STRV

ARCHITECTURES DESIGN SYSTEMS

ARCHITECTURE PATTERNS

01

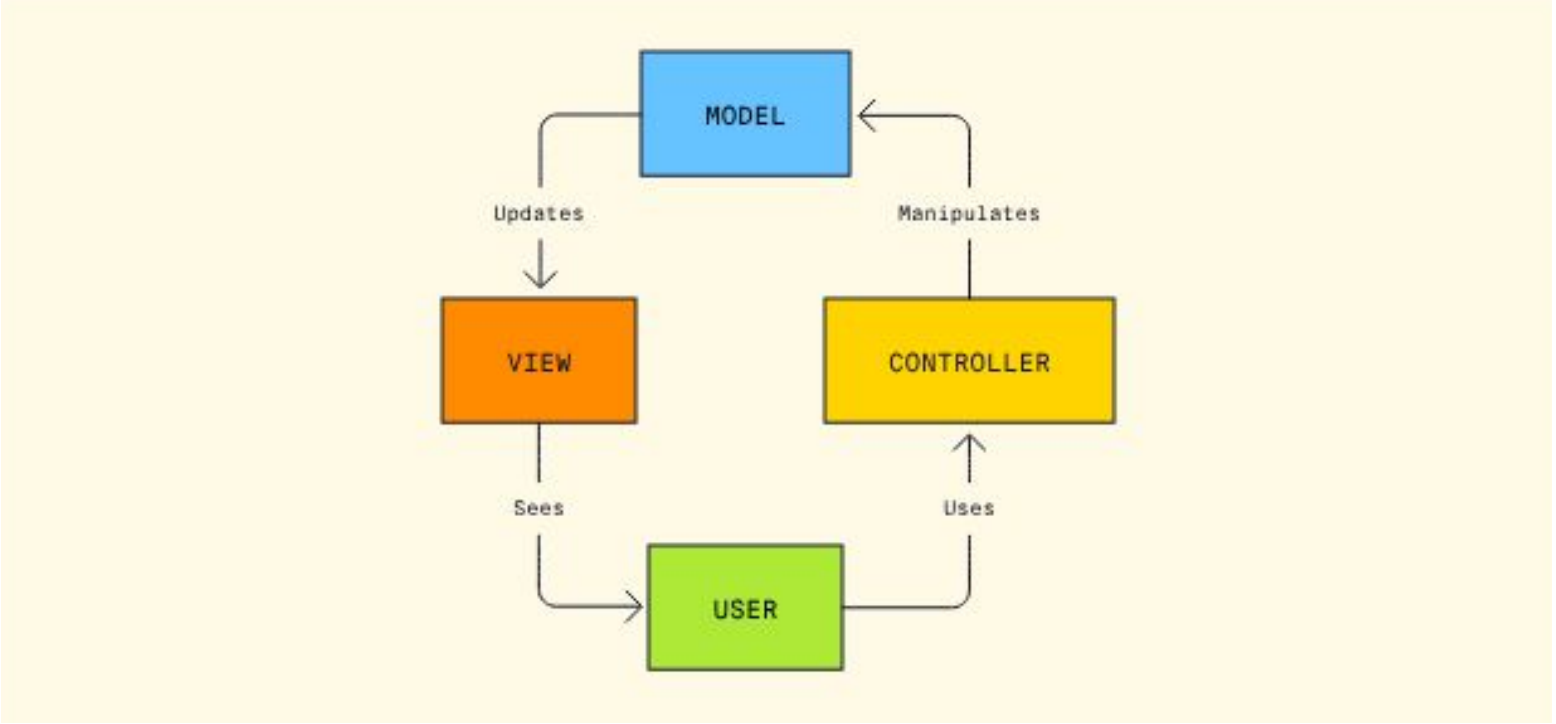
MOTIVATION

- Why should we be concerned about design patterns?
- Easier to add changes / new features
- Working in teams
- It’s not just about folder structure
- **SOLID**

