

10 Things I Do In Every .NET App

Audience

- .NET Developers
- Interested in more maintainable apps
- Hopefully that Venn Diagram is pretty much just a circle

Agenda

- Series of lightning talks
- Better folder structure
- Treat Warnings As Errors
- Logging “Best Practices”
- Global Authorize Attribute via FallbackPolicy
- Remove Server Header
- Don’t use IOption... use this one weird trick instead... 🤪
- Version endpoint
- Code Smells
- HTTP Security Headers
- Build Once, Deploy Many Times

Goals

- Exposure to new ideas
- Takeaway some ideas back to work

Who am I?

- Director of Engineering at Lean TECHniques
- Co-organizer of [Iowa .NET User Group](#)
- Microsoft MVP
- [Friend of Redgate](#)
- Blog at scottsauer.com



Folder Structure



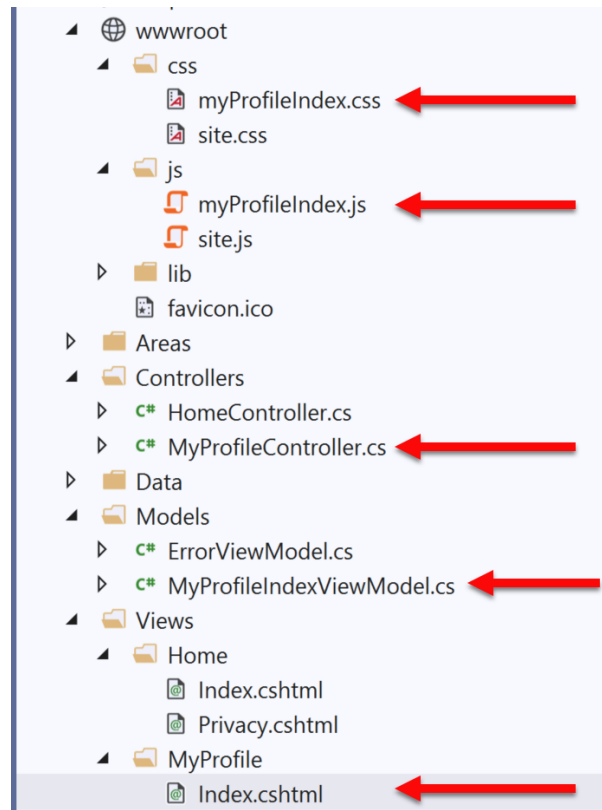
Problem: OOB MVC Folders By Responsibility

- All of these live in their own separate folders and most are required to add a new feature
 - Controllers
 - Views
 - Models
 - wwwroot/css
 - wwwroot/js
- Adds navigation friction
- Scope of a feature is scattered
- Makes it hard to add, delete or extend existing features

