**Introduction to ASP.NET Identity**

ASP.NET Identity API is used for managing user credentials. It replaces the older ASP.NET Membership system.

 It allows users to self-register on the site.  It allows users to select user id and password. Users can also register using the social login providers such as facebook, google, twitter etc. ASP.NET MVC Identity uses the [entity framework Code first](https://www.tektutorialshub.com/entity-framework-tutorial/) and [OWIN](https://www.tektutorialshub.com/owin/).

**Brief History of ASP.NET Membership system**

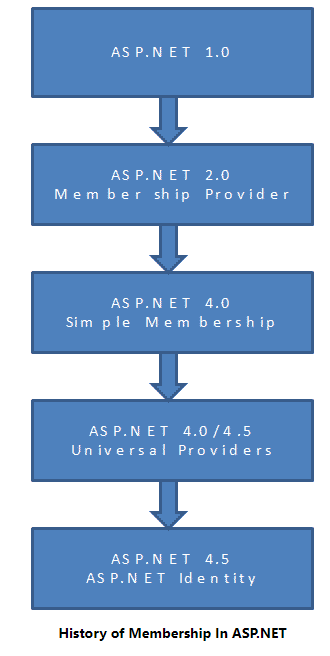
ASP.NET used the forms authentication system to authenticate users to the web application.

Starting from ASP.NET 2.0 introduced the Membership provider.

imple Membership made an entry in ASP.NET 4.0.  A Universal Membership provider in ASP.NET 4/4.5.

All these Membership providers used the abstract base class to manage the user credentials.

Internally they all used Forms Authentication to Provide the Security.



## **Why change**

The ASP.NET membership has worked well over the years.  At the same time developing the web application has also evolved a lot.  The Developers want their users to log in using the social accounts so that they can be provided with rich user experience.

Two-factor authentication has become an essential feature of the applications.

1 - Asp net identity  can be used across all of the ASP.NET frameworks, such as ASP.NET MVC, Web Forms, Web API and SignalR.

2 - ASP.NET Identity comes with API to handle Creating, Editing and deleting of users.

3 - The Developer has now complete control over the user information. You can easily extend the user profile information to include more user information like date of birth or any other information.

4 - Identity uses the entity framework code first to store the user related information to the database.

5 - Identity also provides control over the data persistence. Since it uses the Entity Framework, it can be used to persist data in any datastore supported by EF.

6- Two-factor authentication

7 - The Account Confirmation email can be sent to the user to verify the email ID.

***8- Account lockout*** feature disables the user ***accounts*** if the user enters the wrong password for a specified number of times over a short duration.

9- Unit test

10 - Roles are similar to Roles in the old membership system. These are stored in a separate table.  You can Create, Edit and delete roles

11 - The claim is a piece of information about the user. It could be his name, his ID, email or anything that is important to the application you build. They are stored as a name-value pair. Claims allow developers to add lots more information about the user than the simple username-password provided by the old membership system.

12 – Owin integration: ASP.NET Identity uses OWIN to generate cookies and do authentication. It is not required use OWIN for Identity. You can also use the Forms Authentication module to do the same.

13 - ASP.NET Identity is now not part of the ASP.NET framework. The identity is now available as a download from NuGet packages.

**ASP.NET Identity Tutorial – Getting Started**

(<https://www.tektutorialshub.com/asp-net/asp-net-identity-tutorial-basics/#aspnet-identity-tutorial>)

There are three main Components in the ASP.NET Identity

* Microsoft.AspNet.Identity.Core
* Microsoft.AspNet.Identity.EntityFramework
* Microsoft.AspNet.Identity.Owin

#### [**Microsoft.AspNet.Identity.Core**](https://docs.microsoft.com/en-us/previous-versions/aspnet/dn253016(v=vs.108))

This is the Core module of the Identity System. It contains classes and interfaces related to managing users and roles for ASP.NET Identity.

