Contents

[**1.** **Pakages :** 3](#_Toc110747338)

[**2.** **Configuring/Installing Identity** 3](#_Toc110747339)

[**3.** **Middlewares :** 3](#_Toc110747340)

[**4.** **General** 3](#_Toc110747341)

[**3.1 Get the list of all nuget installed packages** 3](#_Toc110747342)

[**5.** **Enable Razor Runtime Compilation** 3](#_Toc110747343)

[**6.** **Password Restrictions** 3](#_Toc110747344)

[**7.** **Connecting to DataBase** 7](#_Toc110747345)

[**8.** **Creating Data Base from Db Entities** 9](#_Toc110747346)

[**9.** **Making Relashonship between two tables.** 9](#_Toc110747347)

[**10.** **Adding Files/Images to DB** 10](#_Toc110747348)

[**11.** **Saving Data to Database** 12](#_Toc110747349)

[**1** **Via Binding parameters.** 12](#_Toc110747350)

[**2** **Mapping Data to weinto database.** 12](#_Toc110747351)

[**12.** **Getting Data from DataBase** 13](#_Toc110747352)

[**1** **Via Mapping** 13](#_Toc110747353)

[**2** **Via Linq query** 13](#_Toc110747354)

[**13.** **Getting Details of data one by one** 13](#_Toc110747355)

[**1** **We can get data from database based on id.** 13](#_Toc110747356)

[**Note: Provide Id like this** 13](#_Toc110747357)

[**14.** **Editing Data** 14](#_Toc110747358)

[**1** **Via Binding parameters** 14](#_Toc110747359)

[**15.** **Deleting Data from Database.** 14](#_Toc110747360)

[**16.** **Signing Up** 16](#_Toc110747361)

[**17.** **Customization in AspNetUsers table** 19](#_Toc110747362)

[**18.** **Editing signup criteria in Identity** 20](#_Toc110747363)

[**19.** **Signing In Process in Identity** 20](#_Toc110747364)

[**20.** **Signing Out** 21](#_Toc110747365)

[**21.** **Redirect User to login Page while using Authorization** 21](#_Toc110747366)

[**22.** **Return Url** 21](#_Toc110747367)

[**23.** **Get User ID/Authenticated User in controller** 23](#_Toc110747368)

[**24.** **Changing Password** 23](#_Toc110747369)

[**25.** **Logging In Using Third Party Tools** 25](#_Toc110747370)

[**2nd Method** 28](#_Toc110747371)

[**26.** **---------- >> UserSecret.json** 29](#_Toc110747372)

[**27.** **Block External Logins without Authentication** 29](#_Toc110747373)

[ **Allow Only Verified emails** 29](#_Toc110747374)

[**28.** **Generate Default Email Confirmation Link** 30](#_Toc110747375)

[**29.** **Send Email Using Mailkit** 33](#_Toc110747376)

[**30.** **Get User Name & Email + Other Claims** 35](#_Toc110747377)

[**31.** **Resending Email** 35](#_Toc110747378)

[**32.** **Reset/Forgot Password** 36](#_Toc110747379)

[**33.** **LockOut the User** 37](#_Toc110747380)

[**34.** **Token Life Span** 38](#_Toc110747381)

[**35.** **Edit Identity User** 38](#_Toc110747382)

[**36.** **Delete Identity User** 39](#_Toc110747383)

[**37.** **Adding Roles in asp.net core** 40](#_Toc110747384)

[**38.** **Enabling Text Compression** 43](#_Toc110747385)

[4. Add the Response Compression middleware to the request pipeline 43](#_Toc110747386)

[**39.** **Make your website live** 43](#_Toc110747387)

[**40.** **Uploading Asp.net Core Project to GitHub** 44](#_Toc110747388)

[**41.** **Preview an Image before Uploading** 44](#_Toc110747389)

[**42.** **Navigation / HighLight Menues** 44](#_Toc110747390)

[**43.** **Adding Class/Styles to html using jquery / js** 45](#_Toc110747391)

[**44.** **Add/Edit/Delete a note in a bootstrap modal** 46](#_Toc110747392)

[**45.** **Using Dependency Injection for View (Use multiple models in asp.net core)** 47](#_Toc110747393)

[**46.** **Using Partial views** 47](#_Toc110747394)

[**47.** **Add Columns to Asp.netusers Table.** 48](#_Toc110747395)

[**48.** **Showing Bootstrap Notifications** 48](#_Toc110747396)

[**49.** **Generals** 49](#_Toc110747397)

# **Pakages :**

* 1. Microsoft.aspnetcore.identity.entityframeworkcore
  2. Microsoft.aspnetcore.mvc.razor.runtimecompilation
  3. Microsoft.entityframeworkcore.sqlserver
  4. Microsoft.entityframeworkcore.Tools

# **Configuring/Installing Identity**

**Steps :**

1. Install identity.EntityFrameworkcore
2. After installing identity now Enable Authentcation.

// Enabling Authentication

app.UseAuthentication();

1. Now Configure identity.

//Configuring Identity

builder.Services.AddIdentity<IdentityUser, IdentityRole>().AddEntityFrameworkStores<DB\_Context\_Class(In our case it is ‘DB’)>();

builder.Services.AddDbContext<DBase>(ServiceLifetime.Transient);

# **Middlewares :**

UserAuthentication() -------------🡪 to enable Authentication

# **General**

## **3.1 Get the list of all nuget installed packages**

dotnet nuget locals all --list

# **Enable Razor Runtime Compilation**

services.AddRazorPages()

.AddRazorRuntimeCompilation();

# **Password Restrictions**

1. Minimum eight characters, at least one letter and one number:

"^(?=.\*[A-Za-z])(?=.\*\d)[A-Za-z\d]{8,}$"

1. Minimum eight characters, at least one letter, one number and one special character:

"^(?=.\*[A-Za-z])(?=.\*\d)(?=.\*[@$!%\*#?&])[A-Za-z\d@$!%\*#?&]{8,}$"

1. Minimum eight characters, at least one uppercase letter, one lowercase letter and one number:

"^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)[a-zA-Z\d]{8,}$"

1. Minimum eight characters, at least one uppercase letter, one lowercase letter, one number and one special character:

"^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,}$"

1. Minimum eight and maximum 10 characters, at least one uppercase letter, one lowercase letter, one number and one special character:

"^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*?&])[A-Za-z\d@$!%\*?&]{8,10}$"

**Example :**

var regexItem = new Regex("^(?=.\*[A-Za-z])(?=.\*\\d)(?=.\*[@$!%\*#?&])[A-Za-z\\d@$!%\*#?&]{4,}$");

var QUERY = regexItem.IsMatch(signUp.Password);

if (QUERY)

{

context.SignUp.Add(signUp);

context.SaveChanges();

TempData["success"] = true;

return RedirectToAction("LogIn");

}

Services :

services.AddIdentity<IdentityUser,IdentityRole>().AddEntityFrameWorkStores<Name\_of\_DB>();

In order to work with identity we need to inherit IDBContext instead of DBContext in our DB.

-) use of \_Layout,\_ViewStart,\_ViewImports files in .netcore

1 \_ViewStart files are used to write some common code in all view files.

2 Only one \_Layout file can be created so if need to make more files to share some common code then make \_ViewStart files.

3 \_ViewImports file is used to write common directives used in view files.

-) Tag helpers

1 - asp-route-name\_of\_parameter

2 - asp-controller

3 - asp-action

4 - asp-for

5 - asp-validation-for

6 - asp-validation-summary

1 <a asp-route-id="@data.Roll" asp-controller="Home" asp-action="GetStudentByRoll">Get Details</a> => id is parameter of the action method we are

passing data to which.

2 asp-all-route-data => used to pass multiple parameters

3 asp-route => if we are using routing in our action methods then pass the name of the routig in this then u don't need to pass the

action and controller names in anchor tag.

4 asp-append-version="true" => updates cached images when they are changed .

5 <environment include="Development"></environment> => this tag is used to execute code according to the development or production or other environments.

