SWIFTUI

AGENDA

1. INTRODUCTION
2. LAYOUT SYSTEM
3. DATA FLOW
4. ARCHITECTURE
5. UIKIT / APPKIT/WATCHKIT ? SWIFTUI
6. SOME THOUGHTS
7. DEMO

INTRODUCTION

- THE NEW UI FRAMEWORK. INTRODUCED IN WWDC 2019
- WORKS FOR IPAD. MAC. APPLE TV AND WATCH
- THE CONTROLS WILL BE AUTOMATICALLY TRANSLATED FOR YOU

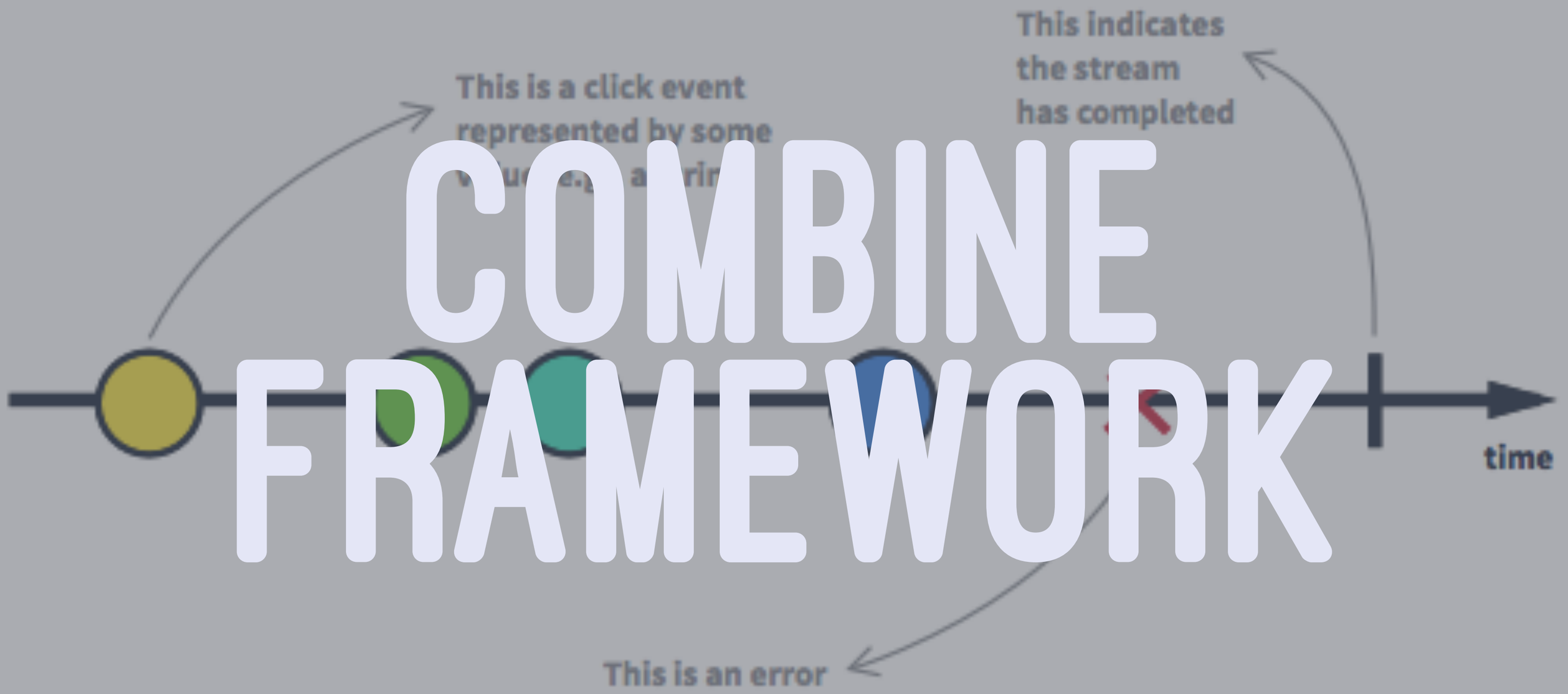
"WRITE ONCE. APPLY EVERYWHERE" 

"LEARN ONCE. APPLY ANYWHERE" 

