

10 Things I Do On 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
- Use FluentValidation
- Remove Server Header
- Don’t use IOptions... use this one weird trick instead... 🧐
- Code Smells
- HTTP Security Headers
- #11, #12, #13
- Rapid fire Bonus!

Goals

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

Who am I?

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





Guillermo Rauch ✓

@rauchg



Every system tends towards complexity, slowness and difficulty

Staying simple, fast and easy-to-use is a battle that must be fought everyday

5:39 PM · Dec 26, 2016 from San Diego, CA · Twitter Web Client

642 Retweets **26** Quote Tweets **1,092** Likes

Folder Structure



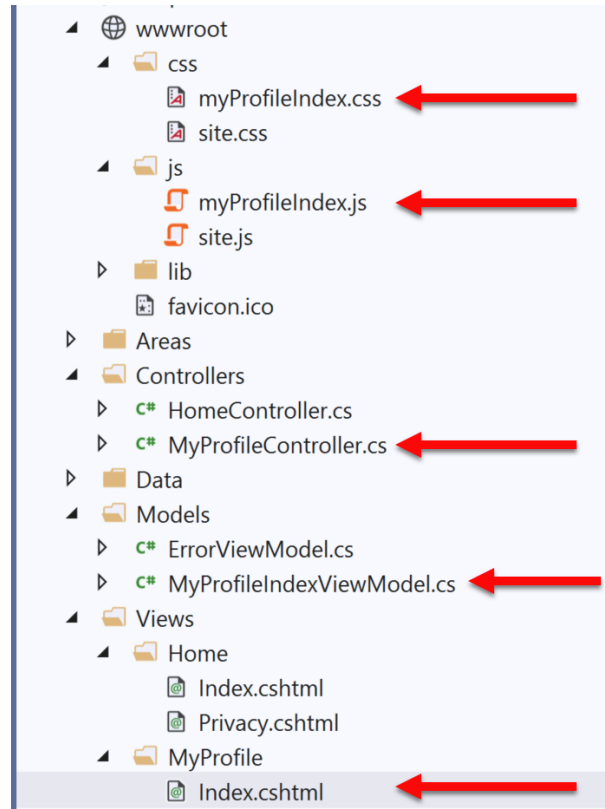
Problem: OOB MVC Folders By Responsibility

