ASP.NET Core Identity

Implementing login/logout, Scaffolding

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

sli.do

#csharp-web

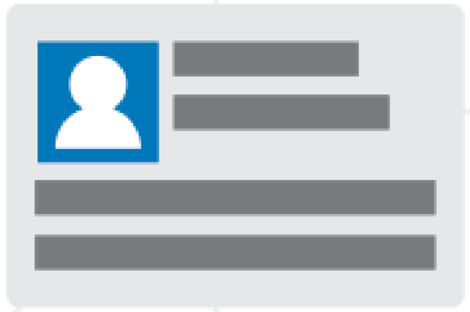
Table of Contents

Authentication vs. Authorization

Authentication vs. Authorization

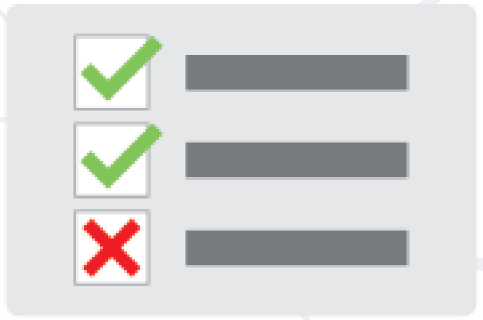
- **Authentication**
  - The process of verifying the identity of a user or computer
  - Prerequisite for authorization
  - Questions: **Who are you**? How you prove it?
  - Credentials can be password, smart card, external token, etc.
- **Authorization**
  - The process of determining what a user is permitted to do on a computer or network
  - Questions: **What are you allowed to do**? Can you see this page?
- You can't authorize a user before authenticating this user

