



CONNECT WITH ME:



BLOG

TRAINING

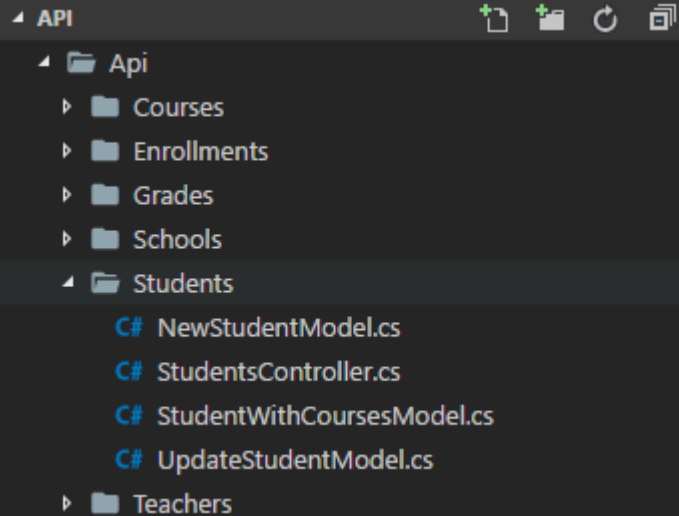
MENTORING

DEV TIPS

ARCHITECTURE EBOOK

TOOLS USED

ABOUT

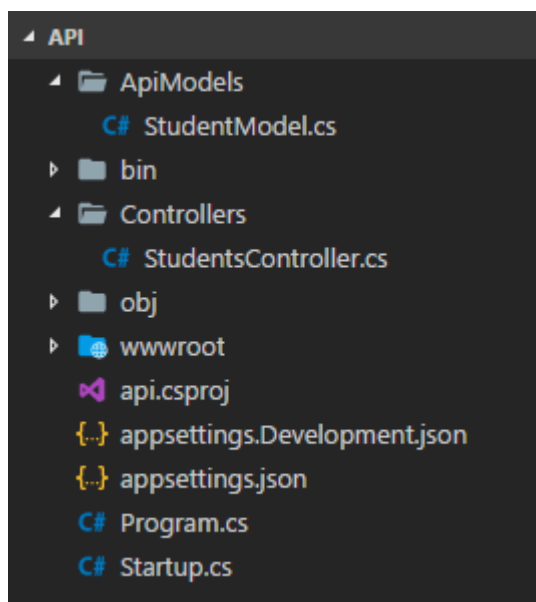


STEVE SMITH 28 FEB 2018 17 COMMENTS

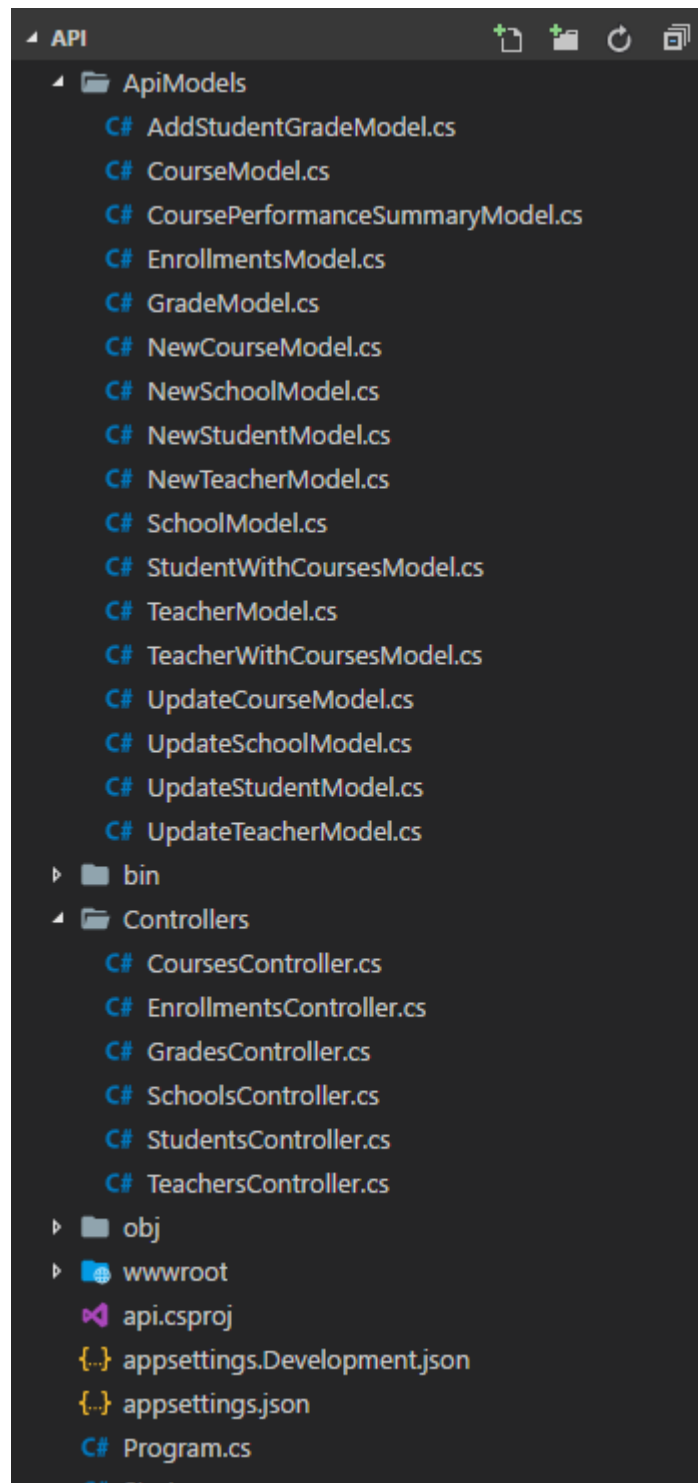
API Feature Folders


I've written about [feature folders for ASP.NET Core](#) before, and how [Razor Pages does a great job of solving this problem](#) for page/view-based endpoints. However, I wanted to take a moment to address APIs, which are an increasingly important part of modern web applications.