- 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

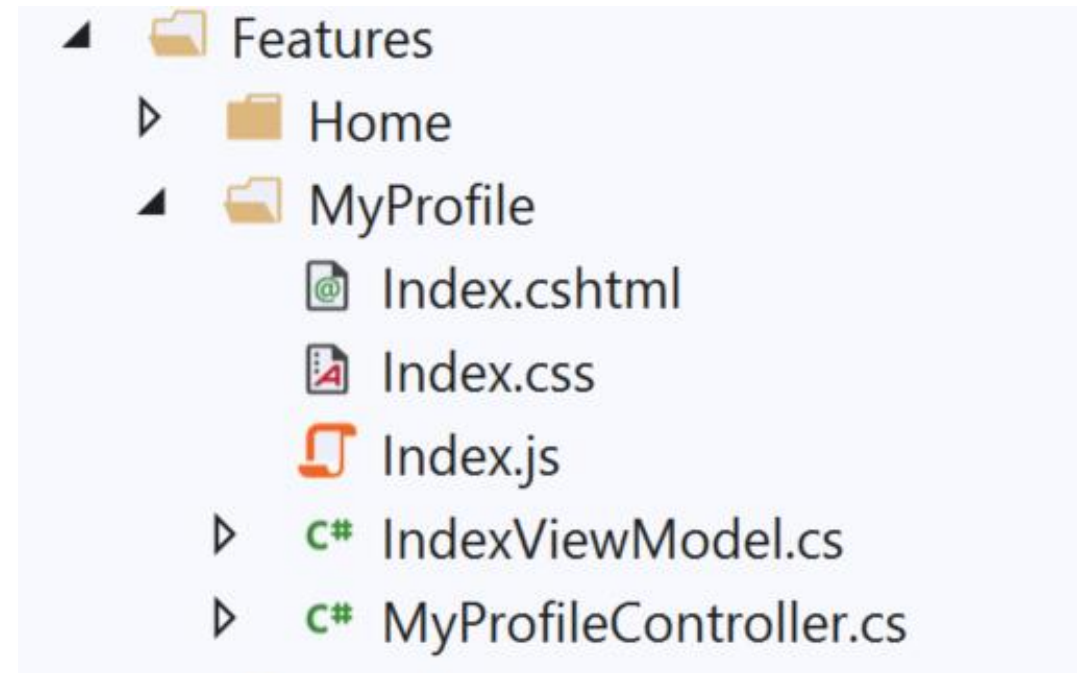
Solution: Use Feature Folders

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

MVC out of the box:

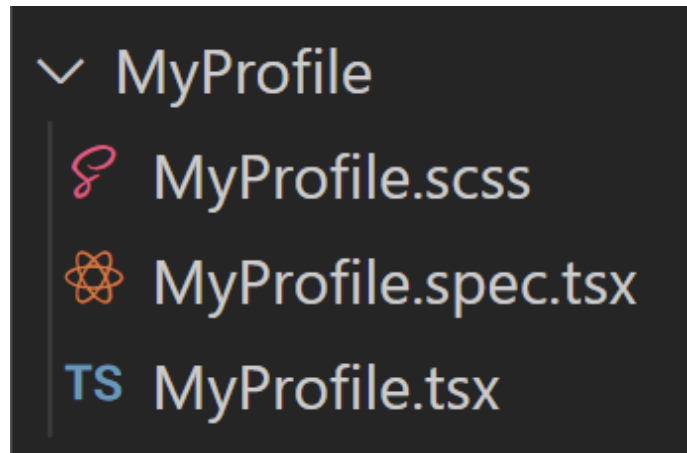


Feature Folders:

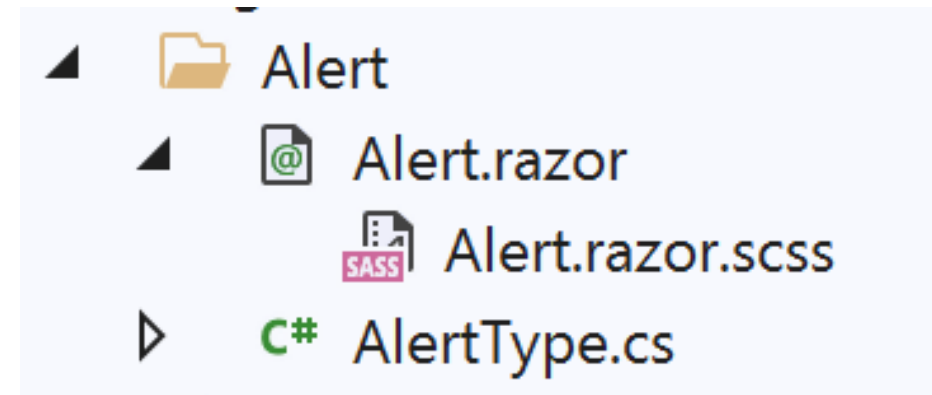


Solution: Use Feature Folders

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
- Slice horizontally only if you publishing packages to NuGet or internally

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](#)



Treat Warnings as Errors

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

```
<PropertyGroup>  
  <TargetFramework>net8.0</TargetFramework>  
  <Nullable>enable</Nullable>  
  <ImplicitUsings>enable</ImplicitUsings>  
  <TreatWarningsAsErrors>true</TreatWarningsAsErrors>  
</PropertyGroup>
```



Logging “Best Practices”



Logging “Best Practices”

