

RxSwift at momondo

Copenhagen Cocoa Meetup
Dennis Torbichuk @dtrukr
28 February 2017

RxSwift at momondo

Focus areas for this presentation:

- Problem of "state" in typical applications
- Brief introduction into RxSwift and how it helps to minimize state
- Practical examples of using RxSwift at momondo

Timetable feature in momondo app

The screenshot shows a mobile application interface for a multi-city trip. At the top, there are status icons for signal strength, battery level (100%), and time (09.41). Below the header, there are navigation buttons: 'Back' (with a left arrow), 'Multi-city trip' (centered), and 'Menu' (with a right arrow). A tab bar at the bottom has two tabs: 'Tickets' (selected, grey background) and 'Timetable' (white background).

Below the tabs, there are four flight options, each with a 'SELECT' button above it:

- Flight 1:** SAS Scandinavian Airlines. Departure: CPH 19.35, Arrival: TXL 20.30, Duration: 55m, Price: 571 DKK. Rating: 😊 10.
- Flight 2:** Norwegian Air International. Departure: CPH 20.10, Arrival: SXF 21.10, Duration: 1h 0m, Price: 563 DKK. Rating: 😊 8,5.
- Flight 3:** easyJet. Departure: CPH 21.35, Arrival: SXF 22.40, Duration: 1h 5m, Price: 572 DKK. Rating: 😢 2,5.

At the bottom of the screen, there is a 'Filter results' button with a filter icon and a large blue circular button with three horizontal lines.

Search

[Save as](#)[Share](#)[Export](#)[Tools](#)

text ~ "builder" AND issuetype = Bug AND project = MIS

Order by ▾

 MIS-1473

Tablet: Ticket builder selection line is missing...

 MIS-240

Crah when closing segment details page in ti...

 MIS-236

Crash when selecting departure for one-way ...

 MIS-1043

Reproducible crash in ticket builder

 MIS-1465

Crash when tapping second leg tab for onew...

 MIS-1474

Second leg result list in ticket builder is empty

 MIS-1469

Tablet: Return leg is incorrectly displayed in ti...

 MIS-1463

Ticket builder result list for second leg is not ...

 MIS-1462

It is not possible to open best fit from ticket b...

 MIS-247

Filters are available in view for selected depa...



1 2 ▶



iOS Search / MIS-240

Crahh when closing segment details page in ticket builder

[Edit](#)

[Comment](#)

[Assign](#)

[More ▾](#)

[Reopen](#)

[QA Review](#)

2 of 61 ▲ ▼

61 bugs

Details

Type:  Bug

Status:  DONE [\(View Workflow\)](#)

Priority:  Critical

Resolution: Done

Affects Version/s: None

Fix Version/s: None

Component/s: None

Labels: [flights](#)

Environment: Branch:

feature/12_ticket_builder

App version: 5.5.1 (1463)

Device: iPhone 6+ OS version: iOS 8.3

Language: English US

Epic Link: [Ticket builder \(Phone\)](#)

Sprint: Sprint 21

People

Assignee:

 Unassigned

[Assign to me](#)

Reporter:

 Jakob Hansen

Votes:

 0 [Vote for this issue](#)

Watchers:

 1 [Start watching this issue](#)

Dates

Created:

20/Apr/15 16:23

Updated:

21/Apr/15 10:27

**All systems have
*essential complexity***

**State also adds
*incidental complexity***

See Moseley and Marks' paper, "Out of the Tar Pit"

state

Your stored values at any given time

State

State of a button:

- **var visible** → 2 states
- **var enabled** → 4 states
- **var selected** → 8 states!
- **var highlighted** → 16 states!!



State is

easy

```
10 INPUT "What is your name: ", U$  
20 PRINT "Hello "; U$  
30 INPUT "How many stars do you want: ", N  
40 S$ = ""  
50 FOR I = 1 TO N  
60 S$ = S$ + "*"  
70 NEXT I  
80 PRINT S$  
90 INPUT "Do you want more stars? ", A$  
100 IF LEN(A$) = 0 THEN GOTO 90  
110 A$ = LEFT$(A$, 1)  
120 IF A$ = "Y" OR A$ = "y" THEN GOTO 30  
130 PRINT "Goodbye "; U$  
140 END
```

