**What are Action Selectors in ASP.NET MVC?**

The Actions are the public methods of a controller in ASP.NET MVC Application that responds to incoming HTTP Requests. The Action Selectors in ASP.NET MVC are the attributes that can be applied to the action methods of a controller and are used to control which action method gets invoked in response to a particular request. That means Action Selectors in ASP.NET MVC Framework help the routing engine to select the correct action method to handle a particular request.

**Types of Action Selectors in ASP.NET MVC.**

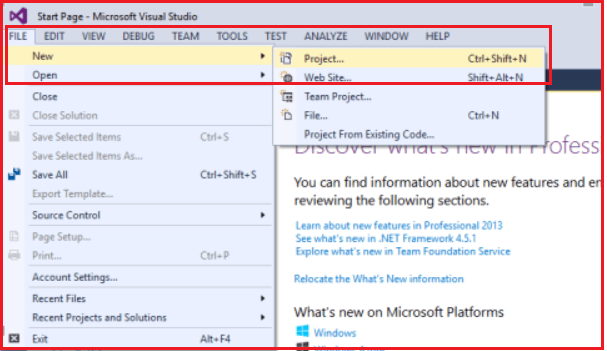
The ASP.NET MVC Framework provides the following three action selector attributes:

1. **ActionName**
2. **ActionVerbs**
3. **NonAction**

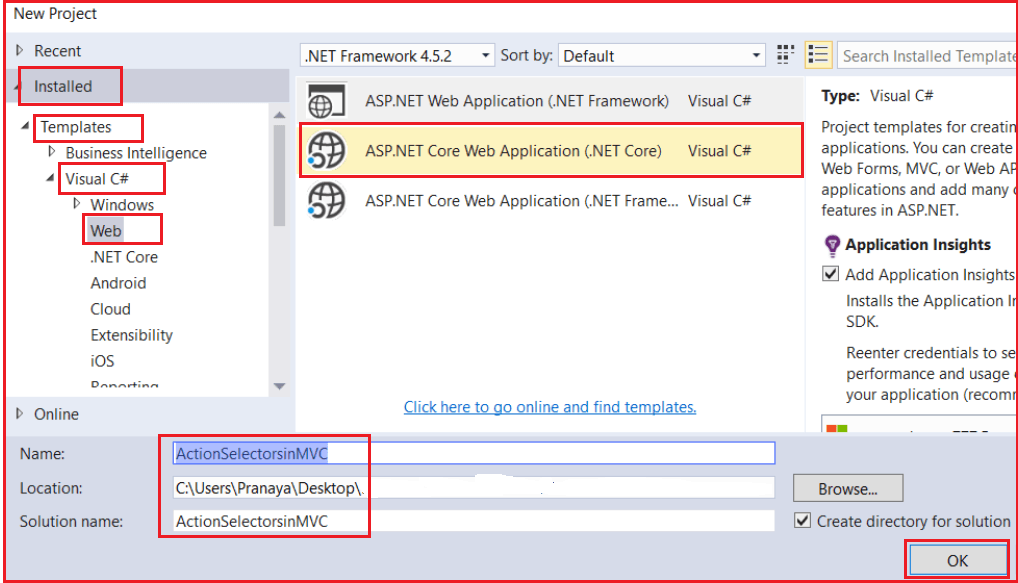
Let us understand the above Action Selectors with examples. To understand this let’s create an empty ASP.NET MVC application.

