**1) What is MVC (Model view controller)?**

Model–view–controller **(MVC) is a Software Architectural Pattern** for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representation of information from the way that information is presented to or accepted from the user.

MVC is a framework for building web applications using a MVC (Model View Controller) design:

* The Model(Business Layer) represents the application core (for instance a list of database records).
* The View(Display Layer) displays the data (the database records).
* The Controller(Input Control) handles the input (to the database records).

**2) Explain MVC application life cycle?**

**Step 1 Fill route:** - MVC requests are mapped to route tables which in turn specify which controller and action to be invoked. So if the request is the first request the first thing is to fill the route table with routes collection. This filling of route table happens in the global.asax file.

**Step 2 Fetch route:-** Depending on the URL sent "UrlRoutingModule" searches the route table to create "RouteData" object which has the details of which controller and action to invoke.

**Step 3 Request context created: -** The "RouteData" object is used to create the "RequestContext" object.

**Step 4 Controller instance created: -** This request object is sent to "MvcHandler" instance to create the controller class instance. Once the controller class object is created it calls the "Execute" method of the controller class.

Creating Response object: - This phase has two steps executing the action and finally sending the response as a result to the view.

**Step 5 Execute Action: -** The "ControllerActionInvoker" determines which action to executed and executes the action.

**Step 6 Result sent: -** The action method executes and creates the type of result which can be a view result , file result , JSON result etc.

**3) What are the advantages of MVC?**

**Multiple view support**

Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time.

**Change Accommodation**

User interfaces tend to change more frequently than business rules (different colors, fonts, screen layouts, and levels of support for new devices such as cell phones or PDAs) because the model does not depend on the views, adding new types of views to the system generally does not affect the model. As a result, the scope of change is confined to the view.

**SoC – Separation of Concerns**

Separation of Concerns is one of the core advantages of ASP.NET MVC . The MVC framework provides a clean separation of the UI, Business Logic, Model or Data.

**More Control**

The ASP.NET MVC framework provides more control over HTML, JavaScript and CSS than the traditional Web Forms.

**Testability**

ASP.NET MVC framework provides better testability of the Web Application and good support for the test driven development too.

**Lightweight**

ASP.NET MVC framework doesn’t use View State and thus reduces the bandwidth of the requests to an extent.

**Full features of ASP.NET**

One of the key advantages of using ASP.NET MVC is that it is built on top of ASP.NET framework and hence most of the features of the ASP.NET like membership providers, roles, etc can still be used.

**4) Filters in MVC**

There are five types of Filters in ASP.NET MVC 5:

**Authentication Filters:**

Authentication filter runs before any other filter or action method. **Authentication confirms that you are a valid or invalid user**. Action filters implements the **IAuthenticationFilter interface**.

**Authorization Filters:**

The AuthorizeAttribute and RequireHttpsAttribute are the examples of Authorization Filters. **Authorization Filters are responsible for checking User Access**; these implement the IAuthorizationFilterinterface in the framework. These filters used to implement authentication and authorization for controller actions. For example, the Authorize filter is an example of an Authorization filter.

**Action Filters:**

Action Filter is an attribute that you can apply to a controller action or an entire controller. This filter will be called before and after the action starts executing and after the action has executed.

Action filters implement the **IActionFilter interface** that have **two methods OnActionExecuting and OnActionExecuted**. OnActionExecuting runs before the Action and gives an opportunity to cancel the Action call. These filters contain logic that is executed before and after a controller action executes, you can use an action filter, for instance, to modify the view data that a controller action returns.

**Result Filters:**

The OutputCacheAttribute class is an example of Result Filters. These implement the **IResultFilter interface** which like the IActionFilter has **OnResultExecuting and OnResultExecuted**. These filters contains logic that is executed before and after a view result is executed. Like if you want to modify a view result right before the view is rendered to the browser.

**ExceptionFilters:**

The HandleErrorAttribute class is an example of ExceptionFilters. These implement the **IExceptionFilter interface** and **they execute if there are any unhandled exceptions thrown during the execution pipeline**. These filters can be used as an exception filter to handle errors raised by either your controller actions or controller action results.

