

ASP.NET Identity 2.1 Accounts Confirmation, and Password Policy Configuration - Part 2

February 3, 2015 By [Taiseer Joudeh](#) – 39 Comments

Be Sociable, Share!



Share

11

G+ Share

4

Email

This is the second part of Building Simple Membership system using ASP.NET Identity 2.1, ASP.NET Web API 2.2 and AngularJS. The topics we'll cover are:

- [Configure ASP.NET Identity with ASP.NET Web API \(Accounts Management\) – Part 1.](#)
- ASP.NET Identity 2.1 Accounts Confirmation, and Password/User Policy Configuration – (This Post)
- [Implement OAuth JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – Part 3](#)
- [ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API – Part 4](#)
- [ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5](#)
- *AngularJS Authentication and Authorization with ASP.NET Web API and Identity 2.1 – Part 6*

The **source code** for this tutorial is available on GitHub.

ASP.NET Identity 2.1 Accounts Confirmation, and Password/User

Policy Configuration

In this post we'll complete on top of what we've already built, and we'll cover the below topics:

- Send Confirmation Emails after Account Creation.
- Configure User (Username, Email) and Password policy.
- Enable Changing Password and Deleting Account.

1. Send Confirmation Emails after Account Creation

ASP.NET Identity 2.1 users table (AspNetUsers) comes by default with a Boolean column named "EmailConfirmed", this column is used to flag if the email provided by the registered user is valid and belongs to this user in other words that user can access the email provided and he is not impersonating another identity. So our membership system should not allow users without valid email address to log into the system.



The scenario we want to implement that user will register in the system, then a confirmation email will be sent to the email provided upon the registration, this email will include an activation link and a token (code) which is tied to this user only and valid for certain period.

Once the user opens this email and clicks on the activation link. and if the token (code) is valid the field

“EmailConfirmed” will be set to “true” and this proves that the email belongs to the registered user.

To do so we need to add a service which is responsible to send emails to users, in my case I’ll use [Send Grid](#) which is service provider for sending emails, but you can use any other service provider or your exchange change server to do this. If you want to follow along with this tutorial you can create a [free account](#) with Send Grid which provides you with 400 email per day, pretty good!

1.1 Install Send Grid

BIT OF TECHNOLOGY

ed
ikes

[ARCHIVE](#)[ABOUT ME](#)[SPEAKING](#)[CONTACT](#)

```
1 install-package Sendgrid
```

1.2 Add Email Service

Now add new folder named “Services” then add new class named “EmailService” and paste the code below:

```
1 public class EmailService : IIdentityMessageService
2 {
3     public async Task SendAsync(IdentityMessage message)
4     {
5         await configSendGridAsync(message);
6     }
7
8     // Use NuGet to install SendGrid (Basic C# client lib)
9
10    private async Task configSendGridAsync(IdentityMessage message)
11    {
12        var myMessage = new SendGridMessage();
13        myMessage.AddTo(message.Destination);
14        myMessage.From = new System.Net.Mail.MailAddress("taiseer@bitoftech.net", "Taiseer Joudeh");
15        myMessage.Subject = message.Subject;
```

```
16 myMessage.Text = message.Body;
17 myMessage.Html = message.Body;
18
19 var credentials = new NetworkCredential(ConfigurationManager.AppSettings["emailService:Account"],
20                                         ConfigurationManager.AppSettings["emailService:Password"]);
21
22 // Create a Web transport for sending email.
23 var transportWeb = new Web(credentials);
24
25 // Send the email.
26 if (transportWeb != null)
27 {
28     await transportWeb.DeliverAsync(myMessage);
29 }
30 else
31 {
32     //Trace.TraceError("Failed to create Web transport.");
33     await Task.FromResult(0);
34 }
35 }
36 }
```

What worth noting here that the class “EmailService” implements the interface “IIdentityMessageService”, this interface can be used to configure your service to send emails or SMS messages, all you need to do is to implement your email or SMS Service in method “SendAsync” and your are good to go.

In our case we want to send emails, so I’ve implemented the sending process using Send Grid in method “configSendGridAsync”, all you need to do is to replace the sender name and address by yours, as well do not forget to add 2 new keys named “emailService:Account” and “emailService:Password” as AppSettings to store Send Grid credentials.