SWIFTUI'S COMPANIONS

```swift
//
//  ContentView.swift
//  Counter
//
//  Created by Phuc Le Dien on 8/1/19.
//  Copyright © 2019 Dwarves Foundation. All rights reserved.
//

import SwiftUI

struct ContentView : View {
    @ObservedObject var state: AppState

    var body: some View {

        NavigationView {
            List {
                NavigationLink(destination: CounterView(state: state)) {
                    Text("Counter demo")
                }
                Navigatio nation: Fa tesPrin taine w(s
                    stat
                    Text vorit rimes")
                }
            }
            .navigationB tle("State mana nt
        }
    }
}

#if DEBUG
struct ContentView_Previews : PreviewProvider {
    static var previews: some View {
        ContentView(state: AppState())
    }
}
#endif
```

CANVAS VIEW



COMBINE FRAMEWORK

HELLO WORLD

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello")
            Text("World")
                .bold()
        }
    }
}
```

HOW DID THEY ACHIEVE THAT 🤔

- IMPLICIT RETURN
- OPAQUE RETURN TYPES
- FUNCTION BUILDER
- VIEW MODIFIER

OPAQUE RETURN TYPES

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello")
            Text("World")
                .bold()
        }
    }
}
```

FUNCTION BUILDER

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello")
            Text("World")
                .bold()
        }
    }
}
```

IT COULD BE THIS 😐

```
VStack(
   Text("Hello"),
   Text("World")
       .bold()
)
```

@VIEWBULDER

Declaration

```
init(alignment: VerticalAlignment = .center, spacing:
    Length? = nil, @ViewBuilder content: () -> Content)
```

```
 VStack {
    Text("Hello")
    Text("World")
        .bold()
}

VStack({
    return ViewBuilder.buildBlock(
        Text("Hello"),
        Text("World")
            .bold()
    )
})
```

```
static func buildBlock() -> EmptyView

static func buildBlock<Content>(Content) -> Content

static func buildBlock<C0, C1>(C0, C1) -> TupleView<(C0, C1)>

static func buildBlock<C0, C1, C2>(C0, C1, C2) -> TupleView<(C0, C1, C2)>

static func buildBlock<C0, C1, C2, C3>(C0, C1, C2, C3) -> TupleView<(C0, C1, C2, C3)>

static func buildBlock<C0, C1, C2, C3, C4>(C0, C1, C2, C3, C4) -> TupleView<(C0, C1, C2, C3, C4)>

static func buildBlock<C0, C1, C2, C3, C4, C5>(C0, C1, C2, C3, C4, C5) -> TupleView<(C0, C1, C2, C3, C4, C5)>

static func buildBlock<C0, C1, C2, C3, C4, C5, C6>(C0, C1, C2, C3, C4, C5, C6) -> TupleView<(C0, C1, C2, C3, C4, C5, C6)>

static func buildBlock<C0, C1, C2, C3, C4, C5, C6, C7>(C0, C1, C2, C3, C4, C5, C6, C7) -> TupleView<(C0, C1, C2, C3, C4, C5, C6, C7)>

static func buildBlock<C0, C1, C2, C3, C4, C5, C6, C7, C8>(C0, C1, C2, C3, C4, C5, C6, C7, C8) -> TupleView<(C0, C1, C2, C3, C4, C5, C6, C7, C8)>

static func buildBlock<C0, C1, C2, C3, C4, C5, C6, C7, C8, C9>(C0, C1, C2, C3, C4, C5, C6, C7, C8, C9) -> TupleView<(C0, C1, C2, C3, C4, C5, C6, C7, C8, C9)>
```

VIEW MODIFERS

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        VStack {
            Text("Hello")
            Text("World")
                .bold()
        }
    }
}
```

CUSTOM VIEWMODIFIER

```
struct PrimaryLabel: ViewModifier {
    func body(content: Content) -> some View {
        content
            .padding()
            .background(Color.red)
            .foregroundColor(Color.white)
            .font(.largeTitle)
    }
}

struct ContentView: View {
    var body: some View {
        Text("Hello, SwiftUI")
            .modifier(PrimaryLabel())
    }
}
```

BEAUTIFY IT 🤩

```
extension View {
    func titleStyle() -> some View {
        self.modifier(PrimaryLabel())
    }
}

struct ContentView: View {
    var body: some View {
        Text("Hello, SwiftUI")
            .titleStyle()
    }
}
```

