

Auth

Autenticación VS Autorización

La autenticación significa validar que alguien es quien dice ser. La autorización significa que alguien tiene permiso para hacer en un recurso lo que intenta hacer.

Función Hash

Es un algoritmo determinista. Para un mismo parámetro de entrada siempre produce el mismo hash. Es muy útil para no guardar una contraseña ni en texto ni encriptada. Se guarda el hash de la contraseña y cuando el usuario decide autenticarse y envía la contraseña, se compara el hash guardado con el hash de la contraseña aportada para ver si son iguales.

El siguiente código demuestra cómo aplicar la función hash a un parámetro de entrada.

```
class Program
{
    static void Main(string[] args)
    {
        var input = "lemoncode";
        var hash = ComputeSha256(input);

        Console.WriteLine($"Input: {input}");
        Console.WriteLine($"Hash: {hash}");

        Console.WriteLine($"Validate again {input}");
        var isSameHash = hash == ComputeSha256(input);
        Console.WriteLine($"Is same hash: {isSameHash}");
    }

    public static string ComputeSha256(string input)
    {
        var inputAsBytes = Encoding.UTF8.GetBytes(input);
        using var hash = SHA256.Create();
        var bytes = hash.ComputeHash(inputAsBytes);

        var readableStringBuilder = new StringBuilder();
        foreach (var hashedByte in bytes)
        {
            readableStringBuilder.Append(hashedByte.ToString("X2"));
        }

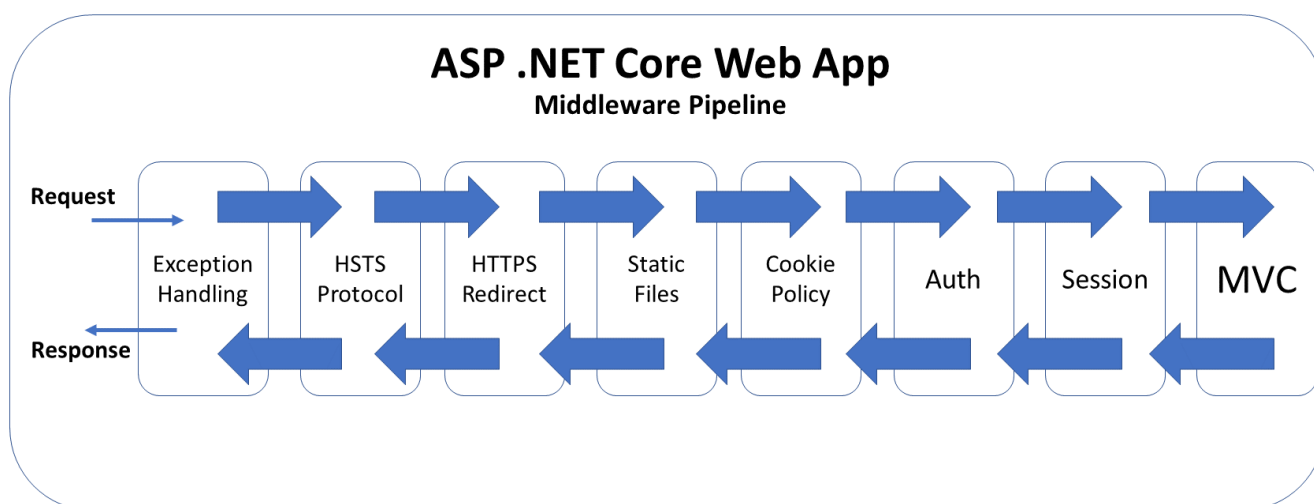
        var output = readableStringBuilder.ToString();
        return output;
    }
}
```

Autenticación Básica

La autenticación (y autorización) más básica posible es la que se puede encontrar en una aplicación web donde se comprueban las credenciales en cada petición web. Estas credenciales pueden ir en una cabecera http **Authorization** a la que se le pasa el valor **Basic** seguido de **username:password** en **base64** (ver <https://www.base64encode.org> para encodear/decodear)

```
Authorization: Basic <username:password>
```

Vamos a abrir una solución existente que ya tengamos, como por ejemplo la web api de soccer, y vamos a enchufarle un *middleware* que deje pasar una petición http solamente si posee credenciales válidas. Si recordamos, los middleware son piezas de software que se pueden encadenar y por las que van pasando (y opcionalmente transformándose) las peticiones http.