After we configured the “EmailService”, we need to hock it with our Identity system, and this is very simple step, open file “ApplicationUserManager” and inside method “Create” paste the code below:

```
1 public static ApplicationUserManager Create(IdentityFactoryOptions<ApplicationUserManager> options, IOwinContext conte
2 {
3     //Rest of code is removed for clarity
4     appUserManager.EmailService = new AspNetIdentity.WebApi.Services.EmailService();
5 }
```

```

6     var dataProtectionProvider = options.DataProtectionProvider;
7     if (dataProtectionProvider != null)
8     {
9         appUserManager.UserTokenProvider = new DataProtectorTokenProvider<ApplicationUser>(dataProtectionProvider.Crea
10    {
11        //Code for email confirmation and reset password life time
12        TokenLifespan = TimeSpan.FromHours(6)
13    };
14    }
15
16    return appUserManager;
17 }

```

As you see from the code above, the “appUserManager” instance contains property named “EmailService” which you set it the class we’ve just created “EmailService”.

Note: There is another property named “SmsService” if you would like to use it for sending SMS messages instead of emails.

Notice how we are setting the expiration time for the code (token) send by the email to 6 hours, so if the user tried to open the confirmation email after 6 hours from receiving it, the code will be invalid.

1.3 Send the Email after Account Creation

Now the email service is ready and we can start sending emails after successful account creation, to do so we need to modify the existing code in the method “CreateUser” in controller “AccountsController“, so open file “AccountsController” and paste the code below at the end of the method:

```

1  //Rest of code is removed for brevity
2
3  string code = await this.AppUserManager.GenerateEmailConfirmationTokenAsync(user.Id);
4
5  var callbackUrl = new Uri(Url.Link("ConfirmEmailRoute", new { userId = user.Id, code = code }));
6
7  await this.AppUserManager.SendEmailAsync(user.Id, "Confirm your account", "Please confirm your account by clicking <a h
8
9  Uri locationHeader = new Uri(Url.Link("GetUserById", new { id = user.Id }));
10
11 return Created(locationHeader, RedirectToAction("Create", "Users"));

```

```
11 return Created(locationHeader, _inMemoryFactory.Create(user));
```

The implementation is straight forward, what we've done here is creating a unique code (token) which is valid for the next 6 hours and tied to this user Id only this happen when calling "GenerateEmailConfirmationTokenAsync" method, then we want to build an activation link to send it in the email body, this link will contain the user Id and the code created.

Eventually this link will be sent to the registered user to the email he used in registration, and the user needs to click on it to activate the account, the route "ConfirmEmailRoute" which maps to this activation link is not implemented yet, we'll implement it the next step.

Lastly we need to send the email including the link we've built by calling the method "SendEmailAsync" where the constructor accepts the user Id, email subject, and email body.

1.4 Add the Confirm Email URL

The activation link which the user will receive will look as the below:

```
1 http://localhost/api/account/ConfirmEmail?userid=xxxx&code=xxxx
```

So we need to build a route in our API which receives this request when the user clicks on the activation link and issue HTTP GET request, to do so we need to implement the below method, so in class "AccountsController" as the new method as the below:

```
1 [HttpGet]
2 [Route("ConfirmEmail", Name = "ConfirmEmailRoute")]
3 public async Task<IHttpActionResult> ConfirmEmail(string userId = "", string code = "")
4 {
5     if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(code))
6     {
7         ModelState.AddModelError("", "User Id and Code are required");
8         return BadRequest(ModelState);
9     }
10 }
```

```
10
11     IdentityResult result = await this.AppUserManager.ConfirmEmailAsync(userId, code);
12
13     if (result.Succeeded)
14     {
15         return Ok();
16     }
17     else
18     {
19         return GetErrorResult(result);
20     }
21 }
```

The implementation is simple, we only validate that the user Id and code is not not empty, then we depend on the method “ConfirmEmailAsync” to do the validation for the user Id and the code, so if the user Id is not tied to this code then it will fail, if the code is expired then it will fail too, if all is good this method will update the database field “EmailConfirmed” in table “AspNetUsers” and set it to “True”, and you are done, you have implemented email account activation!

Important Note: It is recommended to validate the password before confirming the email account, in some cases the user might miss type the email during the registration, so you do not want end sending the confirmation email for someone else and he receives this email and activate the account on your behalf, so better way is to ask for the account password before activating it, if you want to do this you need to change the “ConfirmEmail” method to POST and send the Password along with user Id and code in the request

body, you have the idea so you can implement it by yourself 😊

2. Configure User (Username, Email) and Password policy

2.1 Change User Policy

In some cases you want to enforce certain rules on the username and password when users register into your system, so ASP.NET Identity 2.1 system offers this feature, for example if we want to enforce that our username only allows **alphanumeric** characters and the email associated with this user is unique then all we need to do is to set those properties in class “ApplicationUserManager”, to do so open file

“ApplicationUserManager” and paste the code below inside method “Create”:

```
1 //Rest of code is removed for brevity
2 //Configure validation logic for usernames
3 appUserManager.UserValidator = new UserValidator<ApplicationUser>(appUserManager)
4 {
5     AllowOnlyAlphanumericUserNames = true,
6     RequireUniqueEmail = true
7 };
```