Authentication vs. Authorization



Authentication

Who you are



Authorization

What you can do

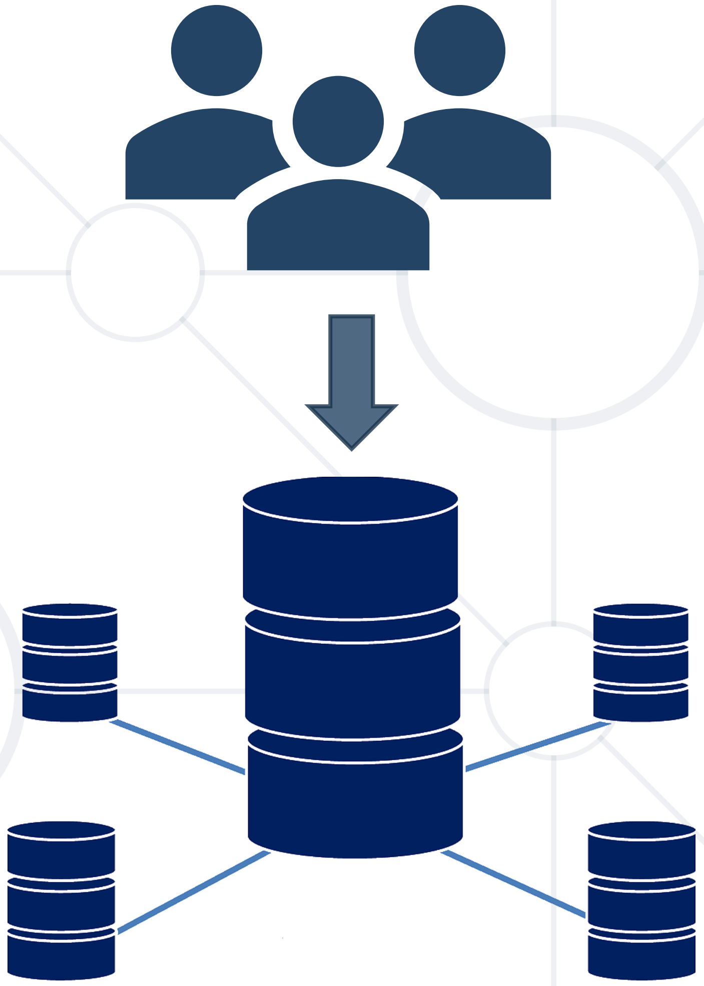
ASP.NET Core Identity

ASP.NET Core Identity

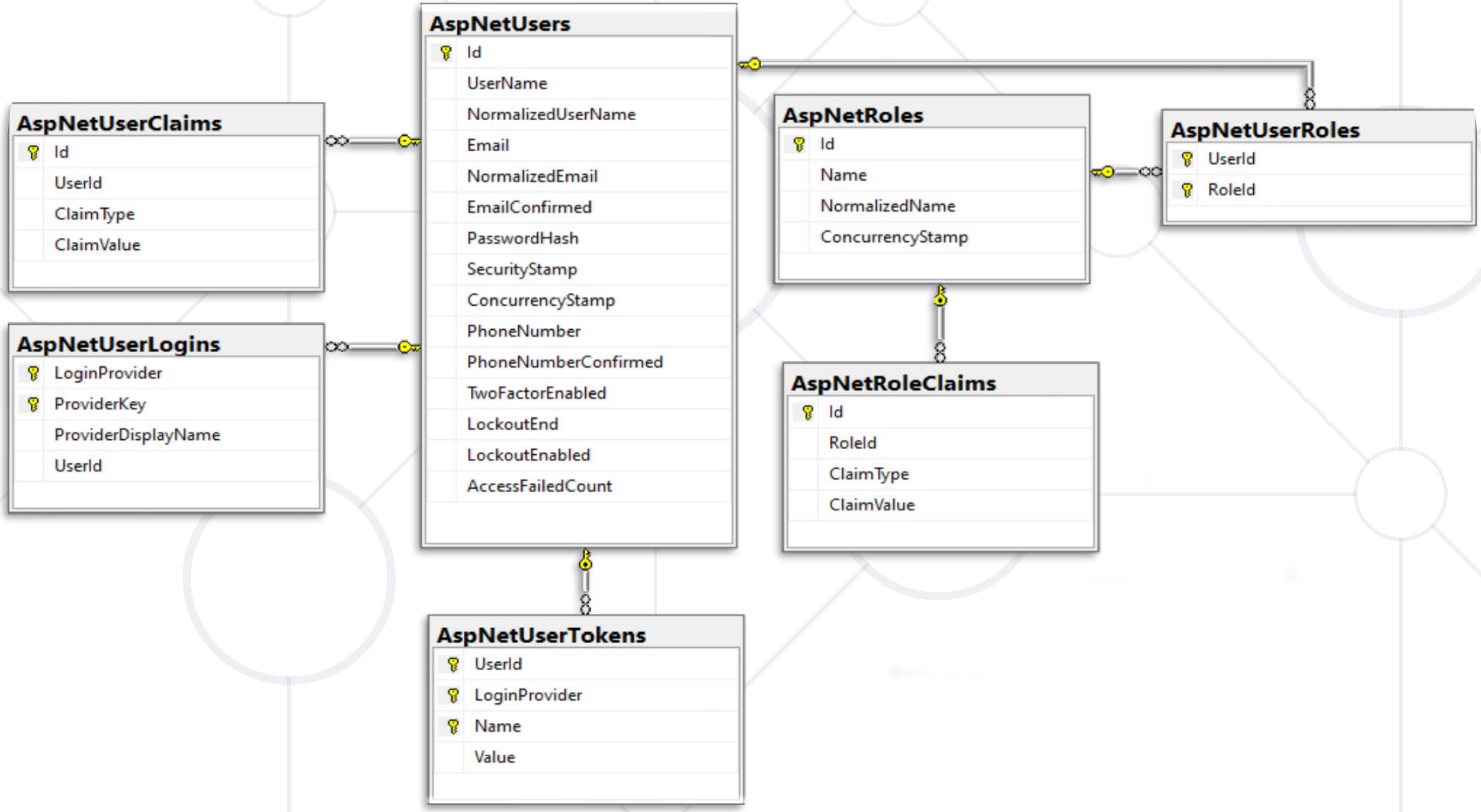
- The **ASP.NET Core Identity** system
  - Authentication and authorization system for ASP.NET Core
  - Supports ASP.NET Core MVC, Pages, Web API (JWT), SignalR
  - Handles **Users**, **User Profiles**, **Login** / **Logout**, **Roles**, etc.
  - Handles cookie consent and GDPR
  - Supports external login providers
    - Facebook, Google, Twitter, etc.
  - Supports database, Azure, Active Directory, Windows Users, etc.

