

# FIT5032 - Internet Applications Development

## WEEK 06 - VALIDATION

Last updated: 14th July 2018

Author: Jian Liew

## Housekeeping

Before we begin, it is highly recommended for you to use your own personal computer for this subject. If you are planning to use the Monash computers, it is highly recommended to save all your work properly. Towards the end of the semester, there will be a portfolio submission where you need to showcase all the work you have done so far. To prevent difficulties during this portfolio submission, it is suggested that you keep all your work neat and organised based on a **week by week structure**.

## Tutorial Structure

The tutorials in this unit are designed to be of a **self-paced and self-taught structure**. So, the aim of your tutor is to aid you in your learning experience. He or she will not go through all the materials that are covered in the labs and will **not do the exercise one by one as a class**. If help is needed, you can either post on the Moodle forums or you can email your tutor asking for clarification. This is slightly different in comparison to other units, where your tutor will lead the discussions. **Please take note of this, you will only be provided help if you make it known that you need help.**

## Objectives

Estimated Time To Complete - 1 hour

Upon the completion of this tutorial, you will gain a basic understanding of

- What a ViewModel is
- How does Visual Studio simplifies the creation of forms.

## DoubtFire Submission

- T6.1 A document explaining what jQuery unobtrusive and how it works in combination with ViewModels.
- T6.2 [Updated 09/08/2017] A document with screenshots showing your week 6 application with validation running. (in a PDF document)

## The View Model

The ViewModel is a class that is created for each form to represent the values of the form.

Normally, it is good practice to create a ViewModel for each and every form.

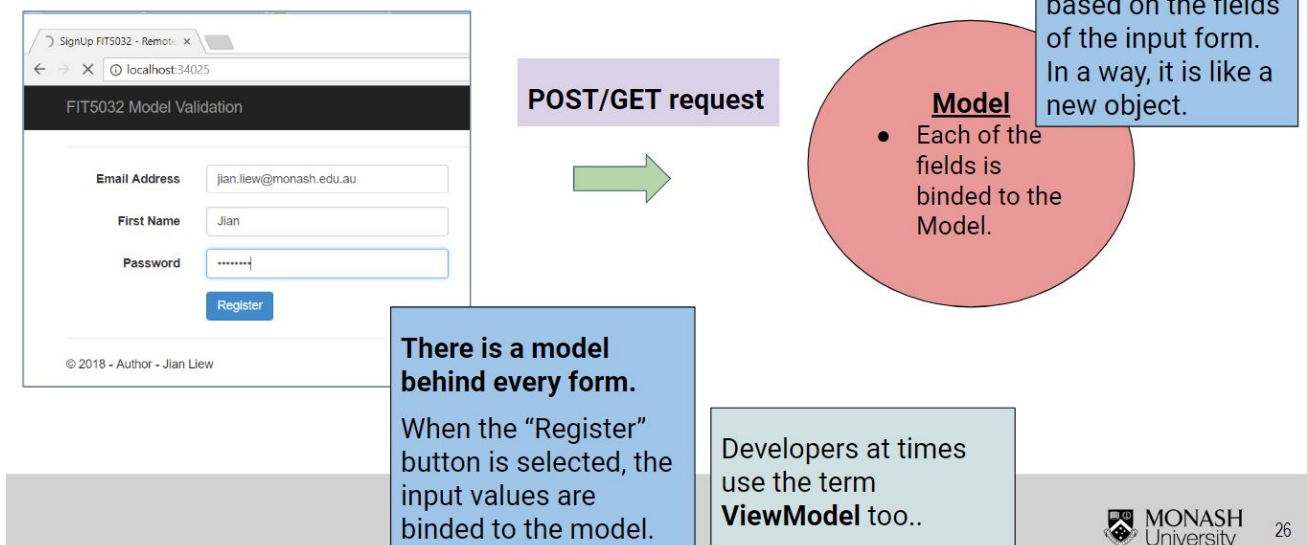
Having a ViewModel simplifies the development significantly as with it, it would be possible to automatically generate codes and use the scaffolding features that are present in Visual Studio.

By default, if you have created the project with Individual accounts enabled, you would notice that it makes good use of this.

For example, you can take a look at

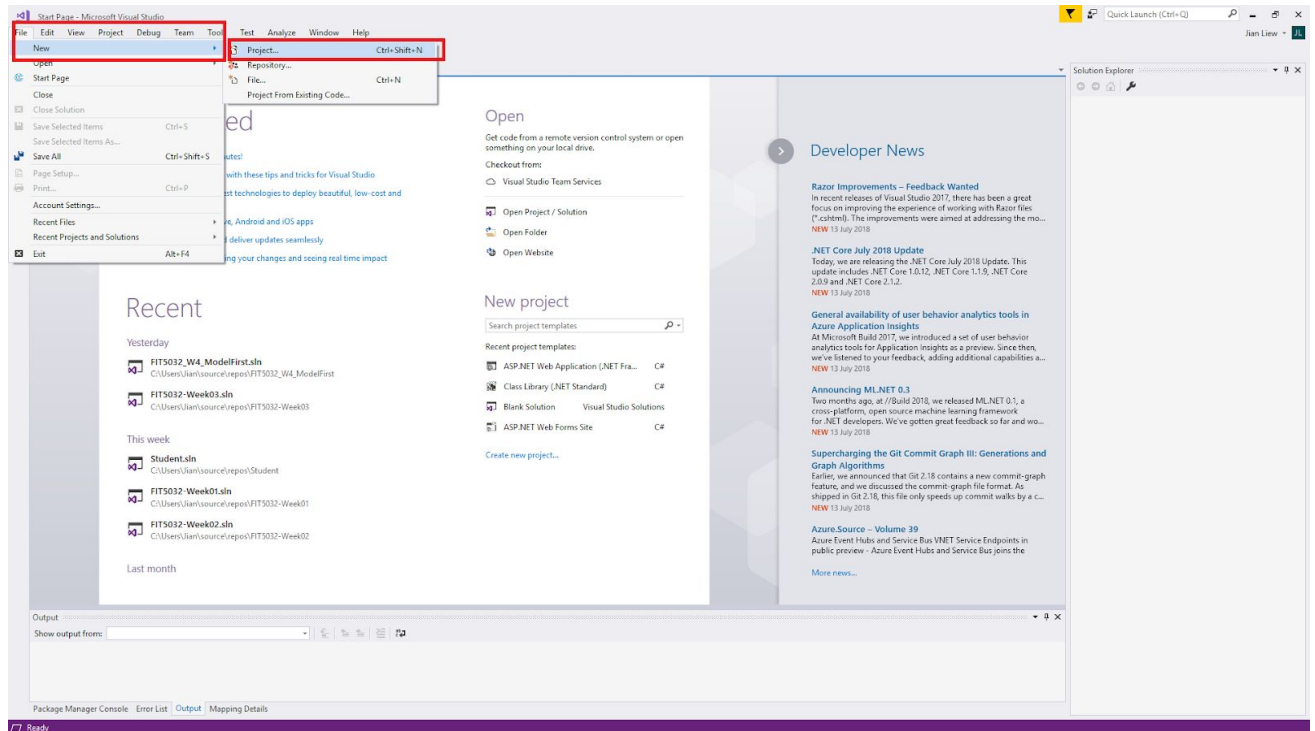
AccountViewModels.cs and ManageViewModels.cs

