



#MonkeyConf

Fabulous Functional Frontends

Mark Allibone

Mobile Lead

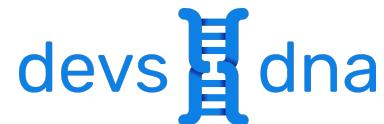
Rey Technology

@mallibone

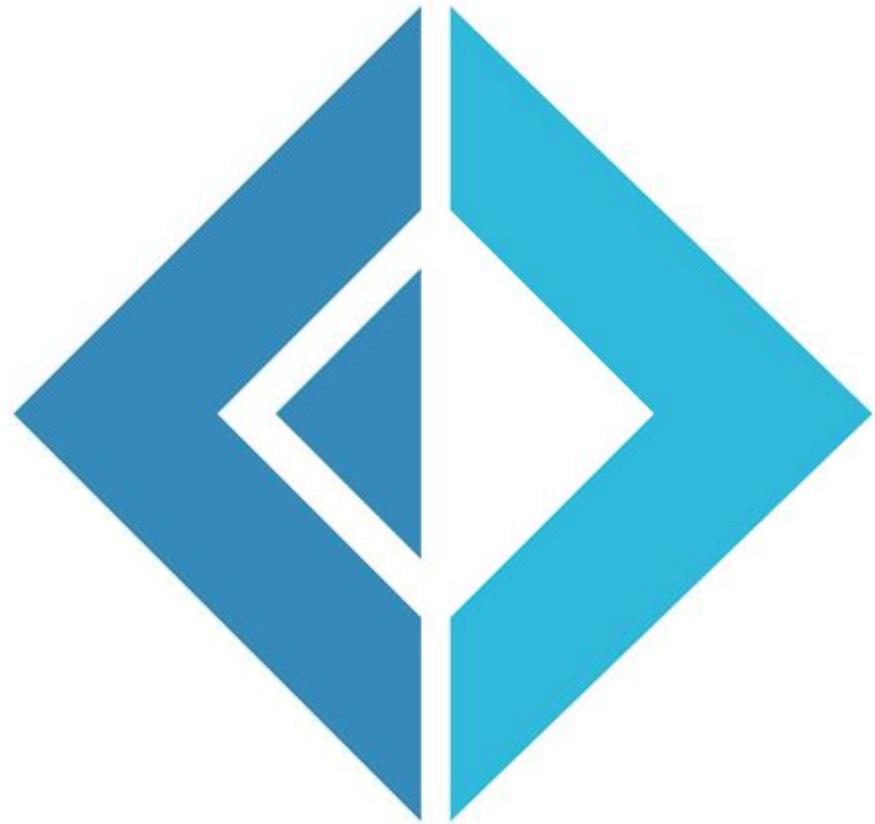
#MonkeyConf



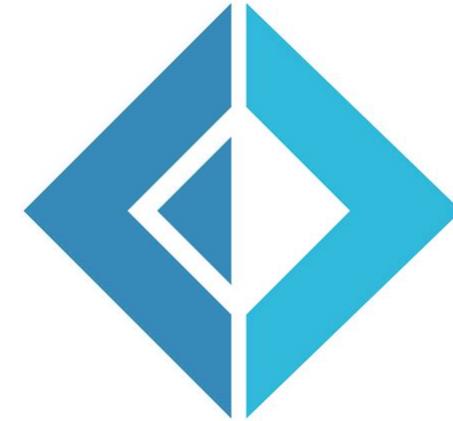
SPONSORS



UXDIVERS
PRODUCT DESIGN STUDIO



- No Nulls
- Immutability
- Declarative Programming
- Based on .Net





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



Fabulous – a MVU framework for F# working with Xamarin.Forms

@mallibone



Why not just Xamarin Forms?!



@mallibone

The screenshot shows the Visual Studio 2019 IDE interface with the following components visible:

- Top Bar:** File, Edit, View, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search Visual Studio (Ctrl+Q).
- Solution Explorer:** Shows the solution structure:
 - Solution 'HelloFSharpConf' (3 of 2 projects)
 - HelloFSharpConf
 - Dependencies
 - AppViewModel.fs
 - AppPage.xaml
 - App.xaml
 - AssemblyInfo.fs
 - HelloFSharpConf.Droid
 - HelloFSharpConf.iOS
- Code Editor:** Displays the `AppPage.fs` file content:

```
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     ()
```
- Output Window:** Shows log output from the application.

```
06-14 06:39:05.177 D/EGL_emulation( 4416): eglGetCurrent: 0xa0aa783250
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: 0xa0aa783250
```
- Bottom Status Bar:** Shows the URL <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

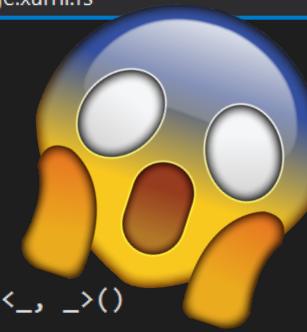
▷ ✓ HelloFSharpConf.Droid

▷ ✓ HelloFSharpConf.iOS

Window Snip

A screenshot of the Visual Studio IDE interface. 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 tab bar at the top shows "HelloFSharpConf" as the active project. Below the menu is a toolbar with various icons. The main workspace displays an F# file named "AppViewModel.fs". The code defines a type `MyViewModel` with properties `text` and `ev`, and methods `createCommand` and `Text`. It also implements the `INotifyPropertyChanged` interface. The code uses F# syntax like `let mutable` and `member` with dot notation. The "text" variable is currently being typed, and the "ev" variable is highlighted in yellow.

```
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
```



```
File Edit View Project Build Debug Test Analyze Tools Extensions Window Help Search Visual Studio (Ctrl+Q) HelloFSharpConf  
Debug Any CPU HelloFSharpConf.Droid my_device (Android 8.1 - API 27)  
AndroidManifest.xml AppViewModel.fs AppPage.xaml.fs  
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          [<>CLIEvent>]  
3          member this.PropertyChanged = ev.Publish  
2  
1          member this.Text  
2              with get() = tex  
1              and set(value) = text <- value  
1                  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. 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 solution configuration "my_device (Android 8.1 - API 27)". The left sidebar features the "Toolbox" and "Test Explorer" panes. The main workspace displays an F# file named "AppViewModel.fs". The code defines a type `MyViewModel` with properties `ev` and `text`, and a command `createCommand`. A red rectangle highlights the command definition. The code also implements the `INotifyPropertyChanged` interface, defining a `Text` property and a `TapCommand`. The code editor shows syntax highlighting for F# and C# components.

