ASP.NET Core Identity

Implementing login/logout, Scaffolding

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

Software University

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

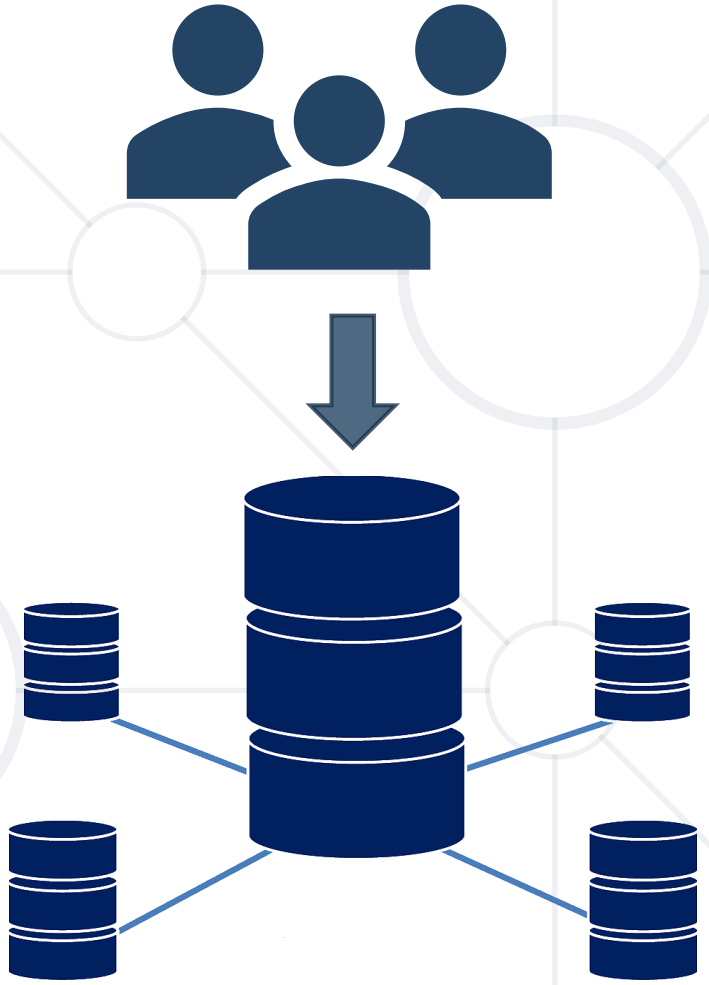
ASP.NET Core Identity

ASP.NET Core Identity

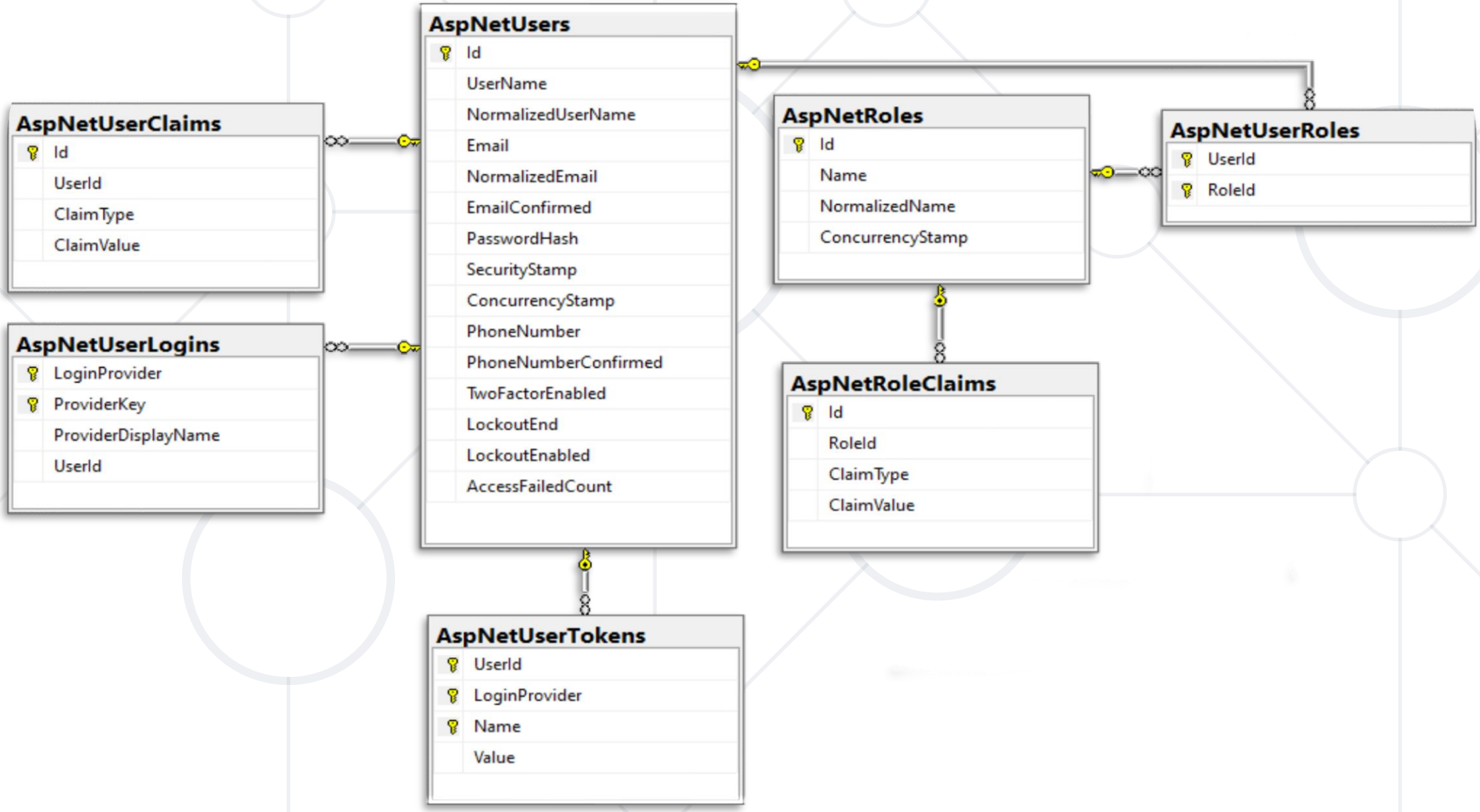
- The **ASP.NET Core Identity** system
  - Authentication and authorization system for ASP.NET Core
  - Supports ASP.NET Core MVC, Pages, Web API (JWT), SignalR
  - Handles **Users**, **User Profiles**, **Login** / **Logout**, **Roles**, etc.
  - Handles cookie consent and GDPR
  - Supports external 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