6 asp-fallback-href="" => if we are using a cdn in our application.In case cdn does not work if the server is down or due to any other reason then

file will be loaded from your provided path.

also set ----> asp-fallback-test-class="sr-only" asp-fallback-test-property="Postion" asp-fallback-test-value="absolute"

7 asp-for => model binder

8 asp-validation-for => used with span tag other wise it will not work

9 asp-validation-summary="All" => used with div tag

values ) => 1 - All

2 - ModelOnly

3 - None

DataType(DataType.Date) => used in server side validation

10 Drop down => <select asp-for="language">

<option value="">Please choose a language. </option>

<option> Hindi </option>

</select>

Note: To pass a list from controller use asp-item + selectlist + viewbag property.

<select asp-for="language" asp-item="new selectlist(ViewBag.LanguageList)">

<option value="">Please choose a language. </option>

<option> Hindi </option>

</select>

11 selectlistitem() => another way of representing data in dropdown from the controller.

group1 = new selectListGroup(){ Name="Group1" }

group2 = new selectListGroup(){ Name="Group1" , Disabled = true }

viewbag.language = new lsit<SelectListItem()>{

new selectListItem(){Text="Hindi" , Value = "1" , Group = group1 },

new selectListItem(){Text="Hindi" , Value = "1" , Group = group1 },

new selectListItem(){Text="Hindi" , Value = "1" , Group = group2 },

new selectListItem(){Text="Hindi" , Value = "1" , Group = group2 }

}

multiple(attribute of select tag) => Selecting multiple options

Make a drop down by enum

HTML.GetEnumSelectList<name\_of\_enum>() => pass this in asp-item();

Use display attribute on enum values to display text in dropdown.How ever written values in enum will be taken as values of dropdown.

<div class="mb-3 d-flex justify-content-center border">

<div class="d-lg-flex">

<label class="p-1 fw-bold">Member Type: </label>

@foreach (var item in Html.GetEnumSelectList<Models.SignUpModel.Select>())

{

<div class="form-check m-1">

<**input** class="form-check-input" **type**="radio" **asp-for**="Member\_Type" id="@item.Value" **name**="type" **value**="@item.Text" tabindex="8">

<label for="@item.Value">@item.Text</label>

</div>

}

</div>

</div>

# **Connecting to DataBase**

* + 1. **Create a new cs file**
    2. **Inherit DBContext Class**
    3. **make constructor like**

public DataBase(DbContextOptions<Data\_Base\_Name> options)

: base(options)

{}

* + 1. **add data\_Entities like you have a model**

public DBset<model\_name> Table\_Name{get;set;}

* + 1. **Now write this functions.**

protected override void onconfiguring(DBContextOptionsBuilder options)

**{**

optionsBuilder.UseSqlServer("connection string");

base.onConfiguring(optionsbuilder);

}

1. **Don't do 5th step if you want to add this line in Program.cs**

builder.Services.AddDbContext<DataBase\_Name>(

options => options.UseSqlServer(connection string)

);

I ) string connectionString = "Server=.;Database=AddNote;User Id=root;Password=;Trusted\_Connection=True;"

**Note :** On problem of adding migration

Unable to create an object of type 'ItemsDB'. For the different patterns supported at design time, see <https://go.microsoft.com/fwlink/?linkid=851728>

Use also plain constructor

public DataContext()  
    {  
    }

# **Creating Data Base from Db Entities**

**Note :** Write [key] attribute on property

e.g. public int id{get;set;} to make it primary key.

1. install Microsoft.EntityFrameWorkCore.Tools
2. get-help entityframework

All changes we do in our dbcontext we will have to add migrations for every change.

1. Add-migration provide\_name\_of\_migration

2 Update-Database

# **Making Relashonship between two tables.**

1. write the name of other table in first table with same type as the table name is in the form of property.
2. write the name of other table in ICollection<Table\_Name> in the form of property.

public Language Language{get;set;}

public ICollection<BookModel> BookModel{get;set;}

17) Custom validation attribute . Inherit ValidationAttribute into the class.

write override keyword and you get the intellisense so it will make a method .

Now you can write any logic into it.

Note: Any property made to this class will be treated as attribute of this class.

18) Custom tag helpers . Inherit TagHelper into the class.

write override keyword and you get the intellisense so it will make a Process method .

public string MyEmail{get;set;};

Logic will be like this.

i ) output.tagname="a"; => Provide name of the tag

ii) output.attributes.setattributes("href",$"mailto{MyEmail}"); =>set value of any attribute.

iii)output.attributes.add("id","email-id"); =>add any attribute of related tag.

iv) output.content.setcontent("myemail") =>will be displayed on the screen.

Note: Any property made to this class will be treated as attribute of this class.

19 ) Overriding html tags .

use [htmltargetelement]

20) Server side validations behave as client side validations automatically.

Just add client side libraries.

i ) jquery.js

ii) jquery.validate.js

ii) jquery.uobtrusive.js

To disable client side validations use

builder.Services.AddRazorPages().AddRazorRuntimeCompilation().AddViewOptions(option =>

{

option.HtmlHelperOptions.ClientValidationEnabled = false;

});

in program.cs

21 ) Create ajax form .

Note : all ajax helpers start with data-ajax.

Data-ajax-success (on completing request successful it runs the function).

# **Adding Files/Images to DB**

1. return type of file is IFormFile.

public Iformfile coverphoto{get;set;}

if you are working with files then use | enctype="multipart/form-data".

1. Use dependency injection.Like this

private readonly IWebHostEnvironment \_webHostEnvironment;

public HomeController(ILogger<HomeController> logger, IWebHostEnvironment webHostEnvironment) ==> constructor

{

\_logger = logger;

\_webHostEnvironment = webHostEnvironment;

}

private string AddFile(AddNote note)

{

string filename = null;

if (note.ProfilePic != null)

{

string folder = "Images/";

filename = (Guid.NewGuid().ToString()) + " " + note.ProfilePic.FileName;

string path = folder + filename;

string serverPath = Path.Combine(environment.WebRootPath, path);

note.ProfilePic.CopyTo(new FileStream(serverPath, FileMode.Create));

}

return filename;

}

**Note :** webhostenvironment gets the path of our server.

**using** (var fileStream = **new** FileStream(filePath, FileMode.Create))

                {

                    model.ProfileImage.CopyTo(fileStream);

                }

1. Use Iformcollection for multiple files.
2. files can be pdf/images or any other kind of files.

# **Replace/Delete old files/Images.**

private string ReplaceFile(news\_events news\_Events)

{

string folder = "Profile\_Pics/";

if(news\_Events.Event\_Image\_Name != null)

{

var oldImage = news\_Events.Event\_Image\_Name;

string oldPath = folder + oldImage;

var Old\_serverPath = Path.Combine(WebHostEnvironment.WebRootPath, oldPath);

System.IO.File.Delete(Old\_serverPath);

}

var formFile = news\_Events.FormFile;

string filename = null;

if (formFile != null)

{

filename = (Guid.NewGuid().ToString()) + " " + formFile.FileName;

string path = folder + filename;

string serverPath = Path.Combine(WebHostEnvironment.WebRootPath, path);

formFile.CopyTo(new FileStream(serverPath, FileMode.Create));

}

return filename;

}

# **Saving Data to Database**

1. **Via Binding parameters.**

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([Bind("NoteTitle,NoteDescription")] AddNote addNote)

{

if (ModelState.IsValid)

{

\_context.Add(addNote);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(addNote);

}

1. **Mapping Data to save into database.**

private int AddNoteToDb(AddNote notes)

{

string UniqueFileName = AddFile(notes);

var note1 = new AddNote()

{

NoteTitle = notes.NoteTitle,

NoteDescription = notes.NoteDescription,

FileName = UniqueFileName

};

context.Notes.Add(note1);

context.SaveChanges();

return notes.ID;

}

Action Method to get data from form

[HttpPost]

public IActionResult AddNote(AddNote note)

{

if (ModelState.IsValid)

{

int result = AddNoteToDb(note);

if (result > 0)

{

ModelState.Clear();

ModelState.AddModelError("", "Data Inserted Successfully");

}

}

return View();

}

# **Getting Data from DataBase**

1. **Via Mapping**

private List<AddNote> Get\_All\_Notes()

