ASP.NET Web API 2, Owin, OAuth, Bearer Token, Refresh Token with custom database (/blog/1236/asp-net-web-api-2-owin-oauth-bearer-token-refresh-token-with-custom-database)

owin bearer token authentication with web api sample token based authentication using asp.net web api 2 owin and identity how to customize authentication to my own set of tables in asp.net web api 2 web api security token example web api token authentication with a custom user database

oauth 2.0 c# web api token based authentication in web api 2 step by step

Token base authentication expires over a fixed time, to overcome on it we need to use the refresh token. We will try to create the token as well as the refresh token after successful login, refresh token will be used to generate a new token if current token is already expired and it is not too late.

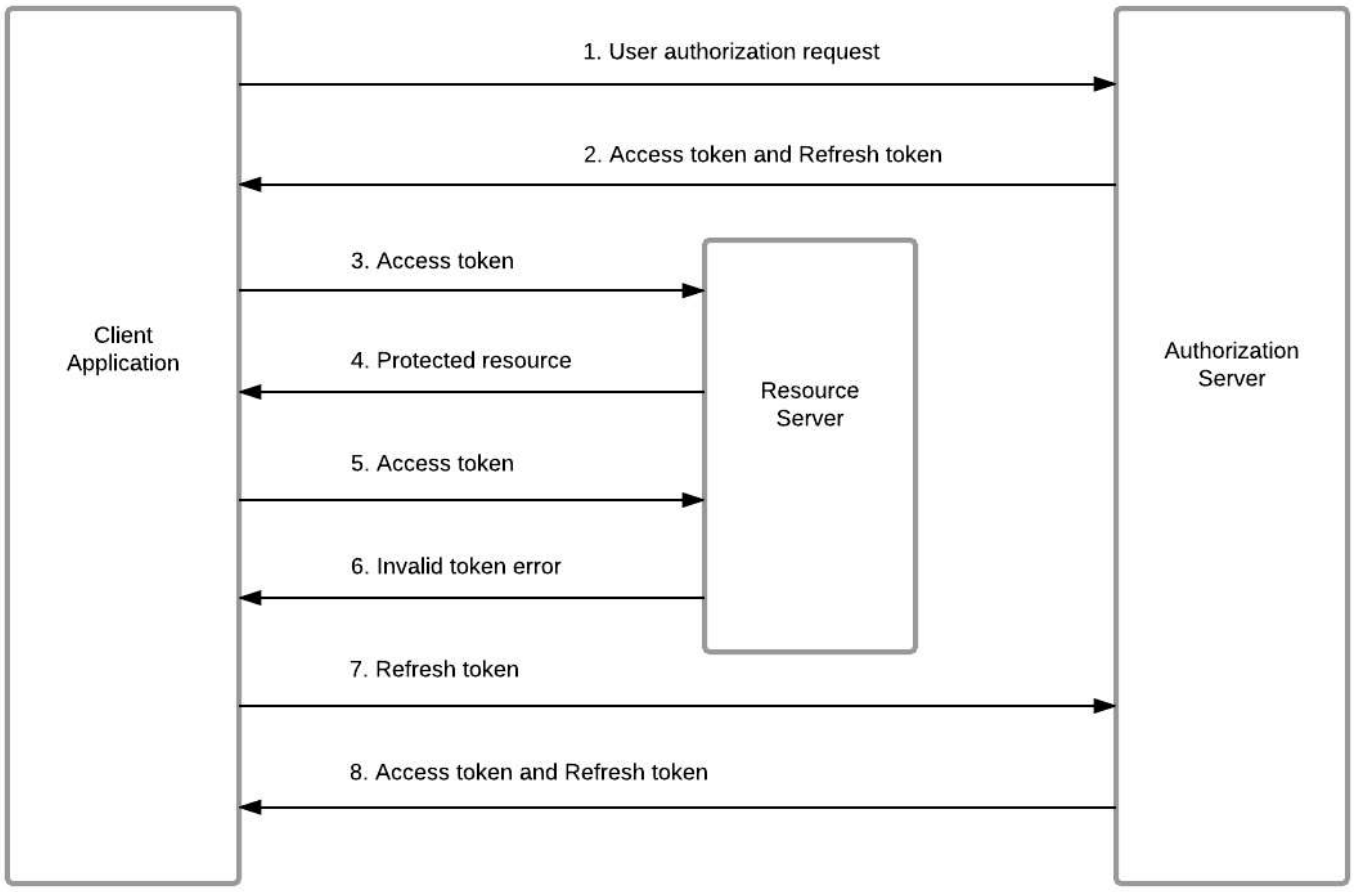
If you are at this page after reading many online articles on how to implement Owin, OAuth, Bearer Token, take a deep breath. We will complete everything step by step until we complete it.

