

**In MVC Application**

1. Set the Lockout Options in the Configure Services Method while configuring IdentityFramework Middleware

options.Lockout.MaxFailedAccessAttempts = 5;

options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(15);

1. Enable Account Lockout functionality in application code at Login()

var result = await signInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, lockoutOnFailure: true);

**ADD THE FOLLOWING CHECK IN THE SAME METHOD**

if(result.IsLockedOut)

{return View("AccountLocked");}

1. In ASPNETUSERS table the three fields LockoutEnd (DateTime), LockoutEnabled (true/false), AccessFailedAccount (integer) are related to the account lockout.
2. Add AccountLocked.cshtml view in the Account Controller

<h3 class="text-danger">

Your account is locked, please try again after sometime or you may

<**a** **asp-action**="ForgotPassword" **asp-controller**="Account">

reset your password by clicking here

</**a**>

</h3>

1. Set Lockout End Date on PasswordReset() in Account Controller

if (await userManager.IsLockedOutAsync(user))

{

await userManager.SetLockoutEndDateAsync(user, DateTimeOffset.UtcNow);

}

In Razor Page Application

1. Set the Lockout Options in the Configure Services Method while configuring IdentityFramework Middleware

No need to change the Middleware Configuration, if the Lockout functionality is revoked as it is already initialized with default values.

options.Lockout.MaxFailedAccessAttempts = 5;

options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(15);

Method One

public static void Add\_Identity\_Config(this IServiceCollection services)

{

// Add Services & Configure their Settings needed by Identity Middleware

services.AddIdentity<ApplicationUser, ApplicationRole>(options =>

{

options.Password.RequiredLength = 6;

options.Password.RequiredUniqueChars = 1;

options.Password.RequireNonAlphanumeric = true;

options.Password.RequireLowercase = true;

options.Password.RequireUppercase = true;

options.Password.RequireDigit = true;

options.Lockout = new LockoutOptions()

{

AllowedForNewUsers = true,

DefaultLockoutTimeSpan = TimeSpan.FromMinutes(15), //new TimeSpan(0, 5, 0),

MaxFailedAccessAttempts = 5

};

})

// Adds a default, self-contained UI for Identity to the application using Razor

// Pages in an area named Identity.

.AddDefaultUI(UIFramework.Bootstrap4)

// Adds an Entity Framework implementation of identity information stores.

// User EF + this UOW to persist the Identity Info

.AddEntityFrameworkStores<ApplicationDbContext>()

// Different Token Providers are used during Login, Register,

// Email Confirmation, Mob. Ph Confirmation & 2FA,

// following statement add Default Token Providers for above.

// We can customize the Token Providers and configure here.

.AddDefaultTokenProviders()

// Add FirstName & Last Name to the User Claims Collection

// Adds an UserClaimsPrincipalFactory for the UserType.

.AddClaimsPrincipalFactory<ApplicationUserClaimsPrincipalFactory>();

}

Method 2

services.AddIdentity<ApplicationUser, IdentityRole>(options =>

{

options.Password.RequiredLength = 10;

options.Password.RequiredUniqueChars = 3;

options.SignIn.RequireConfirmedEmail = true;

options.Tokens.EmailConfirmationTokenProvider = "CustomEmailConfirmation";

options.Lockout.MaxFailedAccessAttempts = 5;

options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(15);

})

.AddEntityFrameworkStores<AppDbContext>()

.AddDefaultTokenProviders()

.AddTokenProvider<CustomEmailConfirmationTokenProvider

<ApplicationUser>>("CustomEmailConfirmation");

services.Configure<DataProtectionTokenProviderOptions>(o =>

o.TokenLifespan = TimeSpan.FromHours(5));

services.Configure<CustomEmailConfirmationTokenProviderOptions>(o =>

o.TokenLifespan = TimeSpan.FromDays(3));

1. Enable Account Lockout functionality in application code at Login Page Page in Account Folder

public async Task<IActionResult> OnPostAsync(string returnUrl = null)

{

ExternalLogins = (await SignInManager.GetExternalAuthenticationSchemesAsync()).ToList();

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

if (ModelState.IsValid)

{

var user = await UserManager.FindByEmailAsync(Input.Email);

// Email confirmation check added here. Now also add the logic to

// the database seed logic that add default user to the DB

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

(await UserManager.CheckPasswordAsync(user, Input.Password)))

{

// Generate Email Confirmation Token for the User.

