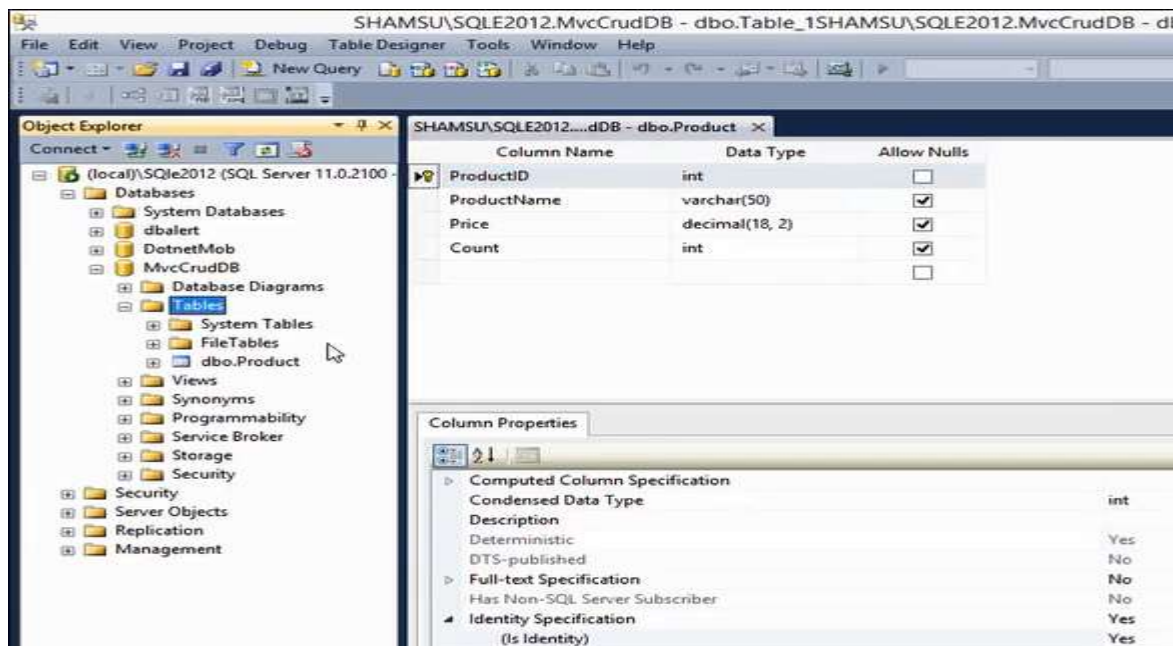


## Travaux pratiques : Gestion des produits (ASP.NET MVC)

**Objectif :** dans ce TP, nous allons créer une petite application MVC qui permet de réaliser les opérations CRUD pour la gestion des produits.

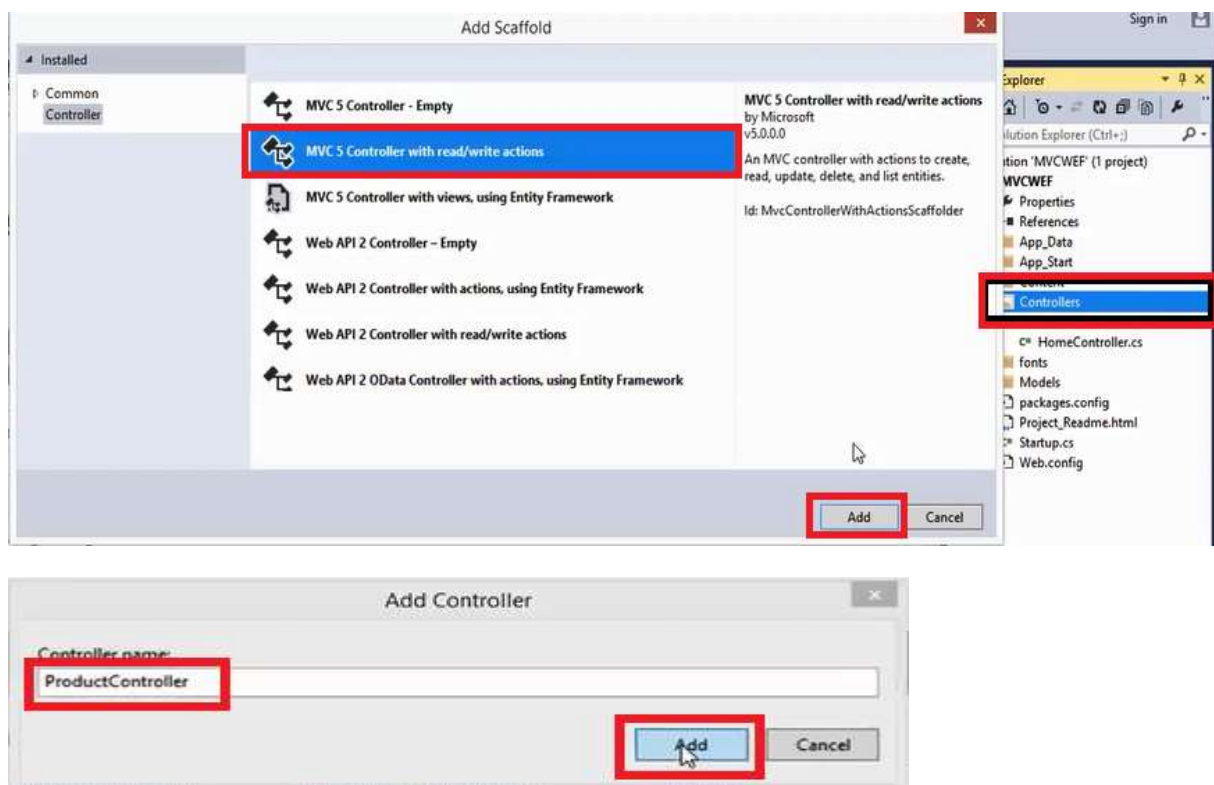
### 1. Créer la base de données :

