**Asp.Net Web API – Token Based Authentication**

TARİH: EYLÜL 2, 2015 | YAZAR: GÖKHAN GÖKALP

Merhaba arkadaşlar, bu makalemde Asp.Net Web API ile RESTful servis geliştirirken **Token Based** bir **Authentication** işlemi nasıl yapıldığına dair örnek bir proje yapacağız.

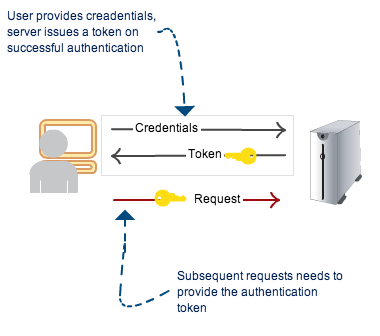
RESTful’ün önemini kısaca hatırlamak gerekirse:

* Fazlasıyla basit ve esneklik sağlamaktadır.
* REST’in **HTTP**protokolü üzerine kurulmuş olmasıyla beraber günümüz modern web dünyasındaki bir çok uygulamalar kendini browser tabanlı uygulamalara bırakıyor ve artık bir çok işlem client-side tabanlı yapıldığı için REST servisleri bize bir artı daha sağlıyor bu anlamda.

Özünde **RESTful** servisleri bize **client-server** arasındaki yapacak olduğumuz veri transferini **SOAP** veya **RPC** gibi kompleks mimariler yerine daha hafif ve esnek bir şekilde yapabilme olanağı sağlıyor.

REST mimarisini hatırladığımıza göre, gelelim şimdi nedir bu **Token Based** **Authentication?**

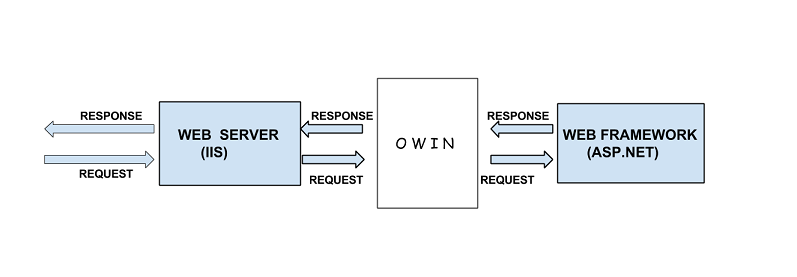
Günümüz çağında geliştirilen neredeyse tüm uygulamaların bir mobil bacağı bulunmaktadır veya mobil tarafına da destek verebilecek şekilde servis mimarileri geliştirilmektedir. REST mimarisi üzerine kurulan bir serviste ise güvenlik işlemlerini ele alabilmek, **client’ı yetkilendirebilmek** için **Token** (Jeton) bazlı bir yetkilendirme işlemi yapılmaktadır.

**Token Based Authentication** işleminin yaşam döngüsüne bakmak istediğimizde ise:[](file:///E:\Asp.Net%20Web%20API%20–%20Token%20Based%20Authentication%20_%20Gökhan%20Gökalp_files\securitypattern.png)

* Client kendi güvenlik bilgilerini girer ve bu bilgiler**Authorization Server**‘a gönderilir
* Authorization Server bu bilgileri doğrulursa, client’a bir **Access Token** Http Response’u döner.
* Client artık erişmek istediği servislere, elde etmiş olduğu Access Token’ı  Http Request’in **Authorization Header**‘ına ekleyerek erişim sağlar.

Bu ön bilgilerden sonra hemen örneğimize geçelim. Öncelikle örneğimizde **Authentication**işlemleri için **OAuth 2.0** protokolü ile sağlayacağız ve bunun için Microsoft’un **Owin** kütüphanesinden yararlanacağız.

Owin temelinde **IIS** ile **Application** arasında kendi pipeline’ını kuruyor ve işlemleri burada handle ediyor. Lightweight bir pipeline’a sahiptir.

[](file:///E:\Asp.Net%20Web%20API%20–%20Token%20Based%20Authentication%20_%20Gökhan%20Gökalp_files\OWIN__2.png)

**AspNetWebAPIOAuth** isminde bir **Asp.Net Web Application** oluşturuyorum. Oluştururken Template kısmından **Empty** seçip Core Referansını ise **Web API** seçerek tamamlıyorum.

