Building Large, Yet Maintainable, ASP.NET Core Apps

Scott Sauber

scottsauber

Slides up at scottsauber.com

Audience

- Existing ASP.NET/ASP.NET Core Developers
- Building “Large” applications
  - Large to me, means a combination of these:
    - Many Controllers and/or Views
    - Lots of business logic
    - High business impact
    - Frequent Change
  - Large != High Traffic
- You may be:
  - Server-rendered HTML via MVC or Razor Pages
  - API’s only via MVC
  - Building backend logic

Audience Goals

- “New” perspective on a few topics
- You’re not going to agree on everything
- Take away some ideas to evaluate in your future workflow tomorrow

Who am I?

- Software Consultant at Lean TECHniques
- .NET Foundation Action Groups
- IADNUG co-organizer
- Blog (primarily ASP.NET Core) at scottsauber.com

- Background in Mortgage, Health Insurance, Ag Industry
  - Users from 1 – 250K
- Highly Regulated
  - Lots of validation
  - Lots of changes

Agenda

- Lightning Talk approach
- Talk about best practices for all these *in a large app*
  - Folder Structure
  - UI Components
  - Code Flow
  - Validation
  - ORM’s
  - Dependency Injection
  - Unit Testing and Assertions
  - DevOps
  - Feature Toggles
  - Microservices

Overall theme

- Improve Maintainability
  - Better practices
  - Consistency
  - Enjoyable to work with
- Create systems that allow developers to make changes as easy, quick, and bug-free as possible, while allowing that trend to continue into the future.

- “Legacy software is software you have no confidence in.”

Assumptions

- I’m assuming you have a traditional LOB app without crazy volume
- The answer is always “It Depends” but I’m going to throw that out the window and give some real guidance.

Guillermo Rauch
@rauchg

Follow

Every system tends towards complexity, slowness and difficulty
Staying simple, fast and easy-to-use is a battle that must be fought everyday

RETWEETS	LIKES
613	1,005

5:39 PM - 26 Dec 2016 from San Diego, CA

16 613 1.0K

Michael Feathers
@mfeathers

Follow

Make something simple and add complexity reluctantly.

9:08 PM - 6 Jul 2017

86 Retweets **137** Likes

3 86 137

Folder Structure

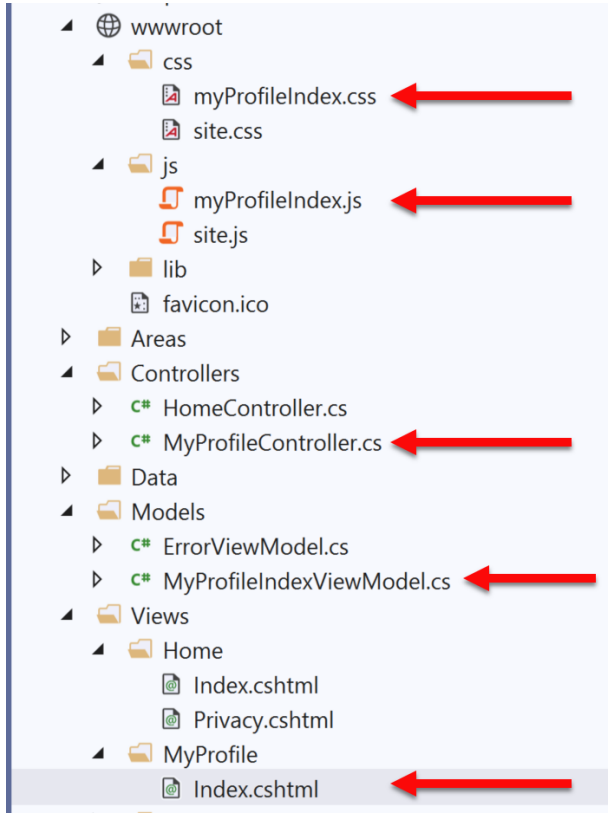
Problem: OOB MVC Folders By Responsibility

