Transforming SwiftUI to JSON (and Vice-Versa) for a fully backend driven UI

"When you need that extra bit of flexibility"

360iDev 2021 - Florian Harr - Betterment

Hi👋

I’m Florian!

- i(Phone)OS since 2008/2009
- Remote @ Betterment
- **Hates to work on “Move the UIButton 5px left” stories**
- DMs open: @CaffeineFlo
- Obvious Dog Fanatic!

Betterment

- **Investment, IRA and 401(k) Provider** with best-in-class automated investing strategies (and human advisors!)
- Strong engineering team and culture with > 100 Engineers across the US
- Offices in NY, Philly, Denver
- Remote positions available
- We’re hiring (https://www.betterment.com/careers/current-openings/)

Transforming SwiftUI to JSON (and Vice-Versa)

for a full-backend driven UI

Entertaining

FUN

Informative

“When you need maximum flexibility

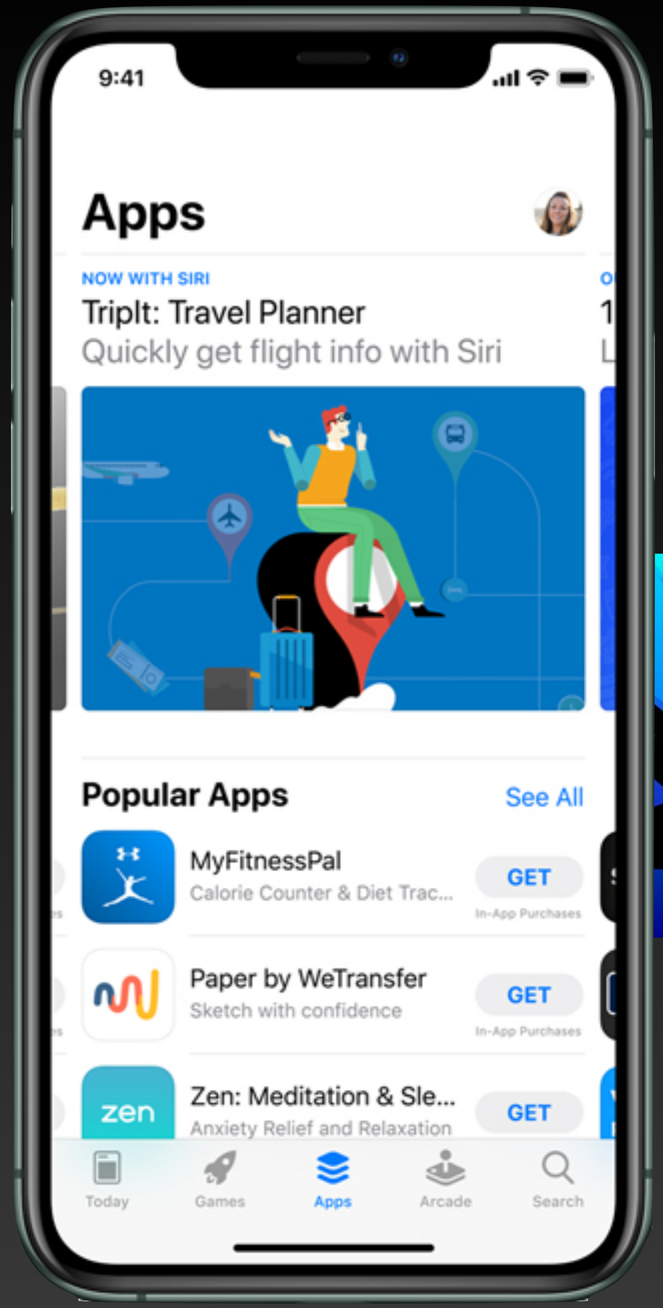
Florian Harr

~~JASON SUDEIKIS~~

TED LASSO

{JSON} +  Swift UI

{JSON}



Swift UI

```swift
struct ItemDetail: View {
    let item: MenuItem

    var body: some View {
        VStack {
            ZStack(alignment: .bottomTrailing) {
                Image(item.mainImage)
                    .resizable()
                    .scaledToFit()

                Text("Photo: \(item.photoCredit)")
                    .padding(4)
                    .background(Color.black)
                    .font(.caption)
                    .foregroundColor(.white)
                    .offset(x: -5, y: -5)
            }

            Text(item.description)
                .padding()

            Button("Order This") {
                order.add(item: item)
            }
            .font(.headline)

            Spacer()
        }
        .navigationTitle(item.name)
        .navigationBarTitleDisplayMode(.inline)
    }
}
// Credit: hackingwithswift.com - iDine - Paul Hudson
```

```swift
class Order: ObservableObject {
    @Published var items = [MenuItem]()

    var total: Int {
        if items.count > 0 {
            return items.reduce(0) { $0 + $1.price }
        } else {
            return 0
        }
    }

    func add(item: MenuItem) {
        items.append(item)
    }

    func remove(item: MenuItem) {
        if let index = items.firstIndex(of: item) {
            items.remove(at: index)
        }
    }
}

struct iDineApp: App {
    @StateObject var order = Order()

    var body: some Scene {
        WindowGroup {
            MainView()
                .environmentObject(order)
        }
    }
}
// Credit: hackingwithsiwft.com - iDine - Paul Hudson
```

```swift
struct HomeView: View {
    @ObservedObject var loginManager: LoginManager

    var body: some View {
        VStack {
            if let user = loginManager.loggedInUser {
                ProfileView(user: user)
            }

            ...
        }
    }
}
// Credit: SwiftBySundell.com - Configuring SwiftUI Views
```

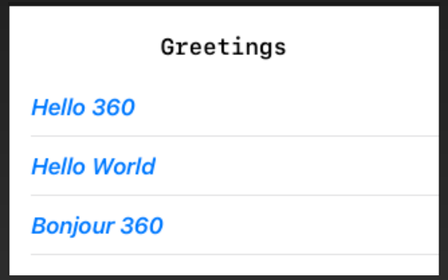
```swift
struct ListView: View {
    var title: String
    var items: [Item]
    @Binding var selectedItem: Item?

    var body: some View {
        VStack {
            Text(title).bold()
            List(items, selection: $selectedItem) { item in
                Text(item.title)
            }
        }.font(.system(.body, design: .monospaced))
    }
}
// Credit: SwiftBySundell.com - Configuring SwiftUI Views
```

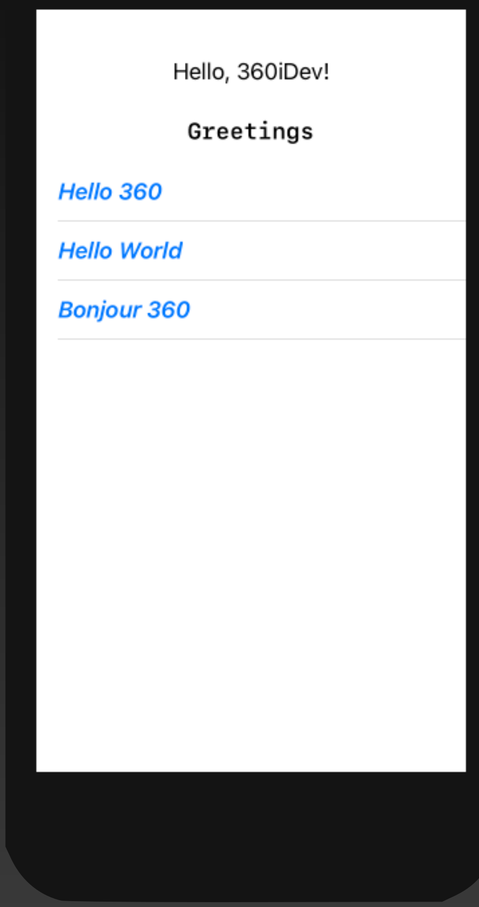
Parameterized SwiftUI Views

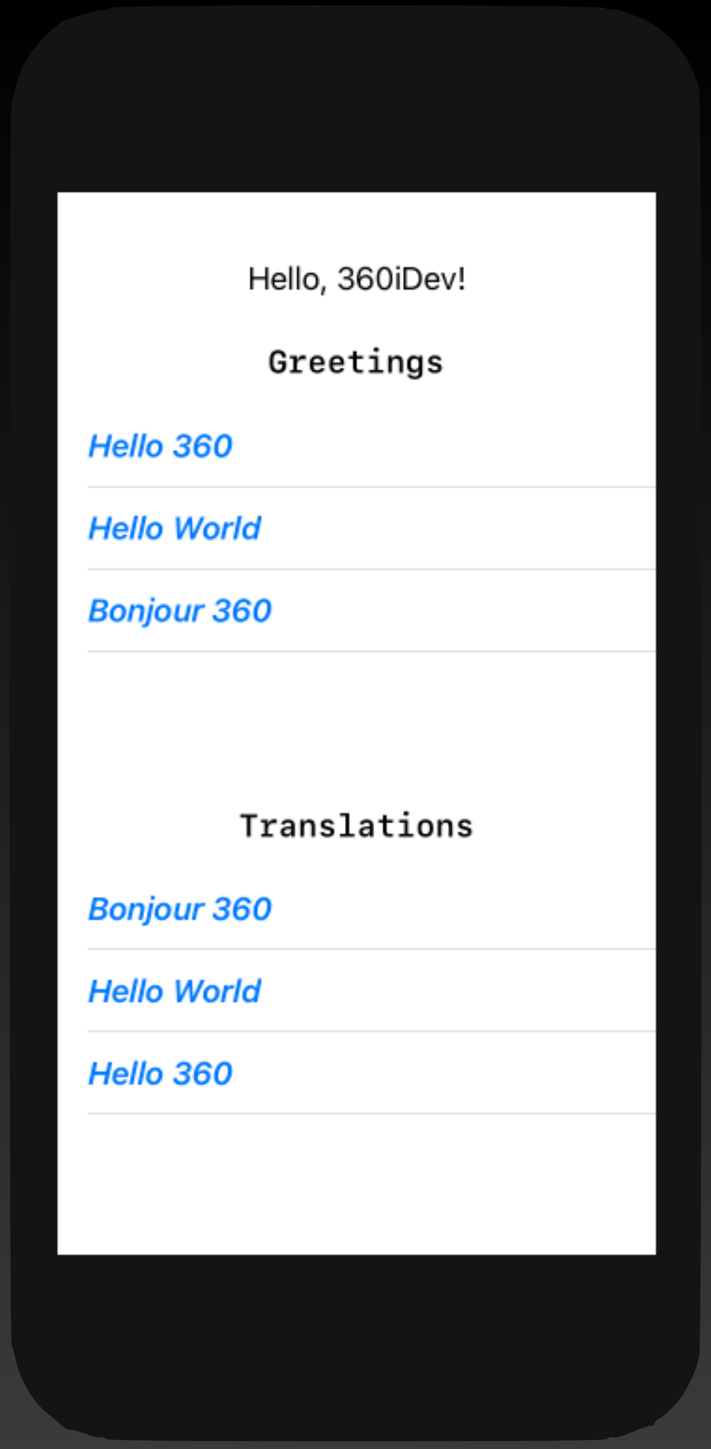
```swift
struct TitleView: View {
    var title: String

    var body: some View {
        Text(title)
            .font(.headline)
            ...
    }
}

var titleView = TitleView(title: "Hello 360iDev")
// Credit: SwiftBySundell.com - Configuring SwiftUI Views
```

