Swift Code Patterns From the Ranch

https://github.com/bignerdranch/RanchWeather

* Built Using Swift 3, Xcode 8b6

June 2, 2014

Swift

Strongly Typed

Value Types

Composition

Protocol-Oriented Programming

Objective-C / UIKit

Weakly Typed

Reference Types

Inheritance

Object-Oriented Programming

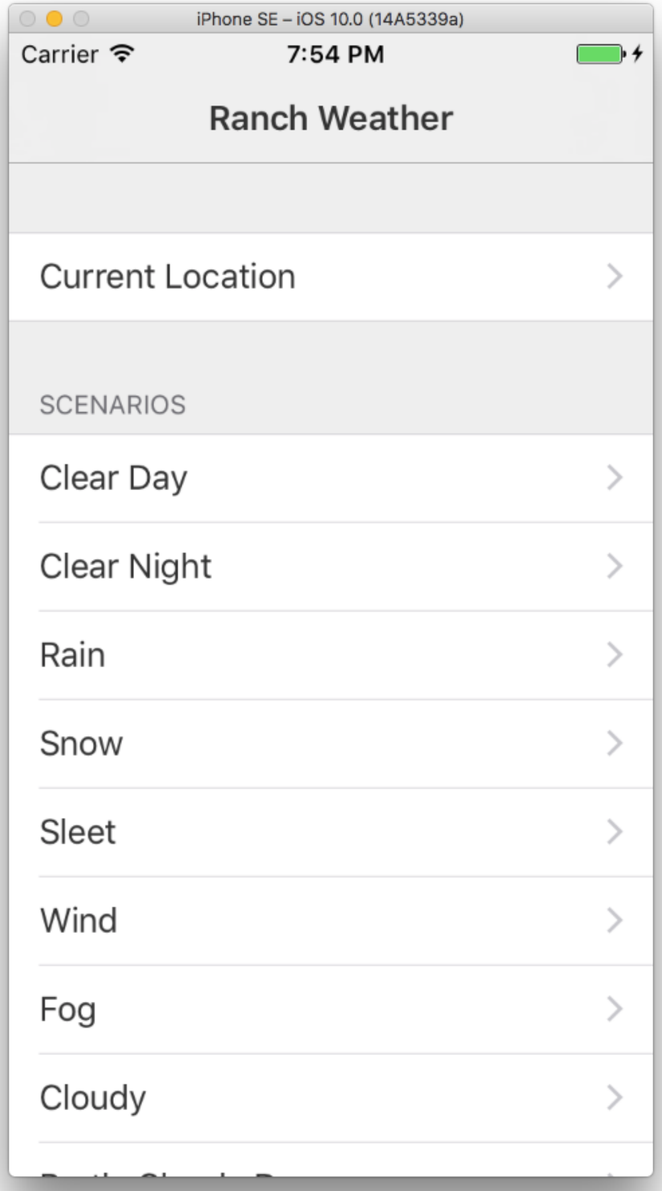
How do we use UIKit
with Swift and honoring its goals?

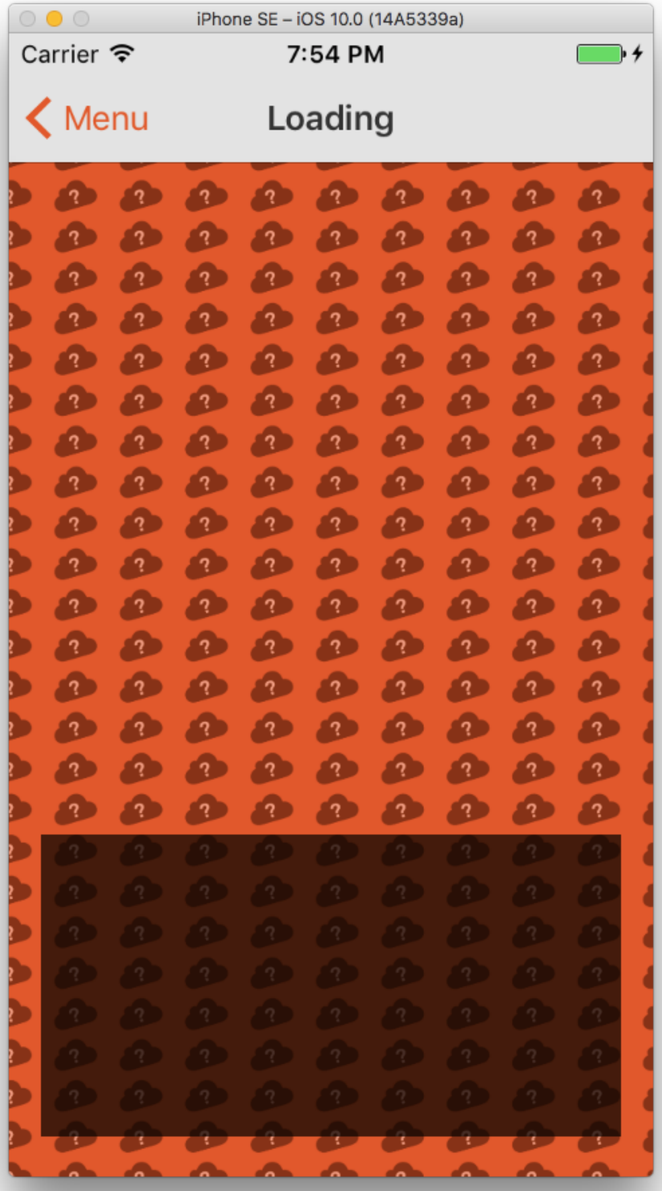
Safe. Fast. Expressive.

https://swift.org/about/

Safe. Expressive.

Safe. Expressive. Testable.


iPhone SE – iOS 10.0 (14A5339a)
Carrier
7:54 PM
Ranch Weather
Current Location
SCENARIOS
Clear Day
Clear Night
Rain
Snow
Sleet
Wind
Fog
Cloudy


iPhone SE – iOS 10.0 (14A5339a)
Carrier
7:54 PM
Menu
Loading


iPhone SE – iOS 10.0 (14A5339a)
Carrier
7:54 PM
Menu
Current Weather
123.4°F
clear-day
Sep 6, 2016, 7:54:34 PM

Initialization

```
class Car {

    let driver: Driver

    init() {

    }
}
```