- 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

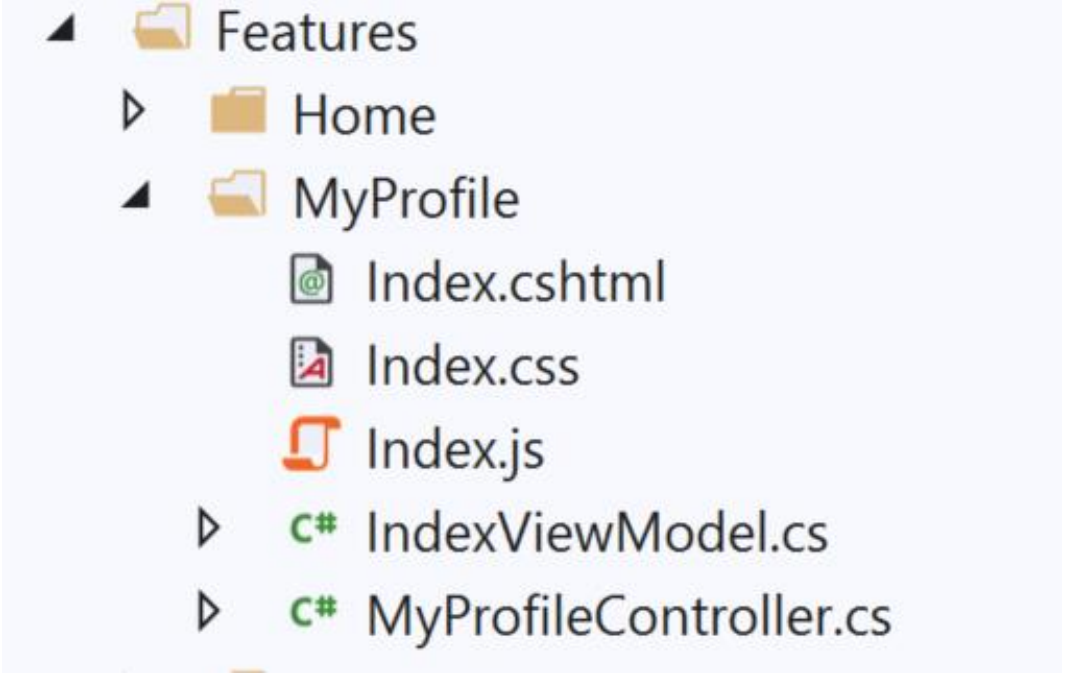
Solution: Use Feature Folders

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

MVC out of the box:



Feature Folders:

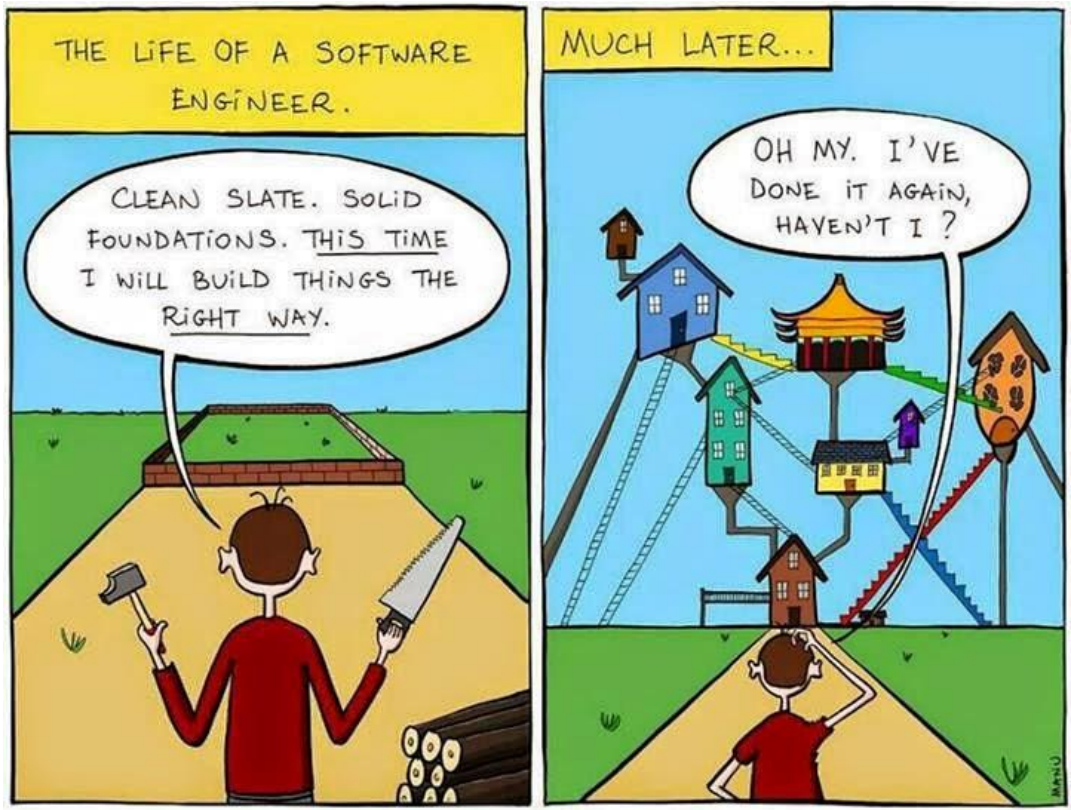


Feature Folder Extra Resources

- How to do this in ASP.NET Core
  - My blog post
  - Steve Smith's Blog on Feature Folders vs. Areas
- How to do this in ASP.NET 4.x
  - Tim Thomas' blog post
- Also used in React, Angular, etc.
  - John Papa

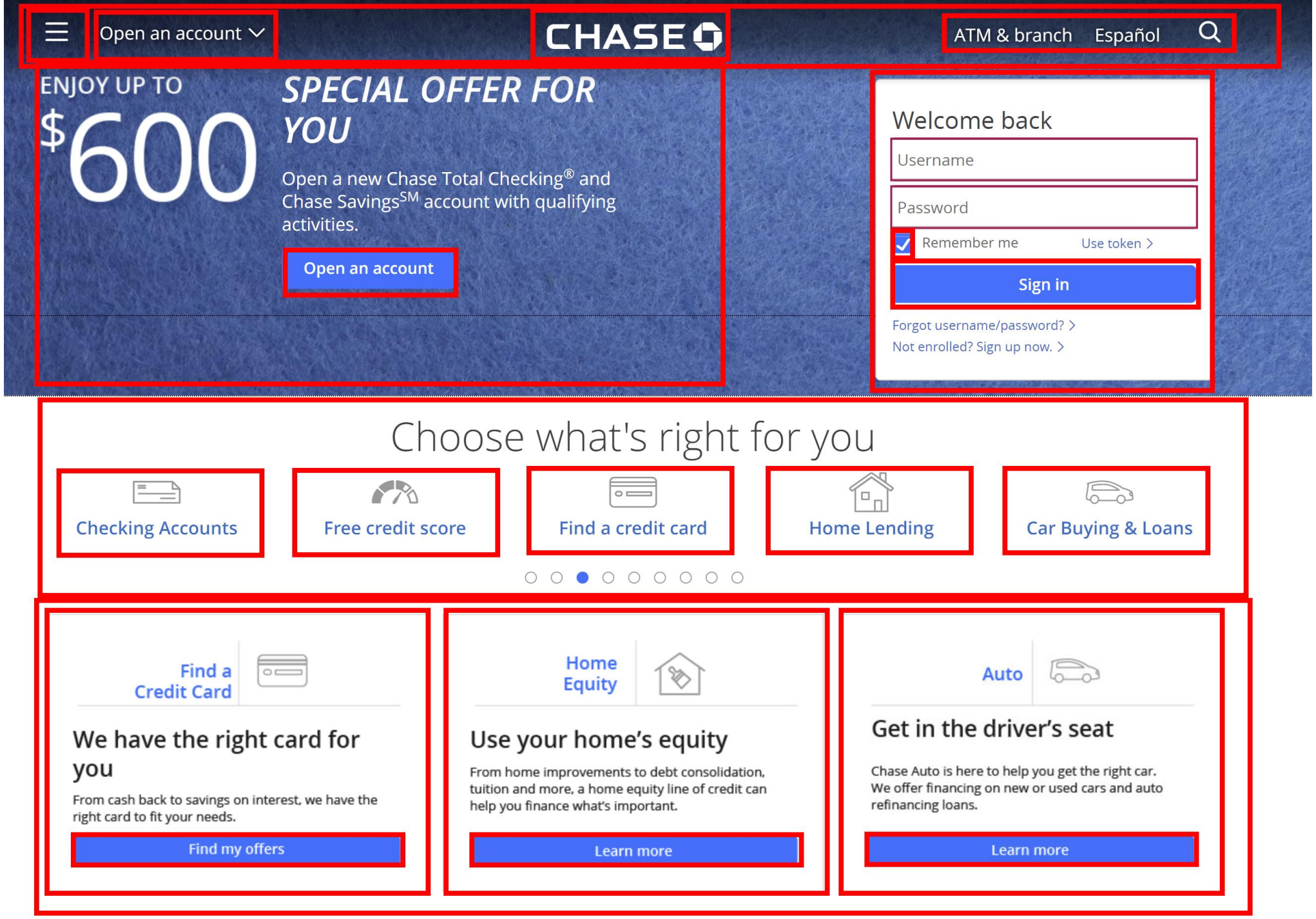
Questions on Feature Folders?