Projemizi oluşturduğumuza göre hemen projemiz üzerine sağ tıklayarak NuGet Package Manager’ı açıp Search kısmından **OAuth** yazarak çıkacak olan sonuçlar içinden**Microsoft.AspNet.WebApi.Owin, Microsoft.Owin.Host.SystemWeb**ve **Microsoft.Owin.Security.OAuth**‘u seçerek projemize kuruyoruz.

Proje içerisine **OAuth** isimli bir klasör ekleyerek servis çalışmaya başlarken **Owin**pipeline’ını ayağa kaldırabilmek için **Startup** sınıfını hazırlamaya başlıyoruz ve içerisinde gerekli konfigürasyon ayarlarını **WebApiConfig**‘e register edip, **Owin Server** üzerinde uygulama oluşurken kullanacağı konfigürasyon ayarınıda belirtiyoruz.

Startup.cs:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | using AspNetWebAPIOAuth.OAuth.Providers;  using Microsoft.Owin;  using Microsoft.Owin.Security.OAuth;  using Owin;  using System;  using System.Web.Http;    [assembly: OwinStartup(typeof(AspNetWebAPIOAuth.OAuth.Startup))]  namespace AspNetWebAPIOAuth.OAuth  {      // Servis çalışmaya başlarken Owin pipeline'ını ayağa kaldırabilmek için Startup'u hazırlıyoruz.      public class Startup      {          public void Configuration(IAppBuilder appBuilder)          {              HttpConfiguration httpConfiguration = new HttpConfiguration();                ConfigureOAuth(appBuilder);                WebApiConfig.Register(httpConfiguration);              appBuilder.UseWebApi(httpConfiguration);          }            private void ConfigureOAuth(IAppBuilder appBuilder)          {              OAuthAuthorizationServerOptions oAuthAuthorizationServerOptions = new OAuthAuthorizationServerOptions()              {                  TokenEndpointPath = new Microsoft.Owin.PathString("/token"), // token alacağımız path'i belirtiyoruz                  AccessTokenExpireTimeSpan = TimeSpan.FromDays(1),                  AllowInsecureHttp = true,                  Provider = new SimpleAuthorizationServerProvider()              };                // AppBuilder'a token üretimini gerçekleştirebilmek için ilgili authorization ayarlarımızı veriyoruz.              appBuilder.UseOAuthAuthorizationServer(oAuthAuthorizationServerOptions);                // Authentication type olarak ise Bearer Authentication'ı kullanacağımızı belirtiyoruz.              // Bearer token OAuth 2.0 ile gelen standartlaşmış token türüdür.              // Herhangi kriptolu bir veriye ihtiyaç duymadan client tarafından token isteğinde bulunulur ve server belirli bir expire date'e sahip bir access\_token üretir.              // Bearer token üzerinde güvenlik SSL'e dayanır.              // Bir diğer tip ise MAC token'dır. OAuth 1.0 versiyonunda kullanılıyor, hem client'a, hemde server tarafına implementasyonlardan dolayı ek maliyet çıkartmaktadır. Bu maliyetin yanı sıra ise Bearer token'a göre kaynak alış verişinin biraz daha güvenli olduğu söyleniyor çünkü client her request'inde veriyi hmac ile imzalayıp verileri kriptolu bir şekilde göndermeleri gerektiği için.              appBuilder.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions());          }      }  } |

Owin ayarlarını başlangıçta içeren sınıfımızı oluşturduk. Sınıf satırlarındaki yorumlarda da belirttiğimiz üzere, Authentication type olarak **Bearer Authentication** kullanacağız. Sebebi ise daha fazla lightweight olup OAuth 2.0 ile standart bir hale gelmesi ve hem client hemde server side için authentication işlemlerini daha fazla kolaylaştırmasıdır. Ayrıca tüm işlemler her ne kadar bir access token üzerinden yürüyecek olsada, **SSL** ile client ile server arasındaki veri güvenliği sağlanmalıdır.

