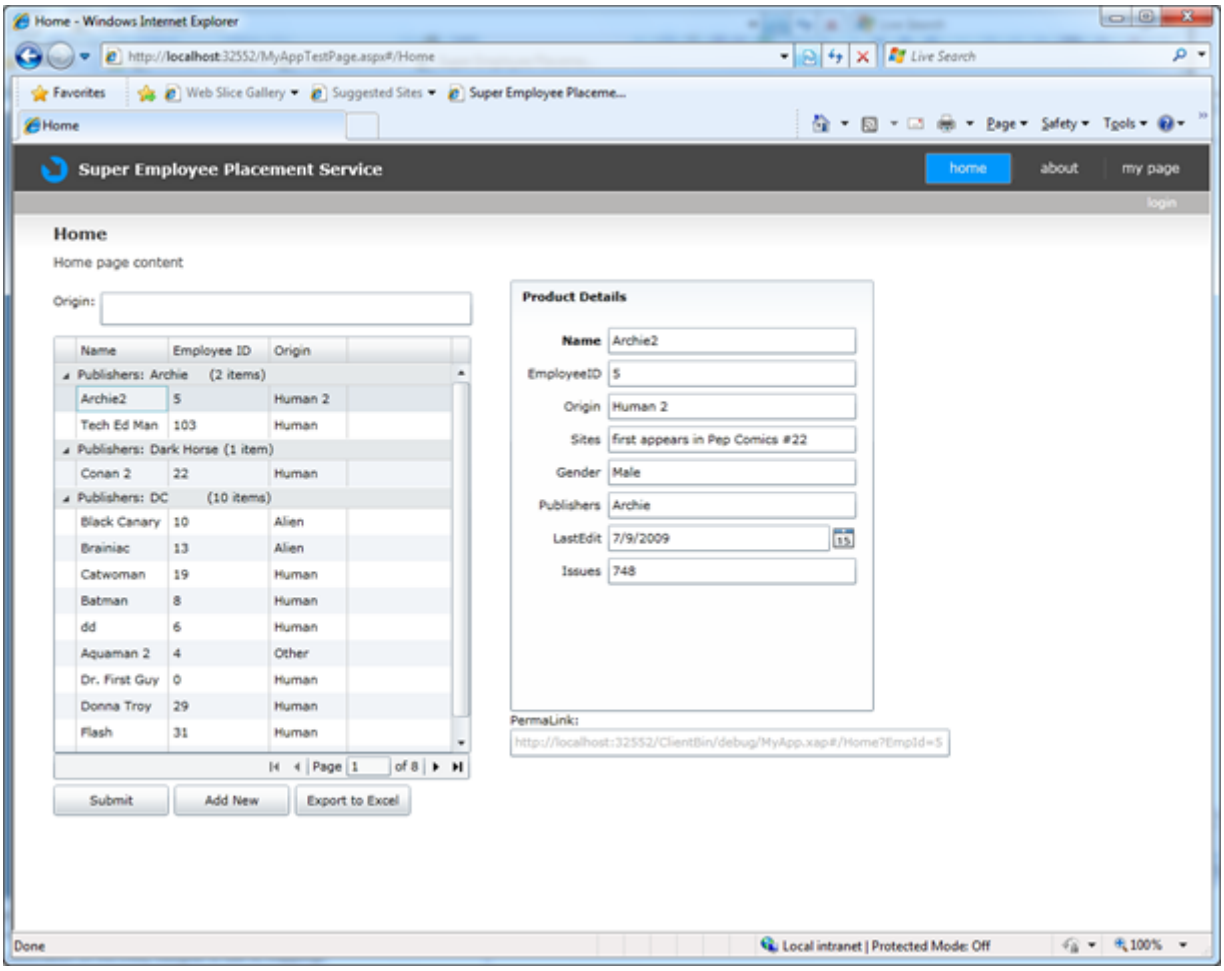
```
[EnableClientAccess]
public class SuperEmployeeDomainService : LinqToEntitiesDomainService<NORTHWNDEntities>
{

    public virtual IQueryable<SuperEmployee> GetSuperEmployees()
    {
        return this.Context.SuperEmployeeSet;
    }

    public override void Submit(ChangeSet changeSet)
    {
        base.Submit(changeSet);
    }

    public virtual void UpdateSuperEmployee(SuperEmployee currentSuperEmployee)
    {
        this.Context.AttachAsModified(currentSuperEmployee,
                                      ChangeSet.GetOriginal(currentSuperEmployee));
    }
```

Lancer la page MyAppTestPage.aspx montrerait que nous avons la même application Silverlight à jour et fonctionnelle..



