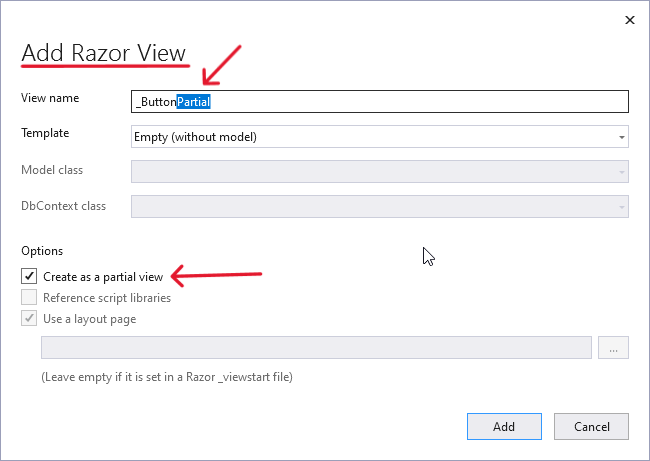
**Partial View:**

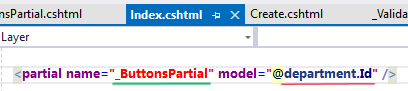
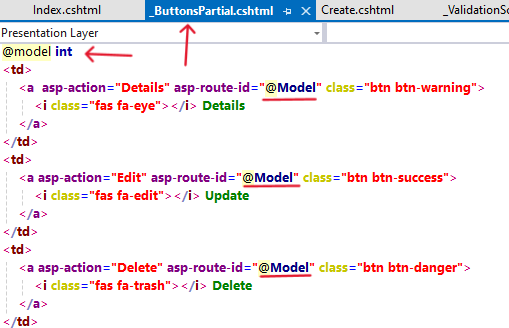
We use partial view to save repeated code

We render it in the views that has the common repeated code

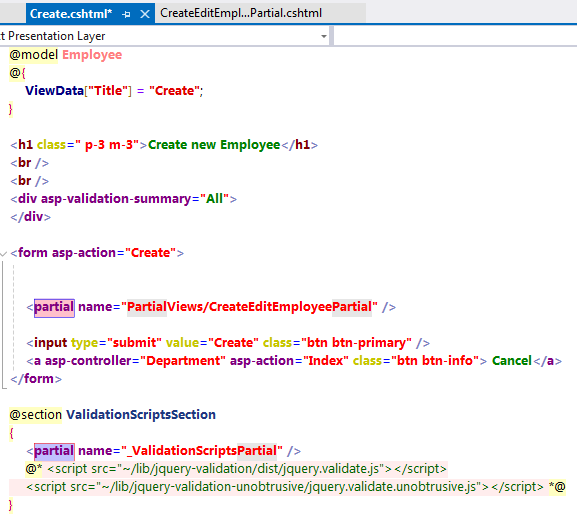
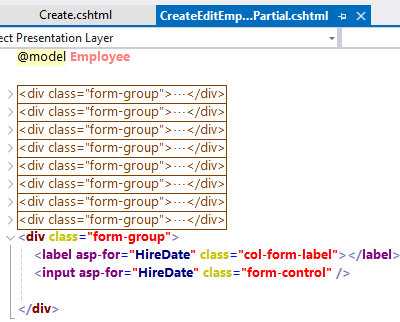
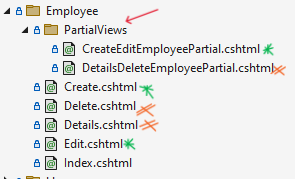
We used to name it \_XPartial 🡺 end its name with Partial word



Remove repeated buttons from Index cshtml files [Department & Employee] and add them in the Partial view of Buttons



Repeated code of Employee views 🡺



**View VS Partial-View Vs Layout VS Section Vs View Start VS View Import** 🡺 Interview Question

View: html page that action [action inside MVC controller] return it as a response

Partial View: repeated part of code written in separate file and rendered in the views containing that repeated code

Layout: structure of the page, views having the same structure use the same layout

View start: we write in it the code that we need to be added in the start of each view [like using a default layout for any view]

View import: the imports we use in more than one view so that we don’t have to repeat the imports using in each view

**Binding:**

Action [index] works as HTTP-Get, send piece of info from action to the view related to this action [the view that the action will return as response]

So binding is sending info from action to view

note the action that has verb get [the action sends info to view] while the action that has verb post [the action receives info from view]

ex: the action Index [http-get] sends model [Employees] to view [the model that will be displayed in the view], the action creates [http-post] receives info from the view [the model that will be created] when click submit button

the direction from view to action or action to view is one way [can’t send info from view to action then receive info from action to view]

the model is the main info send between view ⬄ action

the main info is the model … what if we need to send extra data? We can store the extra info in the storage of the view [Dictionary] as each view has dictionary[storage]

