

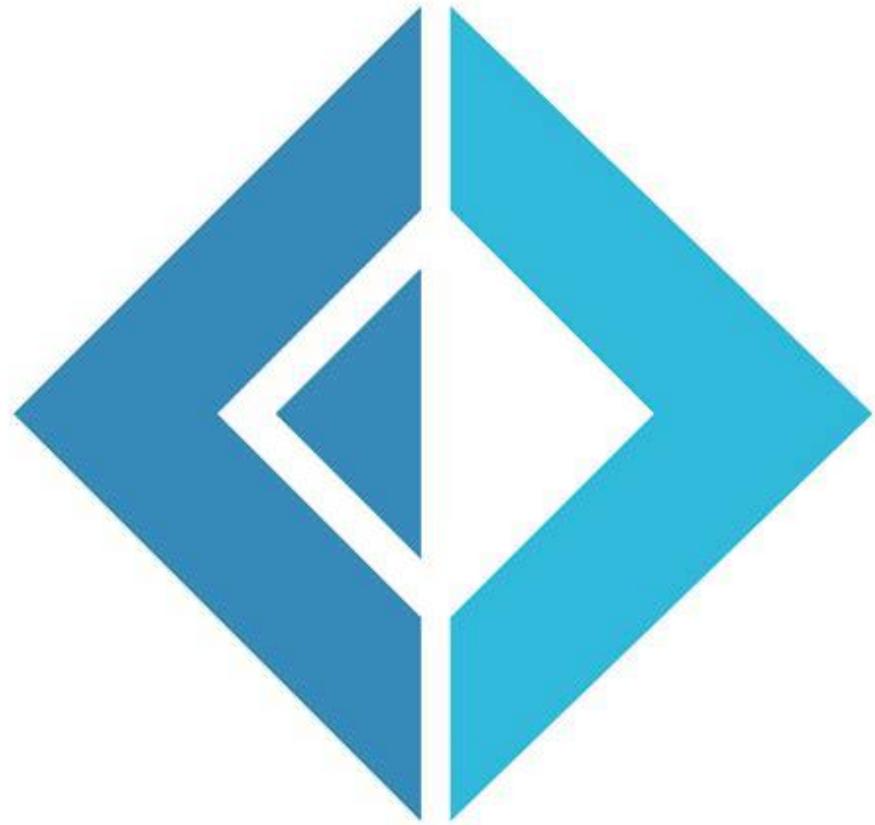
# Fabulous Functional Frontends

Mark Allibone

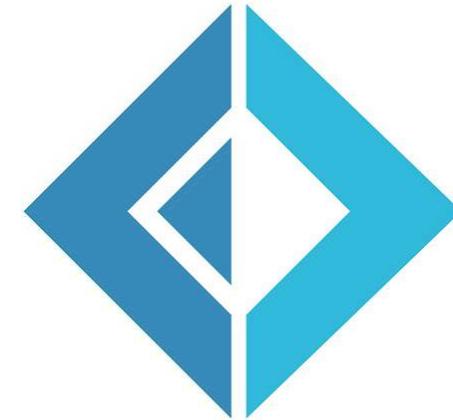
Head of Mobile

Noser Engineering AG

@mallibone

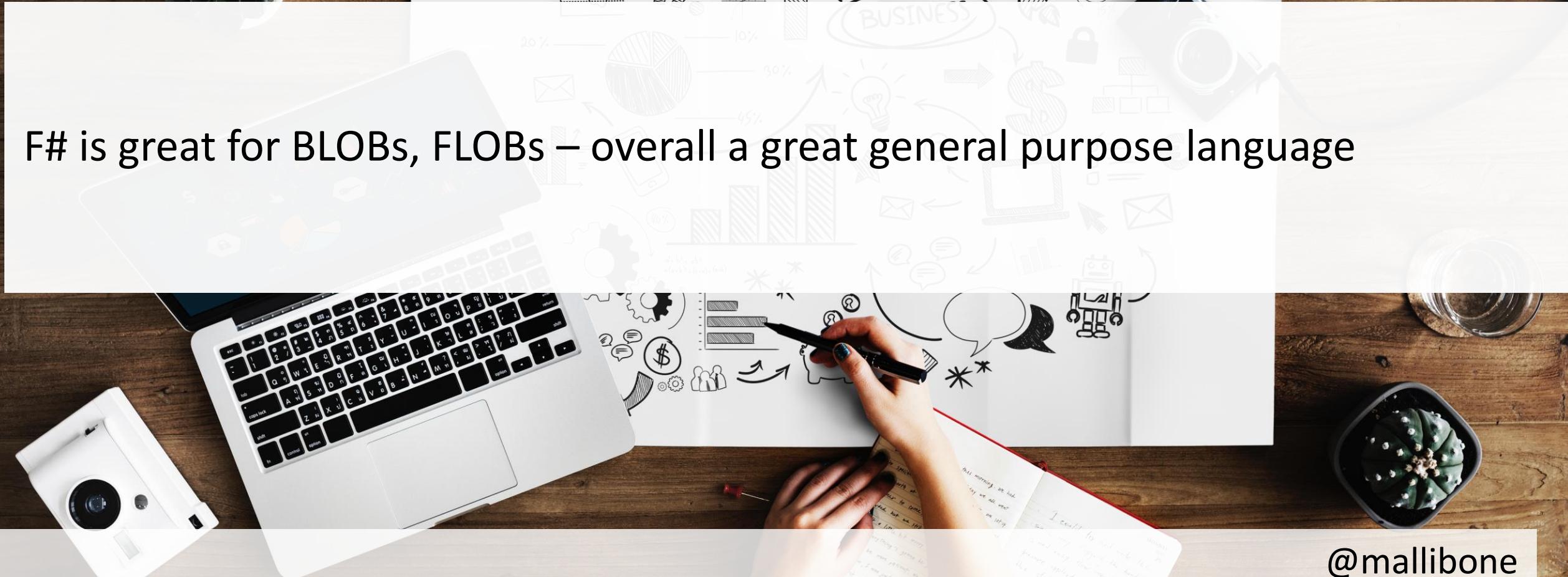


- No Nulls
- No mutable variables
- Data handling
- Declarative Programming
- Based on .Net





F# is great for BLOBs, FLOBs – overall a great general purpose language



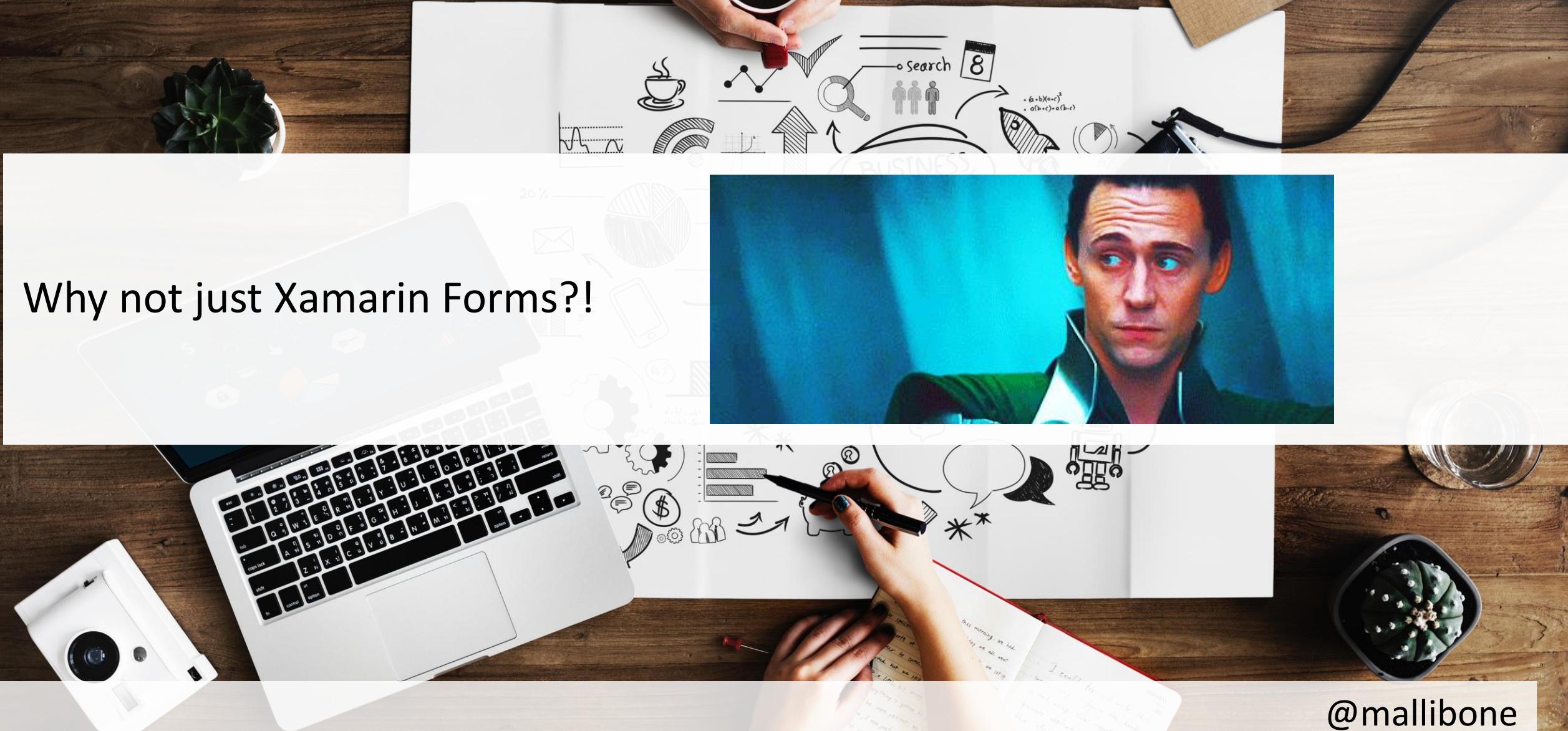
@mallibone



# Fabulous – a functional Wrapper for F# around Xamarin.Forms



@mallibone



Why not just Xamarin Forms?!

@mallibone

A screenshot of the Visual Studio 2019 IDE interface, showing a Xamarin.Forms project named "HelloFSharpConf".

The top navigation bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a Search bar.