```
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
22      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
```

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() = tex
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 the development of a Xamarin application named "HelloSharpConf".

The main window displays the code editor with an F# file named `AppViewModel.fs`. The code defines a `MyViewModel` type with properties for `text` and `TapCommand`, and implements `ICommand` and `INotifyPropertyChanged`.

```
1  namespace HelloSharpConf
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
```

The Solution Explorer on the right shows the project structure, which includes two projects: `HelloSharpConf` and `HelloSharpConf.Droid`.

The Output window at the bottom shows log messages from the Mono runtime, indicating successful EGL initialization and Java interop calls.

```
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)'). 
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: 0xa0a785120: ver 3 0 (tinfo 0xa0a783250)
```



@mallibone

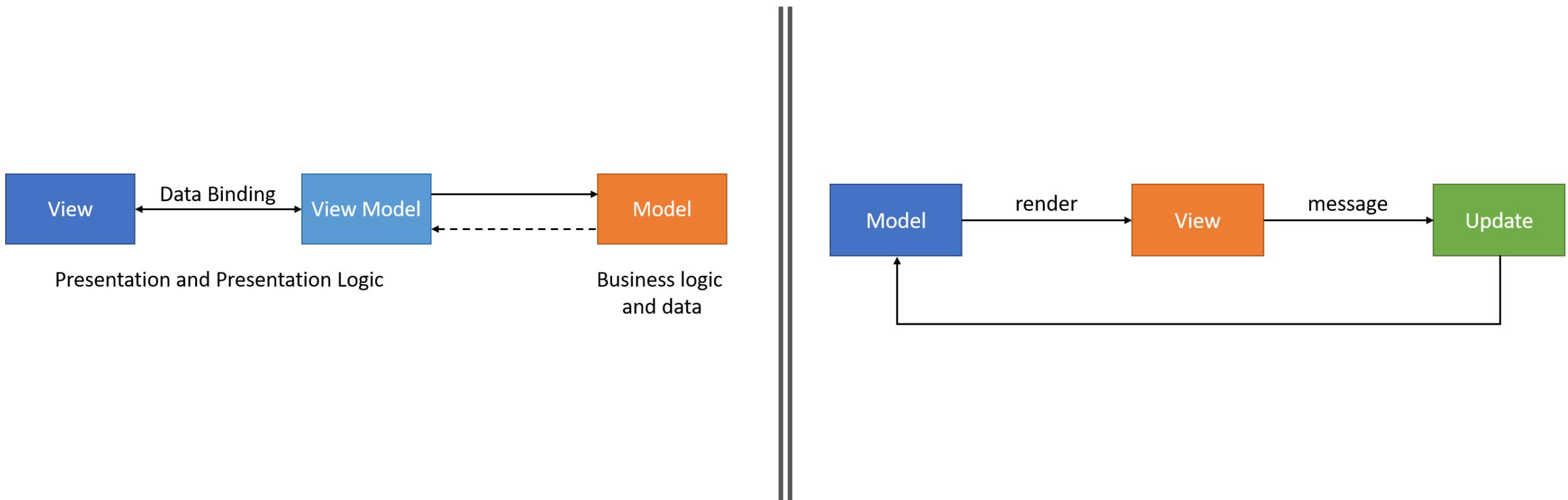


MVU the functional MV* View Pattern

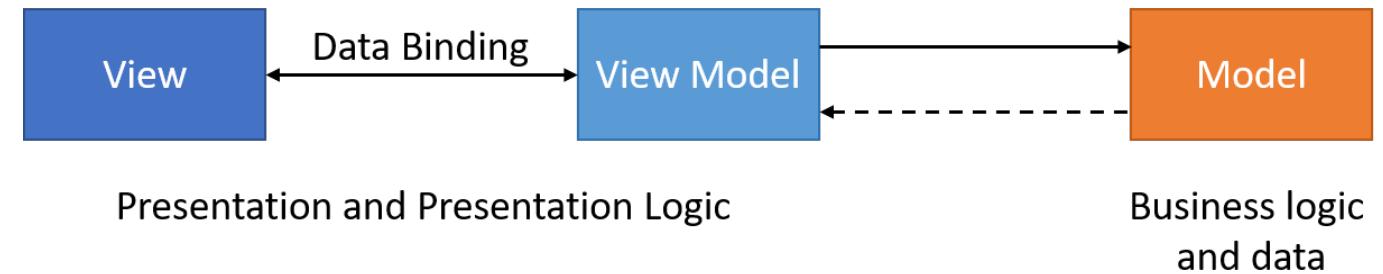


@mallibone

