**Scaffolding**

Introduction: so far manually created the controller and views. This requires a lot of time and manual coding. To overcome this problem using Scaffolding.

\*) scaffolding allows you to create auto generated controllers and corresponding views.

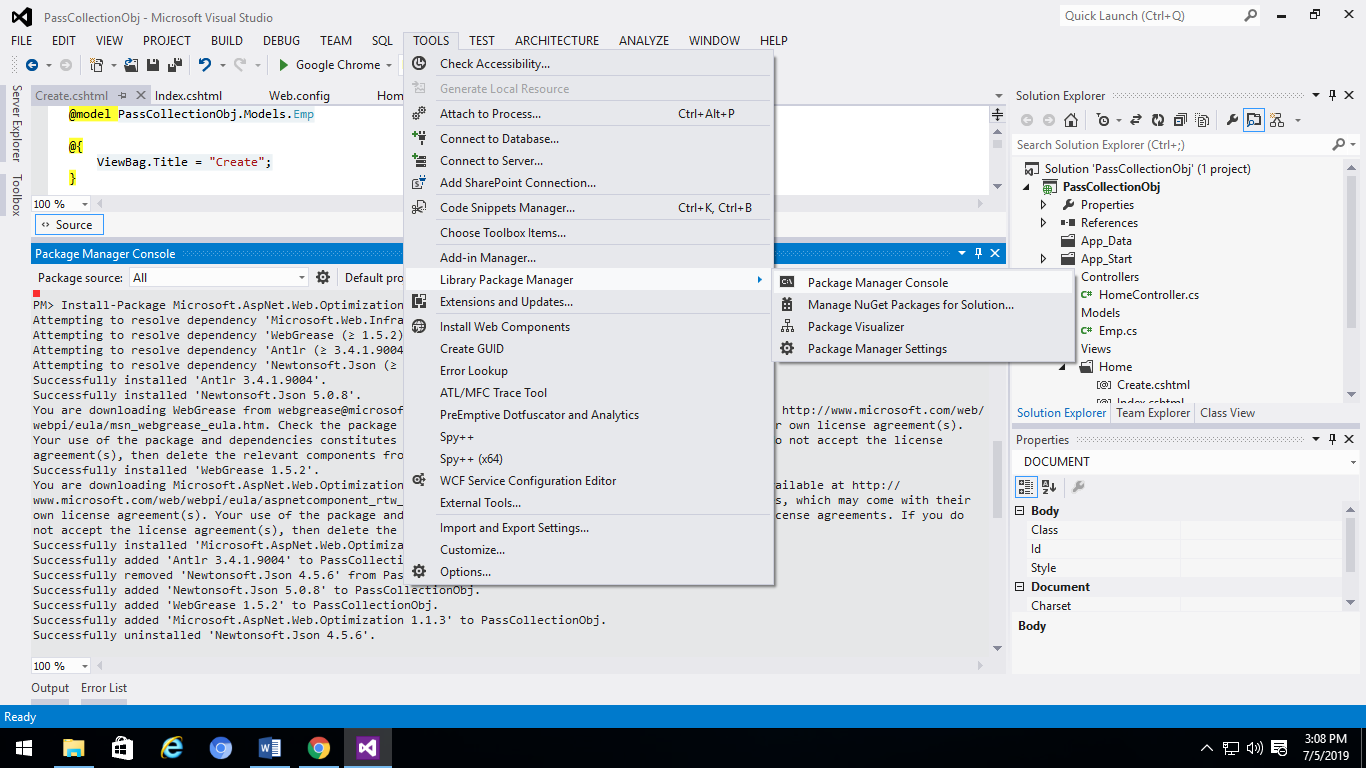
\*) scaffolding uses predefined convention for naming controller and views.

