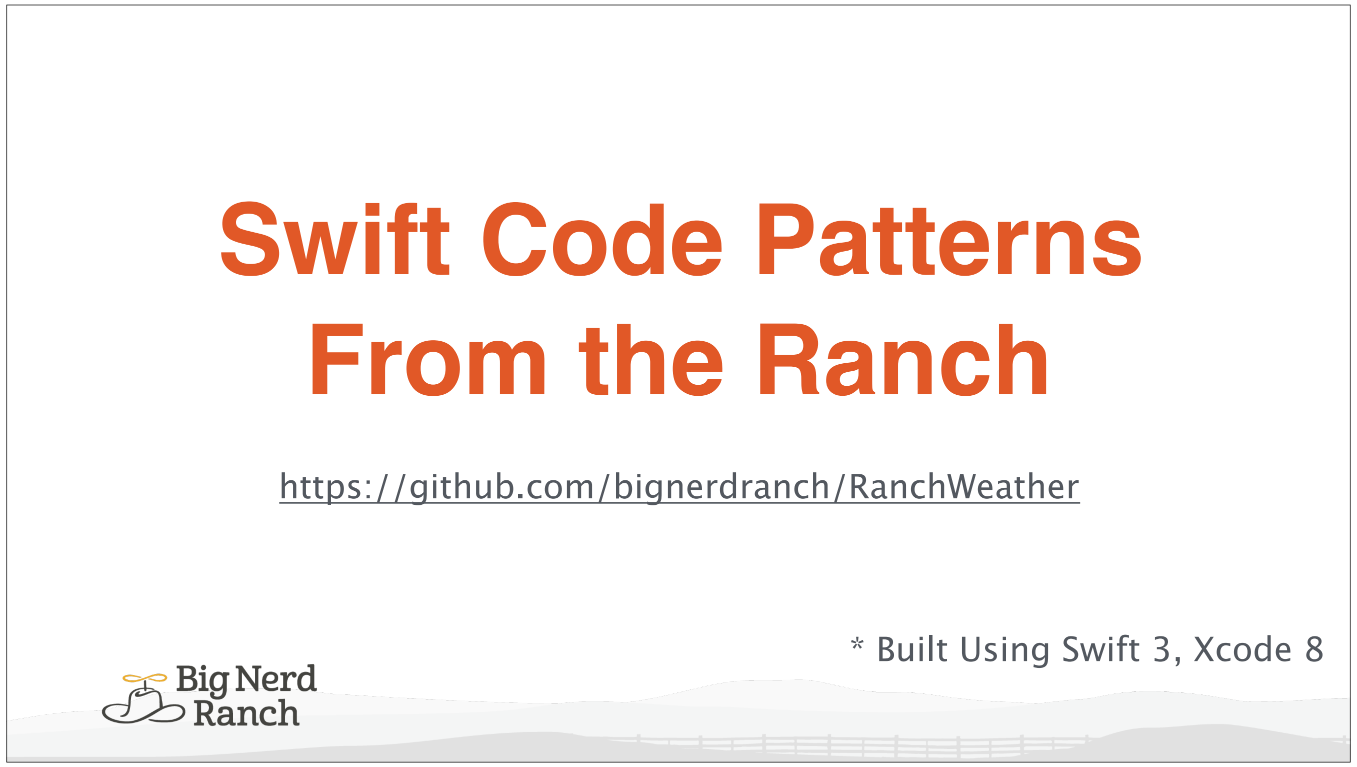


Swift Code Patterns From the Ranch

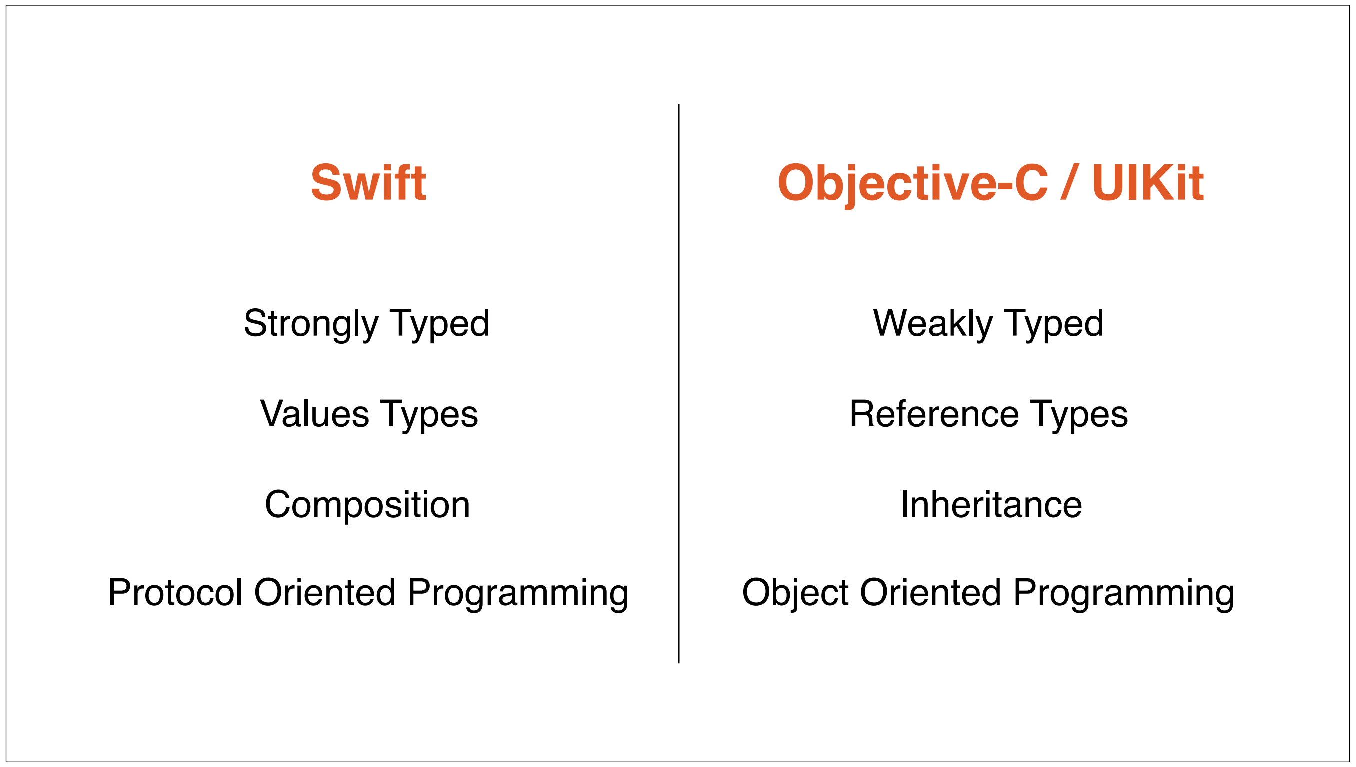
https://github.com/bignerdranch/RanchWeather

* Built Using Swift 3, Xcode 8

This talk has lots of code. If you prefer you can download the sample project and slide PDFs now and reference them as we get going.

June 2, 2014

Do you remember where you were?
Despite being a long time, comfortable Objective-C developer, I was looking forward to the upcoming experimentation.



