WEB524

WEB PROGRAMMING ON WINDOWS

WEEK 7 – LECTURE 1

ASP.NET SECURITY

Today

- At this point, you have learned the foundations to build interactive web apps that work with persistent data.
- Today we will begin to discuss security:
  - Identity management
  - Authentication
  - Authorization
  - Interactive workflows/scenarios that involve security.

Terminology

ASP.NET Identity

- Web apps that you create with Visual Studio use a security system named ASP.NET Identity.
- Its components provide identity management (i.e. user account creation and maintenance), authentication, and authorization.
- ASP.NET Identity is an implementation of widely-used and standards-based approaches to web app security (OAuth2 and OpenID Connect).

Terminology

User Account

- An object that represents a *human* user of a web app.
- Includes identification properties such as a user name and a shared secret (i.e. a password)
- Includes descriptive property such as email address and first/last name.

Terminology

Authentication

- The process of presenting and validating credentials.
- A web app that's configured to use ASP.NET Identity includes the components needed for authentication, including a login page.
- Often abbreviated to AuthN (or authN)

Terminology

Security Principle

- After authentication, the ASP.NET Identity system creates a security principal object and attaches it to the execution context.
- Among other data, this object includes **claims** (descriptive pieces of information about the user).
- The web app returns an authentication cookie in the response to the browser. By convention, the browser will include the cookie in every subsequent request to the web app.

Terminology

Authorization

- After a user authenticates successfully (i.e. logs in), user's claims determine whether they are authorized to perform tasks and access resources in the web app.
- Often abbreviated to AuthZ (or authZ).

Transition Your Knowledge

- As a user, you already have experience with web apps that have security components.
- As a web programmer, you have some experience with security-related coding tasks. In WEB322:
  - You designed and created a credential store to hold usernames and passwords.
  - You coded a login form and the credential validation (authentication) process.
  - An access control list based scheme was used to protect resources.
- In this course:
  - We will use built-in components for all security tasks.
  - There are several customizations that can be made to handle the unique needs of your app.

Recommendation

Do not write your own security infrastructure for web apps!

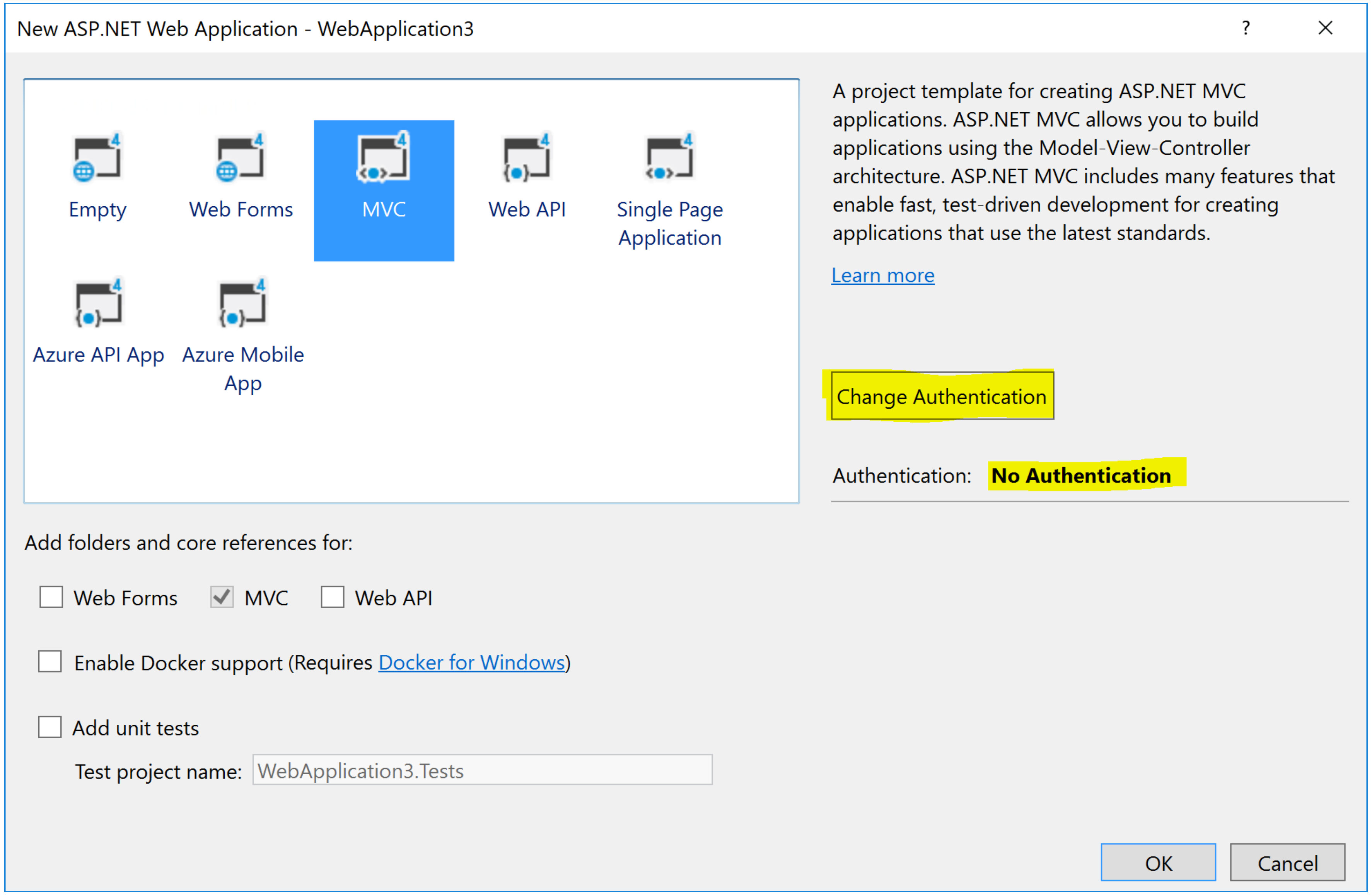
- Not just in this course but also in *real-life*.
- Take advantage of standards-based approaches.
- Many talented and trusted programmers put a lot of time and effort into improving/perfecting modern security infrastructure components.