**Create an empty ASP.NET MVC application.**

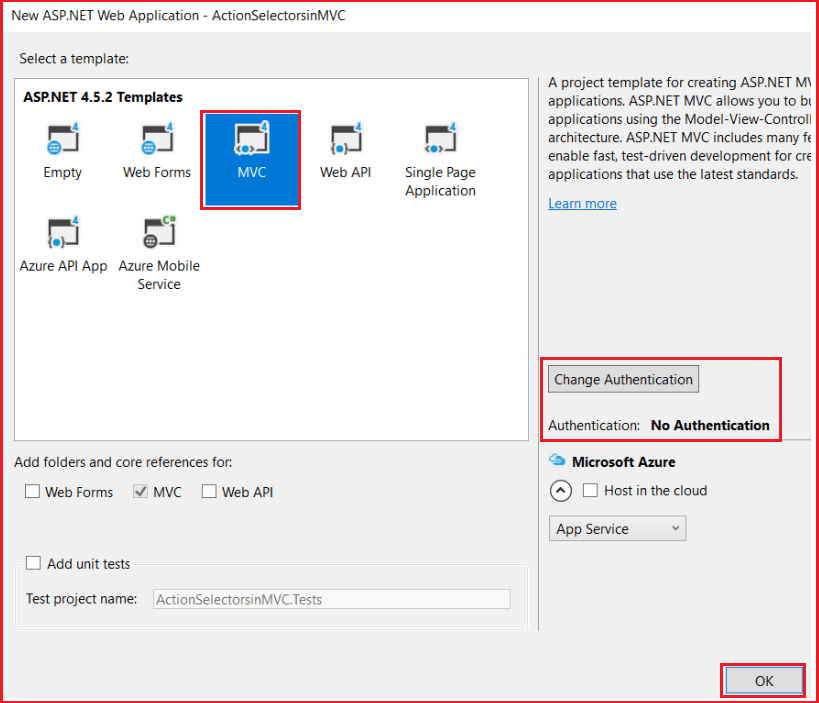
Open Visual Studio and create a new project by selecting **File => New => Project**option from the context menu as shown in the below image.



After clicking on the “**Project**” link a new dialog window will open. In that, we are going to select “**Web”** templates from the left pane, and from the middle pane we need to select the web template as “**ASP.NET Web Application**“. Provide the name and location and finally click on the **OK** button as shown in the below image.



Once you click on the **OK** button a new dialog will pop up with the name “**New ASP.NET Web Application**” for selecting the project Templates. From this dialog, we are going to choose the **MVC** project template and then we are going to choose the Authentication type for doing that just click on the **Change Authentication** button, a new dialog will pop up with the name “**Change Authentication**” here we are going to choose **No Authentication** and finally click on the OK button as shown below.



Once you click on the **OK** button, it will take some time to create the project for us.

**Adding Model classes in the Models folder:**

Right-click on the Models folder and click on **Add => Class** from the context menu and give the class name as **Product.cs.**Once you created the **Product.cs** class file, then copy and paste the following code into it.

**public** **class** Product

**{**

**public** **int** ProductId **{** **get**; **set**; **}**

**public** string ProductName **{** **get**; **set**; **}**

**public** **int** Price **{** **get**; **set**; **}**

**public** string Category **{** **get**; **set**; **}**

**}**

**public** **class** DataAccessElectronics

**{**

**public** List**<**Product**>** GetDataElectronics**()**

**{**

List**<**Product**>** ElectronicsProductList = new List**<**Product**>()**

**{**

new Product**()** **{** ProductId = 1, ProductName = "Desktop", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 2, ProductName = "Laptop", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 3, ProductName = "Router", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 4, ProductName = "Mouse", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 5, ProductName = "USB HDD", Price = 34000, Category = "Electronics" **}**,

new Product**()** **{** ProductId = 6, ProductName = "LCD", Price = 34000, Category = "Electronics" **}**

**}**;

**return** ElectronicsProductList;

**}**

**}**

The above code contains the following classes

1. **Product** – This is an entity class containing properties for Product information.
2. **DataAccessElectronics** – This class contains the method for returning all Electronics products.

**Note:** In a real-world example, the **Entity framework** can be used here to map with the database server.

**Understanding ActionName Action Selector in ASP.NET MVC:**

The ActionName action selector in the ASP.NET MVC Application is used when we want to invoke an action method with a different name, than what is already given to the action method. To understand the need and use of the ActionName selector, let’s modify the HomeController as shown below.

**public** **class** HomeController : Controller

**{**

**public** string Index**()**

**{**

**return** "Index action method invoked";

**}**

**}**

The URL “**/Home/Index**“ would invoke the Index() action method in HomeController. If you want to invoke the**Index()** action method, with the URL “**/Home/List”,** then you need to decorate the action method with the ActionName attribute as shown below.

**public** **class** HomeController : Controller

**{**

**[**ActionName**(**"List"**)]**

**public** string Index**()**

**{**

**return** "Index action method invoked";

**}**

**}**

Now, if you navigate to **/Home/Index**, you will get an error – “**The resource cannot be found**“. At the moment, the **Index**() action method is returning a string, but if it returns a view, should the view be named – **Index or List**? The List should be the view name.

**Let’s modify the Home Controller as shown below.**

Now, the Index Action Method returning a view.