Dans un premier temps il faut créer la base de données « MvcCrudDb » et y créer une table nommée « Product »



### 2. Créer Le projet MVC et le contrôleur :

Ensuite, créer un projet MVC. Puis ajouter un contrôleur nommé « ProductController » :



### 3. Affichage des données :

- Dans 1 : importer les package Data et SqlClient.
- Dans 2 : Déclaration de la chaine de connexion
- Dans 3 : Créer la DataTable pour y stocker les résultats
- Dans 4 : Exécuter la requête de sélection et stocker les résultats dans la DataTable

```

ProductController.cs - Browse - Your ASP.NET application
MVCWEF.Controllers.ProductController - Index()

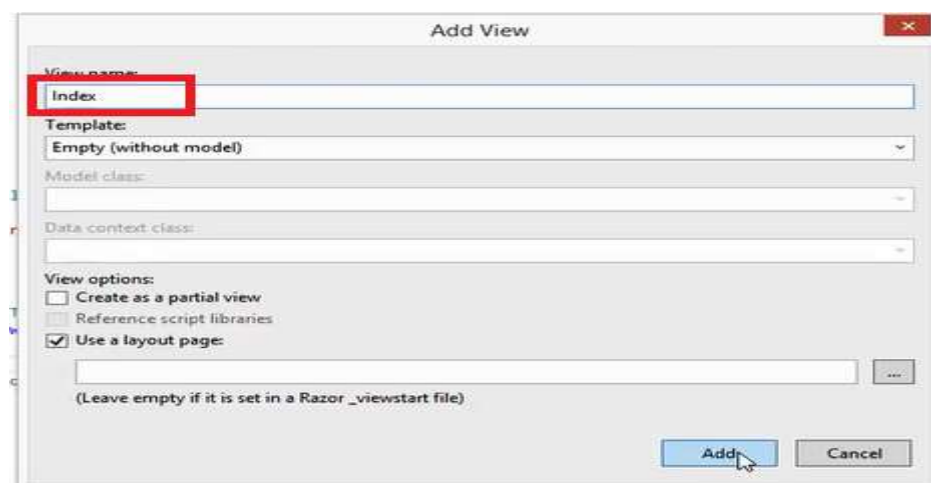
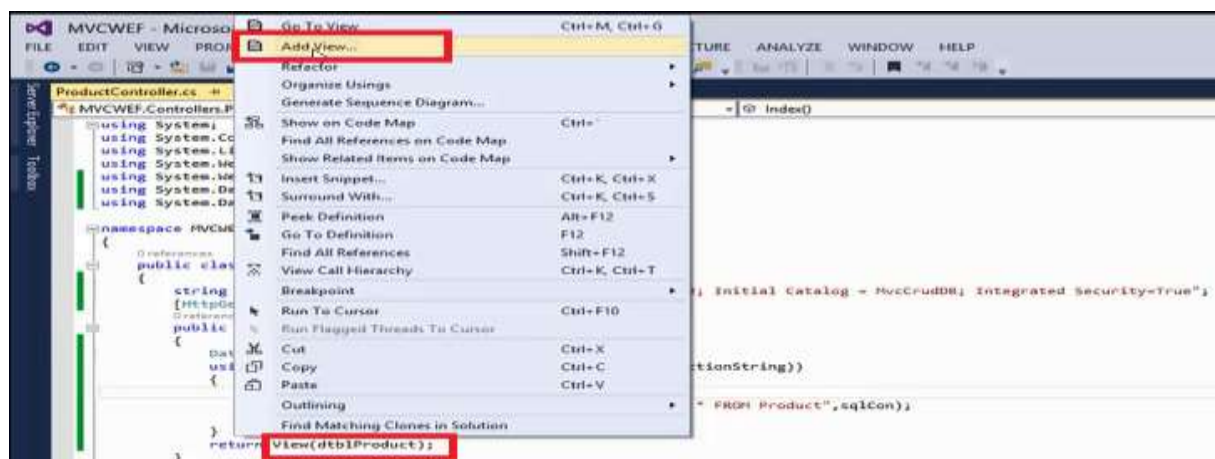
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
1 using System.Data;
using System.Data.SqlClient;

namespace MVCWEF.Controllers
{
    public class ProductController : Controller
    {
2         string connectionString = @"Data Source = (local)\sql2012; Initial Catalog = MvcCrudDB; Integrated Security=True";

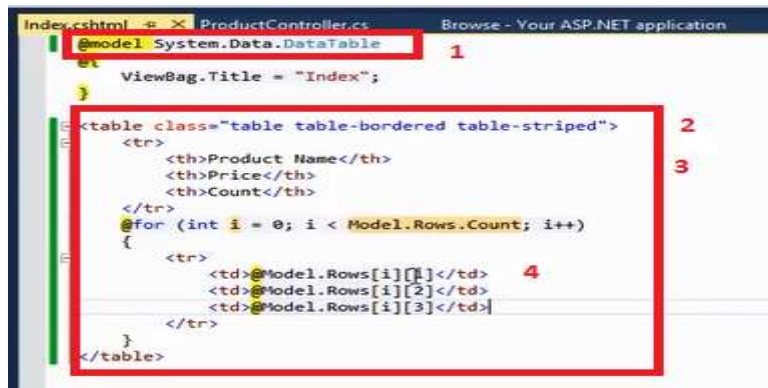
        [HttpPost]
        public ActionResult Index()
        {
3             DataTable dtblProduct = new DataTable();
            using (SqlConnection sqlCon = new SqlConnection(connectionString))
            {
4                 sqlCon.Open();
                SqlDataAdapter sqlDa = new SqlDataAdapter("SELECT * FROM Product", sqlCon);
                sqlDa.Fill(dtblProduct);
5             }
            return View(dtblProduct);
        }

        // GET: /Product/Details/5
        public ActionResult Details(int id)
        {
            return View();
        }
    }
}
  