Hands-on Demo

Start Visual Studio and create a new web project

ASP.NET MVC App With Security

- For the next few weeks, you will no longer use the “Web App Project Template v1” when writing assignments.
- Visual Studio offers a few authentication methods:
  - No Authentication (we have been using this up until today)
  - Individual User Accounts (we will use this for the next few weeks)
  - Work and School Accounts (previously known as Organizational Accounts)
  - Windows Authentication



New ASP.NET Web Application - WebApplication3

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Change Authentication

Authentication: **No Authentication**

Add folders and core references for:

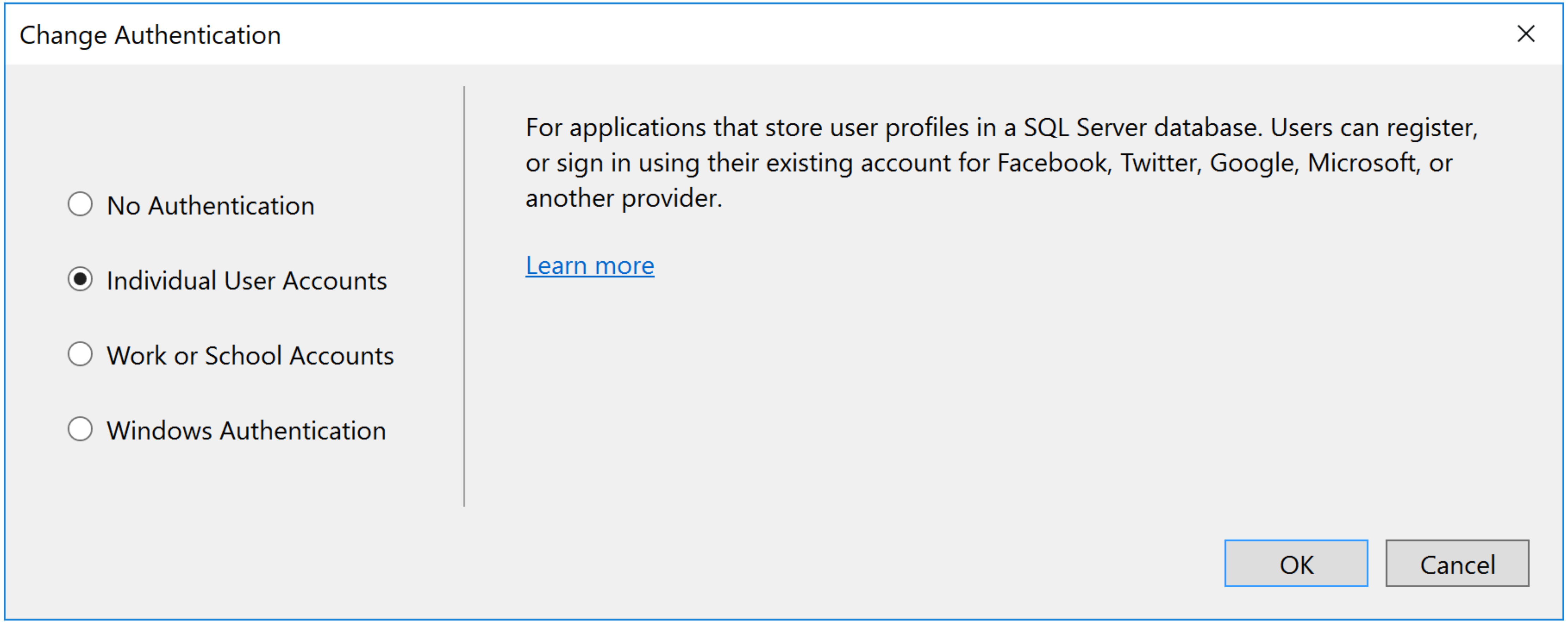
- [ ] Web Forms
- [✓] MVC
- [ ] Web API

- [ ] Enable Docker support (Requires Docker for Windows)

- [ ] Add unit tests

Test project name: WebApplication3.Tests

OK Cancel


Change Authentication

( ) No Authentication
(●) Individual User Accounts
( ) Work or School Accounts
( ) Windows Authentication

For applications that store user profiles in a SQL Server database. Users can register, or sign in using their existing account for Facebook, Twitter, Google, Microsoft, or another provider.

Learn more

Individual User Accounts

- We will use this option for our new web apps.
- The app will use ASP.NET Identity for identity management and authentication.
- Individual User Accounts provide two ways for a user to authenticate:
    1. Local login
    2. Social login

Local Login

- The user can create a user account.
- The app stores user account information (including a password hash) in the database.
- Local components perform authentication.

Social Login

- The user authenticates with an external service such as Facebook, Microsoft, or Google.
- The app creates a user account for the user in the local database but does not store the password hash.