- Use Serilog as logging framework
- Use ILogger everywhere, not Serilog directly
- Use structured logging, not concatenation
 - But... Azure Log Analytics workspaces cap at 500 columns
- Each log should have key bits of information on it
 - User ID, 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
- Stop abusing Information! You probably meant Debug
- 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

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
});
```

Validation



Validation – What's wrong with Data Annotations

- Only work well for simple scenarios
- Hard to make custom ones
- Hard to unit test
- Separate annotations for each property
 - Can get “tall”
- SRP violated
 - Model + Validation combined into one class

Solution: Use FluentValidation

- Fluent interface
- Business rules are easy to maintain and read
- Easy to show a stakeholder
- Easy to test
- Integrates with ModelState.IsValid
- 372M downloads
- <https://github.com/JeremySkinner/FluentValidation>

Template Code:

```
public class RegisterViewModel
{
    [Required]
    [EmailAddress]
    [Display(Name = "Email")]
    2 references | 0 exceptions
    public string Email { get; set; }

    [Required]
    [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    1 reference | 0 exceptions
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    [Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]
    0 references | 0 exceptions
    public string ConfirmPassword { get; set; }
}
```

Refactored with Fluent Validation:

```
public class RegisterViewModel
{
    [Display(Name = "Email")]
    4 references | 0 exceptions
    public string Email { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    5 references | 0 exceptions
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Display(Name = "Confirm password")]
    1 reference | 0 exceptions
    public string ConfirmPassword { get; set; }
}
```

```
public class RegisterViewModelValidator : AbstractValidator<RegisterViewModel>
{
    0 references | 0 exceptions
    public RegisterViewModelValidator()
    {
        RuleFor(m => m.Email).NotEmpty()
            .WithMessage("Email is required.");

        RuleFor(m => m.Email).EmailAddress()
            .WithMessage("Email must be a valid email address.");

        RuleFor(m => m.Password).NotEmpty()
            .WithMessage("Password is required.");

        RuleFor(m => m.Password).MaxLength(100)
            .WithMessage("The password cannot be longer than 100 characters.");

        RuleFor(m => m.Password).MinLength(6)
            .WithMessage("The password must be at least 6 characters long");

        RuleFor(m => m.ConfirmPassword).Equal(m => m.Password)
            .WithMessage("The password and confirmation password do not match.");
    }
}
```

A Rule that only exists if....

```
public class InsuranceEnrollmentValidator : AbstractValidator<InsuranceEnrollment>
{
    0 references
    public InsuranceEnrollmentValidator()
    {
        RuleFor(model => model.Age)
            .Must(age => age < 26)
            .When(model => model.IsDependent)
            .WithMessage("A dependent must be younger than 26.");
    }
}
```

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);
```

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

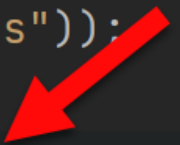
```
public AppSettings AppSettings { get; }

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;  
}
```


