Implementing the Nuts & Bolts of ASP.NET Core Identity

Scott Brady
IDENTITY & ACCESS CONTROL LEAD
@scottbrady91   www.scottbrady91.com

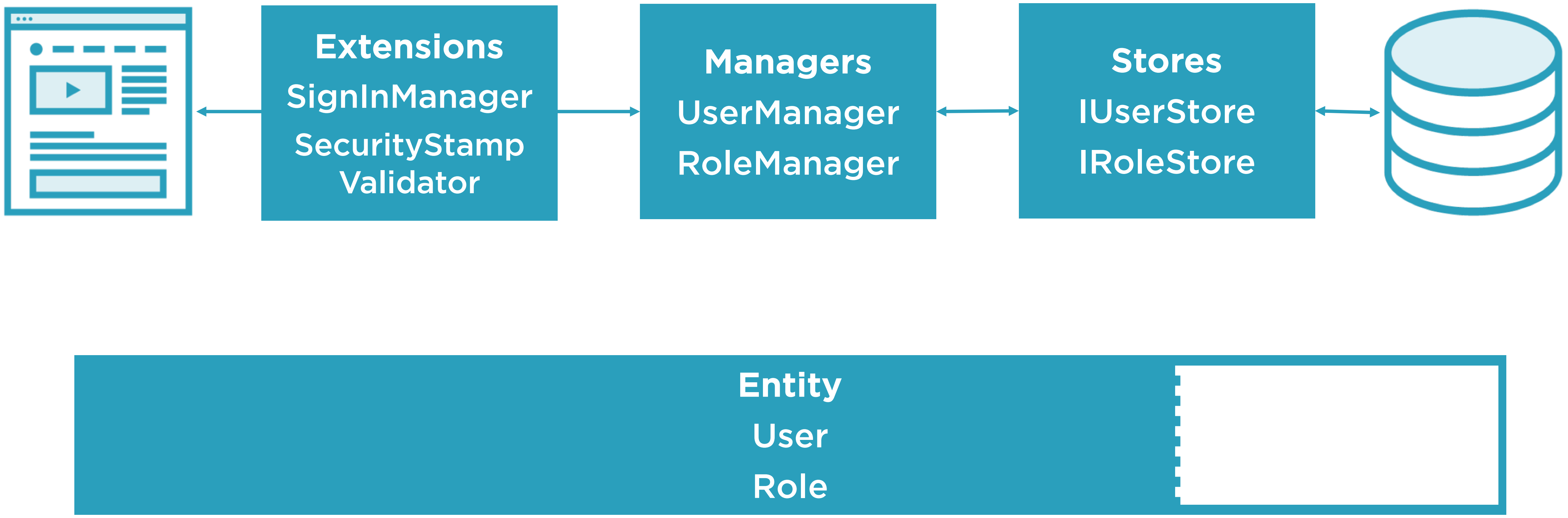
Overview

Library architecture & structure

IUserStore

Implementing ASP.NET Core Identity from scratch

ASP.NET Identity Structure



ASP.NET Identity Package Structure

Microsoft.Extensions.Identity.Core

Microsoft.AspNetCore.Identity

Microsoft.Extensions.Identity.Stores

Microsoft.AspNetCore.Identity
.EntityFrameworkCore

UserManager<TUser>

```
public UserManager(IUserStore<TUser> store,
  IOptions<IdentityOptions> optionsAccessor,
  IPasswordHasher<TUser> passwordHasher,
  IEnumerable<IUserValidator<TUser>> userValidators,
  IEnumerable<IPasswordValidator<TUser>> passwordValidators,
  ILookupNormalizer keyNormalizer,
  IdentityErrorDescriber errors,
  IServiceProvider services,
  ILogger<UserManager<TUser>> logger) { ... }
```

RoleManager<TRole>

```
public RoleManager(IRoleStore<TRole> store,
  IEnumerable<IRoleValidator<TRole>> roleValidators,
  ILookupNormalizer keyNormalizer,
  IdentityErrorDescriber errors,
  ILogger<RoleManager<TRole>> logger) { ... }
```

IUserStore<TUser>

```
Task<IdentityResult> CreateAsync(TUser user, CancellationToken token);

Task<IdentityResult> UpdateAsync(TUser user, CancellationToken token);

Task<IdentityResult> DeleteAsync(TUser user, CancellationToken token);

Task<TUser> FindByIdAsync(string userId, CancellationToken token);

Task<TUser> FindByNameAsync(string normalizedUserName, CancellationToken token);

Task<string> GetUserIdAsync(TUser user, CancellationToken token);

Task<string> GetUserNameAsync(TUser user, CancellationToken token);

Task SetUserNameAsync(TUser user, string userName, CancellationToken token);

Task<string> GetNormalizedUserNameAsync(TUser user, CancellationToken token);

Task SetNormalizedUserNameAsync(TUser user, string normalizedName,
  CancellationToken token);
```

But wait, there’s more!

`IUserPasswordStore`	◀ Password hashes
`IUserEmailStore`	◀ Emails & confirmation
`IUserPhoneNumberStore`	◀ Phone & confirmation
`IUserRoleStore`	◀ Links users & roles
`IUserClaimStore`	◀ Additional claims for users (freeform)
`IUserLockoutStore`	◀ Lockout functionality (e.g. failed attempts)
`IUserSecurityStampStore`	◀ Security stamps
`IQueryableUserStore`	◀ IQueryable access to underlying database
`IUserLoginStore`	◀ External logins (think social)
`IUserAuthenticationTokenStore`	◀ Storage of tokens from external logins
`IUserAuthenticatorKeyStore`	◀ e.g. Google Authenticator
`IUserTwoFactorStore`	◀ Multi-Factor Authentication (MFA)
`IUserTwoFactorRecoveryCodeStore`	◀ MFA recovery

```
internal IUserLockoutStore<TUser> GetUserLockoutStore() {
    var cast = Store as IUserLockoutStore<TUser>;
    if (cast == null) {
        throw new NotSupportedException();
    }
    return cast;
}
```

User Store Support

“All UserManager methods first check to see if the User Store implements the required interfaces”

Demo

Registering ASP.NET Core Identity

Creating a custom user store

User creation & authentication

Mocking the UserManager

```
var store = new Mock<IUserStore<TUser>>();

var mgr = new Mock<UserManager<TUser>>(store.Object, null, null,
null, null, null, null, null, null);

mgr.Object.UserValidators.Add(new UserValidator<TUser>());

mgr.Object.PasswordValidators.Add(new PasswordValidator<TUser>());
```

Summary

Library architecture & structure

IUserStore

Implementing ASP.NET Core Identity from scratch