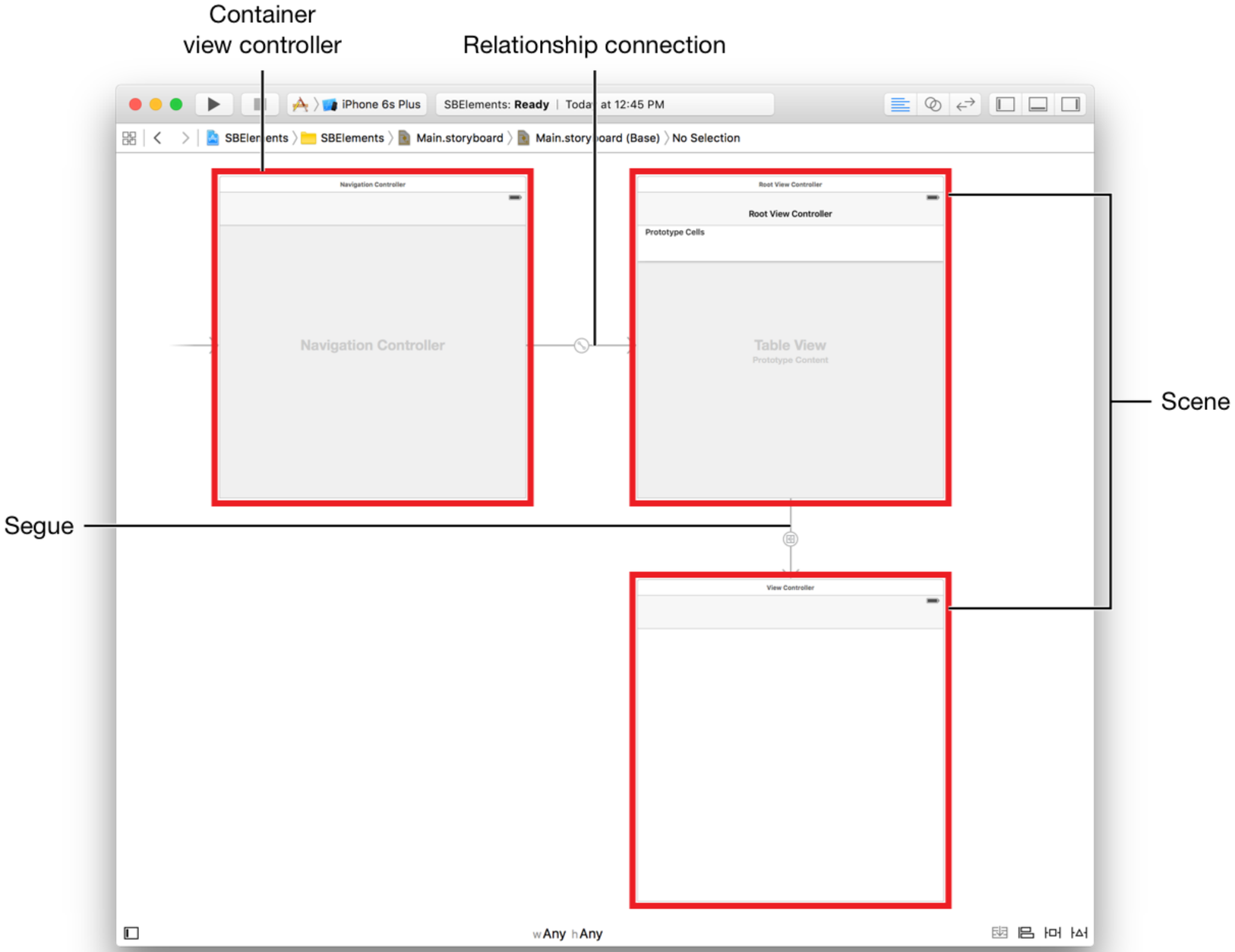
```
class Car {

    let driver: Driver

    init() {

    }
    // Return from initializer without
    // initializing all stored properties.
}
```

```
class Car {

    let driver: Driver

    init(driver: Driver) {
       self.driver = driver
    }

}
```

View Controllers

```
class DisplayViewController: UIViewController {

    let weatherService: WeatherService

}
```

```
init(nibName nibNameOrNil: String?,
       bundle nibBundleOrNil: Bundle?)

init?(coder aDecoder: NSCoder)
```


Container
view controller
Relationship connection
iPhone 6s Plus
SBElements: Ready
Main.storyboard
No Selection
Navigation Controller
Root View Controller
Prototype Cells
Table View
Prototype Content
View Controller
Segue
Scene
wAny hAny

```
class DisplayViewController: UIViewController {

    let weatherService: WeatherService

}
```

```
class DisplayViewController: UIViewController {

    var weatherService: WeatherService?

}
```

```
class DisplayViewController: UIViewController {

    var weatherService: WeatherService!

}
```

Dependency Injection

```swift
// Injectable is a simple protocol to helps enforce Dependency Injection.
// It is typically used on View Controllers where you don't have early
// life-cycle access and need to inject or configure required properties
// after the object has been initialized.
protocol Injectable {

    // When honoring this protocol we expect you to make a method called
    // inject() that has the needed properties for this object.

    // checkDependencies is a method intended to verify that our
    // implicitly unwrapped optionals preconditions have been populated.
    // It should called in an early life-cycle method where the
    // required dependancies should have already been set.
    // For View Controllers, viewDidLoad() is a common choice.
    func checkDependencies()
}
```

```
class WeatherDisplayViewController: UIViewController {
    var weatherService: WeatherService!
    var location: Location!

    override func viewDidLoad() {
        super.viewDidLoad()
        checkDependencies()
        updateUI()
    }
}

//MARK: - Injectable
extension WeatherDisplayViewController: Injectable {
    func inject(weatherService: WeatherService, location: Location) {
        self.weatherService = weatherService
        self.location = location
    }

    func checkDependencies() {
        precondition(weatherService != nil)
        precondition(location != nil)
    }
}
```

Dependency Injection

DRY Strings

```
extension UserDefaults {

    enum Key {
        static let themeIdentifier = "com.ranchweather.userdefaults.theme"
    }

    enum Notifications {
        static let themeDidChange =
           Notification.Name("com.ranchweather.notification.themeDidChange")
    }

}

string(forKey: Key.themeIdentifier)


NotificationCenter.default.post(name: Notifications.themeDidChange, object: self)
```

