***ASP.NET CORE MVC version-5.0.0***

* Create project ASP Web App(MVC)
* Create Model class in models folder
* We downloaded ---Tools--NUGET--Solution---sqlserver-entitycore.sqlserver--Install
* SQL Server Download---Then Download Management Studio- Create a Server-'.'
* Create Data Folder---create class— named ApplicationDbContext:

**public class ApplicationDbContext :DbContext{**

**public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)**

**{**

**}**

**public DbSet<Category> Category { get; set; }**

**}**

* From each model we create we have to create a table for that we do migration as well as we also have to specify

**(public DbSet<modelName> modelName { get; set; }** and then do migration.

* In StartUp.cs file---

**public void ConfigureServices(IServiceCollection services)**

**{**

**services.AddDbContext<ApplicationDbContext>(options =>**

**options.UseSqlServer(**

**Configuration.GetConnectionString("DefaultConnection")));**

**services.AddControllersWithViews();**

**}**

* **Database Settings---** appsettings.json file---Provide

**ConnectionStrings--"DefaultConnection": "Server=.;Database=Rayin;Trusted\_Connection= True;MultipleActiveResultSets=True**"

* Add a package-tools**-entitycore.tools**
* Tools-Nuget-Package Manager Console
* **To Add Table-- add-migration 'name of it'(example-addCategoryToDatabase)**
* **To** **Update DB**-- Package Manager Console**- update-database**
* **To Alter Table/Column- add-migration AddRequiredFields**
* Check in SQL-db created and the table created
* Create Controller for model-same name as model--right click on name of method -add view- create a razor view
* **For each method(action) we need to create a razor view**
* Inside view its all html---for each model we have each view folder-in each view there are views for actions(CRUD operations)
* **The action name(CRUD) has to be same in controller, same name as the file name for view and same for the asp-action where we are specifying in view(button)**
* **View**, we can use **TAG HELPERS**- which are:
* **asp-for**--we can directly access the attribute of model**(@obj.Name**)
* **asp-action--**for post method in form- we can provide an action name specified in controller to call that action
* **asp-validation-for**--we can do validations for each field of model(in model we can specify the condition for each field/prop ex**:[Range(1,int.MaxValue,ErrorMessage = "Display order must be greater than 0")])**
* **asp-route-Id**-- used in edit and delete functionality to get the respective id of the row(we can specify it in the button)
* **asp-controller**- used to specify the controller if we have same action name in more than one controller
* **asp-validation-summary**-- can be two types=ModelOnly/All
* **Controller**--inside controller we will specify diff actions:
* **for add/creating** there is only one action-to add to db and save changes
* **for getting** also we have one action in which we directly get the dta from db in list and display a table in view
* **for edit**, we have to have 2 actions-one is for finding the id and going to that page and another is for updating the database where we take the object as argument
* **for delete**, we have to have 2 actions-one is for finding the id and going to that page and another is for updating the database where we take the object as argument.
* **For validations:**
* **Server-side**-- we can specify the tag helpers in the view(asp-validation) and the conditions on the each field/prop we want:

**[Required]**

**[Range(1,int.MaxValue,ErrorMessage = "Display order must be greater than 0")]**

* **Client-side**---we can specify the tag helpers in the view(asp-validation-summary) and in view at end we can specify this it will do validations:

**@section Scripts**

**{**

**@{<partial name = "\_ValidationScriptsPartial"/>}**

**}**

* If we want to create a model and use another model inside that then we can create an attribute for that model

**[DisplayName(“Category Type”)]**

**public int CategoryId**

**[ForeignKey(Name=” CategoryId”)]**

**public virtual model modelName{}**

* **Eager Loading—**that is used to load the other models from database and their values

**IEnumerable<Product> objList = \_db.Product.Include(u=>u.Category).Include(u=>u.ApplicationType);**

* + Instead of using the below, we should use eager loading

**//foreach (var item in objList) //{ // item.Category = \_db.Category.FirstOrDefault(u => u.Id == item.CategoryId); // item.ApplicationType = \_db.ApplicationType.FirstOrDefault(u => u.Id == item.ApplicationTypeId);**

**//}**

* **UPSERT:-** Here we are doing update and creation at one time
* It can be differentiated based on the id, like(Model.Id!=0) then its update otherwise its create
* In Views- we are using that asp-items for getting Category and ApplicationType
* We are also writing an onclick function for giving an alert if the image is not passed
* We are writing javascript for it:

**function validateInput()**

**{if(document.getElementById("uploadBox").value=="")**

**{ Swal.fire(**

**'Error!',**

**'Please upload an Image!',**

**'error'**

**) return false;**

**}return true;}**

* **VIEWMODEL-**We are creating view model inside model’s folder where we are creating the productVM so that instead of creating object for product again in controller we can directly access from there, also we are specify the IEnumerable<>for both Category and ApplicationType and using them
* **ViewModel** is also used if we are using other models in one model then that we can define in view model and directly from view model we can get all those and use
* We are also creating a constant file for specifying the path(Image Path)
* We can create Partial View, which is used to use it in a view it is not called by controller, it is called in a view itself that is called by Partial view**.**

**@foreach (var prod in Model.Products)**

**{**

**<partial name="\_IndividualProductCard" model="prod" />**

**//display all product**

**}**

* We have to add **Session** then go in **StartUp.cs** file in configuration add services:

**services.AddHttpContextAccessor(); services.AddSession(Options => { Options.IdleTimeout = TimeSpan.FromMinutes(10); Options.Cookie.HttpOnly = true; Options.Cookie.IsEssential = true; });**

**app.UseSession();**

* Create a folder Utilities and a class “**SessionExtensions**”, statis class, add get and set methods in it.

