



CHARLOTTE TECH TALKS

Authentication Using Tokens for AngularJS, OWIN, ASP.NET Web API & Identity

Mark A. Wilson

Senior Developer

MarkW@LogicalAdvantage.com

www.DeveloperInfra.com

@DeveloperInfra

Mark A. Wilson

www.developerinfra.com

@DeveloperInfra

Software craftsman,
consultant and agile
.NET/JavaScript web
developer. User group
leader and event planner.
Loving husband, dog
foster, and Disney
aficionado.



Logical Advantage

www.logicaladvantage.com

Our mission is to partner with our clients to provide strategies and solutions that maximize the ROI of Enterprise Asset Intelligence and Human Capital Development.



Charlotte, NC

**Enterprise Developers
Guild**

www.developersguild.org

Spark Conference

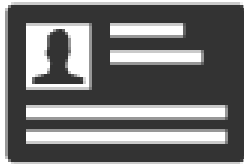
www.sparkconf.org



Authentication Using Tokens for AngularJS, OWIN, ASP.NET Web API & Identity



Authentication



Who You Are



Authentication

Cookie-Based

Pros

- Decades-Old & Widely Used
- Default for New Projects

Cons

- Man-In-The-Middle (MITM)
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

Token-Based

Pros

- Stateless & Scalable Servers
- Mobile Friendly
- Pass Authentication To Other Applications
- Extra Security

Cons

- Cross-Site Scripting (XSS)

- <https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication>
- <http://sittr.us/2011/08/26/cookies-are-bad-for-you.html>



Authentication

Cookie-Based

Pros

- Decades-Old & Widely Used
- Existing Server App Support

Cons

- Man-In-The-Middle (MITM)
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

Token-Based

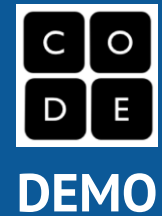
Pros

- Stateless & Scalable Servers
- Mobile Friendly
- Cross-Domain / Cross-Origin Resource Sharing (CORS) AJAX
- Decoupled Token Generation
- CSRF Protected
- CDN Asset Hosting
- Improved Performance
- Standard-based JWT

• <https://auth0.com/blog/2014/01/07/angularjs-authentication-with-cookies-vs-token/>



Token-Based Authentication



Demo

Code based on the online tutorial, "Token Based Authentication using ASP.NET Web API 2, Owin, and Identity" by Taiseer Joudeh, MVP.

token/2014/06/01/token-based-authentication-asp-net-web-api-2-owin-asp-net-identity/

BIT OF TECHNOLOGY ARCHIVE ABOUT ME SPEAKING CONTACT

Microsoft 17 languages. 5 weeks to market. The Dyson 360 Eye™ robot website. Watch the free webinar

Token Based Authentication using ASP.NET Web API 2, Owin, and Identity

June 1, 2014 By Taiseer Joudeh — 722 Comments

Be Social! Share!

Facebook (217) Twitter (231) LinkedIn (68) Google+ (41) Tumblr Email

Last week I was looking at the top viewed posts on my blog and I noticed that visitors are interested in the authentication part of ASP.NET Web API, CORS Support, and how to authenticate users in single page applications built with AngularJS using token based approach.

So I decided to compile mini tutorial of three five posts which covers and connects those topics. In this tutorial we'll build SPA using AngularJS for the front-end, and ASP.NET Web API 2, Owin middleware, and ASP.NET Identity for the back-end.

- AngularJS Token Authentication using ASP.NET Web API 2, Owin, and ASP.NET Identity – Part 2.
- Enable OAuth Refresh Tokens in AngularJS App using ASP.NET Web API 2, and Owin – Part 3.
- ASP.NET Web API 2 external logins with Facebook and Google in AngularJS app – Part 4.
- Decouple OWIN Authorization Server from Resource Server – Part 5.

- New post: Secure ASP.NET Web API 2 using Azure Active Directory, Owin Middleware, and ADAL.
- New post: Two Factor Authentication in ASP.NET Web API & AngularJS using Google Authenticator.

