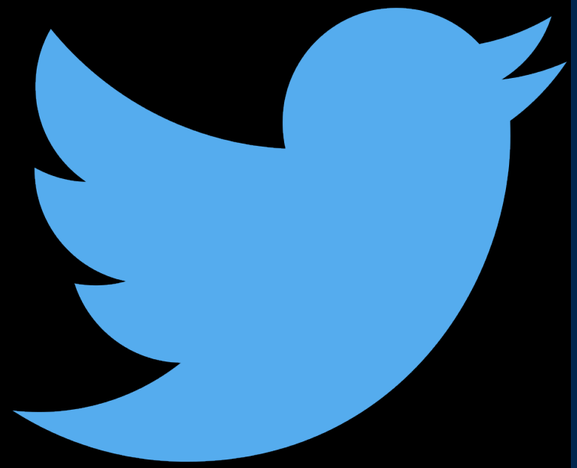
Building Fabric.app with ReactiveCocoa

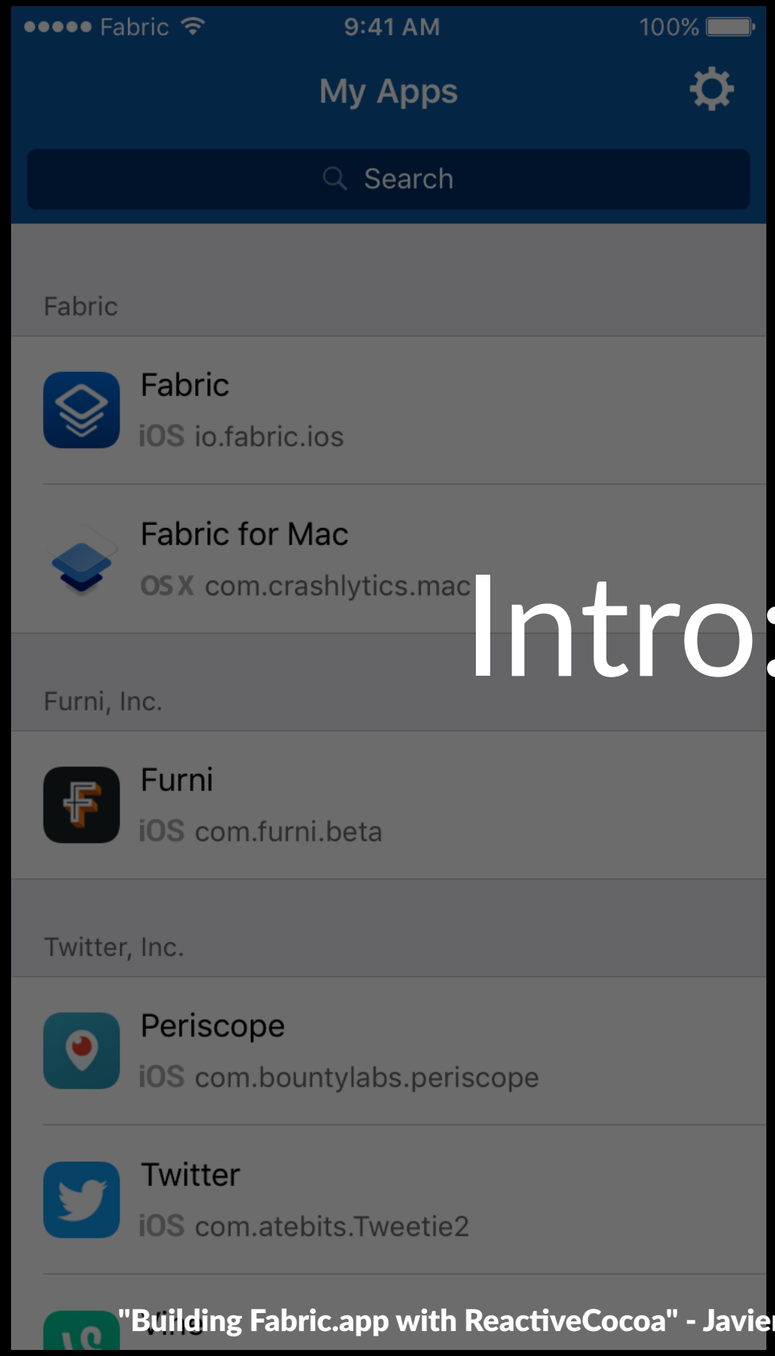
Intro

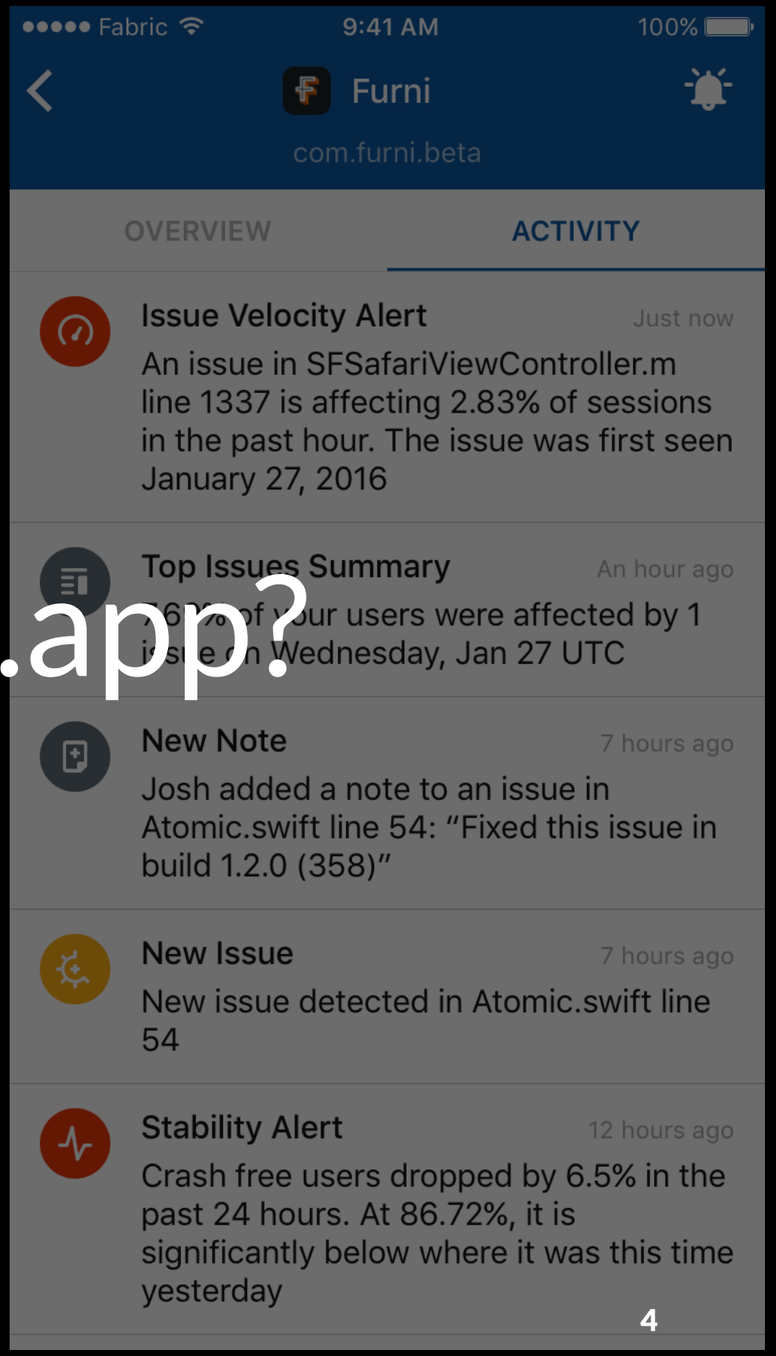


