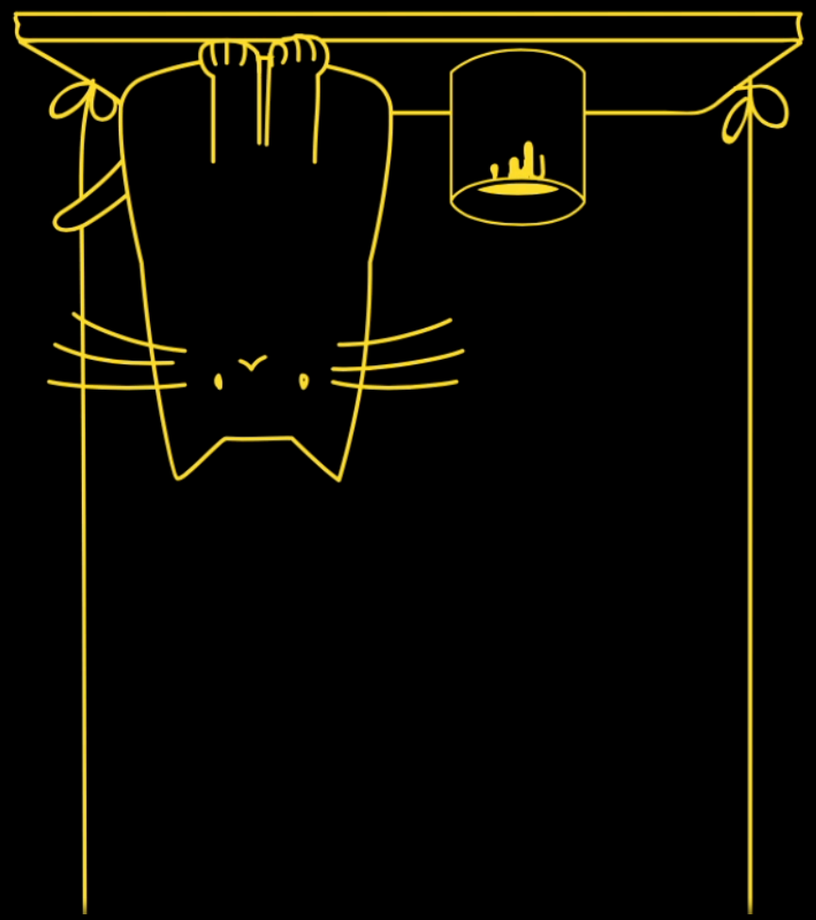


Моделирование состояний экранов

БИБЛИОТЕКА ПРОГРАММИСТА

Дж. Рамбо, М. Блаха

UML 2.0

Объектно-ориентированное моделирование и разработка

2-е издание

- Объектно-ориентированные концепции
- Анализ и проектирование объектно-ориентированной модели
- Реализация проектных решений
- Технологии разработки программного обеспечения

PEARSON Prentice Hall

ПИТЕР

О нас

- ~ 150 экранов
- Аудитория несколько млн
- 1-е место по оценке Роскачества
- Swift 5 / RxSwift / RxDataSources / Moya
- Модульный проект из 14 частей
- 6 разработчиков

Для кого?

- Не получается заранее всё продумать, поэтому часто приходится что-нибудь «допиливать». Хочется более четких и продуманных требований

- Нескончаемые баги после завершения разработки

- Хочется стандартизировать и упростить работу с VIPER / VIP / RIB модулями

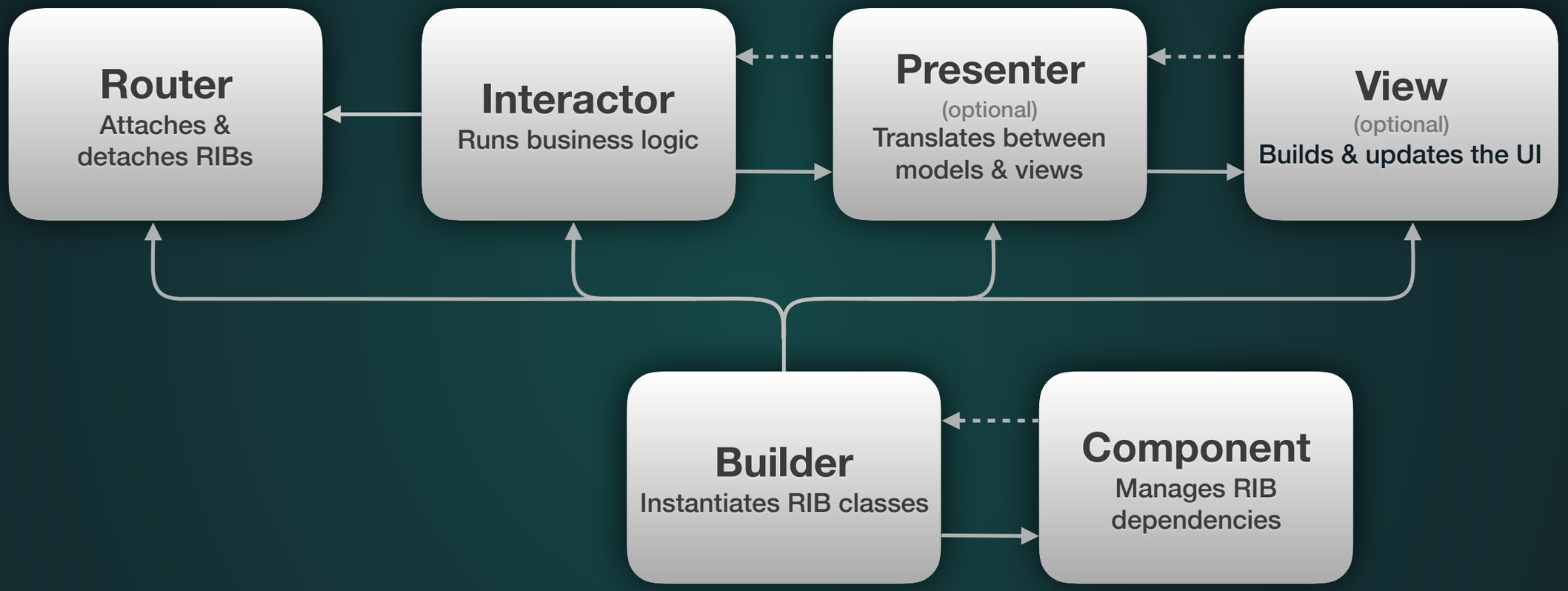
Требования

- Наличие обоснованной необходимости, декомпозированной по пунктам.