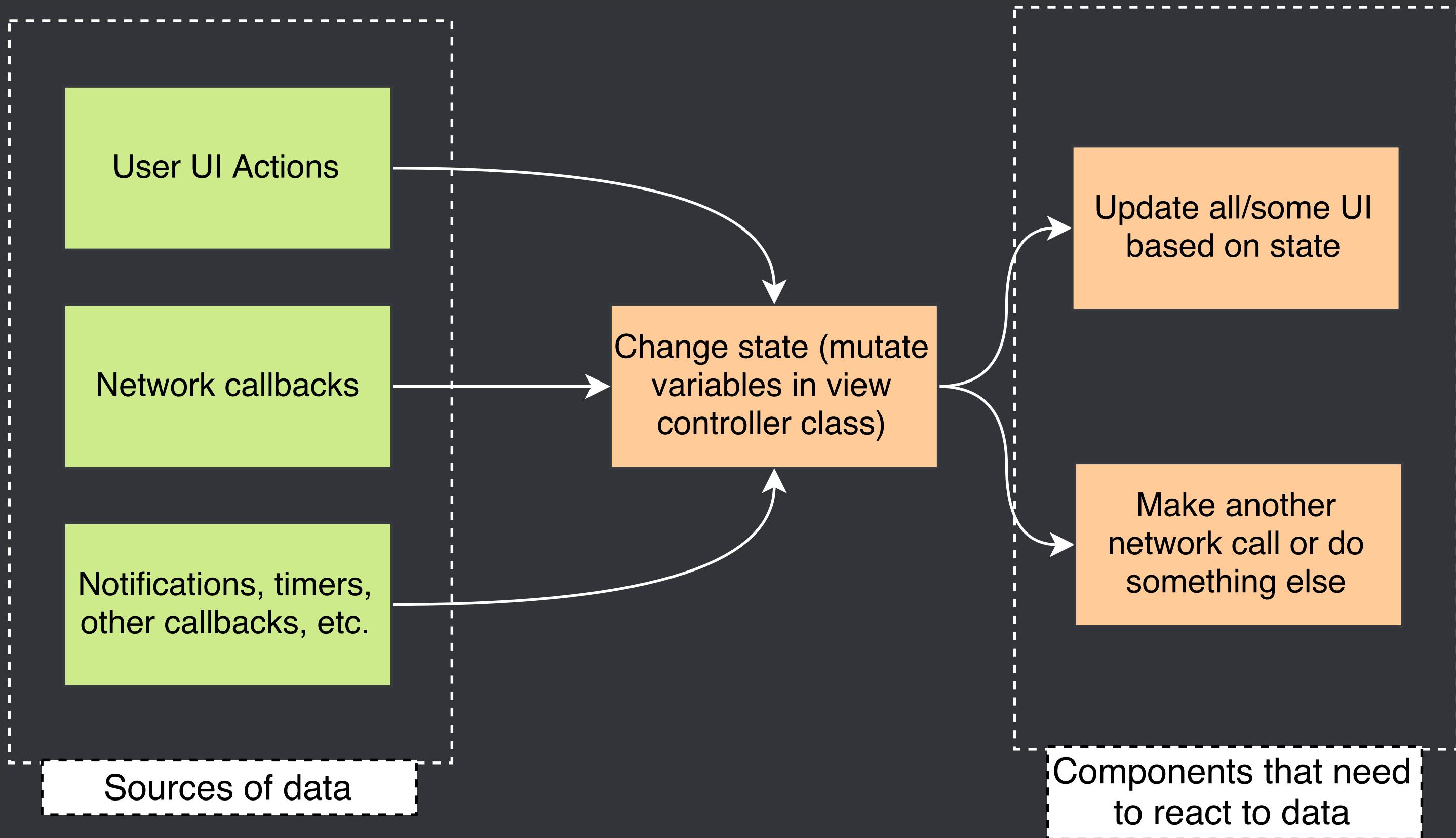
State is unpredictable

The challenge with state is that it actually needs to be handled which becomes a non-trivial problem

- `var visible` → 2 states
- `var enabled` → 4 states
- `var selected` → 8 states!
- `var highlighted` → 16 states!!

16 states need to be handled





GitHub, Inc. github.com/search?o=desc&q=updateui+language% Swift Objective-C

Search updateui language:Swift language:Objective-C

We've found 33,690 code results

Repositories 33,690

Code 33,690

Commits 5,213

Issues 620

Wikis 259

Users

Sort: Recently indexed

timd/ProlosTableCollectionViewsSwift3 – MVVMCell.swift Swift

Showing the top two matches. Last indexed 5 minutes ago.

```
13     var viewModel : Character? {
14         didSet {
15             updateUI()
16         }
17     }
18
19     @IBOutlet weak var nameLabel: UILabel!
...
22     override func awakeFromNib() {
23         super.awakeFromNib()
24         // Initialization code
25         updateUI()
26     }
27
28     override func setSelected(_ selected: Bool, animated: Bool) {
```

rmirza/mindfultr – MainViewController.swift Swift

Showing the top two matches. Last indexed 23 minutes ago.

```
23     private var tap: UITapGestureRecognizer!
24
25     private var lessons: Array<Lesson> = []
26     didSet{
27         updateUI()
...
255         destinationViewController.delegate = self
256     }
```

Advanced search Cheat sheet

Java 3,269
C# 538
JavaScript 507
Objective-C++ 422
C++ 139
JSON 69
Python 16

Advanced search Cheat sheet

```
func updateUI() {  
    label.text = studentName  
    displayRecipes(3)  
}  
  
@IBAction func kannan(sender: UIButton) {  
    studentName = "kannan"  
    updateUI()  
}
```

el2nil/UselessButtons – DetailViewController.swift Swift
Showing the top match. Last indexed on Dec 14, 2016.

```
class DetailViewController: UIViewController {  
    var fillAmount: CGFloat? { didSet { updateUI() } }  
    var labelText: String? { didSet { updateUI() } }  
  
    @IBOutlet private weak var button: CustomButton!  
    @IBOutlet private weak var label: UILabel!
```

griotspeak/2016-11-Examples – ContactViewController.swift Swift
Showing the top three matches. Last indexed on Jan 24.

```
class ContactViewController: UIViewController {  
    var contact: Contact?  
    didSet {  
        updateUI()  
    }  
  
    override func viewWillAppears(animated: Bool) {  
        super.viewWillAppears(animated)  
        updateUI()  
    }  
  
    func updateUI() {
```

Timetable state

```
var segment:TicketBuilderSegment = .Departure ► 2 states  
var direction:TicketBuilderSegment = .Departure ► 4 states  
var departureSorting:FlightSorting = .DepartureSegmentDeparture {} ► 8 states  
var returnSorting:FlightSorting = .ReturnSegmentDeparture {} ► 16 states  
var state:TicketBuilderState ► 32 states
```

More state-like variables with side effects

```
private var selectedDeparturePrice:CurrencyValue?  
private var selectedReturnPrice:CurrencyValue?  
private var selectedPrice:CurrencyValue?  
private var selectedDetailFlight:Flight?  
private(set) var oneway = false  
private var flights:[Flight] { return segment == .Departure ? departureFlights : returnFlights }  
private var sorting:FlightSorting { return segment == .Departure ? departureSorting : returnSorting }  
private var selectedDepartureFlight:Flight? {}  
private var selectedReturnFlight:Flight? {}  
override var result:FlightResult? {}
```



State! State! State! State!