// This code is send to the User' Email for confirmation

var emailConfirmationToken = await UserManager.GenerateEmailConfirmationTokenAsync(user);

// UserId & EmailConfirmationToken is included in the Email Confirmation Link

var callbackUrl = Url.Page(

"/Account/ConfirmEmail",

pageHandler: null,

values: new { userId = user.Id, code = emailConfirmationToken },

protocol: Request.Scheme);

EmailConfirmationToken = callbackUrl;

// Send Email to the users email.

await EmailSender.SendEmailAsync(Input.Email, "Confirm your email",

$"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

return RedirectToPage("./ConfirmEmailInfoAfterLogin");

//ModelState.AddModelError(string.Empty, "Email not confirmed yet");

//return Page();

}

Change the following if you need to reverse the Account Lock functionality

This flag was done true by me, reverse to false if Lockout is disabled

var result = await SignInManager.PasswordSignInAsync(

Input.Email,

Input.Password,

Input.RememberMe,

lockoutOnFailure: true);

if (result.Succeeded)

{

if (!string.IsNullOrEmpty(returnUrl) && Url.IsLocalUrl(returnUrl))

{

Logger.LogInformation("User logged in.");

return LocalRedirect(returnUrl);

}

else

{

// If URL is blank/null or URL not local URL, send user to homepage

return LocalRedirect(Url.Content("~/"));

//return RedirectToAction("index", "home");

}

}

if (result.RequiresTwoFactor)

{

// In case of 2FA, when user provides valid UserName & Password

// he is directed here. in case of wrong username and password

// reult is invalid login.

// WE ARE NOT SENDING ANY CALL/CODE TO THE AUTHENTICATOR APPLICATION.

// HOW THE APPLICATION WILL KNOW THAT USER IS GOING TO LOGIN AND I NEED

// TO PROVIDE A CODE THE USER.

return RedirectToPage("./LoginWith2fa", new { ReturnUrl = returnUrl, RememberMe = Input.RememberMe });

}

Already added in Sample Add, no need to change if Lockout disabled

if (result.IsLockedOut)

{

Logger.LogWarning("User account locked out.");

return RedirectToPage("./Lockout");

}

else

{

ModelState.AddModelError(string.Empty, "Invalid login attempt.");

return Page();

}

}

// If we got this far, something failed, redisplay form

return Page();

}

}

ExternalLogin Page in OnPost() Already added in Sample App, no need to change if Lockout disabled

LoginWith2FA Page in OnPost() Already added in Sample App, no need to change if Lockout disabled

LoginWithRecoverCodes in OnPost() Already added in Sample App, no need to change if Lockout disabled

if (result.IsLockedOut)

{

// it happens when user is already inserted in aspnetusers, aspnetuserlogins table

// but account is locked for sometime.

return RedirectToPage("./Lockout");

}

1. In ASPNETUSERS table the three fields LockoutEnd (DateTime), LockoutEnabled (true/false), AccessFailedAccount (integer) are related to the account lockout.
2. Add Lockout.cshtml view in the Account Folder

Already added in sample app, link to forget password was done by me

@page

@model LockoutModel

@{

ViewData["Title"] = "Locked out";

}

<header>

<h1 class="text-danger">@ViewData["Title"]</h1>

<p class="text-danger">

Your account has been locked out, please try again after sometime, or you may

<**a** **asp-page**="./ForgotPassword">

reset your password by clicking here

</**a**>

</p>

</header>

1. Set Lockout End Date on ResetPassword.cshtml in Account Folder

Comment or delete the following if you need to reverse the Account Lock functionality

if (result.Succeeded)

{

// When password reset successfully, then remove lockout if it is there.

// I have linked the Lockout & Reset Password. If following check is not there

// User still have to wait for LockOutTime expire, even when the Password Reset

if (await \_userManager.IsLockedOutAsync(user))

{

await \_userManager.SetLockoutEndDateAsync(user, DateTimeOffset.UtcNow);

}

// When System State in DB changed, then log that info

// ????????????????????????