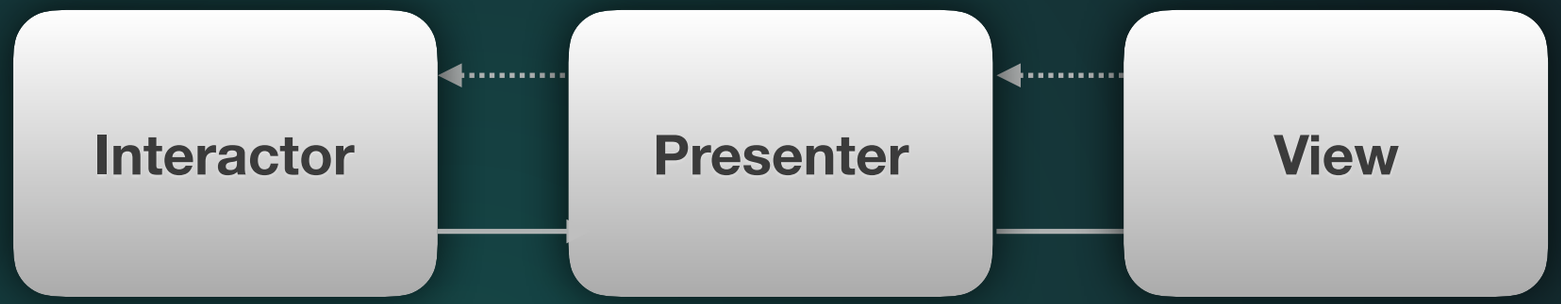
- Понимание азов RxSwift: Observable, PublishRelay, BehaviourRelay, Signal, Driver, ControlEvent, Disposable. Оперыторы: map, compactMap, filter, withLatestFrom, combineLatest, share.

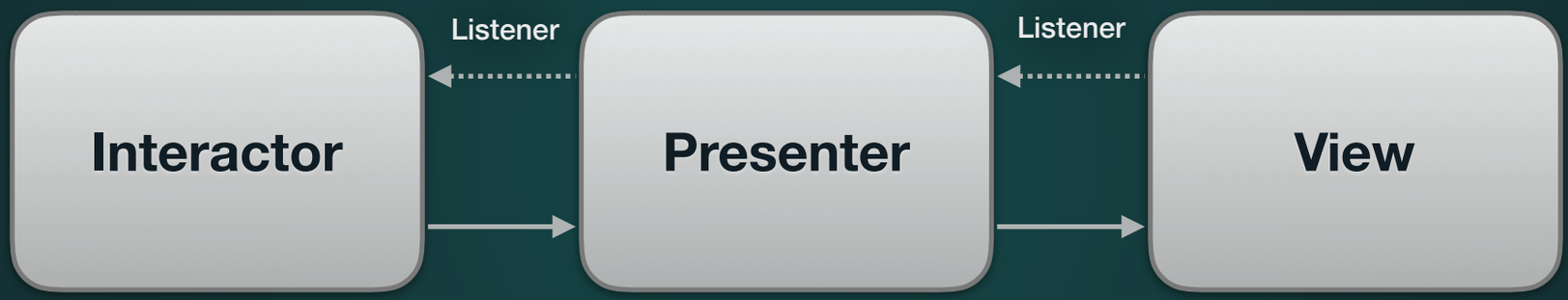
- Понимание вашего целевого архитектурного паттерна.

RIBs


Router
Attaches & detaches RIBs
Interactor
Runs business logic
Presenter
(optional)
Translates between models & views
View
(optional)
Builds & updates the UI
Builder
Instantiates RIB classes
Component
Manages RIB dependencies

Uber

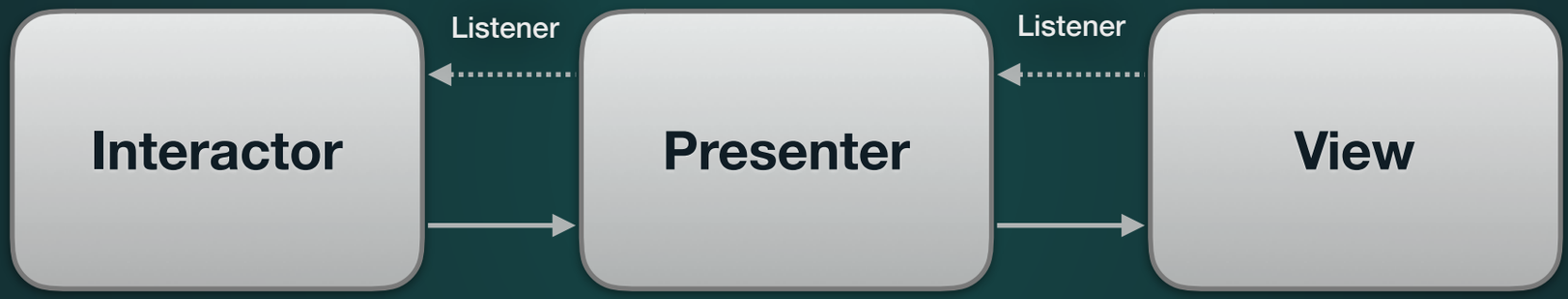

Interactor
Presenter
View


Listener
Listener
Interactor
Presenter
View

```
listener?.showHistoryLoading()
listener?.hideHistoryFailedToLoadZero()
```

```
loginField.isHidden = listener?.contacts.phone == nil
emailField.isHidden = listener?.contacts.email == nil
```

```
let phone = listener?.userPhone ?? ""
```



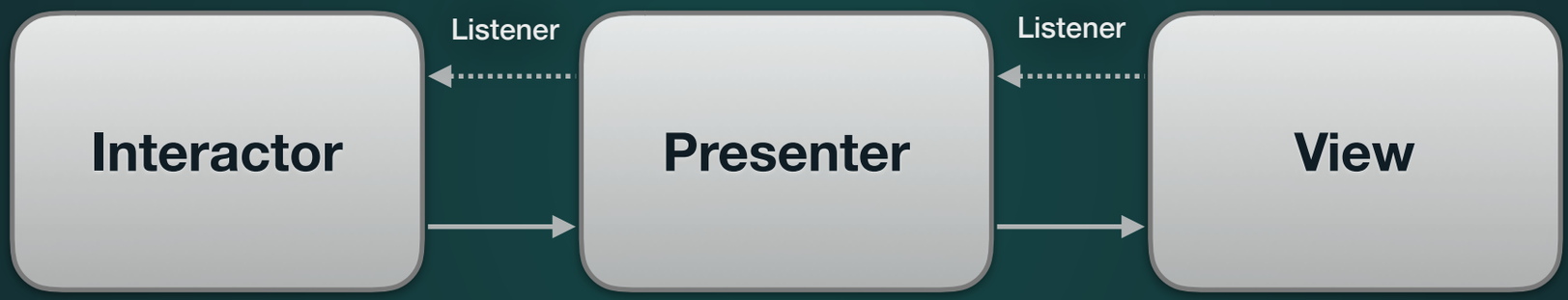

```
if let login = listener?.login {
    loginField.text = login
}
```