48+ states! Handling:

```
if state == .FirstStep {  
    ...  
}  
  
if state == .SecondStep {  
    if indexPath.row == 0 {  
        ...  
    }  
    if indexPath.row == 1 {  
        if showAllFlights {  
            ...  
        } else {  
  
            if selectedOutboundFlight != nil {  
  
                if picker == .Departure {  
                    ...  
                } else {  
                    ...  
                }  
            state = .FirstStep
```

```
    } else {

        if picker == .Return {
            ...
        } else {
            ...
        }
        state = .SecondStep
    }

    updateUI()
    delegateUpstream?.updatedFilter()
    hideHeaderSortingViewAndFilter()
    updateCheckmarkImage()
    tableView.reloadData()
}

} else {
    ...
}
}

if state == .ThirdStep {
    if indexPath.row == 0 {
        ...
    }
    if indexPath.row == 1 {
        state = .FirstStep
        if picker == .Return {
            ...
        } else {
            ...
        }
        selectedOutboundFlight = nil
        selectedHomeboundFlight = nil
        updateUI()
    }
    if indexPath.row == 2 {
        ...
    }
}
```

Minimize state Minimize complexity



quickmeme.com

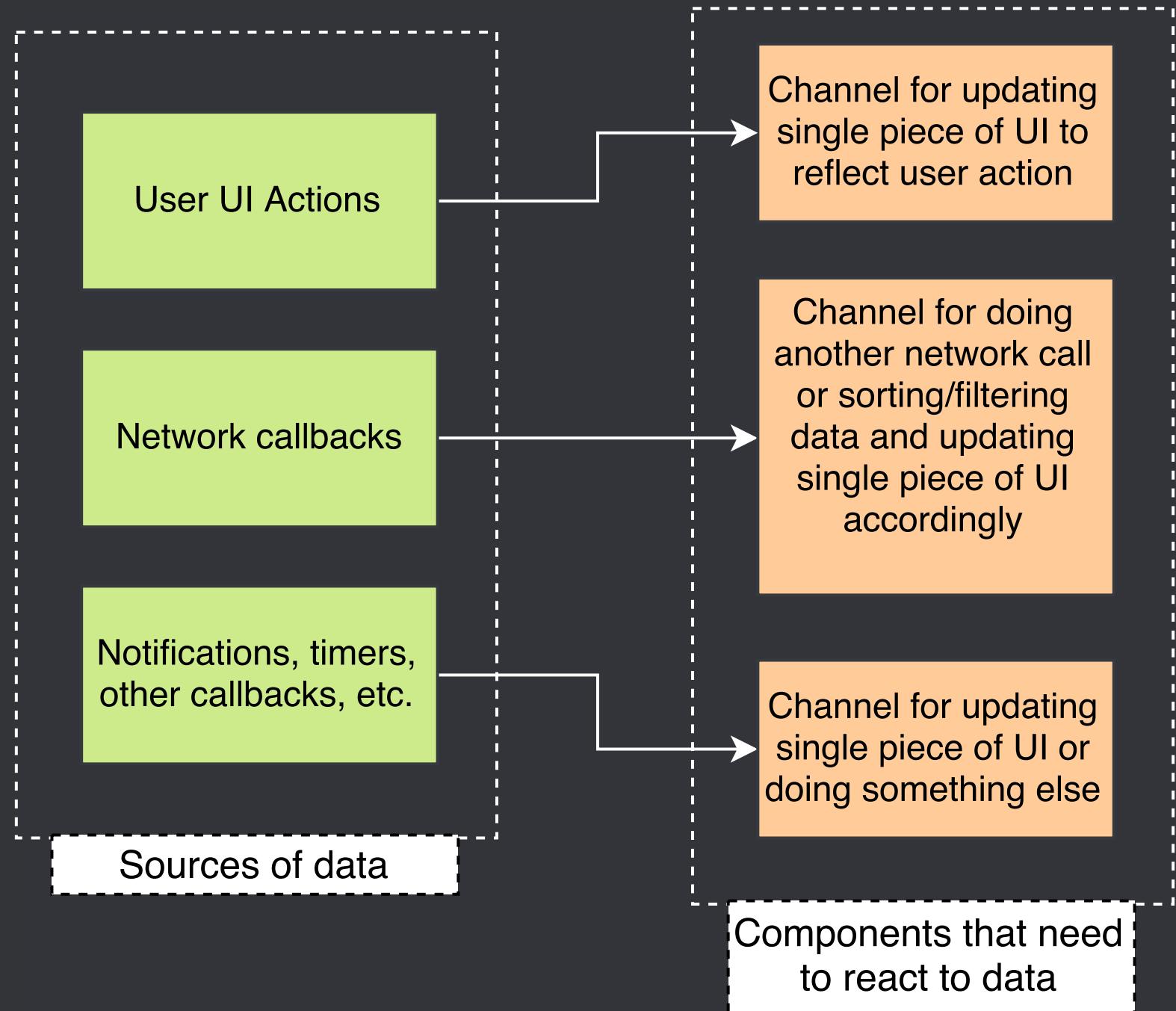
Reactive programming principle

Any “getter” for mutable state causes problems. Instead of using getters, any calculated, generated, loaded or received state values should be immediately sent into a channel and any part of the program that wants access to these values must subscribe to the channel.

— Matt Gallagher

**Reactive programming
manages asynchronous data flows
between sources of data
and components that need to react to that data.**

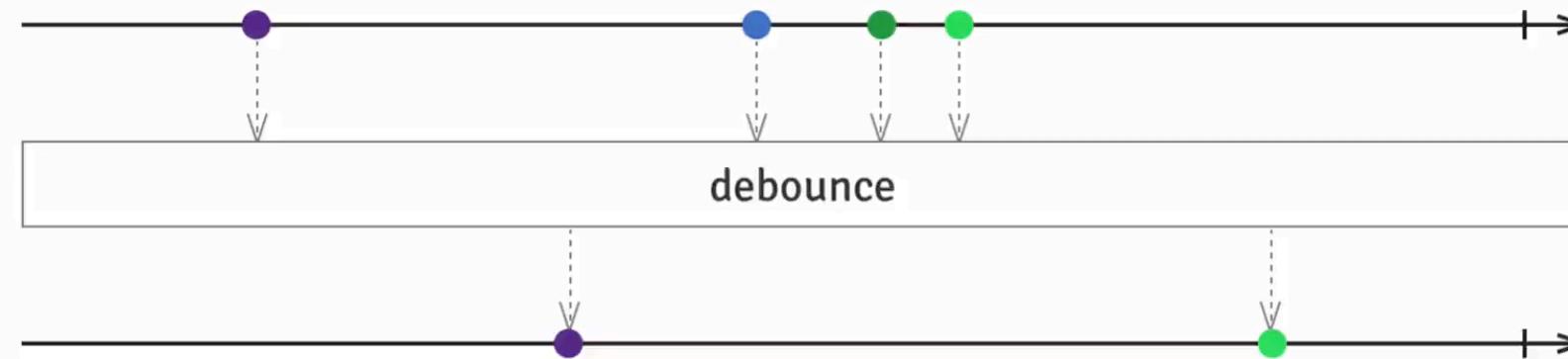
Minimising state and thus reducing complexity



RxSwift

The Observer pattern done right

ReactiveX is a combination of the best ideas from
the **Observer** pattern, the **Iterator** pattern, and **functional programming**



CREATE

Easily create event streams or data streams.

