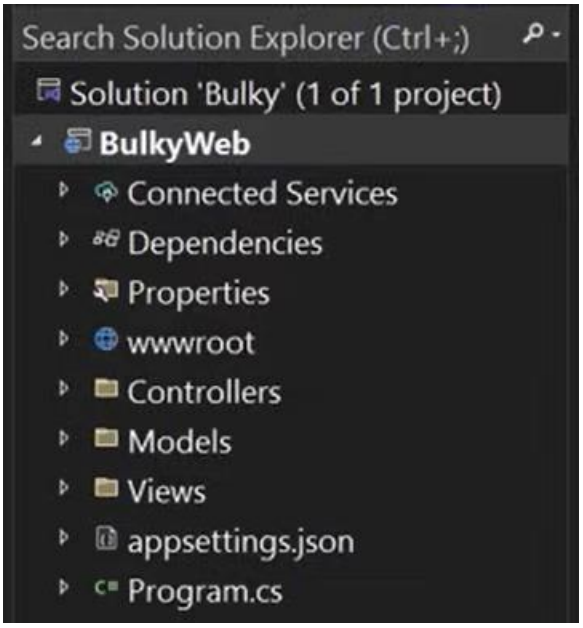
Dot NET core MVC

Tutorial - [Introduction to ASP.NET Core MVC (.NET 8) (youtube.com)]

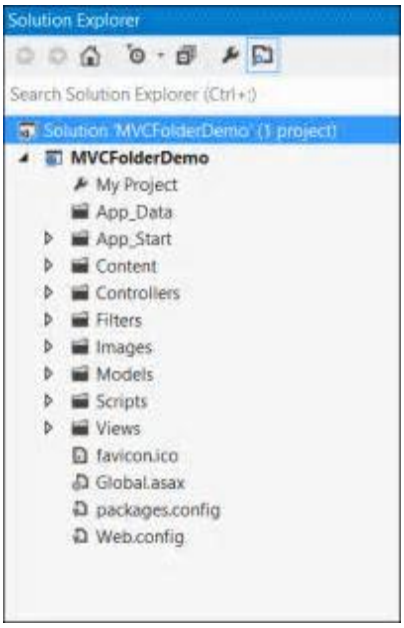
.Net core – fast, open-source, cross platform, Built-in Dependency Injection, Cloud friendly, high performance

Foler Structure –

.net core MVC



.net MVC -



A solution can have multiple projectss

Project File – target Frameowork, nullable, Implicit using

Dependencies – Frameworks -EntityFramework core

Properties – Launch Settings.json – IIS Settings, Profiles – Http, https. IIS

Environment variable – global variable

wwwroot – has static contents – CSS, js, lib-bootstrap, favicon (can have PPT, files, images)

Data – in EFCore, has DBContext.cs files

Migrations – in EFCore, migration files will be stored here

Controllers – handles the user request and acts an interface between Model and View

