Flow Coordinators IRL

Marina Gornostaeva
@hybridcattt

storytel.com

storytel

mofibo

Some context

- Started small, rapid growth, made by one person or an agency
- “Legacy”, old codebase becomes a burden
- Need scalability, flexibility, room for experimentation
- New app in Swift 3

Pragma Conference
pragmaconference.com

Krzysztof Zabłocki
@merowing_

Improve your iOS Architecture with FlowControllers

http://merowing.info/2016/01/improve-your-ios-architecture-with-flowcontrollers/
https://www.youtube.com/watch?v=2GSPPvn0P3Y

Problem

- MVC, MVVM, etc - only the “view controller” layer
  No router or dependency injection. Who and when creates M, V, and VM?

- Multiple presentation types for the same VC
- Reuse same VC in different scenarios
- Something else? Don’t know until it’s too late 😵

AppDelegate
+ window

VCs

VCs

?

```swift
func doneButtonTapped() {
    let vc = NextViewController()

    if !Device.isIPad {
        self.navigationController?.pushViewController(vc, animated: true)
    } else {
        let nav = UINavigationController(rootViewController: vc)
        nav.modalPresentationStyle = UIModalPresentationStyle.popover
        vc.preferredContentSize = CGSize(width: 500, height: 600)

        if let popover = nav.popoverPresentationController {
            popover.delegate = self
            popover.sourceView = self.view
            popover.sourceRect = CGRect(x: 100, y: 100, width: 0, height: 0)
        }
        self.present(nav, animated: true, completion: nil)
    }
}
```

- Unnecessary dependencies, each of the VCs depends on the ones it presents.
  If MVVM - 😱
- Poor reusability
- Spaghetti code everywhere, duplication of “if-else”code
- Actually application logic is spread out
- VC/VMs have side effects, hard to test.

Fix it

```
class MyViewController {
    var onDone: (() -> Void)?

    func doneButtonTapped() {
        self.onDone?()
    }
}

let myVC = MyViewController()
myVC.onDone = {
    // . . .
}
```

```
let vc = createVC()
var executed = false
vc.onDone = {
    executed = true
}
//! add code here to trigger done state
expect(executed).toEventually(beTruthy())
```

- No reference to other screens
- VCs don’t use any UIKit presentation methods
- VCs have interface that other objects register to (delegate or block)
- No need for singletons or crazy amount of dependency injection

Flow Controller

- Simple object that manages a part of the application
- Configures view controllers
- Listens to events on each VC and use that to coordinate between them.
- many more…. but how?

Real life

AppDelegate
+ window

VCs

VCs

?

AppDelegate
+ window

take this window

ApplicationController

VCs

VCs

UIFactory(-es)

take this wireframe and UI factories
coordinate away!

FlowCoordinator(s)

present this, push that
show this error

Wireframe

TabBarWireframe

HamburgerWireframe

iPad?

AppDelegate

```
class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        self.createWindow()
        self.createApplicationController()

        self.applicationController?.finishLaunching()
        return true
    }

    func createApplicationController() {
        guard let window = self.window else {
            fatalError("Failed to install a window.")
        }

        self.applicationController = ApplicationController(window: window)
    }
}
```

