

Customising ASP.Net Identity in Blazor server side

By Joshua Holden

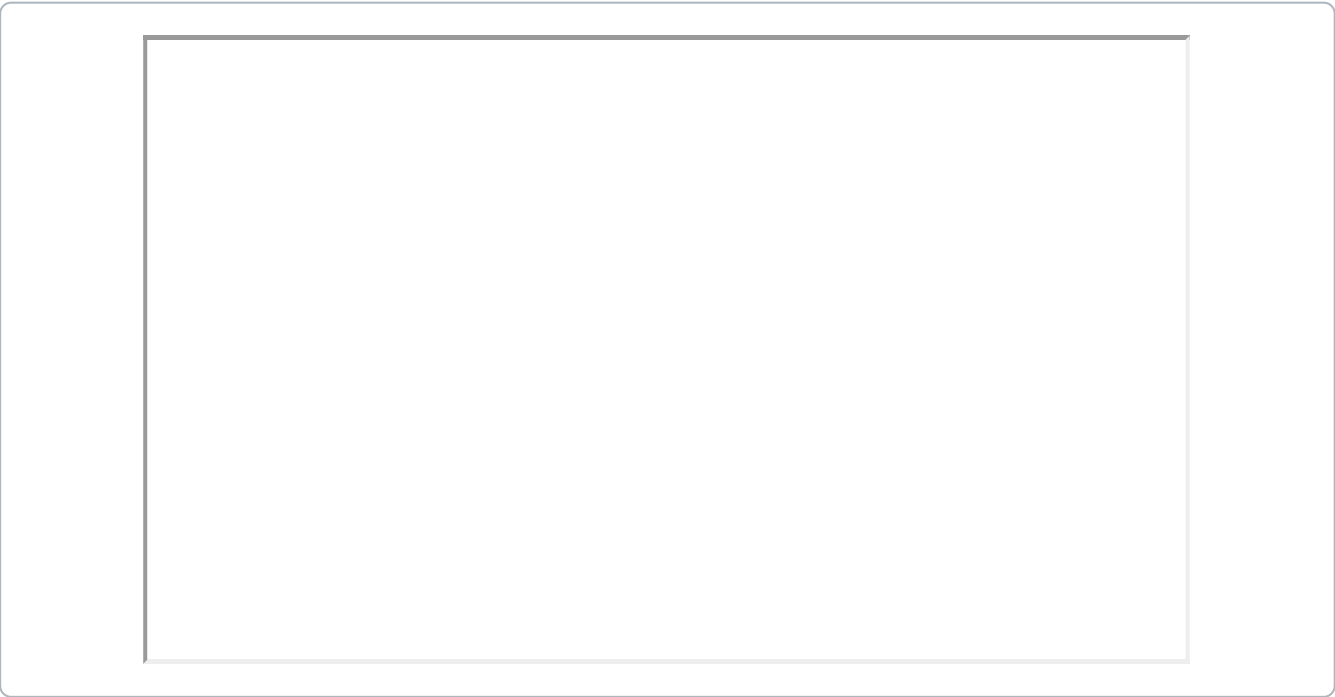
Search blogs

When you create a standard server side Blazor app using the default template with individual user accounts enabled you get a lot out of the box such as user registration, email confirmation, edit profile and role based security.

Blazor uses the core ASP.Net authentication using entity framework to generate the AspNetUsers and related authorisation tables either by as default in a SQL compact database, or by changing the connection string, a different type of SQL database, in this example I will be using on-prem (local) SQL server.

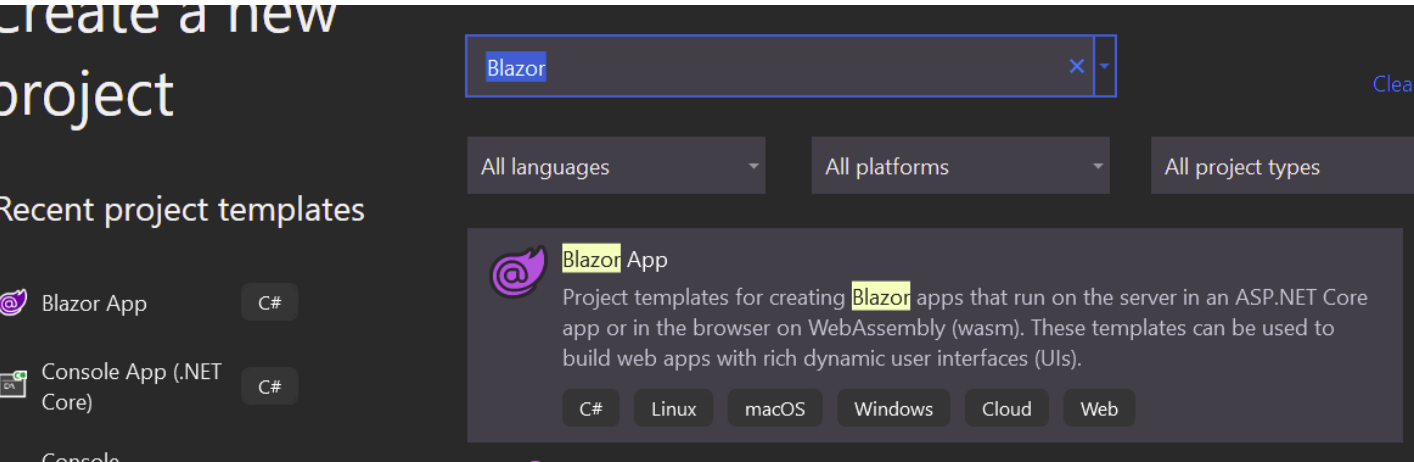
If you are already familiar with setting up individual user accounts with asp.net identity, doing migrations and changing connection strings but just want to know about how to add new fields to the identity user, you may want to skip past the basic functionality section.