{

var note = new List<AddNote>();

var data = context.Notes.ToList();

if (data?.Any() == true)

{

foreach (var item in data)

{

note.Add(new AddNote()

{

ID = item.ID,

NoteTitle = item.NoteTitle,

NoteDescription = item.NoteDescription,

FileName = item.FileName

});

}

}

return note;

}

1. **Via Linq query**

public IActionResult List(int? id)

{

var note = context.Notes.ToList();

return View(note);

}

# **Getting Details of data one by one**

**1 We can get data from database based on id.**

public IActionResult Edit(int? id)

{

var delNote = context.Notes.Find(id);

return View(delNote);

}

**Note: Provide Id like this**

<**a** **asp-action**="Edit" **asp-route-id**="@item.ID">Edit</**a**> |

@Html.ActionLink("Edit", "Edit","Actions", new { id=item.ID })

1. **Via Mapping**

# **Editing Data**

1. **Via Binding parameters**

[HttpPost]

public IActionResult Edit(int id, [Bind("ID,NoteTitle,NoteDescription,ProfilePic,FileName")] AddNote notes)

{

if(id != notes.ID)

{

return NotFound();

}

if (ModelState.IsValid)

{

string uname = AddFile(notes);

notes.FileName = uname;

context.Update(notes);

// context.Entry(notes).State = Microsoft.EntityFrameworkCore.EntityState.Modified;

context.SaveChanges();

}

return View();

}

**Note:** It is a function to get the name of file/Image. And also to save the Image in server folder.

# **Deleting Data from Database.**

[HttpPost]

public IActionResult Delete(int id)

{

var del = context.Notes.Find(id);

context.Notes.Remove(del);

context.SaveChanges();

return redirectToAction(“Index”);

}

In index.cshtml =>

<**a** class="btn btn-danger m-1" **asp-action**="Delete" **asp-controller**="Home" **asp-route-Id**="@note.Id">Delete</**a**>

How to use ajax action link.

@Ajax.ActionLink("Dashboard", "Dashboard", "Home", new { area = "" }, new AjaxOptions { }, new { @class = "link-light btn-primary p-2" })

23 ) Partial views ==> use model="model\_name" tag helper to override the model

<partial name="name\_of\_partial" model="new bookmodel()"/>

foreach( var book in List<Bookmodel>)

then model will be book inside this block.

24) View Components ==> very similar to partial views. View Component does not use model binding.View Component is not the part of http life cycle.Faster than Partial Views.

View components can be used in Dynamic navigation menue(based of role).

Get some related data for a page.eg. related posts,related page & related books.

Note: View components have two files. 1 server side c# file . 2nd .cshtml file.

How to render view components.

i - @await Components.InvokeAsync("Component\_name")

ii- <vc:component\_name></vc:component\_name>

How to make a View component.

i - Inherit Viewcomponent in a new c# file

ii- use IViewComponentResult as return type

File Locations?

c# file can be anywhere.

but .cshtml file needs to be in the shared folder . Make a new folder with name Component then make a new folder inside this with the name of component in the

shared folder.

name of view will be default.cshtml

25) To enable routing use these two

i - userouting()

ii - useendpoints(endpoints=>

{

endpoints.MapDefaultControllerRoute(); ==>>to use attribute routing

endpints.MapControllerRoute({

name:"aboutus",

pattern:"AboutUs"

defaults:{controller=Home}/{action=Index}{id?});

});

26) Constraints => used with routing to make validations on routing input.

route("bookstore/{id:int:min(1){name:alpah:minlength(5)}) by alpha you can put only aphabets.

27) Interfaces

i) Create Interface of DB by clicking on it . then quick actions then extract interface.

ii)Register that interface in the program.cs as a service.

builder.Services.AddScoped<IDB, DB>();

use IDB instead of DB in other files making instance of this method/repository.

28) IConfiguration . => used to read values from appsettings.json file.

.GetValue(),.GetSection(),.Bind() ==>> Used with Iconfiguration instance to read values from appsettings.json file. Bind() method binds the object of json file

to the property of the any other class.

# **Signing Up**

To Work with the security of We have to install Identity framework

Microsoft.aspnetcore.identity.entityframeworkcore

A Universal framework to provide security to any dotnet application.

**Benefits :**

1. User Validation
2. Password Validation
3. Password Hashing
4. Multifactor Authentication
5. LockOut(Block User after n wrong attempts)
6. External Identity(Facebook,Google,Microsoft,Twitter etc)
7. Common framework for all dotnet applications
8. All required tables(aspnetusers,aspnetlogins etc)
9. And Much More

**Note:** We will Insert data into AspNetUsers table of new signups/new Users.

Two very important methods in identity are.

1. User Manger
2. Singin Manager

**Note :** To work with signing in process we need UserManager.

**Steps :**

1. Install identity.EntityFrameworkcore
2. After installing identity now Enable Authentcation.

// Enabling Authentication

app.UseAuthentication();

1. Now Configure identity.

//Configuring Identity

builder.Services.AddIdentity<IdentityUser, IdentityRole>().AddEntityFrameworkStores<DB\_Context\_Class(In our case it is ‘DB’)>();

builder.Services.AddDbContext<DBase>(ServiceLifetime.Transient);

1. In order to work with identity we need to inherit IdentityDBContext instead of DBContext

public class DB : IdentityDbContext

{

public DB(DbContextOptions<DB> options) : base(options)

{

}

public DbSet<AddNote> Notes { get; set; }

}

1. In order to add all tables add a migration and update database.
2. **To add new users**

I ) use usermanager

1. **Create a repository and inject the usermanager**

public class AccountRepository : IAccountRepository

{

private readonly UserManager<IdentityUser> userManager;

public AccountRepository(UserManager<IdentityUser> \_userManager)

{

userManager = \_userManager;

}

public async Task<IdentityResult> CreateUserAsync(SignUp signUp)

{

var user = new IdentityUser()

{

UserName = signUp.Name,

Email = signUp.Email,

};

var result = await userManager.CreateAsync(user, signUp.Password);

return result;

}

}

1. **Now register the interface in the services.**

builder.Services.AddScoped<IAccountRepository, AccountRepository>();

1. **Now inject this interface into your controller and use this.**

private readonly IAccountRepository accountRepository;

public Account(IAccountRepository \_accountRepository)

{

accountRepository = \_accountRepository;

}

[HttpPost]

public async Task<IActionResult> SignUp(SignUp signUp)

{

if (ModelState.IsValid)

{

var result = await accountRepository.CreateUserAsync(signUp);

if (!result.Succeeded)

{

foreach (var errmessage in result.Errors)

{

ModelState.AddModelError("", errmessage.Description);

}

Return View();

}

ModelState.Clear();

}

return View();

}

**Finally:** New user is created in aspnetusers table created by Identity.

# **Customization in AspNetUsers table/Add New Cols to Identity Table**

* Adding some more columns in AspNetUsers class.
* As this class is provided by identity ,so we cannot make any changes to it.
* So to add new columns to this table/class first we will inherit this class in other class .
* Then we will write our new columns in that class then instead of using IdentityUser in the whole application we will use that class name.
* We just need to inherit identityuser to a new cs file and add properties and tell the dbcontext class that we are using custom class

public class DBase:IdentityDbContext<ApplicationUser>

public class ApplicationUser:IdentityUser

{

public string FirstName { get; set; }

public string SecondName { get; set; }

}

In Program.cs file

builder.Services.AddIdentity<ApplicationUser, IdentityRole>().AddEntityFrameworkStores<DB>();

public class AccountRepository : IAccountRepository

{

private readonly UserManager<ApplicationUser> userManager;

public AccountRepository(UserManager<ApplicationUser> \_userManager)

{

userManager = \_userManager;

}

public async Task<IdentityResult> CreateUserAsync(SignUp signUp)

{

var user = new ApplicationUser()

{

UserName = signUp.Name,

Email = signUp.Email,

};

var result = await userManager.CreateAsync(user, signUp.Password);

return result;

}

}

Now you have to add migration,update database and you are done.

