Implementing ASP.NET Identity

Monday, January 20, 2014

5 comments

In previous posts (listed below), we saw how the UserManager class in ASP.NET Identity provides the domain logic for an identity and membership system. Your software makes calls into a UserManager object to register and login users, and the UserManager will then call into a UserStore object to persist and retrieve data.

Microsoft's UserStore class uses the Entity Framework for persistence. If you don't like or can't use Microsoft's UserStore class, then implementing the storage interfaces for ASP.NET identity is **easy**.

The goal of a custom storage class is to provide the basic CRUD operations required by the features your application needs. Since the storage features are factored into granular interfaces, your storage class can pick and choose the interfaces it needs to implement.

Here are the current interfaces you can choose from.

IUserStore<TUser, TKey>

The one required interface in the identity system is IUserStore. In the 2.0 alpha release, a TKey generic parameter allows you to specify the type of the identifier / primary key for a user, which was assumed to be a string in v1.0. This interface has 5 simple CRUD requirements:

- Create a user
- Update an existing user
- Delete a user
- Find a user by ID
- Find a user by username

These methods each require < 10 lines of code, and the count of 10 includes checks for valid parameters and premature disposal. Each method only needs to forward data to a data framework or existing API. For example, a simple implementation of CreateAsync with the Entity Framework might look like the following (where _users is a DbSet<TUser>).

```
public Task CreateAsync(User user)
{
    user.Id = Guid.NewGuid();
```

```
    _users.Add(user);
    return _db.SaveChangesAsync();
}
```

And with MongoDB, users would probably be stored in a MongoCollection:

```
public Task CreateAsync(User user)
{
    user.Id = ObjectId.GenerateNewId();
    _db.Users.Insert(user);
    return Task.FromResult(0);
}
```

All other interfaces derive from IUserStore and add additional functionality. The following interfaces also take TUser and TKey generic type arguments. The generic arguments are omitted from the headings for aesthetic reasons.

IUserPasswordStore

Implement this interface in your custom user store if you want to store users with local hashed passwords. The interface will force you to implement methods to:

- Get a user's password
- Set a user's password
- Check if a user has a password

IUserLoginStore

Implement this interface if you want to store 3rd party user logins, like logins from Twitter, Facebook, Google, and Microsoft. Again, the interface only requires some simple CRUD operations.

- Add a login for a user
- Find a user given their login data
- Get all logins for a user
- Remove a login for a user

IUserClaimStore

Manage System.Security.Claim information for users with three simple methods.

- Get all claims for a user
- Add a claim to a user
- Remove a claim from a user

IUserRoleStore and IRoleStore

Implement IUserRoleStore if you want to associate roles with each user. There are a total of 4 methods required.

- Add a user to a role
- Get all the roles for a user
- Check is a user is in a specific role
- Remove a user from a role

The IRoleStore interface, like IUserStore, is a storage API with CRUD operations for role management. You'll want to implement this interface and pass it to the ASP.NET Identity RoleManager.

IUserStore goes with UserManager; IRoleStore goes with RoleManager.

The IRoleStore interface requires 4 operations.

- Create a role
- Delete a role
- Update a role
- Find a role by ID or name

IUserSecurityStampStore

The security stamp is best explained in a stackoverflow.com answer from team member Hao King.

> … this is basically meant to represent the current snapshot of your user's credentials. So if nothing changes, the stamp will stay the same. But if the user's password is changed, or a login is removed (unlink your google/fb account), the stamp will change. This is needed for things like automatically signing users/rejecting old cookies when this occurs, which is a feature that's coming …

The interface requires only two methods.

- Get the security stamp for a user
- Set the security stamp for a user

IUserEmailStore and IUserConfirmationStore

New in Identity 2.0 are the abilities to confirm users via a token and allow for users to reset their password. To offer both features you'll want these 2 interfaces. When combined they require the following operations.

- Find a user by email address
- Get a user's email address

- Set a user's email address
- Check if a user is confirmed
- Set a user's confirmation flag

IUser*Store Interfaces and UserManager

When you create a new instance of a UserManager you pass in an object implementing at least IUserStore and 0 or more of the other IUser*Store interfaces. If you ask the user manager to do something that isn't supported (like by calling FindByEmailAsync when your custom user store doesn't support IUserEmailStore), the manager object throws an exception.

Working Implementations

There are a few OSS projects out there already providing IUser*Store implementations with various storage mechanisms. You can NuGet these implementations into a project, or peruse the source to get an idea of how to implement a custom user store.

Tugberk Ugurlu has an implementation for RavenDB: AspNet.Identity.RavenDB

Daniel Wertheim has an implementation for CouchDB / Cloudant

SoftFluent has an implementation for CodeFluent Entities.

InspectorIT has an implementation for MongoDB: MongoDB.AspNet.Identity

Stuart Leeks just published a store using Azure Table Storage: AspNet.Identity.TableStorage

ILMServices has an implementation for RavenDB, too: RavenDB.AspNet.Identity

Antônio Milesi Bastos has built a user store using NHibernate: NHibernate.AspNet.Identity

Bombsquad AB provides a user store for Elastic Search: Elastic Identity

aminjam built a user store on top of Redis: Redis.AspNet.Identity

cbfrank has provided T4 Templates to generate EF code for a "database first" user store: AspNet.Identity.EntityFramework

Previous Posts

- ASP.NET Identity with the Entity Framework
- ASP.NET Core Identity
- Customization Options With ASP.NET Identity