SOLID

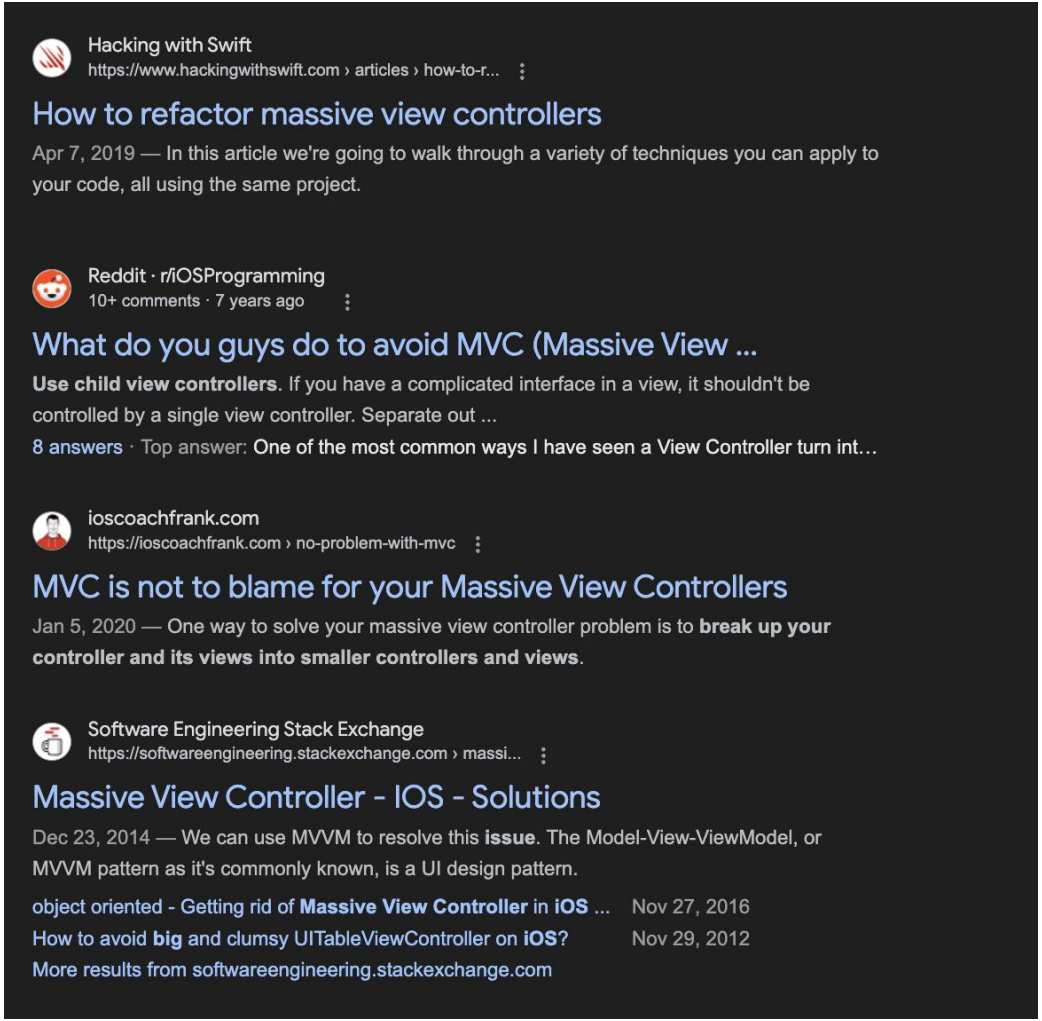
- **S**ingle responsibility principle - class should only do one thing
- **O**pen-closed principle - open for extension closed for modification
- **L**iskov substitution principle - subclass B of class A should be able to replace A without breaking logic
- **I**nterface segregation - split interfaces into multiple ones to avoid implementing things we don’t need
- **D**ependency inversion - classes should depend on abstractions, not concrete implementations

