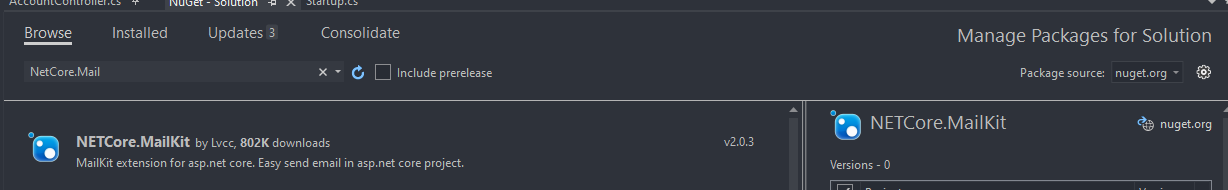
**Section03 Email Verification**

**Notes:-**

**1-we have to download PaperCut**

[PaperCut NG - Download | PaperCut](https://www.papercut.com/products/ng/download/)

**2-install the NetCore.MailKit nugget package as below**



**3-we have to know if we enableRequireConfirmEmail , this means that when you register user you cannot login until the user is confirmed**

**services.AddIdentity<IdentityUser, IdentityRole>(config => {**

**config.Password.RequiredLength = 4;**

**config.Password.RequireDigit = false;**

**config.Password.RequireNonAlphanumeric = false;**

**config.Password.RequireUppercase = false;**

**//it will prevent register the user until confirm email**

**config.SignIn.RequireConfirmedEmail = true;})**

**4-on the startup.csthml we set the following code as below**

**using IdentityExample.Data;**

**using Microsoft.AspNetCore.Builder;**

**using Microsoft.AspNetCore.Hosting;**

**using Microsoft.AspNetCore.Identity;**

**using Microsoft.EntityFrameworkCore;**

**using Microsoft.Extensions.DependencyInjection;**

**using Microsoft.Extensions.Hosting;**

**using System;**

**using NETCore.MailKit.Extensions;**

**using NETCore.MailKit.Infrastructure.Internal;**

**using Microsoft.Extensions.Configuration;**

**namespace IdentityExample{**

**public class Startup{**

**private IConfiguration \_config;**

**public Startup(IConfiguration config){\_config = config;}**

**// This method gets called by the runtime. Use this method to add services to the container.**

**// For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940**

**public void ConfigureServices(IServiceCollection services){**

**//Entity Framework Part**

**//to inject AppDbContext on the whole application to initialize database on the memory**

**services.AddDbContext<AppDbContext>(config =>{**

**config.UseInMemoryDatabase("Memory");});**

**//Identity bridge Part(used to generate repositories that is collection of abstraction methods)**

**//AddDefaultTokenProviders: used to generate Token provider to generate Tokens for reset passwords**

**//AddEntityFrameworkStores: used to link the Identity layer with database to communicate**

**//AddIdentity : registers the services**

**services.AddIdentity<IdentityUser, IdentityRole>(config => {**

**config.Password.RequiredLength = 4;**

**config.Password.RequireDigit = false;**

**config.Password.RequireNonAlphanumeric = false;**

**config.Password.RequireUppercase = false;**

**//it will prevent register the user until confirm email**

**config.SignIn.RequireConfirmedEmail = true;}).AddEntityFrameworkStores<AppDbContext>()**

**.AddDefaultTokenProviders();**

**services.ConfigureApplicationCookie(options =>{**

**options.AccessDeniedPath = "/Home/AccessDenied";**

**options.Cookie.Name = "IdentityCookie";**

**options.Cookie.HttpOnly = true;**

**options.ExpireTimeSpan = new TimeSpan(0, 15, 0);**

**options.LoginPath = "/Account/Login";**

**options.ReturnUrlParameter = "RedirectUrl";**

**options.LogoutPath = "/Account/Logout";});**

**//we add the MailKit middleware and inject the Email section from appsetttings.json**

**services.AddMailKit(config => {**

**var options = new MailKitOptions();**

**config.UseMailKit(\_config.GetSection("Email").Get<MailKitOptions>());});**

**//to apply controllers with views**

**services.AddControllersWithViews();}**

**// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.**

**public void Configure(IApplicationBuilder app, IWebHostEnvironment env){**

**if (env.IsDevelopment()){app.UseDeveloperExceptionPage();}**

**else{**

**app.UseExceptionHandler("/Home/Error");**

**// The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.**

**app.UseHsts();}**

**app.UseHttpsRedirection();**

**app.UseStaticFiles();**

**//we apply the routing middleware to activate routing to whicle endpoint we want**

**app.UseRouting();**

**//means how are you?