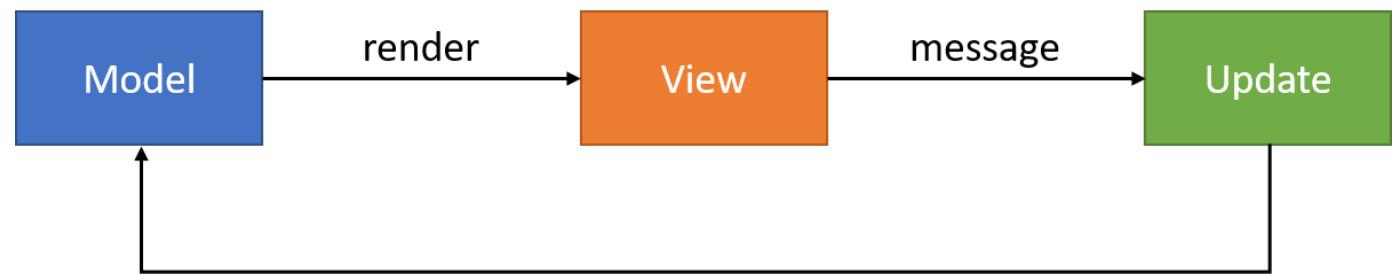
MVVM vs MVU



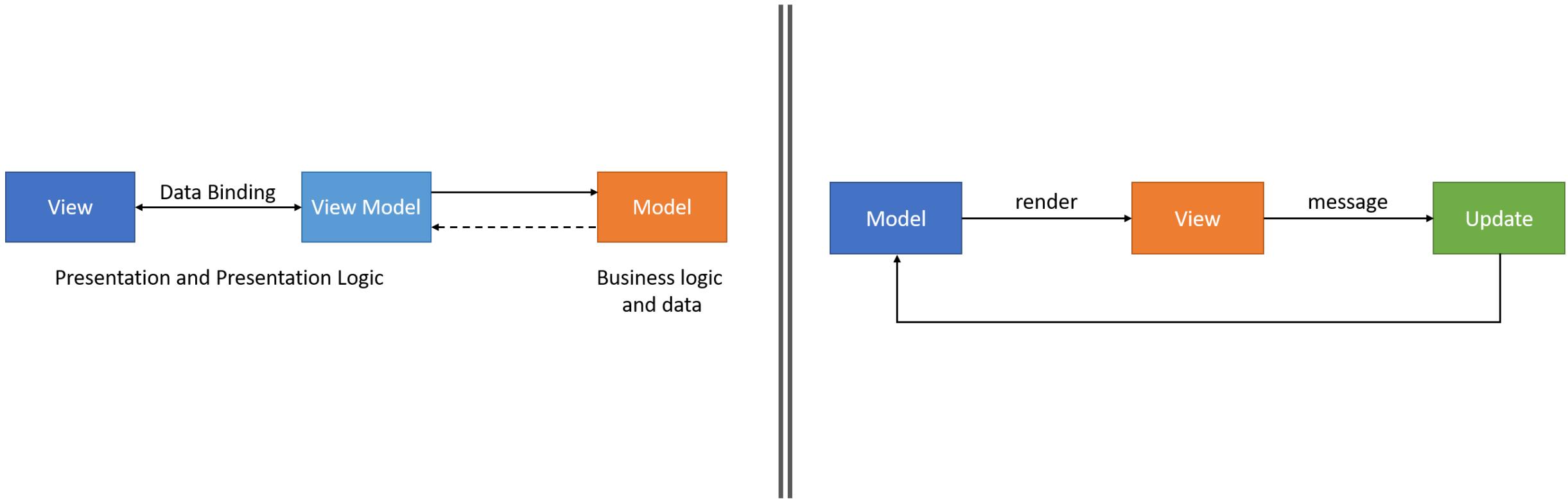
Model View View Model



Model View Update



MVVM vs MVU





NICOTINEBATCH | TUMBLR

FABULOUS

KB3S 

Functional Frontends

Getting started

1. Install Visual Studio or Visual Studio for Mac and enable both Xamarin and .NET Core support, these are listed as 'Mobile development with .NET' and '.NET Core Cross-platform development' respectively.
2. Open a command prompt window and install the template pack by entering:

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

maal@CHREYNB0085 ➤ C:\Work\Playground

[14:19]

> dotnet new fabulous-xf-app -n Gnabber

Gnabber - Microsoft Visual Studio

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

iPhoneSimulator Gnabber.iOS Simulator

Toolbox

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

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

Ready

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

Toolbox

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

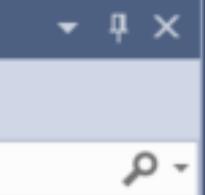
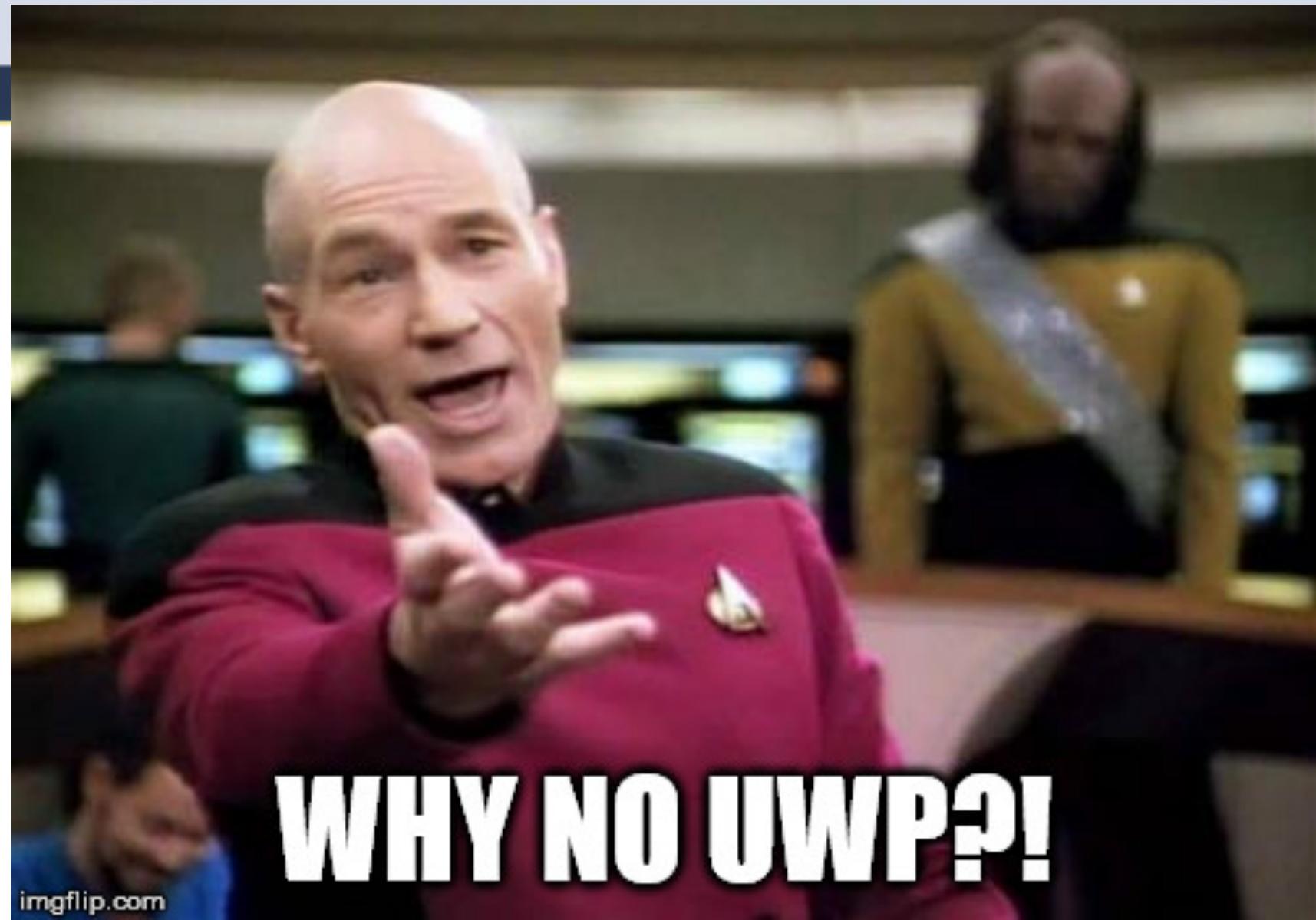
Properties

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

Ready