SOME VIEWS AND CONTROLS

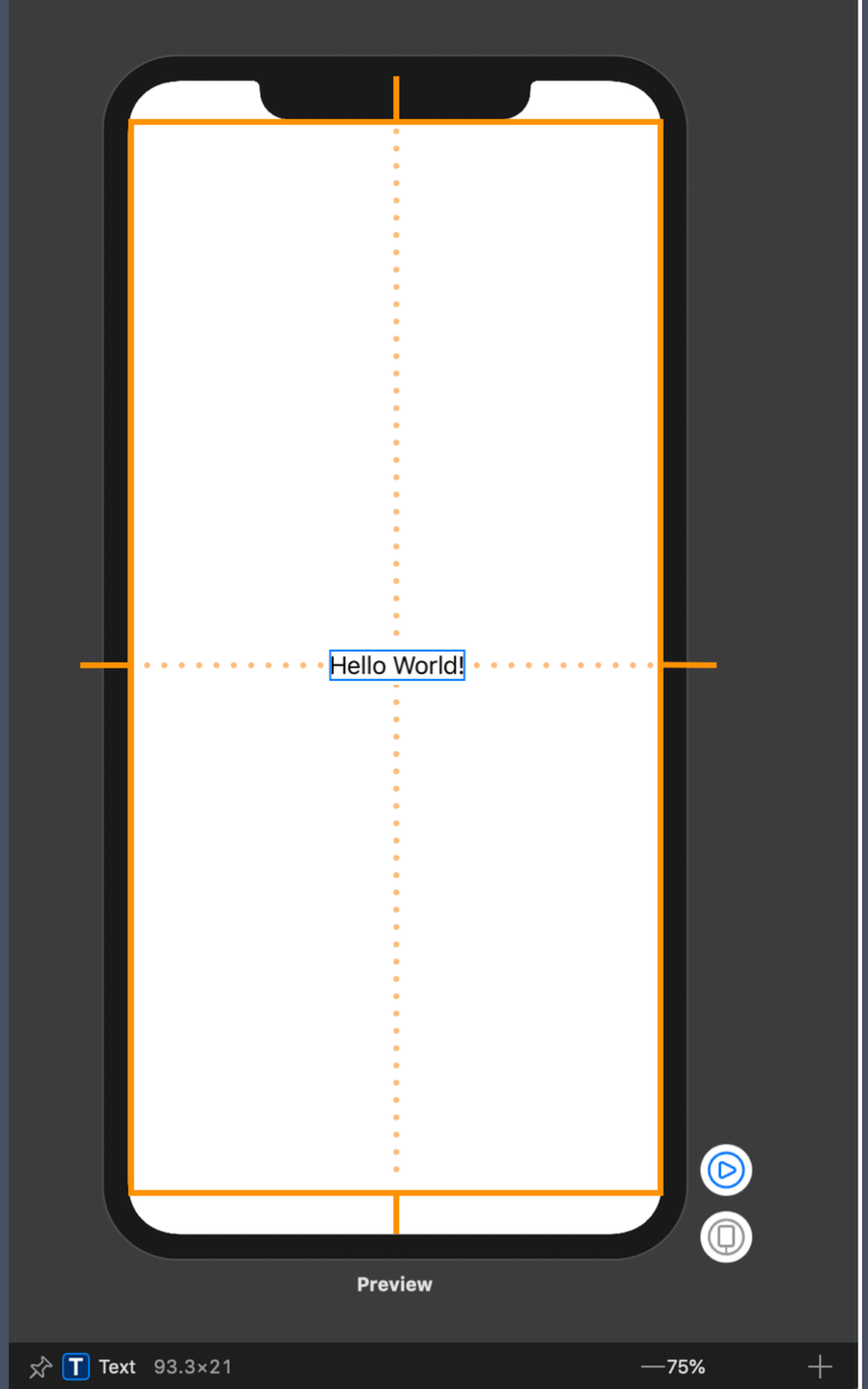
- LIST
- TEXTFIELD
- BUTTON
- PICKER
- TOGGLE
- HSTACK, ZSTACK
- ...