Swift	Objective-C / UIKit
Strongly Typed	Weakly Typed
Values Types	Reference Types
Composition	Inheritance
Protocol Oriented Programming	Object Oriented Programming

Clear these were two very different languages.
Apple Credit: Obj-C Inter-op is very good.
But to use Swift as an OOP language feels wrong.

> How do we execute UIKit using Swift and honoring it's goals?

How do we execute UIKit using Swift and honoring it’s goals?

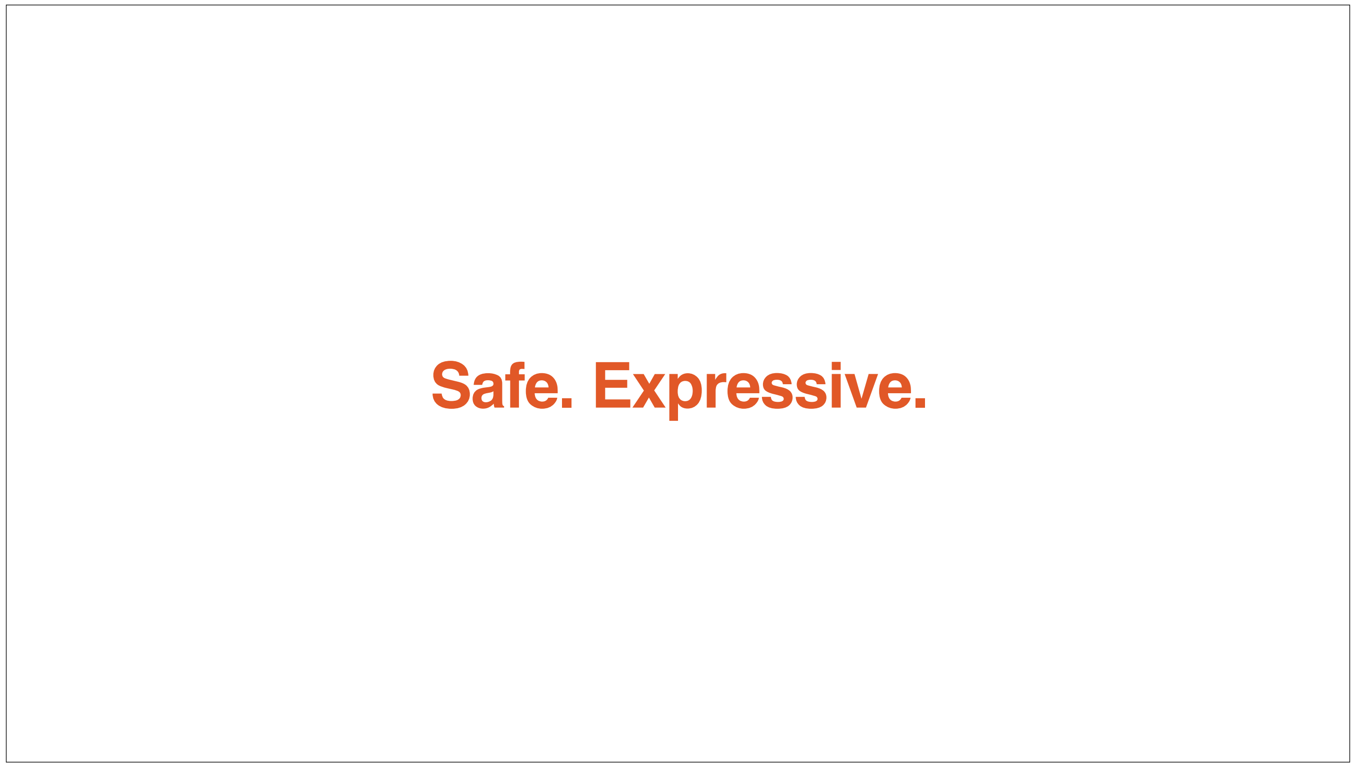
Last 2 Year Challenge
How do we execute UIKit using Swift and honoring Swift Design Goals.

> What are Swift’s Goals?

Safe. Fast. Expressive.

https://swift.org/about/

What are Swift's Goals?



Safe. Expressive.

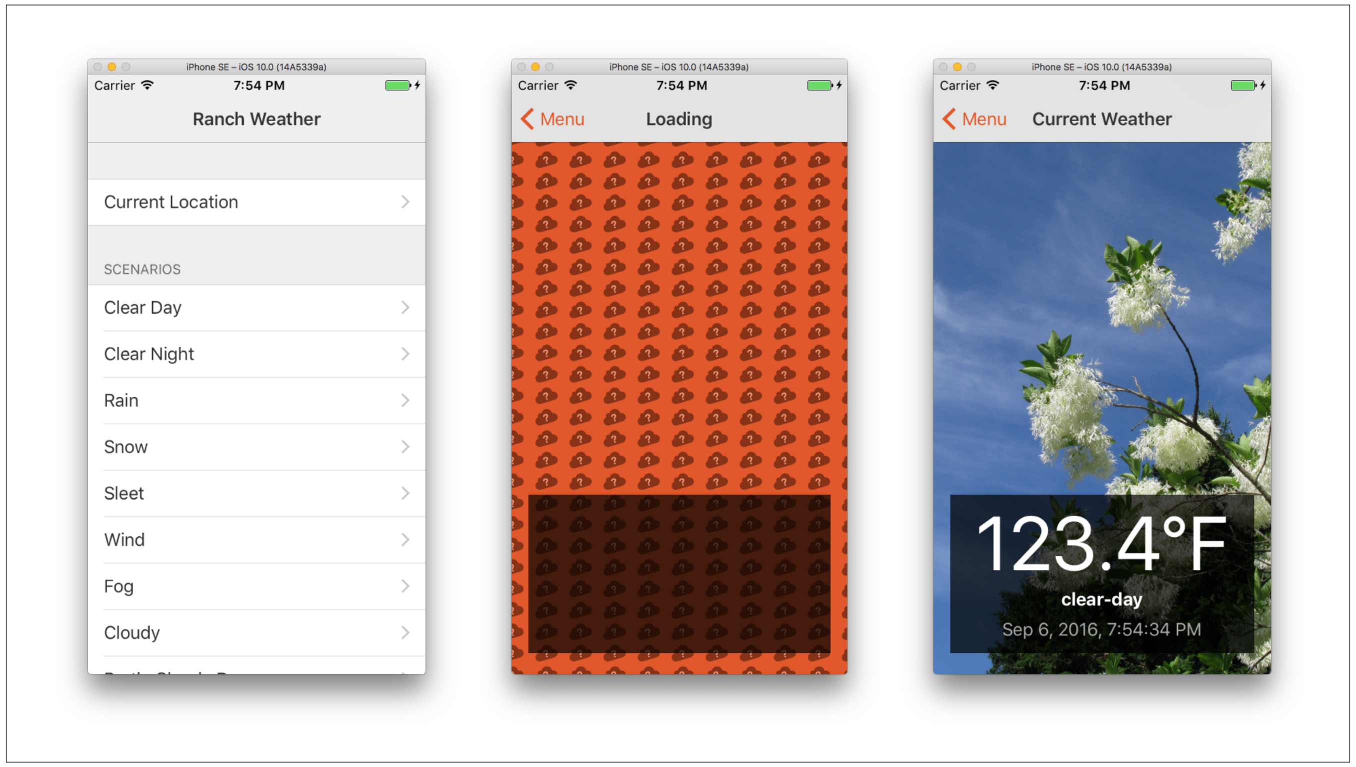
Not going to talk about Fast.