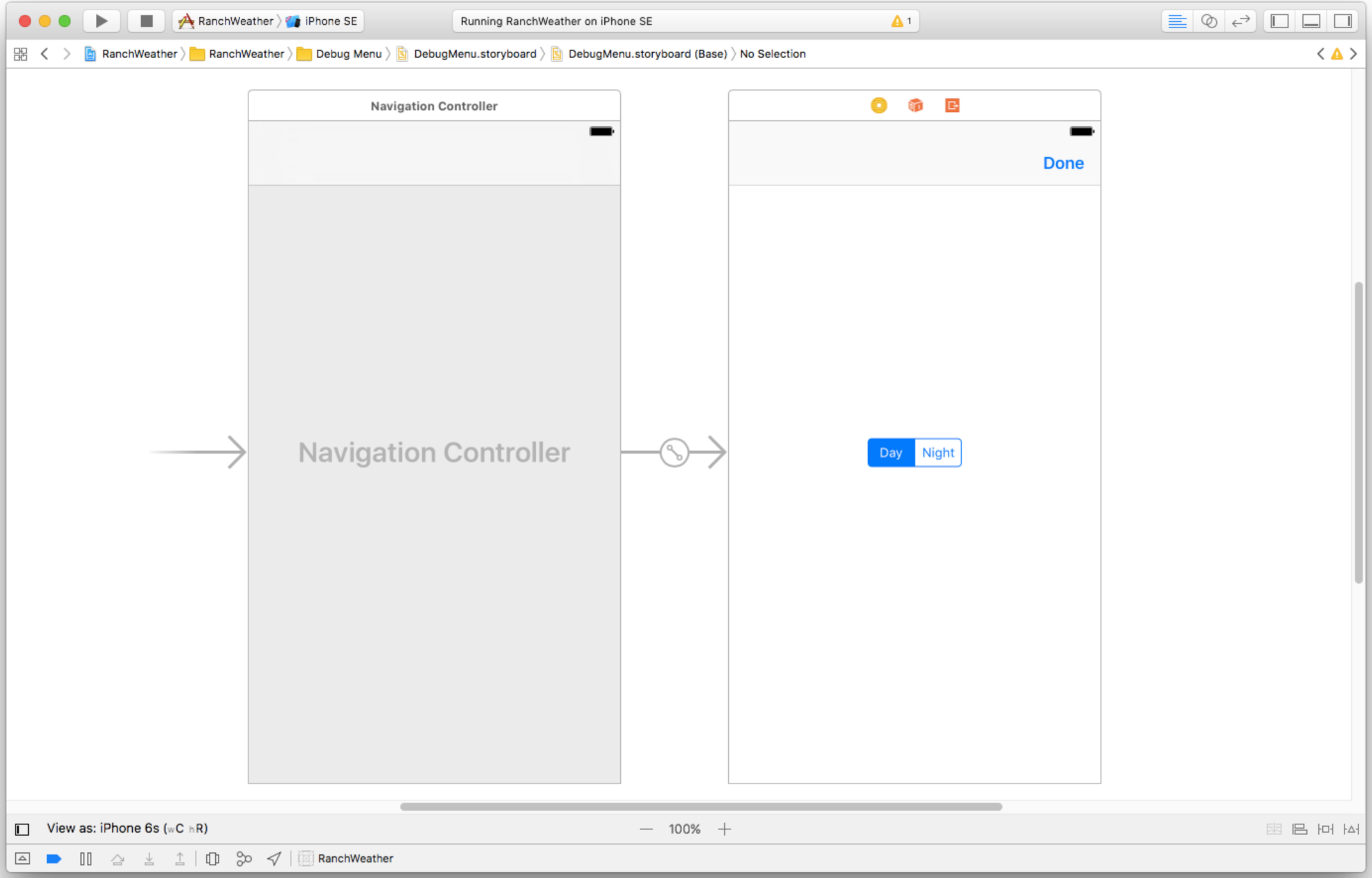
```swift
extension UIImage {
    enum Asset: String {
        case clearDay          = "clear-day"
        case clearNight        = "clear-night"
        case rain              = "rain"
        case snow              = "snow"
        case sleet             = "sleet"
        case wind              = "wind"
        case fog               = "fog"
        case cloudy            = "cloudy"
        case partlyCloudyDay   = "partly-cloudy-day"
        case partlyCloudyNight = "partly-cloudy-night"
    }

    convenience init!(asset: Asset) {
        self.init(named: asset.rawValue)
    }

}

UIImage(asset: .snow)
```

```swift
extension UIStoryboard {
    private enum File: String {
        case main = "Main"
        case weatherDisplay = "WeatherDisplay"
        case feedback = "Feedback"
        case debugMenu = "DebugMenu"
    }

    private convenience init(_ file: File) {
        self.init(name: file.rawValue, bundle: nil)
    }
}

let storyboard = UIStoryboard(.weatherDisplay)
```

View Controllers from Storyboards

```
extension UIStoryboard {
    private enum Identifier: String {
        ...
    }

    private convenience init(_ identifier: Identifier) {
        self.init(name: identifier.rawValue, bundle: nil)
    }
}

let storyboard = UIStoryboard(.WeatherDisplay)
```

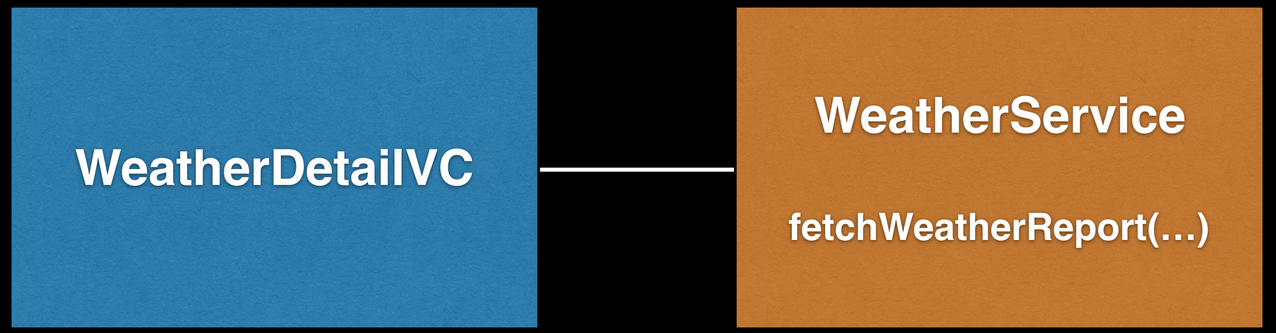
```swift
extension UIStoryboard {

    static func weatherDisplayViewController() -> WeatherDisplayViewController {
        let storyboard = UIStoryboard(.weatherDisplay)
        return storyboard.instantiateInitialViewController() as! WeatherDisplayViewController
    }

}

let vc = UIStoryboard.weatherDisplayViewController()
```


RanchWeather
iPhone SE
Running RanchWeather on iPhone SE
1
RanchWeather
RanchWeather
Debug Menu
DebugMenu.storyboard
DebugMenu.storyboard (Base)
No Selection
Navigation Controller
Navigation Controller
Done
Day
Night
View as: iPhone 6s (wC hR)
100%
RanchWeather

```swift
extension UIStoryboard {

    static func debugViewControllerStack(configure: (DebugMenuViewController) -> Void) ->
        UINavigationController
{
        let navigationController = UIStoryboard(.DebugMenu).
            instantiateInitialViewController() as! UINavigationController
        let debugMenu = navigationController.viewControllers[0]
                        as! DebugMenuViewController
        configure(debugMenu)
        return navigationController
    }

}

let debugNavigationController = UIStoryboard.debugViewControllerStack { (debugVC) in
    debugVC.inject(userDefaults: UserDefaults.standard, delegate: self)
}
present(debugNavigationController, animated: true, completion: nil)
```

