ReactiveUI & Xamarin.Forms

Michael Stonis

Eight-Bot, Inc. & Aurora Controls
Microsoft MVP

@MichaelStonis

ston.is
eight.bot
auroracontrols.app

Reactive UI

- Originally Created by Ani Betts
- MVVM Framework….kind of.
- Declarative, asynchronous-friendly way to build UIs
- Support for major .Net Platforms
  - Xamarin.Forms
  - UWP
  - WPF
  - Avalonia
  - Tizen
  - Native Mobile
  - More…

ReactiveUI Features

- View Model Components
- Reactive Platform Wrappers
- Data Binding
- Commands
- Thread Scheduling
- Helpers (Validation, Event Processing, Logging, Service Locator, etc.)

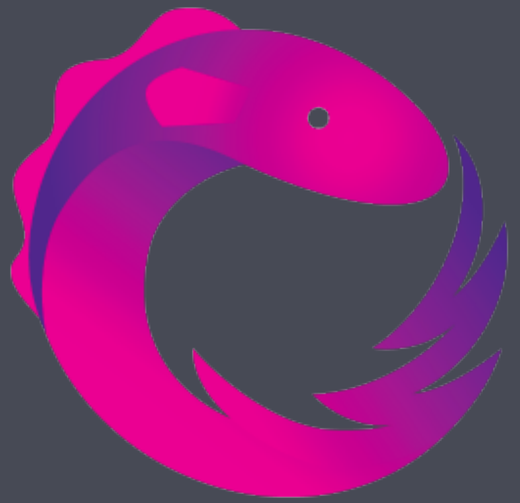
The un-MVVM Framework

- At its core, RxUI is really a bunch of extension methods
- It is extremely **NOT PRESCRIPTIVE**
- Unlike other MVVM frameworks, RxUI leaves it up to the developer to choose which parts they want to use
- There is not exactly one *right way* to use RxUI
- Works great with other MVVM Frameworks like Prism

Reactive Extensions

Splat

Dynamic Data



Reactive Extensions

- API for asynchronous programming using streams
- Listen for events or data notifications
- LINQ-like syntax for data processing
- Operators to make difficult processing easy
- Very popular across languages

http://reactivex.io

It’s Basically LINQ for Events

```
Observable
    .FromEventPattern<TextChangedEventArgs>(
        x ⇒ textEntry.TextChanged += x,
        x ⇒ textEntry.TextChanged -= x)
    .Select(x ⇒ x.EventArgs.NewTextValue)
    .Throttle(TimeSpan.FromSeconds(.75), TaskPoolScheduler.Default)
    .Where(searchText ⇒ !string.IsNullOrWhiteSpace(searchText))
    .SelectMany(
        async searchText ⇒
        {
            var searchService = Locator.Current.GetService<IDuckDuckGoApi>();
            return await searchService.Search(searchText).ConfigureAwait(false);
        })
    .ObserveOn(RxApp.MainThreadScheduler)
    .Do(
        results ⇒
        {
            /* Update the UI */
        })
    .Subscribe();
```

Subscribe to events

Filter Data

New Operators

Async Support

Explicit Scheduling

Splat

- Simple Service Locator
- Light-weight and extremely fast
- Logging
- Cross-Platform type translators

https://github.com/reactiveui/splat

Dynamic Data

- Bridges the world of Rx to collections
- Batman’s utility belt as an ObservableCollection
- Built-in Operators
  - Filtering
  - Sorting
  - Transformation
  - So many more…

https://github.com/ReactiveUI/DynamicData

Features

VS

Xamarin.Forms ViewModel

- Uh, there isn’t one out of the box…
- You likely will need to implement INotifyPropertyChanged
- No good way to prevent property notifications
- No default error handling
- No default instrumentation