MVC - MODEL VIEW CONTROLLER



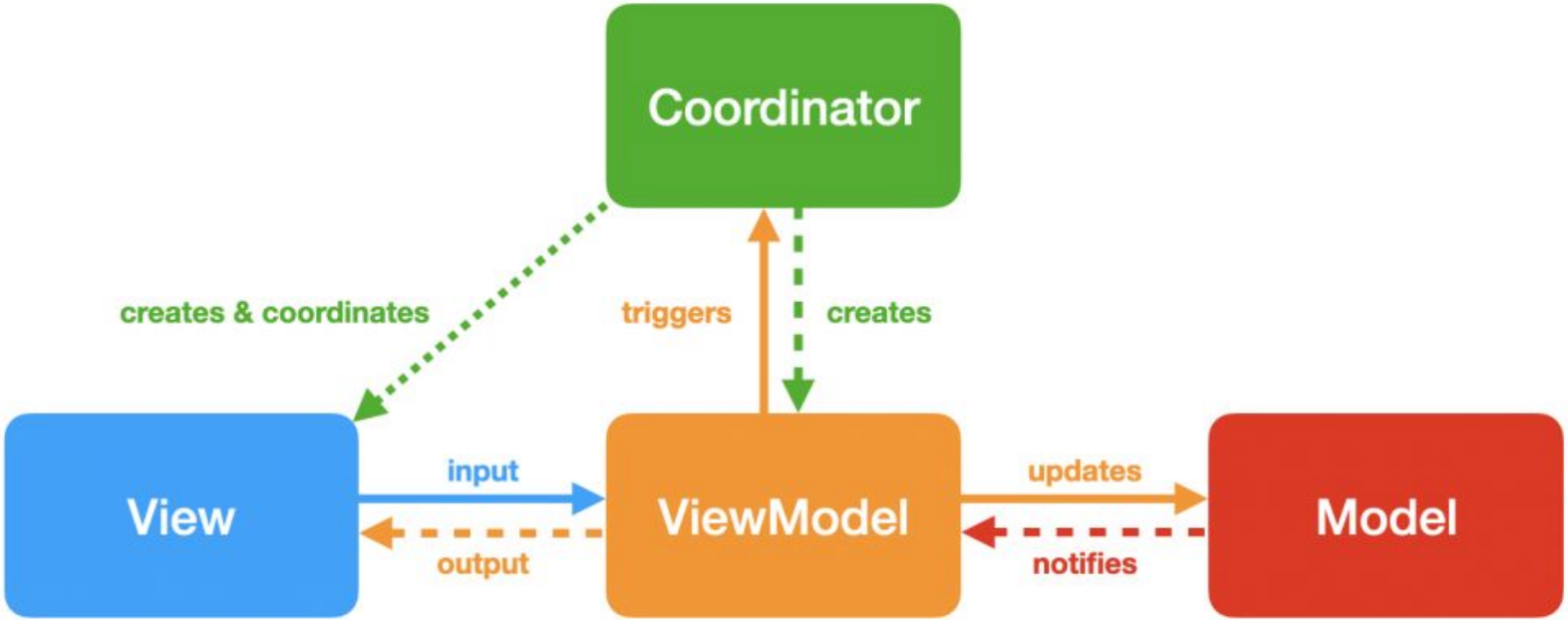
PROBLEM

IT GETS **MASSIVE**


Hacking with Swift
https://www.hackingwithswift.com › articles › how-to-r...
How to refactor massive view controllers
Apr 7, 2019 — In this article we're going to walk through a variety of techniques you can apply to your code, all using the same project.
Reddit · r/iOSProgramming
10+ comments · 7 years ago
What do you guys do to avoid MVC (Massive View ...
Use child view controllers. If you have a complicated interface in a view, it shouldn't be controlled by a single view controller. Separate out ...
8 answers · Top answer: One of the most common ways I have seen a View Controller turn int…
ioscoachfrank.com
https://ioscoachfrank.com › no-problem-with-mvc
MVC is not to blame for your Massive View Controllers
Jan 5, 2020 — One way to solve your massive view controller problem is to break up your controller and its views into smaller controllers and views.
Software Engineering Stack Exchange
https://softwareengineering.stackexchange.com › massi...
Massive View Controller - IOS - Solutions
Dec 23, 2014 — We can use MVVM to resolve this issue. The Model-View-ViewModel, or MVVM pattern as it's commonly known, is a UI design pattern.
object oriented - Getting rid of Massive View Controller in iOS ... Nov 27, 2016
How to avoid big and clumsy UITableViewController on iOS? Nov 29, 2012
More results from softwareengineering.stackexchange.com

MVVM - MODEL VIEW VIEWMODEL

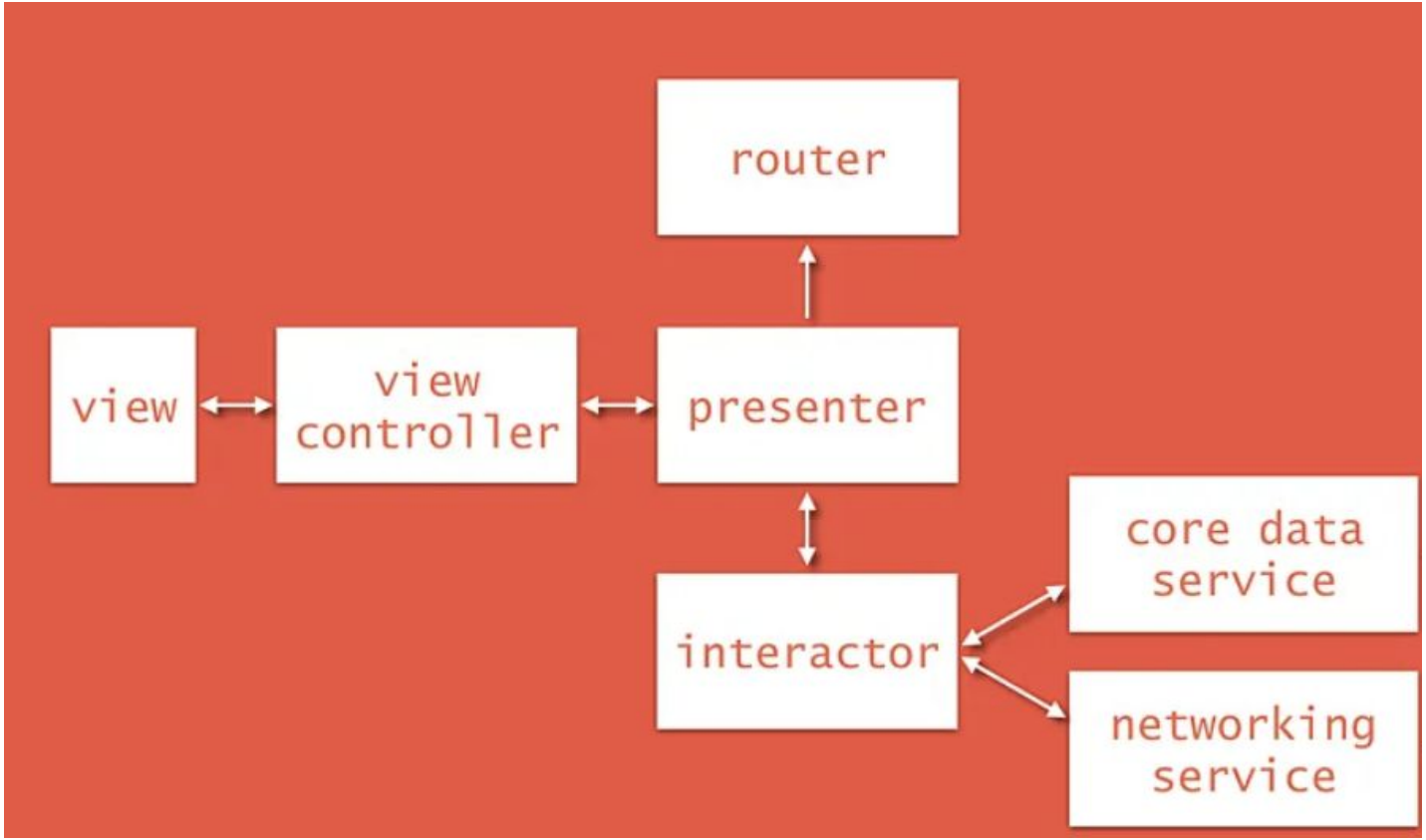
MVVM-C - MODEL VIEW VIEWMODEL COORDINATOR