```

Clic droit sur « View » et ajouter une vue pour afficher les résultats :

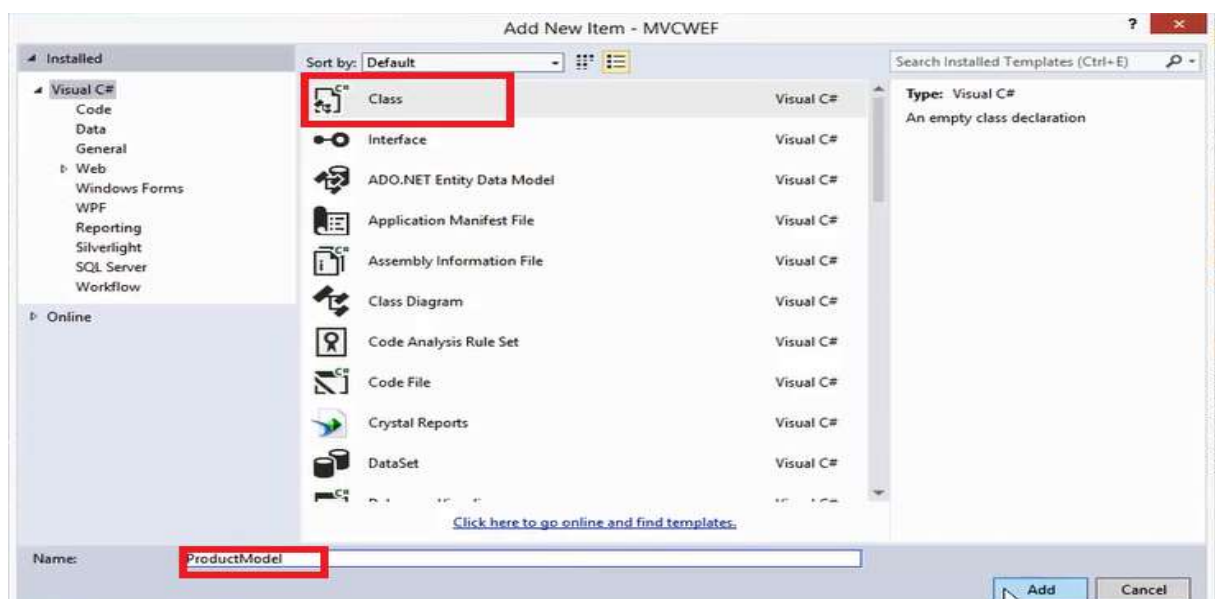
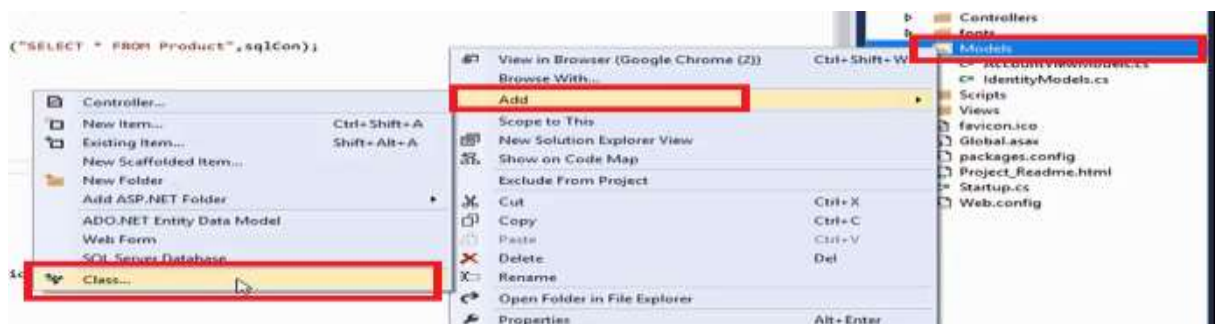
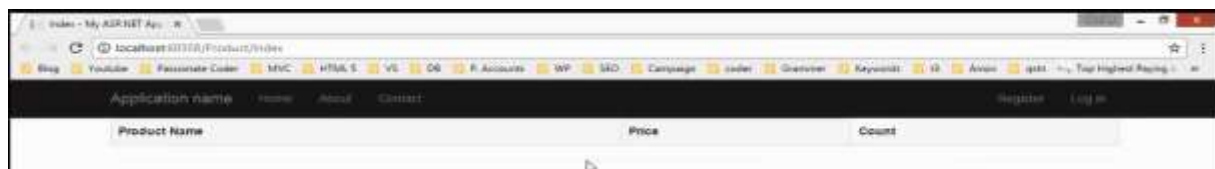