SEE MORE HERE

LAYOUT SYSTEM

LAYOUT PROCESS

1. PARENT PROPOSES SIZE FOR CHILD
2. CHILD CHOOSES ITS SIZE
3. PARENT PLACES CHILD IN PARENT'S COORDINATE SPACE



FRAME

```
struct Frame: View {
    var body: some View {
        Image("swiftui")
            .border(Color.red)
            .frame(width: 80, height: 80)
            .border(Color.blue)
    }
}
```

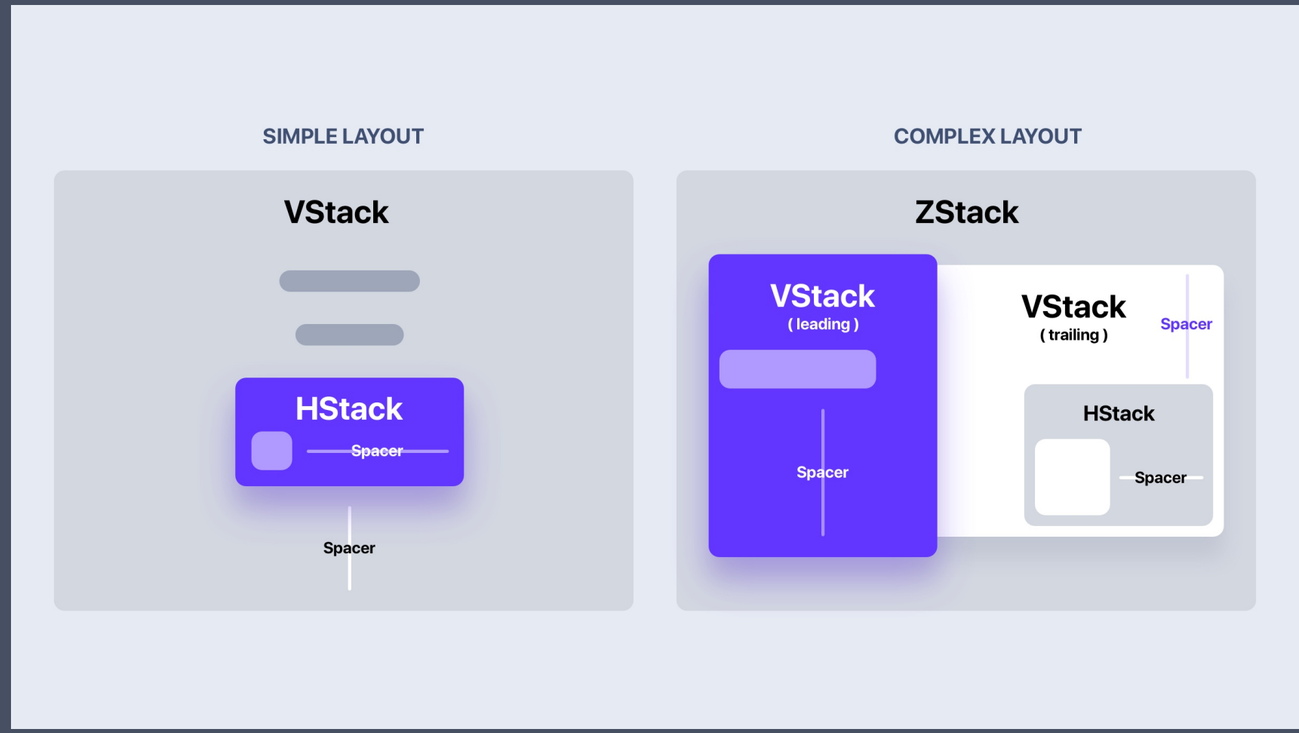
```
struct Frame: View {
    var body: some View {
        Image("swiftui")
            .border(Color.red)
            .frame(width: 80, height: 80,
                   alignment: .topLeading)
            .border(Color.blue)
    }
}
```

```swift
struct Frame: View {
    var body: some View {
        Image("swiftui")
            .resizable()
            .border(Color.red)
            .frame(width: 80, height: 80)
            .border(Color.blue)
    }
}
```

```
struct Frame: View {
    var body: some View {
        Image("swiftui")
            .border(Color.red)
            .frame(width: 40, height: 80)
            .border(Color.blue)
    }
}
```