// Password reset successsfully

return RedirectToPage("./ResetPasswordConfirmation");

}

Prevent attackers from brute force attempt to break password

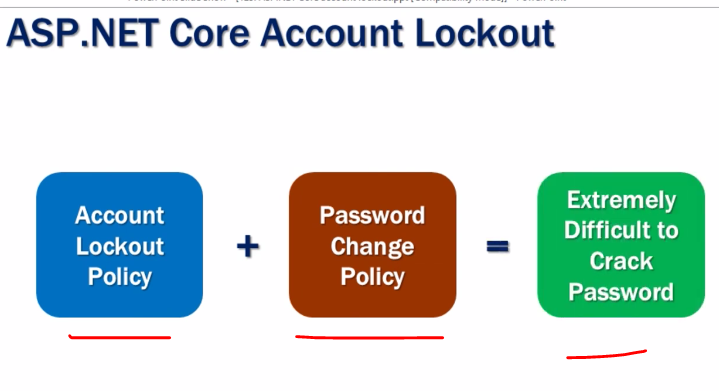
Setting how may failed attempts are allowed, configurable in asp.net mvc

Setting for how much time the account is blocked, configurable in asp.net mvc

Locking account for specified amount of time.

Password change policy is also a related policy.

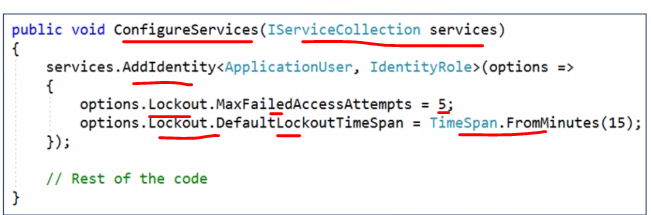
Change password after every month or two



Startup class is like the old Web.config file provides place to add the Middleware and Configure the Services needed for the middleware.

In the startup class we do following things

1. Add the Nuget Package of Middleware
2. Add the middleware to the RPP
3. Add the DEFAULT SERVICES needed for the middleware
4. Add the CONFIGURED/CUSTOMIZED SERVICES needed for the middleware.

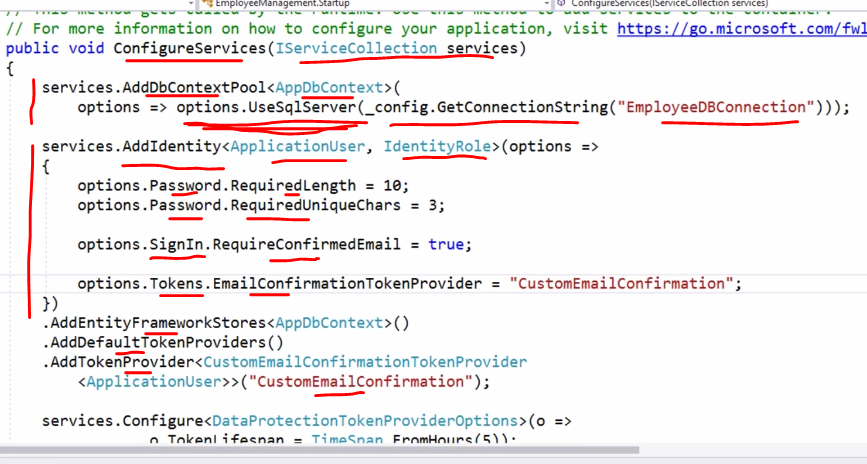


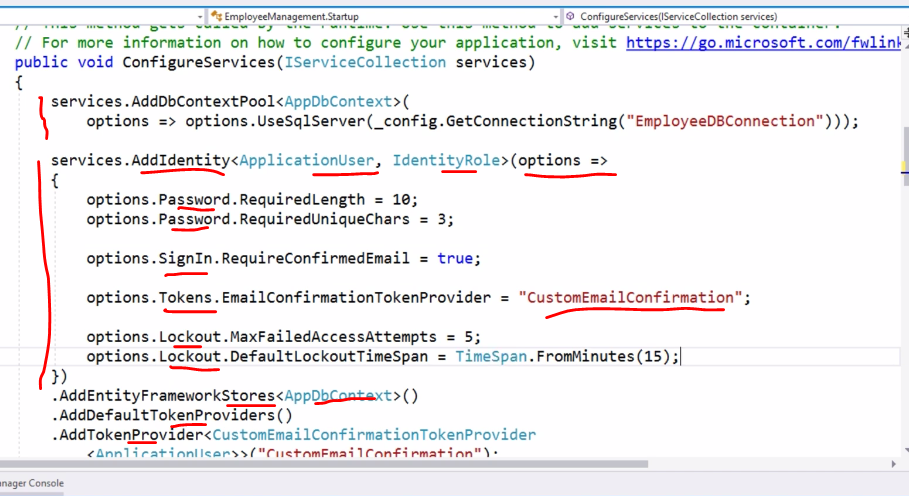
Failed attempt and Time for which the account is locked.