Hello 360iDev

```swift
struct ListView: View {
    var title: String
    var items: [Item]
    @Binding var selectedItem: Item?

    var body: some View {
        VStack {
            Text(title).bold()
            List(items, selection: $selectedItem) { item in
                TitleView(title: item.title)
            }
        }.font(.system(.body, design: .monospaced))
    }
}
```







Parameterized SwiftUI Views

- Initializes Views from backing Data Structure (e.g. Codable JSON Models)
- Works nicely with mostly *“static”* screens
- Clear implementation experience

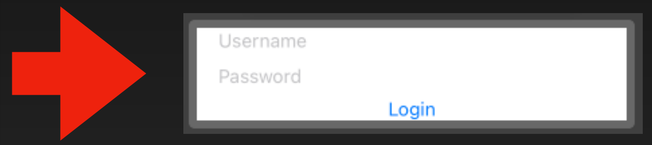
Parameterized SwiftUI Views

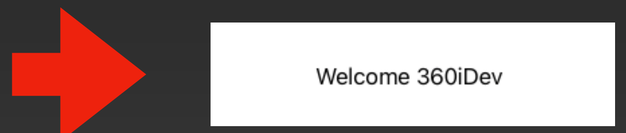
Follow Up Resources

- Configuring SwiftUI Views - SwiftBySundell
- How to create and compose custom views
- Getting started with UIKit in SwiftUI and vice versa

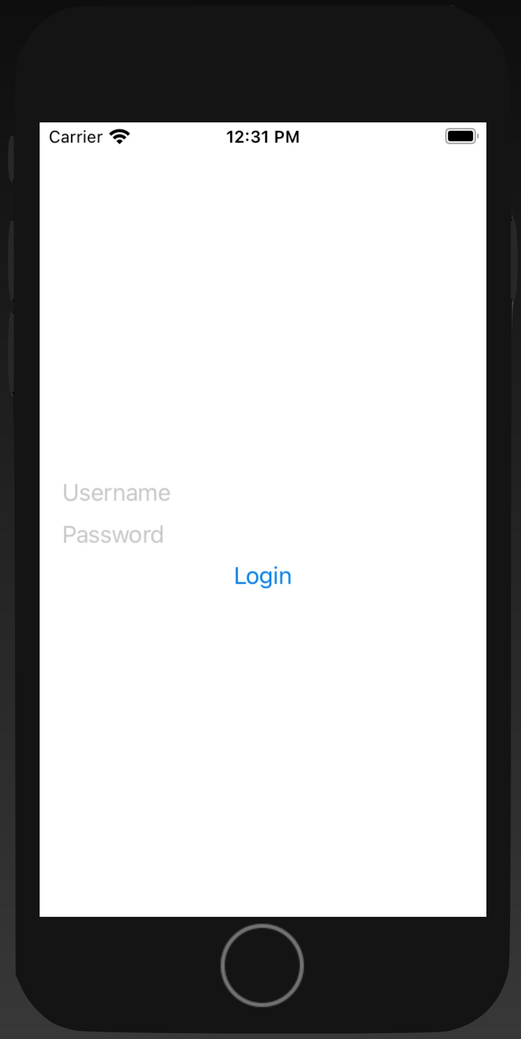
Dynamic Screens with shipped Views

```swift
struct LoginView: View {
    @ObservedObject var loginManager: LoginManager
    @State private var username: String = ""
    @State private var password: String = ""

    var body: some View {
        VStack {
            Group {
                TextField("Username", text: $username)
                TextField("Password", text: $password)
            }
            ...
            Button("Login") {
                loginManager.login(username: username, password: password)
            }
        }
    }
}

struct HomeView: View {
    @StateObject var loginManager = LoginManager()

    var body: some View {
        VStack {
            if let user = loginManager.loggedInUser {
                Text("Welcome \(user)")
            } else {
                LoginView(loginManager: loginManager)
            }
        }
    }
}
```



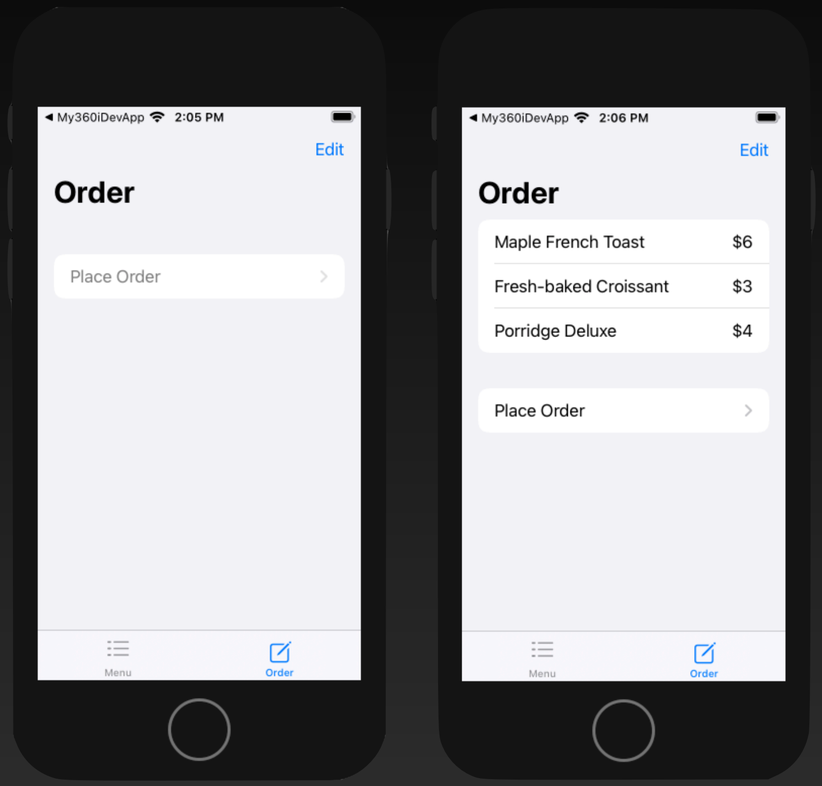


Based on Input:
Logic or API



Dynamic Screens with shipped Views

- Evolution to parameterized SwiftUI Views
- Capable to compose a whole screen from a set of shipped Views
- Can be as “dynamic” as consuming another API at runtime to describe full screens



HackingWithSwift - iDine

Dynamic Screens with shipped Views

Follow Up Resources

- Optional SwiftUI Views - SwiftBySundell
- 5 Steps to Better SwiftUI Views - HackingWithSwift
- Managing scenes in SwiftUI - Swift with Majid

Do you hate UI Bugs?

Would you like to update your UI without releasing a new version?

Would you rather spend your time on actual features that you interviewed for?

Apple certainly does ——->

...and I'll show you how you can do it better...