The demo application can be accessed on (<http://ngAuthenticationWeb.azurewebsites.net/>). The back-end API can be accessed on (<http://ngAuthenticationAPI.azurewebsites.net/>) and both are hosted on Microsoft Azure, for learning purposes feel free to integrate and play with the back-end API with your front-end application. The API supports CORS and accepts HTTP calls from any origin. You can check the [source code](#) for this tutorial on Github.

Home Login Sign Up

AngularJS Authentication

This single page application is built using AngularJS, it is using OAuth bearer token authentication, ASP.NET

JavaScript Diagrams

ABOUT TAISEER Father, MVP (ASP.NET/IIS), Scrum Master, Life Time Learner

CONNECT WITH ME

Microsoft MVP Most Valuable Professional

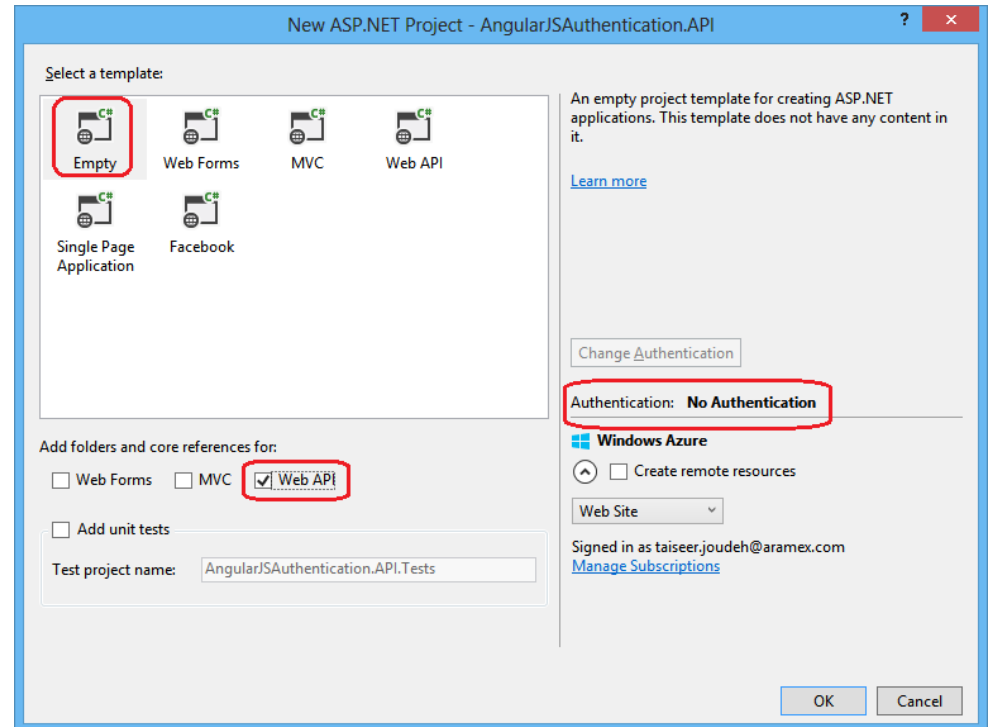
HILTON HHONORS EARN DOUBLE POINTS at over 3,600 hotels OR DOUBLE MILES with over 40 airline partners BOOK NOW

• <http://bitoftech.net/2014/06/01/token-based-authentication-asp-net-web-api-2-owin-asp-net-identity/>



Lightweight Solutions & Projects

Less is more.



NuGet Packages

The magical number 7.