**

**app.UseAuthentication();**

**//we have to make sure that the authorization middleware must set after the routing middleware**

**//we have also to inject the authenticaiton cookie to allow pass the autorization middleware**

**//means : are you allowed?**

**app.UseAuthorization();**

**app.UseEndpoints(endpoints =>{**

**endpoints.MapDefaultControllerRoute();});}}}**

**2-in the appSettings.json we set the following json**

**{"Email": {**

**"Server": "127.0.0.1",**

**"Port": 25,**

**"SenderName": "Anton",**

**"SenderEmail": "m93enbah@hotmail.com"},**

**"Logging": {**

**"LogLevel": {**

**"Default": "Information",**

**"Microsoft": "Warning",**

**"Microsoft.Hosting.Lifetime": "Information"}},**

**"AllowedHosts": "\*"}**

**3-in the AccountController will create two action EmailVerification after register to tell user that he have to verify email after register and VerifyEmail when the user click on the email received and he will see that the email has been confirmed**

**public async Task<IActionResult> Register(UserRegistrationDto request){**

**if (ModelState.IsValid){**

**var userCheck = await userManager.FindByEmailAsync(request.Email);**

**if (userCheck == null){**

**var user = new IdentityUser{**

**UserName = request.Email,**

**NormalizedUserName = request.Email,**

**Email = request.Email,**

**PhoneNumber = request.PhoneNumber,**

**EmailConfirmed = false,**

**PhoneNumberConfirmed = false,};**

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

**if (result.Succeeded){**

**//generation of the email token**

**var code = await userManager.GenerateEmailConfirmationTokenAsync(user);**

**//prepare link to be send body of message through MailKit**

**var link = Url.Action(nameof(VerifyEmail), "Account", new { userId = user.Id, code },Request.Scheme,Request.Host.ToString());**

**await emailManager.SendAsync("mohammedenbah93@gmail.com", "email verify",$"<a href=\"{link}\">Verfiy Email</a>",true);**

**return RedirectToAction("EmailVerification");}**

**else{**

**if (result.Errors.Count() > 0){**

**foreach (var error in result.Errors){ModelState.AddModelError("message", error.Description);}}**

**return View(request);}}**

**else{ModelState.AddModelError("message", "Email already exists.");**

**return View(request);}}**

**return View(request);}**

**public async Task<IActionResult> VerifyEmail(string userId, string code){**

**var user = await userManager.FindByIdAsync(userId);**

**if (user is null){return BadRequest();}**

**var result = await userManager.ConfirmEmailAsync(user, code);**

**if (result.Succeeded){return View();}**

**return BadRequest();}**

**public IActionResult EmailVerification(){return View();}**

**public async Task<IActionResult> VerifyEmail(string userId, string code){**

**var user = await userManager.FindByIdAsync(userId);**

**if (user is null){return BadRequest();}**

**var result = await userManager.ConfirmEmailAsync(user, code);**

**if (result.Succeeded){return View();}**

**return BadRequest();}**

**public IActionResult EmailVerification(){return View();}**

**4-on EmailVerification.,cshtml view we set the following code**

**<h2>Email has been sent</h2>**

**<h3>Please verify it.</h3>**

**5-on VerfiyEmail.cshtml view we set the following code**

**<h2>Thank you for verifying your email please login here</h2>**

**<a href="/Account/Login">Go to Login</a>**

**So the total operation as below**

**1-when the user register it will create user on the Identiy.User and then it will generate code and prepare URL that inject inside email by using MailKit**

**2-then the user see new view called EmailVerfication that you have to check his email to verify email to allow login on the website\**

**3-when he click on the hyperlink it will hit to the VerifyEmail Action that it check for the user and the code is valid or not and if true it will change the EmailConfirmed to true then it redirect to Verify Email view that tell him to go to login page**