
Headspring
Supercharging ASP.NET
Core Applications

About Me

Nolan Egly

Principal Consultant

BIO

Experience with many different languages and always willing to learn more. I take pride in doing good work and prefer working with others that also love software development and aspire to continue improving themselves.

TECHNICAL EXPERTISE (circa 2012)

C#, T-SQL, ASP.NET WebForms and MVC, C, C++, C#, Delphi, Java, VB 6, PL-SQL, HTML, CSS

FUNCTIONAL EXPERTISE

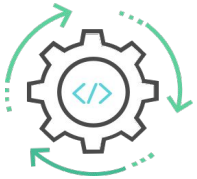
Test Driven Development / Unit Testing

CLIENTS

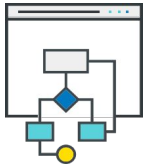
Michael and Susan Dell Foundation
Modern Woodmen
Global Resale
Grifols
USA Compression
YETI
Dell

Headspring

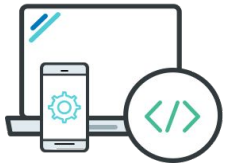
STRATEGY AND CONSULTING



LEGACY SYSTEM MODERNIZATION



ENTERPRISE ARCHITECTURE



APPLICATION DEVELOPMENT

HeadspringLabs

betterway.headspring.com

Truth In Advertising

This talk will

Review the major features of some libraries that can be useful in an ASP.NET Core app - mostly code, some slides

Highlight why a library might be better than the OOTB experience (and also why it may not)

General ‘do this’ and ‘don’t do that’ recommendations based on personal experience

Be a chance to learn from each other

Code and slides will be published to GitHub afterwards

This talk won’t

Be a deep dive on any particular library

Doesn’t cover JS frontends

Agenda

- HtmlTags
- FluentValidation
- Automapper
  - Projecting from a database
- Mediatr
- Vertical Slices (a code organization technique, not a library)


HTMLTAGS

HtmlTags

UI composition

".NET objects for generating HTML"

Single greatest benefit

You can compose methods in C# to standardize the way an app generates common HTML tags. This helps drive a standardized HTML structure, centralizes how common attributes and layout are done, and reduces errors in boilerplate code.