  - **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

Software University

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

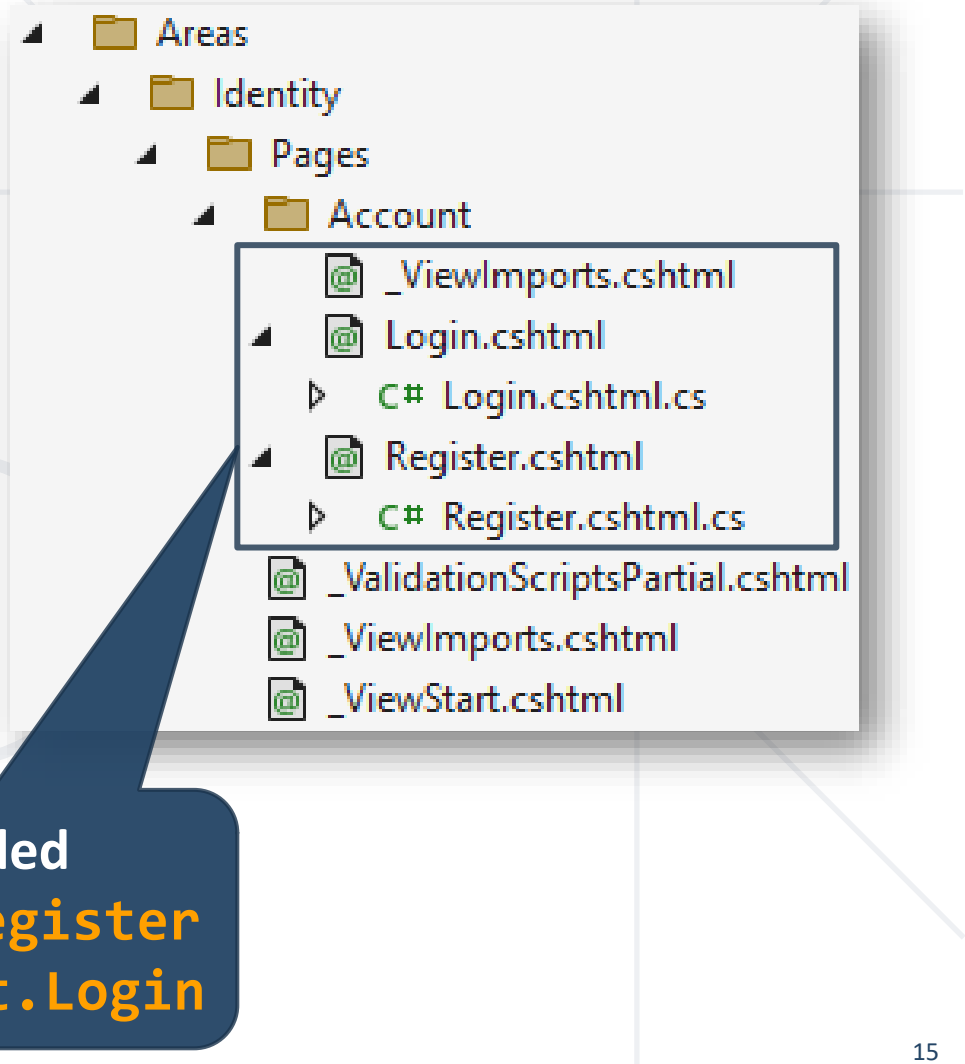
```
builder.Services.AddDefaultIdentity<IdentityUser>(options =>
{
    // Password, lockout, emails, etc.
    options.SignIn.RequireConfirmedAccount = false;