Un ejemplo de middleware básico es el siguiente

```
app.Use(async (context, next) =>
{
    // Do work that doesn't write to the Response.
    await next.Invoke();
    // Do logging or other work that doesn't write to the Response.
});
```

Otro ejemplo que no deja que la request siga hacia capas más profundas sería este

```
app.Run(async context =>
{
    await context.Response.WriteAsync("I hacked the app!");
});
```

Podemos observar lo que ocurre si colocamos este middleware al comienzo del método

`Configure(IApplicationBuilder app)`, y después ver lo que ocurre si lo colocamos al final del método después de los controladores. En el primer caso el middleware cortocircuitará la ejecución. Aunque esto no es deseable para nuestro ejemplo, se trata de una poderosa herramienta para evitar las ejecuciones innecesarias o prohibidas.

Ahora que ya comprendemos mejor el papel que tienen los *middleware*, vamos a crear uno capaz de leer la cabecera `Authorization` para comprobar las credenciales y dejar que la petición siga hacia su destino o se detenga y se devuelva un error 401 Unauthorized.

Comenzamos por una funcionalidad encargada de extraer y decodificar el contenido de una cabecera `Authorization`. Esta pequeña utilidad nos dirá si la cabecera contiene autenticación básica, y extraerá el usuario y la contraseña. Como no guarda estado, y no necesitamos abstraerla, podemos usar una clase estática dentro de nuestra capa de aplicación, ya que no tiene dependencias de terceros.

Por ejemplo, si escribimos antes los tests para practicar TDD, el método debería satisfacer los siguientes tests:

```
public static class TryGetCredentialsTests
{
    public class
    Given_A_Basic_Authorization_Header_With_Valid_Base64_Encoded_Credentials_When_Getting_Values
        : Given_When_Then_Test
    {
        private string _authHeader;
        private string _username;
        private string _password;
        private bool _result;
        private string _expectedUsername;
        private string _expectedPassword;

        protected override void Given()
        {
            _authHeader = "Basic YWRtaW46TGVTMG5Db2RlIQ=="; // Basic
            admin:Lem0nCode! in base64

            _expectedUsername = "admin";
            _expectedPassword = "Lem0nCode!";
        }

        protected override void When()
        {
            _result = AuthorizationUtil.TryGetCredentials(_authHeader, out
            _username, out _password);
        }

        [Fact]
        public void Then_It_Return_True()
        {
            _result.Should().BeTrue();
        }
    }
}
```

```
[Fact]
public void Then_It_Should_Return_The_Username_Decoded()
{
    _username.Should().Be(_expectedUsername);
}

[Fact]
public void Then_It_Should_Return_The_Password_Decoded()
{
    _password.Should().Be(_expectedPassword);
}

}

public class Given_A_Bearer_Authorization_Header_Getting_Values
    : Given_When_Then_Test
{
    private string _authHeader;
    private string _username;
    private string _password;
    private bool _result;
    private string _expectedUsername;
    private string _expectedPassword;

    protected override void Given()
    {
        _authHeader = "Bearer foo";

        _expectedUsername = string.Empty;
        _expectedPassword = string.Empty;
    }

    protected override void When()
    {
        _result = AuthorizationUtil.TryGetCredentials(_authHeader, out
_username, out _password);
    }

    [Fact]
    public void Then_It_Return_False()
    {
        _result.Should().BeFalse();
    }

    [Fact]
    public void Then_It_Should_Return_The_Username_Decoded()
    {
        _username.Should().Be(_expectedUsername);
    }

    [Fact]
    public void Then_It_Should_Return_The_Password_Decoded()
    {
        _password.Should().Be(_expectedPassword);
    }
}
```

```

    }
}
}