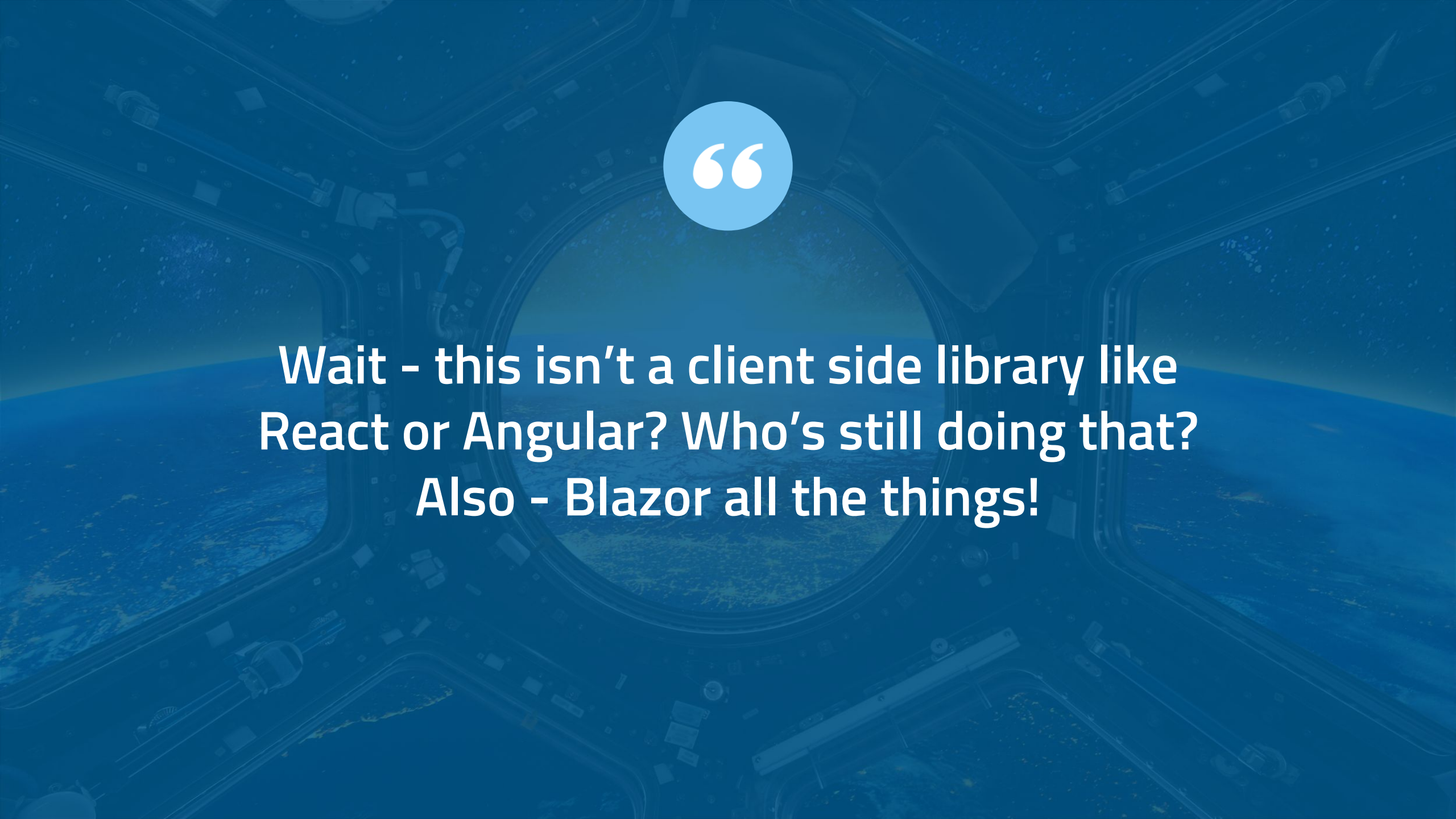
HtmlTags

History

Initial claim to fame was being the first HTMLHelper implementation not based on magic strings

Originally part of a larger effort for a complete framework alternative to the original ASP.NET MVC

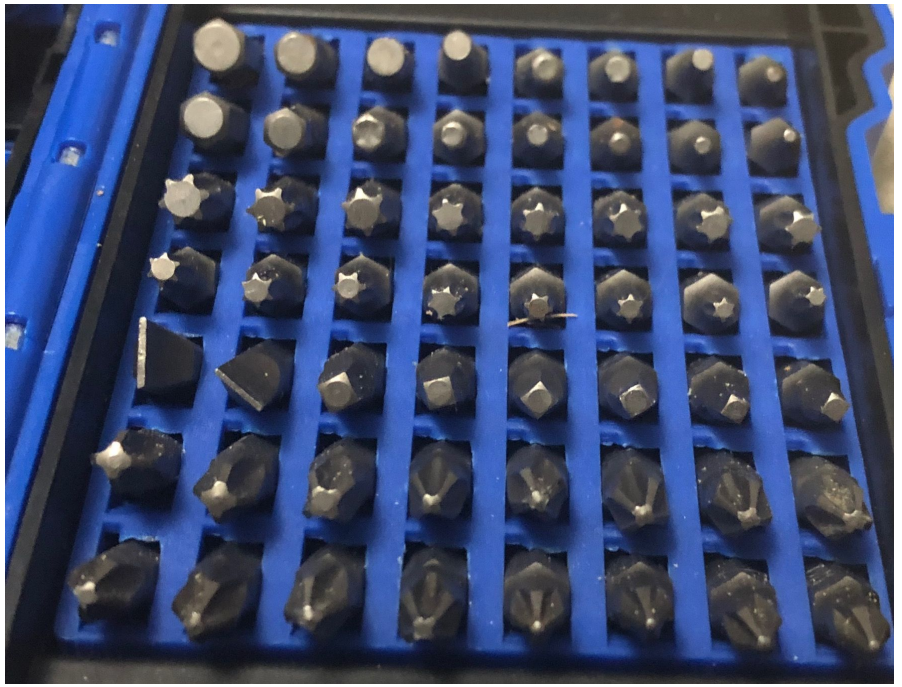
Broken out into a stand-alone tool


Wait - this isn’t a client side library like
React or Angular? Who’s still doing that?
Also - Blazor all the things!

One Size Does Not Fit All

When choosing tools, technological aspects are not always the greatest impacting factor

- Current team lacks JavaScript skills
  - Organization may not be willing to invest in training
  - Project deadline may not allow for learning time
- Negative past experiences
  - Tech lead may have been negatively impacted by previous churn in particular framework
- UX needs are no to low interaction
  - But should probably also be considering a static site generator or CMS at that point



HtmlTags - Points of Interest

General “scale of sample code” disclaimer

Wired up in StartUp

Basic tag methods - Volunteer Create.cshtml
- Point out IHtmlHelper ambiguity, nullable value types on incoming data
- Contrast with Applicant Create.cshtml

Display and DisplayBlock - Volunteer Detail.cshtml
- Point out “fixes” to the library

App-wide conventions for how data types get rendered - TagConventions.cs
- Play with date rendering

Select builders, hidden inputs - Volunteer DetailEdit.cshtml
- Organization - rendering and rehydration
- RowVersion - how configured in conventions, and round-tripping

HtmlTags In the Wild

Some UI Widgets I've seen composed in HtmlTags

A data "graphic" (table with fancy styling and color coding) for compression engine readings
Dashboard readouts of business or department critical metrics
Cards of student information with multiple sections of related data points

HtmlTags - Tripwires

It’s easy to get carried away in how much structure a single method builds up

- One-method-to-render-it-all, but with a ginormous amount of configuration or data passed it
- These usually become difficult to debug or easily change

Resist this temptation!