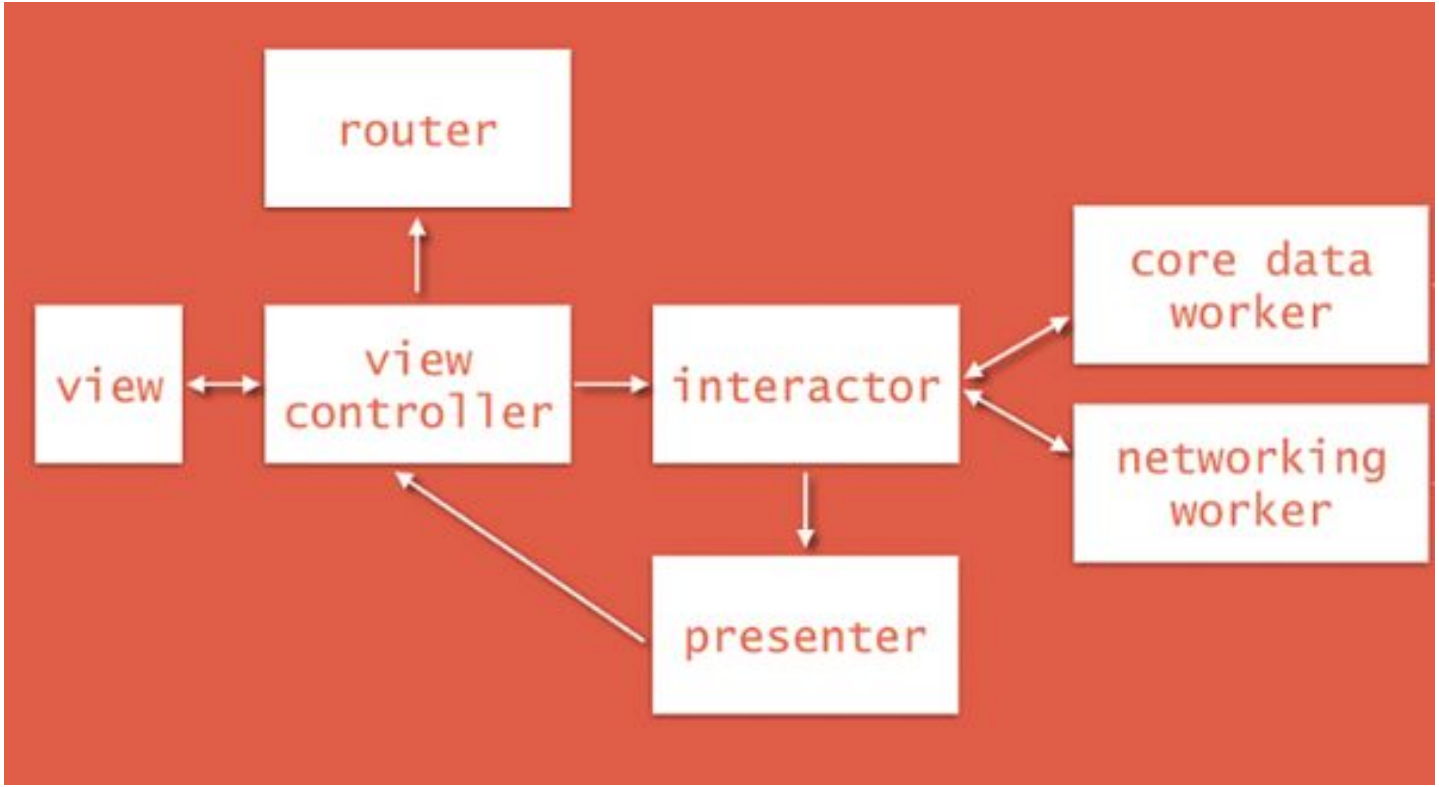
COORDINATOR

- Soroush Khanlou 2015
- Router / Flow
- Delegates navigation
- Delegates navigation
- Encapsulates an entire flow
- Allows easy flows reusability