Default attempts is 5 times and Default Time is 5 minutes

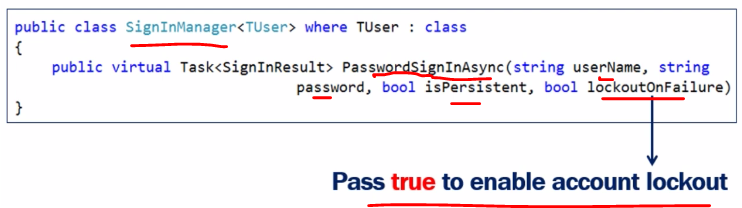
Configuring the Middleware with following aspects

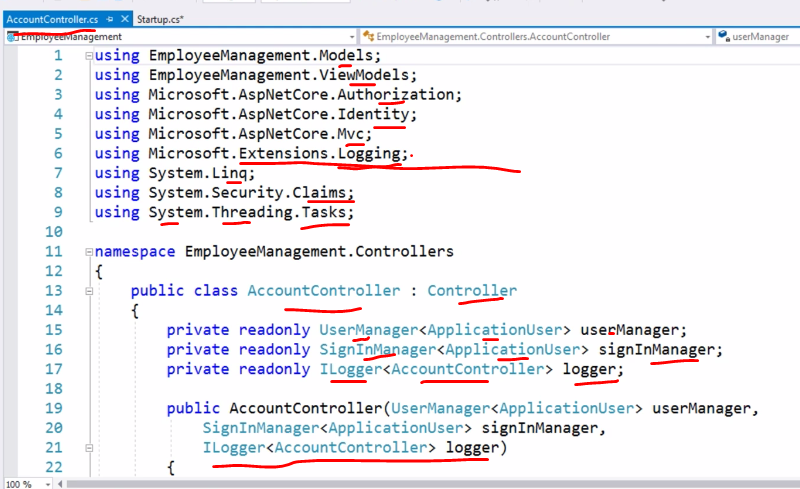
1. Providing the Generic Parameters of the ApplicationUser and Identity role/ApplicationRole. We have to customize the Middleware by these genric parameters
2. Related Options object can also be passed to configure the Middleware. Password, SignIn and Tokens all these aspects of IdentityFramework/AcccessControl can be configured. We can also configure the OptionsObject separately form the Middleware Services Adding step.
3. Adding the default version of the Middleware Services without the Options Object. It will take Default Version of Options object.
4. Adding the customized version of the Middleware Services with the Options Object. It will take the customized Options Object. So, we are adding Middleware Services and Customizing Middleware services at the same time.
5. Adding the Middleware without specifying the Options Object. Customizing the Options Object separately after adding the Middleware Services.
6. Middleware.AddXXXX() to further add more dependencies. These AddXXX() also include Generic Parameters and Options Object and the

