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

1:15 PM / Ballroom C

Premier Staffing Partners

RecruitWise

visit knoxville TENNESSEE

codestock

Mark A. Wilson

www.developerinfra.com

@DeveloperInfra

Logical Advantage

www.logicaladvantage.com

markw@

Charlotte, NC

Enterprise Developers Guild
www.developersguild.org

Spark Conference
www.sparkconf.org

Blend Conference
www.blendconf.com

Sponsors

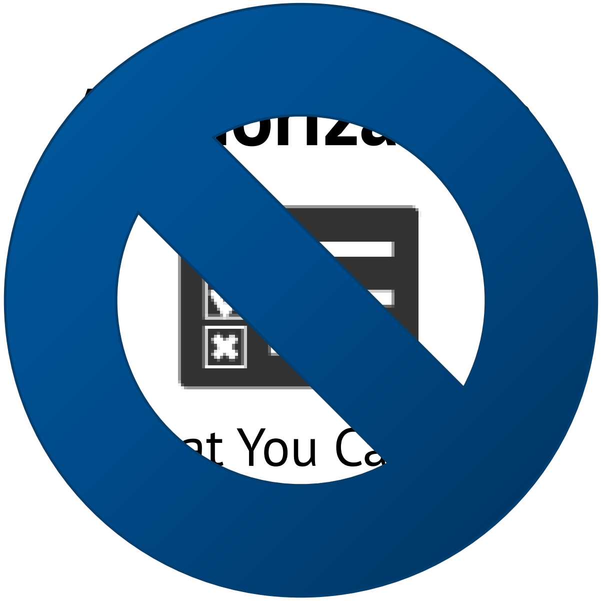
Thank You!

RECRUITWISE
Premier
Staffing Partners
visit knoxville
TENNESSEE
vannmiller
LaunchTennessee
scrippsnetworks
interactive
perfectserve
eventbooking
Clayton homes
Wintellect
Consulting · Training · WintellectNOW
TWSG
The White Stone Group
RESULT STACK
TVA
DEVELOP WITH PURPOSE
DAVE RAMSEY'S DIGITAL DEVELOPMENT TEAM

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

Authentication

Who You Are



Authentication

Cookie-Based

Pros

- Decades-Old & Widely Used

Cons

- Man-In-The-Middle (MITHM)
- 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://sitr.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 (MITHM)
- 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/

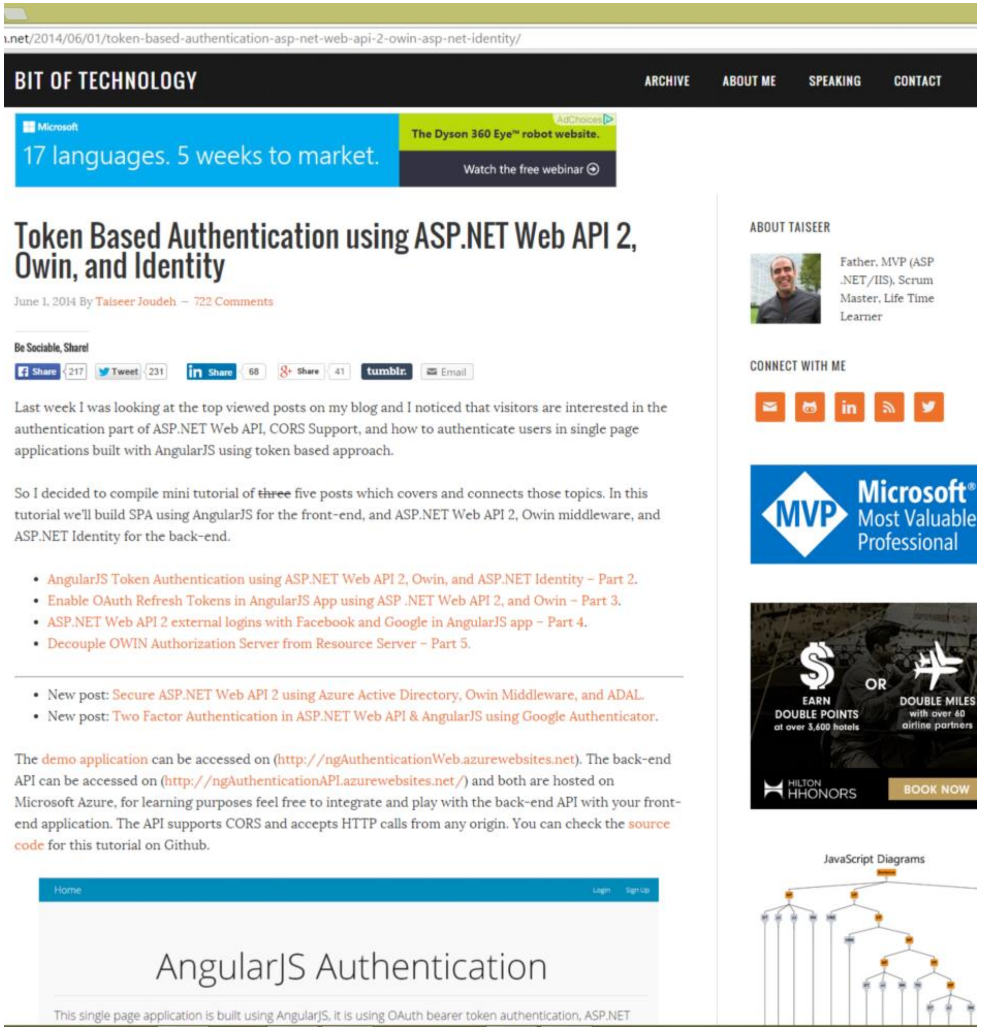
Token-Based Authentication

C O D E

DEMO

Demo

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


.net/2014/06/01/token-based-authentication-asp-net-web-api-2-owin-asp-net-identity/
BIT OF TECHNOLOGY
ARCHIVE
ABOUT ME
SPEAKING
CONTACT

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

June 1, 2014 By Taiseer Joudeh – 722 Comments

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.

AngularJS Authentication

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

ABOUT TAISEER

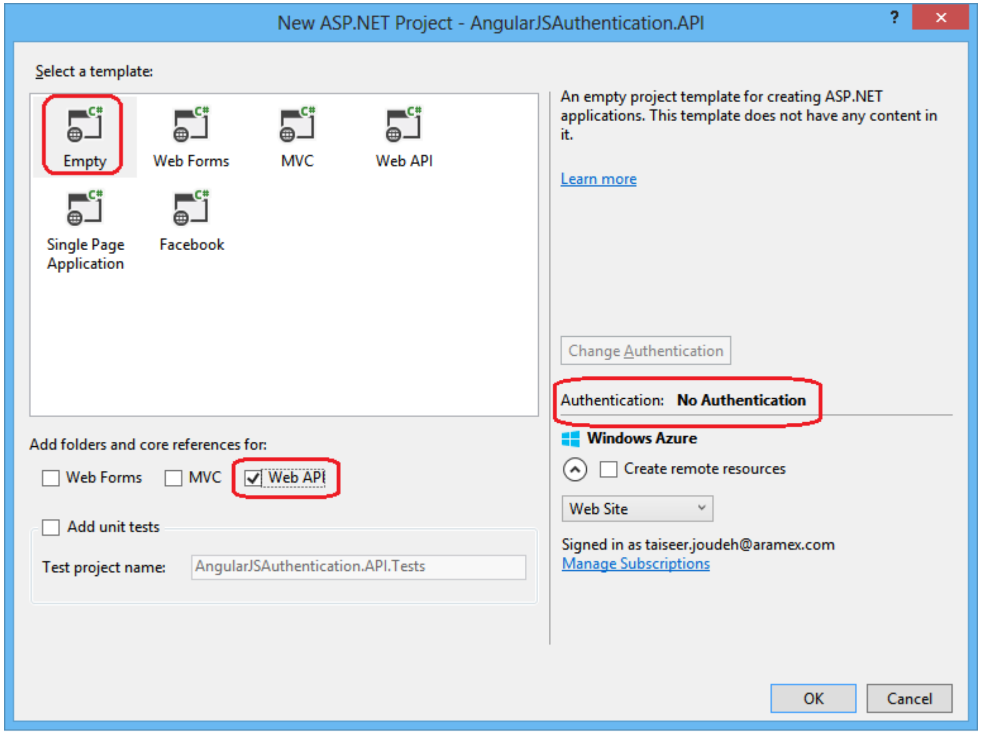
Father, MVP (ASP .NET/IIS), Scrum Master, Life Time Learner

CONNECT WITH ME

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

Lightweight Solutions & Projects

Less is more.