## How Model Binding Works

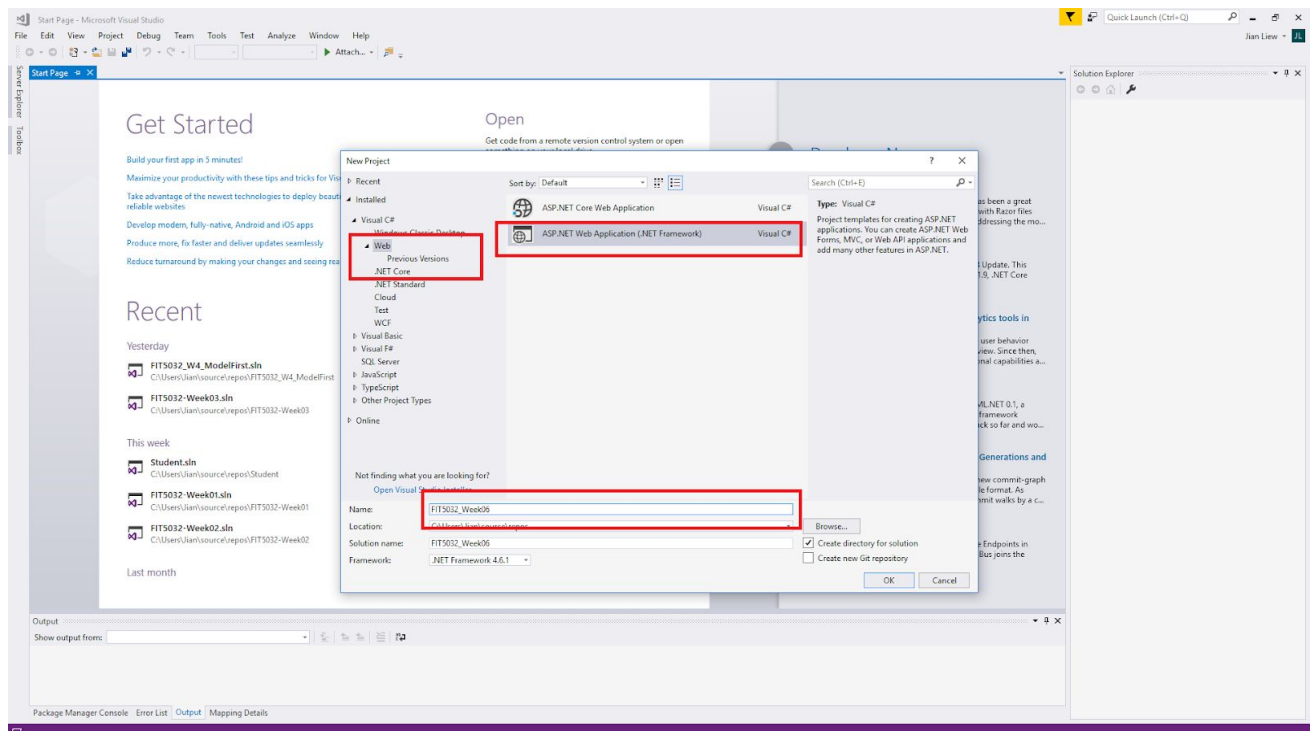


In a way, in combination of using the ViewModel, ModelBinding will be used and it would simplify development significantly. This is one of the benefits of using ASP.NET MVC as it significantly simplifies the creation of forms if used correctly.