- Dans 1 : lier la vue « index » à la DataTable
- Dans 2 : création de la table d'affichage
- Dans 3 : définir les noms des colonnes
- Dans 4 : insérer les valeurs des enregistrements dans la table

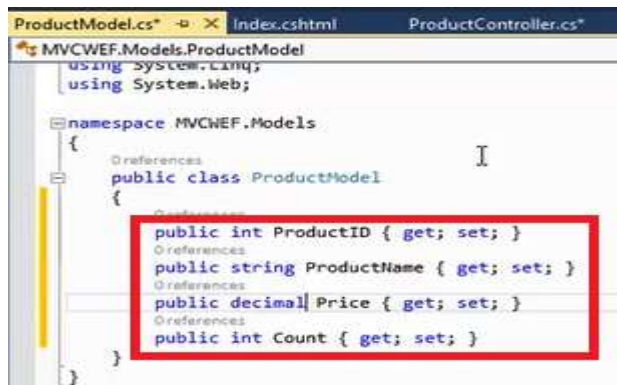


#### 4. Créer un produit (ajouter) :

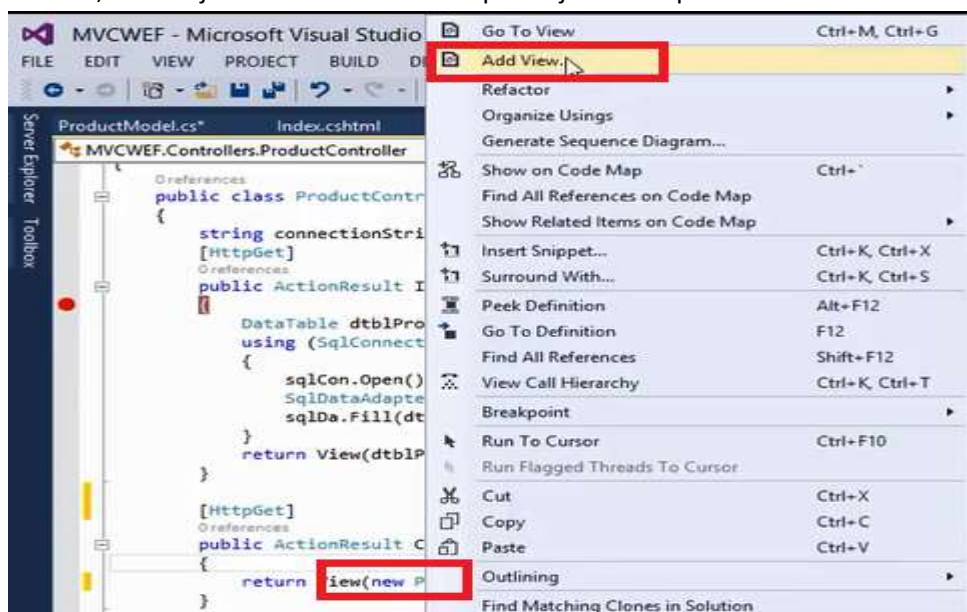
Ensuite il faut créer un modèle de données pour remplir la DataTable à partir de la base de données et afficher les résultats dans la table de la vue « index ».