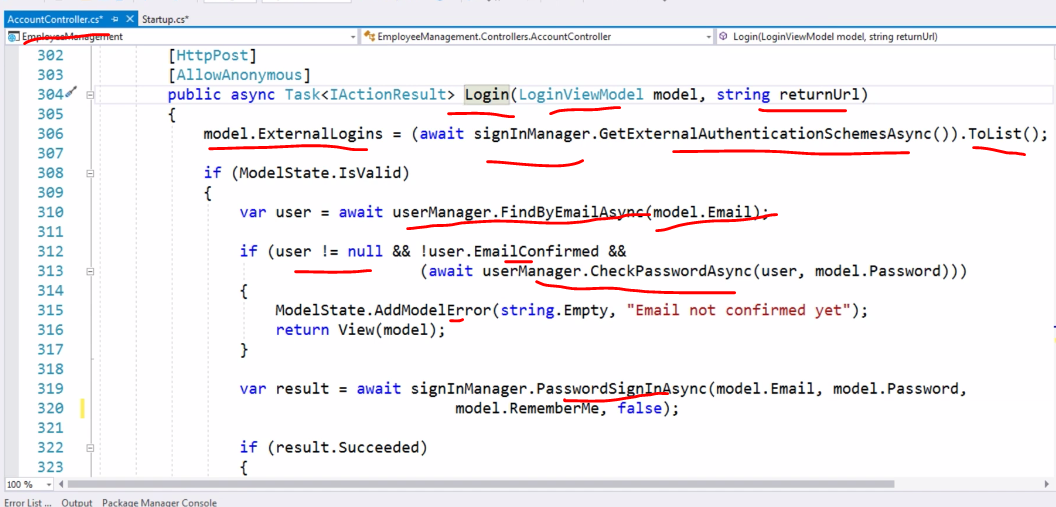


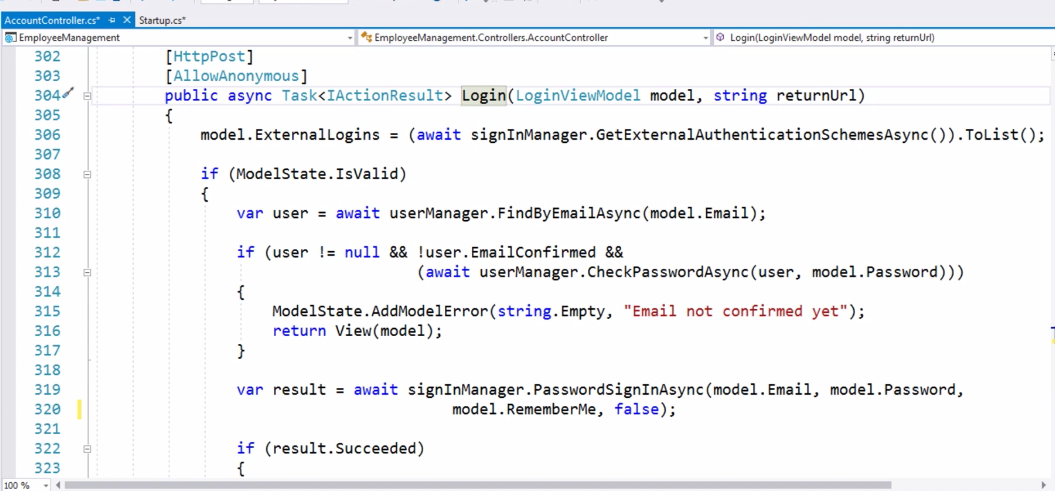


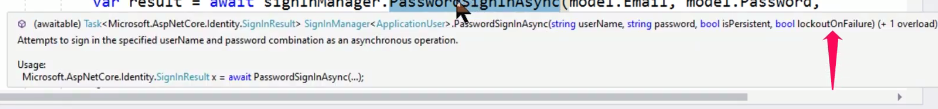
1. Add the Nuget Package of Identity Middleware
2. Lockout DB part of Identity is always there
3. Add the Identity middleware to the RPP
4. Add the DEFAULT SERVICES needed for the Identity middleware
5. Add the CONFIGURED SERVICES needed for Identity middleware.
6. Enable the Lockout Functionality when Signing In. When we are signing in, we need that Lockout Functionality. Sign In user under the lockout functionality.