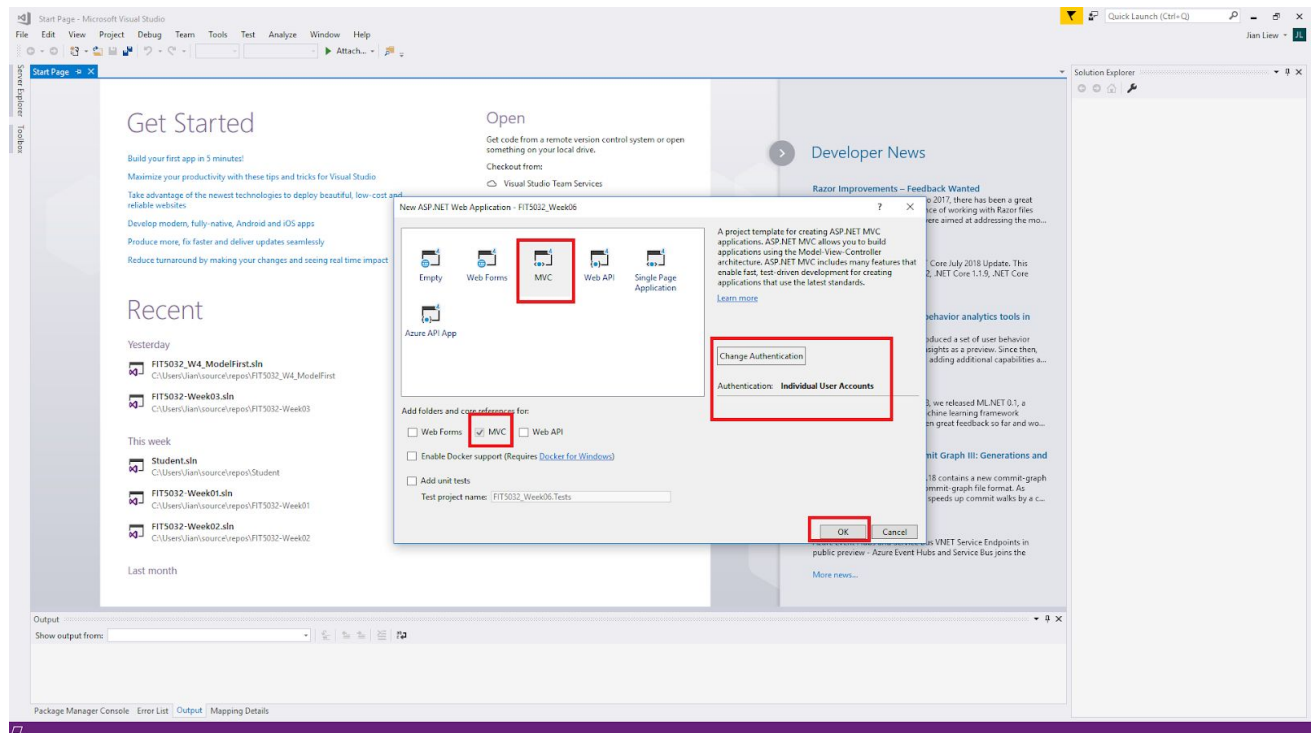
## Step 1



## Step 2

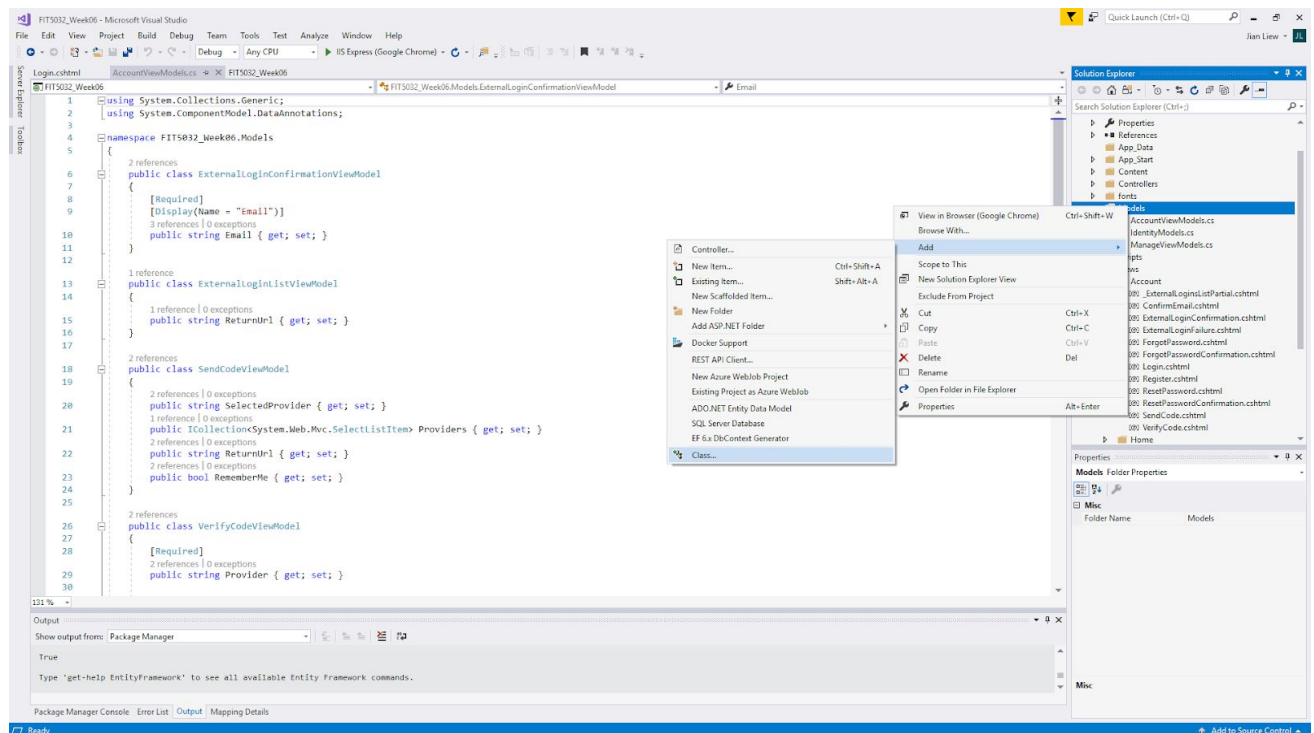


## Step 3

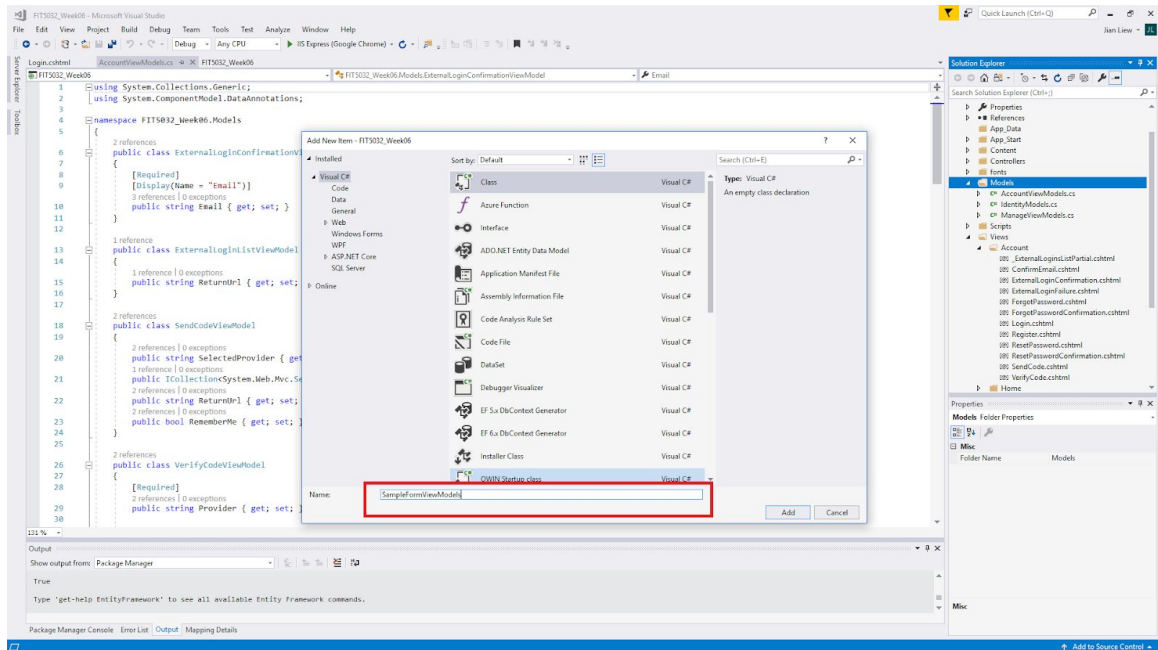


## Step 4

Now I will create my own ViewModel. (Remember that it is just a C# class)



## Step 5



## Step 6

I will now introduce my own class called FormOne.