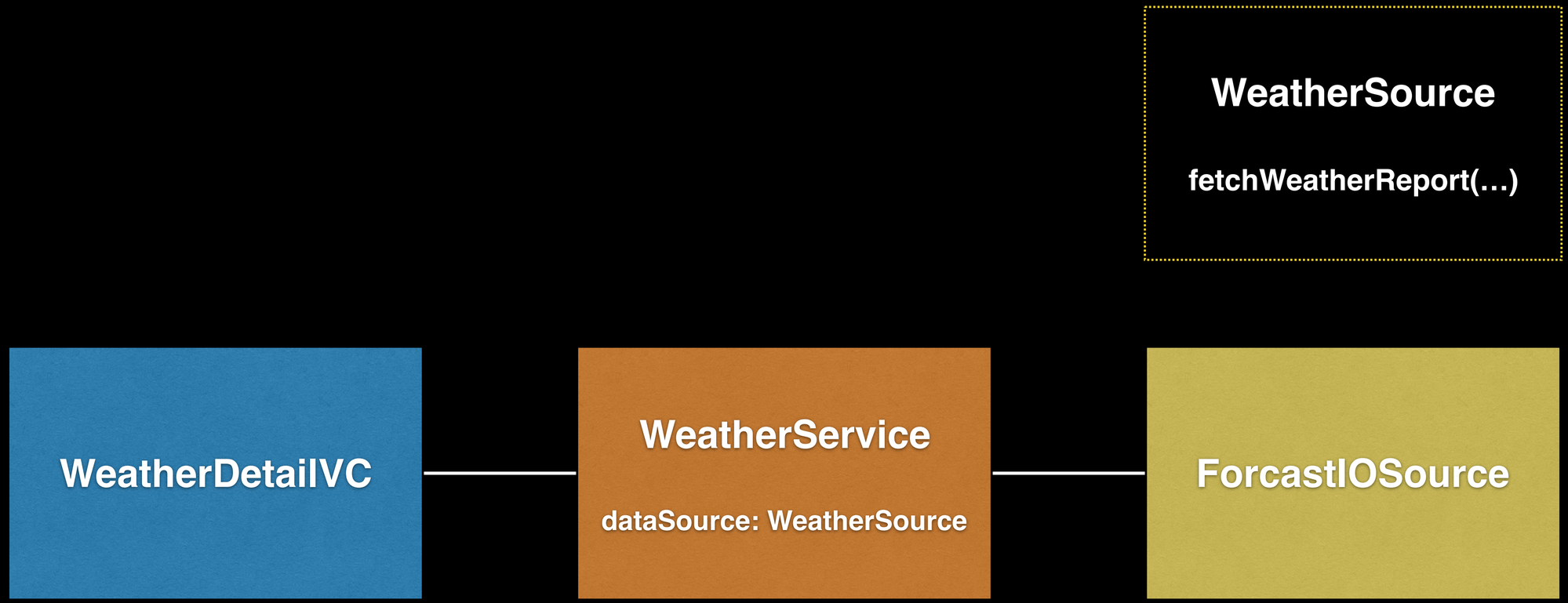
Data Adaptors
(Many DataSources)