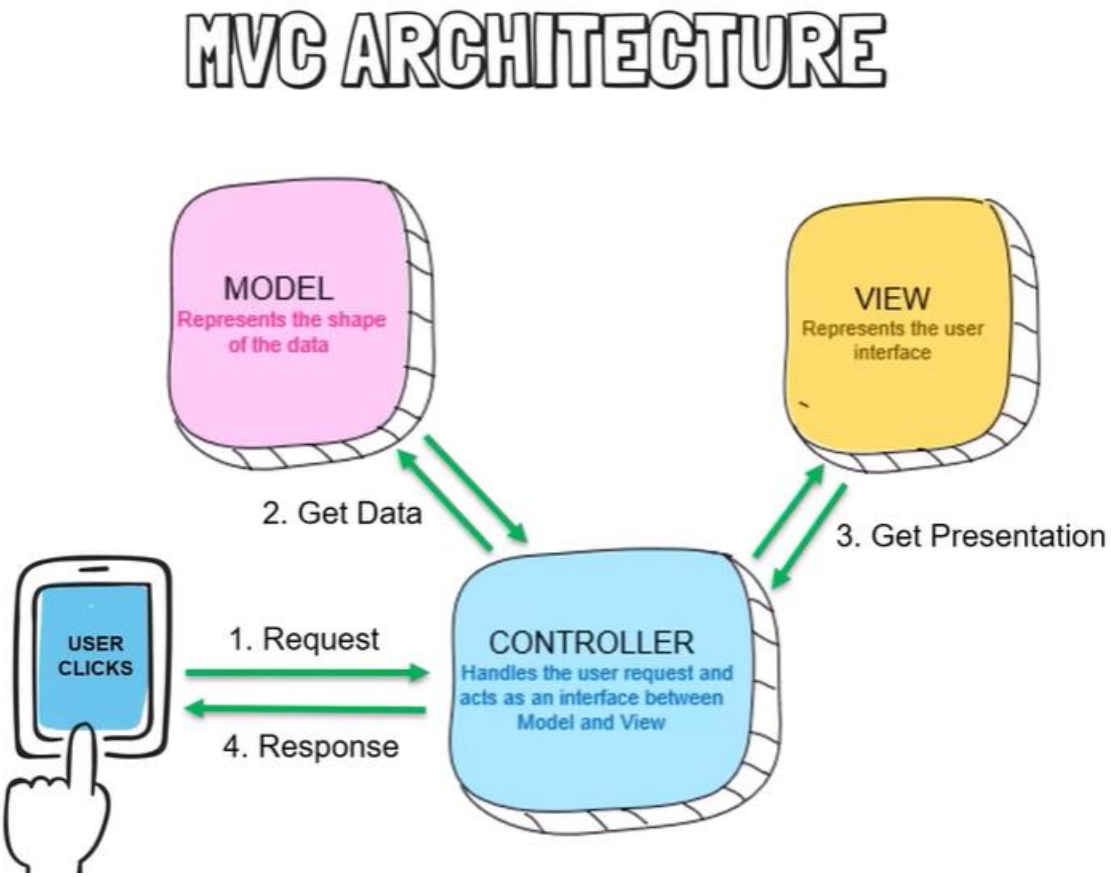
Models – Represents shape of data ,a class file

Views – User interface

appsettings.json / appsettings.Development.json – connection string, any secret key

Program.cs – (also has startup.cs in old .net core) builder, app var builder= WebApplication.CreateBuilder(args);

- **Add services** – builder.Services.AddControllerWithView();, var app=builder.Build();
- **Configure HTTP pipeline** – when a request comes to an application, how it should be handeled
    - App.useHttpRedirection()
    - App.UseStaticFiles();
    - App.UseRouting();
    - App.UseAuthorization();
    - App.MapControllerRoute() – default route
    - App.Run()



Action Methods – endpoints in controller

Routing – states what controller and which action method should be used

The URL pattern for routing is considered after the domain name.

- https://localhost:55555/Category/Index/3
- https://localhost:55555/{controller}/{action}/{id}

URL	Controller	Action	Id
https://localhost:55555/Category/Index	Category	Index	Null
https://localhost:55555/Category	Category	Index	Null
https://localhost:55555/Category/Edit/3	Category	Edit	3
https://localhost:55555/Product/Details/3	Product	Details	3

Controller –

- Should be in controller folder
- Should have controller suffix
- Have action methos

Model –

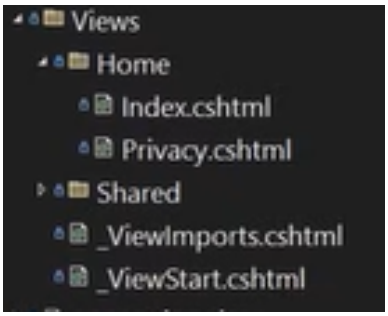
- Cs files has properties
- Represents data
- Not always needed for view
- Data annotation to mark a property as primary key
- [validateNever] to never validate a PROPERTY

```
namespace BulkyWeb.Models
{
    public class Category
    {
        [Key]
        public int Id { get; set; }
        [Required]
        [DisplayName("Category Name")]
        public string Name { get; set; }
        [DisplayName("Display Order")]
        public int DisplayOrder { get; set; }
    }
```

Views –

- Cshtml files

- Folder structure is same as controller
- _layout.cshtml is master page, has renderbody() to render whatever passed by view from controller



_ValidationScriptsPartial – a partial view, has reference for validation scripts

_ViewStart.cshtml – configure default page

```
@{
    Layout = "_Layout";
}
```

_viewImports.cshtml – for global usage in views

```
@using BulkyWeb
@using BulkyWeb.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

IActionResults – interface declared in .net, which has all possible return types

Entity Framework core –

- Add packages using package manager
    - Microsoft.EntityFrameworkcore
    - Microsoft.EntityFrameworkcore.SqlSever
    - Microsoft.EntityFrameworkcore.Tools – for using commands like add-migration
- Add connection string in appsettings.json