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

```
namespace FIT5032_Week06.Models
{
    public class SampleFormViewModels
    {
    }

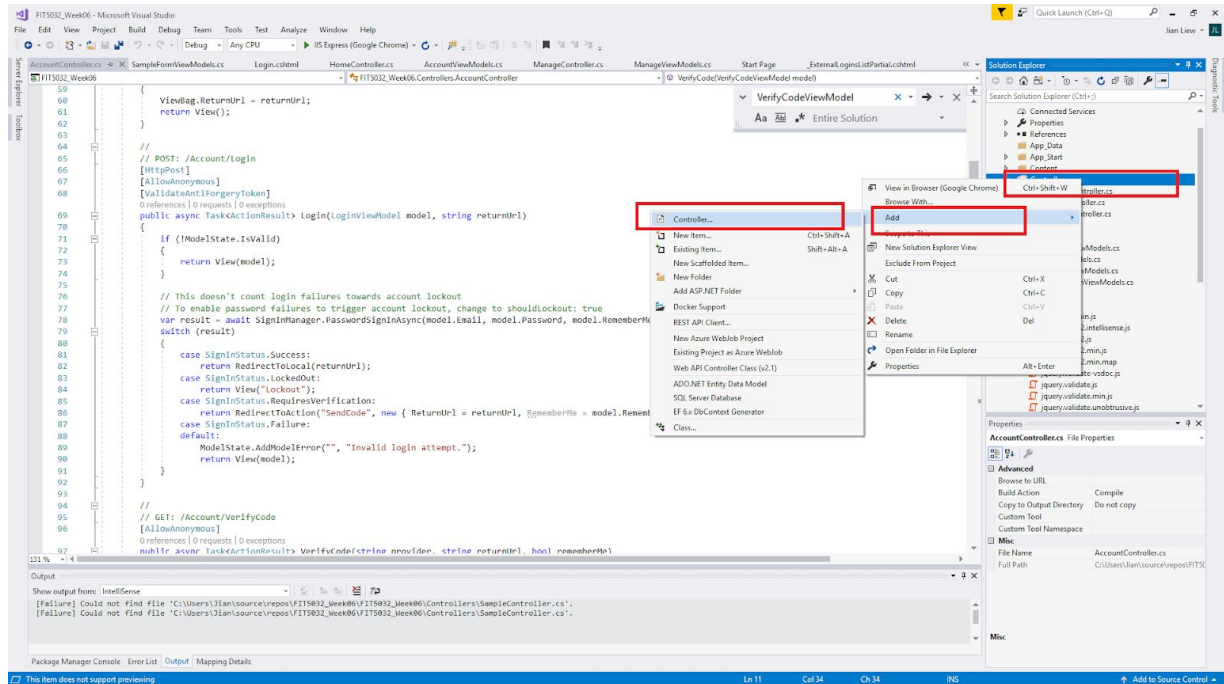
    public class FormOneViewModel
    {
        [Required]
        [Display(Name = "First Name")]
        public string FirstName { get; set; }

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



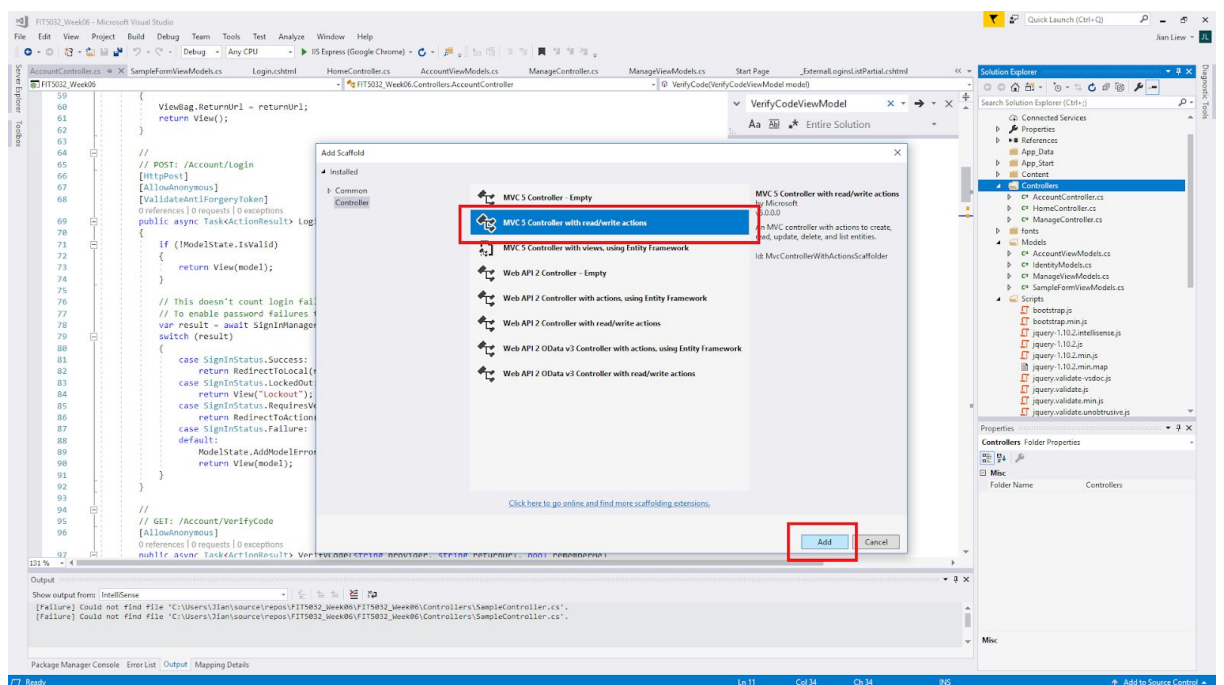
## Step 7

Now, I will need to introduce my own controller. Remember that this is the MVC architecture, so the controller is needed. Right Click the "Controllers" folder and go to "Add" and then Controller



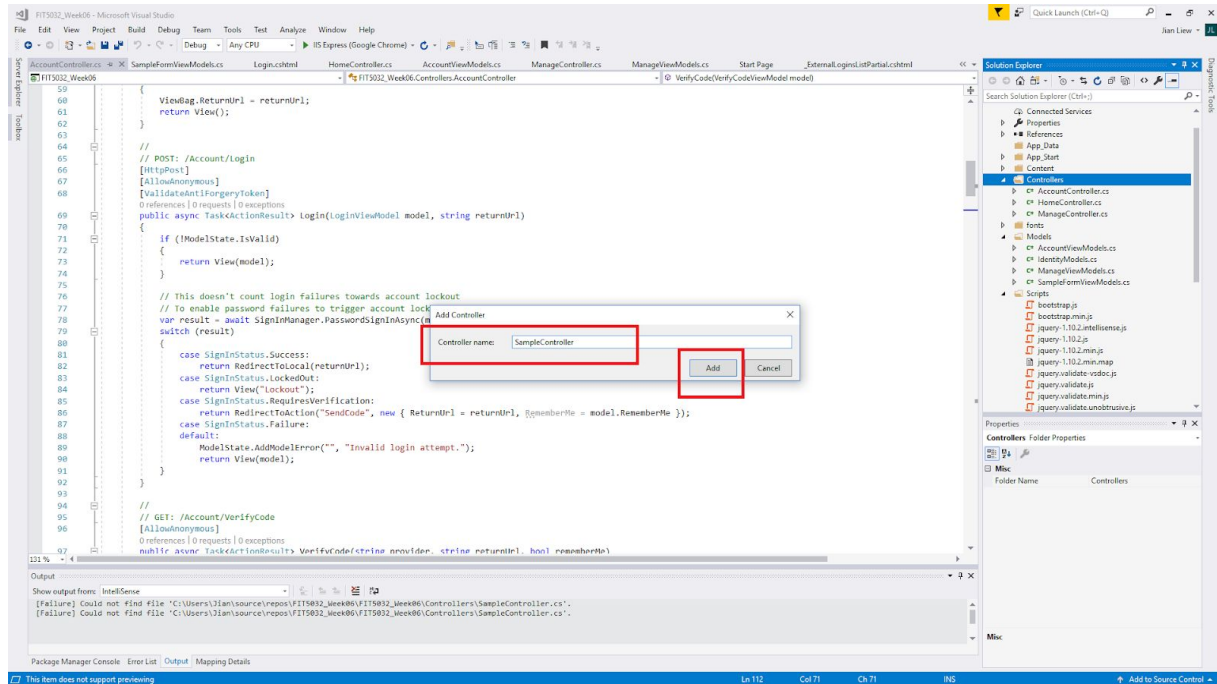
