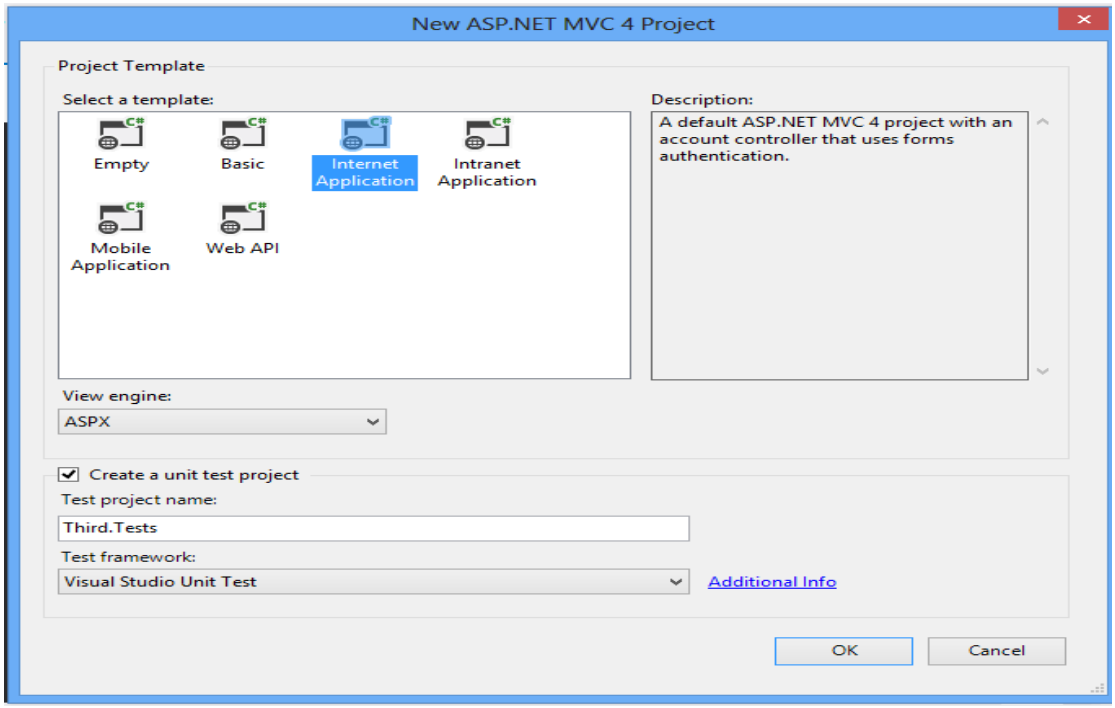
بسم الله الرحمن الرحيم

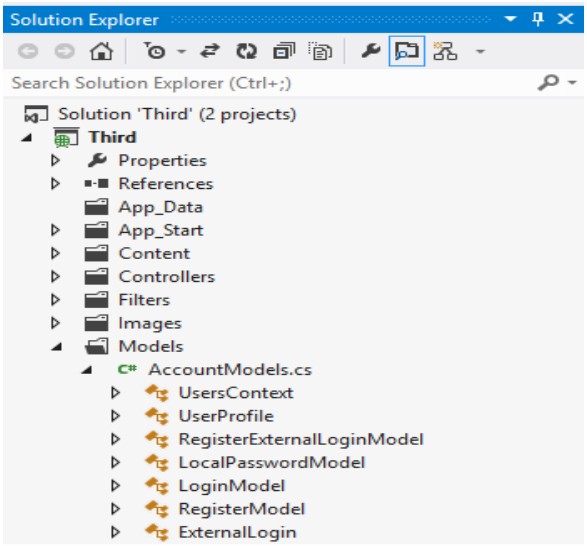
: Models

هو عبارة عن كائن عادي (Object) له خصائص (Properties) وسلوكيات (Behavior) ، يمثل (Business Logic) و (Data Logic) ، مثلاً يمكن أستخدامه لأرسال واسترجاع البيانات من قاعدة البيانات .