**public** **class** HomeController : Controller

**{**

**[**ActionName**(**"List"**)]**

**public** ActionResult Index**()**

**{**

DataAccessElectronics DAE = new DataAccessElectronics**()**;

List**<**Product**>** ElectronicsList = DAE.GetDataElectronics**()**;

**return** View**(**ElectronicsList**)**;

**}**

**}**

Let’s add **Index.cshtml** within the Home folder and then copy and paste the following code into it.

@model IEnumerable**<ActionSelectorsinMVC.Models.Product>**

@{

ViewBag.Title = "Index";

}

**<h2>**Product List**</h2>**

**<table** class="table"**>**

**<tr>**

**<th>**

@Html.DisplayNameFor(model => model.ProductName)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Price)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Category)

**</th>**

**</tr>**

@foreach (var item in Model)

{

**<tr>**

**<td>**

@Html.DisplayFor(modelItem => item.ProductName)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Price)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Category)

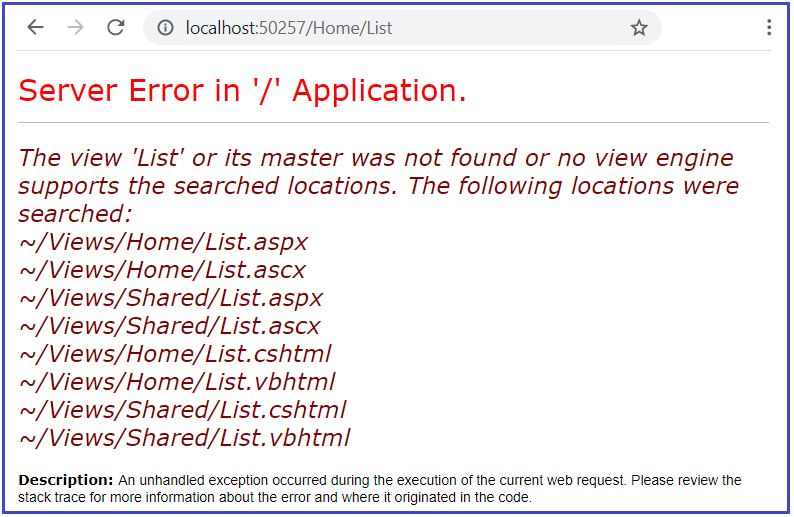
**</td>**

**</tr>**

}

**</table>**

Now run the application and navigate to Home/List and you will get the following error which proves that it will search for a view with the name List.cshtml, not Index.cshtml.



Now add a view with the name **List.cshtml** within the Home folder and copy and paste the following code into it.

@model IEnumerable**<ActionSelectorsinMVC.Models.Product>**

@{

ViewBag.Title = "Index";

}

**<h2>**Product List**</h2>**

**<table** class="table"**>**

**<tr>**

**<th>**

@Html.DisplayNameFor(model => model.ProductName)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Price)

**</th>**

**<th>**

@Html.DisplayNameFor(model => model.Category)

**</th>**

**</tr>**

@foreach (var item in Model) {

**<tr>**

**<td>**

@Html.DisplayFor(modelItem => item.ProductName)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Price)

**</td>**

**<td>**

@Html.DisplayFor(modelItem => item.Category)

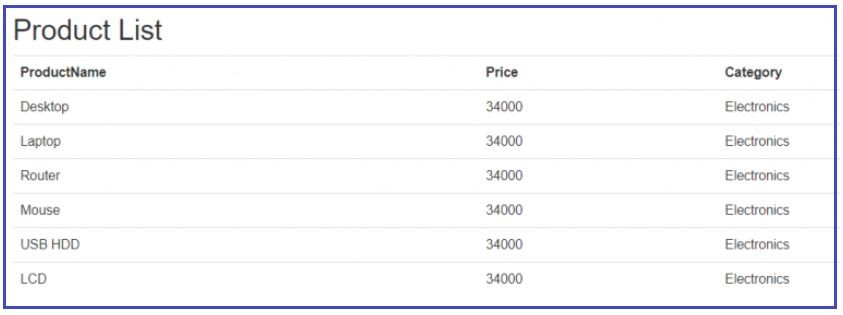
**</td>**

**</tr>**

}

**</table>**

**Now, run the application and navigate to /Home/List and it should display the following output as the response.**



Now for some reason, if you want to use **“Index”** as the view name then we need to use the other overloaded version of the View Extension method which takes the view name as a parameter. So, modify the controller action method as shown below. Here, you can notice, we have passed the first parameter as Index which is nothing but the view name.