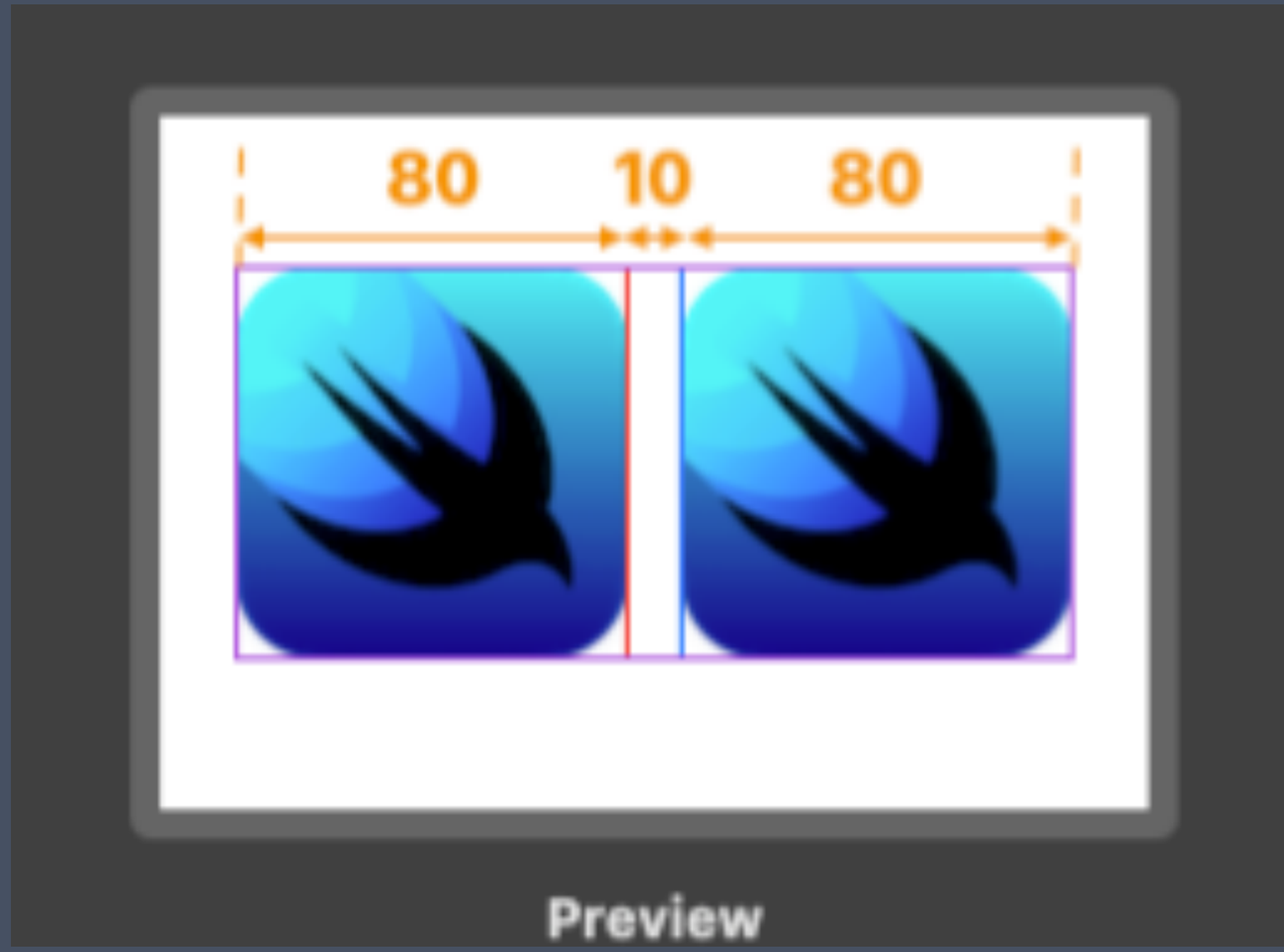
STACKS

- HSTACK - HORIZONTALLY
- VSTACK - VERTICALLY
- ZSTACK - BACK TO FRONT



STACK LAYOUT PROCESS

1. CALCULATE SPACE FOR CHILDVIEW AFTER SUBTRACT INTERNAL SPACING
2. DEVIDE THE REMAIN SPACE LEFT EQUALLY. AND PROPOSE AVAILABLE SIZE FOR THEIR CHILD
3. ALL CHILDREN HAVE SIZES. THE STACK LINES THEM UP WITH THE SPACING AND ALIGNS THEM ACCORDING TO THE SPECIFIED ALIGNMENT.