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



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>

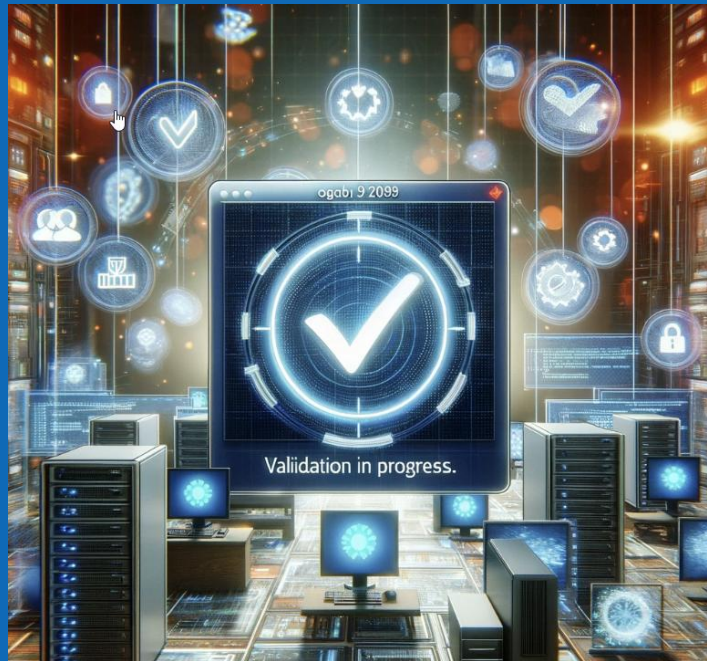
Build Once, Deploy Many Times



Build Once, Deploy Many Times

- Discourages Long Running Branches and GitFlow
- Encourages Trunk Based Development (even with PRs)
- Atlassian calls Gitflow “a legacy workflow”
- Encourages Continuous Integration (the practice)
- Environmental differences (i.e. config, secrets) live in the environment

ValidateOnBuild



ValidateOnBuild

- Singletons can only depend on Singletons
- The “captive dependency” problem
- ASP.NET Core will catch this when running in Development but not other environments
- If you have a custom local env name...

ValidateOnBuild

```
builder.Services.AddSingleton<SingletonThing>();  
builder.Services.AddScoped<ScopedThing>();
```

```
public class SingletonThing(ScopedThing scopedThing)
```

```
Unhandled exception. System.AggregateException: Some services are not able to be constructed (Error while  
validating the service descriptor 'ServiceType: CaptiveDependencyProblem.SingletonThing Lifetime: Singleton  
ImplementationType: CaptiveDependencyProblem.SingletonThing': Cannot consume scoped service 'CaptiveDep  
ndencyProblem.ScopedThing' from singleton 'CaptiveDependencyProblem.SingletonThing'.)  
    at Microsoft.Extensions.DependencyInjection.ServiceProvider..ctor(ICollection`1 serviceDescriptors, Se  
viceProviderOptions options)
```

ValidateOnBuild

```
builder.Host.UseDefaultServiceProvider(config:ServiceProviderOptions =>
{
    config.ValidateOnBuild = true;
});
```

Automated Tests



Automated Testing with xUnit

- You should be writing automated tests
 - Exposes holes in your architecture
 - Proven to be faster long-term
 - Make changes quickly and confidently because have a regression test suite
- Use xUnit or NUnit
 - Just not MSTest which is wayyy more verbose and has less features
- xUnit used by ASP.NET team

Chekhov's Gun



“Remove everything that has no relevance to the story. If you say in the first chapter that there is a rifle hanging on the wall, in the second or third chapter it absolutely must go off. If it's not going to be fired, it shouldn't be hanging there.”

Anton Chekhov



Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    var customer = new Customer
    {
        FirstName = "SpongeBob",
        LastName = "",
        Address = "123 Pineapple",
        BirthDate = new DateOnly(year: 1999, month: 5, day: 1),
    };

    var result = new CustomerValidator().Validate(customer);

    result.Errors.Should().Contain(error: ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    var customer = CreateValidCustomer();
    customer.LastName = "";

    var result = new CustomerValidator().Validate(customer);

    result.Errors.Should().Contain(error : ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    _customer.LastName = "";

    var result = new CustomerValidator().Validate(_customer);

    result.Errors.Should().Contain(error:ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    _customer.LastName = "";

    var result = _customerValidator.Validate(_customer);

    result.Errors.Should().Contain(error:ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

Chekhov's Gun Applied to Testing

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    var customer = new Customer
    {
        FirstName = "SpongeBob",
        LastName = "",
        Address = "123 Pineapple",
        BirthDate = new DateOnly(year: 1999, month: 5, day: 1),
    };

    var result = new CustomerValidator().Validate(customer);

    result.Errors.Should().Contain(error: ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```

```
[Fact]
public void ValidateShouldReturnErrorWhenLastNameIsEmpty()
{
    _customer.LastName = "";

    var result = _customerValidator.Validate(_customer);

    result.Errors.Should().Contain(error: ValidationFailure => error.ErrorMessage == "Last Name is required.");
}
```