New ASP.NET Project - AngularJSAuthentication.API

Select a template:

Empty

Web Forms

MVC

Web API

Single Page Application

Facebook

An empty project template for creating ASP.NET applications. This template does not have any content in it.

Learn more

Change Authentication

Authentication: **No Authentication**

Windows Azure

□ Create remote resources

Web Site

Signed in as taiseer.joudeh@aramex.com

Manage Subscriptions

Add folders and core references for:

□ Web Forms □ MVC ☑ Web API

□ Add unit tests

Test project name: AngularJSAuthentication.API.Tests

OK Cancel

NuGet Packages

The magical number 7.

Microsoft.AspNet.WebApi

Microsoft.AspNet.WebApi.Owin

Microsoft.Owin.Host.SystemWeb

Microsoft.Owin.Security.OAuth

Microsoft.Owin.Cors

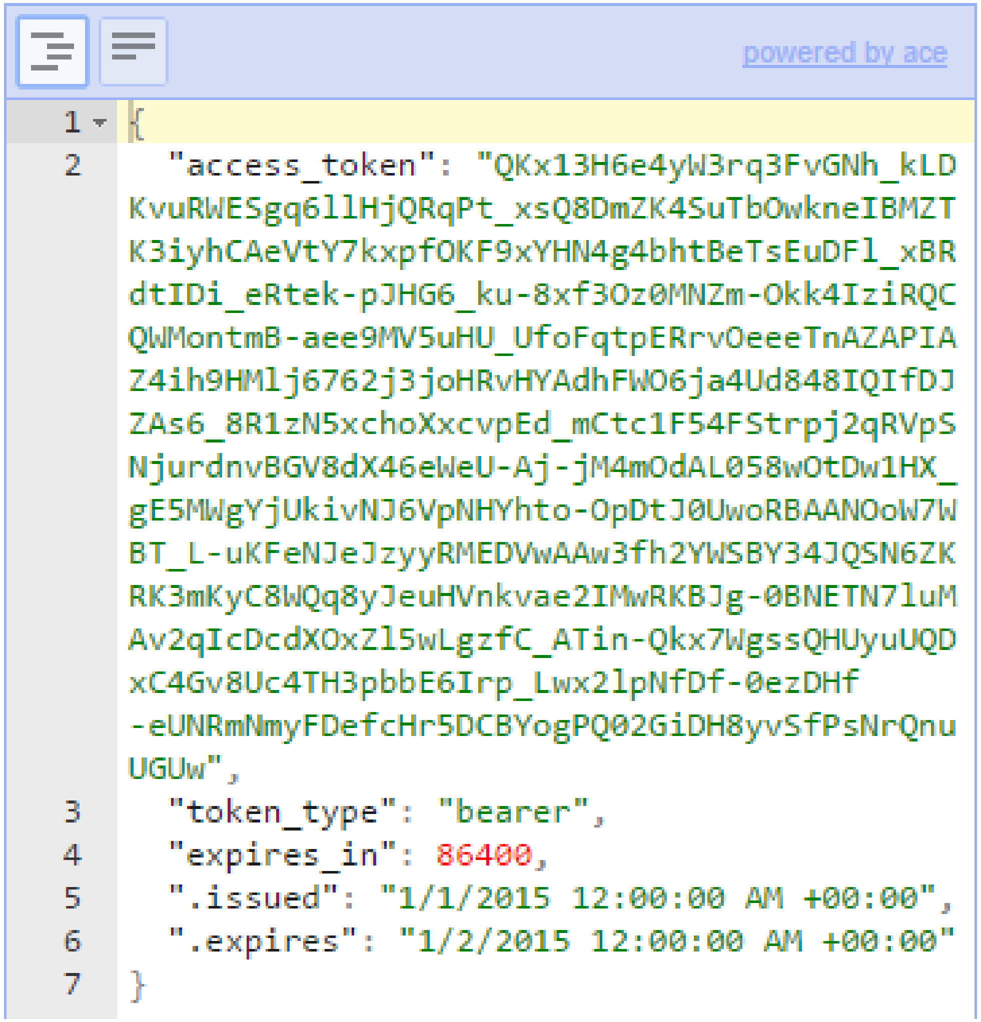
*Microsoft.AspNet.Identity.Owin

*Microsoft.AspNet.Identity.EntityFramework

*Authentication API Project Only

Example Token

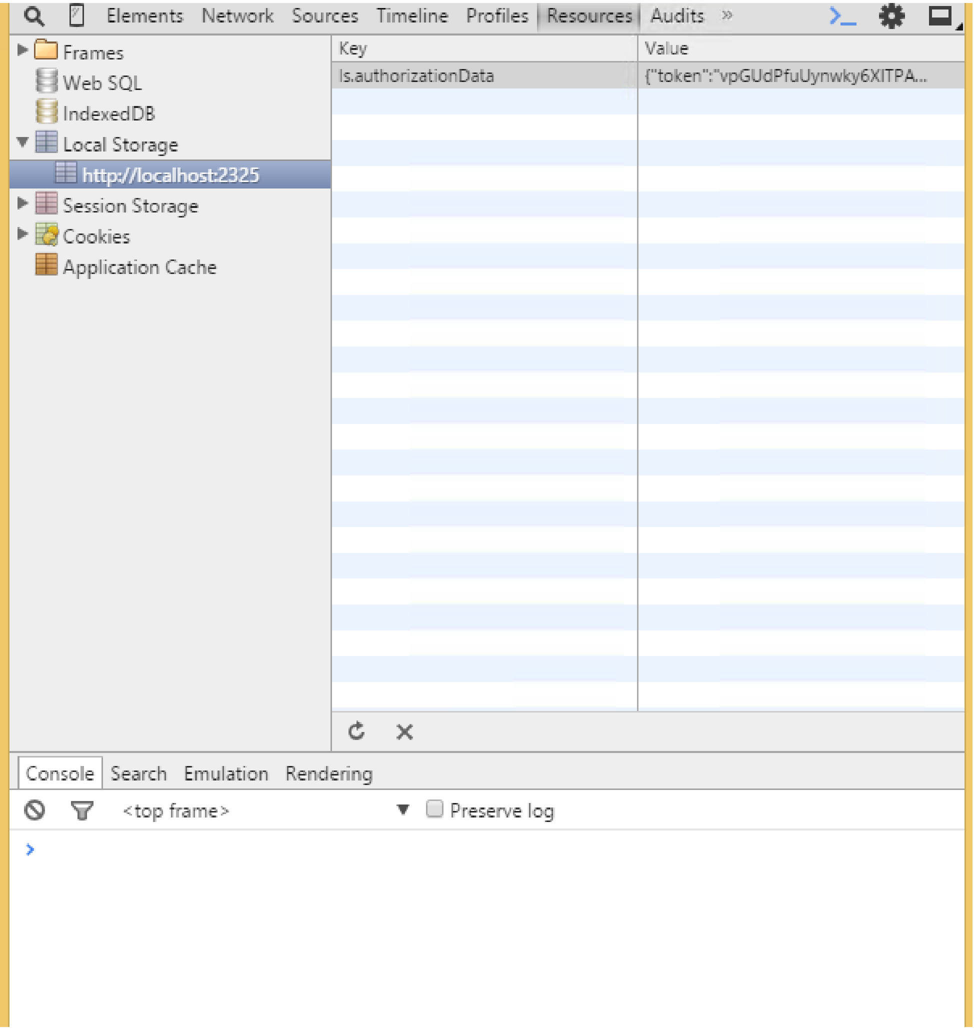
It is JSON.



```
{
  "access_token": "QKx13H6e4yW3rq3FvGNh_kLDKvuRWESgq6llHjQRqPt_xsQ8DmZK4SuTbOwkneIBMZTK3iyhCAeVtY7kxpfOKF9xYHN4g4bhtBeTsEuDFl_xBRdtIDi_eRtek-pJHG6_ku-8xf3Oz0MNZm-Okk4IziRQCQWMontmB-aee9MV5uHU_UfoFqtpERrvOeeeTnAZAPIAZ4ih9HMlj6762j3joHRvHYAdhFWO6ja4Ud848IQIfDJZAs6_8R1zN5xchoXxcvpEd_mCtc1F54FStrpj2qRVpSNjurdnvBGV8dX46eWeU-Aj-jM4mOdAL058wOtDw1HX_gE5MWgYjUkivNJ6VpNHYhto-OpDtJ0UwoRBAANOoW7WBT_L-uKFeNJeJzyyRMEDVwAAw3fh2YWSBY34JQSN6ZKRK3mKyC8WQq8yJeuHVnkvae2IMwRKBJg-0BNETN7luMAv2qIcDcdXOxZl5wLgzfC_ATin-Qkx7WgssQHUyuUQDxC4Gv8Uc4TH3pbbE6Irp_Lwx2lpNfDf-0ezDHf-eUNRmNmyFDefcHr5DCBYogPQ02GiDH8yvSfPsNrQnuUGUw",
  "token_type": "bearer",
  "expires_in": 86400,
  ".issued": "1/1/2015 12:00:00 AM +00:00",
  ".expires": "1/2/2015 12:00:00 AM +00:00"
}
```