See my previous articles on the same topic, without refresh token, which we will extend and provide complete code here:

- OAuth Web API token based authentication with custom database (http://www.advancesharp.com/blog/1216/oauth-web-api-token-based-authentication-with-custom-database)
- OAuth Web API 2 Bearer Token Role base authentication with custom database (http://www.advancesharp.com/blog/1235/oauth-web-api-2-bearer-token-role-base-authentication-with-custom-database)

How refresh token works? A Refresh Token is a special kind of token that can be used to obtain a renewed access token that allows accessing a protected resource at any time until expire.

Let's use the image to understand it:



1. User provided user id/Password to login
2. Received Token say for 20 minute and a refresh token say for 40 minutes
3. Client pass token and access protected resources
4. Successfully accessed protected resource because he have a valid token
5. After 20 minute client again try to access projected token which is already expired.
6. Client receives error 401 - Unauthorized, because his token is already expired
7. Client try to refresh the token which he received at the time of authentication
8. Client receives a new token for 20 minute and a new refresh token for 40 minute.

What will happen when user logged in and left the system for 40 or more minutes? When he will try to access the projected resources he will get 401 - Unauthorized, then try to refresh the token but that is also expired so he will again receive 401 - Unauthorized. Means he need to login again!

Now he have full picture, how token and refresh token works. Time to write the code and implement the same in logic.

Create a new ASP.Net Web Application, give any name you like, click OK, choose Web API, No Authentication and click OK to create the

- Install-Package Microsoft.Owin.Host.SystemWeb
- Install-Package Microsoft.AspNet.Identity.Owin

Now we are fully ready to write the code, create new file `Statup.cs` on the root of the project and add following code in it:

```
public partial class Startup
{
    public static OAuthAuthorizationServerOptions OAuthOptions { get; private set; }

    public void Configuration(IAppBuilder app)
    {
        OAuthOptions = new OAuthAuthorizationServerOptions
        {
            TokenEndpointPath = new PathString("/token"),
            Provider = new OAuthCustomeTokenProvider(), // We will create
            AccessTokenExpireTimeSpan = TimeSpan.FromMinutes(20),
            AllowInsecureHttp = true,
            RefreshTokenProvider = new OAuthCustomRefreshTokenProvider() // We will create
        };
        app.UseOAuthBearerTokens(OAuthOptions);
    }
}

```

Once you will copy the code, it will show some missing refrences, so put the mouse cursor on it and press CTRL + . (DOT) and add the using statements or copy the manually these

```
using Owin;
using System;
using Microsoft.Owin;
using Microsoft.Owin.Security.OAuth;

```

Even after these steps it will be showing compilation error for `OAuthCustomeTokenProvider` and `OAuthCustomRefreshTokenProvider` because we need to write these two methods.

So let's create a new class with name `OAuthCustomeTokenProvider`, we can create this class anywhere better to create a folder **Providers** and create into it. and inherite this class from `OAuthAuthorizationServerProvider`:

```
public class OAuthCustomeTokenProvider: OAuthAuthorizationServerProvider
{
    // we will add code here
}