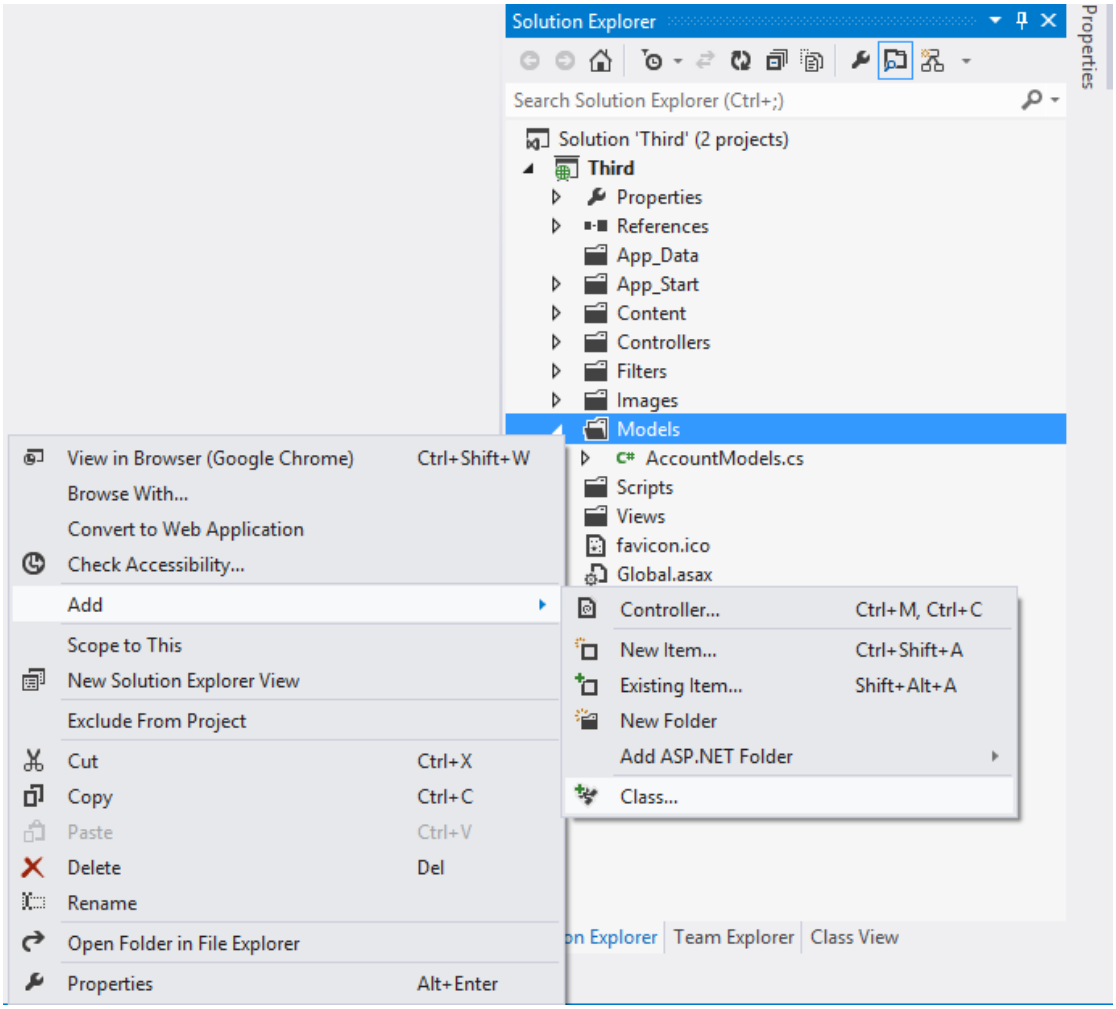
لنعمل مشروع جديد بأسم (Third) :



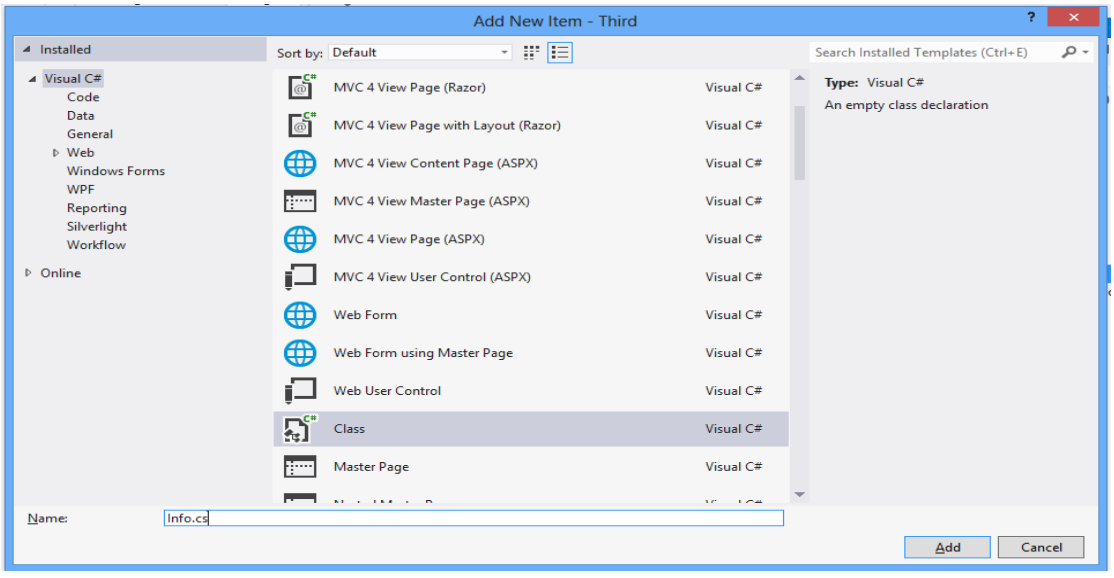
بصورة أفتراضية فأن الفيجوال ستوديو سيعمل لك مجلد (Models)يحوي فقط على مودل واحد (AccountModels) وهذا المودل خاص بـ (Authentication) :



كما تلاحظ اللون الاخضر يشير بأن هذا مودل ، والجوزي الفاتح يشير الى ان هذا (Method) ، دعونا نضيف مودل نسميه (Info.cs) ، من خلال (Right Click) على المجلد (Models) وثم (Add) نختار (Class) :