    options.Password.RequireDigit = false;
})
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

Scaffolding Identity

Scaffolding ASP.NET Core Identity

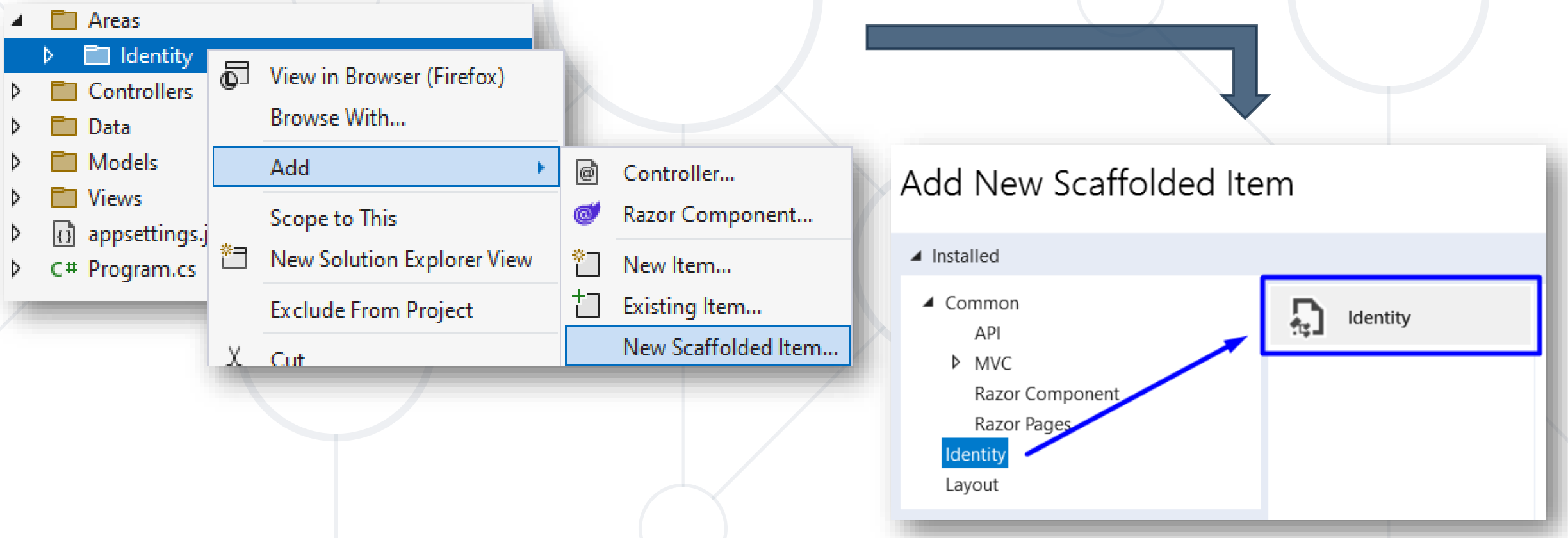
Software University

- 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**



Scaffolding ASP.NET Core Identity in VS

