100 XP

Exercise - Configure Identity support

10 minutes

Sandbox activated! Time remaining: **2 hr 30 min**

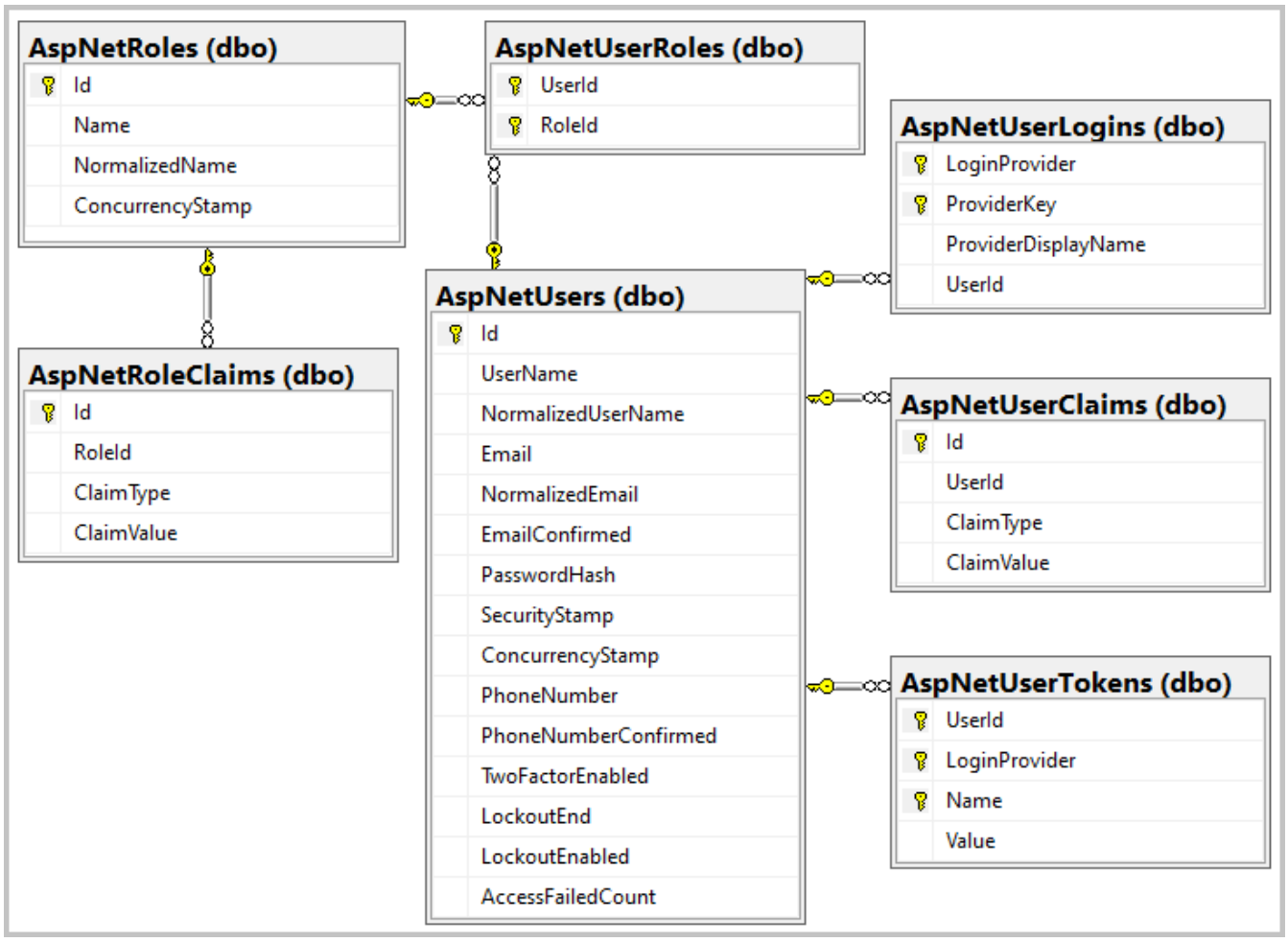
You have used 4 of 10 sandboxes for today. More sandboxes will be available tomorrow.

Choose the ASP.NET Core Identity data store

PostgreSQL | SQL Server

Identity works out-of-the-box without any customization. The standard Identity UI components are packaged in a .NET Standard Razor Class Library (RCL). Because an RCL is used, few files are added to the project.

After applying the initial EF Core migration, the supporting database tables are created. The following diagram depicts the schemas of the supporting tables:



In this unit, Identity will be added to the existing ASP.NET Core Razor Pages project.

