==========

==========

DOCUMENT 9
USER REGISTRATION AND LOGIN SYSTEM
Step by step based

Goal
Build a full account system using Identity.

What you are building
Register page
Register POST to create user
Login page
Login POST to sign user in
Logout POST to sign user out

How to follow this document
You add one part. You run the project. You confirm it works.
Then you move to the next part.

STEP 1
Create the DTOs first

Why start with DTOs
DTOs define what data your forms accept.
They also drive validation using DataAnnotations.
If DTOs are wrong, ModelState fails and your forms never submit properly.

1A. RegisterDTO

RIGHT CLICK ON MODEL FOLDER AND THEN CLICK ON ADD AND THEN CLICK ON CLASS AND THEN NAME IT “RegisterDTO"

```
1. using System.ComponentModel.DataAnnotations;
2.
3. namespace RTWebApplication.DTOs
4. {
5.     public class RegisterDTO
6.     {
7.         [Required]
8.         [EmailAddress]
9.         public string Email { get; set; } = "";
10.
11.        [Required]
12.        [MinLength(8)]
13.        public string Password { get; set; } = "";
14.
15.        [Required]
16.        [Compare("Password")]
17.        public string ConfirmPassword { get; set; } = "";
18.
19.        [Required]
20.        [MaxLength(50)]
21.        public string FirstName { get; set; } = "";
22.
23.        [Required]
24.        [MaxLength(50)]
25.        public string LastName { get; set; } = "";
26.
27.        [Required]
28.        [Phone]
29.        public string PhoneNumber { get; set; } = "";
30.
31.        [Required]
32.        [MaxLength(200)]
33.        public string Address { get; set; } = "";
34.    }
35. }
37.
```

What this does

1. Required blocks empty submissions.
2. EmailAddress checks email format.

3. Compare forces ConfirmPassword to match Password.
4. Phone checks phone number format.

1B. LoginDTO

RIGHT CLICK ON MODEL FOLDER AND THEN CLICK ON ADD AND THEN CLICK ON CLASS AND THEN NAME IT “LoginDTO"

```
1. using System.ComponentModel.DataAnnotations;
2.
3. namespace RTWebApplication.DTOs
4. {
5.     public class LoginDTO
6.     {
7.         [Required]
8.         [EmailAddress]
9.         public string Email { get; set; } = "";
10.
11.         [Required]
12.         public string Password { get; set; } = "";
13.
14.         public bool RememberMe { get; set; }
15.     }
16. }
17.
```

What this does

LoginDTO matches the login form fields.
RememberMe sets a persistent cookie if true preventing the user from getting logged out to quickly from the site.

Checkpoint

Run the project. Fix any missing namespace errors before continuing.

STEP 2
Create the AccountController shell and define Identity services

Why this step exists

1. Controllers need access to Identity services.
2. You do not create users by hand. You use UserManager.
3. You do not sign in by hand. You use SignInManager.

Add AccountController.cs in Controllers.

RIGHT CLICK ON CONTROLLER FOLDER AND THEN CLICK ON ADD AND THEN CLICK ON CONTROLLER AND THEN NAME IT “AccountController" note controller names must end with “Controller”

Only add the top part first.

```
 1. using Microsoft.AspNetCore.Identity;
 2. using Microsoft.AspNetCore.Mvc;
 3. using RTWebApplication.Models;
 4.
 5. namespace RTWebApplication.Controllers
 6. {
 7.     public class AccountController : Controller
 8.     {
 9.         private readonly UserManager<ApplicationUser> userManager;
10.         private readonly SignInManager<ApplicationUser>
signInManager;
11.
12.         public AccountController(
13.             UserManager<ApplicationUser> userManager,
14.             SignInManager<ApplicationUser> signInManager)
15.         {
16.             this.userManager = userManager;
17.             this.signInManager = signInManager;
18.         }
19.     }
20. }
```

21.

What this does
1. UserManager talks to AspNetUsers table and handles passwords securely.
2. SignInManager handles sign in cookies and authentication state.
3. The constructor is dependency injection. ASP.NET Core gives you the services.

How you got to this part
1. You installed Identity packages.
2. You configured Identity in Program.cs.
3. Then you inject services into controllers via the constructor.

Checkpoint
Run the app. It should compile.
No actions yet, so you will not navigate to any account pages.

STEP 3
Add the Register GET action and create the Register view

Why this step exists
GET actions display pages.
You want a route that shows the registration form.

Add this inside AccountController.