```

De esta forma, podemos dar la siguiente solución que satisface ambos tests.

```

public static class AuthorizationUtil
{
    public static bool TryGetCredentials(string authorizationHeader, out string
username, out string password)
    {
        var authHeaderValue =
AuthenticationHeaderValue.Parse(authorizationHeader);
        if (authHeaderValue.Scheme.Equals(AuthenticationSchemes.Basic.ToString(),
StringComparison.OrdinalIgnoreCase))
        {
            var credentials =
                Encoding.UTF8
                    .GetString(Convert.FromBase64String(authHeaderValue.Parameter
?? string.Empty))
                    .Split(new[] { ':' }, 2);
            if (credentials.Length == 2)
            {
                username = credentials[0];
                password = credentials[1];
                return true;
            }
        }

        username = string.Empty;
        password = string.Empty;
        return false;
    }
}

```

Ahora podemos crear un middleware para la pipeline. Por convención, el middleware debe tener un método llamado `Invoke` que acepte, al menos, el `HttpContext`. Este middleware extrae el contenido de la cabecera `http Authorization` y utiliza la funcionalidad anterior para comprobar si es autenticación básica válida y extraer las credenciales. Después compara las credenciales con las que acepta, y que están *hardcodeadas*, y si son válidas deja pasar la petición, pero si son incorrectas devuelve una respuesta http de error 401 `Unauthorized`.

```

public class BasicAuthMiddleware
{
    private readonly RequestDelegate _next;

    public BasicAuthMiddleware(RequestDelegate next)
    {
        _next = next;
    }
}

```

```
}

public async Task Invoke(HttpContext httpContext)
{
    var authHeader = httpContext.Request.Headers["Authorization"].ToString();
    if (string.IsNullOrEmpty(authHeader))
    {
        httpContext.Response.StatusCode = 401;
        return;
    }

    var isValidBasicAuth = AuthorizationUtil.TryGetCredentials(authHeader, out
var username, out var password);
    if (!isValidBasicAuth)
    {
        httpContext.Response.StatusCode = 401;
        return;
    }

    var isValidCredentials = username == "admin" && password == "Lem0nCode!";
    if (!isValidCredentials)
    {
        httpContext.Response.StatusCode = 401;
        return;
    }

    // otherwise all is good to continue processing the http request
    await _next(httpContext);
}
}
```

Es muy importante recordar que para acoplar este middleware en nuestra pipeline, debemos hacerlo dentro del método de `Startup.Configure(IApplicationBuilder app)`, y que el middleware debe estar antes que otros middleware en la pipeline si queremos que la comprobación suceda cuanto antes.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseMiddleware<BasicAuthMiddleware>();

    // aqui el resto de middlewares ya existentes a continuación y en el orden
    deseado
}
```

Si ejecutamos la aplicación, cualquier petición nos devolverá `401 Unauthorized`, como cabría esperar, ya que no portan autenticación básica. Compruébalo abriendo POSTMAN, u otro cliente http visual, y enviando una petición a

```
GET http://localhost:5000/swagger
```

que devolverá 401 Unauthorized.

Ahora añade una cabecera

```
Authorization: Basic YWRtaW46TG5Db2RlIQ==
```

y comprueba que ya devuelve 200 OK.

Prueba ahora a colocar el middleware DESPUÉS de swagger pero ANTES de los controladores

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseDeveloperExceptionPage();
    app.UseOpenApi();

    app.UseMiddleware<BasicAuthMiddleware>();

    // aqui el resto de middlewares ya existentes a continuación y en el orden
    deseado
}
```

Qué feo está eso de hardcodear credenciales o cualquier dato sensible en el código. Si nuestro código fuera open source todo el mundo sabría qué credenciales usamos en producción. Vamos a mejorar un poco el código y a hacer que las credenciales sean configurables. Añade lo siguiente en `appsettings.json`

```
"BasicAuthentication": {
  "Username": "admin",
  "Password": "Lem0nCode!"
}
```