COMBINE

Compose and transform streams with query-like operators.

LISTEN

Subscribe to any observable stream to perform side effects.

Observables are a representation of

any collection of values over
any amount of time

- They can send anything, just like an array can hold anything.
- They're very familiar, and very close to arrays in that they hold stuff.

Differences:

- You can always look into an array and get whatever it has.
- If you're not listening to an observable, then you miss it.

Examples of observables

- Arrays of data
- Network I/O
- UI Events: button taps, any kind of UI interaction
- Notifications
- Delegate callbacks
- KVO, etc.



Everything is a stream

Observables in Rx

Observables are sequences in Rx and described by a push interface

```
enum Event<Element> {
    case Next(Element)           // next element of a sequence
    case Error(ErrorType)        // sequence failed with error
    case Completed               // sequence terminated successfully
}

class Observable<Element> {
    func subscribe(observer: Observer<Element>) -> Disposable
}
```

Observables are lazy

There is one crucial thing to understand about observables.

- When an observable is created, it doesn't perform any work simply because it has been created.
- If you just call a method that returns an **Observable**, no sequence generation is performed, and there are no side effects. Sequence generation starts when **subscribe** method is called

E.g. Let's say you have a method with similar prototype:

```
func searchWikipedia(searchTerm: String) -> Observable<Results> {}

let searchForMe = searchWikipedia("me")

// no requests are performed, no work is being done, no URL requests were fired

let cancel = searchForMe

// sequence generation starts now, URL requests are fired
.subscribeNext { results in
    print(results)
}
```

Transforming Observables

- **buffer**
- **flatMap**
- **map**
- **window**
- ...

Filtering Observables

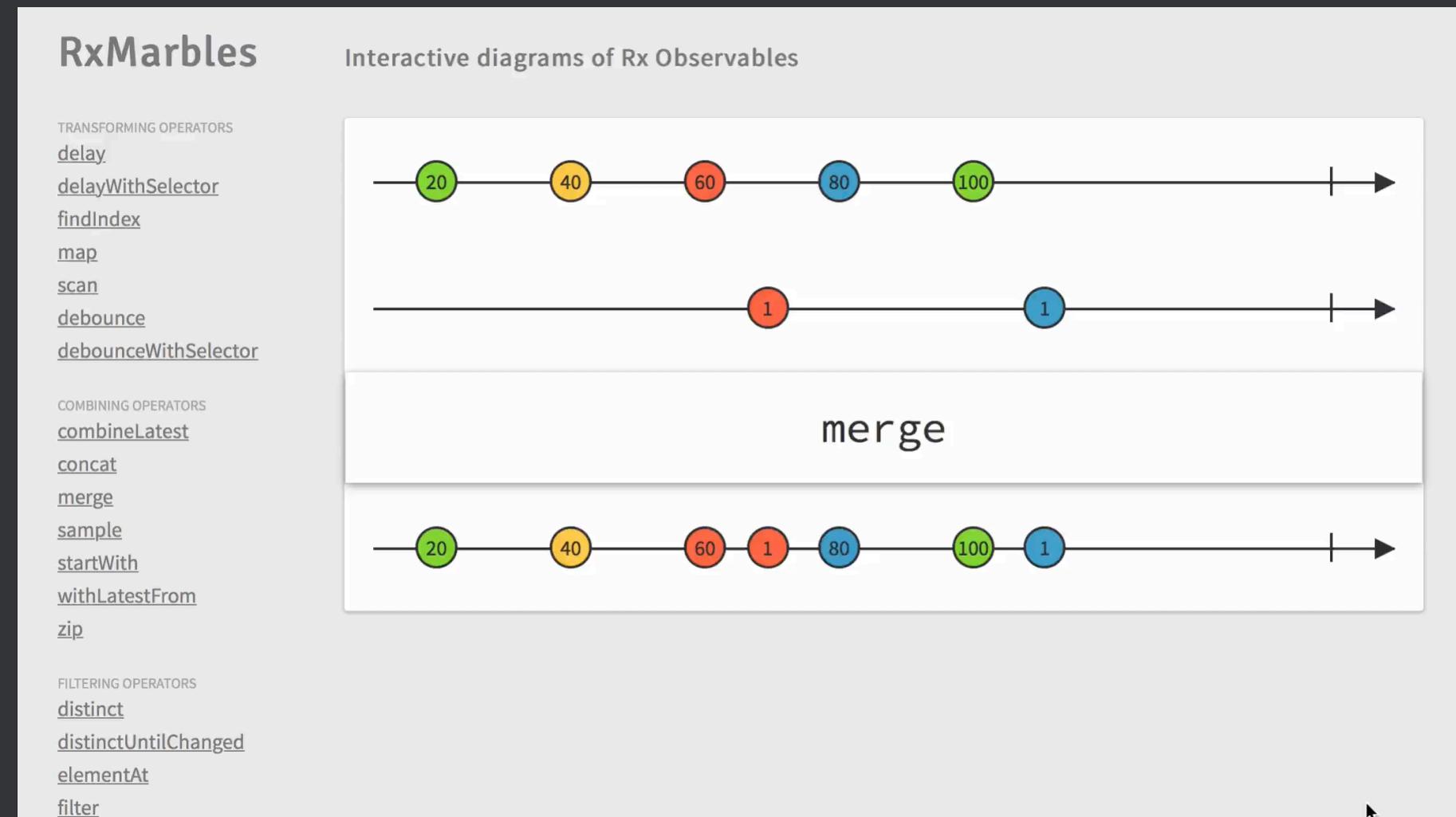
- debounce / throttle
- distinctUntilChanged
- elementAt
- filter
- skip
- ...