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 2
33 1 [let view (model: Model) dispatch =
34 0 [View.ContentPage(
35 1 [content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))
36 0
37 1 [// Note, this declaration is needed if you enable LiveUpdate
38 0
39 1 [let program = Program.mkProgram init update view
40 0
41 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 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

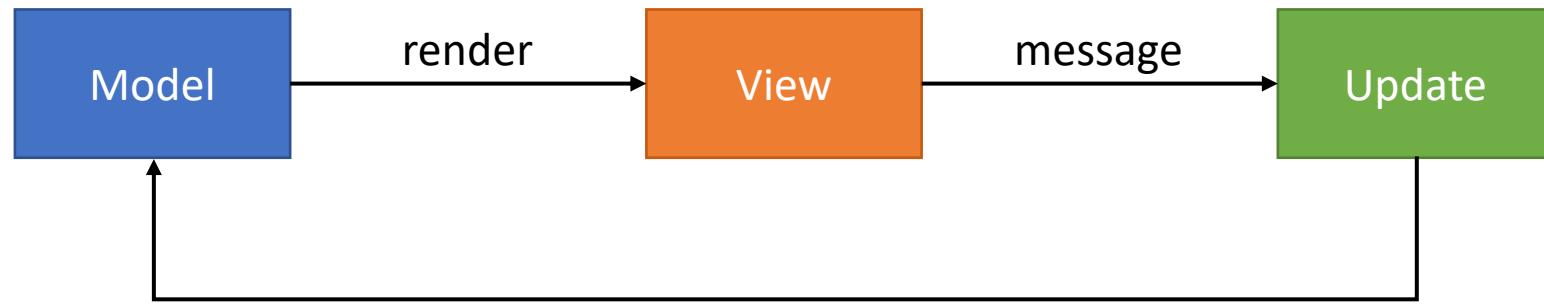
```
7  23 [open Xamarin.Forms
8  22
9  21 [module App =
10 20 [type Model =
11 19 [{ Count : int
12 18 [Step : int
13 17 [TimerOn: bool }
14 16
15 15 [type Msg =
16 14 [| Increment
17 13 [| Decrement
18 12 [| Reset
19 11 [| SetStep of int
20 10 [| TimerToggled of bool
21 9 [| TimedTick
22 8
23 7 [let 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 1
32 2 [let view (model: Model) dispatch =
33 3 [View.ContentPage(content = View.StackLayout(padding = 20.0, verticalOptions = LayoutOptions.Center,...))
34 4 [5
35 6 [// Note, this declaration is needed if you enable LiveUpdate
36 7 [let program = Program.mkProgram init update view
37 8
38 9 [type App () as app =
```

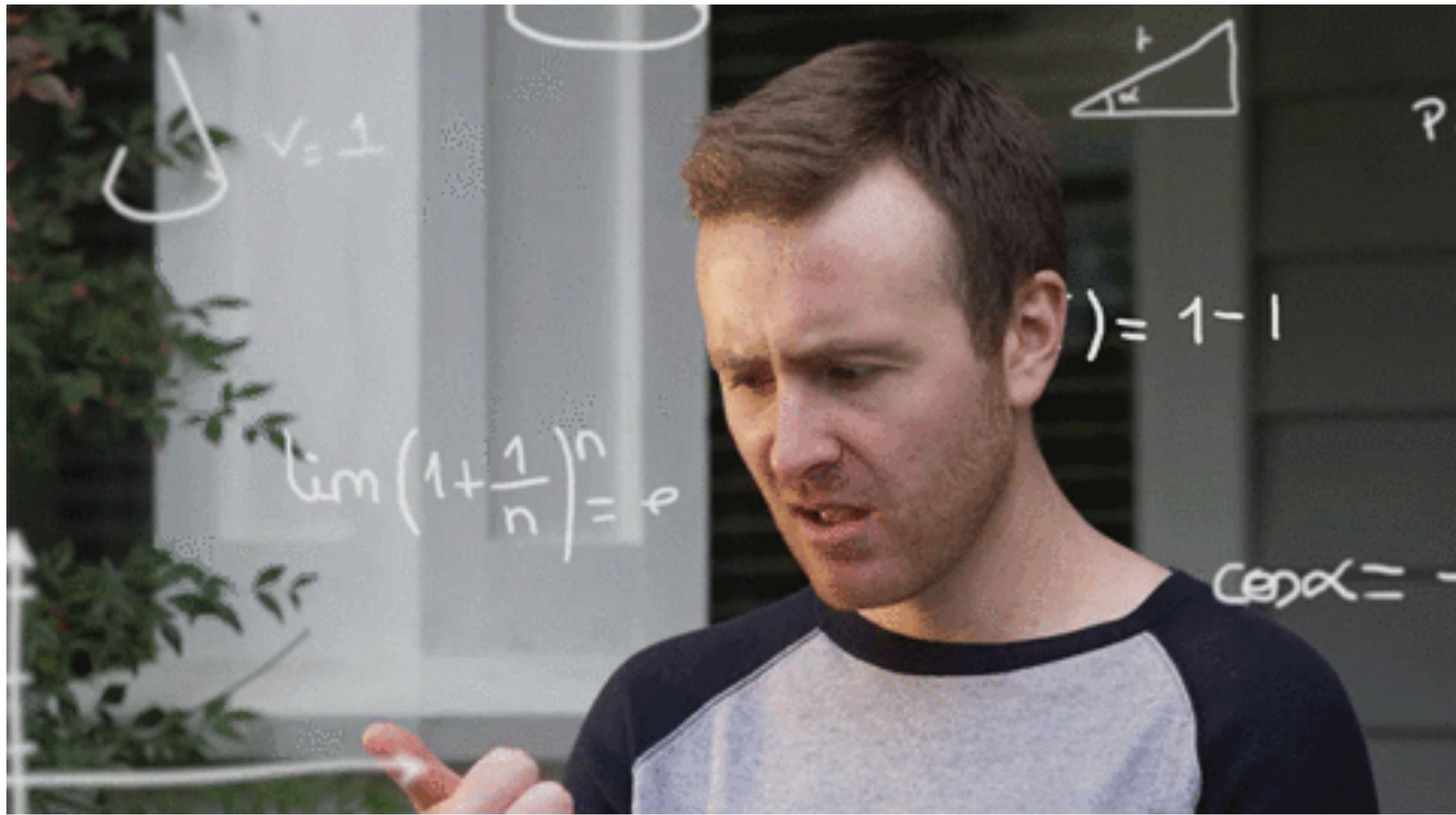


Gnabber.fs