The content discussed on this post can also be found in the below video if you prefer to watch, if not just keep scrolling to the text content below:

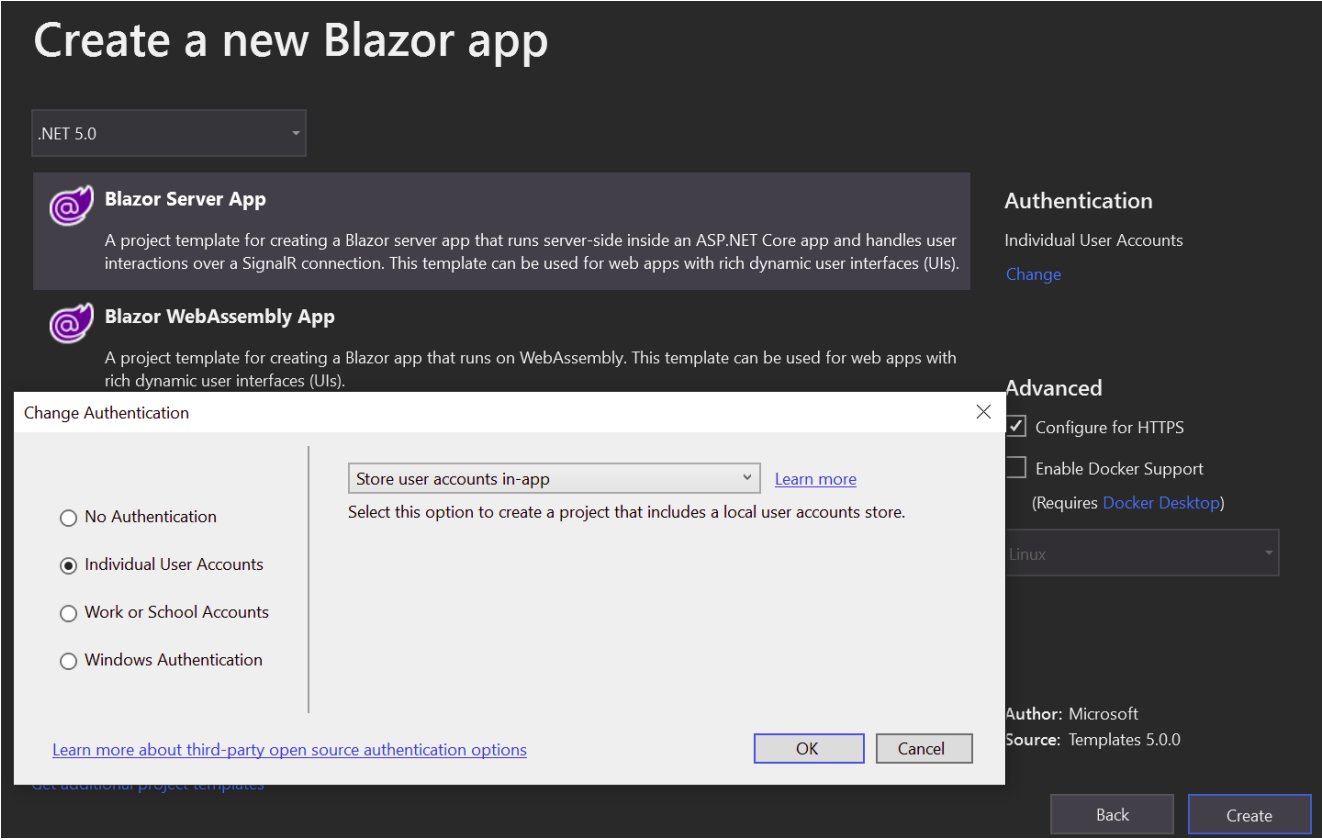


Basic functionality.

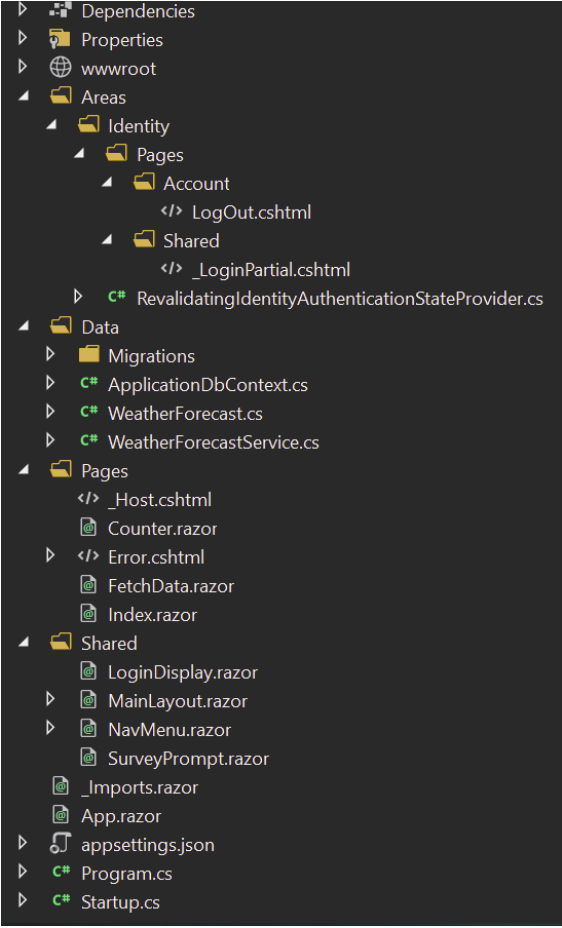
We are going to start by firing up Visual Studio and choosing Blazor app as a new project:



Click next and give your Blazor web application a name, then click next again, choose Blazor server side, click change against authentication and make sure you choose "individual user accounts"



Finally, click create, now we can examine the solution.