Ahora inyecta `IConfiguration` en el método `Invoke` del middleware, ya que Microsoft nos dice que soporta inyección de dependencias, y lee las credenciales de las settings (o de las variables de entorno donde la aplicación esté desplegada) para comprobarlas en tiempo real.

```
public class BasicAuthMiddleware
{
    private readonly RequestDelegate _next;

    public BasicAuthMiddleware(RequestDelegate next)
    {
        _next = next;
    }

    public async Task Invoke(HttpContext httpContext, IConfiguration
configuration)
```

```
{
    var authHeader = httpContext.Request.Headers["Authorization"].ToString();
    if (string.IsNullOrEmpty(authHeader))
    {
        httpContext.Response.StatusCode = 401;
        return;
    }

    var isValidBasicAuth = AuthorizationUtil.TryGetCredentials(authHeader, out
var username, out var password);
    if (!isValidBasicAuth)
    {
        httpContext.Response.StatusCode = 401;
        return;
    }

    // Nuestras credenciales ya no están hardcodeadas
    var storedUsername = configuration.GetValue<string>
("BasicAuthentication:Username");
    var storedPassword = configuration.GetValue<string>
("BasicAuthentication:Password");

    var isValidCredentials = username == storedUsername && password ==
storedPassword;
    if (!isValidCredentials)
    {
        httpContext.Response.StatusCode = 401;
        return;
    }

    // otherwise all is good to continue processing the http request
    await _next(httpContext);
}
}
```

Un momento, si ahora nuestros endpoint están detrás de un middleware que requiere autenticación básica, aunque coloquemos el middleware detrás de la OpenApi de swagger para interactuar con nuestros endpoint, recibiremos un 401 Unauthorized cada vez que alguna petición vaya dirigida a alguno de nuestros endpoint, ¿no?

Así es, pero podemos añadir una funcionalidad en Swagger/OpenApi para que cada petición vaya acompañada de la autenticación básica que indiquemos.

Añade lo siguiente en `OpenApiExtensions.cs` dentro de las opciones de swagger.

```
c.AddSecurityDefinition(
    "basic",
    new OpenApiSecurityScheme
    {
        Name = "Authorization",
        Type = SecuritySchemeType.Http,
```



```
        Scheme = "basic",
        In = ParameterLocation.Header,
        Description = "Basic Authorization header"
    });

c.AddSecurityRequirement(
    new OpenApiSecurityRequirement()
    {
        {
            new OpenApiSecurityScheme
            {
                Reference = new OpenApiReference()
                {
                    Type = ReferenceType.SecurityScheme,
                    Id = "basic"
                }
            },
            new string[]{}
        }
    });
```

Observa cómo ahora podemos "autenticarnos" en swagger y hacer que swagger reenvíe esa cabecera http en cada una de las peticiones a cualquier endpoint.

ASP.NET Core Identity

Es una API que soporta funcionalidad de login gráfica. Controla usuarios, contraseñas, datos de perfil, roles, claims, tokens, confirmación por email y más cosas. Es lo típico para el flujo de Registrar usuarios, login y logout.

No tiene relación con Microsoft Identity Platform, que es una evolución de la plataforma de desarrollo de Azure Active Directory. Tampoco es lo mismo que Identity Server (OAuth2/OpenIdConnect).

Se puede integrar con proveedores externos como Facebook, Google, Twitter, Microsoft, etc.

Se puede utilizar con persistencia en SQL Server, Azure table storage o cualquier otro proveedor. Guarda el hash de contraseñas y también guarda datos personales y claims. No hace falta encriptar información de usuario en una cookie que se pasa de cliente a servidor.

Se utiliza para añadir una interfaz gráfica a aplicaciones web de ASP.NET Core.

1. Crea un repositorio git vacío en GitLab o GitHub, clónalo en tu máquina local y añade un .gitignore con `dotnet new gitignore`. Haz un primer push de tus cambios de la rama `main`.
2. Crea un proyecto MVC con cuentas de usuario individuales (ver <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio>). Utiliza una terminal

```
dotnet new webapp --auth Individual -uld --name Lemoncode.IdentityWeb --
output src/Lemoncode.IdentityWeb
```

```
dotnet new sln

dotnet sln add src/Lemoncode.IdentityWeb
```

Desde la consola se puede compilar toda la solución en cualquier momento con `dotnet build` y ejecutar con `dotnet run --project src/Lemoncode.IdentityWeb`

Esta template genera una aplicación web con Razor e incluye migraciones para crear una base de datos con el esquema adecuado con el proveedor SqlServer LocalDb