Microsoft.AspNet.WebApi

Microsoft.AspNet.WebApi.Owin

Microsoft.Owin.Host.SystemWeb

Microsoft.Owin.Security.OAuth

Microsoft.Owin.Cors

*Microsoft.AspNet.Identity.Owin

*Microsoft.AspNet.Identity.EntityFramework

*Authentication API Project Only



Example Token

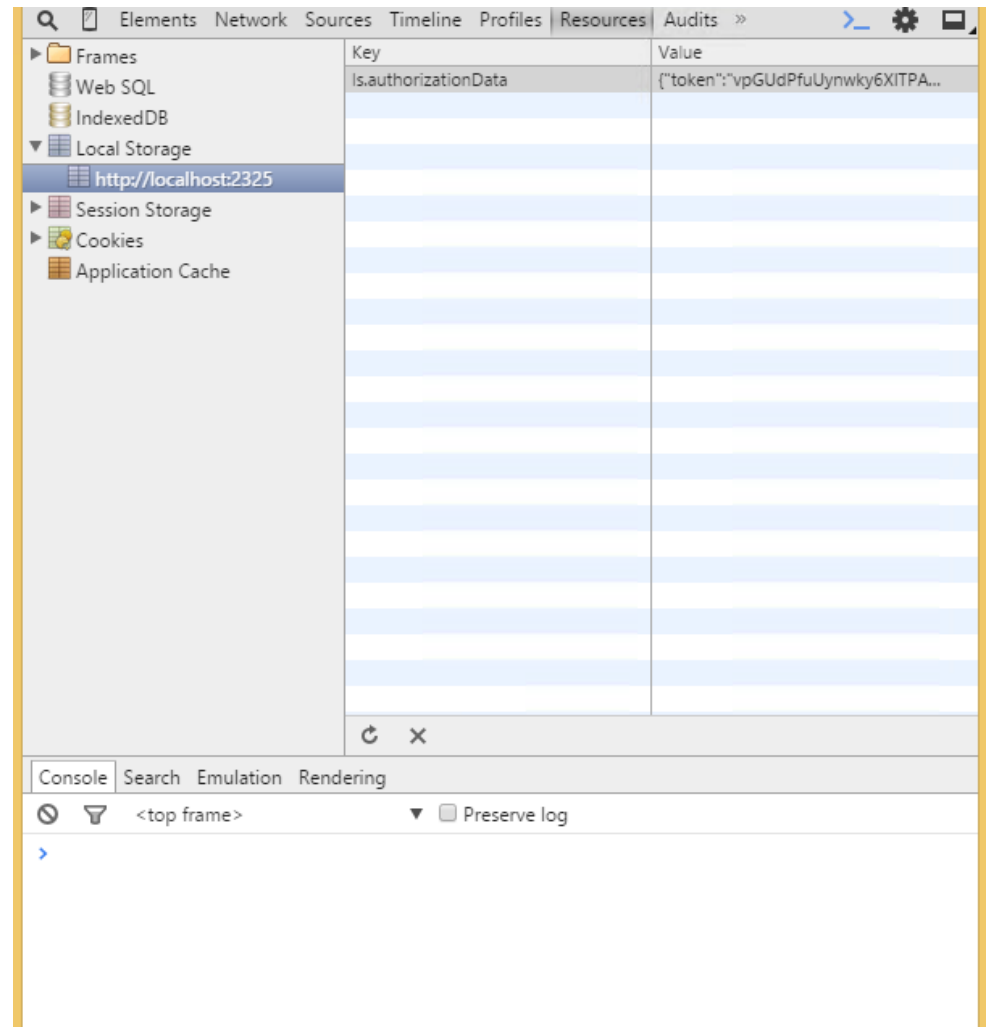
It is JSON.

```
1 {  
2   "access_token": "QKx13H6e4yW3rq3FvGNh_kLD  
KvuRWESgq61lHjQRqPt_xsQ8DmZK4SuTbOwkneIBMZT  
K3iyhCAeVtY7kxpfOKF9xYHN4g4bhtBeTsEuDF1_xBR  
dtIDi_eRtek-pJHG6_ku-8xf30z0MNZm-Okk4IziRQC  
QWMontmB-ae9MV5uHU_UfoFqtpERrvOeeeTnAZAPIA  
Z4ih9HMLj6762j3joHRvHYAdhFW06ja4Ud848IQIfDJ  
ZAs6_8R1zN5xchoXxcvpEd_mCtc1F54FStrpj2qRVpS  
NjurdnvBGV8dX46eWeU-Aj-jM4mOdAL058wOtDw1HX_  
gE5MwgYjUkivNJ6VpNHYhto-OpDtJ0UwoRBAANOow7W  
BT_L-uKFeNJJeJzyyRMEDVwAAw3fh2YWSBY34JQSN6ZK  
RK3mKyC8WQq8yJeuHVnkvae2IMwRKBjg-0BNETN7luM  
Av2qIcDcdX0xZl5wLgzfC_ATin-Qkx7WgssQHUyuUQD  
xC4Gv8Uc4TH3pbbE6Irp_Lwx2lpNfDf-0ezDHf  
-eUNRmNmyFDefcHr5DCBYogPQ02GiDH8yvSfPsNrQnu  
UGUw",  
3   "token_type": "bearer",  
4   "expires_in": 86400,  
5   ".issued": "1/1/2015 12:00:00 AM +00:00",  
6   ".expires": "1/2/2015 12:00:00 AM +00:00"  
7 }
```



Local Storage

Other options include JavaScript or a Cookie.