# **Editing signup criteria in Identity**

Add to Services

builder.Services.Configure<IdentityOptions>(options =>

{

options.Password.RequireDigit = true;

});

**Note :** Configure method provides all customizations of Identity.

# **Signing In Process in Identity**

**Note :** As to work with sign up process in identity we have usermanager class in the same way we have signinmanger class.

private readonly UserManager<ApplicationUser> userManager;

private readonly SignInManager<ApplicationUser> signInManager;

public AccountRepository(UserManager<ApplicationUser> \_userManager, SignInManager<ApplicationUser> \_signInManager)

{

userManager = \_userManager;

signInManager = \_signInManager;

}

public async Task<SignInResult> SignInAsync(SignIn sign)

{

var result = await signInManager.PasswordSignInAsync(sign.Email, sign.Password, sign.Remember\_me, false);

return result;

}

Call this method from the controller

[HttpPost]

public async Task<IActionResult> SignIn(SignIn signIn)

{

if (ModelState.IsValid)

sg {

var result = await accountRepository.SignInAsync(signIn);

if (result.Succeeded)

{

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError("", "Invalid Credentials");

}

return View(signIn);

}

# **Signing Out**

public async Task SignoutAsync()

{

await signIn.SignOutAsync();

}

public async Task<IActionResult> SignOut()

{

await accountRepository.SignoutAsync();

return RedirectToAction("Index", "Home");

}

# **Redirect User to login Page while using Authorization**

builder.Services.ConfigureApplicationCookie(config =>

{

config.LoginPath = "/Account/LogIn";

});

Note : You must use name ‘LogIn’ instead of ‘SignIn’ otherwise you will receive error.

# **Return Url**

* After clicking the authorized page we are redirected to the login page .
* Use localRedirect either use URL.IslocalUrl(returnUrl) to avoid vulnerable redirection.
* Then we login and after login we are redirected to the index page.
* We should be redirected to authorized page after the login where we have clicked.
* For this purpose we use return url.

1. **Set return url property first in login.cshtml form.**

<**form** **asp-action**="LogIn" **asp-controller**="Account" **asp-route-returnurl**="@Context.Request.Query["ReturnUrl"]">

1. **Return url for authorized pages**

public async Task<IActionResult> SignIn(SignIn sign,string returnurl)

{

if (ModelState.IsValid)

{

var result = await accountRepository.SignInAsync(sign);

if (result.Succeeded)

{

if(!string.IsNullOrEmpty(returnurl))

{

return LocalRedirect(returnurl);

}

return RedirectToAction("Index", "Home");

}

ModelState.AddModelError("", "Invalid Credentials");

}

return View(sign);

}

**Claims in Asp.Net**

**Use of Claims**

To get the aspnetusers table value in razor page we need to claim the values of the table with some keys written in the claims.Other wise we will not be able to get the table values in razor page.

Note : we need new cs file for claims . Make a new .cs file .

This is the way you can make custom claims

public class UserClaims : UserClaimsPrincipalFactory<ApplicationUser, IdentityRole>

{

public UserClaims(UserManager<ApplicationUser> userManager, RoleManager<IdentityRole> roleManager, IOptions<IdentityOptions> options) : base(userManager, roleManager, options)

{

}

protected override async Task<ClaimsIdentity> GenerateClaimsAsync(ApplicationUser user)

{

var identity = await base.GenerateClaimsAsync(user);

identity.AddClaim(new Claim("FName", user.FirstName ?? “”));

identity.AddClaim(new Claim("LName", user.SecondName));

return identity;

}

}

Now we need to tell our application that we are using custom version of claims class.

builder.Services.AddScoped <IUserClaimsPrincipalFactory <ApplicationUser>, UserClaims>();

In this ApplicationUser is our user. because

ApplicationUser : IdentityUser

**The way to get the value of claims for our use from identity tables.**

**Using Claims in our views**

Hello @(User.FindFirst("FName").Value + " " + User.FindFirst("LName").Value)

FName & LName are the keys written in the claims to fetch the value from database.

# **Get User ID/Authenticated User in controller**

1. Inject the httpservice for this purpose

public class GetUserId : IGetUserId

{

private readonly IHttpContextAccessor httpContext;

public GetUserId(IHttpContextAccessor \_httpContext)

{

httpContext = \_httpContext;

}

public string GetId()

{

return httpContext.HttpContext.User?.FindFirstValue(ClaimTypes.NameIdentifier);

}

1. User app.authentication() to get authenticated user.

public bool IsAuthenticated()

{

return httpContext.HttpContext.User.Identity.IsAuthenticated;

}

}

1. Register the service in program.cs

builder.Services.AddScoped<IGetUserId, GetUserId>();

1. Inject the service and use it where you want

public HomeController(ILogger<HomeController> logger, IWebHostEnvironment webHostEnvironment, IGetUserId \_getUser)

{

\_logger = logger;

\_webHostEnvironment = webHostEnvironment;

getUser = \_getUser;

}

public async Task<IActionResult> Index()

{

var isauthenicated = getUser.IsAuthenticated;

var id = getUser.GetId();

var data = await GetEmployees();

return View(data);

}

# **Changing Password**

[HttpPost]

public async Task<IActionResult> ChangePassword(ChangePassword changePassword)

{

if (ModelState.IsValid)

{

//Getting user by UserManager

var user = await userManager.GetUserAsync(User);

//Getting User by id

/\*

var userID = signUp\_Functions.GetID();

var user = await userManager.FindByIdAsync(userID); \*/

var result = await userManager.ChangePasswordAsync(user, changePassword.Current\_Password, changePassword.New\_Password);

if (!result.Succeeded)

{

foreach (var err in result.Errors)

{

ModelState.AddModelError("", err.Description);

}

}

else

{

await signUp\_Functions.SignOut();

return RedirectToAction("LogIn", "Account");

}

}

# **Logging In Using Third Party Tools**

1. Install Microsoft.aspnetcore.authentication.google
2. In the LogIn model class

public string ReturnUrl { get; set; }

public IList<AuthenticationScheme> ExternalLogIns { get; set; }

1. In Controller Class

public async Task<IActionResult> LogIn(string returnUrl)

{

SignIn model = new SignIn

{

ReturnUrl = returnUrl,

ExternalLogIns = (await signInManager.GetExternalAuthenticationSchemesAsync()).ToList()

};

return View(model);

}

Getting all external login facilities we have.

1. In the Program.cs

builder.Services.AddAuthentication()

.AddGoogle(options =>

{

options.ClientId = "413683638595-8mcnojhk8efdmgsl07gg5efdimku83ig.apps.googleusercontent.com";

options.ClientSecret = "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";

});

1. In the View we have

<div class="col-md-4">

<h4>External LogIns</h4>

@if (Model.ExternalLogIns.Count == 0)

{

<h6>No External LogIn Providers available</h6>

}

else

{

<**form** method="post" **asp-action**="ExternalLogin" **asp-controller**="Account" **asp-route-returnUrl**="@Model.ReturnUrl">

<div>

@foreach (var provider in Model.ExternalLogIns)

{

<input type="submit" name="provider" title="Login using your @provider.DisplayName account" value="@provider.Name" class="btn btn-outline-primary" />

}

</div>

</**form**>

}

</div>

1. **And our controller has ExternalLogin action like :**

[HttpPost]

public IActionResult Externallogin(string provider, string returnUrl)

{

var redirectUrl = Url.Action("ExternalLoginCallback", "Account",

new { ReturnUrl = returnUrl });

var properties = signInManager.ConfigureExternalAuthenticationProperties(provider, redirectUrl);

return new ChallengeResult(provider, properties);

}

public async Task<IActionResult> ExternalLoginCallback(string returnUrl = null, string remoteError = null)

{

returnUrl = returnUrl ?? Url.Content("~/");

SignIn model = new SignIn

{

ReturnUrl = returnUrl,

ExternalLogIns = (await signInManager.GetExternalAuthenticationSchemesAsync()).ToList()

};

if (remoteError != null)

{

ModelState.AddModelError(string.Empty, $"Error from External Provider : {remoteError}");

return View("Login", model);

}

var info = await signInManager.GetExternalLoginInfoAsync();

if (info == null)

{

ModelState.AddModelError(String.Empty, "Error loading external login information.");

return View("Login", model);