Local Storage

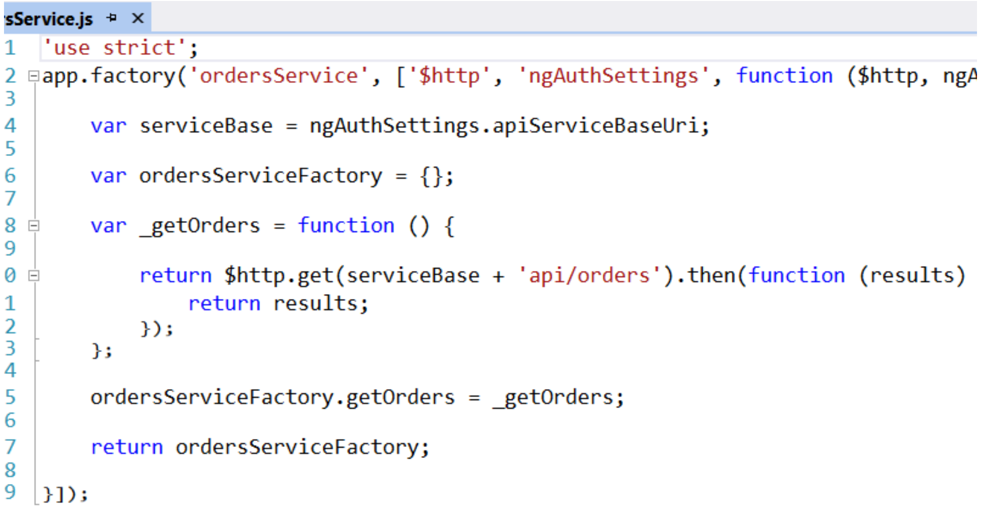
Other options include JavaScript or a Cookie.



Key	Value
ls.authorizationData	{"token":"vpGUdPfuUynwky6XITPA...

AngularJS Service

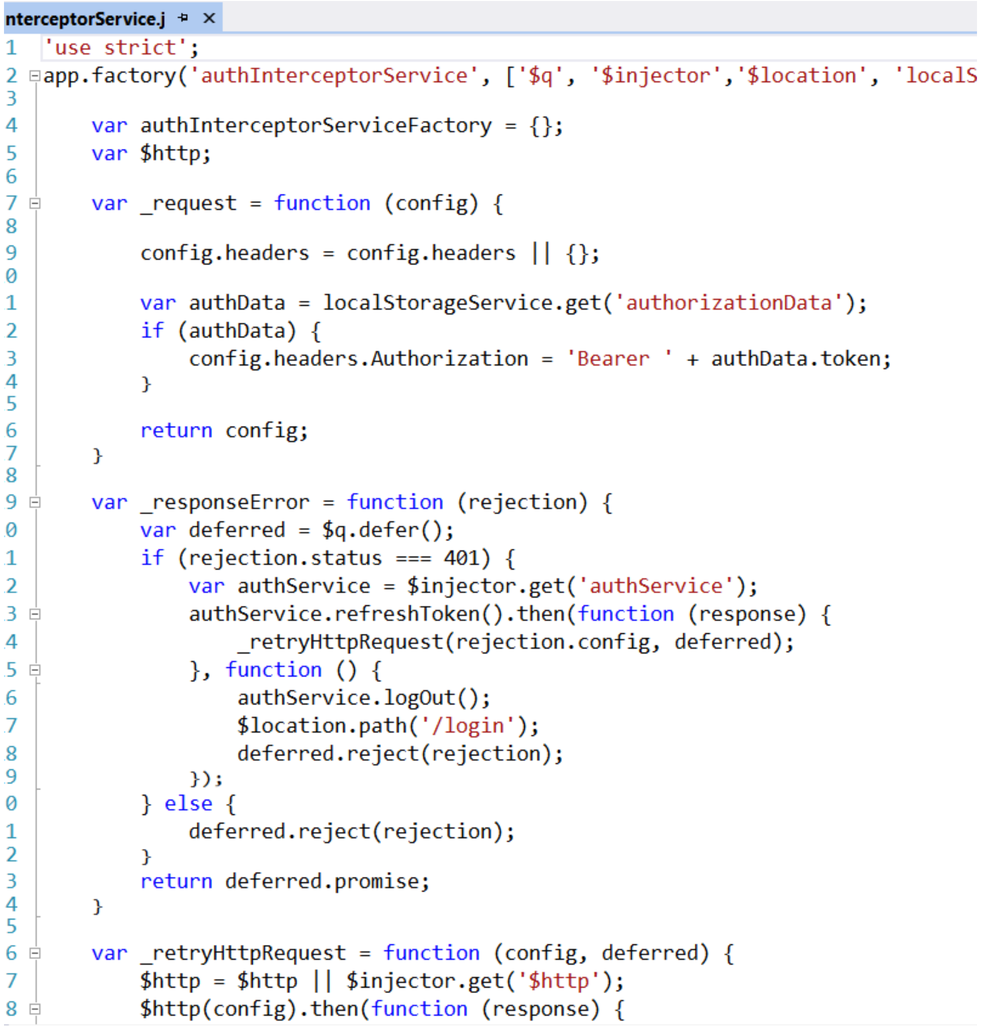
Move along, nothing to see here.



```
'use strict';
app.factory('ordersService', ['$http', 'ngAuthSettings', function ($http, ngA

    var serviceBase = ngAuthSettings.apiServiceBaseUri;

    var ordersServiceFactory = {};

    var _getOrders = function () {

        return $http.get(serviceBase + 'api/orders').then(function (results)
            return results;
        });
    };

    ordersServiceFactory.getOrders = _getOrders;

    return ordersServiceFactory;

}]);
```