- Use tag methods to render a single repeatable element, and then potentially call it from a parent method

Loose rule of thumb

- The method chain of your tag rendering should resemble the different hierarchy levels of the output structure

HtmlTags - Upsides

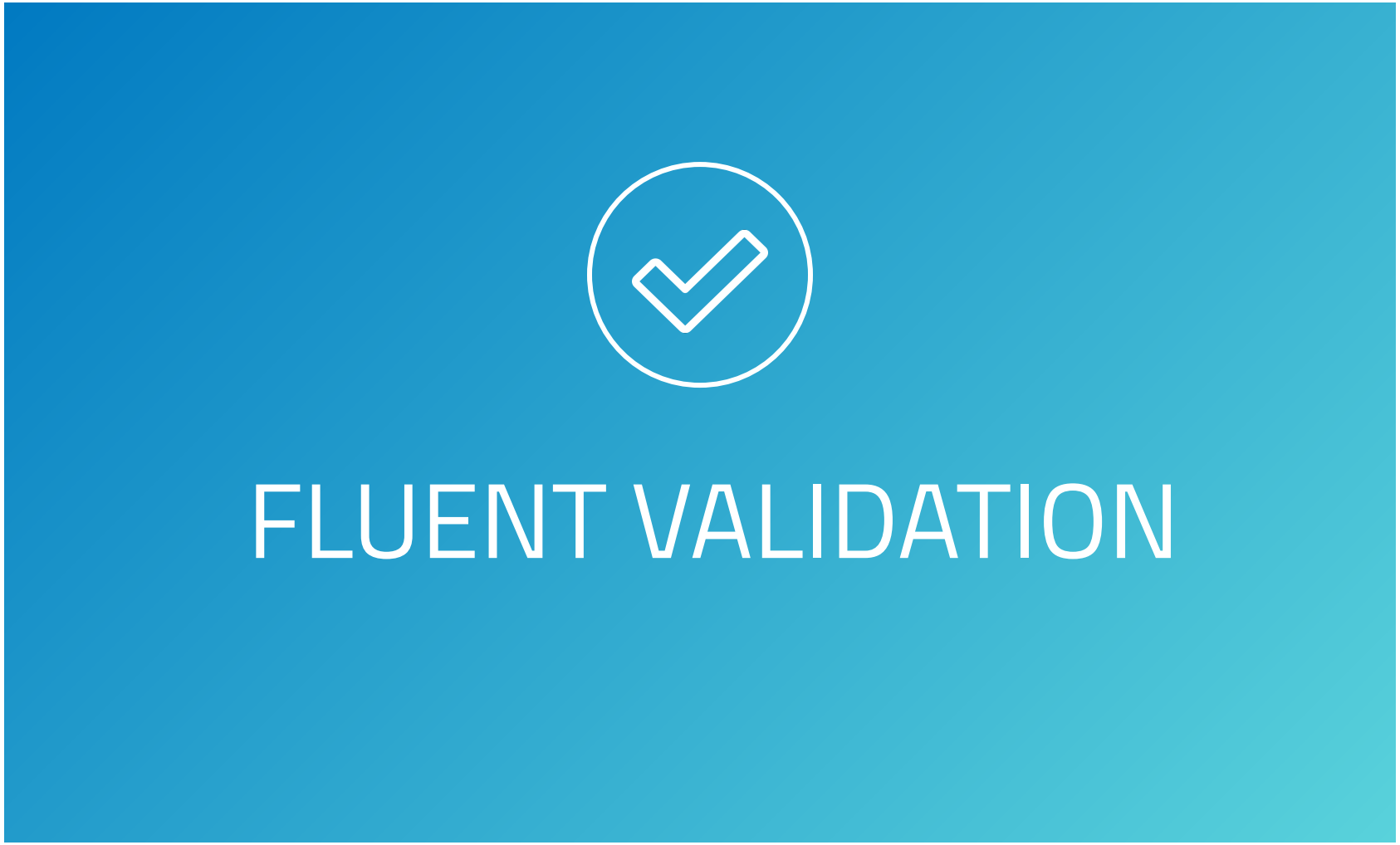
Upsides

- Very powerful tool for creating central implementations to generate common HTML structures
- Conventions make it possible to affect how particular cross cutting properties or data types are rendered

HtmlTags - Downsides

Downsides

- It’s a server side technology
- Documentation is very sparse
  - plan on carefully looking at example projects using it


FLUENT VALIDATION

FluentValidation

Data validation

"A small validation library for .NET that uses a fluent interface and lambda expressions for building validation rules."