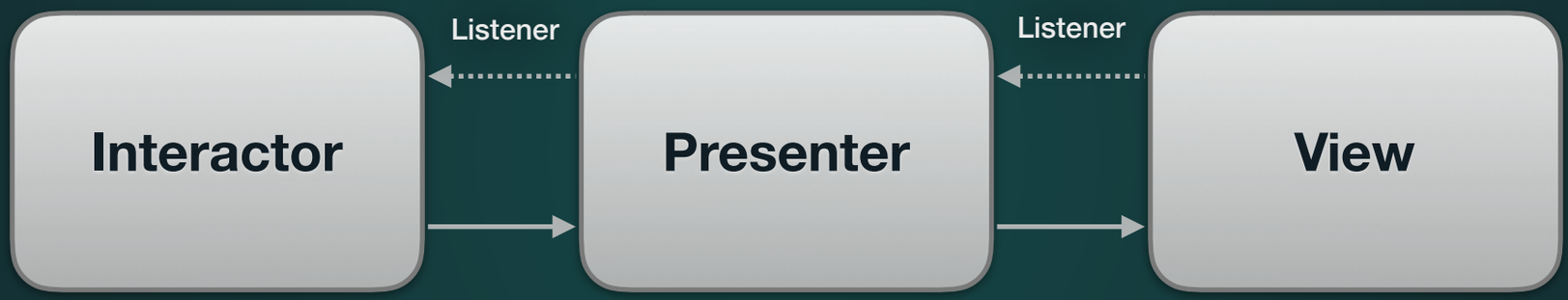
```
let amountStr = String(listener?.obtainAmount() ?? 0)
return listener?.selection.firstIndex(of: amountStr) ?? 1
```

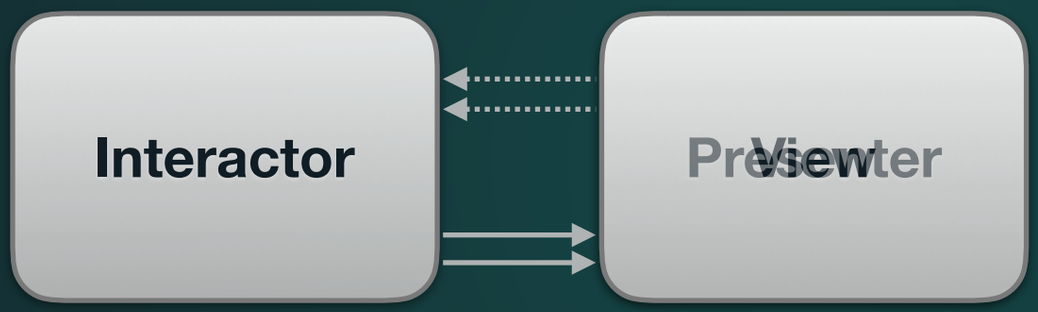
```
guard let steps = listener?.currentSteps,
    let statisticService = listener?.statisticService else {
```

Особенности Presenter'а:

1. Проксирование методов – бесполезный код
2. Сложный Presenter – ему нужно общаться и с Interactor'ом, и с View, и организовать их общение друг с другом.



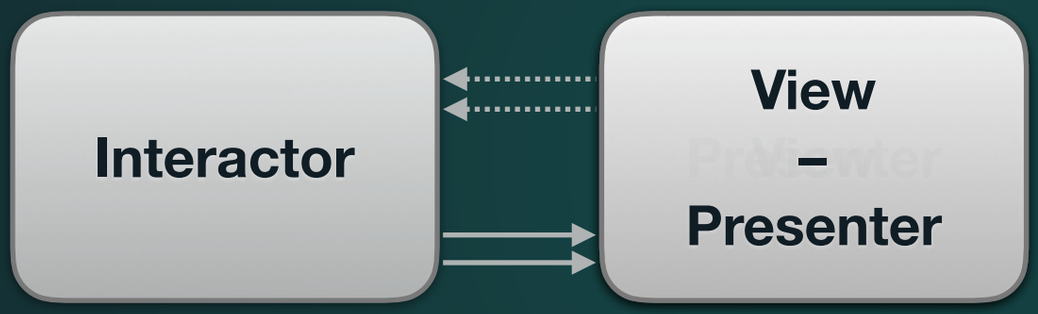

Listener
Listener
Interactor
Presenter
View


Interactor
Presenter
View

index

indexPath

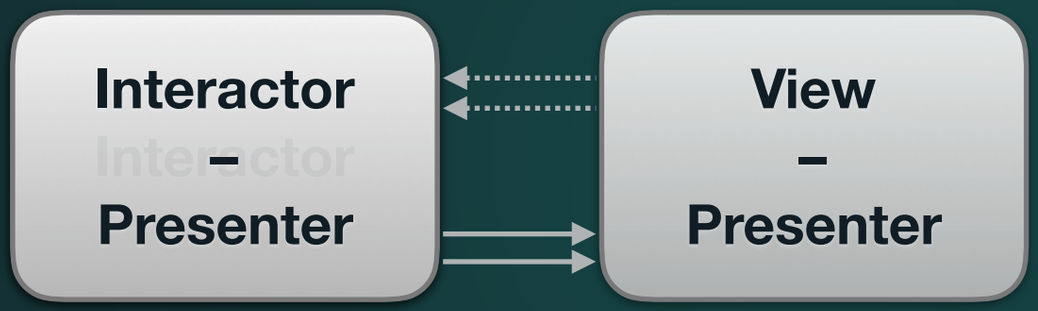
ViewModel



index

indexPath

ViewModel

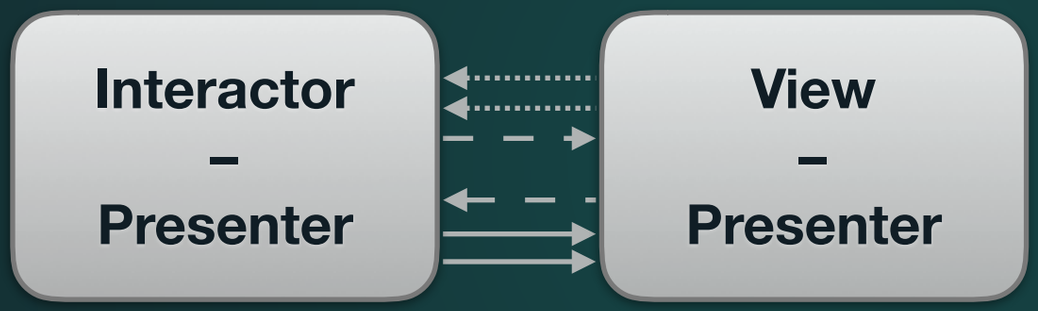


index

indexPath

ViewModel

Rx



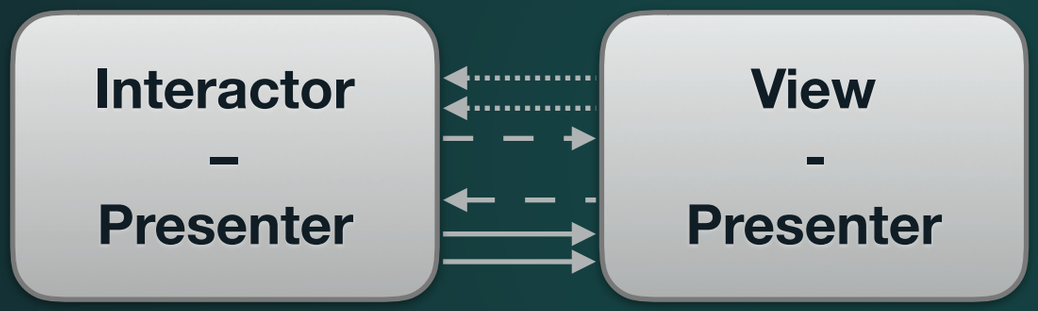
К	О	Л	Ю	П	Р	М	Е	Ч	З
Л	У	О	Н	К	И	О	Н	И	Т
А	В	И	В	И	Н	Ч	Е	С	О
С	К	А	Н	Е	Т	Е	Р	Т	Р
М	У	Т	У	Р	А	У	Ц	Е	Р
Ы	Л	Е	Р	Д	И	С	К	М	Д
Ш	И	Н	А	П	Р	О	Ц	Е	Е
Ь	П	Л	А	Н	У	Ц	Е	С	М
М	У	Ц	Я	Ш	Е	Т	В	С	У
П	Л	А	Т	А	В	Я	Т	О	Р