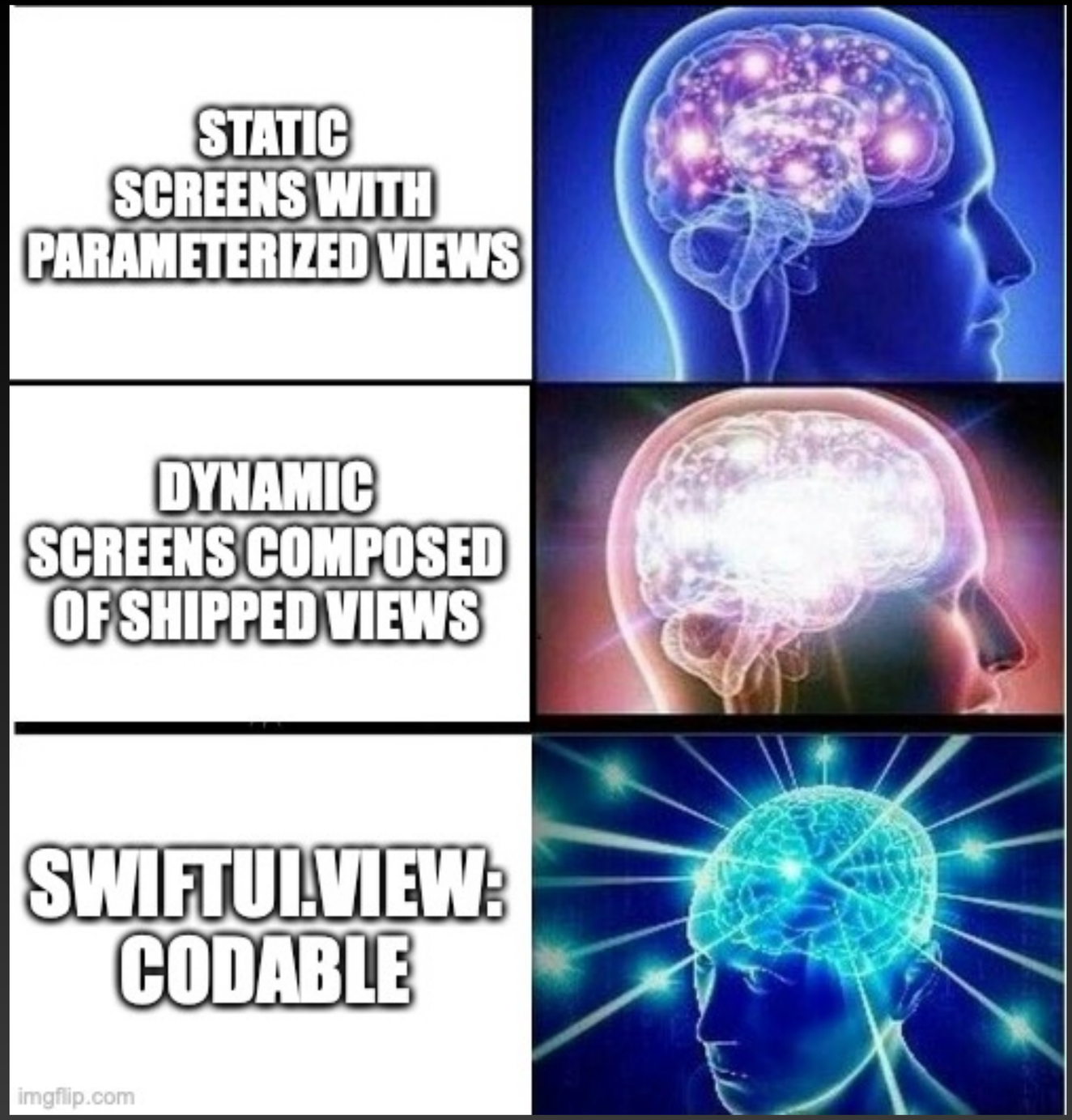
General

```
struct MyView: View, Codable {
    var id: UUID
    var title: String

    var body: some View {
        Text(title)
    }
}
```

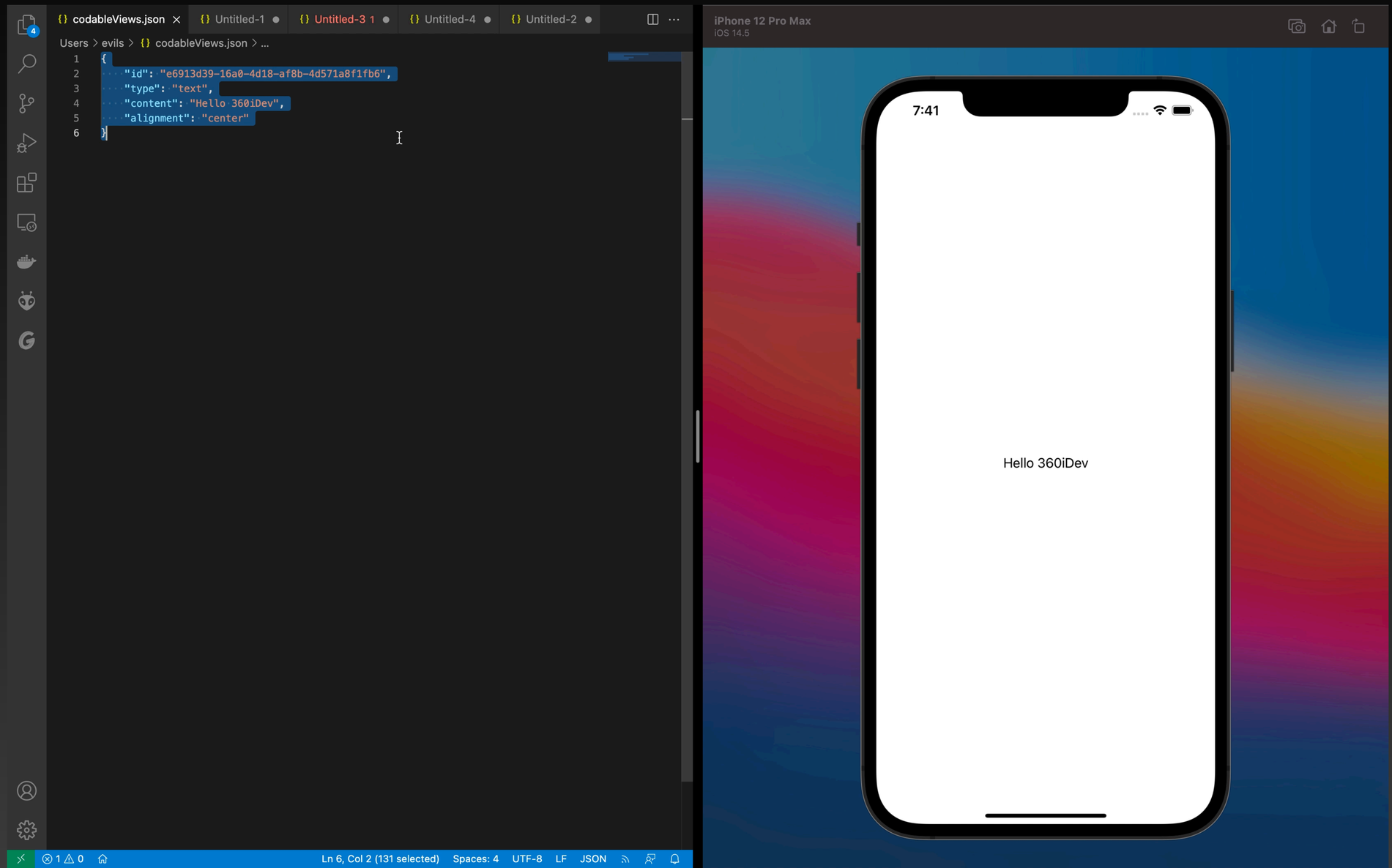
Specific

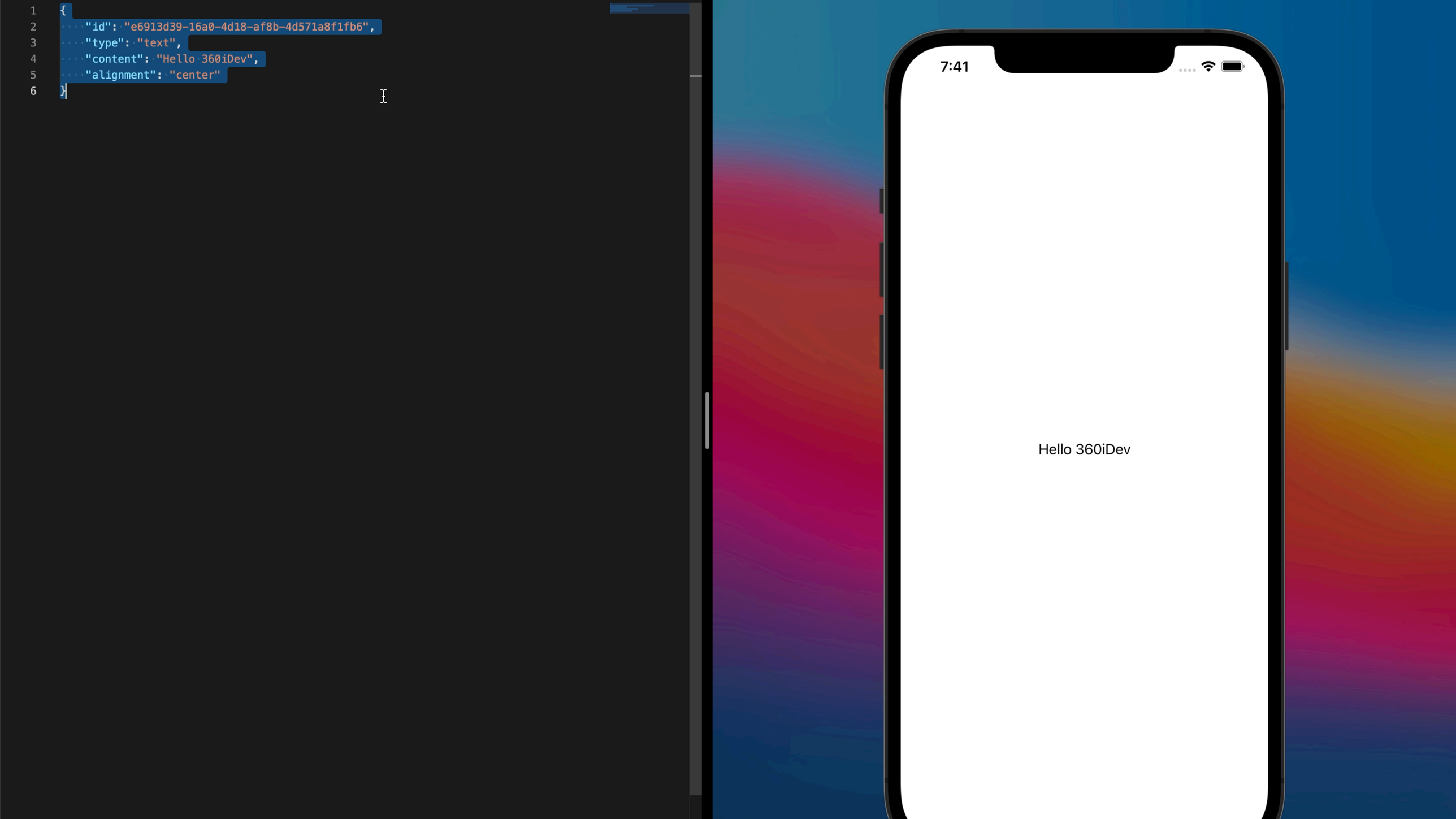
```
struct CodableText: Decodable {
    var alignment: TextAlignment
    var content: String
    var lineLimit: Int?

    public var body: some View {
        Text(content)
            .lineLimit(lineLimit)
            .multilineTextAlignment(alignment)
    }
}
```

