DEV MAGIC FAKE

JUST ONE LINE OF CODE

How to use? Tutorial

By: M.Radwan

TABLE OF CONTENTS

1. Overview

The document is a tutorial that introduces and explains step-by-step guide how to use Dev Magic Fake with ASP.NET MVC 3.0 project. We'll be starting slowly, so beginner level of software development experience is okay.

1.1. Running scenario

We assume a business scenario that has business activities, so we can try Dev Magic Fake, the scenario will contain the following information

We have a company that provide Wi Max communication services to its clients through install this services on products that provided by some vendors.

We have the vendors like (HP, Cisco, DLink) that provide products to our company

We have products that provided by many vendors like (USB modems, switches and routers)

We have Services that installed on the product that provided by the vendors (Like TV and Internet)

1.2. Convention over configuration

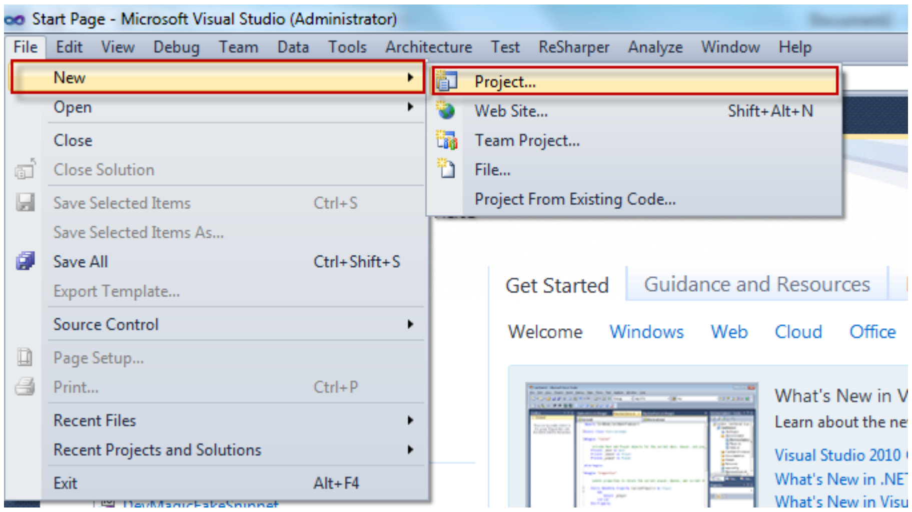
Some of the feature of Dev Magic Fake depends on configuration and naming convention, if we don't want to follow this naming or convention we can overwrite it using the configuration file

2. Create the project that will use Dev Magic Fake

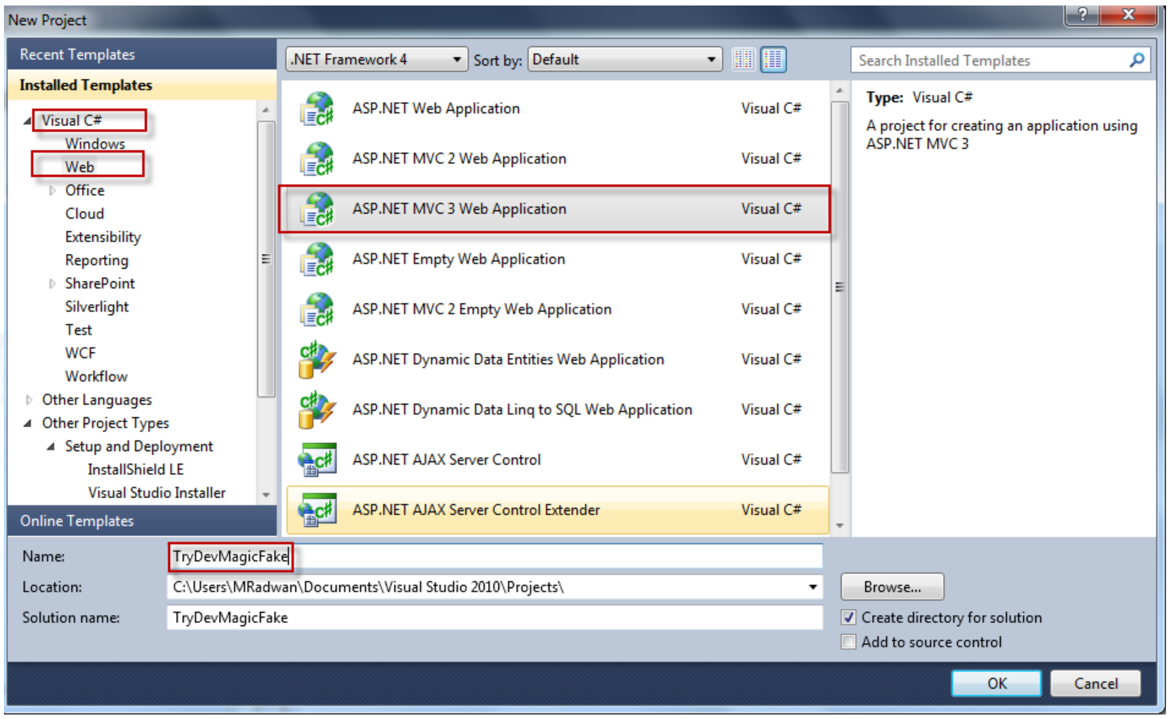
2.1. Creating a newASP.NET MVC 3.0 project

We will start by creating MVC project that will use Dev Magic Fake to implement the previous business scenario

- Open visual studio
- Create new project and select Web Templates


Start Page - Microsoft Visual Studio (Administrator)
File Edit View Debug Team Data Tools Architecture Test ReSharper Analyze Window Help
New
Open
Close
Close Solution
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S
Export Template...
Source Control
Page Setup...
Print... Ctrl+P
Recent Files
Recent Projects and Solutions
Exit Alt+F4
Project...
Web Site... Shift+Alt+N
Team Project...
File...
Project From Existing Code...
Get Started
Guidance and Resources
Welcome Windows Web Cloud Office

- Select ASP.NET MVC3 Project and name the project TryDevMagicFake


New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Office
Cloud
Extensibility
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Setup and Deployment
InstallShield LE
Visual Studio Installer
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
ASP.NET Web Application Visual C#
ASP.NET MVC 2 Web Application Visual C#
ASP.NET MVC 3 Web Application Visual C#
ASP.NET Empty Web Application Visual C#
ASP.NET MVC 2 Empty Web Application Visual C#
ASP.NET Dynamic Data Entities Web Application Visual C#
ASP.NET Dynamic Data Linq to SQL Web Application Visual C#
ASP.NET AJAX Server Control Visual C#
ASP.NET AJAX Server Control Extender Visual C#
Browse...
OK
Cancel

Type: Visual C#

A project for creating an application using ASP.NET MVC 3

Name: TryDevMagicFake

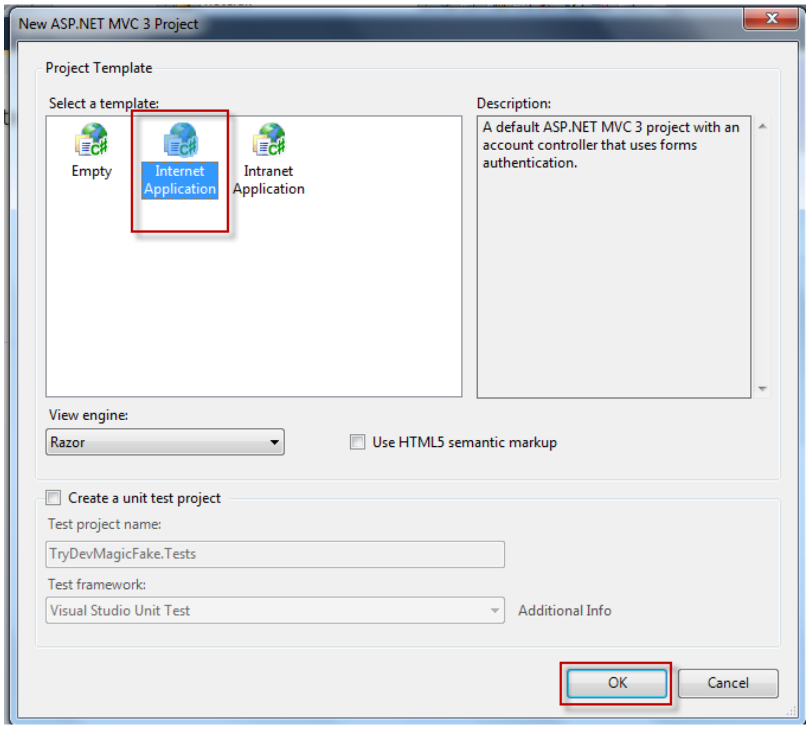
Location: C:\Users\MRadwan\Documents\Visual Studio 2010\Projects\

Solution name: TryDevMagicFake

[✓] Create directory for solution

[ ] Add to source control

- Select Internet application template



New ASP.NET MVC 3 Project

Project Template

Select a template:

Description:

A default ASP.NET MVC 3 project with an account controller that uses forms authentication.

View engine: Razor

Use HTML5 semantic markup

Create a unit test project

Test project name: TryDevMagicFake.Tests

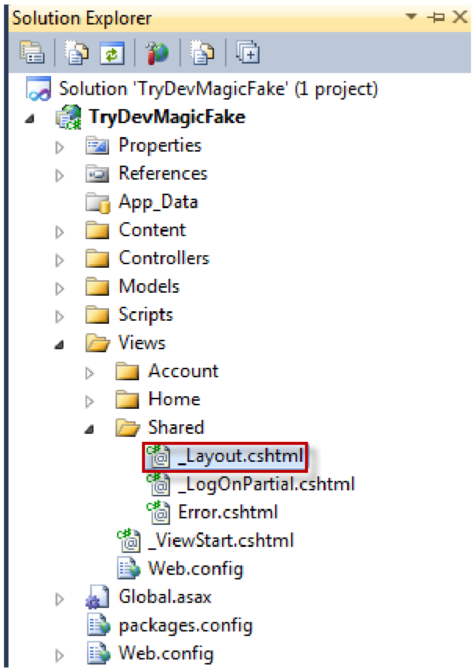
Test framework: Visual Studio Unit Test

Additional Info

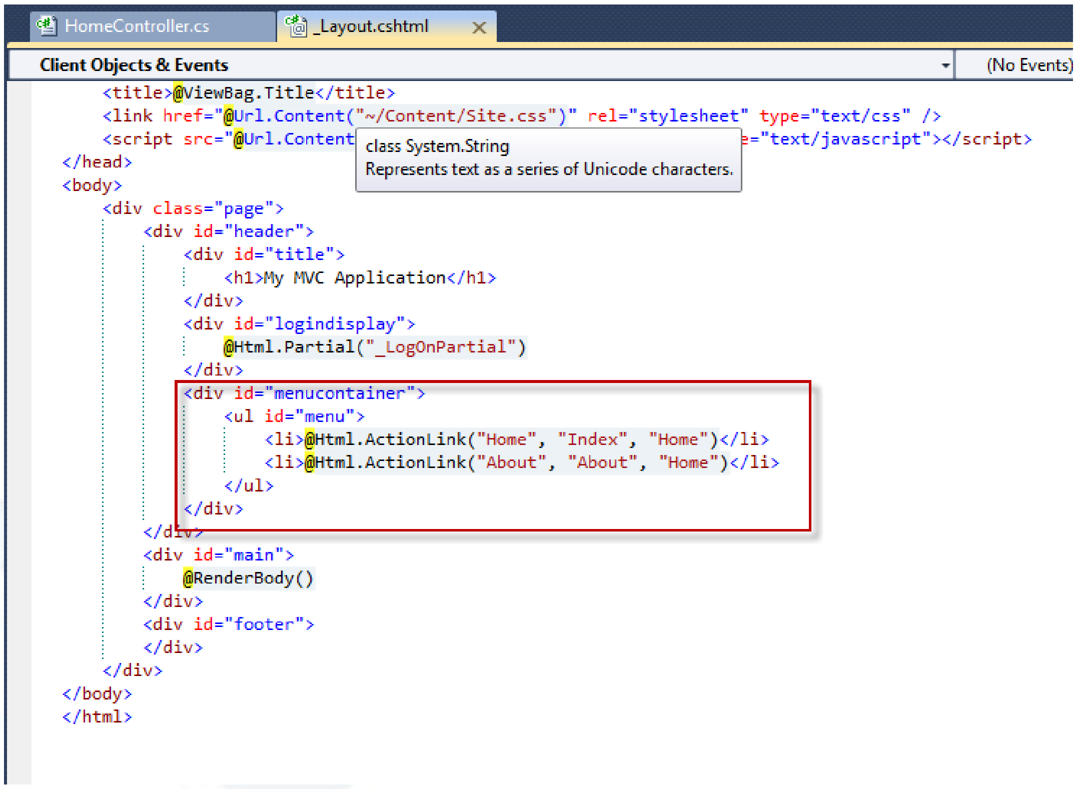
2.2. Add new links for the running scenario to the site navigation

We will add some links to the master page so it will be displayed in any page, the links will be for our previous business scenario like create new vendor, create new product, create new service, vendors etc..

- Open _Layout.cshtml

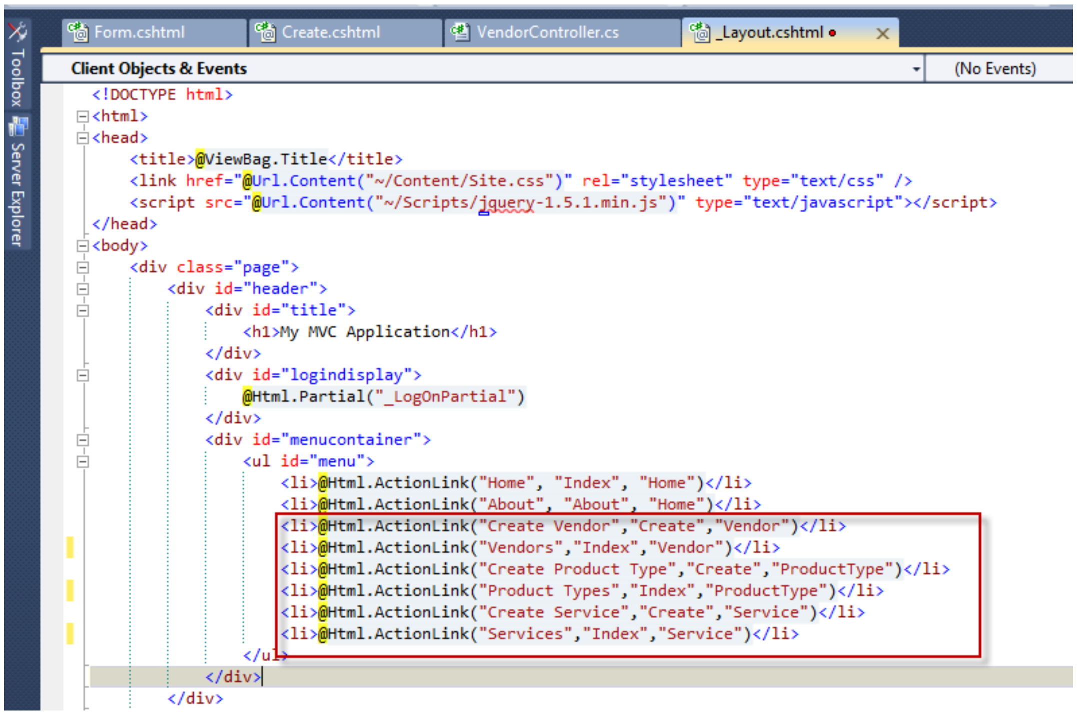


- Examine the following area, for the navigation menu



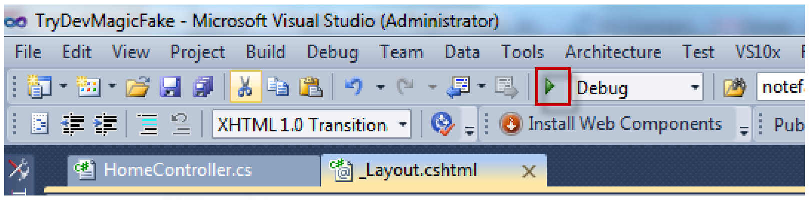
```
        <title>@ViewBag.Title</title>
        <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
        <script src="@Url.Content                                  ="text/javascript"></script>
    </head>
    <body>
        <div class="page">
            <div id="header">
                <div id="title">
                    <h1>My MVC Application</h1>
                </div>
                <div id="logindisplay">
                    @Html.Partial("_LogOnPartial")
                </div>
                <div id="menucontainer">
                    <ul id="menu">
                        <li>@Html.ActionLink("Home", "Index", "Home")</li>
                        <li>@Html.ActionLink("About", "About", "Home")</li>
                    </ul>
                </div>
            </div>
            <div id="main">
                @RenderBody()
            </div>
            <div id="footer">
            </div>
        </div>
    </body>
    </html>
```

- Enter the following links helper methods

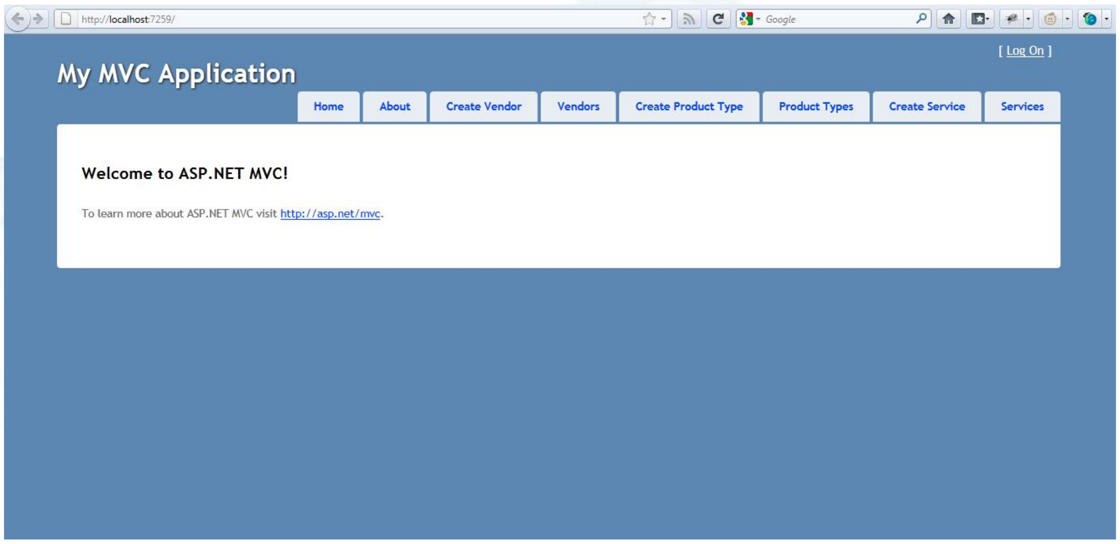


```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
    <link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
    <script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")" type="text/javascript"></script>
</head>
<body>
    <div class="page">
        <div id="header">
            <div id="title">
                <h1>My MVC Application</h1>
            </div>
            <div id="logindisplay">
                @Html.Partial("_LogOnPartial")
            </div>
            <div id="menucontainer">
                <ul id="menu">
                    <li>@Html.ActionLink("Home", "Index", "Home")</li>
                    <li>@Html.ActionLink("About", "About", "Home")</li>
                    <li>@Html.ActionLink("Create Vendor","Create","Vendor")</li>
                    <li>@Html.ActionLink("Vendors","Index","Vendor")</li>
                    <li>@Html.ActionLink("Create Product Type","Create","ProductType")</li>
                    <li>@Html.ActionLink("Product Types","Index","ProductType")</li>
                    <li>@Html.ActionLink("Create Service","Create","Service")</li>
                    <li>@Html.ActionLink("Services","Index","Service")</li>
                </ul>
            </div>
        </div>
```

- Run the application



- We should see now the added links displayed



Welcome to ASP.NET MVC!

To learn more about ASP.NET MVC visit http://asp.net/mvc.

3. Create the business classes that will be used in our scenario

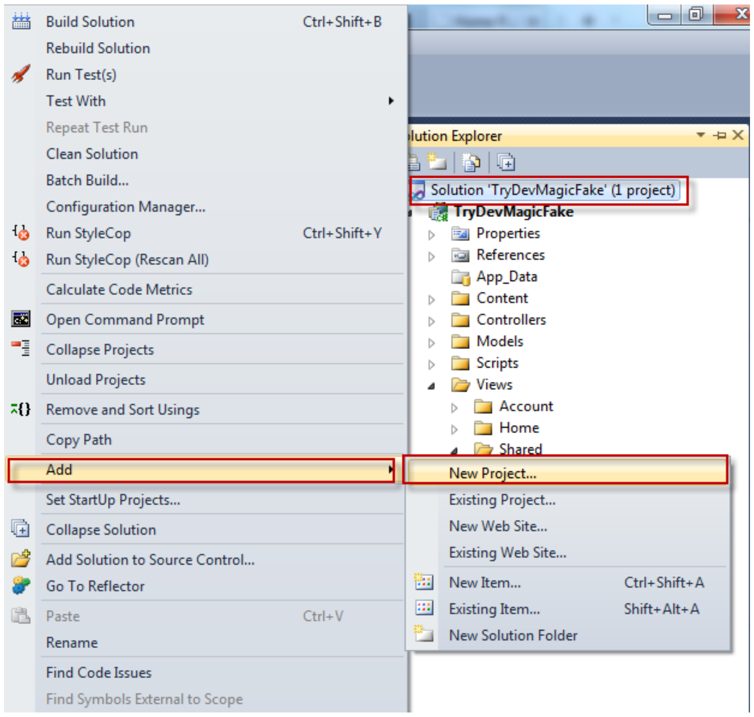
We will create a new class library to hold our business classes that will be used in our tutorial, we have to name this project (**Domain**), if we don't want to follow this convention we can name it anything else and change the default settings in the Dev Magic Fake configuration file (Project configuration file, in our case it's Web.Config), to set the assembly name of our business class, we can also use the MVC project itself to hold our business classes in this case we have to define the classes that will be used by Dev Magic Fake, because we don't want every class in the MVC project to be faked like controller or utilities, we can do this by many option like create all classes in the same name space and set the name space configuration in the Dev Magic Fake configuration file, we can also use Fakeable attribute to mark the class that need to be faked, we can also excluded class by mark the class with NotFakeable attribute, for more information about Dev Magic Fake configuration please see the Codeplex site for more documentation about Dev Magic Fake features and help files

3.1. Create new class library project

We will create 3 classes as the following scenario:

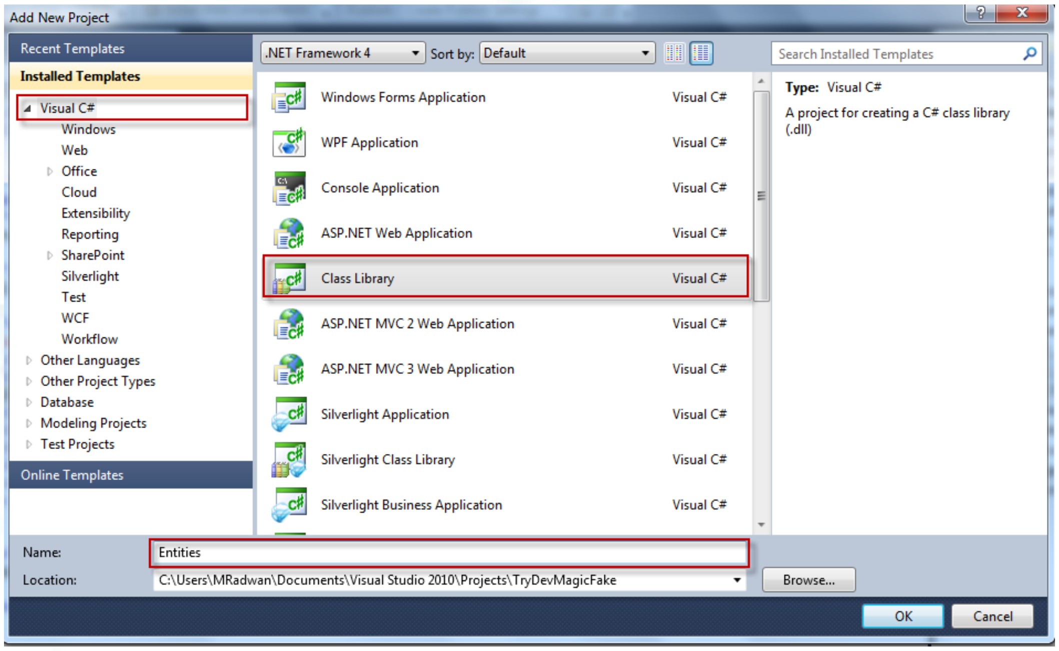
We have Vendor class which is a simple class that only has primitive data types, we have Product type which is a complex class that has primitive data types and only one class user define type (Vendor), we have Service class which is a complex class that has primitive data types and two classes user define types (Vendor and Product type)

- Right click on the solution and select add new project



- Name the project Domain

Note: We will notice that the class library name is **Entities** not **Domain** in all screen images this because these images not updated, but the class library name should be **Domain** to follow the default configuration, otherwise you need to set the assembly name in the configuration file


Add New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Office
Cloud
Extensibility
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Database
Modeling Projects
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
ASP.NET Web Application Visual C#
Class Library Visual C#
ASP.NET MVC 2 Web Application Visual C#
ASP.NET MVC 3 Web Application Visual C#
Silverlight Application Visual C#
Silverlight Class Library Visual C#
Silverlight Business Application Visual C#
Browse...
OK
Cancel

Type: Visual C#

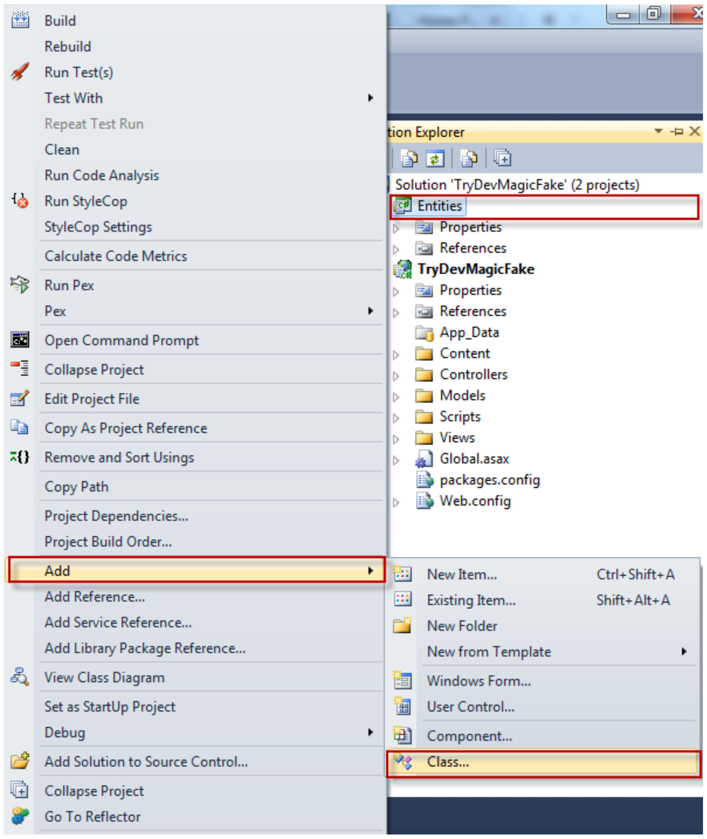
A project for creating a C# class library (.dll)