2.2 Change Password Policy

The same applies for the password policy, for example you can enforce that the password policy must match (minimum 6 characters, requires special character, requires at least one lower case and at least one upper case character), so to implement this policy all we need to do is to set those properties in the same class “ApplicationUserManager” inside method “Create” as the code below:

```
1 //Rest of code is removed for brevity
2 //Configure validation logic for passwords
3 appUserManager.PasswordValidator = new PasswordValidator
4 {
5     RequiredLength = 6,
6     RequireNonLetterOrDigit = true,
7     RequireDigit = false,
8     RequireLowercase = true,
9     RequireUppercase = true,
10 };
```

2.3 Implement Custom Policy for User Email and Password

In some scenarios you want to apply your own custom policy for validating email, or password. This can be done easily by creating your own validation classes and hook it to “UserValidator” and “PasswordValidator” properties in class “ApplicationUserManager”.

For example if we want to enforce using only the following domains (“outlook.com”, “hotmail.com”).

“gmail.com”, “yahoo.com”) when the user self registers then we need to create a class and derive it from “UserValidator<ApplicationUser>” class, to do so add new folder named “Validators” then add new class named “MyCustomUserValidator” and paste the code below:

```
1  public class MyCustomUserValidator : UserValidator<ApplicationUser>
2  {
3
4      List<string> _allowedEmailDomains = new List<string> { "outlook.com", "hotmail.com", "gmail.com", "yahoo.com"
5
6      public MyCustomUserValidator(ApplicationUserManager appUserManager)
7          : base(appUserManager)
8      {
9      }
10
11     public override async Task<IdentityResult> ValidateAsync(ApplicationUser user)
12     {
13         IdentityResult result = await base.ValidateAsync(user);
14
15         var emailDomain = user.Email.Split('@')[1];
16
17         if (!_allowedEmailDomains.Contains(emailDomain.ToLower()))
18         {
19             var errors = result.Errors.ToList();
20
21             errors.Add(String.Format("Email domain '{0}' is not allowed", emailDomain));
22
23             result = new IdentityResult(errors);
24         }
25
26         return result;
27     }
28 }
```

What we have implemented above that the default validation will take place then this custom validation in method “ValidateAsync” will be applied, if there is validation errors it will be added to the existing “Errors” list and returned in the response.

In order to fire this custom validation, we need to open class “ApplicationUserManager” again and hook this custom class to the property “UserValidator” as the code below:

```
1 //Rest of code is removed for brevity
2 //Configure validation logic for usernames
3 appUserManager.UserValidator = new MyCustomUserValidator(appUserManager)
4 {
5     AllowOnlyAlphanumericUserNames = true,
6     RequireUniqueEmail = true
7 };
```

Note: The tutorial code is not using the custom “MyCustomUserValidator” class, it exists in the source code for your reference.

Now the same applies for adding custom password policy, all you need to do is to create class named “MyCustomPasswordValidator” and derive it from class “PasswordValidator”, then you override the method “ValidateAsync” implementation as below, so add new file named “MyCustomPasswordValidator” in folder “Validators” and use the code below:

```
1 public class MyCustomPasswordValidator : PasswordValidator
2 {
3     public override async Task<IdentityResult> ValidateAsync(string password)
4     {
5         IdentityResult result = await base.ValidateAsync(password);
6
7         if (password.Contains("abcdef") || password.Contains("123456"))
8         {
9             var errors = result.Errors.ToList();
10            errors.Add("Password can not contain sequence of chars");
11            result = new IdentityResult(errors);
12        }
13        return result;
14    }
15 }
```

In this implementation we added some basic rule which checks if the password contains sequence of characters and reject this type of password by adding this validation result to the Errors list, it is exactly the same as the custom users policy.

Now to attach this class as the default password validator, all you need to do is to open class

“ApplicationUserManager” and use the code below:

```
1 //Rest of code is removed for brevity
2 // Configure validation logic for passwords
3 appUserManager.PasswordValidator = new MyCustomPasswordValidator
4 {
5     RequiredLength = 6,
6     RequireNonLetterOrDigit = true,
7     RequireDigit = false,
8     RequireLowercase = true,
9     RequireUppercase = true,
10 };
```

All other validation rules will take place (i.e checking minimum password length, checking for special characters) then it will apply the implementation in our “MyCustomPasswordValidator”.

3. Enable Changing Password and Deleting Account

Now we need to add other endpoints which allow the user to change the password, and allow a user in “Admin” role to delete other users account, but those end points should be accessed only if the user is authenticated, we need to know the identity of the user doing this action and in which role(s) the user belongs to. Until now all our endpoints are called anonymously, so lets add those endpoints and we’ll cover the authentication and authorization part next.