In ASP.NET Core (and unlike ASP.NET 5 / Web API 2), Web API controllers are just controllers. You don't need to inherit from a different base type or anything like that. What's more, your API controllers should be returning DTOs that are separate from your underlying domain or data model. In a typical, default project you might have something like this:

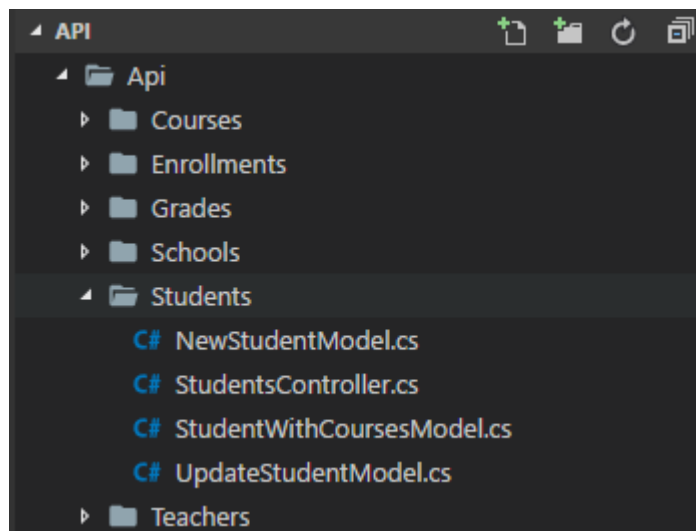


Over time, this tends to grow and can become unwieldy. It's not unusual to have a dozen or more controllers, and your models will likely include things like `NewWhateverModel`, `UpdateWhateverModel`, `WhateverSummaryModel`, etc. as you create models specific to particular API needs. You might start to have a project structure that looks something like this:



 Startup.cs

What I've found to be a better organization is to do away with the Controllers folder (or keep it around if you're using view-based controllers) and instead use feature folders for your APIs. I'm partial to having a root level API folder but if you'd prefer to put your features in the root of the project that would work, too. Within each feature folder, you include the controller along with any model types it needs to work with, like this:



Obviously one benefit of this approach is that it's more cohesive. Things that change together are located physically next to one another, and the friction involved in moving between different folders with too many files in them is greatly reduced. Another nice thing about this approach is that it just works. Unlike view-based controllers (which need something like [this](#)), you don't need to change anything about how ASP.NET Core is configured to have this organization structure work for you.

I'm curious, how many of you are using a non-default organization structure for your ASP.NET (Core) projects? What are the pros and cons you see of your approach?

// ONLINE TRAINING

- › [ASP.NET Core Quick Start](#)
- › [Domain-Driven Design Fundamentals](#)
- › [Refactoring Fundamentals](#)
- › [Kanban Fundamentals](#)
- › [SOLID Principles of OO Design](#)
- › [Pair Programming](#)

// ABOUT ME



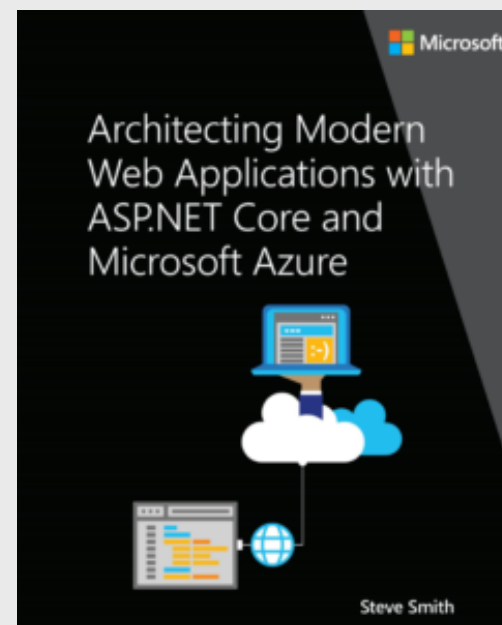
Steve is an experienced software architect and trainer focused on improving team skills with DDD

and ASP.NET Core. His courses on [Pluralsight](#) and [DevIQ](#) help developers write better, more maintainable code. He is

available for [application assessments](#) and [team mentoring](#) engagements.

[Sign up](#) to receive a free developer tip from Steve in your inbox every Wednesday.

// FREE ARCHITECTURE EBOOK



[Get the Book!](#)

// RECENT ARTICLES

- › [Never use the same value for two IDs \(or other values\) in your tests](#)
- › [Configuring a Local Test Email Server](#)
- › [Integrate GitHub and Discord with Webhooks](#)
- › [Stored Procedures, ORMs, and GraphQL](#)
- › [Configuring Logging in Azure App Services](#)

// POPULAR ARTICLES

- › [Force Nuget to Reinstall Packages without Updating](#)
1,097 views
- › [The More You Know The More You Realize You Don't Know](#)
964 views
- › [Easily Add Images to GitHub](#)
874 views
- › [Why Delete Old Git Branches?](#)
466 views
- › [How to Hide the Connection Bar in Remote Desktop Connection \(RDP\)](#)
461 views

// CATEGORIES

- › [Iraq](#) (118)
- › [Personal](#) (7)
- › [Productivity](#) (89)
- › [Security](#) (6)
- › [Software Development](#) (507)
- › [Uncategorized](#) (828)

// RECENT TWEETS BY @ARDALIS

Tweets by @ardalis



Steve "ardalis" Smith Retweeted

**NASA**
@NASA

"American astronauts on American rockets from American soil, showing you what Americans can do when we come together as a team." —
[@Astro_Flow](#) on the [#LaunchAmerica](#) mission:



12h



Steve "ardalis" Smith Retweeted

**Spaceflight Now**
@SpaceflightNow

Russian cosmonaut Ivan Vagner ([@ivan_mks63](#)) took this photo as the International Space Station flew above the Kennedy Space Center yesterday.