Цели

- Не получается заранее всё продумать, поэтому часто приходится что-нибудь «допиливать»
  - Составление диаграмм состояний экранов
  - Получение более полного и точного описания требований на этапе моделирования

- Нескончаемые баги после завершения разработки
  - Минимизировать проблему непредвиденного поведения экранов
  - Сделать код более устойчивым к внесению правок и доработок
  - Минимизировать нарушение SRP принципа.

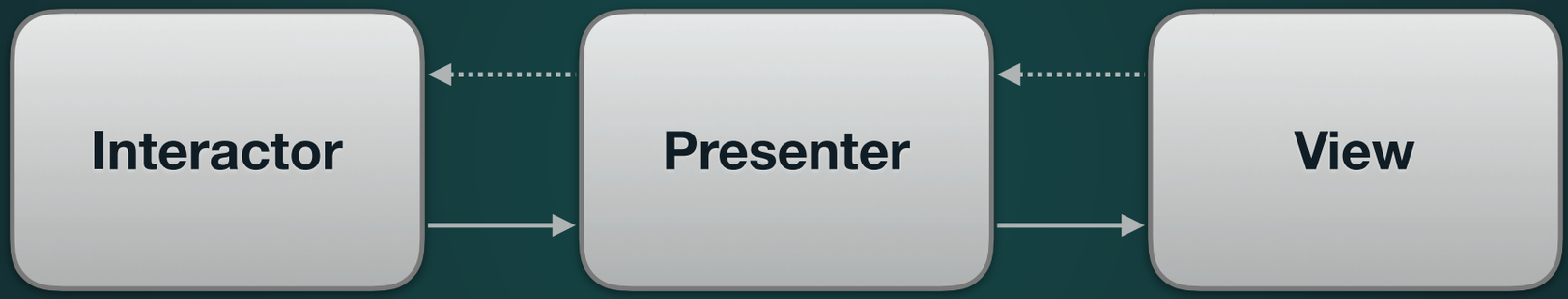
- Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
  - Единый стиль и структура компонентов любого экрана
  - Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
  - Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода.