ASP.NET Core Identity

- Typically, the **ASP.NET Core** identity data is stored in relational database
  - Data is persisted using **Entity Framework Core**
  - You have some control over the internal database schema



Internal Database Schema



ASP.NET Core Identity System Setup

- Setup **ASP.NET Identity**
  - Using the ASP.NET **project templates** from Visual Studio
    - And then customize it

✓ RECOMMENDED

  - **By hand**
    - Install NuGet packages, manual configuration, create EF mappings (models), view models, controllers, views, etc.
- Required NuGet package
  - `Microsoft.AspNetCore.Identity.EntityFrameworkCore`

ASP.NET Core Project Template Authentication

- `ApplicationDbContext.cs`
  - Holds the EF data context
  - Provides access to the application's data using model objects
- `Program.cs`
  - Can configure cookie-based (or JWT) authentication
  - May enable external login (e.g., Facebook login)
  - Can change default identity settings
    - More on those in the next course

ASP.NET Core Project Template Authentication

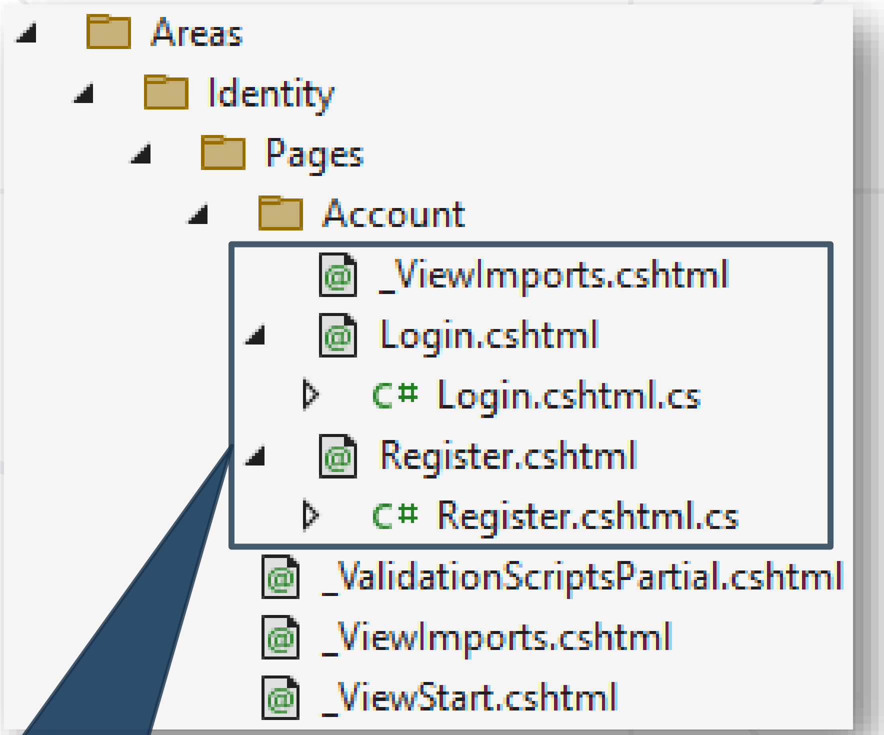
- **Password settings** – can be defined in **`Program.cs`**

```
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
{
    // Password, lockout, emails, etc.
    options.SignIn.RequireConfirmedAccount = false;