The main code editor shows the file `AppPage.xaml.fs` with the following F# code:

```
1  namespace HelloFSharpConf
2
3  open Xamarin.Forms
4  open Xamarin.Forms.Xaml
5
6  type HelloFSharpConfPage() =
7      inherit ContentPage()
8      let _ = base.LoadFromXaml(typeof<HelloFSharpConfPage>)
9
10     do
11         base.BindingContext <- new MyViewModel()
12         ()
13
```

The Solution Explorer on the right lists the solution "HelloFSharpConf" (3 of 2 projects), containing the "HelloFSharpConf" project with files `Dependencies`, `AppViewModel.fs`, `AppPage.xaml`, `App.xaml`, and `AssemblyInfo.fs`, and the "HelloFSharpConf.Droid" project.

The Properties window for the "HelloFSharpConf.Droid" project shows the project file `HelloFSharpConf.Droid.csproj` located at `C:\Work\Talks\201906XamarinExp`.

The Output window at the bottom displays log messages from the Android emulator:

```
06-14 06:39:05.177 D/EGL_emulation( 4416): eglGetCurrent: 0xaa783250
06-14 06:39:05.288 D/Mono   ( 4416): DllImport searching in: '__Internal' ('(null)'). 
06-14 06:39:05.288 D/Mono   ( 4416): Searching for 'java_interop_jnienv_call_nonvirtual_float_method_a'.
06-14 06:39:05.288 D/Mono   ( 4416): Probing 'java_interop_jnienv_call_nonvirtual_float_method_a'.
06-14 06:39:05.288 D/Mono   ( 4416): Found as 'java_interop_jnienv_call_nonvirtual_float_method_a'.
06-14 06:39:05.297 D/EGL_emulation( 4416): eglGetCurrent: 0xaa783250
```

The status bar at the bottom indicates "This item does not support previewing".

<https://github.com/jimbobbennett/HelloFSharpConf>



Live Share



## Solution Explorer



Search Solution Explorer (Ctrl+ü)

✓ Solution 'HelloFSharpConf' (3 of 2 projects)

▲ ✓ HelloFSharpConf

▷ Dependencies

+ AppViewModel.fs

▷ AppPage.xaml

▷ App.xaml

+ AssemblyInfo.fs

▷ ✓ HelloFSharpConf.Droid

▷ ✓ HelloFSharpConf.iOS

Window Snip

A screenshot of the Visual Studio IDE interface, specifically showing the code editor for an F# project named "HelloFSharpConf".

The top navigation bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar labeled "Search Visual Studio (Ctrl+Q)". The title bar shows the project name "HelloFSharpConf" and the current configuration "my\_device (Android 8.1 - API 27)".

The code editor displays the file "AppPage.xaml.fs" with the following F# code:

```
1  namespace HelloFSharpConf
2
3      open Xamarin.Forms
4      open Xamarin.Forms.Xaml
5
6      type HelloFSharpConfPage() =
7          inherit ContentPage()
8          let _ = base.LoadFromXaml(typeof<HelloFSharpConfPage>)
9
10         do
11             base.BindingContext <- new MyViewModel()
12         ()
```

The code uses F# syntax to define a page class that inherits from ContentPage and loads its XAML definition. It then sets the BindingContext to a new instance of a MyViewModel class.

A screenshot of the Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar labeled "Search Visual Studio (Ctrl+Q)". A key icon is located next to the search bar. The title bar shows the project name "HelloFSharpConf" and the current configuration "my\_device (Android 8.1 - API 27)". The toolbar below the title bar contains icons for file operations like Open, Save, and Build.

The left sidebar features the "Toolbox" and "Test Explorer" panes. The main workspace displays an F# code editor with the following content:

```
21  namespace HelloFSharpConf
20
19  type MyViewModel () =
18      let ev = new Event<_,_>()
17      let mutable text = ""
16
15      let createCommand action =
14          let event1 = Event<_, _>()
13          {
12              new System.Windows.Input.ICommand with
11                  member this.CanExecute(obj) = true
10                  member this.Execute(obj) = action(obj)
9                      member this.add_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
8                      member this.remove_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
7
6
5          interface System.ComponentModel.INotifyPropertyChanged with
4              []
3                  member this.PropertyChanged = ev.Publish
2
1
1          member this.Text
2              with get() = text
1                  and set(value) = text <- value
2                      ev.Trigger(this, new System.ComponentModel.PropertyChangedEventArgs("Text"))
3
1          member this.TapCommand = createCommand (fun p -> this.Text <- "Hello Xamarin!")
4
5
```

A screenshot of the Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar labeled "Search Visual Studio (Ctrl+Q)". The title bar shows the project name "HelloFSharpConf" and the current configuration "my\_device (Android 8.1 - API 27)". The left sidebar features the "Toolbox" and "Test Explorer" panes. The main code editor displays an F# file named "AppViewModel.fs". The code defines a type "MyViewModel" with a mutable state "text" and implements the "INotifyPropertyChanged" interface. A red box highlights the declaration of "text". The cursor is positioned over the "get" accessor of the "Text" property, with the word "text" partially typed. The code editor also shows other files like "AndroidManifest.xml" and "AppPage.xaml.fs" in the tabs.

```
21  namespace HelloFSharpConf
20
19  type MyViewModel () =
18      let ev = new Event<_, _>()
17      let mutable text = ""
16
15      let createCommand action =
14          let event1 = Event<_, _>()
13          {
12              new System.Windows.Input.ICommand with
11                  member this.CanExecute(obj) = true
10                  member this.Execute(obj) = action(obj)
9                      member this.add_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
8                      member this.remove_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
7
6
5      interface System.ComponentModel.INotifyPropertyChanged with
4          []
3              member this.PropertyChanged = ev.Publish
2
1
1      member this.Text
2          with get() = text
1          and set(value) = text <- value
2              ev.Trigger(this, new System.ComponentModel.PropertyChangedEventArgs("Text"))
3
1      member this.TapCommand = createCommand (fun p -> this.Text <- "Hello Xamarin!")
4
5
```

A screenshot of the Visual Studio IDE interface. The top menu bar includes File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, and a search bar for "Search Visual Studio (Ctrl+Q)". The title bar shows the project name "HelloFSharpConf" and the current configuration "my\_device (Android 8.1 - API 27)". The left sidebar features the "Toolbox" and "Test Explorer" panes. The main code editor displays an F# file named "AppViewModel.fs". The code defines a type `MyViewModel` with properties `text` and `ev`, and a command `createCommand`. A red rectangle highlights the implementation of `createCommand`. The code also implements the `INotifyPropertyChanged` interface, updating the `Text` property and triggering a `PropertyChanged` event. The code editor uses color-coded syntax highlighting for F# keywords and types.

```
21  namespace HelloFSharpConf
20
19  type MyViewModel () =
18      let ev = new Event<_,_>()
17      let mutable text = ""
16
15      let createCommand action =
14          let event1 = Event<_, _>()
13          {
12              new System.Windows.Input.ICommand with
11                  member this.CanExecute(obj) = true
10                  member this.Execute(obj) = action(obj)
9                      member this.add_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
8                      member this.remove_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
7
6
5      interface System.ComponentModel.INotifyPropertyChanged with
4          []
3          member this.PropertyChanged = ev.Publish
2
1
1      member this.Text
2          with get() = text
1          and set(value) = text <- value
2              ev.Trigger(this, new System.ComponentModel.PropertyChangedEventArgs("Text"))
3
1      member this.TapCommand = createCommand (fun p -> this.Text <- "Hello Xamarin!")
4
5
```

The screenshot shows the Visual Studio IDE interface with the following details:

- Top Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help, Search Visual Studio (Ctrl+Q), HelloFSharpConf.
- Solution Explorer:** Shows a solution named "HelloFSharpConf.Droid" with a file "my\_device (Android 8.1 - API 27)".
- Toolbox:** Standard Visual Studio toolbox items.
- Code Editor:** The "AppViewModel.fs" file is open, showing F# code for a view model. A red rectangle highlights the implementation of the `Text` property and its associated logic.
- Code Preview:** A preview pane on the right shows the resulting XAML or UI representation of the code.

```
21  namespace HelloFSharpConf
20
19  type MyViewModel () =
18      let ev = new Event<_,_>()
17      let mutable text = ""
16
15      let createCommand action =
14          let event1 = Event<_, _>()
13          {
12              new System.Windows.Input.ICommand with
11                  member this.CanExecute(obj) = true
10                  member this.Execute(obj) = action(obj)
9                      member this.add_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
8                      member this.remove_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
7
6
5      interface System.ComponentModel.INotifyPropertyChanged with
4          []
3              member this.PropertyChanged = ev.Publish
2
1
1      member this.Text
2          with get() = text
1          and set(value) = text <- value
2              ev.Trigger(this, new System.ComponentModel.PropertyChangedEventArgs("Text"))
3      member this.TapCommand = createCommand (fun p -> this.Text <- "Hello Xamarin!")
4
5
```

A screenshot of the Visual Studio IDE interface, showing a project named "HelloFSharpConf" with two projects: "HelloFSharpConf" and "HelloFSharpConf.Droid". The main window displays the F# code for "AppViewModel.fs". The code defines a type `MyViewModel` with properties `text` and `TapCommand`, and implements `INotifyPropertyChanged` with a `PropertyChanged` event. The `Text` property uses a mutable binding. The `TapCommand` is a command that triggers a tap event. The Solution Explorer shows the project structure, and the Properties window is visible.

```
1  namespace HelloFSharpConf
2
3  type MyViewModel () =
4      let ev = new Event<_, _>()
5      let mutable text = ""
6
7      let createCommand action =
8          let event1 = Event<_, _>()
9          {
10              new System.Windows.Input.ICommand with
11                  member this.CanExecute(obj) = true
12                  member this.Execute(obj) = action(obj)
13                  member this.add_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
14                  member this.remove_CanExecuteChanged(handler) = event1.Publish.AddHandler(handler)
15          }
16
17      interface System.ComponentModel.INotifyPropertyChanged with
18          [<<<CLIEvent>>]
19          member this.PropertyChanged = ev.Publish
20
21      member this.Text
22          with get() = text
23          and set(value) = text <- value
24          ev.Trigger(this, new System.ComponentModel.PropertyChangedEventArgs("Text"))
25      member this.TapCommand = createCommand (fun p -> this.Text <- "Hello Xamarin!")
26
27
```