**OAuthAuthorizationServerOptions** ayarlarını tanımlarken **Provider**olarak **OAuthAuthorizationServerProvider**sınıfından miras alarak türeteceğimiz **SimpleAuthorizationServerProvider** ‘ı seçtik. Şimdi gelelim bu provider’ın kodlarını incelemeye. Öncesinde daha önce açtığımız OAuth klasörünün içine hemen bir **Providers** isminde klasör daha açarak içerisinde ilgili sınıfımızı oluşturuyoruz.

SimpleAuthorizationServerProvider.cs:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | using Microsoft.Owin.Security.OAuth;  using System.Threading.Tasks;  using System.Security.Claims;    namespace AspNetWebAPIOAuth.OAuth.Providers  {      public class SimpleAuthorizationServerProvider : OAuthAuthorizationServerProvider      {          // OAuthAuthorizationServerProvider sınıfının client erişimine izin verebilmek için ilgili ValidateClientAuthentication metotunu override ediyoruz.          public override async System.Threading.Tasks.Task ValidateClientAuthentication(OAuthValidateClientAuthenticationContext context)          {              context.Validated();          }            // OAuthAuthorizationServerProvider sınıfının kaynak erişimine izin verebilmek için ilgili GrantResourceOwnerCredentials metotunu override ediyoruz.          public override async Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsContext context)          {              // CORS ayarlarını set ediyoruz.              context.OwinContext.Response.Headers.Add("Access-Control-Allow-Origin", new[] { "\*" });                // Kullanıcının access\_token alabilmesi için gerekli validation işlemlerini yapıyoruz.              if (context.UserName == "Gokhan" && context.Password == "123456")              {                  var identity = new ClaimsIdentity(context.Options.AuthenticationType);                    identity.AddClaim(new Claim("sub", context.UserName));                  identity.AddClaim(new Claim("role", "user"));                    context.Validated(identity);              }              else              {                  context.SetError("invalid\_grant", "Kullanıcı adı veya şifre yanlış.");              }          }      }  } |

**OAuthAuthorizationServerProvider**sınıfının iki metodunu ezdiğimizi görüyoruz. Bunlardan birincisi, Client’ı doğrulamak için ki direkt olarak doğruladık biz. İkincisi ise asıl kaynak erişimine verilecek yetkilerin ayarlandığı ana kısım. Öncelikle burada CORS ayarlarını gerçekeleştirdik. Hemen **CORS** nedir hatırlatması yapmak gerekirse:

**CORS** domain’ler arası kaynak paylaşımını sağlamaya yarayan bir mekanizmadır. Bir domain’in bir başka domain’in kaynağını kullanabilmesini sağlar.

Hızlıca CORS’u da tekrardan hatırladığımız üzere devamında koda baktığımızda da net bir şekilde görüldüğü gibi **validation** işlemlerini gerçekleştiriyoruz. Eğer kullanıcı geçerli bir kullanıcı ise bir kimlik yaratıp, **context** üzerinde doğruluyor.

Evet şuan Owin için OAuth 2.0 implementasyonunu gerçekleştirmiş bulunuyoruz. Şimdi gelelim **Controller** üzerinde ki kullanımına. Hemen Controllers kısmına OrdersController ekliyorum ve içine List isminde bir metot tanımlıyorum. Form Authentication’dan da hatırlayabileceğiniz üzere metotların üstüne attirbute olarak [**Authorize**] attributunu ekliyorduk, Owin içinde aynı attribut’u kullanıyoruz.

