100 XP

Exercise - Customize Identity

15 minutes

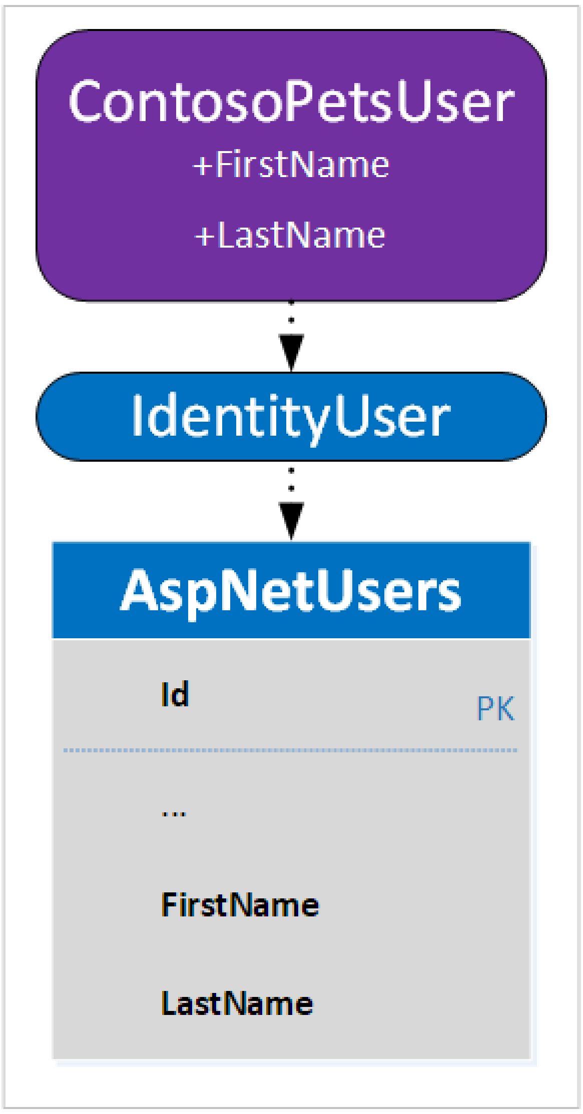
Sandbox activated! Time remaining: **2 min**

You have used 4 of 10 sandboxes for today. More sandboxes will be available tomorrow.

Choose the ASP.NET Core Identity data store

PostgreSQL [✓ SQL Server]

By default, Identity represents a user with an `IdentityUser` class. One way to extend the data being captured at registration time is to create a class deriving from `IdentityUser`. In this unit, a derived class named `ContosoPetsUser` is created. `ContosoPetsUser` will contain properties to store the user's first and last name.



UI changes are also required to collect the additional user profile information. The following steps explain the process of collecting a first and last name for the registered user.

Customize the user account data

1. Add the user registration files to be modified to the project:

.NET Core CLI

```
dotnet aspnet-codegenerator identity \
    --dbContext ContosoPetsAuth \
    --files
"Account.Manage.EnableAuthenticator;Account.Manage.Index;Account.Register" \
    --userClass ContosoPetsUser \
    --force
```

In the preceding command:

- The `--dbContext` option provides the tool with knowledge of the existing `DbContext`-derived class named `ContosoPetsAuth`.

- The `--files` option specifies a semicolon-delimited list of unique files to be added to the *Identity* area.
- The `--userClass` option results in the creation of an `IdentityUser`-derived class named `ContosoPetsUser`.
- The `--force` option causes existing files in the *Identity* area to be overwritten.