Will talk about making your code safer and more expressive.

> and for fun let’s throw in testable as well.

Safe. Expressive. Testable.

and for fun let’s throw in testable as well.


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

RanchWeather

Can’t show you many of our client projects so I’d try to transplant many of our better ideas into this new demo app.

It’s deceptively-simple looking. There is a lot of architecture under the hood. A lot of examples of how to use UIKit with Swift, honoring our goals.

Initialization

Great example of Swift asking us to express our intent.

```swift
class Car {

    let driver: Driver

    init() {

    }
}
```

Q: What’s the problem here?

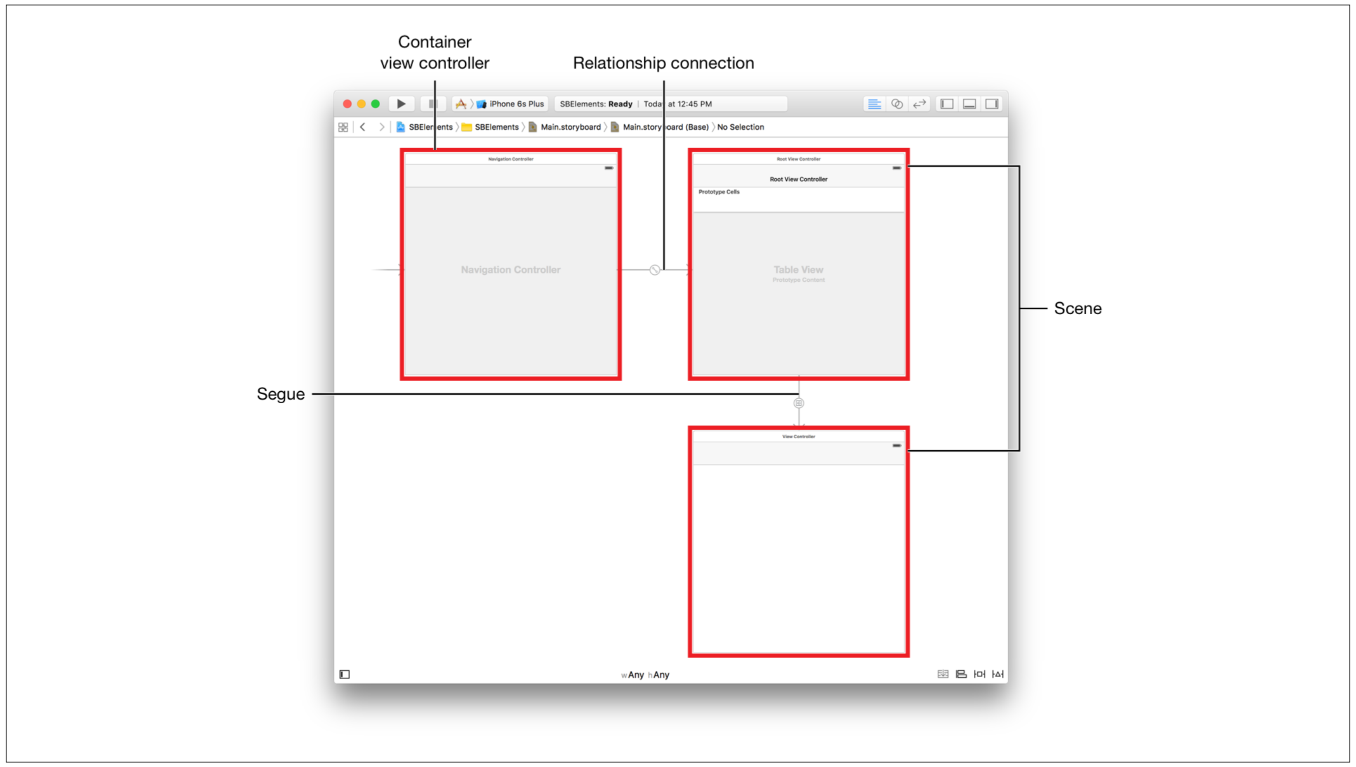
```
class Car {

    let driver: Driver

    init() {

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

```swift
class Car {

    let driver: Driver

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

}
```

View Controllers

```swift
class DisplayViewController: UIViewController {

    let weatherService: WeatherService

}
```

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

init?(coder aDecoder: NSCoder)
```


Container view controller
Relationship connection
Scene
Segue
Navigation Controller
Root View Controller
Prototype Cells
Table View
View Controller
SBElements › SBElements › Main.storyboard › Main.storyboard (Base) › No Selection
iPhone 6s Plus
SBElements: Ready | Today at 12:45 PM
wAny hAny

If we ask Apple how are we to organizer our view controllers, the answer will be Storyboards.

Many benefits. Some problems.

```swift
class DisplayViewController: UIViewController {

    let weatherService: WeatherService

}
```

So what do we do?

```swift
class DisplayViewController: UIViewController {

    var weatherService: WeatherService?

}
```

We make it an optional. But this has negative side effects.

If Swift is about intentions, this is causing us to misrepresent our own.

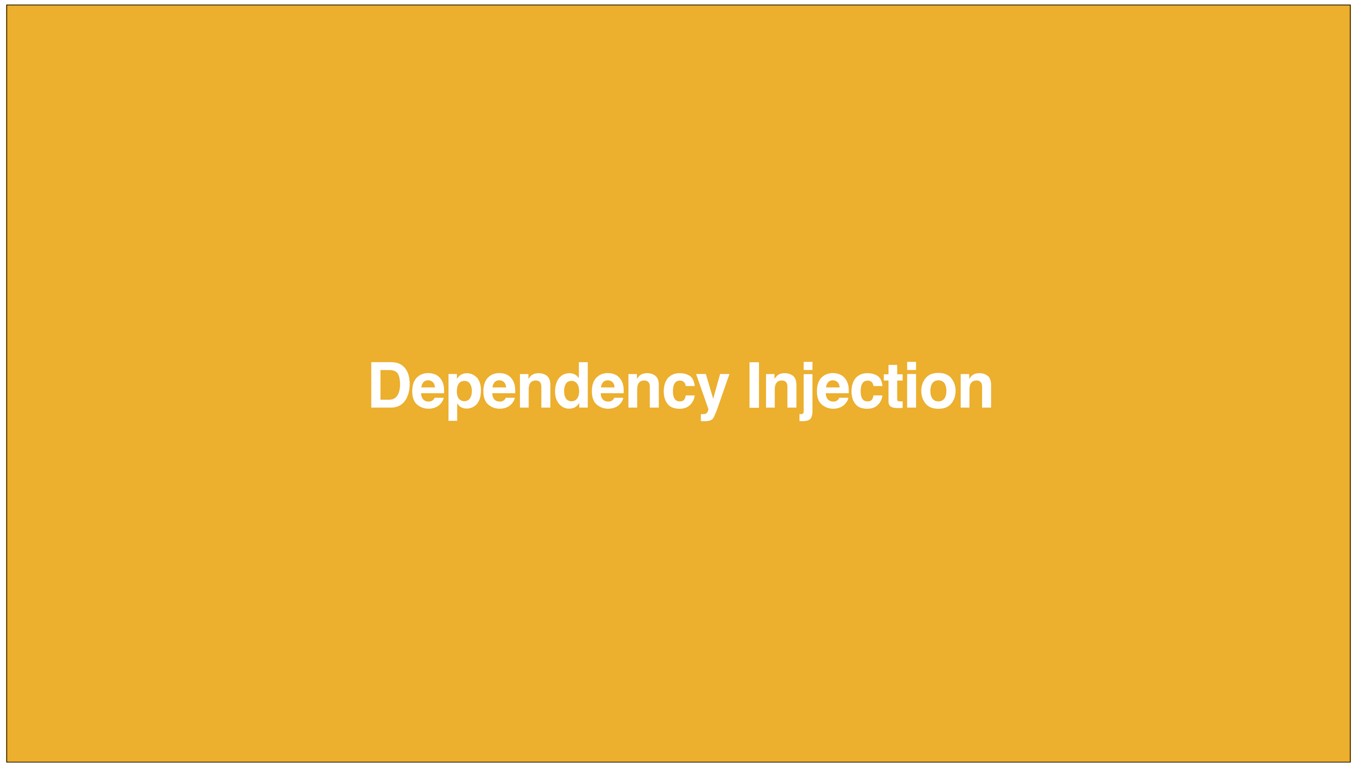
I do not want to make it seems like this LocationViewController has optional Weather Service. This is a required service.

```swift
class DisplayViewController: UIViewController {

    var weatherService: WeatherService!

}
```