AngularJS Interceptor

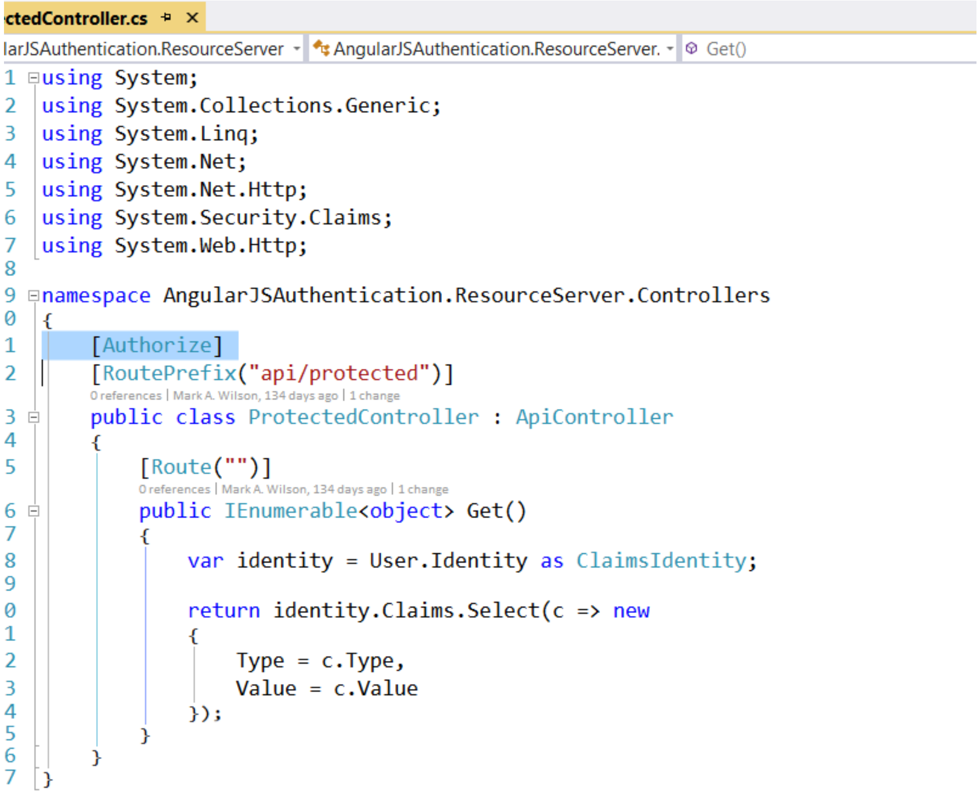
Pre- and post-processing.



```
'use strict';
app.factory('authInterceptorService', ['$q', '$injector','$location', 'localS

    var authInterceptorServiceFactory = {};
    var $http;

    var _request = function (config) {

        config.headers = config.headers || {};

        var authData = localStorageService.get('authorizationData');
        if (authData) {
            config.headers.Authorization = 'Bearer ' + authData.token;
        }

        return config;
    }

    var _responseError = function (rejection) {
        var deferred = $q.defer();
        if (rejection.status === 401) {
            var authService = $injector.get('authService');
            authService.refreshToken().then(function (response) {
                _retryHttpRequest(rejection.config, deferred);
            }, function () {
                authService.logOut();
                $location.path('/login');
                deferred.reject(rejection);
            });
        } else {
            deferred.reject(rejection);
        }
        return deferred.promise;
    }

    var _retryHttpRequest = function (config, deferred) {
        $http = $http || $injector.get('$http');
        $http(config).then(function (response) {
```

ASP.NET Web API Controller

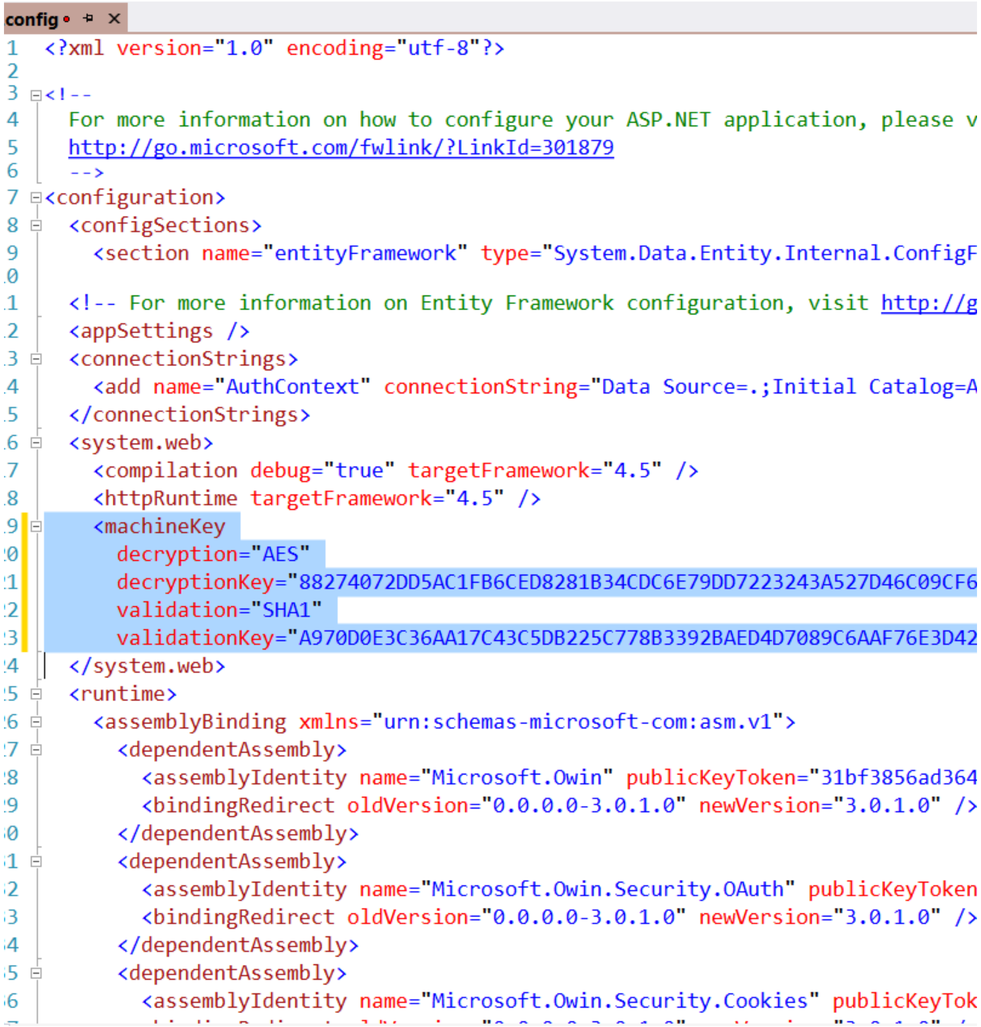
You shall not pass!