[Embed](#)[View on Twitter](#)



 FILED UNDER: [SOFTWARE DEVELOPMENT](#) TAGGED WITH: [ASP.NET](#), [ASP.NET CORE](#), [WEB API](#)

About Steve Smith

Steve is an experienced software architect and trainer, focusing currently on ASP.NET Core and Domain-Driven Design. His courses on [Pluralsight](#) help developers write better, more maintainable code. He is available for [application assessments](#) and [team mentoring engagements](#).

ALSO ON ARDALIS

Configuring a Local Test Email Server

a month ago • 3 comments

It's been a few years since I wrote about using a tool like Smt4Dev for local test ...

Running Integration Tests in Build ...

7 months ago • 11 comments

A pain point for some organizations is figuring out how to run tests that ...

Moving from Controllers and ...


6 months ago • 29 comments

(or Controllers are dinosaurs – it's time to embrace Endpoints) Update Feb ...

Git Autocorrect

4 months ago • 7 comments

I don't know how I discovered this but apparently you can ...

17 Comments Ardalis  [Disqus' Privacy Policy](#)

 Login ▾

 Recommend

 Tweet

 Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

**Connie DeCinko** • 2 years ago

Steve, do you use Services and Repositories and if so, where do you store those?

2 ^ | v • Reply • Share ›

**James Hickey** • 10 months ago

Been doing this, but with nested feature folders since different "features" represent overarching areas of the product (kinda like domains / sub-domains...) and there are MANY features per product in the org I work for.

Works fantastic.

^ | v • Reply • Share ›

**kjbetz** • 2 years ago

I've been doing feature folders for awhile now... following Jimmy Bogard's feature folder / MediatR pattern. So my API structures look like this:

```
--AwesomeApi
----Data
-----Migrations
-----AwesomeApiContext.cs
----Domain
-----Entity1.cs
-----Entity2.cs
-----Entity3.cs
----Features
-----Feature1
-----_Feature1Controller.cs ('_' in name to keep at top of list in code editor)
-----List.cs
-----Details.cs
-----Add.cs
-----Feature2
```

[see more](#)[^](#) | [v](#) • [Reply](#) • [Share](#) ›**Barret** • 2 years ago

I've always been a fan of a common code class library that's separate from any other projects in the solution to host all my models, both the base models and the viewmodels that might be needed in multiple places. I keep them grouped together by function sub folder and I always place the viewmodels in the same .cs file as the base models. If I have viewmodels in the API project, I also tend to include them in the same file as the controller itself. I prefer fewer, larger files with more related functionality where possible.

Generally I will dump all the API controllers into one folder, but I always try to break models down into logical subfolder groupings.

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**ardalis** Mod ➔ Barret • 2 years ago

If that's working for you, stick with it. I personally prefer shorter files and generally one file per class, with the exception of nested classes (where it wouldn't really make sense anyway). I don't like really long classes or files and where I see them I recognize they're generally a code smell and indicate I'm trying to do too much (violating Single Responsibility, etc.). If I combine a bunch of classes into one file it makes it harder to see this kind of thing. I find that using folders rather than files as my grouping mechanism works fine (for me) in the vast majority of cases.

I do admit that when I'm demonstrating some code as part of a training class or presentation, I'll usually just work in one file, but for anything I plan to keep I'll refactor the classes into separate files. Fortunately tooling usually makes this very easy to do (e.g. ctrl+., enter in VS2017).

I do recommend using a shared project for internal APIs/client apps. In order to allow the API and the client code to evolve independently, I recommend putting the shared types into a nuget package, so that it's easy to know exactly what version of the DTOs each application is working with.