Output window:

```
06-14 06:39:05.177 D/EGL_emulation( 4416): eglGetCurrent: 0xa0a785120: ver 3 0 (tinfo 0xa0a783250)
06-14 06:39:05.288 D/Mono   ( 4416): DllImport searching in: '_Internal' ('(null)').
```



F# can do Object Oriented programming but it is functional first



@mallibone

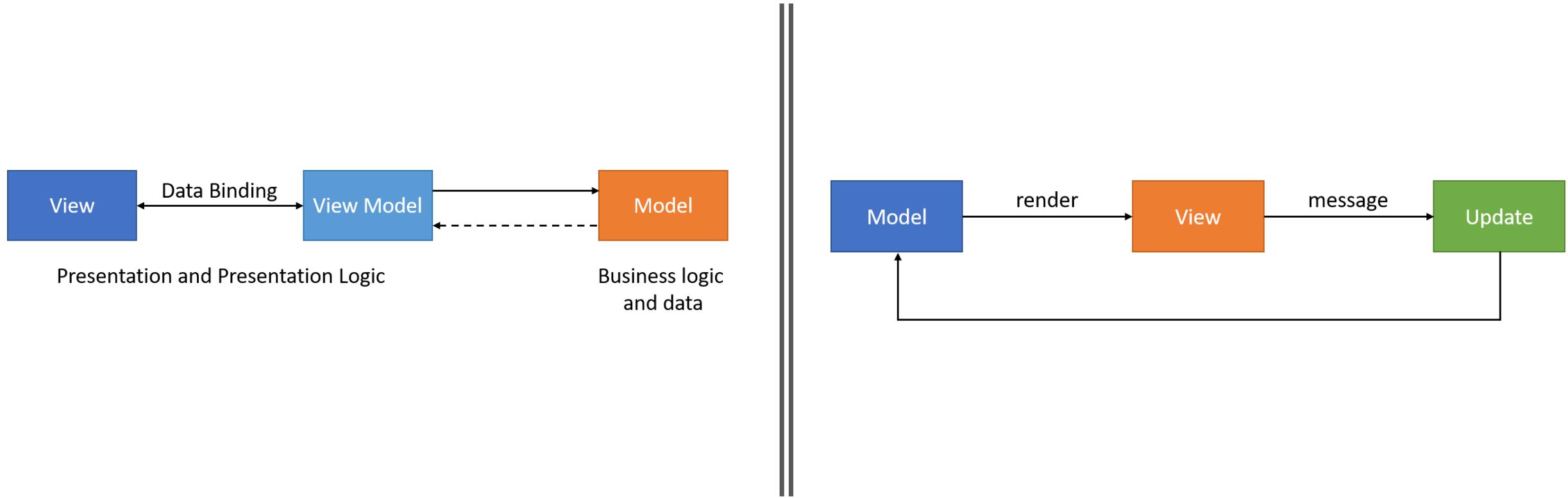


# MVU the functional MV\* View Pattern



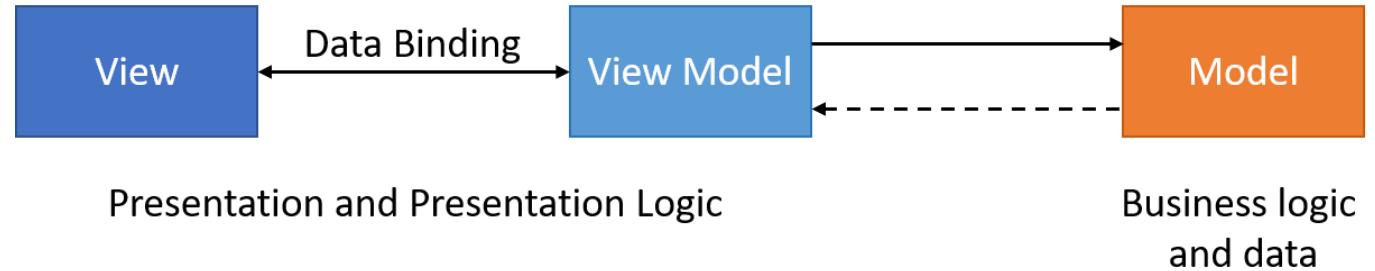
@mallibone

# MVVM vs MVU

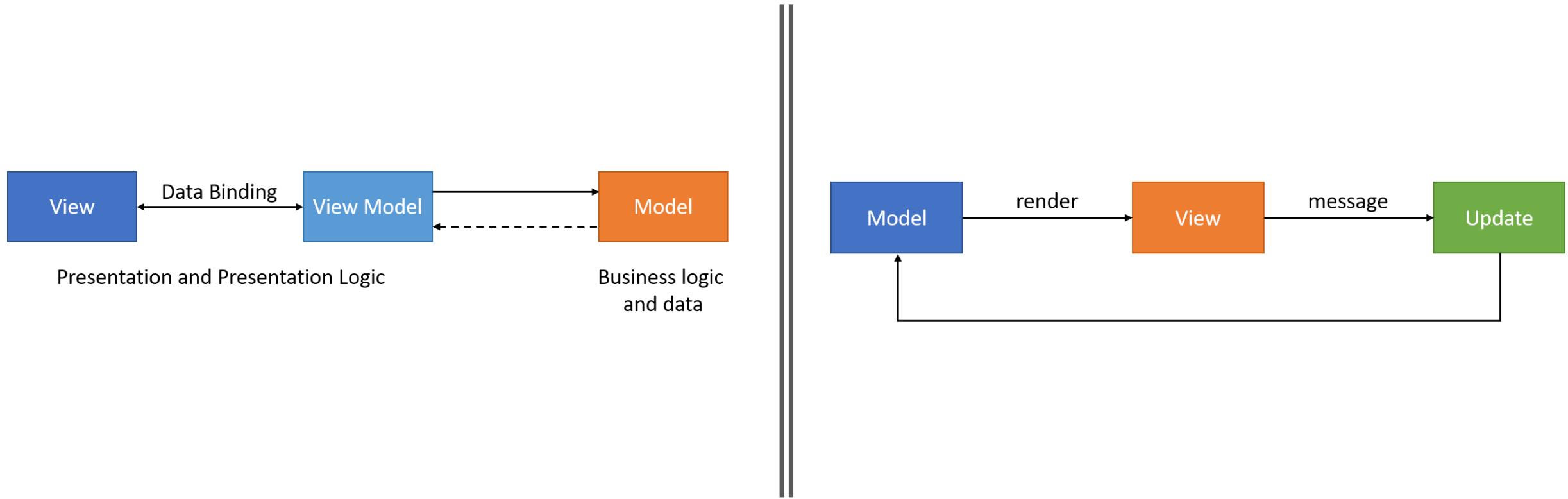


# Model View View Model

---

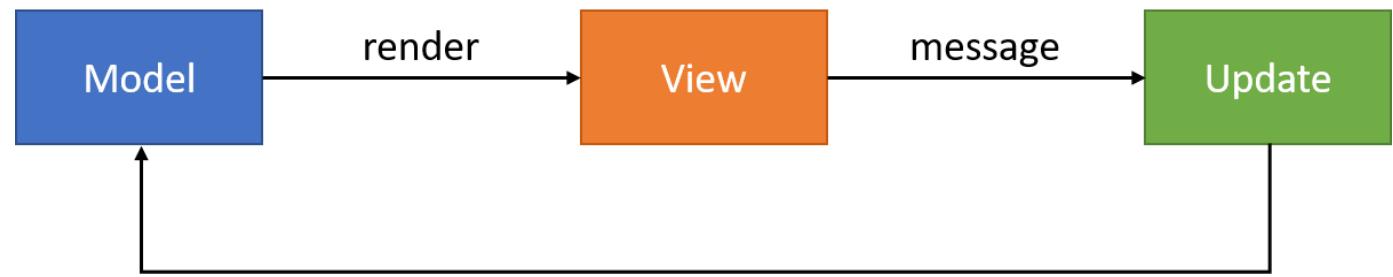


# MVVM vs MVU



# Model View Update

---



A close-up photograph of a fluffy, pinkish-orange chick, likely a penguin or albatross chick, with a large, hooked black beak. The chick is looking down and to its right. The background is a soft-focus, warm-toned landscape.

NICOTINEBATCH | TUMBLR

**FABULOUS**

KBS  
KBS TV

Functional Frontends

## Getting started

1. Install Visual Studio or Visual Studio for Mac and enable both Xamarin and .NET Core support.
2. Install the template pack:

```
dotnet new -i Fabulous.Templates
```

Layout

C:\Work\Playground ~ PowerShell 5.1.17134.407 64-bit (3752)

Reset

C:\Work\Playground> **dotnet new fabulous-app -n Gnabber**

The template "Fabulous App" was created successfully.

Processing post-creation actions...

Description: Opens NewApp.fs in the editor.

Manual instructions:

C:\Work\Playground>

Gnabber - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test ReSharper R Tools Analyze Window Help

iPhoneSimulator Gnabber.iOS Simulator

Gnabber.fs # x

```
open Xamarin.Forms

module App =
    type Model =
        { Count : int
          Step : int
          TimerOn: bool }

    type Msg =
        | Increment
        | Decrement
        | Reset
        | SetStep of int
        | TimerToggled of bool
        | TimedTick

    let initModel = { Count = 0; Step = 1; TimerOn=false }

    let init () = initModel, Cmd.none

    let timerCmd = ...

    let update msg model =
        match msg with...

    let view (model: Model) dispatch =
        View.ContentPage(
            content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))

    // Note, this declaration is needed if you enable LiveUpdate
    let program = Program.mkProgram init update view

type App () as app =
    inherit Application ()

    let runner =
        App.program
    #if DEBUG
    |> Program.withConsoleTrace
    #endif
    |> Program.runWithDynamicView app

    #if DEBUG
    |> Program.enableLiveUpdate()
    #endif

    // Uncomment this code to save the application state to app.Properties using Newtonsoft.Json
    // See https://fsprojects.github.io/Fabulous/tools.html for further instructions.