Single greatest benefit

Unlike the built in attribute approach, validations are not coupled directly to the model being validated. This greatly increases the ability to easily deal with interesting variations without spreading validation across different objects.

FluentValidation

History

Initial claim to fame was “it’s fluent!”

Implemented several validation patterns that weren’t baked into .NET at the time

Straightforward to write your own validators

FluentValidation - Points of Interest

Wire up in StartUp

Example of simple validation - Volunteers Create

- Fluent chains
- Expression based (no magic strings)
- Inheritence based

Example of complex/conditional validation - Applicants Create

- Mix of annotations and FV
- Must/When

FluentValidation

Supports manual validation for edge-case scenarios

```
Customer customer = new Customer();
CustomerValidator validator = new CustomerValidator();

ValidationResult results = validator.Validate(customer);

if(! results.IsValid) {
  foreach(var failure in results.Errors) {
    Console.WriteLine("Property " + failure.PropertyName + " failed validation. Error was: " + failure.E
  }
}
```

FluentValidation

Expression approach makes it easy to completely customize a validator

```
public class PersonValidator : AbstractValidator<Person> {
  public PersonValidator() {
   RuleFor(x => x.Pets).Custom((list, context) => {
     if(list.Count > 10) {
       context.AddFailure("The list must contain 10 items or fewer");
     }
   });
  }
}
```

FluentValidation - client side

Most of the built in validations work with ASP.NET's client side validation, but not all of them. Check the docs.

Custom validators or conditional clauses will only work server side.

FluentValidation - Points of Interest (skip)

Fully asynchronous validation calls - Applicants Create.cshtml

- Form post hijack in site.js
- Validation errors handled serverside in ValidatorPageFilter.cs
- Business exceptions handled serverside in ExceptionPageFilter.cs, clientside in site.js

Not FV specific, could be done with annotations too

FluentValidation - Tripwires

Because it's so easy to write your own validators, it can be tempting to put ALL of your validation in Validators

- Custom validator that takes a DbContext in the ctor
- Do database reads for record checks similar to "user name must be unique"

Resist this temptation!

- Potential performance hit
  - Accessing database at the same time you're validating simple criteria like required and max length
- Putting that much complexity in the validator makes automated testing more difficult
- Moving "business logic" validation out of the backend service/handler limits "headless" reuse

Keep the two core types of validation in mind, and put them in the right place

- Data record validation - FluentValidation
  - IsRequired, max length 50, must be a number
- Business logic validation - backend
  - Unique value, running totals