    options.Password.RequireDigit = false;
})
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

Scaffolding Identity

Scaffolding ASP.NET Core Identity

- Since **ASP.NET Core 2.2**, **Identity** is provided as a **Razor Class Library**
- The **scaffolder** can be configured to generate source code
  - If you need to modify the code and change the behavior
- Most of the necessary code is generated by the **scaffolder**

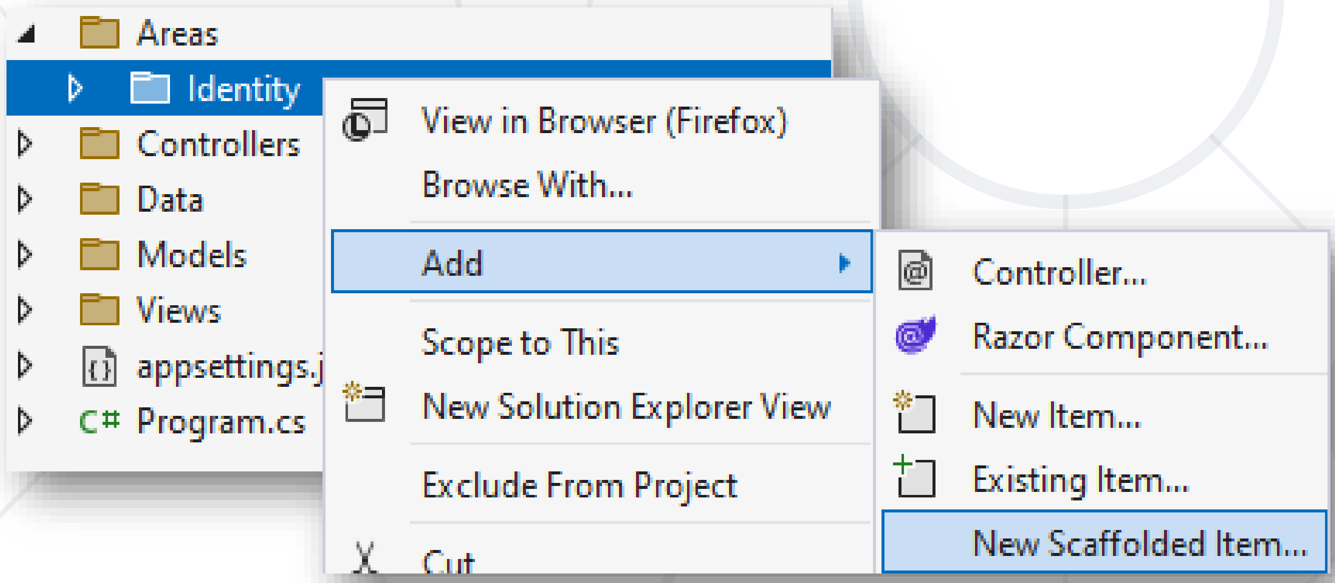


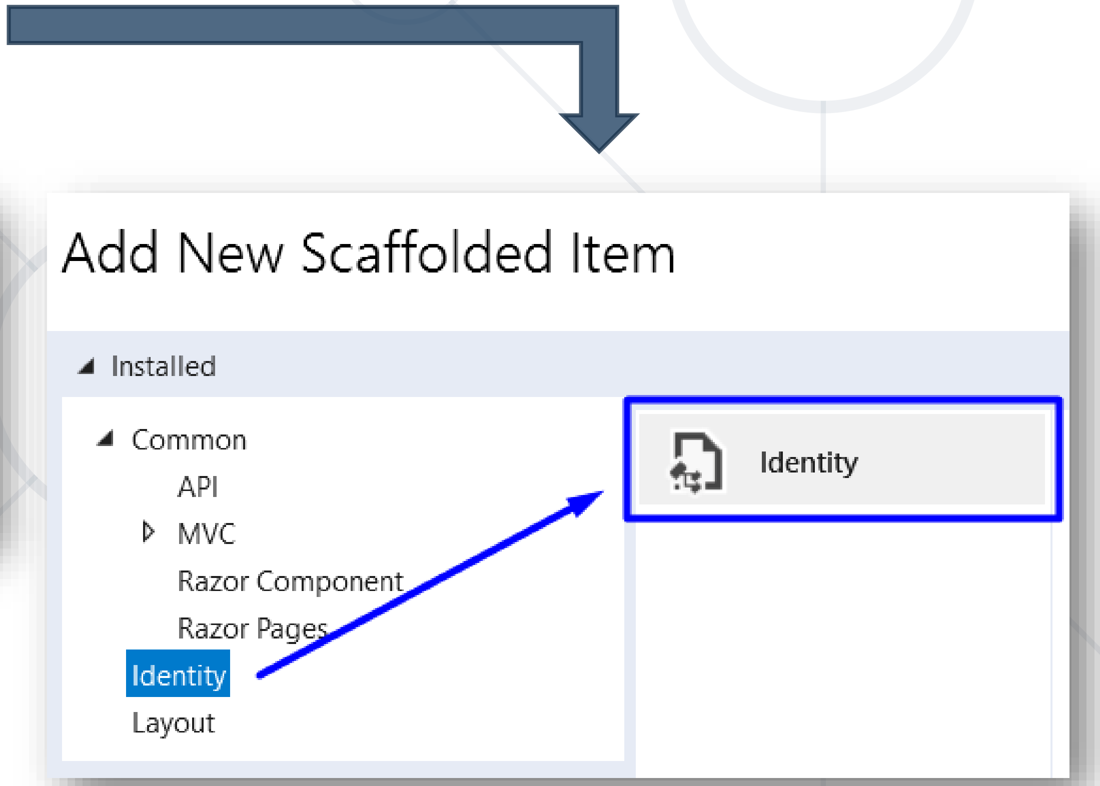
```
◢ Areas
  ◢ Identity
    ◢ Pages
      ◢ Account
          _ViewImports.cshtml
        ◢ Login.cshtml
          ▷ Login.cshtml.cs
        ◢ Register.cshtml
          ▷ Register.cshtml.cs
        _ValidationScriptsPartial.cshtml
        _ViewImports.cshtml
        _ViewStart.cshtml
```

Scaffolded `Account.Register` and `Account.Login`

Scaffolding ASP.NET Core Identity in VS

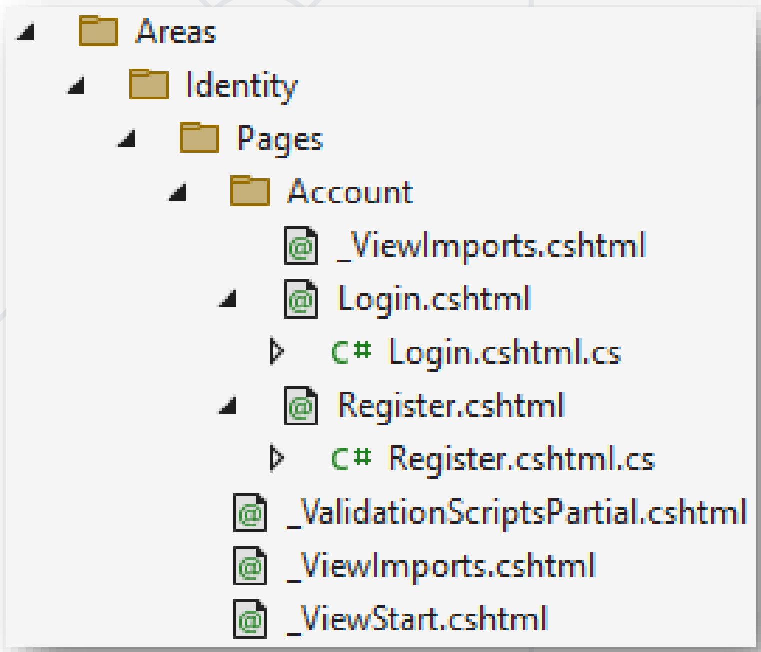
- Scaffold Identity pages by adding a **new scaffolded identity item**

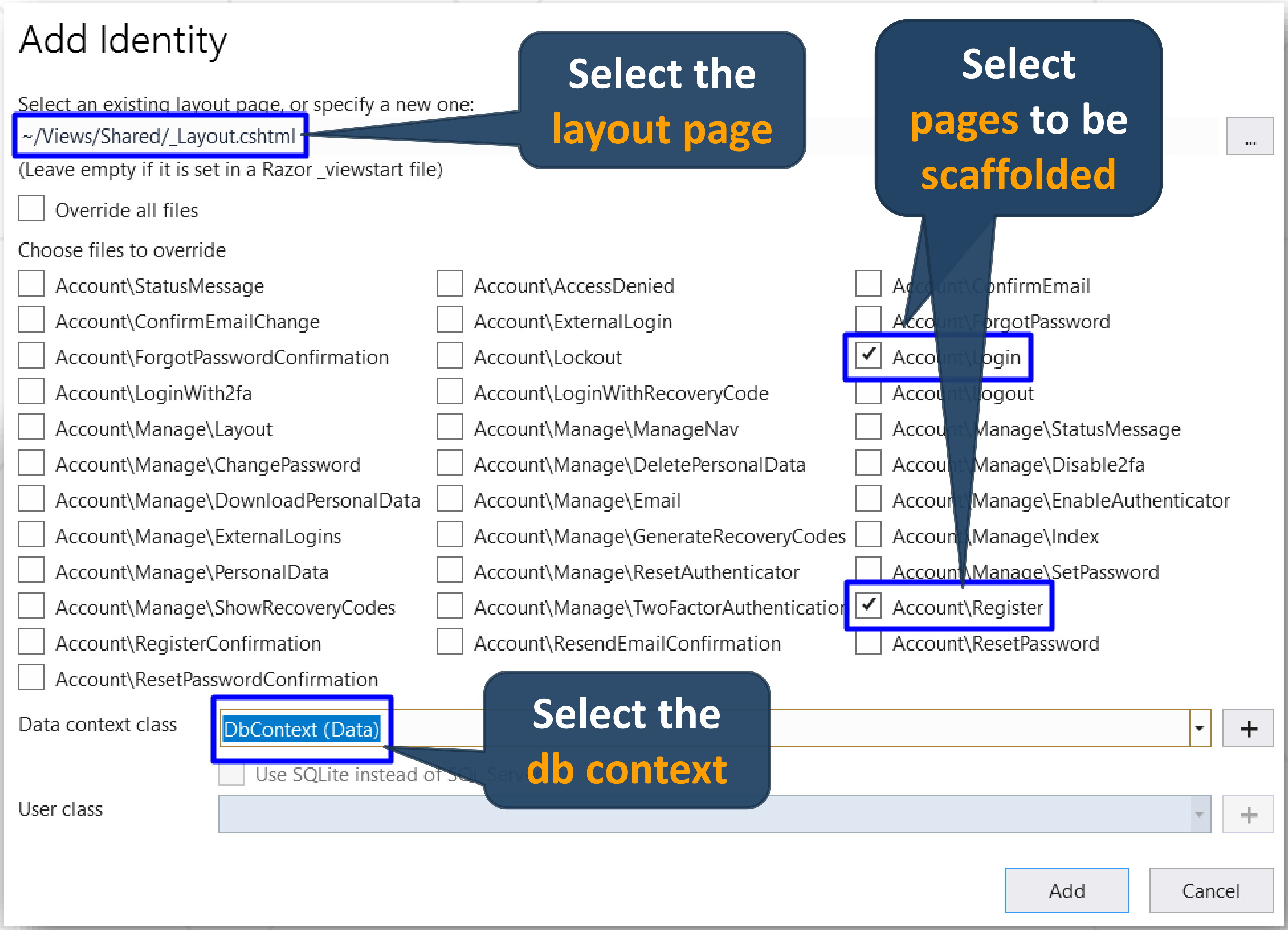



Add New Scaffolded Item

Scaffolding ASP.NET Core Identity in VS

- **Scaffolded pages** will be part of the "**/Areas/Identity**" folder





Add Identity

Select an existing layout page, or specify a new one:

~/Views/Shared/_Layout.cshtml

(Leave empty if it is set in a Razor _viewstart file)

[ ] Override all files

Choose files to override

[ ] Account\StatusMessage	[ ] Account\AccessDenied	[ ] Account\ConfirmEmail
[ ] Account\ConfirmEmailChange	[ ] Account\ExternalLogin	[ ] Account\ForgotPassword
[ ] Account\ForgotPasswordConfirmation	[ ] Account\Lockout	[✓] Account\Login
[ ] Account\LoginWith2fa	[ ] Account\LoginWithRecoveryCode	[ ] Account\Logout
[ ] Account\Manage\Layout	[ ] Account\Manage\ManageNav	[ ] Account\Manage\StatusMessage
[ ] Account\Manage\ChangePassword	[ ] Account\Manage\DeletePersonalData	[ ] Account\Manage\Disable2fa
[ ] Account\Manage\DownloadPersonalData	[ ] Account\Manage\Email	[ ] Account\Manage\EnableAuthenticator
[ ] Account\Manage\ExternalLogins	[ ] Account\Manage\GenerateRecoveryCodes	[ ] Account\Manage\Index
[ ] Account\Manage\PersonalData	[ ] Account\Manage\ResetAuthenticator	[ ] Account\Manage\SetPassword
[ ] Account\Manage\ShowRecoveryCodes	[ ] Account\Manage\TwoFactorAuthentication	[✓] Account\Register
[ ] Account\RegisterConfirmation	[ ] Account\ResendEmailConfirmation	[ ] Account\ResetPassword
[ ] Account\ResetPasswordConfirmation		

Data context class: DbContext (Data)

[ ] Use SQLite instead of SQL Server

User class:

Select the **layout page**