3. Podemos sustituir el *connection string* para que, en lugar de utilizar LocalDb, utilice Sql Server (ver Anexo I). En `appsettings.json` sustituye:

```
"ConnectionStrings": {
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-
Lemoncode.IdentityWeb-53bc9b9d-9d6a-45d4-8429-
2a2761773502;Trusted_Connection=True;MultipleActiveResultSets=true"
}
```

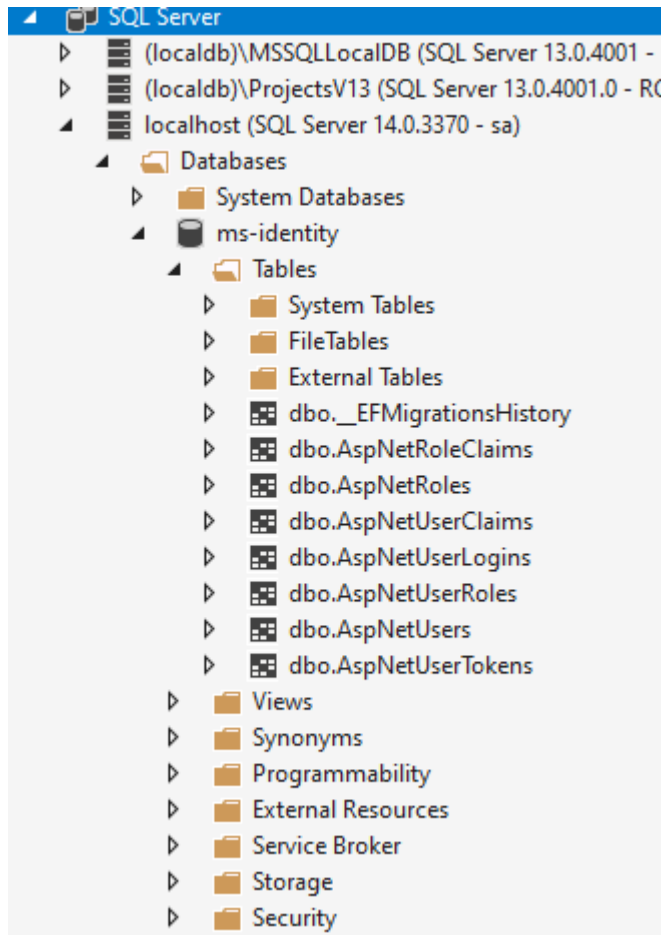
por lo siguiente

```
"ConnectionStrings": {
  "DefaultConnection": "Server=localhost;Database=ms-
identity;user=sa;password=Lem0nCode!"
}
```

4. Aplica las migraciones

```
dotnet ef database update --project src/Lemoncode.IdentityWeb --startup-
project src/Lemoncode.IdentityWeb
```

Observa que, si todo ha ido bien conectando con el servidor de SQL Server, se habrá creado una base de datos con un esquema listo para ser utilizado.



5. Ejecuta la aplicación y prueba a registrar un usuario
6. Observa la base de datos, se habrá creado un usuario en AspNetUsers, entre otras cosas. Observa también que el **EmailConfirmed** para ese usuario estará a **False** y que, por tanto, el usuario no podrá autenticarse hasta que ese valor esté a **True**. Prueba a cambiar el valor y a autenticarte.

Identity Server

Identity Server es un framework open source que podemos utilizar para crear nuestra propia aplicación capaz de autenticar, guardar identidades, emitir tokens y gestionar *single sign on*.

Es una implementación de OAuth2 y OpenID Connect (i.e: emite *id_token* y *access_token*) y soporta diferentes flujos o tipos de *grant* dependiendo de si se desea utilizar de forma interactiva con clientes que son redirigidos a una web para autenticarse, o de si se trata de una aplicación que no requiere la interacción del usuario, etc.

Identity Server tiene una web de demostración <https://demo.identityserver.io> que se puede utilizar para hacer pruebas de autenticación. También cuenta con otra web de demostración con una API <https://demo.identityserver.io/api/test> que se puede utilizar para hacer pruebas de autorización con el *access_token* conseguido en la url anterior

Si abrimos POSTMAN, o cualquier otro cliente http, podemos comprobar que la llamada a la api

```
GET https://demo.identityserver.io/api/test
```

nos devuelve un error 401 Unauthorized porque no tenemos permiso de acceso a ese recurso. Necesitaremos, como se verá más adelante, un token de acceso o *access_token* emitido por una autoridad en quien esta API confíe, y que tenga permiso de acceso a ese recurso en concreto.