FluentValidation - Upsides

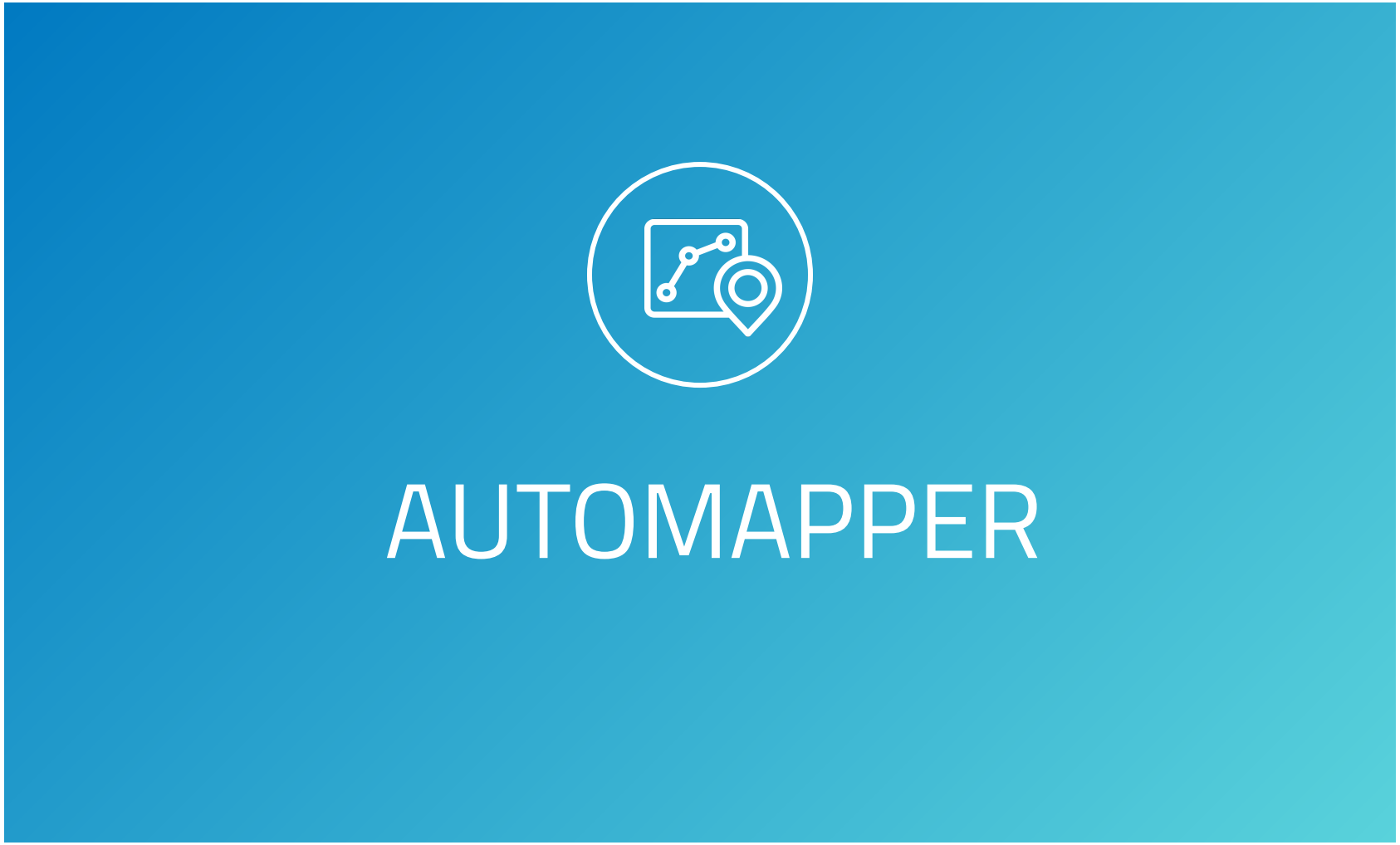
Upsides

- Decoupled approach leads to easier flexibility
  - Save Draft functionality
- Custom validators are easy to test

FluentValidation - Downsides

Downsides

- Because it’s so easy, can accidentally put too much in the validator


AUTOMAPPER

AutoMapper

Object mapping

“An object-object mapper.
Because Mapping code is boring.”

Single greatest benefit

Reduce boilerplate code of assigning properties from one object to another, and at the same time helping enforce a convention for naming class properties.

AutoMapper

History

Extracted from a project with many large data concepts

Used to have static configuration options that could be difficult to stub

AutoMapper

The big picture - why are we doing mapping in the first place?

Keeping the “why” in mind helps us choose the “when”

AutoMapper

Mapping is most useful when crossing boundaries, usually with DTOs

From the docs:

"AutoMapper uses a convention-based matching algorithm to match up source to destination values. AutoMapper is geared towards model projection scenarios to flatten complex object models to DTOs and other simple objects, whose design is better suited for serialization, communication, messaging, or simply an anti-corruption layer between the domain and application layer."

AutoMapper - Points of Interest

Wired in StartUp

Flat example - Applicants Create

Composite example - Volunteers Detail

- Auto-flattening
- Customizing mapping

Reverse map - Volunteers DetailEdit

AutoMapper - Developer Niceties

Mappings are dynamically compiled at runtime, not at compile time

It's possible to have a mapping error, and not know it until the program is actually run.

Automapper has a built in feature to validate all of the mapping configurations. Writing an automated test for this will significantly help catch mapping issues before the code ships.

```
var config = AutoMapperConfiguration.Configure();

config.AssertConfigurationIsValid();
```

AutoMapper - Developer Niceties

Debugging complex mappings

If you're having trouble understanding why a particular mapping isn't behaving as expected, you can get AutoMapper to export the execution plan so you can review it

The docs recommend several ways of how to conveniently view the plan

Note: if you're resorting to this, maybe you're mapping is to complex

```
var configuration = new MapperConfiguration(cfg => cfg.CreateMap<Foo, Bar>());
var executionPlan = configuration.BuildExecutionPlan(typeof(Foo), typeof(Bar));
```

AutoMapper - Conventions

Sensible defaults out of the box, but can be customized

- Property name parsing based on Pascal convention
  - Can be overridden at the global level, or profile level
  - Useful for using AutoMapper with other data sources or JSON serialization
- Validates all Destination properties are mapped to
  - Can configure this per-profile to validate the Source instead

AutoMapper - Customizing a Mapping

Three primary options to override convention

- Ignore the destination
  - Useful when the property is something that you'll set manually
- Projection
  - Customize the mapping with specific links that don't follow the general conventions
- Add an extension
  - Write custom converters or resolvers to execute your own C# code when mapping between two objects

AutoMapper - Ignore

Simply specify what destination member AutoMapper should not validate or populate

```
public class MappingProfile : Profile
{
    public MappingProfile() =>
        CreateMap<CreateCommand, Course>()
            .ForMember(c => c.Id, opt => opt.Ignore());
}
```