1. Static Screens with parameterized Views
2. Dynamic Screens composed of shipped Views
3. SwiftUI.View: Codable

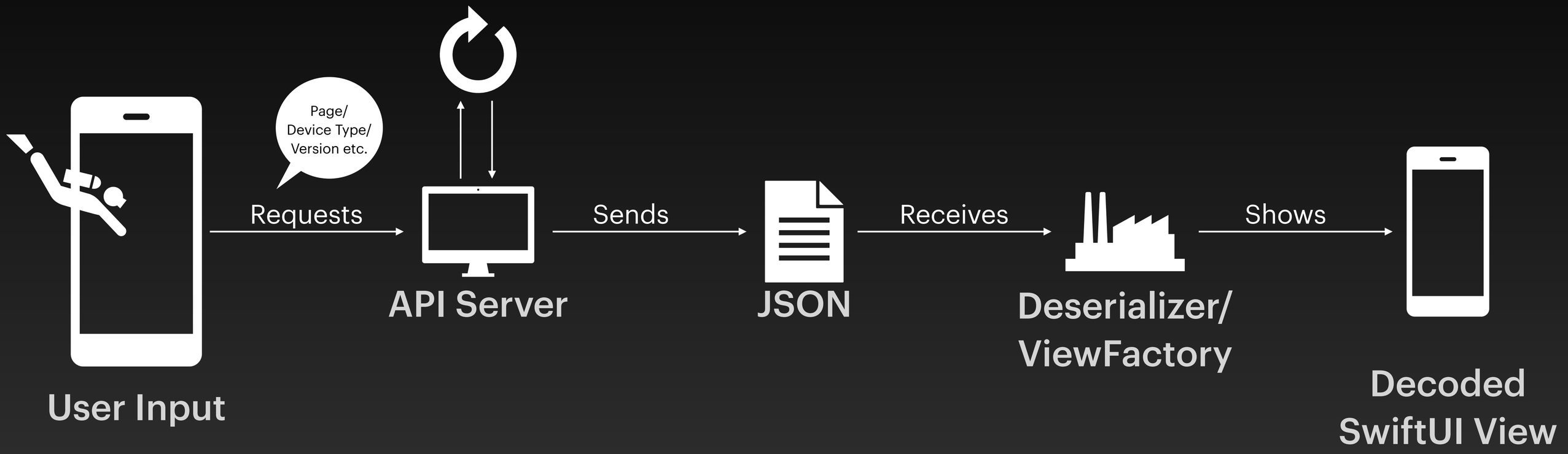


This one weird SwiftUI Trick ...




{
"id": "e6913d39-16a0-4d18-af8b-4d571a8f1fb6",
"type": "text",
"content": "Hello 360iDev",
"alignment": "center"
}
7:41
Hello 360iDev

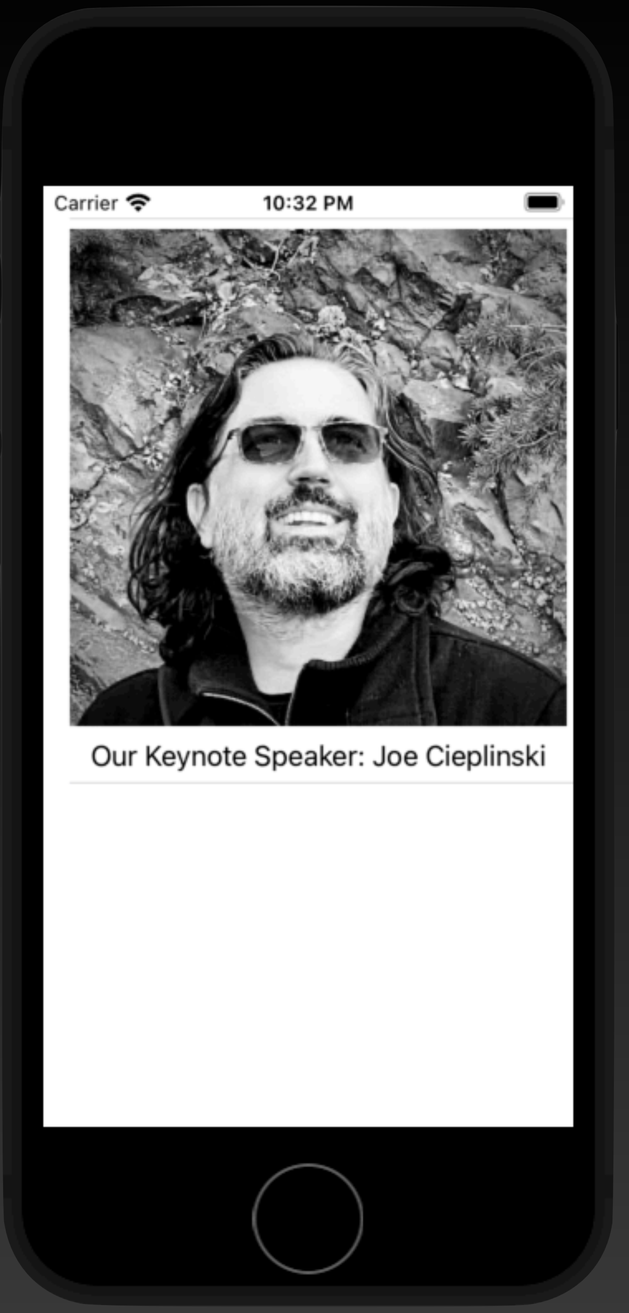

JSON
Deserializer/
ViewFactory
Decoded
SwiftUI View


Page/
Device Type/
Version etc.
Requests
Sends
Receives
Shows
User Input
API Server
JSON
Deserializer/
ViewFactory
Decoded
SwiftUI View

```
{
    "type": "list",
    "id": "3a0fdf2d-aa76-40e2-b584-537bbd6f25af",
    "padding": [0, 0, 0, 0],
    "components": [
        {
            "type": "vStack",
            "id": "fc586d08-91fe-43a3-8da6-98a14b172bbc",
            "alignment": "leading",
            "components": [
                {
                    "type": "image",
                    "id": "49b7a18c-3f8a-4732-bbd0-26feed28e3e7",
                    "aspectRatio": 1.5,
                    "url":
"https://360idev.com/.../2qa8uykx4JOMrvU01Eu4LmY2SLz2-300x300.jpg",
                },
                {
                    "type": "text",
                    "id": "e6913d39-16a0-4d18-af8b-4d571a8f1fb5",
                    "content": "Our Keynote Speaker: Joe Cieplinski",
                    "fontFamily": "Escrow Condensed",
                    "fontWeight": 700,
                    "fontStyle": "normal",
                    "fontSize": 32
                }
            ]
        }
    ]
}
```


Our Keynote Speaker: Joe Cieplinski

```
{
    "type": "list",
    "id": "3a0fdf2d-aa76-40e2-b584-537bbd6f25af",
    "padding": [0, 0, 0, 0],
    "components": [
        {
            "type": "vStack",
            "id": "fc586d08-91fe-43a3-8da6-98a14b172bbc",
            "alignment": "leading",
            "components": [
                {
                    "type": "image",
                    "id": "49b7a18c-3f8a-4732-bbd0-26feed28e3e7",
                    "aspectRatio": 1.5,
                    "url":
"https://360idev.com/.../2qa8uykx4JOMrvU01Eu4LmY2SLz2-300x300.jpg",
                },
                {
                    "type": "text",
                    "id": "e6913d39-16a0-4d18-af8b-4d571a8f1fb5",
                    "content": "Our Keynote Speaker: Joe Cieplinski",
                    "fontFamily": "Escrow Condensed",
                    "fontWeight": 700,
                    "fontStyle": "normal",
                    "fontSize": 32
                }
            ]
        }
    ]
}
```

Bonus: Abstraction

Allows for Model Variants with dynamicMemberLookup

```
protocol CodableViewVariant {
    var id: UUID { get }
    var padding: [Int] { get }
}

enum CodableView {
    case list(CodableList)
    case text(CodableText)
    case button(CodableButton)
}
```

CodableView

```
public struct TextComponent: Decodable, CodableViewVariant {
    public let id: UUID
    public var content: String
    public var fontFamily: String
    public var fontWeight: String
    public var fontStyle: FontStyle
    public var fontSize: CGFloat
    public var lineLimit: Int?
    public var alignment: TextAlignment
}
```

```
public struct TextView: View {
    public var model: TextComponent

    public var body: some View {
        Text(model.content)
            .lineLimit(model.lineLimit)
            .applyCommonTextModifiers(from: model)
    }
}
```