**5) Attributes in MVC**

[Key] - to mark the property as primary key

[Display(Name = "Is Active?")]

public bool Active { get; set; }

[Required(ErrorMessage = "Please write your LastName")]

public string LastName{ get; set; }

[StringLength(20, MinimumLength = 4, ErrorMessage = "Must be at least 4 characters long.")]

public string FirstName { get; set; }

[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]

// [DisplayFormat(NullDisplayText = "Not specified")]

public DateTime? BirthDate{ get; set; }

[Range(0, 18, ErrorMessage = "The value must be between 0 and 18")]

public int Age { get; set; }

[Compare("ConfirmPassword", ErrorMessage = "Sorry, both should match")]

public string Password { get; set; }

[RegularExpression(@"\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*")]

public string Email { get; set; }

**6) List out different return types of a controller action method?**

There are total nine return types we can use to return results from controller to view.

**ViewResult (View):** This return type is used to return a webpage from an action method.

**PartialviewResult (Partialview):** This return type is used to send a part of a view which will be rendered in another view.

**RedirectResult (Redirect):** This return type is used to redirect to any other controller and action method depending on the URL.

**RedirectToRouteResult (RedirectToAction, RedirectToRoute):** This return type is used when we want to redirect to any other action method.

**ContentResult (Content):** This return type is used to return HTTP content type like text/plain as the result of the action.

**jsonResult (json):** This return type is used when we want to return a JSON message.

**javascriptResult (javascript):** This return type is used to return JavaScript code that will run in browser.

FileResult (File): This return type is used to send binary output in response.

**EmptyResult:** This return type is used to return nothing (void) in the result.

**7) Explain attribute based routing in MVC?**

In ASP.NET MVC 5.0 we have a new attribute route, By using the "Route" attribute we can define the URL structure. For example in the below code we have decorated the "GotoAbout" action with the route attribute. The route attribute says that the "GotoAbout" can be invoked using the URL structure "Users/about".

public class HomeController: Controller

{

[Route("Users/about")]

Public ActionResult GotoAbout()

{

return View();

}

}

**8) If we have multiple filters, what’s the sequence for execution?**

Authentication Filters

Authorization filters

Action filters

Response filters

Exception filters

**9) Explain RenderSection in MVC?**

RenderSection() is a method of the WebPageBase class. Scott wrote at one point, The first parameter to the "RenderSection()" helper method specifies the name of the section we want to render at that location in the layout template. The second parameter is optional, and allows us to define whether the section we are rendering is required or not. If a section is "required", then Razor will throw an error at runtime if that section is not implemented within a view template that is based on the layout file (that can make it easier to track down content errors). It returns the HTML content to render.

<div id="body">

@RenderSection("featured", required: false)

<section class="content-wrapper main-content clear-fix">

@RenderBody()

</section>

</div>

**10) Explain what is connected and dis-connected data access in ADO.NET?**

**Connected data access**: Through the DataReader objects of data-provider you can have connected data access. It provides fast access to data, and it does not allow editing.

**Dis-connected data access:** Through the DataAdapter object, disconnected data access is achieved. The dataset works independently of the database, and the data is editable.

**11) What is the difference between each version of MVC 2, 3 , 4, 5 and 6?**

**MVC 6**

1. ASP.NET MVC and Web API has been merged in to one.
2. Side by side - deploy the runtime and framework with your application
3. No need to recompile for every change. Just hit save and refresh the browser.
4. Dependency injection is inbuilt and part of MVC.
5. Everything packaged with NuGet, Including the .NET runtime itself.
6. New JSON based project structure.
7. Compilation done with the new Roslyn real-time compiler.