3.1 Add Change Password Endpoint

This is easy to implement, all you need to do is to open controller “AccountsController” and paste the code below:

```
1 [Route("ChangePassword")]
2 public async Task<IHttpActionResult> ChangePassword(ChangePasswordBindingModel model)
3 {
```

```
4         if (!ModelState.IsValid)
5         {
6             return BadRequest(ModelState);
7         }
8
9         IdentityResult result = await this.AppUserManager.ChangePasswordAsync(User.Identity.GetUserId(), model.Old
10
11         if (!result.Succeeded)
12         {
13             return GetErrorResult(result);
14         }
15
16         return Ok();
17     }
```

Notice how we are calling the method “ChangePasswordAsync” and passing the authenticated User Id, old password and new password. If you tried to call this endpoint, the extension method “GetUserId” will not work because you are calling it as anonymous user and the system doesn’t know your identity, so hold on the testing until we implement authentication part.

The method “ChangePasswordAsync” will take care of validating your current password, as well validating your new password policy, and then updating your old password with new one.

Do not forget to add the “ChangePasswordBindingModel” to the class “AccountBindingModels” as the code below:

```
1     public class ChangePasswordBindingModel
2     {
3         [Required]
4         [DataType(DataType.Password)]
5         [Display(Name = "Current password")]
6         public string OldPassword { get; set; }
7
8         [Required]
9         [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
10        [DataType(DataType.Password)]
11        [Display(Name = "New password")]
12        public string NewPassword { get; set; }
13    }
```

```
14 [Required]
15 [DataType(DataType.Password)]
16 [Display(Name = "Confirm new password")]
17 [Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]
18 public string ConfirmPassword { get; set; }
19
20 }
```

3.2 Delete User Account

We want to add the feature which allows a user in “Admin” role to delete user account, until now we didn’t introduce Roles management or authorization, so we’ll add this end point now and later we’ll do slight modification on it, for now any anonymous user can invoke it and delete any user by passing the user Id.

To implement this we need add new method named “DeleteUser” to the “AccountsController” as the code below:

```
1 [Route("user/{id:guid}")]
2 public async Task<IHttpActionResult> DeleteUser(string id)
3 {
4
5     //Only SuperAdmin or Admin can delete users (Later when implement roles)
6
7     var appUser = await this.AppUserManager.FindByIdAsync(id);
8
9     if (appUser != null)
10    {
11        IdentityResult result = await this.AppUserManager.DeleteAsync(appUser);
12
13        if (!result.Succeeded)
14        {
15            return GetErrorResult(result);
16        }
17
18        return Ok();
19    }
20
21    return NotFound();
22
23 }
24 }
```

This method will check the existence of the user id and based on this it will delete the user. To test this method we need to issue HTTP DELETE request to the end point “api/accounts/user/{id}”.

The **source code** for this tutorial is available on [GitHub](#).

In the **next post** we'll see how we'll implement Json Web Token (JWTs) Authentication and manage access for all the methods we added until now.

Follow me on Twitter [@tjoudeh](#)

References

- [Featured Image Source](#)

Be Sociable, Share!



Share

11

G+ Share

4

Email

Like this:

Loading...

Related Posts

[ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5](#)

[ASP.NET Identity 2.1 with ASP.NET Web API 2.2 \(Accounts Management\) – Part 1](#)

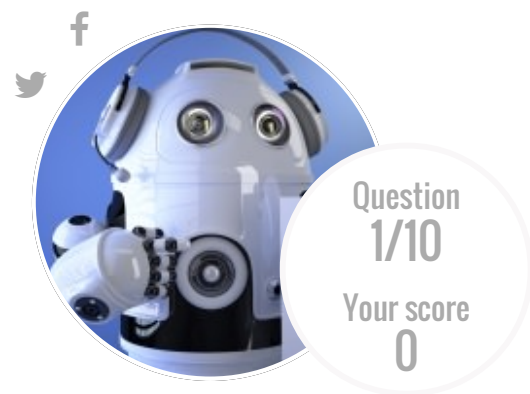
[AngularJS Authentication Using Azure Active Directory Authentication Library \(ADAL\)](#)

[Two Factor Authentication in ASP.NET Web API & AngularJS using Google Authenticator](#)

[Secure ASP.NET Web API 2 using Azure Active Directory, Owin Middleware, and ADAL](#)

Filed Under: [ASP.NET](#), [ASP.NET Identity](#), [ASP.Net Web API](#), [Web API Tutorial](#)

Tagged With: [Token Authentication](#), [Tutorial Web API 2](#)




Are You a True Robot Overload?