Central Package Management



Problem

- Same package in multiple projects across your solution
- Annoying to keep version in sync
- Tests projects especially
- I find most people don't know Central Package Management exists

Solution: Central Package Management

- Added in .NET 6
- Create Directory.Packages.props at the root and define packages and versions there
- Omit Version attribute in PackageReference
- All packages will have to omit the Version attribute
- Can override with VersionOverride attribute

Solution: Central Package Management

```
1      <!-- In Directory.Packages.props 🙌🙌 -->
2      <Project>
3      <PropertyGroup>
4          <ManagePackageVersionsCentrally>true</ManagePackageVersionsCentrally>
5      </PropertyGroup>
6      <ItemGroup>
7          <PackageVersion Include="xunit" Version="2.6.6" />
8      </ItemGroup>
9  </Project>
10
```

```
<!-- In consuming csproj's 🙌🙌 -->
<ItemGroup>
    <PackageReference Include="xunit"/>
</ItemGroup>
```

BONUS



Ever wonder what SQL Queries EF is making?

- Switch log level of `Microsoft.EntityFrameworkCore.Database.Command` to Information
- Only do this for local development

Ever wonder what SQL Queries EF is making?

```
{
  "Logging": {
    "LogLevel": {
      // others omitted
      "Microsoft.EntityFrameworkCore.Database.Command": "Information"
    }
  }
}
```

```
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (12ms) [Parameters=[@p0='?' (Size = 36), @p1='?' (DbType = Int32), @p2='?' (Size = 36),
      @p3='?' (Size = 18), @p4='?' (DbType = Boolean), @p5='?' (DbType = Boolean), @p6='?' (DbType = DateTimeOffset), @
      p7='?' (Size = 18), @p8='?' (Size = 18), @p9='?' (Size = 84), @p10='?', @p11='?' (DbType = Boolean), @p12='?' (Si
      ze = 32), @p13='?' (DbType = Boolean), @p14='?' (Size = 18)], CommandType='Text', CommandTimeout='30']
      INSERT INTO "AspNetUsers" ("Id", "AccessFailedCount", "ConcurrencyStamp", "Email", "EmailConfirmed", "Locko
      utEnabled", "LockoutEnd", "NormalizedEmail", "NormalizedUserName", "PasswordHash", "PhoneNumber", "PhoneNumberCon
      firmed", "SecurityStamp", "TwoFactorEnabled", "UserName")
      VALUES (@p0, @p1, @p2, @p3, @p4, @p5, @p6, @p7, @p8, @p9, @p10, @p11, @p12, @p13, @p14);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[@__normalizedEmail_0='?' (Size = 18)], CommandType='Text', CommandTim
      eout='30']
      SELECT "a"."Id", "a"."AccessFailedCount", "a"."ConcurrencyStamp", "a"."Email", "a"."EmailConfirmed", "a"."L
      ockoutEnabled", "a"."LockoutEnd", "a"."NormalizedEmail", "a"."NormalizedUserName", "a"."PasswordHash", "a"."Phone
      Number", "a"."PhoneNumberConfirmed", "a"."SecurityStamp", "a"."TwoFactorEnabled", "a"."UserName"
      FROM "AspNetUsers" AS "a"
      WHERE "a"."NormalizedEmail" = @__normalizedEmail_0
      LIMIT 2
```

More Bonus!

- CI/CD Pipelines
- Confident Green
- Feature Toggles
- Work in small batches
- Deploy frequently (should be daily or more frequent)
- Don't split your frontend and your BFF in another repo
- Don't put your Infrastructure as Code in another repo

Real benefits of these practices

- 1 company went from deploying to Prod 12x a year (rolling back half of those) to 2000x a year (rolling back <1% of those)
- Faster, more reliable delivery of value to users

Questions?

ssauber@leantechniques.com

Thanks!