ApplicationController

```
protocol MainFlowCoordinator: class {
    func installUI()
    func flushAllUI()
}

class ApplicationController {
    let flowCoordinator: MainFlowCoordinator

    convenience init(window: UIWindow) {
        let tabBarWireframe = ApplicationTabBarWireframe(window: window)

        let loginUIFactory = LoginUIFactory(. . .)
        let explorationUIFactory = ExplorationUIFactory(. . .)

        let appFlowCoordinator = ApplicationFlowCoordinator(wireframe: tabBarWireframe,
                                                            loginFactory: loginUIFactory,
                                                            explorationFactory: explorationUIFactory)

        self.init(flowCoordinator: appFlowCoordinator)
    }

    init(flowCoordinator: MainFlowCoordinator, . . .) {
        self.flowCoordinator = flowCoordinator
    }

    func finishLaunching() {     // <---- called by AppDelegate
        self.flowCoordinator.installUI()
    }
    func wentToBackground() {     // <---- called by AppDelegate
        self.flowCoordinator.flushAllUI()
    }
}
```


```
typealias RootLevelViewControllerRepresentable = ViewControllerRepresentable & RootLevelRepresentable

typealias ModallyPresentableViewController = ViewControllerRepresentable & ModallyPresentable

typealias FullscreenMinimizableViewController = ViewControllerRepresentable & FullscreenMinimizable


/// A class that provides an abstracted interface for UI navigation.
protocol ApplicationWireframe: class {

    var rootViewControllers: [UIViewController] { get }

    func install(rootItems items: [RootLevelViewControllerRepresentable])

    func push(viewController: UIViewController, from sender: UIViewController)

    func presentModally(_ viewController: ModallyPresentableViewController, from sender: UIViewController)

    func dismiss(_ viewController: ModallyPresentableViewController)

    func presentError(_ errorDisplayable: ErrorDisplayable, from sender: UIViewController)

    func installMinimizable(_ viewController: FullscreenMinimizableViewController, animated: Bool)
    func removeMinimizable(animated: Bool)
    func minimizeFullscreenView(animated: Bool)
    func maximizeMinimizableView(animated: Bool)

}
```

```swift
class ApplicationTabBarWireframe: ApplicationWireframe {

    let window: UIWindow
    let tabbarController: UITabBarWithMinimizableViewController

    var animateAllTransitions: Bool = true

    init(window: UIWindow) {
        self.window = window
        self.tabbarController = UITabBarWithMinimizableViewController()
    }

    func install(rootItems items: [RootLevelViewControllerRepresentable]) {
        self.window.rootViewController = self.tabbarController
        // Use tabbar items
        self.tabbarController.viewControllers = . . .
    }
}



// MARK: - TabBar Related -

extension RootLevelRepresentation {
    func makeTabBarItem() -> UITabBarItem {
        let image = (self.imageName != nil) ? UIImage(named: self.imageName!) : nil
        let tabbarItem = UITabBarItem(title: self.title, image: image, tag: 0)
        return tabbarItem
    }
}
```

```
class ApplicationTabBarWireframe: ApplicationWireframe {

    func presentError(_ errorDisplayable: ErrorDisplayable, from sender: UIViewController) {

        let alert = UIAlertController(title: errorDisplayable.title,
                                      message: errorDisplayable.message,
                                      preferredStyle: UIAlertControllerStyle.alert)

        alert.addAction(UIAlertAction(title: errorDisplayable.dismissButtonStyle.displayableString,
                                      style: UIAlertActionStyle.default,
                                      handler: { [weak alert] (action: UIAlertAction) in
            alert?.dismiss(animated: self.animateAllTransitions,
                           completion: nil)
        }))

        sender.present(alert, animated: self.animateAllTransitions, completion: nil)
    }
}
```

```swift
class ApplicationTabBarWireframe: ApplicationWireframe {

    func presentModally(_ presentable: ModallyPresentableViewController,
                        from sender: UIViewController) {

        let viewController = presentable.asViewController
        if let dismissButtonStyle = presentable.allowedModalDismissButtonStyle {
            viewController.tbw_configureDismissButton(style: dismissButtonStyle,
                                                      action: { [weak self, weak presentable] in
                guard let sself = self, let ppresentable = presentable else { return }

                sself.dismiss(ppresentable)
            })
        }

        let navigationController = viewController as? UINavigationController ??
                                   UINavigationController(rootViewController: viewController)
        navigationController.delegate = self.navigationControllerDelegate

        sender.present(navigationController, animated: self.animateAllTransitions, completion: nil)
    }
}
```

Flow Coordinator

```
class ApplicationFlowCoordinator: MainFlowCoordinator {

    let wireframe: ApplicationWireframe

    let demoPagesFactory: DemoParentPagesUIFactory
    let loginFactory: LoginUIFactory
    let explorationFactory: ExplorationUIFactory

    func installUI() {
        let vc1 = self.demoPagesFactory.makePageOne()
        vc1.rootLevelRepresentation.imageName = "demo_circle_icon"
        self.configurePageOne(vc1)

        let vc2 = self.demoPagesFactory.makePageTwo()
        vc2.rootLevelRepresentation.imageName = "demo_square_icon"
        self.configurePageTwo(vc2)

        let vc3 = self.demoPagesFactory.makePageThree()
        vc3.rootLevelRepresentation.imageName = "demo_circle_icon"
        self.configurePageThree(vc3)

        let rootItems: [RootLevelViewControllerRepresentable] = [vc1, vc2, vc3]
        self.wireframe.install(rootItems: rootItems)
    }
}
```