VIPER

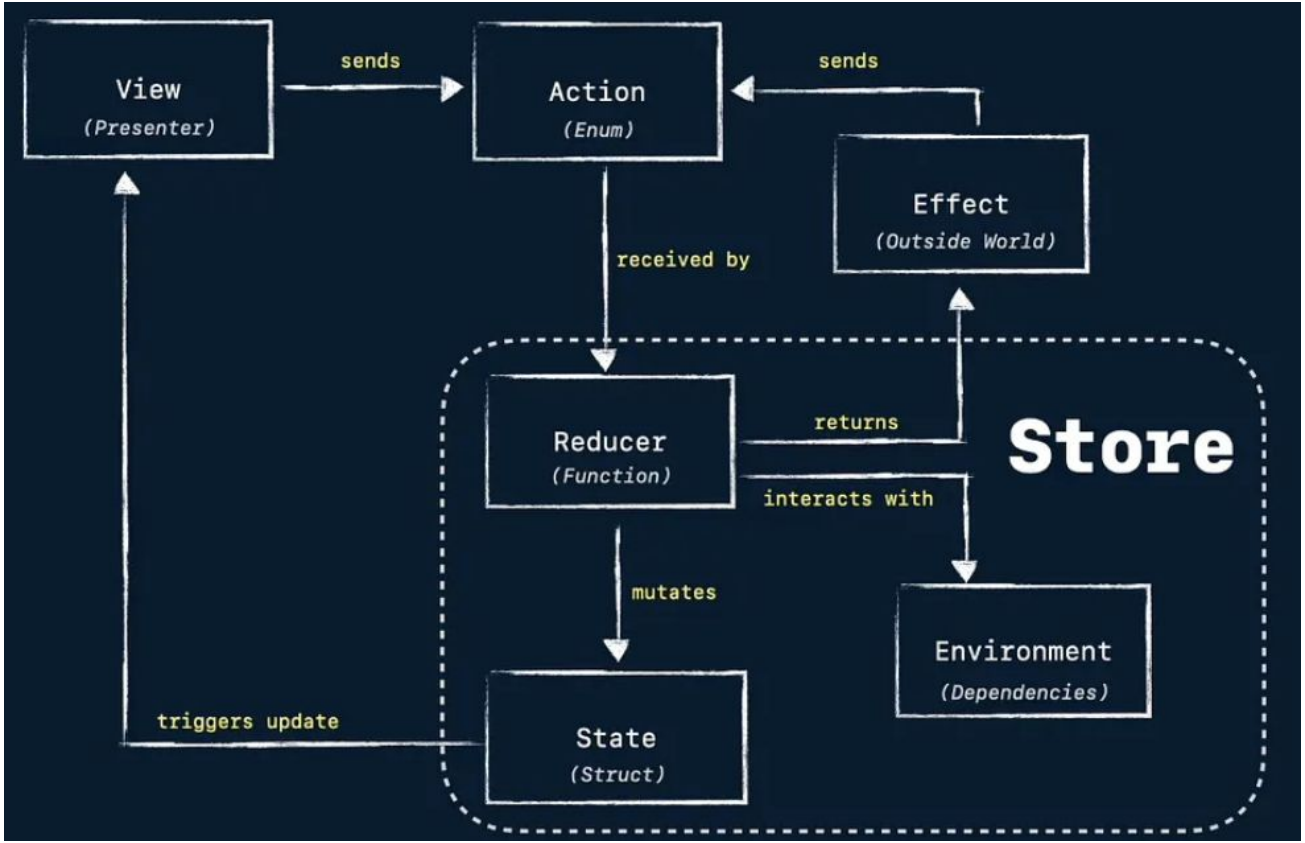

router
view
view controller
presenter
core data service
interactor
networking service

VIP


router
view
view controller
interactor
core data worker
networking worker
presenter

WHAT IS REDUX? 🤷‍♂️

THE COMPOSABLE ARCHITECTURE (TCA)



Nothing is fixed. Take the good ideas and create your own patterns that suits your needs

Someone clever

DEPENDENCY INJECTION

02

DEPENDENCY INJECTION

Implementation of Dependency Inversion principle (SOLI**D**):

The strategy of depending upon interfaces or abstract functions and classes rather than upon concrete functions and classes.

Passing a concrete dependency (e.g. view model or service) from outside of the object

- Removed knowledge about implementation details where it is not needed
- Helps with interchangeability
- Allows better testability

```
final class LocationService {
    func updateLocation() -> CLLocation {
        CLLocation(
            latitude: Double.random(in: -90...90),
            longitude: Double.random(in: -90...90)
        )
    }
}

final class ViewModel: ObservableObject {
    @Published var location = CLLocation()
    private let locationService = LocationService()

    init() {}

    func didLoad() {
        location = locationService.updateLocation()
    }
}
```

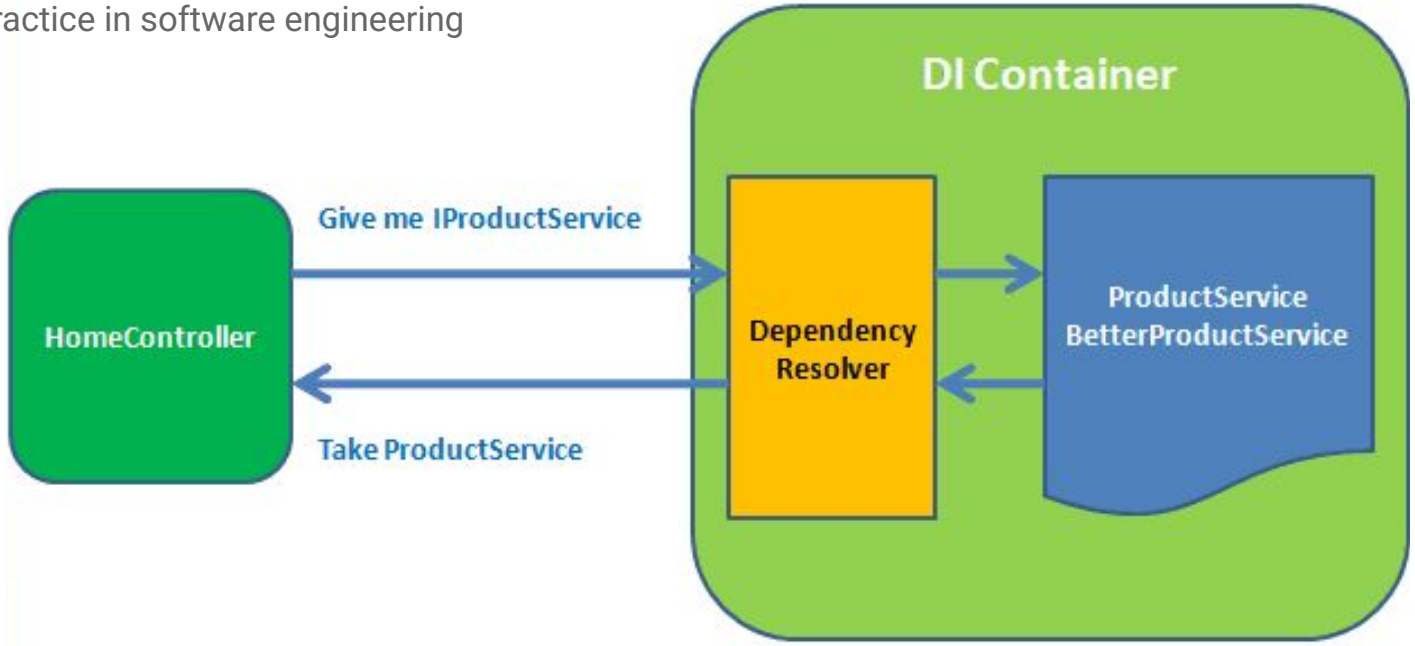
Almost there…

```
final class LocationService {
    func updateLocation() -> CLLocation {
        CLLocation(
            latitude: Double.random(in: -90...90),
            longitude: Double.random(in: -90...90)
        )
    }
}

final class ViewModel: ObservableObject {
    @Published var location = CLLocation()
    private let locationService: LocationService

    init(locationService: LocationService) {
        self.locationService = locationService
    }

    func didLoad() {
        location = locationService.updateLocation()
    }
}
```

```swift
protocol LocationServicing {
    func updateLocation() -> CLLocation
}

final class LocationService: LocationServicing {
    func updateLocation() -> CLLocation {
        CLLocation(
            latitude: Double.random(in: -90...90),
            longitude: Double.random(in: -90...90)
        )
    }
}

final class ViewModel: ObservableObject {
    @Published var location = CLLocation()
    private let locationService: LocationServicing

    init(locationService: LocationServicing) {
        self.locationService = locationService
    }

    func didLoad() {
        location = locationService.updateLocation()
    }
}
```

