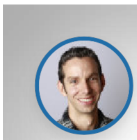


Andrew Lock | .NET Escapades

Sponsored by Nick Chapsas—Want to learn how to build elegant REST APIs in .NET? Get **5% off** Nick's latest course **"From Zero to Hero: REST APIs in .NET"**!

September 19, 2023 ~15 min read

ASP.NET CORE .NET 8 ASP.NET CORE IDENTITY MINIMAL APIS AUTH

Should you use the .NET 8 Identity API endpoints?

Share on:

In my previous post I described the new ASP.NET Core Identity API endpoints introduced in .NET 8. In this post I look more at the design of these endpoints, the implications of using them, and finally, whether or not I think it's a good idea to use them at all.

> **Anyone who's heard of Betteridge's law can probably guess where I'm going with this, but it's not *quite* as cut-and-dried as that.**

The opinions in this post are obviously just my own, but they're based on both how the ASP.NET team have described the APIs, what other people think about the design, and also my own feelings. My thoughts are predominantly based on the implementation as of .NET 8 preview 7.

Andrew Lock | .Net Escapades

Want an email when there's new posts?

Subscribe

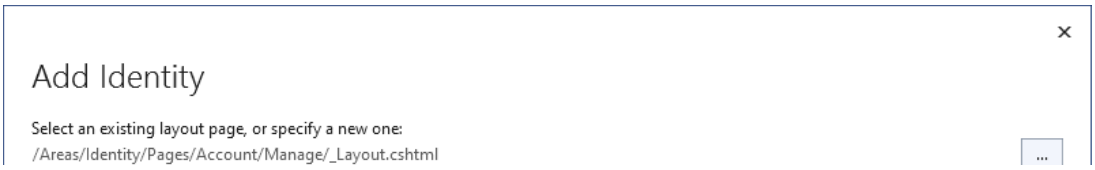
What are the new Identity API endpoints?

As I described in my previous post, the Identity API endpoints are new in .NET 8, and are meant to provide an "API" version of the default Razor Pages UI.

The ASP.NET Core Identity framework consists of multiple layers:

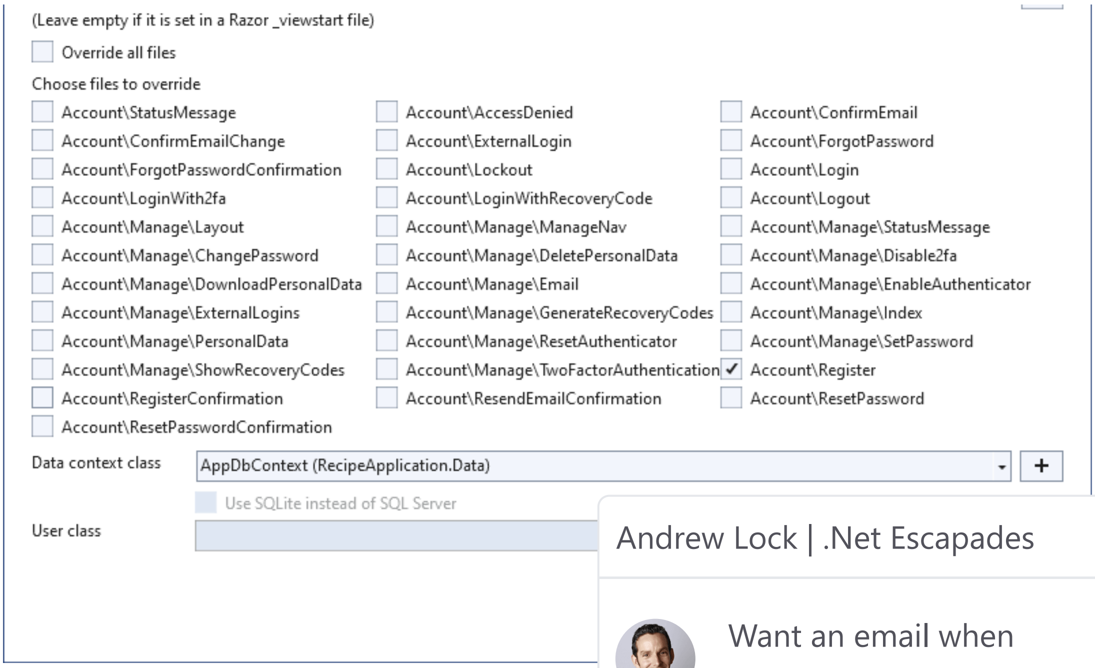
- Data model consisting of types like `IdentityUser` and `IdentityRole`
- Stores for persisting the data model to or from a database
- Manager classes for handling sign-in and out

The default Identity UI Razor Pages add another layer over these, providing multiple pages for managing users and for authenticating.