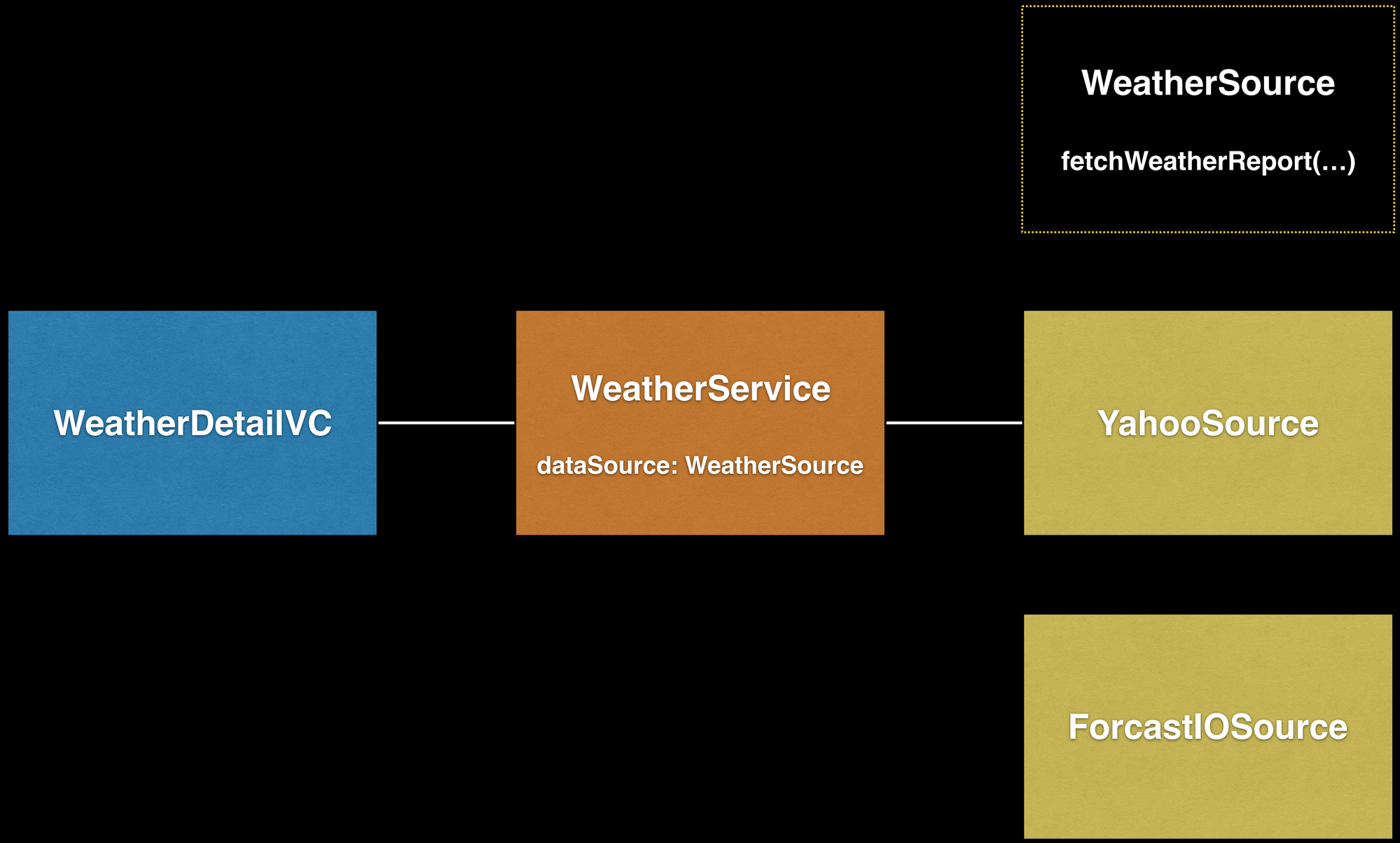

WeatherDetailVC

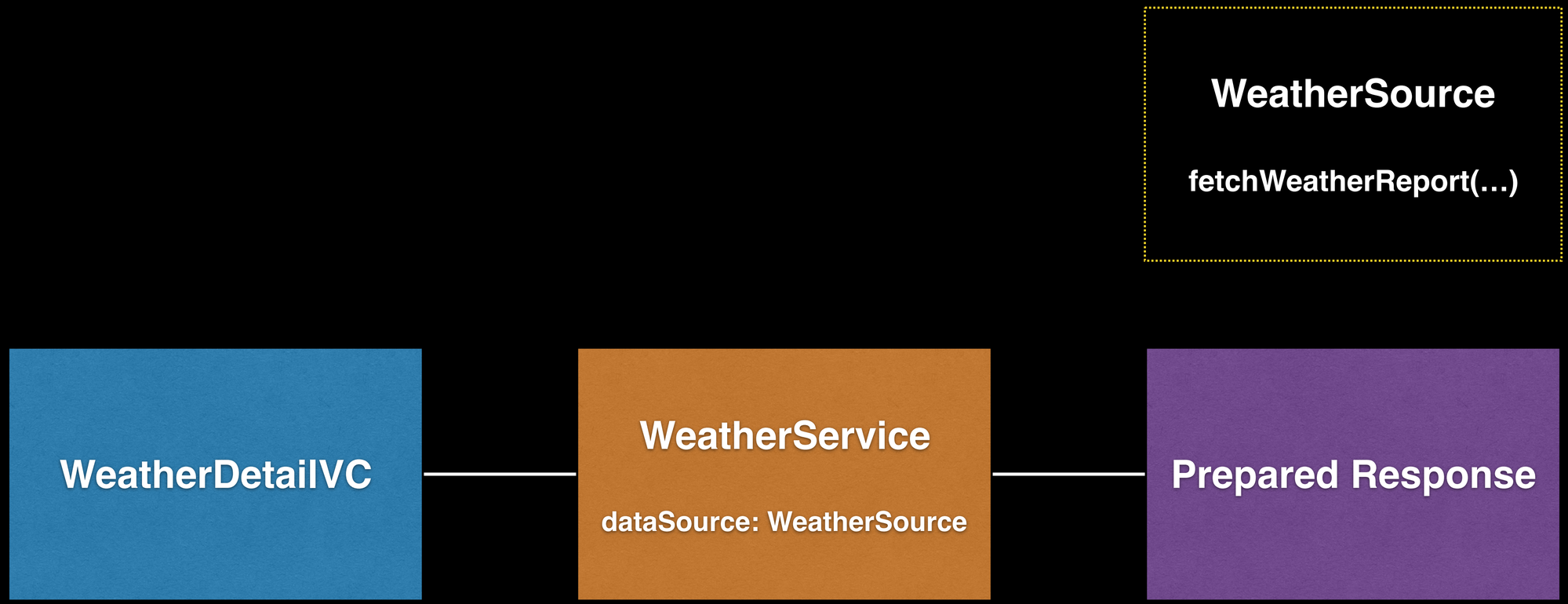

WeatherDetailVC
WeatherService
fetchWeatherReport(…)

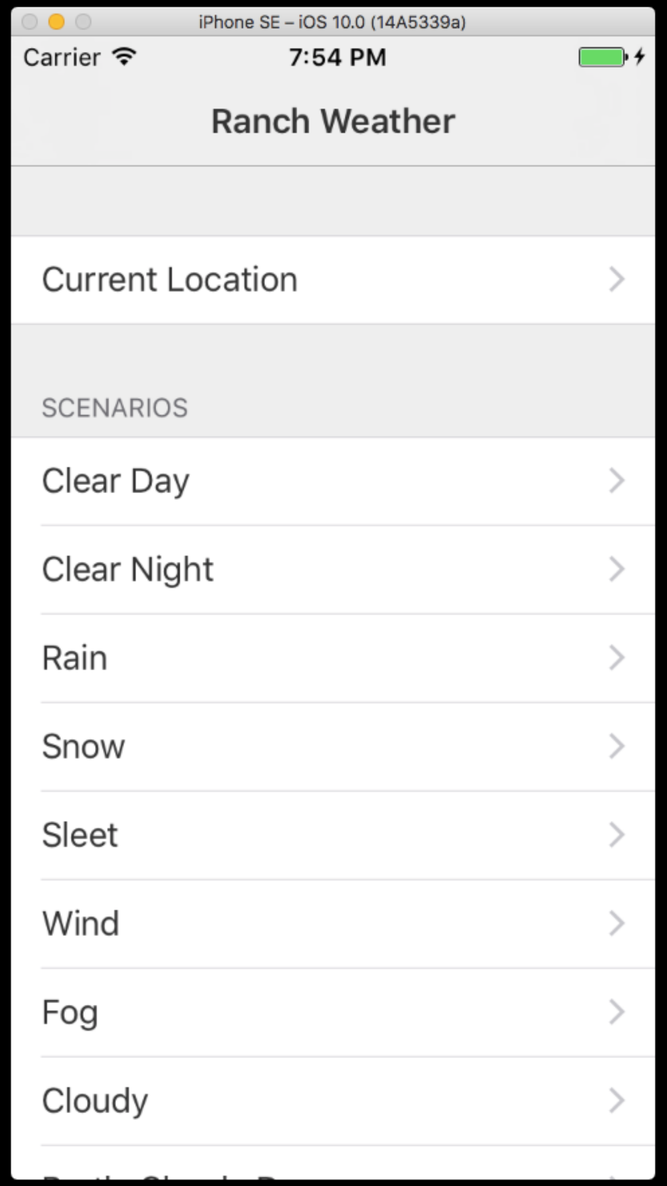

WeatherDetailVC
WeatherService
fetchWeatherReport(…)
WeatherSource
fetchWeatherReport(…)