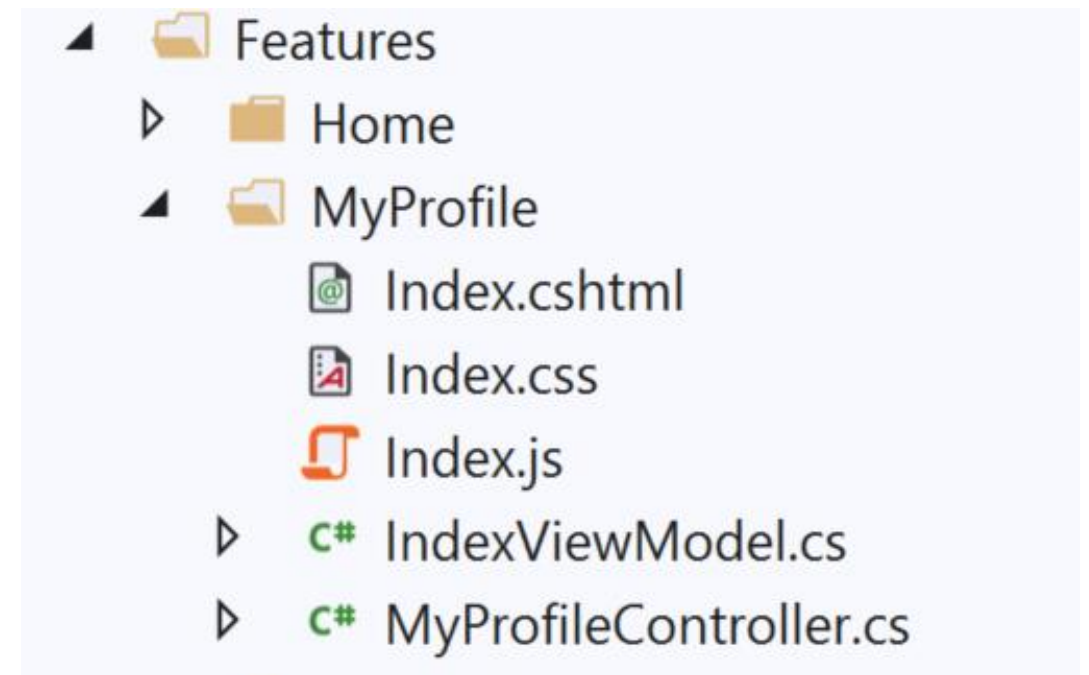
Solution: Use Feature Folders

- Grouping by Feature, not by Responsibility, results in easier maintenance
- Related things remain together (High Cohesion)

MVC out of the box:

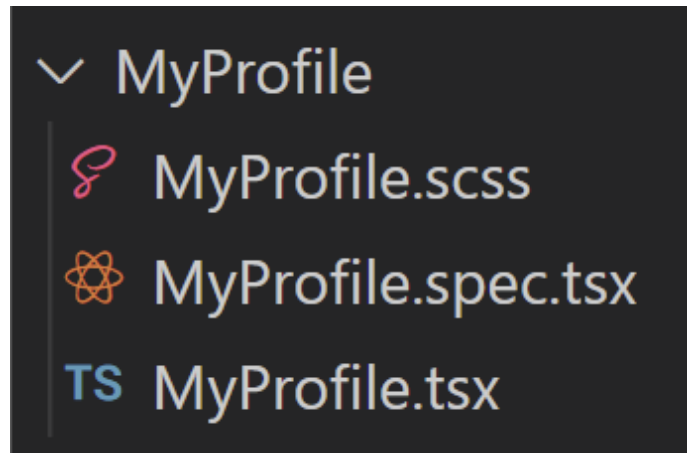


Feature Folders:

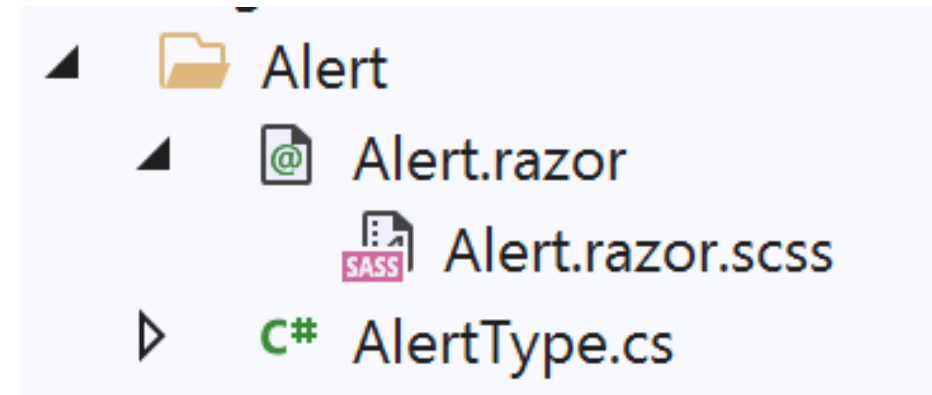


Solution: Use Feature Folders

React:



Blazor:



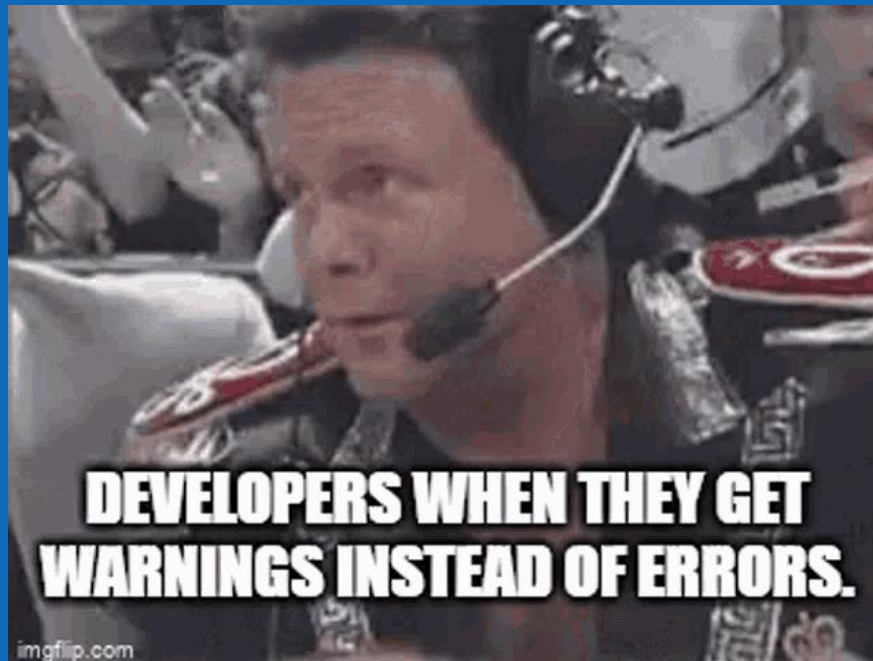
But mah layers!!!1one

- I used to do horizontal csproj's
- .Data, .Business, .Common, .Models, .API
- Now I just do two projects
- .Core and .API (or .Web or .Console or whatever entrypoint is)
- I slice the folders here vertically too

Feature Folder Extra Resources

- Soap analogy
- How to do this in ASP.NET Core
 - [My blog post](#)
 - [Steve Smith's Blog on Feature Folders vs. Areas](#)
- Refactoring to Vertical Slice architecture (featureFolders++)
 - [Derek Comartin](#)


Questions?



Treat Warnings as Errors

- Build/Compiler warnings don't exist in my world
 - Error or Nothing

```
<PropertyGroup>  
  <TargetFramework>net6.0</TargetFramework>  
  <Nullable>enable</Nullable>  
  <TreatWarningsAsErrors>true</TreatWarningsAsErrors>  
</PropertyGroup>
```



Questions?

Logging “Best Practices”



Logging “Best Practices”

- Use Serilog as logging framework
- Use ILogger everywhere, not Serilog directly
- Use structured logging, not concatenation
- Each log should have key bits of information on it
 - User, Correlation ID, Request URL, App Version, etc.
- “We need to log that”
- Logs vs Metrics vs Audits

Logs

- Developer focused
- Example: log an exception or log response from external API
- Log Levels
 - Debug vs Information vs Warning vs Error vs Critical
- How long does my log store keep logs?
- How reliable is my log delivery system
 - It's okay to not have 100% guaranteed delivery of logs
- How does Serilog work?

Metrics

- Two types
- Application
 - CPU, Network, Response Times, Queue Depth, etc.
- Business
 - How many times did someone click that button
- How long do we need to keep Metric data?
- It may or may not be acceptable to miss some data
- What data store are we using?

Audits

- Recording who, did what, and when in your application
- Usually for legal, compliance, or traceability reasons
- Losing any data is unacceptable
- Store audits with the same data store as the data that's being audited

Questions?

Global Authorize Attribute via FallbackPolicy



Problem: Security is Opt-In

- You have to remember to add a [Authorize] attribute everywhere
- Or you have to remember to inherit from a custom BaseController
- You forget? Oops you're wide open to the world!

Solution: FallbackPolicy

- A Fallback Policy is the policy that gets evaluated if no other policy is specified

```
builder.Services.AddAuthorization(options =>
{
    options.FallbackPolicy = new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser() // AuthorizationPolicyBuilder
        .Build(); // AuthorizationPolicy
});
```


Questions?

Remove the Server Header



Remove the Server Header

- By default ASP.NET Core adds a “Server Header”
- This says “Kestrel”
- This exposes to black hats what you’re running
- Focuses exploiting known CVEs

```
builder.WebHost.UseKestrel(options => options.AddServerHeader = false);
```

Questions?

Don't use IOption...