Add Identity

Select an existing layout page, or specify a new one:
/Areas/Identity/Pages/Account/Manage/_Layout.cshtml



(Leave empty if it is set in a Razor _viewstart file)

□ Override all files

Choose files to override

□ Account\StatusMessage
□ Account\AccessDenied
□ Account\ConfirmEmail
□ Account\ConfirmEmailChange
□ Account\ExternalLogin
□ Account\ForgotPassword
□ Account\ForgotPasswordConfirmation
□ Account\Lockout
□ Account\Login
□ Account\LoginWith2fa
□ Account\LoginWithRecoveryCode
□ Account\Logout
□ Account\Manage\Layout
□ Account\Manage\ManageNav
□ Account\Manage\StatusMessage
□ Account\Manage\ChangePassword
□ Account\Manage\DeletePersonalData
□ Account\Manage\Disable2fa
□ Account\Manage\DownloadPersonalData
□ Account\Manage\Email
□ Account\Manage\EnableAuthenticator
□ Account\Manage\ExternalLogins
□ Account\Manage\GenerateRecoveryCodes
□ Account\Manage\Index
□ Account\Manage\PersonalData
□ Account\Manage\ResetAuthenticator
□ Account\Manage\SetPassword
□ Account\Manage\ShowRecoveryCodes
□ Account\Manage\TwoFactorAuthentication
☑ Account\Register
□ Account\RegisterConfirmation
□ Account\ResendEmailConfirmation
□ Account\ResetPassword
□ Account\ResetPasswordConfirmation

Data context class: AppDbContext (RecipeApplication.Data)

