100 XP

Exercise - Configure multi-factor authentication

10 minutes

Sandbox activated! Time remaining: 1 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, the ASP.NET Core project templates using Identity include multi-factor authentication support for TOTP authenticator apps. The Razor Pages template's **Configure authenticator app** form provides a 32-character registration key to seed the token value. In this unit, you'll customize the **Configure authenticator app** form to provide a QR code containing the registration key. The QR code will serve as an alternative authentication mechanism to the 32-character key.

Multiple strategies exist for generating the QR code. An example includes using a third-party JavaScript library. In this unit, however, a third-party NuGet package is used to generate the QR code with C#. The resulting QR code image is injected into an HTML placeholder element as a base-64 encoded string.

Examine the QR code generation infrastructure

To avoid distracting from the focus on Identity, the boilerplate to support QR code generation has been provided. The supporting changes include:

- The NuGet package, `QRCoder`, has already been installed in the project.
- All interactions with the `QRCoder` library are abstracted away in the *Services/QRCodeService.cs* file. The `QRCodeService` class:
  - Uses constructor injection to gain access to an instance of the library's `QRCodeGenerator` class.
  - Exposes the `GetQRCodeAsBase64` method to return the base-64 encoded string. The QR code dimensions are determined by the integer value passed to `GetGraphic`. In this case, the generated QR code will be composed of blocks sized four pixels squared.

- `QRCodeService` is registered as a singleton service in the IoC container within *Startup.cs*.

Customize multi-factor authentication

1. Open *Areas/Identity/Pages/Account/Manage/EnableAuthenticator.cshtml.cs* and make the following changes and save:

   a. Add the following property to the `EnableAuthenticatorModel` class to store the QR code's base-64 string representation:

   C# Copy

   ```
   public class EnableAuthenticatorModel : PageModel
   {
       private readonly UserManager<ContosoPetsUser> _userManager;
       private readonly ILogger<EnableAuthenticatorModel> _logger;
       private readonly UrlEncoder _urlEncoder;

       public string QrCodeAsBase64 { get; set; }
   ```

   b. Incorporate the highlighted changes in the `OnGetAsync` page handler:

   C# Copy

   ```
   public async Task<IActionResult> OnGetAsync([FromServices] QRCodeService
   qrCodeService)
   {
       var user = await _userManager.GetUserAsync(User);
       if (user == null)
       {
           return NotFound($"Unable to load user with ID
   '{_userManager.GetUserId(User)}'.");
       }

       await LoadSharedKeyAndQrCodeUriAsync(user);
       QrCodeAsBase64 = qrCodeService.GetQRCodeAsBase64(AuthenticatorUri);

       return Page();
   }
   ```

   In the preceding page handler, parameter injection provides a reference to the `QRCodeService` singleton service. `QRCodeService` is responsible for interactions with a third-party library that generates QR codes.

   c. Add the following `using` statement to the top of the file to resolve the reference to `QRCodeService`. Save your changes.

C# Copy

```csharp
using ContosoPets.Ui.Services;
```

2. In *Areas/Identity/Pages/Account/Manage/EnableAuthenticator.cshtml*, make the following highlighted changes and save:

CSHTML Copy

```cshtml
<li>
    <p>Scan the QR Code or enter this key <kbd>@Model.SharedKey</kbd> into
your two factor authenticator app. Spaces and casing do not matter.</p>
    <div class="alert alert-info">To enable QR code generation please read
our <a href="https://go.microsoft.com/fwlink/?
Linkid=852423">documentation</a>.</div>
    <div id="qrCode">
        <img alt="embedded QR code"
src="data:image/png;base64,@Model.QrCodeAsBase64" />
    </div>
    <div id="qrCodeData" data-url="@Html.Raw(@Model.AuthenticatorUri)"></div>
</li>
```

The preceding markup embeds the base-64 encoded image in the page.

Test multi-factor authentication

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. Navigate to the site and log in with either registered user (if not already logged in). Select **Hello, [First name] [Last name]!** link to navigate to the profile management page, and then select **Two-factor authentication**.

Notice the presence of the following message on the page:

Console Copy

```
Privacy and cookie policy have not been accepted.
You must accept the policy before you can enable two factor authentication.
```

4. Click the **Accept** link in the privacy and cookie use policy banner to accept the policy. Refresh the page.

   A cookie named *.AspNet.Consent* is created to mark acceptance of the policy. The cookie expires one year from the acceptance date.

5. Select the **Add authenticator app** button.

6. Follow the on-screen instructions to register and verify your authenticator app for this user.

   Using Microsoft Authenticator on Android as an example, follow these steps to add the account to the app:
   a. Open the Microsoft Authenticator app.
   b. Select the kebab menu (vertical ellipsis) in the upper right.
   c. Select **Add account**.
   d. Select **Other account (Google, Facebook, etc.)**.
   e. Scan the QR code as indicated.
   f. Select **Finish** to verify the 32-character key.

7. Enter the verification code provided by your TOTP app in the **Verification Code** text box.

8. Select **Verify**.

   Upon successful verification, the page displays a **Your authenticator app has been verified** banner and some recovery codes.

9. Run the following command to see the effect on the `AspNetUsers` table's `TwoFactorEnabled` column:

   Bash Copy

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

   For the logged in user, the output shows that the `TwoFactorEnabled` column is equal to `1`. Because multi-factor authentication hasn't been enabled for the other registered user, the record's column value is `0`.

10. Select **Logout**, and then log in again with the same user.

11. Enter the verification code from the TOTP authenticator app in the **Authenticator code** text box. Select the **Log in** button.

12. Select **Hello, [First name] [Last name]!**. Then, select the **Two-factor authentication** tab.

    Because Microsoft Authenticator has been set up, the following buttons appear:

    - **Disable 2FA**
    - **Reset recovery codes**
    - **Set up authenticator app**
    - **Reset authenticator app**

Next unit: Exercise - Use claims with policy-based authorization

Continue >

Azure Cloud Shell

```
rajani_net@Azure:~/aspnet-learn/src/ContosoP
```