List - tap on a row

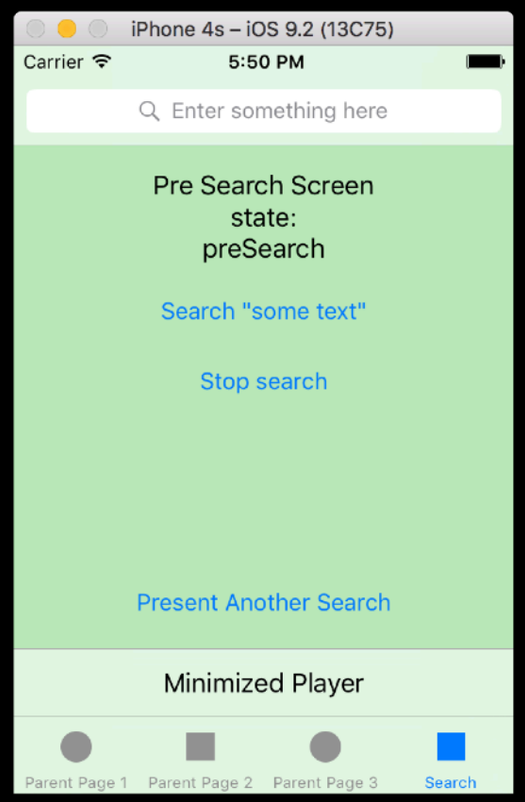
```
class ApplicationFlowCoordinator: MainFlowCoordinator {

    func pushProductList(listId: String, from viewController: UIViewController) {

        let productListVC = self.explorationFactory.makeProductList(listId: listId)
        productListVC.st_prefersTransparentNavigationBar = true

        productListVC.onProductRowPressed =
                  { [weak self, weak productListVC] (productBox: ProductBox) in

             guard let sself = self, let sender = productListVC else {
                 return
             }

             sself.pushProductDetailsPage(productBox: productBox, from: sender)
        }

        self.wireframe.push(viewController: productListVC, from: viewController)
    }
}
```

Dismiss-on-done

```
class ApplicationFlowCoordinator: MainFlowCoordinator {

    func presentLogin(from viewController: UIViewController) {

        let loginVC = self.loginFactory.makeLoginViewController()
        loginVC.allowedModalDismissButtonStyle = ModalDismissButtonStyle.done

        loginVC.onLoginSucceeded = { [weak self, weak loginVC] in
            guard let sself = self, let vc = loginVC else { return }
            DispatchQueue.main.asyncAfter(deadline: .now() + .milliseconds(500), execute: {
                sself.wireframe.dismiss(vc)
            })
        }

        loginVC.onErrorOccurred = { [weak self, weak loginVC] (errorToDisplay: ErrorDisplayable) in
            guard let sself = self, let vc = loginVC else { return }

            sself.wireframe.presentError(errorToDisplay, from: vc)
        }

        self.wireframe.presentModally(loginVC, from: viewController)
    }
}
```

Custom navigation

```
class ApplicationFlowCoordinator: MainFlowCoordinator {

    func addMinimizablePlayer() {

        let controls = self.demoPagesFactory.makeMinimizedPlayerControls()
        self.configureDemoPlayerControls(controls)

        let minimizable = self.consumptionFactory.makeFullscreenMinimizable(controlsVC: controls)
        minimizable.onMinimizedPlayerPressed = { [weak self] in
            self?.wireframe.maximizeMinimizableView(animated: true)
        }
        self.wireframe.installMinimizable(minimizable, animated: true)
    }

    func configureDemoPlayerControls(_ vc: DemoMinimizedControlsViewController) {

        vc.onAddPlayerPressed = { [weak self] in
            self?.addMinimizablePlayer()
        }
        vc.onClosePlayerPressed = { [weak self] in
            self?.wireframe.removeMinimizable(animated: true)
        }
        vc.onMinimizePressed = { [weak self] in
            self?.wireframe.minimizeFullscreenView(animated: true)
        }
        vc.onMaximizePressed = { [weak self] in
            self?.wireframe.maximizeMinimizableView(animated: true)
        }
    }
}
```



- Each view controller is absolutely reusable
- **How to present** is separated from **what to present**
- **When to present** is separated from **what & how**
- No hacks, no boilerplate: each custom presentation style is a supported feature
- Easy to A/B test or change UIs

Thank you!

Marina Gornostaeva
@hybridcattt

storytel.com