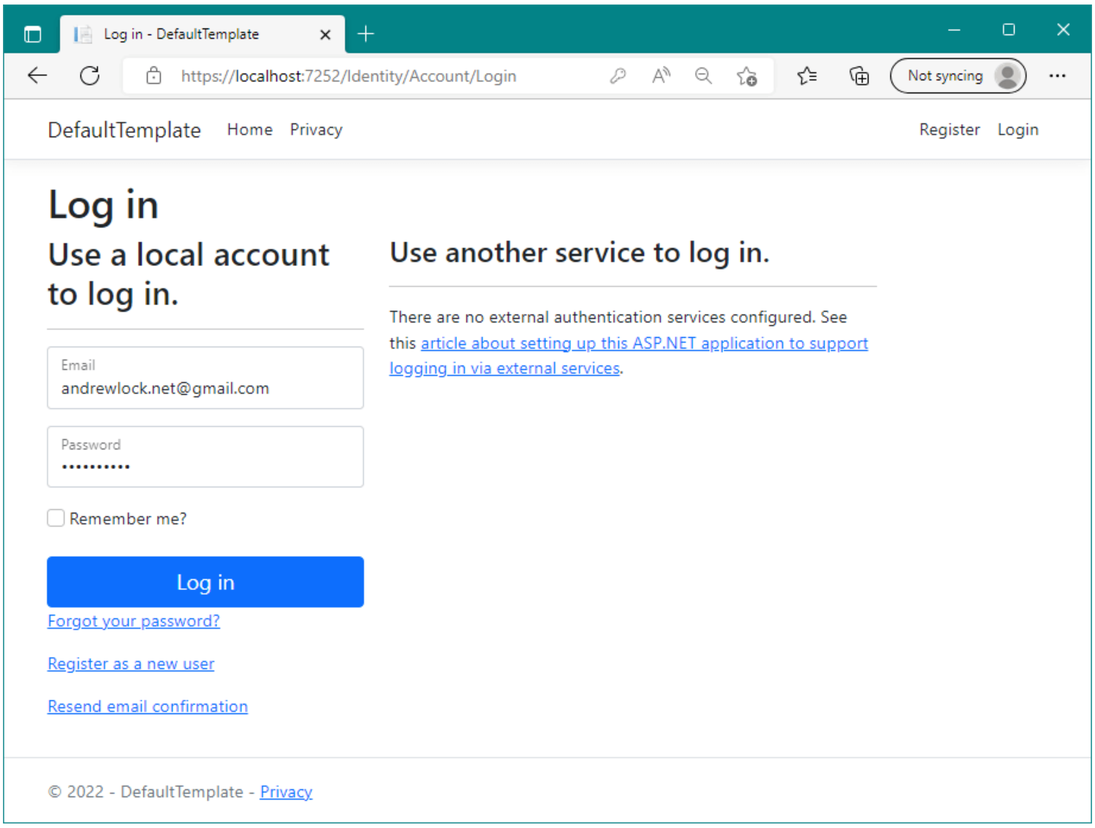
□ Use SQLite instead of SQL Server

User class

Andrew Lock | .Net Escapades

Want an email when there's new posts?

These Razor Pages add standard cookie-based authentication, external login capability (through Google/Facebook/any OpenID Connect Identity Provider), multi-factor authentication (MFA) support etc.



Log in

Use a local account to log in.

Email: andrewlock.net@gmail.com

Password: ••••••••••

□ Remember me?

Log in

Forgot your password?

Register as a new user

Resend email confirmation

Use another service to log in.

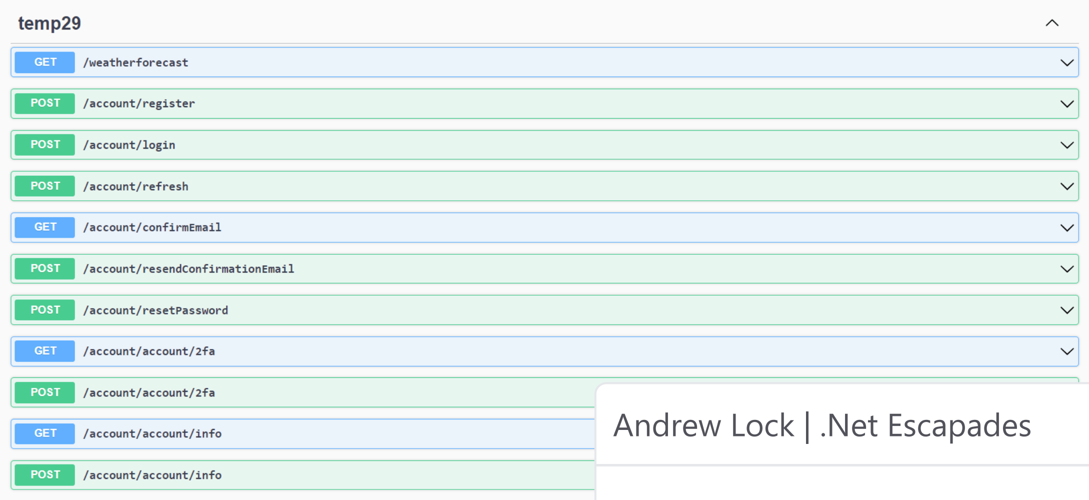
There are no external authentication services configured. See this article about setting up this ASP.NET application to support logging in via external services.

© 2022 - DefaultTemplate - Privacy

The new Identity endpoints can be added with a single line:

```
app.MapGroup("/account").MapIdentityApi<AppUser>();
```

and your app will have a number of new endpoints



temp29

GET /weatherforecast
POST /account/register
POST /account/login
POST /account/refresh
GET /account/confirmEmail
POST /account/resendConfirmationEmail
POST /account/resetPassword
GET /account/account/2fa
POST /account/account/2fa
GET /account/account/info
POST /account/account/info

Andrew Lock | .Net Escapades

Want an email when there's new posts?

So what's the use case for the new endpoi

What problem are they trying to solve?

The new Identity endpoints added in .NET 8 are intended to make it easier to add ASP.NET Core Identity to API server applications and front-end SPA-style client applications, such as those built with JavaScript or Blazor. To better understand why the endpoints were added, it's worth considering the existing recommended solution.