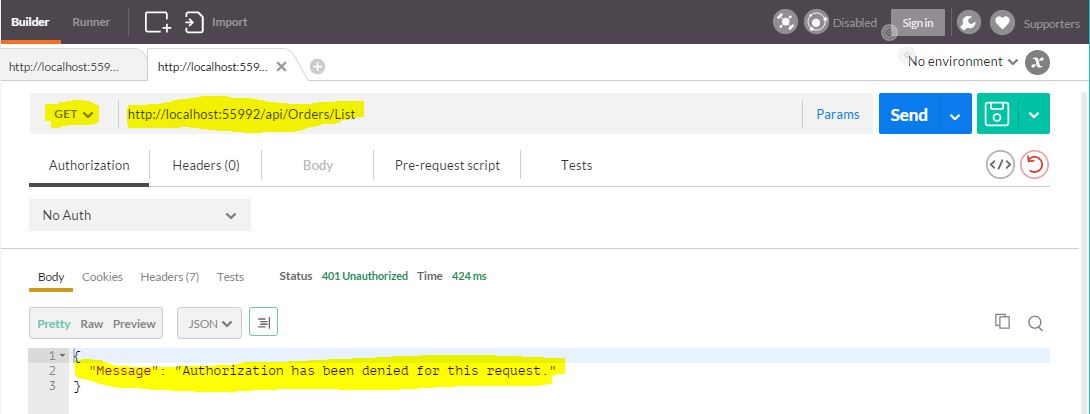
OrdersController.cs:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | using System.Collections.Generic;  using System.Web.Http;    namespace AspNetWebAPIOAuth.Controllers  {      public class OrdersController : ApiController      {          [HttpGet]          [Authorize]          public List<string> List()          {              List<string> orders = new List<string>();                orders.Add("Elma");              orders.Add("Armut");              orders.Add("Erik");                return orders;          }      }  } |

Api tarafında herşey hazır olduğuna göre projemizi test edebiliriz. Ben tool olarak Postman’ı tercih ediyorum siz isterseniz Fiddler Composer’da kullanabilirsiniz. Postman data gönderirken bana daha fazla esneklik sağlıyor açıkcası. 🙂

Öncelikle direkt olarak ilgili api metodumuza erişmeye çalıştığımızda alacağımız sonuca bir bakalım:

[](file:///E:\Asp.Net%20Web%20API%20–%20Token%20Based%20Authentication%20_%20Gökhan%20Gökalp_files\oauth_hata.jpg)

**api/Orders/List** url’i ile GET isteği attığımızda Authorization hatası aldığımızı görüyoruz. Öncelikle**/token** path’i ile belirttiğimiz adrese gidip geçerli bir **access\_token** almalıyız.

Bunun için **POST** tipinde**/token** url’ine Headers’a ve Body’e bir kaç parametre set ederek gitmemiz gerekmektedir.

Headers’e eklenecek parametreler:

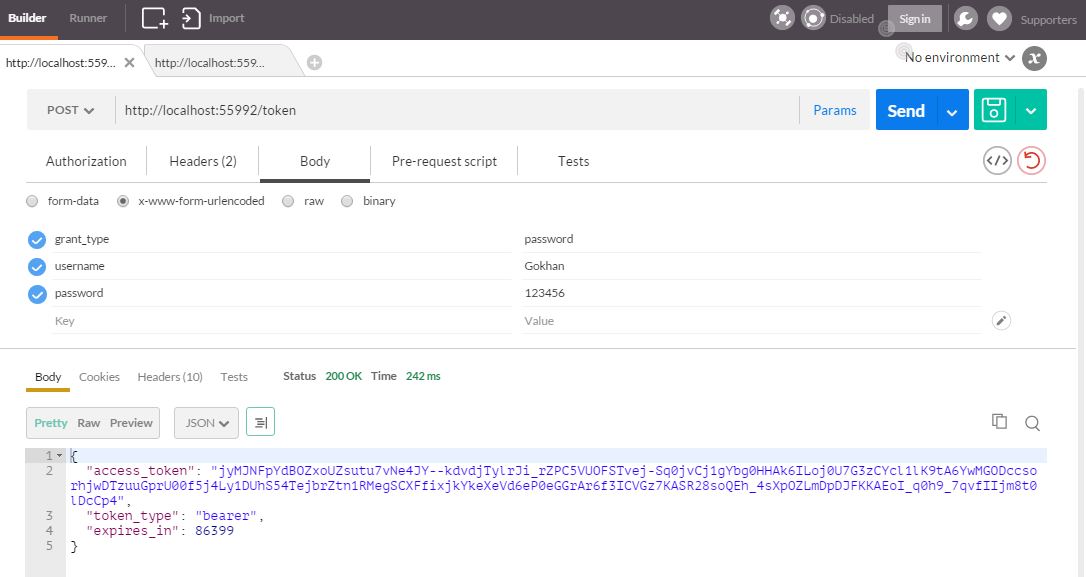
**Header**: Accept                **Value**: application/json  
**Header**: Content-Type     **Value**: application/x-www-form-urlencoded