The word "robot" comes from the Czech words "robota" which translates to "forced labor"


TRUE ☒ FALSE ☐

1 2 3 4 5 6 7 8 9 10

Advertisement:



WRITE FOR THE
WORLD'S TOP BRANDS



JOIN TODAY

Comments



Liam says

February 3, 2015 at 2:45 am

Good article, but you should really change all the references to 'He/him' etc to something gender neutral.

Reply



Taiseer Joudeh says

February 3, 2015 at 4:08 am

Thanks Liam for your comment, It was a mistake, all is fixed now, thanks again 😊

Reply



Akinsanya Olanrewaju says

February 9, 2015 at 12:14 pm

Your Post have been a life saving material for us in my region, with all your past post and this, you have change our thinking toward DotNet Development. Thanks.

Please we need this series to be completed on time, so as to use it to complete our school project.
(We are student from Africa), We enjoy your series.

Thanks.

Reply



Taiseer Joudeh says

February 13, 2015 at 8:06 pm

Glad to hear this, happy to know that I'm helping students in other contents to learn some cool technologies, good luck 😊

Reply



AndreasF says

February 10, 2015 at 1:16 pm

When is the next post? Waiting for it!
Nice posts. Identity is hard for me to understand because it has so many functions.
It's the first time i begin understanding it. Thanks!

Reply



Taiseer Joudeh says

February 10, 2015 at 3:56 pm

Glad it was useful, most probably tomorrow 😊 keep tuned!

Reply



Kurai says

February 11, 2015 at 3:29 pm

Been here in the past months and I am learning new things in every post you have.. -I'm just another kid who wants to be something great.. and I tell you.. when that happen.. I will put your name as one my "hero"... Thank you so much...

Reply



Taiseer Joudeh says

February 11, 2015 at 3:40 pm

Thanks for your sweet message, I'm really happy to know that my contribution is helping a lot of people out there, love what you do and have passion for it and I'm sure you will be a shining star =)

Reply



Tony Pollard says

February 12, 2015 at 6:32 pm

I've been following you for some time now, and I appreciate your approach to start with an empty project and add what you need with adequate explanation as you go. I find myself eager and watching for your next post as you are regular reading for me. Great job Taiseer!

Reply



Taiseer Joudeh says

February 13, 2015 at 8:00 pm

Thank you Tony for your comment, glad to know that posts are useful. Part 3 is ready and should be published this Monday.

Reply



Miguel Delgado says

February 16, 2015 at 8:04 am

Hello Taisser, thanks for all your posts and very happy to see you MVP.

Alas, I hit a bit of a bump implementing your code. I create the user and while creating the link to send the confirmation email, it raises an exception:

on the instruction

```
var callbackUrl = new Uri(Url.Link("ConfirmEmailRoute", new { userId = user.Id, code = code }));
```

the Url.Link raises the exception

```
{ "Value cannot be null.\r\nParameter name: request" }
```

I checked the function signature and it matches...

Do you have any idea on how to fix this... I can always hardcode the uri, but I'd rather suffer the pains of hell...

Thanks in advance

Miguel Delgado

Reply



Taiseer Joudeh says

February 16, 2015 at 10:23 am

Hi Miguel,

Thanks for your nice words,

Double check that you have created and named the route "ConfirmEmailRoute" correctly, as well you are passing the exact parameters. This is only what I can think of now.

Download the repo and compare the code, I'm sure there is simple glitch there.

Hope this helps!

Reply



Diego says

February 19, 2015 at 3:33 am

Hi Taiseer,

What's the point in creating an email confirmation api endpoint?

The user will see nothing else than a white screen in the browser.

The logical thing would be creating a web page to show a confirmation message, but I guess in this post you just wanted to focus on webapi.

Apart from that, please tell me what you think about this

In my case, I'm implementing all this from a mobile application and I'm trying to make things easy to the user. Imagine the situation: a user registers in the app and receives a message asking him/her to check the email. The email will redirect the user to a web page (confirmation page), and then, that user has to go back to the app. 4 step process (including registration form) is a bit overhead IMO.

So I was thinking about sending a short code (like the kind of sms confirmation codes) to the email so the user can just write it in a textbox (inside the mobile app) instead of using urls. I'm not sure if there is any way to modify the confirmation token asp.net generates to make for example, a 4 character number code. In that case you could see the code in the push notification of your email box and even no need to open it

Any thoughts?

Reply



Taiseer Joudeh says

February 19, 2015 at 8:47 pm

Hi Diego,

As you suggested you need to build a GUI on your system where in contains a link for the "ConfirmEmailRoute" end point, once the user invokes this endpoint (click on that link) and you receive 200 OK status from the API, your GUI needs to display confirmation message and maybe redirect him to the application as this [SO question](#). Never tried it before 😊

Regarding send SMS, as you suggested here you are going to send the SMS to the email so the user might

need to open the email to read the SMS code there and then close the email and go back to application to enter it, so the same number of steps.