#### [**Microsoft.AspNet.Identity.EntityFramework**](https://docs.microsoft.com/en-us/previous-versions/aspnet/dn253028(v=vs.108))

This is an [Entity Framework](https://www.tektutorialshub.com/entity-framework-tutorial/) namespace specific to ASP.NET Identity. This has a concrete Implementation of the Interfaces defined in the Microsoft.ASPNet.Identity.Core namespace, which will persist the ASP.NET Identity data and schema to SQL Server.

#### [**Microsoft.AspNet.Identity.Owin**](https://docs.microsoft.com/en-us/previous-versions/aspnet/dn337153(v=vs.108))

The Identity System uses the Owin middleware components for Cookie Authentication and provides methods to implement external logins such as Facebook, Google and Twitter.

Install-Package Microsoft.AspNet.Identity.Core

Install-Package Microsoft.AspNet.Identity.EntityFramework

Install-Package Microsoft.aspnet.identity.owin

he ASP.NET Identity uses [Entity framework](https://www.tektutorialshub.com/entity-framework-tutorial/) to connect to the database and store the user’s information. To do that, we need to create a database. Let us use the localdb to do that.

Open the**web.config** and add the following connection string under configuration section.

Open the**web.config** and add the following connection string under configuration section.

<connectionStrings>

    <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\ASPNetIdentity.mdf;Initial Catalog=ASPNetIdentity;Integrated Security=True"

providerName="System.Data.SqlClient" />

</connectionStrings>

### IdentityUser

We need a class to represent the user,  where all the user-related information’s are kept.

This Interface is defined in the Microsoft.AspNet.Identity namespace. It requires an ID and Name field for every user.

The system generates a GUID and stores it as a string in this field. You can override this field with an integer or any other type

IdentityUser class extends the IUser interface by adding the minimum required fields for our user model.

public virtual string Email { get; set; }

public virtual bool EmailConfirmed { get; set; }

public virtual TKey Id { get; set; }

public virtual bool LockoutEnabled { get; set; }

public virtual DateTime? LockoutEndDateUtc { get; set; }

public virtual ICollection<TLogin> Logins { get; }

public virtual string PasswordHash { get; set; }

public virtual string PhoneNumber { get; set; }

public virtual bool PhoneNumberConfirmed { get; set; }

public virtual ICollection<TRole> Roles { get; }

public virtual string SecurityStamp { get; set; }

public virtual bool TwoFactorEnabled { get; set; }

public virtual string UserName { get; set; }

But it is advisable to define our own class and inherit it from IdentityUser.  You can add your custom fields to this class.

public class ApplicationUser :IdentityUser

{

//You can extend this class by adding additional fields like Birthday

}

**The Entity Framework requires [DbContext](https://www.tektutorialshub.com/entity-framework/ef-dbcontext/)**, which is responsible for interacting with the entity model and the database.

[Microsoft.AspNet.Identity.EntityFramework](https://www.tektutorialshub.com/entity-framework/ef-introduction/) namespace implements  the IdentityDbContext<TUser> which inherits from *[DbContext](https://www.tektutorialshub.com/entity-framework/ef-dbcontext/)*.

This class provides the ready *[DbSets](https://www.tektutorialshub.com/entity-framework/ef-dbset/)* for *Users and Roles.*

### Configuring ASP.NET Identity

Next, we need to configure the Identity to use UserManager and UserStore.

Let us create class **IdentityConfig.cs** under the **App\_start** folder as shown below.  Remove the App\_start from the namespace.

**User store** provides the methods to persist the user, user roles, user claims etc to the database.

The UserManager<TUser> is a concrete class that manages the user.

This Class Creates, Updates, and Deletes the Users. It has methods to find a user by ID, User Name and email. It also provides the functionality for adding  Claims, removing claims, add and removing roles, etc. It also generates password hash, Validates Users etc.

UserStore must be passed to the UserManager in its Constructor. You can pass any object that implements IUserStore interface.