Let’s improve expressing out intentions.

This is required, so we’ll make it an implicitly unwrapped optional — but not it’s unsafe.

Let’s make it safer.



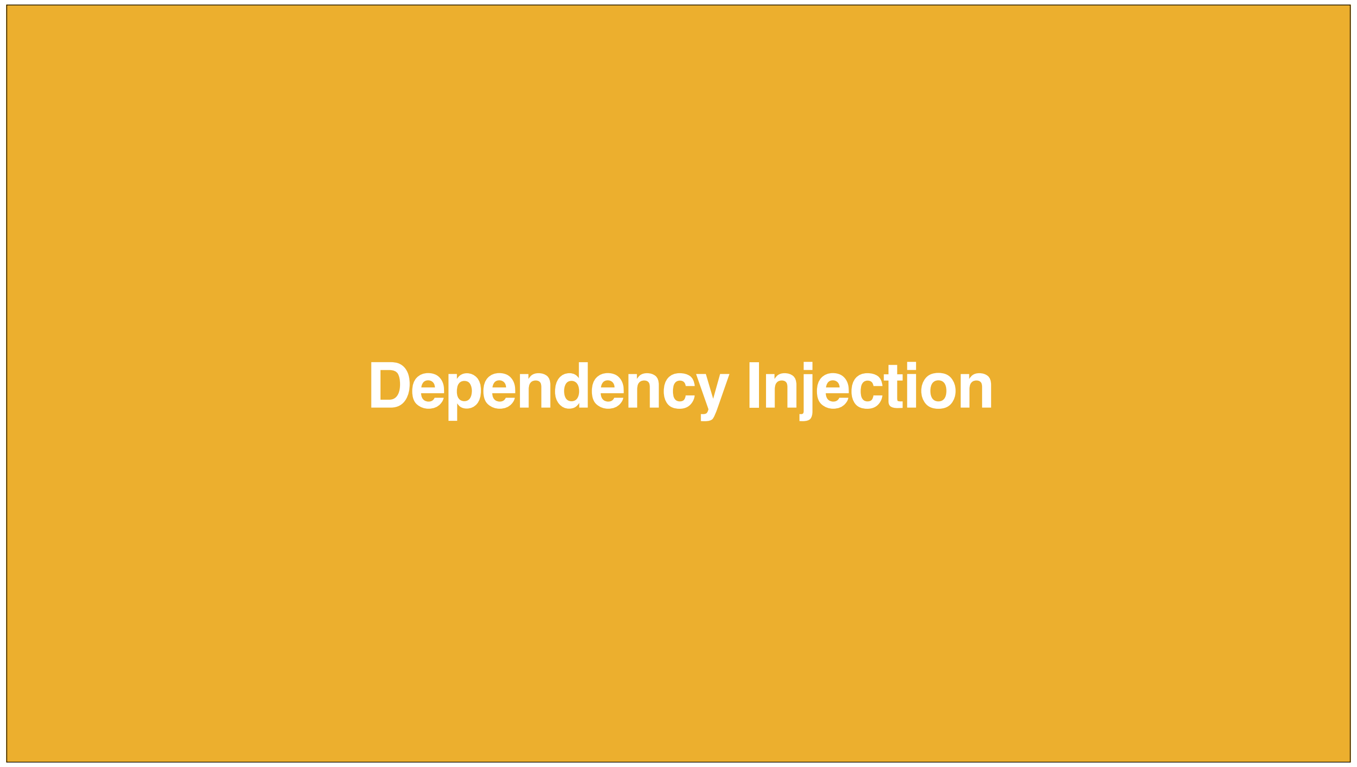
Dependency Injection

Every object should be self-contained. It should never reach out into the global space. You could configure or inject it with every object, service or store it will need.

By following DI your code can be more modular and more easily testable.

```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.

    // assertDependencies is a method intended to verify that our
    // implicitly unwrapped optionals 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 assertDependencies()
}
```

```swift
class WeatherDisplayViewController: UIViewController {

    var weatherService: WeatherService!
    var locationService: LocationService!

    override func viewDidLoad() {
        super.viewDidLoad()
        assertDependencies()
    }

}

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

    func assertDependencies() {
        assert(weatherService != nil)
        assert(locationService != nil)
    }
}
```



Dependency Injection

Big topic.
Helps keep view controllers less brittle, easier to test.
Very important for some things we’ll explore later.

DRY Strings

Q: Know what DRY stands for?

Moving to a lighter topic, DRY Strings.
Using Swift types like structs and enums to help symbolize terms that would normally be strings.
Avoid typos. Get code completion. Get complier help.

```swift
extension UserDefaults {

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

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

}

string(forKey: Key.themeIdentifier)


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

We'll talk more about UserDefaults later but here is an example of isolating strings for the keys.

Another example is notification names. New in Swift 3 this is actually a type.