It should replace the method with name Index()

```
1. [HttpGet]
2. public IActionResult Register()
3. {
4.     return View();
5. }
6.
```

What it does

1. When user visits /Account/Register, MVC runs this action.
2. It returns Views/Account/Register.cshtml.

How to create the view

1. Right click View()
2. Add View
3. Name it Register
4. Place it in Views/Account

Create Views/Account/Register.cshtml.

```
 1. @model RegisterDTO
 2.
 3. <h2>Register</h2>
 4.
 5. <form asp-action="Register" method="post">
 6.     <div>
 7.         <label>Email</label>
 8.         <input asp-for="Email" />
 9.         <span asp-validation-for="Email"></span>
10.     </div>
11.
12.     <div>
13.         <label>Password</label>
14.         <input asp-for="Password" type="password" />
15.         <span asp-validation-for="Password"></span>
16.     </div>
17.
18.     <div>
19.         <label>Confirm Password</label>
```

```
        <input asp-for="ConfirmPassword" type="password" />
        <span asp-validation-for="ConfirmPassword"></span>
    </div>

    <div>
        <label>First Name</label>
        <input asp-for="FirstName" />
        <span asp-validation-for="FirstName"></span>
    </div>

    <div>
        <label>Last Name</label>
        <input asp-for="LastName" />
        <span asp-validation-for="LastName"></span>
    </div>

    <div>
        <label>Phone Number</label>
        <input asp-for="PhoneNumber" />
        <span asp-validation-for="PhoneNumber"></span>
    </div>

    <div>
        <label>Address</label>
        <input asp-for="Address" />
        <span asp-validation-for="Address"></span>
    </div>

    <button type="submit">Register</button>
</form>

```

What this view does

1. asp-for connects the inputs to RegisterDTO properties.
2. asp-validation-for shows error messages from ModelState.

Checkpoint

Run the app and go to /Account/Register.
The page should load with form fields.

STEP 4
Add the Register POST action to create a user

Why this step exists
1. POST actions process form data.
2. You validate input.
3. Then you create the user in the database using UserManager.

add the POST. BELOW the Register GET method

```
 1. [HttpPost]
 2. public async Task<IActionResult> Register(RegisterDTO
registerDTO)
 3. {
 4.     if (!ModelState.IsValid)
 5.     {
 6.         return View(registerDTO);
 7.     }
 8.
 9.     var newUser = new ApplicationUser
10.     {
11.         UserName = registerDTO.Email,
12.         Email = registerDTO.Email,
13.         FirstName = registerDTO.FirstName,
14.         LastName = registerDTO.LastName,
15.         PhoneNumber = registerDTO.PhoneNumber,
16.         Address = registerDTO.Address,
17.         CreatedAt = DateTime.UtcNow
18.     };
19.
20.     var result = await userManager.CreateAsync(newUser,
registerDTO.Password);
21.
22.     if (result.Succeeded)
23.     {
24.         await userManager.AddToRoleAsync(newUser, "User");
25.         return RedirectToAction("Login", "Account");
26.     }
27.
28.     foreach (var error in result.Errors)
29.     {
```

```
30.             ModelState.AddModelError("", error.Description);
31.             return View(registerDTO);
32.         }
33.
34.         return View();
35.     }
36.
```

What it does, in order

1. ModelState.IsValid checks RegisterDTO attributes.
2. Creates a new ApplicationUser object.
3. CreateAsync saves the user and hashes password.
4. If successful, assigns User role.
5. Redirects to Login page.
6. If failed, it adds error messages to ModelState so the view shows them.

Important observation about the error loop

we return inside the foreach.
That means only the first error shows.

Better pattern so all errors show

```
1. foreach (var error in result.Errors)
2. {
3.     ModelState.AddModelError("", error.Description);
4. }
5. return View(registerDTO);
6.
```

Checkpoint

Run the app.
Register with a new email.
After submit, you should be redirected to Login.

Database check
AspNetUsers should contain the new user.
AspNetUserRoles should contain a row linking the user to User role.

STEP 5
Add Login GET action and create the Login view

Why this step exists
Users need a page to type credentials.

Add to AccountController.

```
1. [HttpGet]
2. public IActionResult Login()
3. {
4.     return View();
5. }
6.
```

Create Views/Account/Login.cshtml.

```
 1. @model LoginDTO
 2.
 3. <h2>Login</h2>
 4.
 5. @if (ViewBag.ErrorMessage != null)
 6. {
 7.     <p>@ViewBag.ErrorMessage</p>
 8. }
 9.
10. <form asp-action="Login" method="post">
11.     <div>
```

```
        <label>Email</label>
        <input asp-for="Email" />
        <span asp-validation-for="Email"></span>
    </div>

    <div>
        <label>Password</label>
        <input asp-for="Password" type="password" />
        <span asp-validation-for="Password"></span>
    </div>

    <div>
        <label>Remember Me</label>
        <input asp-for="RememberMe" type="checkbox"
/>
    </div>

    <button type="submit">Login</button>
</form>

```

Explanation

1. ViewBag.ErrorMessage displays a custom login failure message.
2. Model validation still displays field level errors.

Checkpoint

Go to /Account/Login and confirm the page loads.