AngularJS Service

Move along, nothing to
see here.

```
sService.js  ✎ ✕
1  'use strict';
2  app.factory('ordersService', ['$http', 'ngAuthSettings', function ($http, ngA
3
4      var serviceBase = ngAuthSettings.apiServiceBaseUri;
5
6      var ordersServiceFactory = {};
7
8      var _getOrders = function () {
9
10         return $http.get(serviceBase + 'api/orders').then(function (results)
11             return results;
12         });
13     };
14
15     ordersServiceFactory.getOrders = _getOrders;
16
17     return ordersServiceFactory;
18
19 }]);
```



AngularJS Interceptor

Pre- and post-processing.

```
nterceptorService.js x
1 'use strict';
2 app.factory('authInterceptorService', ['$q', '$injector', '$location', 'localStorageService',
3
4     var authInterceptorServiceFactory = {};
5     var $http;
6
7     var _request = function (config) {
8
9         config.headers = config.headers || {};
10
11         var authData = localStorageService.get('authorizationData');
12         if (authData) {
13             config.headers.Authorization = 'Bearer ' + authData.token;
14         }
15
16         return config;
17     }
18
19     var _responseError = function (rejection) {
20         var deferred = $q.defer();
21         if (rejection.status === 401) {
22             var authService = $injector.get('authService');
23             authService.refreshToken().then(function (response) {
24                 _retryHttpRequest(rejection.config, deferred);
25             }, function () {
26                 authService.logout();
27                 $location.path('/login');
28                 deferred.reject(rejection);
29             });
30         } else {
31             deferred.reject(rejection);
32         }
33         return deferred.promise;
34     }
35
36     var _retryHttpRequest = function (config, deferred) {
37         $http = $http || $injector.get('$http');
38         $http(config).then(function (response) {
```



ASP.NET Web API Controller

You shall not pass!

```
ctedController.cs  x
larJSAuthentication.ResourceServer - AngularJSAuthentication.ResourceServer - Get()
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Net;
5 using System.Net.Http;
6 using System.Security.Claims;
7 using System.Web.Http;
8
9 namespace AngularJSAuthentication.ResourceServer.Controllers
10 {
11     [Authorize]
12     [RoutePrefix("api/protected")]
13     public class ProtectedController : ApiController
14     {
15         [Route("")]
16         public IEnumerable<object> Get()
17         {
18             var identity = User.Identity as ClaimsIdentity;
19
20             return identity.Claims.Select(c => new
21             {
22                 Type = c.Type,
23                 Value = c.Value
24             });
25         }
26     }
27 }
```



Shared MachineKey

Do not use the same keys for dev & prod.

```
config • X
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <!--
4   For more information on how to configure your ASP.NET application, please v
5   http://go.microsoft.com/fwlink/?LinkId=301879
6   -->
7 <configuration>
8   <configSections>
9     <section name="entityFramework" type="System.Data.Entity.Internal.ConfigF
10
11   <!-- For more information on Entity Framework configuration, visit 
```



Demo

Code based on the online tutorial, "Token Based Authentication using ASP.NET Web API 2, Owin, and Identity" by Taiseer Joudeh, MVP.

bitoftech.net/2014/06/01/token-based-authentication-asp-net-web-api-2-owin-asp-net-identity/

BIT OF TECHNOLOGY ARCHIVE ABOUT ME SPEAKING CONTACT

Microsoft 17 languages. 5 weeks to market. The Dyson 360 Eye™ robot website. Watch the free webinar

Token Based Authentication using ASP.NET Web API 2, Owin, and Identity

June 1, 2014 By Taiseer Joudeh — 722 Comments

Be Socialable, Share!