```
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 = ...]
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 =
```

Model View Update





- Driven from the model not the view
- State changes are defined with messages
- Single place for change - the update method
- Can be based on Xamarin Forms





Con un diseño como ese no
habrá regalos para ti bajo el
árbol de navidad



CSS

Doing it with style



@mallibone



XAML vs Coded UI

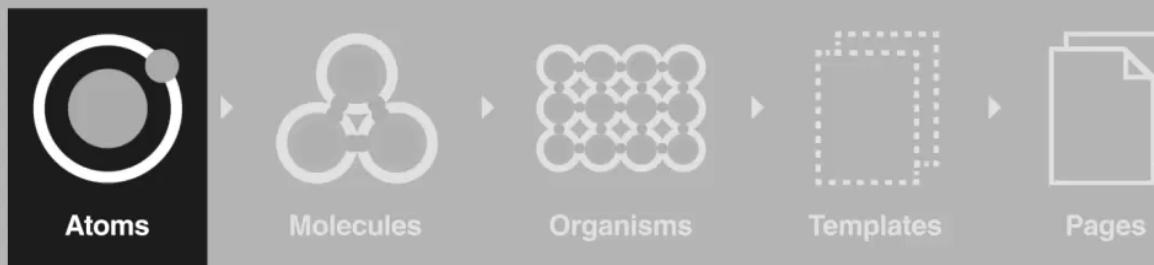
The background of the image is a close-up photograph of ink swirling in water. The ink forms large, billowing clouds in various colors, including vibrant orange, yellow, and hints of purple and red. The ink is suspended against a solid black background, creating a stark contrast that emphasizes the fluid shapes and color gradients of the ink.