STEP 6

Add Login POST action to sign the user in

Why this step exists

This creates the authentication cookie.

Add below Login GET.

```
 1. [HttpPost]
 2. public async Task<IActionResult> Login(LoginDTO
loginDTO)
 3. {
 4.     if (!ModelState.IsValid)
 5.     {
 6.         return View(loginDTO);
 7.     }
 8.
 9.     var signIn = await
signInManager.PasswordSignInAsync(
10.         loginDTO.Email,
11.         loginDTO.Password,
12.         loginDTO.RememberMe,
13.         false);
14.
15.     if (signIn.Succeeded)
16.     {
17.         return RedirectToAction("Index", "Home");
18.     }
19.     else
20.     {
21.         ViewBag.ErrorMessage = "Invalid Login
Attempt";
22.     }
23.
24.     return View(loginDTO);
25. }
26.
```

What it does

1. PasswordSignInAsync checks credentials.
2. If correct, it sets the login cookie.
3. If wrong, it shows an error message and returns the view.

Checkpoint
Login with the user you registered.
After login, you should be sent to Home/Index.

STEP 7
Add Logout POST action

Why this step exists
Logout clears the login cookie.

```
1. [HttpPost]
 2. public async Task<IActionResult> Logout(LoginDTO
loginDTO)
 3. {
 4.     if (signInManager.IsSignedIn(User))
 5.     {
 6.         await signInManager.SignOutAsync();
 7.     }
 8.
 9.     return RedirectToAction("Index", "Home");
10. }
11.
```

What it does
1. IsSignedIn(User) checks if the current user has an auth cookie.
2. SignOutAsync clears it.

Cleaner version

```
1. [HttpPost]
2. public async Task<IActionResult> Logout()
3. {
4.     await signInManager.SignOutAsync();
```

```
5.     return RedirectToAction("Index", "Home");
6. }
7.
```

Checkpoint
Login, then trigger logout from a button.
Confirm you are logged out.

STEP 8
Add Login, Register, Logout links to _Layout

Why this step exists
enables navigation without typing URLs.

Open Views/Shared/_Layout.cshtml and add Identity injections at the top.

```
1. @using Microsoft.AspNetCore.Identity
2. @using RTWebApplication.Models
3. @inject SignInManager<ApplicationUser> SignInManager
4. @inject UserManager<ApplicationUser> UserManager
5.
```

Then add links in nav.

```
 1. @if (SignInManager.IsSignedIn(User))
 2. {
 3.     <li class="nav-item">
 4.         <form asp-controller="Account" asp-
action="Logout" method="post">
 5.             <button type="submit" class="nav-link
btn btn-link text-dark">Logout</button>
 6.         </form>
 7.     </li>
 8. }
```

```
 9. else
10. {
11.     <li class="nav-item">
12.         <a class="nav-link text-dark" asp-
controller="Account" asp-action="Register">Register</a>
13.     </li>
14.     <li class="nav-item">
15.         <a class="nav-link text-dark" asp-
controller="Account" asp-action="Login">Login</a>
16.     </li>
17. }
18.
```

Explanation

1. You show Register and Login when logged out.
2. You show Logout when logged in.
3. Logout is a POST form, which is better practice.

==========

==========

COMMON ERRORS AND HOW TO FIX

Error 1

InvalidOperationException. Unable to resolve service for UserManager or SignInManager

Cause

Identity not registered in Program.cs

Fix

Ensure AddIdentity and AddEntityFrameworkStores exist.
Ensure app.UseAuthentication is present before UseAuthorization.

Error 2

Register POST runs but user not created

Cause
ModelState invalid
Fix
Check validation rules in RegisterDTO.
Ensure ConfirmPassword is included in form and matches Password.

Error 3
CreateAsync fails with password errors
Cause
Password does not meet policy
Fix
Either change password rules in Program.cs, or use a stronger password.

Error 4
Login always fails even with correct credentials
Causes
UserName does not match what you log in with
Authentication middleware missing
Fix
Set UserName = Email at registration, which you already do.
Add app.UseAuthentication in Program.cs.

Error 5
Logout does nothing
Cause
Logout link is GET not POST, or the form is not inside a form tag
Fix
Use a POST form to call Logout action.

Error 6
Roles not assigned on register
Cause
User role does not exist yet
Fix
Run your RolesAndSeedAdminUser seeding at startup first.

Error 7
Only one Identity error shows during registration
Cause
Return inside foreach loop
Fix
Move the return outside the loop so all errors are added.