**MVC 5**

1. Asp.Net Identity
2. Attribute based routing
3. Bootstrap in the MVC template
4. Filter overrides
5. Authentication Filters

**MVC 4**

1. ASP.NET Web API
2. New mobile project template
3. Refreshed and modernized default project templates
4. Many new features to support mobile apps

**MVC 3**

1. Razor
2. HTML 5 enabled templates
3. JavaScript and Ajax
4. Support for Multiple View Engines
5. Model Validation Improvements

**MVC 2**

1. Templated Helpers
2. Client-Side Validation
3. Areas
4. Asynchronous Controllers
5. Html.ValidationSummary Helper Method
6. DefaultValueAttribute in Action-Method Parameters
7. Binding Binary Data with Model Binders
8. DataAnnotations Attributes
9. Model-Validator Providers
10. New RequireHttpsAttribute Action Filter

**12) What is the difference between “HTML.TextBox” and “HTML.TextBoxFor”?**

Both provide the same HTML output, **“HTML.TextBoxFor” is strongly typed while “HTML.TextBox” isn’t**. Below is a simple HTML code which just creates a simple textbox with “FirstName” as name.

**Html.TextBox("FirstName ")**

Below is “Html.TextBoxFor” code which creates HTML textbox using the property name ‘FirstName” from object “m”.

**Html.TextBoxFor(m =>m.FirstName)**

**13) Where is the route mapping code written?**

The route mapping code is written in "RouteConfig.cs" file and registered using "global.asax" application start event.

**14) What is the difference between Temp data, View, and View Bag?**

In ASP.NET MVC there are three ways to pass/store data between the controllers and views.

**ViewData**

* ViewData is used to pass data from controller to view.
* It is derived from ViewDataDictionary class.
* It is available for the current request only.
* **Requires typecasting for complex data type and checks for null values to avoid error**.
* If redirection occurs, then its value becomes null.

**ViewBag**

* ViewBag is also used to pass data from the controller to the respective view.
* ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
* It is also available for the current request only.
* If redirection occurs, then its value becomes null.
* **Doesn’t require typecasting for complex data type**.

**TempData**

* TempData is derived from TempDataDictionary class
* TempData is used to pass data from the current request to the next request
* It keeps the information for the time of an HTTP Request. This means only from one page to another. It helps to maintain the data when we move from one controller to another controller or from one action to another action
* It requires typecasting for complex data type and checks for null values to avoid error. Generally, it is used to store only one time messages like the error messages and validation messages

**15) Explain attribute based routing in MVC?**

In ASP.NET MVC 5.0 we have a new attribute route, By using the "Route" attribute we can define the URL structure. For example in the below code we have decorated the "GotoAbout" action with the route attribute. The route attribute says that the "GotoAbout" can be invoked using the URL structure "Users/about".

public class HomeController: Controller

{

[Route("Users/about")]

Public ActionResult GotoAbout()

{

return View();

}

}

**16) How to implement Forms authentication in MVC?**

ASP.NET forms authentication occurs after IIS authentication is completed. You can configure forms authentication by using forms element with in web.config file of your application. The default attribute values for forms authentication are shown below:

<system.web>

<authenticationmode="Forms">

<formsloginUrl="Login.aspx" protection="All" timeout="30" name=".ASPXAUTH" path="/" requireSSL="false" slidingExpiration="true" defaultUrl="default.aspx" cookieless="UseDeviceProfile" enableCrossAppRedirects="false" />

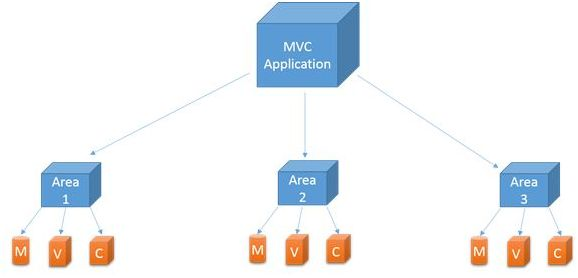
</authentication>

</system.web>

The FormsAuthentication class creates the authentication cookie automatically when SetAuthCookie() or RedirectFromLoginPage() methods are called. The value of authentication cookie contains a string representation of the encrypted and signed FormsAuthenticationTicket object.

**17) What is Areas in MVC?**

From ASP.Net MVC 2.0 Microsoft provided a new feature in MVC applications, Areas. Areas are just a way to divide or “isolate” the modules of large applications in multiple or separated MVC. like:



When you add an area to a project, a route for the area is defined in an AreaRegistration file. The route sends requests to the area based on the request URL. To register routes for areas, you add code to the Global.asax file that can automatically find the area routes in the AreaRegistration file.