```

110 %

Error List

Entire Solution 0 Errors 0 Warnings 0 Messages Build + IntelliSense

Solution Explorer

Search Solution Explorer (Ctrl+U)

Solution 'Gnabber' (3 projects)

- GNabber
- Dependencies
- NuGet
- SDK

F# Gnabber.fs

Gnabber.Android

Gnabber.iOS

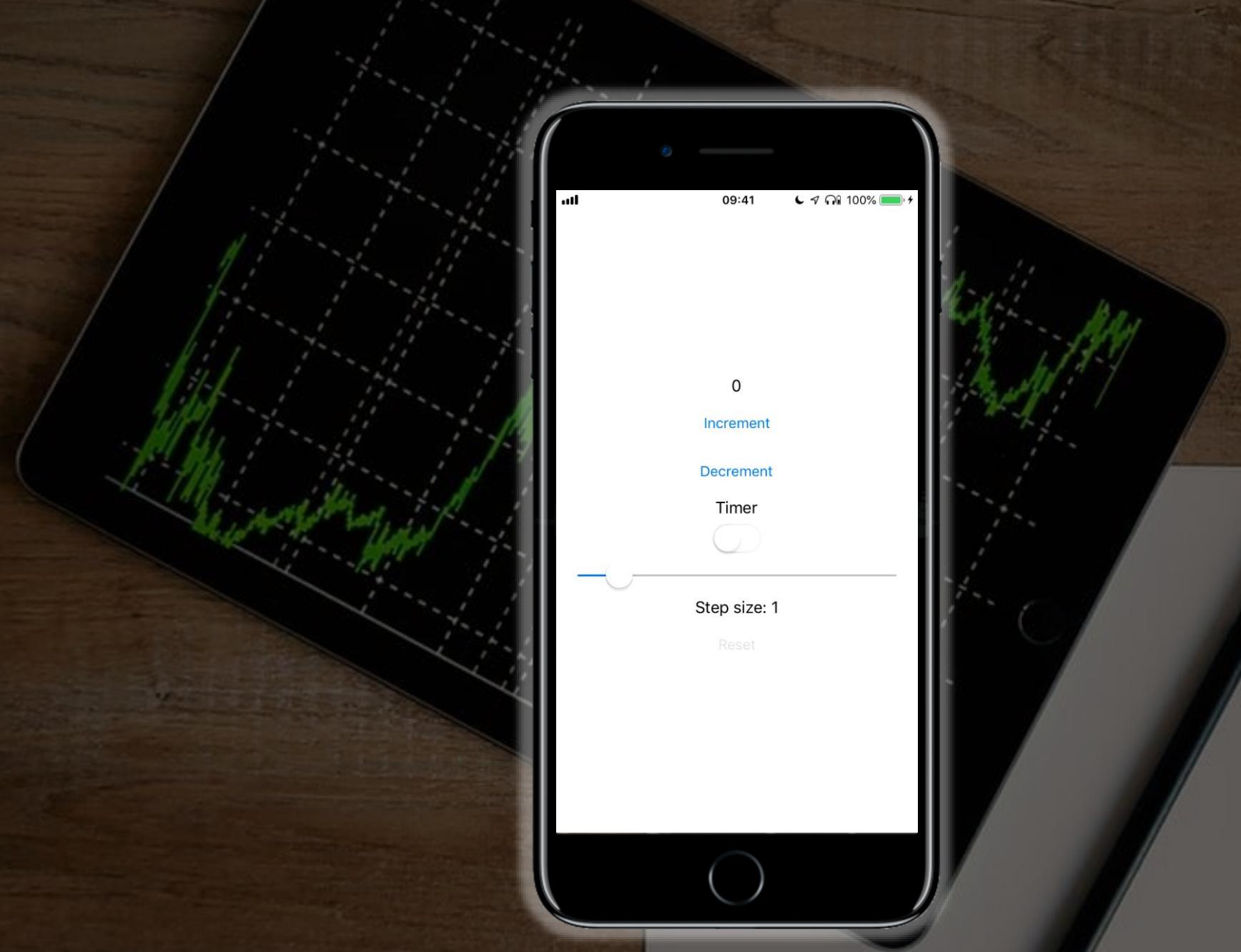
Solution Explorer Team Explorer

Properties

Ready Data Tools Operations Package Manager Console Error List Output F# Interactive

Ln 37 Col 26 Ch 57 INS

Add to Source Control



@mallibone

Gnabber - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test ReSharper R Tools Analyze Window Help

iPhoneSimulator Gnabber.iOS Simulator

Gnabber.fs # x

```
open Xamarin.Forms

module App =
    type Model =
        { Count : int
          Step : int
          TimerOn: bool }

    type Msg =
        | Increment
        | Decrement
        | Reset
        | SetStep of int
        | TimerToggled of bool
        | TimedTick

    let initModel = { Count = 0; Step = 1; TimerOn=false }

    let init () = initModel, Cmd.none

    let timerCmd = ...

    let update msg model =
        match msg with...

    let view (model: Model) dispatch =
        View.ContentPage(
            content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))

    // Note, this declaration is needed if you enable LiveUpdate
    let program = Program.mkProgram init update view

type App () as app =
    inherit Application ()

    let runner =
        App.program
        #if DEBUG
        |> Program.withConsoleTrace
        #endif
        |> Program.runWithDynamicView app

    #if DEBUG
    // Uncomment this line to enable live update in debug mode.
    // See https://fsprojects.github.io/Fabulous/tools.html for further instructions.
    //do runner.EnableLiveUpdate()
    #endif

    // Uncomment this code to save the application state to app.Properties using Newtonsoft.Json
    // See https://fsprojects.github.io/Fabulous/models.html for further instructions.
```

110 %

Error List

Entire Solution 0 Errors 0 Warnings 0 Messages Build + IntelliSense

Solution Explorer

Search Solution Explorer (Ctrl+U)

Solution 'Gnabber' (3 projects)

- GNabber
- Dependencies
- NuGet
- SDK

F# Gnabber.fs

Gnabber.Android

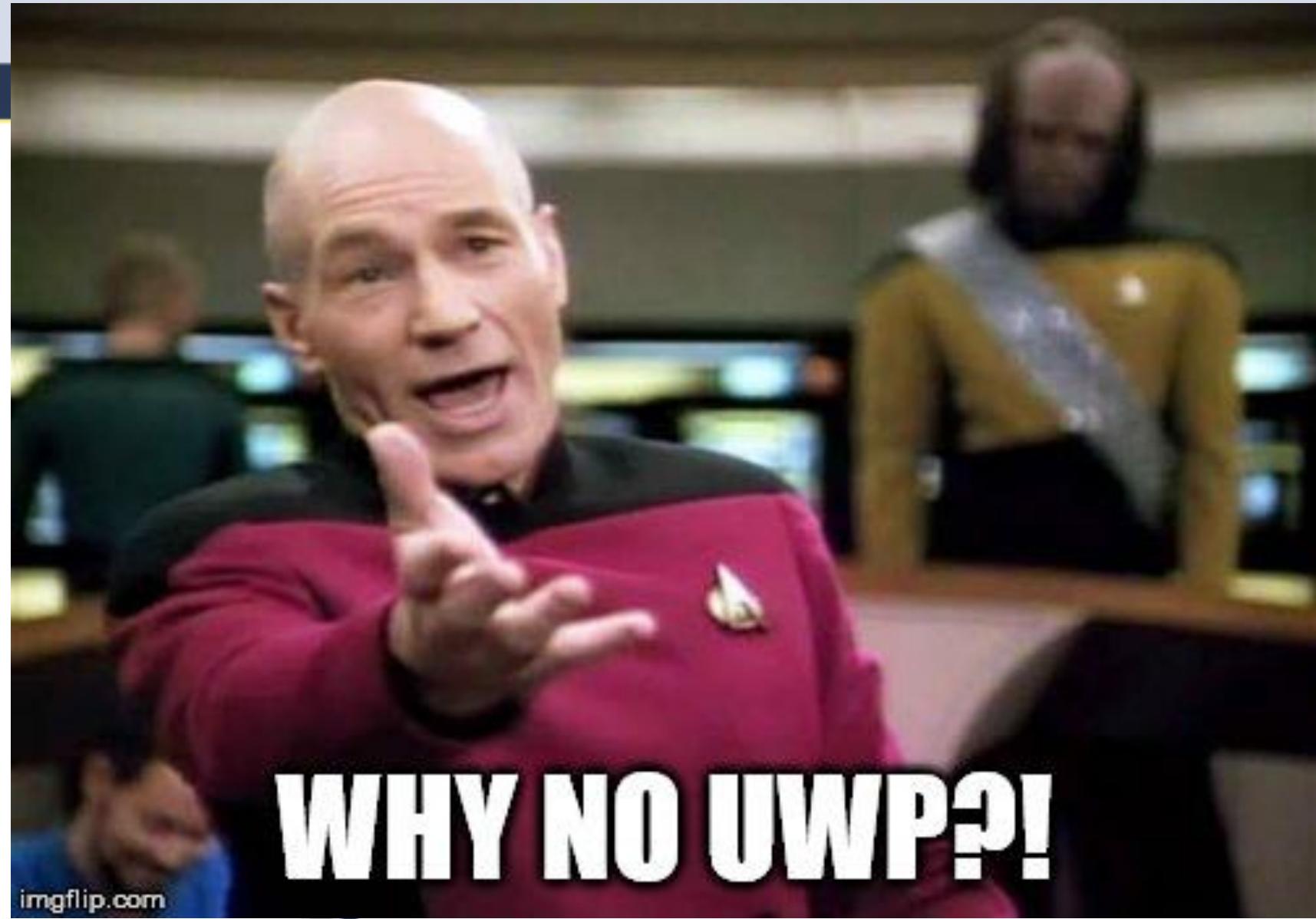
Gnabber.iOS

Properties

Ready Data Tools Operations Package Manager Console Error List Output F# Interactive

Ln 37 Col 26 Ch 57 INS

Add to Source Control





Gnabber.fs

```
7 23 open Xamarin.Forms
8 22
9 21 module App =
10 20 |> type Model =
11 19 |>| Count : int
12 18 |>| Step : int
13 17 |>| TimerOn: bool }
14 16
15 15 |> type Msg =
16 14 |>| Increment
17 13 |>| Decrement
18 12 |>| Reset
19 11 |>| SetStep of int
20 10 |>| TimerToggled of bool
21 9 |>| TimedTick
22 8
23 7 let initModel = { Count = 0; Step = 1; TimerOn=false }
24 6
25 5 let init () = initModel, Cmd.none
26 4
27 3 let timerCmd = ...
28 2
29 1 let update msg model =
30 0 |> match msg with...
31 1
32 2 let view (model: Model) dispatch =
33 3 View.ContentPage(
34 4 |> content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))
35 5
36 6 // Note, this declaration is needed if you enable LiveUpdate
37 7 let program = Program.mkProgram init update view
38 8
39 9 type App () as app =
```