Demo

@mallibone

- Interactive Development
- Create reusable UI
- You can create good looking UIs
- UI is based on Xamarin Forms





The background of the slide features a vibrant, abstract pattern of ink swirling in water against a black background. The colors include bright orange, yellow, pink, purple, and white, creating a dynamic and organic feel.

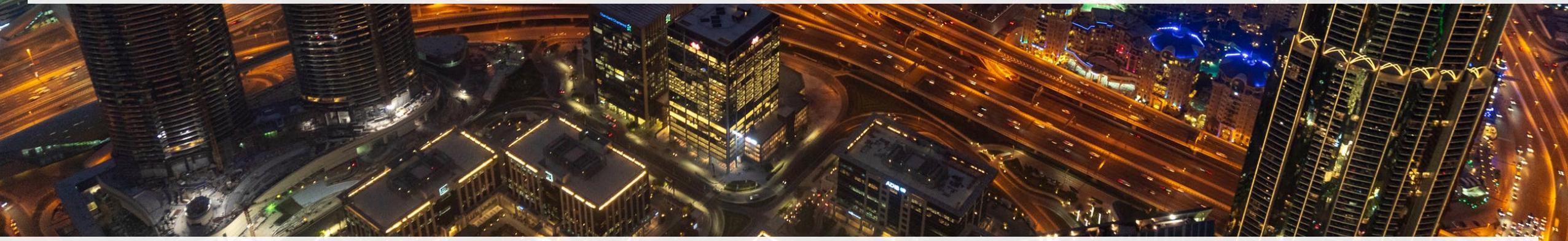
Live Update ❤️ Coded UI

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

@mallibone



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



@mallbone



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

Feedback Edit Share Theme Sign in