As well I'm not sure if you can generate friendly token (4 digits) and attach it to the user Id so the validation for this token happens seamlessly as it happens in method "AppUserManager.ConfirmEmailAsync", I need to check this.

Reply



Diego says

February 19, 2015 at 9:46 pm

Hi Taiseer,

About the friendly token:

If you do something like this in your asp.net userManager...

```
> IDataProtector dataProtector = dataProtectionProvider.Create("ASP.NET Identity");  
> this.UserTokenProvider = new DataProtectorTokenProvider(dataProtector);
```

you get a huge token (kind of OAuth token), but I realized you can get a 6 digit "token" if you do this instead:

```
> this.UserTokenProvider = new EmailTokenProvider();
```

Not sure about the impact in the application or any related security issues... but I can say the confirmation works fine as well in this way.

About the confirmation message:

I'm not using legacy MVC in my project. Just webapi so I'm writing an html response like this:

```
[ActionName("emailconfirmation")]  
public async Task ConfirmEmailAccount(string userId, string token)  
{
```

```
IdentityResult result = await aspNetUserManager.ConfirmEmailAsync(userId, token);
```

```
if (result.Succeeded)
```

```
{
```

```
    string body = "bla bla bla"; // this will be a proper web page
```

```
    var response = new HttpResponseMessage(HttpStatusCode.OK);
```

```
    response.Content = new StringContent(body, Encoding.UTF8, "text/html");
```

```
    return response;
```

```
}
```

```
else
```

```
{
```

```
    GetErrorResult(result.Errors);
```

```
    return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ModelState);
```

```
}
```

```
}
```

About steps overhead and usability in mobile applications:

I've been reading about this in stack-overflow and many people agree that email confirmations on mobile applications are a no-go for many users. Of course that depends on your project needs but reasonable in many cases. So a nice way to go would be:

1. You send the welcome-confirmation email
2. The user can start using the app without confirming
3. After a time period you remind the user to confirm the account (by email)
4. After another time period, if the account has not been confirmed, you just delete it (this can be done in a background scheduled job)

That way, the users just trying to test your app will be able to do it quickly. And if they are interested after all, clicking a link shouldn't be a problem.

Thanks for your answer!

ps: is there anyway to tag text as code in the comments?

Reply



Diego says

February 19, 2015 at 9:56 pm

I forgot to mention about redirecting user to the app from the email box.

That's a nice solution if the user opens the email in the mobile device where the app is.

This can be done (I'm using Xamarin) like this: <http://developer.xamarin.com/recipes/cross-platform/app-links/app-links-ios/> and this: <http://developer.xamarin.com/recipes/cross-platform/app-links/app-links-android/>

Reply



Akinsanya Olanrewaju says

March 13, 2015 at 1:56 pm

I ran the source code from the updated one from github, after update the package manager, I want to the create a new account from the create end-point

<http://prntscr.com/6g70sh>

This is the error i get <http://prntscr.com/6g70z4>

I have done all i needed to do, i dont know am doing wrong

1. <http://prntscr.com/6g7180>

2. <http://prntscr.com/6g71s5> : Here, this is the credentials from my signup in sendgrid

Can someone help me out.

Reply



Taiseer Joudeh says

March 16, 2015 at 5:15 pm

This issue with the SendGrid service trying to send confirmation email, make sure you are using the same NuGet package I have used in the project by checking packages.config file and double check that SendGrid username and password are set correctly in web.config as you obtained them from SendGrid after creating a SendGrid account.

Reply



Regan says

March 14, 2015 at 7:47 pm

I dont get the interface >> IIdentityMessageService

Reply



Taiseer Joudeh says

March 16, 2015 at 5:20 pm

What is not clear about it? It interface used to separate the implementation of your SMS service or Email service from Web API logic, it has method named "SendAsync" in order to implement your sending logic in it

Reply



Simon says

March 25, 2015 at 11:02 pm

Hi Taiseer,

First off I would like to thank you for your post and all the information concerning JWT. I am currently doing an internship in a company where I was asked to do a small authentication and authorization using Identity + JWT. I ran into a small bug and I don't know if its me that did not implement your explications properly but for some reason I cannot seem to be able to do any calls to Identity, for example `User.Identity.GetUserId();` is always null. It feels like the data/information from my token does not get passed. Would you be able to give me any pointers or would you have any idea? I have been looking at your AngularJSAuthentication solution and all your other post and I can't seem to be able to find out what it is.

Thank you,

Simon.

Reply



Taiseer Joudeh says

March 27, 2015 at 3:51 pm

Hi Simon, are you calling the method `User.Identity.GetUserId()`; inside a protected controller? A controller or action method attributed with `[Authorize]`? If yes then the Identity should be set, can you try checking the property `User.Identity.Name` too? Does it return the authenticated `UserId`?