Gnabber.fs

```
7 23 [open Xamarin.Forms
8 22
9 21 [module App =
10 20 | type Model =
11 19 | { Count : int
12 18 | Step : int
13 17 | TimerOn: bool }
14 16
15 15 [type Msg =
16 14 | Increment
17 13 | Decrement
18 12 | Reset
19 11 | SetStep of int
20 10 | TimerToggled of bool
21 9 | TimedTick
22 8
23 7 let initModel = { Count = 0; Step = 1; TimerOn=false }
24 6
25 5 let init () = initModel, Cmd.none
26 4
27 3 [let timerCmd = ...
28 2
29 1 [let update msg model =
30 0 | [match msg with...]
31 2
32 1 [let view (model: Model) dispatch =
33 0 | View.ContentPage(
34 1 | content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))
35 5
36 6 // Note, this declaration is needed if you enable LiveUpdate
37 7 let program = Program.mkProgram init update view
38 8
39 9 [type App () as app =
```



Gnabber.fs

```
7 23 open Xamarin.Forms
8 22
9 21 module App =
10 20 | type Model =
11 19 | { Count : int
12 18 | Step : int
13 17 | TimerOn: bool }
14 16
15 15 | type Msg =
16 14 |     Increment
17 13 |     Decrement
18 12 |     Reset
19 11 |     SetStep of int
20 10 |     TimerToggled of bool
21 9 |     TimedTick
22 8
23 7 let initModel = { Count = 0; Step = 1; TimerOn=false }
24 6
25 5 let init () = initModel, Cmd.none
26 4
27 3 let timerCmd = ...
28 2
29 1 let update msg model =
30 0 | match msg with...
31 1
32 1 let view (model: Model) dispatch =
33 0 |     View.ContentPage(
34 1 |         content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))
35 0
36 1 // Note, this declaration is needed if you enable LiveUpdate
37 0 let program = Program.mkProgram init update view
38 1
39 0 type App () as app =
```



Gnabber.fs

```
7 23 open Xamarin.Forms
8 22
9 21 module App =
10 20 |> type Model =
11 19 |>| Count : int
12 18 |>| Step : int
13 17 |>| TimerOn: bool }
14 16
15 15 |> type Msg =
16 14 |>| Increment
17 13 |>| Decrement
18 12 |>| Reset
19 11 |>| SetStep of int
20 10 |>| TimerToggled of bool
21 9 |>| TimedTick
22 8
23 7 let initModel = { Count = 0; Step = 1; TimerOn=false }
24 6
25 5 let init () = initModel, Cmd.none
26 4
27 3 let timerCmd = ...
28 2
29 1 let update msg model =
30 0 |> match msg with...
31 1
32 0 let view (model: Model) dispatch =
33 1 |> View.ContentPage(
34 0 |> content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))
35 1
36 0 // Note, this declaration is needed if you enable LiveUpdate
37 1 let program = Program.mkProgram init update view
38 0
39 1 type App () as app =
```

Gnabber.fs

```
7 23 open Xamarin.Forms
8 22
9 21 module App =
10 20 |> type Model =
11 19 |> { Count : int
12 18 |> Step : int
13 17 |> TimerOn: bool }
14 16
15 15 |> type Msg =
16 14 |> Increment
17 13 |> Decrement
18 12 |> Reset
19 11 |> SetStep of int
20 10 |> TimerToggled of bool
21 9 |> TimedTick
22 8
23 7 let initModel = { Count = 0; Step = 1; TimerOn=false }
24 6
25 5 let init () = initModel, Cmd.none
26 4
27 3 let timerCmd = ...
28 2
29 1 let update msg model =
30 0 |> match msg with ...
31 2
32 1 let view (model: Model) dispatch =
33 0 |> View.ContentPage(
34 1 |> content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))
35 5
36 4 // Note, this declaration is needed if you enable LiveUpdate
37 3 let program = Program.mkProgram init update view
38 2
39 1 type App () as app =
```



Gnabber.fs

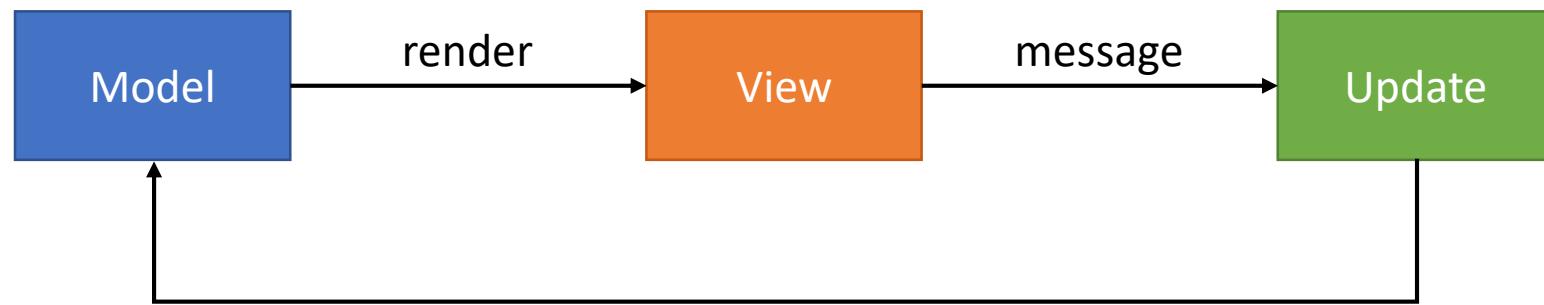
```
7 23 [open Xamarin.Forms
8 22
9 21 [module App =
10 20 | type Model =
11 19 | { Count : int
12 18 | Step : int
13 17 | TimerOn: bool }
14 16
15 15 [type Msg =
16 14 | Increment
17 13 | Decrement
18 12 | Reset
19 11 | SetStep of int
20 10 | TimerToggled of bool
21 9 | TimedTick
22 8
23 7 let initModel = { Count = 0; Step = 1; TimerOn=false }
24 6
25 5 let init () = initModel, Cmd.none
26 4
27 3 [let timerCmd = ...]
31 2
32 1 [let update msg model =
33 0 | match msg with...|]
44 1
45 2 [let view (model: Model) dispatch =
46 3 | View.ContentPage(
47 4 | | content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))
58 5
59 6 // Note, this declaration is needed if you enable LiveUpdate
60 7 let program = Program.mkProgram init update view
61 8
62 9 [type App () as app =
```