```
appsettings.json + × Program.cs  Applica...text.cs
Schema: https://json.schemastore.org/appsettings.json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Server=.;Database=Bulky;Trusted_Connection=True;TrustServerCertificate=True"
  }
}
```

- Add new folder – Data and create a class dbContext.cs. Dervie from DBContext and add DBSet property

```
using Microsoft.EntityFrameworkCore;

namespace BulkyWeb.Data
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
        {

        }

        public DbSet<Category> Categories { get; set; }
    }
}
```

- Seeding data in DBContext (Optional)

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Category>().HasData(
        new Category { Id = 1, Name = "Action", DisplayOrder = 1 },
        new Category { Id = 2, Name = "SciFi", DisplayOrder = 2 },
        new Category { Id = 3, Name = "History", DisplayOrder = 3 }
        );
}
```

- Register DB context in Program.cs

```
// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<ApplicationDbContext>(options=>
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));
```

- In package manager console – add-migration name
- Update-database

Retrieving values from DB

```
public class CategoryController : Controller
{
    private readonly ApplicationDbContext _db;
    public CategoryController(ApplicationDbContext db)
    {
        _db= db;
    }
    public IActionResult Index()
    {
        List<Category> objCategoryList = _db.Categories.ToList();
        return View();
    }
}
```

For strongly typed view – add @model List<class> and use value by Model.property

To use c# code in view – user @

```
@model List<Category>

    <div class="container">

<table class="table table-bordered table-striped">
    <thead>
    <tr>
        <th>
            Category Name
        </th>
        <th>
            Display Order
        </th>
    </tr>
    </thead>
    <tbody>
        @foreach(var obj in Model.OrderBy(u=>u.DisplayOrder))
        {
            <tr>
                <td>@obj.Name</td>
                <td>
                    @obj.DisplayOrder
```

URL redirection

```
<div class="col-6 text-end">
    <a asp-controller="Category" asp-action="Create" class="btn btn-primary">
    <i class="bi bi-plus-circle"></i>  Create New Category
    </a>
</div>
```

Tag Helpers –

Asp-for=”Property”

Asp-validation-summary – all, none, ModelOnly

```
<form method="post">

    <div class="border p-3 mt-4">
        <div class="row pb-2">
            <h2 class="text-primary">Create Category</h2>
            <hr/>
        </div>
        <div asp-validation-summary="All"></div>
        <div class="mb-3 row p-1">
            <label asp-for="Name"  class="p-0"></label>
            <input asp-for="Name" class="form-control"/>
            <span asp-validation-for="Name" class="text-danger"></span>
        </div>
        <div class="mb-3 row p-1">
            <label asp-for="DisplayOrder" class="p-0"></label>
            <input asp-for="DisplayOrder" class="form-control" />
            <span asp-validation-for="DisplayOrder" class="text-danger"></span>
        </div>
```

Form submission -

```
public IActionResult Create()
{
    return View();
}
[HttpPost]
public IActionResult Create(Category obj)
{
    if (obj.Name == obj.DisplayOrder.ToString())
    {
        ModelState.AddModelError("name", "The DisplayOrder cannot exactly
    }
    if (ModelState.IsValid)
    {
        _db.Categories.Add(obj);
        _db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View();
```

Client-side validation –

Add below in view-

```
@section Scripts{
    @{
    <partial name="_ValidationScriptsPartial"/>
    }
```

Edit – update operation

For passing ID – asp-route-ID

Remember to hide ID in view if other that ID is used for ID name so it wont give 0 as id to post. .update will create a new record if id is 0

```
<td>
    <div class="w-75 btn-group" role="group">
        <a asp-controller="Category" asp-action="Edit" asp-route-categoryId="@obj.Id"
            <i class="bi bi-pencil-square"></i> Edit
        </a>
            <a asp-controller="Category" asp-action="Delete" class="btn btn-danger mx-
                <i class="bi bi-trash-fill"></i> Delete
            </a>
```

```
public IActionResult Edit(int? id)
{
    if(id==null || id == 0)
    {
        return NotFound();
    }
    Category categoryFromDb = _db.Categories.Find(id);
    if (categoryFromDb == null)
    {
        return NotFound();
    }
    return View(categoryFromDb);
}
```

Find only works on ID

```
[HttpPost]
public IActionResult Edit(Category obj)
{
    if (ModelState.IsValid)
    {
        _db.Categories.Update(obj);
        _db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View();

}
```