```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 Identifier: String {
        case Main
        case WeatherDisplay
        case Feedback
        case DebugMenu
    }
}
```

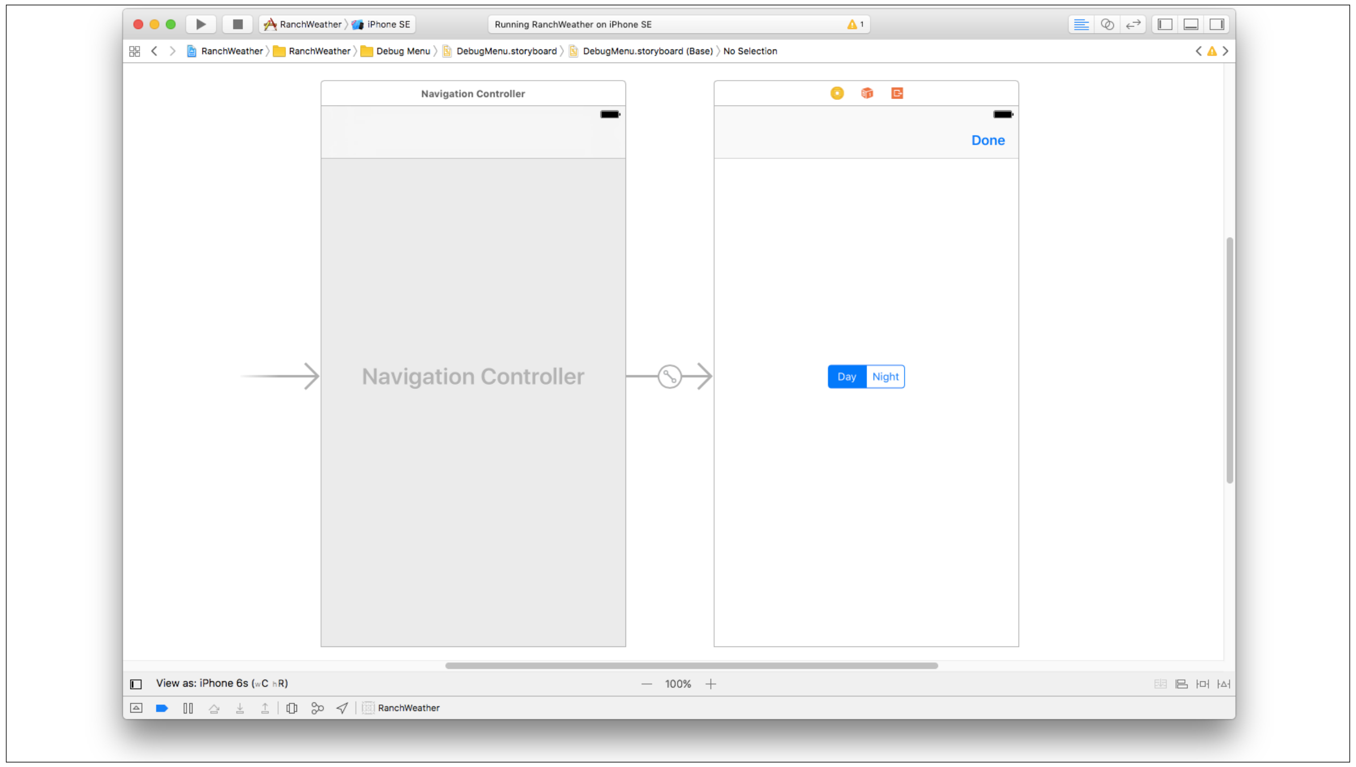
Storyboard names.

Storyboard Management

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

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

let storyboard = UIStoryboard(.WeatherDisplay)
```

In theory you could get a storyboard with this convenience initializer, but we actually keep it private.

```swift
extension UIStoryboard {

    static func weatherDisplayViewController() -> WeatherDisplayViewController {
        return UIStoryboard(.WeatherDisplay).instantiateInitialViewController() as!
WeatherDisplayViewController
    }

}

let vc = UIStoryboard.weatherDisplayViewController()
```


RanchWeather 〉 iPhone SE
Running RanchWeather on iPhone SE
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)
```

Async, Result, Error Enums