Individual User Accounts

- For both local and social login, ASP.NET Identity uses standards-based OAuth2 for authentication.
- The database is managed by *Entity Framework Code First* technologies and all of the tables are represented by entity classes that can be modified.
- You can easily customize the security- and profile-related data to fit your app's needs.
- ASP.NET Identity is a good choice if you are creating a public-facing web site which is mainly for users on the web.

Work and School Accounts

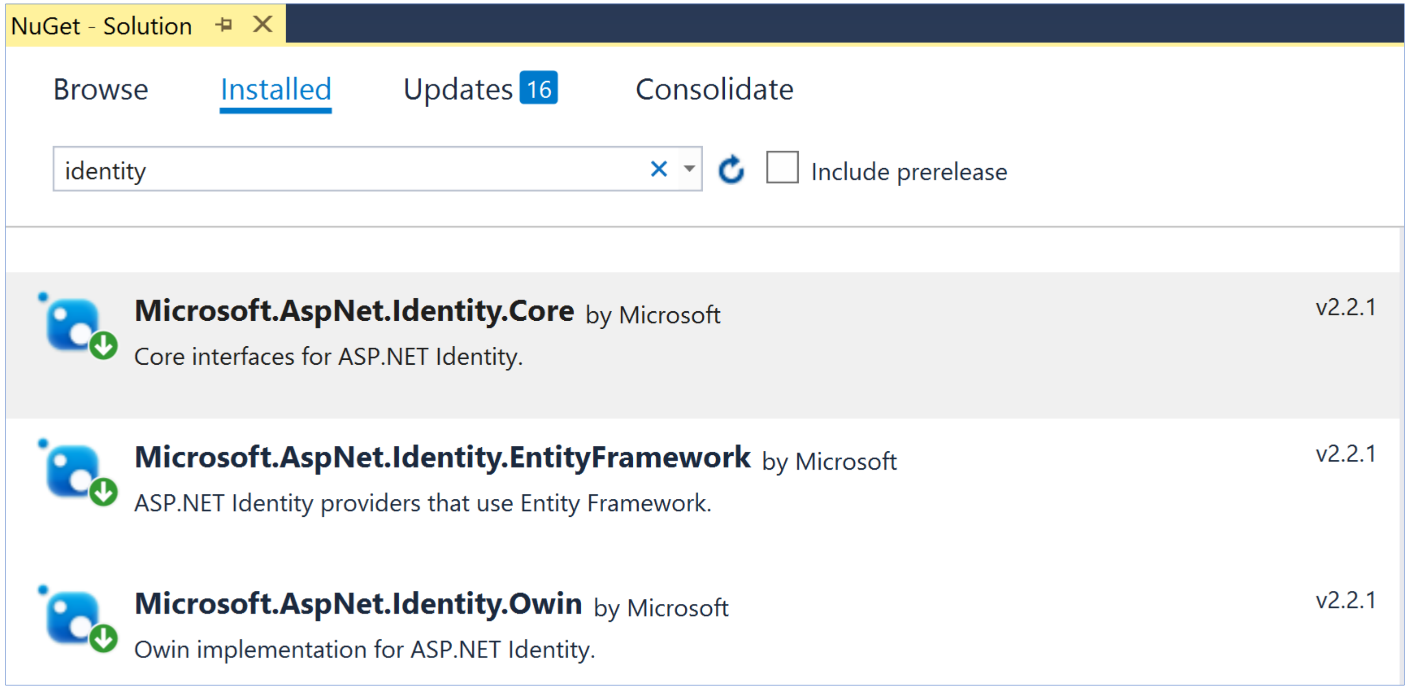
- Previously known as *Organizational Accounts*.
- A good choice if your organization uses *Microsoft Active Directory* or *Microsoft Office 365*
- Enables single-sign-on for employees and business partners.

Windows Authentication

- A good choice if your organization is creating an internal-facing app to be used ONLY by employees who have *Active Directory* accounts.
- Seneca College uses an Active Directory
  - If you deploy your web app using Windows Authentication, users can log in with their Seneca username and password.
  - Typically when logged in to your computer (within the Active Directory), you are not prompted to log in.

ASP.NET Identity Components

- If you open the **NuGet Package Manager** and search for "Identity", you will see at least three installed packages.



Microsoft.AspNet.Identity.Core by Microsoft v2.2.1
Core interfaces for ASP.NET Identity.

Microsoft.AspNet.Identity.EntityFramework by Microsoft v2.2.1
ASP.NET Identity providers that use Entity Framework.