Consider the following scenario:

- You have a simple application consisting of an ASP.NET Core backend that provides APIs (for example using minimal APIs, WebAPI controllers, or gRPC endpoints etc).
- You have a client side SPA application that talks to the API (written using Angular, Vue, or Blazor for example).
- You want to add user accounts (and so authentication and authorization) to your app.

Currently (without the Identity endpoints) there are three main approaches you could take to the solution, and then variations on these:

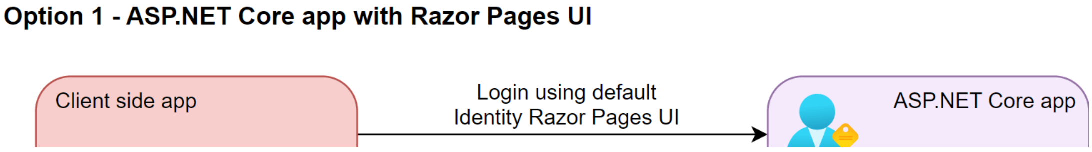
1. Add ASP.NET Core Identity directly to your API app, using the the default Razor Pages UI.
2. Add OpenID Connect (OIDC) to your API app, and defer user

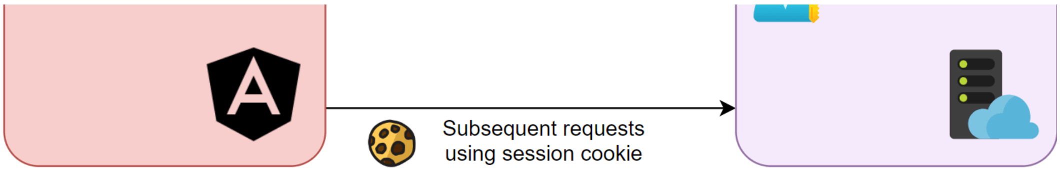
2. Add OpenID Connect (OIDC) to your API app, and defer user management to a third-party, OIDC identity provider like Okta, Google, or Entra ID or Azure Active Directory B2C.
3. Create your own standalone "Identity" app using [IdentityServer]() or [OpenIddict]() (for example) along with ASP.NET Core Identity for user management. There are a couple of variations on this approach:
   - Use the default Razor Pages UI. This app manages user accounts, as well handling password management, account recovery, and MFA etc
   - Use the default Razor Pages UI w authentication to an external prov gives you the flexibility of manag but you move the responsibility f etc to a third-party.

Andrew Lock | .Net Escapades

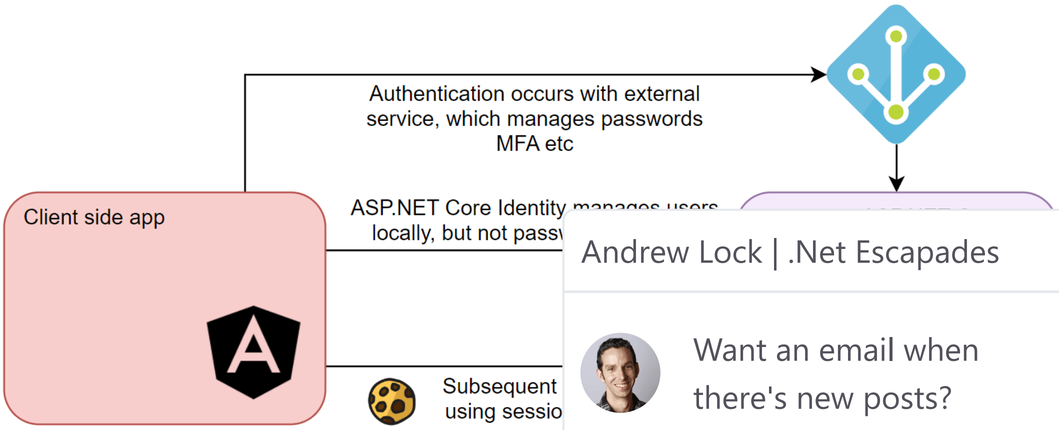
Want an email when there's new posts?