Facebook (217) Twitter (231) LinkedIn (68) Google+ (41) Tumblr Email

Last week I was looking at the top viewed posts on my blog and I noticed that visitors are interested in the authentication part of ASP.NET Web API, CORS Support, and how to authenticate users in single page applications built with AngularJS using token based approach.

So I decided to compile mini tutorial of three five posts which covers and connects those topics. In this tutorial we'll build SPA using AngularJS for the front-end, and ASP.NET Web API 2, Owin middleware, and ASP.NET Identity for the back-end.

- AngularJS Token Authentication using ASP.NET Web API 2, Owin, and ASP.NET Identity – Part 2.
- Enable OAuth Refresh Tokens in AngularJS App using ASP.NET Web API 2, and Owin – Part 3.
- ASP.NET Web API 2 external logins with Facebook and Google in AngularJS app – Part 4.
- Decouple OWIN Authorization Server from Resource Server – Part 5.

- New post: Secure ASP.NET Web API 2 using Azure Active Directory, Owin Middleware, and ADAL.
- New post: Two Factor Authentication in ASP.NET Web API & AngularJS using Google Authenticator.

The demo application can be accessed on (<http://ngAuthenticationWeb.azurewebsites.net/>). The back-end API can be accessed on (<http://ngAuthenticationAPI.azurewebsites.net/>) and both are hosted on Microsoft Azure, for learning purposes feel free to integrate and play with the back-end API with your front-end application. The API supports CORS and accepts HTTP calls from any origin. You can check the [source code](#) for this tutorial on Github.

Home Login Sign Up

AngularJS Authentication

This single page application is built using AngularJS, it is using OAuth bearer token authentication, ASP.NET

ABOUT TAISEER

Father, MVP (ASP.NET/IIS), Scrum Master, Life Time Learner

CONNECT WITH ME

Facebook Twitter LinkedIn Google+ RSS YouTube

Microsoft MVP Most Valuable Professional

HILTON HHONORS EARN DOUBLE POINTS at over 3,600 hotels OR DOUBLE MILES with over 40 airline partners BOOK NOW

JavaScript Diagrams



Lessons Learned

- Use HTTPS over TLS/SSL!
- Forms Authentication
- Third-Party Logins (Facebook, Twitter, etc.)
- Custom Grant Type
- Entity Framework
- [Public-facing] Production Ready?
 - Auth0.com
 - OAuth.io
 - Thinktecture IdentityServer
 - DotNetOpenAuth
 - Spring Social for .NET
 - etc.



OAuth Refresh Tokens



Example Token

It is a GUID.

```
1 {  
2   "access_token": "QKx13H6e4yW3rq3FvGNh_kLDKvuRWESgq6l  
1HjQRqPt_xsQ8DmZK4SuTbOwkneIBMZTK3iyhCAeVtY7kxpfOKF9xY  
HN4g4bhtBeTsEuDF1_xBRdtIDi_eRtek-pJHG6_ku-8xf30z0MNZm  
-Okk4IziRQCQWMontmB-ae9MV5uHU_UfoFqtpERrvOeeeTnAZAPIA  
Z4ih9HMLj6762j3joHRvHYAdhFW06ja4Ud848IQIfDJZAs6_8R1zN5  
xchoXxcvpEd_mCtc1F54FStrpj2qRVpSNjurdnvBGV8dX46eWeU-Aj  
-jM4mOdAL058wOtDw1HX_gE5MWgYjUkivNJ6VpNHYhto-OpDtJ0Uwo  
RBAANOoW7WBT_L-uKFeNJeJzyyRMEDVwAAw3fh2YWSBY34JQSN6ZKR  
K3mKyC8WQq8yJeuHVnkvae2IMwRKBjg-0BNETN7luMAv2qIcDcdX0x  
Z15wLgzfC_ATin-Qkx7WgssQHuyuUQDxC4Gv8Uc4TH3pbbE6Irp_Lw  
x2lpNfDf-0ezDHf-eUNRmNmyFDefcHr5DCBYogPQ02GiDH8yvSfPsN  
rQnuUGUw",  
3   "token_type": "bearer",  
4   "expires_in": 1800,  
5   ".issued": "Thu, 01 Jul 2015 12:00:00 GMT",  
6   ".expires": "Thu, 01 Jul 2015 12:30:00 GMT",  
7   "refresh_token": "873d2a0150ef419bb92523295764910e",  
8   "as:client_id": "ngAuthApp"  
9 }
```



Lessons Learned

Pros

- Updating access token content.
- Maintaining authenticated users list.
- Revoking access from authenticated users.
- Prompting to login multiple times not necessary.