Tip

Run the following command from the project root to view valid values for the `--files` option:

.NET Core CLI

Copy

```
dotnet aspnet-codegenerator identity --listFiles
```

The following files are added to the *Areas/Identity* directory:

- **Data/**
  - *ContosoPetsUser.cs*
- **Pages/**
  - *_ViewImports.cshtml*
  - **Account/**
    - *_ViewImports.cshtml*
    - *Register.cshtml*
    - *Register.cshtml.cs*
    - **Manage/**
      - *_ManageNav.cshtml*
      - *_ViewImports.cshtml*
      - *EnableAuthenticator.cshtml*
      - *EnableAuthenticator.cshtml.cs*
      - *Index.cshtml*
      - *Index.cshtml.cs*
      - *ManageNavPages.cs*

Additionally, the *Data/ContosoPetsAuth.cs* file, which existed before running the preceding command, was overwritten because the `--force` option was used. The `ContosoPetsAuth` class declaration now references the newly created user type of `ContosoPetsUser`:

C#

Copy

```
public class ContosoPetsAuth : IdentityDbContext<ContosoPetsUser>
```

The *EnableAuthenticator* Razor page was scaffolded, though it won't be modified until later in the module.

2. In the `Configure` method of *Areas/Identity/IdentityHostingStartup.cs*, the call to `AddDefaultIdentity` needs to be made aware of the new Identity user type. Incorporate the following highlighted change, and save the file.

```C#
services.AddDefaultIdentity<ContosoPetsUser>()
    .AddDefaultUI()
    .AddEntityFrameworkStores<ContosoPetsAuth>();
```

3. Update *Pages/Shared/_LoginPartial.cshtml* to incorporate the following highlighted changes. Save your changes.

```CSHTML
@using Microsoft.AspNetCore.Identity
@using ContosoPets.Ui.Areas.Identity.Data
@inject SignInManager<ContosoPetsUser> SignInManager
@inject UserManager<ContosoPetsUser> UserManager

<ul class="navbar-nav">
```

The preceding changes update the user type passed to both `SignInManager<T>` and `UserManager<T>` in the `@inject` directives. Instead of the default `IdentityUser` type, `ContosoPetsUser` user is now referenced. The `@using` directive was added to resolve the `ContosoPetsUser` references.

Pages/Shared/_LoginPartial.cshtml is physically located outside of the *Identity* area. Consequently, the file wasn't updated automatically by the scaffold tool. The appropriate changes had be made manually.

Tip

As an alternative to manually editing the *_LoginPartial.cshtml* file, it can be deleted prior to running the scaffold tool. The *_LoginPartial.cshtml* file will be recreated with references to the new `ContosoPetsUser` class.

4. Update *Areas/Identity/Data/ContosoPetsUser.cs* to support storage and retrieval of the additional user profile data. Make the following changes:

a. Add the `FirstName` and `LastName` properties:

C# Copy

```csharp
public class ContosoPetsUser : IdentityUser
{
    [Required]
    [MaxLength(100)]
    public string FirstName { get; set; }

    [Required]
    [MaxLength(100)]
    public string LastName { get; set; }
}
```

The properties in the preceding snippet represent additional columns to be created in the underlying `AspNetUsers` table. Both properties are required and are therefore annotated with the `[Required]` attribute. The `[Required]` attribute also results in a non-null constraint in the underlying database table column. Additionally, the `[MaxLength]` attribute indicates that a maximum length of 100 characters is allowed. The underlying table column's data type is defined accordingly.

b. Add the following `using` statement to the top of the file. Save your changes.

C# Copy

```csharp
using System.ComponentModel.DataAnnotations;
```

The preceding code resolves the data annotation attributes applied to the `FirstName` and `LastName` properties.