}

var signinResult = await signInManager.ExternalLoginSignInAsync(info.LoginProvider, info.ProviderKey, isPersistent: false, bypassTwoFactor: true);

if (signinResult.Succeeded)

{

return LocalRedirect(returnUrl);

}

else

{

var email = info.Principal.FindFirstValue(ClaimTypes.Email);

if (email != null)

{

var user = await userManager.FindByEmailAsync(email);

if (user == null)

{

user = new IdentityUser

{

UserName = info.Principal.FindFirstValue(ClaimTypes.Email),

Email = info.Principal.FindFirstValue(ClaimTypes.Email)

};

await userManager.CreateAsync(user);

}

await userManager.AddLoginAsync(user, info);

await signInManager.SignInAsync(user, isPersistent: false);

return LocalRedirect(returnUrl);

}

ViewBag.ErrorTitle = $"Email claim not Recieved from:{info.LoginProvider}";

ViewBag.ErrorMessage = $"Please Contact support on Pragim@pragim.com";

return View("Error");

}

## **2nd Method**

[HttpPost]

public IActionResult ExternalLogins(string returnUrl = null)

{

// Request a redirect to the external login provider.

string provider = "Google";

var redirectUrl = Url.Action(nameof(ExternalLoginCallback), "Account", new { returnUrl });

var properties = signInManager.ConfigureExternalAuthenticationProperties(provider, redirectUrl);

return Challenge(properties, provider);

}

[TempData]

public string ErrorMessage { get; set; }

[HttpGet]

[AllowAnonymous]

public async Task<IActionResult> ExternalLoginCallback(string returnUrl = null, string remoteError = null)

{

if (remoteError != null)

{

ErrorMessage = $"Error from external provider: {remoteError}";

return RedirectToAction(nameof(ExternalLogins));

}

var info = await signInManager.GetExternalLoginInfoAsync();

if (info == null)

{

return RedirectToAction(nameof(ExternalLogins));

}

// Sign in the user with this external login provider if the user already has a login.

var result = await signInManager.ExternalLoginSignInAsync(info.LoginProvider, info.ProviderKey, isPersistent: false, bypassTwoFactor: true);

if (result.Succeeded)

{

return RedirectToAction("Index", "Home");

//\_logger.LogInformation("User logged in with {Name} provider.", info.LoginProvider);

//return RedirectToAction(nameof(returnUrl));

}

else

{

string email = string.Empty;

string firstName = string.Empty;

string lastName = string.Empty;

string profileImage = string.Empty;

//get google login user infromation like that.

if (info.Principal.HasClaim(c => c.Type == ClaimTypes.Email))

{

email = info.Principal.FindFirstValue(ClaimTypes.Email);

}

if (info.Principal.HasClaim(c => c.Type == ClaimTypes.GivenName))

{

firstName = info.Principal.FindFirstValue(ClaimTypes.GivenName);

}

if (info.Principal.HasClaim(c => c.Type == ClaimTypes.GivenName))

{

lastName = info.Principal.FindFirstValue(ClaimTypes.Surname);

}

if (info.Principal.HasClaim(c => c.Type == "picture"))

{

profileImage = info.Principal.FindFirstValue("picture");

}

var user = new IdentityUser { UserName = email, Email = email, EmailConfirmed = true };

var result2 = await userManager.CreateAsync(user);

if (result2.Succeeded)

{

result2 = await userManager.AddLoginAsync(user, info);

if (result2.Succeeded)

{

//do somethng here

}

}

return View("Login");

}

}

# **---------- >> UserSecret.json**

Cut the important information like database connections string from appsetting.json file and paste it in this file .

# **Block External Logins without Authentication**

**1st Method**

builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>

{

options.SignIn.RequireConfirmedEmail = true;

}).AddEntityFrameworkStores<DB>();

**2nd Method**

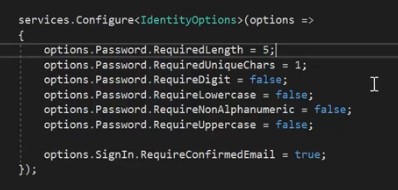
builder.Services.Configure<IdentityOptions>(options =>

{

options.SignIn.RequireConfirmedEmail = true;

});

* **Allow Only Verified emails**



**3rd Method**

var user = await userManager.FindByNameAsync(signIn.Name);

if (user != null && !user.EmailConfirmed)

{

ModelState.AddModelError("", "Please confirm your email");

return View("SignIn", signin);

}

# **Generate Default Email Confirmation Link**

We need to send the whole link to our email so

1. **In program.cs enable token providers**

builder.Services.AddIdentity<IdentityUser, IdentityRole>()

.AddEntityFrameworkStores<DBase>()

.AddDefaultTokenProviders();

1. **Call below method like this and provide user to it.**

var user = await userManager.FindByNameAsync(signUp.Name);

if (user != null)

GeneratingEmailLinkAndSendingEmail(user);

1. **Generating Full Link**

public async Task GeneratingEmailLinkAndSendingEmail(IdentityUser user)

{

var token = HttpUtility.UrlEncode(await userManager.GenerateEmailConfirmationTokenAsync(user));

var userID = user.Id;

var domain = "https://localhost:7179";

var confirmEmail = $"Account/ConfirmEmail?userID={userID}&token={token}";

string fulLink = Path.Combine(domain, confirmEmail);

CLink = fulLink;

if (!string.IsNullOrEmpty(CLink))

{

//var confirmationLink = Url.Action("ConfirmEmail", "Account",

// new { userID = user.Id, token = token });

await sEmail(user, token);

}

}

**Note :** ConfirmEmail action method.

public async Task<IActionResult> confirmEmail(string userID, string token)

{

token = token.Replace(' ', '+');

if (!string.IsNullOrEmpty(userID) && !string.IsNullOrEmpty(token))

{

var result = await userManager.ConfirmEmailAsync(await userManager.FindByIdAsync(userID), token);

if (result.Succeeded)

{

ViewBag.success = true;

}

}

Console.WriteLine("emaail confirmed succeessfully.");

return RedirectToAction("Index", "Home");

}

1. **sEmail method is like this**

public string CLink { get; set; }

public async Task sEmail(IdentityUser user, string token)

{

//string fileData = System.IO.File.ReadAllText("F:\\ASP.net core learning\\Website[27 - Jan 2022]\\Website[27 - Jan 2022]\\Templates\\template.cshtml");

MimeMessage message = new MimeMessage();

message.From.Add(new MailboxAddress("Email Confirmation Providers", "ahmadfrazahm5@gmail.com"));

message.To.Add(new MailboxAddress(user.UserName, user.Email));

message.Subject = "Please confirm your email by clicking on the confirmation link below.";

message.Body = new TextPart("html")

{

//Text = fileData

Text = CLink

};

try

{

using (var client = new SmtpClient())

{

await client.ConnectAsync("smtp.gmail.com", 465, true);

//// Note: only needed if the SMTP server requires authentication

await client.AuthenticateAsync("ahmadfrazahm5@gmail.com", "03066854864AS@DF");

await client.SendAsync(message);

await client.DisconnectAsync(true);

}

}

catch(Exception ex)

{

ModelState.AddModelError("", "Basically You don't have any internet.\nException Details : " + ex.Message);

}

}//end of method

# **Send Email Using Mailkit**

using System;

using MailKit.Net.Smtp;

using MailKit;

using MimeKit;

var file = File.ReadAllText("F:\\ASP.net core learning\\Send\_Email\_Using\_Mailkit\\Send\_Email\_Using\_Mailkit\\template.html");

var message = new MimeMessage();

message.From.Add(new MailboxAddress("Sending 2nd bar", "joey@friends.com"));

message.To.Add(new MailboxAddress("To 2nd Account", "ahmadfrazahm@gmail.com"));

message.Subject = "This is test mail";

message.Body = new TextPart("html")

{

Text = file

};

try

{

using (var client = new SmtpClient())

{

client.Connect("smtp.gmail.com", 465, true);

// Note: only needed if the SMTP server requires authentication

client.Authenticate("ahmadfrazahm5@gmail.com", "03066854864AS@DF");

client.Send(message);

client.Disconnect(true);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

* **Sending Email Dynamically**

1. **Model Class**

Also create a model class having properties like , Name,Email of sender , subject and description of problem.