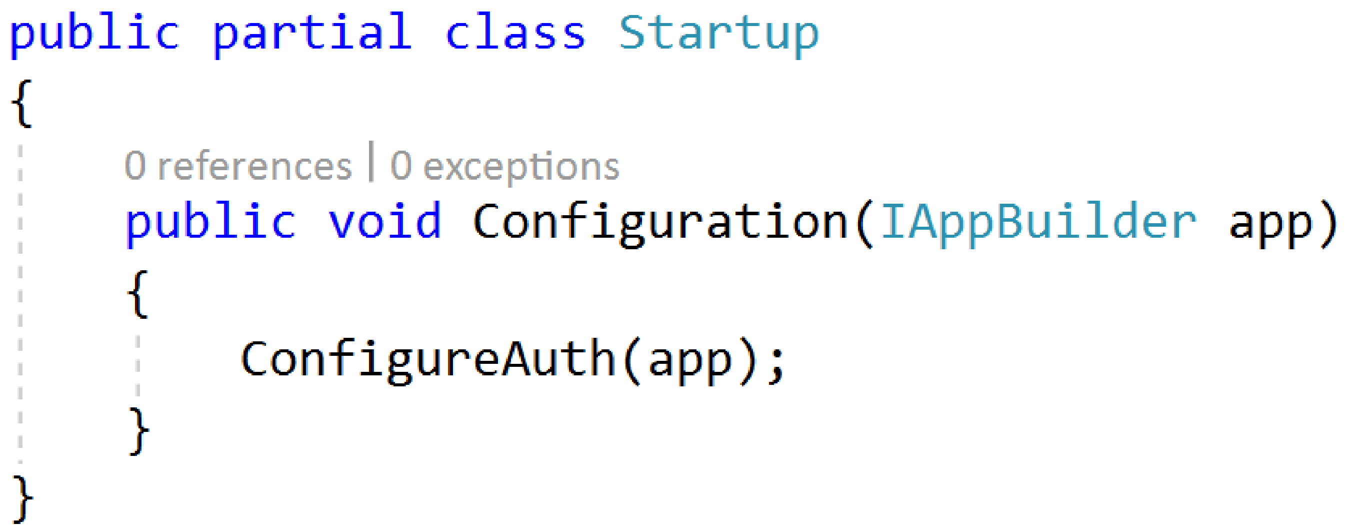
Microsoft.AspNet.Identity.Owin by Microsoft v2.2.1
Owin implementation for ASP.NET Identity.

ASP.NET Identity Components

- The Microsoft-provided components are visible in the **NuGet Package Manager** or as “assemblies” in the **References** branch of your project in Solution Explorer.
- Notable assemblies include a number of **Microsoft.AspNet.Identity** assemblies and others in the **Microsoft.Owin.Security** group.
- Other security-related components can be seen in source code files.
- **OWIN** - Open Web Interface for .NET

Initial Configuration of Security

- Look at the **Startup** class in the root of the project.
- When the app loads for the first time, it instantiates this class and runs the **Configure()** method
- By default, the method simply calls the **ConfigureAuth()** method



```
public partial class Startup
{
    0 references | 0 exceptions
    public void Configuration(IAppBuilder app)
    {
        ConfigureAuth(app);
    }
}
```

Loading the Components at App Startup

- The class' implementation is split over two source code files.
- Notice the "partial" keyword in the class definition statement.
- The other part of the implementation is in the App_Start/Startup.Auth.cs code file.
- Its **ConfigureAuth()** method loads and configures the security components.

ConfigureAuth()

It does several very important tasks:

1. Open and use the data store (i.e. the LocalDB database):

```
app.CreatePerOwinContext(ApplicationDbContext.Create);
```

2. Load the “user manager” and authentication (“sign in manager”) components:

```
app.CreatePerOwinContext<ApplicationUserManager>
     (ApplicationUserManager.Create);

app.CreatePerOwinContext<ApplicationSignInManager>
     (ApplicationSignInManager.Create);
```

   The user and sign in manager classes are defined in a companion source code file,App_Start/IdentityConfig.cs.

ConfigureAuth()

3. Load / activate and configure HTTP “cookie authentication” for the app:

```
app.UseCookieAuthentication(new
CookieAuthenticationOptions…
```

Locate the security-related persistence components

- In Solution Explorer, open the Models folder to locate the **IdentityModels.cs** code file.
- Its classes enable the creation, storage, and management of user accounts.

ApplicationUser

- The **ApplicationUser** class models a user account.
- We can extend the class by adding our own descriptive properties.
- Check out the existing properties in its superclass **IdentityUser<TKey, TLogin, TRole, TClaim>**
- Right-click on IdentityUser and choose “Go To Definition” (or F12) to see all of the properties.

