



* Action result – return type, helper – actual return value
* When we have multiple return type based on condition, then specify ***IActionResult*** as a return Type. It is parent of all, so it will be able to return any type from above.
* Entity Framework for database

1. Microsoft.EntityFrameworkCore.SqlServer
2. ………………………………………………….Relational
3. Microsoft.EntityFrameworkCore

* Here 1st is dependent on 2 and 2nd is on 3rd
* To enable migrations – install – Microsoft.EntityFrameworkCore.Tools
* Annotation for PK – [Key]
* Use toastr js for batter notification
* Use partial view for temporary data notification
* Use ‘**TempData’** for showing notification on view page which will be destroyed after refresh of any pages
  + **Seek & Peek** method of TempData,
  + MUST Take a look at that
* Add custom validation in ModelState
  + **ModelState.AddModelError(keyName, ErrorMessage);** **-- Add custom validation to ModelState**
  + Keep this inside any if block which will check some condition for our custom validation.
  + But default custom validation will not be on client side
  + We can also override any existing key
  + <div **asp-validation-summary=”All ”**></div> -- Validation summary tag helper
* N-tier Architecture
  + Project.DataAccess – Everything related to data, Like ApplicationDbContext, Migrations, Repositories, IRepositories
  + Project.Models – for all the models, ViewModels
  + Project.Utility – For all the generic utility, like email options, payment settings, session extensions, Static details OR constants of our website
* Repository pattern
* IRepository -> Repository = All common Create, Read, Find & Remove method, generic, no class specific, This will be of generic type repository, which will take class name
* IUnitOfWork -> UnitOfWork = All related to database like \_Db.SaveChanges(), Only One Dependency Injection register in program.cs file, This file will be parent riper for all the repositories,
* One dependency Injection will be sufficient for all the repository
* We don’t need to Inject it for all the repositories
* And in all the controllers we will only create instance of IUnitOfWork repository, So whenever we’re suing more than one model/repository in our controller, we don’t need to create object of each repository in our controller, One – IunitOfWork will be sufficient for all repos
* Areas in .Net Core
* Like admin area, Customer area
* Each area will have their own Model View Controller
* Also specify area annotation above the controller like : [Area(“Admin”)]
* **SelectListItem** Class – Take a look in Microsoft documentation in ASP.NET Core 6.0
* Rendered as an HTML <option> element with specified attributes values.
* NameSpace - Microsoft.AspNetCore.Mvc.Rendering;
* Package – Microsoft.AspNetCore.Mvc.ViewFeatures;
* [ValidateAntyForgeryToken] annotation is also inside this same package
* Use JS libraries like : Sweet alerts, Toastr JS, Bootswatch for bootstrap themes
* Tools -> debugging -> enable automatically close = FOR console app, No need of ReadKey() to stop window
* To parse html content ***@Html.Raw(Model.Product.Description)***
* Make note that – For ***disabled input fileds values will not be passed to Controller***
* In some cases we might not want some of the properties of the model to validate,
* In that case specify following data annotation to the peroperty of Model
* **[ValidateNever]**
* This is used, When we have property that refer to another parent table model
* When We have some property inside Model BUT WE DON’T WANT IT TO MAP TO THE TABLE OF DB In that case specify following [DataAnnotation]
* **[NotMapped]**
* So property with this DataAnnotation will not be mapped to Table in DB && also not considered in migration
* **Select All & ctrl + k + d –** To align Code properly, IN **Visual Studio**
* In the View File, to get the QueryParameter Values:
* @{

**var status = Context.Request.Query[“status”];**

}

* This will give the value of query parameter named as status from URL