Name: Entities

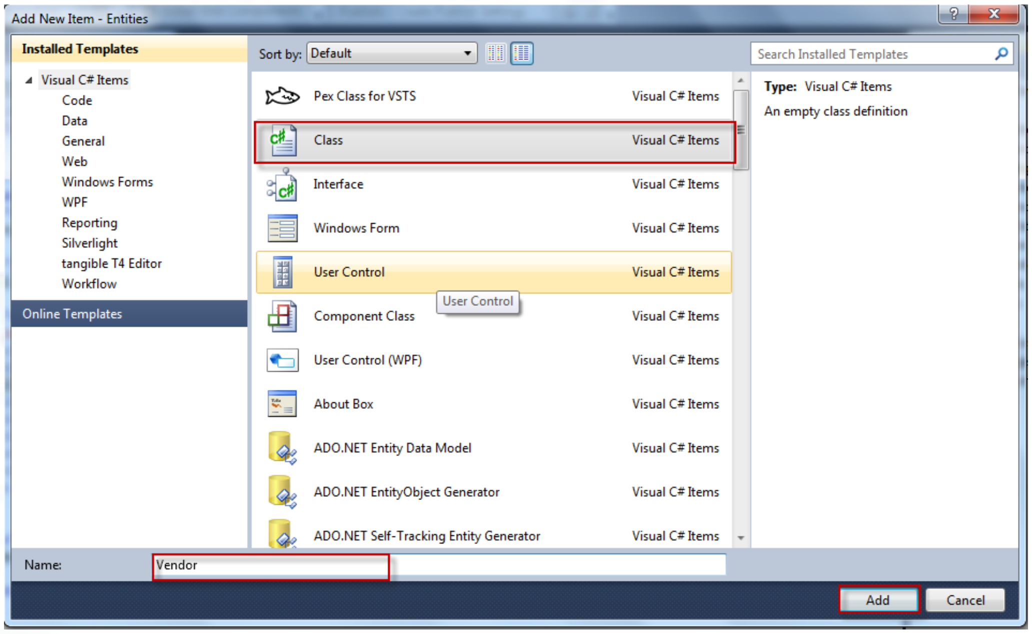
Location: C:\Users\MRadwan\Documents\Visual Studio 2010\Projects\TryDevMagicFake

3.2. Add business classes, Vendor, Product Type and Service

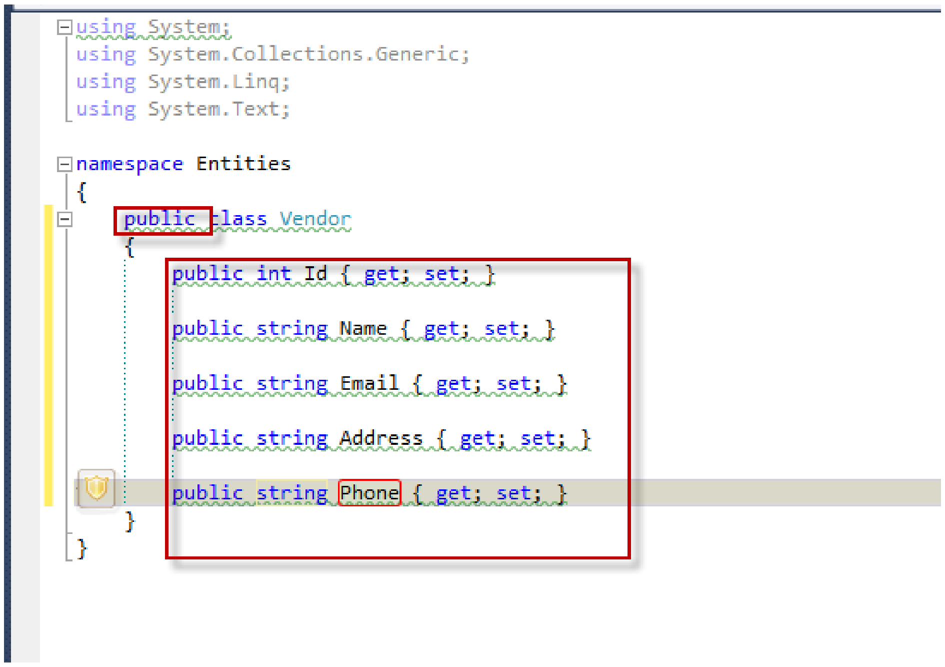
- Add new Class



- Name the class Vendor

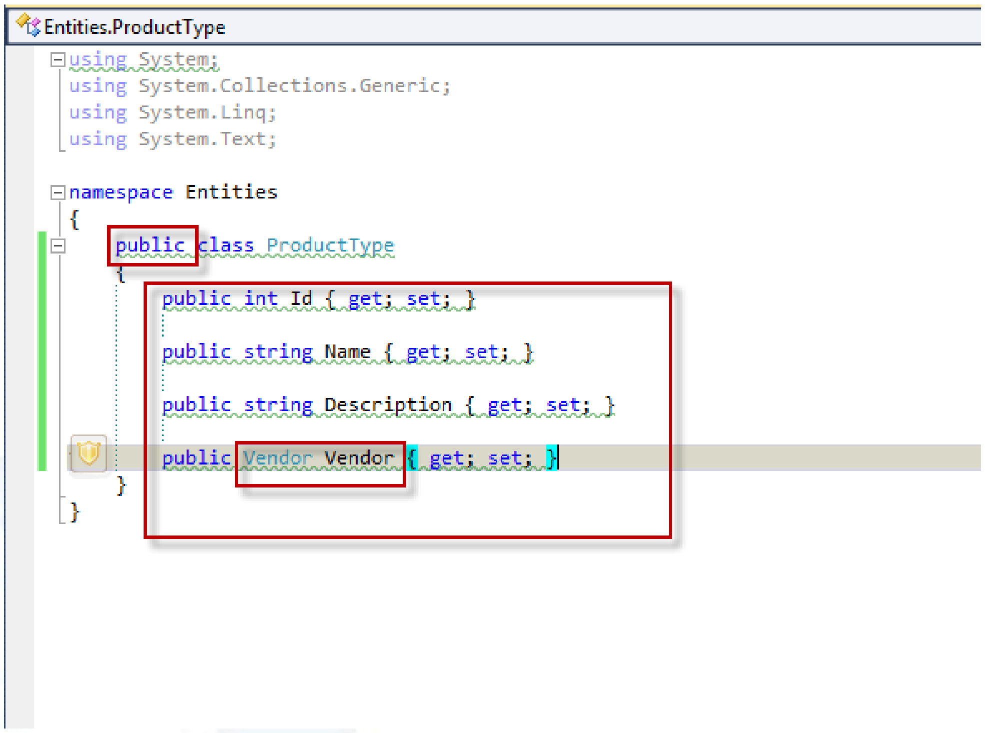

Add New Item - Entities
Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Silverlight
tangible T4 Editor
Workflow
Online Templates
Sort by: Default
Search Installed Templates
Pex Class for VSTS Visual C# Items
Class Visual C# Items
Interface Visual C# Items
Windows Form Visual C# Items
User Control Visual C# Items
Component Class Visual C# Items
User Control (WPF) Visual C# Items
About Box Visual C# Items
ADO.NET Entity Data Model Visual C# Items
ADO.NET EntityObject Generator Visual C# Items
ADO.NET Self-Tracking Entity Generator Visual C# Items
User Control
Type: Visual C# Items
An empty class definition
Name: Vendor
Add
Cancel

- Change the access modifier to be public and enter the following properties



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Entities
{
    public class Vendor
    {
        public int Id { get; set; }

        public string Name { get; set; }

        public string Email { get; set; }

        public string Address { get; set; }

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

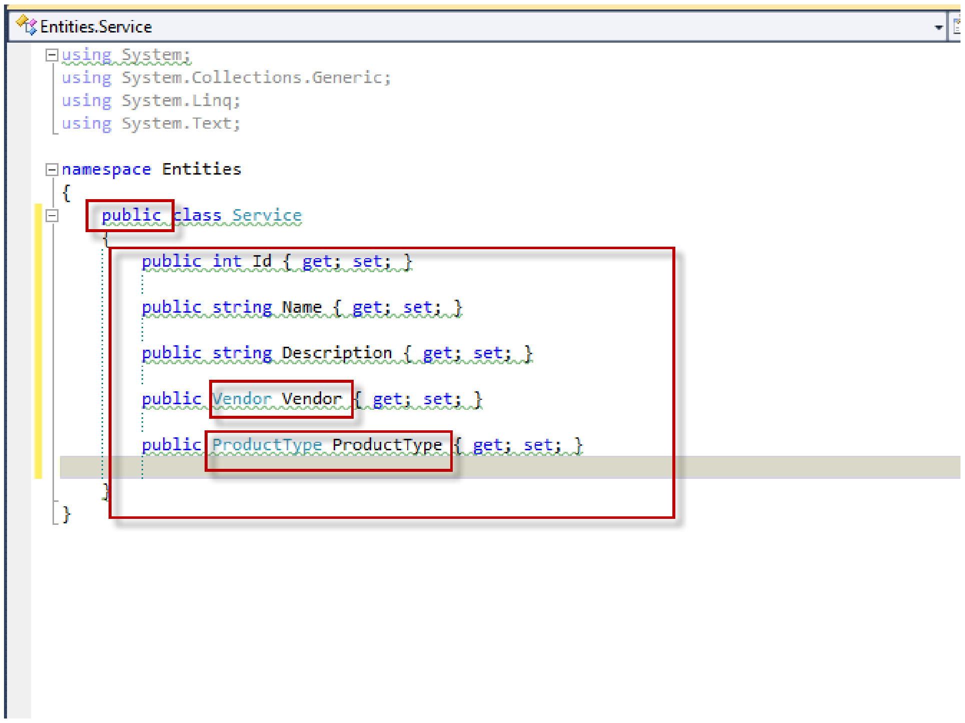
- Create Product Type class and change the access modifier to be public and enter the following properties



Entities.ProductType

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Entities
{
    public class ProductType
    {
        public int Id { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }

        public Vendor Vendor { get; set; }
    }
}
```

- Create class service and enter the following properties



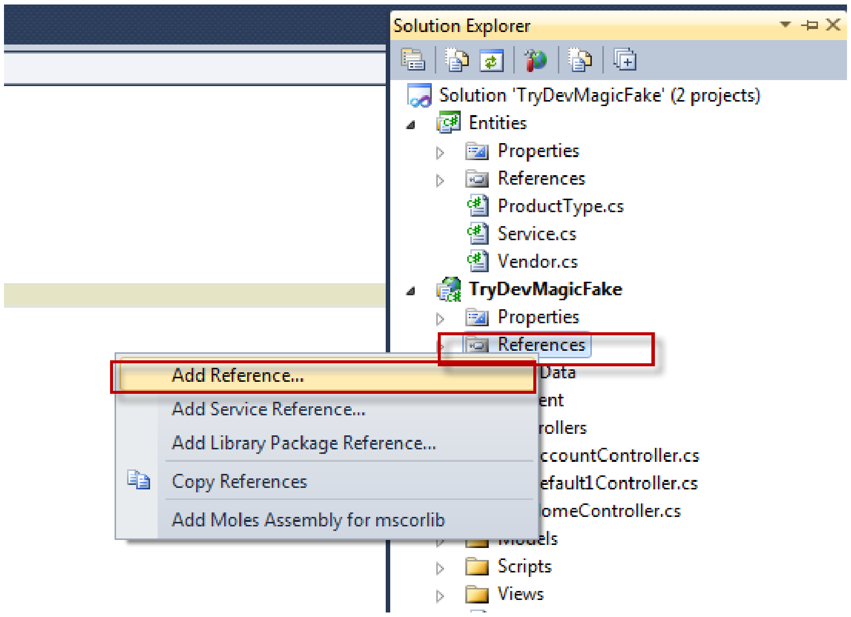
Entities.Service

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Entities
{
    public class Service
    {
        public int Id { get; set; }

        public string Name { get; set; }

        public string Description { get; set; }

        public Vendor Vendor { get; set; }

        public ProductType ProductType { get; set; }
    }
}
```

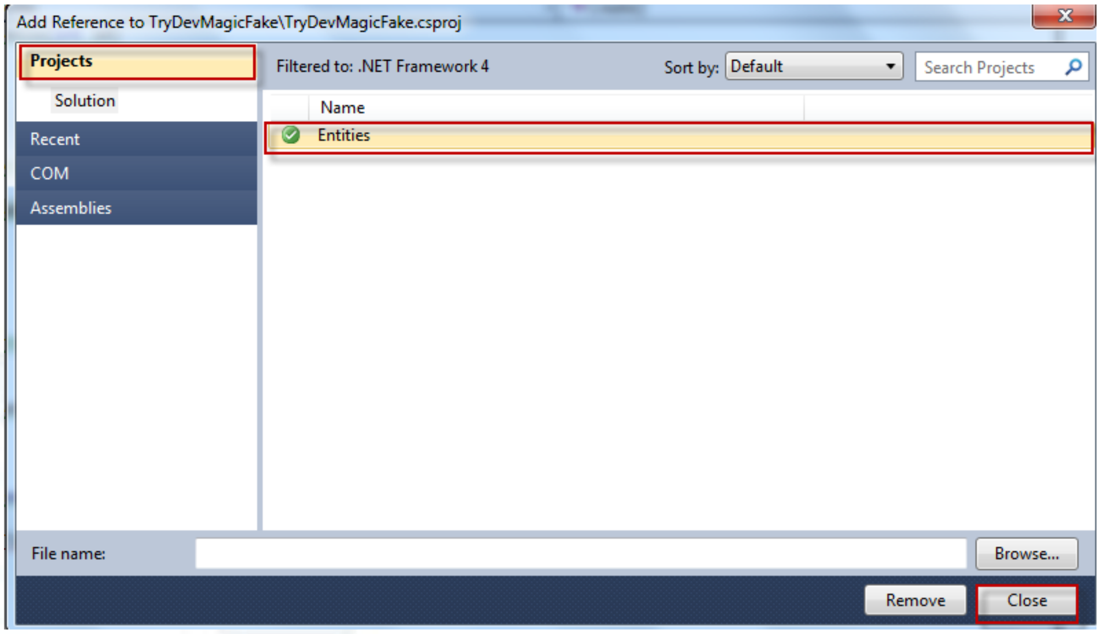
4. Reference project Domain, Install and link Dev Magic Fake

4.1. Add reference to the Domain class library

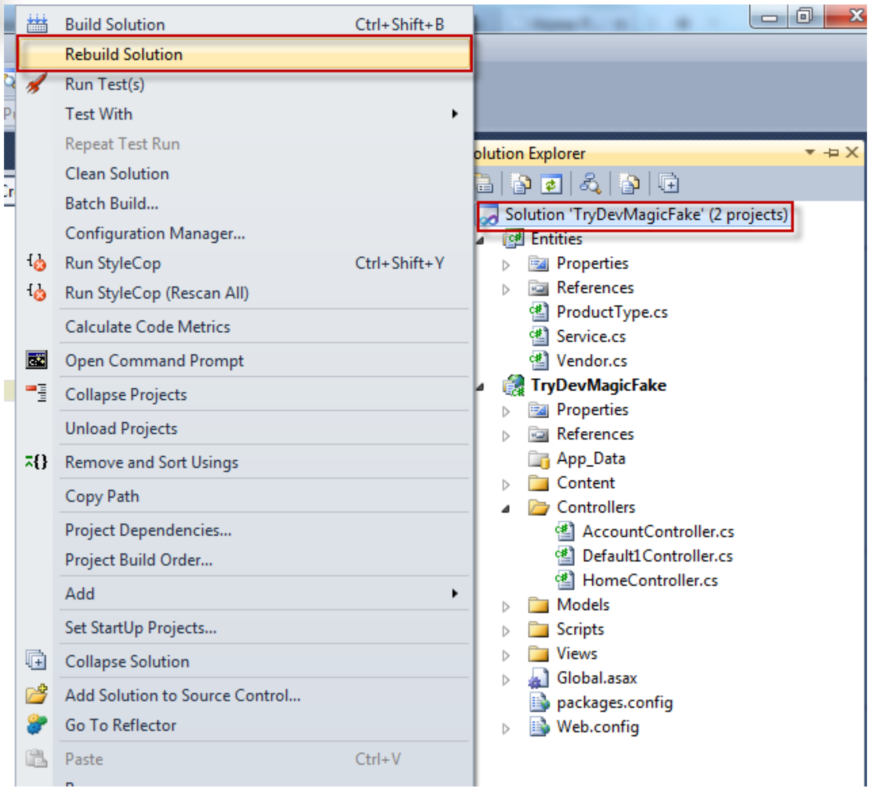
- Right click on the References and select add reference



- Select project and select **Domain** (entities before)



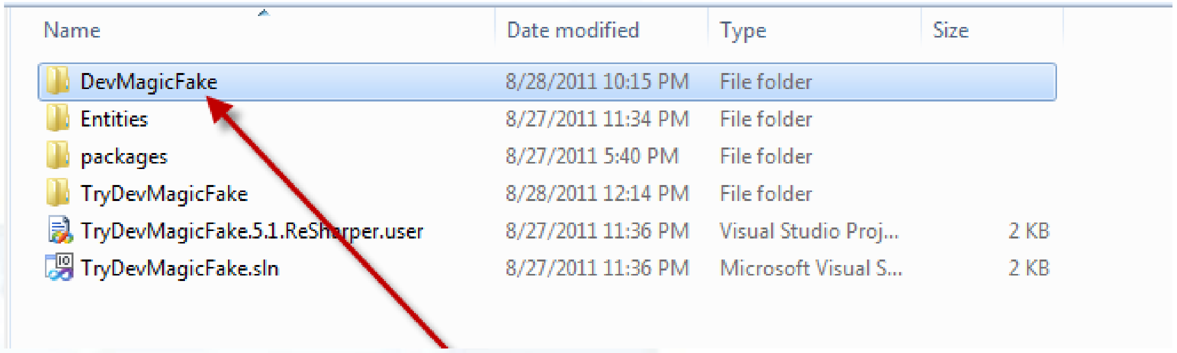
- Rebuild the solution as the following:



In order to use Dev Magic Fake we need to add reference to its assembly, it better to copy it to our project directory

4.2. Copy Dev Magic Fake to our project

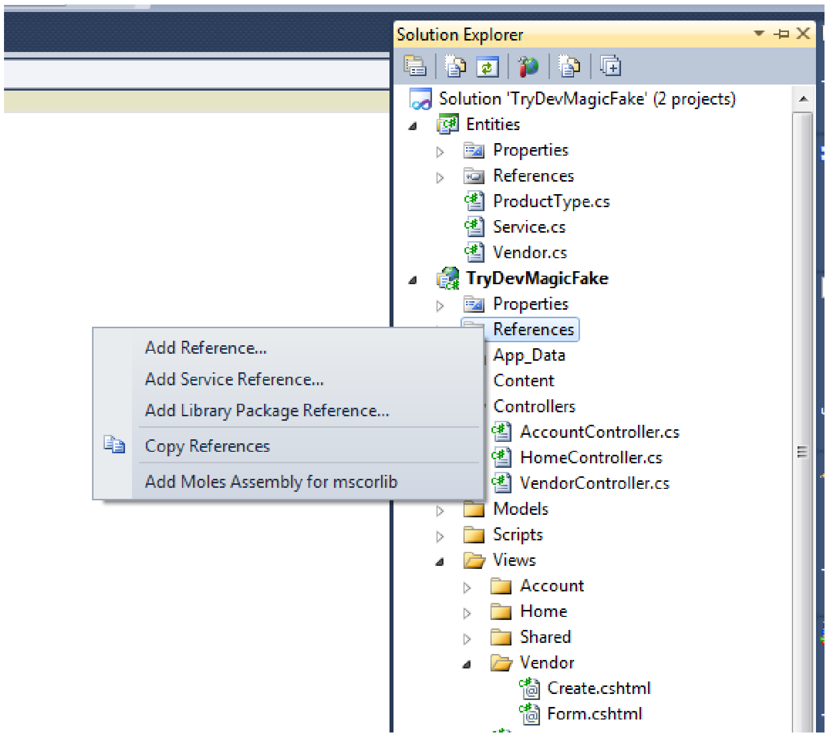
- Right click on the Dev Magic Fake folder and select copy and paste in the TryDevMagicFake folder



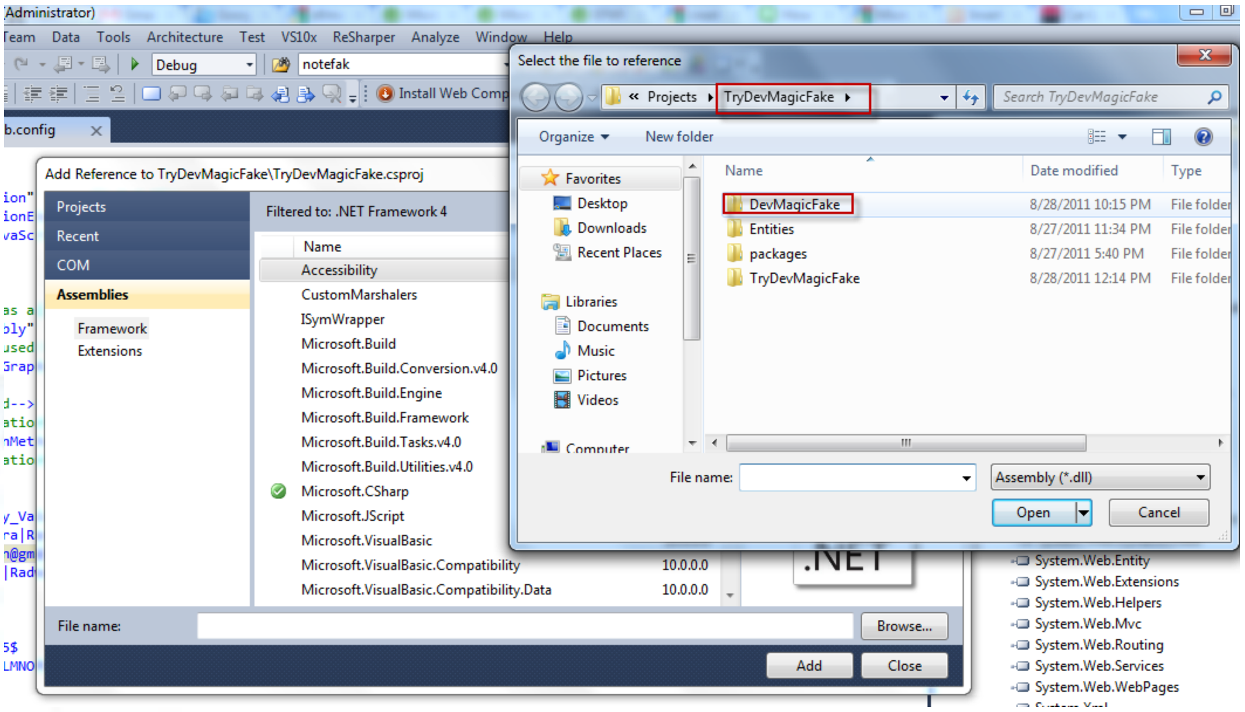
Name	Date modified	Type	Size
DevMagicFake	8/28/2011 10:15 PM	File folder	
Entities	8/27/2011 11:34 PM	File folder	
packages	8/27/2011 5:40 PM	File folder	
TryDevMagicFake	8/28/2011 12:14 PM	File folder	
TryDevMagicFake.5.1.ReSharper.user	8/27/2011 11:36 PM	Visual Studio Proj...	2 KB
TryDevMagicFake.sln	8/27/2011 11:36 PM	Microsoft Visual S...	2 KB

4.3. Add reference to Dev Magic Fake Framework

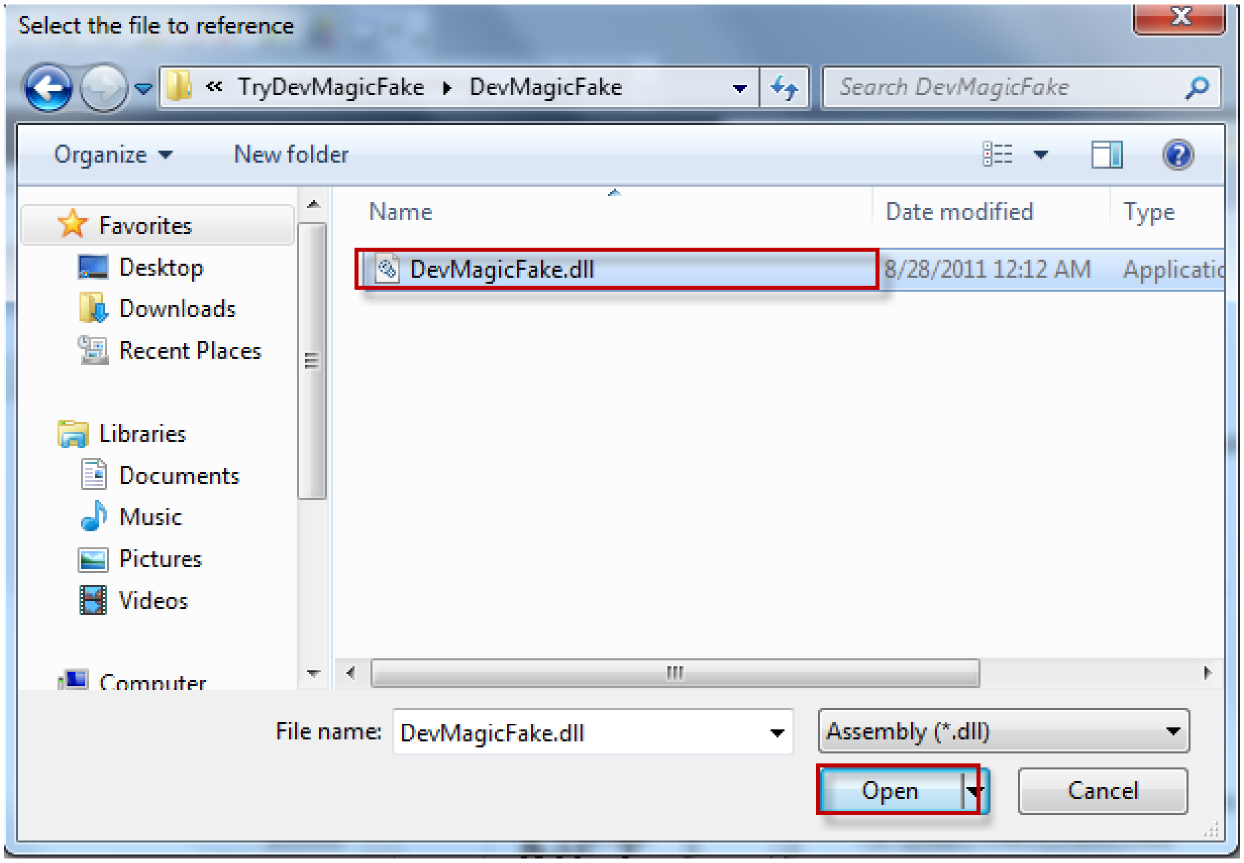
- Right click on the References and select add reference