UI Composition



Make as many UI Components as you can

- Benefits
  - SRP for the UI
  - Easier to reason about
  - Reusability
  - Eliminates div soup
- Component architecture is here to stay
  - React, Vue, Angular, AngularJS 1.5+, Knockout 3.2+, Blazor
- Partials
- View Components
- Blazor Components
- Child Actions (in ASP.NET 4)


Open an account
CHASE
ATM & branch
Español
ENJOY UP TO
$600
SPECIAL OFFER FOR YOU
Open a new Chase Total Checking® and Chase Savings℠ account with qualifying activities.
Open an account
Welcome back
Username
Password
Remember me
Use token
Sign in
Forgot username/password?
Not enrolled? Sign up now.
Choose what's right for you
Checking Accounts
Free credit score
Find a credit card
Home Lending
Car Buying & Loans
Find a Credit Card
We have the right card for you
From cash back to savings on interest, we have the right card to fit your needs.
Find my offers
Home Equity
Use your home's equity
From home improvements to debt consolidation, tuition and more, a home equity line of credit can help you finance what's important.
Learn more
Auto
Get in the driver's seat
Chase Auto is here to help you get the right car. We offer financing on new or used cars and auto refinancing loans.
Learn more

Questions on Components?

Code Flow/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:

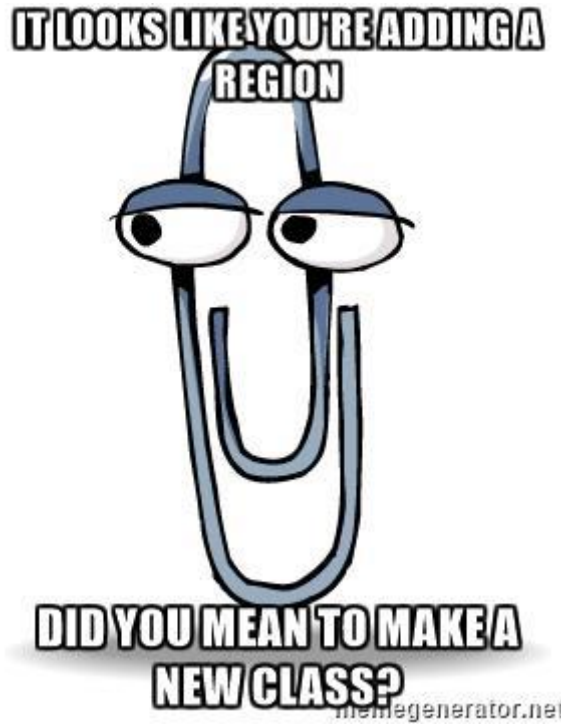
```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{
    returnUrl = returnUrl ?? Url.Content("~/");
    if (ModelState.IsValid)
    {
        var user = new IdentityUser { UserName = Input.Email, Email = Input.Email };
        var result = await _userManager.CreateAsync(user, 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);
}
```

Code smells

- Not hard and fast rules, just my “warning light”
- Methods > 30 lines
- Classes > 200 lines
- Regions
  - You probably should’ve added a new class or method instead



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

Questions on Code Flow/Smells?

Validation

Validation – What's wrong with OOB Options

- 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
- Writing own Custom Validation classes
  - Lose client-side hooks Data Annotations provides

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
- Same Client-Side validation as Data Annotations
- 20.2M 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).MaximumLength(100)
            .WithMessage("The password cannot be longer than 100 characters.");