We have got the required classes in place. Let us now try to add a user and test it.

public async Task<string> AddUser()

        {

            ApplicationUser user;

            ApplicationUserStore Store = new ApplicationUserStore(new ApplicationDbContext());

            ApplicationUserManager userManager = new ApplicationUserManager(Store);

            user = new ApplicationUser

            {

                UserName = "TestUser",

                Email = "TestUser@test.com"

            };

            var result = await userManager.CreateAsync(user);

            if (!result.Succeeded)

            {

                return result.Errors.First();

            }

            return "User Added";

        }

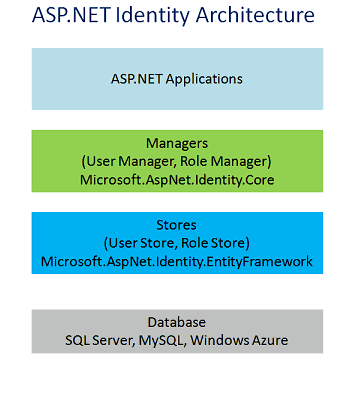
## Identity Tables

Open the database, you will find that the following tables are automatically added by the Identity system

AspNetUserLogins  
This table to hold information about 3rd party/external logins like facebook, google etc

AspNetUserClaims  
Stores Claims associated with the user

Recap:



### Managers

ASP.Net Identity has high-level classes called Managers, which is used by our application to manage Identity models like users, roles, claims etc. Currently, ASP.NET Identity Provides two managers. UserManger andRoleManager

### Identity Models

The Identity Models represent the domain model. ASP.NET Identity provides the basic Interface for these models. For Example, IUser Interface for User Model and IRole for Roles, IClaim for Claims.

### Use Store

The Managers uses the UseStore to talk to the Stores. The stores define how the users, roles are persisted to the database. The User Manager is decoupled from the UserStore.

This decoupling enables us to create our own implementation of User Store

**ASP.NET Identity Tutorial – OWIN Authentication Middleware**

In this tutorial, we are going to build the login page and look at how to Authenticate the user using the OWIN Middleware authentication component.

The OWIN authentication middleware is used for authenticating users.

In older ASP.NET Applications, we used *Forms authentication module* to authenticate the users into our application.

The OWIN authentication middleware can issue authentication cookies on its own or it can use the external logins like facebook, google, twitter etc to do that.

The OWIN authentication middleware is platform agnostic. It can be used in ASP.NET MVC, webforms or ASP.NET core.

Install this:

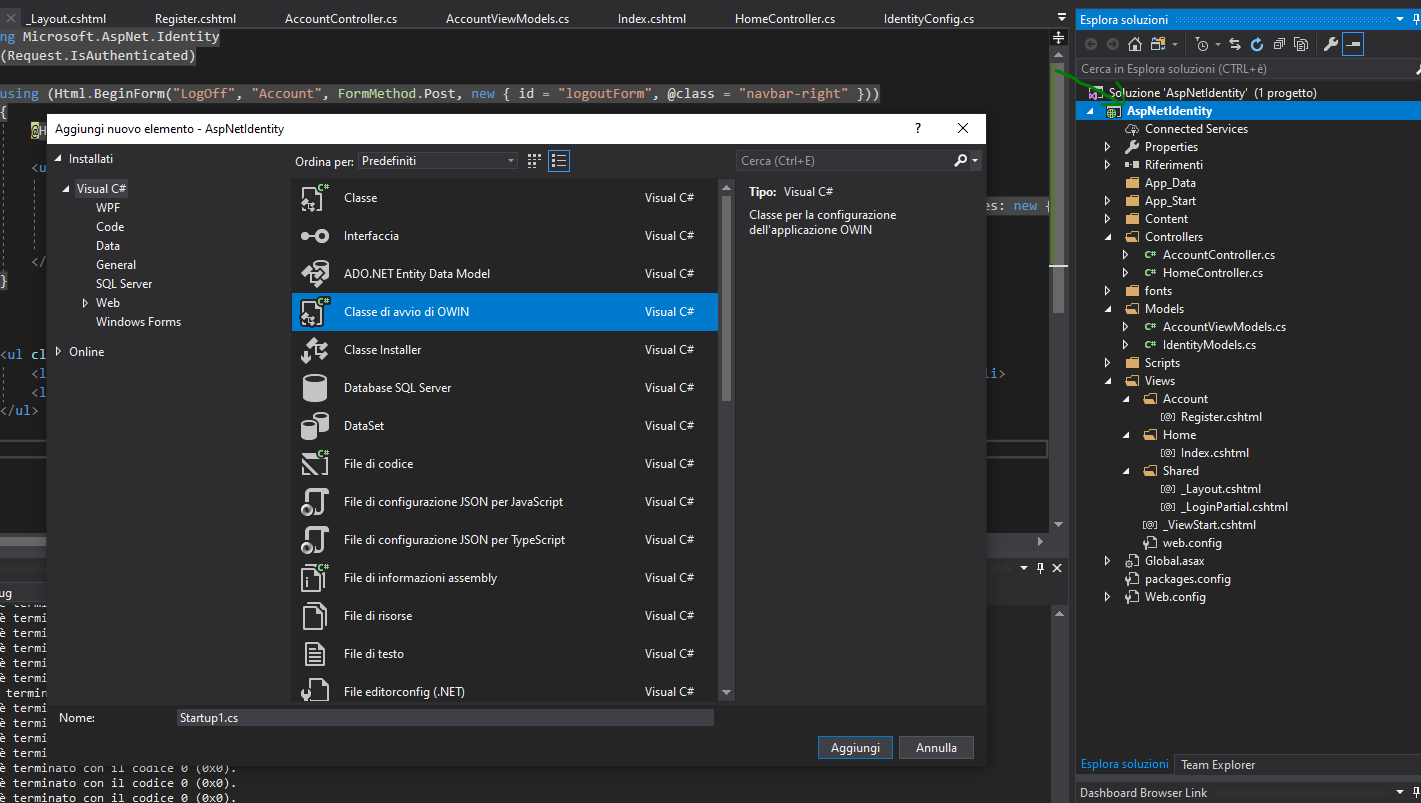
Install-Package Microsoft.Owin.Host.systemWeb

Every OWIN enabled Application must have one startup class where you need to register the **OWIN Middleware** used by the application pipeline.

The Startup class must have **Configuration(IAppBuilder)** method.

This method is then executed by the OWIN runtime.

OWIN Startup class can be added by selecting the root folder of the Project and clicking on Add -> Add New Item. Select the OWIN Startup class from the list of Options. Enter the Name as Startup and click on Add. This will add the Owin startup class to the project.



We can use the OWIN context objects to store the user Manager object and retrieve it whenever it is needed. This is done in the ***CreatePerOwinContext*** method of the ***AppBuilder*** class. The Instance of the AppBuilder is supplied by the Host at the runtime.

The ***CreatePerOwinContext*** creates a single instance of the object for each request handled by the application. The Instance is tied to the request lifecycle.

This instance is then stored in the OwinContext and can be used throughout the application. You can access the instance from the Context.Get() method.

The app.CreatePerOwinContext<T>() method registers a callback method, which will return the new instance of the specified type.

## Sign in Manager

**ignInManager** is a concrete class which handles the user sign in from the application.

**SignInManager** takes **ApplicationUser**and **authenticationManager** in its constructor.

n the code below we create **ApplicationSignInManager** inheriting from the **SignInManager <TUser,TKey>.**

The cookie authentication is now handled by the **Microsoft.owin.Security.Cookies** middleware

We use extension method **UseCookieAuthentication** to configure this middleware.

The ***UseCookieAuthentication*** method takes the AuthenticationType parameter where we need to specify the Authentication Type we are going to use. We also need to specify the ***loginpath*** which is the path where unauthenticated users are redirected.

We make a call to PasswordSignInAsync method of the SignInManager.  This method validates the user and issues authentication cookie. It Returns SignInStatus which indicates whether the user was able to log in or not

## ASP NET Core Identity

ASP NET Core Identity is used for managing user credentials in the ASP.NET environment.

It provides the necessary functionality to manage users, roles, claims etc.