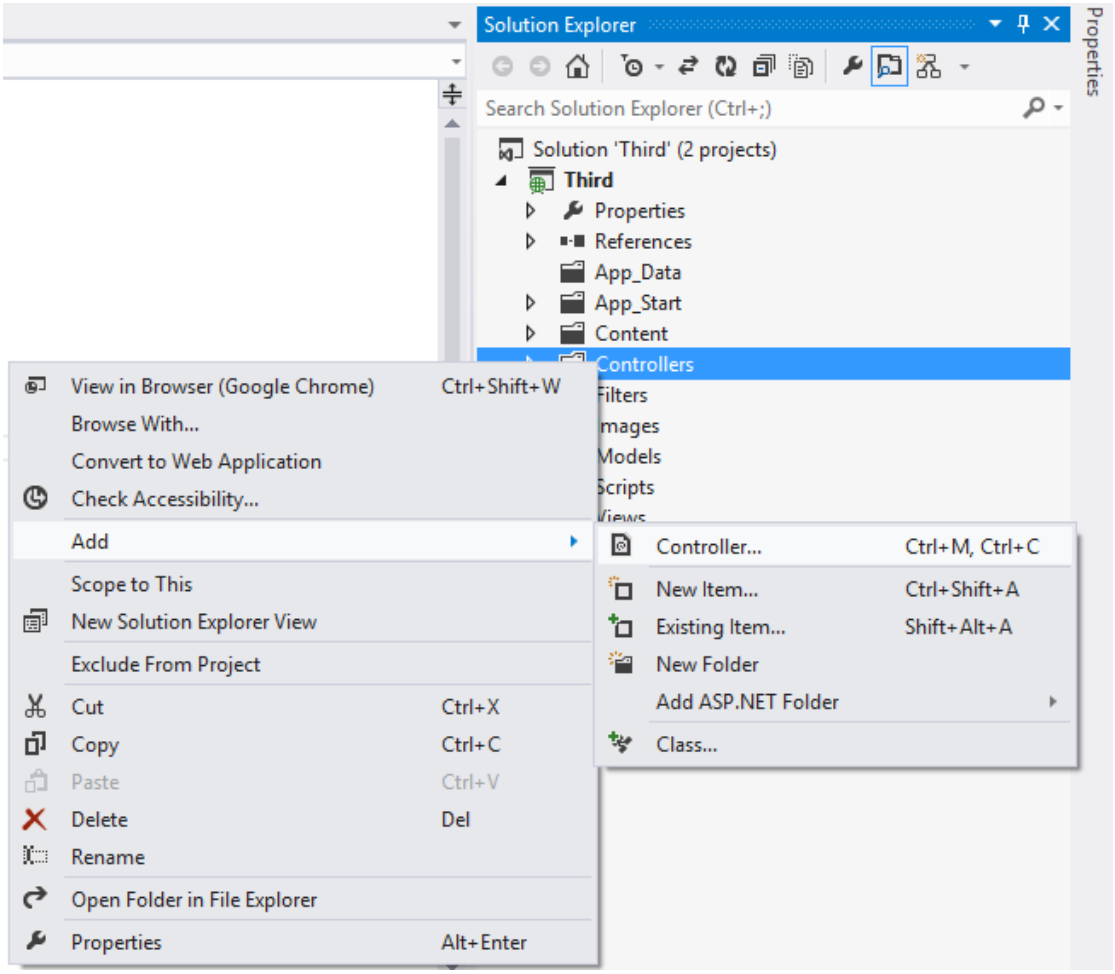
ثم :



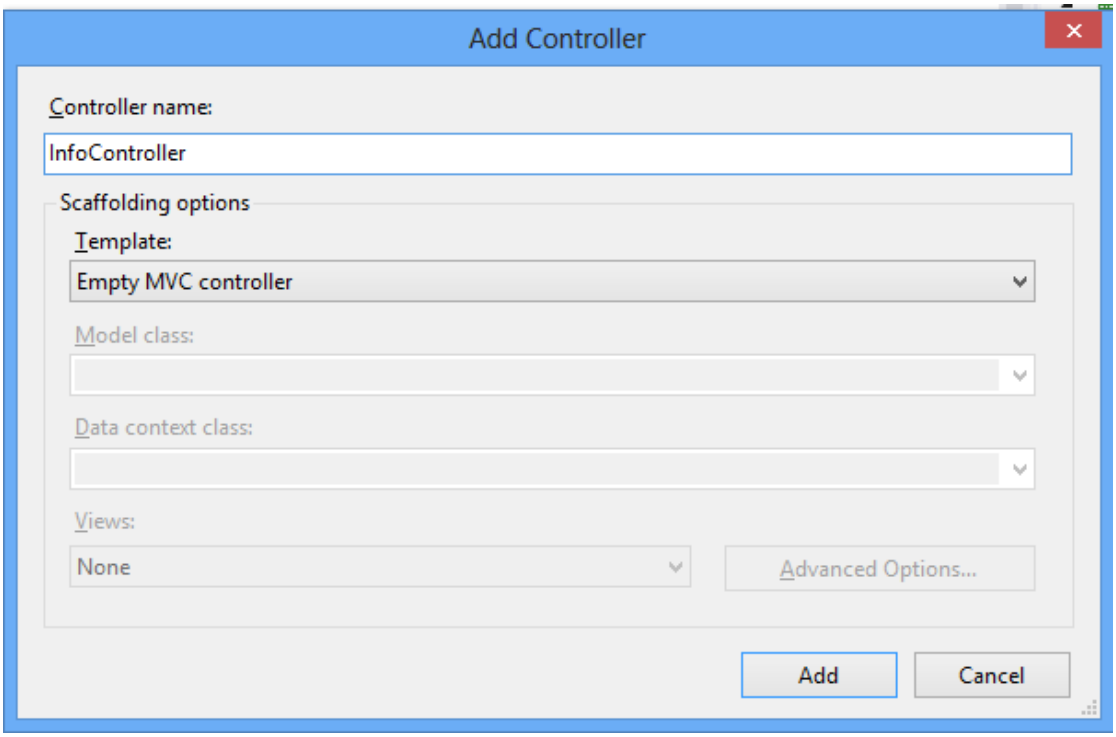
سنضيف فيه ٣ متغيرات (خصائص) :