1. **In SendEmail class we have**

using MailKit.Net.Smtp;

using MailKit;

using MimeKit;

namespace Send\_Email\_\_2\_14\_2022\_.Models

{

public class SendEmail

{

public string To { get; set; }

public string Name { get; set; }

public string Subject { get; set; }

public string Description { get; set; }

string fileData = File.ReadAllText("F:\\ASP.net core learning\\Send Email [2-14-2022]\\Send Email [2-14-2022]\\Email Template\\template.cshtml");

public async Task sEmail()

{

MimeMessage message = new MimeMessage();

message.From.Add(new MailboxAddress("Sending Email", "ahmadfrazahm5@gmial.com"));

message.To.Add(new MailboxAddress(To, "ahmadfrazahm@gmail.com"));

message.Subject = "This is test mail";

message.Body = new TextPart("html")

{

Text = fileData

};

using (var client = new SmtpClient())

{

await client.ConnectAsync("smtp.gmail.com", 465, true);

// Note: only needed if the SMTP server requires authentication

await client.AuthenticateAsync("ahmadfrazahm5@gmail.com", "03066854864AS@DF");

await client.SendAsync(message);

await client.DisconnectAsync(true);

}

}//end of method

}

}

1. **In Controller class we have**

[HttpPost]

public IActionResult ContactUs(EmailModel model)

{

SendEmail email = new SendEmail()

{

To = model.To

};

if (ModelState.IsValid)

{

ViewBag.To = model.To;

ViewBag.From = model.From ;

ViewBag.Body = model.Body ;

}

email.sEmail();

return View();

}

# **Get User Name & Email + Other Claims**

var useremail = User.FindFirst(ClaimTypes.Email).Value;

var userName = User.Identity.Name;

**Note :** User ClaimTypes property to get current user email and phone no and other things.

# **Resending Email**

1. **In controller we have**

From the signup controller we return values of email and username that is created that are needed to get the user either by using email or by using his name to get the id of the same user to generate the full confirmation link.

Like this

return RedirectToAction(nameof(LogIn), new { email = signUp.Email, name=signUp.Name });

1. **And from the login controller we receive those values by parameter binding.**

public IActionResult LogIn(string name, string email)

{

return View();

}

Because we have values of email and name sended by the login controller that are gained by parameter binding we create a form with hidden fields of name and email so their value can be sended to the ‘resendEmail’ controller.

1. **In view we have**

@if (TempData["EmailResended"] != null)

{

<div class="alert alert-success" role="alert">

<**form** **asp-action**="ResendEmail" **asp-controller**="Account">

<**input** **type** ="hidden" **asp-for**="email"/>

<**input** **type** = "hidden" **asp-for**= "Name" />

<h4 class="alert-heading">Email resended successfully.</h4>

<hr>

<p class="mb-0">We have sent you an email for confirmation.Please confirm your email Before Loging In.</p>

<input type = "submit" value = "Resend Email" />

</**form**>

</div>

}

Resend Email Method that is called from the View to resend confirmation link.We get the value of the email and Name by parameter binding from the view.

1. **In controller we have.**

[HttpPost]

public async Task<IActionResult> Resendemail(string email, string Name)

{

var user = await GetUserByNameAsync(Name);

if (user != null)

{

if (!user.EmailConfirmed)

{

await signUp\_Functions.sendingEmail(user);

TempData["EmailResended"] = "true";

}

else

{

ViewBag.EmailIsConfirmed = true;

TempData["AlreadyConfirmed"] = "true";

}

}

return RedirectToAction(nameof(LogIn), new { email = email, name = Name });

}

private async Task<IdentityUser> GetUserByNameAsync(string name)

{

return await userManager.FindByNameAsync(name);

}

# **Reset/Forgot Password**

1. **First we make a model with name & email properties to take them as input.**

//Reset Password

public IActionResult ResetPassword()

{

return View();

}

1. **Call the reset password method.**

[HttpPost]

public async Task<IActionResult> ResetPassword(ResetPassword resetPassword)

{

await ResetPassword(resetPassword.Name, resetPassword.Email);

return View();

}

1. **Generate a resetPassword token like this as well as we need full Link.**

public async Task sendingResetPasswordMail(IdentityUser user)

{

var token = HttpUtility.UrlEncode(await userManager.GeneratePasswordResetTokenAsync(user));

var userID = user.Id;

var domain = "https://localhost:7179";

var confirmEmail = $"Account/ResetPasswordModel?userID={userID}&token={token}";

string fulLink = Path.Combine(domain, confirmEmail);

CLink = fulLink;

if (!string.IsNullOrEmpty(CLink))

{

//var confirmationLink = Url.Action("ConfirmEmail", "Account",

// new { userID = user.Id, token = token });

await sEmail(user, token);

}

}

1. **Now Make a method to reset the password .**

private async Task ResetPassword(string name, string email)

{

var user = await GetUserByNameAsync(name);

if (user != null)

{

if (user.Email == email)

{

await signUp\_Functions.sendingResetPasswordMail(user);

TempData["resetpaswwordEmailSent"] = "Successfull";

}

else

{

TempData["EmailisnotValid"] = "error";

}

}

}

# **LockOut the User**

1. **Enable lockingout in signInResult**

public async Task<Microsoft.AspNetCore.Identity.SignInResult> SignInResult(LogIn logIn)

{

var result = await signInManager.PasswordSignInAsync(logIn.Name, logIn.Password, true, true);

return result;

}

1. **Now Configure the lockout attempts + Time according to your self.**

builder.Services.Configure<IdentityOptions>(options =>

{

options.SignIn.RequireConfirmedEmail = true;

options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromSeconds(30);

options.Lockout.MaxFailedAccessAttempts = 3;

});

# **Token Life Span**

builder.Services.Configure<DataProtectionTokenProviderOptions>(options =>

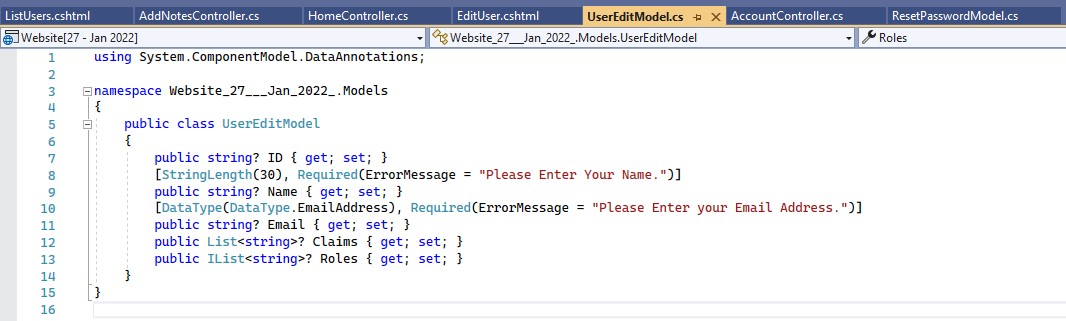
{

options.TokenLifespan = TimeSpan.FromMinutes(5);

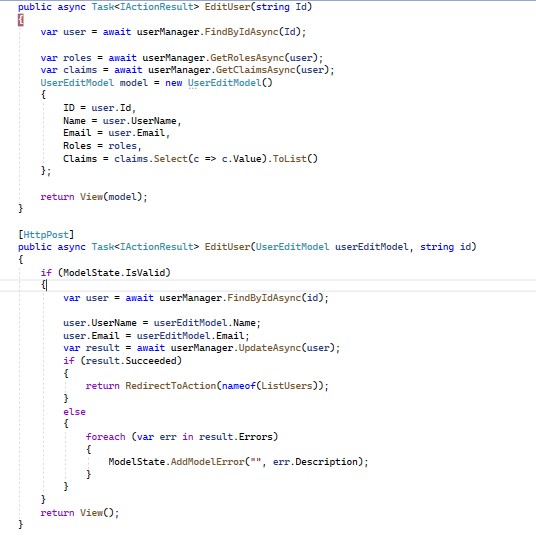
});