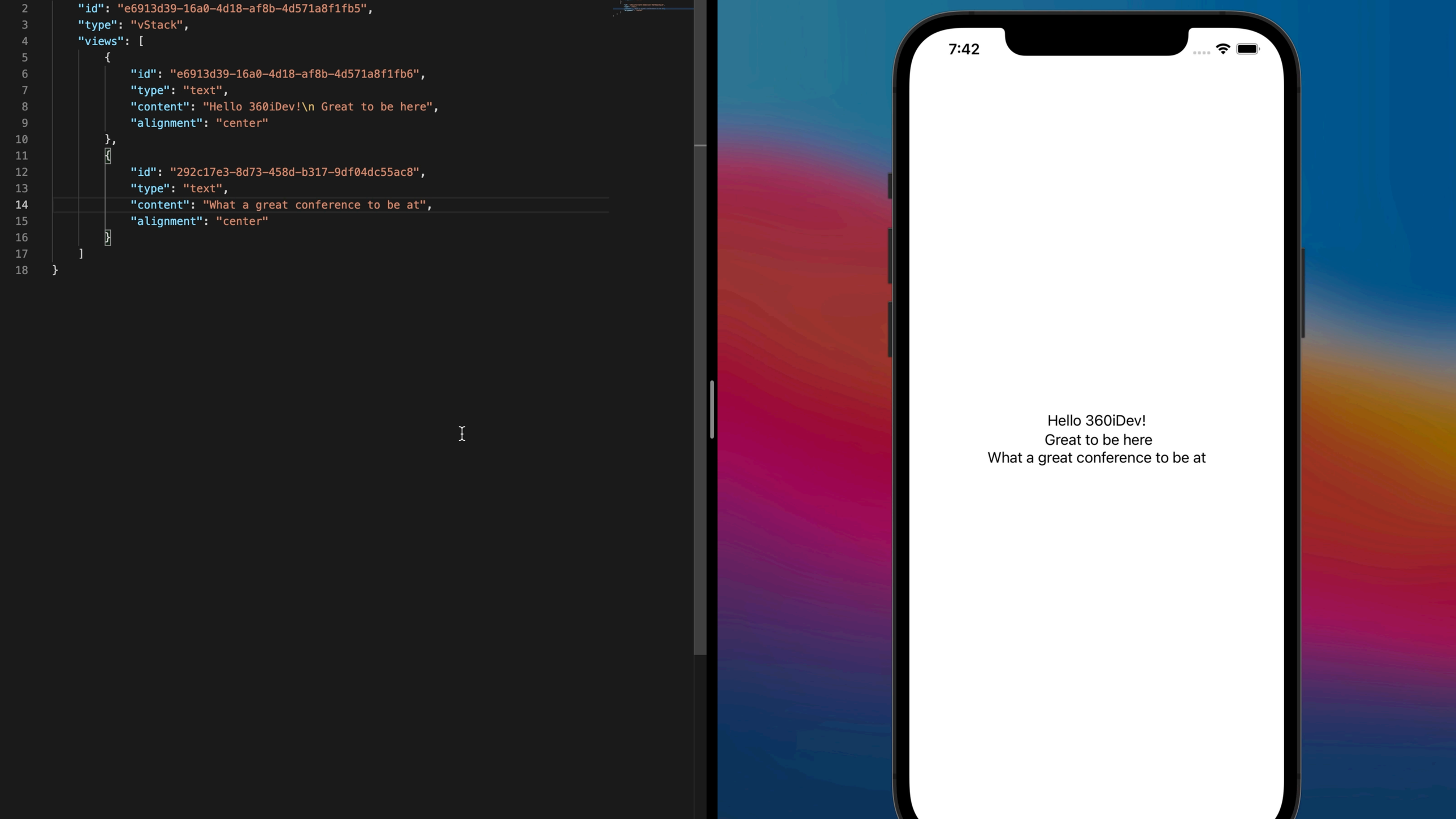
Deserializer

```
extension CodableView: Decodable {
    enum CodingKeys: CodingKey {
        case type
    }

    init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        let type = try container.decode(String.self, forKey: .type)

        switch type {
        case "list":
            self = .list(try CodableList(from: decoder))
        case "text":
            self = .text(try CodableText(from: decoder))
        case "button":
            self = .button(try CodableButton(from: decoder))
        default:
            fatalError("Unknown Type received")
        }
    }
}
```

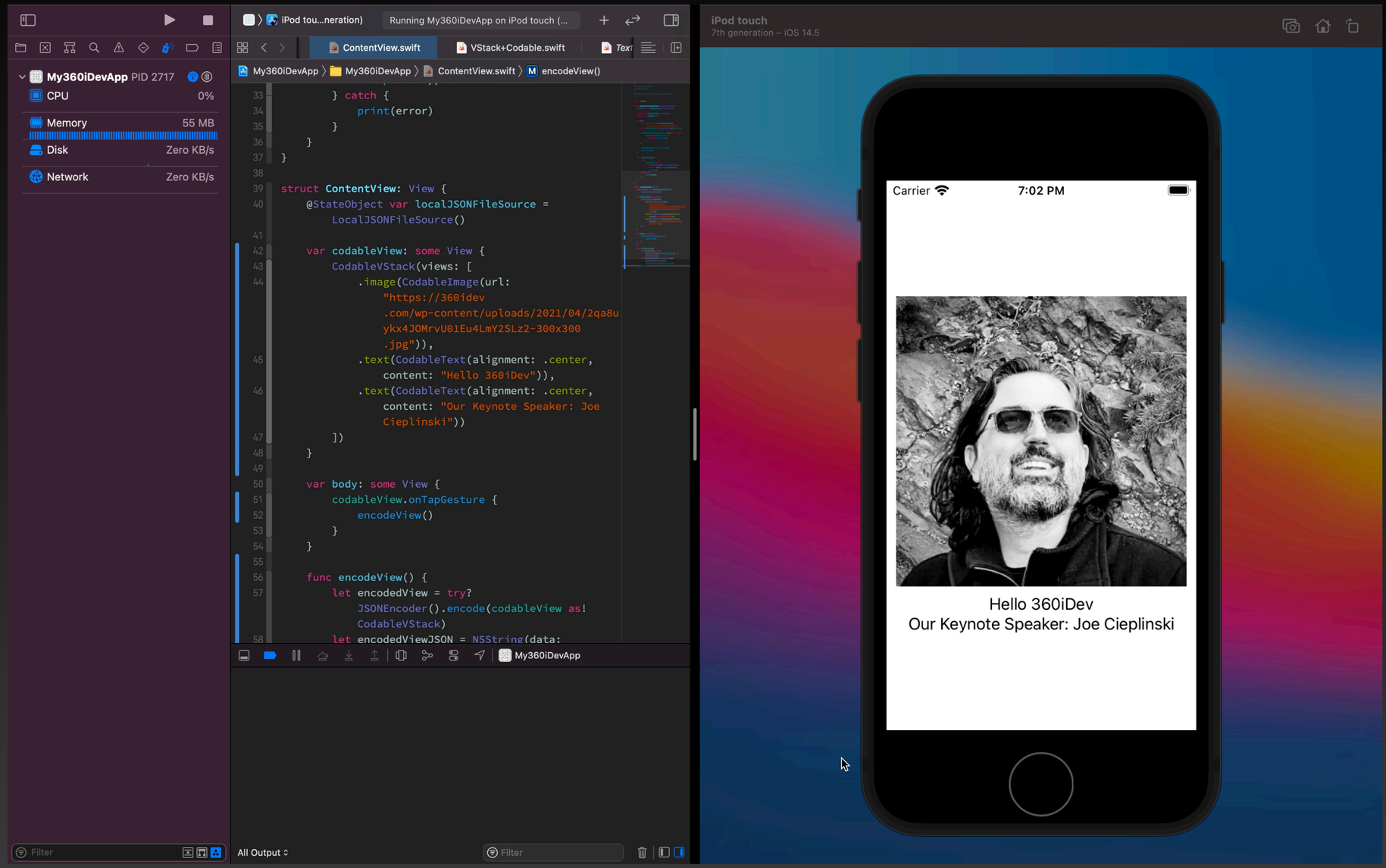
https://www.swiftbysundell.com/articles/handling-model-variants-in-swift/