ReactiveObject

- Base ViewModel Object for RxUI
- Intelligent Property Change Notifications
- Provides the ability to suppress and delay change notifications out of the box
- Support for automatically persisting the data based on changes and time
- Logging support
- Error routing

```
public class ViewModel : ReactiveObject
{
    [Reactive] public string FirstName { get; set; }
    [Reactive] public string LastName { get; set; }

    public string FormattedName { [ObservableAsProperty] get; }

    public BetterViewModel()
    {
        this.WhenAnyValue(
            vm => vm.FirstName,
            vm => vm.LastName,
            (first, last) => $"{last}, {first}")
        .ToPropertyEx(this, x => x.FormattedName);
    }
}
```

```
var simpleViewModel = new SimpleViewModel();

simpleViewModel.FirstName = "Michael";
simpleViewModel.LastName = "Stonis";

using(simpleViewModel.SuppressChangeNotifications())
{
    simpleViewModel.FirstName = "Miguel";
    simpleViewModel.FirstName = "Mike";
}
```

```
var simpleViewModel = new SimpleViewModel();

simpleViewModel.FirstName = "Michael";
simpleViewModel.LastName = "Stonis";

using(simpleViewModel.SuppressChangeNotifications())
{
    simpleViewModel.FirstName = "Miguel";
    simpleViewModel.FirstName = "Mike";
}
```

```
var simpleViewModel = new SimpleViewModel();

simpleViewModel.FirstName = "Michael";
simpleViewModel.LastName = "Stonis";

using(simpleViewModel.DelayChangeNotifications())
{
    simpleViewModel.FirstName = "Miguel";
    simpleViewModel.FirstName = "Mike";
}
```

```
var simpleViewModel = new SimpleViewModel();

simpleViewModel.FirstName = "Michael";
simpleViewModel.LastName = "Stonis";

using(simpleViewModel.DelayChangeNotifications())
{
    simpleViewModel.FirstName = "Miguel";
    simpleViewModel.FirstName = "Mike";
}
```

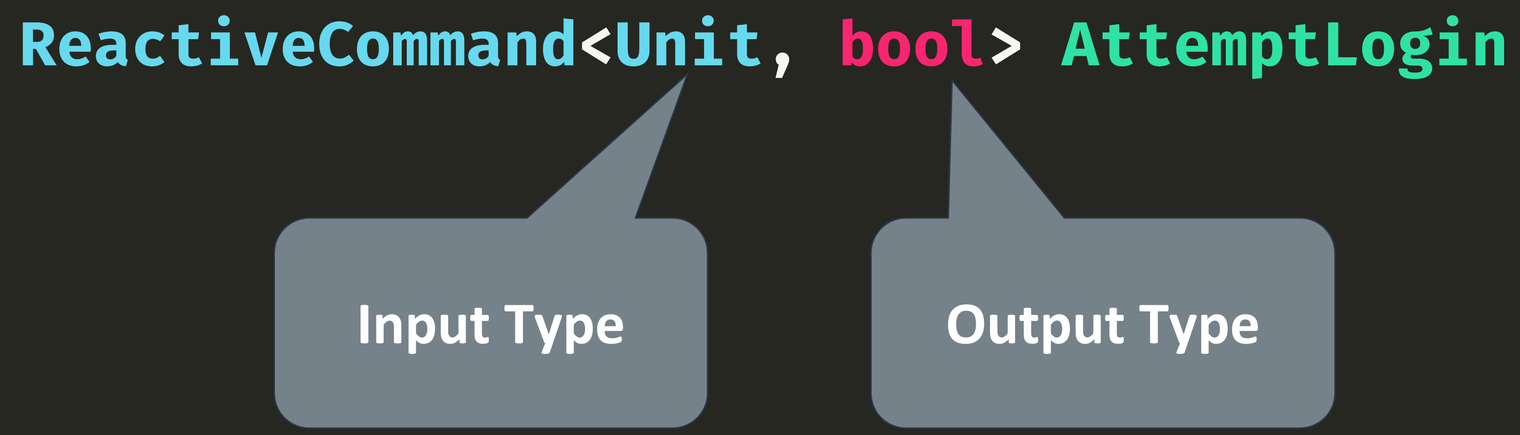
```
this.ThrownExceptions
    .Do(ex ⇒ Debug.WriteLine($"Exception: {ex}"))
    .Subscribe()
    .DisposeWith(ViewModelBindings);

this.Log()
    .Info("Everything is looking good!");
```

Xamarin.Forms Command

- No great out of the box support for async/await
- No support for cancellation
- CanExecute functionality is minimal at best
- No typed input
- No notification upon completion