La classe nommée « ProductModel.cs », elle contient l'ensemble des actions. Il faut déclarer les propriétés GET et SET de chaque attribut de la classe « ProductModel.cs »



Ensuite, il faut ajouter la vue « create » pour ajouter un produit :



Cette vue se présente sous forme d'un formulaire qui contient des zones de saisie de chaque attribut de la classe ProductModel.



```

@model MVCWEF.Models.ProductModel

@{
    ViewBag.Title = "Create";
}

<h2>Create</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>ProductModel</h4>
        <hr />
        @Html.ValidationSummary(true)

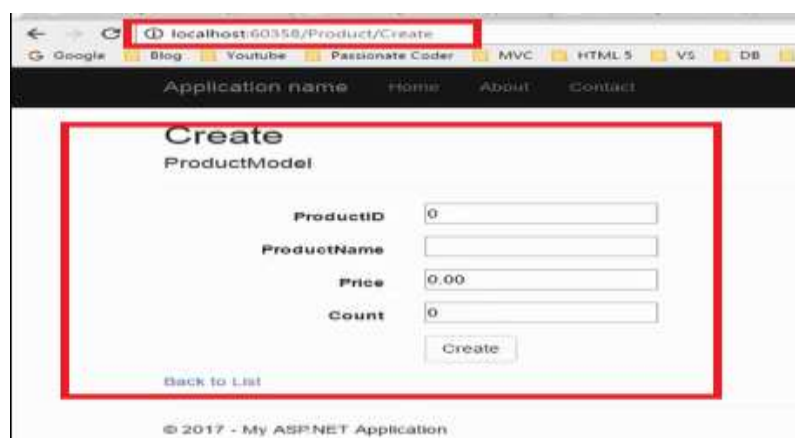
        <div class="form-group">
            @Html.LabelFor(model => model.ProductID, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.ProductID)
                @Html.ValidationMessageFor(model => model.ProductID)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.ProductName, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.ProductName)
                @Html.ValidationMessageFor(model => model.ProductName)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Price, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Price)
                @Html.ValidationMessageFor(model => model.Price)
            </div>
        </div>
    </div>
}

```

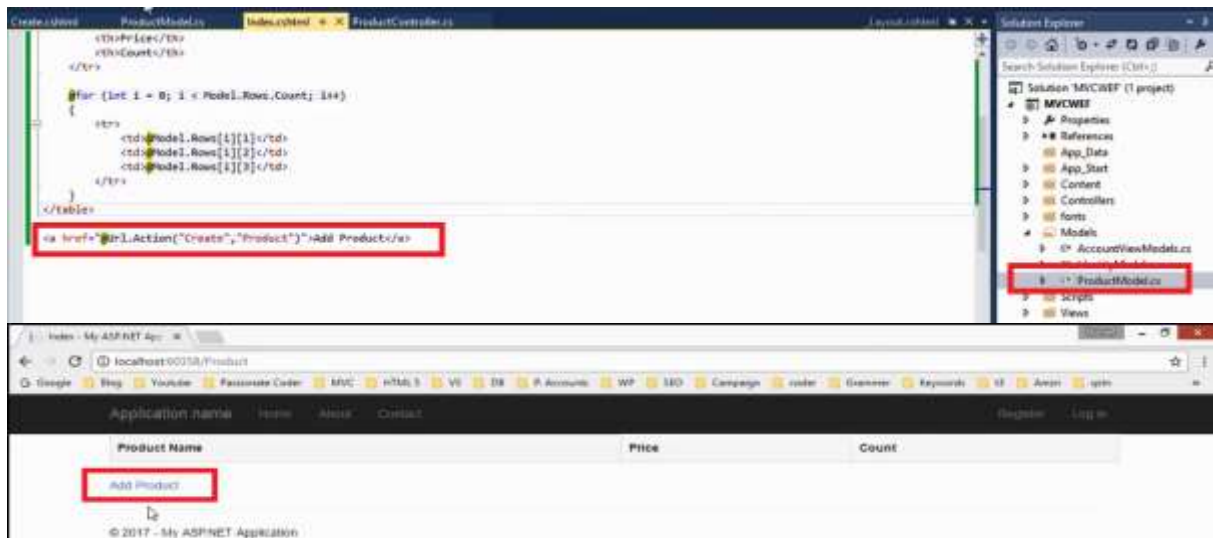
Générer le projet et voilà la vue « Create »