DATA FLOW

PROPERTY WRAPPERS

EXAMPLE

```swift
var text = " \n Hello, World! \n\n    "

let trimmedText = text.trimmingCharacters(in: .whitespacesAndNewlines)

print(trimmedText) // "Hello, World!"
```

WITH PROPERTY WRAPPERS

```
@Trimmed
var text = " \n Hello, World! \n\n     "

print(text) // "Hello, World!"
```

```
@propertyWrapper
public struct Trimmed {
    private var value: String!
    private let characterSet: CharacterSet

    public var wrappedValue: String {
        get { value }
        set { value = newValue.trimmingCharacters(in: characterSet) }
    }

    public init(wrappedValue: String) {
        self.characterSet = .whitespacesAndNewlines
        self.wrappedValue = wrappedValue
    }

    public init(wrappedValue: String, characterSet: CharacterSet) {
        self.characterSet = characterSet
        self.wrappedValue = wrappedValue
    }
}
```

PROPERTY WRAPPERS IN SWIFTUI

- @STATE
- @BINDING
- @OBSERVEDOBJECT
- @ENVIRONMENTOBJECT
- @ENVIRONMENT

@STATE

```
struct ProductsView: View {
    let products: [Product]

    @State private var showFavorited: Bool = false

    var body: some View {
        List {
            Button(
                action: { self.showFavorited.toggle() },
                label: { Text("Change filter") }
            )

            ForEach(products) { product in
                if !self.showFavorited || product.isFavorited {
                    Text(product.title)
                }
            }
        }
    }
}
```

@BINDING

```
struct FilterView: View {
    @Binding var showFavorited: Bool

    var body: some View {
        Toggle(isOn: $showFavorited) {
            Text("Change filter")
        }
    }
}

struct ProductsView: View {
    let products: [Product]

    @State private var showFavorited: Bool = false

    var body: some View {
        List {
            FilterView(showFavorited: $showFavorited)

            ForEach(products) { product in
                if !self.showFavorited || product.isFavorited {
                    Text(product.title)
                }
            }
        }
    }
}
```

@OBSERVEDOBJECT

```
import Combine

final class PodcastPlayer: ObservableObject {
    let objectWillChange = PassthroughSubject<Void, Never>()

    private(set) var isPlaying: Bool = false {
        willSet { objectWillChange.send() }
    }

    func play() {
        isPlaying = true
    }

    func pause() {
        isPlaying = false
    }
}
```

EVEN SHORTER 🤩

```
import Combine

final class PodcastPlayer: ObservableObject {

    @Published private(set) var isPlaying: Bool = false

    func play() {
        isPlaying = true
    }

    func pause() {
        isPlaying = false
    }
}
```

```
struct EpisodesView: View {
    @ObservedObject var player: PodcastPlayer
    let episodes: [Episode]

    var body: some View {
        List {
            Button(
                action: {
                    if self.player.isPlaying {
                        self.player.pause()
                    } else {
                        self.player.play()
                    }
            }, label: {
                        Text(player.isPlaying ? "Pause": "Play")
                }
            )
            ForEach(episodes) { episode in
                Text(episode.title)
            }
        }
    }
}
```

@ENVIRONMENTOBJECT

```
class SceneDelegate: UIResponder, UIWindowSceneDelegate {

    var window: UIWindow?

    func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {
        let window = UIWindow(frame: UIScreen.main.bounds)
        let episodes = [
            Episode(id: 1, title: "First episode"),
            Episode(id: 2, title: "Second episode")
        ]

        let player = PodcastPlayer()
        window.rootViewController = UIHostingController(
            rootView: EpisodesView(episodes: episodes)
                .environmentObject(player)
        )
        self.window = window
        window.makeKeyAndVisible()
    }
}
```

```
struct EpisodesView: View {
    @EnvironmentObject var player: PodcastPlayer
    let episodes: [Episode]

    var body: some View {
        List {
            Button(
                action: {
                    if self.player.isPlaying {
                        self.player.pause()
                    } else {
                        self.player.play()
                    }
            }, label: {
                    Text(player.isPlaying ? "Pause": "Play")
                }
            )
            ForEach(episodes) { episode in
                Text(episode.title)
            }
        }
    }
}
```

@OBSERVEDOBJECT AND @ENVIRONMENTOBJECT 🤔

A -> B -> C -> D

@ENVIRONMENT

```
struct CalendarView: View {
    @Environment(\.calendar) var calendar: Calendar
    @Environment(\.locale) var locale: Locale
    @Environment(\.colorScheme) var colorScheme: ColorScheme

    var body: some View {
        return Text(locale.identifier)
    }
}
```