Nous pouvons maintenant nous concentrer sur la partie ASP.NET MVC.

Dans le controlleur, ouvrez HomeContollers.cs.

```
1: [HandleError]
2: public class HomeController : Controller
3: {
4:     SuperEmployeeDomainService domainService =
5:         DomainServiceProxy.Create<SuperEmployeeDomainService>();
```

La ligne 5 appelle une méthode factory qui crée un wrapper de DomainService.. Cela vous permet d'avoir une syntaxe propre et d'appel direct pour le DomainService, mais permet au système de démarrer à travers son pipeline standard de validation, authorisation, etc. Notez qu'avec l'actuel CTP, vous aurez besoin de réferencer DomainServiceExtensions.dll de cet exemple pour avoir cette fonctionnalité.

```
1: public ActionResult Index()
2:  {
3:      return View("Index", domainService.GetSuperEmployees());
4:  }
```

Après, l'index est très facile.. Nous passons simplement les résultats en appelant GetSuperEmployee() tel qu'il est dans notre modèle. Cela nous permet de partager n'importe quelle logique métier qui filtre les résultats... Je peux écrire une fois et la partager entre mon application Silverlight et mon application ASP.NET MVC.

Rien de remarquable à propos de la vue... Index.aspx dans le répertoire \Views.

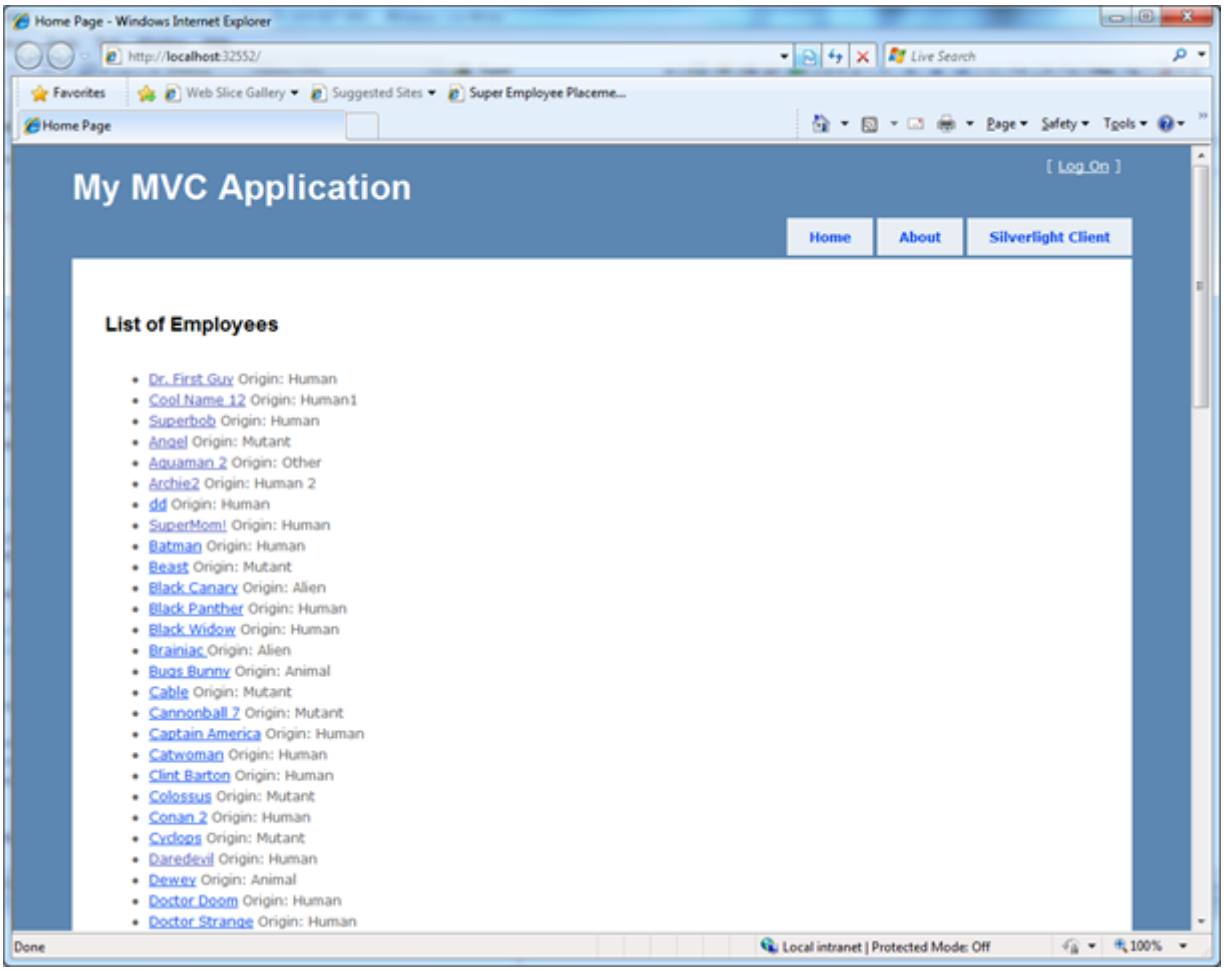
```
1: <%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
2:  Inherits="System.Web.Mvc.ViewPage<IQueryable<MyApp.Web.Models.SuperEmployee>>" %>