Interactor
-
Presenter
View
-
Presenter

Interactor

Presenter

View


Interactor
Presenter
View

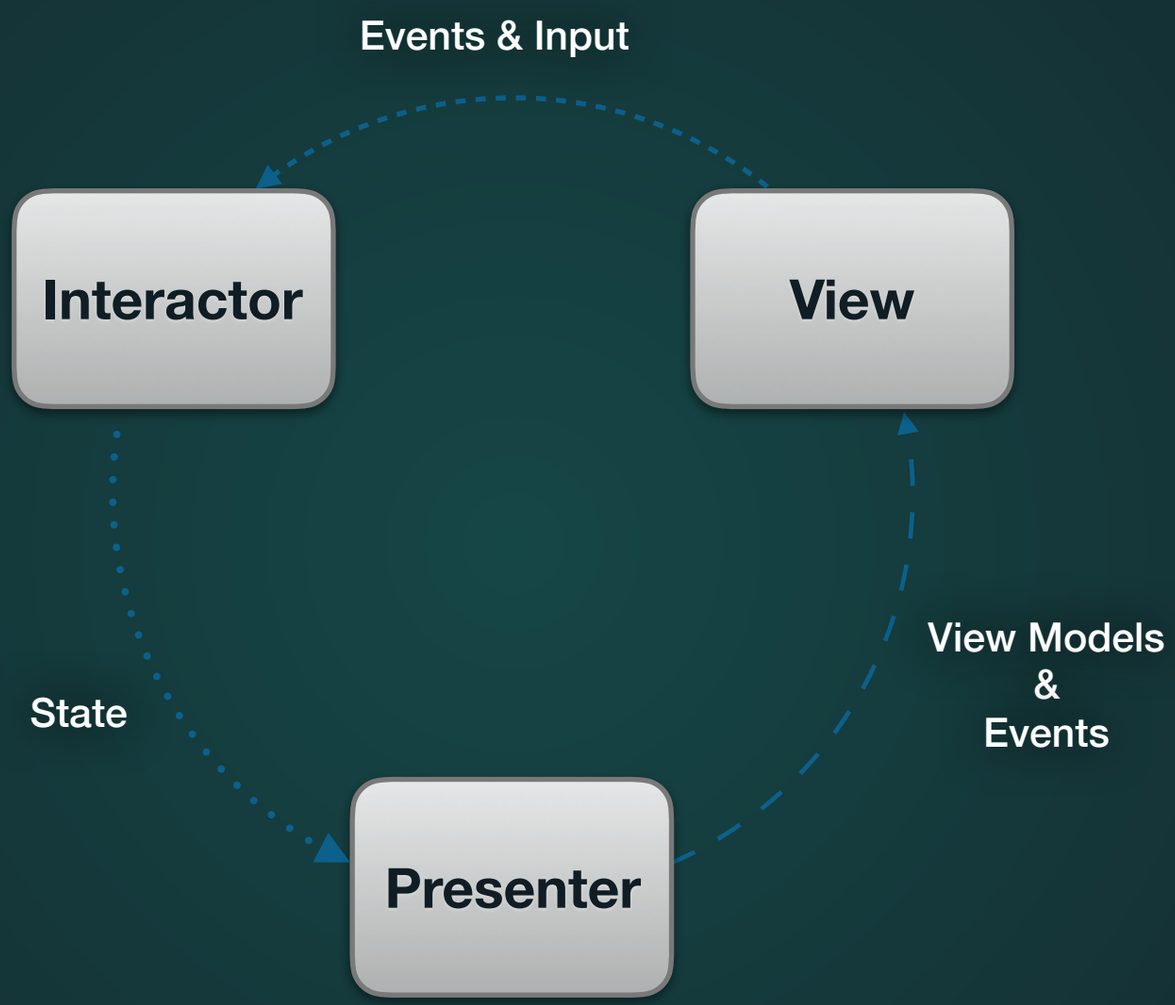
Interactor

Presenter

View

Unidirectional data flow

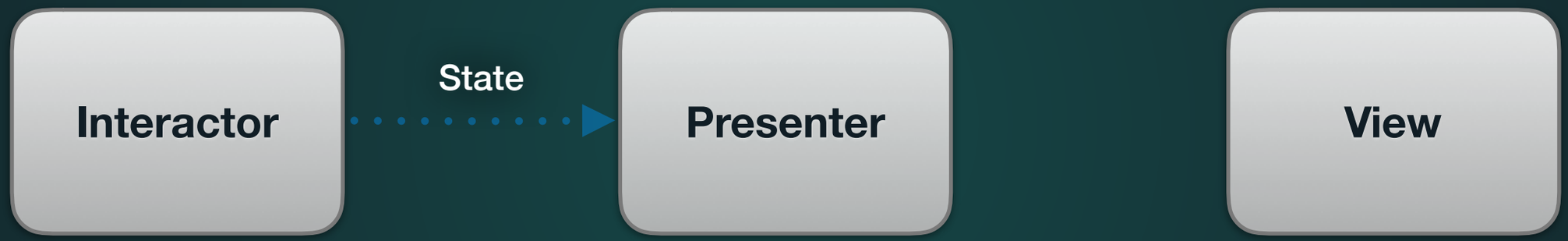
Unidirectional data flow

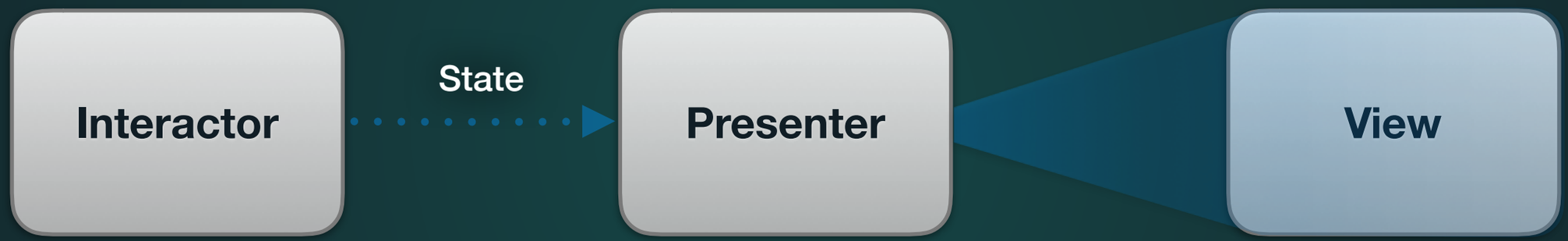


Interactor

Presenter

View


Interactor
State
Presenter
View


Interactor
State
Presenter
View

Interactor

Что происходит

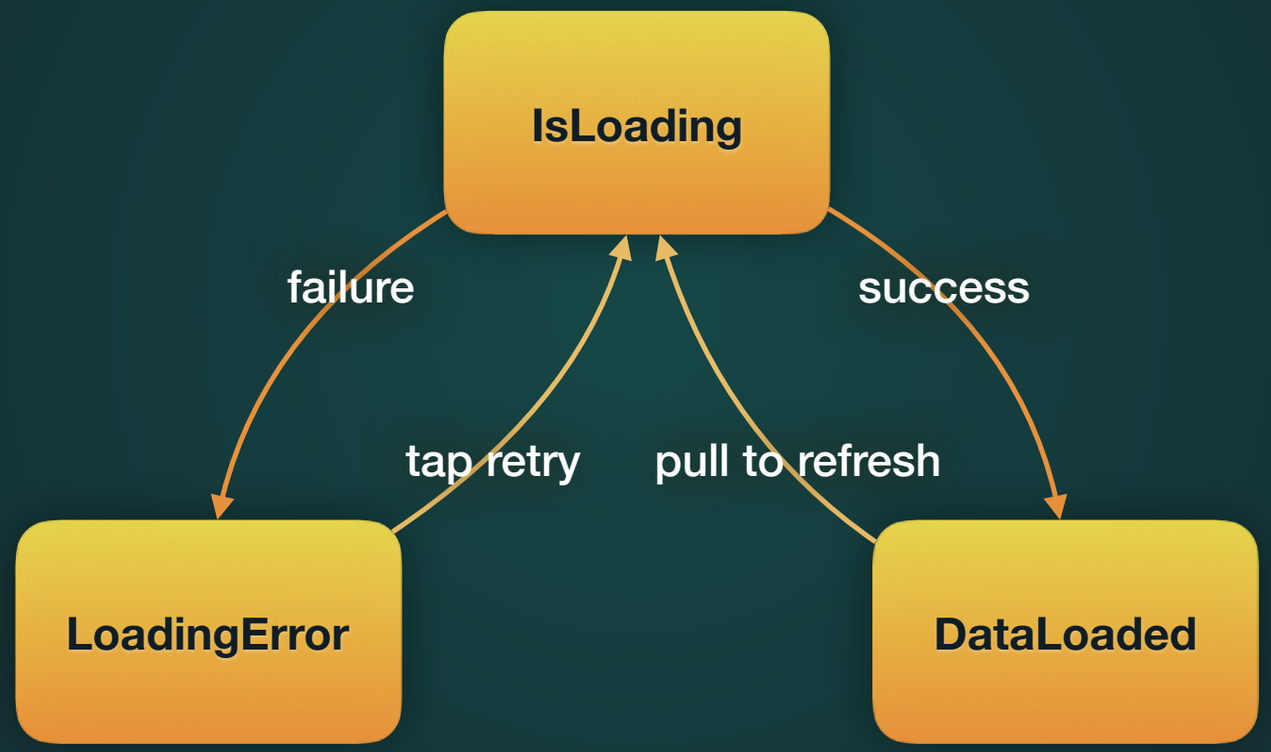
Presenter

Что нужно показать

View

Как нужно показать

Состояния экранов



NSOperation

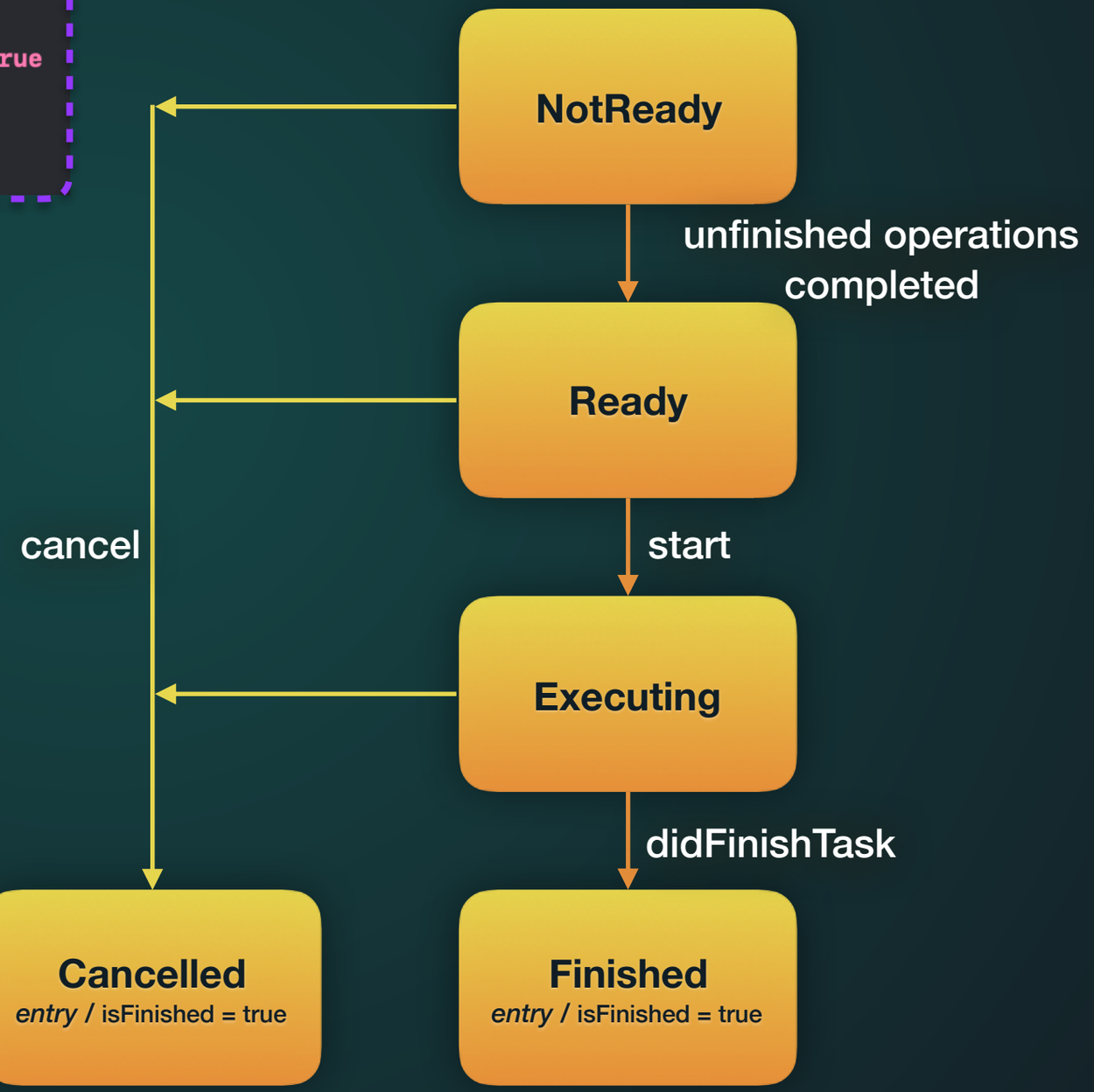
var isReady: Bool

var isExecuting: Bool

var isFinished: Bool

var isCancelled: Bool

```
var isExecuting: Bool {
    switch self {
    case .executing: return true
    default: return false
    }
}
```



GameplayKit Programming Guide

Fig

GKStateMachine

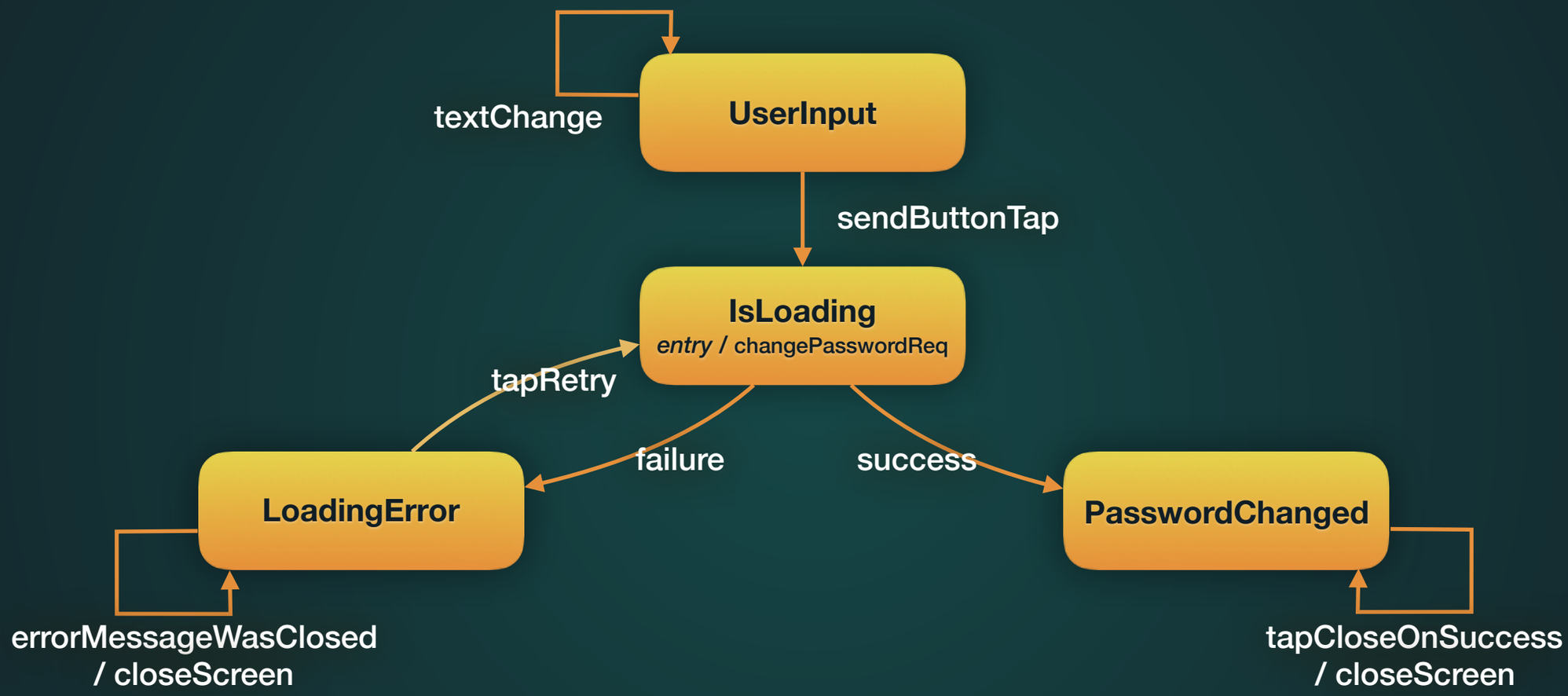
A finite-state machine—a collection of state objects that each define logic for a particular state of gameplay and rules for transitioning between states.

Overview

In GameplayKit, you subclass `GKState` to define each state and the rules for allowed transitions between states, and use a `GKStateMachine` instance to manage a machine that combines several states. This system provides a way to organize code in your game by organizing state-dependent actions into methods that run when entering a state, when exiting a state, and periodically while in a state (for example, on every animation frame your game renders).

- **Running.** While in this state, loop the run animation. If the player presses the jump button, switch to the Jump state. If the player runs off a ledge, switch to the Falling state.
- **Jumping.** On entering this state, play a sound effect, and start a one-time jumping animation. While in this state, move the character up a short distance (per frame), decelerating with gravity. When upward speed reaches zero, switch to the Falling state.
- **Falling.** On entering this state, start a one-time falling animation. While in this state, move the character down a short distance (per frame). On reaching the ground, switch to the Running state and play a one-time landing animation and sound effect.