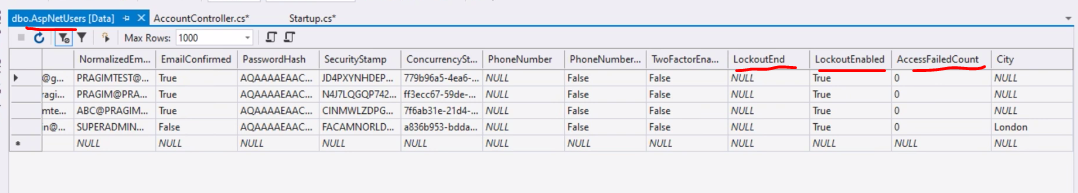


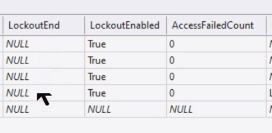


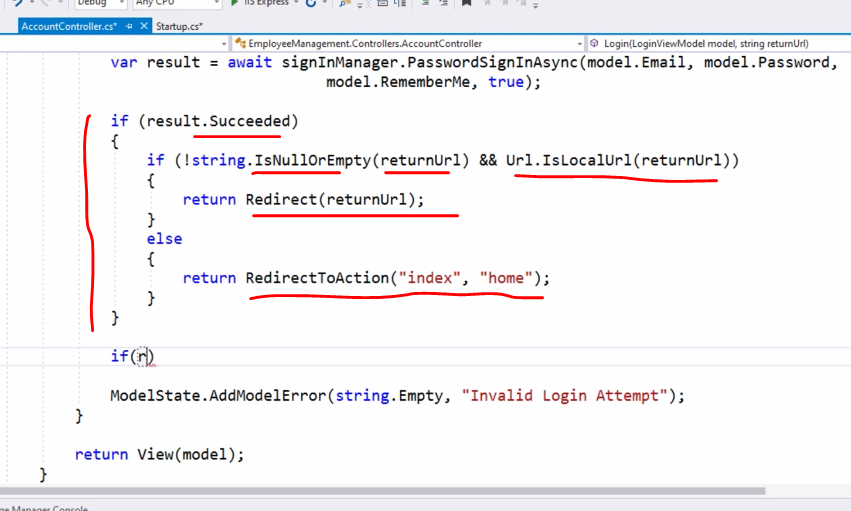


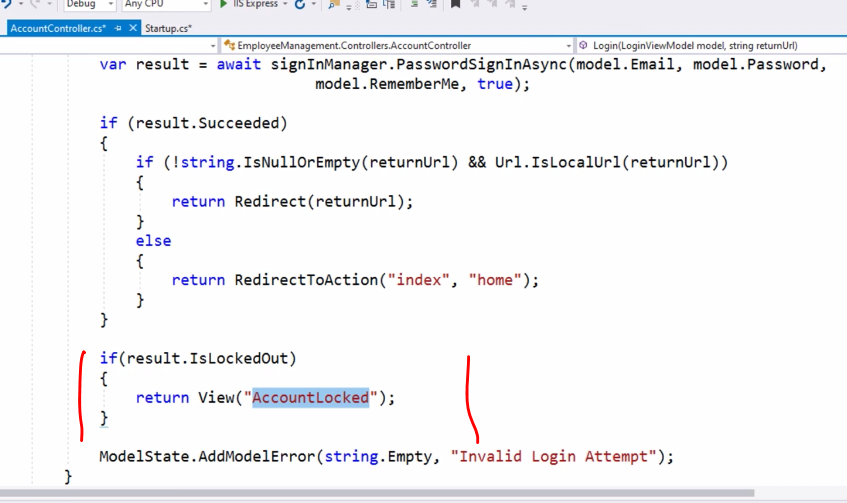


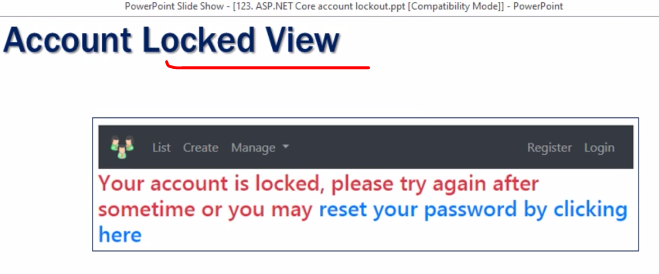


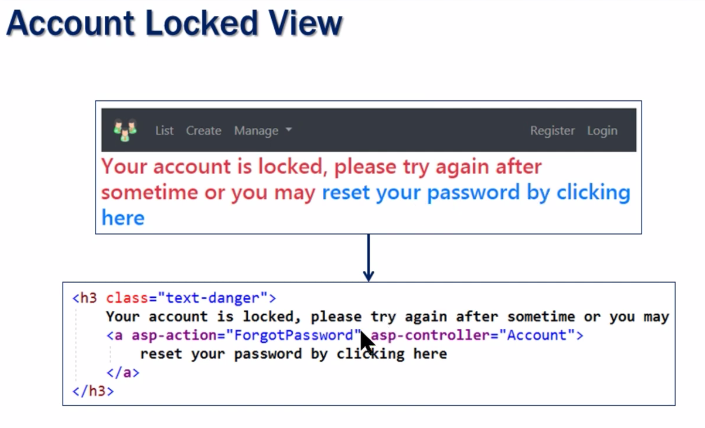


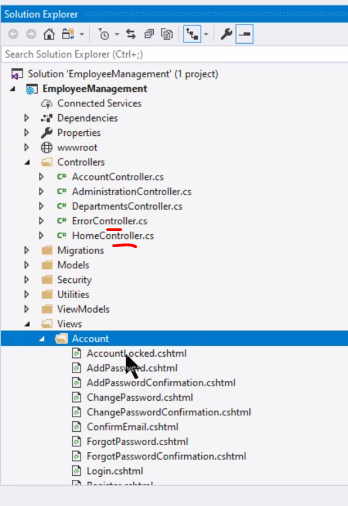


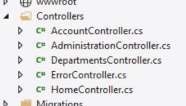


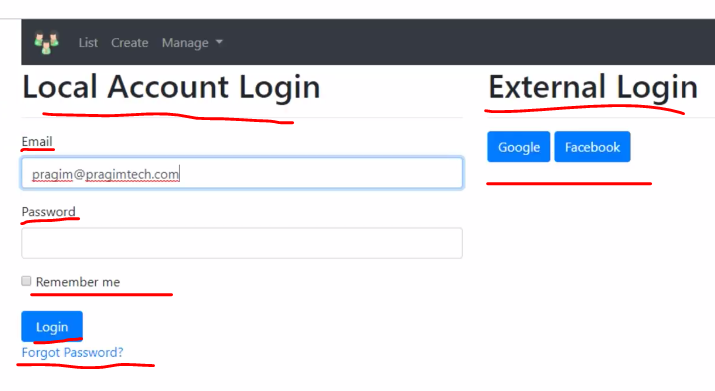


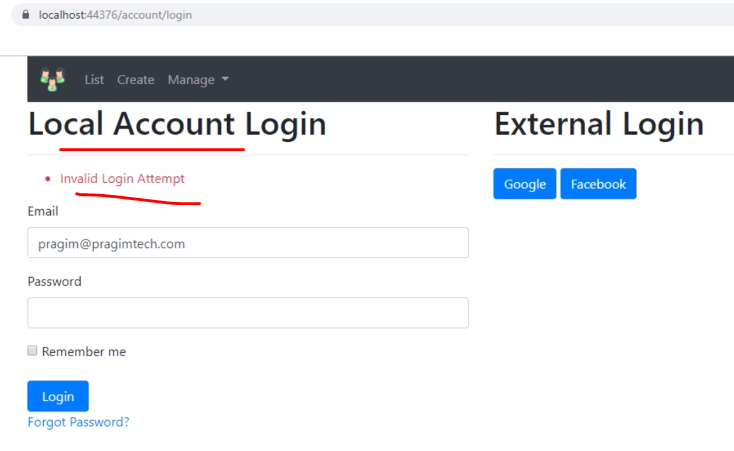