Combining Observables

- `merge`
- `combineLatest`
- `zip`
- ...

Learn operators

A website for experimenting with diagrams of Rx Observables, for learning purposes: <http://rxmarbles.com>

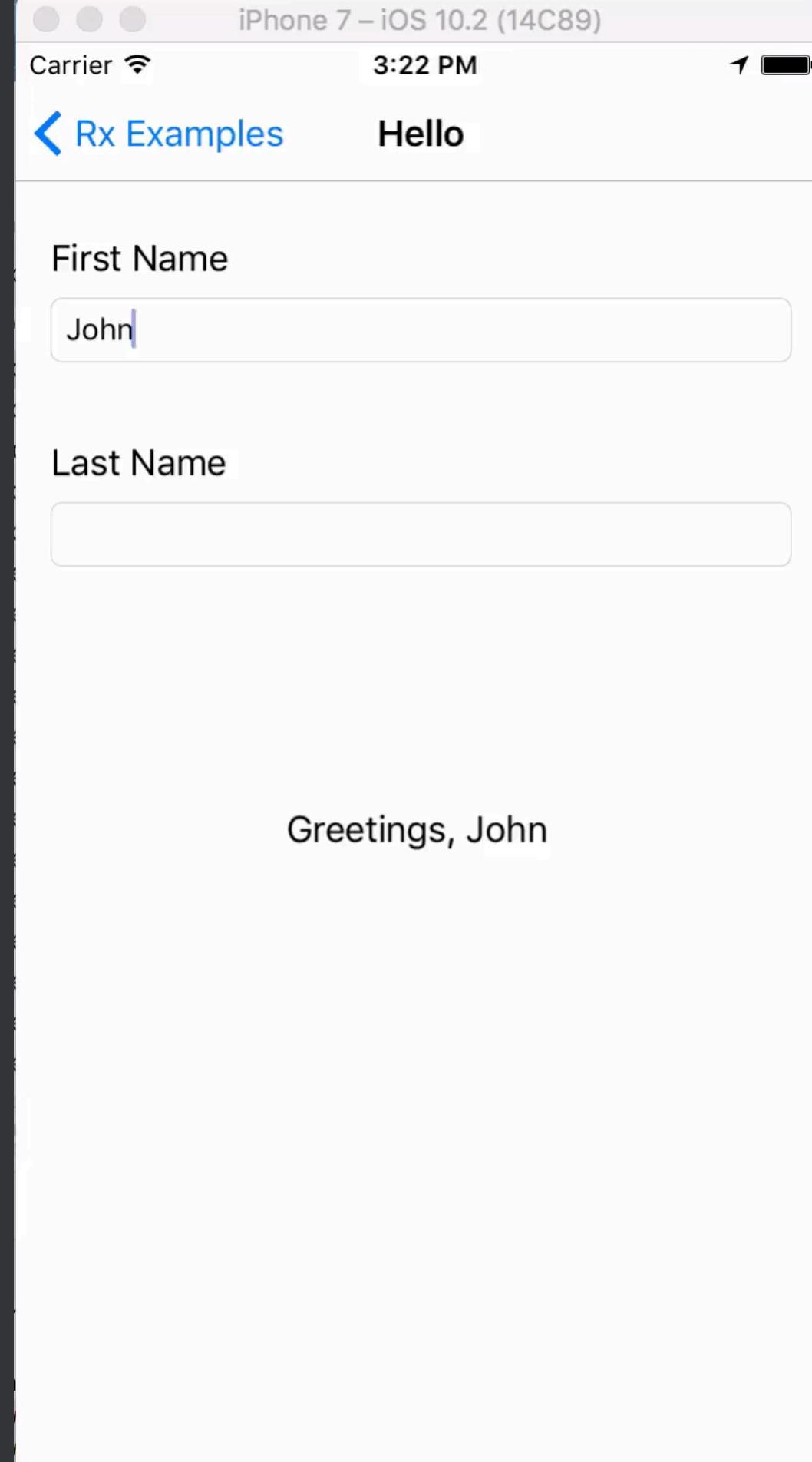


Bindings

```
Observable.combineLatest(firstName.rx.text, lastName.rx.text)
    { $0 + " " + $1 }
    .map { "Greetings, \"\($0)\"" }
    .bindTo(greetingLabel.rx.text)
```

This also works with **UITableViews** and
UICollectionViews.

```
viewModel
    .rows
    .bindTo(tableView.rx.items(cellId: "Cell", cellType: Cell.self))
    { (_, viewModel, cell) in
        cell.title = viewModel.title
        cell.url = viewModel.url
    }
    .disposed(by: disposeBag)
```



Before implementing RxSwift: 15 state-like variables in TimeTable VC and many lines of code to handle those states.

After implementing RxSwift: 3 Observables to handle all possible states

```
let flightsObservable = ReplaySubject<[[TimetableSegment]]>.create(bufferSize: 1)
let tabbarObservable = ReplaySubject<Int>(0).create(bufferSize: 1)
let selectedSegmentObservable = ReplaySubject<(Int, [TimetableSegment?])>.create(bufferSize: 1)
```

- flightsObservable: network request results, sorting and filtering results
- tabbarObservable: tabbar user actions
- selectedSegmentObservable: selecting a particular flight user action

```
//This observable flow will fire up whenever there is a new FlightResult OR tabbar index changed
Observable.combineLatest(flightsObservable.asObservable(), tabbarObservable.asObservable()) {($0.0, $0.1)}
    .map {(segmentedFlights, segmentIndex) -> [TimetableSectionModel] in

        let sections: [TimetableSectionModel] = [
            .flightSection(items: segmentedFlights[segmentIndex])
                .map({ (item) -> TimetableSectionItem in
                    .flightItem(item)
                })])
    }

    return sections
}

.asDriver(onErrorJustReturn: [])
.drive(self.tableView.rx.items(dataSource: self.dataSource))

//This observable flow will fire up whenever there is at least one segment selected
selectedSegmentObservable.asObservable()
    .filter{$0.1.flatMap({$0}).count > 0}
    .subscribe(onNext: { [weak self] _ in
        self?.masterDelegate?.hideFilter()
    }).addDisposableTo(disposeBag)
```