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

namespace Third.Models
{
    public class Info
    {
        public int ID { get; set; }
        public string Category { get; set; }
        public DateTime Create_Date { get; set; }

    }
}
```

ثم نضيف كونترولر ، (Right Click) على المجلد (Controllers) ونختار (Add) ثم (Controller) :



يجب ان يكون الكونترولر بنفس أسم المودل (Info) :

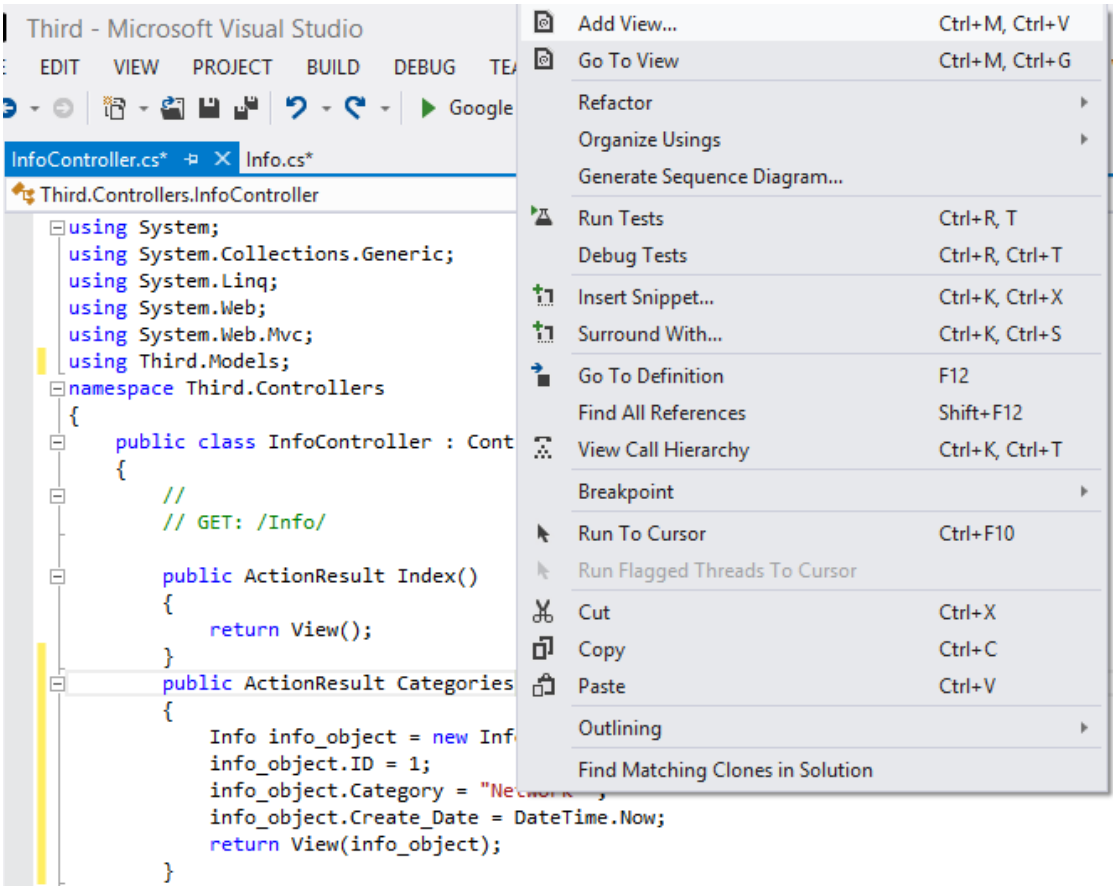


نعمل أكشن جديد (Categories) ونعرف داخله (Instance) من المودل (Info) لكي نملأه بالمعلمات

.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Third.Models;
namespace Third.Controllers
{
    public class InfoController : Controller
    {
        //
        // GET: /Info/

        public ActionResult Index()
        {
            return View();
        }
        public ActionResult Categories()
        {
            Info info_object = new Info();
            info_object.ID = 1;
            info_object.Category = "Network ";
            info_object.Create_Date = DateTime.Now;
            return View(info_object);
        }

    }
}
```

كما تلاحظ بالكود أعلاه بعد أن ملأنا المودل بالبيانات قمنا بأرساله للفيو ، الان يجب أن نعمل فيو بأسم الاكشن (Categories) ، من (Right Click) على الاكشن ونختار (Add View) :