ViewFactory

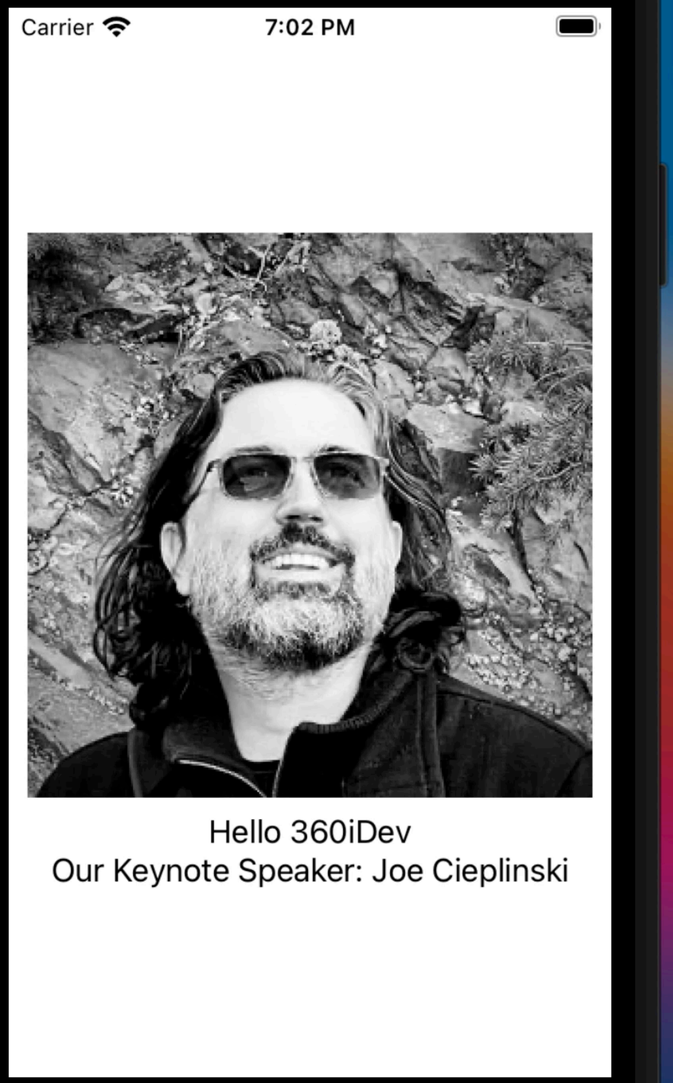
```
public struct ViewFactory {
    /// Constructs a `View` from a `Component`.
    ///
    /// - Parameter component: The component
    /// from which to construct a `View`.
    /// - Returns: A `View` constructed
    /// from the given `Component`.
    @ViewBuilder static func view(for
                                  codableView: CodableView)
                                                -> some View {
        switch codableView.type {
        case let .list(list):
            ListView(model: list)
        case let .text(text):
            TextView(model: text)
        case let .button(button):
            ButtonView(model: button)
        }
    }
}
```


"id": "e6913d39-16a0-4d18-af8b-4d571a8f1fb5",
"type": "vStack",
"views": [
{
"id": "e6913d39-16a0-4d18-af8b-4d571a8f1fb6",
"type": "text",
"content": "Hello 360iDev!\n Great to be here",
"alignment": "center"
},
{
"id": "292c17e3-8d73-458d-b317-9df04dc55ac8",
"type": "text",
"content": "What a great conference to be at",
"alignment": "center"
}
]
}
7:42
Hello 360iDev!
Great to be here
What a great conference to be at

Creating Valid SwiftUI JSON Code



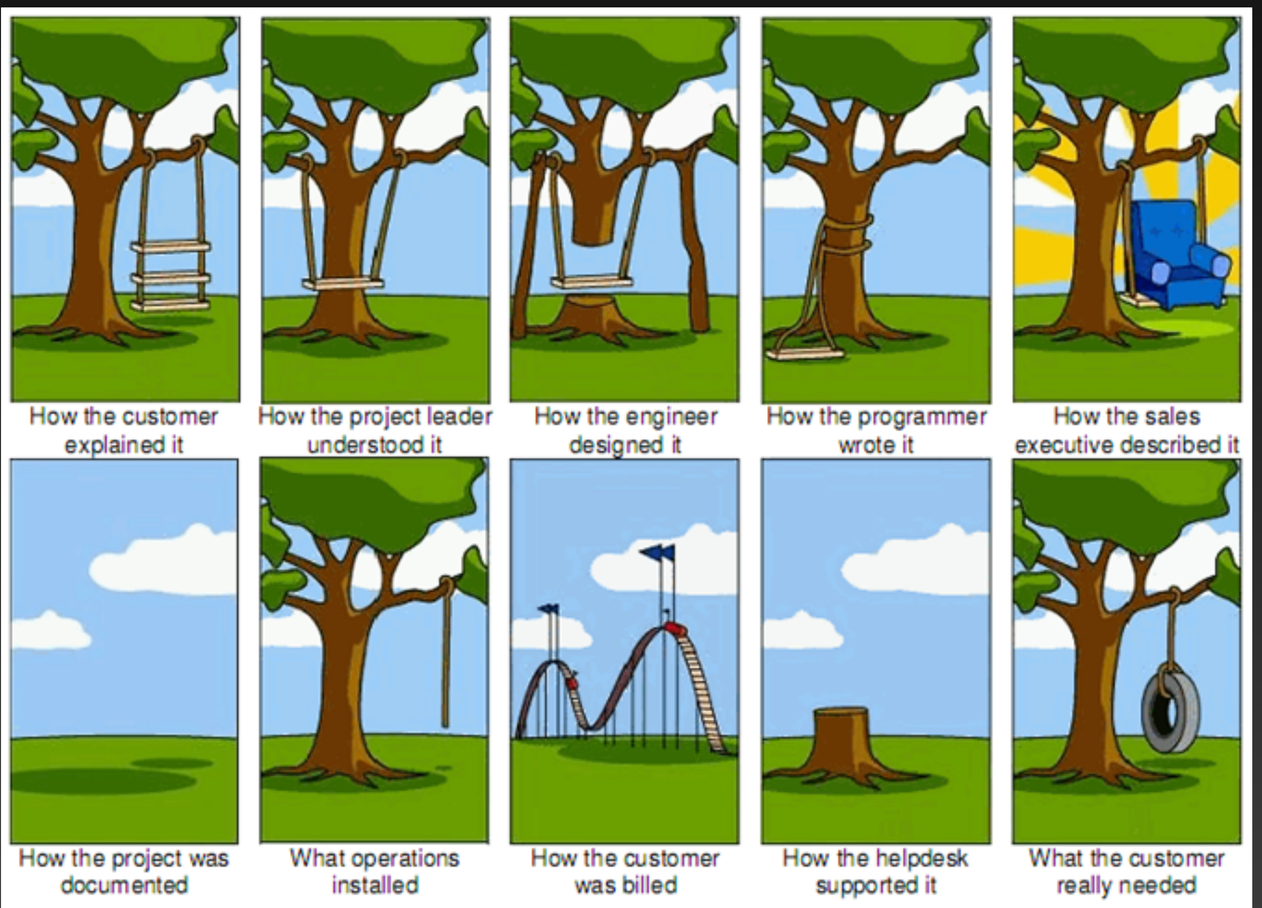
```
} catch {
    print(error)
}
}
}

struct ContentView: View {
    @StateObject var localJSONFileSource =
        LocalJSONFileSource()

    var codableView: some View {
        CodableVStack(views: [
            .image(CodableImage(url:
                "https://360idev
                .com/wp-content/uploads/2021/04/2qa8u
                ykx4JOMrvU01Eu4LmY2SLz2-300x300
                .jpg")),
            .text(CodableText(alignment: .center,
                content: "Hello 360iDev")),
            .text(CodableText(alignment: .center,
                content: "Our Keynote Speaker: Joe
                Cieplinski"))
        ])
    }

    var body: some View {
        codableView.onTapGesture {
            encodeView()
        }
    }

    func encodeView() {
        let encodedView = try?
            JSONEncoder().encode(codableView as!
            CodableVStack)
        let encodedViewJSON = NSString(data:
```



- Encode what you need
- *Encodable* extensions for standard views
- Grab the JSON output form the console or write more advanced tooling

```
func encode(to encoder: Encoder) throws {
    var container = encoder.container(keyedBy: CodingKeys.self)

    switch self {
    case let .button(codableButton):
        try container.encode("button", forKey: .type)
        try codableButton.encode(to: encoder)
    case let .image(codableImage):
        try container.encode("image", forKey: .type)
        try codableImage.encode(to: encoder)
    case let .list(codableList):
        try container.encode("list", forKey: .type)
        try codableList.encode(to: encoder)
    case let .text(codableText):
        try container.encode("list", forKey: .type)
        try codableText.encode(to: encoder)
    case let .vStack(codableVStack):
        try container.encode("vStack", forKey: .type)
        try codableVStack.encode(to: encoder)
    }
}
```