ApplicationDbContext

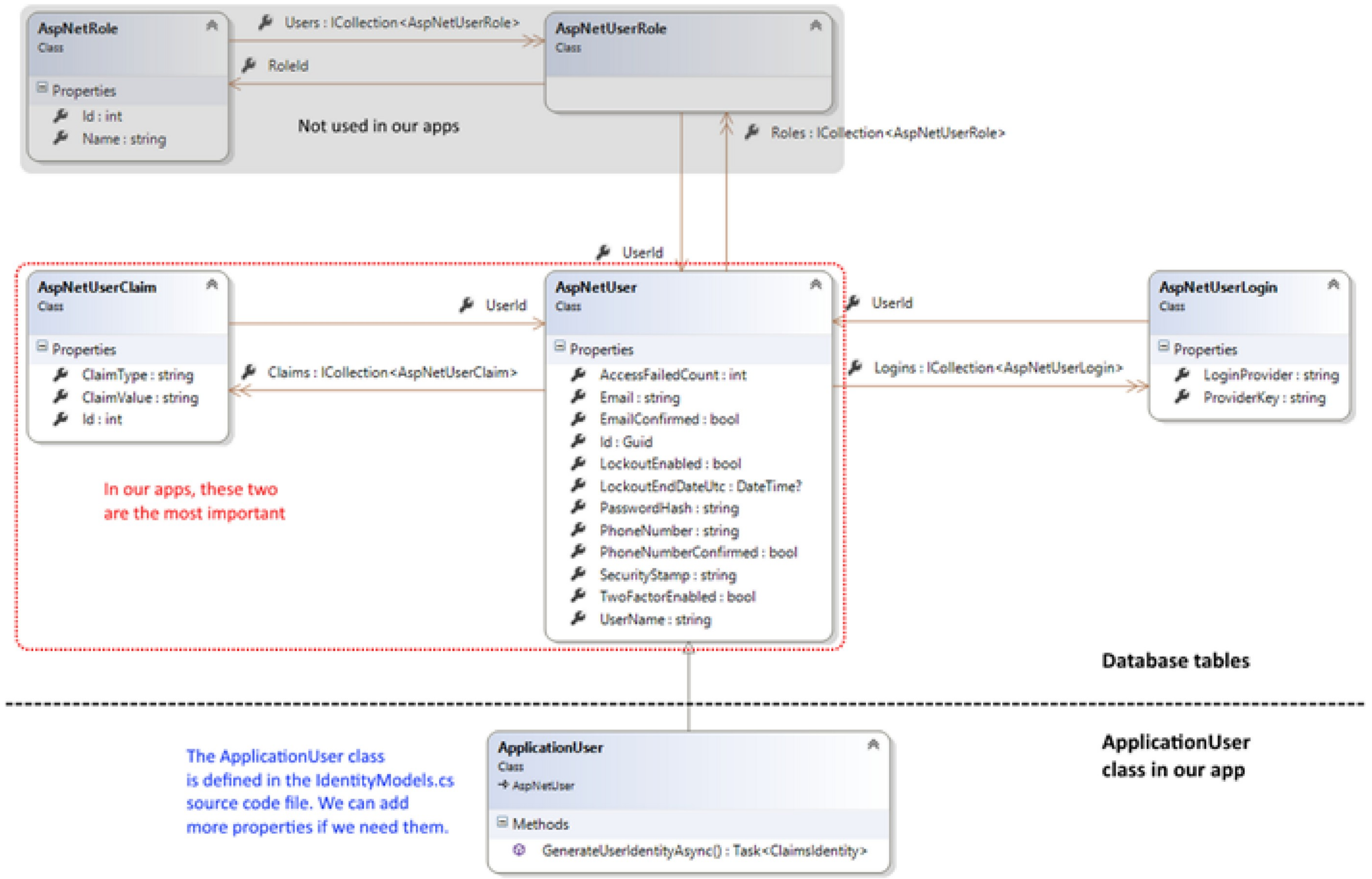
- The **ApplicationDbContext** class manages the persistent data store (the database).
- We can extend this class by adding our own **DbSet<TEntity>** properties for our web app's entity collections.
- The class inherits from **IdentityDbContext<TUser>** and ultimately **DbContext** just like the data context that we have been using for the past several weeks.

Account & Manage View Models

- Locate the **AccountViewModels.cs** and **ManageViewModels.cs** source code files.
- **AccountViewModels.cs**
  - Includes the view model classes needed by the “Account” controller.
  - Used for the user interface views such as register, login, and forgot password.
  - Typically used for tasks done by unauthenticated users.
- **ManageViewModels.cs**
  - Includes the view model classes needed by the “Manage” controller.
  - Used for the user interface views such as change password.
  - Typically used for tasks done by authenticated users.

Database Schema

- When the database is accessed for the first time, the Entity Framework infrastructure creates a number of tables in the database which support identity management.
- All of the tables begin with the prefix “AspNet”.
- The most important tables include:
  - AspNetUser
  - AspNetUserClaim



The ApplicationUser class is defined in the IdentityModels.cs source code file. We can add more properties if we need them.

Security-related controller, view model, and view components

- In Solution Explorer, open the Controllers folder.
- **Account** controller
  - Has actions that enable a user to register for a new account, login, and do other tasks.
  - It is typically used for tasks done by unauthenticated users.
- **Register** action
  - They enable a user to register/create a new user account.
  - Review the Register method pair.
  - Review the view model that support this task (RegisterViewModel).
  - Review the Register.cshtml view (in Views/Account folder) - this renders the UI for new account registration.

Security-related controller, view model, and view components

- **Login** action
  - Enables a user to login/authenticate.
  - Review the Login method pair.
  - This line of code performs authentication:
    `var result = await SignInManager.PasswordSignInAsync…`
  - Review the view models that support this task (LoginViewModel).
  - Review the Login.cshtml view (in Views/Account folder) - this renders the UI for login.
- **Manage** controller
  - Includes tasks that are done by authenticated users such as password change and password reset.

Shared view components

- In the Views/Shared folder, there is a new _LoginPartial.cshtml view.
- It holds markup and code for rendering content on the _Layout.cshtml view.
- If the user is authenticated, it displays links that enable the user to manage their account and logout.
- If the user is not authenticated, it displays links for new account registration and login.