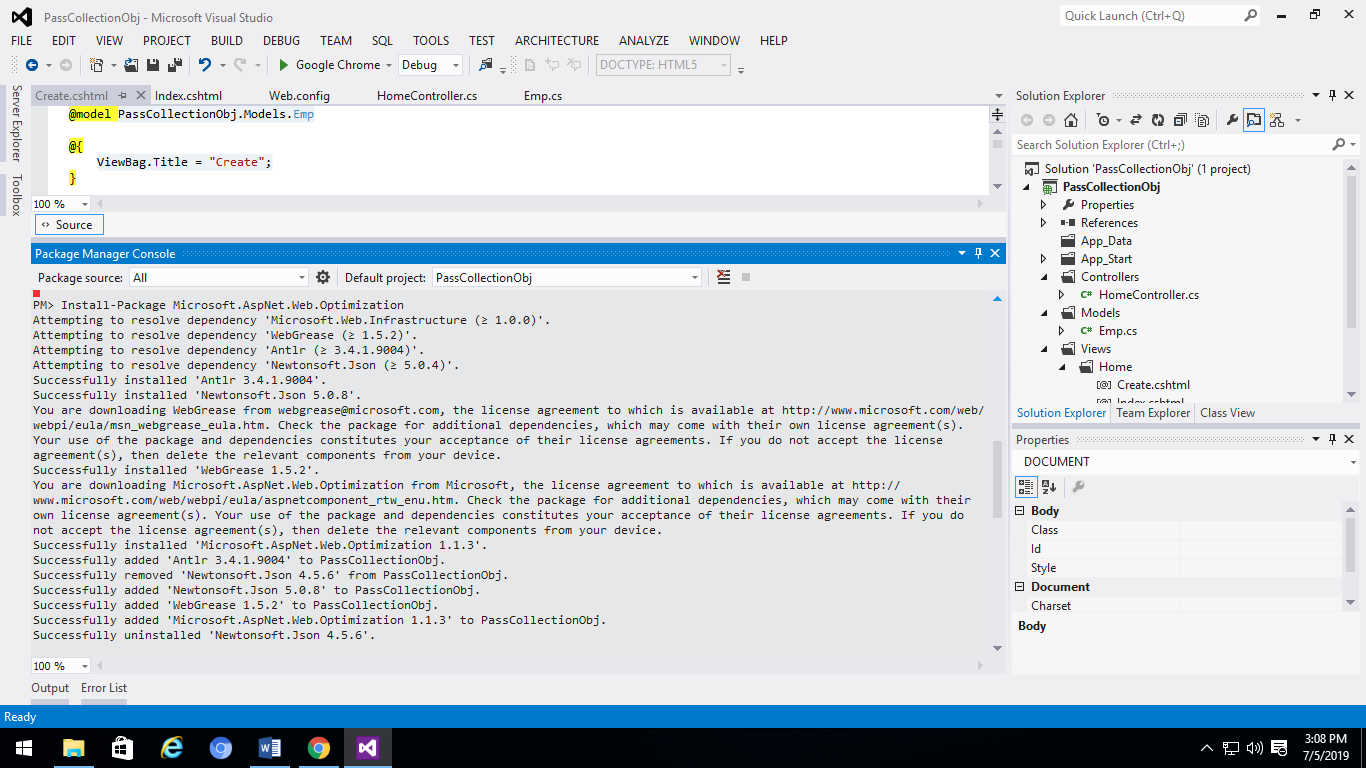
Def:

Scaffolding is a technique used by many MVC frameworks like ASP.NET MVC, Ruby on Rails, Cake PHP and Node.JS etc., to generate code for basic CRUD (create, read, update, and delete) operations against your database effectively. Further you can edit or customize this auto generated code according to your need.

**Scaffolding provides various templates for creating Controllers and associative views.**

1. **Empty MVC controller – derived from Controller class – only one action method index-no code.**
2. **MVC controller with empty read/write actions: action method (Index,Details,create,Edit,Delete) – return some code inside them**
3. **API Controller with empty read/write action: derived from APIController**