DEPENDENCY INJECTION

- Dependency container
- Standard practice in software engineering



```
static func registerUseCases(to container: DIContainer) {
    container.register(ConnectUserUseCasing.self) {
        ConnectUserUseCase(
            credentialsManager: container.resolve()
        )
    }

    container.register(DisconnectUserUseCasing.self) {
        DisconnectUserUseCase(
            credentialsManager: container.resolve()
        )
    }

    container.register(UpdateConnectedUserUseCasing.self)
        UpdateConnectedUserUseCase(
            credentialsManager: container.resolve()
        )
    }
```

```
container.register(ProfileViewModel.self) { resolver in
    ProfileViewModel(
        profileUseCase: resolver.resolve(ProfileUseCaseProtocol.self)!,
        analyticsManager: resolver.resolve(AnalyticsManaging.self)!,
        localAuthUseCase: resolver.resolve(LocalAuthenticationUseCasing.self)!,
        pushNotificationUseCase: resolver.resolve(PushNotificationUseCasing.self)!
    )
}.inObjectScope(.transient)
```

STRV

DESIGN SYSTEMS

WHAT IS IT?

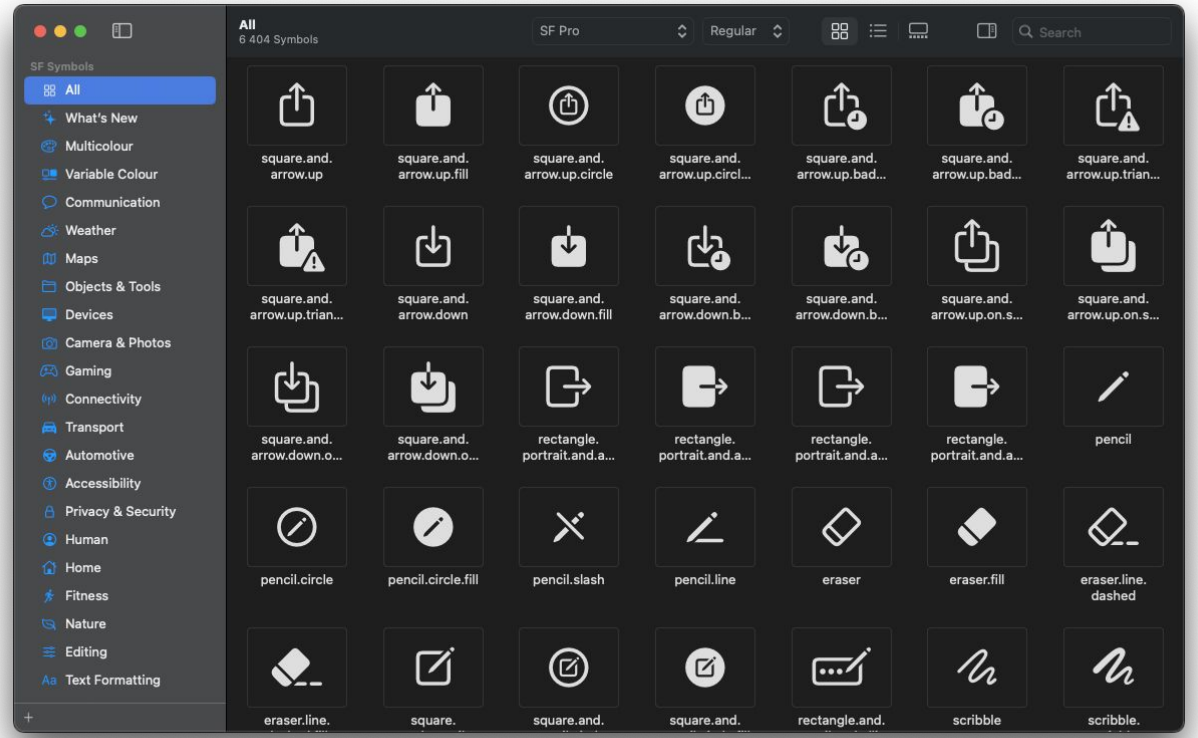
- Components
  - Button, Switch, Alert
  - Label, TextField, Search
- Styles
  - Colors, Gradients
  - Font, Spacing (Padding)

- Composition
  - Lists, Collections, Stacks
    Pages, Section
- Guidelines
  - Rules for Accessibility
  - Rules for Layout

Native DS

- Native Apple Design System
  - SF Symbols
  - UI Components
- Native Android Design System
  - Material Design