- Browse to Dev Magic Fake folder



- Select DevMagicFake.dll

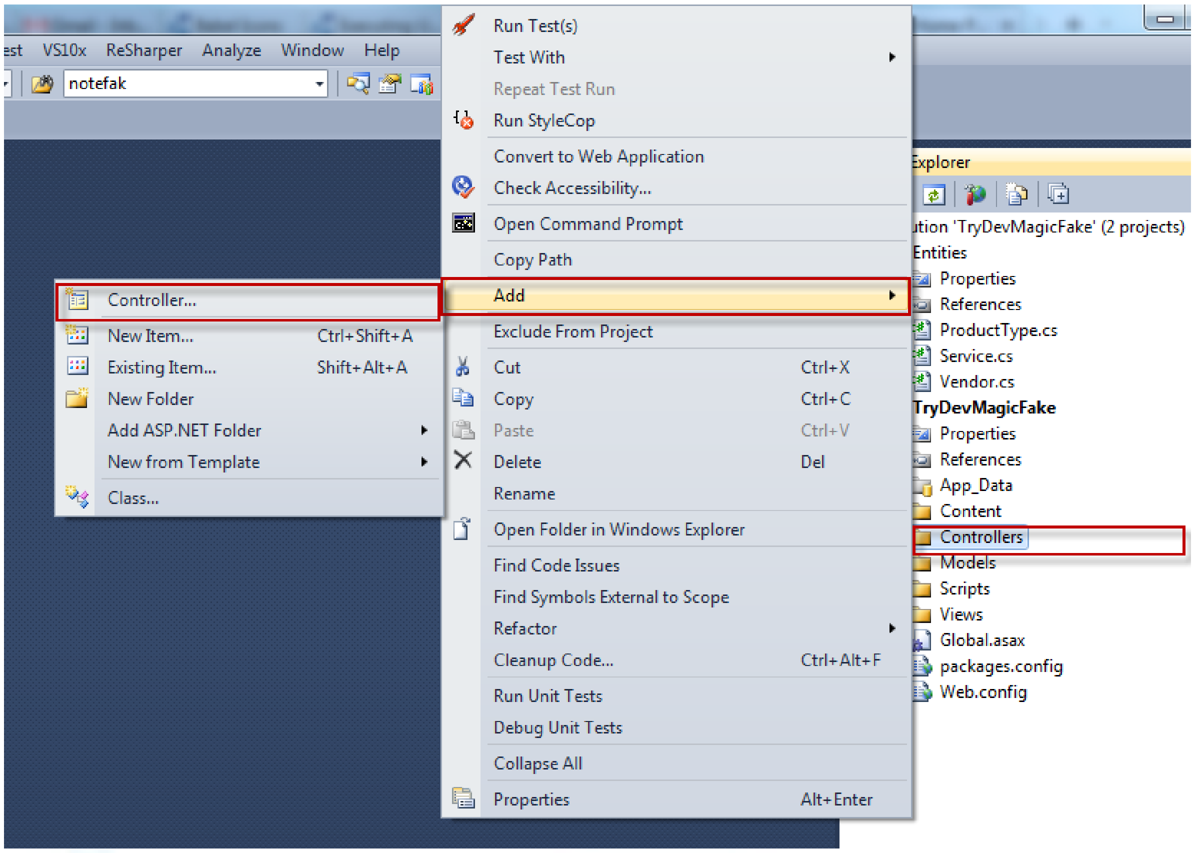


5. Save simple type using Dev Magic Fake

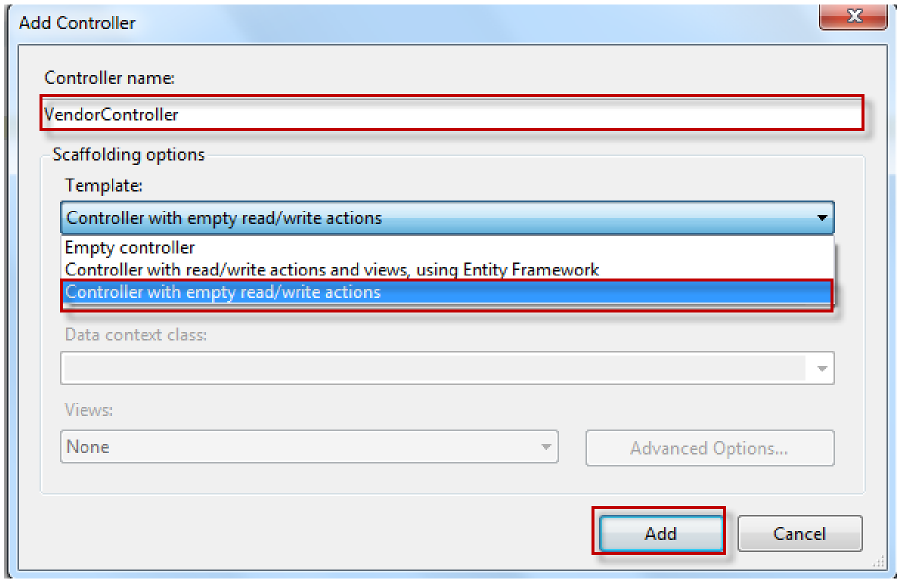
The simple type is the user defines type that has only primitive data types (Integer, String, Double, And Boolean etc.)In this section we will describe how to save simple type, in our scenario this belong to (Vendor)

5.1. Add Vendor Controller

- Right click on the controls folder and click add controller



- Write Vendor, Verify that your settings are as shown below


Add Controller

Controller name: VendorController

Scaffolding options

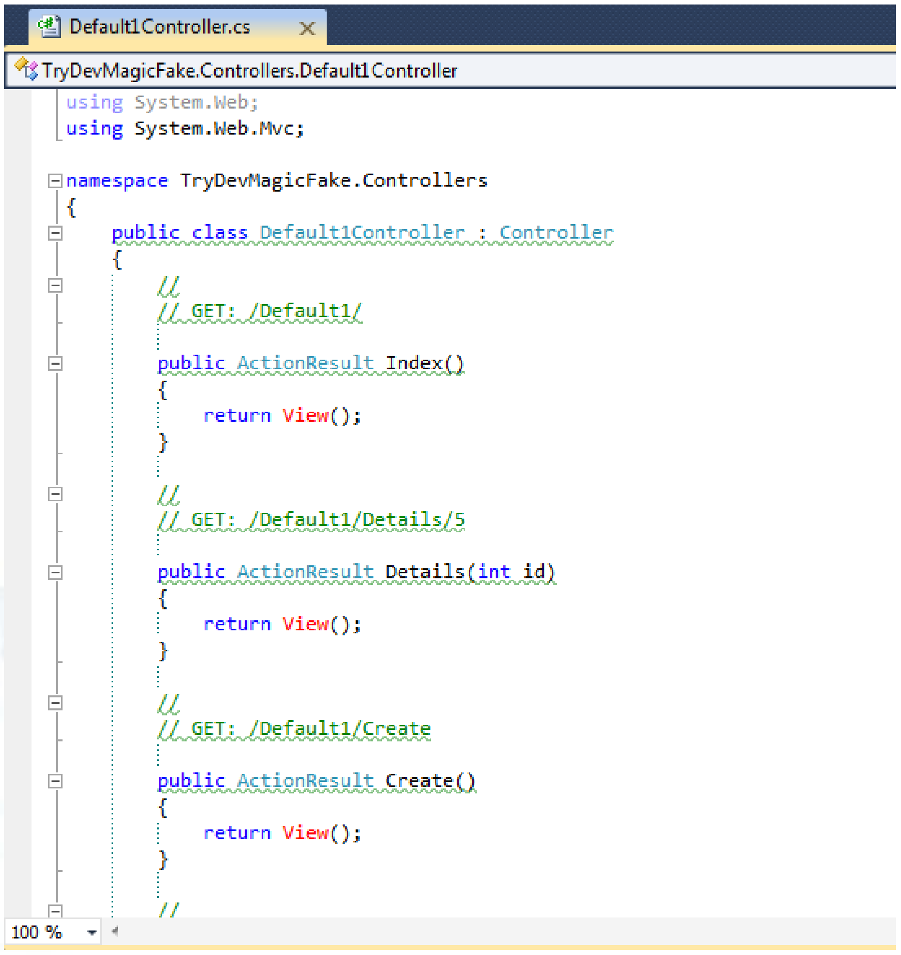
Template: Controller with empty read/write actions

- Empty controller
- Controller with read/write actions and views, using Entity Framework
- Controller with empty read/write actions

Data context class:

Views: None

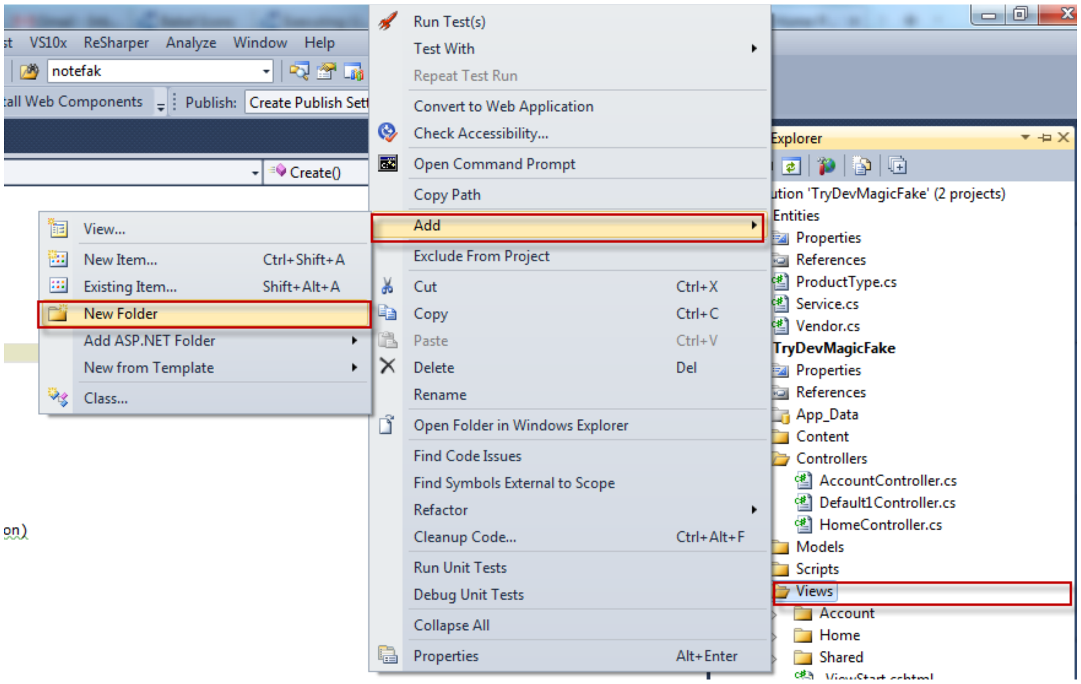
- The vendor control page should now look like this



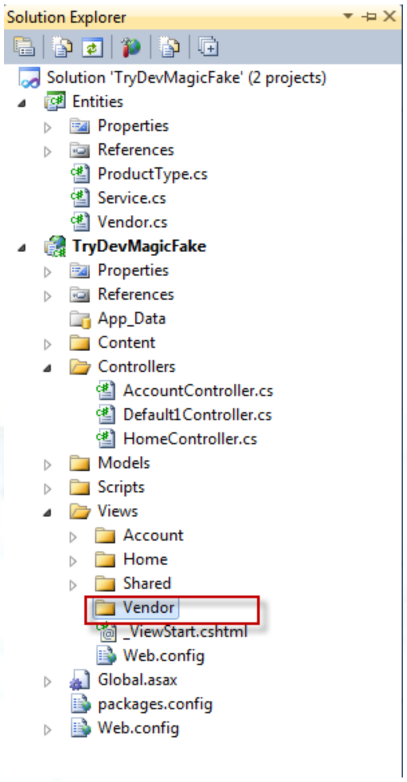
```
using System.Web;
using System.Web.Mvc;

namespace TryDevMagicFake.Controllers
{
    public class Default1Controller : Controller
    {
        //
        // GET: /Default1/

        public ActionResult Index()
        {
            return View();
        }

        //
        // GET: /Default1/Details/5

        public ActionResult Details(int id)
        {
            return View();
        }

        //
        // GET: /Default1/Create

        public ActionResult Create()
        {
            return View();
        }

        //
```

5.2. Add Vendor Views

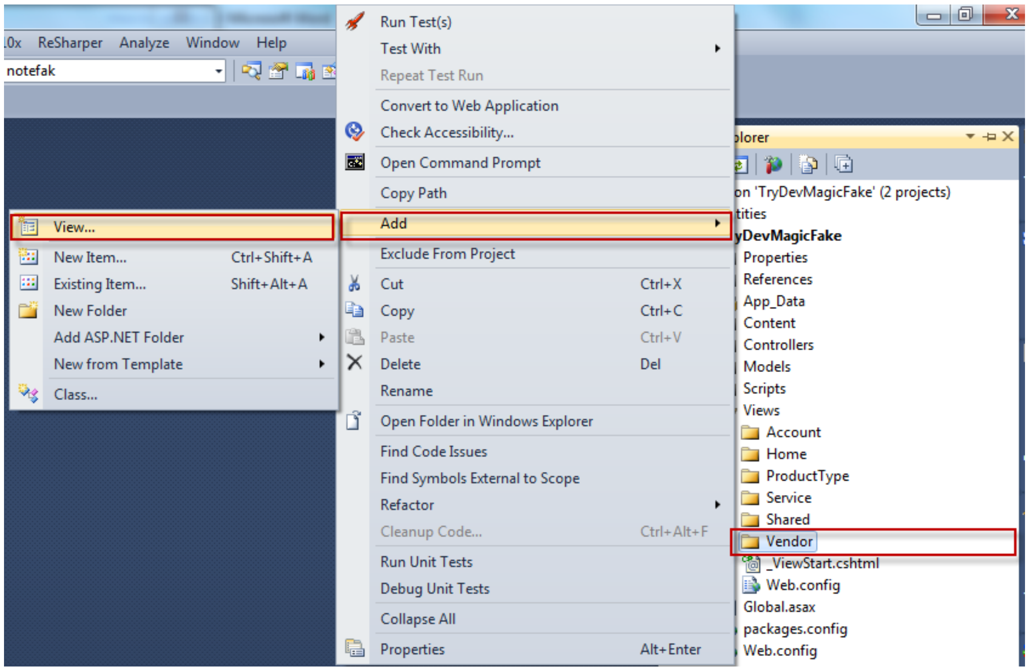
- Right click on the views folder and click add new folder and name it Vendor



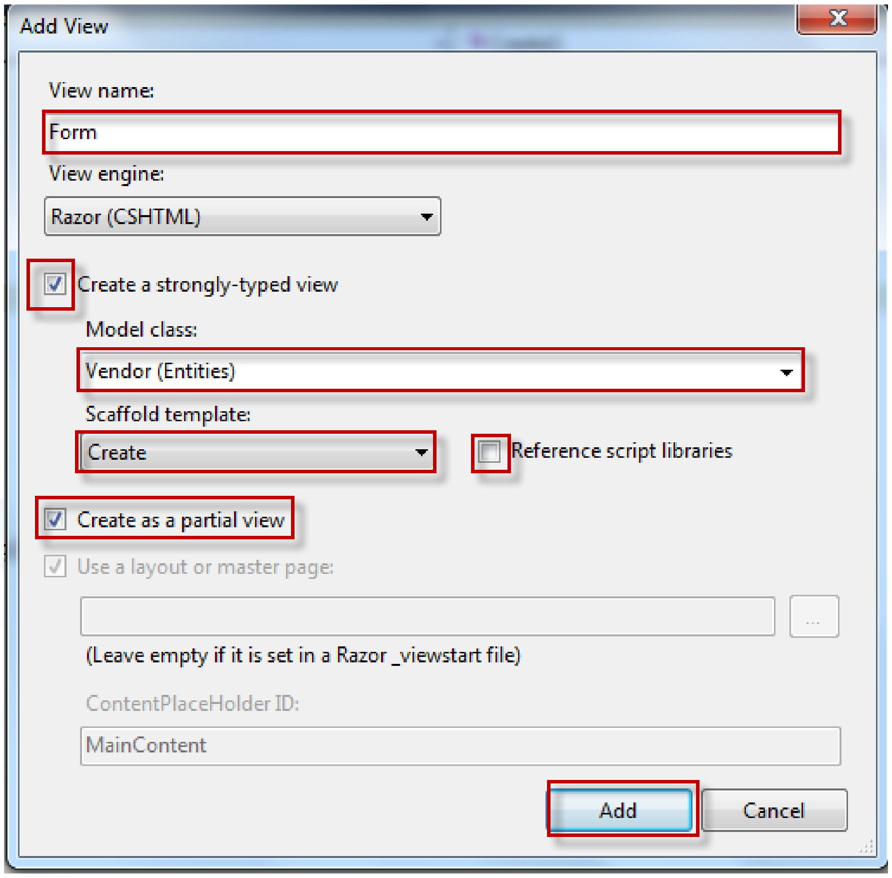
- The solution explorer should now look like this



- Right click on vendor under views folder and select add new view



- Verify that your settings are as shown below


Add View

View name:
Form

View engine:
Razor (CSHTML)

[✓] Create a strongly-typed view

Model class:
Vendor (Entities)

Scaffold template:
Create

[ ] Reference script libraries

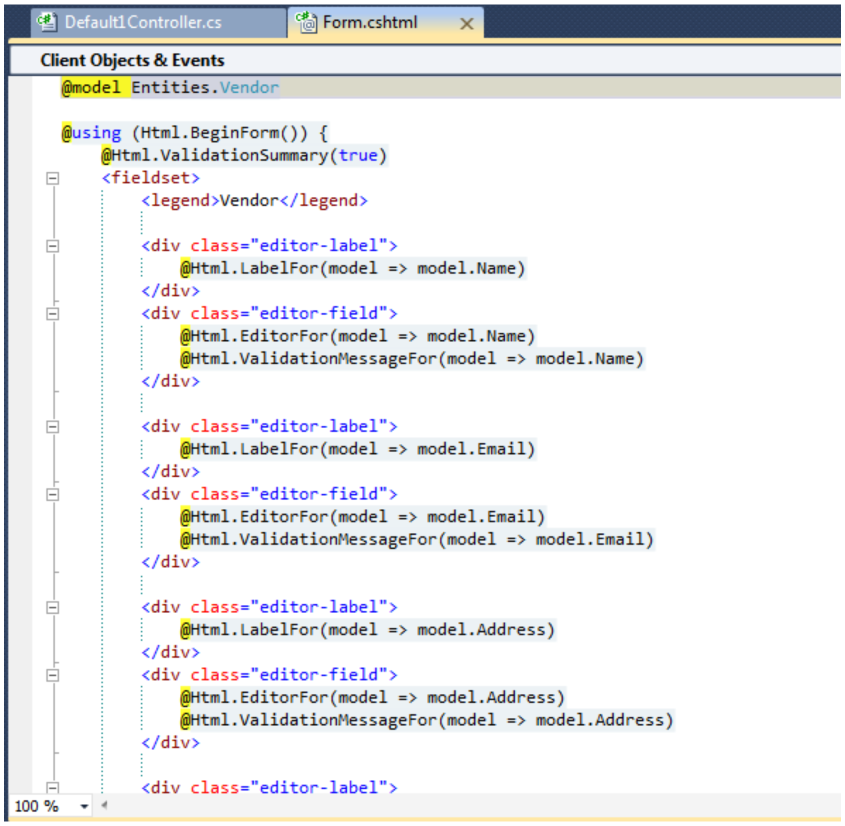
[✓] Create as a partial view

[✓] Use a layout or master page:

(Leave empty if it is set in a Razor _viewstart file)

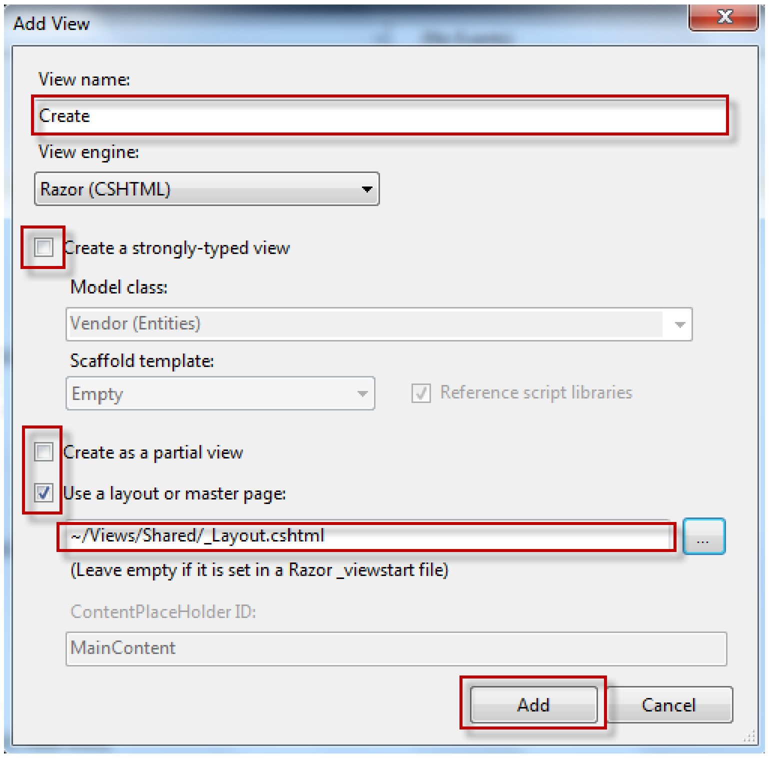
ContentPlaceHolder ID:
MainContent

- The Vendor Form view page will be like the following:



```
@model Entities.Vendor

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Vendor</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Email)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Email)
            @Html.ValidationMessageFor(model => model.Email)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Address)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Address)
            @Html.ValidationMessageFor(model => model.Address)
        </div>

        <div class="editor-label">
```

- Create a new view (Create), Verify that your settings are as shown below



View name: Create

View engine: Razor (CSHTML)

☐ Create a strongly-typed view

Model class: Vendor (Entities)

Scaffold template: Empty ☑ Reference script libraries

☐ Create as a partial view

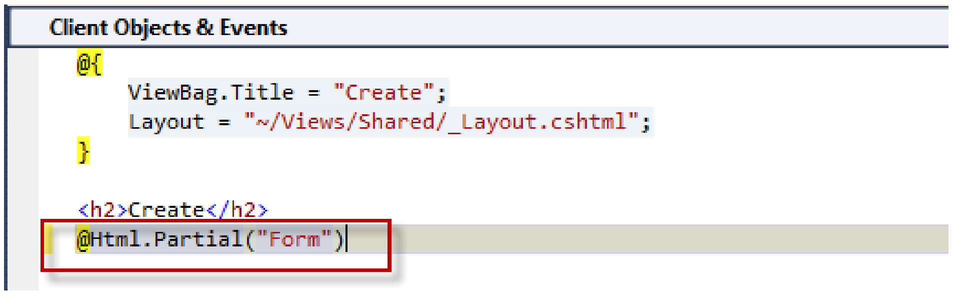
☑ Use a layout or master page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID: MainContent

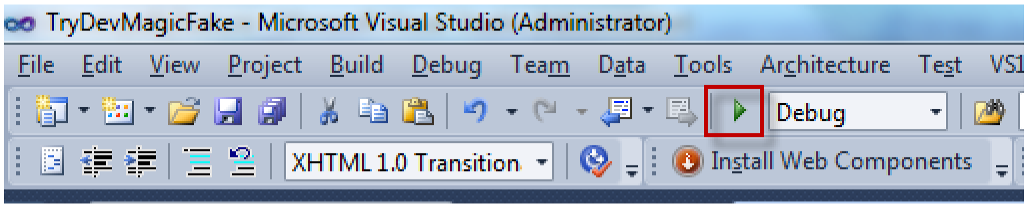
- Load partial view for the form view as the following:



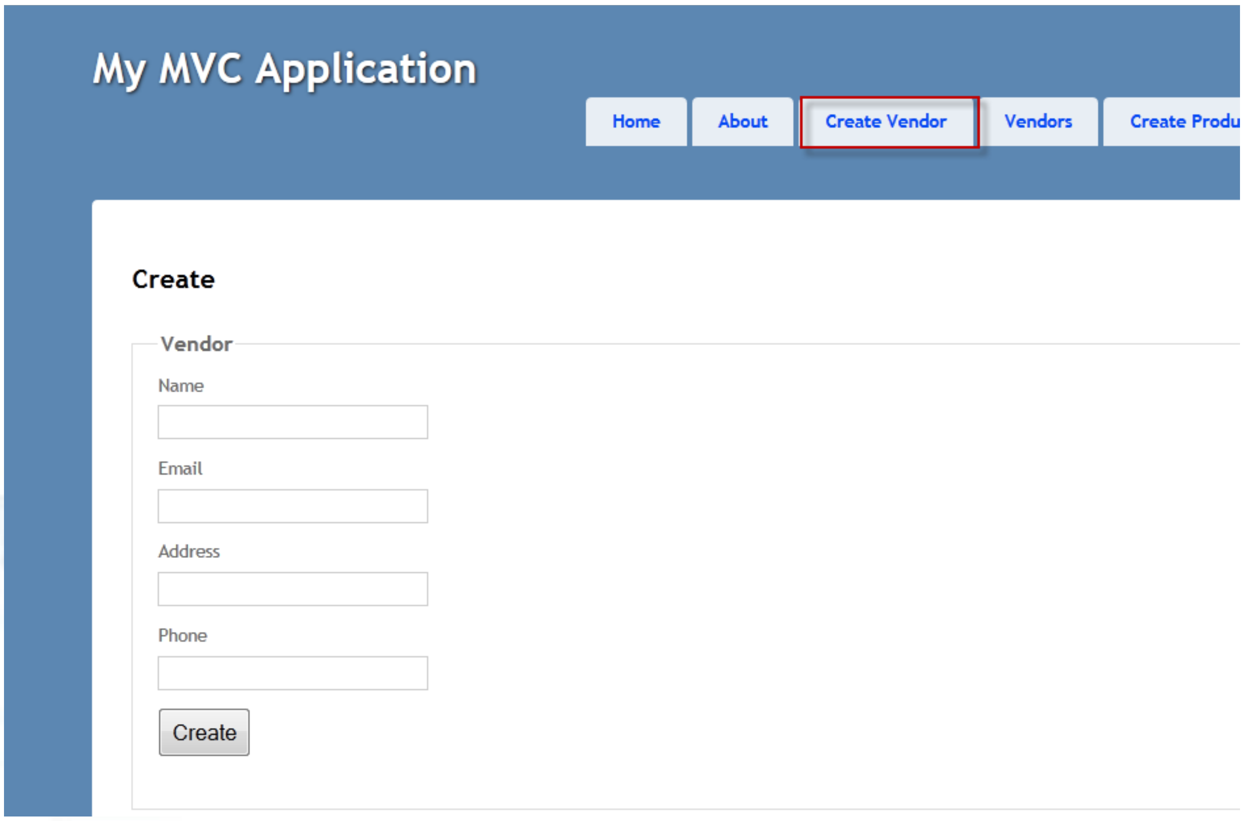
Client Objects & Events

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

<h2>Create</h2>
@Html.Partial("Form")
```

Run the application


TryDevMagicFake - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test VS1
Debug
XHTML 1.0 Transition
Install Web Components

- The page should now look like this



My MVC Application

Home | About | Create Vendor | Vendors | Create Produ

Create

Vendor

Name

Email

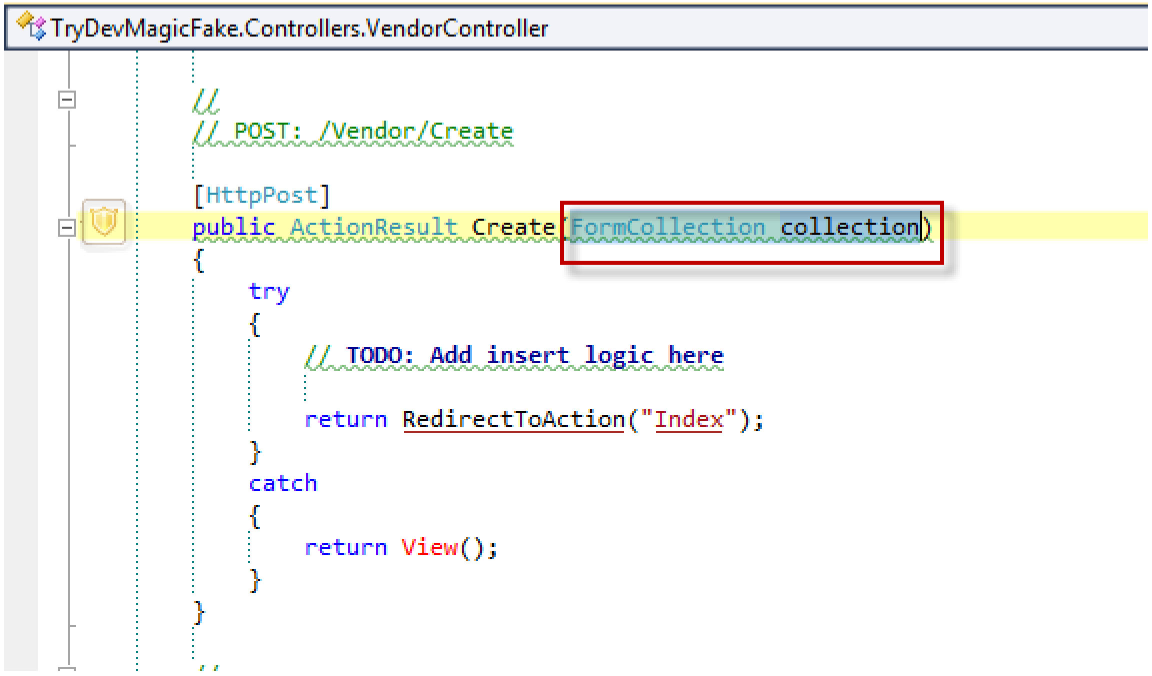
Address

Phone

Create

5.3. Use Dev Magic Fake in the Vendor Controller action methods

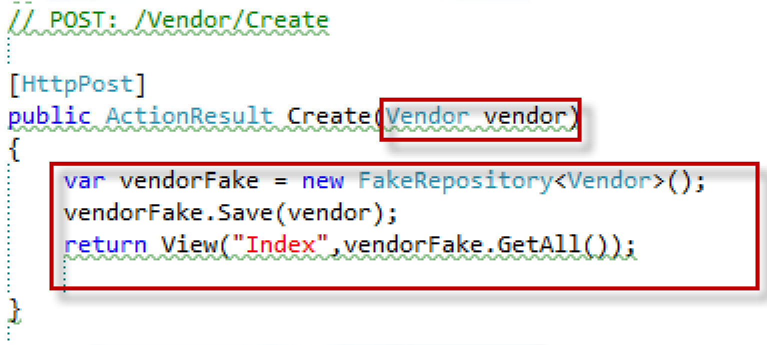
- Add the content of the Create (Post) action method, we will remove the FormCollection



TryDevMagicFake.Controllers.VendorController

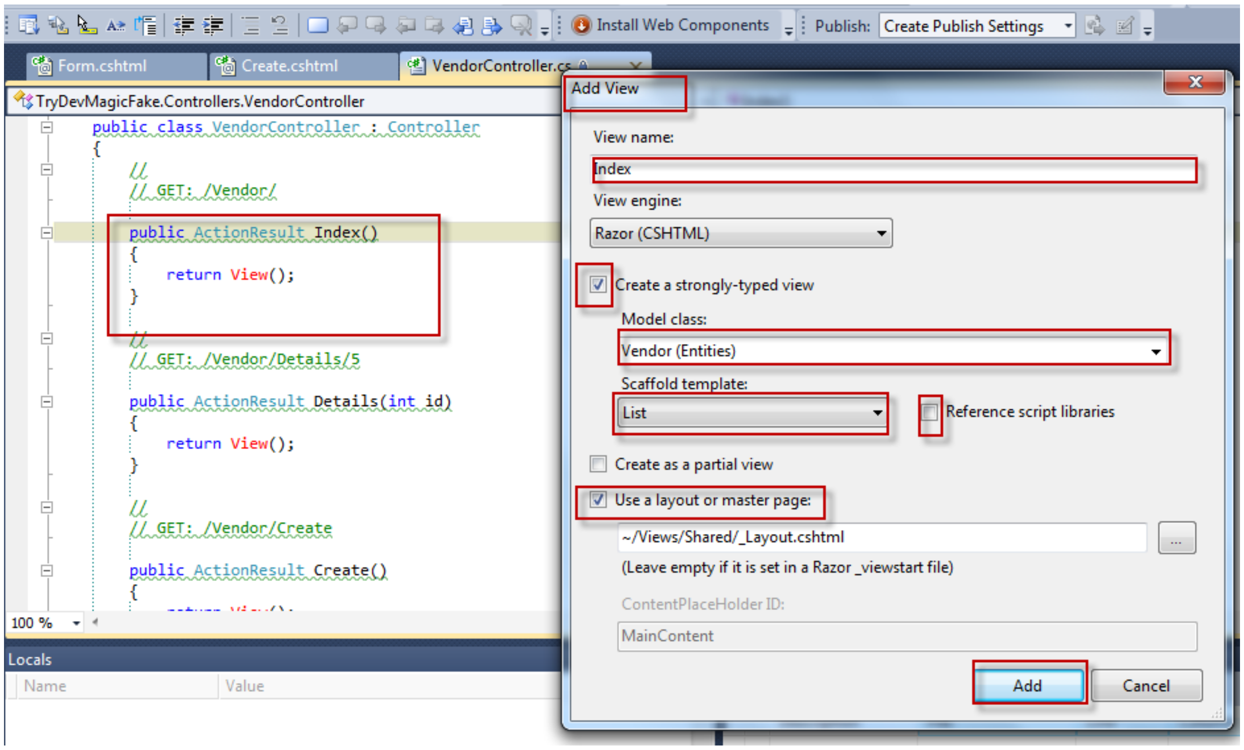
```
//
// POST: /Vendor/Create

[HttpPost]
public ActionResult Create(FormCollection collection)
{
    try
    {
        // TODO: Add insert logic here

        return RedirectToAction("Index");
    }
    catch
    {
        return View();
    }
}
```

- Add the vendor as a parameter and call the save method of the Fake Repository, we can see hear in this instance we create FakeRepository of one type because we want to save simple type



```
// POST: /Vendor/Create

[HttpPost]
public ActionResult Create(Vendor vendor)
{
    var vendorFake = new FakeRepository<Vendor>();
    vendorFake.Save(vendor);
    return View("Index",vendorFake.GetAll());

}
```

- Add an index view for vendor as the following:



```
TryDevMagicFake.Controllers.VendorController
public class VendorController : Controller
{
    //
    // GET: /Vendor/

    public ActionResult Index()
    {
        return View();
    }

    //
    // GET: /Vendor/Details/5

    public ActionResult Details(int id)
    {
        return View();
    }

    //
    // GET: /Vendor/Create

    public ActionResult Create()
    {
```

Add View

View name: Index

View engine: Razor (CSHTML)

[✓] Create a strongly-typed view

Model class: Vendor (Entities)

Scaffold template: List

[ ] Reference script libraries

[ ] Create as a partial view

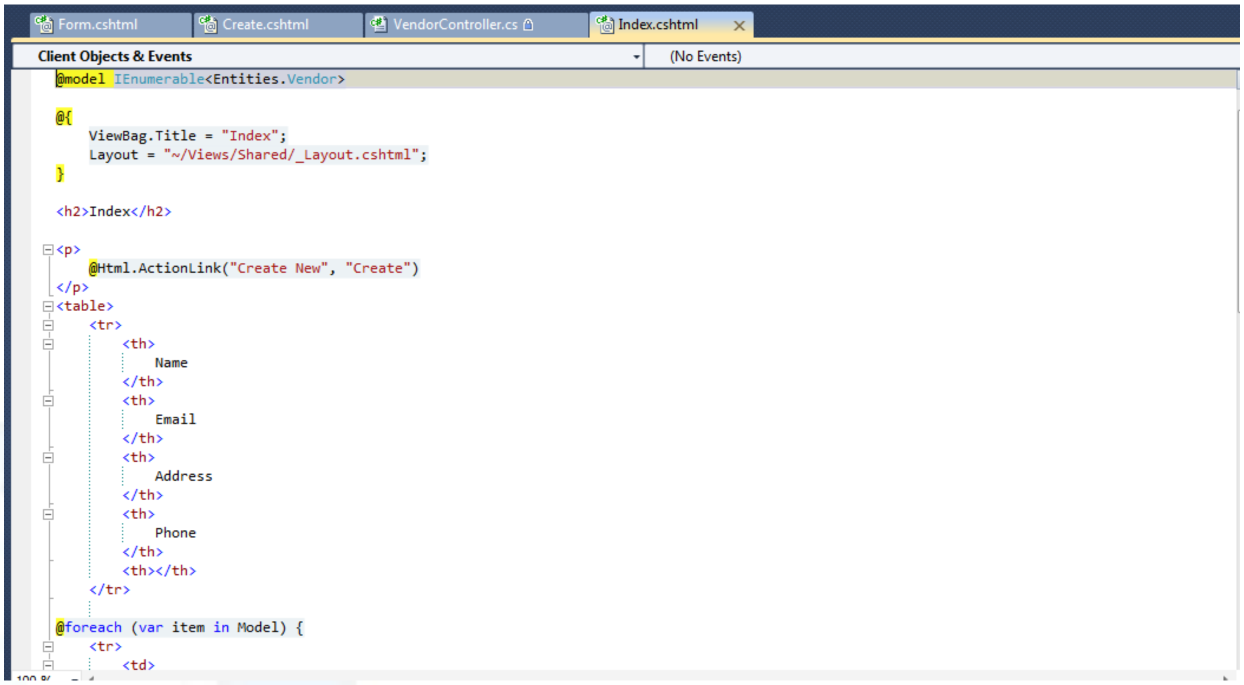
[✓] Use a layout or master page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

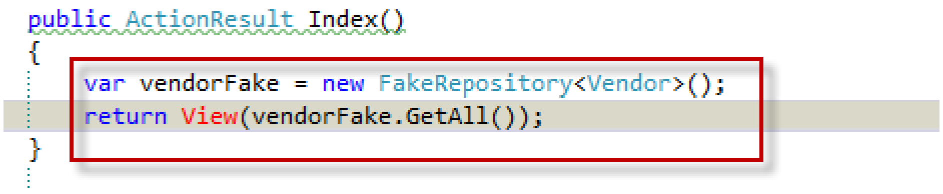
ContentPlaceHolder ID: MainContent

- The index page should now look like this



```
@model IEnumerable<Entities.Vendor>

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

<h2>Index</h2>

<p>
    @Html.ActionLink("Create New", "Create")
</p>
<table>
    <tr>
        <th>
            Name
        </th>
        <th>
            Email
        </th>
        <th>
            Address
        </th>
        <th>
            Phone
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
```

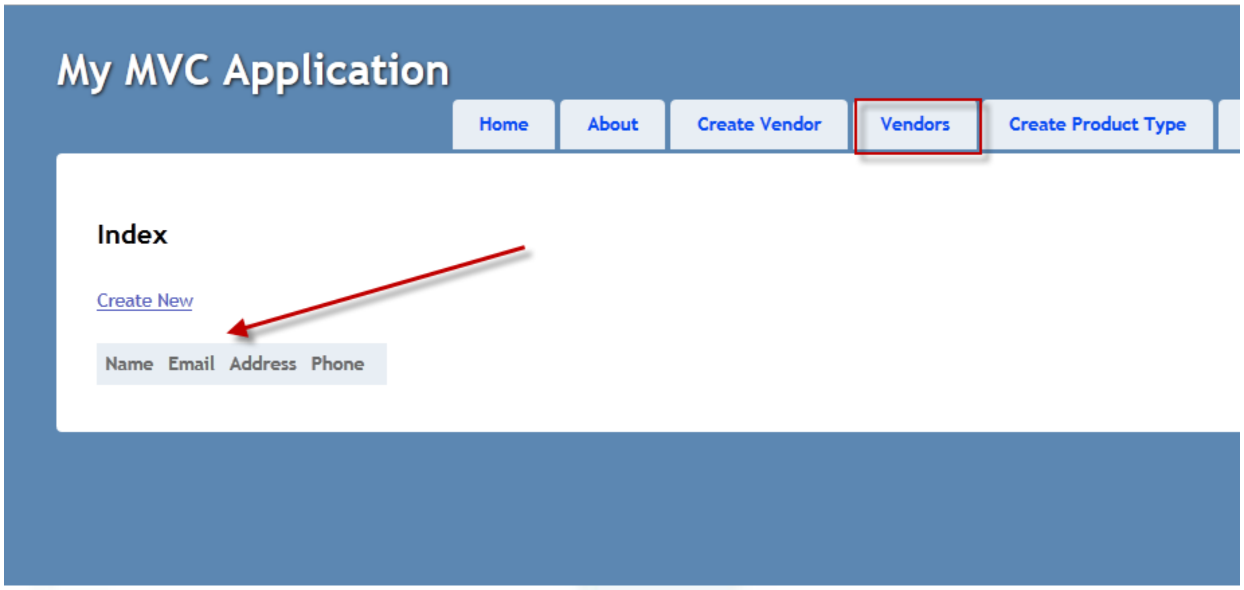
- Add the following code to the index action method so to display all the vendors when we open the index page



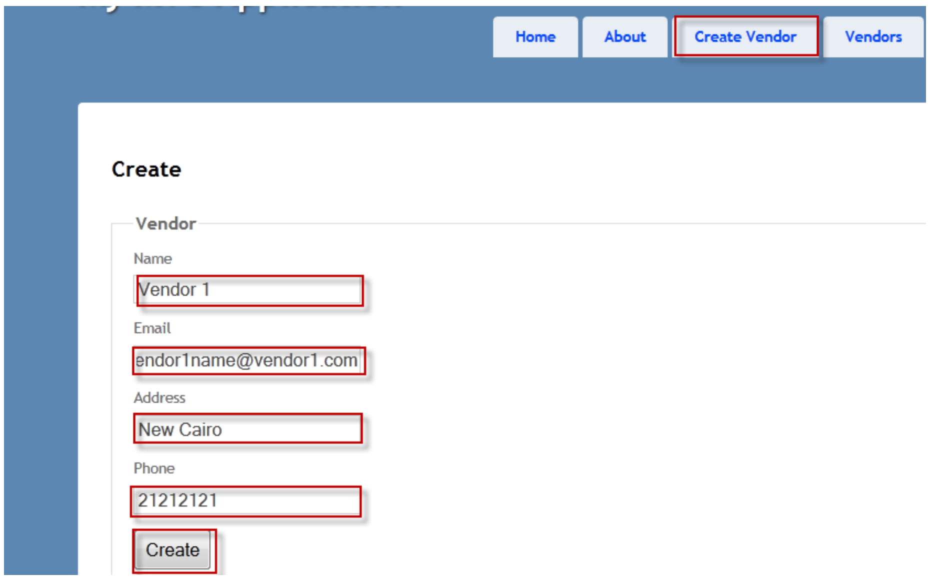
```
public ActionResult Index()
{
    var vendorFake = new FakeRepository<Vendor>();
    return View(vendorFake.GetAll());
}
```

5.4. Examine the result of Dev Magic Fake

- Run the application and open the Vendor's page, we will not find any displayed items because there is no vendor added



- Open the Create Vendor page and start adding the first Vendor data



Create

Vendor

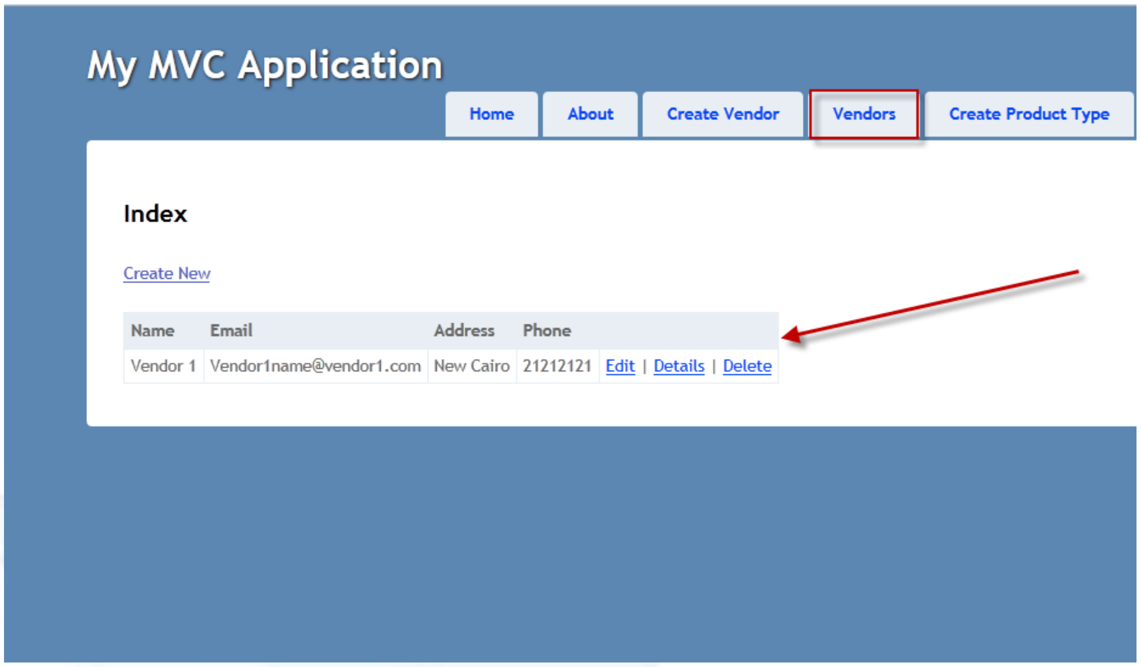
Name: Vendor 1

Email: endor1name@vendor1.com

Address: New Cairo

Phone: 21212121

- Open the Vendors page
- We should see the page display the created vendor, try to add many vendors and see how this reflect to the Vendors page



Index

Create New

Name	Email	Address	Phone	
Vendor 1	Vendor1name@vendor1.com	New Cairo	21212121	Edit | Details | Delete

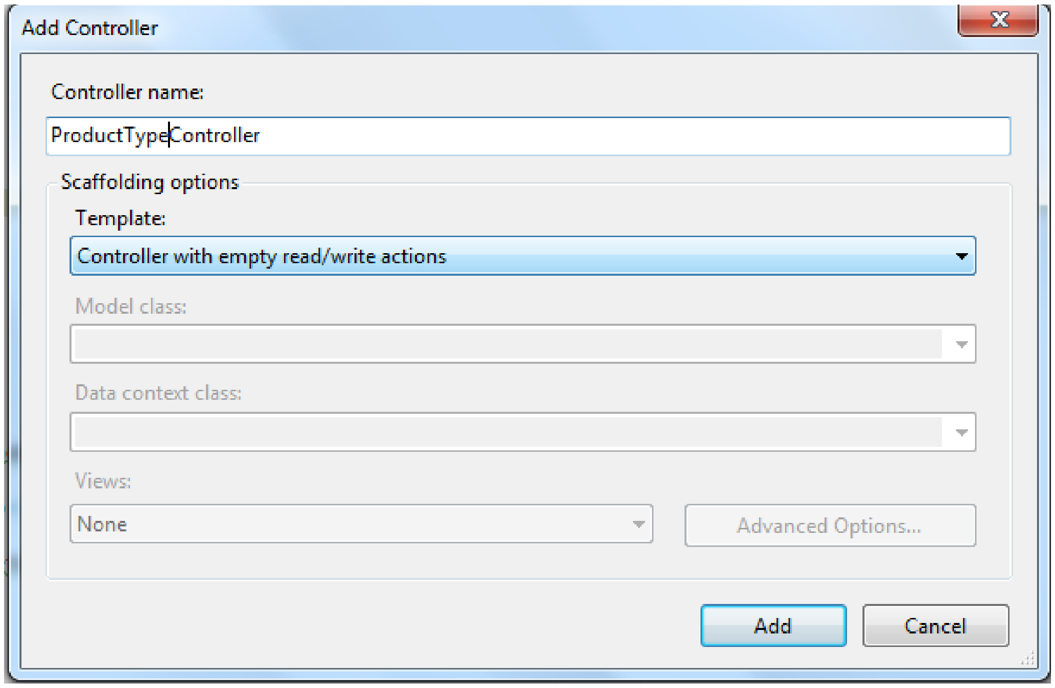
After we saw how to save simple type, in the next section we will see how to save complex object like product type and service

6. Save complex type that has one sub type using Dev Magic Fake

The complex type means it has properties of another user define types, in this section we will describe how to save complex type that has only one user define type, in our scenario this belong to (Product Type)

6.1. Add Product Type Controller

- Add the Product Type controller as the following:

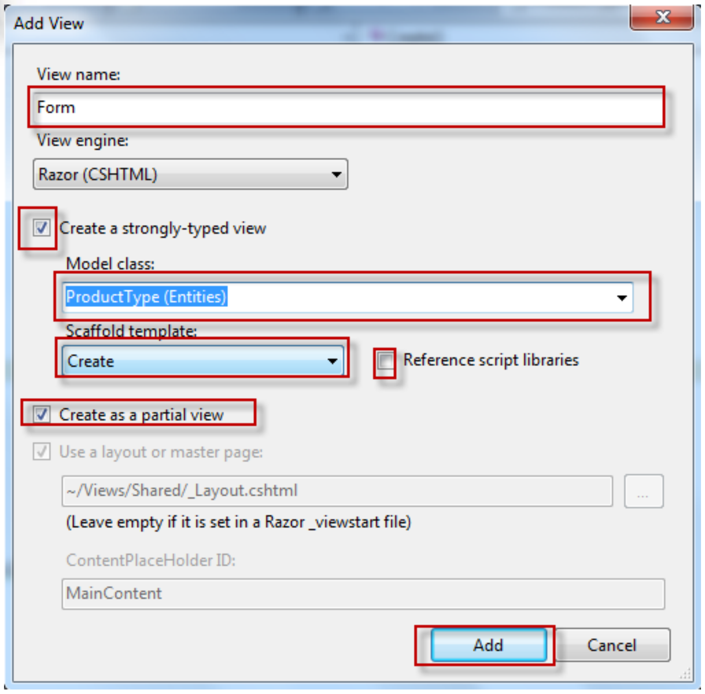


ProductTypeController

Controller with empty read/write actions

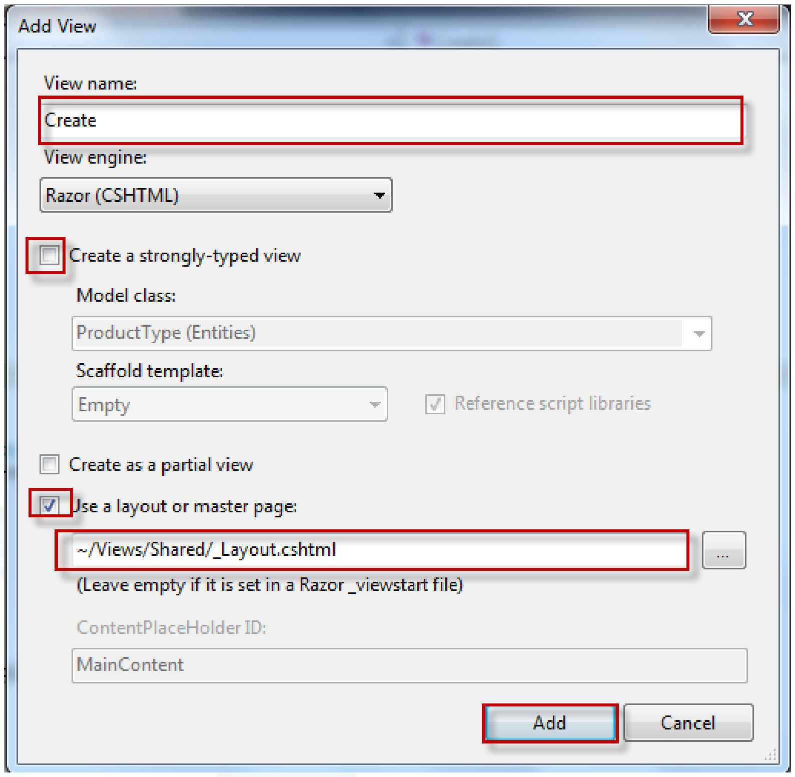
6.2. Add Product Type Views

- Add a view (Form) based on strong type as the following:


Add View

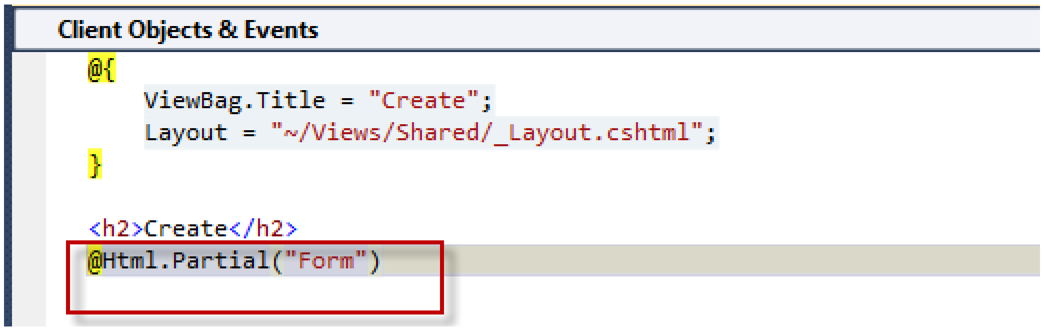
View name:
Form
View engine:
Razor (CSHTML)
☑ Create a strongly-typed view
Model class:
ProductType (Entities)
Scaffold template:
Create
☐ Reference script libraries
☑ Create as a partial view
☑ Use a layout or master page:
~/Views/Shared/_Layout.cshtml
(Leave empty if it is set in a Razor _viewstart file)
ContentPlaceHolder ID:
MainContent

- Create a new view (Create), Verify that your settings are as shown below


Add View

View name:
Create
View engine:
Razor (CSHTML)
☐ Create a strongly-typed view
Model class:
ProductType (Entities)
Scaffold template:
Empty
☑ Reference script libraries
☐ Create as a partial view
☑ Use a layout or master page:
~/Views/Shared/_Layout.cshtml
(Leave empty if it is set in a Razor _viewstart file)
ContentPlaceHolder ID:
MainContent

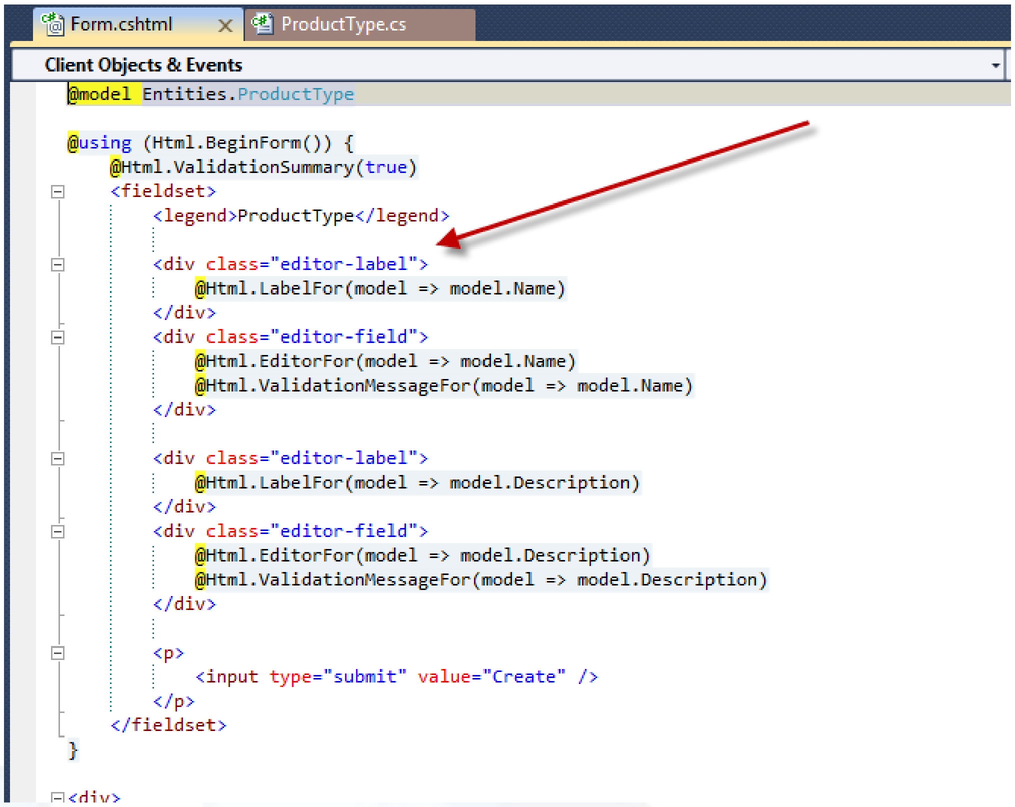
- Load partial view for the form view as the following:



Client Objects & Events

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

<h2>Create</h2>
@Html.Partial("Form")
```