**Create Scaffold Template.**

**Models File(Emp.cs)**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Scaff1.Models

{

public class Emp

{

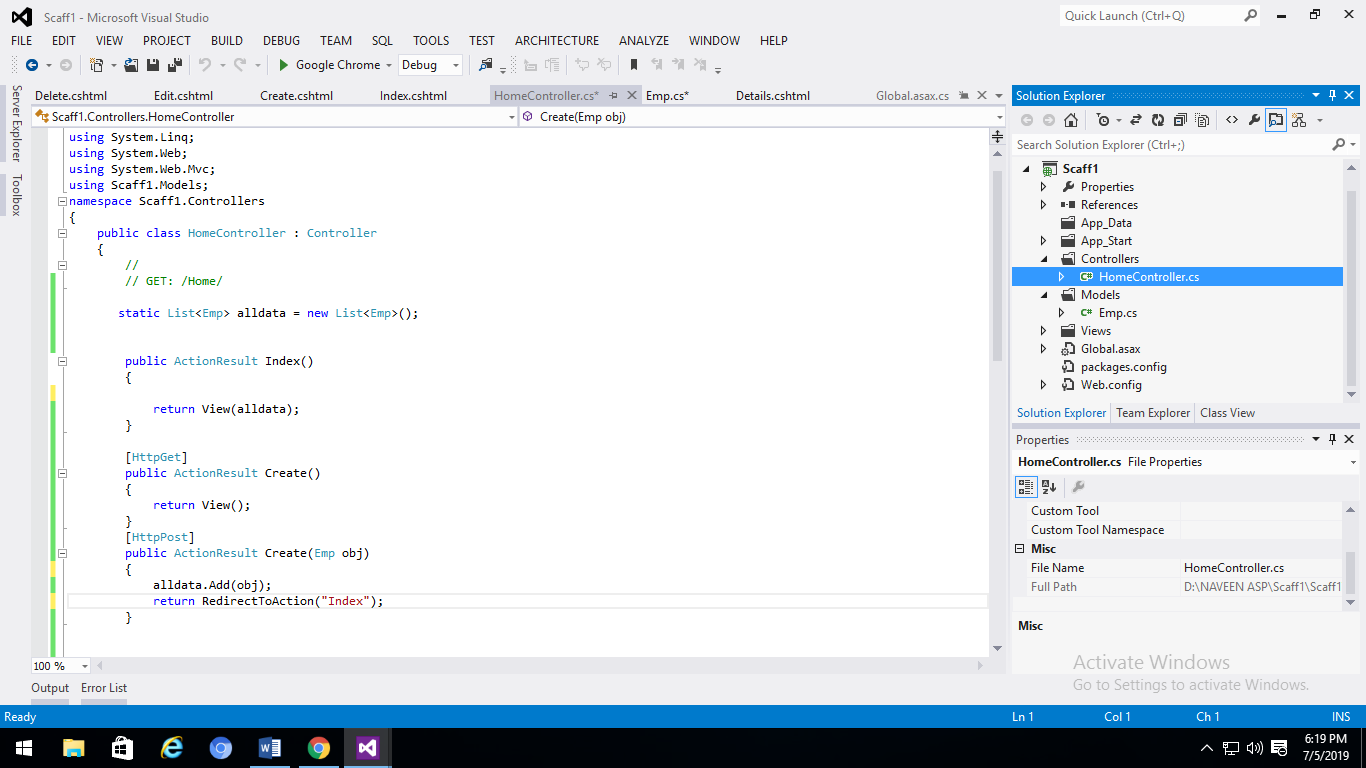
public int eno { get; set; }

public String ename { get; set; }

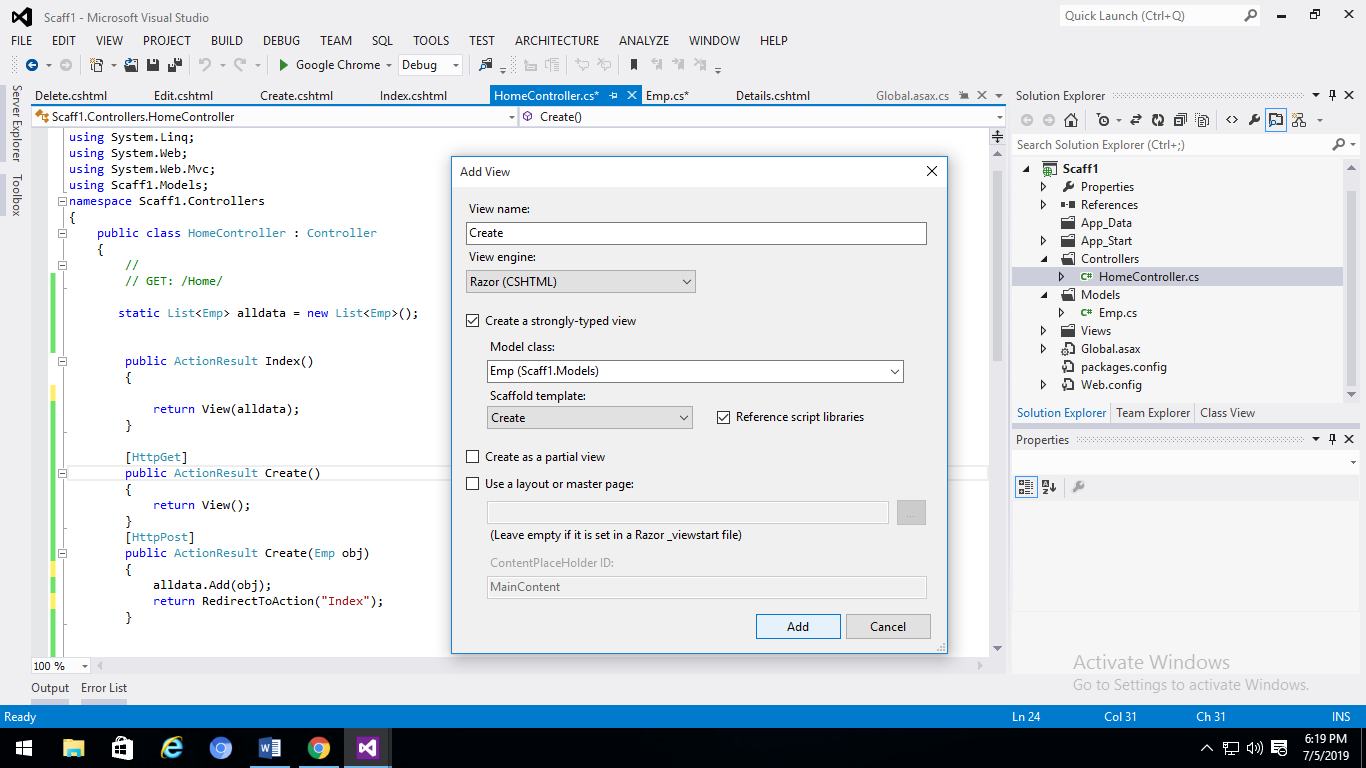
public int esal { get; set; }

}

}



**Right click on Create (httpGet)🡪addview 🡪……add button press.**



**View file is automatically Generated(: views ->> home 🡪 Create.cshtml)**

@model Scaff1.Models.Emp

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width" />

<title>Create</title>

</head>

<body>

<script src="~/Scripts/jquery-1.7.1.min.js"></script>

<script src="~/Scripts/jquery.validate.min.js"></script>

<script src="~/Scripts/jquery.validate.unobtrusive.min.js"></script>

@using (Html.BeginForm()) {

@Html.ValidationSummary(true)

<fieldset>

<legend>Emp</legend>

<div class="editor-label">

@Html.LabelFor(model => model.eno)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.eno)

@Html.ValidationMessageFor(model => model.eno)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.ename)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.ename)

@Html.ValidationMessageFor(model => model.ename)

</div>

<div class="editor-label">

@Html.LabelFor(model => model.esal)

</div>

<div class="editor-field">

@Html.EditorFor(model => model.esal)

@Html.ValidationMessageFor(model => model.esal)

</div>

<p>

<input type="submit" value="Create" />

</p>

</fieldset>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

</body>

</html>

**Note: when user click submit button(create) 🡪 HTTPPost method called at controller file**

**Ex:**

[HttpGet] // default no need to type [HttpGet] Annodataion

// controller to view

public ActionResult Create()

{

return View();

}

[HttpPost] // view to contrlller (very important)

public ActionResult Create(Emp obj)

{

alldata.Add(obj);

return RedirectToAction("Index");

}

**Result:**