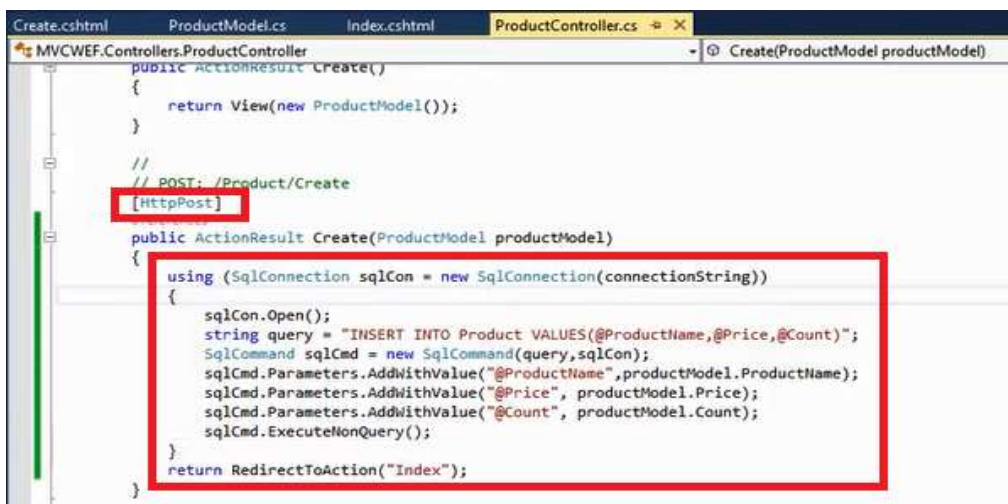
Voici le formulaire par défaut proposé par l'application, nous voulons changer la mise en forme et supprimer le ProductID et sa zone de texte.



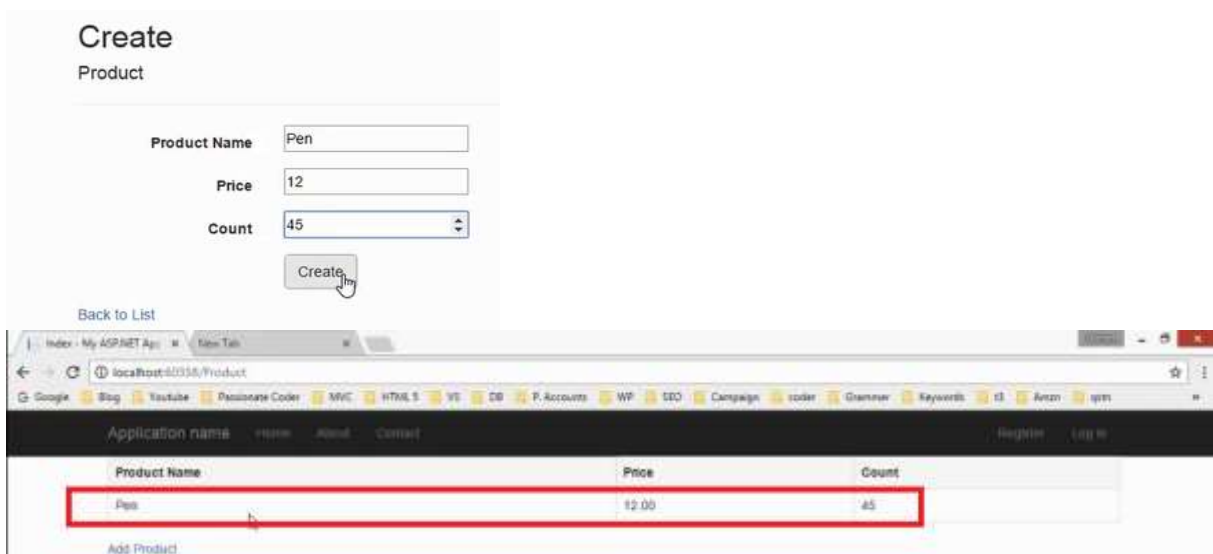
Puis, on ajoute le lien pour l'action **Create()** du contrôleur « **Product** » le lien s'appelle « **Add Product** »



Dans l'action `Create()` de `ProductController` ([HttpPost] :



Régénérer le projet :



## 5. Rechercher (éditer) un produit :

```
// GET: /Product/Edit/5
public ActionResult Edit(int id)
{
    ProductModel productModel = new ProductModel();
    DataTable dtblProduct = new DataTable();
    using (SqlConnection sqlCon = new SqlConnection(connectionString))
    {
        sqlCon.Open();
        string query = "SELECT * FROM Product Where ProductID = @ProductID";
        SqlDataAdapter sqlDa = new SqlDataAdapter(query, sqlCon);
        sqlDa.SelectCommand.Parameters.AddWithValue("@ProductID", id);
        sqlDa.Fill(dtblProduct);

        if (dtblProduct.Rows.Count == 1)
        {
            productModel.ProductID = Convert.ToInt32(dtblProduct.Rows[0][0].ToString());
            productModel.ProductName = dtblProduct.Rows[0][1].ToString();
            productModel.Price = Convert.ToDecimal(dtblProduct.Rows[0][2].ToString());
            productModel.Count = Convert.ToInt32(dtblProduct.Rows[0][3].ToString());
            return View(productModel);
        }
        else
        {
            return RedirectToAction("Index");
        }
    }
}
```