Reply



felipefurlan says

August 12, 2015 at 6:59 pm

Hello Taisser, how are you doing? I'm having this issue too. The `User.Identity.Name` is OK, but the ID is

always null. When I inspect the `User.Identity` on Immediate Window I receive the following:

```
{System.Security.Claims.ClaimsIdentity}  
[System.Security.Claims.ClaimsIdentity]: {System.Security.Claims.ClaimsIdentity}  
AuthenticationType: "Bearer"  
IsAuthenticated: true  
Name: "bla@bla.com.br"
```

I'm not sure if I'm missing something here.

Reply



Akinsanya Olanrewaju says

April 1, 2015 at 2:16 pm

Hi,

```
List _allowedEmailDomains = new List { "outlook.com", "hotmail.com", "gmail.com", "yahoo.com" };
```

How do i create a CustomUserValidator for user and admin email separately

My application will be split into two admin registration and user registration

I want the users to register with any email address (e.g gmail.com, yahoo.com) while the admin will only register with the official company email (e.g [admin@company.com](#))

How can i seperate the logic for this or conditionally configure this, since the validation logic for the user admin are called from one single point in the startup.cs

Is it advisable to create the API for admin seperate from the user, or i can use thesame API for both logic.

Thanks

[Reply](#)



Ernst Bolt says

April 8, 2015 at 10:06 am

Hi Taiseer,

Microsoft.Owin.Testing doesn't provide a `DataProtectionProvider` in the options parameter of `ApplicationUserManager.Create()`. I've changed the method to:

...

```
var dataProtectionProvider = options.DataProtectionProvider;
if (dataProtectionProvider != null)
{
    appUserManager.UserTokenProvider = new
    DataProtectorTokenProvider(dataProtectionProvider.Create("ASP.NET Identity"))
    {
        //Code for email confirmation and reset password life time
        TokenLifespan = TimeSpan.FromHours(6)
    };
}
else
{
    var provider = new Microsoft.Owin.Security.DataProtection.DpapiDataProtectionProvider("ASP.NET Identity");
    UserManager userManager = new UserManager(new UserStore());
    appUserManager.UserTokenProvider = new DataProtectorTokenProvider(provider.Create("ASP.NET Identity")) {

        //Code for email confirmation and reset password life time
        TokenLifespan = TimeSpan.FromHours(6)

    };
}
```

...

Now it's working.

Reply



GRAPHC_coder says

May 15, 2015 at 9:09 am

Hi Taiseer,

Great post! It really helps me on my project.

Just a quick heads up for those who might experience same as mine.

I was setting up our own SMTP server and testing email confirmation through Postman, "Invalid Token" happened for URL confirmation.

Solution:

<http://tech.trailmax.info/2015/05/asp-net-identity-invalid-token-for-password-reset-or-email-confirmation/>

It works for me:

CreateUser() add

```
code = System.Web.HttpUtility.UrlEncode(code);
```

ConfirmEmail() add

```
code = System.Web.HttpUtility.UrlDecode(code);
```

Reply



Taiseer Joudeh says

May 15, 2015 at 11:48 pm

Thanks for sharing this, what you are doing is better practice because the token might contain unsafe URL chars which needs to be Url encoded before.

Reply



Doug says

June 27, 2015 at 3:11 am

The callback URL is for a GET to the API, but how do you handle the case where the front end is solely Angular? How can I get the ConfirmEmail API function to, upon execution from the email link click event, redirect back to my front end Angular site after updating the EmailConfirmed flag?

Reply



Mcshaz says

July 10, 2015 at 1:33 am

Thanks for posting this.

just an expansion on the comment:

“Notice how we are setting the expiration time for the code (token) send by the email to 6 hours”

so far, my testing on the Indentity 2.1 framework would indicate that the token contains datetime information on when it was sent, rather than any data on when it is to expire. The ConfirmEmailAsync method then seems to apply the TokenLifespan as part of validating the token. The relevance of this is that changing TokenLifespan (in testing or production) will be effective retrospectively on tokens previously sent

Testing or production) will be effective retroactively on tokens previously sent.

Reply



Nathan says

August 4, 2015 at 9:45 pm

Did you find that GenerateEmailConfirmationTokenAsync is creating a 500 character token? i.e.

<http://localhost/api/account/ConfirmEmail?userid=xxxx&code=>

Sending such a long URL is running up against spam filters in our case.

Reply



Taiseer Joudeh says

August 8, 2015 at 7:11 am

Hi Nathan,

To be honest I didn't notice that huge number of generated characters, were you able to find solution for this issue?