```swift
struct WeatherService {

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

// NEED EXAMPLE
```

Error Handling with Enums

Theming

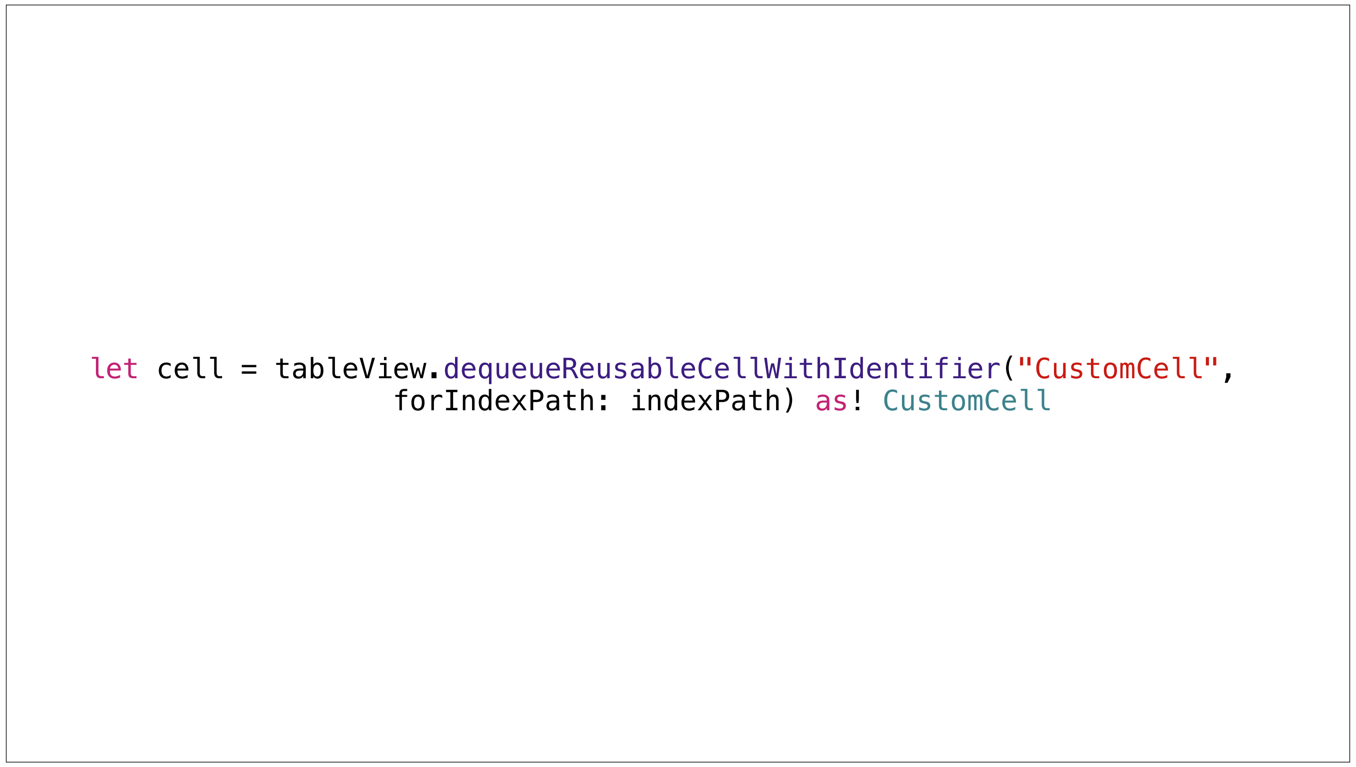
UIColor

Localization

User Defaults

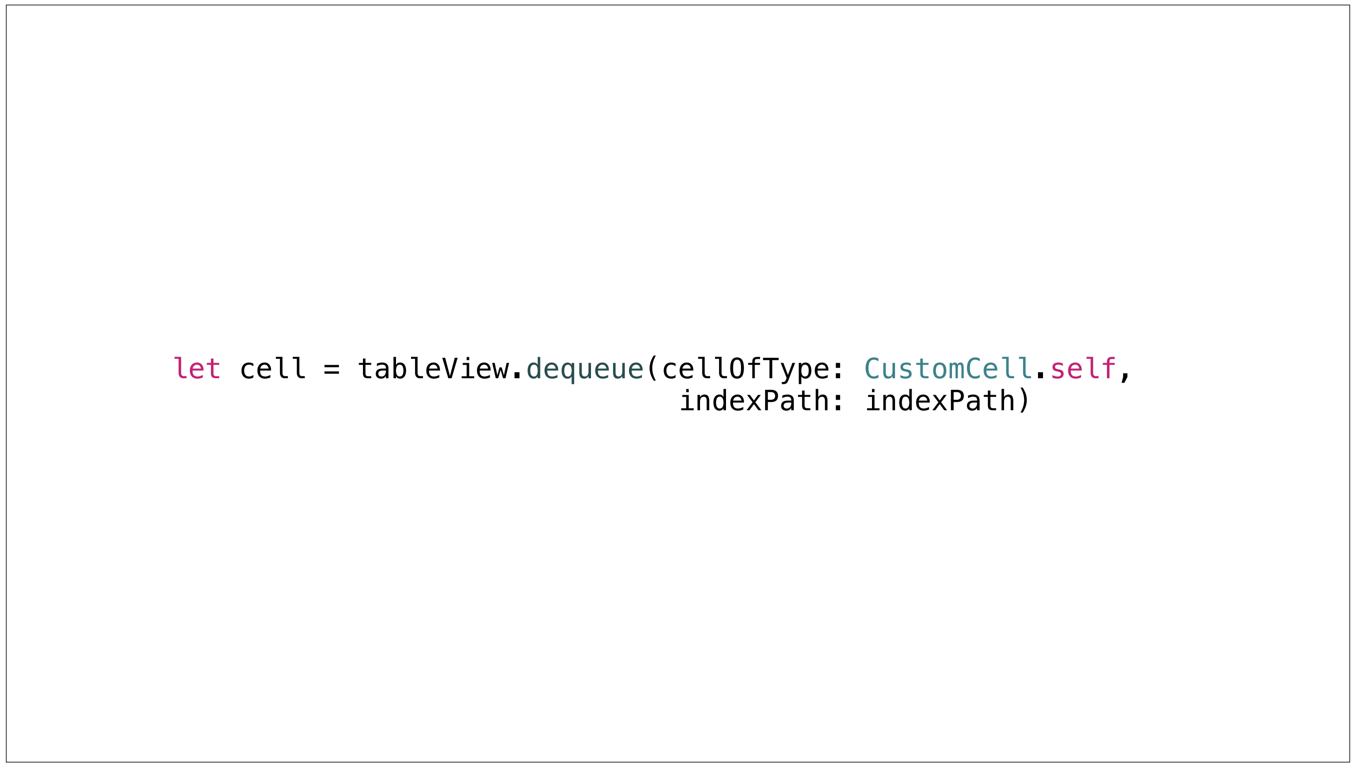
UIAlertController Helper

Dequeueing TableView Cells



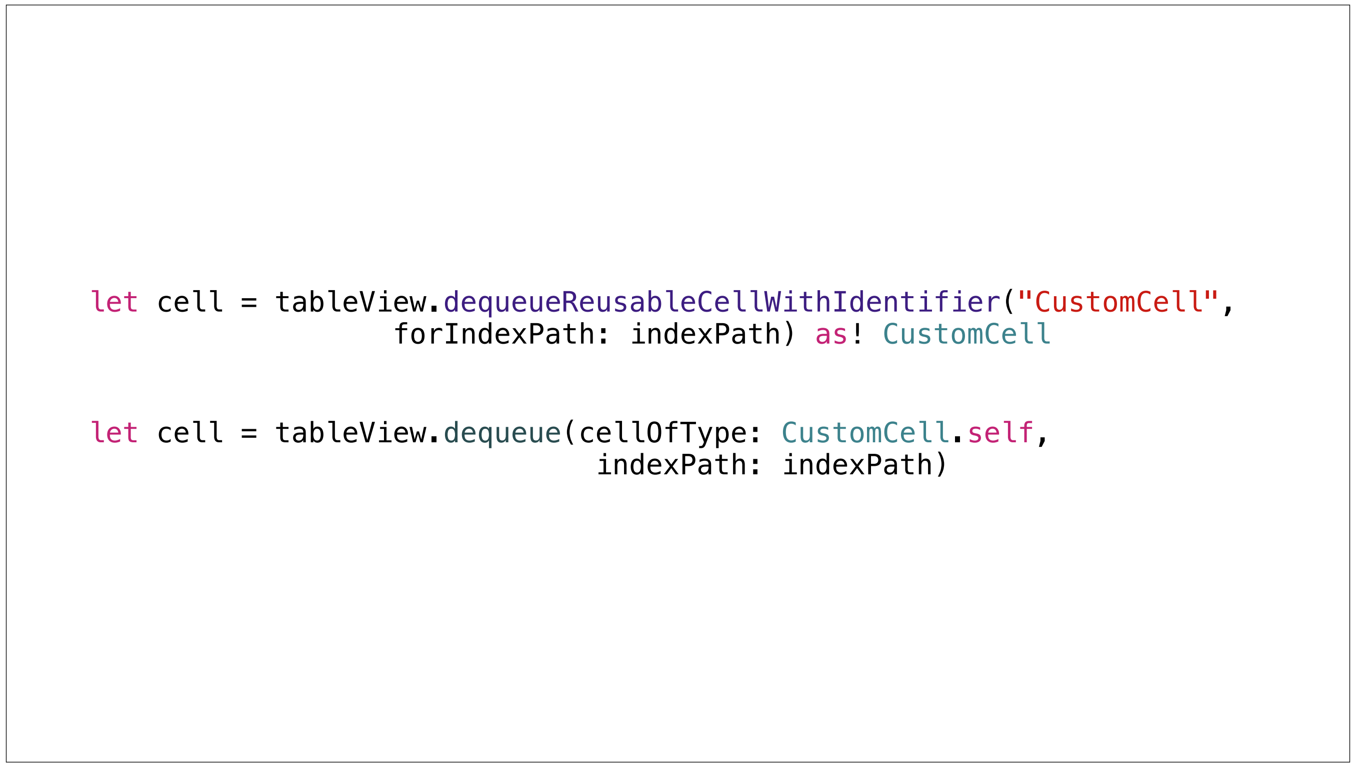
```
let cell = tableView.dequeueReusableCellWithIdentifier("CustomCell",
                 forIndexPath: indexPath) as! CustomCell
```

New method does two things in one, handles casting and reuse id.



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

New method does two things in one, handles casting and reuse id.



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


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

New method does two things in one, handles casting and reuse id.