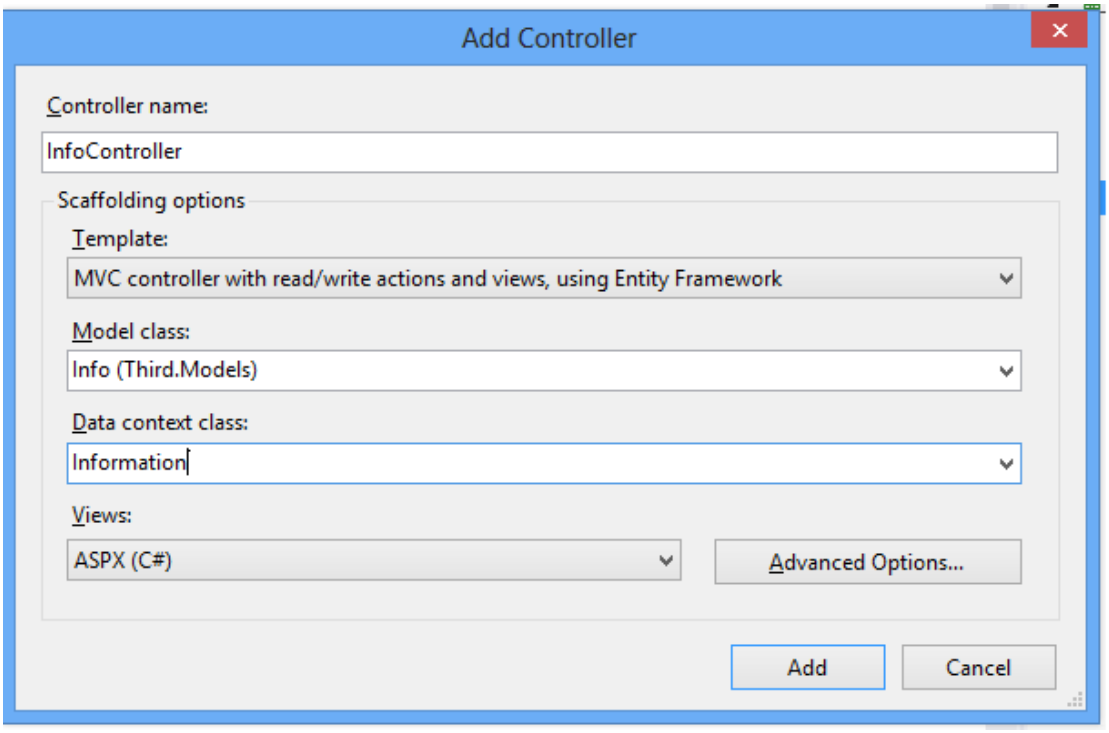
ونستدعي المودل في الفيو :

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage<dynamic>" %>

<asp:Content ID="Content1" ContentPlaceHolderID="TitleContent"
runat="server">
    Categories
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" runat="server">
<h1> <%= Model .ID +" : "+Model .Category %></h1>
    <br />
    <p><%= Model .Create_Date %></p>
</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderID="FeaturedContent"
runat="server">
</asp:Content>