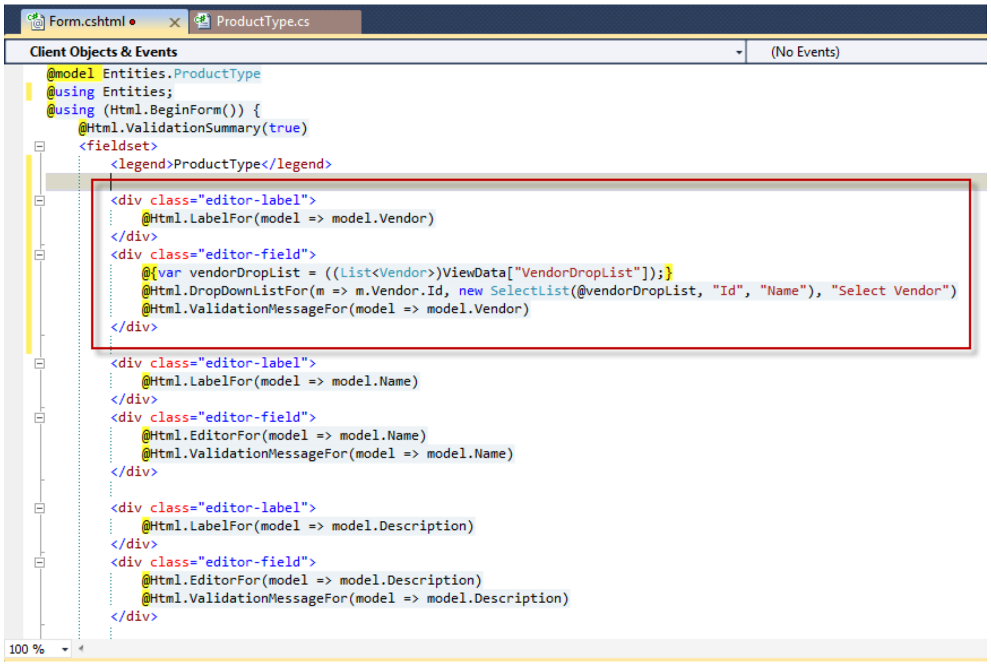
- Open the Form view of the Product Type, and locate the arrow



Client Objects & Events

```
@model Entities.ProductType

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>ProductType</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Description)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Description)
            @Html.ValidationMessageFor(model => model.Description)
        </div>

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

<div>
```

- Add the following code in the previous area in the image; this will display the Vendors drop down list when we open the create product type page



Form.cshtml ProductType.cs

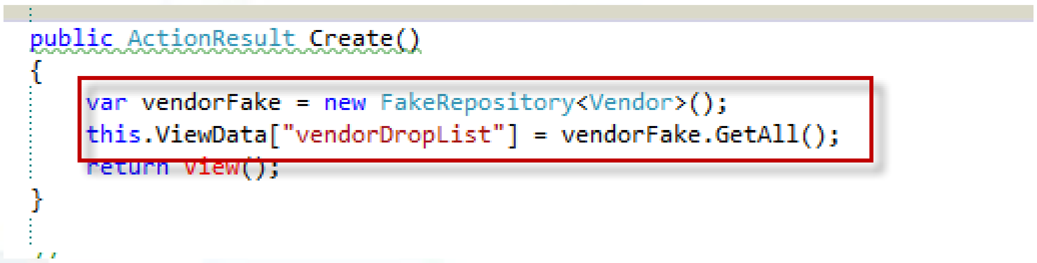
Client Objects & Events (No Events)

```
@model Entities.ProductType
@using Entities;
@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>ProductType</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Vendor)
        </div>
        <div class="editor-field">
            @{var vendorDropList = ((List<Vendor>)ViewData["VendorDropList"]);}
            @Html.DropDownListFor(m => m.Vendor.Id, new SelectList(@vendorDropList, "Id", "Name"), "Select Vendor")
            @Html.ValidationMessageFor(model => model.Vendor)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Description)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Description)
            @Html.ValidationMessageFor(model => model.Description)
        </div>
```

100 %

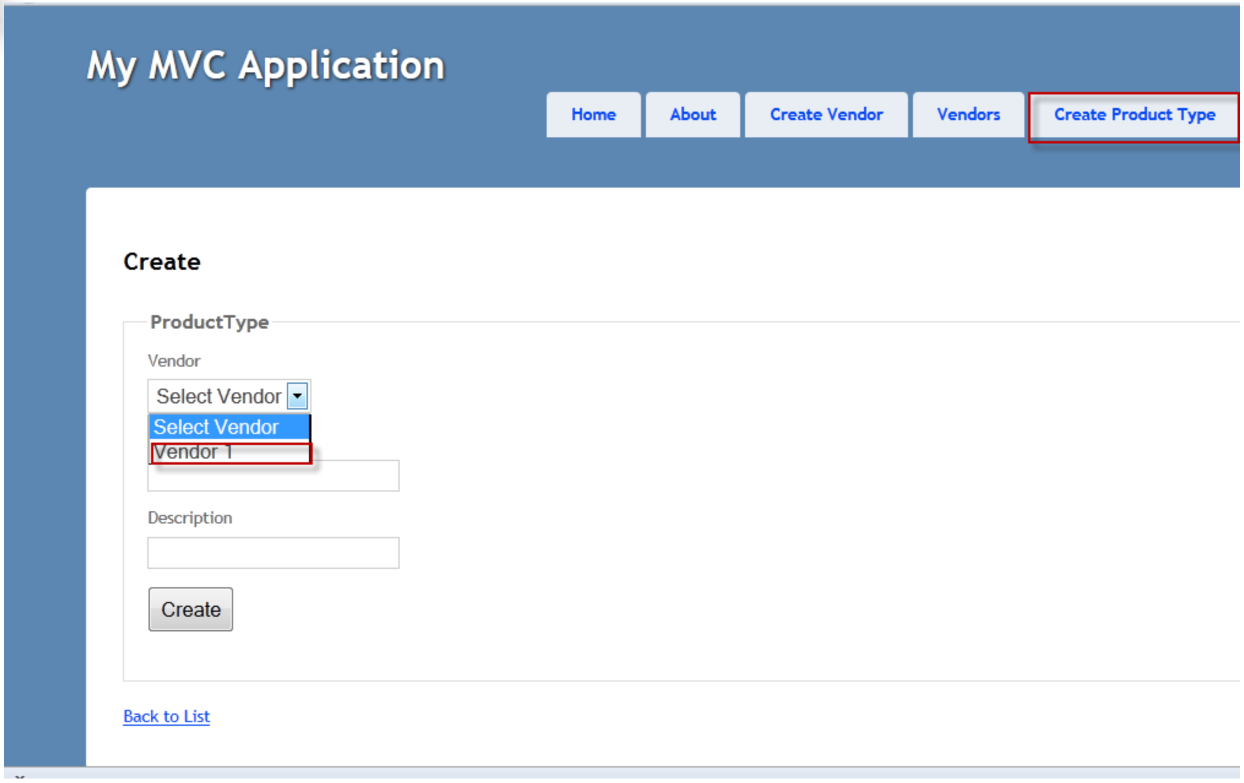
6.3. Use Dev Magic Fake in the Product type Controller action methods

- Add the content of the Create (Get) action method of the Product Type

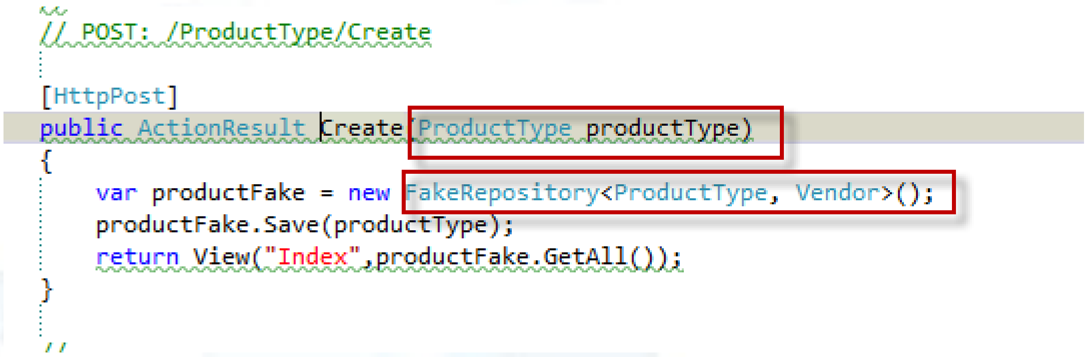


```
public ActionResult Create()
{
    var vendorFake = new FakeRepository<Vendor>();
    this.ViewData["vendorDropList"] = vendorFake.GetAll();
    return View();
}
```

Run the application, open create product type page, we will find the vendor drop down list display all vendors, if there is no vendors, we can go to create vendor and add vendor as need and open create Product Type again and we will see the list filled with the vendors added

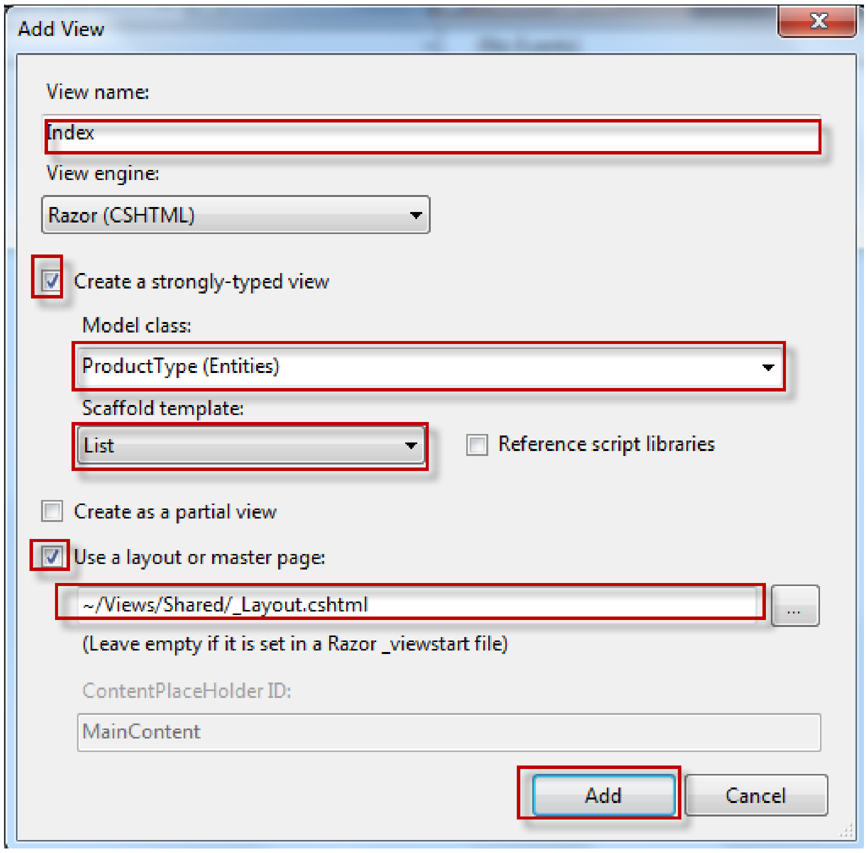


- Add the content of the Create (Post) action method of the Product Type, we can see hear in this instance we create FakeRepository of two types the main and the sub, this because we want to save complex object, the sub (Vendor) will only has id and name because the drop down list only has this values (Vendor drop down)



```
// POST: /ProductType/Create

[HttpPost]
public ActionResult Create(ProductType productType)
{
    var productFake = new FakeRepository<ProductType, Vendor>();
    productFake.Save(productType);
    return View("Index",productFake.GetAll());
}
```

- Add an index view for Product Type as the following:


Add View

View name:

Index

View engine:

Razor (CSHTML)

[✓] Create a strongly-typed view

Model class:

ProductType (Entities)

Scaffold template:

List

[ ] Reference script libraries

[ ] Create as a partial view

[✓] Use a layout or master page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:

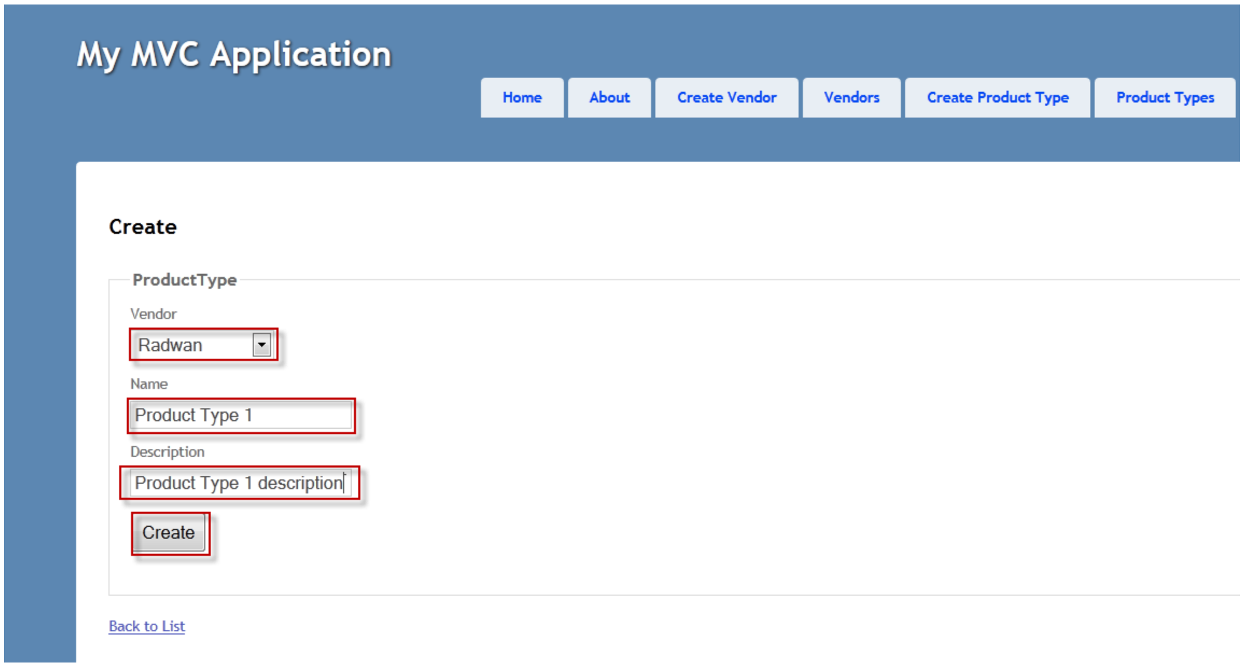
MainContent

- Add the following code to the index action method to display all the Product Type when we open the product type index page

```
public ActionResult Index()
{
    var products = new FakeRepository<ProductType>();
    return View(products.GetAll());
}
```

6.4. Examine the result of Dev Magic Fake

- Run the application and open create product type and see how the vendor list is loaded.



Create

ProductType

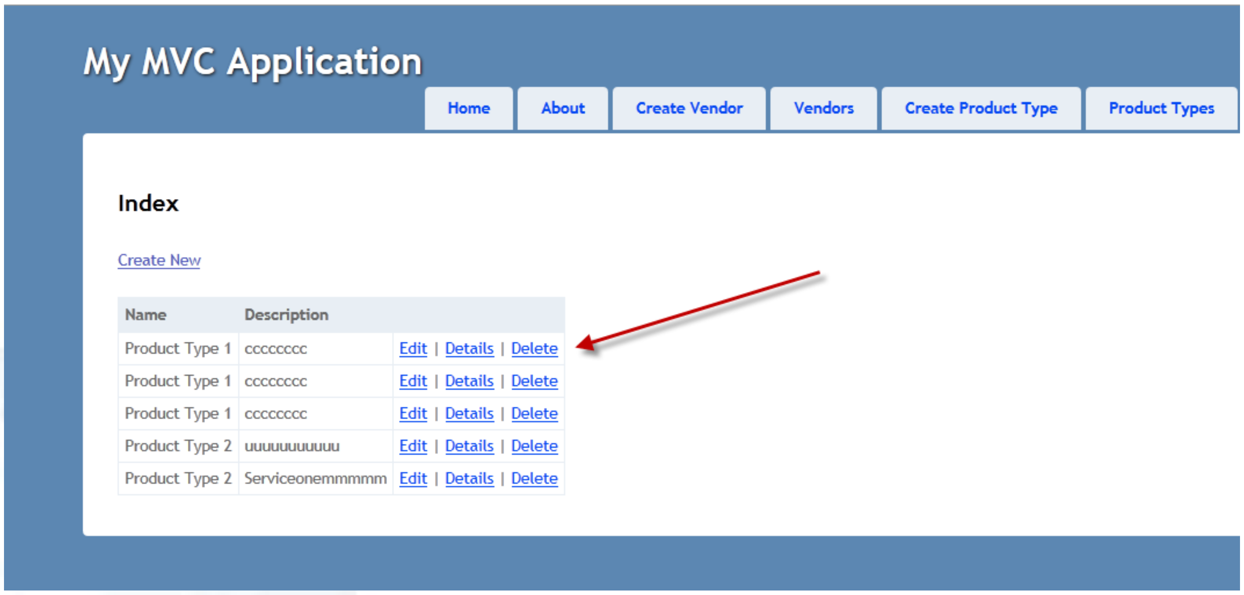
Vendor: Radwan

Name: Product Type 1

Description: Product Type 1 description

Create

- Create some product types and open the product types page, we will find that all the product types that has been added are displayed



Index

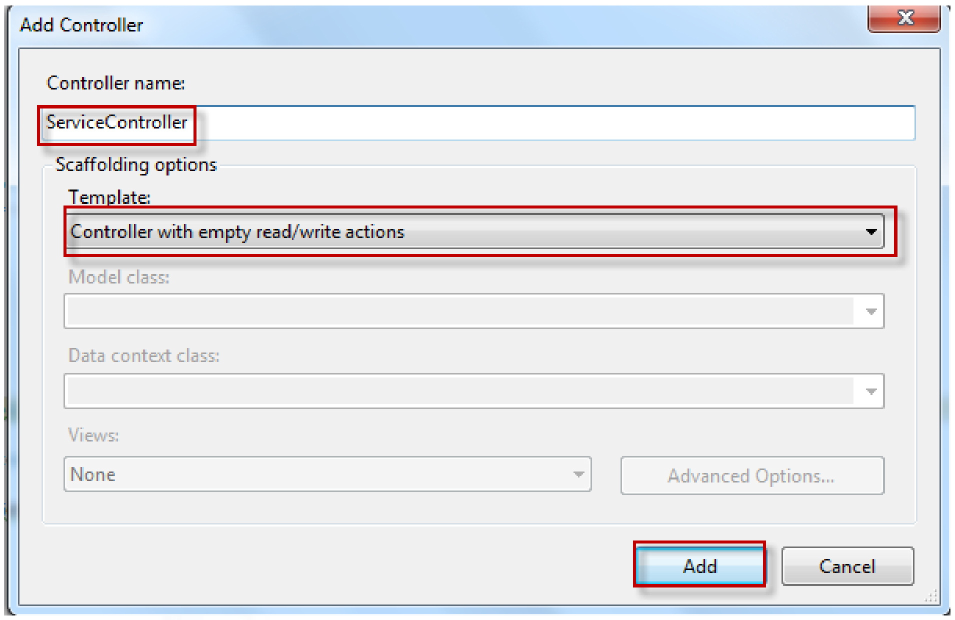
Name	Description	
Product Type 1	cccccccc	Edit | Details | Delete
Product Type 1	cccccccc	Edit | Details | Delete
Product Type 1	cccccccc	Edit | Details | Delete
Product Type 2	uuuuuuuuuuu	Edit | Details | Delete
Product Type 2	Serviceonemmmmm	Edit | Details | Delete

7. Save complex type that has two sub types using Dev Magic Fake

The complex type means it has properties of another user define types In this section we will describe how to save complex type that has two properties of user define type, in our scenario this belong to (Service)

7.1. Add Service Controller

- Add the Service controller as the following:



Add Controller

Controller name: ServiceController

Scaffolding options

Template: Controller with empty read/write actions

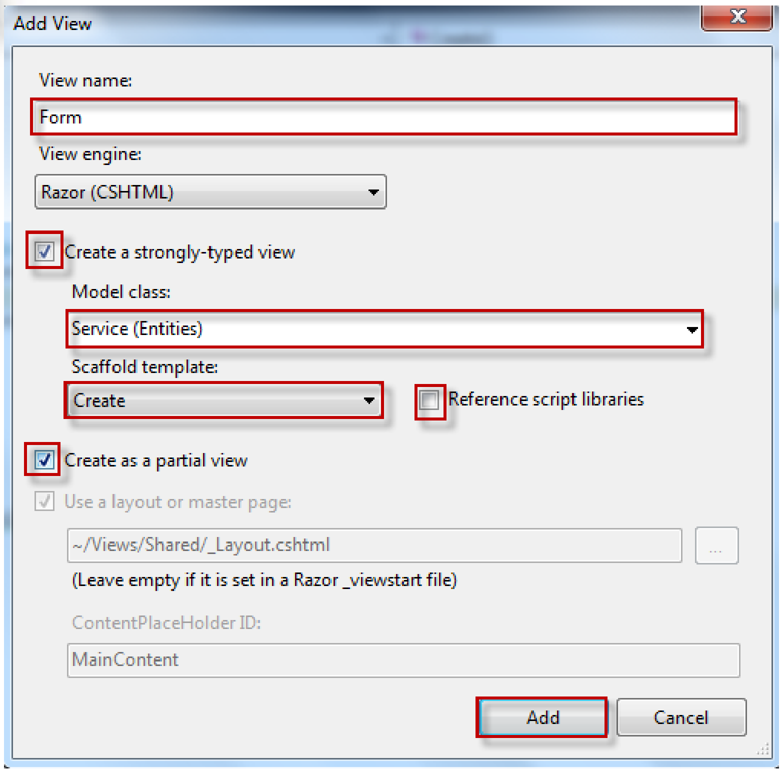
Model class:

Data context class:

Views: None

7.2. Add Service Views

- Add a view (Form) based on strong type as the following:


Add View

View name: Form

View engine: Razor (CSHTML)

☑ Create a strongly-typed view

Model class: Service (Entities)

Scaffold template: Create ☐ Reference script libraries

☑ Create as a partial view

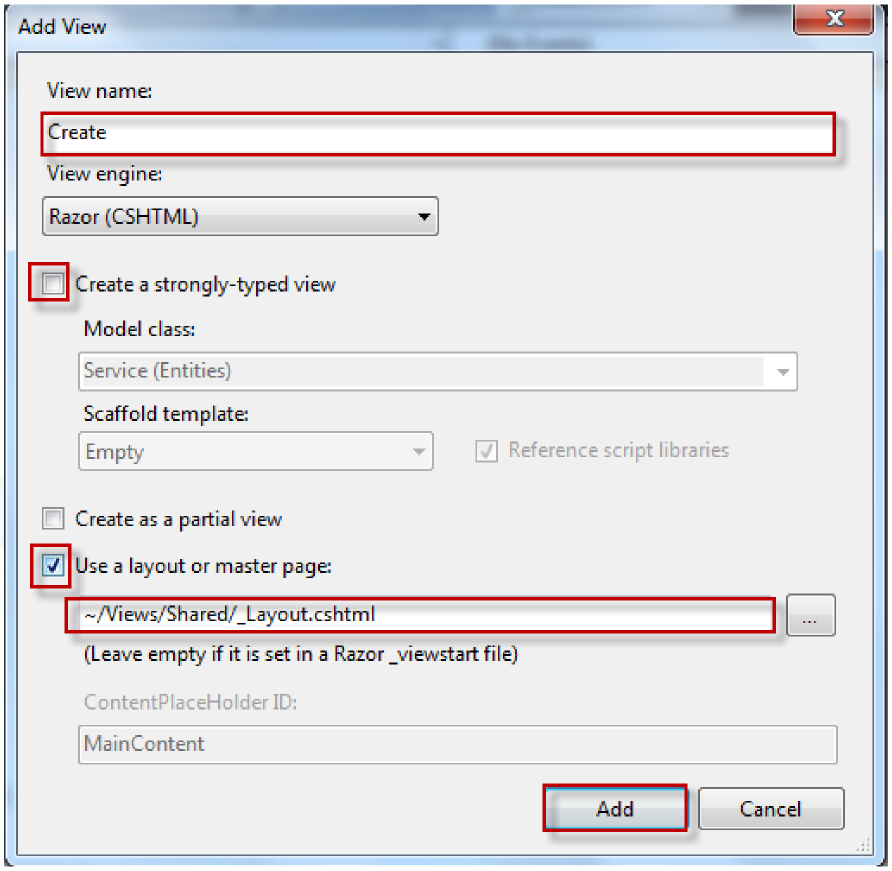
☑ Use a layout or master page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID: MainContent