ctedController.cs
larJSAuthentication.ResourceServer
AngularJSAuthentication.ResourceServer.
Get()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Security.Claims;
using System.Web.Http;

namespace AngularJSAuthentication.ResourceServer.Controllers
{
    [Authorize]
    [RoutePrefix("api/protected")]
    0 references | Mark A. Wilson, 134 days ago | 1 change
    public class ProtectedController : ApiController
    {
        [Route("")]
        0 references | Mark A. Wilson, 134 days ago | 1 change
        public IEnumerable<object> Get()
        {
            var identity = User.Identity as ClaimsIdentity;

            return identity.Claims.Select(c => new
            {
                Type = c.Type,
                Value = c.Value
            });
        }
    }
}
```

Shared MachineKey

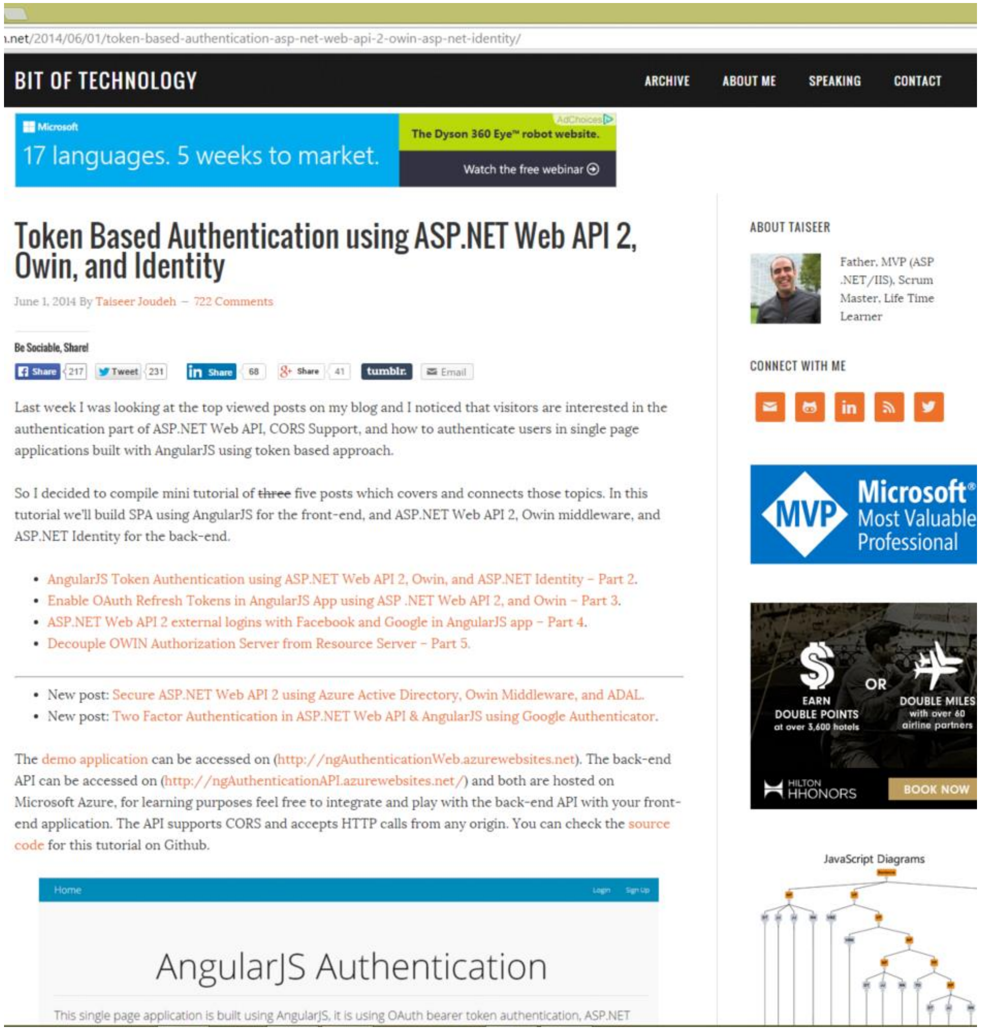
Do not use the same keys for dev & prod.



```
<?xml version="1.0" encoding="utf-8"?>

<!--
  For more information on how to configure your ASP.NET application, please v
  http://go.microsoft.com/fwlink/?LinkId=301879
  -->
<configuration>
  <configSections>
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigF

  <!-- For more information on Entity Framework configuration, visit http://g
  <appSettings />
  <connectionStrings>
    <add name="AuthContext" connectionString="Data Source=.;Initial Catalog=A
  </connectionStrings>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5" />
    <machineKey
      decryption="AES"
      decryptionKey="88274072DD5AC1FB6CED8281B34CDC6E79DD7223243A527D46C09CF6
      validation="SHA1"
      validationKey="A970D0E3C36AA17C43C5DB225C778B3392BAED4D7089C6AAF76E3D42
  </system.web>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.Owin" publicKeyToken="31bf3856ad364
        <bindingRedirect oldVersion="0.0.0.0-3.0.1.0" newVersion="3.0.1.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.Owin.Security.OAuth" publicKeyToken
        <bindingRedirect oldVersion="0.0.0.0-3.0.1.0" newVersion="3.0.1.0" />
      </dependentAssembly>
      <dependentAssembly>
        <assemblyIdentity name="Microsoft.Owin.Security.Cookies" publicKeyTok
```

Demo

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



.net/2014/06/01/token-based-authentication-asp-net-web-api-2-owin-asp-net-identity/
BIT OF TECHNOLOGY
ARCHIVE
ABOUT ME
SPEAKING
CONTACT

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

June 1, 2014 By Taiseer Joudeh – 722 Comments

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.

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

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

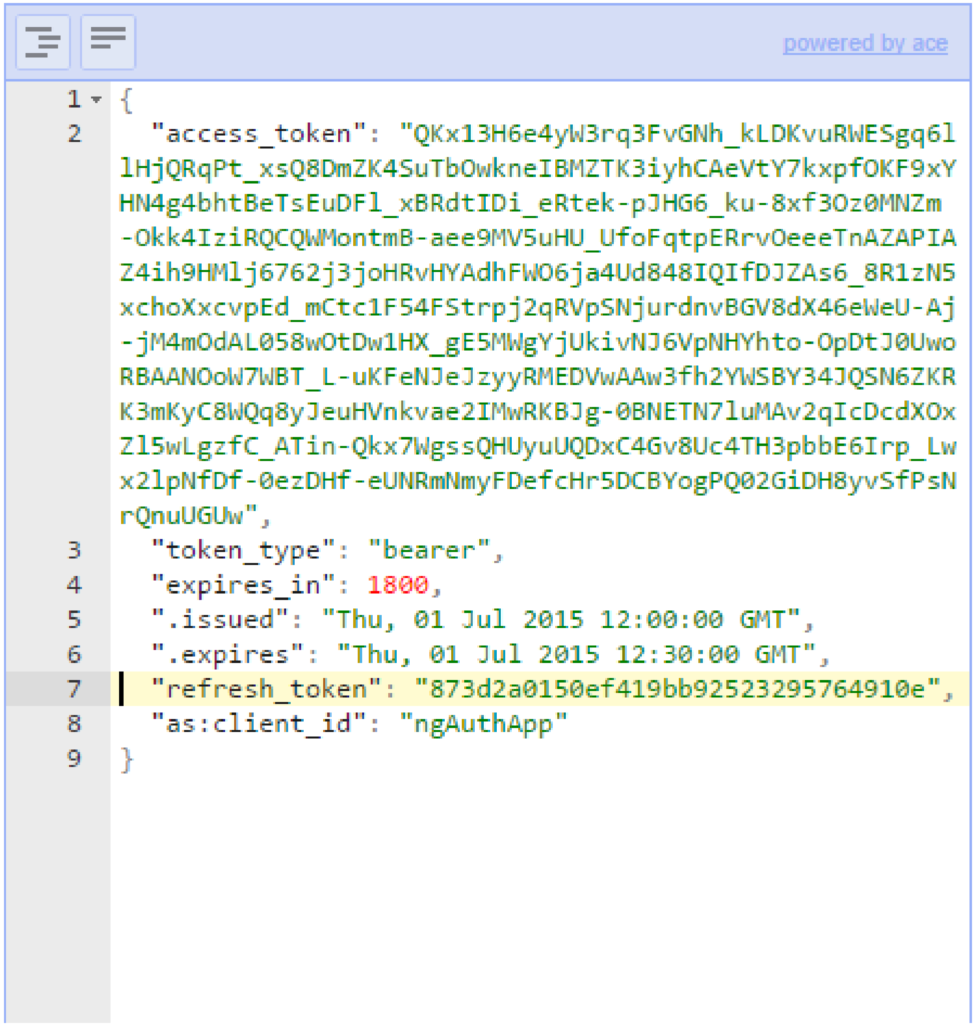
Lessons Learned

- 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.