**public** **class** HomeController : Controller

**{**

**[**ActionName**(**"List"**)]**

**public** ActionResult Index**()**

**{**

DataAccessElectronics DAE = new DataAccessElectronics**()**;

List**<**Product**>** ElectronicsList = DAE.GetDataElectronics**()**;

**return** View**(**"Index", ElectronicsList**)**;

**}**

**}**

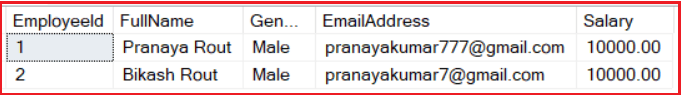
With the above changes in place, now when you run the application and navigate to **/Home/List**, it will display the results by using the Index view.

##### ****Action Verb Action Selector in ASP.NET MVC Application:****

We need to use the **Action Verb Selector**when we want to control the invocation of an action method based on the request type in the ASP.NET MVC Application. We can define two different action methods with the same name but one action method responds to an HTTP Get request while the other action method responds to an HTTP Post request.

##### ****Database Required:****

We are going to use the following Employee table in this demo.



Please use the below SQL Script to create the Employee table with the required test data.

CREATE TABLE Employee**(**

**[**EmployeeId**]** **[int]** PRIMARY KEY IDENTITY**(**1,1**)**,

**[**FullName**]** **[**nvarchar**](**100**)**,

**[**Gender**]** **[**nvarchar**](**10**)**,

**[**EmailAddress**]** **[**nvarchar**](**100**)**,

**[**Salary**]** DECIMAL**(**18, 2**)**

**)**

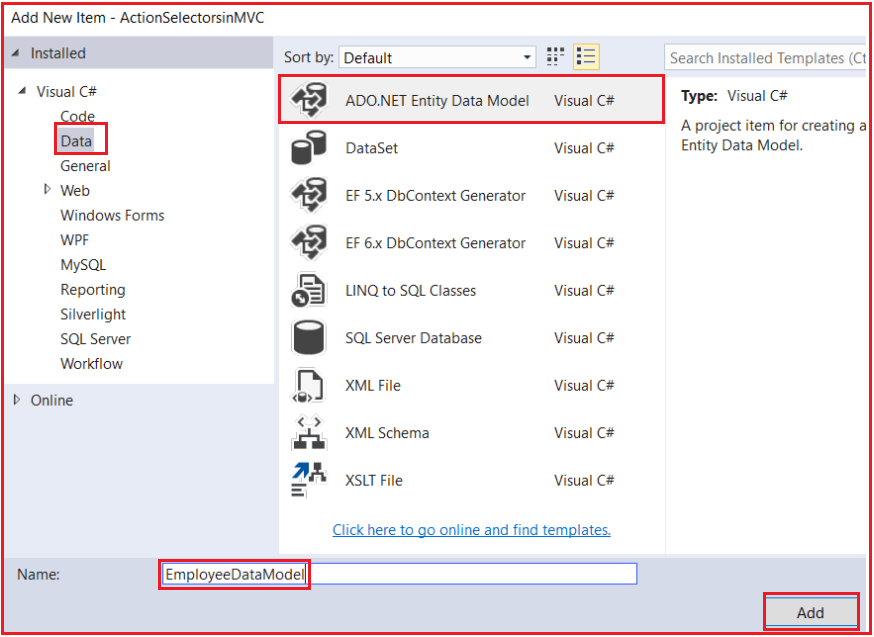
-- Populate the following test data

Insert **into** Employee values**(**'Pranaya Rout', 'Male', 'pranayakumar777@gmail.com', 10000**)**