Update the database

1. Create and apply an EF Core migration to update the underlying data store:

.NET Core CLI Copy

```
dotnet ef migrations add UpdateUser && \
    dotnet ef database update
```

The `UpdateUser` EF Core migration applied a DDL change script to the `AspNetUsers` table's schema. Specifically, `FirstName` and `LastName` columns were added, as seen in the following migration output excerpt:

Console Copy

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (37ms) [Parameters=[], CommandType='Text',
CommandTimeout='30']
    ALTER TABLE [AspNetUsers] ADD [FirstName] nvarchar(100) NOT NULL DEFAULT
N'';
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (36ms) [Parameters=[], CommandType='Text',
CommandTimeout='30']
    ALTER TABLE [AspNetUsers] ADD [LastName] nvarchar(100) NOT NULL DEFAULT
N'';
```

Complete the following steps to analyze the impact of the `UpdateUser` EF Core migration on the `AspNetUsers` table's schema. You'll gain an understanding of the impact extending the Identity data model has on the underlying data store.

2. Run the following command to view the table schema:

Bash Copy

```bash
db -Q "SELECT COLUMN_NAME, IS_NULLABLE, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH
AS MAX_LENGTH FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME='AspNetUsers'"
-Y 20
```

The following output displays:

Console Copy

```
COLUMN_NAME          IS_NULLABLE DATA_TYPE            MAX_LENGTH
-------------------- ----------- -------------------- -----------
Id                   NO          nvarchar                     450
UserName             YES         nvarchar                     256
NormalizedUserName   YES         nvarchar                     256
Email                YES         nvarchar                     256
NormalizedEmail      YES         nvarchar                     256
EmailConfirmed       NO          bit                         NULL
PasswordHash         YES         nvarchar                      -1
SecurityStamp        YES         nvarchar                      -1
ConcurrencyStamp     YES         nvarchar                      -1
PhoneNumber          YES         nvarchar                      -1
PhoneNumberConfirmed NO          bit                         NULL
TwoFactorEnabled     NO          bit                         NULL
LockoutEnd           YES         datetimeoffset              NULL
LockoutEnabled       NO          bit                         NULL
AccessFailedCount    NO          int                         NULL
FirstName            NO          nvarchar                     100
LastName             NO          nvarchar                     100
```

The `FirstName` and `LastName` properties in the `ContosoPetsUser` class correspond to the `FirstName` and `LastName` columns in the preceding output. A data type of `nvarchar(100)` was assigned to each of the two columns because of the `[MaxLength(100)]` attributes. The non-null constraint was added because of the `[Required]` attributes. Existing rows show empty strings in the new columns.

3. Run the following command to view the primary key for the table:

Bash Copy

```bash
db -i $setupWorkingDirectory/list-aspnetusers-pk.sql -Y 15
```

The following output shows that the `Id` column is the unique identifier for a user account:

Console Copy

```console
Table           Column          Primary key
--------------- --------------- ---------------
AspNetUsers     Id              PK_AspNetUsers
```

Customize the user registration form

1. In *Areas/Identity/Pages/Account/Register.cshtml*, add the following highlighted markup:

CSHTML Copy

```cshtml
<form asp-route-returnUrl="@Model.ReturnUrl" method="post">
    <h4>Create a new account.</h4>
    <hr />
    <div asp-validation-summary="All" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Input.FirstName"></label>
        <input asp-for="Input.FirstName" class="form-control" />
        <span asp-validation-for="Input.FirstName" class="text-danger">
</span>
    </div>
    <div class="form-group">
        <label asp-for="Input.LastName"></label>
        <input asp-for="Input.LastName" class="form-control" />
        <span asp-validation-for="Input.LastName" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Input.Email"></label>
        <input asp-for="Input.Email" class="form-control" />
        <span asp-validation-for="Input.Email" class="text-danger"></span>
    </div>
```

With the preceding markup, **First name** and **Last name** text boxes are added to the user registration form.

2. In *Areas/Identity/Pages/Account/Register.cshtml.cs*, add support for the name text boxes.

   a. Add the `FirstName` and `LastName` properties to the `InputModel` nested class:

C# Copy

```csharp
public class InputModel
{
    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at
max {1} characters long.", MinimumLength = 1)]
    [Display(Name = "First name")]
    public string FirstName { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at
max {1} characters long.", MinimumLength = 1)]
    [Display(Name = "Last name")]
    public string LastName { get; set; }