Select **pages** to be **scaffolded**

Select the **db context**

IdentityUser

ASP.NET Core User Manager

- `UserManager<TUser>` - APIs for managing users in a persistence store

`AddClaimAsync(…)`	`DeleteAsync(…)`
`AddPasswordAsync(…)`	`FindByIdAsync(…)`
`AddToRoleAsync(…)`	`FindByNameAsync(…)`
`AddToRolesAsync(…)`	`GetClaimsAsync(…)`
`ChangeEmailAsync(…)`	`GetRolesAsync(…)`
`ChangePasswordAsync(…)`	`GetUserAsync(…)`
`CheckPasswordAsync(…)`	`GetUserIdAsync(…)`
`ConfirmEmailAsync(…)`	`RemoveClaimAsync(…)`
`CreateAsync(…)`	`ValidateUserAsync(…)`

ASP.NET Core SignIn Manager

- **`SignInManager<TUser>`** – APIs for user sign in

`AddClaimsAsync(…)`	`FindByEmailAsync(…)`	`GenerateChangePhoneNumberTokenAsync(…)`
`AddToRoleAsync(…)`	`FindByIdAsync(…)`	`GenerateEmailConfirmationTokenAsync(…)`
`IsInRoleAsync(…)`	`FindByNameAsync(…)`	`GeneratePasswordResetTokenAsync(…)`
`GetUserId(…)`	`GetClaimsAsync(…)`	`GetAuthenticationTokenAsync(…)`
`ChangeEmailAsync(…)`	`GetEmailAsync(…)`	`IsEmailConfirmedAsync(…)`
`ConfirmEmailAsync(…)`	`GetRolesAsync(…)`	`CreateSecurityTokenAsync(…)`
`CreateAsync(…)`	`GetUserAsync(…)`	`ResetPasswordAsync(…)`
`DeleteAsync(…)`	`CheckPasswordAsync(…)`	`RemoveFromRoleAsync(…)`
`Dispose(…)`	`UpdatePassword(…)`	`RemoveClaimsAsync(…)`

Register