@Javi

Outline

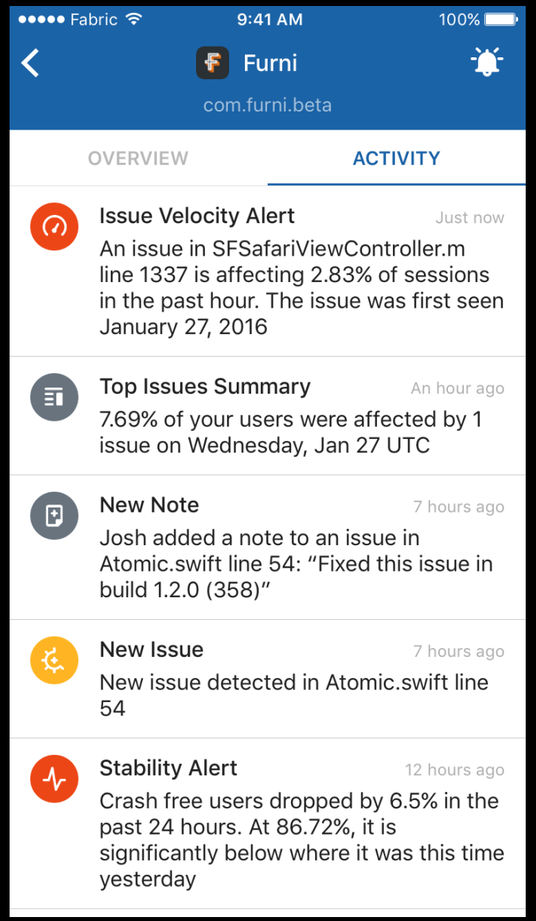
- Intro
- History
- Contributing to ReactiveCocoa
- Fabric App Architecture
- RAC examples from Fabric App