    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    public string Email { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at
max {1} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation
password do not match.")]
    public string ConfirmPassword { get; set; }
}
```

   The `[Display]` attributes define the label text to be associated with the text boxes.

   b. Modify the `OnPostAsync` method to set the `FirstName` and `LastName` properties on the `ContosoPetsUser` object. Make the following highlighted changes:

C# Copy

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (ModelState.IsValid)
    {
        var user = new ContosoPetsUser
        {
            FirstName = Input.FirstName,
            LastName = Input.LastName,
            UserName = Input.Email,
            Email = Input.Email,
        };
        var result = await _userManager.CreateAsync(user, Input.Password);
        if (result.Succeeded)
        {
```

The preceding change sets the `FirstName` and `LastName` properties to the user input from the registration form.

Customize the site header

Update *Pages/Shared/_LoginPartial.cshtml* to display the first and last name collected during user registration. The highlighted lines in the following snippet are needed:

CSHTML Copy

```
@using Microsoft.AspNetCore.Identity
@using ContosoPets.Ui.Areas.Identity.Data
@inject SignInManager<ContosoPetsUser> SignInManager
@inject UserManager<ContosoPetsUser> UserManager

<ul class="navbar-nav">
@if (SignInManager.IsSignedIn(User))
{
    ContosoPetsUser user = await UserManager.GetUserAsync(User);
    var fullName = $"{user.FirstName} {user.LastName}";

    <li class="nav-item">
        <a id="manage" class="nav-link text-dark" asp-area="Identity" asp-
page="/Account/Manage/Index" title="Manage">Hello, @fullName!</a>
    </li>
    <li class="nav-item">
        <form id="logoutForm" class="form-inline" asp-area="Identity" asp-
page="/Account/Logout" asp-route-returnUrl="@Url.Page("/Index", new { area = ""
})">
            <button id="logout" type="submit" class="nav-link btn btn-link text-
dark">Logout</button>
        </form>
    </li>
}
```

```
else
{
    <li class="nav-item">
        <a class="nav-link text-dark" id="register" asp-area="Identity" asp-
page="/Account/Register">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link text-dark" id="login" asp-area="Identity" asp-
page="/Account/Login">Login</a>
    </li>
}
</ul>
```

Customize the profile management form

1. In *Areas/Identity/Pages/Account/Manage/Index.cshtml*, add the following highlighted markup. Save your changes.

   CSHTML Copy

   ```
   <form id="profile-form" method="post">
       <div asp-validation-summary="All" class="text-danger"></div>
       <div class="form-group">
           <label asp-for="Input.FirstName"></label>
           <input asp-for="Input.FirstName" class="form-control" />
           <span asp-validation-for="Input.FirstName" class="text-danger">
   </span>
       </div>
       <div class="form-group">
           <label asp-for="Input.LastName"></label>
           <input asp-for="Input.LastName" class="form-control" />
           <span asp-validation-for="Input.LastName" class="text-danger"></span>
       </div>
       <div class="form-group">
           <label asp-for="Username"></label>
           <input asp-for="Username" class="form-control" disabled />
       </div>
   ```

2. In *Areas/Identity/Pages/Account/Manage/Index.cshtml.cs*, make the following changes to support the name text boxes.

   a. Add the `FirstName` and `LastName` properties to the `InputModel` nested class:

   C# Copy

   ```
   public class InputModel
   {
   ```

```
    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at
max {1} characters long.", MinimumLength = 1)]
    [Display(Name = "First name")]
    public string FirstName { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at
max {1} characters long.", MinimumLength = 1)]
    [Display(Name = "Last name")]
    public string LastName { get; set; }

    [Phone]
    [Display(Name = "Phone number")]
    public string PhoneNumber { get; set; }
}
```

b. Incorporate the highlighted changes in the `LoadAsync` method:

C# Copy

```
private async Task LoadAsync(ContosoPetsUser user)
{
    var userName = await _userManager.GetUserNameAsync(user);
    var phoneNumber = await _userManager.GetPhoneNumberAsync(user);

    Username = userName;

    Input = new InputModel
    {
        PhoneNumber = phoneNumber,
        FirstName = user.FirstName,
        LastName = user.LastName,
    };
}
```

The preceding code supports retrieving the first and last names for display in the corresponding text boxes of the profile management form.

c. Incorporate the highlighted changes in the `OnPostAsync` method. Save your changes.