[^](#) | [v](#) • [Reply](#) • [Share](#) ›**George Paoli** • 2 years ago • edited

Hi, see with my projects are organized in ASP.NET Core Web API:

- Api (csproj)
 - Feature1
 - Feature1Controller.cs
 - Model.cs (all DTOs for controller)
 - Feature2
 - Feature2Controller.cs
 - Model.cs (all DTOs for controller)
- Domain (csproj - share with only a before API)
 - Entities (all EF entities and EF Context)
 - Modules
 - Feature1
 - Model.cs (all DTOs for service)
 - Feature1Service.cs (business logic)
 - IFeature1Service.cs (interface for DI)
 - Feature2

[see more](#)

^ | ▾ • Reply • Share ›



Tanvir Ahmad Arjel → George Paoli • 7 months ago

- Filter (custom ASP.NET Core Filters)
- Middlewares (custom ASP.NET Core Middlewares)

... Should not be in Infrastructure in DDD concept. These two are sole part of Web Project.

^ | ▾ • Reply • Share ›



ardalis Mod → George Paoli • 2 years ago

Yes, this is pretty much exactly what I recommend today. See also my Clean Architecture repo on GitHub (that you'll find looks very similar). <https://github.com/ardalis/...>

^ | ▾ • Reply • Share ›



Connie DeCinko → ardalis • 2 years ago

Is the repository pattern still valid? I found a couple devs who say Core 2.1 eliminates the

need for repo/UoW.

^ | v • Reply • Share ›



ardalis Mod → Connie DeCinko • 2 years ago

Of course they're still valid. The purpose of the pattern (Repository) is to remove direct dependency on any particular persistence implementation from the application. Thus, it's literally impossible for any particular implementation to serve that purpose. If you want to hard-code your application to a particular data access tool or technology, go ahead, but you won't get the benefits a repository offers. For many applications, that may be fine, but that's a question of that application's requirements, not of whether the pattern or a particular ORM are useful in general.

More on this here:

<http://www.weeklydevtips.co...>

<http://www.weeklydevtips.co...>

<http://www.weeklydevtips.co...>

1 ^ | v • Reply • Share ›



Connie DeCinko → ardalis • 2 years ago

Wow, your post talks about the exact article I came across saying we don't need repository with EF Core. Gotta go read your other related posts. What about using a service? I have a background in WebForms and old MVC and I want my future greenfield projects to use modern methods.

^ | v • Reply • Share ›



ardalis Mod → Connie DeCinko • 2 years ago

A service is just a class that does some operation. I use services for many different things. They're all over the place in modern, clean architecture-style applications. Whether or not a particular operation makes sense to have in a service versus somewhere else (on an entity, in a controller, etc.) is a design decision, but certainly it would be silly to say "services are bad" as a general statement.

4 ^ | v • Reply • Share ›



Dave Brock • 2 years ago

This is an interesting concept that will definitely help me to flatten my project structure a little bit. Do you

find this to be a better approach (from a maintenance perspective) over something like a domain model project that stores all the models?

^ | v • Reply • Share ›



ardalis Mod → Dave Brock • 2 years ago

This isn't either/or with domain models - domain models should be separate from API models and I would keep them in a separate project. You don't want to expose your domain/data model over your API - instead you want to have separate DTOs as your API "wire format". These are the models that can be placed in the same folder as the controller using this approach, as they should be tightly coupled to that controller, but not to much else in the project/solution. Does that make sense?

2 ^ | v • Reply • Share ›



Dave Brock → ardalis • 2 years ago

Ah, yes. Of course. Thanks for clarifying!

^ | v • Reply • Share ›



Ken Bonny • 2 years ago • edited

It's quite funny that I wanted to make a project about using feature folders myself, until I found Ode To Code's feature folders (<https://github.com/OdeToCode...>) I'm now in the process of adding areas to that nuget package.

I'll have to check out how easy it is to add that structure to an API project. I like having my models close to my controllers.

I like your pages approach, but very easy to abuse.

^ | v • Reply • Share ›



ardalis Mod → Ken Bonny • 2 years ago

What's "easy to abuse" with pages? I'm assuming you mean Razor Pages, which I didn't discuss in this post so perhaps you're talking about my MSDN article?

^ | v 1 • Reply • Share ›