```
extension UITableView {

    func dequeue<V: UITableViewCell>(cellOfType type: V.Type,
           indexPath: NSIndexPath, reuseIdentifier: String? = nil) -> V {
        let identifier = reuseIdentifier ?? String(type)
        return dequeueReusableCellWithIdentifier(identifier,
                                              forIndexPath: indexPath) as! V
    }

}

String(type) // old way NSStringFromClass(type)
```

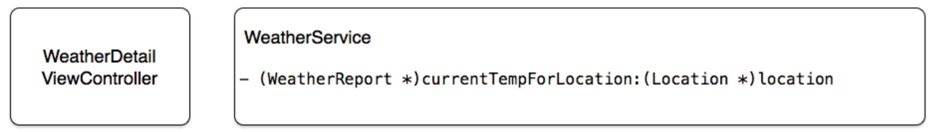
Optional argument, generic of type.
Problem: Should enforce usage.

Problem

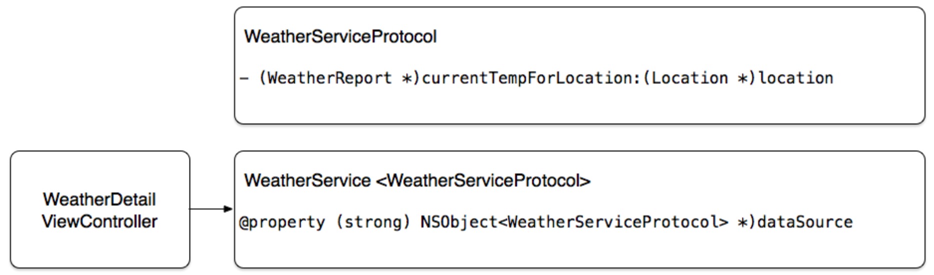
Pain Threshold

Launch / Debug Menus

Many DataSources Pattern

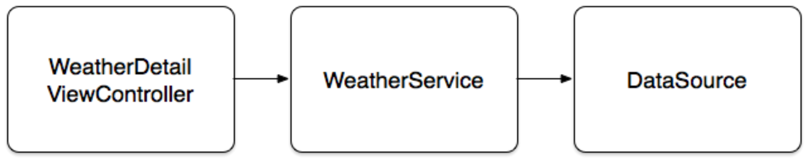


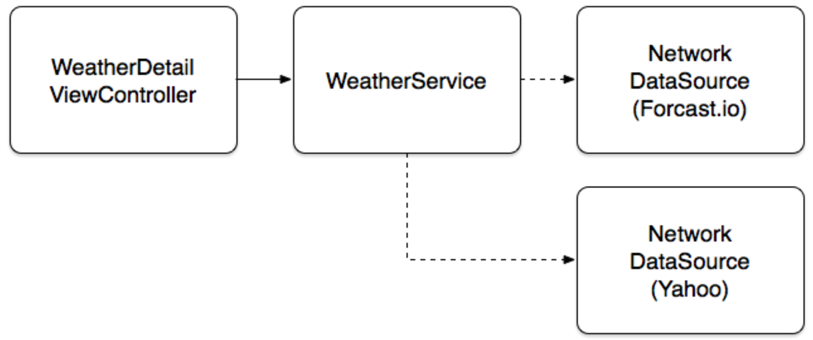
```
- (WeatherReport *)currentTempForLocation:(Location *)location
```

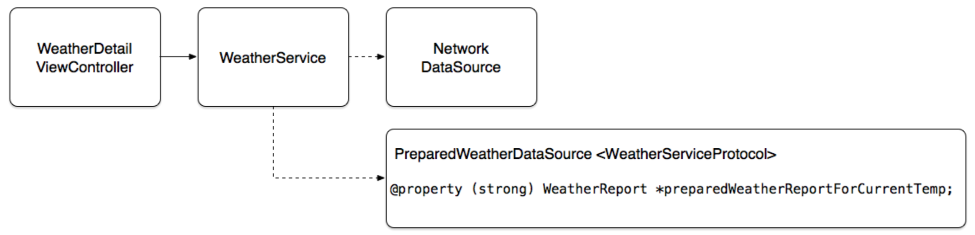

WeatherServiceProtocol
WeatherDetail ViewController
WeatherService <WeatherServiceProtocol>

```
- (WeatherReport *)currentTempForLocation:(Location *)location
```

```
@property (strong) NSObject<WeatherServiceProtocol> *)dataSource
```

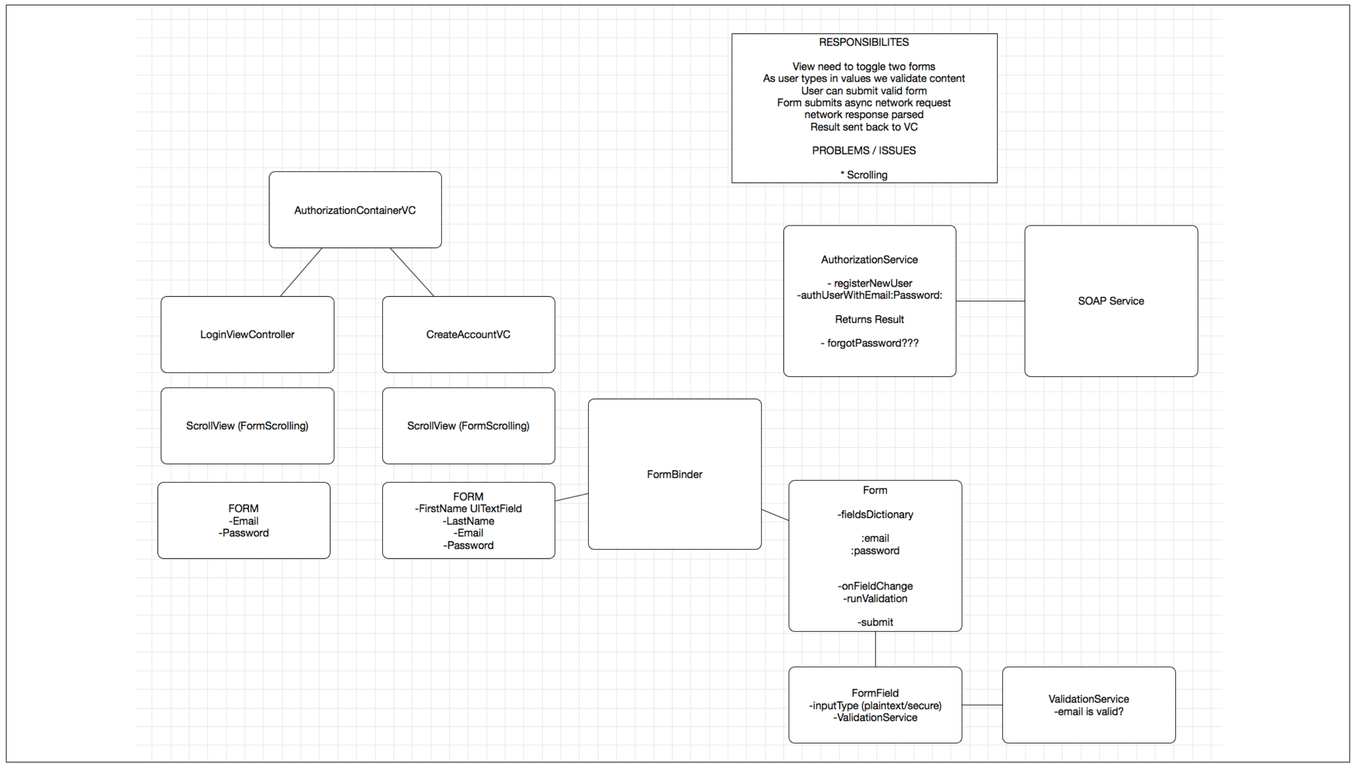

WeatherDetail ViewController
WeatherService
DataSource


WeatherDetail ViewController
WeatherService
Network DataSource (Forcast.io)
Network DataSource (Yahoo)


WeatherDetail ViewController
WeatherService
Network DataSource
PreparedWeatherDataSource <WeatherServiceProtocol>