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.

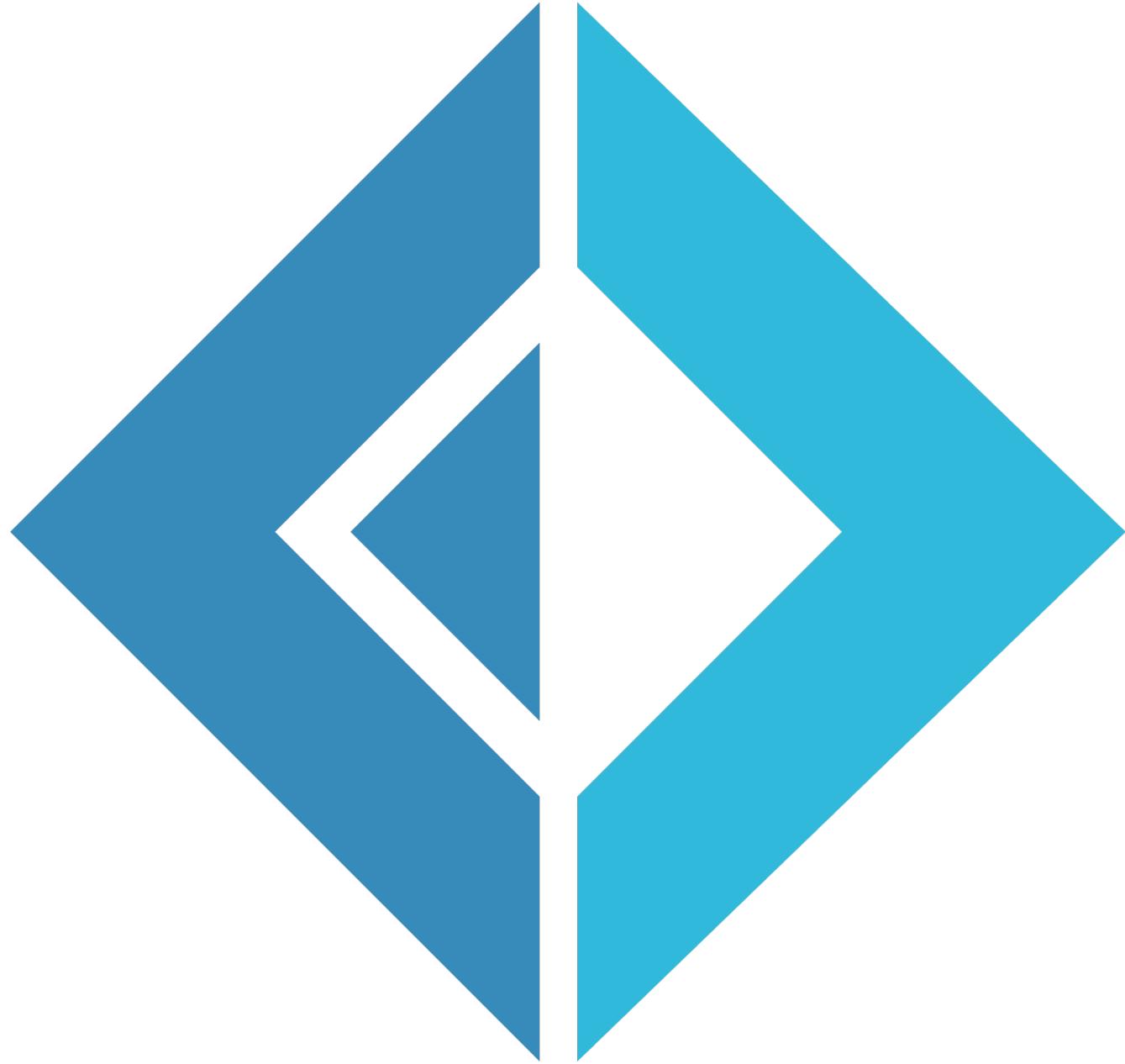
Feature Guides

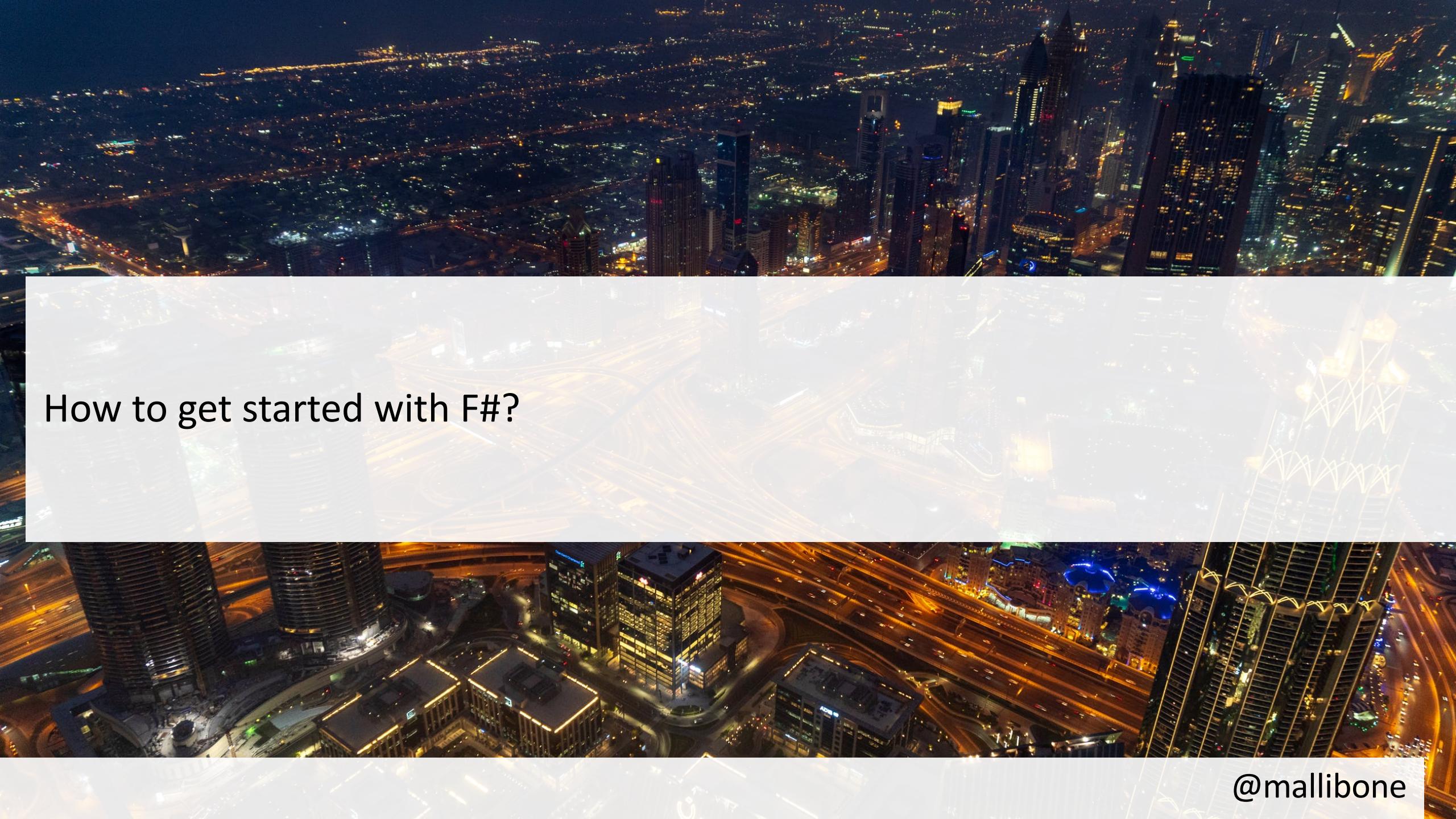
Follow the guides to integrate these Xamarin.Essentials features into your applications:

- [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?

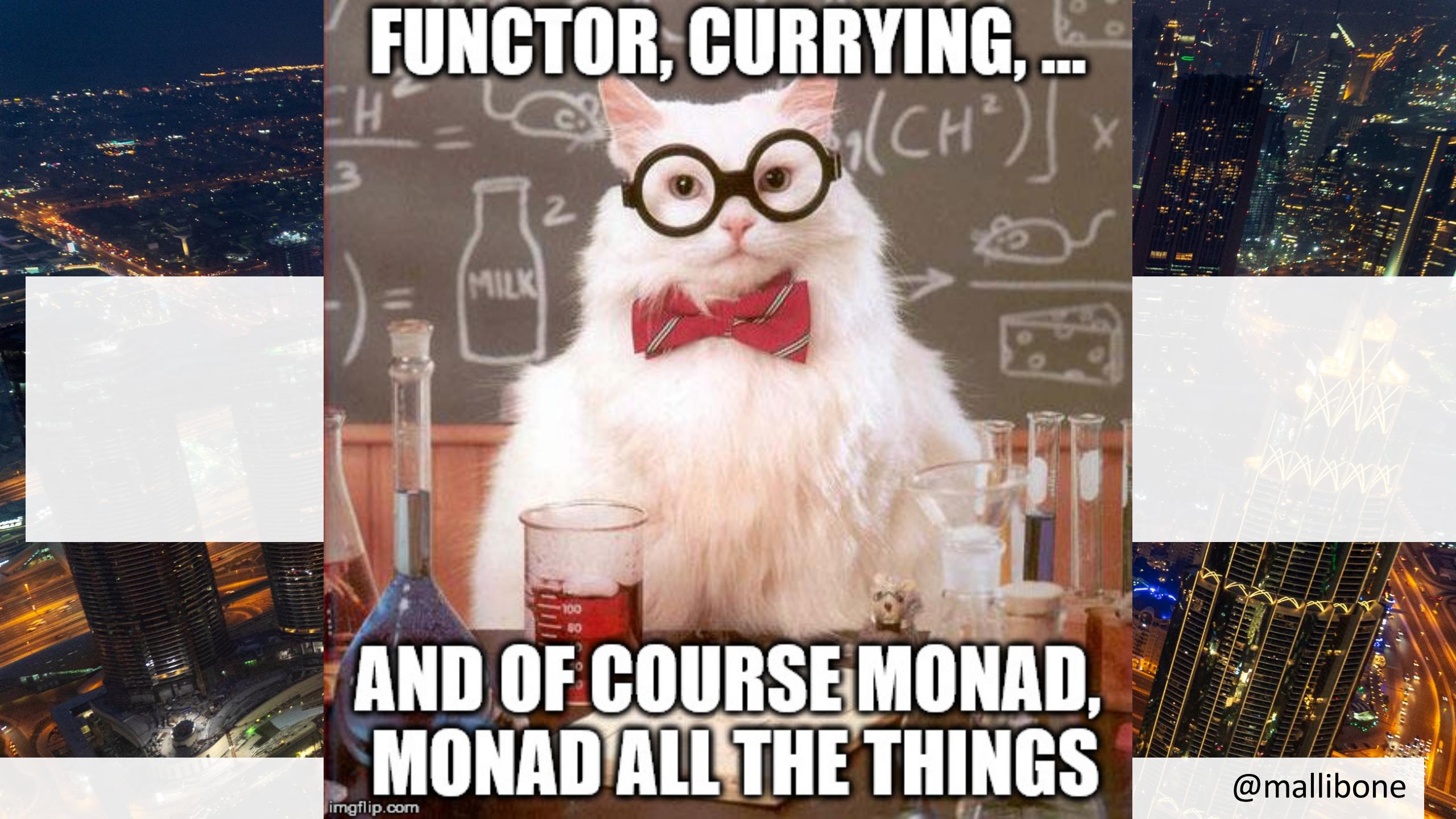
@mallbone





How to get started with F#?

@mallbone



FUNCTOR, CURRYING,...

AND OF COURSE MONAD,
MONAD ALL THE THINGS

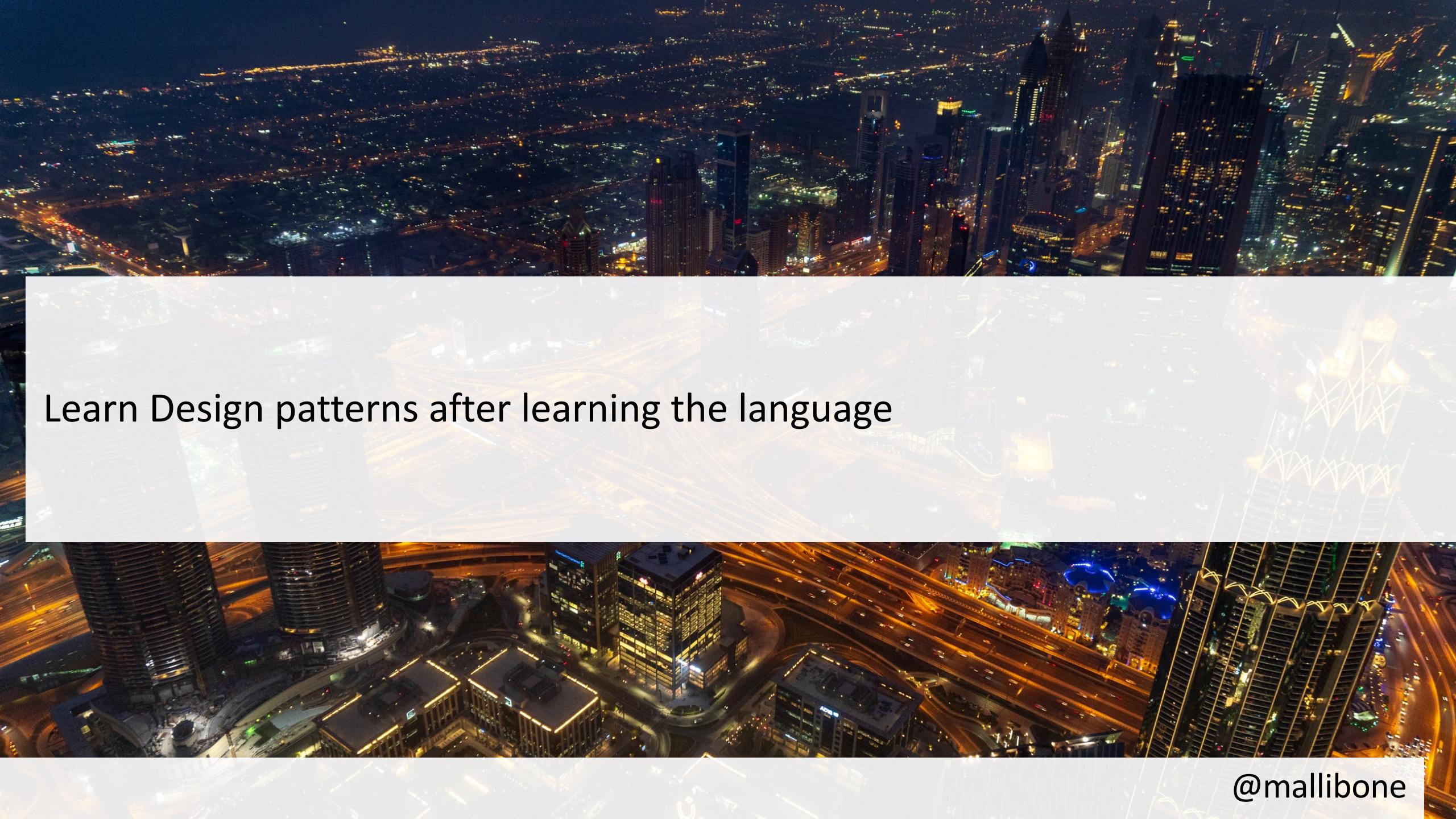


"HOLY FORKING SHIRT!"

#THEGOODPLACE



@mallbone



Learn Design patterns after learning the language

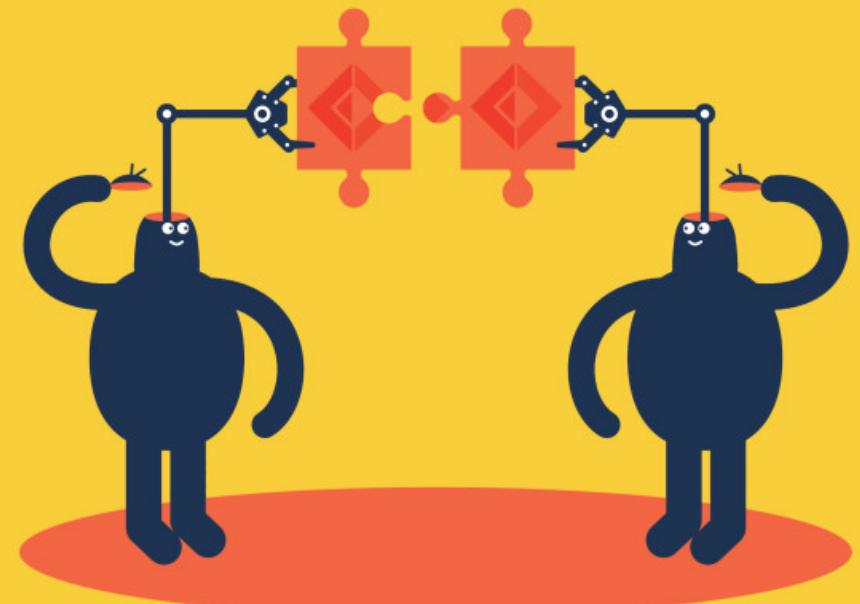


How to get started with F#?



GET PROGRAMMING WITH **F#**

A guide for .NET developers