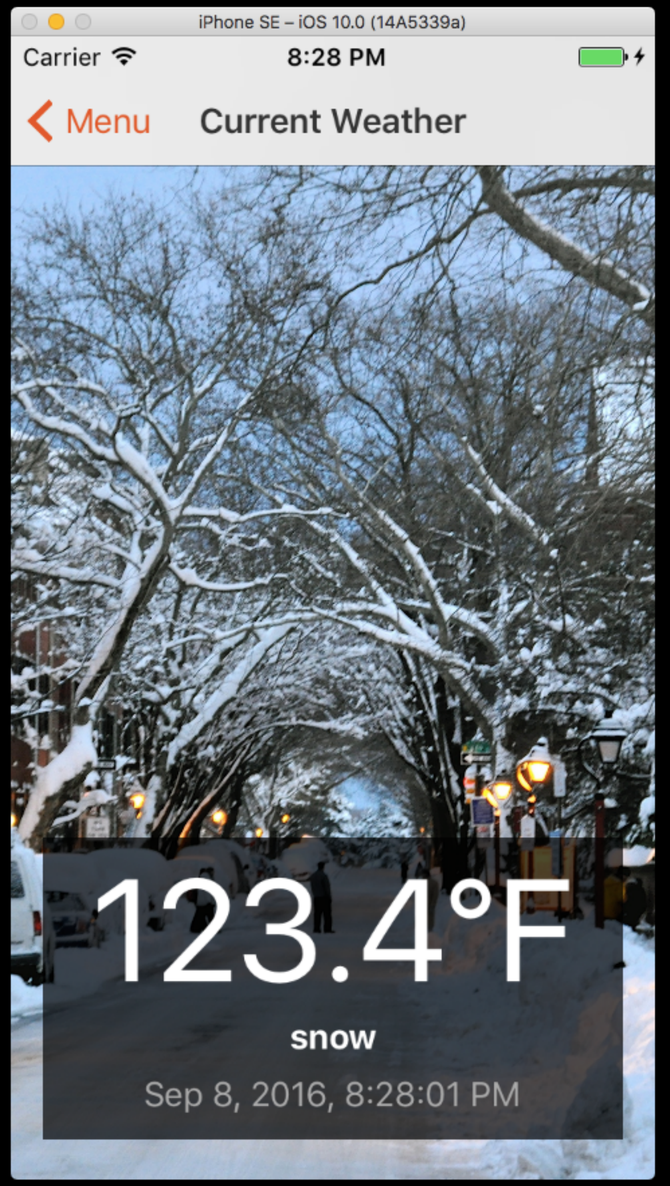

WeatherDetailVC
WeatherService
dataSource: WeatherSource
WeatherSource
fetchWeatherReport(…)

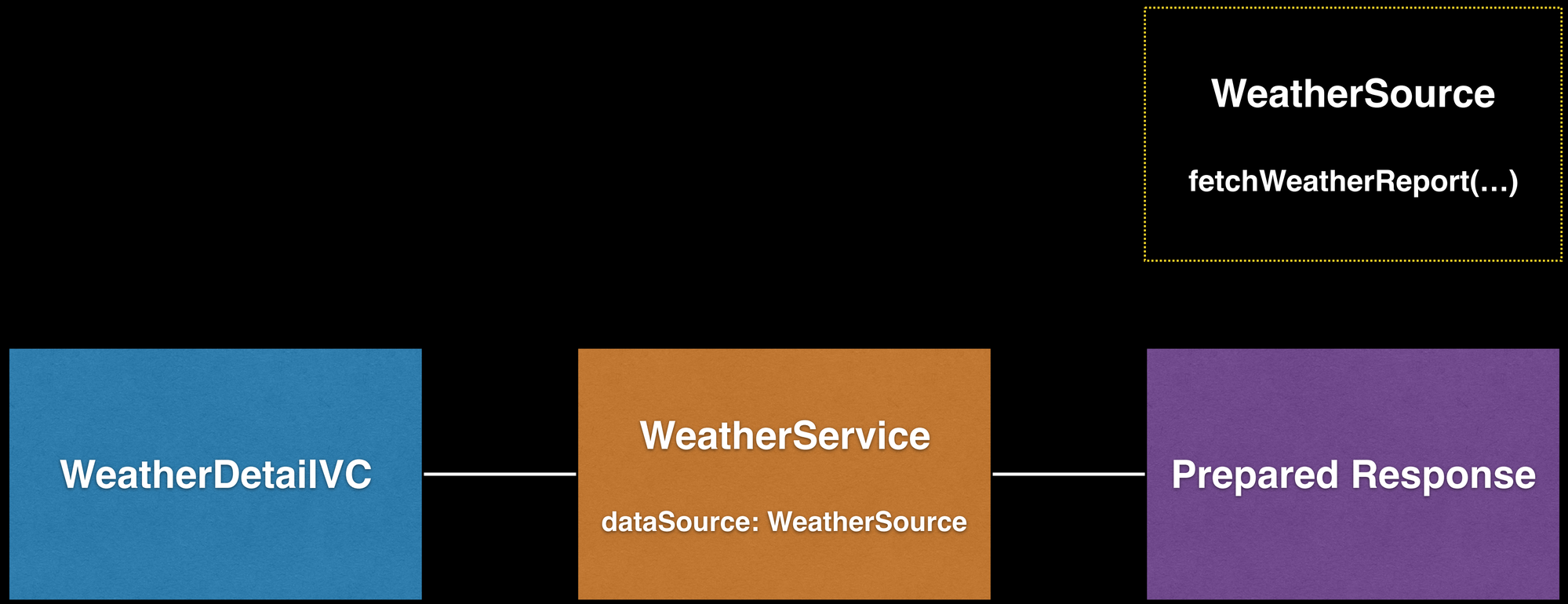

WeatherSource
fetchWeatherReport(…)
WeatherDetailVC
WeatherService
dataSource: WeatherSource
ForcastIOSource


WeatherSource
fetchWeatherReport(…)
WeatherDetailVC
WeatherService
dataSource: WeatherSource
YahooSource
ForcastIOSource


WeatherSource
fetchWeatherReport(…)
WeatherDetailVC
WeatherService
dataSource: WeatherSource
Prepared Response


iPhone SE – iOS 10.0 (14A5339a)
Carrier
7:54 PM
Ranch Weather
Current Location
SCENARIOS
Clear Day
Clear Night
Rain
Snow
Sleet
Wind
Fog
Cloudy


iPhone SE – iOS 10.0 (14A5339a)
Carrier
8:28 PM
Menu
Current Weather
123.4°F
snow
Sep 8, 2016, 8:28:01 PM


WeatherSource
fetchWeatherReport(…)
WeatherDetailVC
WeatherService
dataSource: WeatherSource
Prepared Response