Cons

- Adding a lot of complexity.
- Using third-party logins?



Custom ASP.NET Identity Provider



Custom Provider

For when you want something other than Entity Framework or a GUID primary key.

The screenshot shows the ASP.NET Identity documentation page for custom storage providers. The page is titled "Overview of Custom Storage Providers for ASP.NET Identity" and is authored by Tom FitzMacken, last updated on October 13, 2014. The page includes a table of contents on the left with links to "Getting Started with ASP.NET Identity", "Releases", "Features & API", "Extensibility", "Overview of Custom Storage Providers for ASP.NET Identity", "Change Primary Key for Users in ASP.NET Identity", "Implementing a Custom MySQL ASP.NET Identity Storage Provider", "Migrations", "Security", and "Additional Resources". The main content area starts with an introduction to ASP.NET Identity as an extensible system for creating custom storage providers. It then lists the software versions used in the tutorial and provides a detailed introduction to the architecture, explaining the roles of managers and stores. The page also includes a sidebar with a "Find. Debug. Fix. FASTER." banner for Slackify and a "FREE TRIAL" offer.

asp.net/identity/overview/extensibility/overview-of-custom-storage-providers-for-aspnet-identity

Microsoft Search ASP.NET Language Sign In

ASP.NET Home Get Started Learn Hosting Downloads Community Forums Help

ASP.NET Identity: Overview Samples

Follow us on Twitter Facebook

We Redefined AD Management for 10,000+ Happy Customers

Active Directory Management & Reporting Solution Free Trial

Getting Started with ASP.NET Identity

Releases

Features & API

Extensibility

Overview of Custom Storage Providers for ASP.NET Identity

Change Primary Key for Users in ASP.NET Identity

Implementing a Custom MySQL ASP.NET Identity Storage Provider

Migrations

Security

Additional Resources

Find. Debug. Fix. FASTER. Smarter Errors & Logs for Developers

slackify FREE TRIAL

Overview of Custom Storage Providers for ASP.NET Identity

By Tom FitzMacken | last updated October 13, 2014

Like 145 Tweet 33 Print

ASP.NET Identity is an extensible system which enables you to create your own storage provider and plug it into your application without re-working the application. This topic describes how to create a customized storage provider for ASP.NET Identity. It covers the important concepts for creating your own storage provider, but it is not step-by-step walkthrough of implementing a custom storage provider.

For an example of implementing a custom storage provider, see [implementing a Custom MySQL ASP.NET Identity Storage Provider](#).

This topic was updated for ASP.NET Identity 2.0.

Software versions used in the tutorial

Introduction

By default, the ASP.NET Identity system stores user information in a SQL Server database, and uses Entity Framework Code First to create the database. For many applications, this approach works well. However, you may prefer to use a different type of persistence mechanism, such as Azure Table Storage, or you may already have database tables with a very different structure than the default implementation. In either case, you can write a customized provider for your storage mechanism and plug that provider into your application.

ASP.NET Identity is included by default in many of the Visual Studio 2013 templates. You can get updates to ASP.NET Identity through [Microsoft.AspNet.Identity.EntityFramework](#) NuGet package.

This topic includes the following sections:

- Understand the architecture
- Understand the data that is stored
- Create the data access layer
- Customize the user class
- Customize the user store
- Customize the role class
- Customize the role store
- Reconfigure application to use new storage provider
- Other implementations of custom storage providers

Understand the architecture

ASP.NET Identity consists of classes called managers and stores. Managers are high-level classes which an application developer uses to perform operations, such as creating a user, in the ASP.NET Identity system. Stores are lower-level classes that specify how entities, such as users and roles, are persisted. Stores are closely coupled with the persistence mechanism, but managers are decoupled from stores which means you can replace the persistence mechanism without disrupting the entire application.