```

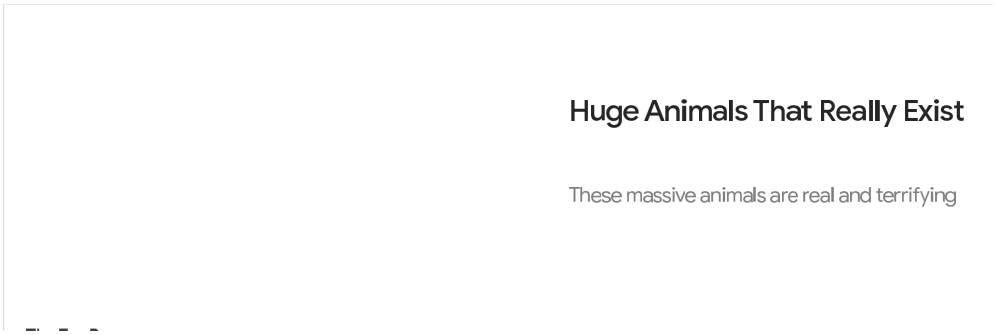
In above class override `GrantResourceOwnerCredentials` method so we can use the custom database for authentication and token creation, add following code in it:

```
public override Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
{
    return Task.Factory.StartNew(() =>
    {
        var userName = context.UserName;
        var password = context.Password;
        var userService = new UserService();
        var user = userService.Validate(userName, password);
        if (user != null)
        {
            var claims = new List<Claim>()
            {
                new Claim(ClaimTypes.Sid, Convert.ToString(user.Id)),
                new Claim(ClaimTypes.Name, user.Name),
                new Claim(ClaimTypes.Email, user.Email)
            };
            foreach (var role in user.Roles)
                claims.Add(new Claim(ClaimTypes.Role, role));

            var data = new Dictionary<string, string>
            {
                { "userName", user.Name },
                { "roles", string.Join(",", user.Roles)}
            };
            var properties = new AuthenticationProperties(data);

            ClaimsIdentity oAuthIdentity = new ClaimsIdentity(claims,
                Startup.OAuthOptions.AuthenticationType);

            var ticket = new AuthenticationTicket(oAuthIdentity, properties);
            context.Validated(ticket);
        }
        else
        {
            context.SetError("invalid_grant", "Either email or password is incorrect");
        }

    });
}

```

If you don't understand any part of the code, don't be worried, I will provide you complete detail and code at the end of this article.

As you note and will not get any code is `UserService`, this will be your service/repository/class which will be used to authenticate the user id and password.

If it is a valid user then we go ahead and add different properties and roles to create the token which can be retrieved at any time latter.



I thinks it's better to give those code, otherwise you will scratching your head, what's going on, bear with me and create this service and user model to get user detail. Create a model `user` in Models folder and add following code:

```
1. public class User
2. {
3.     public int Id { get; set; }
4.     public String Name { get; set; }
5.     public String Email { get; set; }
6.     public String Password { get; set; }
7.     public string[] Roles { get; set; }
8. }
9.
```

Nothing special, just a simple class with some properties.

Create a user service interface and a concrete User Service class anywhere or keep then a new folder say services (it's up to you) like this:

```
// IUserService interface
public interface IUserService
{
    User Validate(string email, string password);
    List<User> GetUserList();
    User GetUserById(int id);
    List<User> SearchByName(string name);
}

// UserService concrete class
public class UserService: IUserService
{

    private List<User> userList = new List<User>();
    public UserService()
    {
        for (var i = 1; i <= 10; i++)
        {
            userList.Add(new User
            {
                Id = i,
                Name = $"User {i}",
                Password = $"pass{i}",
                Email = $"user{i}@dummy.com",
                Roles = new string[] { i % 2 == 0 ? "Admin" : "User" }
            });
        }
    }

    public User Validate(string email, string password)
        => userList.FirstOrDefault(x => x.Email == email && x.Password == password);

    public List<User> GetUserList() => userList;

    public User GetUserById(int id)
        => userList.FirstOrDefault(x => x.Id == id);

    public List<User> SearchByName(string name)
        => userList.Where(x => x.Name.Contains(name)).ToList();
}

```

It is just for testing purpose, because I guess, you can create it and authenticate user from database easily.

If you feel lost, go back and see `OAuthCustomeTokenProvider` class and we added on method `GrantResourceOwnerCredentials` , fix the UserService references.