También podemos enviar una petición HTTP al servidor de demostración de Identity Server para conocer el *discovery document* y ver las opciones disponibles de autenticación y solicitud de *id_token* y *access_token*

```
GET https://demo.identityserver.io/.well-known/openid-configuration
```

nos devuelve un 200 OK con

```
{
  "issuer": "https://demo.identityserver.io",
  "jwks_uri": "https://demo.identityserver.io/.well-known/openid-configuration/jwks",
  "authorization_endpoint": "https://demo.identityserver.io/connect/authorize",
  "token_endpoint": "https://demo.identityserver.io/connect/token",
  "userinfo_endpoint": "https://demo.identityserver.io/connect/userinfo",
  "end_session_endpoint": "https://demo.identityserver.io/connect/endsession",
  "check_session_iframe": "https://demo.identityserver.io/connect/checksession",
  "revocation_endpoint": "https://demo.identityserver.io/connect/revocation",
  "introspection_endpoint": "https://demo.identityserver.io/connect/introspect",
  "device_authorization_endpoint":
    "https://demo.identityserver.io/connect/deviceauthorization",
  "frontchannel_logout_supported": true,
  "frontchannel_logout_session_supported": true,
  "backchannel_logout_supported": true,
  "backchannel_logout_session_supported": true,
  "scopes_supported": [
    "openid",
    "profile",
    "email",
    "api",
    "api.scope1",
    "api.scope2",
    "scope2",
    "policyserver.runtime",
    "policyserver.management",
    "offline_access"
  ],
  "claims_supported": [
    "sub",
    "name",
    "family_name",
    "given_name",
    "middle_name",
    "nickname",
    "preferred_username",
    "profile",
    "picture",
```

```
    "website",
    "gender",
    "birthdate",
    "zoneinfo",
    "locale",
    "updated_at",
    "email",
    "email_verified"
  ],
  "grant_types_supported": [
    "authorization_code",
    "client_credentials",
    "refresh_token",
    "implicit",
    "password",
    "urn:ietf:params:oauth:grant-type:device_code"
  ],
  "response_types_supported": [
    "code",
    "token",
    "id_token",
    "id_token token",
    "code id_token",
    "code token",
    "code id_token token"
  ],
  "response_modes_supported": [
    "form_post",
    "query",
    "fragment"
  ],
  "token_endpoint_auth_methods_supported": [
    "client_secret_basic",
    "client_secret_post"
  ],
  "id_token_signing_alg_values_supported": [
    "RS256"
  ],
  "subject_types_supported": [
    "public"
  ],
  "code_challenge_methods_supported": [
    "plain",
    "S256"
  ],
  "request_parameter_supported": true
}
```