This repository's size is over 1 GB. Learn how to reduce your repository size.

momondo iOS / iOS / SearchOld

Source

release_6.2.1 | Full commit

FlightTicketBuilderViewControllerPhone.swift

Blame | Raw | ▾

1 // FlightTicketBuilderViewControllerPhone.swift
2 // Momondo
3 //
4 // Created by Aron Lindberg on 18/03/15.
5 // Copyright (c) 2015 Momondo. All rights reserved.
6 //
7 import UIKit
8 import MomondoKit
9
10 enum TicketBuilderSegmentType {
11 case Departure
12 case Return
13 }
14
15 enum TicketBuilderState {
16 case FirstStep
17 case SecondStep
18 case ThirdStep
19 }
20
21 class FlightTicketBuilderViewControllerPhone: FlightTicketBuilderViewController, UITableViewDataSource, UITableViewDelegate {
22 let ticketBuilderIntroDisplayController: MMInfoDisplayController = MMInfoDisplayController(topAlignment: true, rightAlign: true)
23
24 enum TabPriceType {
25 case Outbound
26 case Homebound
27 }
28
29 enum HideSortingSeparators: Int {
30 case HideDeparture = 3
31 case HideArrival = 4
32 case HideTime = 5
33 case HideMoney = 6
34 }
35
36 @IBOutlet weak var homeboundLabel: UILabel!
37 @IBOutlet weak var outboundLabel: UILabel!
38 @IBOutlet weak var tabSelectorConstraint: NSLayoutConstraint!
39 @IBOutlet weak var tabSelectorView: UIView!
40 @IBOutlet weak var tableView: UITableView!
41 @IBOutlet weak var departureSortingButton: UIButton!
42 @IBOutlet weak var arrivalSortingButton: UIButton!
43 @IBOutlet weak var timeSortingButton: UIButton!
44 @IBOutlet weak var cheapestSortingButton: UIButton!
45 @IBOutlet weak var sortingButtons: [UIButton]!
46 @IBOutlet weak var homeboundCheckmarkImage: UIImageView!
47 @IBOutlet weak var outboundCheckmarkImage: UIImageView!
48 @IBOutlet var headerView: UIView!
49 @IBOutlet weak var departureSeparatorImage: UIImageView!
50 @IBOutlet weak var arrivalSeparatorImage: UIImageView!
51 @IBOutlet weak var timeSeparatorImage: UIImageView!
52 @IBOutlet weak var homeboundButton: UIButton!
53 @IBOutlet weak var boundsViewContainer: UIView!
54 @IBOutlet weak var tableViewLeftConstraint: NSLayoutConstraint!
55 @IBOutlet weak var tableViewRightConstraint: NSLayoutConstraint!
56
57 private(set) var picker: TicketBuilderSegmentType = .Departure
58 private(set) var direction: TicketBuilderSegmentType = .Departure
59 private(set) var oneWay = false
60 private var selectedOutboundFlight: Flight?
61 private var selectedHomeboundFlight: Flight?
62 private var detailsSelectedFlight: Flight?
63 private var selectedFlight: Flight? {
64 if let selectedOutboundFlight = selectedOutboundFlight {
65 return selectedOutboundFlight
66 } else if let selectedHomeboundFlight = selectedHomeboundFlight {
67 return selectedHomeboundFlight
68 } else {
69 return nil
70 }
71 }
72
73
74 if(firstHomeboundButtonTap) {
75 firstHomeboundButtonTap = false
76 MMUsageTracking.instance().trackEvent(MMMGAIEventFlightSearchResultsPage_First_switch_of_sorting_panes_result)
77 }
78 updateTabbar(true, changePickerType:true)
79 }
80
81 // MARK: Sorting action buttons
82 @IBAction func departureSortingTapped(sender: UIButton) {
83 MMUsageTracking.instance().trackEvent(MMMGAIEventFlightSearchResultsPage_Use_the_filters_by_departure_timetable_results)
84 updateSortingButtons(sender.tag)
85 }
86
87 @IBAction func arrivalSortingTapped(sender: UIButton) {
88 MMUsageTracking.instance().trackEvent(MMMGAIEventFlightSearchResultsPage_Use_the_filters_by_arrival_timetable_results)
89 updateSortingButtons(sender.tag)
90 }
91
92 @IBAction func timeSortingTapped(sender: UIButton) {
93 MMUsageTracking.instance().trackEvent(MMMGAIEventFlightSearchResultsPage_Use_the_filters_by_time_timetable_results)
94 updateSortingButtons(sender.tag)
95 }
96
97 @IBAction func cheapestSortingTapped(sender: UIButton) {
98 MMUsageTracking.instance().trackEvent(MMMGAIEventFlightSearchResultsPage_Use_the_filters_by_price_timetable_results)
99 updateSortingButtons(sender.tag)
100 }
101 }

811 100

Blog | Support | Plans & pricing | Documentation | API | Site status | Version info | Terms of service | Privacy policy

JIRA Software | Confluence | Bamboo | SourceTree | HipChat

Atlassian

Timetable view controller before changes

- 811 lines of code
- 15 state-like variables with unclear side-effects
- Missing handling of some states
- 61 bugs reported by QA - most relate to poor state handling
- Confusing code, nobody has clear picture of what's going on
- Tight coupling between components in view controller
- Hard to extend functionality

Bitbucket Teams Projects Repositories Snippets Find a repository...

momondo iOS / iOS / Search

Source

Search / View Management / Flights Ticket Builder / FlightTimeTableViewController.swift