We want to use the refresh token so we need to override `GrantRefreshToken` method

```
public override Task GrantRefreshToken(OAuthGrantRefreshTokenContext context)
{
    var newIdentity = new ClaimsIdentity(context.Ticket.Identity);

    var newTicket = new AuthenticationTicket(newIdentity, context.Ticket.Properties);
    context.Validated(newTicket);

    return Task.FromResult<object>(null);
}

```

We need to override two other methods `ValidateClientAuthentication` and `TokenEndpoint` to make the authentication working properly, just do the copy and paste following code in the same class:

```
public override Task ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)
{
    if (context.ClientId == null)
        context.Validated();

    return Task.FromResult<object>(null);
}


public override Task TokenEndpoint(OAuthTokenEndpointContext context)
{
    foreach(KeyValuePair<string, string> property in context.Properties.Dictionary)
    {
        context.AdditionalResponseParameters.Add(property.Key, property.Value);
    }
    return Task.FromResult<object>(null);
}

```

If you come till this point then we are almost completed only one more class for refresh authentication token providers. So let's create a new class inside the providers folder with name `OAuthCustomRefreshTokenProvider` and inherite this class from `IAuthenticationTokenProvider`

```
public class OAuthCustomRefreshTokenProvider: IAuthenticationTokenProvider
{
    // Add a static variable
    private static ConcurrentDictionary<string, AuthenticationTicket> _refreshTokens = new ConcurrentDictionary<string, AuthenticationTicket>();

    // We will add code here
}

```

Since we inherited from `IAuthenticationTokenProvider` interface so we need to implement following methods in this class. We will use only `CreateAsync` and `ReceiveAsync` but still we need to implement Create and Receive synchronous methods, so we will throw error from them.

```
public interface IAuthenticationTokenProvider
{
    void Create(AuthenticationTokenCreateContext context);
    Task CreateAsync(AuthenticationTokenCreateContext context);
    void Receive(AuthenticationTokenReceiveContext context);
    Task ReceiveAsync(AuthenticationTokenReceiveContext context);
}

```

Create a new method into this class with name `CreateAsync` and add following code into it:

```
public async Task CreateAsync(AuthenticationTokenCreateContext context)
{
    var guid = Guid.NewGuid().ToString();
    /* Copy claims from previous token
     ********************************/
    var refreshTokenProperties = new AuthenticationProperties(context.Ticket.Properties.Dictionary)
    {
        IssuedUtc = context.Ticket.Properties.IssuedUtc,
        ExpiresUtc = DateTime.UtcNow.AddMinutes(40)
    };
    var refreshTokenTicket = await Task.Run(() => new AuthenticationTicket(context.Ticket.Identity, refreshTokenProperties));

    _refreshTokens.TryAdd(guid, refreshTokenTicket);

    // consider storing only the hash of the handle
    context.SetToken(guid);
}

```

This will be used to create a refresh token, which we will use to create a new token on expiry of current token.

Add ReceiveAsync method and add following code in it. Note it will read Authorization header and remove them because we will create a new token and refresh token:

```
public async Task ReceiveAsync(AuthenticationTokenReceiveContext context)
{
    AuthenticationTicket ticket;
    string header = await Task.Run(() => context.OwinContext.Request.Headers["Authorization"]);

    if (_refreshTokens.TryRemove(context.Token, out ticket))
        context.SetTicket(ticket);
}

```

Add Create and Receive for synchronous method, it is not needed because we are creating then asynchronously so just throw error from then, see the code:

```
public void Create(AuthenticationTokenCreateContext context)
{
    throw new NotImplementedException();
}
public void Receive(AuthenticationTokenReceiveContext context)
{
    throw new NotImplementedException();
}

```

We completed every thing, now time to fix the startup.cs missing references, open the file and add the using statements for missing namespaces or use CTRL + DOT. Once done try to build and run the application.

Let's try to create a token, if you noticed, we use the email as user[N]@dummy.com and password pass[N].