Service Interface

```
struct WeatherService {

    func fetchWeatherReport(latitude: Double, longitude: Double,
         completion: @escaping (_ result: WeatherService.Result) -> Void)
    {
        dataSource.fetchWeatherReport(latitude: latitude, longitude: longitude,
           completion: completion)
    }

}
```

```
struct WeatherService {

    enum Result {
        case success(WeatherReport)
        case failure(WeatherService.Error)
    }

    func fetchWeatherReport(latitude: Double, longitude: Double,
         completion: @escaping (_ result: WeatherService.Result) -> Void)
    {
        dataSource.fetchWeatherReport(latitude: latitude, longitude: longitude,
            completion: completion)
    }

}
```

```
struct WeatherService {

    enum Result {
        case success(WeatherReport)
        case failure(WeatherService.Error) // Many would use ErrorType
    }

}
```

```
struct WeatherService {

    enum Result {
        case success(WeatherReport)
        case failure(WeatherService.Error) // Many would use ErrorType
    }

    enum Error {
        case invalidLatitude
        case invalidLongitude
        case parseError
        case noPreparedResponse
        case networkError(NSError)
        case noDataError
    }

}
```

```
enum ValidatorError: Equatable {
    case MinLengthInvalid(actual: Int, required: Int)
    case MaxLengthInvalid(actual: Int, allowed: Int)
    case RequiredCharacterMissing(requiredSet: NSCharacterSet)
    case RequiredPrefixCharacterMissing(requiredPrefixSet: NSCharacterSet)
    case DoesNotMatchRegex(regex: String)
}

// Taken From Swift2 project, forgive the CamelCase enum names.
```

```
struct WeatherService {

    enum Result {
        case success(WeatherReport)
        case failure(WeatherService.Error)
    }

    enum Error {}

    func fetchWeatherReport(latitude: Double, longitude: Double,
         completion: @escaping (_ result: WeatherService.Result) -> Void)
    {
        dataSource.fetchWeatherReport(latitude: latitude, longitude: longitude,
            completion: completion)
    }

}
```

Theming

UIColor

```swift
private extension UIColor {
    convenience init(hex: UInt32) {
        // Note: This is called the bit shift operator
        let red   = CGFloat((hex >> 16) & 0xff) / 255
        let green = CGFloat((hex >>  8) & 0xff) / 255
        let blue  = CGFloat((hex      ) & 0xff) / 255
        self.init(red: red, green: green, blue: blue, alpha: 1)
    }
}

let yellow = UIColor(hex: 0xECB02F)
// Note: These are called hexadecimal literals
```

```
private enum BNRColors {
    static let offWhite  = UIColor(hex: 0xEEEEEE)
    static let offBlack  = UIColor(hex: 0x333333)
    static let red       = UIColor(hex: 0xE15827)
    static let yellow    = UIColor(hex: 0xECB02F)
}
```

```
enum Theme: String {
    case day   = "com.ranchweather.theme.day"
    case night = "com.ranchweather.theme.night"

    var backgroundColor: UIColor {
        switch self {
        case .day:   return BNRColors.offWhite
        case .night: return BNRColors.offBlack
        }
    }
    var textColor: UIColor {
        switch self {
        case .day:   return BNRColors.offBlack
        case .night: return BNRColors.offWhite
        }
    }
    var tintColor: UIColor {
        switch self {
        case .day:   return BNRColors.red
        case .night: return BNRColors.yellow
        }
    }
    var preferredStatusBarStyle: UIStatusBarStyle {
        switch self {
        case .day:   return UIStatusBarStyle.default
        case .night: return UIStatusBarStyle.lightContent
        }
    }

}
```

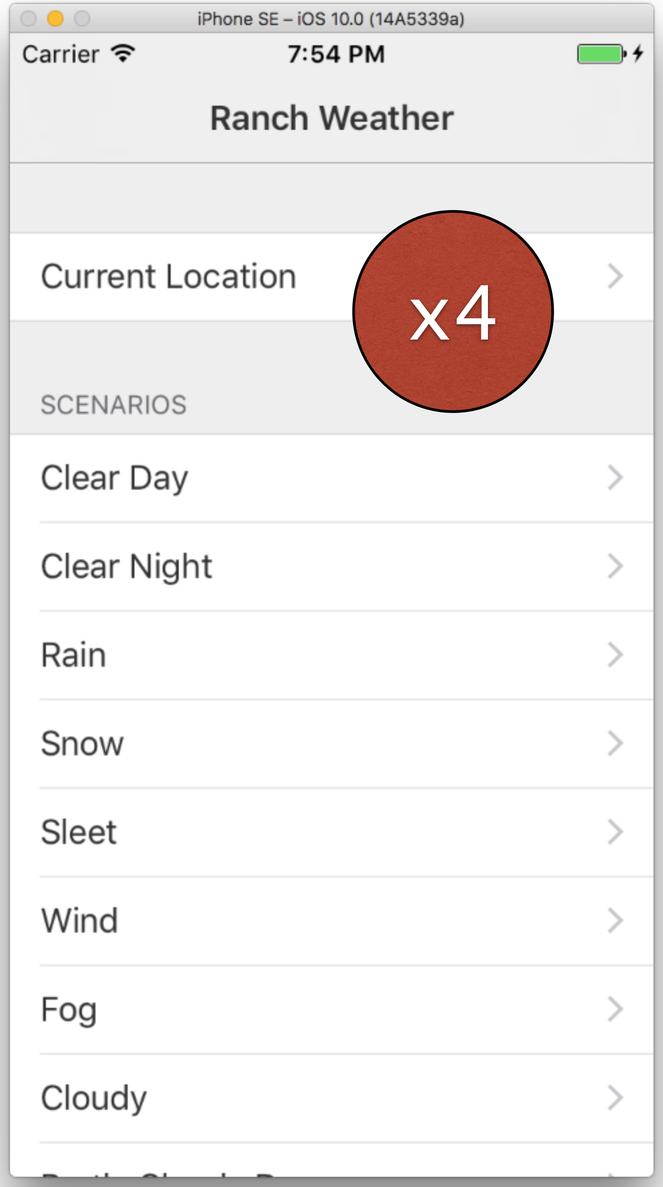
```
struct Themer {
    let theme: Theme

    init(theme: Theme) {
        self.theme = theme
        updateGlobalThemeSettings()
    }
}
```