Смена пароля



Смена пароля

UserInput

IsLoading

LoadingError

PasswordChanged

Смена пароля

UserInput

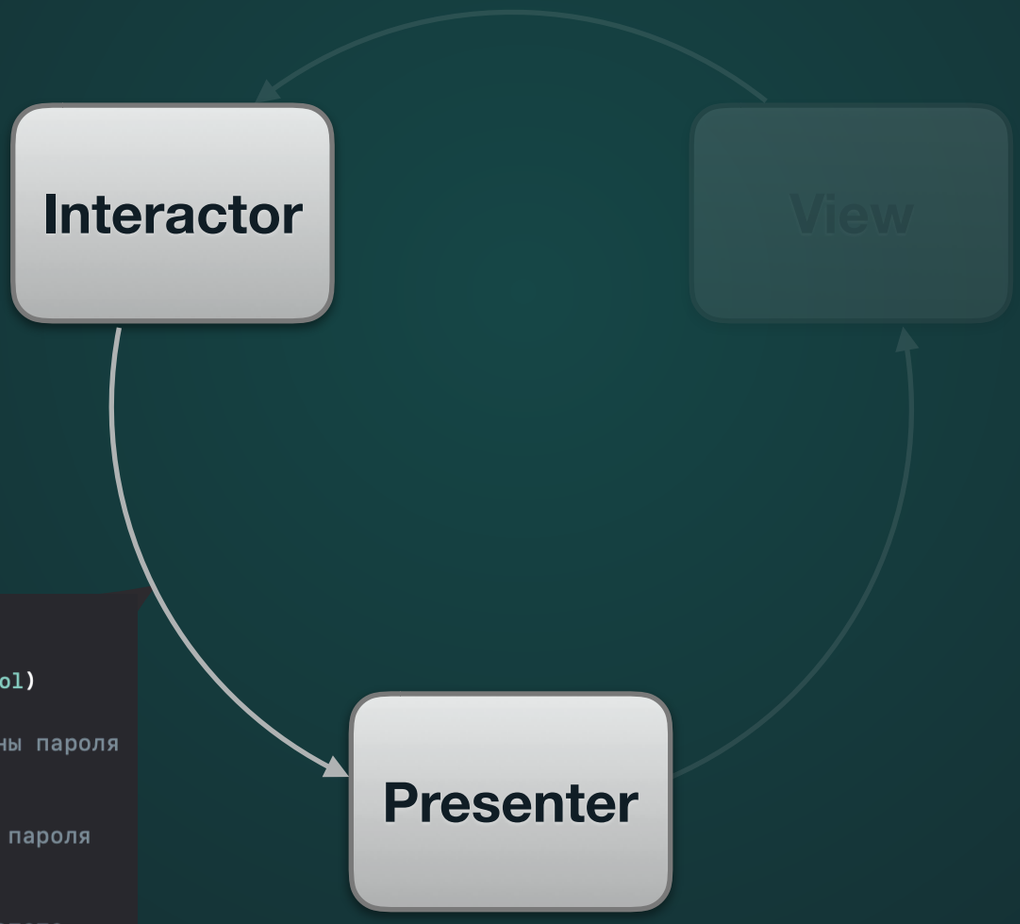
IsLoading

LoadingError

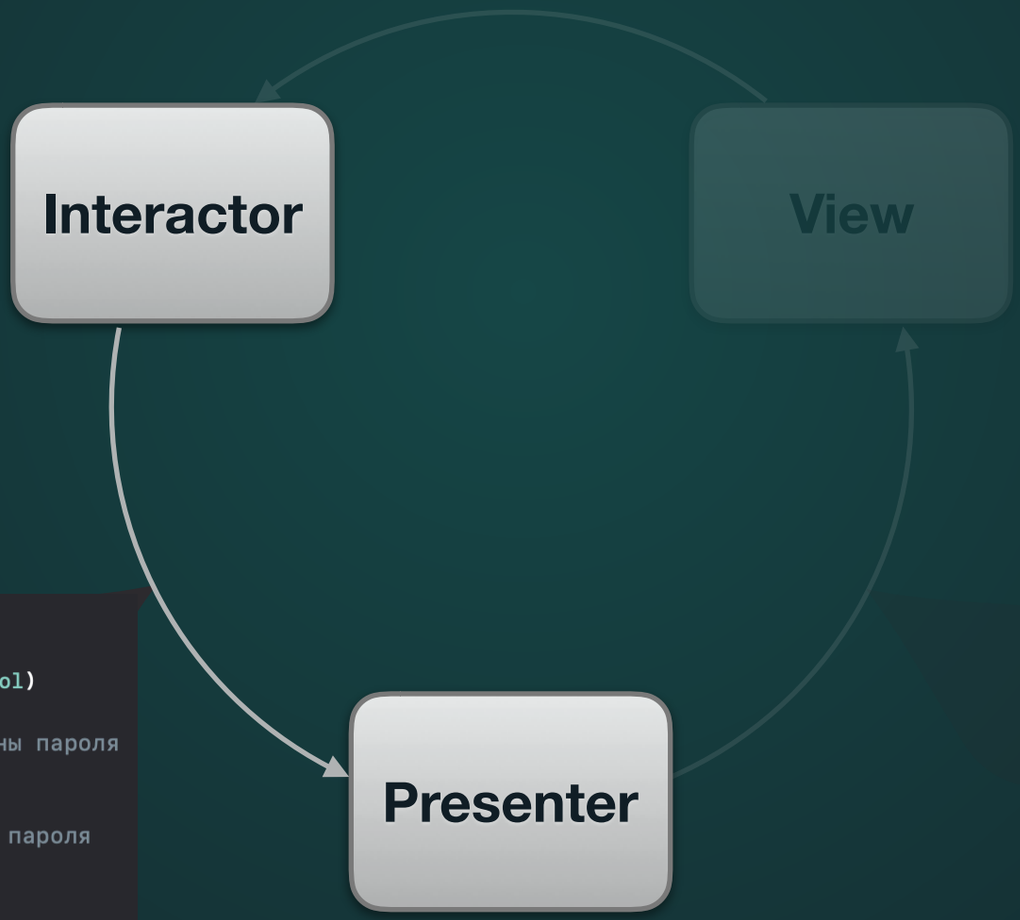
PasswordChanged

Смена пароля

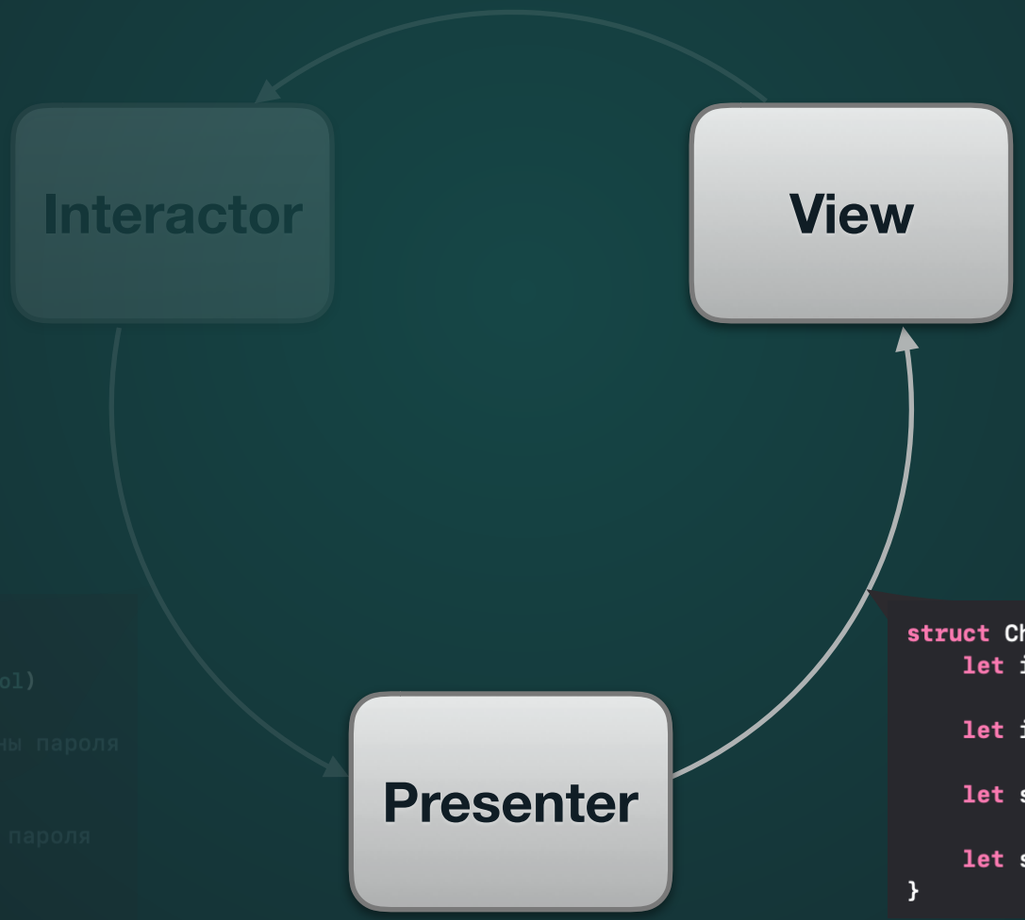
```
enum ChangePasswordInteractorState {
    /// Заполнение пароля пользователем
    case userInput(password: String, isValid: Bool)

    /// Ожидание ответа сервера на зпрос смены пароля
    case isLoading

    /// Ошибка ответа сервера на зпрос смены пароля
    case loadingError(Error)

    /** Это финальное состояние. Переход из этого
      состояния в другие невозможен. */
    case passwordChanged
}
```




```
enum ChangePasswordInteractorState {
    /// Заполнение пароля пользователем
    case userInput(password: String, isValid: Bool)

    /// Ожидание ответа сервера на зпрос смены пароля
    case isLoading

    /// Ошибка ответа сервера на зпрос смены пароля
    case loadingError(Error)

    /** Это финальное состояние. Переход из этого
      состояния в другие невозможен. */
    case passwordChanged
}
```




```
enum ChangePasswordInteractorState {
    /// Заполнение пароля пользователем
    case userInput(password: String, isValid: Bool)

    /// Ожидание ответа сервера на зпрос смены пароля
    case isLoading

    /// Ошибка ответа сервера на зпрос смены пароля
    case loadingError(Error)

    /** Это финальное состояние. Переход из этого
     состояния в другие невозможен. */
    case passwordChanged
}
```




```
enum ChangePasswordInteractorState {
    /// Заполнение пароля пользователем
    case userInput(password: String, isValid: Bool)

    /// Ожидание ответа сервера на зпрос смены пароля
    case isLoading

    /// Ошибка ответа сервера на зпрос смены пароля
    case loadingError(Error)

    /** Это финальное состояние. Переход из этого
     состояния в другие невозможен. */
    case passwordChanged
}
```

```
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

```swift
protocol ChangePasswordPresentableListener: AnyObject {
    var isLoading: Observable<Bool> { get }

    func send(password: String)

    func closeChangePassword()

    func passwordChanged()
}
```

```swift
protocol ChangePasswordPresentable: Presentable {
    var listener: ChangePasswordPresentableListener? { get set }

    func showSuccess(viewModel: ZeroScreenConfig)
    func showActionSheetFor(error: Error?)
}
```

Раньше

```swift
struct ChangePasswordViewOutput {
    let newPassword: Observable<String>

    /// Нажатие на анопку отправить (приводит к отпрвке пароля на
    let didTapSendButton: ControlEvent<Void>

    /// Нажатие на крестик в NavBar'е
    let didTapCloseButton: ControlEvent<Void>