AutoMapper - Projection

Use expressions to link a source and destination property together

This example shows splitting a single DateTime into two fields for UI rendering

```
public class CalendarEvent
{
        public DateTime Date { get; set; }
        public string Title { get; set; }
}
```

```
public class CalendarEventForm
{
        public DateTime EventDate { get; set; }
        public int EventHour { get; set; }
        public int EventMinute { get; set; }
        public string Title { get; set; }
}
```

```
var configuration = new MapperConfiguration(cfg =>
  cfg.CreateMap<CalendarEvent, CalendarEventForm>()
        .ForMember(dest => dest.EventDate, opt => opt.MapFrom(src => src.Date.Date))
        .ForMember(dest => dest.EventHour, opt => opt.MapFrom(src => src.Date.Hour))
        .ForMember(dest => dest.EventMinute, opt => opt.MapFrom(src => src.Date.Minute)));
```

AutoMapper - Use an Extension

There are different scopes available, choose the extension type that fits your need

- Value converter
  - Translates from a single source member to a single destination member
- Value resolver
  - Translates from multiple source members to a single destination member
- Type converter
  - Translates a source object to a destination object

AutoMapper - Database Projection

The Queryable Extensions let you query only the database fields you're mapping to

- Adds an extension method **ProjectTo** on the IQueryable used with the ORM
- Only supports the operations that IQueryable understands
- Can help reduce network traffic and database load originating from your application

Point of Interest - Volunteer Detail

AutoMapper - Tripwires

Because AutoMapper provides straightforward customization, it can be easy to end up in “every problem is a nail” territory

- Excessive or inappropriate automapping
- One symptom of this is a “pipeline” of multiple map steps on a single operation
- Remember mapping is most useful at context boundaries

Having maps with more customization than convention

- Automapper provides extension points, but it’s more infrastructure and less readability then simply writing “boilerplate” assignment code
- Rough rule of thumb, more than 20% customization in mappings is a red flag to abandon auto mapping

Performance (small risk)

- Because mappings are dynamically compiled, there is a perf hit when evaluating them as app starts up
  - Hundreds of mappings could take a second or two
- As a result, AutoMapper does lazy evaluation by default to avoid a startup freeze
  - But this can be configured otherwise. Test the startup time if you disable this

AutoMapper - Upsides

Upsides

- Reduces time and errors associated with writing object mapping code
- Mature tool with well thought out configuration and extensibility

AutoMapper - Downsides

Downsides

- Involves some “automagic”
  - Removal of static interfaces has helped reduced this in newer versions
- Dynamic compilation could have a small perf impact on very large apps
- Can require discipline to evaluate whether a complex scenario should just be coded manually


MEDIATR

Mediatr

Code decoupling

"Low-ambition library trying to solve a simple problem - decoupling the in-process sending of messages from handling messages."

Single greatest benefit

Flexible request sending or event signaling within a single system without tightly coupling the various parties together. This makes it possible to cleanly separate the code handling user requests from the user interface (HTTP, XAML, etc.), and for unrelated features that should remain in separate code files to notify about events.