<asp:Content ID="Content4" ContentPlaceHolderID="ScriptsSection"
runat="server">
</asp:Content>
```

بعد الأستعراض سيكون العرض هكذا :

حتى هنا أعتقد الامور بسيطة ، لو تعمقنا أكثر فأنها سوف تزداد تعقيداً بعض الشي ، دعونا نعمل مثال آخر لكن هذه المرة سوف نبني قاعدة بيانات فيها جدول (Info) ، ولكي نبني قاعدة بيانات امامنا طريقين كلاهما يوصلنا لنفس الهدف ، الطريق الاولى أما ان ننشئ قاعدة بيانات وفيها جدول يدوياً ثم نستخدم تقنية (Entity Framework) لتولد لنا مودل من نفس الجدول ، او نستخدم (Code First) وهي أن نكتب كود المودل وأعتماداً عليه يتم توليد جدول اوتوماتيكياً .

الان لو أردنا أستخدام (Code First) وذلك ببناء جدول في قاعدة البيانات من المودل يكون من خلال أضافة كونترولر ، لكن قبل هذا يجب ان نحذف الكونترولر (Info) ثم (Right Click) على المجلد (Controller) ونضيف كونترولر جديد . :

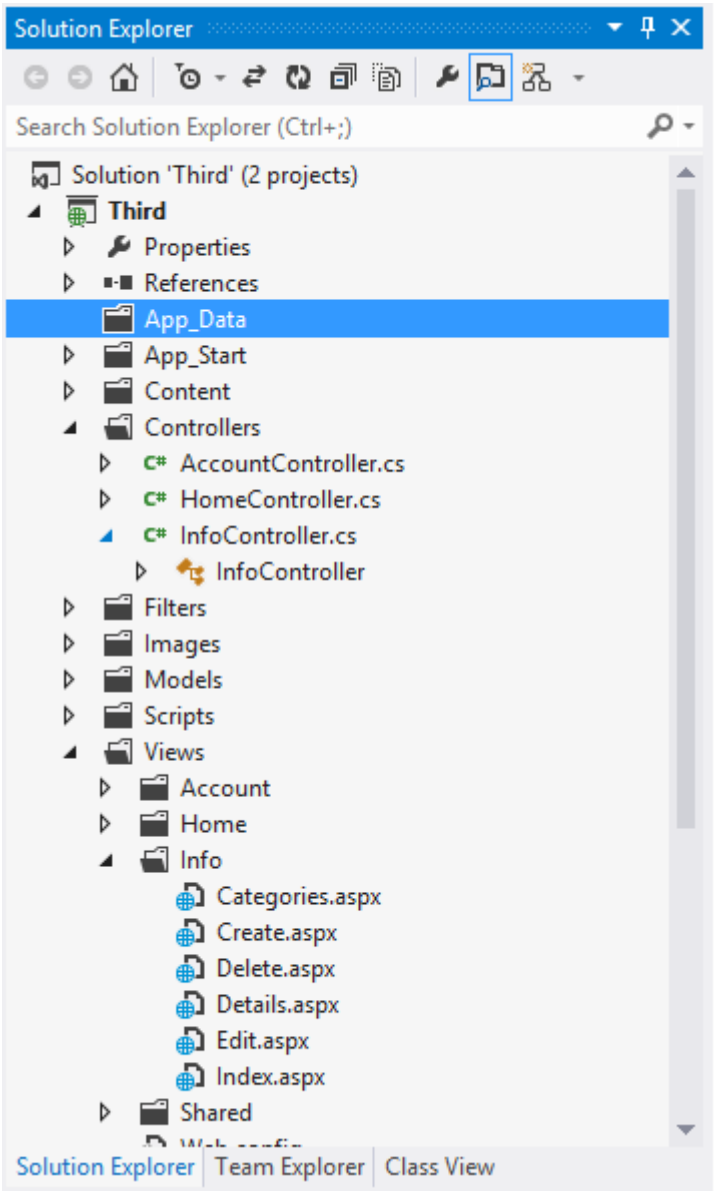


كما تلاحظ فأن الكونترولر(Info) سنختار له من (Template) (MVC Controller with read /write action views, using entity Framework) بينما قمنا بتسمية (Data Context Class) له (Information)

وهي تمثل اسم قاعدة البيانات ، وأخترنا المودل (Info) .

انقر على (add) :

كما ستلاحظ انه قام بأضافة كونترولر من المودل الموجود فيه عدة أكشن وكذلك أضاف لكل أكشن فيو خاص به . :



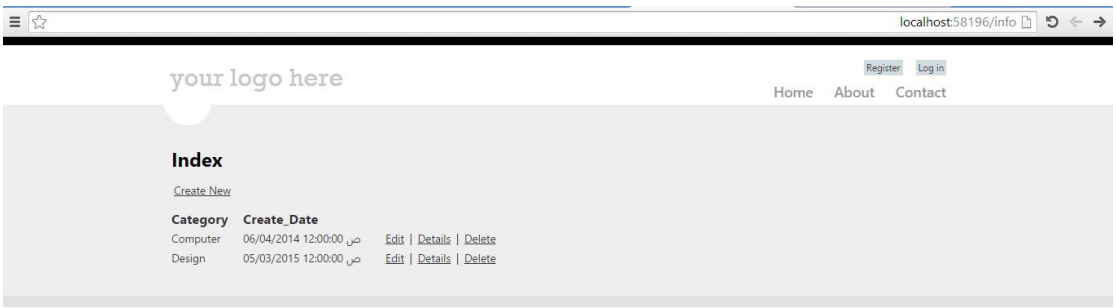

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using Third.Models;

namespace Third.Controllers
{
    public class InfoController : Controller
```

```
{
    private Information db = new Information();

    //
    // GET: /Info/

    public ActionResult Index()
    {
        return View(db.Infoes.ToList());
    }

    //
    // GET: /Info/Details/5

    public ActionResult Details(int id = 0)
    {
        Info info = db.Infoes.Find(id);
        if (info == null)
        {
            return HttpNotFound();
        }
        return View(info);
    }

    //
    // GET: /Info/Create

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

    //
    // POST: /Info/Create

    [HttpPost]
    public ActionResult Create(Info info)
    {
        if (ModelState.IsValid)
        {
            db.Infoes.Add(info);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        return View(info);
    }

    //
    // GET: /Info/Edit/5

    public ActionResult Edit(int id = 0)
    {
        Info info = db.Infoes.Find(id);
        if (info == null)
        {
            return HttpNotFound();
        }
        return View(info);
    }

    //
```