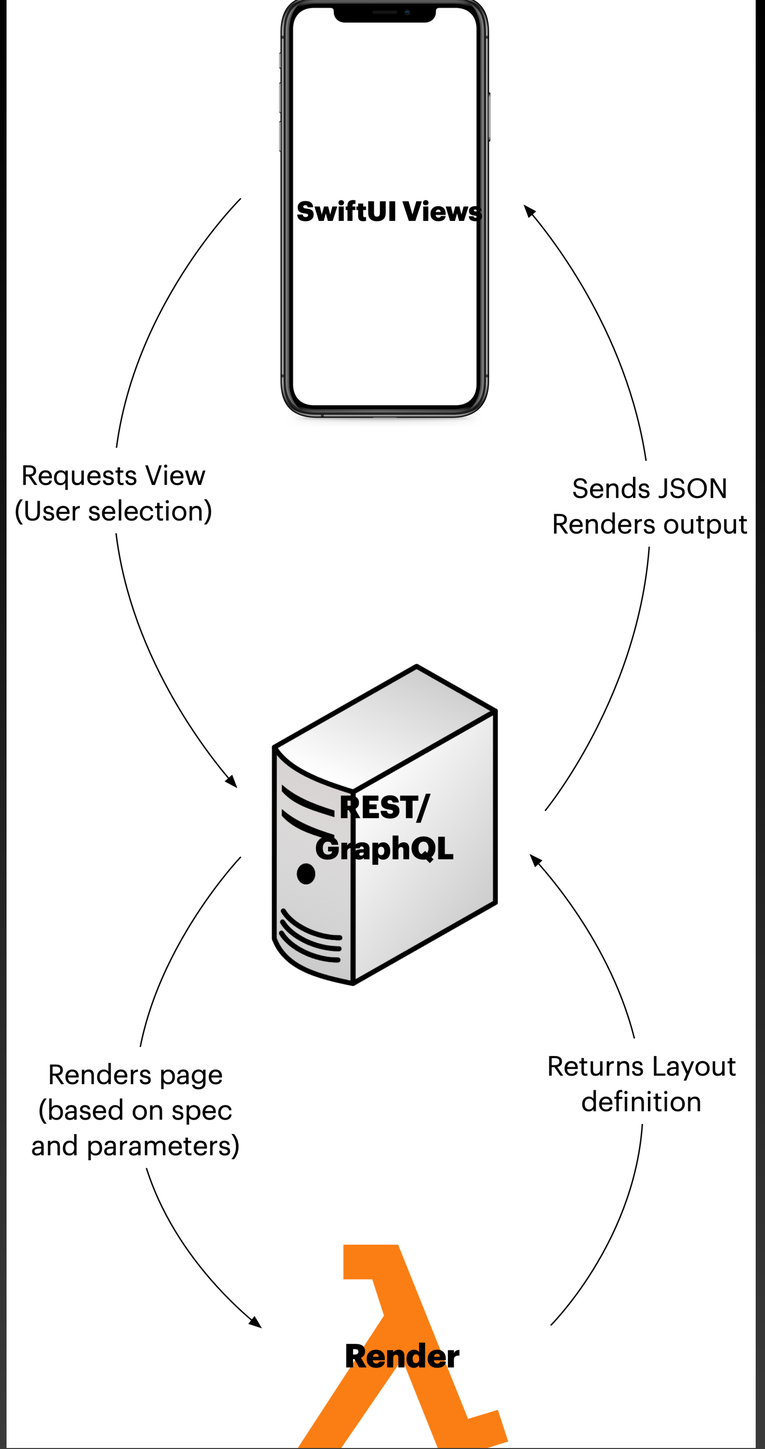
Pros


How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

&


ANOTHER PRO AND CON LIST?
I'M GOING TO KILL MYSELF.

Cons



Re-imagine App Screens On-The-Fly

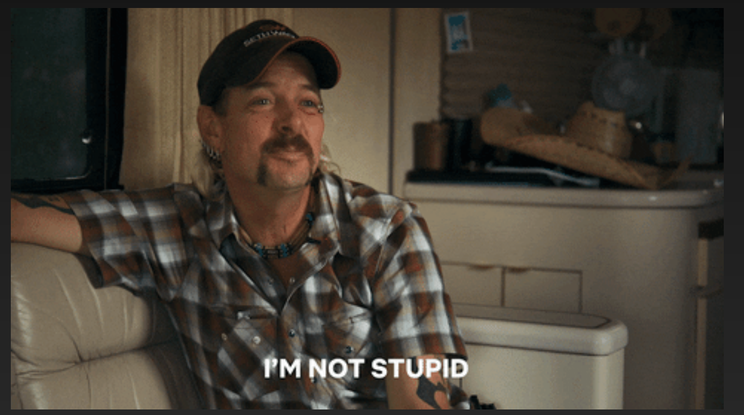
Server Side Rendering isn’t a brand-new technology and has been used on (reactive) web for a long time.

There are many use-cases and benefits where this architecture enables a flexibility that is hard to replicate otherwise. It comes with trade-offs though and SwiftUI (in it’s current state) requires a good bit of boilerplate code to make it work.

Measure Once, Cut Twice or Measure Twice, Cut Once?

Not a general purpose solution for all situations.
Adds complexity, need for maintenance and additional edge cases.

Evaluate your needs properly and see if your tools (and technologies) fit your needs.



Lessons Learned

- Every “layer” adds exponential complexity
- “Thin” translation layer (JSON) isn’t that thin
- Iterate quick early on
- Move as much to the backend as possible
- Build strong tooling suit to help debugging and communication