ARCHITECTURE

MVVM

```
struct SearchView: View {
    @ObservedObject var viewModel: SearchViewModel

    var body: some View {
        VStack {
            TextField("Search", text: $viewModel.query)
            List(viewModel.songs) {
                Text($0.name)
            }
        }
    }
}

final class SearchViewModel: ObservableObject {
    @Published var query: String = ""
    @Published private(set) var songs: [Song] = []
    private var cancellable: AnyCancellable?

    init(service: SearchService) {
        cancellable = $query
            .throttle(for: .milliseconds(300), scheduler: DispatchQueue.main, latest: true)
            .removeDuplicates()
            .flatMap {
                service.searchSongs(query: $0).catch {
                    _ in Just([])
                }
            }
            .receive(on: DispatchQueue.main)
            .sink { [unowned self] in self.songs = $0 }
    }
}
```

Dimillian / MovieSwiftUI

Watch 112 | Unstar 4.2k | Fork 330

Code | Issues 8 | Pull requests 1 | Actions | Projects 1 | Wiki | Security | Insights

SwiftUI & Combine app using MovieDB API. With a custom Flux (Redux) implementation.

316 commits | 2 branches | 0 packages | 0 releases | 9 contributors | Apache-2.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

Dimillian Make the blur always opaque in movie detail — Latest commit 7a23a6b 16 days ago

MovieSwift		16 days ago
images		8 months ago
sketches		9 months ago
.gitignore		9 months ago
LICENSE		8 months ago
Readme.md		2 months ago

REDUX

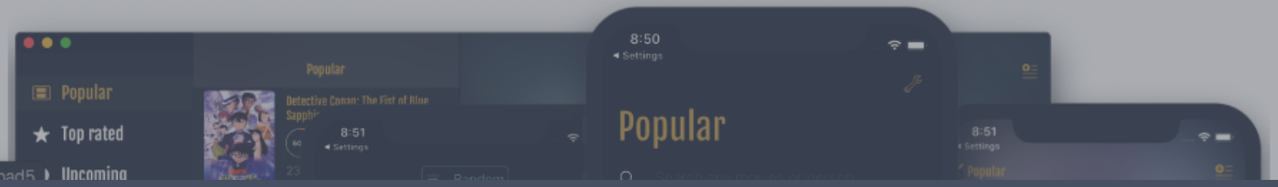
CHECK HERE

Readme.md

MovieSwiftU

MovieSwiftUI is an application that uses the MovieDB API and is built with SwiftUI. It demos some SwiftUI (& Combine) concepts. The goal is to make a real world application using SwiftUI only. It'll be updated with new features as they come to the SwiftUI framework.

I have written a series of articles that document the design and architecture of the app: Making a Real World Application With SwiftUI.



UIKIT ? SWIFTUI

SWIFTUI DOESN'T SUPPORT YOUR FAVORITE COMPONENT? 😞

DIY 😴

```
struct TextView: UIViewRepresentable {
    @Binding var text: String

    func makeUIView(context: Context) -> UITextView {
        return UITextView()
    }

    func updateUIView(_ uiView: UITextView, context: Context) {
        uiView.text = text
    }
}

struct ContentView : View {
    @State var text = ""

    var body: some View {
        TextView(text: $text)
            .frame(minWidth: 0, maxWidth: .infinity, minHeight: 0, maxHeight: .infinity)
    }
}
```

SOME THOUGHS 

LIMITED

- LIMITED API COVERAGE
- LIMITED ADOPTION
- LIMITED SUPPORT

IMPLICIT WRAPS

```
VStack {
    Text("abc")
        .bold()
        .padding(.all)
}
```

WHY NOT THIS?

```
VStack {
    Padding {
        Text("abc").bold()
    }
}
```

CHILD PRIVACY INVASION

```
NavigationView {
    List {...}
        .navigationBarTitle(Text("Rooms"))
}
```

SMART DEFAULTS

```
HStack {
    Text("★★★★★")
    Text("Avocado Toast).font(.title)
    ...
}
```

SEE THE PADDING HERE? NO? BUT LOOK AT THE PICTURE!

DEMO

THANK YOU!

@PHUCLEDIEN