La méthode (action) Edit consiste à rechercher un produit par son id

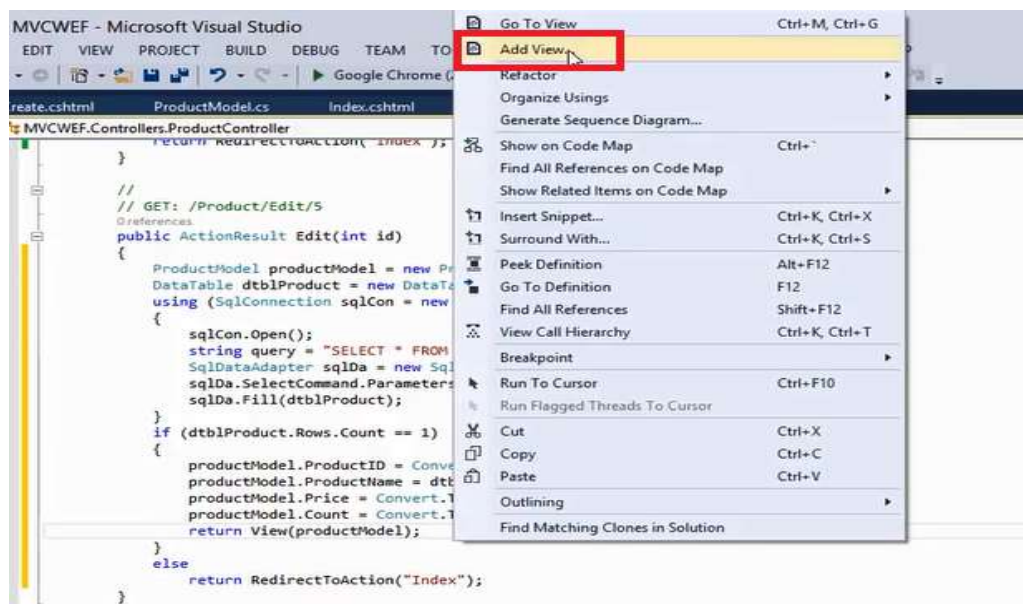
on crée un produit et un datatable ou on va stocker les résultats de l'ordre select dans la datatable

on cherche le produit par son id et on charge le résultat dans la datatable

on teste s'il existe une ligne de résultat on va l'afficher

sinon on se redirige vers la page index

Ensuite, on doit ajouter la vue pour éditer les résultats de la recherche :



Et voici la vue Edit.cshtml

```

@model MVCMEF.Models.ProductModel
@{
    ViewBag.Title = "Edit";
}

<h2>Edit</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>ProductModel</h4>
        <hr />
        @Html.ValidationSummary(true)
        <div class="form-group">
            @Html.LabelFor(model => model.ProductID, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.ProductID)
                @Html.ValidationMessageFor(model => model.ProductID)
            </div>
        </div>

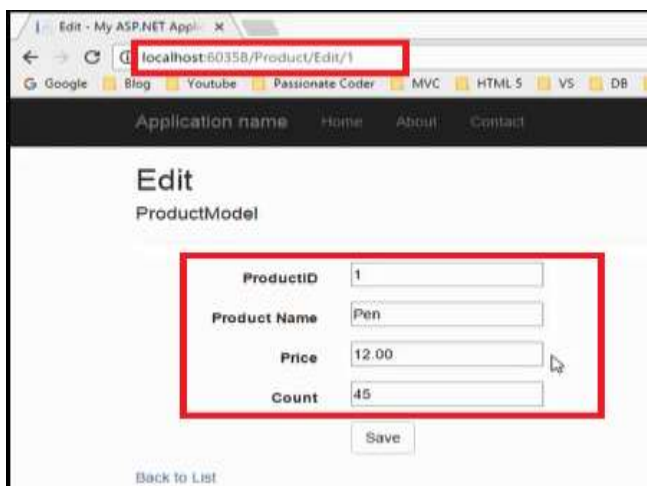
        <div class="form-group">
            @Html.LabelFor(model => model.ProductName, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.ProductName)
                @Html.ValidationMessageFor(model => model.ProductName)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Price, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Price)
                @Html.ValidationMessageFor(model => model.Price)
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Count, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Count)
                @Html.ValidationMessageFor(model => model.Count)
            </div>
        </div>
    </div>
}

```

Régénérer le projet et taper l'URL .../Edit/1 qui permet d'éditer les informations du produit Id=1 :



On veut alors modifier « ProductModel » par « Product » et on veut cacher le champ productID

```

@model MVCMEF.Models.Product
@{
    ViewBag.Title = "Edit";
}

<h2>Edit</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Product</h4>
        <hr />
        @Html.ValidationSummary(true)
        @Html.HiddenFor(model => model.ProductID)
        <div class="form-group">
            @Html.LabelFor(model => model.ProductName, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.ProductName)
                @Html.ValidationMessageFor(model => model.ProductName)
            </div>
        </div>
    </div>
}