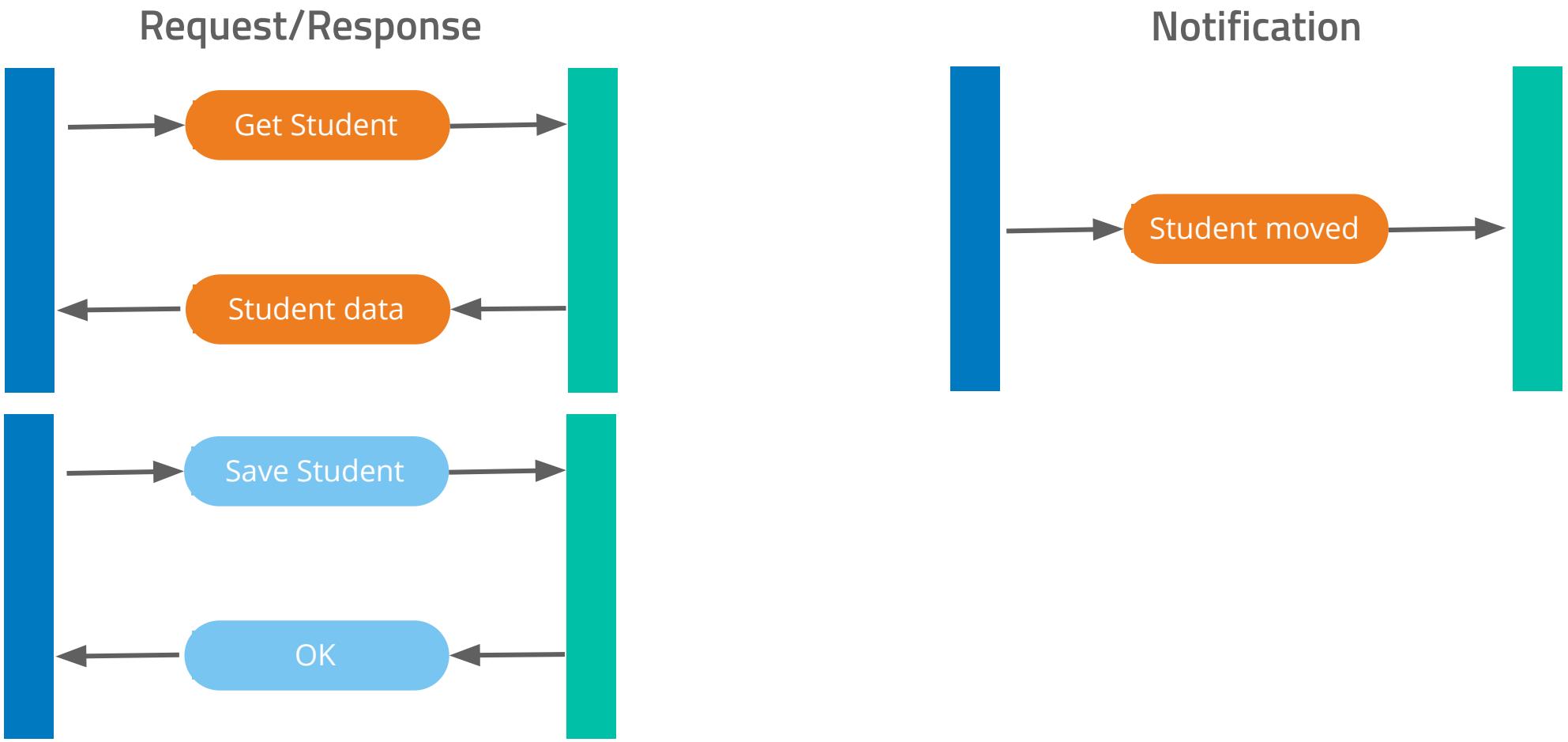
Mediatr

History

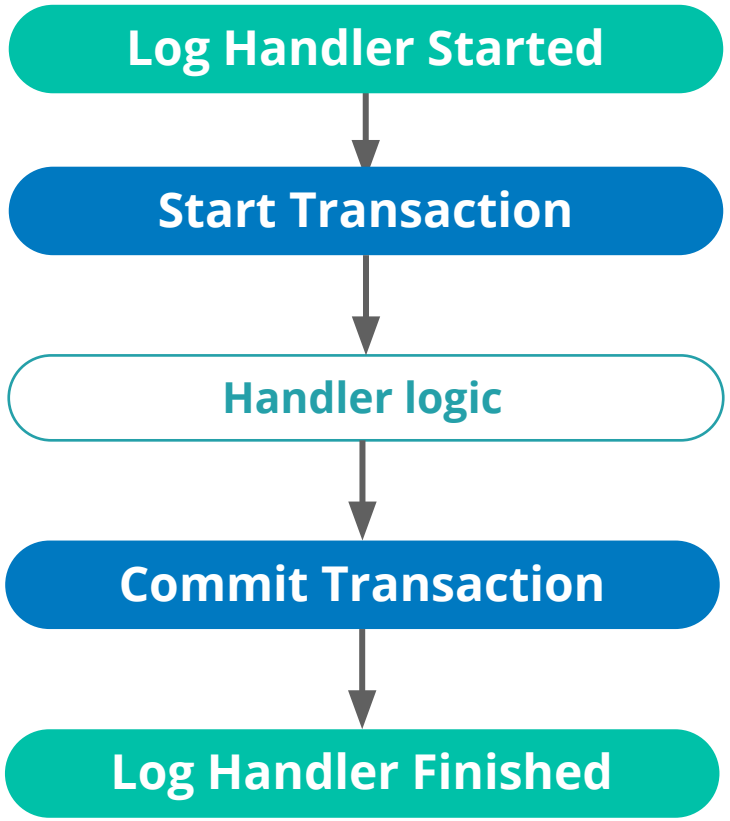
Prototype was in a codebase that needed to signal across independent features when something interesting happened (e.g. when a customer's third order becomes payment past due, add a note on customer service file)

Quickly started getting applied as a way to formalize a boundary between UI and domain