- Create a new view (Create), Verify that your settings are as shown below


Add View

View name: Create

View engine: Razor (CSHTML)

☐ Create a strongly-typed view

Model class: Service (Entities)

Scaffold template: Empty ☑ Reference script libraries

☐ Create as a partial view

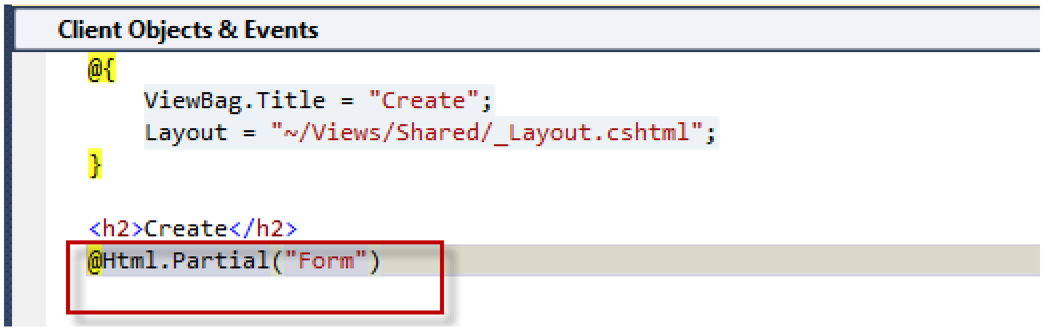
☑ Use a layout or master page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID: MainContent

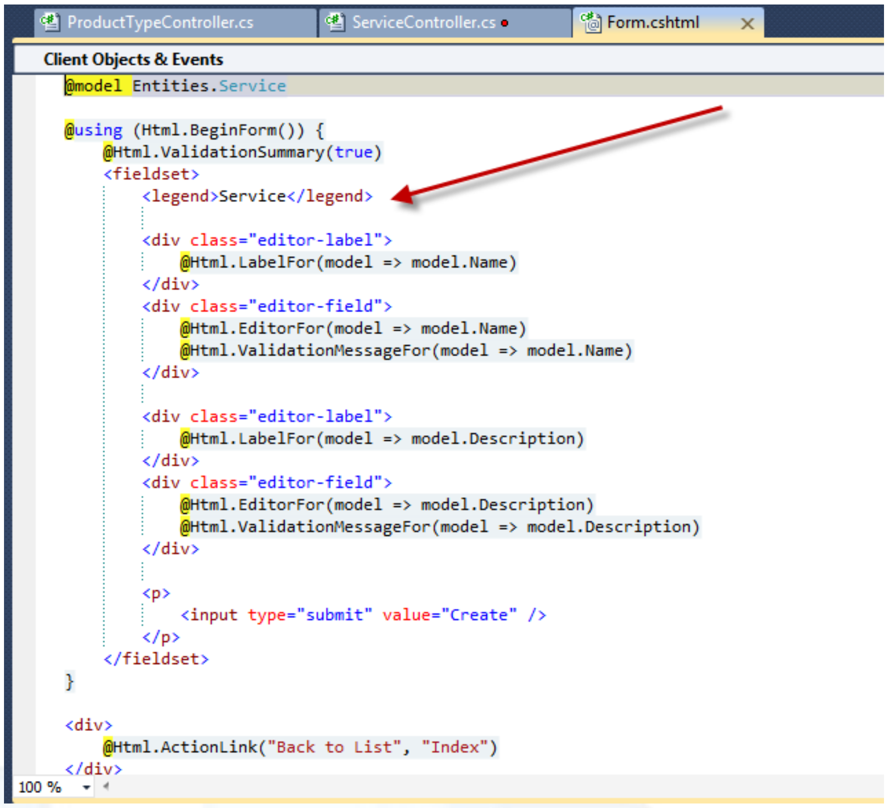
- Load partial view for the form view as the following:



Client Objects & Events

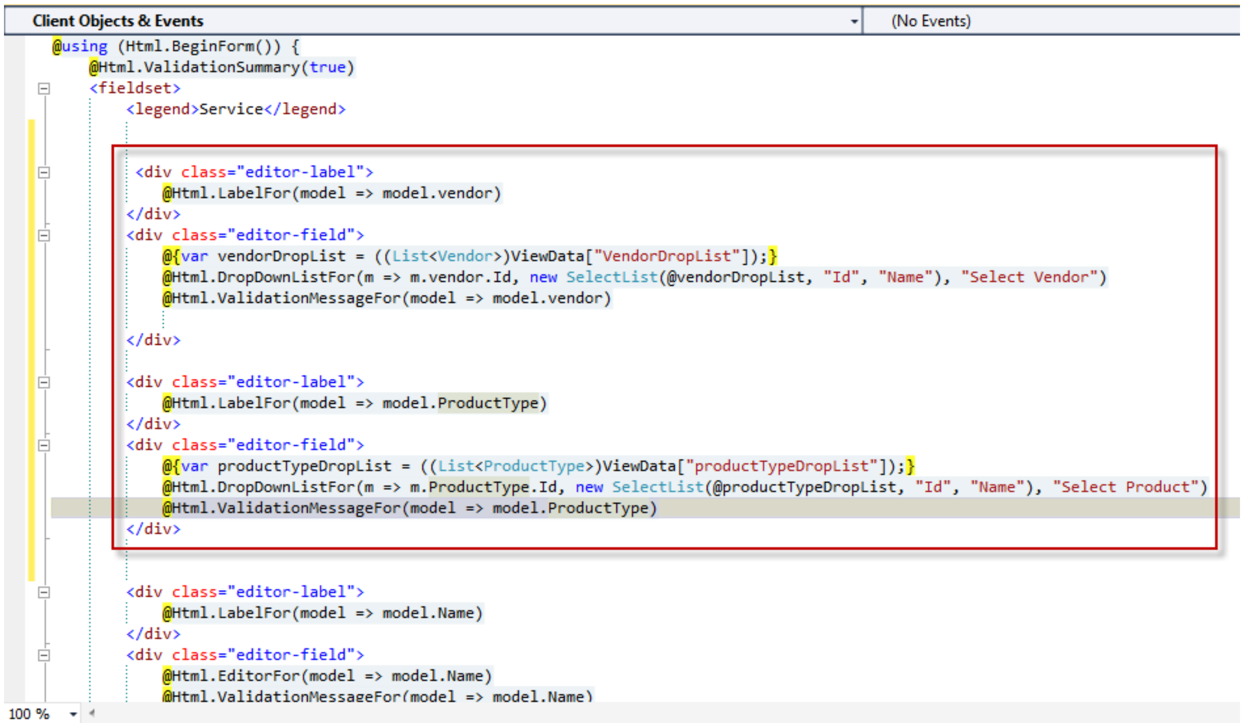
```
@{
    ViewBag.Title = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Create</h2>
@Html.Partial("Form")
```

- Open the Form view of the service, and locate the arrow



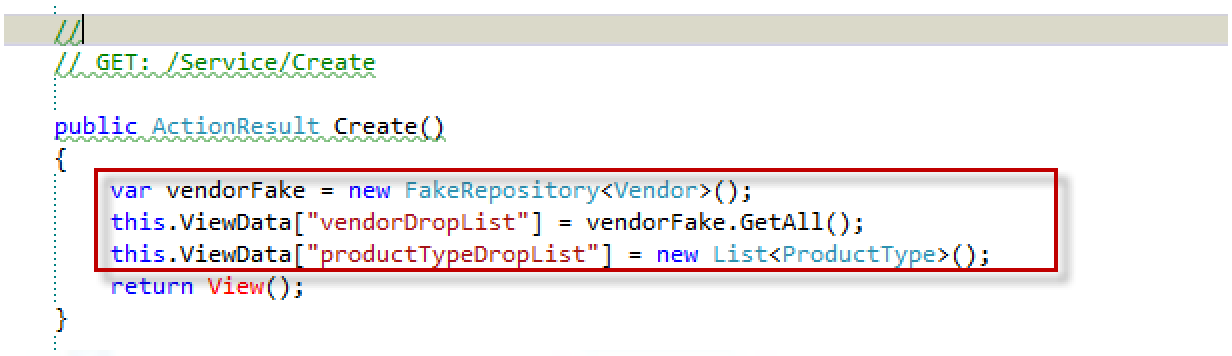
```
@model Entities.Service

@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Service</legend>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Description)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Description)
            @Html.ValidationMessageFor(model => model.Description)
        </div>

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

- Add the following code in the previous area in the image; this will display the Vendors drop down list and the Product Type list



```
@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <fieldset>
        <legend>Service</legend>

         <div class="editor-label">
            @Html.LabelFor(model => model.vendor)
        </div>
        <div class="editor-field">
            @{var vendorDropList = ((List<Vendor>)ViewData["VendorDropList"]);}
            @Html.DropDownListFor(m => m.vendor.Id, new SelectList(@vendorDropList, "Id", "Name"), "Select Vendor")
            @Html.ValidationMessageFor(model => model.vendor)

        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.ProductType)
        </div>
        <div class="editor-field">
            @{var productTypeDropList = ((List<ProductType>)ViewData["productTypeDropList"]);}
            @Html.DropDownListFor(m => m.ProductType.Id, new SelectList(@productTypeDropList, "Id", "Name"), "Select Product")
            @Html.ValidationMessageFor(model => model.ProductType)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
```

- Add the content of the Create (Get) action method of the Service

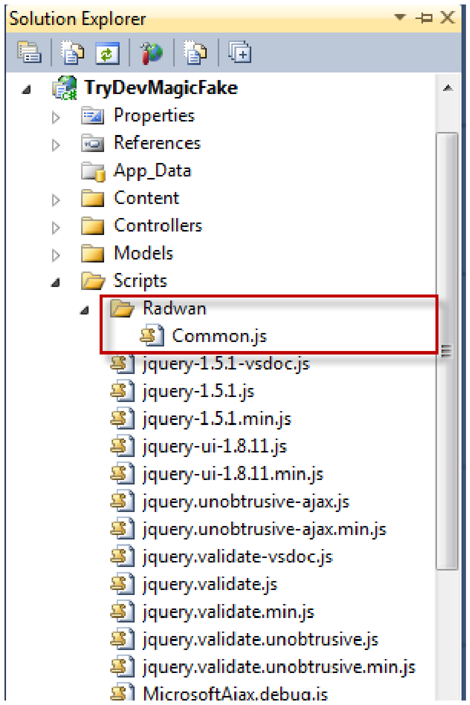


```
//
// GET: /Service/Create

public ActionResult Create()
{
    var vendorFake = new FakeRepository<Vendor>();
    this.ViewData["vendorDropList"] = vendorFake.GetAll();
    this.ViewData["productTypeDropList"] = new List<ProductType>();
    return View();
}
```

7.3. Fill Product Type drop down by Vendor in the Vendor drop down

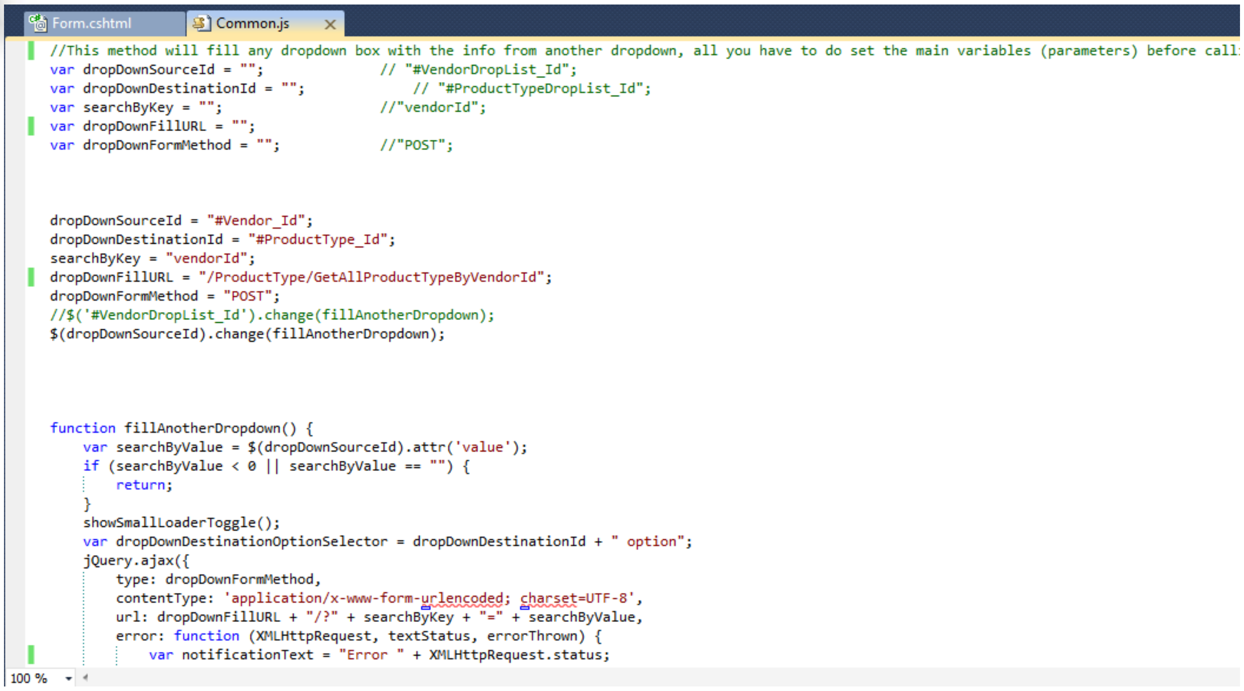
We don't have this section in the previous complex save, because in our business scenario to add service we have to add it to product that provided by specific vendor, so we will select the vendor from the drop down list so the product drop down list will be filled by all products provided to our company by this vendor

- Create folder Radwan under scripts as the following:



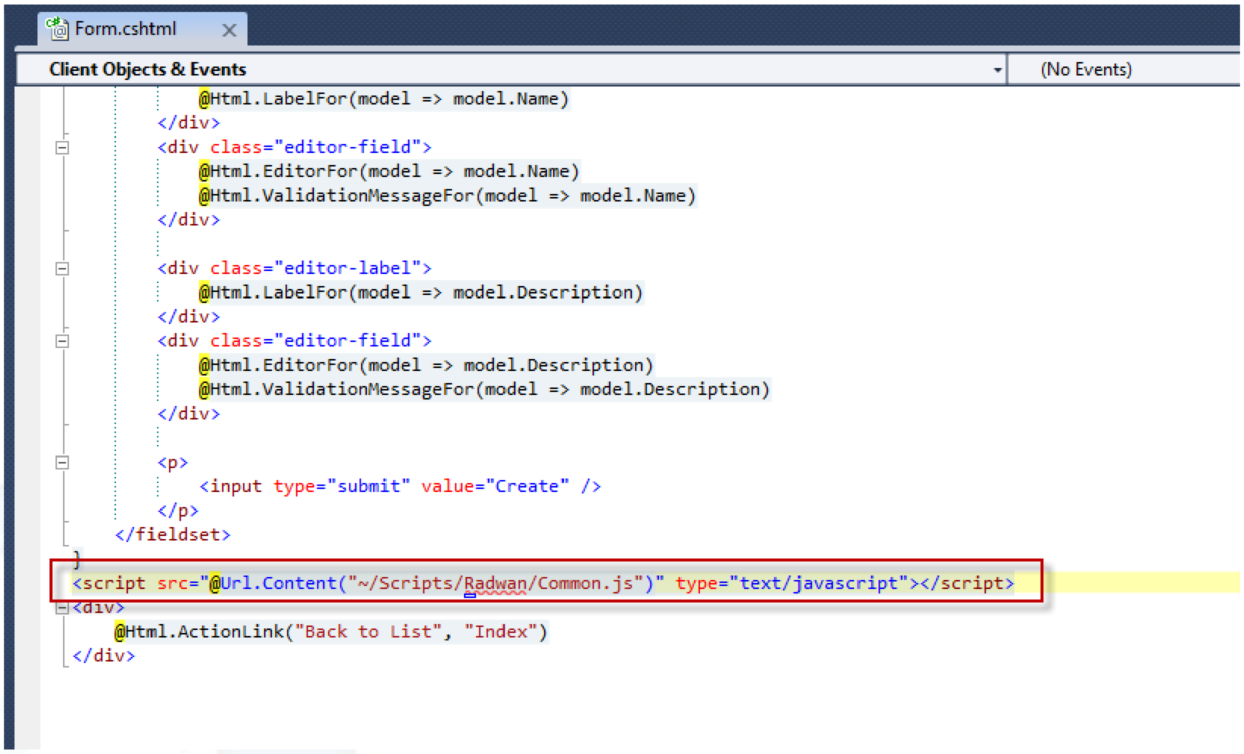
This is JQuery script file and responsible for parsing the JSON to fill the drop down of the product type with the vendor Id, the JQuery file look like this, we can find the file in downloaded project

(This part out of the scope)



```
//This method will fill any dropdown box with the info from another dropdown, all you have to do set the main variables (parameters) before call
var dropDownSourceId = "";              // "#VendorDropList_Id";
var dropDownDestinationId = "";             // "#ProductTypeDropList_Id";
var searchByKey = "";                   //"vendorId";
var dropDownFillURL = "";
var dropDownFormMethod = "";            //"POST";



dropDownSourceId = "#Vendor_Id";
dropDownDestinationId = "#ProductType_Id";
searchByKey = "vendorId";
dropDownFillURL = "/ProductType/GetAllProductTypeByVendorId";
dropDownFormMethod = "POST";
//$('#VendorDropList_Id').change(fillAnotherDropdown);
$(dropDownSourceId).change(fillAnotherDropdown);



function fillAnotherDropdown() {
    var searchByValue = $(dropDownSourceId).attr('value');
    if (searchByValue < 0 || searchByValue == "") {
        return;
    }
    showSmallLoaderToggle();
    var dropDownDestinationOptionSelector = dropDownDestinationId + " option";
    jQuery.ajax({
        type: dropDownFormMethod,
        contentType: 'application/x-www-form-urlencoded; charset=UTF-8',
        url: dropDownFillURL + "/?" + searchByKey + "=" + searchByValue,
        error: function (XMLHttpRequest, textStatus, errorThrown) {
            var notificationText = "Error " + XMLHttpRequest.status;
```

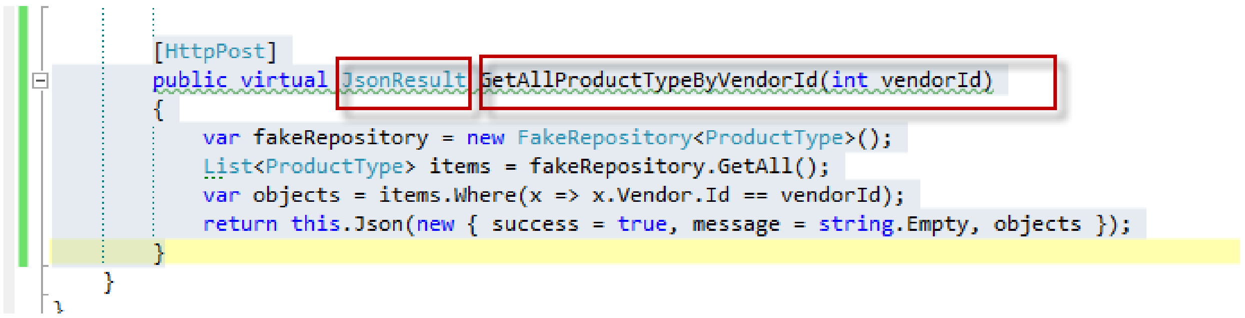
- Link the script to the Service Form view as the following:



```
            @Html.LabelFor(model => model.Name)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.Description)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.Description)
            @Html.ValidationMessageFor(model => model.Description)
        </div>

        <p>
            <input type="submit" value="Create" />
        </p>
    </fieldset>
}
<script src="@Url.Content("~/Scripts/Radwan/Common.js")" type="text/javascript"></script>
<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

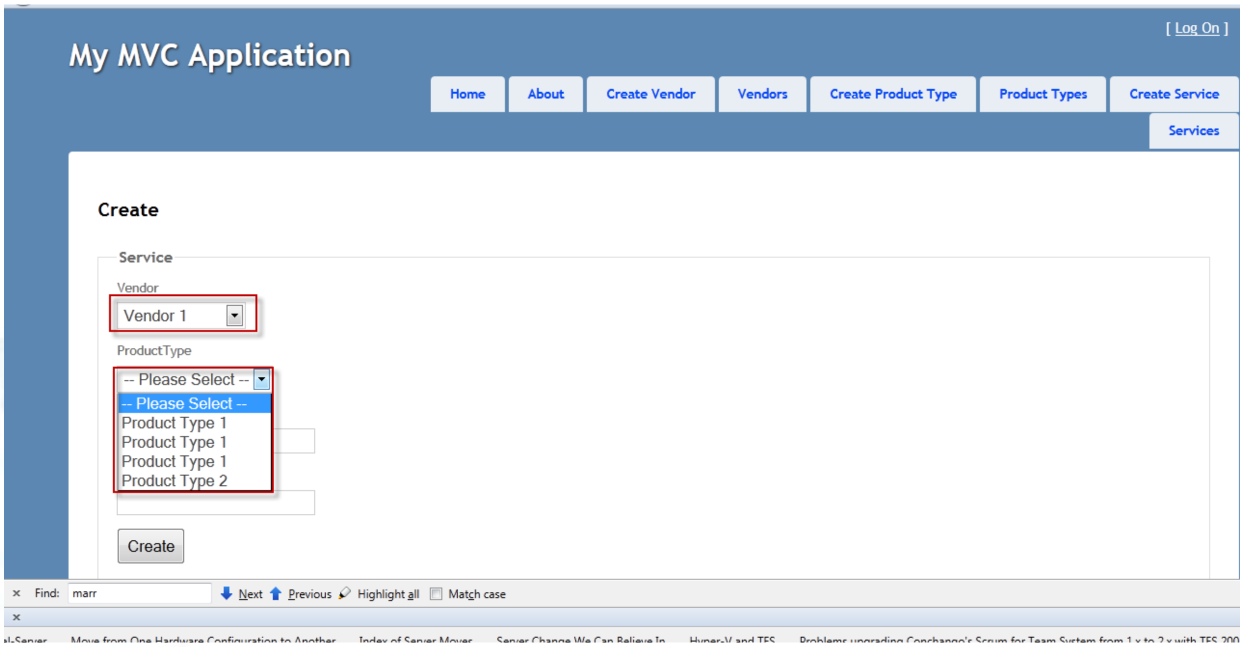
We will need to create action method that return JSON of all product types by vendor Id, so when the vendor drop list change the product type drop list will be filled with all products for this vendor

- Add GetAllProductTypeByVendorId action method to product type controller



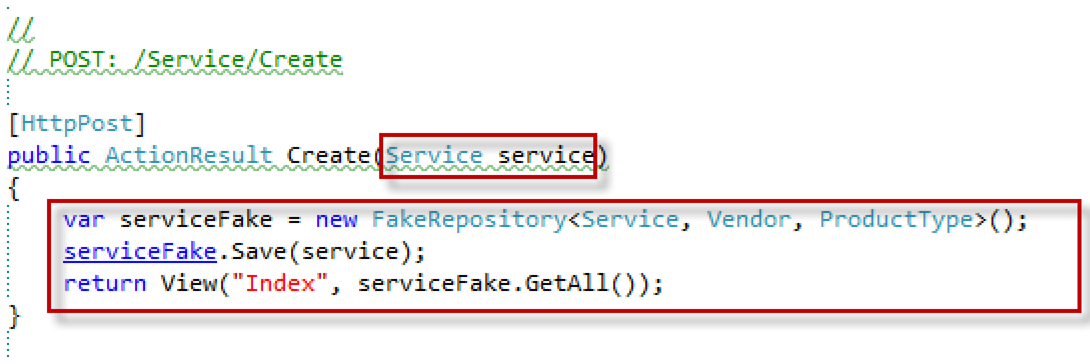
```
        [HttpPost]
        public virtual JsonResult GetAllProductTypeByVendorId(int vendorId)
        {
            var fakeRepository = new FakeRepository<ProductType>();
            List<ProductType> items = fakeRepository.GetAll();
            var objects = items.Where(x => x.Vendor.Id == vendorId);
            return this.Json(new { success = true, message = string.Empty, objects });
        }
    }
}
```

Run the application and open create service page and see how the vendor list is loaded. If not? Create some vendors and product type, reopen the create service page again, we will find that all vendors exist in the drop down list and if we select vendor from the drop down list the product type drop list will be filled by the products that provided by the selected vendor.