# **Edit Identity User**

1. **First of All make a User Edit Model class.Also Make a view of this class**

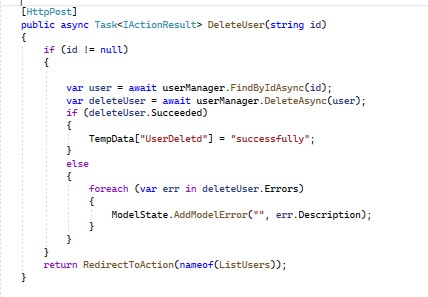


1. **In Controller we can update Identity User like this.**



# **Delete Identity User**

1. **In Controller class we have**



1. **Use this return type Model in view to delete identity user**

@model IEnumerable<Microsoft.AspNetCore.Identity.IdentityUser>

# **Adding Roles in asp.net core**

1. **Get List of roles**

public IActionResult Roles()

{

var roles = roleManager.Roles.ToList();

return View(roles);

}

1. **Create Any role you want Role**

[HttpPost]

public async Task<IActionResult> CreateRole(CreateRole createRole)

{

if (ModelState.IsValid)

{

var UserRole = new IdentityRole();

UserRole.Name = createRole.RoleName;

var result = await roleManager.CreateAsync(UserRole);

if (result.Succeeded)

{

return RedirectToAction(nameof(Roles));

ModelState.AddModelError("", "Successfully added a role.");

}

else

{

foreach (var err in result.Errors)

{

ModelState.AddModelError("", err.Description);

}

}

}

return View();

}

1. **Get List of users & also made their id and name and check if that user is in role then checkbox is true otherwise false.**

[HttpGet]

public async Task<IActionResult> AddUsers(string Role\_id)

{

ViewBag.roleId = Role\_id;

var role = await roleManager.FindByIdAsync(Role\_id);

if (role == null)

{

ModelState.AddModelError("", $"Role\_id is not valid {Role\_id}");

}

var model = new List<AddUsers>();

foreach (var user in userManager.Users)

{

var AddUsers = new AddUsers()

{

ID = user.Id,

UserName = user.UserName

};

if (await userManager.IsInRoleAsync(user, role.Name))

{

AddUsers.isMember = true;

}

else

{

AddUsers.isMember = false;

}

model.Add(AddUsers);

}

return View(model);

}

1. **Get all users that have been added to role and check if** 
   1. Checkbox is not selected then user is removed if he is in role .
   2. Checkbox is selected then user is add to role if he is not added to role already.
   3. Else continue the execution for further users.

[HttpPost]

public async Task<IActionResult> AddUsers(List<AddUsers> users, string Role\_id)

{

var role = await roleManager.FindByIdAsync(Role\_id);

if (role == null)

{

ModelState.AddModelError("", $"Role with id = {Role\_id} cannot be found.");

}

if (ModelState.IsValid)

{

for (int i = 0; i < users.Count; i++)

{

var user = await userManager.FindByIdAsync(users[i].ID);

if (users[i].isMember && !(await userManager.IsInRoleAsync(user, role.Name)))

{

var result = await userManager.AddToRoleAsync(user, role.Name);

if (result.Succeeded)

{

ModelState.AddModelError("", "User has been added successfully to role.");

return RedirectToAction(nameof(Roles));

}

}

else if (!users[i].isMember && await userManager.IsInRoleAsync(user, role.Name))

{

var result = await userManager.RemoveFromRoleAsync(user, role.Name);

if (result.Succeeded)

{

ModelState.AddModelError("", "User has been removed successfully from role.");

return RedirectToAction(nameof(Roles));

}

}

else

{

continue;

}

}

}

return View();

}

1. **AddUsers.cshtml file is like this.**

@model List<AddUsers>

@{

ViewData["Title"] = "Index";

var id = ViewBag.roleId;

}

<**form** **asp-action**="AddUsers" **asp-controller** = "Home" **asp-route-Role\_id** =@id method="post">

<div class="card" style="width: 18rem;">

<ul class="list-group list-group-flush">

<li class="list-group-item"> <h1>List of Users</h1> </li>

<li class="list-group-item">

@if (Model.Any())

{

for (int i = 0; i < Model.Count; i++)

{

<**input** **asp-for**="@Model[i].isMember" class="form-check-input"/>

<**label** **asp-for**="@Model[i].isMember"> @Model[i].UserName</**label**>

<**input** **type**="hidden" **asp-for**="@Model[i].ID"/>

<**input** **type**="hidden" **asp-for**="@Model[i].UserName"/>

<br />

}

}

else

{

<h6>No roles yet.</h6>

}

</li>

<li class="list-group-item">

<button class = "btn-primary btn">Save</button>

</li>

</ul>

</div>

</**form**>

# **Enabling Text Compression**

1 install  Microsoft.AspNetCore.ResponseCompression package.

in program.cs

1. services.Configure<GzipCompressionProviderOptions>
2. (options => options.Level = CompressionLevel.Optimal);
3. services.AddResponseCompression(options =>
4. {
5. options.Providers.Add<GzipCompressionProvider>();
6. });
7. services.Configure<GzipCompressionProviderOptions>
8. (options => options.Level = CompressionLevel.Optimal);
9. services.AddResponseCompression(options =>
10. {
11. options.Providers.Add<GzipCompressionProvider>();
12. });

## 4. Add the Response Compression middleware to the request pipeline

**app.UseResponseCompression();**

# **Make your website live**

Go to heruko.com and connect your github repository in which files are present .

Now provide a buildpack link of asp.net core.

For buildpack of asp.net core go to jincode github repo and copy the link of that repo and provide it in heruko and your website is live.

# **Uploading Asp.net Core Project to GitHub**

1. **General Information to add to the .gitignore file for a asp.net core project to upload it to the github.**

**File :** [.gitignore.txt](file:///C:\Users\Faraz\AppData\Roaming\Microsoft\Word\.gitignore.txt)

# **Preview an Image before Uploading**

View side is like this =>

<div class="text-center m-auto">

<img id="img" style="height:20rem; text-align:center;" alt="No Image Available" />

</div>

Script is like this =>

<script type="text/javascript">

function ShowPreview(input) {

if (input.files && input.files[0]) {

var ImageDir = new FileReader();

ImageDir.onload = function(e) {

$('#img').attr('src', e.target.result);

}

ImageDir.readAsDataURL(input.files[0]);

}

}

</script>

Server side is like =>

<div class="form-group">

<**label** **asp-for**="Picture"></**label**> <br />

<**input** **asp-for**="Picture" class="form-control" onchange="ShowPreview(this)"/>

</div>

# **Navigation / HighLight Menues**

* 1. **In asp.net core MVC**

class = @(ViewContext.RouteData.Values["Action"].toString()=="Index(Action Name)"?"btn-primay":"")

Example :

<ul class="nav">

<li class="@(ViewContext.RouteData.Values["Action"].ToString()== "SignUp" ? "btn-primary" : "")">

@Ajax.ActionLink("SignUp", "SignUp", "DB", new { area = "" }, new AjaxOptions { }, new { @class = "ActionHover nav-link text-light d-flex" })

</li>

<li class="@(ViewContext.RouteData.Values["Action"].ToString() == "LogIn" ? "btn-primary" : "")">

@Ajax.ActionLink("LogIn", "LogIn", "DB", new { area = "" }, new AjaxOptions { }, new { @class = "nav-link text-light d-flex ActionHover" })

</li>

</ul>

* 1. **In asp.net Razor Pages**

register the **IHttpContextAccessor** with the correct lifetime (singleton):

services.AddHttpContextAccessor();

Now we should be able to access current **HttpContext** inside our views:

@inject Microsoft.AspNetCore.Http.IHttpContextAccessor Accessor

<a class="navbar-item @(Accessor.HttpContext.Request.Path.Value == "/" ? "nav-item-selected" : "" )" href="/">Home</a>

# **Adding Class/Styles to html using jquery / js**

$(function() {

$('.menuItem').hover( function(){

$(this).css('background-color', '#F00');

},

function(){

$(this).css('background-color', '#000');

});

});

$('#mydiv').addClass('newclass');