So let's try to first user, with fiddler or any Chrome Advance Rest Client:

- URL: http://localhost:50353/token (http://localhost:50353/token)
- Method: POST
- Body: userName = user1@dummy.com
- Body: password = pass1
- Body: grant_type = password

And here is the output:

```
{
  "access_token": "T-8NHXhRTAK3M-W9gJ5hBEECKOS_BA66mKFBehe5iTkwI8Y3DnGjL4tTR-IAkqszrLty1BZQS-nN2KW_NK_tpR9UhAS05C29TsISF-j682Vbxd3AEnJou
Nud7o5RLXXrGWG3EVq_dS9gzWrIkhywD0CxgKEOitqCY9QqqDMZtmKITLFKKZ08pm8Twb8OPsppTfngzWP7vlE9chwOl4WCOpcoWgG9u--CvxYs6rORnW0i_Jw3drGFQ38wVaoFF
BWj-JvM35OXly9gEdgmXcXC6rrC26kmd_obmRUq8N47WpW_x52N_CM0q76AavZvCbzamiuLKFaXx-VtFMQ0bEBUFMaCXPskJHaUeRi7rhRvdFEOGMqGQEP5e7i_I4Rx8HRB",
  "token_type": "bearer",
  "expires_in": 1199,
  "refresh_token": "c4c8a27f-807f-46fd-b502-f2c5557c59b4",
  "userName": "User 1",
  "roles": "User",
  ".issued": "Mon, 08 Oct 2018 14:57:11 GMT",
  ".expires": "Mon, 08 Oct 2018 15:17:11 GMT"
}
```

How to check whether this token is working or not? When we created our application by default Visual Studio created one controller named `ValuesController` with some action method, let's use that by add [Authorize] attribute on entire controller, so no action method will be called without authentication.

Let's try to call the Get method on the controller, first without providing any token, just by using the URL

- URL : http://localhost:50353/api/values (http://localhost:50353/api/values)
- Method: GET

We will get error saying `"Message": "Authorization has been denied for this request."`

Now let's try to pass our created Authentication Bearer Token with following values:

- URL : http://localhost:50353/api/values (http://localhost:50353/api/values)
- Method: GET
- Header: Authorization = Bearer T-8NHXhRT....I4Rx8HRB

And now we are getting values properly, it means everything working as we expected.

Now how to refresh a token by using our refresh token?

We can use the same method which we used to create the token but this time we are not going to provide the Id & password but the refresh token and change the grant_type, see this

```
- URL: http://localhost:50353/token (http://localhost:50353/token)
- Method: POST
- Body: refresh_token = c4c8a27f-807f-46fd-b502-f2c5557c59b4
- Body: grant_type = refresh_token
```

And now we will receive a new token and a new refresh token, the same method is getting called so same result will be returned but with new values.

I know, I split code to explain method by method so some of you will face problem to get the complete code in proper class, so let me copy to main methods completely as it is working. These two classes are created in Providing folder, project name I used `Refresh_Token` so you will need to fix the namespace according to your project name and namespace.

Complete code for `OAuthCustomeTokenProvider` class:

```
using System;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.Owin.Security;
using System.Collections.Generic;
using Microsoft.Owin.Security.OAuth;
using Refresh_Token.Services;

namespace Refresh_Token.Providers
{
    public class OAuthCustomeTokenProvider: OAuthAuthorizationServerProvider
    {
        #region[GrantResourceOwnerCredentials]
        public override Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)
        {
            return Task.Factory.StartNew(() =>
            {
                var userName = context.UserName;
                var password = context.Password;
                var userService = new UserService();
                var user = userService.Validate(userName, password);
                if (user != null)
                {
                    var claims = new List<Claim>()
                    {
                        new Claim(ClaimTypes.Sid, Convert.ToString(user.Id)),
                        new Claim(ClaimTypes.Name, user.Name),
                        new Claim(ClaimTypes.Email, user.Email)
                    };
                    foreach (var role in user.Roles)
                        claims.Add(new Claim(ClaimTypes.Role, role));

                    var data = new Dictionary<string, string>
                    {
                        { "userName", user.Name },
                        { "roles", string.Join(",", user.Roles)}
                    };
                    var properties = new AuthenticationProperties(data);

                    ClaimsIdentity oAuthIdentity = new ClaimsIdentity(claims,
                        Startup.OAuthOptions.AuthenticationType);

                    var ticket = new AuthenticationTicket(oAuthIdentity, properties);
                    context.Validated(ticket);
                }
                else
                {
                    context.SetError("invalid_grant", "Either email or password is incorrect");
                }
            });
        }
        #endregion

        #region[GrantRefreshToken]
        public override Task GrantRefreshToken(OAuthGrantRefreshTokenContext context)
        {
            // Change authentication ticket for refresh token requests
            var newIdentity = new ClaimsIdentity(context.Ticket.Identity);
            // newIdentity.AddClaim(new Claim("newClaim", "newValue"));

            var newTicket = new AuthenticationTicket(newIdentity, context.Ticket.Properties);
            context.Validated(newTicket);
```

```
67.
68.         #region[ValidateClientAuthentication]
69.         public override Task ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)
70.         {
71.             if (context.ClientId == null)
72.                 context.Validated();
73.
74.             return Task.FromResult<object>(null);
75.         } /**/
76.         #endregion
77.
78.         #region[TokenEndpoint]
79.         public override Task TokenEndpoint(OAuthTokenEndpointContext context)
80.         {
81.             foreach (KeyValuePair<string, string> property in context.Properties.Dictionary)
82.             {
83.                 context.AdditionalResponseParameters.Add(property.Key, property.Value);
84.             }
85.             return Task.FromResult<object>(null);
86.         }
87.         #endregion
88.     }
89. }
90.
```