Delete –

We cannot use 2 methods with same name and sign to use ActionName in Post

```
public IActionResult Delete(int? id)
{
    if (id == null || id == 0)
    {
        return NotFound();
    }
    Category? categoryFromDb = _db.Categories.Find(id);
    //Category? categoryFromDb1 = _db.Categories.FirstOrDefault(u=>u.Id==id);
    //Category? categoryFromDb2 = _db.Categories.Where(u=>u.Id==id).FirstOrDefault();

    if (categoryFromDb == null)
    {
        return NotFound();
    }
    return View(categoryFromDb);
}
[HttpPost, ActionName("Delete")]
public IActionResult DeletePOST(int? id)
{
    if (ModelState.IsValid)
    {
```

```
[HttpPost, ActionName("Delete")]
public IActionResult DeletePOST(int? id)
{
    Category? obj = _db.Categories.Find(id);
    if (obj == null)
    {
        return NotFound();
    }
    _db.Categories.Remove(obj);
    _db.SaveChanges();
    return RedirectToAction("Index");
}
```

Disable all fields

```
<div class="mb-3 row p-1">
    <label asp-for="DisplayOrder" class="p-0"></label>
    <input asp-for="DisplayOrder" disabled class="form-control" />

</div>
```

TempData used for displaying notification, available for one request

```
TempData["success"] = "Category created successfully";
```

```
@if (TempData["success"] != null)
{
    <h2>@TempData["successs"]</h2>
}
```

Use razor pages, when ther is no controllers/model/view. It has only pages

N-tier Atchitecture –

Has separate class library for Data Access, Models, Utility and use it in MVC Web app

Add reference to web project for all 3 class library and add packages at library level

DI Service Lifetimes –

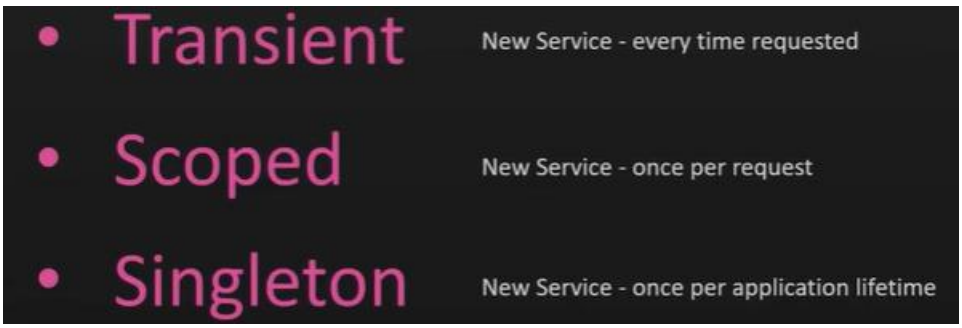
Transient –

- Simple and safest
- Whenever we want an implementation create an object and provide
- Everytime a service is requested, it creates an object

Scoped (recommended)–

- Depend on HTTP request
- Object created per request, it will be used within the HTTP request
- For the next request new implementation will be created