- <http://www.asp.net/identity/overview/extensibility/overview-of-custom-storage-providers-for-aspnet-identity>



Architecture

Your application interacts with the managers, and stores interact with the data access layer.

ASP.NET Application

Managers

(UserManager, RoleManager)

Stores

(UserStore, RoleStore)

Data Access Layer

(Dapper.NET)

Data Source

(SQL Server, MongoDB, MySQL, etc.)

- <http://www.asp.net/identity/overview/extensibility/overview-of-custom-storage-providers-for-aspnet-identity>



UserManager

All configuration.

```
ApplicationUserManager - x
MMA3.Security.AspNet.Identity - TRG.EMMA3.Security.AspNet.Identity.App - Create(IdentityFactoryOptions<Ap
1 reference | Mark A. Wilson, 54 days ago | 2 changes
1 public static ApplicationUserManager Create(IdentityFactoryOptions<Ap
2     IOWinContext context)
3 {
4     var manager =
5         new ApplicationUserManager(
6             new UserStore<ApplicationUser>(new ApplicationDatabaseCon
7
8         // Configure validation logic for usernames
9         manager.UserValidator = new UserValidator<ApplicationUser, int>(m
10     {
11         AllowOnlyAlphanumericUserNames = false,
12         RequireUniqueEmail = false
13     });
14     // Configure validation logic for passwords
15     manager.PasswordValidator = new PasswordValidator
16     {
17         RequireDigit = false,
18         RequiredLength = 8,
19         RequireLowercase = false,
20         RequireNonLetterOrDigit = false,
21         RequireUppercase = false
22     };
23     // Configure user lockout defaults
24     manager.UserLockoutEnabledByDefault = true;
25     manager.DefaultAccountLockoutTimeSpan = TimeSpan.FromMinutes(5);
26     manager.MaxFailedAccessAttemptsBeforeLockout = 5;
27
28     IDataProtectionProvider dataProtectionProvider = options.DataProt
29     if (dataProtectionProvider != null)
30     {
31         manager.UserTokenProvider =
32             new DataProtectorTokenProvider<ApplicationUser, int>(data
33     }
34
35     return manager;
36 }
```



UserStore

Gets/Sets don't alter data
in the Data Source (DB).

```
Store.cs
1 using System;
2 using System.Collections.Generic;
3 using System.Data.SqlClient;
4 using System.Linq;
5 using System.Threading.Tasks;
6 using Dapper;
7 using Microsoft.AspNet.Identity;
8 using TRG.EMMA3.Security.AspNet.Identity.Models;
9 using TRG.EMMA3.Security.AspNet.Identity.TsqlQueries;
10
11 namespace TRG.EMMA3.Security.AspNet.Identity.Stores
12 {
13     /// <summary>
14     ///     Responsible for the persistence of a user. Uses an <see cref="int
15     /// </summary>
16     /// <typeparam name="TUser">The user model</typeparam>
17     public sealed class UserStore<TUser> :
18         IUserEmailStore<TUser, int>,
19         IUserLockoutStore<TUser, int>,
20         IUserLoginStore<TUser, int>,
21         IUserPasswordStore<TUser, int>,
22         IUserPhoneNumberStore<TUser, int>,
23         IUserSecurityStampStore<TUser, int>,
24         IUserTwoFactorStore<TUser, int>,
25         IQueryableUserStore<TUser, int>
26     where TUser : User
27     {
28         private readonly string _connectionString;
29
30         /// <summary>
31         ///     Creates an instance of the store
32         /// </summary>
33         /// <remarks>
34         ///     NEVER call the UserStore directly in code! It only serves as
35         ///     storage provider for ASP.NET Identity <see cref="IUserManager{
```



Lessons Learned

- Pretty easy to create.
- Permits integer primary key.
- Interact with managers, not stores.
- Gets/Sets don't alter data in the Data Source (DB).
- In OAuthConfig, configure the database context, user manager, and role manager to use a single instance per request.