Looking at the above structure of the default application there doesn't seem to be much going on, no ApplicationUser class or registration pages, etc, this is because Blazor at runtime injects the pages and classes required by ASP.net identity in and if you want to alter them you will have to manually create the pages to override them (or you can add them individually as scaffolded items from the menu).

To create the required tables, first of all, we will need to update the default connection string to point at a database and table, if you are opting to use the SQLCompact database then feel free to skip the following part and jump to the part of the adding migration below.

First of all, fire up SSMS and create a new empty database and add a SQL user account to connect to it And although not necessarily secure, for ease map the account to the database you have created as DBO, in a real environment it's probably better to grant only the permissions required to each table for the user, once this has been done and a new database has been created and a SQL account created with permissions flick back into visual studio and open up the appsettings.json file, we need to replace the existing default connection string to point at our new database.

The existing connection string will be pointing to a localdb, and will look something like this

```
"ConnectionStrings": {  
  "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-  
AuthorisationDemo-BC1AB14F-C507-4594-B1B0-  
250B0A2E609B;Trusted_Connection=True;MultipleActiveResultSets=true"  
}
```

replace this with your SQL connection string mine looks like this:

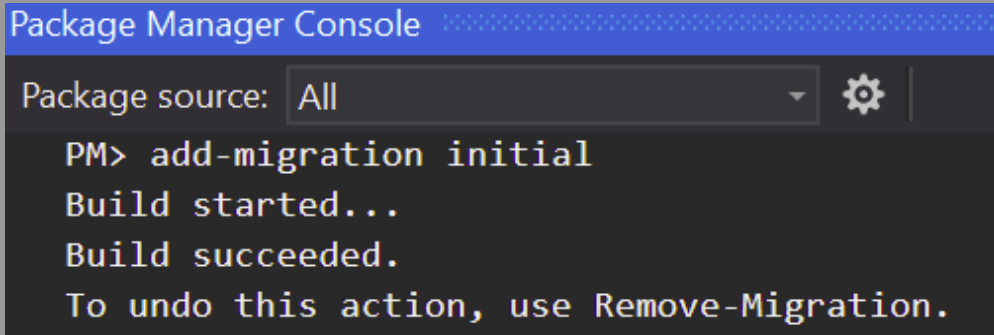
```
Server=localhost;Database=IdentityDemo;User  
Id=IdentityDemoSqlUser;Password=Password123;
```

The final config should now look like this:

```
"ConnectionStrings": {  
  "DefaultConnection": "Server=localhost;Database=IdentityDemo;User  
Id=IdentityDemoSqlUser;Password=Password123;"  
}
```

Now the application is configured to point at our new database we can run the initial migration and update the database to build the required authorisation tables.

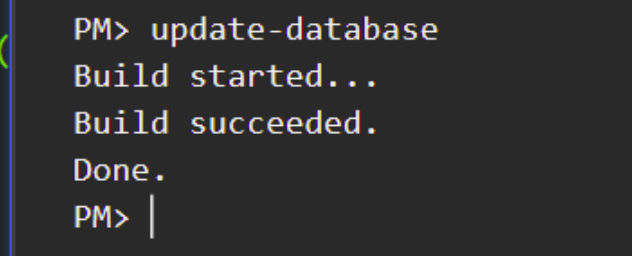
In Visual Studio click on the tools menu item in the top menu bar, then "Nuget Package Manager", and finally "Package Manager Console", once this is done the package manager console should open at the bottom of the screen, in the console type: "add-migration initial" end press enter:



```
Package Manager Console  
Package source: All  
PM> add-migration initial  
Build started...  
Build succeeded.  
To undo this action, use Remove-Migration.
```

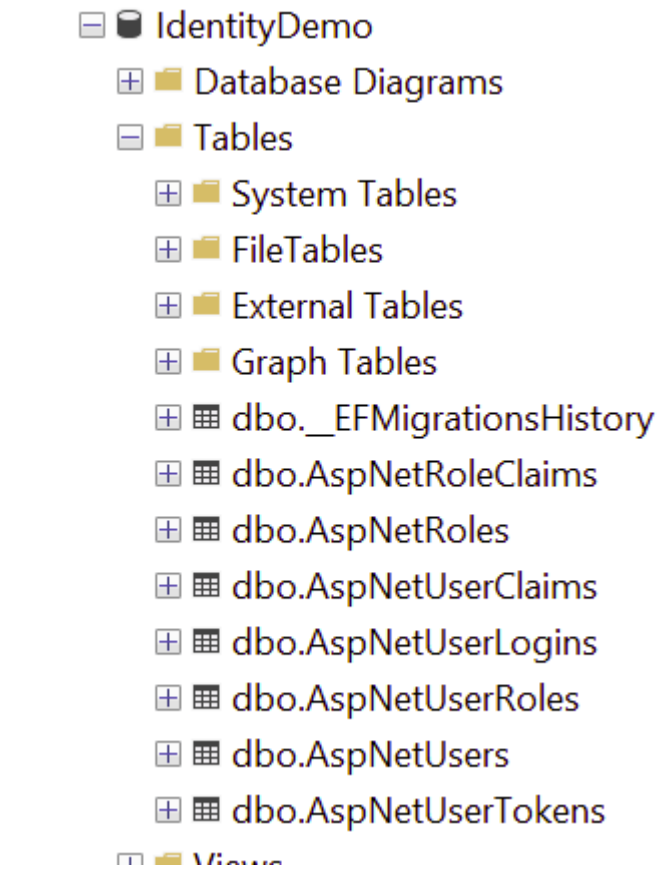
If everything is successful, the NPM (Nuget package manager) console should reflect the above screenshot, adding a migration builds all of the scripts required to update your database and add (or remove if reverting to a previous migration) the tables/objects that have been altered since the last migration.