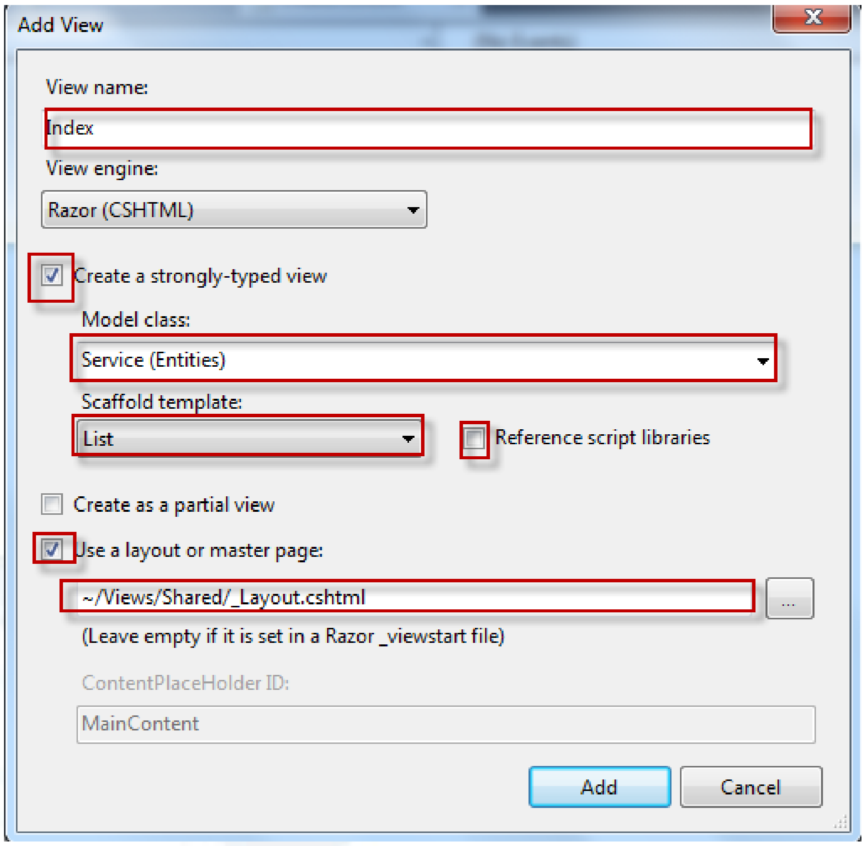
7.4. Use Dev Magic Fake in the Service Controller action methods

- Add the content of the Create (Post) action method of the Service, we can see hear in this instance we create FakeRepository of three types the main and two subs, this because we want to save complex object, the subs will (Vendor and Product type) only have id and name because the drop down list only has this values (Vendor drop down) and (Product drop down)



```
//
// POST: /Service/Create

[HttpPost]
public ActionResult Create(Service service)
{
    var serviceFake = new FakeRepository<Service, Vendor, ProductType>();
    serviceFake.Save(service);
    return View("Index", serviceFake.GetAll());
}
```

- Add an index view for service as the following:



Add View

View name:
Index

View engine:
Razor (CSHTML)

[✓] Create a strongly-typed view

Model class:
Service (Entities)

Scaffold template:
List

[ ] Reference script libraries

[ ] Create as a partial view

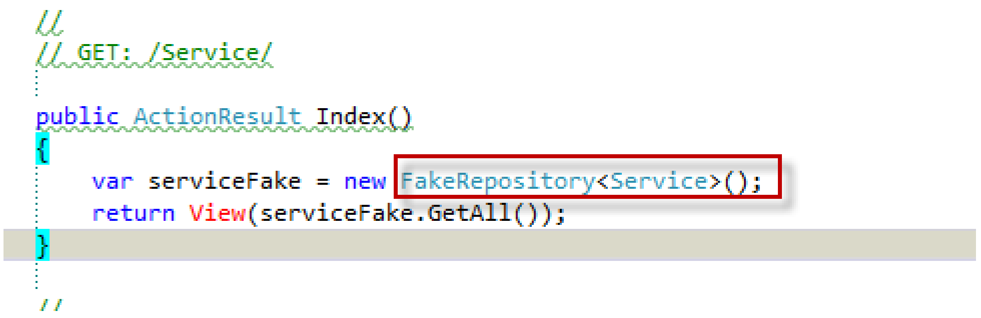
[✓] Use a layout or master page:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

ContentPlaceHolder ID:
MainContent

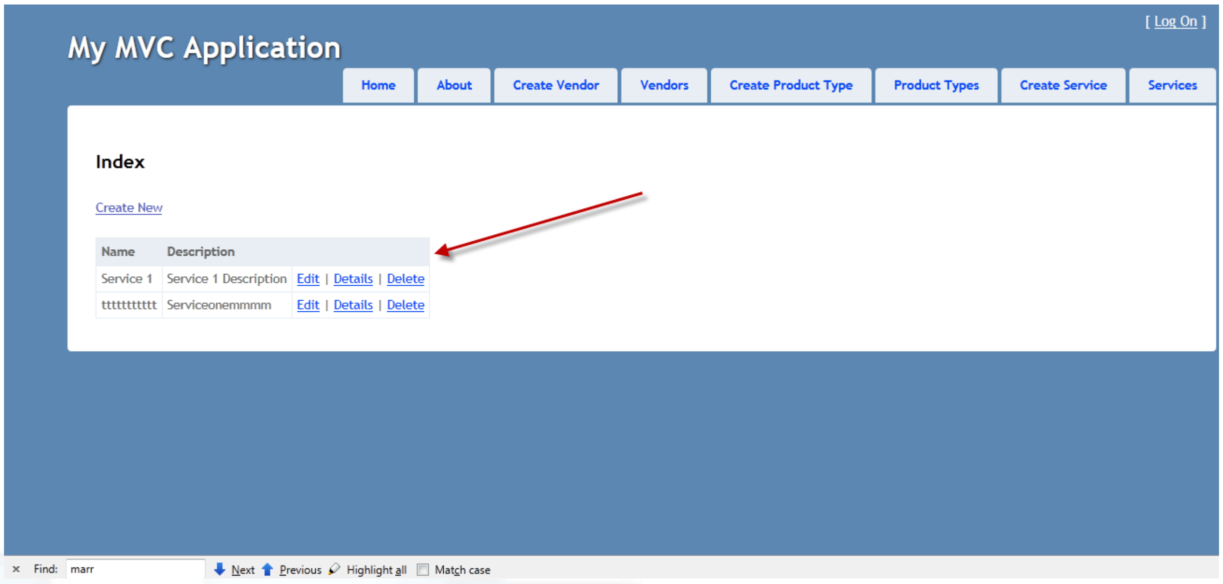
- Add the following code to the index action method to display all the services when we open the service index page



```
//
// GET: /Service/

public ActionResult Index()
{
    var serviceFake = new FakeRepository<Service>();
    return View(serviceFake.GetAll());
}

//
```

7.5. Examine the result of Dev Magic Fake

- Run the application and open create service and see how the vendor list is loaded.
- Create some services.
- Open services page, we will find that all the service that has been added are displayed



Index

Create New

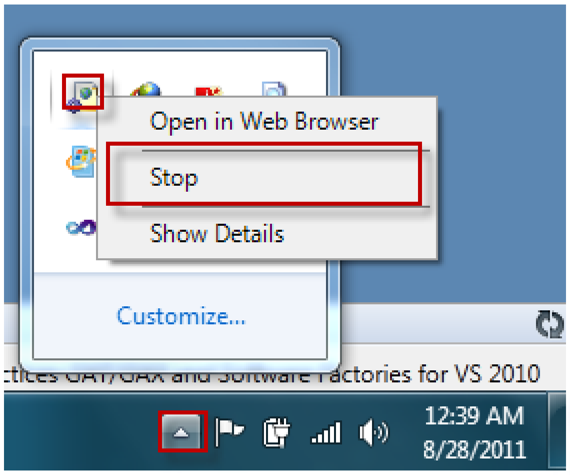
Name	Description	
Service 1	Service 1 Description	Edit | Details | Delete
tttttttttt	Serviceonemmmm	Edit | Details | Delete

8. Enable Dev Magic Fake to save data permanently

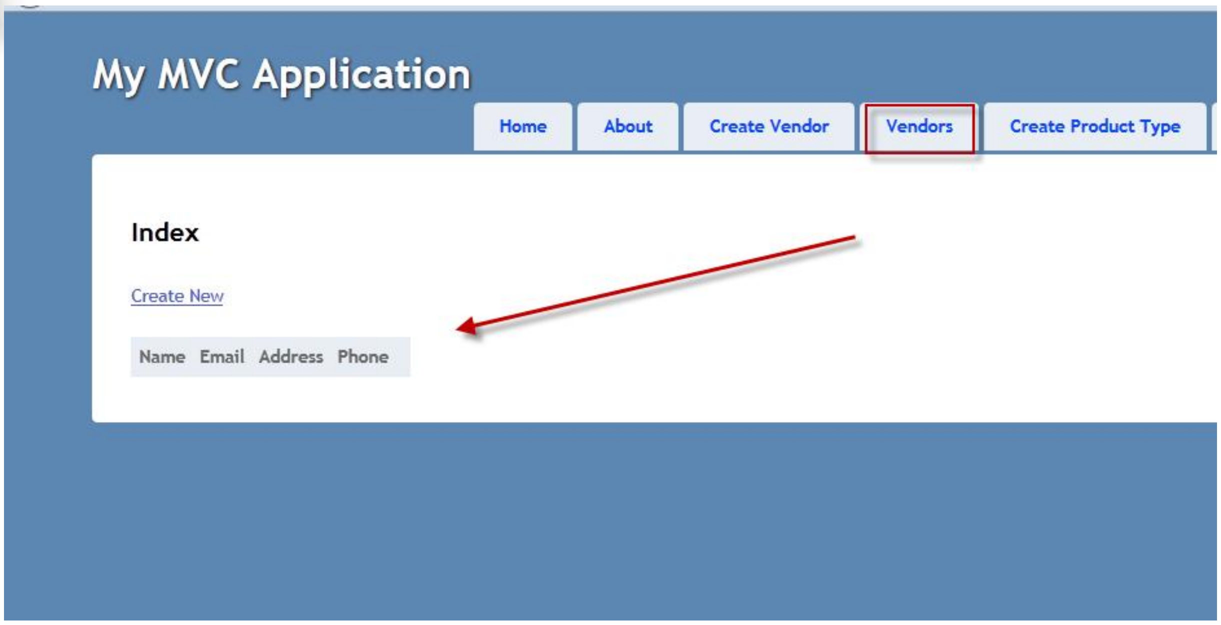
While we success saving instances of Vendor, Product Type and Service without having any information about these class before, but this saving not permanent, if the web server closed or restarted we will lose all saved data, so in this section we will describe how Dev Magic Fake can make the save process permanent.

8.1. Examine how the saved data is not permanent

- Run the application and add some Vendors, Product Types and Services
- Open the open the Vendors, Product Type and Services pages and examine how the data is exist
- Close the application and stop the development server

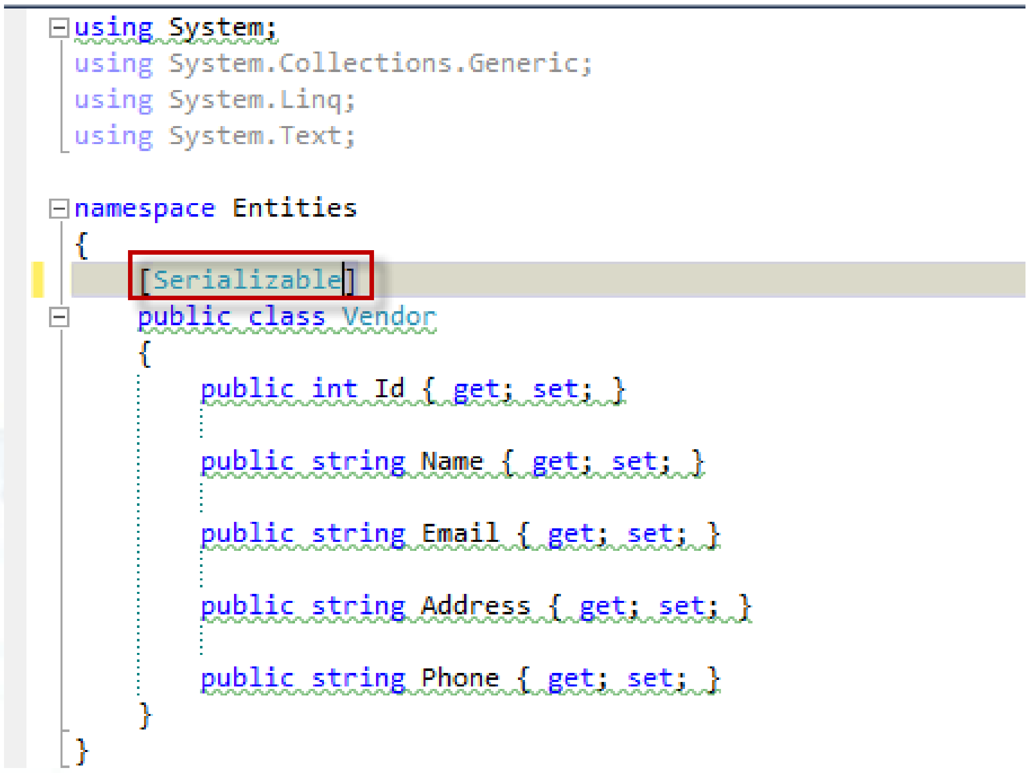


Run the application again and open the Vendors, Product Type and Services pages, we will find that the items that we saved are gone



8.2. Add Serializable attribute

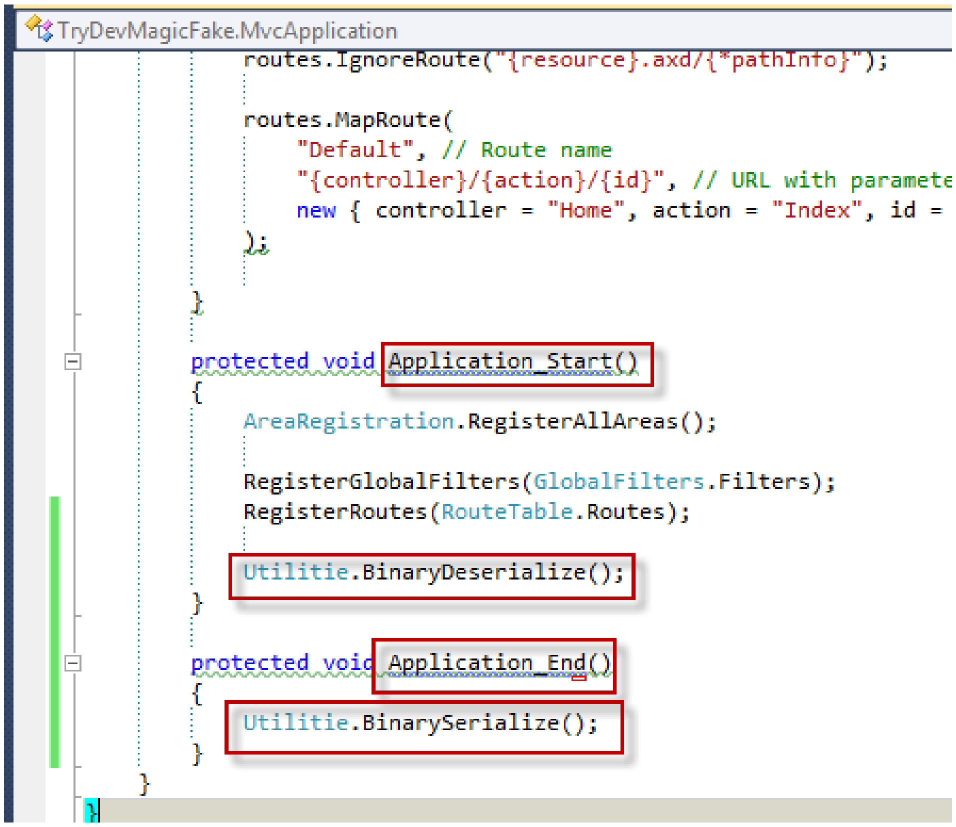
- We want to make our saved data permanent all the time, so we will add attribute [Serializable] for all needed classes to be saved permanently



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Entities
{
    [Serializable]
    public class Vendor
    {
        public int Id { get; set; }

        public string Name { get; set; }

        public string Email { get; set; }

        public string Address { get; set; }

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

8.3. Call Dev Magic Fake data permanent methods

- Open the Global.asax file and calling the Dev Magic Fake serialization methods as the following:



TryDevMagicFake.MvcApplication

```
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with paramete
                new { controller = "Home", action = "Index", id =
            );

        }

        protected void Application_Start()
        {
            AreaRegistration.RegisterAllAreas();

            RegisterGlobalFilters(GlobalFilters.Filters);
            RegisterRoutes(RouteTable.Routes);

            Utilitie.BinaryDeserialize();
        }

        protected void Application_End()
        {
            Utilitie.BinarySerialize();
        }
    }
}
```

- Run the application again, add some Vendors, Product Types and Services, stop the development server and re run the application and open the Vendors, Product Type and Services pages, we will find that all saved items are existing

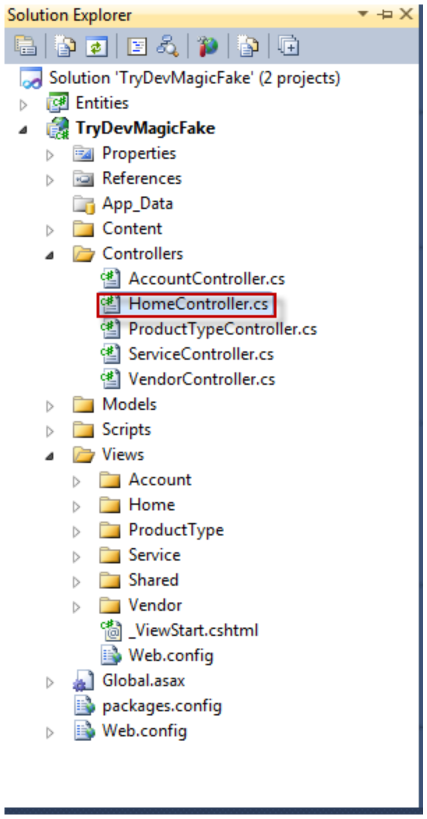
9. Generate data using Dev Magic Fake

Dev Magic Fake has features for data generation; in this section we will examine some of these features.

9.1. Generate data using default data generation

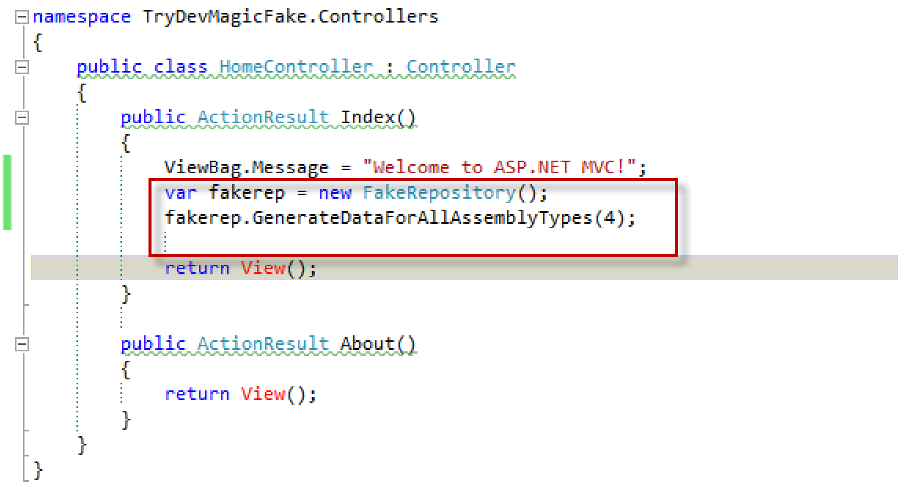
Dev Magic Fake can generate data for all classes that exist in specific assembly

- Open the home controller



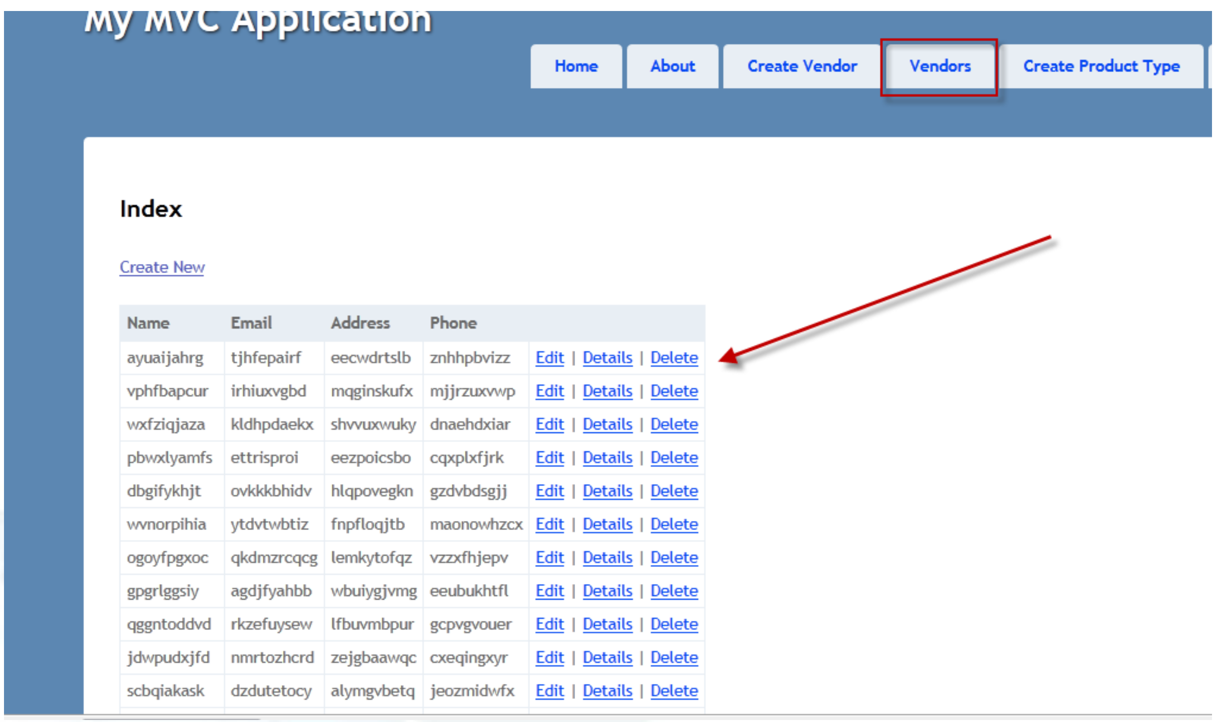
```
Solution 'TryDevMagicFake' (2 projects)
  Entities
  TryDevMagicFake
    Properties
    References
    App_Data
    Content
    Controllers
      AccountController.cs
      HomeController.cs
      ProductTypeController.cs
      ServiceController.cs
      VendorController.cs
    Models
    Scripts
    Views
      Account
      Home
      ProductType
      Service
      Shared
      Vendor
      _ViewStart.cshtml
      Web.config
    Global.asax
    packages.config
    Web.config
```

- Call the generate data method in the home index method as the following:



```
namespace TryDevMagicFake.Controllers
{
    public class HomeController : Controller
    {
        public ActionResult Index()
        {
            ViewBag.Message = "Welcome to ASP.NET MVC!";
            var fakerep = new FakeRepository();
            fakerep.GenerateDataForAllAssemblyTypes(4);

            return View();
        }

        public ActionResult About()
        {
            return View();
        }
    }
}
```

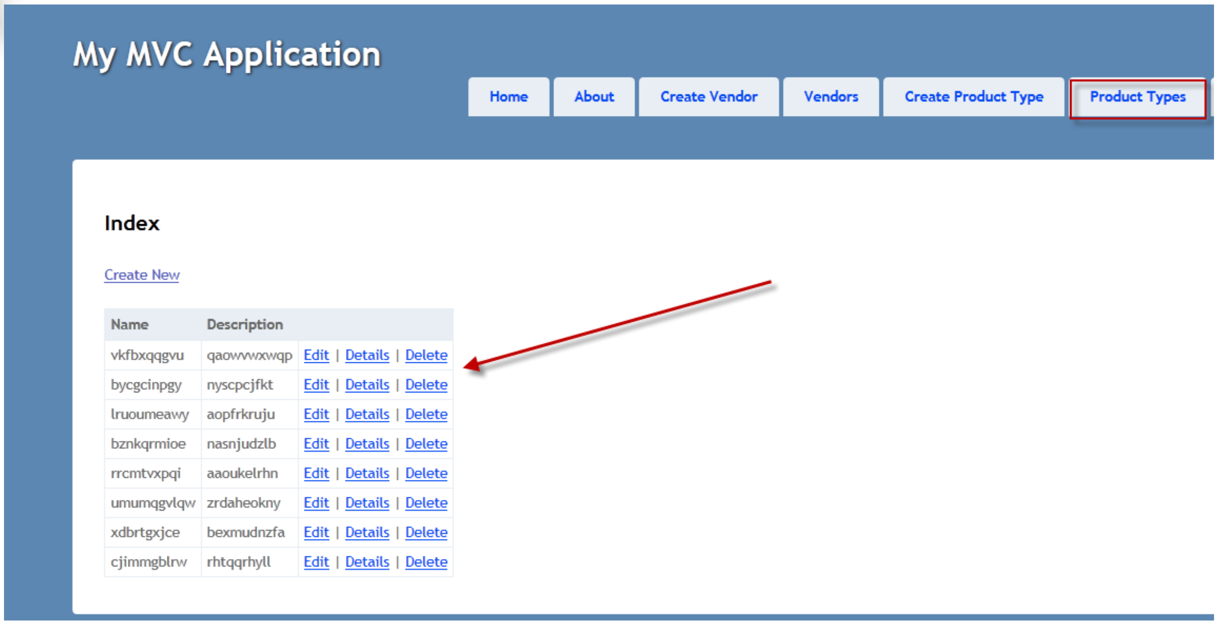
Run the application, open the vendors, the product types, the services pages, we will find and see the generated data for these types



Index

Create New

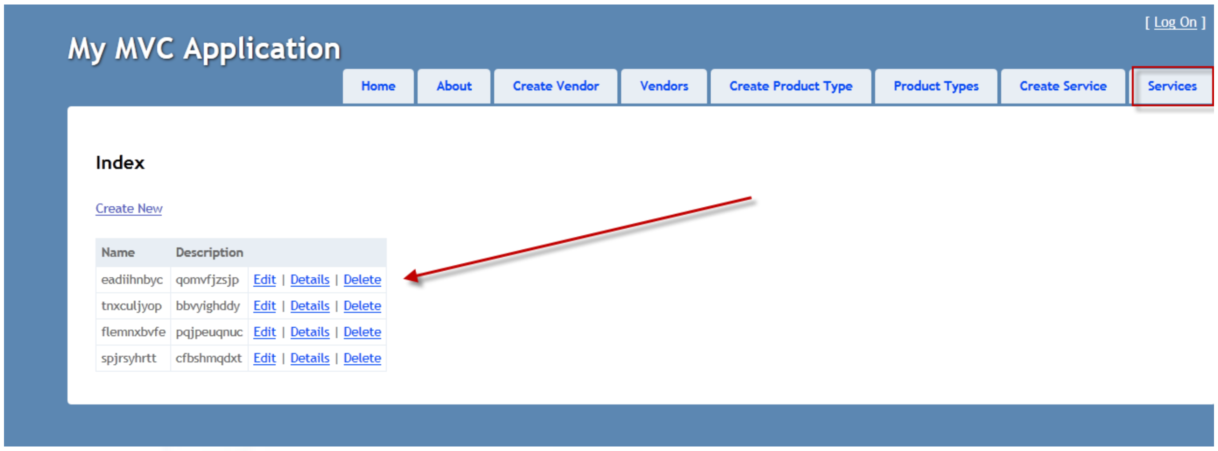
Name	Email	Address	Phone	
ayuaijahrg	tjhfepairf	eecwdrtslb	znhhpbvizz	Edit | Details | Delete
vphfbapcur	irhiuxvgbd	mqginskufx	mjjrzuxvwp	Edit | Details | Delete
wxfziqjaza	kldhpdaekx	shvvuxwuky	dnaehdxiar	Edit | Details | Delete
pbwxlyamfs	ettrisproi	eezpoicsbo	cqxplxfjrk	Edit | Details | Delete
dbgifykhjt	ovkkkbhidv	hlqpovegkn	gzdvbdsgjj	Edit | Details | Delete
wvnorpihia	ytdvtwbtiz	fnpfloqjtb	maonowhzcx	Edit | Details | Delete
ogoyfpgxoc	qkdmzrcqcg	lemkytofqz	vzzxfhjepv	Edit | Details | Delete
gpgrlggsiy	agdjfyahbb	wbuiygjvmg	eeubukhtfl	Edit | Details | Delete
qggntoddvd	rkzefuysew	lfbuvmbpur	gcpvgvouer	Edit | Details | Delete
jdwpudxjfd	nmrtozhcrd	zejgbaawqc	cxeqingxyr	Edit | Details | Delete
scbqiakask	dzdutetocy	alymgvbetq	jeozmidwfx	Edit | Details | Delete