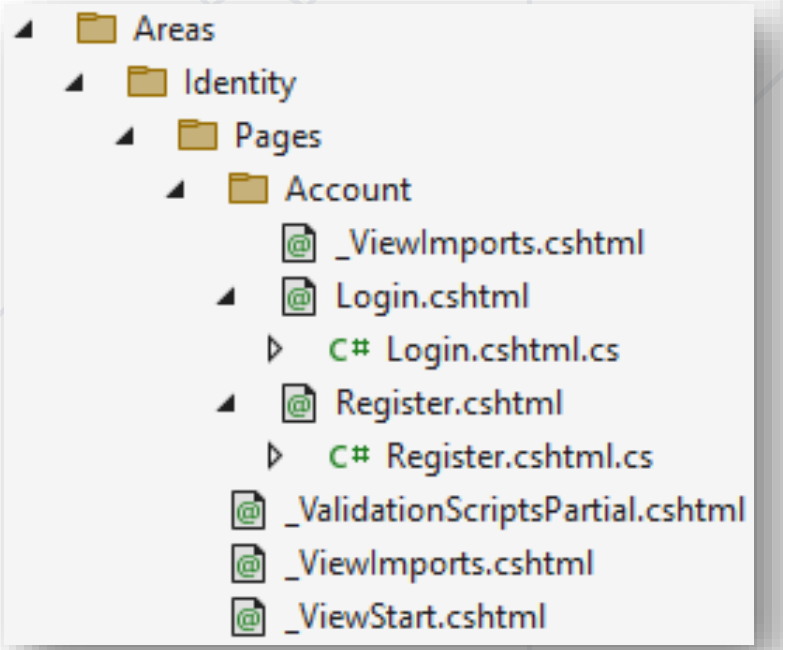
- Scaffold Identity pages by adding a **new scaffolded identity item**

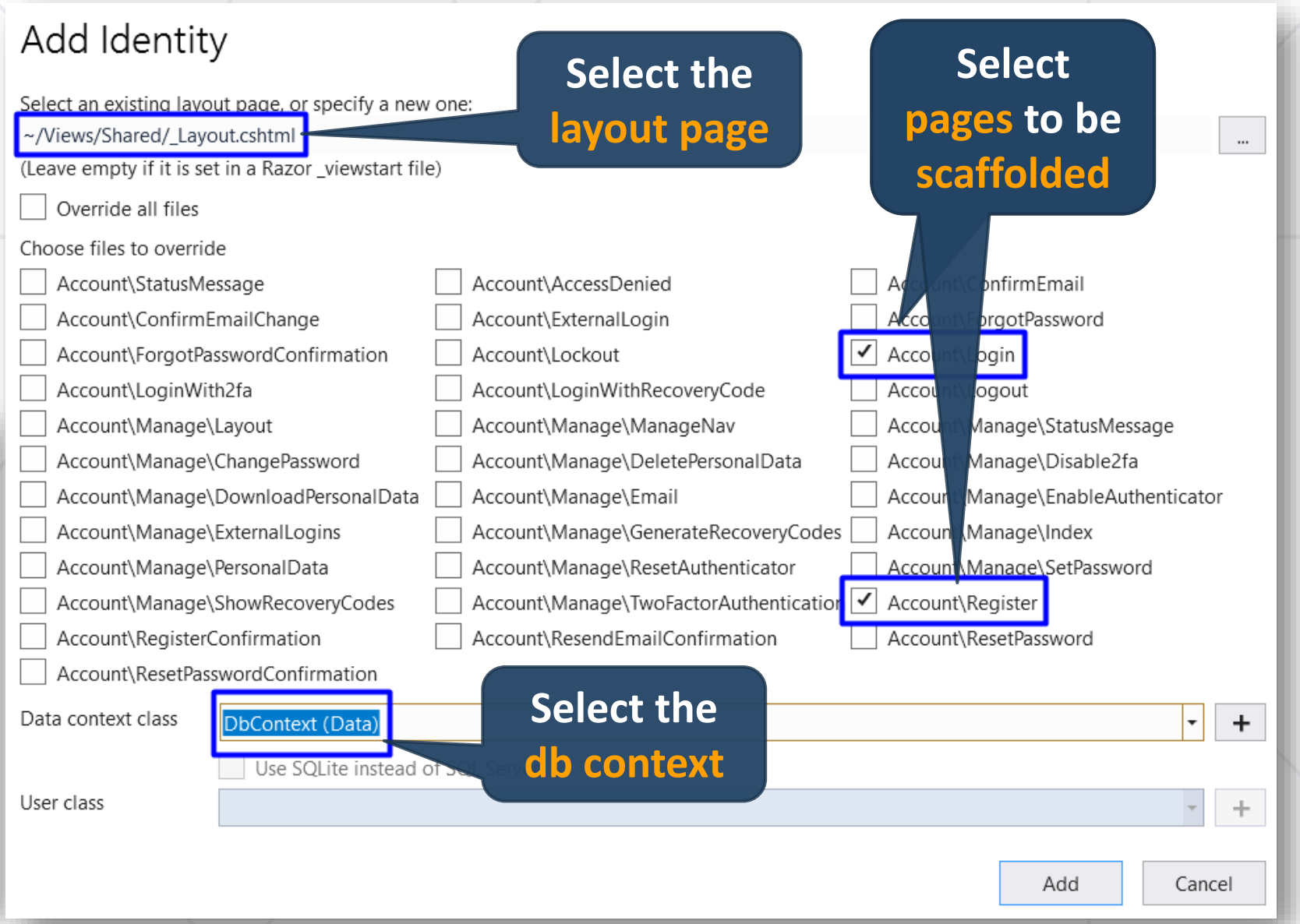


Scaffolding ASP.NET Core Identity in VS

Software University

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





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

Software University