Now, we need to issue one final command: "Update-Database":



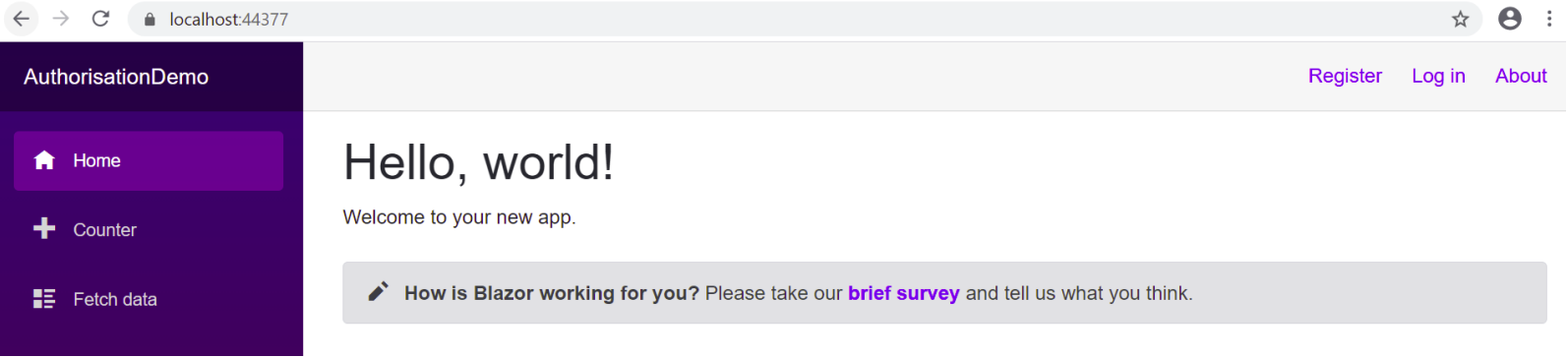
```
PM> update-database  
Build started...  
Build succeeded.  
Done.  
PM> |
```

Again, if successful, the NPM console should say done and reflect the above screenshot, now we can flick over to SSMS and confirm our tables have been created:

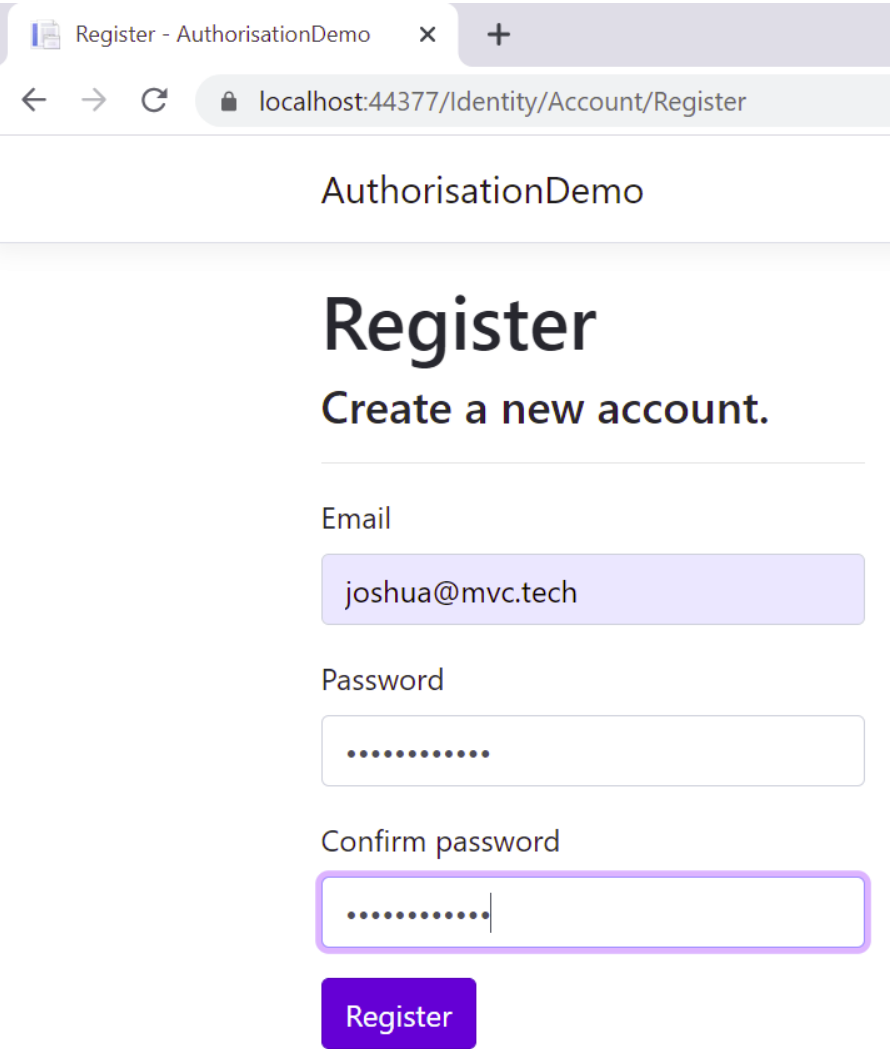


Above, you can now see the database has a bunch of tables related to user accounts and authorisation.