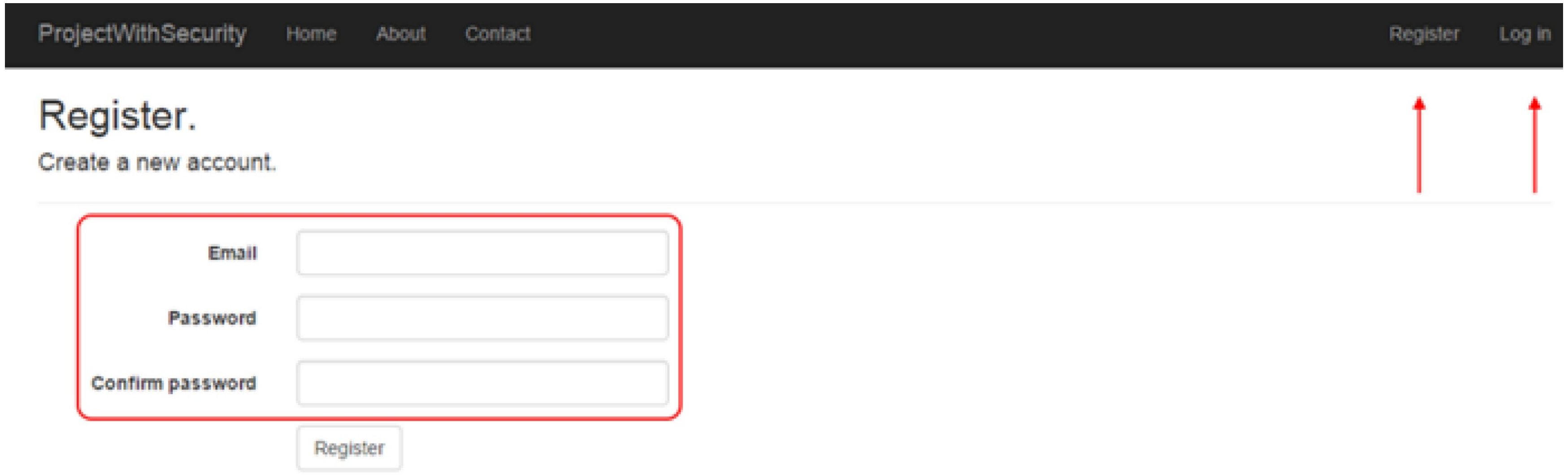
Introduction to common security-related workflows

Common Security-Related Workflows

- There are three common security-related workflows in a web app:
  1. New user account registration.
  2. Login / authentication.
  3. Protecting and accessing protected resources.

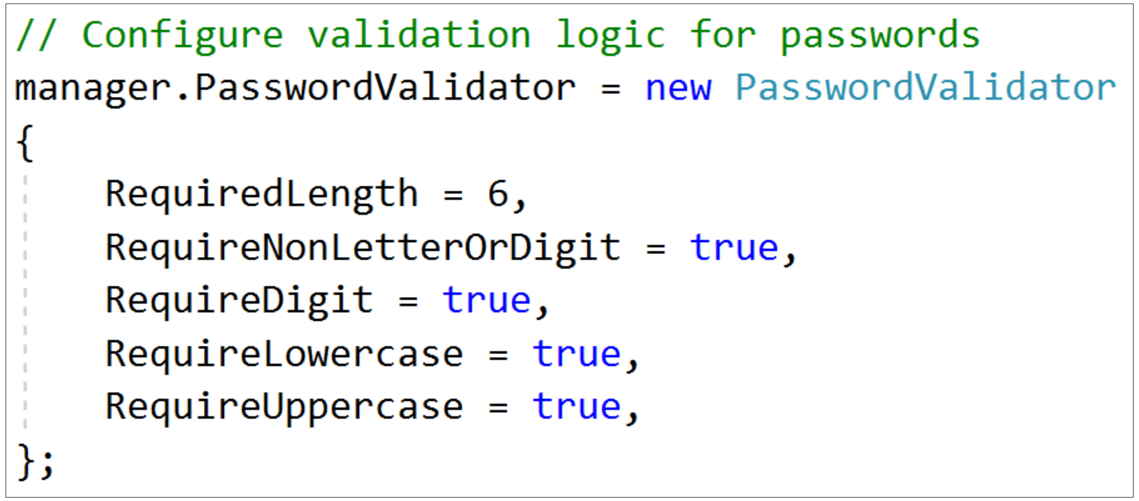
New user account registration

- A new web app (with security) does NOT include any user accounts.
- After loading the app, notice the Register link in the upper-right area of the page. Click it to navigate to a new account registration page.



ASP.NET Default Configuration

- The username is an email address. It must be a unique value so it can be reliably used as a unique identifier.
- The password must follow some rules, found in the **PasswordValidator** property, in the **Create()** method of the **ApplicationUserManager** class (in App_Start/IdentityConfig.cs).



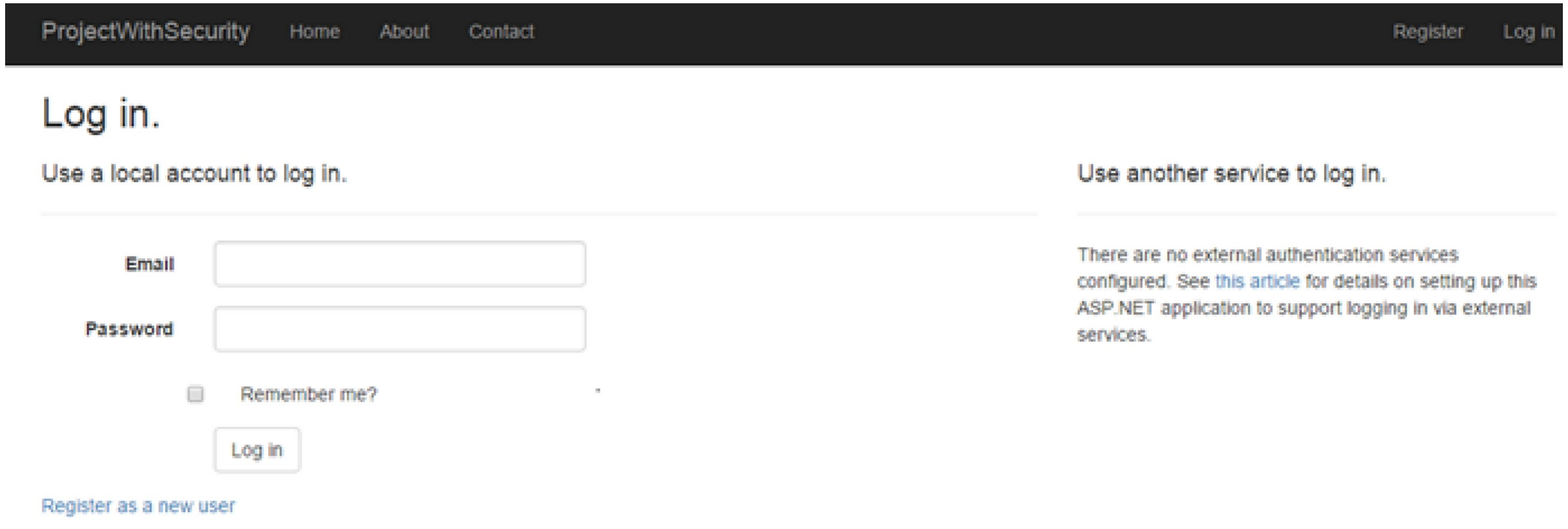
```
// Configure validation logic for passwords
manager.PasswordValidator = new PasswordValidator
{
    RequiredLength = 6,
    RequireNonLetterOrDigit = true,
    RequireDigit = true,
    RequireLowercase = true,
    RequireUppercase = true,
};
```