Body’e eklenecek parametreler:

data tipi x-www-form-urlencoded olarak seçilip,

**Key**: grant\_type                 **Value**: password  
**Key**: username                  **Value**: Gokhan (Kullanıcı adınız)  
**Key**: password                   **Value**: 123456 (Şifreniz)

İlgili bilgileri girdikten sonra POST işlemini gerçekleştirelim ve gelen sonuca bakalım:

[](file:///E:\Asp.Net%20Web%20API%20–%20Token%20Based%20Authentication%20_%20Gökhan%20Gökalp_files\oauth_token.jpg)

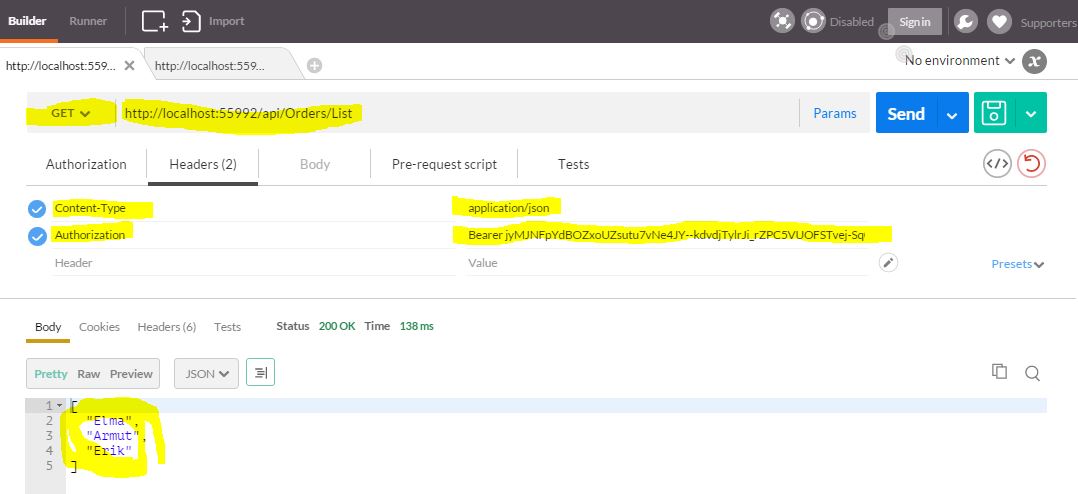
Geriye dönen JSON sorgusunda **access\_token** oluşmuş ve expires\_in süresi ile geldiğini görüyoruz. Bu süreyi hatırlarsak Startup kısmında konfigürasyon bölümünde **AccessTokenExpireTimeSpan** propertysi ile vermiştik.

Artık bu token’ı kullanarak tekrardan **api/Orders/List** url’ine tekrardan bir GET sorgusunda bulunalım. Fakat bu sefer ilgili token’ı Header’a ekleyerek gönderiyoruz.

Headers’e eklenecek parametreler:

**Header**: Content-Type      **Value**: application/json  
**Header**: Authorization       **Value**: Bearer jyMJNFpYdBOZxoUZsutu7vNe4JY–kdvdjTylrJi\_rZPC5VUOFSTvej-Sq0jvCj1gYbg0HHAk6ILoj0U7G3zCYcl1lK9tA6YwMGODccsorhjwDTzuuGprU00f5j4Ly1DUhS54TejbrZtn1RMegSCXFfixjkYkeXeVd6eP0eGGrAr6f3ICVGz7KASR28soQEh\_4sXpOZLmDpDJFKKAEoI\_q0h9\_7qvfIIjm8t0lDcCp4

Token tipimiz **Bearer** olduğu için headerda Authorization kısmının değerine access\_token’ı girmeden önce Bearer tag’ini ekleyip daha sonrasında access\_token’ı ekliyoruz.

[](file:///E:\Asp.Net%20Web%20API%20–%20Token%20Based%20Authentication%20_%20Gökhan%20Gökalp_files\oauth_access.jpg)

Servisten başarıyla bilgileri çektiğimizi görüyoruz. Bir sonraki Web API konumda ise Custom Token Based Authentication işlemi nasıl gerçekleştirilebilir hakkında bir şeyler yazmayı planlıyorum. Şimdilik sağlıcakla kalın.