Now it's time to fire up the applicaton by pressing f5 in visual studio, if all goes to plan you should be greeted with a web browser window like below:

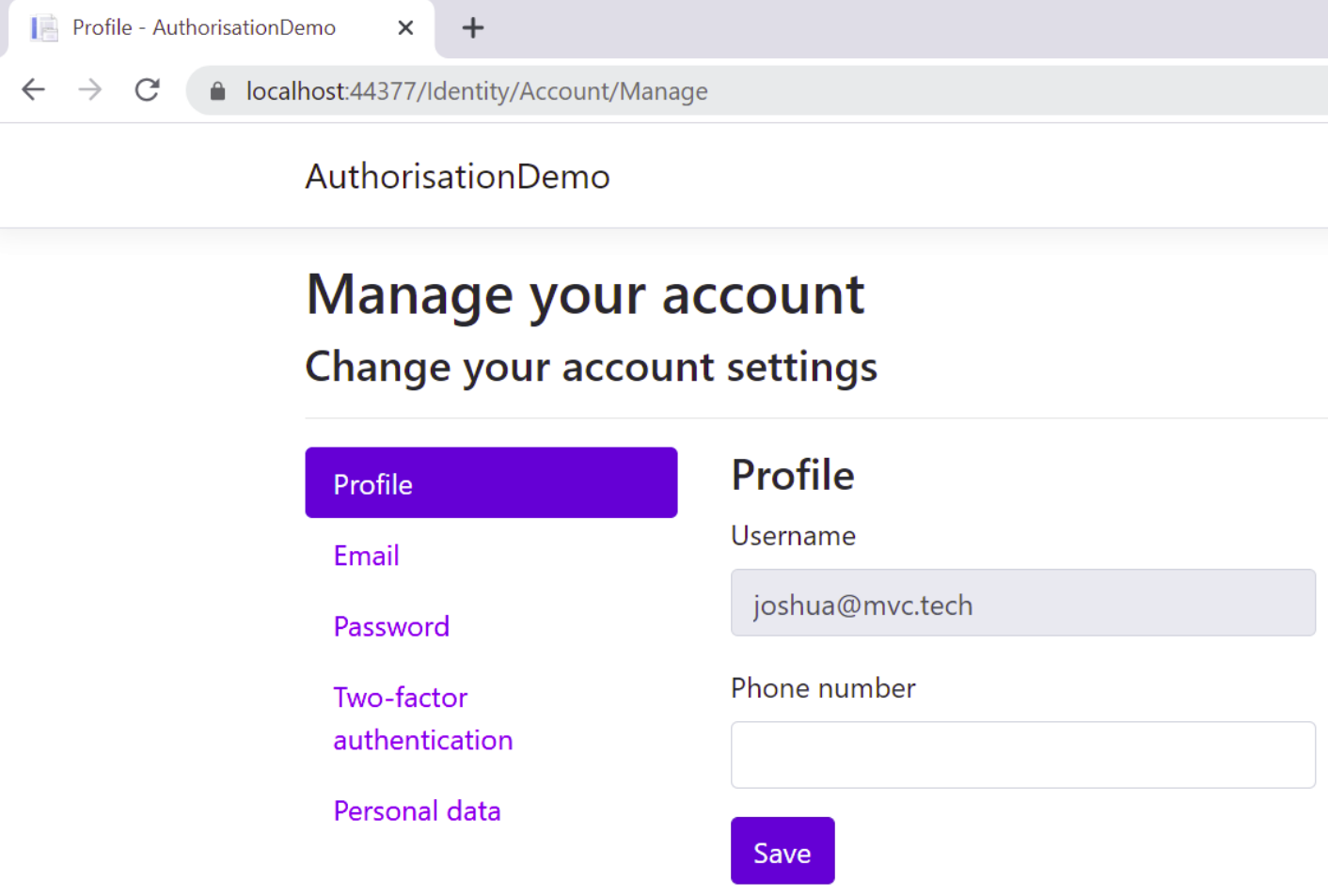


Click Register and bash in an email adress and password (note, it's possible and not terribly complex to add google authentication or facebook etc to this page if required), and click register:



Since we have not configured an SMTP account to use with authorisation and doing so is out of scope of this post (clicking the docs shown on the link will explain in detail how to do this) it will show a link in the browser asking you to confirm the account instead of emailing it, click the confirm link then login with your newly created credentials.

If successfull, the website should now greet you and offer the option to log out, click on the greeting text "Hello, email@address" and it should take you to your account managment page:



Here, a whole host of options are presented such as adding a microsoft 2 factor authenticator to your account, changing your password and more.