Source Diff History

94bafdb 2017-01-11 Full commit Blame Raw

```
// FlightTimeTableViewController.swift
// Momondo
//
// Created by Dennis Torbichuk on 11/09/16.
// Copyright (c) 2015 Momondo. All rights reserved.

import UIKit
import Flights
import MonondoKit
import Slash
import RxSwift
import RxDataSources
import RxCocoa

class FlightTimeTableViewController: UIViewController, FlightResultsChildProtocol, UITableViewDelegate, MonondoAnimatedTa
{
    let disposeBag = DisposeBag()
    let dataSource = RxTableViewSectionedReloadDataSource<TimetableSectionModel>()

    @IBOutlet weak var navigationBar: MonondoNavigationBar!
    @IBOutlet weak var segmentControl: MonondoSegmentedControl!
    @IBOutlet weak var tableView: UITableView!
    @IBOutlet fileprivate weak var flightDetailsButton: UIButton!
    @IBOutlet fileprivate var tabBar: MonondoAnimatedTabBar!

    @IBOutlet weak var tableViewLeftConstraint: NSLayoutConstraint!
    @IBOutlet weak var tableViewRightConstraint: NSLayoutConstraint!

    var tabBarSegmentViews:[FlightResultsTimeTableSegmentView] = []
    var manager: FlightManager!
    weak var masterDelegate: FlightResultsMasterProtocol?
    func newSearchStarted() {}
    func searchFinished() {}

    fileprivate var flightsObservable = ReplaySubject<[TimetableSegment]>.create(bufferSize: 1)
    fileprivate var tabBarObservable = Variable<Int>()
    fileprivate var selectedSegmentObservable = ReplaySubject<Int, [TimetableSegment?>>.create(bufferSize: 1)
    fileprivate var selectedTrackingIndex: Int?
    var selectedIndex: Int = 0

    var result: FlightResult? {
        didSet {
            if let _ = manager, let result = result {
                if let selectedTrackingIndex = selectedTrackingIndex {
                    let count = result.segmentedFlights[selectedTrackingIndex].count
                    let feature = GMEventLogger.feature(resultActive: false, parameters: manager.searchParameters)
                    let label = "button | flight \(selectedTrackingIndex+1)"
                    GMEventLogger.currentLogger().resultEvent(.FlightSearchTimeTableSelectSegment, numberOfFlights: count)
                }
                flightsObservable.onNext(result.segmentedFlights)
            }
        }
    }

    // MARK: General functions

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        updateTabbarSegmentViews()
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        updateTabbarSegmentViews()

        flightDetailsButton.isHidden = true
        tabBar.delegate = self
    }
}
```

371 }
372 await original: TiptableSectionModel.items: [Item] {
373 if (lf === origin)
374 }
375 , 376 LOC)