        RuleFor(m => m.Password).MinimumLength(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.");
    }
}
```

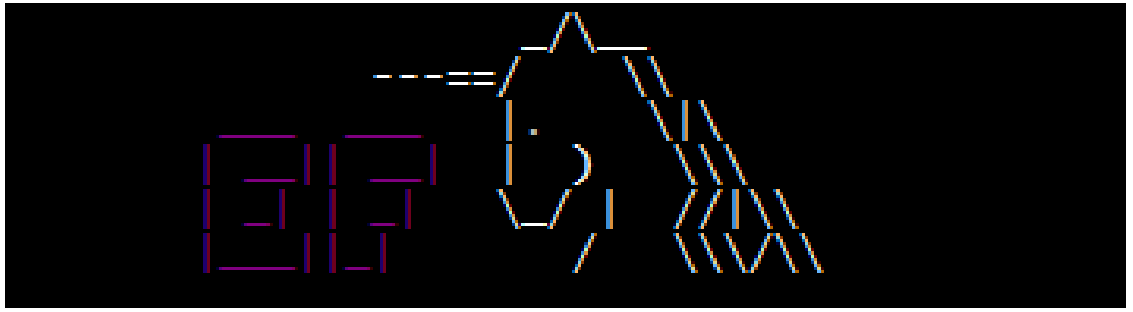
Questions on Fluent Validation?

ORM’s

Don’t use raw ADO - Use an ORM

Entity Framework – Full ORM

- Pros
  - Developer Productivity
  - Compile-time safety with LINQ Queries
  - Extremely quick to add new CRUD operations
  - Built in Unit of Work
  - Migration support
- Cons
  - Less performant
  - Less control over the queries generated
  - Heavier



Dapper – Micro ORM

- Pros
  - Performance near ADO
  - More control over the queries
  - Extremely simple to setup
  - Stack Overflow beta tests
- Cons
  - SQL strings = Big column name refactorings are harder
  - Less features than EF

ORM usage comparison

Entity Framework:

```
public List<Customer> GetCustomersByState(string state)
{
    var context = new ApplicationDbContext();
    return context.Customers.Where(c => c.State == state).ToList();
}
```

Dapper:

```
public List<Customer> GetCustomersByState (string state)
{
    var connection = new SqlConnection();

    string sql = @"SELECT *
                   FROM Customers
                   WHERE State = @State";

    var parameters = new
    {
        State = state
    };