And the second one for refresh token `OAuthCustomRefreshTokenProvider` complete code:

```
using System;
using System.Threading.Tasks;
using Microsoft.Owin.Security;
using System.Collections.Concurrent;
using Microsoft.Owin.Security.Infrastructure;

namespace Refresh_Token.Providers
{
    public class OAuthCustomRefreshTokenProvider: IAuthenticationTokenProvider
    {
        private static ConcurrentDictionary<string, AuthenticationTicket> _refreshTokens = new ConcurrentDictionary<string, AuthenticationTicket>();

        #region[CreateAsync]
        public async Task CreateAsync(AuthenticationTokenCreateContext context)
        {
            var guid = Guid.NewGuid().ToString();
            /* Copy claims from previous token
             *********************************/
            var refreshTokenProperties = new AuthenticationProperties(context.Ticket.Properties.Dictionary)
            {
                IssuedUtc = context.Ticket.Properties.IssuedUtc,
                ExpiresUtc = DateTime.UtcNow.AddMinutes(40)
            };
            var refreshTokenTicket = await Task.Run(() => new AuthenticationTicket(context.Ticket.Identity, refreshTokenProperties));

            _refreshTokens.TryAdd(guid, refreshTokenTicket);

            // consider storing only the hash of the handle
            context.SetToken(guid);
        }
        #endregion

        #region[ReceiveAsync]
        public async Task ReceiveAsync(AuthenticationTokenReceiveContext context)
        {
            AuthenticationTicket ticket;
            string header = await Task.Run(() => context.OwinContext.Request.Headers["Authorization"]);

            if (_refreshTokens.TryRemove(context.Token, out ticket))
                context.SetTicket(ticket);
        }
        #endregion

        #region[Create & Receive Synchronous methods]
        public void Create(AuthenticationTokenCreateContext context)
        {
            throw new NotImplementedException();
        }
        public void Receive(AuthenticationTokenReceiveContext context)
        {
            throw new NotImplementedException();
        }
        #endregion
    }
}

```

I would love to have comments to improve anything.

Soon I will use the same code to authentication Angular 6 application with complete code, like how to pass token in every HttpRequest, how to use Refresh Token if current token is expired etc.

Angular 6 Web API 2 Bearer Token Authentication add to header with HttpInterceptor (http://www.advancesharp.com/blog/1237/angular-6-web-api-2-bearer-token-authentication-add-to-header-with-httpinterceptor)

Having 13+ years of experience in Microsoft Technologies (C#, ASP.Net, MVC and SQL Server). Worked with Metaoption LLC, for more than 9 years and still with the same company. Always ready to learn new technologies and tricks.

owin oauth bearer-token web-api

By Ali Adravi (/users/100000) On 08 Oct, 18 Viewed: 16,298

Other blogs you may like

OAuth Web API 2 Bearer Token Role base authentication with custom database (/blog/1235/oauth-web-api-2-bearer-token-role-base-authentication-with-custom-database)