The following diagram shows these options in another format:


Option 1 - ASP.NET Core app with Razor Pages UI
Client side app
Login using default Identity Razor Pages UI
ASP.NET Core app



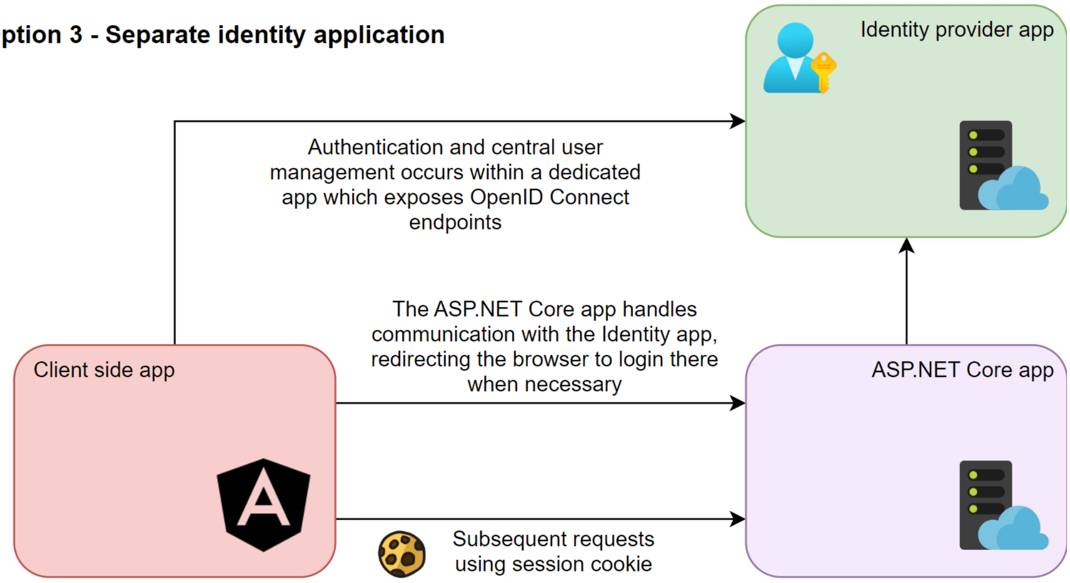
Option 2 - ASP.NET Core app with Open ID Connect and external authentication


Authentication occurs with external service, which manages passwords MFA etc
Client side app
ASP.NET Core Identity manages users locally, but not passw
Subsequent
using sessio

Andrew Lock | .Net Escapades

Want an email when there's new posts?

Option 3 - Separate identity application


Identity provider app
Authentication and central user management occurs within a dedicated app which exposes OpenID Connect endpoints
The ASP.NET Core app handles communication with the Identity app, redirecting the browser to login there when necessary
Client side app
ASP.NET Core app
Subsequent requests
using session cookie

I'll get into the subtleties of each of these approaches later but it's important to recognise that while having choices can be a good thing, it can also make it harder to know what the "right" choice is for your situation.

Each approach has a different level of complexity associated with it, and is a combination of multiple factors:

- Do you need to integrate with a separate service (e.g. Okta, Azure AD)?
- Do you need to build, deploy, and manage a separate app (an IdentityServer app)?
- Do you need to understand OpenID Connect (everything except option

- Do you need to understand OpenID Connect (everything except option 1)?
- How well does the solution "fit in" to your existing app's paradigms (does the "login page" in your app feel like *part* of your app, or something completely separate)?

On most measures, option 1 (add ASP.NET Core Identity to your app and use the default Razor Pages UI) could be considered the simplest. You can add ASP.NET Core Identity and the default Razor Pages UI to your app by adding a few packages, updating your database sc services.

However, the "simplicity" of that approach how well does it fit into your existing app' is likely "not very well" if you're adding this to an API + SPA app.

Andrew Lock | .Net Escapades

Want an email when there's new posts?

- The full-page refreshes of Razor Pages can feel like a clunky step back in the context or an SPA that otherwise uses client-side navigation
- The default styling of Razor Pages uses bootstrap. If your app is using something else, you may need to update the Razor for 30+ pages to match your app's style.
- Having an app that serves both Razor Pages and API endpoints can sometimes be more confusing in the backend, particularly when it comes to error responses, as the app needs to respond in different ways in different cases (return HTML for Razor Page requests and JSON for API requests). This is doable, but is just more complicated.
- *Traditionally* people like (or have been told) to use bearer tokens for authentication when calling APIs. The default Razor Pages UI uses traditional cookie-based authentication, not bearer tokens.