ReactiveCommand

- A *really good* implementation of ICommand
- Supports synchronous and asynchronous operations OOTB
- Practical and easy-to-use support for command disabling
- Pretty dang easy support for command cancellation
- Great integration with Xamarin.Forms


ReactiveCommand<Unit, bool> AttemptLogin
Input Type
Output Type

```
ReactiveCommand
    .CreateFromTask(
        async () =>
        {
            this.Log().Info("Starting Login");
            var result = await ProcessLogin();
            this.Log().Info($"Finished Login");

            return result;
        },
        this.WhenAnyValue(x => x.Username, x => x.Password,
            (username, password) =>
                !string.IsNullOrWhiteSpace(username) &&
                !string.IsNullOrWhiteSpace(password)));
```

```
ReactiveCommand
    .CreateFromTask(
        async () =>
        {
            this.Log().Info("Starting Login");
            var result = await ProcessLogin();
            this.Log().Info($"Finished Login");

            return result;
        },
        this.WhenAnyValue(x => x.Username, x => x.Password,
            (username, password) =>
                !string.IsNullOrWhiteSpace(username) &&
                !string.IsNullOrWhiteSpace(password)));
```

Can create from synchronous, observable, and other commands too

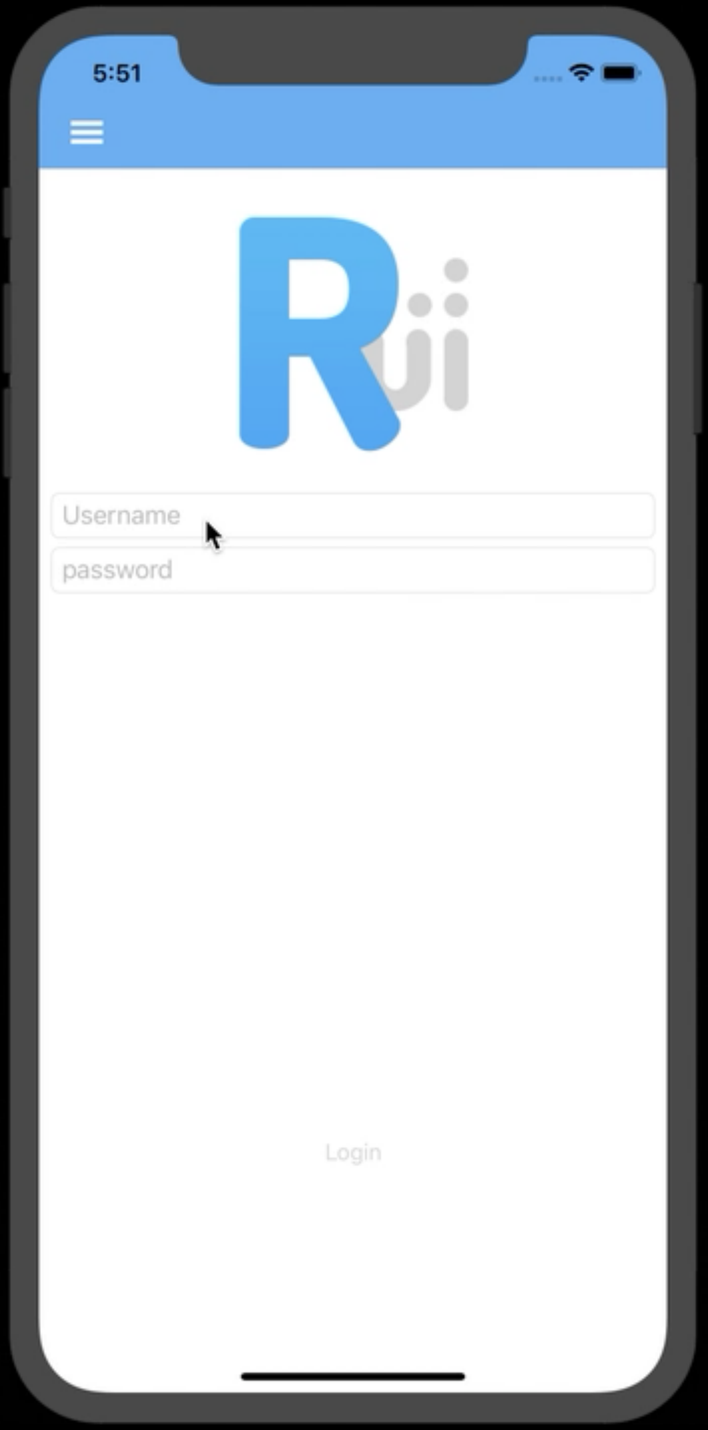
```
ReactiveCommand
    .CreateFromTask(
        async () =>
        {
            this.Log().Info("Starting Login");
            var result = await ProcessLogin();
            this.Log().Info($"Finished Login");

            return result;
        },
        this.WhenAnyValue(x => x.Username, x => x.Password,
            (username, password) =>
                !string.IsNullOrWhiteSpace(username) &&
                !string.IsNullOrWhiteSpace(password)));
```