Security Profiles

- The user accounts that are created all have the same security profile.
- No account is more powerful or privileged than another.
- We can use claims to fix this.

Login / Authentication

- At the end of the new account registration process, the app authenticates you and you are logged in.
- After logging out, you can use the login page to reauthenticate.



Log in.

Use a local account to log in.

Email

Password

Remember me?

Log in

Register as a new user

Use another service to log in.

There are no external authentication services configured. See this article for details on setting up this ASP.NET application to support logging in via external services.

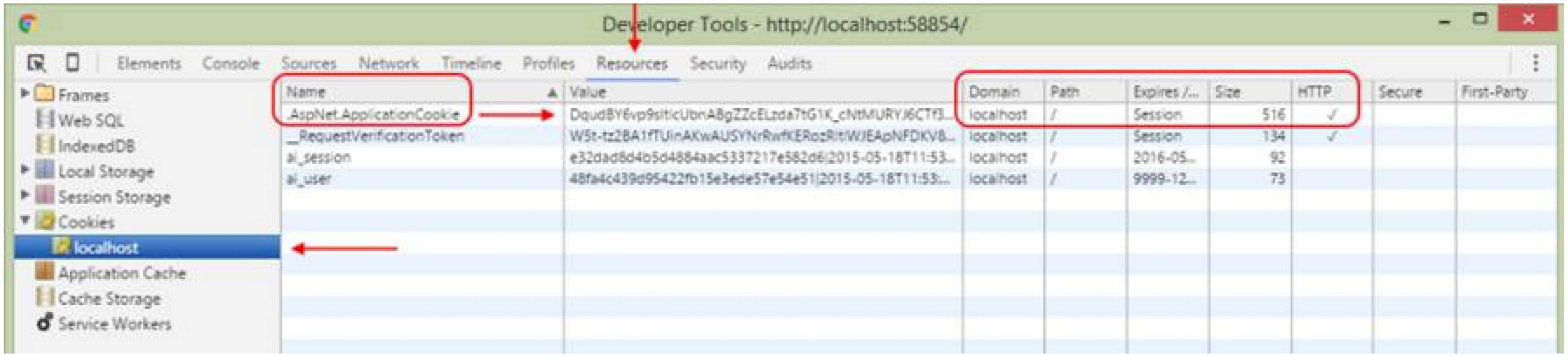
Login / Authentication

- Remember: authentication is the process of presenting and validating credentials.
- During authentication, ASP.NET Identity:
  - Creates a security principal object and attaches it to the execution context.
  - The authentication process will create a data package (an HTTP cookie) and return it in the response to the browser.
  - The data package includes information about the authenticated user (like username and claims) but it does not include sensitive or secret data.
- On subsequent requests, the browser automatically sends the data package (HTTP cookie) back to the server.

Login / Authentication

- You can quickly determine whether you have successfully authenticated… Look in the upper-right area of the page and it will show a greeting and a logout link if you are signed in.
- You can use the web browser developer tools – F12 in any browser – to inspect the HTTP cookie. For example, using Google Chrome:
    1. Press F12 to open the Developer Tools panel or window.
    2. Select the “Resources” item on the menu.
    3. On the left-side navigator, open Cookies, then localhost. You will see a cookie named “.AspNet.ApplicationCookie“. The browser will include the cookie with every subsequent request to the web app.

Chrome Developer Tools



Name	Value	Domain	Path	Expires /...	Size	HTTP	Secure	First-Party
.AspNet.ApplicationCookie	DqudBY6vp9slticUbnABgZZcELzda7tG1K_cNtMURYJ6CTf3...	localhost	/	Session	516	✓		
__RequestVerificationToken	W5t-tz2BA1fTUinAKwAUSYNrRwfKERozRltWJEApNFDKV8...	localhost	/	Session	134	✓		
ai_session	e32dad8d4b5d4884aac5337217e582d6|2015-05-18T11:53...	localhost	/	2016-05...	92			
ai_user	48fa4c439d95422fb15e3ede57e54e51|2015-05-18T11:53:...	localhost	/	9999-12...	73			

Protecting and accessing protected resources

- When a web app receives a request with an HTTP cookie, the security infrastructure validates the cookie. If valid:
  - It creates an IPrincipal object and attaches it to the request.
  - Using this object, information about the authenticated user is available to your code while the request makes its way through the request-processing pipeline.
  - The user's claims determine whether they are authorized to perform tasks and access resources in the web app.
- **[Authorize]** attribute
  - Can be placed on the controller class declaration and/or on its methods.
  - In its simplest form, its presence requires the request (from the browser) to be authenticated. In other words, it must include the HTTP cookie that indicates that authentication has happened.
  - There are other forms that you will learn about later on.