Add Identity to the project

1. Install the ASP.NET Core code scaffolder:

.NET Core CLI Copy

```
dotnet tool install dotnet-aspnet-codegenerator --version 3.1.2
```

The following output appears:

Console Copy

```
You can invoke the tool from this directory using the following commands:
'dotnet tool run dotnet-aspnet-codegenerator' or 'dotnet dotnet-aspnet-
codegenerator'.
Tool 'dotnet-aspnet-codegenerator' (version '3.1.2') was successfully
installed. Entry is added to the manifest file /home/<USER>/aspnet-
learn/src/ContosoPets.Ui/.config/dotnet-tools.json.
```

The scaffolder is a .NET Core tool that will:

- Be used to add the default Identity components to the project.
- Enable customization of Identity UI components in the next unit.
- Be invoked via `dotnet aspnet-codegenerator` in this module.

2. Add the following NuGet packages to the project:

.NET Core CLI Copy

```
dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design --version
3.1.2 && \
    dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore --
version 3.1.3 && \
    dotnet add package Microsoft.AspNetCore.Identity.UI --version 3.1.3 && \
    dotnet add package Microsoft.EntityFrameworkCore.Design --version 3.1.3
&& \
    dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version
3.1.3
```

These packages install code generation templates and dependencies that are used by the scaffolder.

Tip

To view the available generators:

- In the command shell, run `dotnet aspnet-codegenerator -h`.
- When in Visual Studio, right-click the project in **Solution Explorer** and select **Add** > **New Scaffolded Item**.

3. Use the scaffolder to add the default Identity components to the project. Run the following command from the project root:

.NET Core CLI Copy

```
dotnet aspnet-codegenerator identity \
    --useDefaultUI \
    --dbContext ContosoPetsAuth
```

In the preceding command:

- The generator identified as `identity` is used to add Identity framework to the project.
- The `--useDefaultUI` option indicates that an RCL containing the default UI elements will be used. Bootstrap will be used to style the components.
- The `--dbContext` option to indicate the name of an EF Core database context class to generate.

4. Update file explorer by clicking the editor's refresh icon.

An *Areas* directory structure appears in the project root:

- *Areas*
  - *Identity*
    - *Data*
      - *ContosoPetsAuth.cs*
    - *Pages*
      - *_ValidationScriptsPartial.cshtml*
      - *_ViewStart.cshtml*
    - *IdentityHostingStartup.cs*

Areas provide a way to partition an ASP.NET Core web app into smaller functional groups.

Configure the database connection

1. Replace the `Configure` method of *Areas/Identity/IdentityHostingStartup.cs* with the following code:

C# Copy

```
public void Configure(IWebHostBuilder builder)
{
    builder.ConfigureServices((context, services) => {
        var connBuilder = new SqlConnectionStringBuilder(

context.Configuration.GetConnectionString("ContosoPetsAuthConnection"))
        {
            UserID = context.Configuration["DbUsername"],
            Password = context.Configuration["DbPassword"]
        };

        services.AddDbContext<ContosoPetsAuth>(options =>
            options.UseSqlServer(connBuilder.ConnectionString));

        services.AddDefaultIdentity<IdentityUser>()
            .AddDefaultUI()
            .AddEntityFrameworkStores<ContosoPetsAuth>();
    });
}
```

In the preceding code:

- The Azure Key Vault configuration provider is implicitly used to retrieve the database username and password:

  C# Copy

  ```
  UserID = context.Configuration["DbUsername"],
  Password = context.Configuration["DbPassword"]
  ```

- The database username and password are injected into the connection string stored in *appsettings.json*.
- The EF Core database context class, named `ContosoPetsAuth`, is configured with the appropriate connection string.
- The Identity services are registered, including the default UI, token providers, and cookie-based authentication.

2. Also in *IdentityHostingStartup.cs*, add the following code to the block of `using` statements at the top. Save your changes.

C# Copy

```
using Microsoft.Data.SqlClient;
```

The preceding code resolves the reference to the `SqlConnectionStringBuilder` class in the `Configure` method.

3. In the `Configure` method of *Startup.cs*, replace the `// Add the app.UseAuthentication code` comment with the following code. Save your changes.

C# Copy

```
app.UseAuthentication();
```

The preceding code enables authentication capabilities. More specifically, an instance of the ASP.NET Core authentication middleware is added to the app's HTTP request-handling pipeline.

4. Run the following command to print the database connection string to the console. Copy the connection string to your clipboard.

Bash Copy

```
echo $dbConnectionString
```

5. In *appsettings.json*, replace the connection string for `ContosoPetsAuthConnection` with the connection string from the previous step. Save your changes.