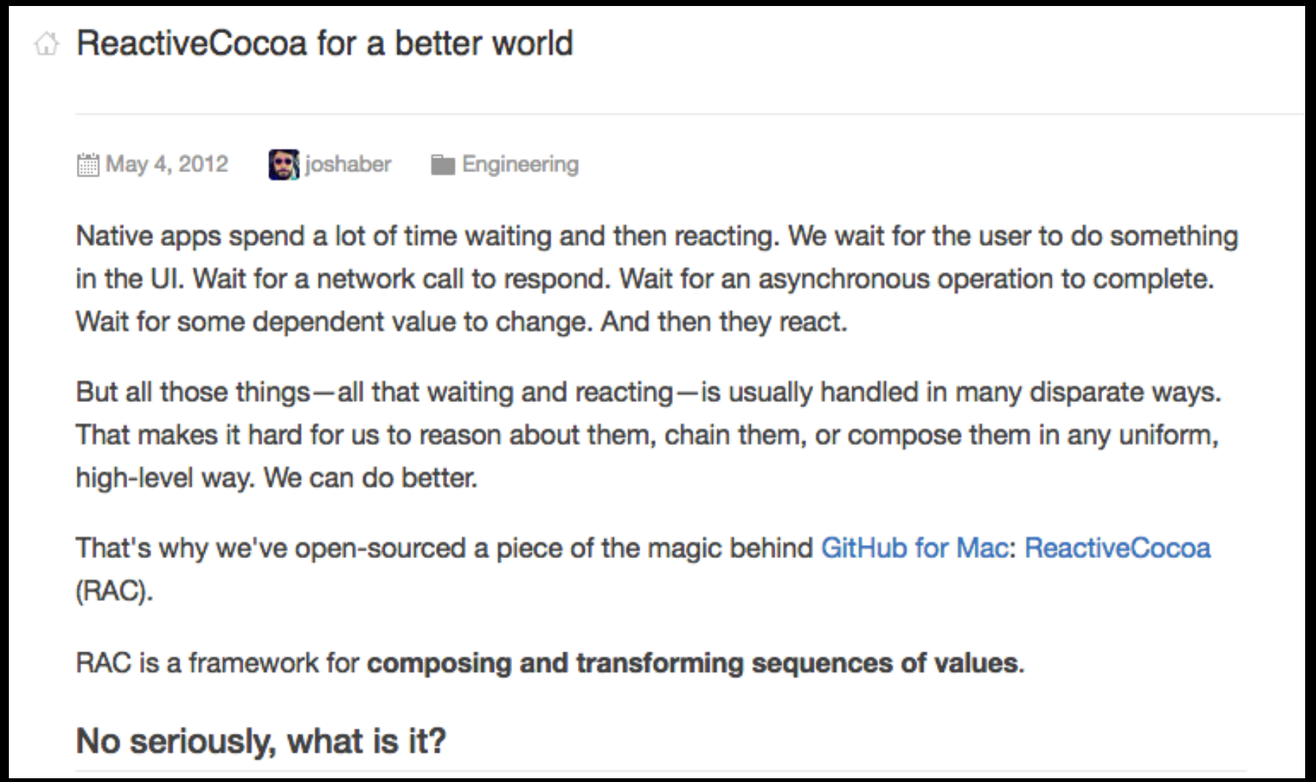


Intro: What is Fabric.app?

What is Fabric.app?



History



ReactiveCocoa for a better world

May 4, 2012 joshaber Engineering

Native apps spend a lot of time waiting and then reacting. We wait for the user to do something in the UI. Wait for a network call to respond. Wait for an asynchronous operation to complete. Wait for some dependent value to change. And then they react.

But all those things—all that waiting and reacting—is usually handled in many disparate ways. That makes it hard for us to reason about them, chain them, or compose them in any uniform, high-level way. We can do better.

That's why we've open-sourced a piece of the magic behind GitHub for Mac: ReactiveCocoa (RAC).

RAC is a framework for **composing and transforming sequences of values**.

No seriously, what is it?

https://github.com/blog/1107-reactivecocoa-for-a-better-world

History

- ReactiveCocoa 1: May 2012 (Objective-C)
- ReactiveCocoa 2: September 2013 (Objective-C)
- ReactiveCocoa 3: September 2015 (Swift 1)
- ReactiveCocoa 4: January 2016 (Swift 2)


3 3G
14:19
97%
MY PEBBLE
Smartwatch+
Reminders
HTTP Request
Music
foursquare
Steven's Philly Cheesesteaks
1121 Tasman Dr
Sunnyvale, CA
5 / 8 SLOTS USED
LOCKER
NOT SURE IF IT'S
13:50:30
OR
Fri 01-31-2014
PHILOSORAP...
REVOLUTION

Contributing to ReactiveCocoa

Contributing to ReactiveCocoa: Coding

- Refactoring
- New tests
- Proposing new operators
- Compatibility with new Swift versions

Contributing to ReactiveCocoa: Other ways!

- Replying to issues
- Writing / improving docs
- Helping other users
- Evangelizing

Fabric App Architecture