MediatR - Message Types



MediatR - Configurable Pipelines

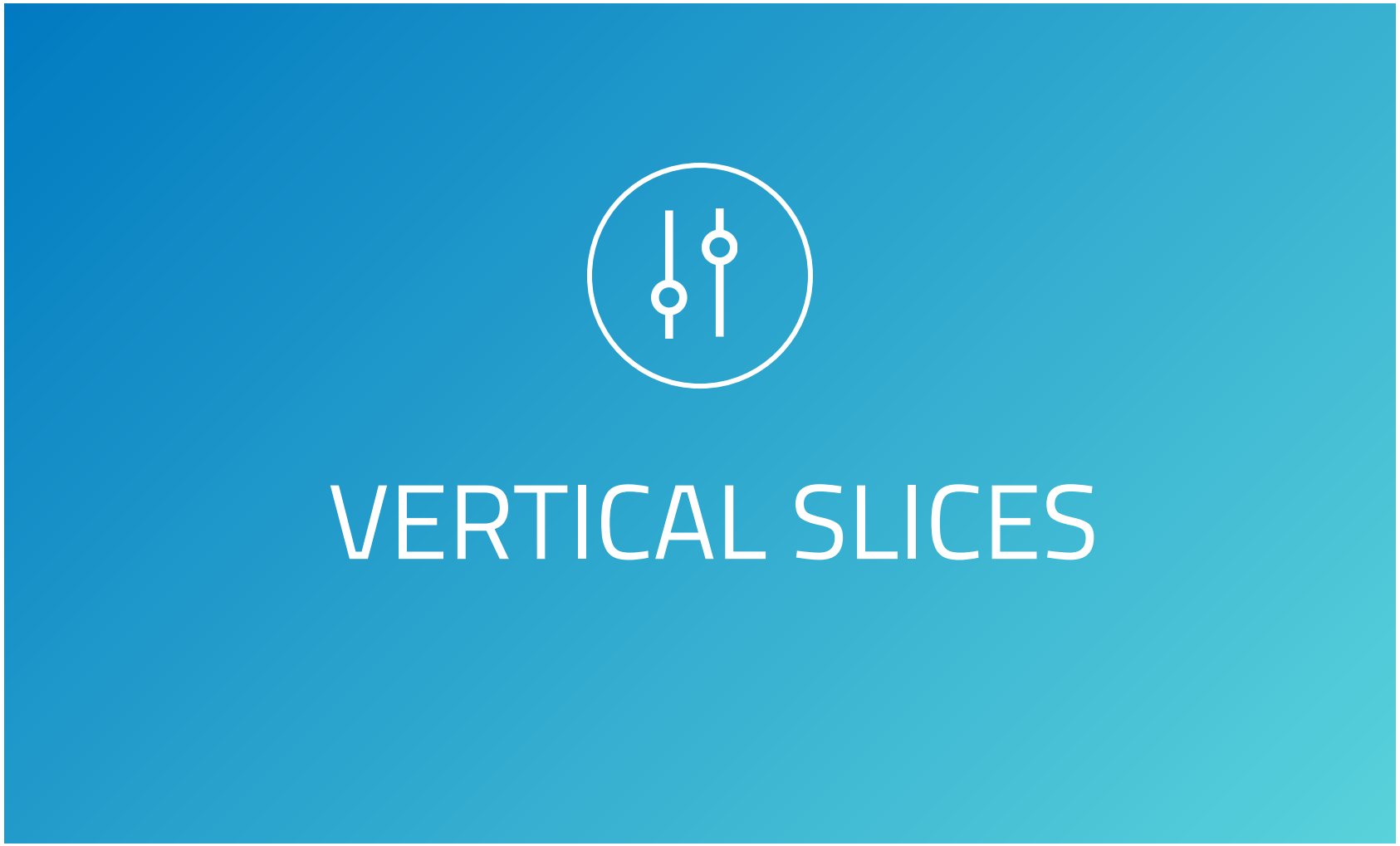


Mediatr - Points of Interest

Wired in StartUp
- Pipelines

Query request - Applicants Index
- Query object with no parameters

Query request and Command request - Volunteers Edit


VERTICAL SLICES

Vertical Slices

(not a library)

A code organizing technique that groups classes based on what business feature they support

Vertical Slices

Code organized by business feature

Student file	Class folder
Student page	Class page
Student queries	Class queries
Student commands	Class commands
Student domain	Class domain

Code organized by technical layer

Layer	Code
UI layer	Student page Class page
Services or Controller layer	Student service Class controller
Domain or Business Logic layer	Student class Class BAL
Data Access layer	Student repository Class DAL

Vertical Slices

Upsides

- At coding time
  - most of the relevant code is all in file or folder together
- At compile time
  - reduces the number of project files that have to be built
  - Personal experience - project that went from a handful to 50 projects
- At deployment time
  - Can have more targeted flexibility and isolation, if needed
  - Company experience - medical software that had a highly regulated feature

Vertical Slices

Downsides

- Trusts the developer to refactor
  - Can end up with overly complex subfeatures if feature expansion isn't restructured along the way
- Not one single implementation pattern
  - Newer team members may be unsure how to structure a nontrivial feature

Vertical Slices

Other Resources

Short blog post comparing vertical slice architecture with a layered one
https://jimmybogard.com/vertical-slice-architecture/

NDC presentation on Vertical Slices
https://www.youtube.com/watch?v=SUiWfhAhgQw

Thank you!

Nolan Egly

Principal Consultant

Twitter
@nolanegly

GitHub
nolanegly

LinkedIn
nolanegly

Email
nolan.egly@headspring.com
nolan@nolanegly.com


Headspring
Q & A