The Identity API endpoints were added with the intention of improving this scenario, as I'll discuss in the next section.

How do the Identity API endpoints aim to solve the problem?

The Identity endpoints added in .NET 8 were intended to simplify the scenario of adding user accounts to an ASP.NET Core API app which is used by an SPA (or mobile) frontend by:

- Exposing minimal API endpoints for managing your user, largely equivalent to the Razor Page Identity UI pages.
- Adding an endpoint for retrieving *bearer tokens*, which can be used for authentication.
- Configuring the app to support beare

It's fairly obvious how these steps tackle th the previous section. By exposing minimal functionality, you can build the UI in your S the APIs without the impedance-mismatch of using Razor Pages. No need for full-page refreshes; no clashes in styling; everything feels "native" to your app.

Andrew Lock | .Net Escapades

Want an email when there's new posts?

Additionally, by adding bearer token authentication, your app can easily use the "expected default" mechanisms often suggested for SPAs. On mobile, using cookies is particularly hard, so access tokens is generally the suggested approach for calling APIs.

So on the face of it, this all seems like a good thing, right? It makes a key scenario easier, by lots of measures. It seems like they should be easy to recommend?

Well...it depends. But currently, in my opinion, there's a lot of reasons *not* to use them.

Reasons you shouldn't use the Identity API endpoints

The Identity endpoints have received a fair amount of publicity for .NET 8, which is not surprising, because many people have been asking for something like this for some time.

Unfortunately, I feel like there are some fundamental problems with the

Identity endpoints, such that I wouldn't recommend you use them. Some of these problems are temporary things that could easily be addressed in future versions. Others are fundamental problems with the Identity endpoints *as a concept*.

> **My opinions in this area have been heavily influenced by discussing these APIs with other authentication- and security-focused developers. I don't pretend to fully understand the security aspects of the Identity endpoints, but when it makes people who *do* understand them uneasy, I get uneasy too 😅**

In the following sections, I'll discuss some
two points are fundamental issues with the
endpoints. The latter points are problems that could feasibly be addressed in future versions of the Identity endpoints.

Andrew Lock | .Net Escapades

Want an email when there's new posts?

- [1. Bearer tokens are falling out of favour for security reasons]
- [2. Leads to inflexible architectures]
- [3. There's a lot missing (that you need to build it yourself)]
- [4. There may be endpoints you don't want, which you can't remove]
- [5. You can't customise the endpoints]

I'll address each of these issues in the following sections.

1. Bearer tokens in the browser should generally be avoided where possible

This is probably the biggest issue with the Identity endpoints, while also potentially being the most controversial complaint as well. With the rise in prominence of single-page/browser-based (JavaScript) applications, the received wisdom has been that "you should use bearer tokens for authentication", rather than cookies.

However, the modern opinion of most security professionals has shifted. These days, it's generally recommended that you *avoid ever having bearer tokens stored in the browser*, due to the complexities involved in securely *obtaining* a token and and in safely *storing* the token while keeping the value secret.

> **This isn't just vague opinions, there's a draft RFC from the IETF discussing recommended architectures, and they mostly focus on how to keep the tokens out of the browser. I'll describe each of these approaches in detail in a future post.**

The recommended approach when your frontend SPA is served from the same domain as your API (and so can shar authentication. Browsers have built-in tech accumulated over 20 years, such as `httpOı` cookies. When coupled with anti-forgery t CSRF in older browsers), using cookies for authentication represents a simple, easy, secure solution.

Andrew Lock | .Net Escapades

Want an email when there's new posts?

If you're *not* hosting your app and API on the same domain, you can still use cookies for authentication with the backend for fronted (BFF) pattern. This is an increasingly common pattern (similarly recommended by the IETF) in which you use cookies to authenticate with a backend .NET app, and then that app handles retrieving and storing the access tokens. With the BFF pattern, the tokens are never sent to the browser.