Timetable view controller after changes

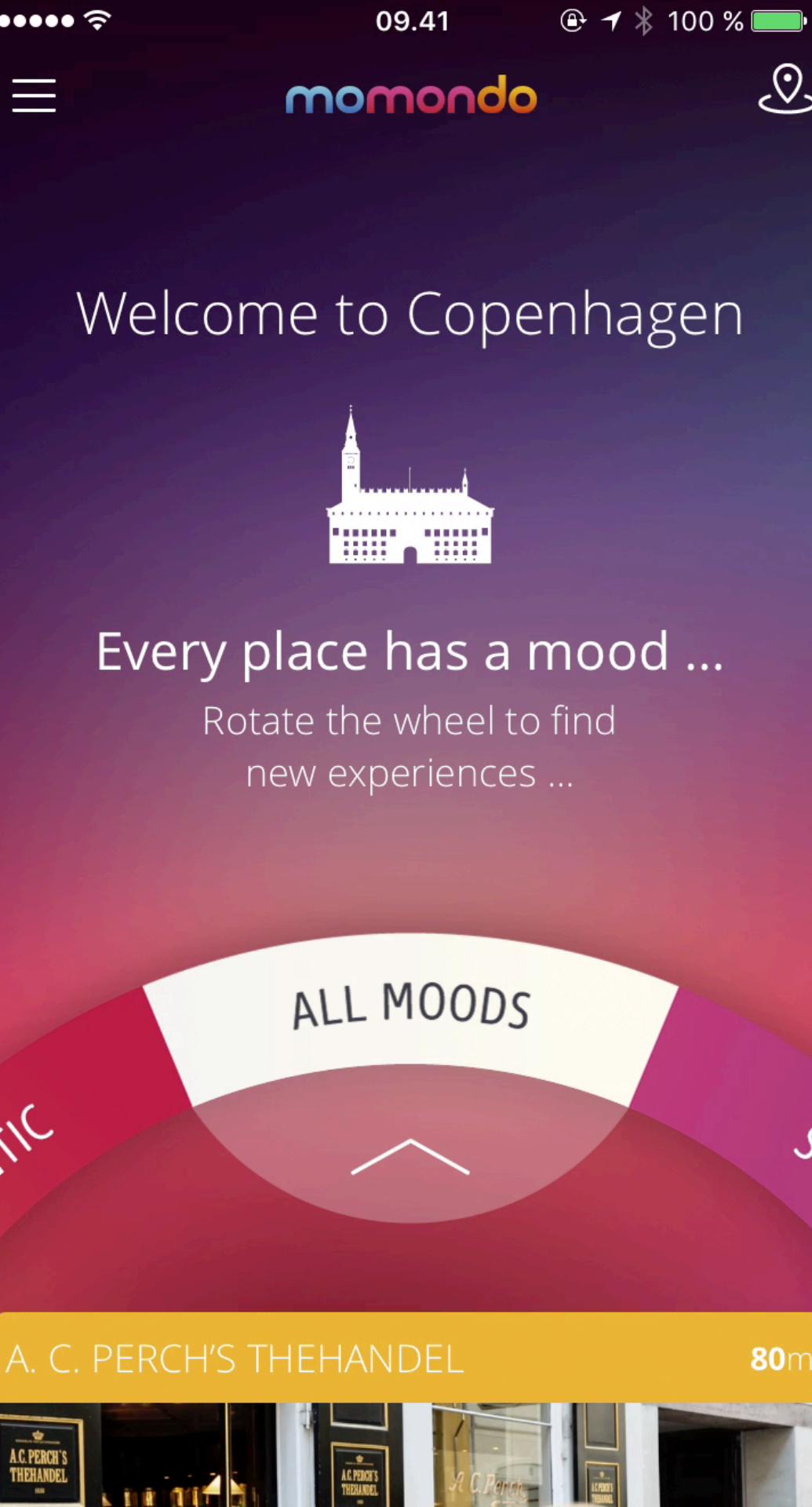
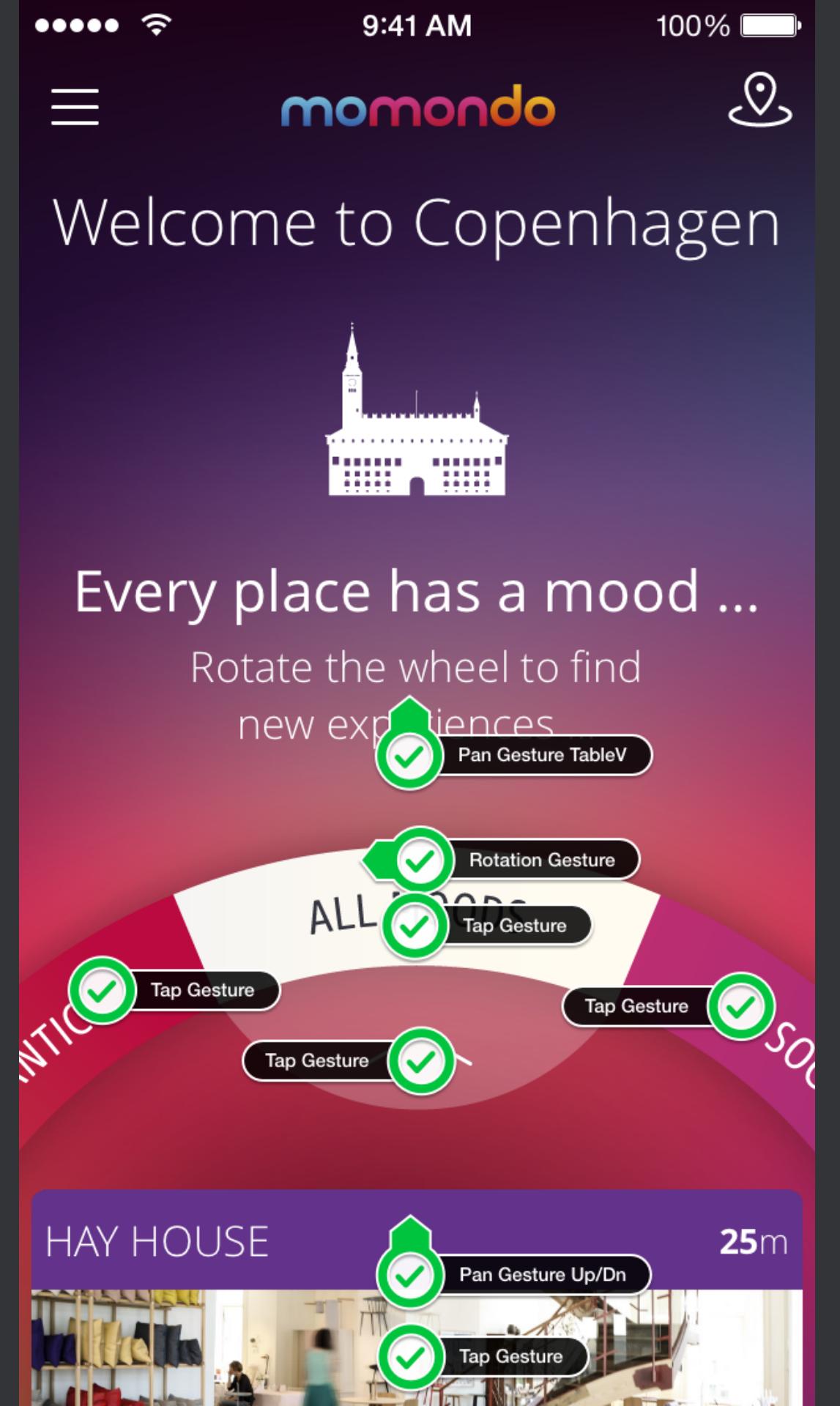
- 376 lines of code: 2.15x less code
 - 0 state-like variables, instead clearly defined channels or flows
 - Very few bugs reported by QA after extending functionality to support up-to 4 flights
 - Concise code that clearly demonstrates relationship between sources of data and components that need to react to changes of this data
 - No coupling between components in view controller
 - Other features implemented using Rx from the beginning, like new destination

Before following reactive programming principles

- Brief discussion with peers on technical implementation
- Developer goes into crunch mode adding UI components one by one, adding data sources, state variables and connecting everything together

After we started following reactive programming principles

- Team discussion to identify all possible inputs and components that need to react to changes
- Identify and mockup channels or flows which consist of sources of data, observable chains and subscribers that react to changes.
- Discuss the flows with the team/QA to ensure everything is covered
- Implement flows





Welcome to Copenhagen



Every place has a mood ...

Rotate the wheel to find
new experiences ...



HAY HOUSE

25m



What happens when you drag the wheel up?

1. Wheel moves up
2. Text on the wheel fades out
3. Wheel glow fades out
4. Wheel scales down
5. Wheel tab fades out
6. Introduction text moves up proportionally
7. Introduction text fades out
8. Onboarding arrow fades out
9. Logo fades out
10. Background image fades out
11. List of cards moves up
12. Specific gesture recognisers become enabled/disabled based on the wheel drag progress
13. Flights banner moves up (closes)

How all of these things placed on 4 different view controllers are kept in sync and move together?

Conclusion: Benefits of using Reactive Programming

In short, using Rx will make your code:

- Reusable <- Because it's composable
- Declarative <- Because definitions are immutable and only data changes
- Understandable and concise <- Raising the level of abstraction and removing transient states
- Less stateful <- Because of unidirectional data flows

Cons

- Steep learning curve
- Requires changing the way you think
- Requires mature development team
- Forces one to think more and code less
- Introducing 3rd party dependency
- Some operators and concepts can be confusing in the beginning.



References

Here are some links to material referenced in the talk:

- RxSwift and ReactiveX
- What is reactive programming and why should I use it? by Matt Gallagher
- Out of the Tar Pit by Moseley and Marks
- Mutability, aliasing, and the caches you didn't know you had by Andy Matuschak
- Enemy of the state by Justin Spahr-Summers
- "Advanced iOS Application Architecture and Patterns" from WWDC 2014 by Andy Matuschak and Colin Barrett

**Thanks for not falling
asleep. Questions?**