This is the work that the command will do

```
ReactiveCommand
    .CreateFromTask(
        async () =>
        {
            this.Log().Info("Starting Login");
            var result = await ProcessLogin();
            this.Log().Info($"Finished Login");

            return result;
        },
        this.WhenAnyValue(x => x.Username, x => x.Password,
            (username, password) =>
                !string.IsNullOrWhiteSpace(username) &&
                !string.IsNullOrWhiteSpace(password)));
```

```
ReactiveCommand
    .CreateFromTask(
        async () =>
        {
            this.Log().Info("Starting Login");
            var res
            this.Lo

            retur
        },
        this.WhenAnyValue(x => x.Username, x => x.Password,
            (username, password) =>
                !string.IsNullOrWhiteSpace(username) &&
                !string.IsNullOrWhiteSpace(password)));
```

This will control whether the command *CAN* be executed


5:51
Username
password
Login

Xamarin.Forms Bindings

- Value Converters are a ton of fun
- Can be difficult to determine binding mode
- No direct way to bind *ANY* event to a command
- Can’t batch bindings
- No great way to bind to the completion of an event or notification
- Bindings are getting better though with things like Multi-Bindings
- Check out the C# Markup Bindings!

Bindings & Update Notifications

- Two-Way and One-Way Bindings
- Built-In Type Conversion and Easy Value Conversion Overrides
- Command Binding
- Control when binding updates source
- Support for Xamarin.Forms, Xamarin.iOS, Xamarin.Android, WPF, Windows Forms, Avalonia, UWP, etc.

The Binding Host View

The Notifying Property

```
this.Bind(ViewModel, vm => vm.Username, view => view.username.Text)
```

The ViewModel Property

The View/Control Property

```
this.OneWayBind(ViewModel, vm => vm.WorkStatus, view => view.workStatus.Text)
```

```
this.BindCommand(ViewModel, vm => vm.StartWorking, view => view.startWork)
```

```
this.WhenAnyValue(x => x.ViewModel.UsernameValidation.IsValid)
    .Select(isValid => isValid ? Color.Green : Color.Red)
    .ObserveOn(RxApp.MainThreadScheduler)
    .BindTo(this, view => view.username.BackgroundColor)
```

```
this.WhenAnyObservable(x => x.ViewModel.CancelProcessing)
    .ObserveOn(RxApp.MainThreadScheduler)
    .Do(
        async _ =>
        {
            await this.DisplayAlert(
                "Cancelled", "Processing Cancelled", "OK");
        })
    .Subscribe()
    .DisposeWith(viewDisposables);
```

```
this.WhenAnyValue(x => x.ViewModel.UsernameValidation.IsValid)
    .Select(isValid => isValid ? Color.Green : Color.Red)
    .ObserveOn(RxApp.MainThreadScheduler)
    .BindTo(this, view => view.username.BackgroundColor)
```

```
this.WhenAnyObservable(x => x.ViewModel.CancelProcessing)
    .ObserveOn(RxApp.MainThreadScheduler)
    .Do(
        async _ =>
        {
            await this.DisplayAlert(
                "Cancelled", "Processing Cancelled", "OK");
        })
    .Subscribe()
    .DisposeWith(viewDisposables);
```

```
this.WhenAnyValue(x ⇒ x.ViewModel.Load)
    .IsNotNull()
    .ObserveOn(RxApp.MainThreadScheduler)
    .Do(
        async load ⇒
        {
            _contentContainer.Opacity = 0f;

            _logo.Source = load.VendorImageUrl;
            _vendorName.Text = load.Vendor;
            _loadId.Text = load.LoadId;
            _payment.Text = load.Payment.ToCurrencyDisplay();
            _pickupDatetime.Text = load.PickupDateTime.ToLongDisplay();
            _locationPickup.Text = load.PickupLocation;
            _locationDestination.Text = load.DestinationLocation;
            _totalDistance.Text = $"{load.TotalDistance:n0} mi";
            _totalStops.Text = $"{load.TotalStops} stops";
            _payloadInfo.Text = $"{load.Equipment} / {load.Commodity} / {load.Weight}";

            await _contentContainer.FadeTo(1f, Durations.FadeIn, easing: Easing.CubicInOut);
        })
    .Subscribe()
    .DisposeWith(ControlBindings);
```