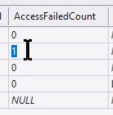


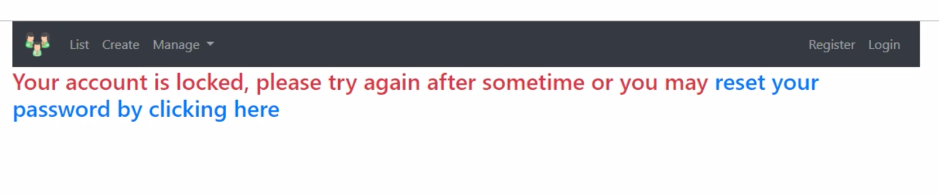








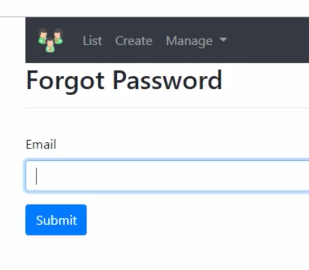


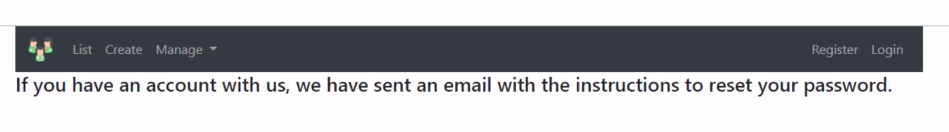


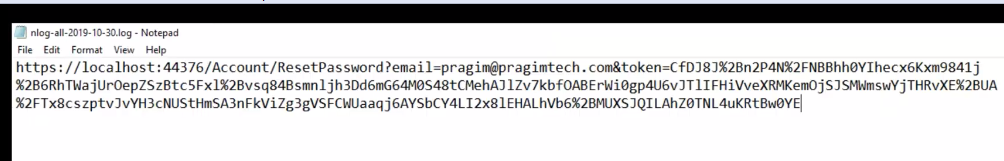


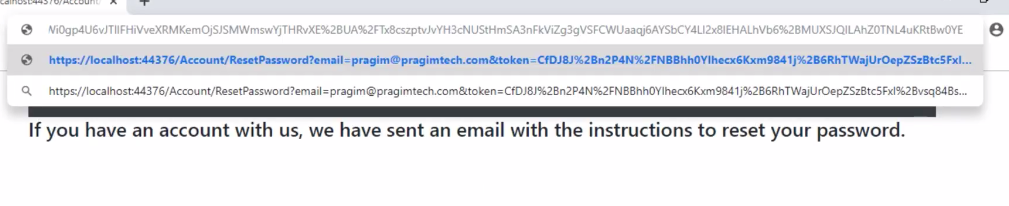
Now 2 options

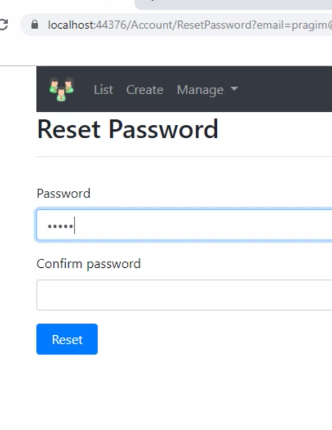
1. Wait for 15 minutes and retry the old correct password
2. Reset the password