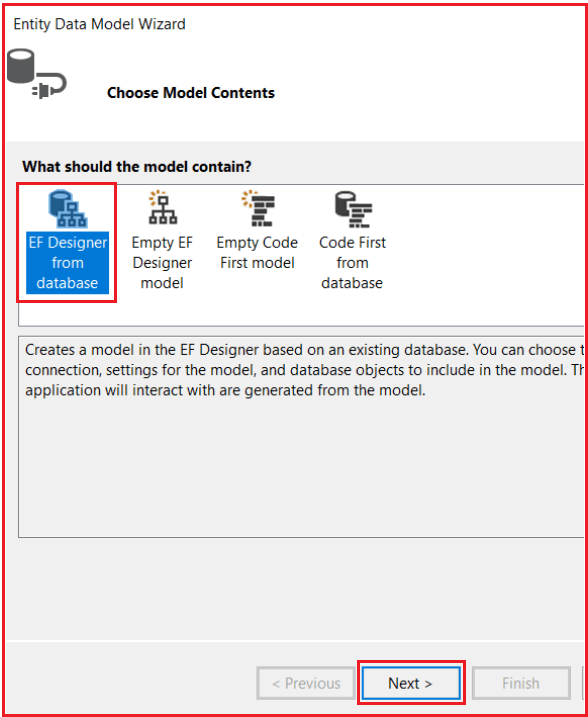
Insert **into** Employee values**(**'Bikash Rout', 'Male', 'pranayakumar7@gmail.com', 10000**)**

##### ****Adding ADO.NET Entity Data Model:****

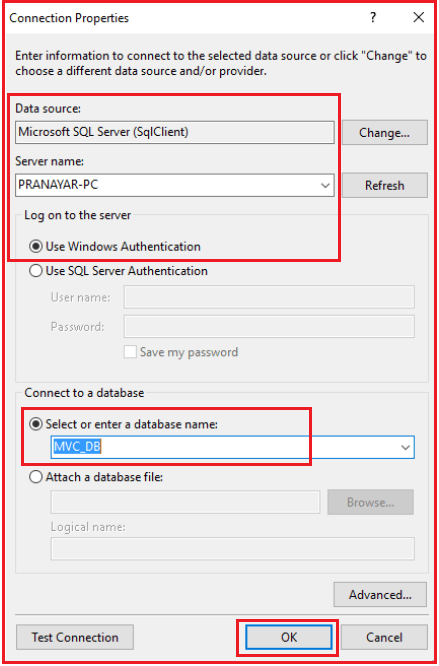
Right-click on the Models folder then select **Add => New Item** from the context menu. Then select **ADO.NET Entity Data Model**, provide a meaningful name such as **EmployeeDataModel** and click on the **Add** button as shown in the image below.



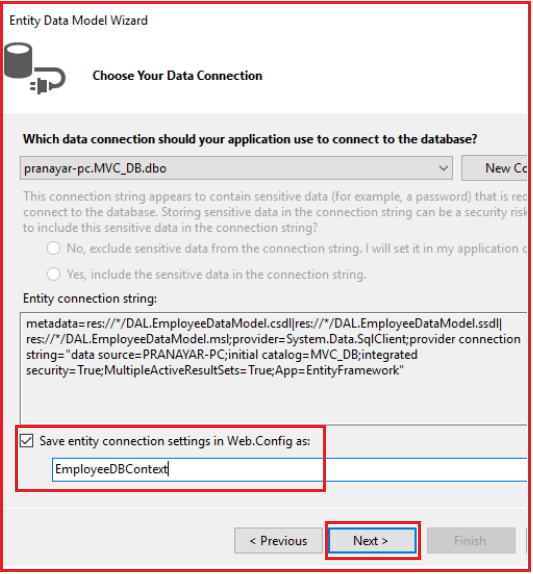
Here, we are going to use **Entity Framework Database First Approach** to interact with the database. So, from the **Choose Entity Model Wizard** window, select **Generate From Database** and click on the **Next** button as shown below.



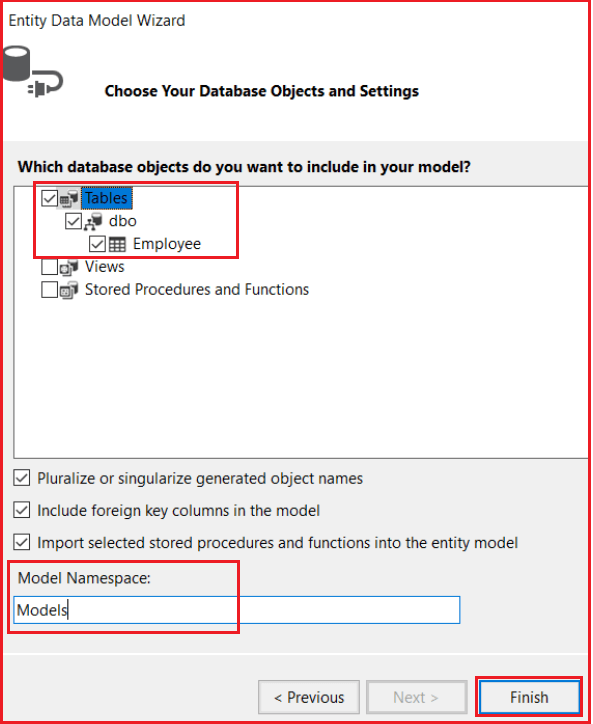
On the next screen, click on the **New Connection** tab and provide the necessary details to interact with the database. Then select the database and click on the **OK** button as shown below.