> **If you're interested in implementing the BFF pattern, Duende provide a self-contained solution based around IdentityServer. Alternatively, Damien Bowden has a plethora of templates and sample apps using different frameworks on his GitHub account which he keeps up to date.**

I could hammer on this point even more, and when discussing this with others *many* more subtle issues arise, especially if we extend our thinking to mobile apps too. But the key point I'm making here is that the pattern that the Identity endpoints use—send a username and password and get back access and refresh tokens—is *not* recommended. Sure, people have asked for it, no doubt. But it's hard to say that this is a pattern you should be using

these days, even in a "toy" application, when there are other (better) options available.

> **If you're going to *insist* on having tokens in the browser, the [IETF have various recommendations on approaches to take](). These are all through the lens of OAuth 2.0/OpenID Connect, but they generally include more protections than the flows suggested by the Identity endpoints.**

In defence of the Identity endpoints, the `/login` endpoint *can* be used in "cookie mode". With this approach, you send a username and password, and the sign in process sets an authentication cookie (and doesn't return a token), just like if you had logged in using the Razor Pages UI.

```
### Login (Cookie mode, persist cooki
POST http://localhost:5117/account/lo
Content-Type: application/json

{
  "username": "andrew@example.com",
  "password": "SuperSecret1!"
}
```

Andrew Lock | .Net Escapades

Want an email when there's new posts?

From a security point of view, there doesn't seem to be anything fundamentally wrong with this approach as far as I can tell. You're using "standard" cookie authentication, as recommended, and as long as you make sure to include CSRF anti-forgery tokens in your requests, you're avoiding all the pitfalls of bearer tokens.

Using the cookie implementation avoids the worst offender in the Identity APIs, but you still may want to reconsider whether the simplistic Identity endpoint approach is the best one to take.

2. You're tying yourself to a specific authentication architecture

The Identity APIs were created on the premise that you want to add user accounts *directly* to an application. That feels like the simplest "grow up path" from not having any accounts, but the trouble is, it ties you strongly to this one approach. And in any realistic business setting, it's likely *not* the approach you want.

First of all, the Identity endpoints don't support any sort of external authentication mechanism. That means users of your app will always need a dedicated username and password for this app alone. That may be fine for toy apps. But if you're building an app for a business, whether it's a line-of-business app or public-facing, that's likely not the approach you want longer term.

For line-of-business apps, your company likely already has some sort of single-sign-on central authentication mech workspaces account, an Entra ID account, c authentication, you'll generally want to del Identity provider, instead of every internal and password.

Andrew Lock | .Net Escapades

Want an email when there's new posts?

Even for public-facing applications, you may want to consider the fact that you might ultimately have *multiple* apps. In that case, you likely *don't* want customers to have different accounts on each app, and instead a central account would be preferable. By splitting authentication into a separate service from the get-go you're future proofing against a common scenario.

By using the Identity APIs and directly sending a username and password to the backend, you're strongly coupling your application to the backend. A far more extensible approach is to delegate the authentication to an external application, while using cookie authentication with the backend application, similar to the approach Damien Bowden describes in his recent post.

3. There's important missing functionality in the endpoints

The first two points I've covered are enough (in my opinion) to advise against using the Identity API endpoints in any production app (unless you're using them in cookie-mode, but even then...). But I'm sure there are people who will want to use them anyway. In which case, it's worth bearing in mind that the Identity endpoints don't necessarily include everything you need.

Firstly, you need to build the UI. That's obviously a given, seeing as it's one of the main reasons given for wanting the Identity APIs in the first place! But compared to the Razor Pages Identity UI, the Identity endpoints are missing quite a few APIs. For example:

- View Personal Data (for GDPR)
- Delete Personal Data (GDPR)
- Download Personal Data (GDPR)
- Manage external Logins
- Generate Recovery Codes

These may not be critical for you, but then
news is you can "just" reimplement these y
then, that was the state prior to the introdu

Andrew Lock | .Net Escapades

Want an email when there's new posts?

Either way, you better make sure you've im
custom endpoints and on the client side correctly, and aren't accidentally exposing any vulnerabilities. After all, the user account management logic is arguably some of the most important in your app, and you're on the hook to implement all of it!