**18) What is the difference between ActionResult and ViewResult?**

* ActionResult is an abstract class while ViewResult derives from the ActionResult class.
* ActionResult has several derived classes like ViewResult, JsonResult, FileStreamResult, and so on.
* ActionResult can be used to exploit polymorphism and dynamism. So if you are returning different types of views dynamically, ActionResult is the best thing. For example in the below code snippet, you can see we have a simple action called DynamicView. Depending on the flag (IsHtmlView) it will either return a ViewResult or JsonResult.

public ActionResult DynamicView()

{

if (IsHtmlView)

return View(); // returns simple ViewResult

else

return Json(); // returns JsonResult view

}

**19) What is the Html.AntiForgeryToken helper function for?**

Basically the antiforgery tokens stop anyone from submitting requests to your site that are generated by a malicious script not generated by the actual user. When you use it, your form will contain a hidden field and a corresponding cookie will also be set in the browser. Then, when the form is submitted, the hidden field is checked against the cookie value (assuming that ValidateAntiForgeryTokenAttribute is used): if the field and the cookie match then the form post is probably genuine; if they don't then it's probably not.

**20) Difference between static class and singleton?**

1- Static is a keyword and Singleton is a design pattern.

2- Singleton gives you an object, while static classes just provides static methods.

3- Singleton is an object creational pattern with one instance of the class.

4- Singleton can implement interfaces, inherit from other classes and it aligns with the OOPS concepts.

5- Singleton object can be passed as a reference whereas static object can't passed as a reference to other methods or objects.

6- Singleton supports object disposal.

7- Singleton object is stored on heap whereas static class is stored on stack

8- Singleton object can be cloned as well.

**21) Difference between @Html.Action and @Html.RenderAction**

**@Html.Action**

This Html.Action renders partial view as an HTML string so we can store it in another string variable. It is string return type method so first it returns result as a string then renders result to response.

**@Html.RenderAction**

This is also same as Html.Action but **main difference is that it renders result directly to response that’s why it is more efficient** if the action returns a large amount of HTML over @Html.Action.

**22) Difference between NonAction and ChildActionOnly Attribute**

1. An action method is a public method in a controller that can be invoked using a URL. So, by default, if you have any public method in a controller then it can be invoked using a URL request. ***To restrict access to public methods in a controller, NonAction attribute can be used****.*
2. Child action methods are different from NonAction methods, in that NonAction methods cannot be invoked using Action() or RenderAction() helpers.
3. ***Child action methods will not respond to URL requests***. If an attempt is made, a runtime error will be thrown stating - **Child action is accessible only by a child request.**
4. ***Child action methods can be invoked by making child request from a view using Action*() and RenderAction() Html helpers**.
5. An action method doesn’t need to have [ChildActionOnly] attribute to be used as a child action, but use this attribute to prevent if you want to prevent the action method from being invoked as a result of a user request.
6. Child actions are typically associated with partial views, although this is not compulsory.
7. Using child action methods, it is possible to cache portions of a view. This is the main advantage of child action methods.

[ChildActionOnly]

public string GetEmployeeCount()

{

return "Employee Count = " + db.employees.Count().ToString() + " @ " + DateTime.Now.ToString();

}

Calling above method using **@Html.Action("GetEmployeeCount")**  will not give error.

[NonAction]

public string Test()

{

return "Employee Count = " + db.employees.Count().ToString() + " @ " + DateTime.Now.ToString();

}

Calling above method using **@Html.Action("GetEmployeeCount")**  will give error.

**23) Benefits of Unit Tests**

1. Find bugs early
2. Significantly reduce production bugs
3. Unit Test make complex code easy to understand
4. Provide Documentation
5. Save Development Time
6. Easier to change and refactor code

**24) Dependency Injection**

With DI it's easy to share data and functionality as the angular Injector Provide Singleton i.e. single instance of the service

**25) What is Routing in MVC?**

Routing is a mechanism in MVC that decides which action method of a controller class to execute. Without routing there's no way an action method can be mapped to a request. Routing is a part of the MVC architecture so ASP.NET MVC supports routing by default.