Fabric App Architecture

- FabricAPI.framework:
  - Networking
  - Models
- Fabric App:
  - View Controllers
  - View Models

Examples of Usage of ReactiveCocoa in the Fabric App

Examples - Networking

```
final class AuthenticatedFabricAPI {
    func applications() -> SignalProducer<[Application], FabricAPIError> {
        return apiNetworking.requestJSONProducer(
                URL: APIURL(path: "api/v3/projects"),
                method: .GET
            )
            .attemptMap(Application.decodeObjectsInJSON)
            .observeOn(UIScheduler())
    }
}
```

DataLoadState

```
final class MyViewModel {
    var data: MyEntity?
}
```

DataLoadState

```
enum DataLoadState<DataType> {
    case Loading
    case Failed
    case Loaded(DataType)
}
```

DataLoadState

```
enum DataLoadState<DataType> {
    case Loading
    case Failed
    case Loaded(DataType)
}

extension SignalProducerType {
    func materializeToLoadState() -> SignalProducer<DataLoadState<Value>, NoError> {
        let producer = self
            .map(DataLoadState.Loaded)
            .startWithValue(DataLoadState.Loading)

        return producer.ignoreErrors(replacementValue: DataLoadState<Value>.Failed)
    }
}
```

```swift
extension SignalProducerType where Value: DataLoadState {
    func ignoreLoadingAndErrorsAfterSuccess() -> SignalProducer<DataLoadState<Value.DataType>, Error> {
        var hasSuccededOnce = false

        return self.filter { value in
            defer {
                if value.success {
                    hasSuccededOnce = true
                }
            }

            return !hasSuccededOnce || value.success
        }
}
```