## Step 8

I will create a controller with **Read / Write Actions**.



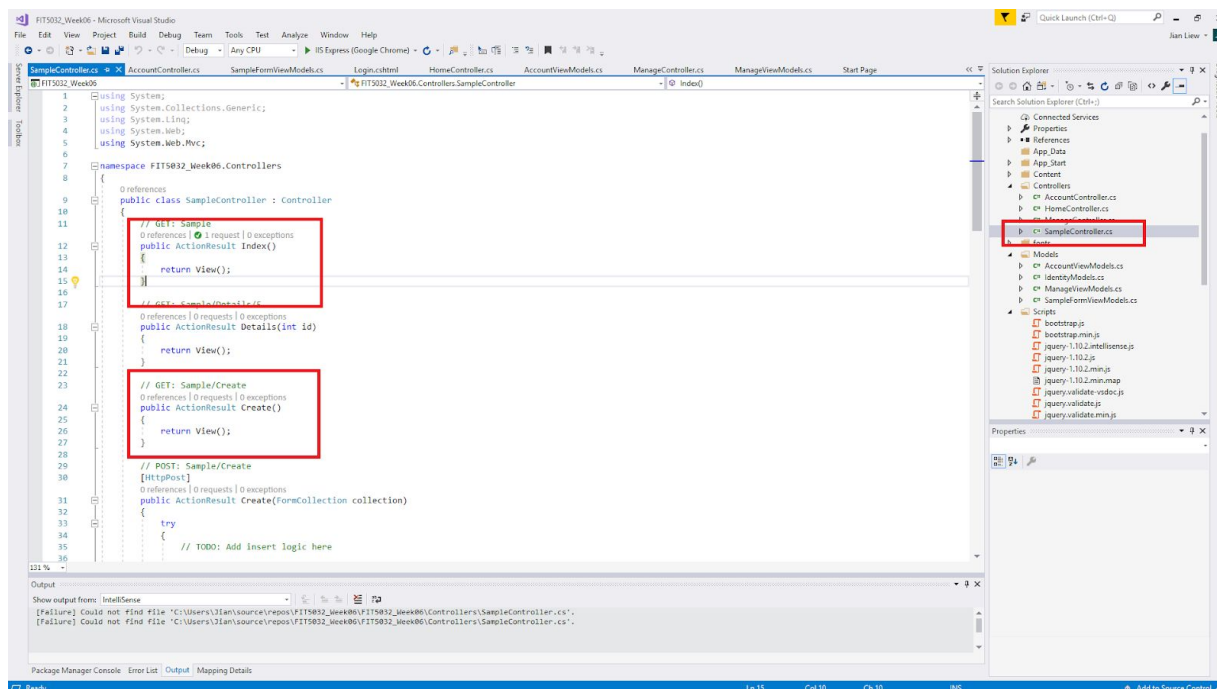
## Step 9

The name would be "SampleController"



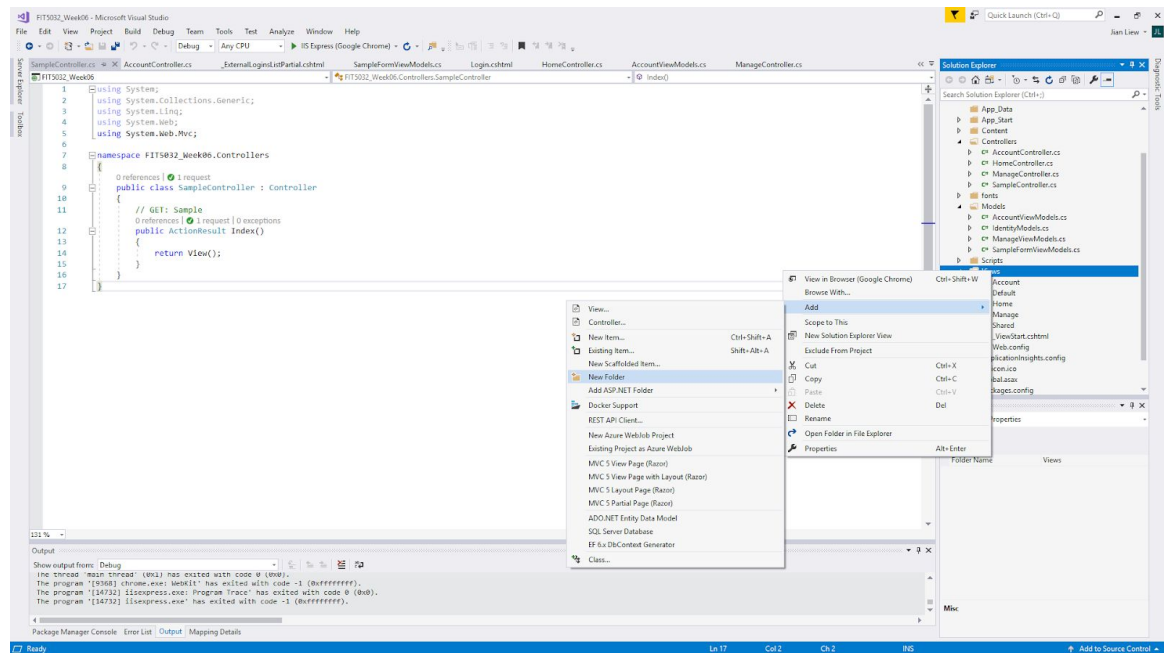
## Step 10

The SampleController.cs will be populated with default codes.

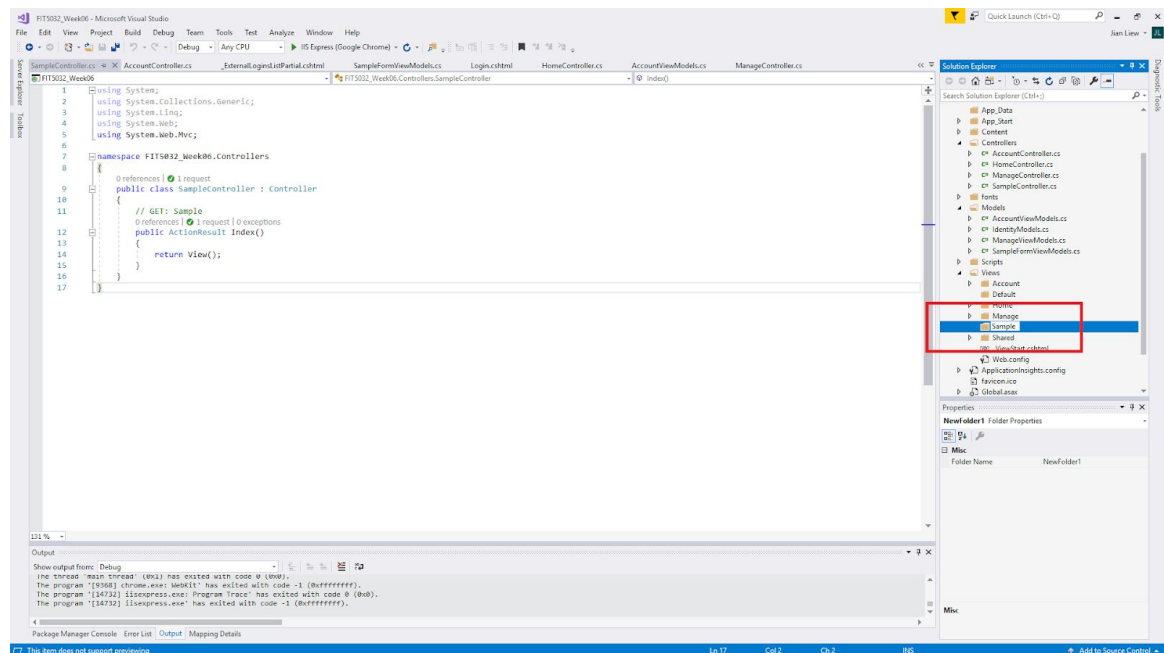


## Step 11

We will now create our View page.



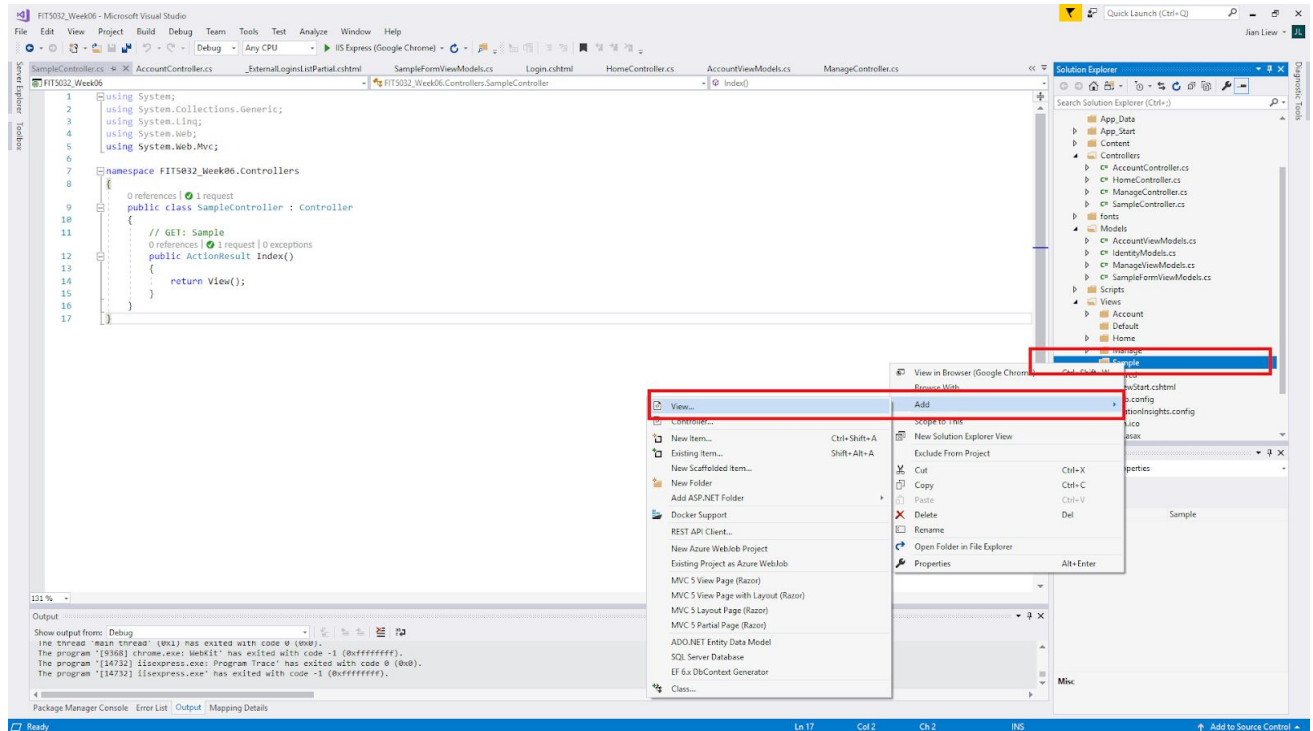
I will create a sub folder under the Views folder and call it "Sample".



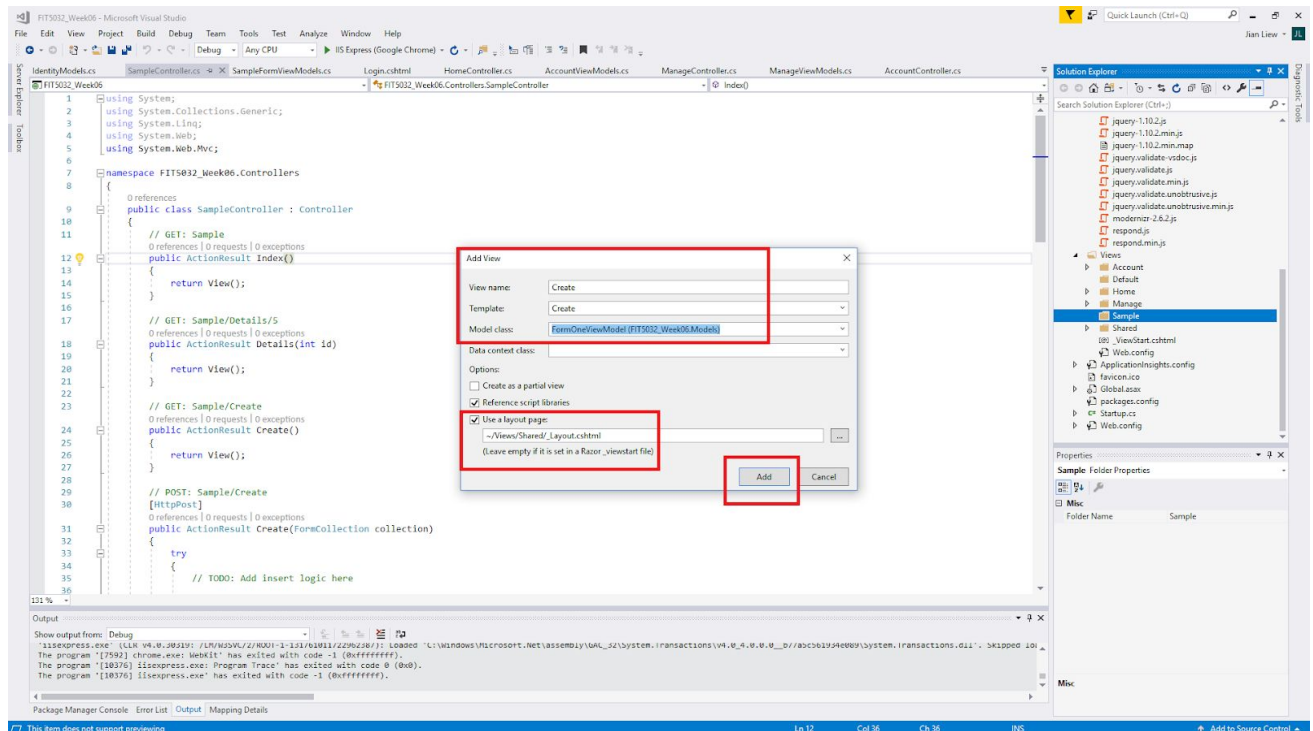


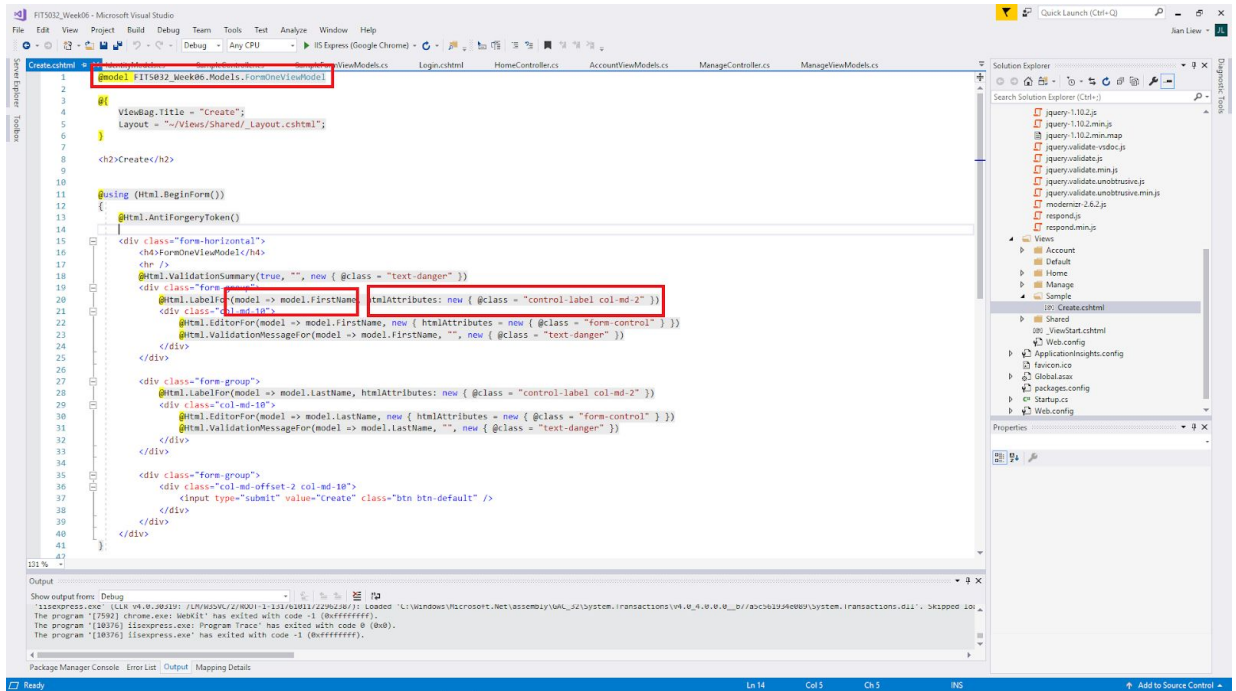
## Step 12

After that create Views for your ViewModel. (This View should be inside the Sample folder)



## Step 12





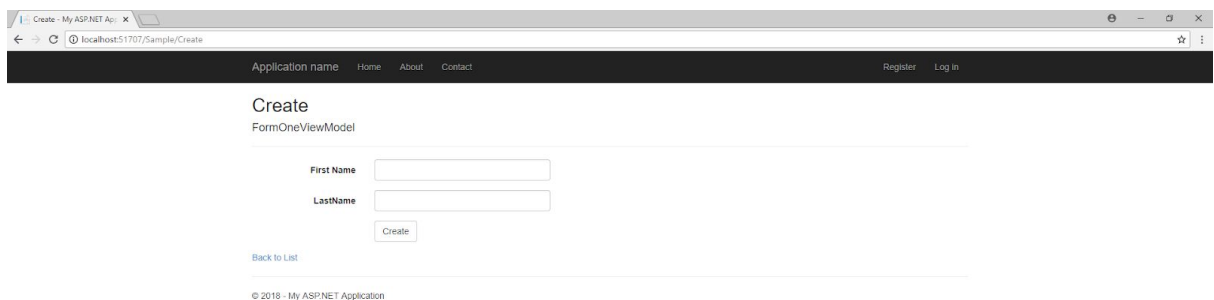
```