**26) Difference between TextBoxFor and EditorFor**

**TextBoxFor:** It will render like text input html element corresponding to specified expression. In simple word it will always render like an input textbox irrespective datatype of the property which is getting bind with the control.

**EditorFor:** This control is bit smart. It renders HTML markup based on the datatype of the property. E.g. suppose there is a boolean property in model. To render this property in the view as a checkbox either we can use CheckBoxFor or EditorFor. Both will be generate the same markup.

What is the advantage of using EditorFor?

As we know, depending on the datatype of the property it generates the html markup. So suppose tomorrow if we change the datatype of property in the model, no need to change anything in the view. EditorFor control will change the html markup automatically.

**27) How route table is created in ASP.NET MVC?**

When an MVC application first starts, the Application\_Start() method is called. This

method, in turn, calls the RegisterRoutes() method. The RegisterRoutes() method

creates the route table.

**28) Ajax Helpers**

Install-Package **jquery.unobstrusive.ajax.js**

Microsoft.jQuery.Unobtrusive.Ajax

1. @Ajax.ActionLink("View  All Student Info", "AllStudent", "Home", **new** AjaxOptions
2. {
3. UpdateTargetId = "divAllStudent",
4. OnBegin = "fnOnBegin",
5. InsertionMode = InsertionMode.Replace,
6. HttpMethod = "GET",
7. LoadingElementId = "imgloader",
8. OnSuccess= "fnSuccess",
9. Confirm="Do you want to get all student info ?????"
10. }, **new** { @**class** = "btn btn-default" })
11. Allow cache

**By Default it is false**, it specifies whether you want to cache the page requested by the browser or not by specifying its value as true or false.

1. Confirm

Like jQuery confirm dialog.

1. HttpMethod

Specifies the HTTP request method that is either GET or POST. **By default it is POST.**

1. InsertionMode

Specify the way of populating the target container. The possible values are InsertAfter, InsertBefore and **Replace (which is the default).**

1. LoadingElementDuration

Here you can specify the value in milliseconds that controls the duration of the animation loading element.

1. LoadingElementId

HTML element that will be displayed when AJAX call is in-progress.

Generally we used this property to display the loader element i.e.

<div id="imgloader" style="display:none;position:absolute;top:50%;left:50%;padding:2px;">

<img src="~/Content/loader.gif" />

</div>

1. OnBegin

This property specifies the name of the java script function that is called just before Ajax starts. So here you can perform some validation or other operation that is required before Ajax starts

1. OnComplete

Name of the java script function that will be called when response data has been represented by the AJAX call.

1. OnFailure

Name of the java script function to call when AJAX request returns error.

1. OnSuccess

Name of the java script function to call after the AJAX request returns successfully

1. UpdateTargetId

This property is used to specify the ID of HTML element that is updated by using the response;

1. Url

Specify the URL to make call i.e. External URL link

**return View() :** It tells MVC to generate an HTML template to be displayed and sends it to the browser without making a new request.

**return Redirect() :** Makes a new request for the new action but in order to use this method we have to specify the full URL.

**return RedirectToRoute()** : Redirect to action from the specified URL defined in the route table that is defined in RouteConfig.cs file.

**return RedirectToAction() :** To redirect to a different action which can be in the same or different controller.

**return RedirectPermanent()** **:** Sends the browser an HTTP 301 (Moved Permanently) status code whereas Redirect will send an HTTP 302 status code.

**Example**

Let's say that you have users in your system. You also have an option to delete existing users. Your website has a resource /user/{userid} that displays the details of a given user. If the user has been deleted, you must redirect to the /user/does-not-exist page. In this case:

If the user will never be restored again, you should use RedirectPermanent so the browser can go directly to /user/does-not-exist in subsequent requests even if the URL points to /user/{userid}.

**Session** state can be **stored** in one of the following modes:

In - Process: **Stored** in the same **ASP**.**Net** Process.

State Server: **Stored** in the some other system.

SQL Server: **Stored** in the SQLServer database.