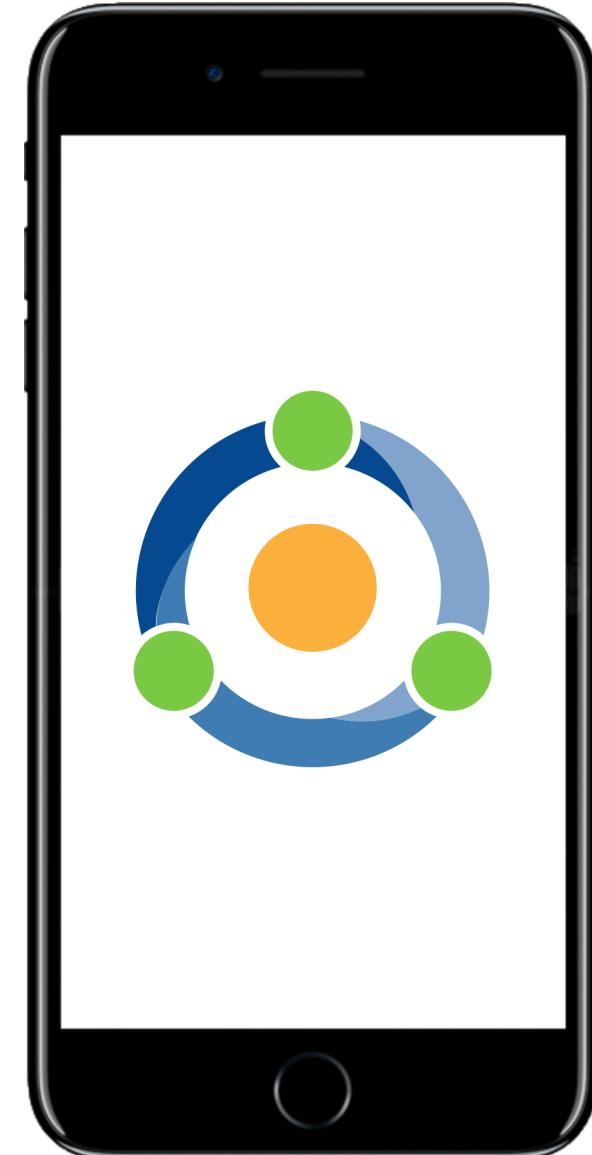


Isaac Abraham

Forewords
by Dustin Campbell
and Tomas Petricek

 MANNING

Takeaways



Takeaways

- F# is not scary it's just your friendly functional first .NET language
- Model View Update simplifies state handling in UI
- Fabulous is everywhere
- Have a Fabulous Day!





Thank you for your time!



Mark Allibone



@mallibone



Rey Technology



<https://nullpointers.io>



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



<https://fsharpforfunandprofit.com/>



<https://mallibone.com>



Microsoft®
Most Valuable
Professional



Gracias!

Gracias a nuestros Sponsors.
Sin ellos el evento no sería posible.

**plain
concepts** 

devs  **dna**

 **Beezy**


NAMOCODE

 **UXDIVERS**
PRODUCT DESIGN STUDIO