4. You can't change which endpoints are added

Just as you need to add any endpoints which are missing (which is feasible, but annoying), you also can't *remove* any endpoints from your app. That means the `/register` endpoint is *always* there, so anyone will be able to create a new account on your app, whether you want them to or not.

With the Razor Pages default UI, you can at least "override" the pages, and can remove any functionality you want. With the current Identity API endpoints, that's not possible. That functionality *may* come in the future, but as far as I can tell, for .NET 8, it won't be there.

Even if you ignore all the potential security or architecture concerns, this one feels like it could well end up being a blocker for many people.

5. You can't change how the endpoints work

In a similar vein, just as you can't remove any endpoints, you also can't *change* any endpoints. The current implementation has no facility for customisation, so if they don't work *exactly* as you want, then you'll need to implement the whole API surface yourself from scratch.

> **As an aside, there isn't a "scaffold" approach that works with the new Identity endpoints. The team have discussed a simple option that "dumps" the code for the endpoints in your app so you can customise them if you need to. This obviously solves points 4 and 5, though it also leaves you on the hook for maintaining all those endpoints which may not be something you want to take on...**

Ideally, you won't need to customise the e common cases where you might want to:

Andrew Lock | .Net Escapades

Want an email when there's new posts?

- Collecting additional claims when you to provide a nickname at the same time as they register, for example, then you're out of luck currently.
- Implementing additional MFA flows. If you want to add support for FIDO2, WebAuthn, or Duo Security then you can't use the Identity endpoints.

It's worth noting that as the endpoints use the standard ASP.NET Core Identity managers and stores, anything you can implement at *that* layer, should work without any issues. For example you can customise the password requirements to update the `IdentityOptions` and that will all flow through to your endpoints.

My overall recommendation

I'll keep this short, because if you've read the post, you'll know my feelings already. Fundamentally, I think baking a token endpoint directly into ASP.NET Core Identity is going to encourage people to use a non-standard, less-secure solution, when we have many good alternatives. The fact that people are (I'm sure) clamouring for it doesn't feel like a good enough reason, as it's *very* unlikely people are choosing to take this approach *while having a full understanding of the implications*.

Purely from an architectural point of view, I can't really recommend you use the Identity API endpoints, but worst case, as long as you use them in cookie mode, you should be ok from the security point of view. But I would still

recommend using external authentication instead where possible.

Whether the Identity APIs are otherwise suitable for your needs will depend very much on your requirements. If you want to customise any of the flows (registration flows for example), or you want to remove any endpoints (such as the registration endpoint) then the Identity API endpoints won't work for you currently. Although I expect these limitations will be addressed in a future version of .NET.

Summary

Andrew Lock | .Net Escapades

Want an email when there's new posts?

In this post I provided an overview of the r in .NET 8. I described how they're trying to solve the problem of the impedance mismatch between client-side apps and the Razor Pages Default Identity UI. I then described some of the reasons why I think this approach is misguided and could lead to problems for you down the line.

Overall, providing APIs for generating custom access tokens for storage in the browser feels like a step backwards from the increasing trend towards using cookies. Patterns like the BFF pattern provide recommended, extensible, solutions. "Simple" is all well and good, but it comes at a cost in the Identity APIs, and that cost is likely not apparent unless you're steeped in authentication security as your bread and butter.

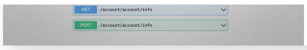
FOLLOW ME

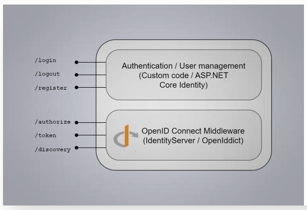
ENJOY THIS BLOG?

Buy me a coffee

PREVIOUS

Introducing the Identity API endpoints: Exploring the .NET 8 preview - Part 8




/login
/logout
/register
/authorize
/token
/discovery
Authentication / User management (Custom code / ASP.NET Core Identity)
OpenID Connect Middleware (IdentityServer / OpenIddict)

NEXT

Can you use the .NET 8 Identity API endpoints with IdentityServer?

2 reactions

☺ 👍 2

6 comments · 12 replies – *powered by giscu*

Andrew Lock | .Net Escapades

Want an email when there's new posts?

© 2023 Andrew Lock | .NET Escapades. All Rights Reserved. | Image credits