The `ConnectionStrings` section should look similar to the following JSON:

JSON Copy

```
"ConnectionStrings": {
    "ContosoPetsAuthConnection": "Data Source={HOST
NAME}.database.windows.net;Initial Catalog=ContosoPets;Connect
Timeout=30;Encrypt=True;TrustServerCertificate=False;ApplicationIntent=ReadWr
ite;MultiSubnetFailover=False"
}
```

6. Run the following command to build the app:

.NET Core CLI Copy

```
dotnet build
```

The build succeeds with no warnings. If the build fails, check the output for troubleshooting information.

Update the database

1. Install the Entity Framework Core migration tool:

.NET Core CLI Copy

```
dotnet tool install dotnet-ef --version 3.1.3
```

The following output appears:

Console Copy

```
You can invoke the tool from this directory using the following commands:
'dotnet tool run dotnet-ef' or 'dotnet dotnet-ef'.
Tool 'dotnet-ef' (version '3.1.3') was successfully installed. Entry is added
to the manifest file /home/<USER>/aspnet-
learn/src/ContosoPets.Ui/.config/dotnet-tools.json.
```

The migration tool is a .NET Core tool that will:

- Generate code called a migration to create and update the database that supports the Identity entity model.
- Execute migrations against an existing database.
- Be invoked via `dotnet ef` in this module.

2. Create and run an EF Core migration to update the database:

.NET Core CLI Copy

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

The `CreateIdentitySchema` EF Core migration applied a Data Definition Language (DDL) change script to create the tables supporting Identity. For example, the following excerpt depicts a `CREATE TABLE` statement generated by the migration:

Console Copy

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (98ms) [Parameters=[], CommandType='Text',
CommandTimeout='30']
      CREATE TABLE [AspNetUsers] (
```

```
            [Id] nvarchar(450) NOT NULL,
            [UserName] nvarchar(256) NULL,
            [NormalizedUserName] nvarchar(256) NULL,
            [Email] nvarchar(256) NULL,
            [NormalizedEmail] nvarchar(256) NULL,
            [EmailConfirmed] bit NOT NULL,
            [PasswordHash] nvarchar(max) NULL,
            [SecurityStamp] nvarchar(max) NULL,
            [ConcurrencyStamp] nvarchar(max) NULL,
            [PhoneNumber] nvarchar(max) NULL,
            [PhoneNumberConfirmed] bit NOT NULL,
            [TwoFactorEnabled] bit NOT NULL,
            [LockoutEnd] datetimeoffset NULL,
            [LockoutEnabled] bit NOT NULL,
            [AccessFailedCount] int NOT NULL,
            CONSTRAINT [PK_AspNetUsers] PRIMARY KEY ([Id])
        );
```

3. Run the following command to list the tables in the database:

Bash Copy

```bash
db -Q "SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE
TABLE_SCHEMA='dbo' ORDER BY TABLE_NAME" -Y 25
```

The following output appears, which confirms the creation of the tables.

Console Copy

```console
TABLE_NAME
-------------------------
__EFMigrationsHistory
AspNetRoleClaims
AspNetRoles
AspNetUserClaims
AspNetUserLogins
AspNetUserRoles
AspNetUsers
AspNetUserTokens

(8 rows affected)
```

Add the login and registration links

1. In *Pages/Shared/_Layout.cshtml*, replace the `@* Add the _LoginPartial partial view *@` comment with the following. Save your changes.

CSHTML Copy

```
<partial name="_LoginPartial" />
```

The preceding markup renders the `_LoginPartial` partial view within the header of any page that uses the default layout. `_LoginPartial` was added by the Identity scaffold. This partial view presents the user with **Login** and **Register** links if the user isn't signed in.

2. 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.

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

Azure CLI Copy

```
az webapp up
```

> **Note**
>
> The *.azure/config* file in the project root contains the configuration values used by `az webapp up`.

4. Run the following command to view the app's URL. Navigate to that URL in your browser.

Bash Copy

```
echo $webAppUrl
```

5. Click the **Register** link in the app's header. Complete the form to create a new account.

   After successful registration:

   - You're redirected to the homepage.
   - The app's header displays **Hello [Email address]!** and a **Logout** link.
   - A cookie named *.AspNetCore.Identity.Application* is created. Identity preserves user sessions with cookie-based authentication.

6. Click the **Logout** link in the app's header.

   After successfully logging out, the *.AspNetCore.Identity.Application* cookie is deleted to terminate the user session.

Next unit: Exercise - Customize Identity

Continue >

Azure Cloud Shell

```
Requesting a Cloud Shell.Succeeded.
Connecting terminal...

The following variables are used in this mod
webAppUrl: https://webapp138379996.azurewebs
dbConnectionString: Data Source=azsql1383799
g=ContosoPetsAuth;Connect Timeout=30;Encrypt
licationIntent=ReadWrite;MultiSubnetFailover
```