```

```
3:
4: <asp:Content ID="indexTitle" ContentPlaceHolderID="TitleContent" runat="server">
5:     Home Page
6: </asp:Content>
7:
8: <asp:Content ID="indexContent" ContentPlaceHolderID="MainContent" runat="server">
9:
10: <h2>List of Employees</h2>
11: <ul>
12: <% foreach (var emp in Model) { %>
13:
14: <li>
15:     <%=Html.ActionLink(emp.Name, "Details", new { id = emp.EmployeeID })%>
16:     Origin:
17:     <%=Html.Encode(emp.Origin)%>
18: </li>
19: <% } %>
20:
21: </ul>
22:
23: </asp:Content>
```

En ligne deux, nous configurons le modèle type pour être IQueryable<SuperEmployee>... Notez que c'est exactement ce que mon DomainService retournait.

Les lignes 15 à 18, j'ai un accès fortement type à SuperEmployee.

Voici à quoi cela ressemble :



Après, regardons à l'action dans le contrôlleur pour la view Details..

```
1: public ActionResult Details(int id)
2: {
3:     var q = domainService.GetSuperEmployees();
4:     var emp = q.Where(e=>e.EmployeeID==id).FirstOrDefault();
5:
6:     return View(emp);
7: }
```

De nouveau, très simple, ici nous faisons une requête Linq sur les résultats en appelant notre logique métier. La chose cool sur la composition de Linq est que nous passons d'ici, au DomainService, au modèle Entity Framework et jusqu'au chemin vers la base de données où est exécuté un code tsql efficace.

La vue Details.aspx est aussi simple que la vue ci-dessus, on accède juste au modèle.