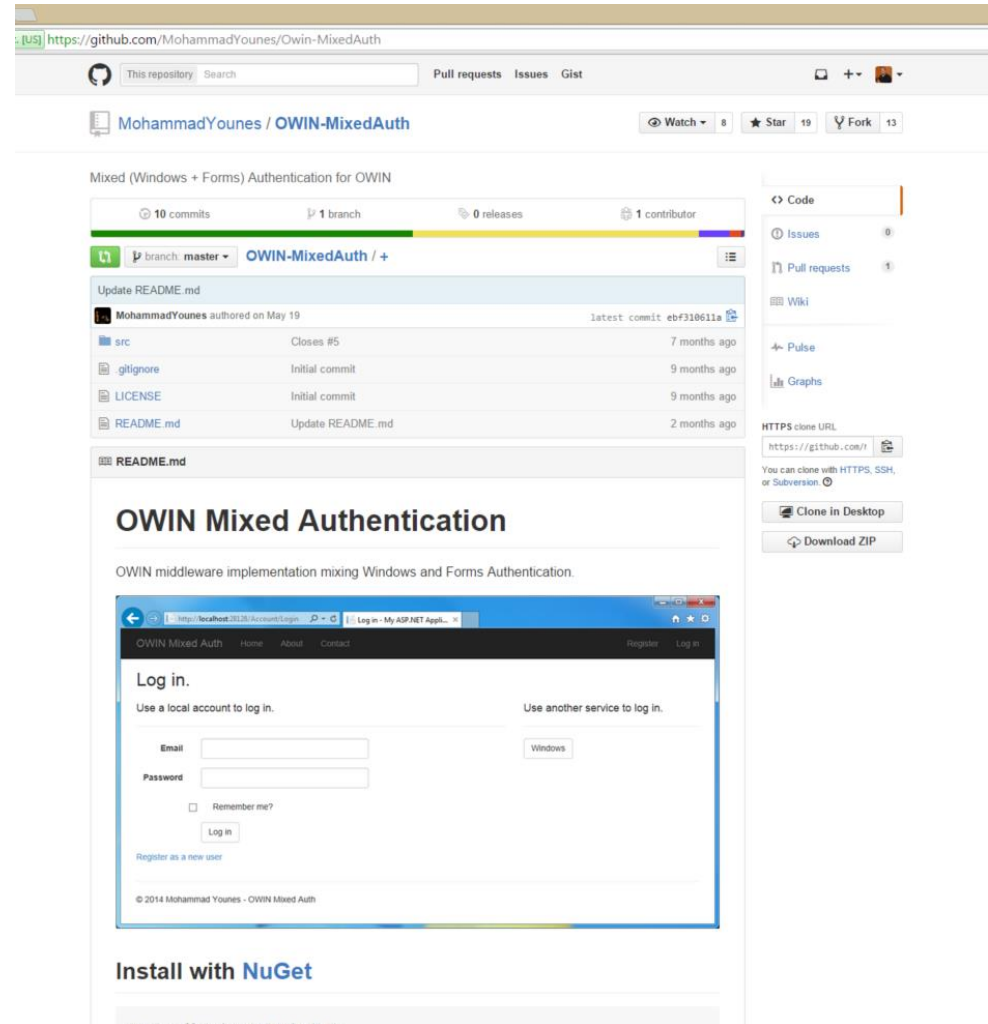


Mixed Authentication



OWIN MixedAuth

Third-party Windows
login provider.



- <https://github.com/MohammadYounes/Owin-MixedAuth>



Lessons Learned

- Proof of concept?
- Enable Windows Authentication.
- Required small code changes.
- Uses a pop-up window.
- Uses a cookie.



Resources

- **Fiddler**
(<http://www.telerik.com/fiddler>)
- **Token Based Authentication using ASP.NET Web API 2, Owin, and Identity**
(<http://bitoftech.net/2014/06/01/token-based-authentication-asp-net-web-api-2-owin-asp-net-identity/>)
- **Overview of Custom Storage Providers for ASP.NET Identity**
(<http://www.asp.net/identity/overview/extensibility/overview-of-custom-storage-providers-for-aspnet-identity>)
- **OWIN Mixed Authentication**
(<https://github.com/MohammadYounes/Owin-MixedAuth>)





Authentication Using Tokens for AngularJS, OWIN, ASP.NET Web API & Identity

Mark A. Wilson

Senior Developer

MarkW@LogicalAdvantage.com

www.DeveloperInfra.com

@DeveloperInfra