**Model Binding**

<https://www.tutorialspoint.com/asp.net_mvc/asp.net_mvc_model_binding.htm#:~:text=ASP.NET%20MVC%20model%20binding,browser%20in%20an%20HTTP%20request.&text=Model%20binding%20is%20a%20well,and%20the%20C%23%20action%20methods>.

ASP.NET MVC model binding allows you to map HTTP request data with a model. It is the process of creating .NET objects using the data sent by the browser in an HTTP request. The ASP.NET Web Forms developers who are new to ASP.Net MVC are mostly confused how the values from View get converted to the Model class when it reaches the Action method of the Controller class, so this conversion is done by the Model binder.

Model binding is a well-designed bridge between the HTTP request and the C# action methods. It makes it easy for developers to work with data on forms (views), because POST and GET is automatically transferred into a data model you specify. ASP.NET MVC uses default binders to complete this behind the scene.

Let’s take a look at a simple example in which we add a ‘Create View’ in our project from the last chapter and we will see how we get these values from the View to the EmployeeController action method.

Following is the Create Action method for POST.

// POST: Employee/Create

[HttpPost]

public ActionResult Create(FormCollection collection){

try{

// TODO: Add insert logic here

return RedirectToAction("Index");

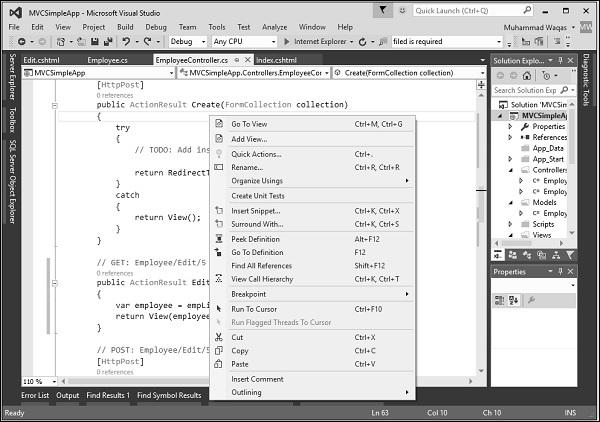
}catch{

return View();

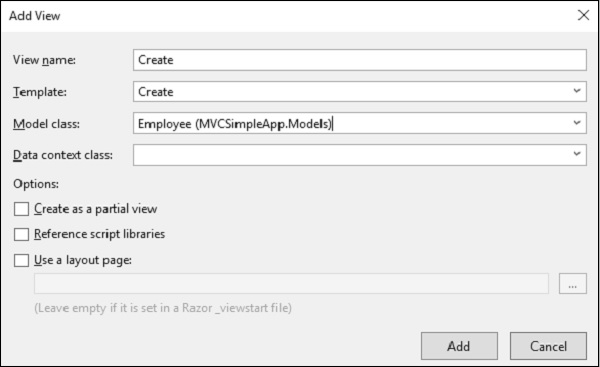
}

}

Right-click on the Create Action method and select Add View…



It will display the Add View dialog.



As you can see in the above screenshot, the default name is already mentioned. Now select Create from the Template dropdown and Employee from the Model class dropdown.

You will see the default code in the Create.cshtml view.

@model MVCSimpleApp.Models.Employee

@{

Layout = null;

}

<!DOCTYPE html>

<html>

<head>

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

<title>Create</title>

</head>

<body>

@using (Html.BeginForm()){

@Html.AntiForgeryToken()

<div class = "form-horizontal">

<h4>Employee</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

<div class = "form-group">

@Html.LabelFor(model => model.Name, htmlAttributes:

new{ @class = "control-label col-md-2" })

<div class = "col-md-10">

@Html.EditorFor(model => model.Name, new{ htmlAttributes =

new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Name, "",

new{ @class = "text-danger" })

</div>

</div>

<div class = "form-group">

@Html.LabelFor(model => model.JoiningDate, htmlAttributes:

new{ @class = "control-label col-md-2" })

<div class = "col-md-10">

@Html.EditorFor(model => model.JoiningDate, new{ htmlAttributes =

new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.JoiningDate, "",

new { @class = "text-danger" })

</div>

</div>

<div class = "form-group">

@Html.LabelFor(model => model.Age, htmlAttributes:

new { @class = "control-label col-md-2" })

<div class = "col-md-10">

@Html.EditorFor(model => model.Age, new { htmlAttributes =

new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Age, "", new{ @class = "text-danger" })

</div>

</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>

</div>

}

<div>

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

</div>

</body>

</html>

When the user enters values on Create View then it is available in FormCollection as well as Request.Form. We can use any of these values to populate the employee info from the view.