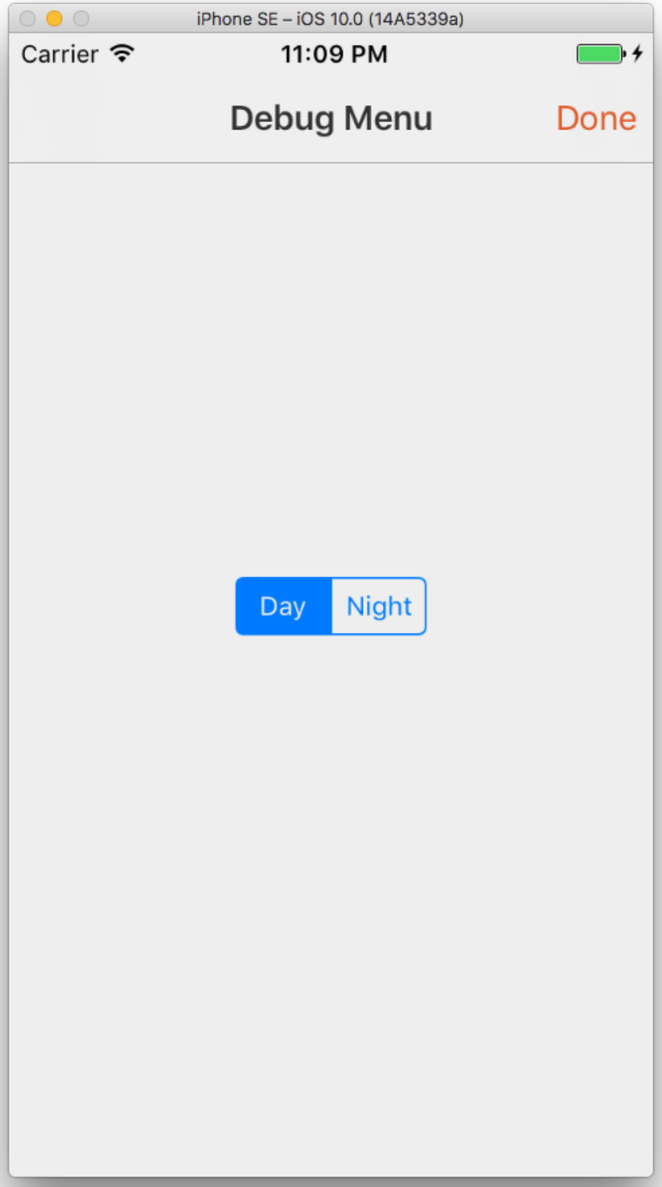
```
struct Themer {
    let theme: Theme

    init(theme: Theme) {
        self.theme = theme
        updateGlobalThemeSettings()
    }

    func updateGlobalThemeSettings() {
        updateNavigationBar()
        reloadWindow()
    }

    func updateNavigationBar() {
        let apperance = UINavigationBar.appearance()
        apperance.barTintColor = theme.backgroundColor
        apperance.titleTextAttributes = [NSForegroundColorAttributeName: theme.textColor]
        apperance.tintColor = theme.tintColor
    }
```

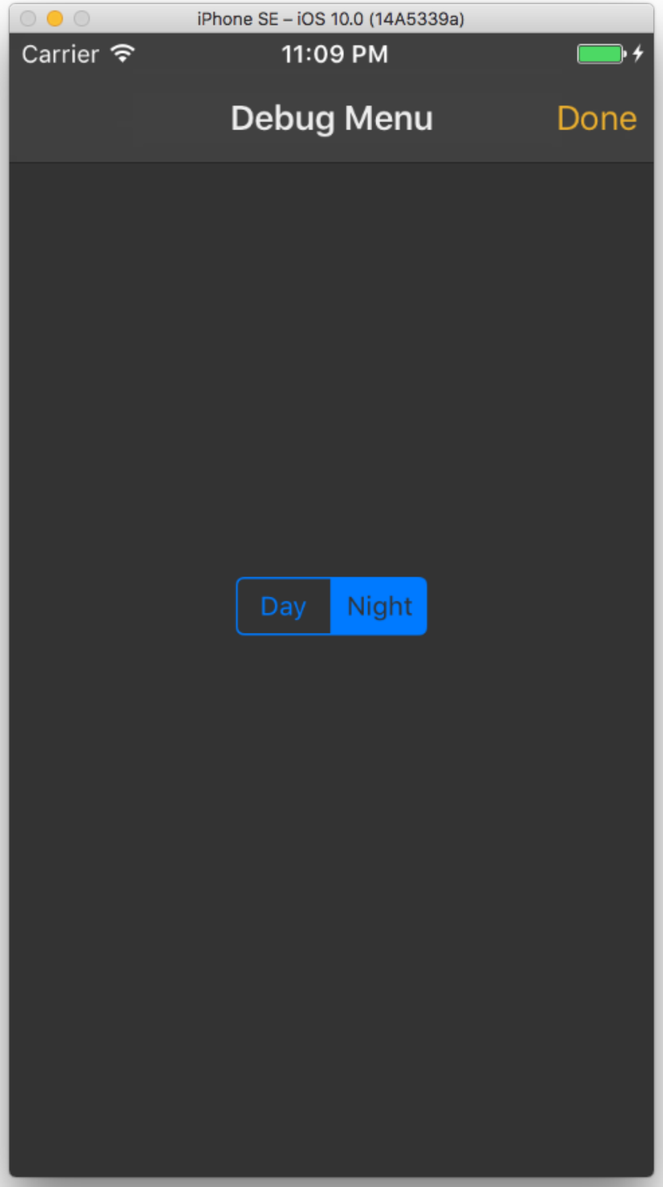
```
struct Themer {

    ...

    // Manual calls

    func theme(_ tableView: UITableView) {
        tableView.backgroundColor = theme.tableViewBackgroundColor
    }

    func theme(_ cell: UITableViewCell) {
        cell.backgroundColor = theme.tableViewCellBackgroundColor
        cell.textLabel?.textColor = theme.tableViewCellTextColor
    }

    func theme(_ view: UIView) {
        view.backgroundColor = theme.tableViewBackgroundColor
    }
}
```

Theming

Debug Menus


iPhone SE – iOS 10.0 (14A5339a)
Carrier
7:54 PM
Ranch Weather
Current Location
x4
SCENARIOS
Clear Day
Clear Night
Rain
Snow
Sleet
Wind
Fog
Cloudy


iPhone SE – iOS 10.0 (14A5339a)
Carrier
11:09 PM
Debug Menu
Done
Day
Night


iPhone SE – iOS 10.0 (14A5339a)
Carrier
11:09 PM
Debug Menu
Done
Day
Night

Story Time

Dequeueing TableView Cells

```
let cell = tableView.dequeueReusableCellWithIdentifier("CustomCell",
                    forIndexPath: indexPath) as! CustomCell
```

```
let cell = tableView.dequeue(cellOfType: CustomCell.self,
                                        indexPath: indexPath)
```

```
let cell = tableView.dequeueReusableCellWithIdentifier("CustomCell",
                    forIndexPath: indexPath) as! CustomCell


let cell = tableView.dequeue(cellOfType: CustomCell.self,
                                    indexPath: indexPath)
```

Problem

Pain Threshold

Safe. Expressive. Testable.

Big Nerd Ranch Open Source

Big Nerd Ranch loves open source. We love it so much, that sometimes we even open up some of our own source. And then we put a link to it here:

Android

Simple Item Decoration

A library for adding dividers and offsets to Android's RecyclerView using RecyclerView.ItemDecoration.

More info | View on Github

RecyclerView MultiSelect

RecyclerView MultiSelect is a tool to help implement single or multichoice selection on RecyclerView items.

More info | View on Github

Expandable RecyclerView

A custom RecyclerView which allows for an expandable view to be attached to each ViewHolder

More info | View on Github

iOS

Core Data Stack

The BNR Core Data Stack is a small framework, written in Swift, that makes it easy to

Freddy

Parsing JSON elegantly and safely can be hard, but Freddy is here to help. Freddy is a

Deferred

Lets you work with values that haven't been determined yet, like an array that's

RanchWeather

https://github.com/bignerdranch/RanchWeather

Thank you.