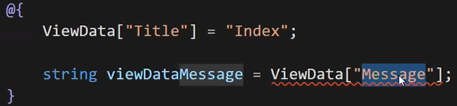
set the data we need to use in the view inside the action [in the action method in the controllers] then use it in the view [set the data in the view storage and use it in the View]

we can access the View Dictionary using 2 ways:

* ViewData[old]: more safe and faster than ViewBag, uses Dictionary [key value pairs]

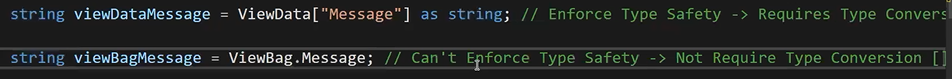
Compiler enforces type safety … requires type casting/conversion

Faster as its type is detected in compilation time not the runtime



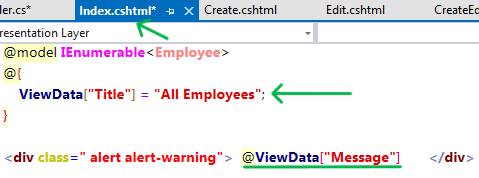
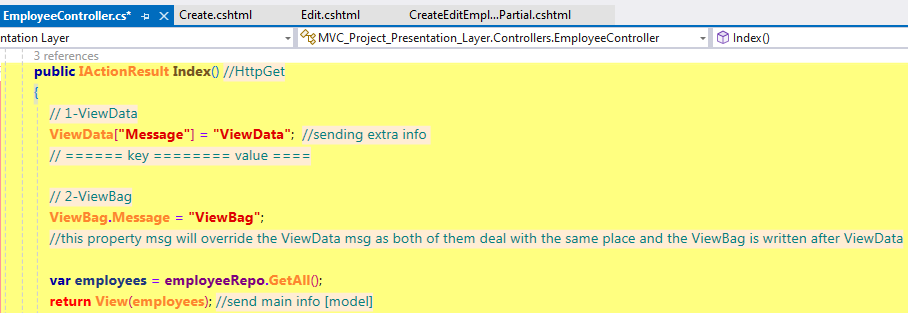
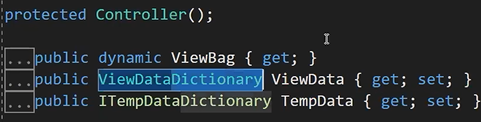
* ViewBag[new]: based on dynamic keyword [dynamic property]🡺 which means it will detect the type of the object in the runtime that’s why ViewData is better and faster, we use it to access the view dictionary in a different way

Don’t need to worry about the casting just use the dynamic ViewBag property and CLR will determine its type



Each View has its own dictionary, so if we used the key of index view in the create view we’ll find out that it will be null [if page is redirected to another page it will look for the key value in the dictionary of the redirected view]

The controller class inherit those 2 properties from Class Controller [not ControllerBase as ControllerBase doesn’t deal with views]



Note: ViewData & ViewBag store in the same place

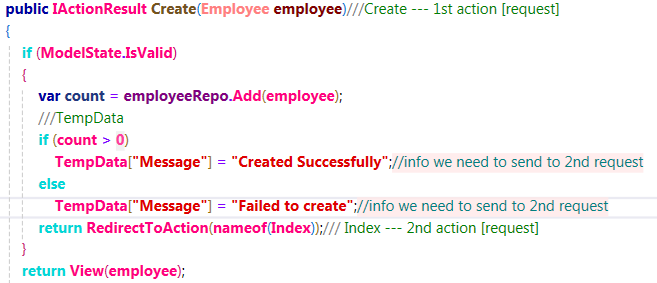
* TempData: is used to send data between 2 consecutive requests [from current request to next request]

Inside the Create action we have another action [redirect to another view] Index action

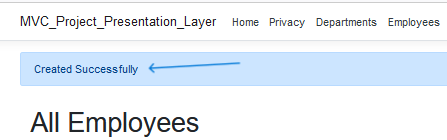
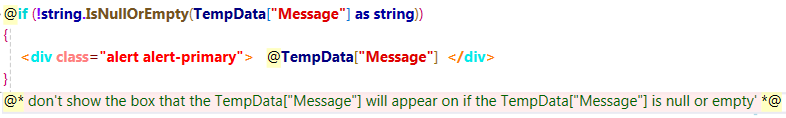
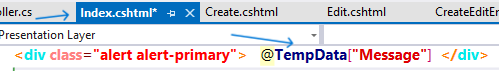
We need to send info from the Create action to the Index action… send a piece of information to display it in the next action [after the Create action send me info about the object if it is created or not]