Let’s use the following code to create the Employee using FormCollection.

// POST: Employee/Create

[HttpPost]

public ActionResult Create(FormCollection collection){

try {

Employee emp = new Employee();

emp.Name = collection["Name"];

DateTime jDate;

DateTime.TryParse(collection["DOB"], out jDate);

emp.JoiningDate = jDate;

string age = collection["Age"];

emp.Age = Int32.Parse(age);

empList.Add(emp);

return RedirectToAction("Index");

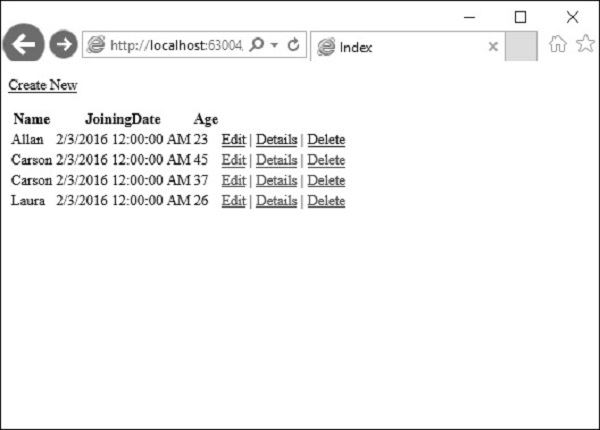
}catch {

return View();

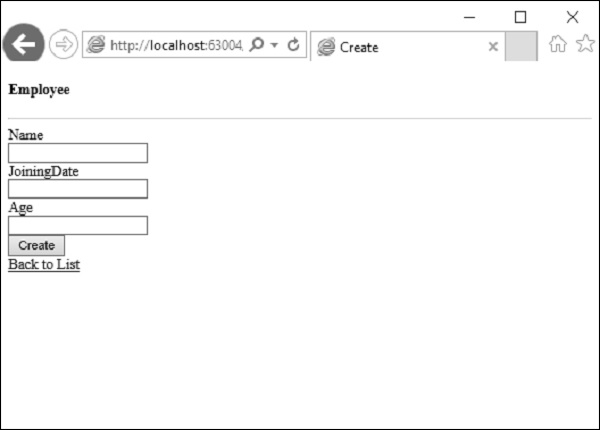
}

}

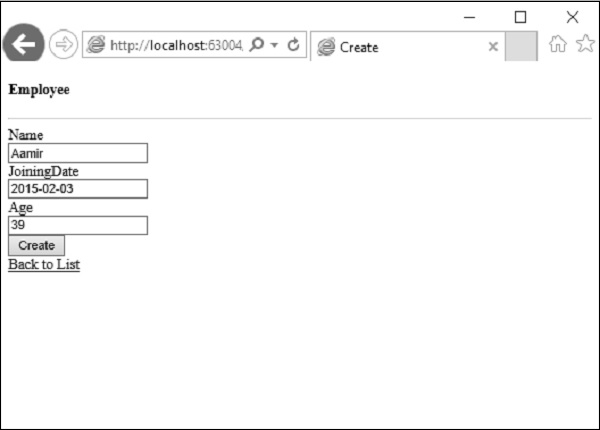
Run this application and request for this URL http://localhost:63004/Employee/. You will receive the following output.



Click the ‘Create New’ link on top of the page and it will go to the following view.



Let’s enter data for another employee you want to add.



Click on the create button and you will see that the new employee is added in your list.



In the above example, we are getting all the posted values from the HTML view and then mapping these values to the Employee properties and assigning them one by one.