Podemos observar que la URL para solicitar tokens es <https://demo.identityserver.io/connect/token> y que hay diferentes tipos de flujo (o *grants*) y recursos (o *scopes*) soportados, entre otras cosas.

```
POST https://demo.identityserver.io/connect/token
Content-Type: application/x-www-form-urlencoded

client_id=m2m&
client_secret=secret&
grant_type=client_credentials&
scope=api
```

```
{
    "access_token":
        "eyJhbGciOiJSUzI1NiIsImtpZCI6IjQzNkI5QTU0NUYzMEl5OEMxMDNCQUY4MjYzNkMyN0FBIiwidHlwIjoieXNQrand0In0.eyJybmYiOiE2MzQ0NjEzNjgsImV4cCI6MTYzNDQ2NDk2OCwiaXNzIjoiaHR0cHM6Ly9kZW1vLm1kZW50aXR5c2VydmVyLmlvIiwiaXVkiJoiYXBpIiwiaY2xpZW50X2lkIjoibTJtIiwianRpIjoikJEREIwOUJFQjhFMkQwOUeYNjU4MEI4Mzc5NzEyQUYiLCJpYXQiOiJE2MzQ0NjEzNjgsInNjb3BlIjpBImFwaSjd0fQ.01027vH8rS8WfPrex2CJV4HWApJkBNXD676Yl6ju9FuEt8kBeheJQ6CU2SC20KQIj0Dh8wEy-DShZ8_nrhDp0Kb07oPaQKXBVhm09wrGbJot8nJOBWeBcwDb80eyUpxbaR-WizlKCCypwY03fI4cJ9uK86A-Qp9gebkbqJkyr71qyObLYSxTkB7kDW7qmiTvCvBR1lUYFXkazRfksTPmOHJ5Ltc3xIKSzYaVDuHknFYWBn9WmZuPcs2ah9DIHT4S2VZGallXGwpMzUJ9KQVS_w9rhQYW6pVerBvXK-Nh24VXD8v3D0YrSmHCS7kxHjfi9dPVmT04xlingYP1i0snA",
    "expires_in": 3600,
    "token_type": "Bearer",
    "scope": "api"
}
```

```
GET https://demo.identityserver.io/api/test  
Authorization: Bearer  
eyJhbGciOiJSUzI1NiIsImtpZCI6IjQzNkI5QTE0NUYzMEl5OEMxMDNCQUY4MjYzNmMyN0FBiwiwidHlwIjoiyXQrand0In0.eyJyYmYiOiJlE2MzQ0NjEzNjgsImV4cCI6MTYzNDQ2NDk2OCwiaXNzIjoiaHR0cHM6Ly9kZW1vLmlkZW50aXR5c2VydmVybWlviIiwiaWF0IjoiYXVkJoiYXBpIiwiaY2xpZW50X2lkIjoibTJTtIiwianRpIjoicjQkIEREREwOUjFQjhFMkQwOEYnJuU4MEI4Mzc5NzEyQUYiLCJpYXQiOiJlE2MzQ0NjEzNjgsInNjb3B1IjpjbImFwaSjdFQ.01027vH8rS8WfPrex2CJV4HWApJkBNXD676Yl6ju9FuET8kBeheJQ6CU2SC20KQIj0Dh8wEy-
```

```
DShZ8_nrhDp0Kb07oPaQKXBVhm09wrgbJot8nJOBWeBcwDb80eyUpxbaR-  
WizlKCCypwY03fI4cJ9uK86A-  
Qp9gebkbQJkyrR71qyObLYSxTkB7kDW7qmiTvCvBR1lUYFXkazRfksTPmOHJ5Ltc3xIKSzYaVDuHknFYWB  
n9WmZuPcs2ah9DIHT4S2VZGal1XGwPmzUJ9KQVS_w9rhQYw6pVerBvXK-  
Nh24VXD8v3DOYrSmHCS7kxHjfi9dPVmT04xliNGyP1i0snA
```

ahora nos devuelve un 200 OK con el siguiente contenido en el que simplemente nos muestra el contenido de nuestro token de una forma más amigable.

```
[  
  {  
    "type": "nbf",  
    "value": "1634461368"  
  },  
  {  
    "type": "exp",  
    "value": "1634464968"  
  },  
  {  
    "type": "iss",  
    "value": "https://demo.identityserver.io"  
  },  
  {  
    "type": "aud",  
    "value": "api"  
  },  
  {  
    "type": "client_id",  
    "value": "m2m"  
  },  
  {  
    "type": "jti",  
    "value": "BBDDb09BEB8E2D09A26580B8379712AF"  
  },  
  {  
    "type": "iat",  
    "value": "1634461368"  
  },  
  {  
    "type": "scope",  
    "value": "api"  
  }  
]
```

Anexo I - SQL Server con Docker

Crea un contenedor de SqlServer para tener un servidor de base de datos corriendo en local

```
docker run -e "ACCEPT_EULA=Y" \  
-e "SA_PASSWORD=Lem0nCode!" \  
-e "MSSQL_PID=Express" \  
-p 1433:1433 \  
--name sqlserver \  
-d mcr.microsoft.com/mssql/server:2017-latest-ubuntu
```

Asegúrate de que el contenedor extá en ejecución con

```
docker ps
```

Si ya tenías ese contenedor (u otro), puedes simplemente arrancarlo

```
docker start sqlserver
```

Opcionalmente instala descarga e instala Azure Data Studio para tu sistema operativo

<https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio?view=sql-server-ver15>