1  @model FIT5032_Week06.Models.FormOneViewModel
2
3  ViewBag.Title = "Create";
4  Layout = "~/Views/Shared/_Layout.cshtml";
5
6
7
8  <h2>Create</h2>
9
10
11
12  @using (Html.BeginForm())
13  {
14      @Html.AntiForgeryToken()
15
16      <div class="form-horizontal">
17          <h3>FormOneViewModel</h3>
18          <div class="form">
19              @Html.ValidationSummary(true, "", new { @class = "text-danger" })
20              @Html.LabelFor(model => model.FirstName, new { @class = "control-label col-md-2" })
21              @Html.EditorFor(model => model.FirstName, new { htmlAttributes = new { @class = "form-control" } })
22              @Html.ValidationMessageFor(model => model.FirstName, "", new { @class = "text-danger" })
23          </div>
24
25          <div class="form-group">
26              @Html.LabelFor(model => model.LastName, new { @class = "control-label col-md-2" })
27              @Html.EditorFor(model => model.LastName, new { htmlAttributes = new { @class = "form-control" } })
28              @Html.ValidationMessageFor(model => model.LastName, "", new { @class = "text-danger" })
29          </div>
30
31          <div class="form-group">
32              <div class="col-md-offset-2 col-md-10">
33                  <input type="submit" value="Create" class="btn btn-default" />
34              </div>
35          </div>
36      </div>
37  }
38
39
40
41
  
```

Here, the codes will be automatically generated once again.

The reason I used the View Name "Create" is because the automatically generated controller automatically maps the path. You will also notice that there is a backing view model for this page. The **first request to the page is normally the GET request** and after the submit it does a POST.

After you have completed this step, you can hit the run button.



Notice that Visual Studio simplifies the creation of these forms, if you know how to do it correctly.

Please take note that the names are very important because this is how the default mapping works.

Besides that, you will also notice that if you play around, that the client side validation are also automatically generated. However if you notice, if the form is submitted successfully there will an error. We will now go ahead and fix it up.

I will now make the following changes in SampleController.cs. I will change the entire HTTP post method to the following. So, **instead of using a Form Collection, I will use the backing model.**

This is what we call the binding model. So when the values are submitted, the values are bound to this model. I will also redirect him back to the same View instead of going to the Index page which does not exist.