```
// 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**

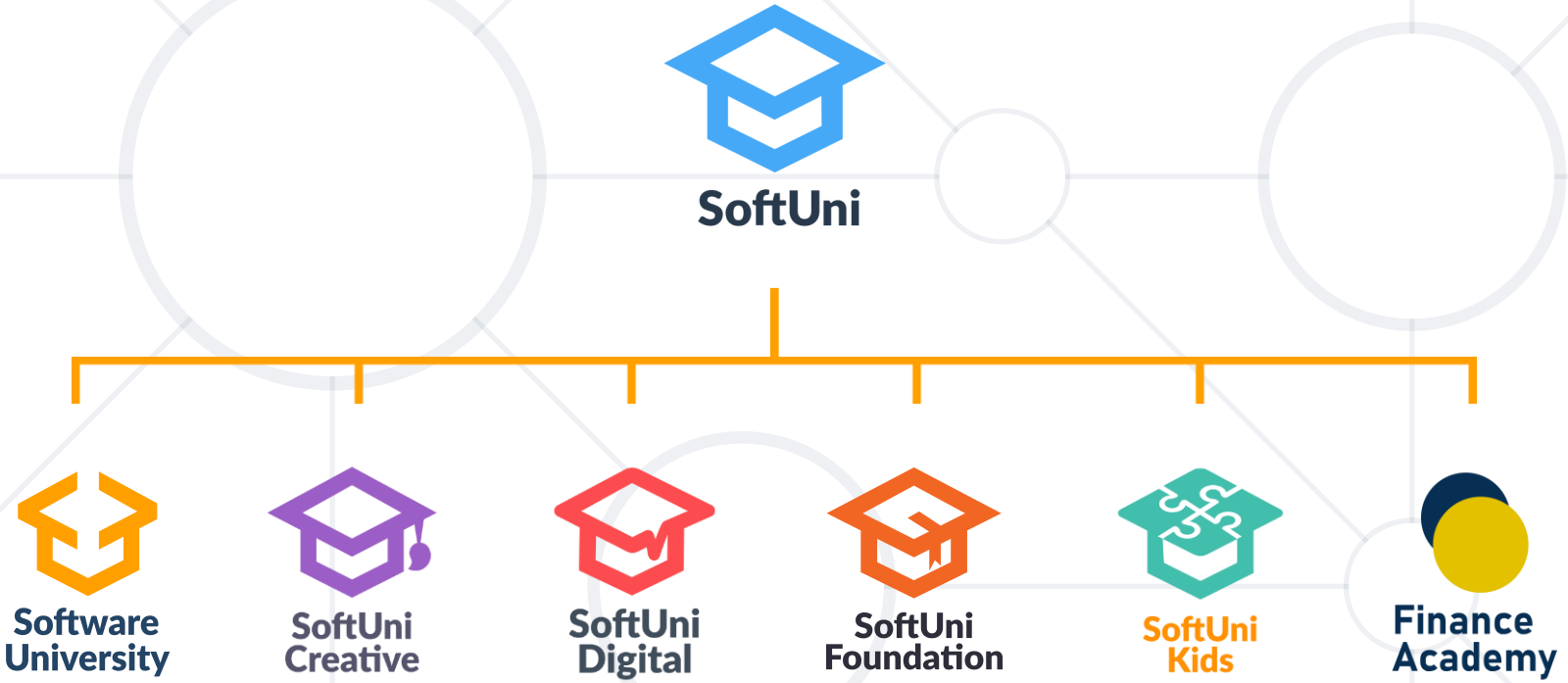
```
[Authorize]
public class AccountController : Controller
{
  // GET: /Account/Login (anonymous)
  [AllowAnonymous]
  public async Task<IActionResult> Login(string returnUrl) { … }

  // 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?



SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

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

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg
- © Software University – https://softuni.bg