TempData is similar to ViewData in structure both of them are dictionary but with different usage

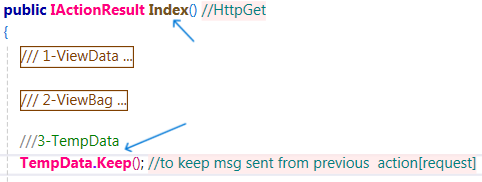
TempData & ViewData have different dictionaries



display the message in the Index View



If we need to keep the TempData sent to the index to send it to another action [another request] go to the Index action and Keep it 🡺

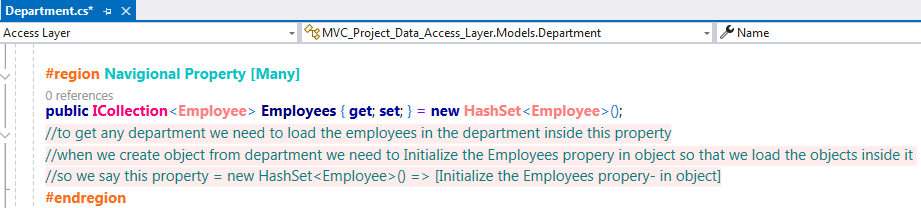


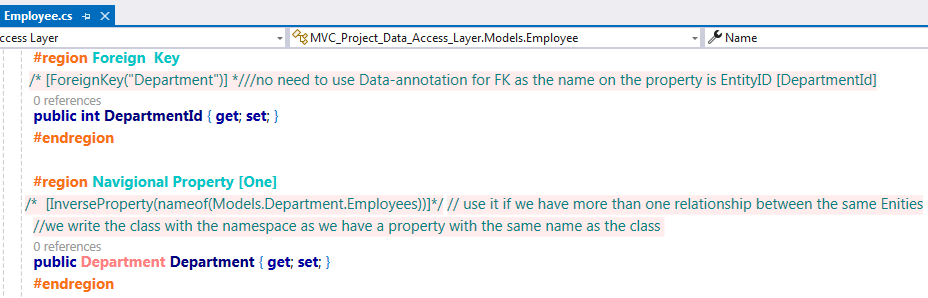
<https://www.c-sharpcorner.com/blogs/viewdata-vs-viewbag-vs-tempdata-in-mvc1>

**Employee Department Relationship [one Department🡺 many Employee]**

PK of 1 🡪 many [add it as a FK in Employee table]

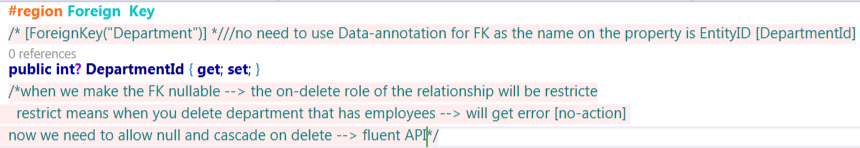
As we are working Code 1st so we’ll do that using navigational property

Go to the DAL project 🡺 Employee Model [Employee only works in one Department] 

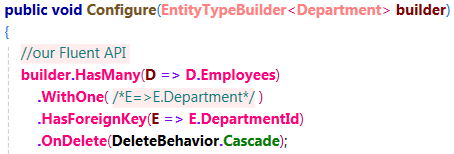


If The Fk doesn’t allow null 🡺 on-delete [when we delete department that has employees] action will be cascade[delete all employees in that department] while if at allows null the action will be no-action

To make it allow null and on-delete cascade 🡺do it fluent API



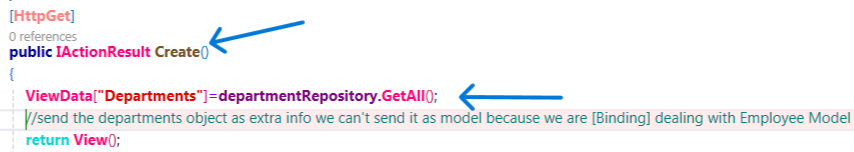
We have navigational property in both sides so we can make configures [fluent API] in any configuration class of them [remember if we make navigational property in one side (Department) we go and continue our configuration in the same side (DepartmentConfigurations)]



**Select**

We need to add input for the Employee create and update pages so that user can choose the Department from that input [dropdown list]

* Allow DI for DepartmentRepository in the EmployeeController Constructor and make a read-only attribute for that object of type IDepartmentRepository so that we can save the objects we get from the constructor in it and use it in the create method… the department repository is responsible for getting departments data from database
* Send the data we get from the DepartmentRepository as extra info using ViewData so that we can display them in the Employee Create/update view



Using the ViewData in the partial view🡺