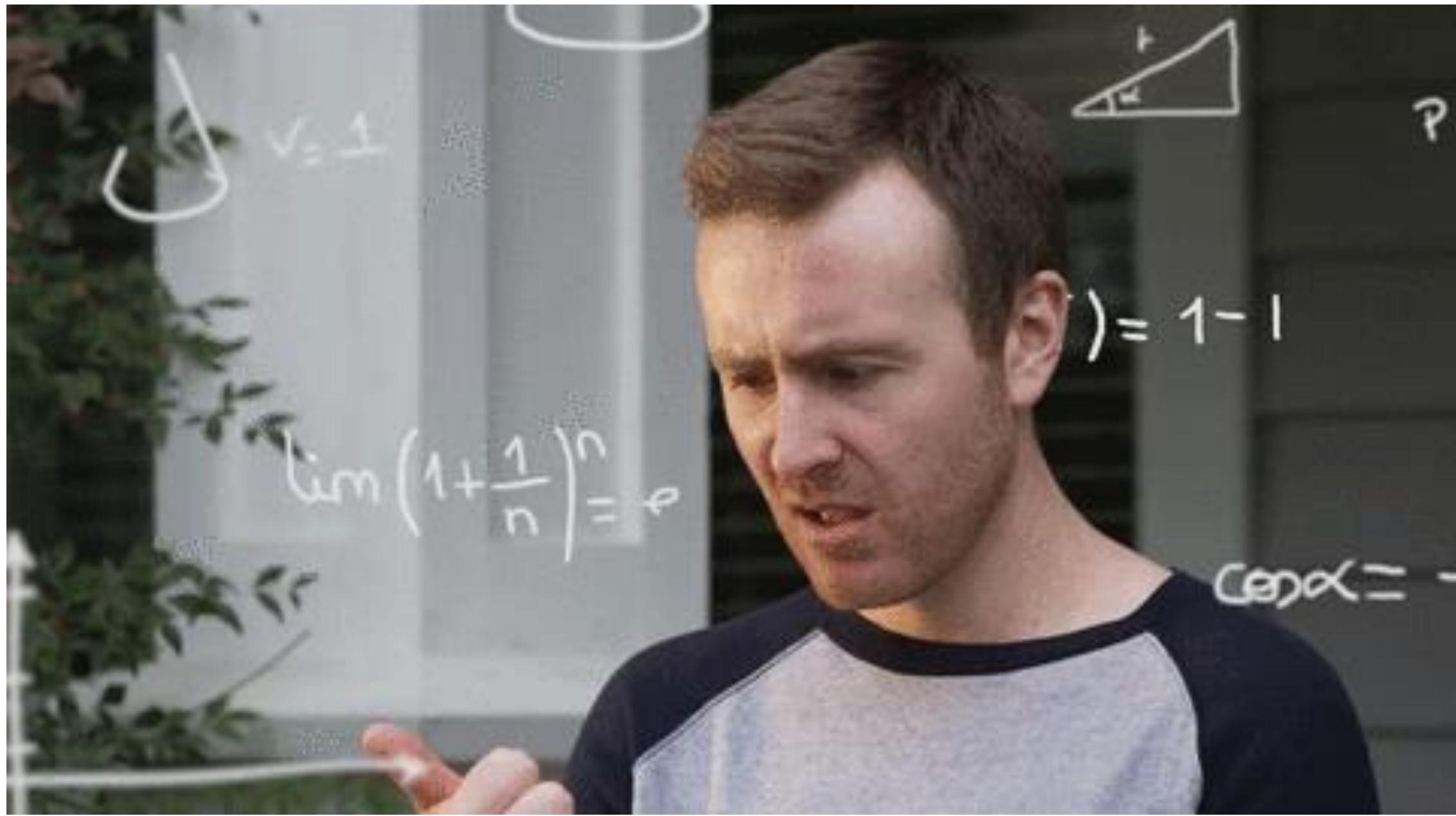
# DEMO



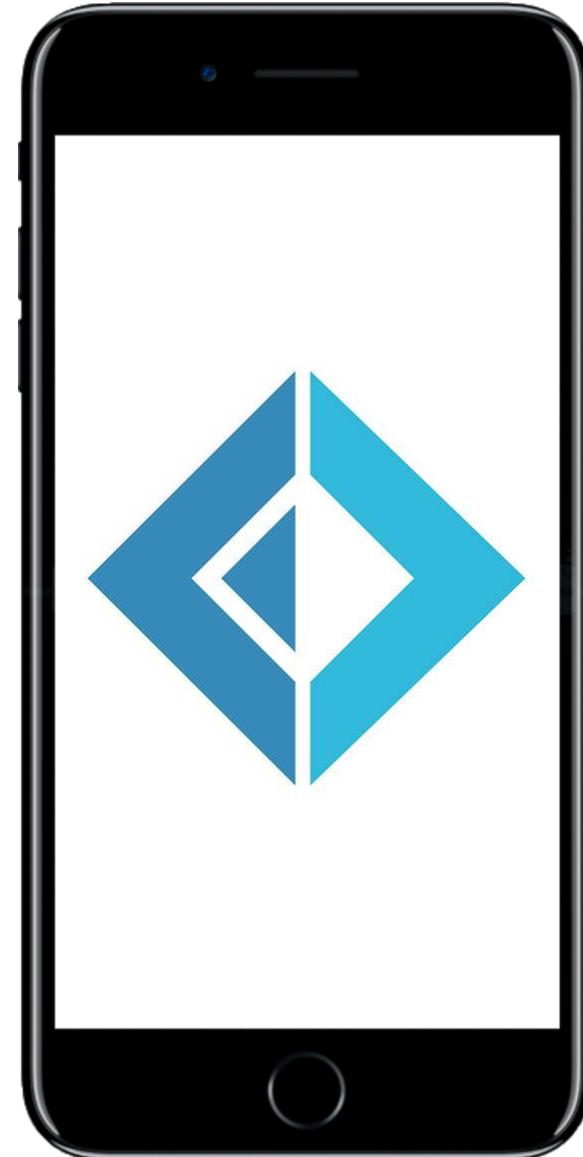
@mallibone

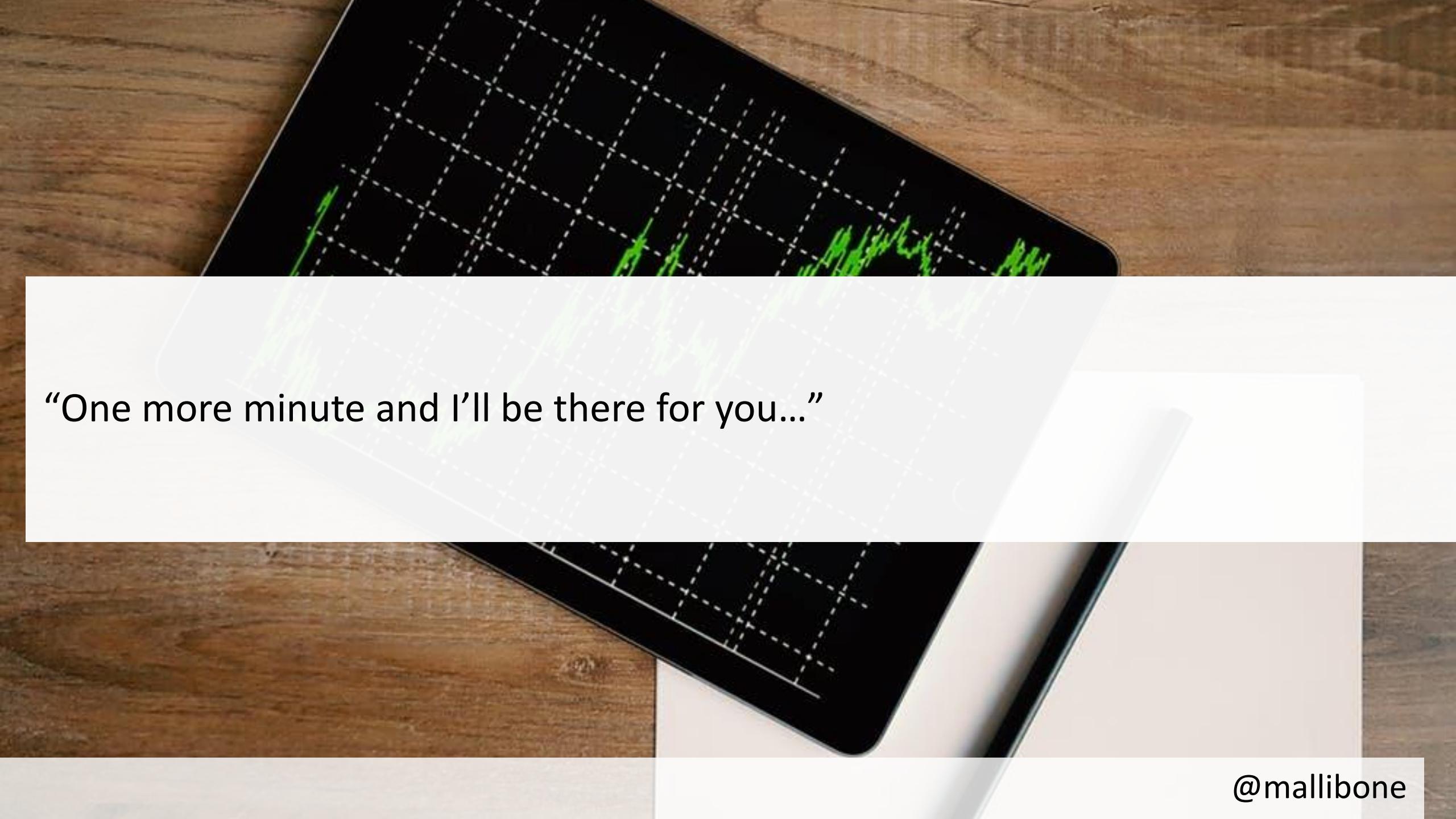
# Model View Update





- Driven from the model not the view
- States changes are defined with messages
- Single place for change in the update method
- Based on Xamarin Forms





“One more minute and I’ll be there for you...”



Commands are for Dispatching Work

DEMO



@mallibone



FASCINATING

NOW MAKE IT PRETTY

imgflip.com

@mallibone



CSS



Doing it with style

@mallibone





The background of the slide features a vibrant, abstract cloud of ink or paint suspended in water against a black background. The ink forms large, billowing shapes in shades of orange, yellow, red, and purple, creating a dynamic and organic visual texture.

Demo

@mallibone



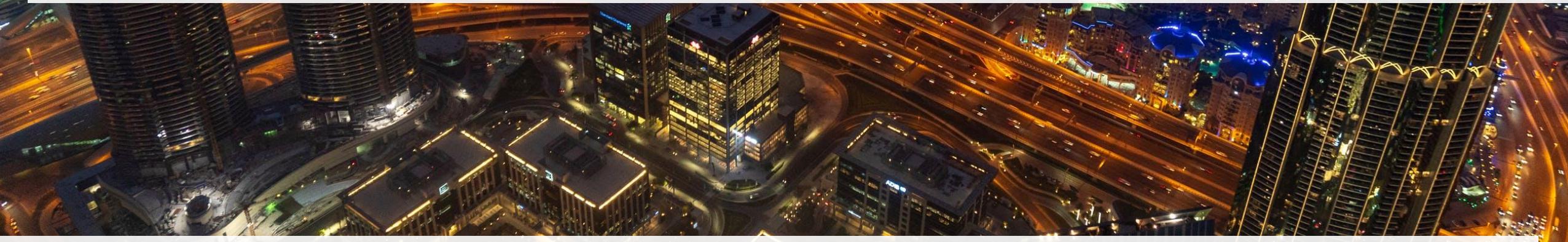
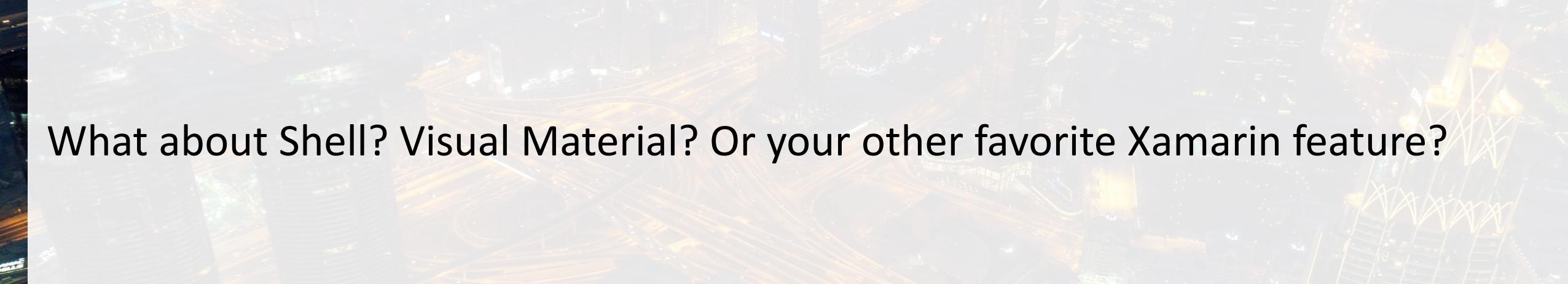
Live Update ❤️ Coded UI

<https://fsprojects.github.io/Fabulous/tools.html>

@mallibone



What about Shell? Visual Material? Or your other favorite Xamarin feature?