Examples - View Models

```
typealias ApplicationLoadState = DataLoadState<[Application]>

final class ApplicationListViewModel {
    let applications: AnyProperty<ApplicationLoadState>
    private let applicationsMutableProperty = MutableProperty(ApplicationLoadState.loading())

    init(fabricAPI: AuthenticatedFabricAPI) {
        self.applications = AnyProperty(self.applicationsMutableProperty)

        self.applicationsMutableProperty <~ fabricAPI.applications().materializeToLoadState()
    }
}
```

Examples - Consuming Data From a View Model

```
self.viewModel.applications.producer.startWithNext { applicationsLoadState in
    switch applicationsLoadState {
        case .Loading:
            label.text = "Loading..."

        case .Failed:
            label.text = "Error loading applications!"

        case .Loaded(let applications):
            reloadTableView(applications: applications)
    }
}
```

ReactiveCocoa Extensions

```swift
extension SignalProducerType {
    func startWithValue(value: Value) -> SignalProducer<Value, Error> {
        return SignalProducer(value: value).concat(self.producer)
    }
}
```

ReactiveCocoa Extensions

```
extension SignalProducerType {
    func startWithNil() -> SignalProducer<Value?, Error> {
        return self
            .map(Optional.init)
            .startWithValue(nil)
}
```

ReactiveCocoa Extensions

```
extension SignalProducerType {
    func ignoreErrors(
        replacementValue replacementValue: Self.Value? = nil
    ) -> SignalProducer<Value, NoError> {
        return self.flatMapError { _ in
            return replacementValue.map(SignalProducer.init) ?? .empty
        }
    }
```

ReactiveCocoa Extensions

```
extension SignalProducerType {
    func failRandomly(withError error: Self.Error) -> SignalProducer<Value, Error> {
        return self.attemptMap { value in
            let shouldFail = arc4random() % 3 == 0

            return shouldFail ? Result(error: error) : Result(value: value)
    }
}
```

ReactiveCocoa Extensions

```
extension NSProcessInfo {
    var lowPowerModelEnabledProducer: SignalProducer<Bool, NoError> {
        return NSNotificationCenter.defaultCenter()
            .rac_notifications(NSProcessInfoPowerStateDidChangeNotification, object: nil)
            .map { _ in return () }
            .startWithValue(())
            .map { [unowned self] _ in
                return self.lowPowerModeEnabled
            }
            .skipRepeats(==)
    }
}
```

ReactiveCocoa Extensions

```
let shouldReload = combineLatest(
    viewIsOnScreen,
    NSProcessInfo.processInfo().lowPowerModelEnabledProducer.map { !$0 }
).map { $0 && $1 }

let reloadPeriodically = shouldReload
    .flatMap(.Latest) { [unowned self] shouldReload in
            return shouldReload ?
                timer(30, onScheduler: scheduler).map { _ in () }
                : .empty
    }

let request = reloadPeriodically.flatMap(.Latest) { someAPIRequest }
```

ReactiveCocoa Extensions

```
extension SignalProducerType {
    func continueWhenApplicationIsBackgrounded(
        taskName taskName: String,
        timeoutError: Self.Error
    ) -> SignalProducer<Value, Error> {
}
```

ReactiveCocoa Extensions

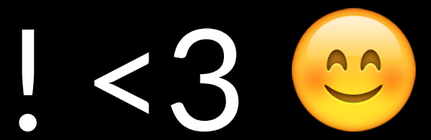
```
extension SignalProducerType {
    func repeatWith(
        producer: SignalProducer<(), NoError>,
        throttleWithInterval: NSTimeInterval,
        onScheduler scheduler: DateSchedulerType
    ) -> SignalProducer<Value, Error> {
        return SignalProducer(value: ()).concat(producer)
            .throttle(throttleWithInterval, onScheduler: scheduler)
            .promoteErrors(Error)
            .flatMap(.Concat) { _ in
                return self.producer
            }
    }
}
```

ReactiveCocoa Extensions

```
self.api.request(parameterFoo: bar)
    .repeatWith(
        viewWillAppearProducer,
        throttleWithInterval: 60,
        onScheduler: QueueScheduler.mainQueueScheduler
    )
```

Conclusion

Questions?

Thank you! <3 

See you next year!