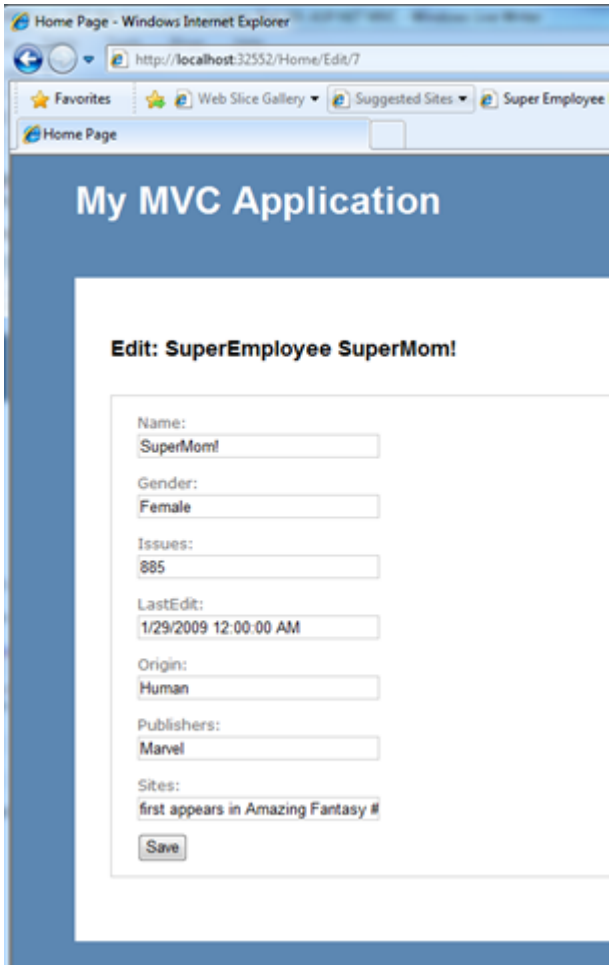
Ok - donc, le cas de la lecture est facile, qu'en est-il de la mise à jour ?

Regardons à l'action Edit dans le controlleur..

```
1: public ActionResult Edit(int id)
2: {
3:     var q = domainService.GetSuperEmployees();
4:     var emp = q.Where(e => e.EmployeeID == id).FirstOrDefault();
5:
6:     return View(emp);
7: }
```

De nouveau, très semblable à l'action Details que nous avons vu... cela remplit simplement les champs.

Maintenant, jettons un coup d'oeil à la vue edit.aspx pour cette action. Nous avons besoin d'un simple formulaire d'entrée de données..



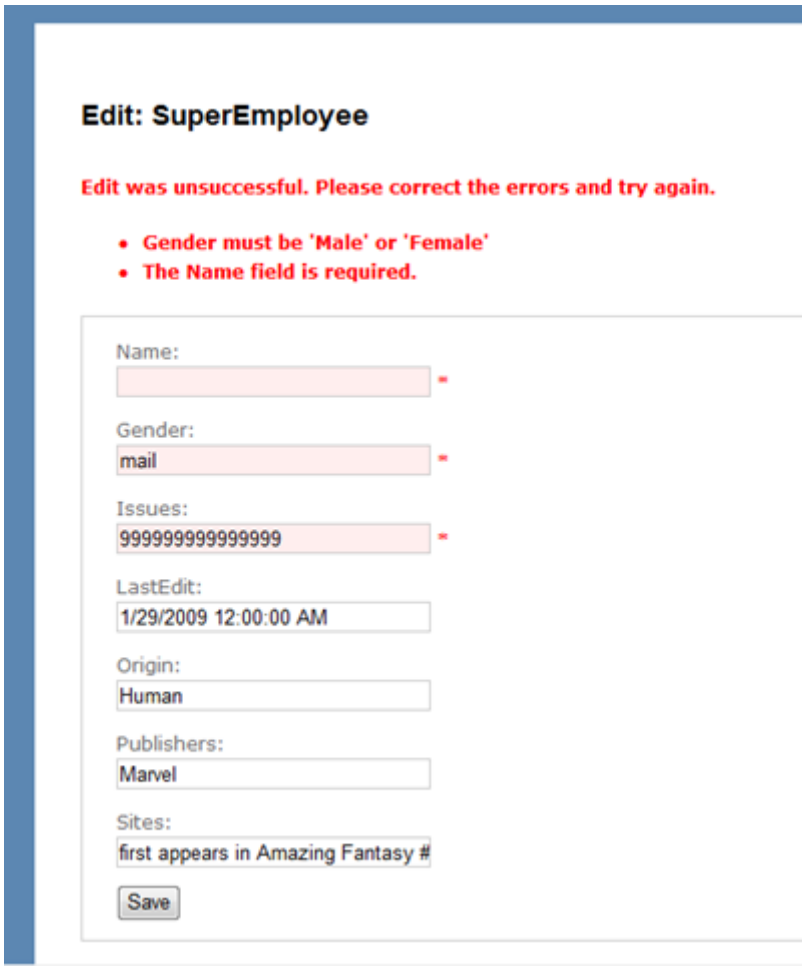
Premièrement, nous ajoutons un résumé de validation en haut du formulaire. C'est là que nous afficherons tous les problèmes de validation pour cette page.

```
<%= Html.ValidationSummary("Edit was unsuccessful. Please correct the errors and
```