```
[HttpPost]
public ActionResult Create(FormOneViewModel model)
{
    try
    {
        String FirstName = model.FirstName;

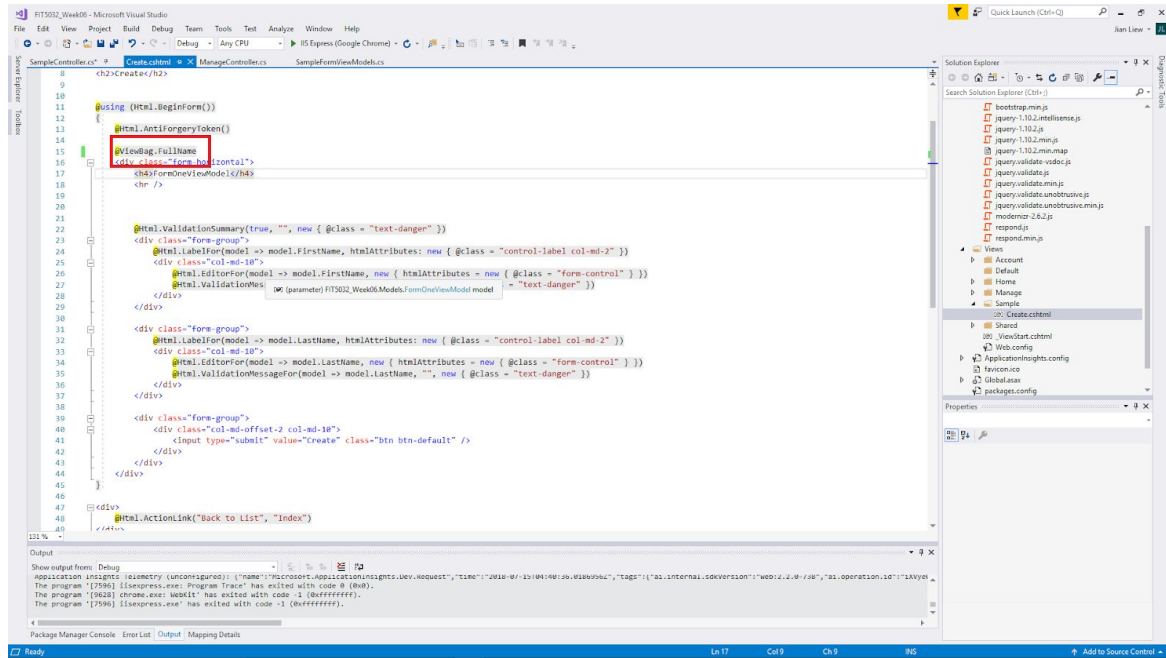
        String LastName = model.LastName;