In the next step provide a meaningful name such as “**EmployeeDBContext**” for the Connection String that is going to create in the **Web.config** file and click on the **Next** button as shown below.



In the next step From Choose your database objects screen, choose the **Employee** table, provide the namespace name and click on the **Finish** button as shown below.



##### ****Adding Employee Controller:****

Let’s Add Employee Controller to the Controller Folder and copy and paste the following code. In the below code, the “Edit” action method that is decorated with the **[AcceptVerbs(HttpVerbs.Get)]**accept verb responds to the GET request, whereas the “Edit” action method which is decorated with **[AcceptVerbs(HttpVerbs.Post)]**accept verb responds to POST request. The default is GET. So, if we don’t decorate an action method with any accept verb, then, by default, the method responds to the GET request. In our example Index is not decorated with any action verb so it by default responds to the GET attribute.

**namespace** *ActionSelectorsinMVC.Controllers*

**{**

**public** **class** EmployeeController : Controller

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

**public** ActionResult Index**()**

**{**

List**<**Employee**>** ListEmployees = dbContext.Employees.ToList**()**;

**return** View**(**ListEmployees**)**;

**}**

**[**AcceptVerbs**(**HttpVerbs.Get**)]**

**public** ActionResult Edit**(int** id**)**

**{**

Employee employee = dbContext.Employees.Where**(**x =**>** x.EmployeeId == id**)**.FirstOrDefault**()**;

**return** View**(**employee**)**;

**}**

**[**AcceptVerbs**(**HttpVerbs.Post**)]**

**public** ActionResult Edit**(**Employee employee**)**

**{**

**if** **(**ModelState.IsValid**)**

**{**

dbContext.Entry**(**employee**)**.State = EntityState.Modified;

dbContext.SaveChanges**()**;

**return** RedirectToAction**(**"Index"**)**;

**}**

**return** View**(**employee**)**;

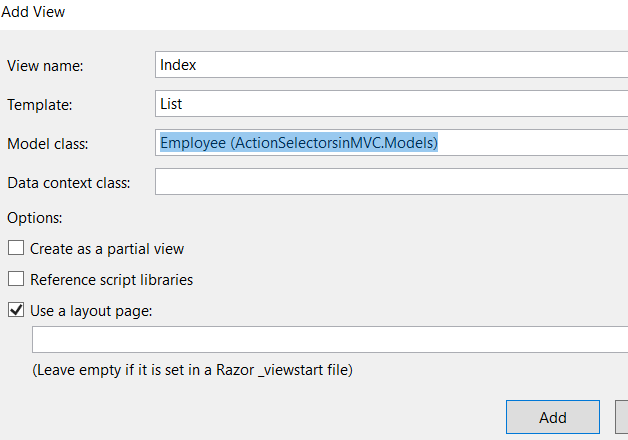
**}**

**}**

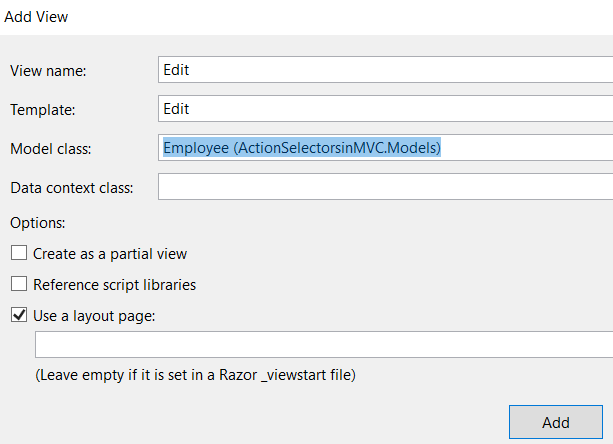
**}**

##### ****Add Index and Edit view.****

Right-click on Index view and select Add View and provide the following details and click on the Add button as shown below.