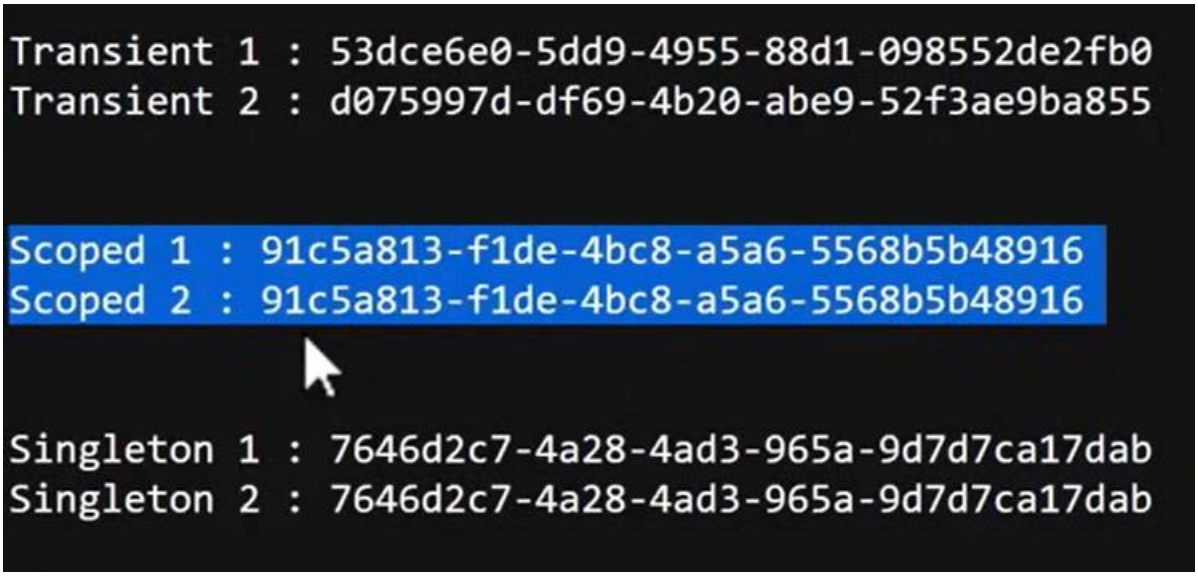
Singleton –



Registering -

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddSingleton<ISingletonGuidService,SingletonGuidService>();
builder.Services.AddTransient<ITransientGuidService, TransientGuidService>();
builder.Services.AddScoped<IScopedGuidService, ScopedGuidService>();
```



Repository Pattern -

Separation of access to database

Acts as intermediary between the domain model layers and data mapping

Create an Interface

Create an implemntaion of the interface

Inject the respository to the service with Custructor DI

Areas –

Add new scaffolded item – area

```
namespace BulkyBookWeb.Areas.Admin.Controllers
{
    [Area("Admin")]
    public class CategoryController : Controller
    {
        private readonly IUnitOfWork _unitOfWork;
```

```
app.MapControllerRoute(
    name: "default",
    pattern: "{area=Customer}/{controller=Home}/{action=Index}/{id?}");
```

```
<li class="nav-item">
    <a class="nav-link" asp-area="Admin" asp
</li>
<li class="nav-item">
    <a class="nav-link" asp-area="Customer"
</li>
```

Foreign key –

```
public int CategoryId { get; set; }
[ForeignKey("CategoryId")]
public Category Category { get; set; }
```

```
_db.Products.Include(u => u.Category);
```

Dropdown –

```
public IActionResult Create()
{
    IEnumerable<SelectListItem> CategoryList = _unitOfWork.Category
        .GetAll().Select(u => new SelectListItem
        {
            Text = u.Name,
            Value = u.Id.ToString()
        });

    //ViewBag.CategoryList = CategoryList;
    ViewData["CategoryList"] = CategoryList;

    return View();
}
```

```
<select asp-for="CategoryId" asp-items="ViewBag.CategoryList" class="form-select
    <option disabled selected>--Select Category--</option>
</select>
```

```
<select asp-for="CategoryId" asp-items="@(ViewData["CategoryList"] as IEnumerable<SelectListItem
    <option disabled selected>--Select Category--</option>
</select>
```

VIEWBAG

- ViewBag transfers data from the Controller to View, not vice-versa. Ideal for situations in which the temporary data is not in a model.
- ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0
- Any number of properties and values can be assigned to ViewBag
- The ViewBag's life only lasts during the current http request. ViewBag values will be null if redirection occurs.
- ViewBag is actually a wrapper around ViewData.

VIEWDATA

- ViewData transfers data from the Controller to View, not vice-versa. Ideal for situations in which the temporary data is not in a model.
- ViewData is derived from ViewDataDictionary which is a dictionary type.
- ViewData value must be type cast before use.
- The ViewData's life only lasts during the current http request. ViewData values will be null if redirection occurs.

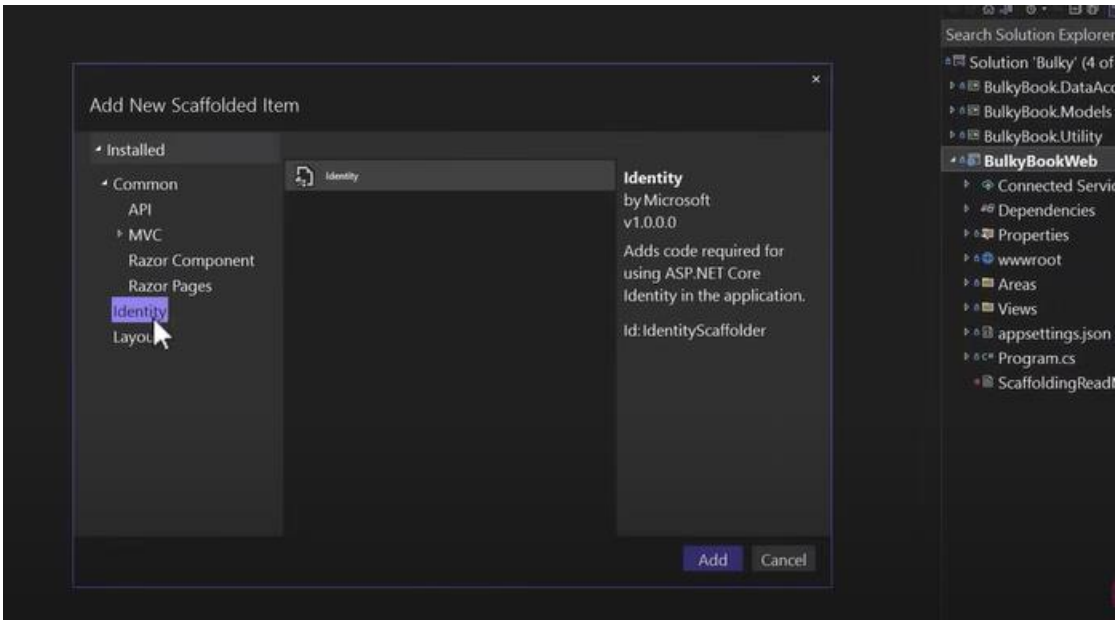
ViewBag internally inserts data into ViewData dictionary. So the key of ViewData and property of ViewBag must **NOT** match

TEMPDATA

- TempData can be used to store data between two consecutive requests.
- TempData internaly use Session to store the data. So think of it as a short lived session.
- TempData value must be type cast before use. Check for null values to avoid runtime error.
- TempData can be used to store only one time messages like error messages, validation messages.

ASP.NET Identity – (Razor Pages)

- add New scaffolded item to the web app



- add identitydbcontext instead of dbContext in deriving and add below