    return connection.Query<Customer>(sql, parameters).ToList();
}
```

Other ORM things

- Both manage connection lifetimes
- Both give you SQL Injection protection
- I use both depending on the job
- Common for us to start with EF and supplement with Dapper
  - EF for Developer Productivity
  - Dapper for hot paths
  - Dapper for queries that are tough to write in LINQ
- DON’T WRITE YOUR OWN ORM

Questions on ORM’s?

Dependency Injection

Dependency Injection and IoC containers

- DI/IoC helps you loosely couple your “large” apps
- Lets you unit test anything if done correctly
- I use Microsoft.Extensions.DependencyInjection
  - Scrutor for auto-registration

Common DI Pitfalls

- "New Is Glue"
- "Static Cling"
- DateTime.Now (and variants)
  - Instead inject in IClock or have method take in DateTime? and default to Now
- HttpContext
  - IHttpContextAccessor in ASP.NET Core
- ConfigurationManager in ASP.NET 4.x
  - Instead, inject a POCO Settings class

Questions on Dependency Injection?

Automated Testing

Automated Testing with xUnit

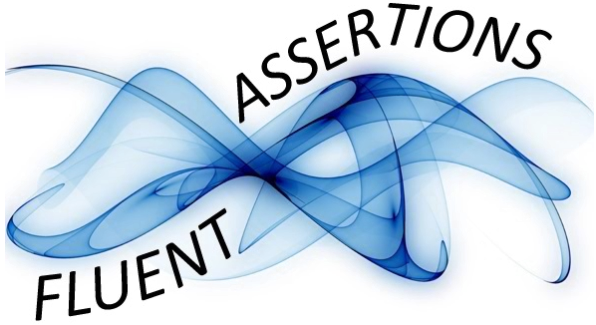
- You should be writing automated tests
  - Exposes holes in your architecture
  - Proven to be faster long-term
    - Make changes quickly and confidently because have a regression test suite
- “Start at the back”
- Use xUnit or NUnit
  - Just not MSTest which is wayyy more verbose and has less features
- xUnit used by ASP.NET team
- I used to use NUnit and switched to xUnit
- NUnit more boilerplate
  - No [TestFixture] in xUnit
  - No [SetUp] method just use a constructor in xUnit

Problem: OOB Assertion Libraries Annoy Me

- Assert.Equal(value, value)
  - Hard to remember that it's Assert.Equal(expected, actual)
  - Can lead to funky looking assertion failures if you flip them
- No "Because" string in xUnit's assertions

Solution: Use FluentAssertions for assertions

- Adds a Should() extension method to object
  - result.Should().Be(0);
  - Easier to read than Assert.Equal(0, result);



- Because string to explain why
  - can12YearOldDriveResult.Should().Be(false, “because you must be 16 years old to drive.”)
- 28.7M downloads

Questions on Automated Testing?

DevOps

The Expert Beginner
@ExpertBeginner1

Follow

The most important thing you can do is have well-defined handoff procedures between Dev and DevOps and between DevOps and Ops.

9:24 AM - 14 Sep 2017

43 Retweets **54** Likes

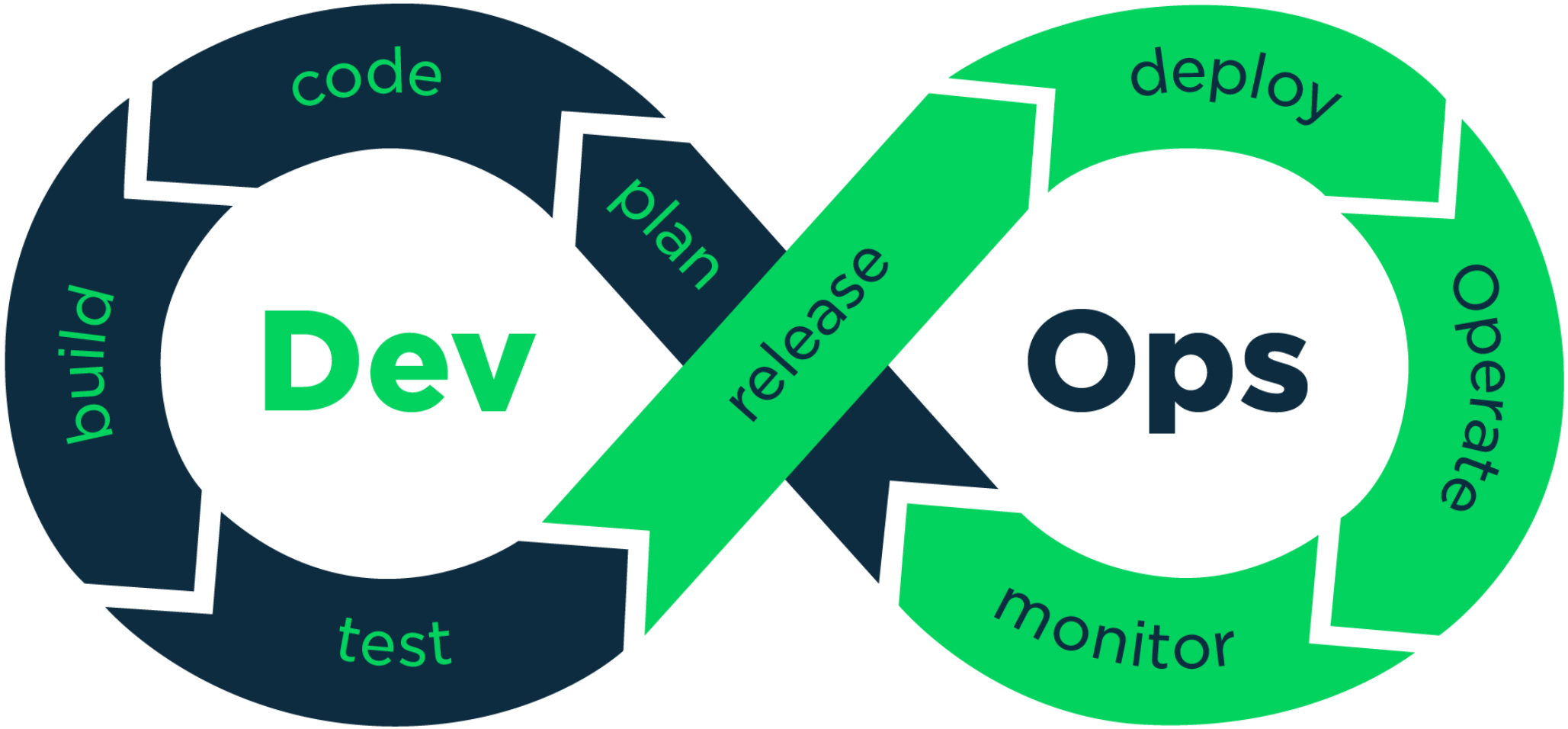
5 43 54

What is DevOps

"DevOps is the union of people, process, and products to enable continuous delivery of value to our end users."

- Donovan Brown

DevOps Pipeline