Après, nous incluons du HTML standard pour chaque champ que nous devons remplir :

```
1: <p>
2:     <label for="Name">Name:</label>
3:     <%= Html.TextBox("Name", Model.Name) %>
4:     <%= Html.ValidationMessage("Name", "*") %>
5: /p>
6: <p>
7:     <label for="Name">Gender:</label>
8:     <%= Html.TextBox("Gender", Model.Gender) %>
9:     <%= Html.ValidationMessage("Gender", "*")%>
10: </p>
```

Notez que dans les lignes 4 et 9, nous avons ajouté des hooks pour la validation...



Si il y a des champs que nous ne voulons pas afficher, nous devons quand même les inclure afin qu'ils soient dans les données du postback...

```
<p>
<%= Html.Hidden("EmployeeID", Model.EmployeeID)%>
</p>
```

L'autre chose, c'est que nous voulons dans les données renvoyées au serveur l'ensemble des données originales non éditées.. C'est comme ça que nous pouvons faire les vérifications d'accès concurrents avec la base de données.

```
<%= Html.Hidden("orginal.EmployeeID", Model.EmployeeID)%>
<%= Html.Hidden("orginal.Gender", Model.Gender)%>
<%= Html.Hidden("orginal.Issues", Model.Issues)%>
<%= Html.Hidden("orginal.LastEdit", Model.LastEdit)%>
<%= Html.Hidden("orginal.Name", Model.Name)%>
<%= Html.Hidden("orginal.Origin", Model.Origin)%>
<%= Html.Hidden("orginal.Publishers", Model.Publishers)%>
<%= Html.Hidden("orginal.Sites", Model.Sites)%>
```

Notez ici la convention de nommage dans original.blahh.. comme vous verrez plus tard, cela coincide avec le nom de l'argument sur l'action du controlleur. Retournons à l'action du controlleur et regardons..

```
1: [AcceptVerbs(HttpVerbs.Post)]
2: public ActionResult Edit(SuperEmployee emp, SuperEmployee orginal)
3: {
4:
5:     if (ModelState.IsValid)
6:     {
7:         domainService.AssociateOriginal(emp, orginal);
8:         domainService.UpdateSuperEmployee(emp);
```

```
9:          return RedirectToAction("Details", new { id = emp.EmployeeID });
10:     }
11:
12:     return View(emp);
13:
14: }
```

Notez que ma méthode prend deux arguments, emp, qui est l'employé actuel tel qu'il a été mis à jour à partir du formulaire et l'employéé "original" qui a provient des champs html cachés.

Après en ligne 7, nous associons la valeur orginale avec l'employé... Cela se fait simplement lors de l'appel de ChangeSet.GetOriginal() de la classe DomainService qui retournera la bonne valeur. Après, nous appelons UpdateSuperEmployee() dans la logique métier. Ici nous faisons toutes les validations définies pour le modèle incluant l'exécution du code des validations personnalisées. Si la validation rate, l'information d'erreur sera affichée dans le formulaire. Sinon, les données sont finalement sauvées en base de données.

Ce que j'ai montré est le développement d'un application ASP.NET MVC qui partage la même logique applicative que le client Silverlight. Cela inclut des choses telles que validation des données. Pour en revenir à notre sujet, voici un exemple de la même erreur de validation dans le client Silverlight, puis dans le client ASP.NET MVC...