```json
{
  "access_token": "QKx13H6e4yW3rq3FvGNh_kLDKvuRWESgq6l1HjQRqPt_xsQ8DmZK4SuTbOwkneIBMZTK3iyhCAeVtY7kxpfOKF9xYHN4g4bhtBeTsEuDFl_xBRdtIDi_eRtek-pJHG6_ku-8xf3Oz0MNZm-Okk4IziRQCQWMontmB-aee9MV5uHU_UfoFqtpERrvOeeeTnAZAPIAZ4ih9HMlj6762j3joHRvHYAdhFWO6ja4Ud848IQIfDJZAs6_8R1zN5xchoXxcvpEd_mCtc1F54FStrpj2qRVpSNjurdnvBGV8dX46eWeU-Aj-jM4mOdAL058wOtDw1HX_gE5MWgYjUkivNJ6VpNHYhto-OpDtJ0UwoRBAANOoW7WBT_L-uKFeNJeJzyyRMEDVwAAw3fh2YWSBY34JQSN6ZKRK3mKyC8WQq8yJeuHVnkvae2IMwRKBJg-0BNETN7luMAv2qIcDcdXOxZl5wLgzfC_ATin-Qkx7WgssQHUyuUQDxC4Gv8Uc4TH3pbbE6Irp_Lwx2lpNfDf-0ezDHf-eUNRmNmyFDefcHr5DCBYogPQ02GiDH8yvSfPsNrQnuUGUw",
  "token_type": "bearer",
  "expires_in": 1800,
  ".issued": "Thu, 01 Jul 2015 12:00:00 GMT",
  ".expires": "Thu, 01 Jul 2015 12:30:00 GMT",
  "refresh_token": "873d2a0150ef419bb92523295764910e",
  "as:client_id": "ngAuthApp"
}
```

Lessons Learned

Pros

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