However, in the profile only a phone number can be provided, to customise this we want to add an additional user profile field, "Date of Birth" (Note, all these pages are automagically generated and do not exist in your app, so will need to be created and overridden)

Customising Identity User

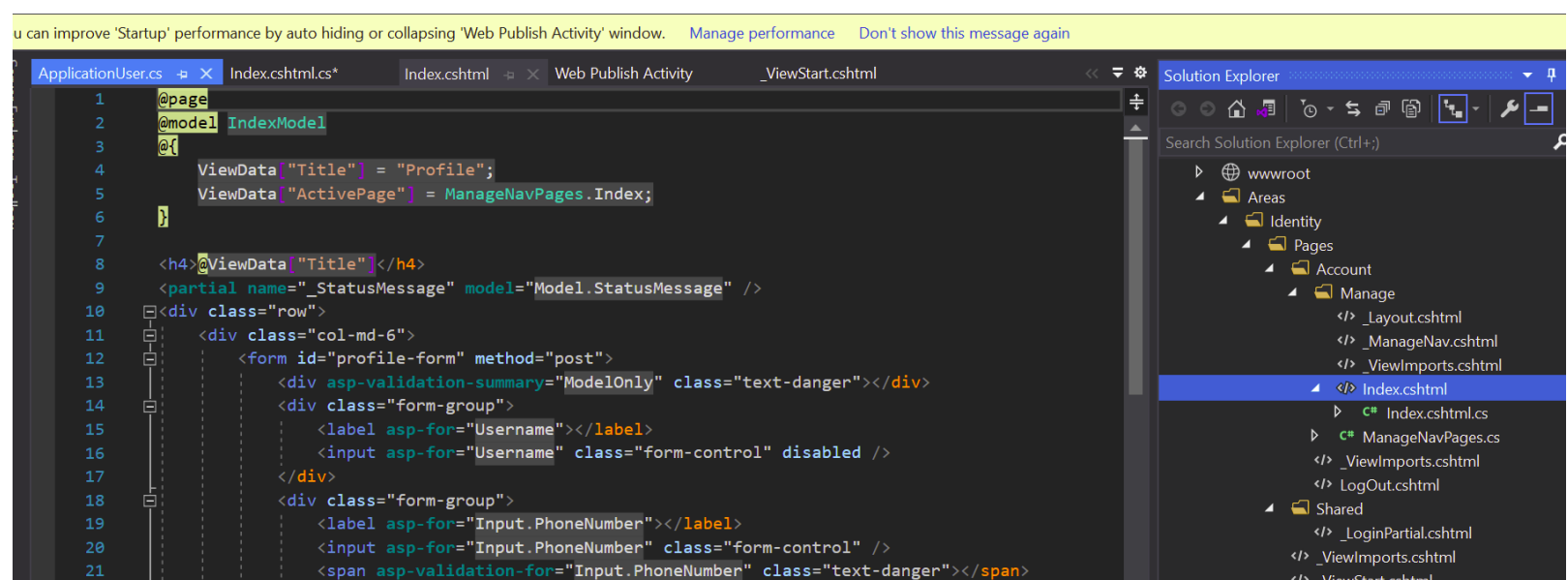
Up to now, we have only discussed standard functionality, now that's out of the way we can start doing fun stuff, adding in new functionality.

The first thing we need to do is scaffold up the Manage user pages so we can add in new fields, Right click on the "Data" folder in solution explorer, click: add => new scaffold item:

Then in the resulting window, click Identity:

Then select the following, ensuring you select the ApplicationDbContext:

Once you have done this, navigate in the solution explorer to: Areas => Identity => Pages => Account => Manage and click on Index.cshtml, note that you can see the phone number HTML field, we are going to add a new field called Date of birth, but first we need to alter the IdentityUser account so the DateOfBirth property can be saved back to the database:

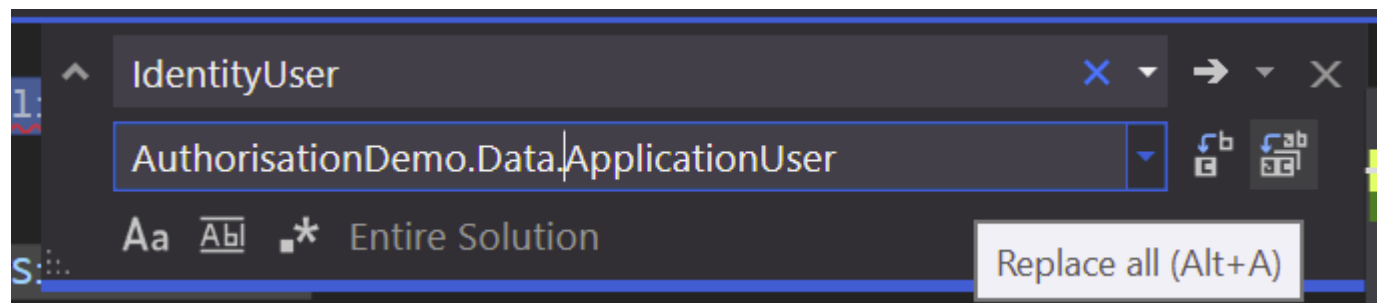


The first thing we need to do now is right click on the "Data" folder and create a new class called ApplicationUser, this needs to inherit from IdentityUser and have the new custom date of birth property, the code for this should look like this:

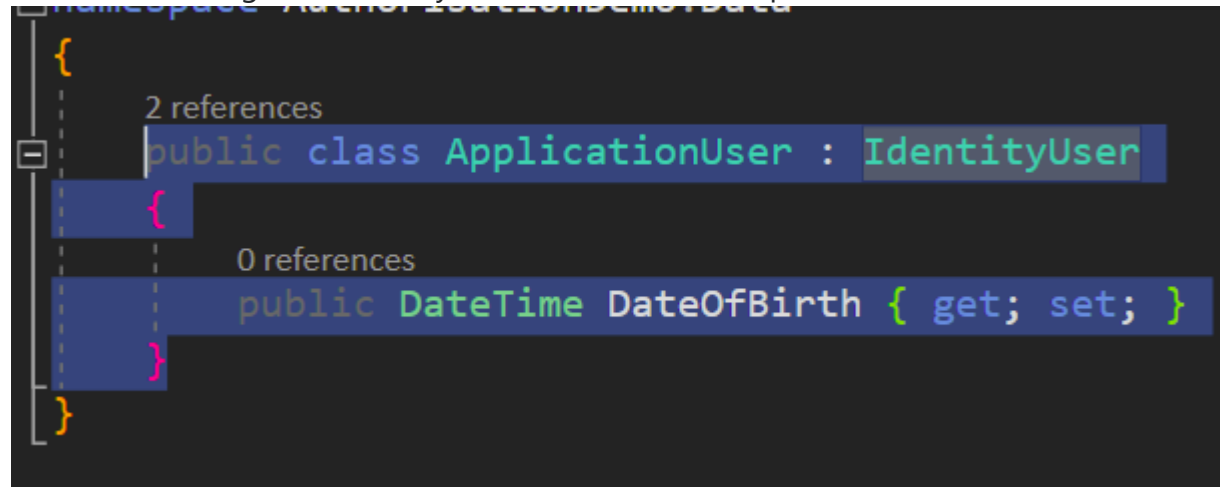
```
public class ApplicationUser : IdentityUser
{
    public DateTime DateOfBirth { get; set; }
}
```

the second thing we need to do now is to replace globally across the application all references to IdentityUser and replace it with ApplicationName.Data.ApplicationUser (Replace ApplicationName with your app name so the namespace matches), this should update around

53 records (Using the full namespace to avoid going into each view and adding: @using AuthorisationDemo.Data) :



Once you have replaced all instances of IdentityUser with ApplicationName.Data.ApplicationUser, go back into the ApplicationUser class and update it back to inheriting from IdentityUser as the find and replace will have broken the code here:

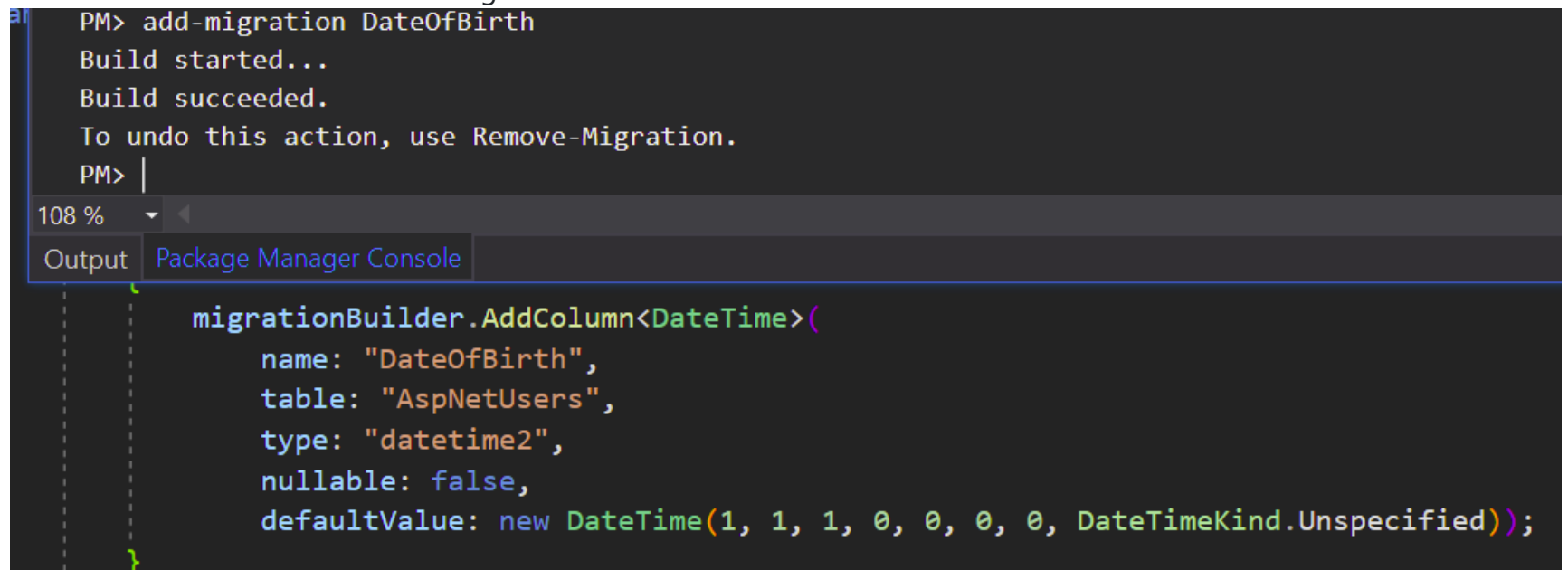


The final thing we need to do now before adding the new migration and updating the database is to update the ApplicationDbContext so it inherits from IdentityDbContext<ApplicationUser> instead of just IdentityDbContext, also we need to add in the on model building code, the final amended class should look like this:

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
        : base(options)
    {
    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);
    }
}
```

Now back into NPM and run: add-migration DateOfBirth:



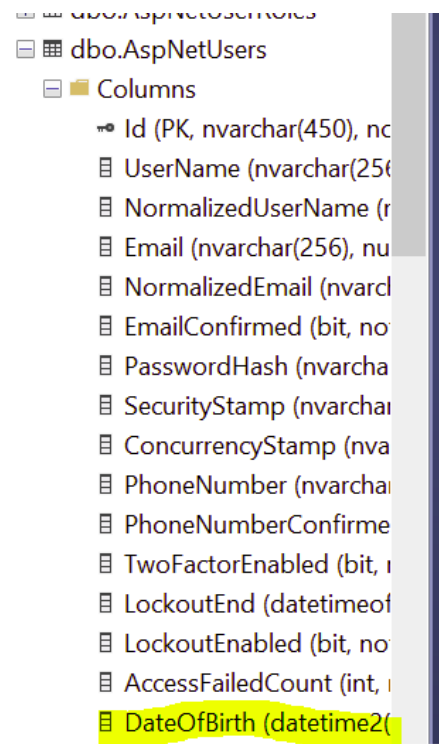
Once completed, it should generate the migration code for our new column shown above, we can now run: update-database:

```
PM> update-database
Build started...
Build succeeded.
Done.
PM> |
```

108 %

Output Package Manager Console

Once this has completed successfully as shown above, flick across into SSMS and refresh the table, we should see our new column in the AspNetUsers table:



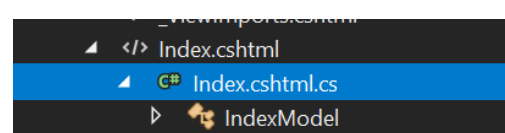
Going back into Visual Studio now, let's edit the Areas/Identity/Pages/Manage/Index.cshtml page and add the following date of birth field below the phone number:

```
<div class="form-group">
    <label asp-for="Input.DateOfBirth"></label>
    <input asp-for="Input.DateOfBirth" type="date" class="form-control" />
    <span asp-validation-for="Input.DateOfBirth" class="text-danger"></span>
</div>
```

The code should look something like this now:

```
<div class="row">
    <div class="col-md-6">
        <form id="profile-form" method="post">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="Username"></label>
                <input asp-for="Username" class="form-control" disabled />
            </div>
            <div class="form-group">
                <label asp-for="Input.PhoneNumber"></label>
                <input asp-for="Input.PhoneNumber" class="form-control" />
                <span asp-validation-for="Input.PhoneNumber" class="text-danger"></span>
            </div>
            <div class="form-group">
                <label asp-for="Input.DateOfBirth"></label>
                <input asp-for="Input.DateOfBirth" type="date" class="form-control" />
                <span asp-validation-for="Input.DateOfBirth" class="text-danger"></span>
            </div>
            <button id="update-profile-button" type="submit" class="btn btn-primary">Update Profile</button>
        </form>
    </div>
</div>
```

Finally we need to edit the code behind for this page by clicking the arrow next to the index.cshtml page and opening the Index.cshtml.cs code page:



In this code we need to update the viewmodel to contain the new field as get/save it back to the database, copy and paste in the following code over the top of the existing:

```

public partial class IndexModel : PageModel
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly SignInManager<ApplicationUser> _signInManager;

    public IndexModel(
        UserManager<ApplicationUser> userManager,
        SignInManager<ApplicationUser> signInManager)
    {
        _userManager = userManager;
        _signInManager = signInManager;
    }

    public string Username { get; set; }

    [TempData]
    public string StatusMessage { get; set; }

    [BindProperty]
    public InputModel Input { get; set; }

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

        public DateTime DateOfBirth { get; set; }
    }

    private async Task LoadAsync(ApplicationUser user)
    {
        var dbuser = await _userManager.GetUserAsync(User);
        var userName = dbuser.UserName;
        var phoneNumber = dbuser.PhoneNumber;
        var DOB = dbuser.DateOfBirth;
        Username = userName;

        Input = new InputModel
        {
            PhoneNumber = phoneNumber,
            DateOfBirth = DOB
        };
    }

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

        await LoadAsync(user);
        return Page();
    }

    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();
    }
    if(Input.DateOfBirth!= user.DateOfBirth)
    {
        user.DateOfBirth = Input.DateOfBirth;
    }
    if(Input.PhoneNumber != user.PhoneNumber)
    {
        user.PhoneNumber = Input.PhoneNumber;
    }
    var success = await _userManager.UpdateAsync(user);
    if(!success.Succeeded)
    {
        StatusMessage = "Unexpected error when trying to update user.";
        return RedirectToPage();
    }

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

That's it, we should now be done, lets fire up the web app and view the user profile again:

Profile

Email

Password

Two-factor authentication

Personal data

Profile

Username

joshua@mvc.tech

Phone number

01904339454

DateOfBirth

23/06/1984

Save

As you can see now, the additional date of birth field now shows and if we populate it and click save:

Profile

Your profile has been updated

Username

joshua@mvc.tech

Phone number

01904339454

DateOfBirth

23/06/1984

Save

You get a nice little message informing you the changes have been saved back to the database.

This concludes this introduction to customising IdentityUser in Blazor, additional tables can be linked in a similar manner using fluent annotation but this is out of scope of the current post.

Thanks for reading and happy coding!

Comments

Comments are closed

© 2020 Copyright MVCTech