```

on change le nom de productModel to product

on cache le productID

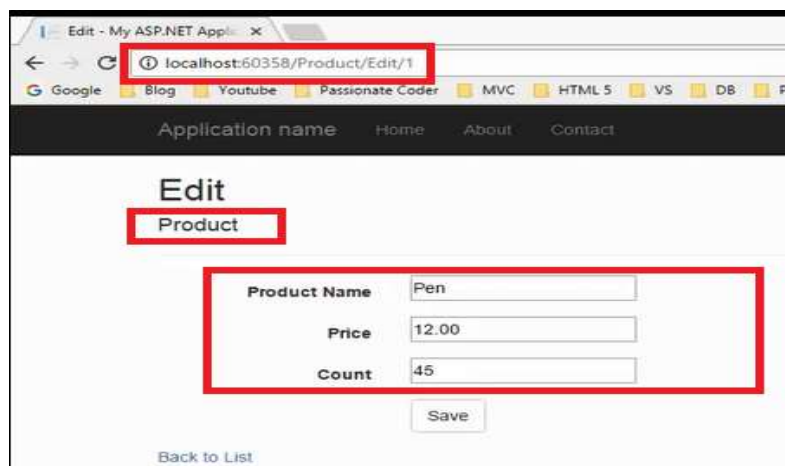
On ajoute le bouton « Edit » dans la 4<sup>ème</sup> colonne de la table :



```

<table class="table table-bordered table-striped" >
  <tr>
    <th>Product Name</th>
    <th>Price</th>
    <th>Count</th>
    <th></th>
  </tr>
  <tr>
    <td>@Model.Rows[i][1]</td>
    <td>@Model.Rows[i][2]</td>
    <td>@Model.Rows[i][3]</td>
    <td>
      <a href="@Url.Action("Edit", "Product", new {id=@Model.Rows[i][0]})">Edit</a>
    </td>
  </tr>
</table>

```



### 1. Modifier (update) un produit :

Maintenant si on veut enregistrer des modifications sur le produit édité, nous passons au contrôleur et nous rectifions la méthode Edit pour [HttpPost]

```

//
// POST: /Product/Edit/5
[HttpPost]
public ActionResult Edit(int id, FormCollection collection)
{
    try
    {
        // TODO: Add update logic here
        return RedirectToAction("Index");
    }
    catch
    {
        return View();
    }
}

```

Elle devient :

```

public ActionResult Edit(ProductModel productModel)
{
    using (SqlConnection sqlCon = new SqlConnection(connectionString))
    {
        sqlCon.Open();
        string query = "UPDATE Product SET ProductName = @ProductName , Price= @price , Count = @Count Where ProductID = @ProductID";
        SqlCommand sqlCmd = new SqlCommand(query, sqlCon);
        sqlCmd.Parameters.AddWithValue("@ProductID", productModel.ProductID);
        sqlCmd.Parameters.AddWithValue("@ProductName", productModel.ProductName);
        sqlCmd.Parameters.AddWithValue("@Price", productModel.Price);
        sqlCmd.Parameters.AddWithValue("@Count", productModel.Count);
        sqlCmd.ExecuteNonQuery();
    }
    return RedirectToAction("Index");
}

```

on effectue les modifications du produit édité

les nouvelles valeurs à insérer sont obtenues des zones de texte de la vue edit

exécution de la requête

Régénérer le projet et faites des modifications puis « Save »

**Edit Product**

Product Name

Price

Count

[Back to List](#)

Product Name	Price	Count	
Pen 1	12.00	100	Edit

### 1. Supprimer (Delete) un produit :

Maintenant pour supprimer un Produit nous allons dans l'action « delete » dans le ProductController

```

MVCWEF.Controllers.ProductController
Delete(int id)
{
    // GET: /Product/Delete/5
    public ActionResult Delete(int id)
    {
        using (SqlConnection sqlCon = new SqlConnection(connectionString))
        {
            sqlCon.Open();
            string query = "DELETE FROM Product Where ProductID = @ProductID";
            SqlCommand sqlCmd = new SqlCommand(query, sqlCon);
            sqlCmd.Parameters.AddWithValue("@ProductID", id);
            sqlCmd.ExecuteNonQuery();
        }
        return RedirectToAction("Index");
    }
}

```

```

<table class="table table-bordered table-striped">
<tr>
<th>Product Name</th>
<th>Price</th>
<th>Count</th>
<th></th>
</tr>
<tr>
<td>@Model.Rows[i][1]</td>
<td>@Model.Rows[i][2]</td>
<td>@Model.Rows[i][3]</td>
<td>
<a href="@Url.Action(\"Edit\", \"Product\", new {id=@Model.Rows[i][0]})">Edit</a>
<a href="@Url.Action(\"Delete\", \"Product\", new {id=@Model.Rows[i][0]})">Delete</a>
</td>
</tr>
</table>

```

Ajout de lien de suppression

Après exécution du projet, nous avons le bouton « Delete » pour supprimer un enregistrement



On peut ajouter un nouveau Product et les supprimer par le bouton « Delete »