@mallibone



NuGet

@mallibone

Xamarin.Essentials - Xamarin | M + ...

https://docs.microsoft.com/en-us/xamarin/essentials/ NoserEngineering

Microsoft | Xamarin Getting Started Android iOS Mac Xamarin.Forms Samples APIs All Microsoft Search

Docs / Xamarin / Xamarin.Android / Xamarin.Essentials

Filter by title

Xamarin.Android

Get Started

Get Started

Setup and Installation

Hello, Android

Hello, Android Multiscreen

Xamarin for Java Developers

Application Fundamentals

User Interface

Platform Features

Xamarin.Essentials

Xamarin.Essentials

Getting Started

Accelerometer

App Information

Barometer

Battery

Clipboard

Color Converters

Compass

Connectivity

Detect Shake

Device Display Information

Device Information

Email

File System Helpers

Flashlight

Geocoding

Geolocation

Gyroscope

Launcher

Magnetometer

Main Thread

Download PDF

# Xamarin.Essentials

04/22/2019 • 2 minutes to read • Contributors

Xamarin.Essentials provides developers with cross-platform APIs for their mobile applications.

Android, iOS, and UWP offer unique operating system and platform APIs that developers have access to all in C# leveraging Xamarin. Xamarin.Essentials provides a single cross-platform API that works with any Xamarin.Forms, Android, iOS, or UWP application that can be accessed from shared code no matter how the user interface is created.

## Get Started with Xamarin.Essentials

Follow the [getting started guide](#) to install the `Xamarin.Essentials` NuGet package into your existing or new Xamarin.Forms, Android, iOS, or UWP projects.

## Feature Guides

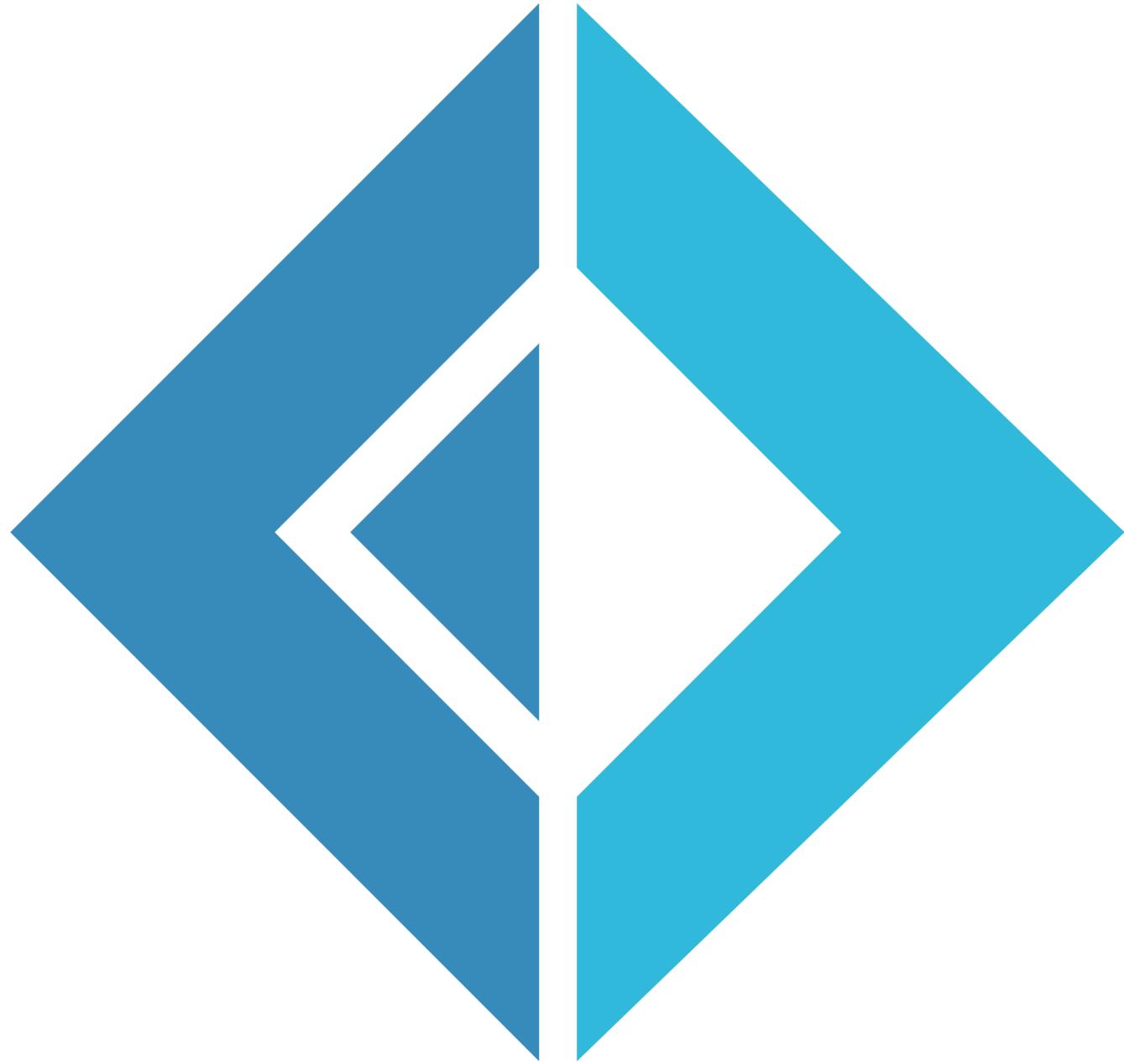
Follow the guides to integrate these Xamarin.Essentials features into your applications:

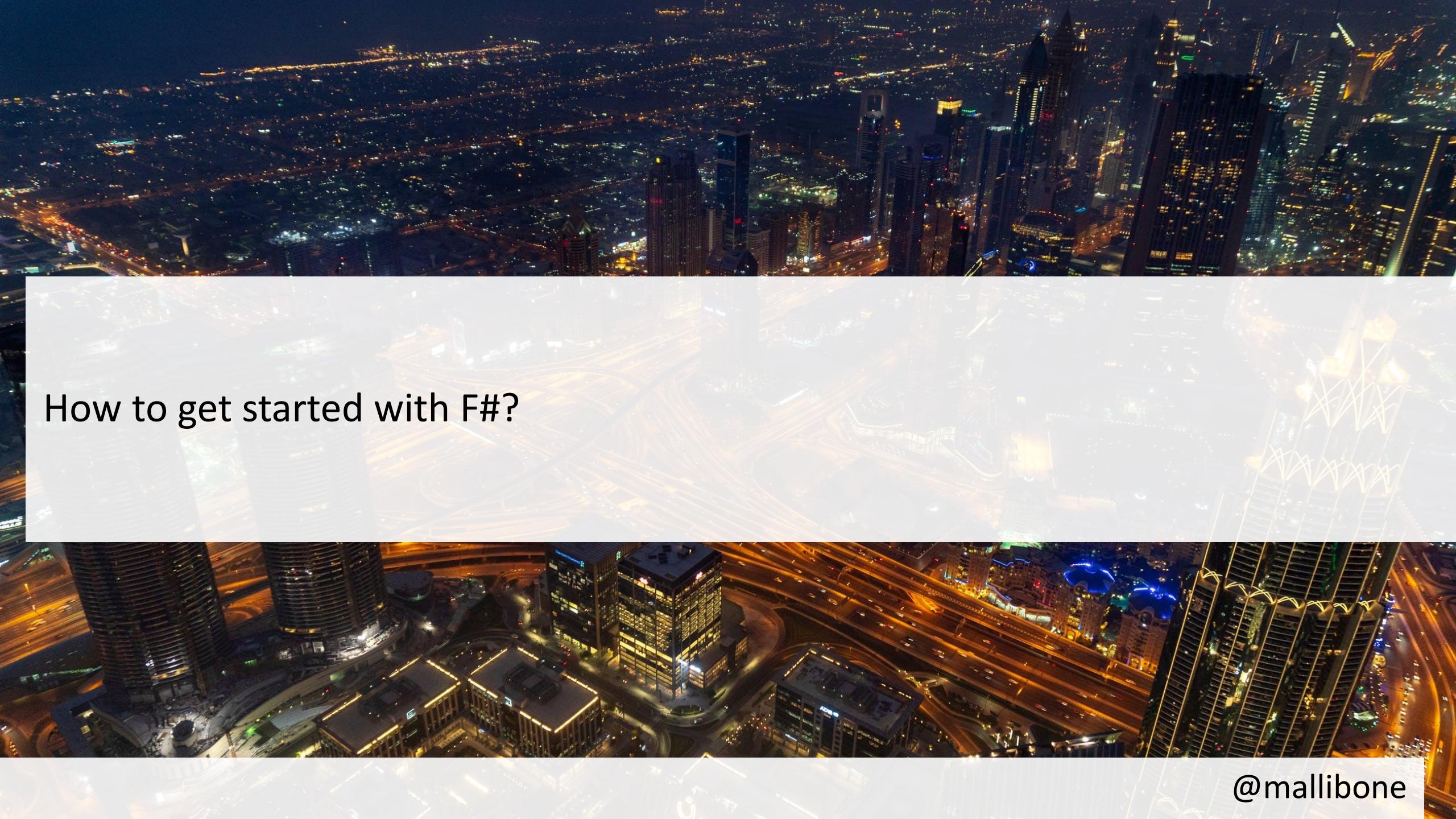
- Accelerometer – Retrieve acceleration data of the device in three dimensional space.
- App Information – Find out information about the application.
- Barometer – Monitor the barometer for pressure changes.
- Battery – Easily detect battery level, source, and state.
- Clipboard – Quickly and easily set or read text on the clipboard.
- Color Converters – Helper methods for `System.Drawing.Color`.
- Compass – Monitor compass for changes.
- Connectivity – Check connectivity state and detect changes.
- Detect Shake – Detect a shake movement of the device.
- Device Display Information – Get the device's screen metrics and orientation.
- Device Information – Find out about the device with ease.
- Email – Easily send email messages.
- File System Helpers – Easily save files to app data.
- Flashlight – A simple way to turn the flashlight on/off.
- Geocoding – Geocode and reverse geocode addresses and coordinates.
- Geolocation – Retrieve the device's GPS location.
- Gyroscope – Track rotation around the device's three primary axes.
- Launcher – Enables an application to open a URI by the system.
- Magnetometer – Detect device's orientation relative to Earth's magnetic field.
- MainThread – Run code on the application's main thread.
- Maps – Open the maps application to a specific location.
- Open Browser – Quickly and easily open a browser to a specific website.
- Orientation Sensor – Retrieve the orientation of the device in three dimensional space.
- Phone Dialer – Open the phone dialer.