SF SYMBOLS

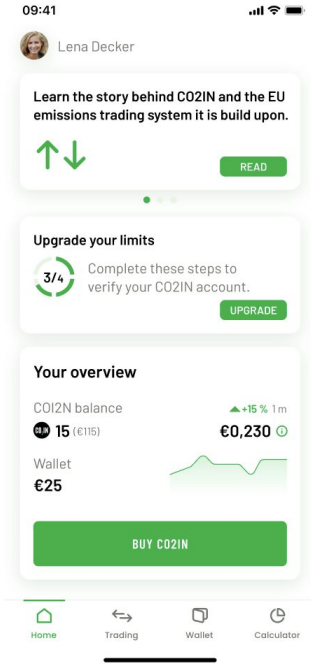

All
6 404 Symbols
SF Pro
Regular
Search
SF Symbols
All
What's New
Multicolour
Variable Colour
Communication
Weather
Maps
Objects & Tools
Devices
Camera & Photos
Gaming
Connectivity
Transport
Automotive
Accessibility
Privacy & Security
Human
Home
Fitness
Nature
Editing
Text Formatting
square.and.arrow.up
square.and.arrow.up.fill
square.and.arrow.up.circle
square.and.arrow.up.circl...
square.and.arrow.up.bad...
square.and.arrow.up.bad...
square.and.arrow.up.trian...
square.and.arrow.up.trian...
square.and.arrow.down
square.and.arrow.down.fill
square.and.arrow.down.b...
square.and.arrow.down.b...
square.and.arrow.up.on.s...
square.and.arrow.up.on.s...
square.and.arrow.down.o...
square.and.arrow.down.o...
rectangle.portrait.and.a...
rectangle.portrait.and.a...
rectangle.portrait.and.a...
rectangle.portrait.and.a...
pencil
pencil.circle
pencil.circle.fill
pencil.slash
pencil.line
eraser
eraser.fill
eraser.line.dashed
eraser.line.
square.
square.and.
square.and.
rectangle.and.
scribble
scribble.

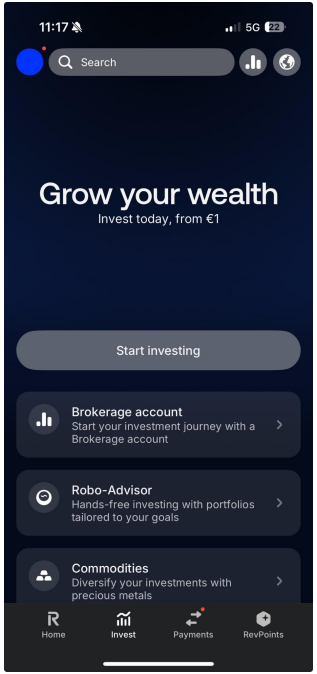
WHY WE NEED IT

- **The Need for Consistency**
- Professional look & feel
- Speed up development
- Single source of truth
- Brings order into a chaotic world of UI

WHY CUSTOM DS?

- Unique brand identity
- Differentiate from others
- Apple's components are not enough
- Consistency across platforms





DEMO

COLORS

```
enum Colors {
    static let primary = UIComponentsModule.color(named: "Base/primary")
    static let background = UIComponentsModule.color(named: "Base/background-base")

    enum Contrast {
        public static let gray100 = UIComponentsModule.color(named: "Contrast/gray-100")
        public static let gray500 = UIComponentsModule.color(named: "Contrast/gray-500")
        public static let gray800 = UIComponentsModule.color(named: "Contrast/gray-800")
    }

    enum Functional {
        public static let error = UIComponentsModule.color(named: "Functional/error")
        public static let warning = UIComponentsModule.color(named: "Functional/warning")
        public static let success = UIComponentsModule.color(named: "Functional/success")
        public static let info = UIComponentsModule.color(named: "Functional/info")
    }
}
```

COLORS

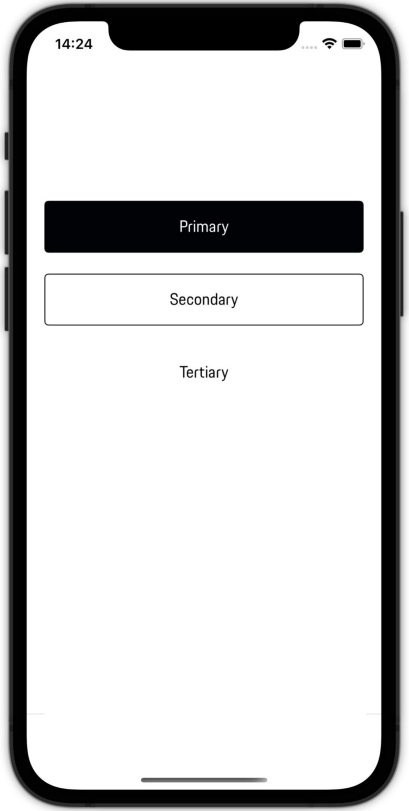
```
var subtitleView: some View {
    Text(model.subtitle)
        .foregroundStyle(Colors.Contrast.gray800.color)
        .font(Fonts.subheadline.font)
        .padding(Spacings.small)
}
```