Right-click on the Edit action method and click on Add View and provide the following details and click on the Add button as shown below.



##### ****HttpGet and HttpPost Attribute in ASP.NET MVC:****

Instead of using **[AcceptVerbs(HttpVerbs.Get)]** and **[AcceptVerbs(HttpVerbs.Post)]** attribute, we can also use HttpGet and HttpPost attributes as shown in the below code. This is an alternative approach to using the AcceptVerbs attribute and the behavior is going to be the same i.e. the action method which decorates with the HttpGet attribute will only respond to GET Request whereas the action method which decorates with HttpPost attribute will only respond to the Post Request.

**public** **class** EmployeeController : Controller

**{**

EmployeeDBContext dbContext = new EmployeeDBContext**()**;

**public** ActionResult Index**()**

**{**

List**<**Employee**>** ListEmployees = dbContext.Employees.ToList**()**;

**return** View**(**ListEmployees**)**;

**}**

**[**HttpGet**]**

**public** ActionResult Edit**(int** id**)**

**{**

Employee employee = dbContext.Employees.Where**(**x =**>** x.EmployeeId == id**)**.FirstOrDefault**()**;

**return** View**(**employee**)**;

**}**

**[**HttpPost**]**

**public** ActionResult Edit**(**Employee employee**)**

**{**

**if** **(**ModelState.IsValid**)**

**{**

dbContext.Entry**(**employee**)**.State = EntityState.Modified;

dbContext.SaveChanges**()**;

**return** RedirectToAction**(**"Index"**)**;

**}**

**return** View**(**employee**)**;

**}**

**}**

##### ****Understanding Non-Action Attribute in the ASP.NET MVC Application with an Example.****

**public** **class** HomeController : Controller

**{**

**[**HttpGet**]**

**public** string Method1**()**

**{**

**return** "<h1>Method 1 Invoked</h1>";

**}**

**[**HttpGet**]**

**public** string Method2**()**

**{**

**return** "<h1>Method 2 Invoked</h1>";

**}**

**}**

As you can see in the above code, we have 2 public methods i.e. Method1() and Method2() in HomeController. As the above two methods are public methods, so, we can access these two methods using the below URL.

**Method1 can be invoked using URL /Home/Method1**  
**Method2 can be invoked using URL /Home/Method2**

Let’s say **Method2**() is doing some internal work, and we don’t want it to be invoked using a URL request. To achieve this, we need to decorate **Method2**() with the **NonAction** attribute as shown in the below code.

**[**NonAction**]**

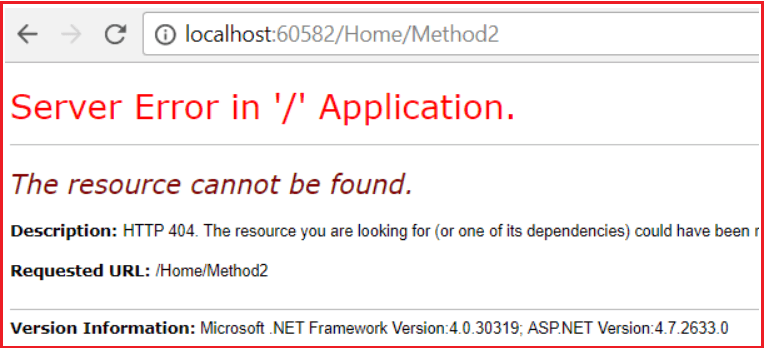
**public** string Method2**()**

**{**

**return** "<h1>Method 2 Invoked</h1>";

**}**

Now, if we navigate to URL **/Home/Method2**, we will get an error saying – **The resource cannot be found**as shown in the image below.



##### ****Another way to restrict access to methods in a controller is by making them private.****

**private** string Method2**()**

**{**

**return** "<h1>Method 2 Invoked</h1>";

**}**

In general, it’s a bad design to have a public method in a controller that is not an action method. If we have any such method for performing business calculations, it should be somewhere in the model and not in the controller. However, if for some reason, if you want to have public methods in a controller and you don’t want to treat them as actions, then use the Non-Action Selector Attribute in ASP.NET MVC Application.