Continuous Integration and Delivery

- Continuous Integration Server
  - Automated builds running on an independent build server
  - Automatically running automated tests
  - Creates an output
- Continuous Delivery
  - Takes the output from the build server and deploys it
  - Go through different environments
  - Automated configuration of VM, IIS, folder permissions, Azure WebApp, etc.
- These two together will be transformational for you and your company

Continuous Integration with TeamCity

- Free for up to 100 build configs
  - No support or $1999/year with support
- NuGet Restores
- MSBuild
  - Don’t need to install VS on your build server
- Run automated tests
- Produces an output
- Zips up and sends to Octopus Deploy

Continuous Delivery with Octopus Deploy

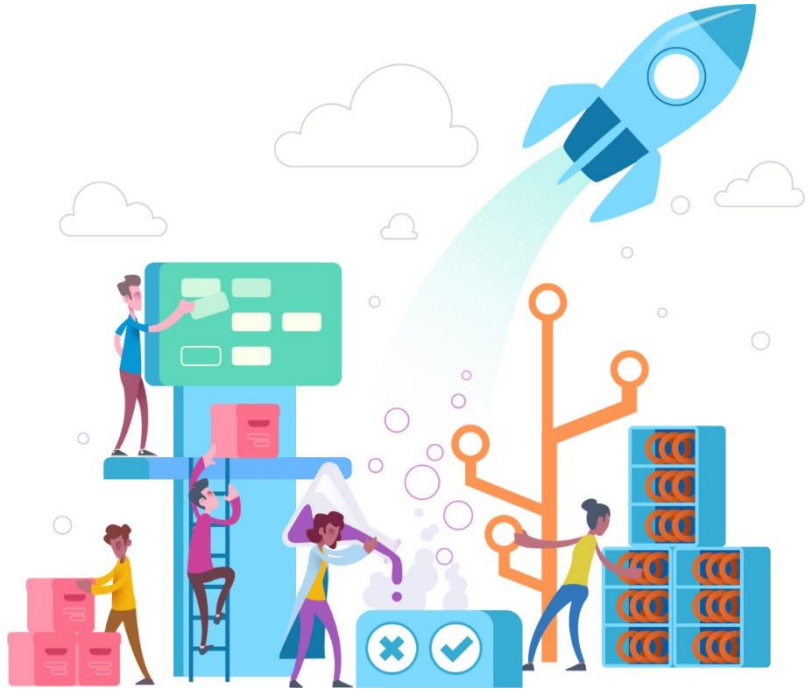
- Lets you promote your package through different environments
- Controls all appsettings/web.config scoped to each environment
- Auto-creates IIS Site and App Pool for you
- Deploys and rollbacks are click of a button
- Lets you run any scripts you want to as part of your deployment
- Easy to install and easy to use
- Professional – 25 deployment targets $2300/yr
- Full Audit Trail
  - Who/What/When changed
- My single favorite piece of software – ever

TeamCity + Octopus Deploy enables…

- Consistency
  - Machine repeats the same steps over and over again
- Consistency promotes Confidence
  - Passes build
  - Passes automated tests
  - Configuration is correct
  - Let’s deploy it to Prod!
- Confidence enables agility
  - Mid-day publishes are no big deal
    - For a single app, our record is 24 publishes to Prod in 1 day with 2 developers

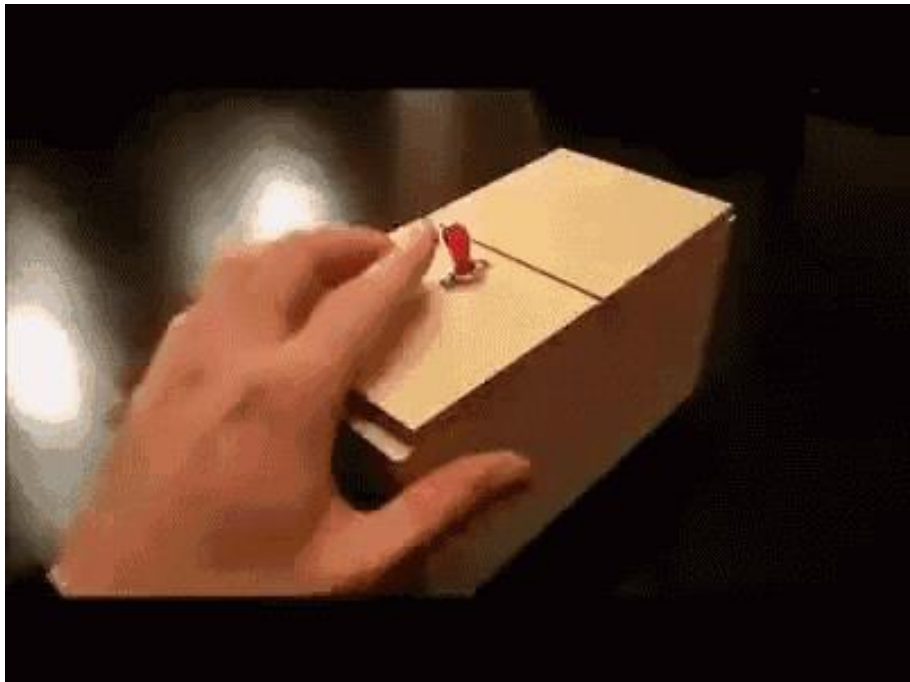
Azure DevOps

- Can handle Source Control, Builds, Deploys
- Octopus is still more full featured
- But AzDO gives you 1 product
- Also can do work item tracking



Questions on DevOps?

Feature Toggles



Feature Toggles

- No long lived branches
  - No environment branches
- Commit to master instead!
  - Behind an if statement
- Reduce merge conflicts
- Simplest way: config files
  - appsettings.{Environment}.json
- Commit to cleaning up after it’s live
- Clearly separate out your Feature Toggles to make it obvious what Feature Toggles exist
  - Have a FeatureToggles class or section in your config file

Questions on Feature Toggles?

Microservices

<Something about Microservices />

- When I say Microservices I mean some out of process call of some sort (web service, queue, etc.)
- Microservices add operational complexity
  - Health Checks
    - Is it running? Are the dependencies ok?
  - Load Balancing