Problem: IOptions is annoying

- Dependency on Microsoft.Extensions.Options down in other csproj's
- Testing IOptions is slightly annoying with Options.Create()
- .Value everywhere adds friction

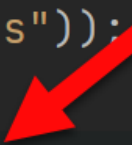
```
public AppSettings AppSettings { get; }

public IndexModel(IOptions<AppSettings> appSettings)
{
    AppSettings = appSettings.Value;
}
```



Solution: Register your Options class directly

```
services.Configure<AppSettings>(Configuration.GetSection(key: "AppSettings"));  
services.AddSingleton<IOptions<AppSettings>>(() =>  
    registeredServices.GetRequiredService<IOptions<AppSettings>>().Value);
```



```
public AppSettings AppSettings { get; }  
  
public IndexModel(AppSettings appSettings)  
{  
    AppSettings = appSettings;  
}
```

Questions?

Version Endpoint



Version Endpoint

- Versioning the API, right? Nope!
- What version of your app are you running?
- /api/version
- Format:
 - <Date>.<BuildNumber>.<ShortGitSha>
 - 20230617.1000.abc1234
- Generate version in CI and store in a file read by endpoint, why a file?
- Version goes on every log
- Bonus: notify SPA of new deployment for reload purposes

Questions?

Code Smells



Structuring a method

- Happy Path always at the bottom of the method
 - Don't want to scan for “what happens when all goes well” and find it in the middle of a method
- Use return's instead of nested if => else

Razor Pages Template Code:

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (ModelState.IsValid)
    {
        var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
        var result = await _userManager.CreateAsync(user, Input.Password);
        if (result.Succeeded)
        {
            _logger.LogInformation("User created a new account with password.");

            var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
            var callbackUrl = Url.Page("/Account/ConfirmEmail", null, new { userId = user.Id, code = code }, Request.Scheme);

            await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
                $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

            await _signInManager.SignInAsync(user, isPersistent: false);
            return LocalRedirect(returnUrl);
        }
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }
    }

    // If we got this far, something failed, redisplay form
    return Page();
}
```

Refactored with Happy Path At The Bottom:

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (!ModelState.IsValid)
        return Page();

    var user = new IdentityUser { Username = Input.Email, Email = Input.Email };
    var result = await _userManager.CreateAsync(user, Input.Password);

    if (!result.Succeeded)
    {
        foreach (var error in result.Errors)
        {
            ModelState.AddModelError(string.Empty, error.Description);
        }

        return Page();
    }

    _logger.LogInformation("User created a new account with password.");

    var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
    var callbackUrl = Url.Page("/Account/ConfirmEmail", null, new { userId = user.Id, code = code }, Request.Scheme);

    await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
        $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

    await _signInManager.SignInAsync(user, isPersistent: false);
    return LocalRedirect(returnUrl);
}
```

The Indentation Proclamation

- The more indented your code is, the harder it is to follow
- Nested if's, nested loops, etc.
- ...I made this up

Code smells

- Not hard and fast rules, just my “warning light”
- Methods > 20 lines
- Classes > 200 lines
- Regions
 - You probably should’ve added a new class or method instead



Questions?

HTTP Security Headers



HTTP Security Headers

- Tells a browser what extra rules to enforce
- Protects against MITM, clickjacking, cross-site scripting, and more
- Be careful!
- `NetEscapades.AspNetCore.SecurityHeaders`

Resources

- My talk deep diving on this topic:
<https://www.youtube.com/watch?v=7MWXTXjtl8s>

Questions?

Build Once, Deploy Many Times



Build Once, Deploy Many Times

- Discourages Long Running Branches and GitFlow
- Encourages Trunk Based Development (even with PRs)
- Encourages Continuous Integration (the practice)
- Easier testing
- Environmental differences (i.e. config, secrets) live in the environment

Questions?

Bonus!

- CI/CD Pipelines
- Write tests
- Work in small batches
- Deploy frequently (should be daily or more frequent)
- Don't put your Infrastructure as Code in another repo

Bonus!

- CI/CD Pipelines
- Write tests
- Work in small batches
- Deploy frequently (should be daily or more frequent)
- Don't put your Infrastructure as Code in another repo

Real benefits of these practices

- 1 web app went from deploying to Prod 12x a year to 1000x a year
- Faster, more reliable delivery of value to users

Questions?

Contact: ssauber@leantechniques.com



Slides at scottsauber.com

Thanks!