Is this page helpful? X

Yes No

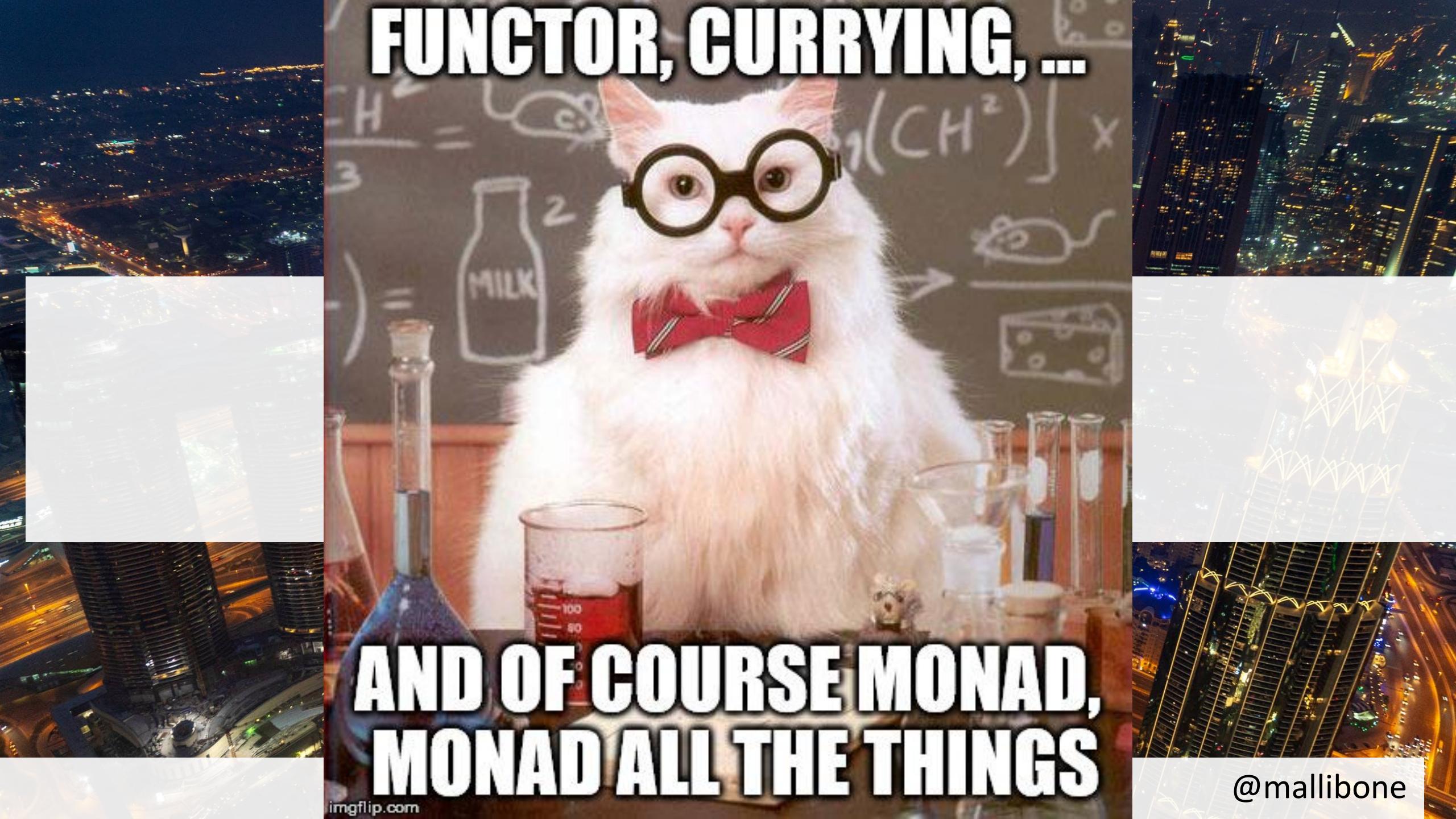
@mallibone





# How to get started with F#?

@mallibone



# FUNCTOR, CURRYING,...

AND OF COURSE MONAD,  
MONAD ALL THE THINGS

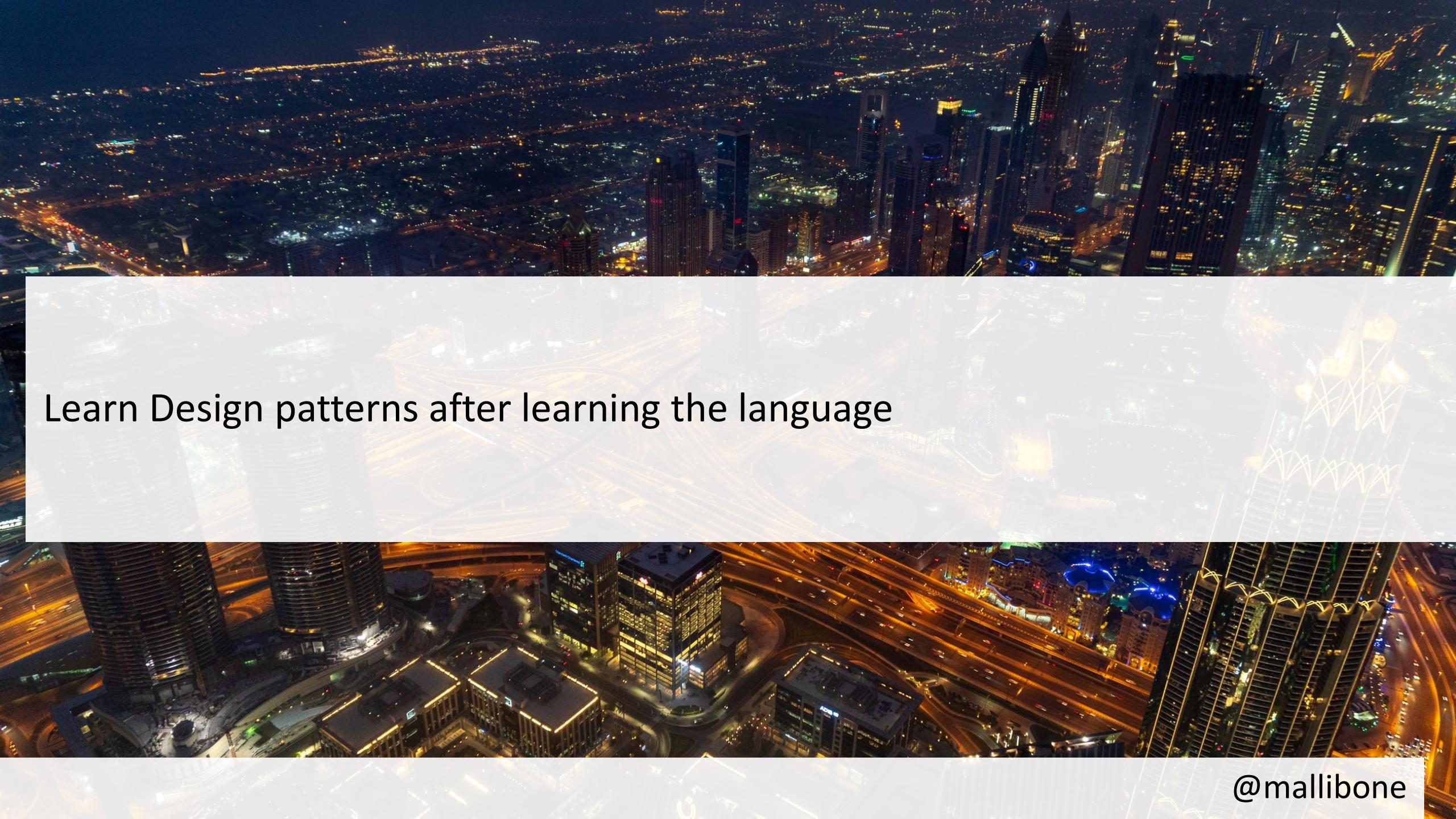


"HOLY FORKING SHIRT!"

#THEGOODPLACE



@mallibone



Learn Design patterns after learning the language

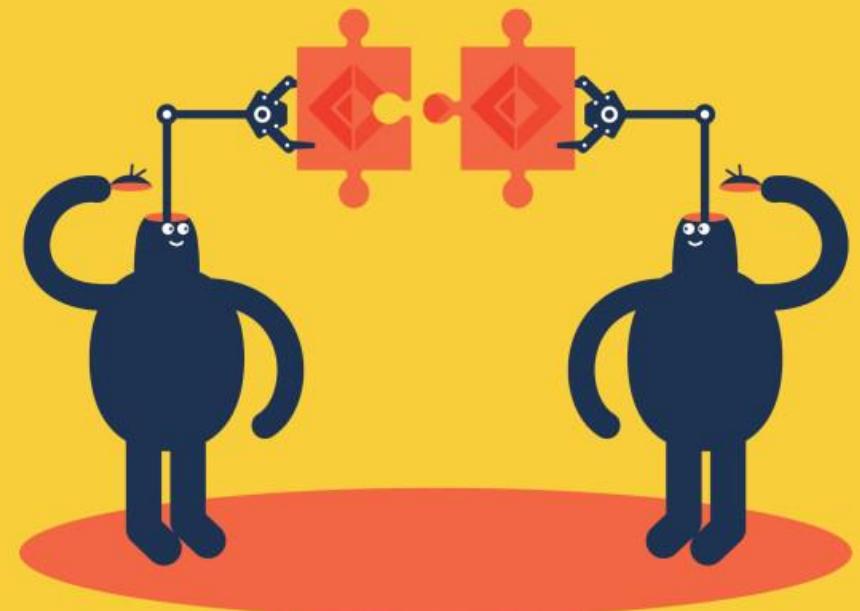


How to get started with F#?



# GET PROGRAMMING WITH **F#**

A guide for .NET developers



Isaac Abraham

Forewords  
by Dustin Campbell  
and Tomas Petricek

 MANNING

# Takeaways



# Takeaways

- F# is not scary it's just yet another .Net language
- Model View Update simplifies state handling in UI
- Fabulous is everywhere
- Have a Fabulous Day!



A professional headshot of a young man with short brown hair, smiling at the camera. He is wearing a light blue button-down shirt.

# Thank you for your time!



Mark Allibone



@mallibone



Head of Mobile, Noser Engineering AG



<https://fsprojects.github.io/Fabulous>

<https://fsharpforfunandprofit.com/>

<https://mallibone.com>



Microsoft®  
Most Valuable  
Professional