In this case, we will also be doing the type casting wherever the posted values are not of the same format as of the Model property.

This is also known as manual binding and this type of implementation might not be that bad for simple and small data model. However, if you have huge data models and need a lot of type casting then we can utilize the power and ease-of-use of ASP.NET MVC Model binding.

Let’s take a look at the same example we did for Model binding.

We need to change the parameter of Create Method to accept the Employee Model object rather than FormCollection as shown in the following code.

// POST: Employee/Create

[HttpPost]

public ActionResult Create(Employee emp){

try{

empList.Add(emp);

return RedirectToAction("Index");

}catch{

return View();

}

}

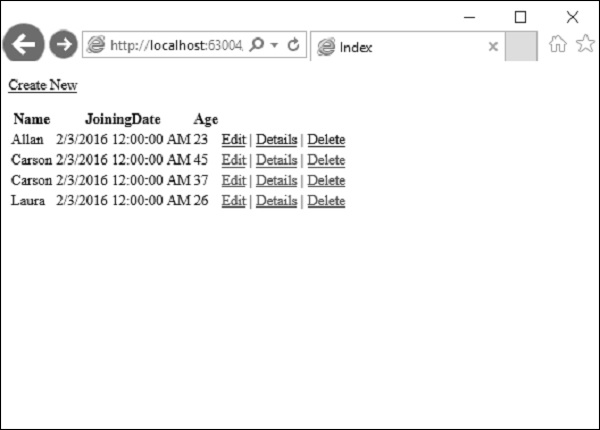
Now the magic of Model Binding depends on the id of HTML variables that are supplying the values.

For our Employee Model, the id of the HTML input fields should be the same as the Property names of the Employee Model and you can see that Visual Studio is using the same property names of the model while creating a view.

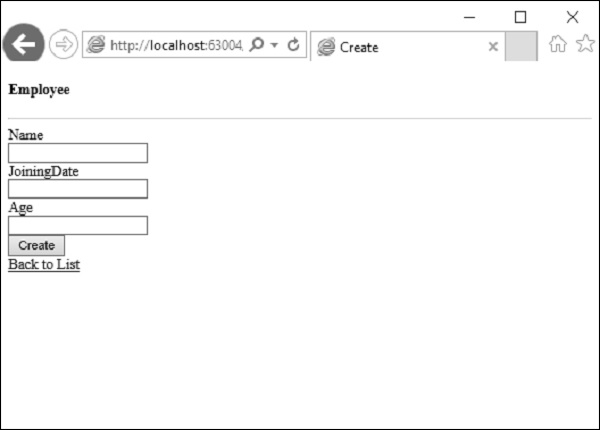
@Html.EditorFor(model => model.Name, new { htmlAttributes = new { @class = "form-control" } })

The mapping will be based on the Property name by default. This is where we will find HTML helper methods very helpful because these helper methods will generate the HTML, which will have proper Names for the Model Binding to work.

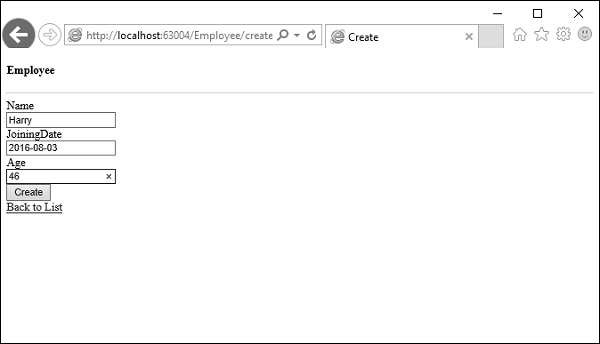
Run this application and request for the URL **http://localhost:63004/Employee/**. You will see the following output.



Let’s click on the Create New link on the top of the page and it will go to the following view.



Let’s enter data for another employee that we want to add.



Now click the create button and you will see that the new employee is added to your list using the ASP.Net MVC model binding.



**Send data from View to Action**

1. Using Traditional approach (**Request["txtAmount"].ToString()**)
2. Using the FormCollection Object
3. Using the Parameters
4. Strongly type model binding to view