        ViewBag.FullName = FirstName + " " + LastName;

        return View();
    }
    catch
    {
        return View();
    }
}
```

After that,

I will introduce the ViewBag value in the View page that was created earlier.



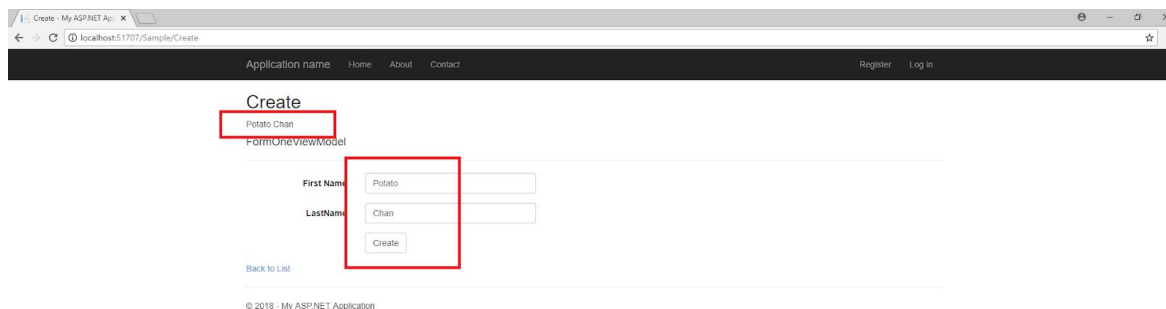
```

using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    @ViewBag.FullName
    <div class="form-group">
        <div class="form-control">
            @Html.EditorFor(model => model.FirstName, new { htmlAttributes = new { @class = "form-control" } })
        </div>
        @Html.ValidationMessageFor(model => model.FirstName, "", new { @class = "text-danger" })
    </div>
    <div class="form-group">
        @Html.EditorFor(model => model.LastName, new { htmlAttributes = new { @class = "form-control" } })
        @Html.ValidationMessageFor(model => model.LastName, "", new { @class = "text-danger" })
    </div>
    <div class="form-group">
        <div class="col-md-offset-2 col-md-10">
            <input type="submit" value="Create" class="btn btn-default" />
        </div>
    </div>
    @Html.ActionLink("Back to List", "Index")
}

```

If everything was done correctly.

You should see the value after the form is submitted.



You should now have a general idea of how the ViewModel and Model Binding is related to the controllers and how they interact with each other.

## Conclusion

Upon the completion of this tutorial, you would gain a basic understanding on how to use attribute validation. You can also use these attributes at the model instead of the view model. However, there are times when you would prefer something more custom instead of the automatic generated ones, one of the more common libraries to do this using [FluentValidation](#).

### DoubtFire Submission

- T6.1 A document explaining what jQuery unobstructive and how it works in combination with ViewModels.
- T6.2 [Updated 09/08/2017] A document with screenshots showing your week 6 application with validation running. (in a PDF document)