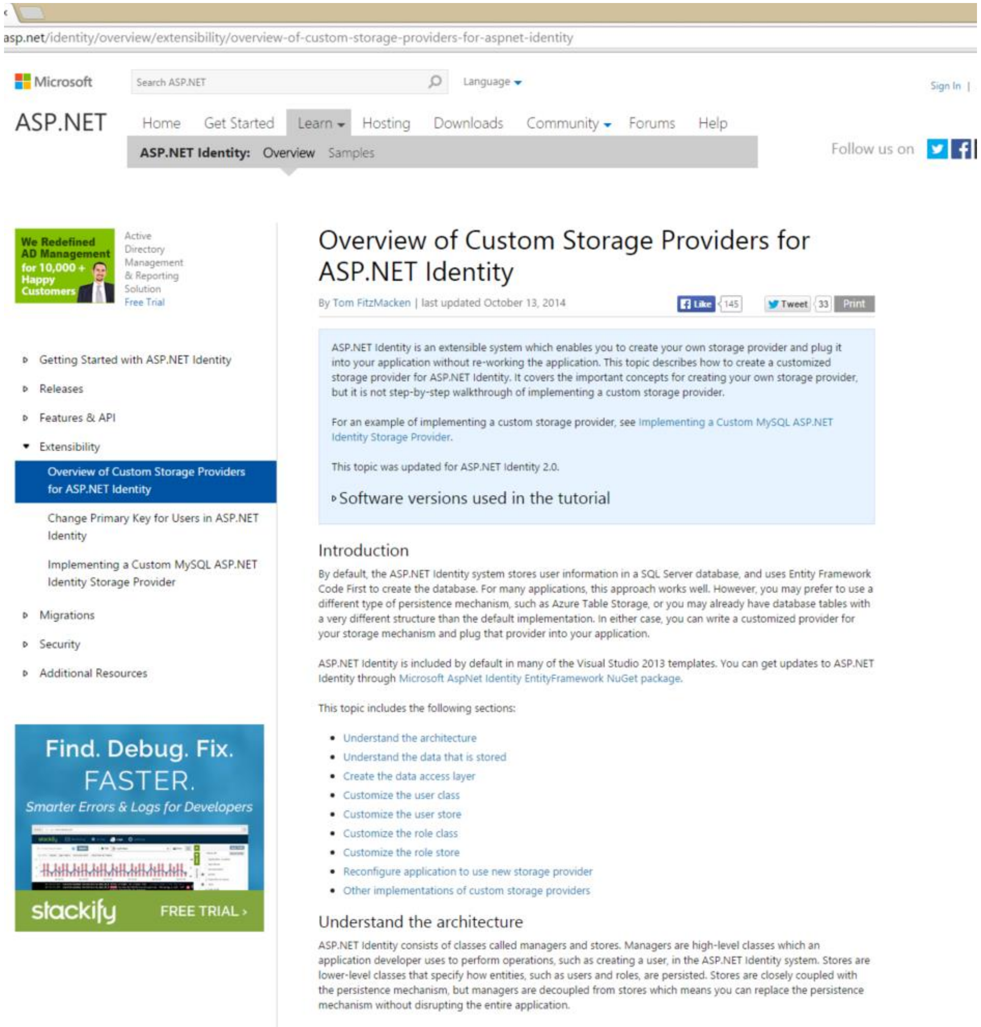
Cons

- 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.



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

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

Overview of Custom Storage Providers for ASP.NET Identity

By Tom FitzMacken | last updated October 13, 2014

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

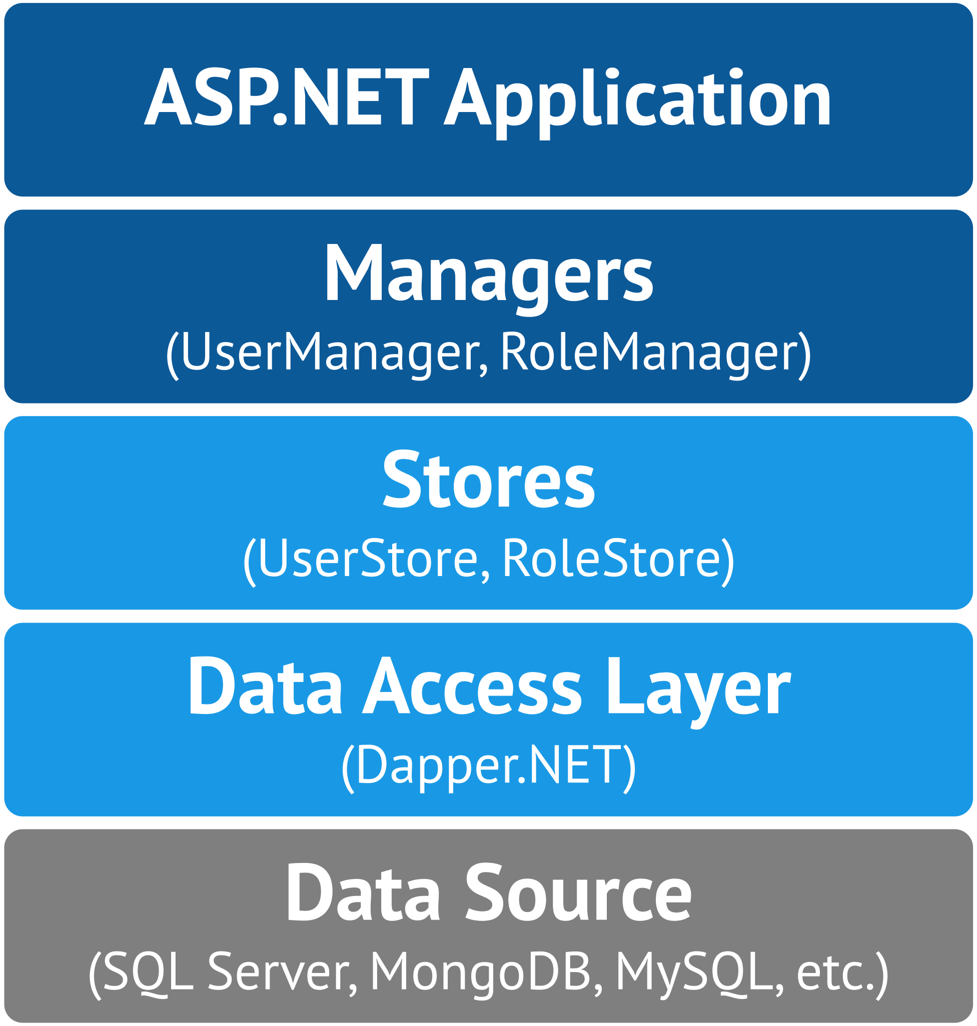
Understand the architecture

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)

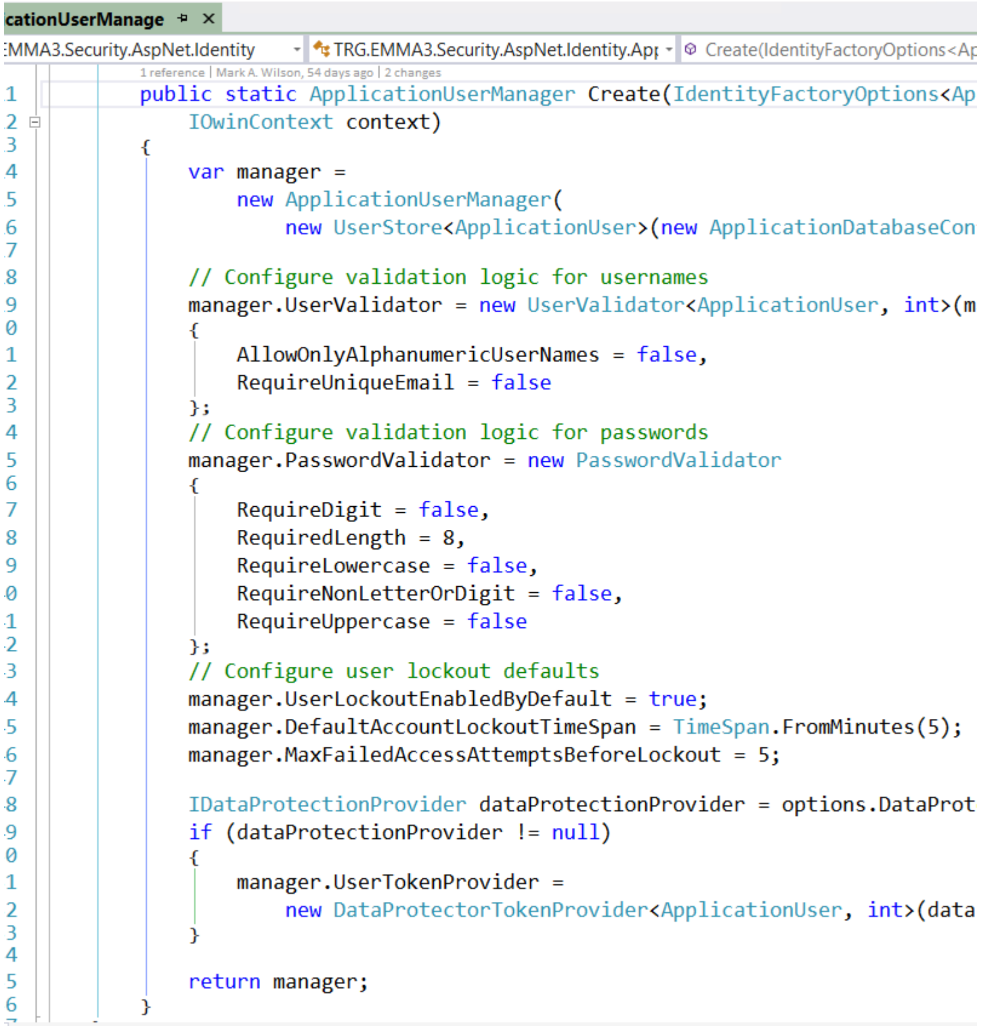
Data Access Layer
(Dapper.NET)

Data Source
(SQL Server, MongoDB, MySQL, etc.)

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

UserManager

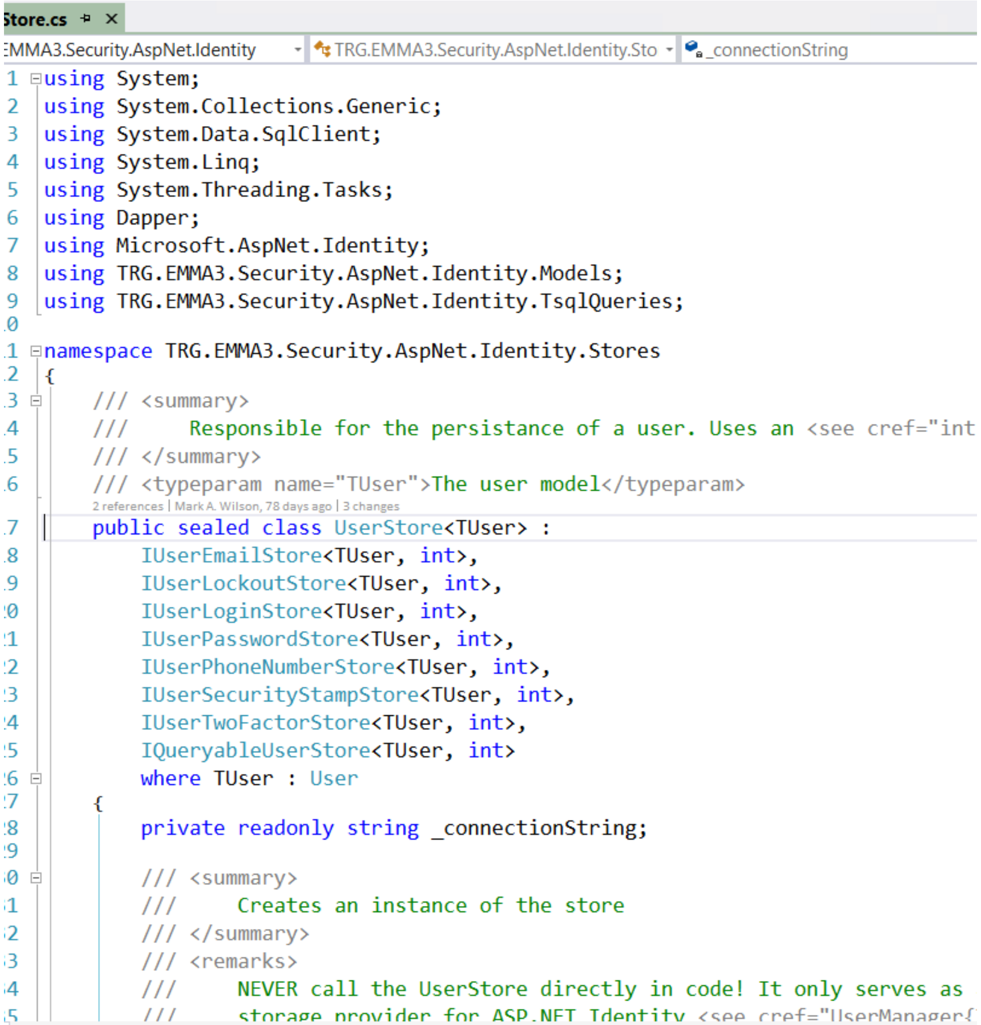
All configuration.


cationUserManage
:MMA3.Security.AspNet.Identity
TRG.EMMA3.Security.AspNet.Identity.Ap
Create(IdentityFactoryOptions<Ap
1 reference | Mark A. Wilson, 54 days ago | 2 changes