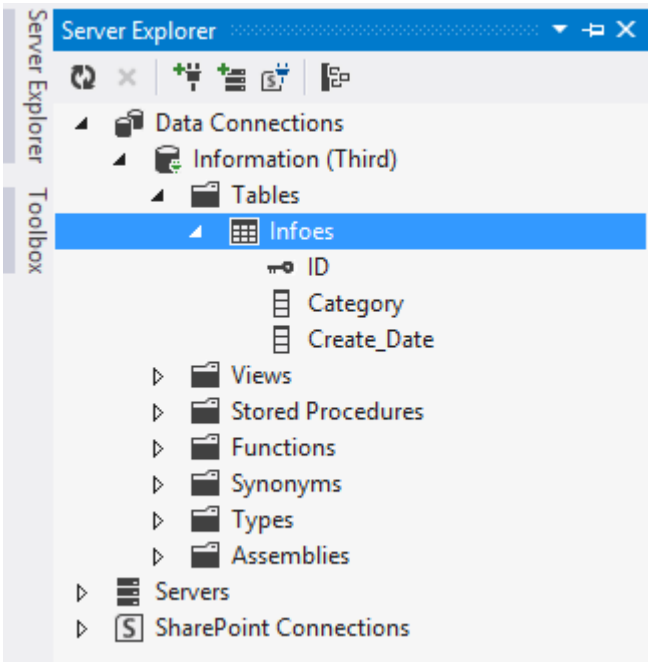
```
        // POST: /Info/Edit/5

        [HttpPost]
        public ActionResult Edit(Info info)
        {
            if (ModelState.IsValid)
            {
                db.Entry(info).State = EntityState.Modified;
                db.SaveChanges();
                return RedirectToAction("Index");
            }
            return View(info);
        }

        //
        // GET: /Info/Delete/5

        public ActionResult Delete(int id = 0)
        {
            Info info = db.Infoes.Find(id);
            if (info == null)
            {
                return HttpNotFound();
            }
            return View(info);
        }

        //
        // POST: /Info/Delete/5

        [HttpPost, ActionName("Delete")]
        public ActionResult DeleteConfirmed(int id)
        {
            Info info = db.Infoes.Find(id);
            db.Infoes.Remove(info);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        protected override void Dispose(bool disposing)
        {
            db.Dispose();
            base.Dispose(disposing);
        }
    }
}
```

وبالامكان التعديل على أي أكشن .

بعد التنفيذ :



بأمكانك الان ان تضيف وتعدل البيانات وحتى الحذف ، وقد أضاف لنا قاعدة بيانات بأسم (Information) وهو أسم (Data Context) التي وضعناها سابقآ وفيها جدول (Infoes) :



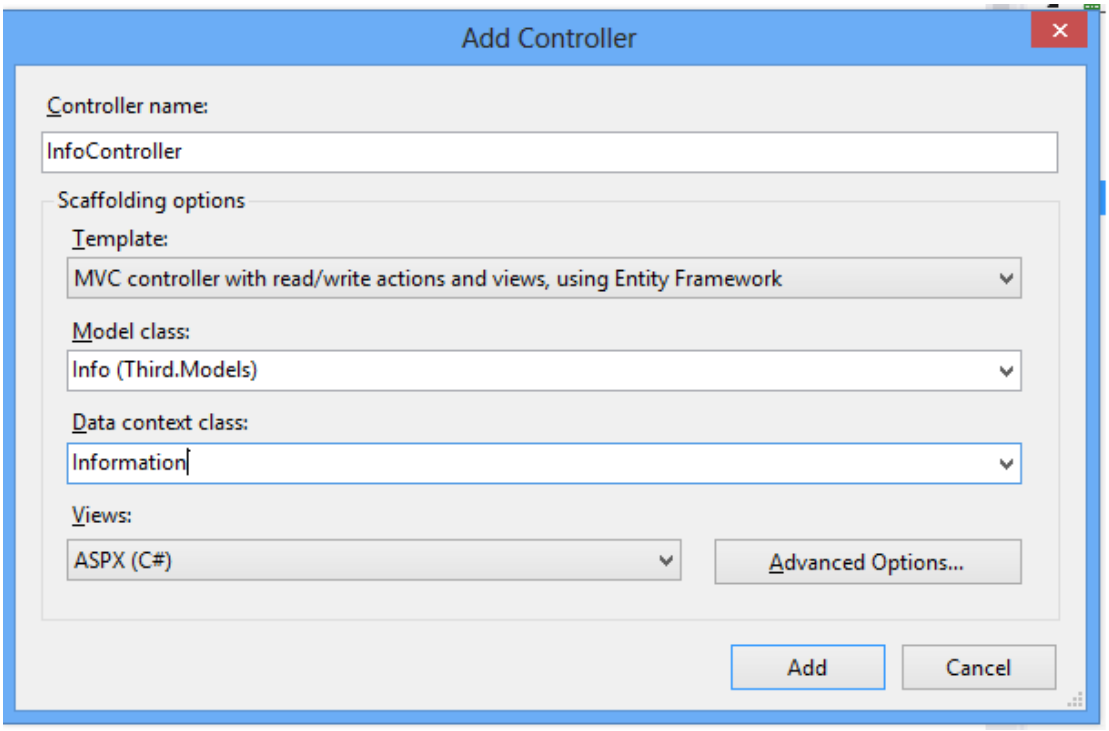
هناك عدة أمور ربما تكون غامضة في المثال أعلاه ، سنتطرق لبعضها الان ومنها :

Entity Framewrok :

الـ (MVC) ليس فيها نوع محدد من (Data Access) تستخدمها بصورة مباشرة للتعامل مع قاعدة البيانات وأنما هناك العديد من مكتبات (Data Access) بأمكانك أن تستخدمها للعمل مع قاعدة البيانات واحده من هذه المكتبات هي (Entity Framework) ومختصرها (EF) وهذه الاخيرة تمتاز بالمرونة وتساعد المطوير بالاستعلام والتعديل على البيانات المخزنة في قاعدة البيانات بطريقة الكائنات الموجهة (Object Oriented) ، وهي جزء من بيئة (.Net Framework) ، الـ (EF) تعطينا عدة طرق تقنية مختلفة لتعريف مودل البيانات (Data Model) واحدة من هذه الطرق تسمى (Code First) وهذه الطريقة تعني انشاء قاعدة البيانات بصورة تلقائية من المودل وكائناتها مثل (الجداول و العلاقات الخ) عند تنفيذ البرنامج بمعنى أوضح اننا نكتب كود في المودل (حسب شروط معينة) وأعتمادآ على هذا الكود سيتم انشاء قاعدة البيانات ووجداولها .