Xamarin.Forms ObservableCollection

- Barebones implementation
- Sledgehammer approach to updates
- Doesn’t support batch updates
- Each update/insert/delete triggers a notification
- No built-in filtering, transformation, etc.

DynamicData

- Observable Lists and Caches
- Granular controls for change notification including batch updates
- Built-In support for filtering, transformation, sorting, aggregation
- Provides the ability to listen to property changes of items
- Highly optimized and extremely fast

```
_resultsSourceList =
    new SourceList<TaskResultCaptureViewModel>()
        .DisposeWith(ViewModelBindings);

_resultsSourceList
    .Connect()
    .BufferIf(this.WhenAnyValue(x => x.Categories).Select(x => x == null))
    .GroupOn(x => x.TaskInstance.CategoryId)
    .Transform(x =>
        new TasksByCategory
        {
            CategoryId = x.GroupKey.Value,
            Category = Categories.FirstOrDefault(c => c.Id == x.GroupKey.Value),
            TaskResultCaptures = x.List.Items.ToList()
        })
    .Sort(OrderedComparer<TasksByCategory>.OrderBy(x => x.Category.Name))
    .ObserveOn(RxApp.MainThreadScheduler)
    .Bind(out var results)
    .DisposeMany()
    .Subscribe()
    .DisposeWith(ViewModelBindings);
```