1. **Changing class**

document.getElementById('mydiv').className = 'newclass';

1. **Adding new Class**

document.getElementById('mydiv').className += ' newClass';

1. **Setting Styles directly**

document.getElementById('divErrorBlock').setAttribute("class", "alert alert-warning");

1. **Toggling Class**

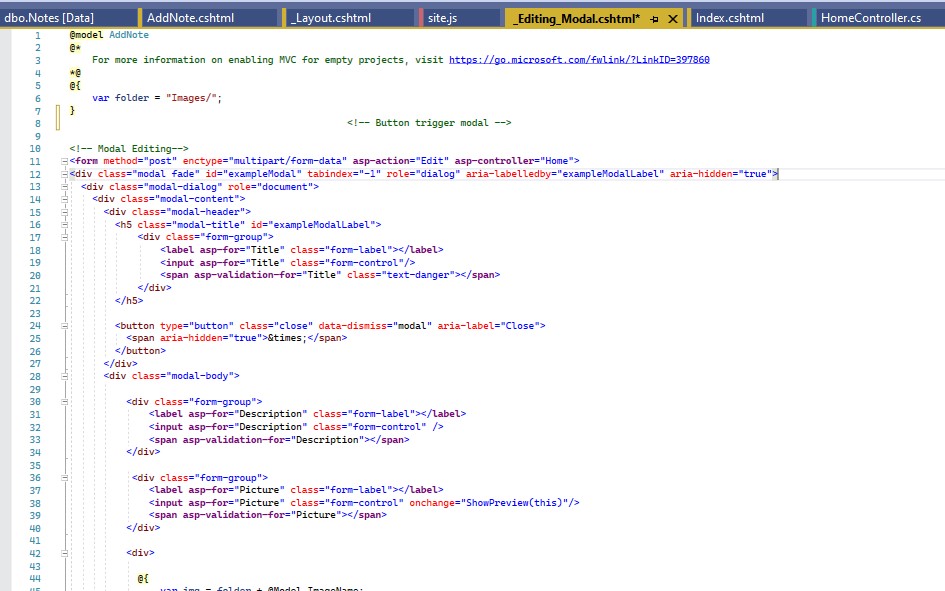
<script>

$('.Hover').hover(function(){$(this).toggleClass('shadow');});

</script>

# **Add/Edit/Delete a note in a bootstrap modal**

* First of all create a partial view and paste the bootstrap modal here just like below in picture.



* Write this line in the page in which you want to popup modal .

<div id="placeHoderHere"></div>

* Action method is like this.

public IActionResult EditEvent(int? id)

{

var \_event = DBase.News\_Events.Where(x=> x.id == id).FirstOrDefault();

return PartialView("\_Edit",\_event);

}



* Add the button in the page in which you want to popup modal (by clicking which modal should be shown).

<!-- Button trigger modal -->

<button class="card-link btn btn-outline-info my-2" type="button" data-toggle="ajax-modal" data-target="#\_EditEvent" data-url="@Url.Action($"EditEvent/{id}","Dashboard")">Edit</button>

* Write following code in site.js file

// Write your JavaScript code.

$(function () {

var placeholder = $('#placeHoderHere');

$('button[data-toggle="ajax-modal"]').click(function (event) {

var url = $(this).data('url');

var decodedUrl = decodeURIComponent(url);

$.get(decodedUrl).done(function (data) {

placeholder.html(data);

placeholder.find('.modal').modal('show');

})

})

placeholder.on('click', '[data-save="modal"]', function (event) {

event.preventDefault();

var form = $(this).parents('.modal').find('form');

var actionUrl = form.attr('action');

var sendData = form.serialize();

$.post(actionUrl, sendData).done(function (data) {

placeholder.find('.modal').modal('hide');

})

})

});

# **Using Dependency Injection for View (Use multiple models in asp.net core)**

1. Register your model in program.cs file

builder.Services.AddScoped<Crud\_Operations\_Thu\_\_6\_16\_22\_\_.Models.emailModel>();

1. Use Inject keyword in view.

@inject Crud\_Operations\_Thu\_\_6\_16\_22\_\_.Models.emailModel sendEmail

# **Using Partial views**

1. Use partial tag helper

<**partial** **name**="\_SendEmail" **model**="new emailModel()"/>

1. Use html helper

@await Html.PartialAsync("\_SendEmail",new emailModel());

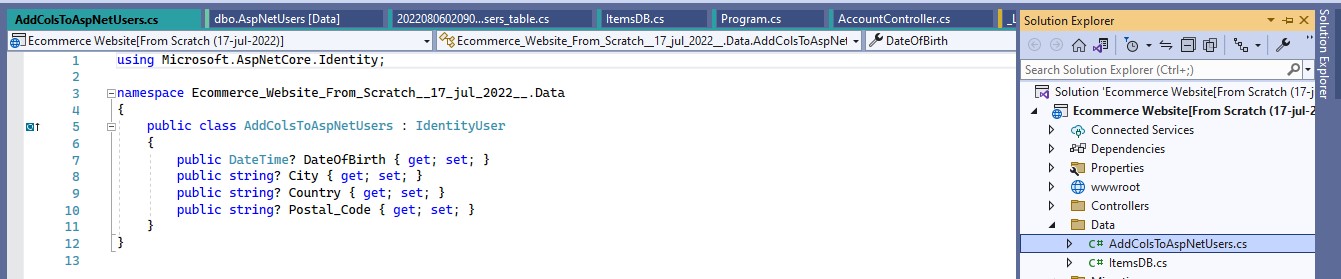
@Html.Partial("\_SendEmail",new emailModel());

@{ Html.RenderPartial("\_SendEmail",new emailModel());}

@{awiat Html.RenderPartialAsync("\_SendEmail",new emailModel());}

# **Add Columns to Asp.netusers Table.**

1. Make a new class and inherit identityuser class in it.
2. The properties you define are added to the table as columns.



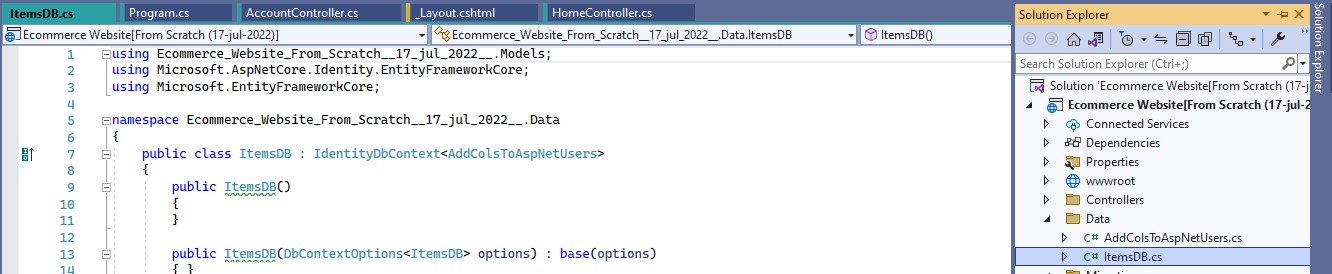
1. Now change IdentityUser to className(e.g. AddColsToAspNetUsers) in all files.
2. Also change it in the program.cs file.Just like below.

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

builder.Services.AddIdentity<AddColsToAspNetUsers,IdentityRole>().AddEntityFrameworkStores<ItemsDB>();

1. Be sure to add it in the database file where IdentityDbContext is inherited as well.Just like below.

IdentityDbContext< AddColsToAspNetUsers>



1. Now add a new migration and update dabase.
2. New columns will be added in the asp.netusers table.

# **Showing Bootstrap Notifications**

TempData["success"] = "data inserted";

@if(TempData["success"] != null)

{

<div class="alert alert-success" role="alert">

User Account has been created successfully.

</div>

}

# **Generals**

1. Injecting signinmanager/usermanager service

@using Microsoft.AspNetCore.Identity

@inject SignInManager<IdentityUser> signInManager

# **Adding multiple constructors in razor pages**

* It can cause problem some time
* Add a parameterless constructor
* Use the following attribute on the second constructor

[ActivatorUtilitiesConstructor]