[AllowAnonymous]

- Review the **AccountController** class.
  - It includes an [Authorize] attribute on the class. This means that a request MUST be authenticated to run any method in the class.
  - Notice the Login() and Register() methods. They include the [AllowAnonymous] attribute. This means that ANY request will be able to run these methods.

Security Principal

- A security principal is an object that represents the security context under which the code is running.
- It can be inspected by your program code, so that you have decision-making information in your code path.
- In a controller:
  - The information is in the **User** property, it's simply the top-level **User** property.
  - If you want to determine whether a request is authenticated, the **Request.IsAuthenticated** property will tell you.
- In the manager object:
  - It's the **HttpContext.Current.User** property
  - If you want to determine whether a request is authenticated, the **HttpContext.Current.Request.IsAuthenticated** property will tell you.

SecurityIntro Example

- Open the **SecurityIntro** code example in Visual Studio.
- Review the code as you continue with the following tasks.
- The "TODO" comment tokens (on the Task List) are numbered so sort the list and run through them in sequence.
- Run the app, create a new user account, and login.
- Use the nav menu links to perform the tasks on the menu. Note that you have access to some information but not all the content.

Summary

Security task	Responsibility	Code assets and tasks
Identity management	ASP.NET Identity	• ApplicationUser class • Customized data context (and security-related tables in the app's database) • Account controller and view to register a new user account • Manage controller and view for account management tasks
Authentication	ASP.NET Identity	• Account controller and view for login
Authorization	Web Programmer (You)	• Authorize attribute on controller class and/or actions

Problem

- You learned that the user accounts created in a web app, using ASP.NET Identity, all have the same security profile.
- No account is more powerful or privileged than another.
- For most real-world applications, this is a huge problem! Most apps need user accounts with different security profiles.

Introduction to Claims

- A **claim** is a statement that one subject makes about itself or another subject.
- A **statement** is a piece of descriptive information about a subject.
- A **subject** is a participant of an application.
- A subject could be a user or a corporate body or a programmable object (e.g. a security provider).

Claim Statement Example

- A claim statement, appears in the format: `Claim type = Value`
- The claim type is a string. There's a list of standard, predefined, or "well known" claim types, which are URIs.
- Here are some examples; assume that the subject is a user (and professor), Peter McIntyre.

- User name:
  Known informally as "name".
  For example: name = pmcintyr

Claim Statement Example

- Date of birth:
  Known informally as “dateofbirth”.
  For example: dateofbirth = 1980-05-15T08:00:00Z

- Role:
  Often used as a security-oriented access role
  Known informally as “role” or “role claim”.
  For example:
    - role = employee
    - role = faculty
    - role = coordinator

- Surname:
  Known informally as “surname”
  For example: surname = McIntyre

Claim Statement Example

- A claim type string can also be a simple string instead of a URI. This is useful if you are defining a custom claim type for local use within the app or the app ecosystem.

- Full name:
  The user’s full (readable) name (e.g. first name, and last name) is Peter McIntyre.
  For example: fullname = “Peter McIntyre”

- Driver’s license:
  The user’s driver’s license number is M12345809515.
  For example: driverlicense = “M12345809515″

Claims management and issuance

- While a claim is a statement about a subject, claims are managed and issued by an identity authority (which is the ASP.NET Identity system in our web app).
- A claim can be used by an application to authorize a user to access resources and/or perform tasks.
- For our web app, claims are packaged in an authentication cookie, after a user successfully authenticates.
- The result of a successful authentication is a cookie that (among other data) includes claims.
- Our web app must trust the identity authority.
  - This is done by sharing a cryptographically-strong ‘machine key’ value among the identity authority and your app.
  - If you separate the identity authority and web app, you will have to configure this value. Although we do not do this in this WEB524 course, you may want to do it in the future,

SecurityClaimsIntro Code Example

- Open the SecurityClaimsIntro code example and run the app.
- The "TODO" comment tokens (on the Task List) are numbered so you can sort the list and go through them in sequence.
- It is similar to the SecurityIntro code example above but it has a number of important modifications. These modification will form the base of all our future apps that use security:
  - Claims-aware app
  - Role claims to help with power and privilege questions
  - Descriptive claims to help determine access and capability
  - Modified Register code assets
  - Enables fine-grained access to controller actions

Resources

- Introduction to ASP.NET Identity
- OAuth2 Authorization Framework (and in Wikipedia) (skim/read)
- OpenID Connect (and in Wikipedia) (skim/read)
- Getting Started with OWIN and Katana (skim only, because we use the components differently)
- Wikipedia article on claims-based identity
- Claims-Based Identity Term Definitions
- ASP.NET Identity: ClaimType Fields
- ASP.NET Identity: How to: Create a Custom Claim
- Textbook (Professional ASP.NET MVC 5): Chapter 7
  - Ignore the part on “Windows Authentication”
  - Ignore any content on “roles”, “RoleManager”, and so on
  - Ignore – for now – discussion about external security providers