Connect to list updates

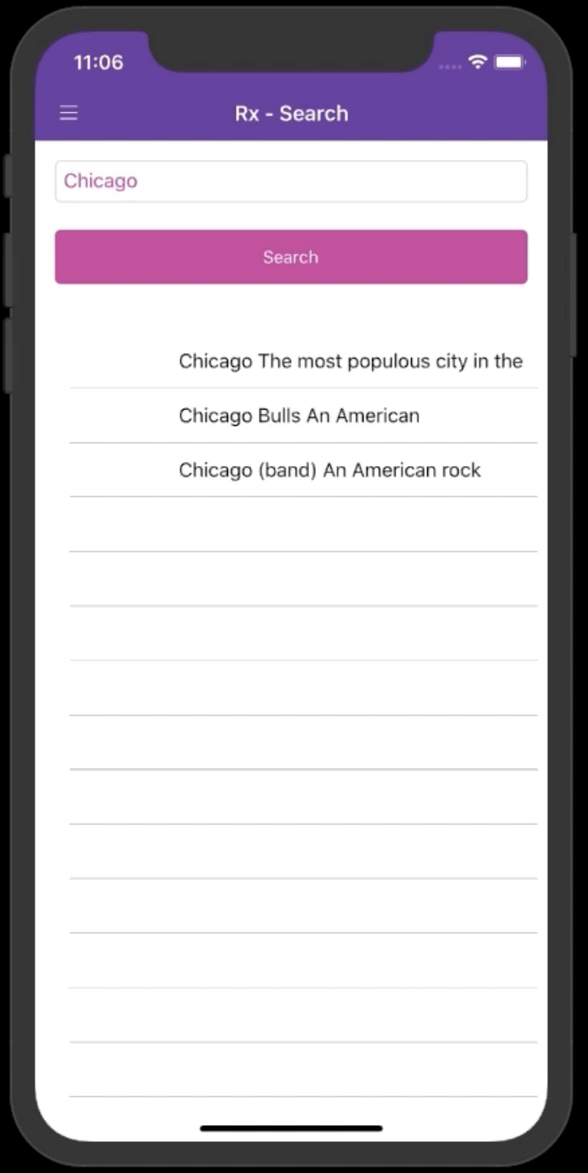
Buffer changes

Group Data

Transform Data

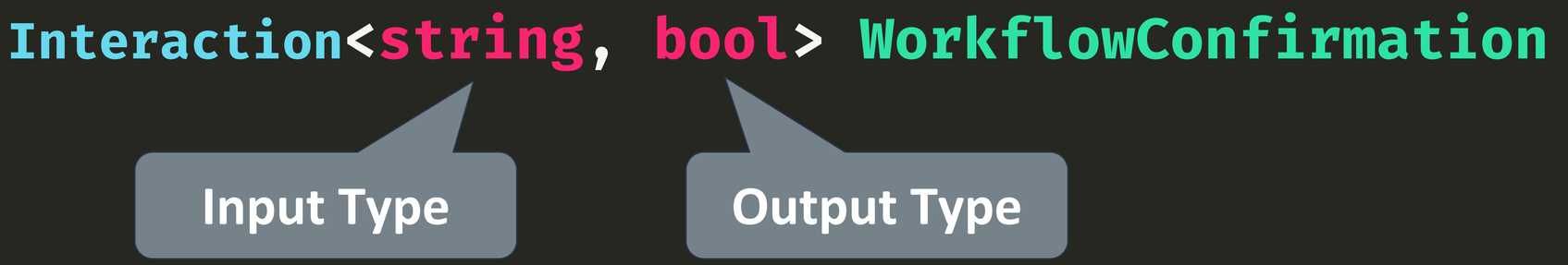
Efficiently Sort Results

Send Updates


11:06
Rx - Search
Chicago
Search
Chicago The most populous city in the
Chicago Bulls An American
Chicago (band) An American rock

Interactions

- Simple way for our VM to kick out to an outside process and get feedback
- Great for when you need user input from an alert or a picker
- Allows us to send and receive typed data
- Can be mocked out for unit tests


Interaction<string, bool> WorkflowConfirmation
Input Type
Output Type

```
ReactiveCommand
    .CreateFromTask(
        async () =>
        {
            var doesWantToWork =
                await WorkflowConfirmation
                    .Handle("Are you sure you want to work?");

            WorkStatus =
                 doesWantToWork
                    ? "I guess we can work, but I'm not happy about it"
                    : "Please don't scare us next time and just ";
        })
```

This is in the ViewModel and knows nothing about the implementation. So, this is nice and unit testable.

```
this.ViewModel
    .WorkflowConfirmation
    .RegisterHandler(
        async x =>
        {
            var result =
                await DisplayAlert("Work Status", x.Input, "yes", "no");

            x.SetOutput(result);
        })
```

Here is an example of a handler that is provided in a Xamarin.Forms page

```
this.ViewModel
    .WorkflowConfirmation
    .RegisterHandler(
        async x =>
        {
            var result =
                await DisplayAlert("Work Status", x.Input, "yes", "no");

            x.SetOutput(result);
        })
```