```
public static ApplicationUserManager Create(IdentityFactoryOptions<Ap
    IOwinContext context)
{
    var manager =
        new ApplicationUserManager(
            new UserStore<ApplicationUser>(new ApplicationDatabaseCon

    // Configure validation logic for usernames
    manager.UserValidator = new UserValidator<ApplicationUser, int>(m
    {
        AllowOnlyAlphanumericUserNames = false,
        RequireUniqueEmail = false
    };
    // Configure validation logic for passwords
    manager.PasswordValidator = new PasswordValidator
    {
        RequireDigit = false,
        RequiredLength = 8,
        RequireLowercase = false,
        RequireNonLetterOrDigit = false,
        RequireUppercase = false
    };
    // Configure user lockout defaults
    manager.UserLockoutEnabledByDefault = true;
    manager.DefaultAccountLockoutTimeSpan = TimeSpan.FromMinutes(5);
    manager.MaxFailedAccessAttemptsBeforeLockout = 5;

    IDataProtectionProvider dataProtectionProvider = options.DataProt
    if (dataProtectionProvider != null)
    {
        manager.UserTokenProvider =
            new DataProtectorTokenProvider<ApplicationUser, int>(data
    }

    return manager;
}
```

UserStore

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


Store.cs
EMMA3.Security.AspNet.Identity
TRG.EMMA3.Security.AspNet.Identity.Sto
_connectionString
2 references | Mark A. Wilson, 78 days ago | 3 changes

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Threading.Tasks;
using Dapper;
using Microsoft.AspNet.Identity;
using TRG.EMMA3.Security.AspNet.Identity.Models;
using TRG.EMMA3.Security.AspNet.Identity.TsqlQueries;

namespace TRG.EMMA3.Security.AspNet.Identity.Stores
{
    /// <summary>
    ///     Responsible for the persistance of a user. Uses an <see cref="int
    /// </summary>
    /// <typeparam name="TUser">The user model</typeparam>
    public sealed class UserStore<TUser> :
        IUserEmailStore<TUser, int>,
        IUserLockoutStore<TUser, int>,
        IUserLoginStore<TUser, int>,
        IUserPasswordStore<TUser, int>,
        IUserPhoneNumberStore<TUser, int>,
        IUserSecurityStampStore<TUser, int>,
        IUserTwoFactorStore<TUser, int>,
        IQueryableUserStore<TUser, int>
        where TUser : User
    {
        private readonly string _connectionString;

        /// <summary>
        ///     Creates an instance of the store
        /// </summary>
        /// <remarks>
        ///     NEVER call the UserStore directly in code! It only serves as
        ///     storage provider for ASP.NET Identity <see cref="UserManager{
```

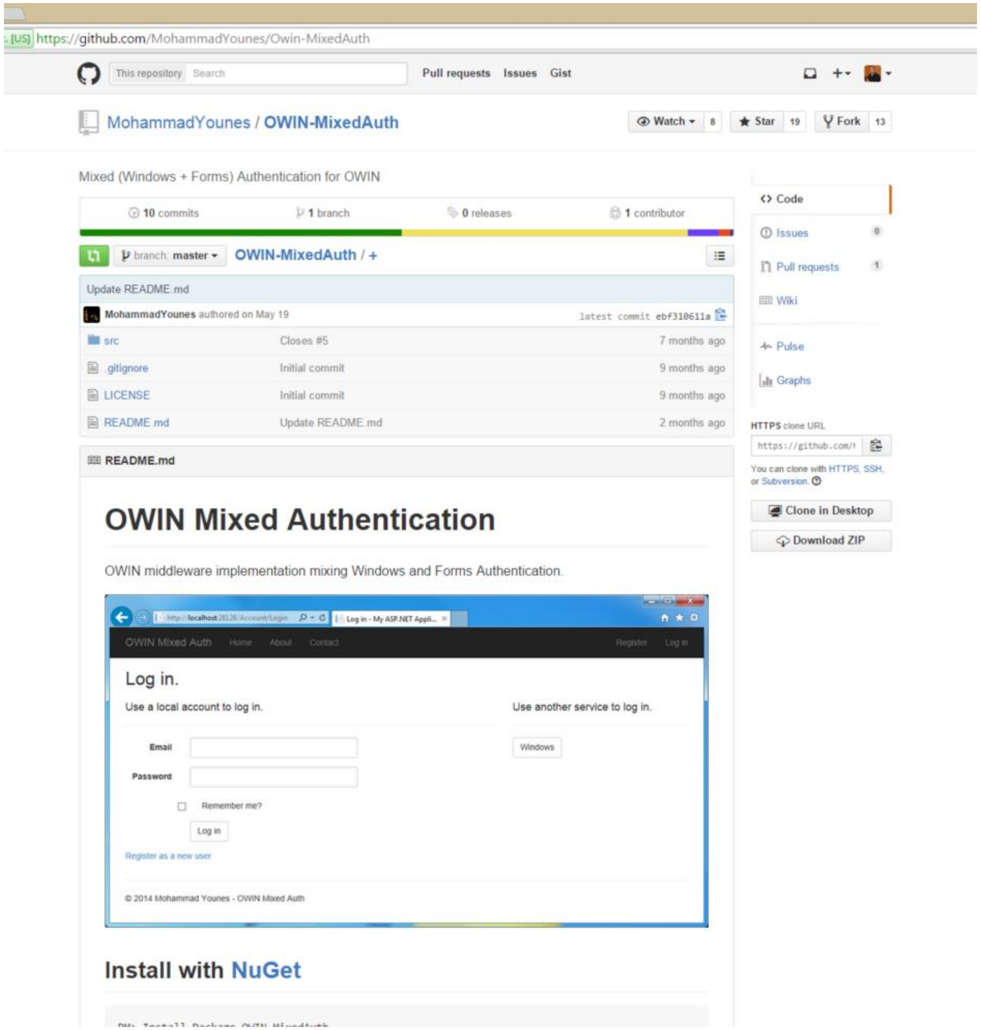
Lessons Learned

- 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.



You can clone with HTTPS, SSH, or Subversion.

Mixed (Windows + Forms) Authentication for OWIN

Update README.md

MohammadYounes authored on May 19 — latest commit ebf310611a

src	Closes #5	7 months ago
.gitignore	Initial commit	9 months ago
LICENSE	Initial commit	9 months ago
README.md	Update README.md	2 months ago

OWIN Mixed Authentication

OWIN middleware implementation mixing Windows and Forms Authentication.

http://localhost:28128/Account/Login
Log in - My ASP.NET Appli...
OWIN Mixed Auth
Home
About
Contact
Register
Log in
Log in.
Use a local account to log in.
Use another service to log in.
Email
Password
□ Remember me?
Log in
Windows
Register as a new user
© 2014 Mohammad Younes - OWIN Mixed Auth

Install with NuGet

```
PM> Install-Package OWIN-MixedAuth
```

- 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