**public static class SessionExtensions { public static void Set<T>(this ISession session, string key, T value) { session.SetString(key, JsonSerializer.Serialize(value)); } public static T Get<T>(this ISession session, string key)**

**{**

**var value = session.GetString(key);**

**return value == null ? default : JsonSerializer.Deserialize<T>(value);**

**} }**

* **For registration/Login:**
  + Go in to applicationDbContext and inherit that class:
    - **public class ApplicationDbContext : IdentityDbContext**
* Install **Microsoft**.**AspNetCore.Identity.EntityFrameworkCore and Microsoft.AspNetCore.Identity.UI**
* In Startup.cs file add
* **app.useAuthentication()**
* **services.AddDefaultIdentity<IdentityUser>().AddEntityFrameworkStores<ApplicationDbContext>();**
* Also add migration:
* **add-migration scaffoldIdentityRazorClassLib**
  + ASP.NET has inbuilt method to add Identity,
    - **Right click on Project---Add—Add Scaffolded Item----override all files----select layout.cshtml---select applicationDbContext**
    - This will add a folder named Areas in your project, these pages are Razor Pages not MVC pages so we have to add tht in Startup.cs file in endpoints and then we can use.

**endpoints.MapRazorPages();**

* We have an inbuilt functionality for it, in Shared-\_LoginPartialView

Add it before <ul class>

**<partial name="\_LoginPartial"/>**

* + Create a model-ApplicationUser inherits IdentityUser
    - **public class ApplicationUser : IdentityUser**
    - in ApplicationDbContext—add
    - **public DbSet< ApplicationUser>{get; set;}**
    - **add-migration AddApplicationUserToTable, update-database**
  + In layout.cshtml file, on button of Register/login we have use
    - **asp-area=”Identity” asp-page=”/Account/Register”(path comes here)**
    - this is same as controller and action this also we can use as it is inbuilt functionality.
* **Authorization**: If we have provide authorization then just add **[Authorize]** tag on top of Controller
* We are using **BindProperty** with the help of which we can directly use that object inside method.

Example: **[BindProperty]**

**public ProductUserVm ProductUserVM{get; set;}**

* To get User details we have inbuilt package that helps in getting all that

**var claimsIdentity = (ClaimsIdentity)User.Identity;**

**var claim = claimsIdentity.FindFirst(ClaimTypes.NameIdentifier);**

**OR**

**//var userId = User.FindFirstValue(ClaimTypes.Name);**

* **MAILJET:**
* We will create a class in Utility Folder for sending mail **“EmailSender”.**
* We have to create an account in Mailjet, so send email. From there we have to go in **account settings** and get the apikey and secretkey and use them in code
* The asp.net has inbuilt library for mailjet install the **extension**
* Also in Startup.cs file, we have to add:
  + **services.AddTransient<IEmailSender, EmailSender>();**
* In mailjet, u will get **the whole code** copy paste make the req changes
* For sending email also, we have an inbuilt method in **Register.cshtml.cs** file as “**SendEmailAsync**”, u can call it directly in your controller
* We can store the apikey and secretkey in **appsettings** file and then use it in method using **IConfiguartion** library
* We will create a file in Utility folder as “mailsettings” where we will define two attributres ApiKey and SecretKey and then using this we populate the values in that file using **IConfiguartion** library and then use in EmailSender file.
* **Authorization**: if the **user is not Admin** then he/she should only see be authorized for CartController/HomeController for other we will add :
* **[Authorize(Roles=WC.AdminRole)]**
* **Using Microsoft.AspNetCore.Authorization;**
* Also in **Layout.cshtml** file we have to add the condition for the content management dropdown list**:**
* **If(User.IsInRole(WC.AdminRole))**
* **In Register.cshtml.cs,** also we will put a condition in **“OnPostAsync”** method thatif the user is admin then only he/she can create/register a user with Admin role otherwise he/she has to be a customer role only.
* Also in **WC**(constants file) we will define a variable for “**EmailAdmin**”, so that the emails are delivered from admin account.
* **AZURE DEPLOYMENT**
  + **Create a sql database in azure, create sql server, configure Database as Basic, 0.5 GB**
  + **Then in ssms(sql studio) right click on urd b the task🡪deploy on azure sql db🡪take the server name from azure then password and connect, also don’t signin—cancel, then go to sql db in azure-set firewall🡪 clientip given provide tht in client and end IP, allow azure firewall setting🡪Yes-Save**
  + **Then in ssms- connect and wait**
  + **Then connect to that server in ssms**
  + **Then go in VS change ethe connection styring, copy it from azure and paste here, please provide the password in connection string and save**
  + **Then rebuild the solution**
  + **Then click on project and publish, create an app instance and publish**