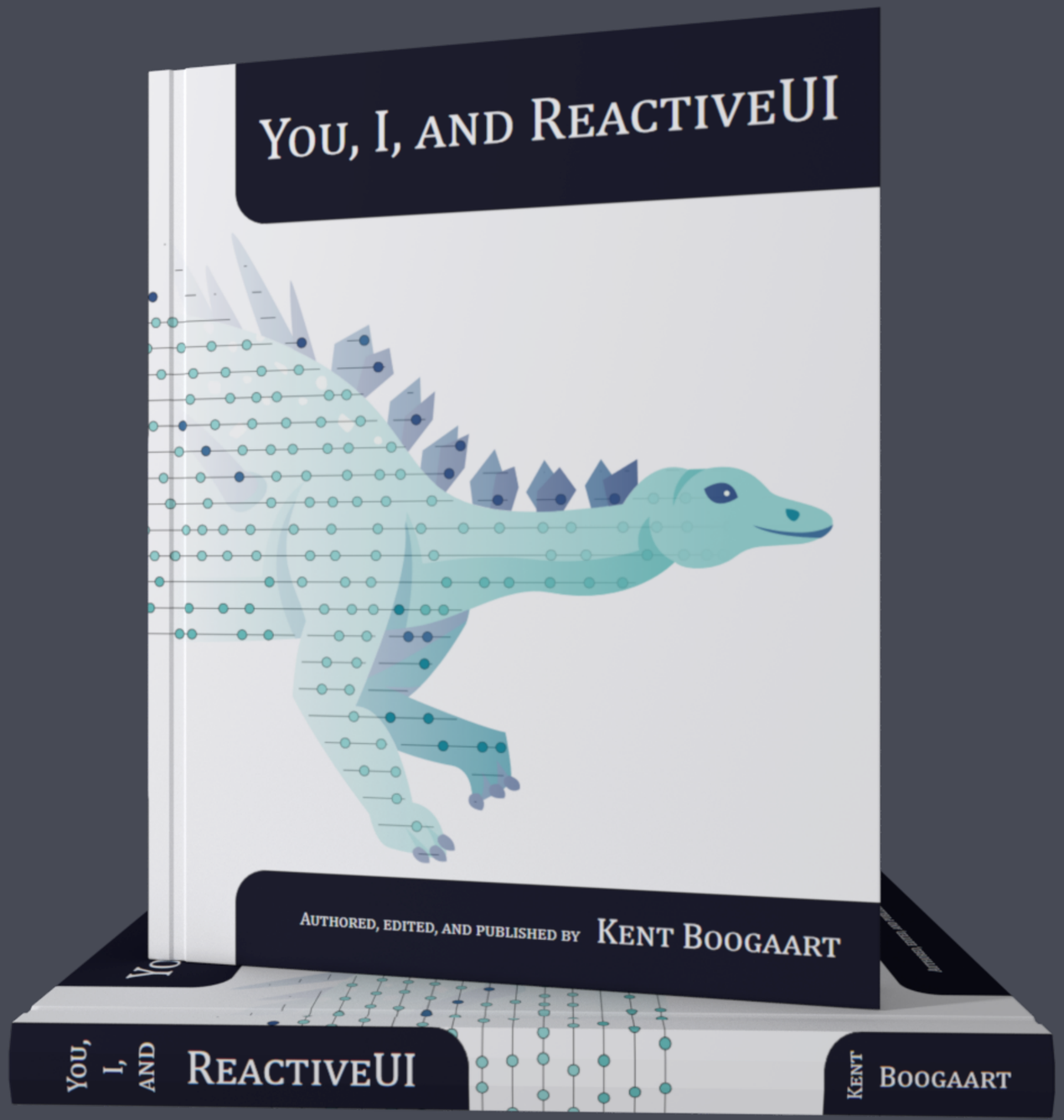
```
this.ViewModel
    .WorkflowConfirmation
    .RegisterHandler(
        async x =>
        {
            var result =
                await DisplayAlert("Work Status", x.Input, "yes", "no");

            x.SetOutput(result);
        })
```

We can call UI components and gather the output values we want from Dialogs, Cameras, File Pickers, etc.

```
this.ViewModel
    .WorkflowConfirmation
    .RegisterHandler(
        async x =>
        {
            var result =
                await D                                    ut, "yes", "no");

            x.SetOutput(result);
        })
```

We can send the data back to the ViewModel from when we are done

DEMO TIME


You, I, and ReactiveUI
Authored, edited, and published by Kent Boogaart
You, I, and ReactiveUI
Kent Boogaart

Glenn Watson

Rodney Littles II

Artyom Gorchakov

Colt Bauman

Roland Pheasant

Geoffrey Huntley

Kent Boogaart

Ani Betts

Michael Stonis

Eight-Bot, Inc. & Aurora Controls
Microsoft MVP

@MichaelStonis

ston.is
eight.bot
auroracontrols.app