  - Service Discovery (via Consul or something simple like config files with primary/backup)
  - Perf is worse
- Microservices add development complexity
  - HTTP call (most likely) instead of a method call
  - If calling from C#, likely want a client library wrapper around HttpClient sending JSON
  - Dev environment needs multiple projects running
  - Cross-cutting things (users, logging, common packages, versions, etc.)

<Something about Microservices />

- Benefits
  - Iterate on microservices separate from other app (SRP)
  - Reuse
  - Scale independently
  - Easy to reason about in small chunks
- You are not Google/Facebook/Netflix/Amazon
- Use where it makes sense:
  - Needs scale
  - High CPU usage
  - Isolate an annoying dependency
  - Separate teams
  - Hard costs is a priority over engineering costs (i.e. Lambda, Azure Functions)

Daniel Vassallo
@dvassallo

Step 1: Forget that all these things exist: Microservices, Lambda, API Gateway, Containers, Kubernetes, Docker.

Anything whose main value proposition is about “ability to scale” will likely trade off your “ability to be agile & survive”. That’s rarely a good trade off.

4/25

5:20 PM · Jul 25, 2019 from Seattle, WA · Twitter for iPad

53 Retweets **177** Likes

Michael Feathers
@mfeathers

Follow

Make something simple and add complexity reluctantly.

9:08 PM - 6 Jul 2017

86 Retweets **137** Likes

3 86 137

Questions on Microservices?

Real benefits of these practices

- 1 web app went from deploying to Prod 20x a year to deploying to Prod over 500x a year.
- Faster delivery of value to users.

Summary

- Feature Folders > MVC Folders
- SRP for UI with Components
- Happy Path at the Bottom of a Method
- FluentValidation over Data Annotations/Custom
- Use an ORM
- Avoid new, Statics, HttpContext, ConfigurationManager for testing
- Automate your Build and Release pipeline
- Use Feature Toggles over environment branches
- Resist Microservices until there's a good reason not to

Closing

- Hope you got at least one idea out of this.
- You likely don’t agree with everything I just said.
- Remember – point is to deliver value to your end users as quickly and reliably as possible and allowing that trend to continue in the future.
- Don’t get caught up in what library you’re using
  - If you use NUnit instead of xUnit and have no reason to switch – who cares?
  - Important thing is you’re doing automated testing.

Questions?

- Feel free to reach out on Twitter (@scottsauber) if you think of a question later
- Slides posted on my blog (scottsauber.com) and I'll tweet them out
- Don't forget to fill out evals, please

Thanks!