```
public async Task<IActionResult> Register()
{
   var user = CreateUser();
   var result = await _userManager.CreateAsync(user, "S0m3@Pa$$");

   if (result.Succeeded)
     // User registered successfuly
}
```

```
 private IdentityUser CreateUser()
 {
      try
      {
            return Activator.CreateInstance<IdentityUser>();
      }
      // catch exception if not successful
}
```

Login

```
public async Task<IActionResult> Login()
{
   bool rememberMe = true;
   bool shouldLockout = false;
   var signInStatus = await _signInManager.PasswordSignInAsync(
    "John", "S0m3@Pa$$", rememberMe, shouldLockout);

   if (signInStatus.Succeeded)
   {
       // Sucessful login
   }
   else
   {
      // Login failed
   }
}
```

Logout

```
public async Task<IActionResult Logout()
{
    await _signInManager.SignOutAsync();
}
```

Check the Currently Logged-In User

```
// GET: /Account/Roles (for logged-in users only)
[Authorize]
public ActionResult Roles()
{
    var currentUser = await userManager.GetUserAsync(this.User);
    var roles = await userManager.GetRolesAsync(currentUser);
    ...
}
```

```
// GET: /Account/Data (for logged-in users only)
[Authorize]
public ActionResult Data()
{
    var currentUser = await userManager.GetUserAsync(this.User);
    var currentUserUsername = await userManager.GetUserNameAsync(currentUser);
    var currentUserId = await userManager.GetUserIdAsync(currentUser);
    ...
}
```

Authorization

- Use the [**Authorize**] and [**AllowAnonymous**] attributes to configure **Authorized** / **Anonymous access** for **Controller** / **Action**

```
[Authorize]
public class AccountController : Controller
{
  // GET: /Account/Login (anonymous)
  [AllowAnonymous]
  public async Task<IActionResult> Login(string returnUrl) { … }

  // POST: /Account/LogOff (for logged-in users only)
  [HttpPost]
  public async Task<IActionResult> Logout() { … }
}
```

Summary

1. Authentication vs. Authorization
2. **ASP.NET Core Identity**
3. **Scaffolding** Identity
4. **IdentityUser**

Questions?

Software University

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SOFTWARE GROUP

DXC TECHNOLOGY

createX

Postbank

Решения за твоето утре

SUPER HOSTING .BG

DRAFT KINGS

VIVACOM

INDEAVR

Serving the high achievers

AMBITIONED

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License