Create Token with user credential & roles and authorize action methods based on role in Web API is the topic we will cover in this article. We would need to pass token in every request and decorate action methods with [Authorize(Roles = "Admin, Manager") etc. that's only the code we will need to...
— By Ali Adravi On 04 Oct 2018 Viewed: 6,330

OAuth Web API token based authentication with custom database (/blog/1216/oauth-web-api-token-based-authentication-with-custom-database)

Token base authentication with custom database by using OAuth in Web API is not complicated but documents are not very clear, many people try it and ended up with scratching their head, but you are on the right page so you will not be one of them. It is not required to create an empty project...
— By Ali Adravi On 11 Sep 2017 Viewed: 17,355

Angular 6 Web API 2 Bearer Token Authentication add to header with HttpInterceptor (/blog/1237/angular-6-web-api-2-bearer-token-authentication-add-to-header-with-httpinterceptor)

Security is the main feature of any application, we will use in this article Web API 2 bearer token, created through Owin oAuth, which we created in our previous article. Pass Bearer token with every HttpRequest with the help of HttpInterceptor. In this article we will see only the authentication...
— By Ali Adravi On 14 Oct 2018 Viewed: 8,748

Angular 4 upload files with data and web api by drag & drop (/blog/1218/angular-4-upload-files-with-data-and-web-api-by-drag-drop)

Upload file with data by clicking on button or by drag and drop, is the topic of this article. We will try to create an image gallery by uploading files and save data into database and images into a folder. We will try to create a component in a way that it can be used anywhere in entire...
— By Ali Adravi On 24 Sep 2017 Viewed: 40,862

What is ASP.Net Web API, why & when to use (/blog/1213/what-is-asp-net-web-api-why-when-to-use)

What is ASP.Net Web API, what is RESTFul service, What is the difference between WCF and Web API and when to use over other. These are the main points which we going to discuss in this starting article, then we will create our own Web API project and look each and every single type of methods...
— By Ali Adravi On 13 Jul 2017 Viewed: 1,349

9 Comments Login ▾

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Share Best Newest Oldest

Mansur Haider

4 years ago

I am getting unsupported_grant_type error message while call http://localhost:62670/token with post

body : {
"userName" : "user1@dummy.com",
"password" : "pass1",
"grant_type": "password"

}

output:
{
"error": "unsupported_grant_type"
}

i followed step by step this article. please suggest me my possible mistake.
Or would be helpful if you provide code with zip file, writer promised to share code in below but he might forgot to mention the download link.

2 0 • Reply • Share ›

Страхиња Чолић → Mansur Haider

4 years ago

If u are using Postman, select x-www-form-urlencoded body when posting request. That worked for me.

0 0 • Reply • Share ›

Mansur Haider

4 years ago

very useful article, could you please share the code reference as zip project. ?

2 1 • Reply • Share ›

CM

a year ago

Thanks for the tutorial, however I have a question: how do you revoke a refresh token?
They are not stored in any table, so how would you revoke one for security reasons?

0 0 • Reply • Share ›

Ravi Chanduri

Post A Blog (/blog/postablog)	
Write A Blog (/blog/create)	
ASP.Net (/blog/articles/asp-net?tab=)	49
MVC (/blog/articles/mvc?tab=)	37
GridView (/blog/articles/gridview?tab=)	5
C# (/blog/articles/csharp?tab=)	38
DataTable (/blog/articles/datatable?tab=)	3
Ajax (/blog/articles/ajax?tab=)	9
LINQ To SQL (/blog/articles/linq-to-sql?tab=)	3
SQL Server (/blog/articles/sql-server?tab=)	32
Angularjs (/blog/articles/angularjs?tab=)	34
ReactJS (/blog/articles/reactjs?tab=)	1
Web API (/blog/articles/web-api?tab=)	5
JavaScript (/blog/articles/javascript?tab=)	13
jQuery (/blog/articles/jquery?tab=)	9
HTML (/blog/articles/html?tab=)	8
CSS (/blog/articles/css?tab=)	12
XML (/blog/articles/xml?tab=)	6
OOPs (/blog/articles/oops?tab=)	13
Web.Config (/blog/articles/web-config?tab=)	2
Routing (/blog/articles/routing?tab=)	3