Reply



Ali Morlo says



Ali Morlo says

August 13, 2015 at 4:21 pm

Slm from Africa,

thx for the stuff

i'm unable to pull the namespace for `AspNetIdentity.WebApi.Services.EmailService()`

Reply



Ali Morlo says

August 13, 2015 at 4:35 pm

I Got it Sorry, Just Need Some rest

Reply



Taiseer Joudeh says

August 15, 2015 at 1:09 am

No problem Ali 😊

Reply



Michael Heribert says

September 10, 2015 at 5:35 pm

Thank you for the good article! Although i followed your steps exactly, there seems to be an issue, at least for me: When i send a create-user request, the programme creates a new user, but does not call the EmailService class so no email is sent. Is this a known issue? Thanks in advance 😊

Reply



Taiseer Joudah says

September 10, 2015 at 6:37 pm

Hi Michael, glad you liked it, I guess you are missing registering the service in AppUserManager as this **LOC**. If it is already there, check that your SendGrid Apild and Secret are correct. Hope this will help.

Reply



Michael Heribert says

September 10, 2015 at 7:50 pm

Hi, thank you for your quick response! I checked the registering of the EmailService in AppUserManager, that wasn't the problem. The credentials for the SendGridAPI are fine aswell. But by inserting Debug.writeline commands i found out that the ApplicationUserManager class is not called. I tried your downloaded your version from github and there it worked just perfectly and the output-

then you downloaded your version from github and there it worked just perfectly and the output-
commands were shown in the console. Do you know anything else that could be the problem?

Reply



mazin says

September 18, 2015 at 8:51 am

Hi Taiseer, Great article, all your articles have helped me a lot, thanks so much.

I'm still a bit confused about the emailservice, I followed your AngularJsAuthentication tutorials and so i dont have a ApplicationUserManager class or a "Create" method to plug in the the EmailService with Identity system, where should I place the body of code?

Reply



Taiseer Joudeh says

September 18, 2015 at 2:34 pm

Hi Mazin,

If you need to use the email service to send emails, then you have to user the ASP.NET Identity system and create an instance of the UserManager, and assign the "EmailService" property to your email sending logic, hope this somehow clarifies your concern.

Reply

Leave a Reply



ABOUT TAISEER



Father, MVP (ASP



.NET/IIS), Scrum
Master, Life Time
Learner

CONNECT WITH ME



Advertise Here

Advertise Here

JavaScript Diagrams



Click here for GoJS

Advertise Here

RECENT POSTS

[ASP.NET Web API Claims
Authorization with ASP.NET Identity
2.1 – Part 5](#)

[ASP.NET Identity 2.1 Roles Based
Authorization with ASP.NET Web API
– Part 4](#)

[Implement OAuth JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – Part 3](#)

[ASP.NET Identity 2.1 Accounts Confirmation, and Password Policy Configuration – Part 2](#)

[Interview with John about establishing a successful blog](#)

BLOG ARCHIVES

Select Month ▼

LEAVE YOUR EMAIL AND KEEP TUNED!

Sign up to receive email updates on every new post!

SUBSCRIBE

RECENT POSTS

TAGS

[ASP.NET Web API Claims Authorization with ASP.NET Identity 2.1 – Part 5](#)

[ASP.NET Identity 2.1 Roles Based Authorization with ASP.NET Web API – Part 4](#)

[Implement OAuth JSON Web Tokens Authentication in ASP.NET Web API and Identity 2.1 – Part 3](#)

[ASP.NET Identity 2.1 Accounts Confirmation, and Password Policy Configuration – Part 2](#)

[Interview with John about establishing a successful blog](#)

[AJAX](#) [AngularJS](#) [API](#) [API Versioning](#)

[ASP.NET](#) [Attribute Routing](#) [Authentication](#)

[Autherization Server](#) [basic authentication](#) [C#](#) [CacheCow](#)

[Client Side Templating](#) [Code First](#) [Dependency Injection](#)

[Entity Framework](#) [ETag](#) [Foursquare API](#)

[HTTP Caching](#) [HTTP Verbs](#) [IMDB API](#) [IoC](#) [Javascript](#) [jQuery](#) [JSON](#)

[JSON Web Tokens](#) [JWT](#) [Model Factory](#) [Ninject](#) [OAuth](#)

[OData](#) [Pagination](#) [Resources Association](#) [Resource Server](#)

[REST](#) [RESTful](#) [Single Page](#)

[Applications](#) [SPA](#) [Token Authentication](#)

[Tutorial](#) [Web API](#) [Web API 2](#) [Web](#)

[API Security](#) [Web Service](#) [wordpress.com](#)

[wordpress.org](#)

CONNECT WITH ME



SEARCH

Copyright © 2015 ·eleven40 Pro Theme · Genesis Framework by StudioPress · WordPress · [Log in](#)