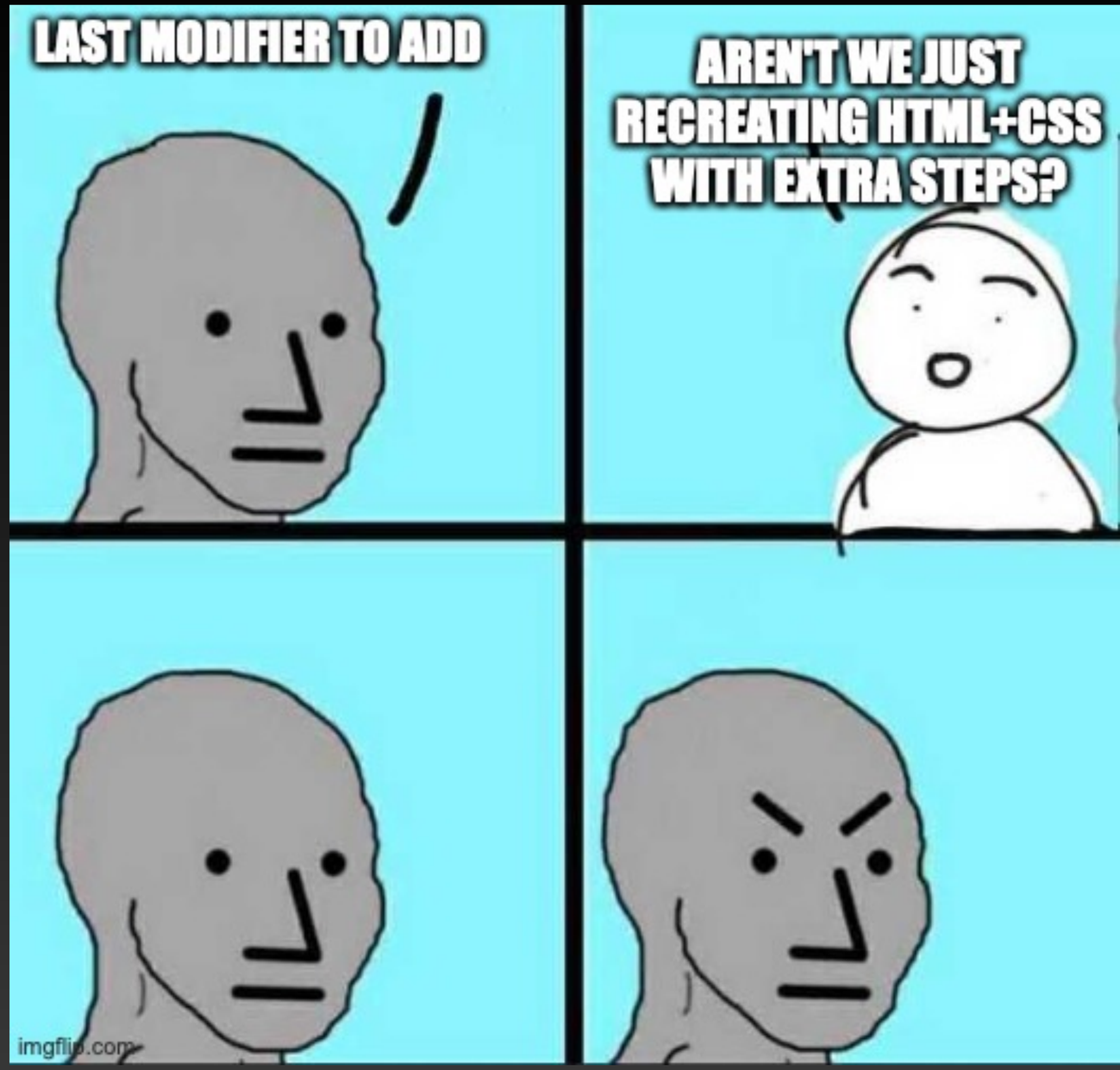
K.I.S.S

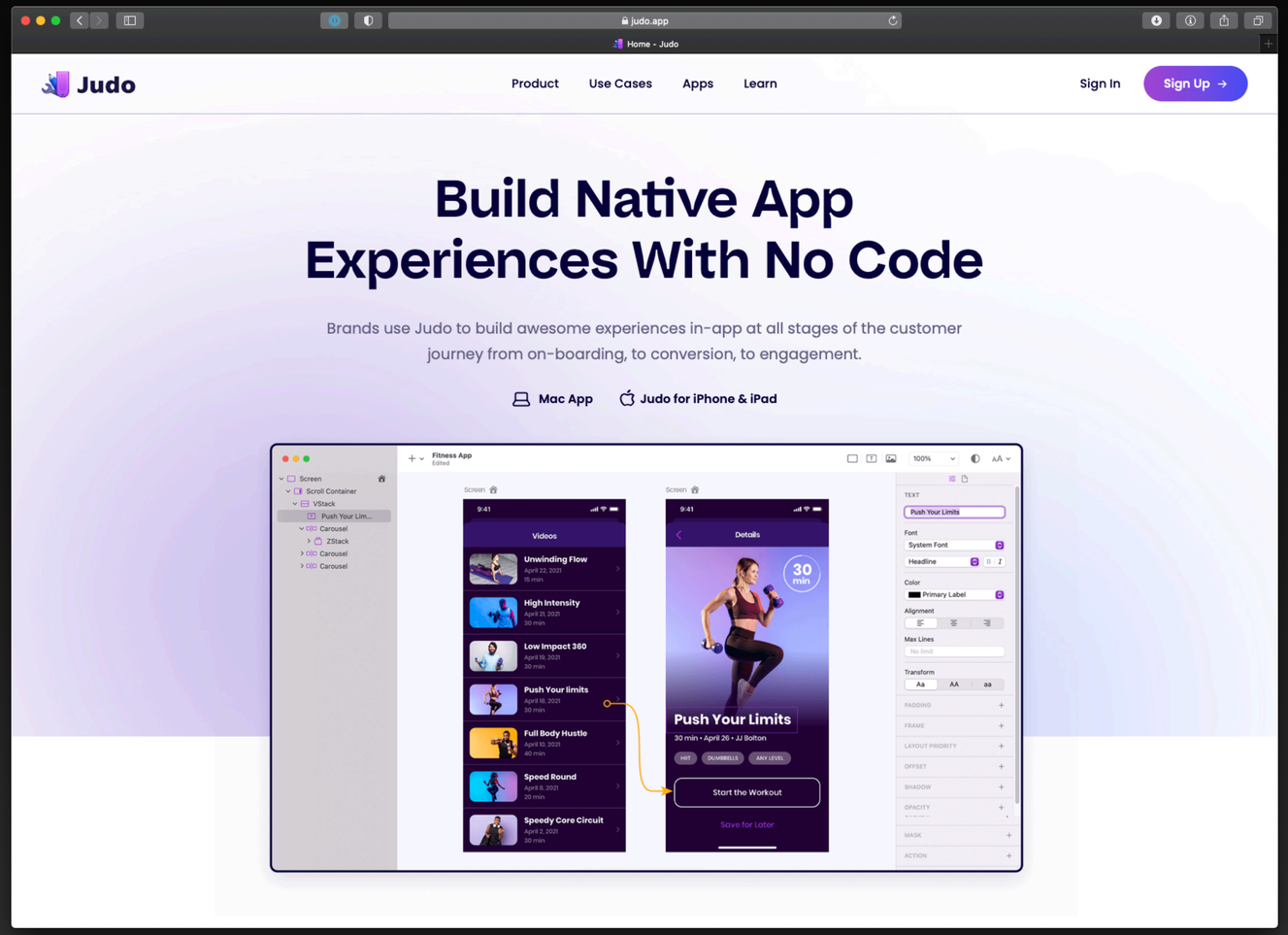
Keep it simple, stupid

K.I.S.S

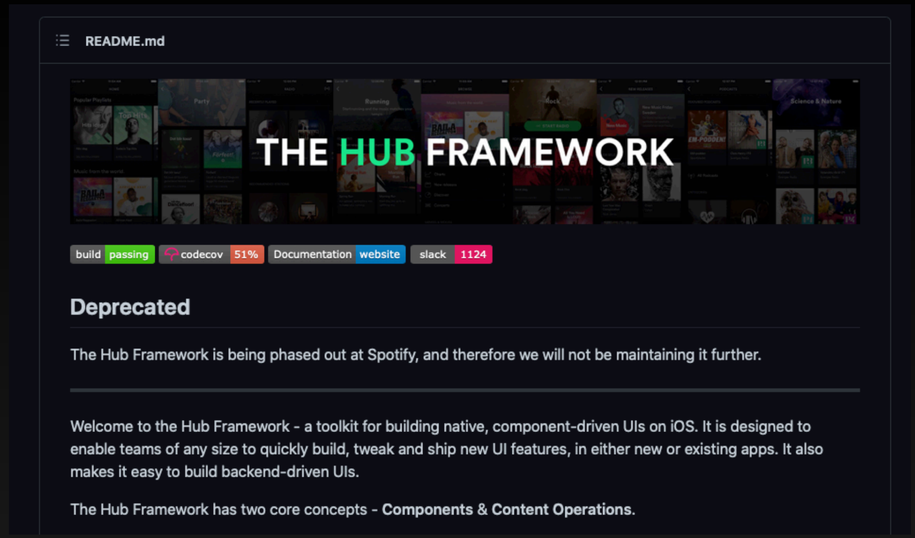
Keep it simple, stupid (where possible)

- Each view, parameter and modifier adds complexity/edge cases/potential outcomes
- Versioning will eventually add another layer on top
- “Actions” or function calls can be extremely complex and potentially dangerous
- Technologies often solve specific problems, double check your use-cases

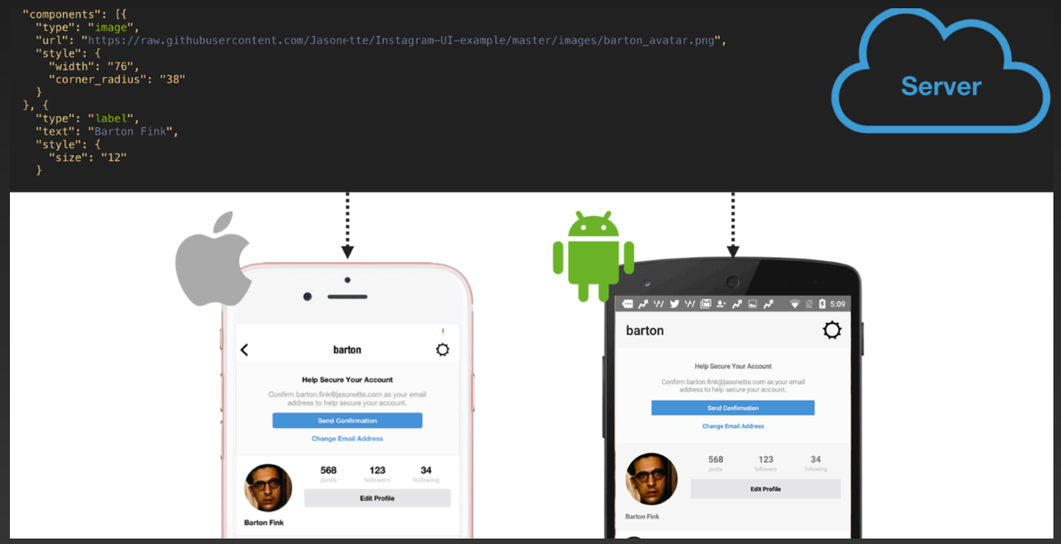




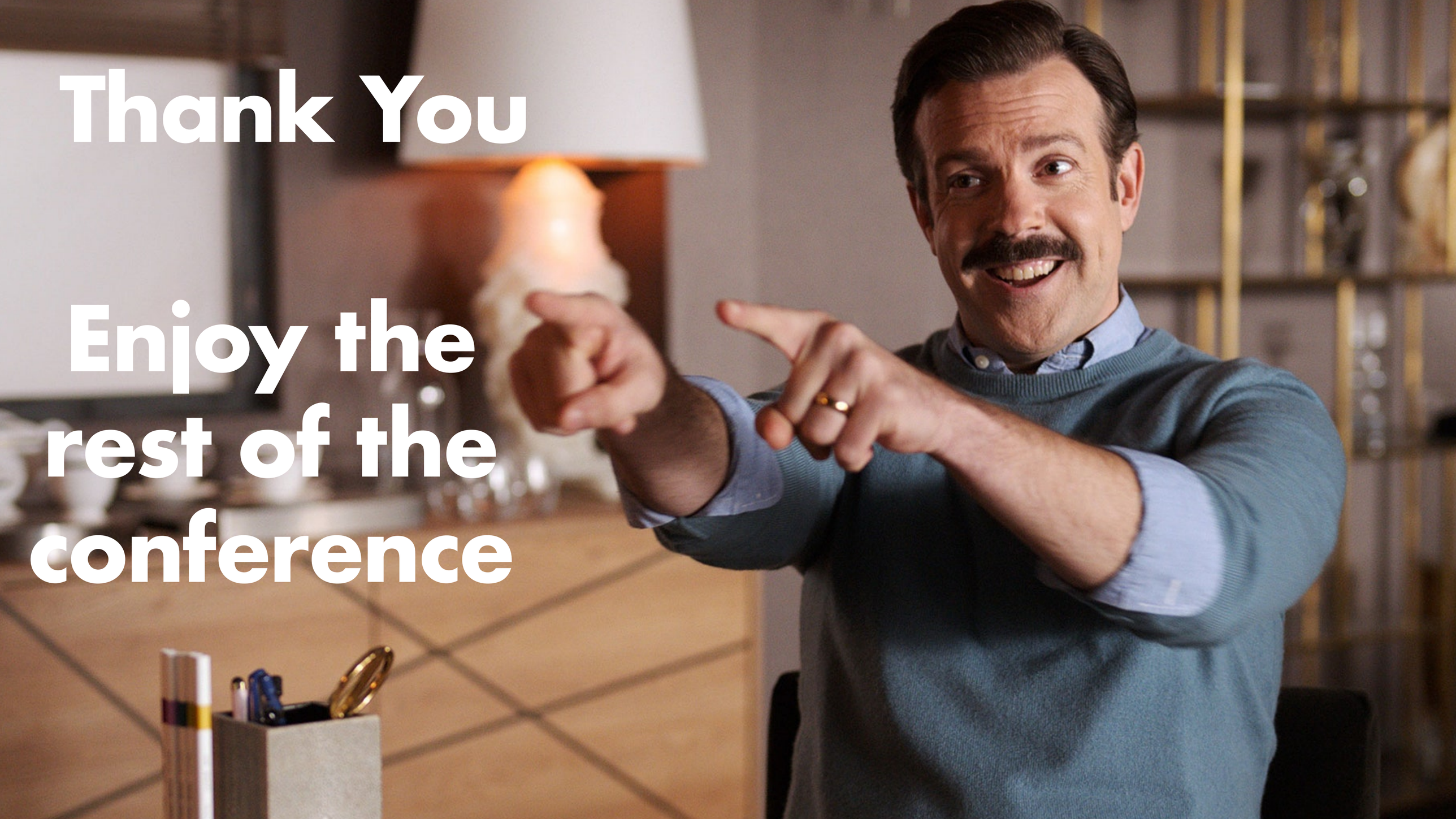
judo.app



Spotify HUB Framework



jasonette.com


Thank You
Enjoy the rest of the conference

- @CaffeineFlo on Twitter
- 360iDev Slack - CaffeineFlo
- GitHub Repo - https://github.com/caffeineflo/360iDev2021

Link to business card

Questions ?