Index

Create New

Name	Description	
vkfbxqqgvu	qaowvwxwqp	Edit | Details | Delete
bycgcinpgy	nyscpcjfkt	Edit | Details | Delete
lruoumeawy	aopfrkruju	Edit | Details | Delete
bznkqrmioe	nasnjudzlb	Edit | Details | Delete
rrcmtvxpqi	aaoukelrhn	Edit | Details | Delete
umumqgvlqw	zrdaheokny	Edit | Details | Delete
xdbrtgxjce	bexmudnzfa	Edit | Details | Delete
cjimmgblrw	rhtqqrhyll	Edit | Details | Delete



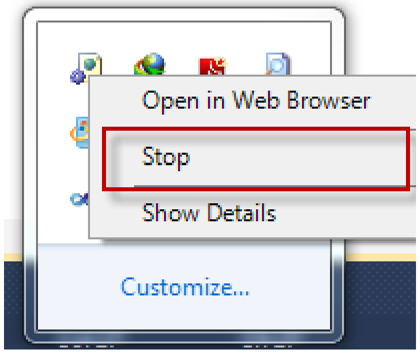
Index

Create New

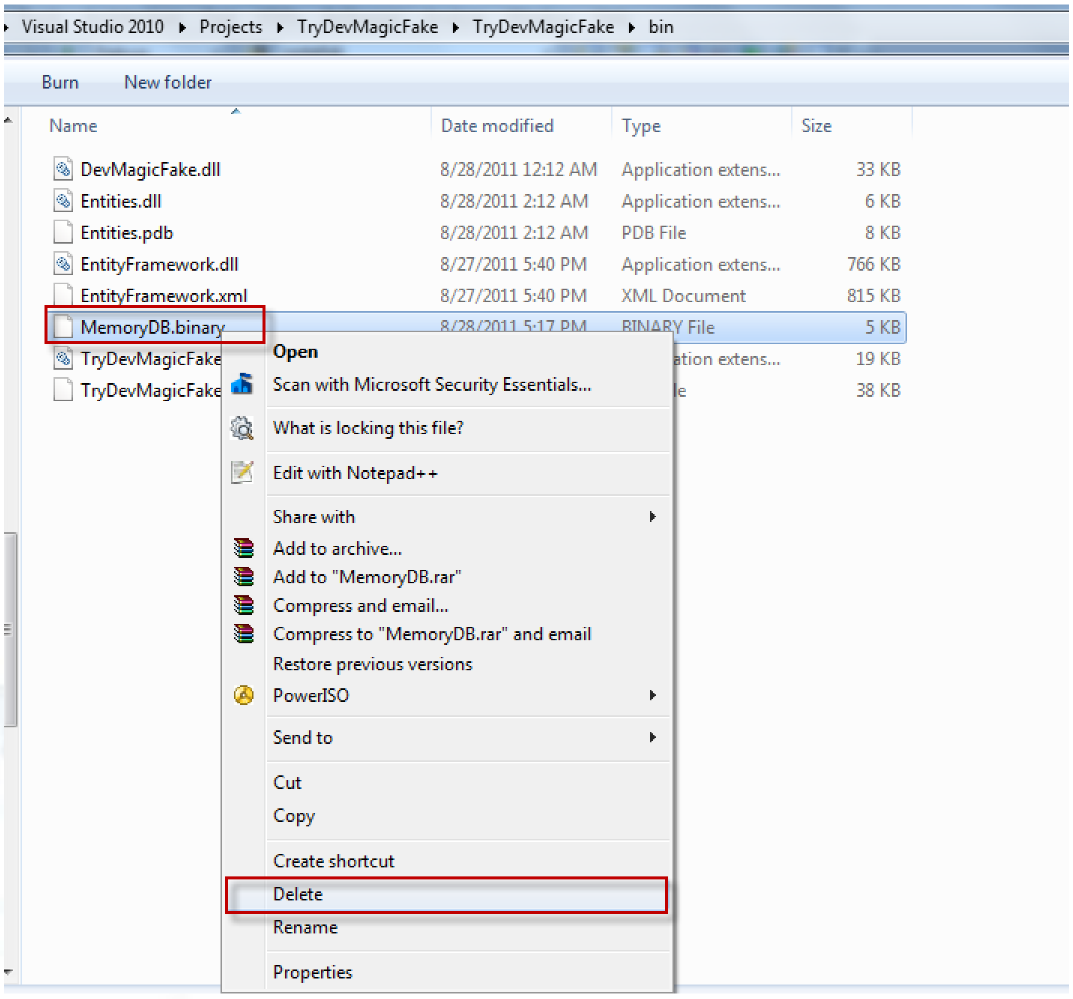
Name	Description	
eadiihnbyc	qomvfjzsjp	Edit | Details | Delete
tnxculjyop	bbvyighddy	Edit | Details | Delete
flemnxbvfe	pqjpeuqnuc	Edit | Details | Delete
spjrsyhrtt	cfbshmqdxt	Edit | Details | Delete

9.2. Generate data using Range and Data type configuration

- Stop the Development server



- Open the bin folder of the web project TryDevMagicFake project, delete MemoryDB.binary file


Visual Studio 2010 ▸ Projects ▸ TryDevMagicFake ▸ TryDevMagicFake ▸ bin
Burn
New folder
Open
Scan with Microsoft Security Essentials...
What is locking this file?
Edit with Notepad++
Share with
Add to archive...
Add to "MemoryDB.rar"
Compress and email...
Compress to "MemoryDB.rar" and email
Restore previous versions
PowerISO
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties

Name	Date modified	Type	Size
DevMagicFake.dll	8/28/2011 12:12 AM	Application extens...	33 KB
Entities.dll	8/28/2011 2:12 AM	Application extens...	6 KB
Entities.pdb	8/28/2011 2:12 AM	PDB File	8 KB
EntityFramework.dll	8/27/2011 5:40 PM	Application extens...	766 KB
EntityFramework.xml	8/27/2011 5:40 PM	XML Document	815 KB
MemoryDB.binary	8/28/2011 5:17 PM	BINARY File	5 KB
TryDevMagicFake		ation extens...	19 KB
TryDevMagicFake		le	38 KB

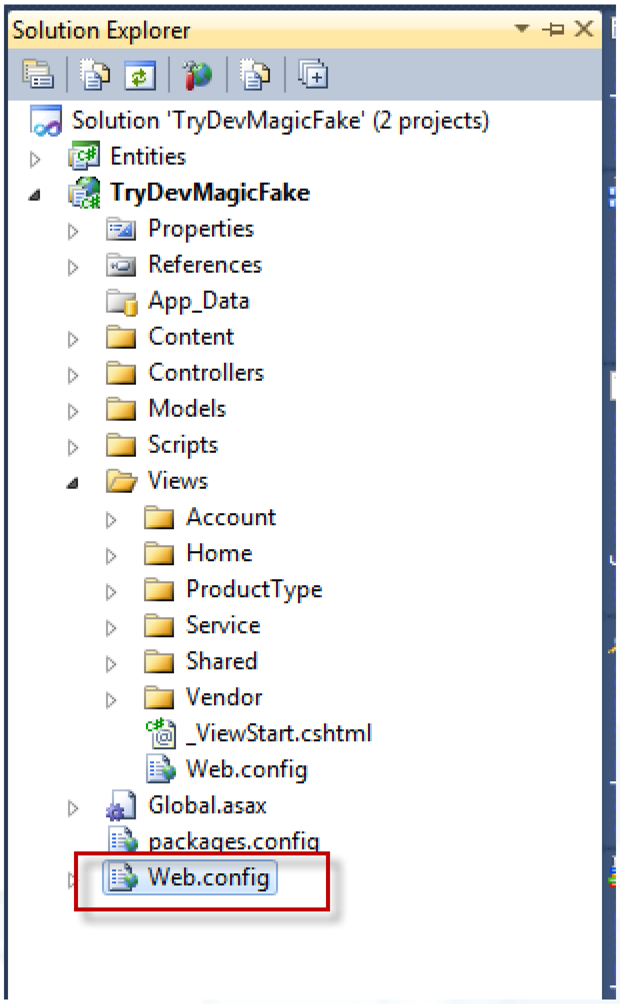
This file has all the saved data; we will delete this file to empty any earlier data generation in the previous example

Note:

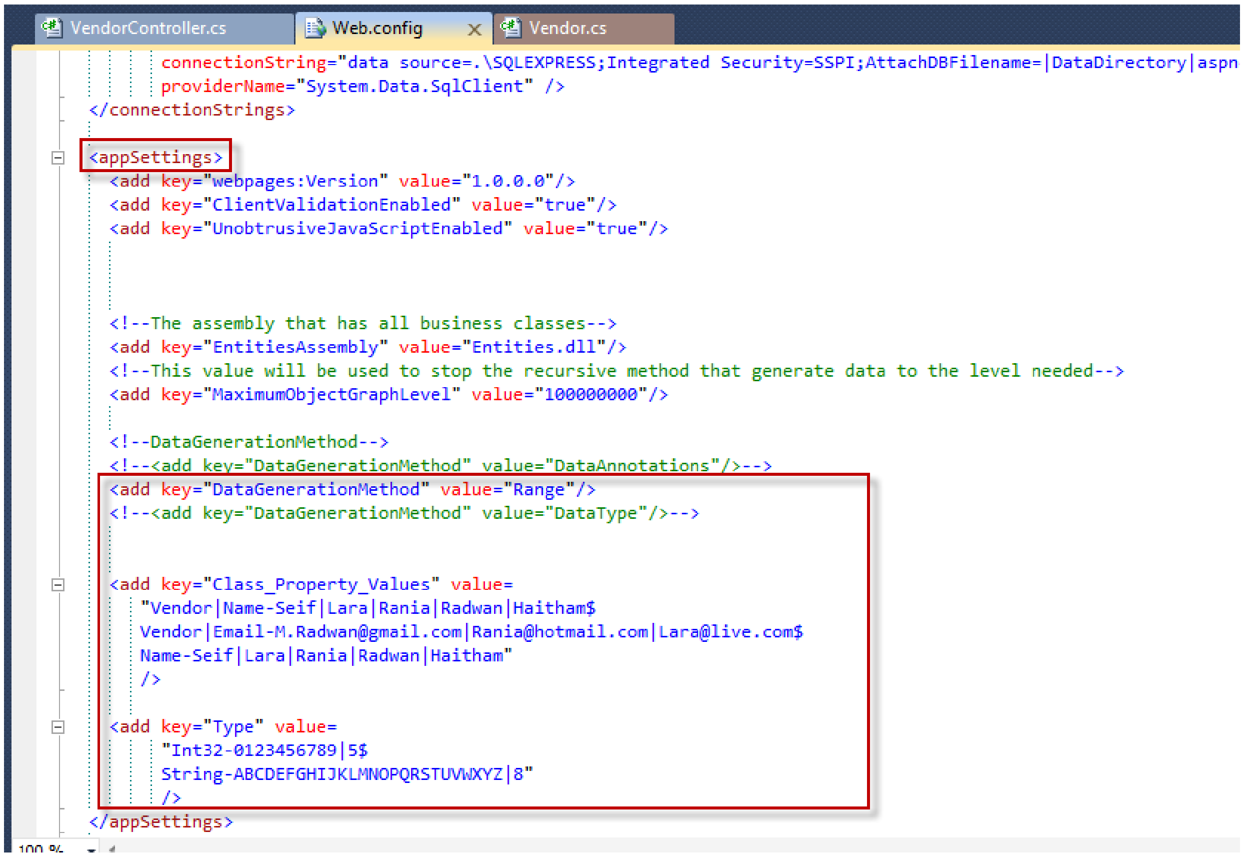
If we change any type definition we have to delete this file otherwise an exception will be thrown.

9.3. Change the data generation configuration

- Open the web.config file



- Enter the data generation settings as the following:



VendorController.cs | Web.config | Vendor.cs

```
        connectionString="data source=.\SQLEXPRESS;Integrated Security=SSPI;AttachDBFilename=|DataDirectory|aspn
        providerName="System.Data.SqlClient" />
  </connectionStrings>

  <appSettings>
    <add key="webpages:Version" value="1.0.0.0"/>
    <add key="ClientValidationEnabled" value="true"/>
    <add key="UnobtrusiveJavaScriptEnabled" value="true"/>


    <!--The assembly that has all business classes-->
    <add key="EntitiesAssembly" value="Entities.dll"/>
    <!--This value will be used to stop the recursive method that generate data to the level needed-->
    <add key="MaximumObjectGraphLevel" value="100000000"/>

    <!--DataGenerationMethod-->
    <!--<add key="DataGenerationMethod" value="DataAnnotations"/>-->
    <add key="DataGenerationMethod" value="Range"/>
    <!--<add key="DataGenerationMethod" value="DataType"/>-->


    <add key="Class_Property_Values" value=
       "Vendor|Name-Seif|Lara|Rania|Radwan|Haitham$
       Vendor|Email-M.Radwan@gmail.com|Rania@hotmail.com|Lara@live.com$
       Name-Seif|Lara|Rania|Radwan|Haitham"
       />

    <add key="Type" value=
         "Int32-0123456789|5$
         String-ABCDEFGHIJKLMNOPQRSTUVWXYZ|8"
         />
  </appSettings>
```

As we can see we have 3 types of data generation method Range, Data type and Data annotation (not supported in this release)

If we select Range we can start add the range as the following:

Class Name|Property Name-Value1|Value2|Value3|Value N$

This will randomly generate data for this property for <u>this</u> class

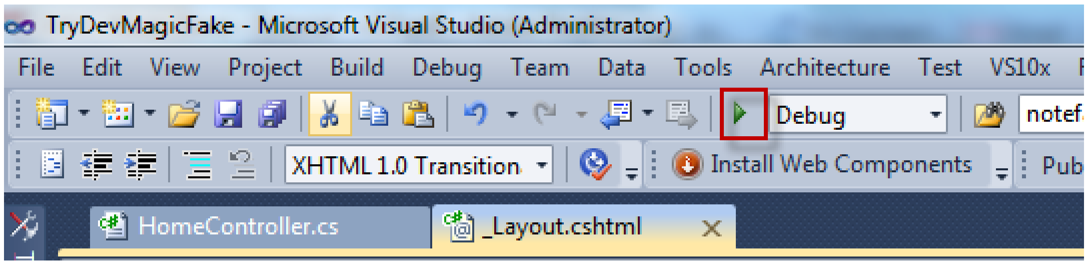
Or

Property Name-Value1|Value2|Value3|Value N$

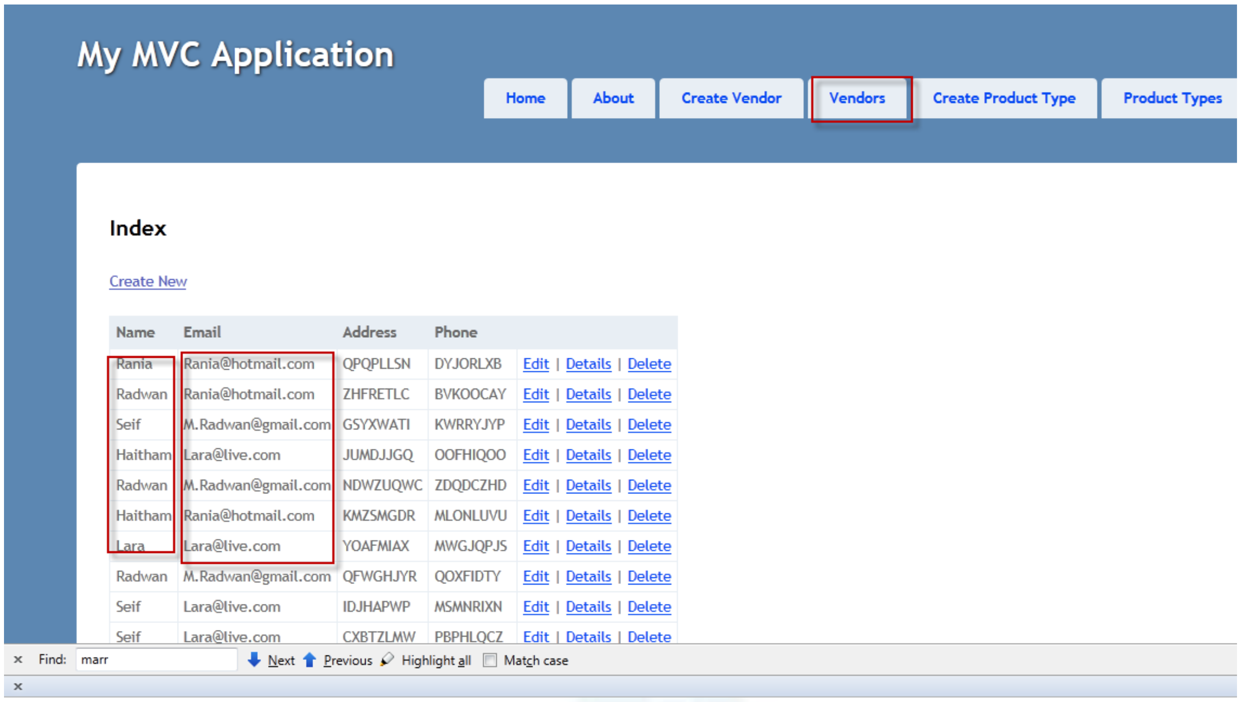
This will randomly generate data for this property for <u>any</u> class

9.4. Examine the result of changing the data generation configuration

- Run the application


TryDevMagicFake - Microsoft Visual Studio (Administrator)
File Edit View Project Build Debug Team Data Tools Architecture Test VS10x
Debug
notef
XHTML 1.0 Transition
Install Web Components
Pub
HomeController.cs
_Layout.cshtml

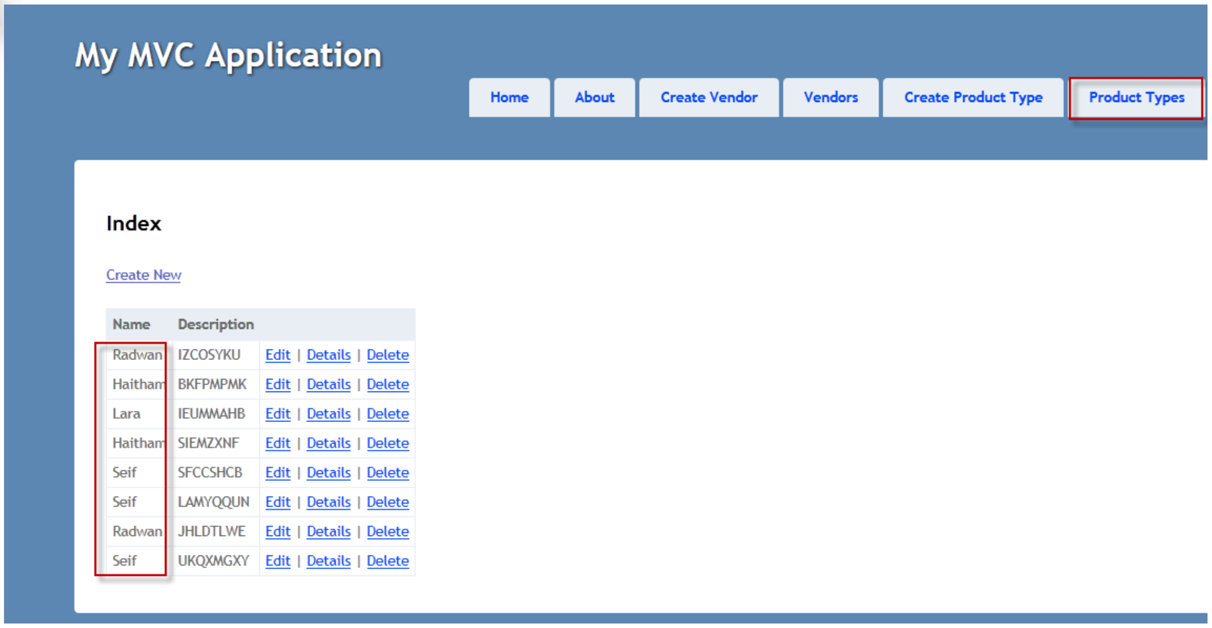
- Open Vendors, Product Types and Service page and see the generated data



Index

Create New

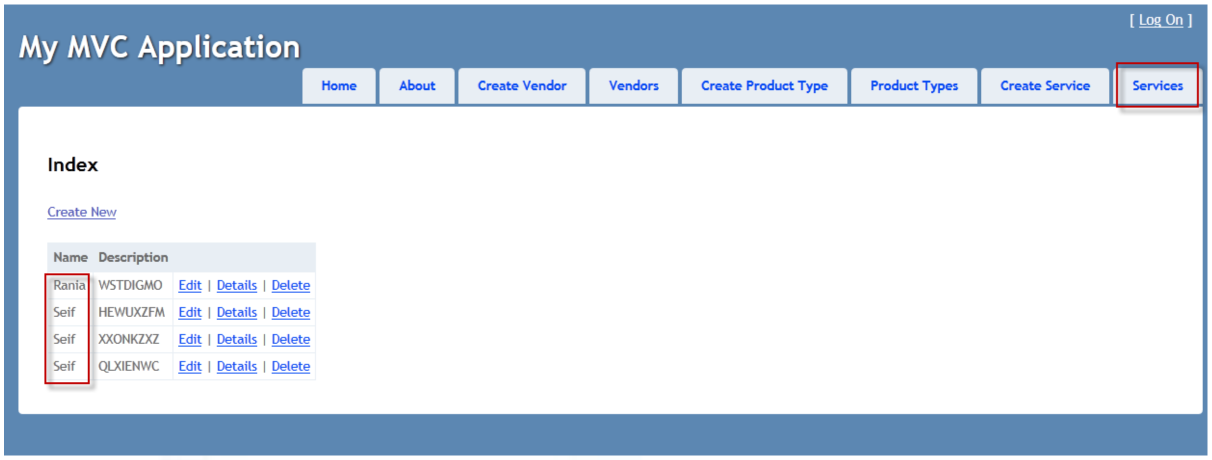
Name	Email	Address	Phone	
Rania	Rania@hotmail.com	QPQPLLSN	DYJORLXB	Edit | Details | Delete
Radwan	Rania@hotmail.com	ZHFRETLC	BVKOOCAY	Edit | Details | Delete
Seif	M.Radwan@gmail.com	GSYXWATI	KWRRYJYP	Edit | Details | Delete
Haitham	Lara@live.com	JUMDJJGQ	OOFHIQOO	Edit | Details | Delete
Radwan	M.Radwan@gmail.com	NDWZUQWC	ZDQDCZHD	Edit | Details | Delete
Haitham	Rania@hotmail.com	KMZSMGDR	MLONLUVU	Edit | Details | Delete
Lara	Lara@live.com	YOAFMIAX	MWGJQPJS	Edit | Details | Delete
Radwan	M.Radwan@gmail.com	QFWGHJYR	QOXFIDTY	Edit | Details | Delete
Seif	Lara@live.com	IDJHAPWP	MSMNRIXN	Edit | Details | Delete
Seif	Lara@live.com	CXBTZLMW	PBPHLQCZ	Edit | Details | Delete



Index

Create New

Name	Description	
Radwan	IZCOSYKU	Edit | Details | Delete
Haitham	BKFPMPMK	Edit | Details | Delete
Lara	IEUMMAHB	Edit | Details | Delete
Haitham	SIEMZXNF	Edit | Details | Delete
Seif	SFCCSHCB	Edit | Details | Delete
Seif	LAMYQQUN	Edit | Details | Delete
Radwan	JHLDTLWE	Edit | Details | Delete
Seif	UKQXMGXY	Edit | Details | Delete



Index

Create New

Name	Description	
Rania	WSTDIGMO	Edit | Details | Delete
Seif	HEWUXZFM	Edit | Details | Delete
Seif	XXONKZXZ	Edit | Details | Delete
Seif	QLXIENWC	Edit | Details | Delete

10. Features that didn't covered by this tutorial

This tutorial not cover all features of Dev Magic Fake, actually it only covered 10% of its features, so this tutorial just the first step of how to use Dev Magic Fake, for more information about Dev Magic Fake and how to use please see the Codeplex site for more documentation about Dev Magic Fake features and help files

11. Summary

We saw in the previous tutorial how to use Dev Magic fake in a simple scenario, we save simple type, complex type that has only one sub type and complex type that has two sub types, Dev Magic Fake has many features like saving complex object that has collection, , complex object that need to save without retrieved it's nested object, actually Dev Magic Fake released to provide faking for complex scenarios.

Dev Magic Fake is a faking framework that gives us the ability to focus on how to complete, verify and test the application behaviors and response without focus on coding or developing the underline layers until the application features finished, tested and approved

So the main goal of the Dev Magic Fake is to give us the ability to:

- Implementing "Develop By Feature" approach by Agile methodology
- Complete the feature without coding the underline layers
- Give the ability to creating a passed successful unit testing to test the behavior and response of the application without completing the underline layers
- Give the ability to creating a passed successful UI test to test the behavior and response of the application UI without complete the underline layers
- Create faking code with no effort
- Create faking code in no time
- Permanent data storage
- Minimum effort for replacing the faking code with the real one

12. About the Author

M.Radwan is a Lead Architect, Configuration Manager and Build Engineer, with more than 9 years of software architecture, design, development, and management experience, specializing in Microsoft technologies and Agile methodology. Consulting and coaching clients in Egypt, KSA, Libya and Kuwait.

M.Radwan Focus on :C# / .NET, ASP.NET, MVC, JQuery, Test-Driven Development, MS Build, TFS, MS Team build, Application Architectures, Agile, Process Automation And Improvement, Configuration Management, Automation all tasks related to software development activities, this include but not limited for Development, Build, Configuration, Deployment, Test, etc.

M.Radwan is M.Sc. of computer sciences and information technology in Agile Methodology

M.Radwan holds number of Microsoft certifications including MCT, MCPD, MCITP in EPM, MCTS (7), MCSD, MCAD and CIW

M.Radwan believes that we have to learn from our mistakes and this what we called experience and the only way for productivity is to automate this experience.

http://social.msdn.microsoft.com/profile/M.Radwan

http://www.codeplex.com/site/users/view/mradwan

http://www.linkedin.com/in/mohamedahmedradwan

http://twitter.com/#!/mradwan06

http://stackoverflow.com/users/386323/m-radwan

http://www.youtube.com/user/MRadwanMSF

http://www.facebook.com/M.Radwan.TFS

13. Feedback

I would love to hear about what I do well and how I can improve, if there is anything you don't like or have an idea or enhancement, please email me on mradwan.automationplanet@gmail.com or contribute to the CodePlex discussions page for Dev Magic Fake, if you would like to contribute to this project as a developer, technical writer, or any other role please let me know.

Thanks

M.Radwan