```
@property (strong) WeatherReport *preparedWeatherReportForCurrentTemp;
```

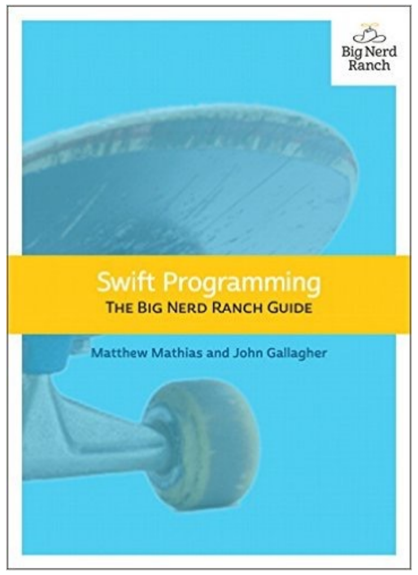
Forms

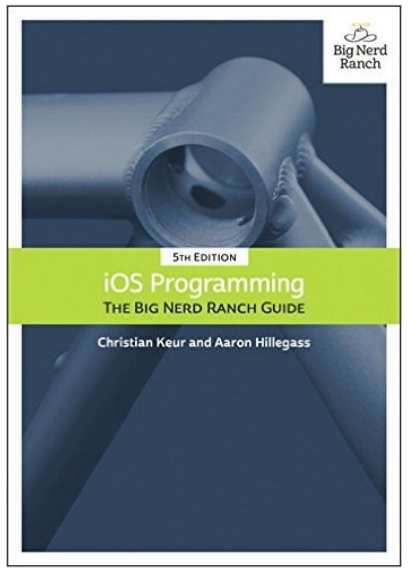

RESPONSIBILITES
View need to toggle two forms
As user types in values we validate content
User can submit valid form
Form submits async network request
network response parsed
Result sent back to VC
PROBLEMS / ISSUES
* Scrolling
AuthorizationContainerVC
LoginViewController
CreateAccountVC
ScrollView (FormScrolling)
ScrollView (FormScrolling)
FORM
-Email
-Password
FORM
-FirstName UITextField
-LastName
-Email
-Password
FormBinder
AuthorizationService
- registerNewUser
-authUserWithEmail:Password:
Returns Result
- forgotPassword???
SOAP Service
Form
-fieldsDictionary
:email
:password
-onFieldChange
-runValidation
-submit
FormField
-inputType (plaintext/secure)
-ValidationService
ValidationService
-email is valid?

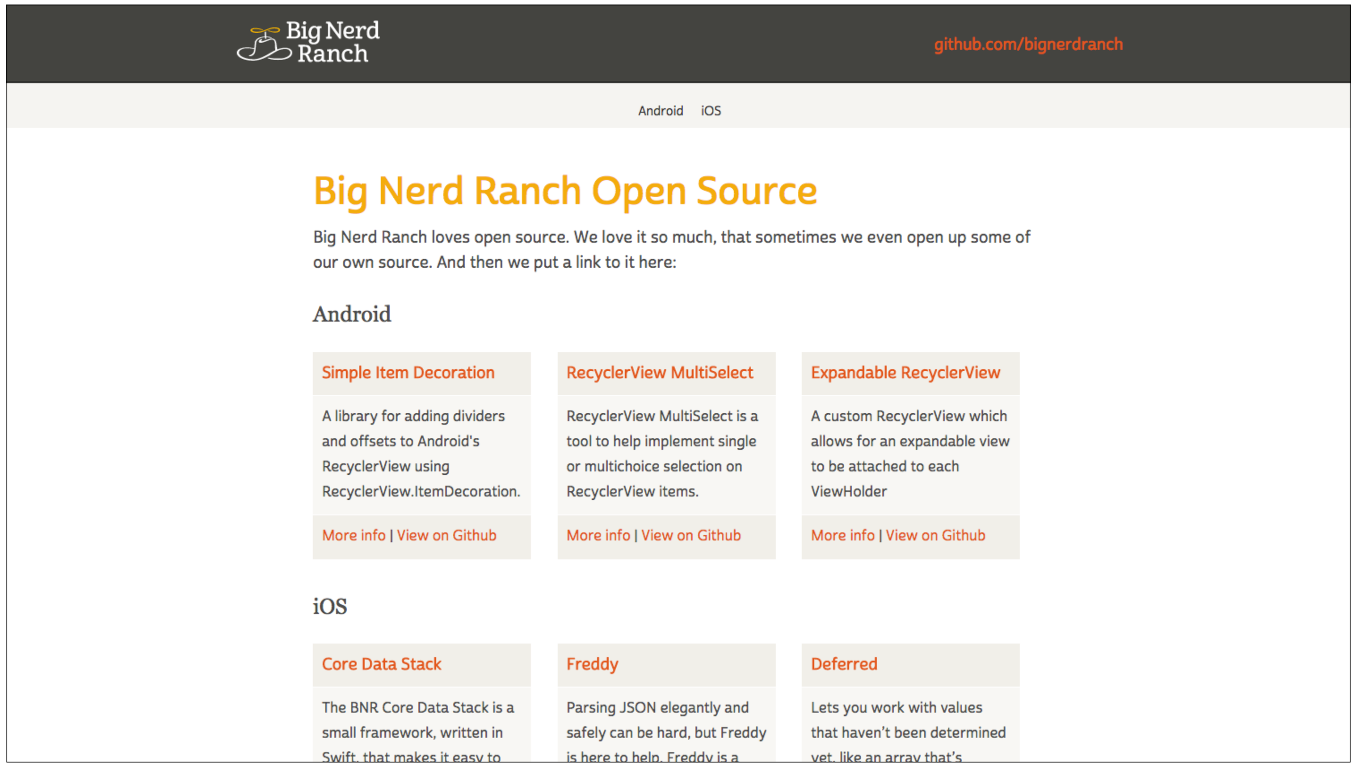
Thank you.

Submit your own patterns!

Resources


Big Nerd Ranch
Swift Programming
The Big Nerd Ranch Guide
Matthew Mathias and John Gallagher


Big Nerd Ranch
5th Edition
iOS Programming
The Big Nerd Ranch Guide
Christian Keur and Aaron Hillegass


Big Nerd Ranch

github.com/bignerdranch

Android iOS

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

Mike Zornek
Philadelphia
Big Nerd Ranch, Instructor / Developer
Long Apple History

Swift Code Patterns From the Ranch

https://github.com/bignerdranch/RanchWeather

Big Nerd Ranch