```
public class ApplicationDbContext : IdentityDbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
    {

    }
```

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{

    base.OnModelCreating(modelBuilder);
```

- remove the extra db context added by scaffolding in Identity folder
- add useAuthentication() and razor pages in program.cs

```
builder.Services.AddRazorPages();
builder.Services.AddScoped<IUnitOfWork, UnitOfWork>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this for
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();
app.UseAuthentication();
app.UseAuthorization();
app.MapRazorPages();
app.MapControllerRoute(
    name: "default",
    pattern: "{area=Customer}/{controller=Home}/{action=Index}/{id?}");

app.Run();
```

- add in layout page

```
<partial name="_LoginPartial" />
```

- add migration and update db

- for adding role

```
builder.Services.AddIdentity<IdentityUser,IdentityRole>().AddEntityFrameworkStores<ApplicationDbContext>();
```

```
private readonly SignInManager<IdentityUser> _signInManager;
private readonly RoleManager<IdentityRole> _roleManager;
private readonly UserManager<IdentityUser> _userManager;
private readonly IUserStore<IdentityUser> _userStore;
private readonly IUserEmailStore<IdentityUser> _emailStore;
private readonly ILogger<RegisterModel> _logger;
private readonly IEmailSender _emailSender;

public RegisterModel(
    UserManager<IdentityUser> userManager,
    RoleManager<IdentityRole> roleManager,
    IUserStore<IdentityUser> userStore,
    SignInManager<IdentityUser> signInManager,
    ILogger<RegisterModel> logger,
    IEmailSender emailSender)
{
    _roleManager=roleManager;
```

- Assigning a role –

```
if (result.Succeeded) {
    _logger.LogInformation("User created a new account with password.");

    if (!String.IsNullOrEmpty(Input.Role)) {
        await _userManager.AddToRoleAsync(user, Input.Role);
    }
    else {
        await _userManager.AddToRoleAsync(user, SD.Role_Customer);
    }
```

*add creating token

```
builder.Services.AddIdentity<IdentityUser,IdentityRole>().AddEntityFrameworkStores<ApplicationDbContext>().AddDefaultTokenProviders();
```

- To restrict access

```
[Area("Admin")]
[Authorize(Roles = SD.Role_Admin)]
public class CategoryController : Controller
{
```

- To map path (add it after adding identity)

```
builder.Services.AddIdentity<IdentityUser,IdentityRole>().AddEntityFrameworkStores<ApplicationDbContext>().AddDefaultTokenProviders();
builder.Services.ConfigureApplicationCookie(options => {
    options.LoginPath = $"/Identity/Account/Login";
    options.LogoutPath = $"/Identity/Account/Logout";
    options.AccessDeniedPath = $"/Identity/Account/AccessDenied";
});
```