C# Copy

```
public async Task<IActionResult> OnPostAsync()
{
    var user = await _userManager.GetUserAsync(User);
    if (user == null)
    {
        return NotFound($"Unable to load user with ID
'{_userManager.GetUserId(User)}'.");
    }
    if (!ModelState.IsValid)
    {
        await LoadAsync(user);
        return Page();
    }

    user.FirstName = Input.FirstName;
    user.LastName = Input.LastName;
    await _userManager.UpdateAsync(user);

    var phoneNumber = await _userManager.GetPhoneNumberAsync(user);
    if (Input.PhoneNumber != phoneNumber)
    {
        var setPhoneResult = await _userManager.SetPhoneNumberAsync(user,
Input.PhoneNumber);
        if (!setPhoneResult.Succeeded)
        {
            var userId = await _userManager.GetUserIdAsync(user);
            throw new InvalidOperationException($"Unexpected error
occurred setting phone number for user with ID '{userId}'.");
        }
    }

    await _signInManager.RefreshSignInAsync(user);
    StatusMessage = "Your profile has been updated";
    return RedirectToPage();
}
```

The preceding code supports updating the first and last names in the database's `AspNetUsers` table.

Build, deploy, and test

1. Run the following command to build the app:

   .NET Core CLI   Copy

   ```
   dotnet build --no-restore
   ```

   The `--no-restore` option is included because no NuGet packages were added since the last build. The build process bypasses restoration of NuGet packages and succeeds with no warnings. If the build fails, check the output for troubleshooting information.

2. Deploy the app to Azure App Service by running the following command:

   Azure CLI   Copy

   ```
   az webapp up
   ```

3. In your browser, navigate to the app. Select **Logout** if you're still logged in.

   Tip

   If you need the URL to your app, display it with the following command:

   Bash Copy

   ```bash
   echo $webAppUrl
   ```

4. Select **Register** and use the updated form to register a new user.

   Note

   The validation constraints on the **First name** and **Last name** fields reflect the data annotations on the `FirstName` and `LastName` properties of `InputModel`.

   After registering, you're redirected to the homepage. The app's header now contains **Hello, [First name] [Last name]!**.

5. Run the following command to confirm that the first and last names are stored in the database:

   Bash Copy

   ```bash
   db -Q "SELECT UserName, Email, FirstName, LastName FROM dbo.AspNetUsers" -Y
   25
   ```

   A variation of the following output displays:

   Console Copy

   ```console
   UserName                  Email                     FirstName
   LastName
   ------------------------- ------------------------- -------------------------
   -------------------------
   kai.klein@contoso.com     kai.klein@contoso.com
   jana.heinrich@contoso.com jana.heinrich@contoso.com Jana
   Heinrich
   ```

   The first user registered prior to adding `FirstName` and `LastName` to the schema. Consequently, the associated `AspNetUsers` table record doesn't have data in those columns.

Test the changes to the profile management form

1. In the web app, log in with the first user you created.
2. Click the **Hello, !** link to navigate to the profile management form.

   > ⓘ **Note**
   >
   > The link doesn't display correctly because the `AspNetUsers` table's row for this user doesn't contain values for `FirstName` and `LastName`.

3. Enter valid values for **First name** and **Last name**. Select **Save**.

   The app's header updates to **Hello, [First name] [Last name]!**.

Next unit: Exercise - Configure multi-factor authentication

Continue >