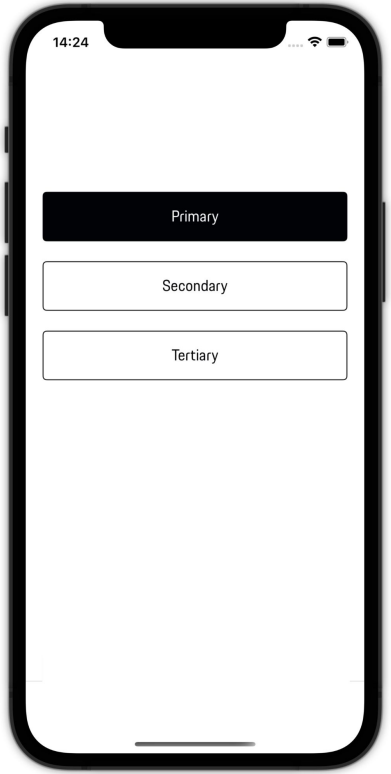
DEMO 2

STRV

ABSTRACTIONS

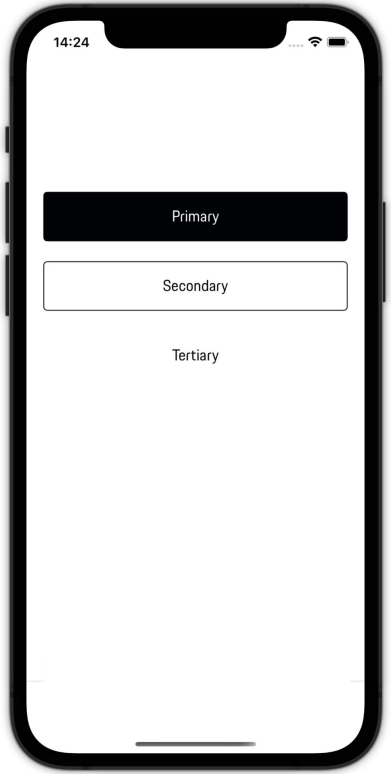
GROUPS


14:24
Primary
Secondary
Tertiary


14:24
Primary
Secondary
Tertiary

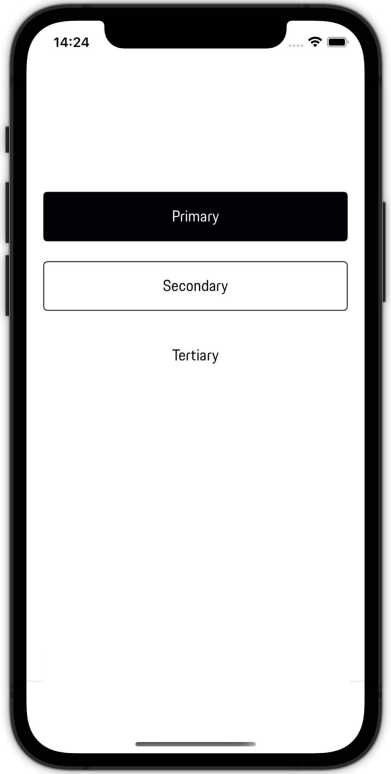
GROUPS

```
var body: some View {
    PrimaryButton()
    SecondaryButton()
    TertiaryButton()
}
```



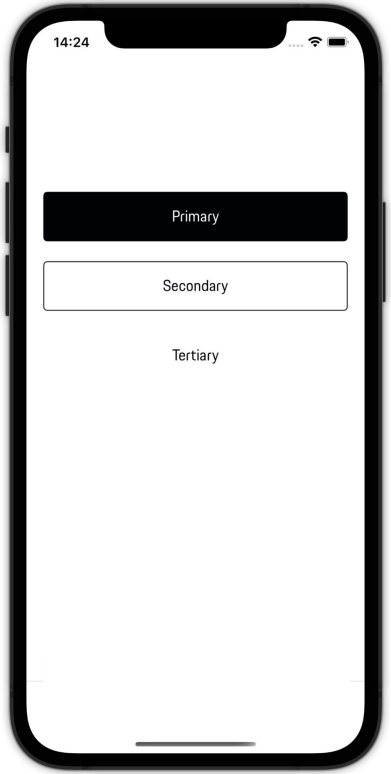
GROUPS

```
var body: some View {
    MyButton(type: .primary)
    MyButton(type: .secondary)
    MyButton(type: .tertiary)
}
```



ABSTRACTION

- Less change in screens
- More changes in components
- Easy to do changes



APPEARANCE / CONFIG

- Conforming to protocols
  - Default values
- Separation of values from implementation
- Strict for changes

APPEARANCE

```
protocol ButtonAppearance {
    func height(for style: ButtonStyle) -> CGFloat
    func width(for style: ButtonStyle) -> CGFloat
    func backgroundColor(for style: ButtonStyle) -> Color
    func font(for style: ButtonStyle) -> Font
}
```

APPEARANCE

```
protocol WorldButtonAppearance: ButtonAppearance {}

extension WorldButtonAppearance {
    func height(for style: ButtonStyle) -> CGFloat {
        40
    }
    func width(for style: ButtonStyle) -> CGFloat {
        .infinity // (for full width)
    }
    func backgroundColor(for style: ButtonStyle) -> Color {
        Colors.Contrast.gray100
    }
    func font(for style: ButtonStyle) -> Font {
        Fonts.body.font
    }
}
```

```
struct MyButton: View {
    let appearance: ButtonAppearance
    let buttonStyle: ButtonStyle
    init(appearance: ButtonAppearance, buttonStyle: ButtonStyle) {
        self.appearance = appearance
        self.buttonStyle = buttonStyle
    }

    var body: some View {
        Button(action: {}) {
            Text("Button")
                .background(
                    appearance.backgroundColor(for: buttonStyle)
                )
                .font(
                    appearance.font(for: buttonStyle)
                )
                .frame(
                    width: appearance.width(for: buttonStyle),
                    height: appearance.height(for: buttonStyle)
                )
        }
    }
}
```

Versioning DS

- Refactor existing one
  - Risky solution
  - Release all at once
- Start over
  - New Naming
  - Deprecating old components (hundreds of warnings)
- Appearance solution
  - Create new appearances + new components
  - Feature flagged -> for development, use new components, for production, still old one
  - Deprecate old components with pink color (no warnings)

Struct vs Modifier?

```
Button {
    print()
} label: {
    Text("Tap here")
}
.buttonStyle(PrimaryButtonStyle())
```

```
PrimaryButton {
    print()
} label: {
    Text("Tap here")
}
```

QUESTIONS

STRV

THANK YOU!

Martin Vidovic, iOS Engineer at STRV

STRV