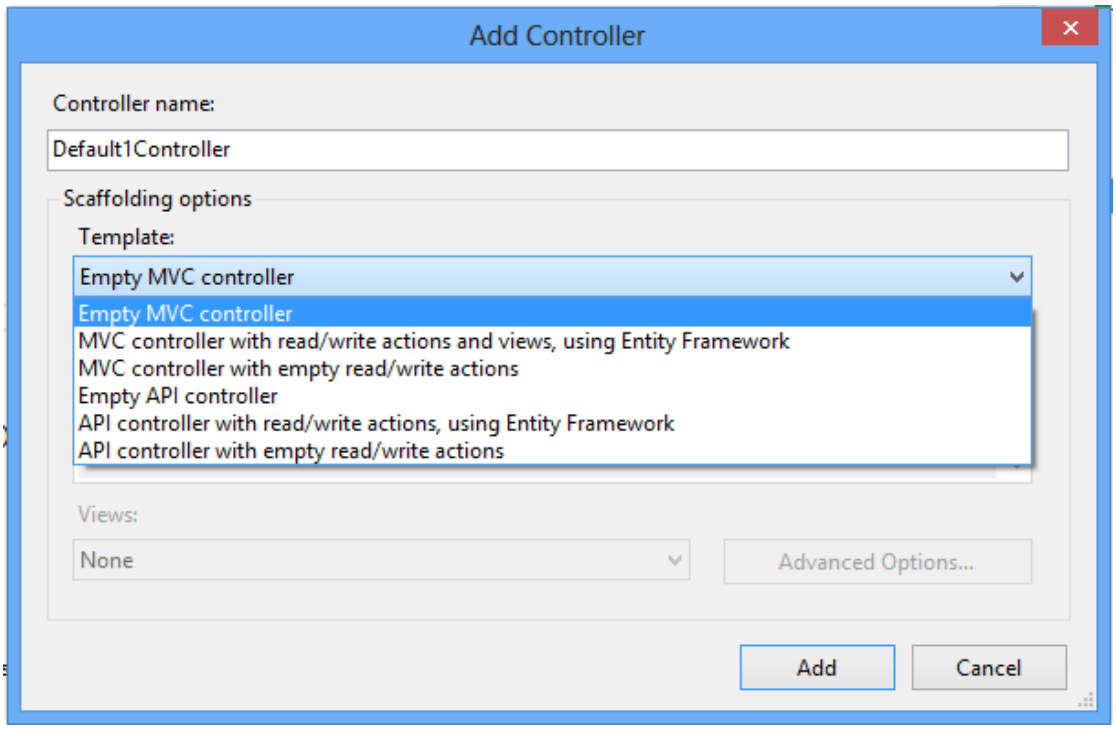
Scaffolding :

في (MVC) فأن (Scaffolding) تستطيع ان تولد لك كود معياري تحتاجه في علميات الانشاء والقراءة والتعديل والحذف (CRUD) في التطبيق .



قوالب (Scaffolding Templates) تستطيع أن تختير نوع التعاريف للمودل وأعتمادا عليه تولد لنا كونترولر مع الفيو الخاص به ، مثلا هي ستعرف ماهو أسم الكونترولر وما أسم الفيو ، وماهو الكود الضروري لعناصرهم ، وفي أي مجلد ستضعهم في التطبيق .

لا تعتقد أن (Scaffolding) تبني لك تطبيق كامل وانما هي تساعدك في بناء بعض العناصر بصورة افتراضية تستطيع التعديل عليها بما يتلائم مع تطبيقك ، هناك عدة انواع من القوالب هي :



- ☒ **Empty MVC Controller : هذا النوع يضيف لنا كونترولر بأسم أنت تحدده داخل المجلد (Controllers) داخله أكشن واحد أسمه (Index) فارغ من الكود ولا ينشئ أي فيو له.**
- ☒ **MVC Controller with read/write action and view using EF : - هذا القالب ليس فقط يولد لك كونترولر داخله الاكشنات ((Index ,Create ,Details , Edit , Delete) وانما يولد لك معها جميع الفيو والكود الضروري لأرسال واستلام العملومات من قاعدة البيانات .**
- ☒ **MVC Controller with empty read /write action :- هذا القالب يضيف لنا كونترولر للشمروع بداخله عدة أكشنات هي (Index ,Create ,Details , Edit , Delete) وينشئ لكل أكشن فيو خاص به ، ولكل أكشن كود خاص به يمكن التعديل عليه .**
- ☒ **API Controller with empty read/write action : - هذا القالب يضيف لنا كونترولر مشتق من (API Controller) نستطيع أستخدامة في بناء تطبيقات (WEB API) (لنا كلام مفصل حول هذا الموضوع فيما بعد) .**

الى هنا أنتهى الدرس الثالث .

نلتقي أن شاء الله في الدرس الرابع

تحياتي

محمد الساعدي

https://mohamediddan.wordpress.com/