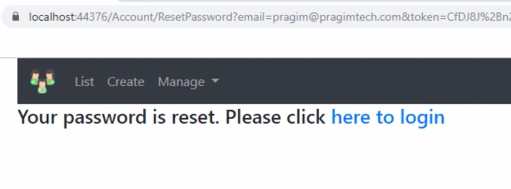




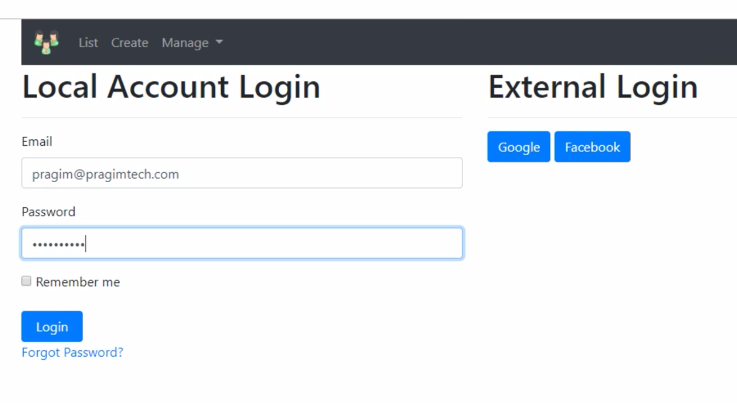


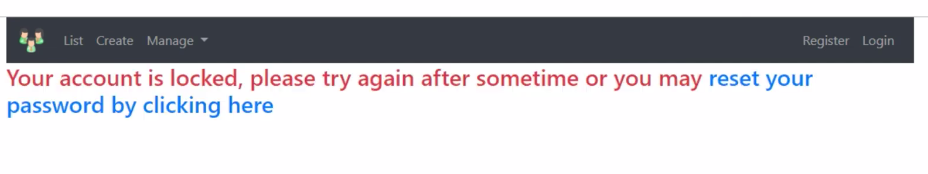






New Password is given





Password is reset, now you know the password but the Account is still locked. You need to wait for 15 mintues and apply new correct password.

Another options is When You Reset the password, the Unlock the Account as well by setting the LockData to Null or Now value