    /// Нажатие в ZeroScreen'е на кнопку "закрыть" после успешной
    let didTapCloseAfterChangingSuccess: ControlEvent<Void>

    /// Событие,означающее что сообщение об ошибке было закрыто
    let errorMessageWasClosed: ControlEvent<Void>
}
```

```swift
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Сейчас

```swift
protocol ChangePasswordPresentableListener: AnyObject {
    var isLoading: Observable<Bool> { get }

    func send(password: String)

    func closeChangePassword()

    func passwordChanged()
}
```

```swift
protocol ChangePasswordPresentable: Presentable {
    var listener: ChangePasswordPresentableListener? { get set }

    func showSuccess(viewModel: ZeroScreenConfig)
    func showActionSheetFor(error: Error?)
}
```

Раньше

```swift
struct ChangePasswordViewOutput {
    let newPassword: Observable<String>

    /// Нажатие на анопку отправить (приводит к отпрвке пароля на
    let didTapSendButton: ControlEvent<Void>

    /// Нажатие на крестик в NavBar'е
    let didTapCloseButton: ControlEvent<Void>

    /// Нажатие в ZeroScreen'е на кнопку "закрыть" после успешной
    let didTapCloseAfterChangingSuccess: ControlEvent<Void>

    /// Событие,означающее что сообщение об ошибке было закрыто
    let errorMessageWasClosed: ControlEvent<Void>
}
```

```swift
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Сейчас

```swift
protocol ChangePasswordPresentableListener: AnyObject {
    var isLoading: Observable<Bool> { get }

    func send(password: String) !

    func closeChangePassword()

    func passwordChanged()
}
```

```swift
struct ChangePasswordViewOutput {
    let newPassword: Observable<String>

    /// Нажатие на анопку отправить (приводит к отпрвке пароля на
    let didTapSendButton: ControlEvent<Void>

    /// Нажатие на крестик в NavBar'е
    let didTapCloseButton: ControlEvent<Void>

    /// Нажатие в ZeroScreen'е на кнопку "закрыть" после успешной
    let didTapCloseAfterChangingSuccess: ControlEvent<Void>

    /// Событие,означающее что сообщение об ошибке было закрыто
    let errorMessageWasClosed: ControlEvent<Void>
}
```

?

```swift
protocol ChangePasswordPresentable: Presentable {
    var listener: ChangePasswordPresentableListener? { get set }

    func showSuccess(viewModel: ZeroScreenConfig)
    func showActionSheetFor(error: Error?)
}
```

?

```swift
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Раньше

Сейчас

```swift
protocol ChangePasswordPresentableListener: AnyObject {
    var isLoading: Observable<Bool> { get }

    func send(password: String) !

    func closeChangePassword()

    func passwordChanged()
}
```

```swift
struct ChangePasswordViewOutput {
    let newPassword: Observable<String>

    /// Нажатие на анопку отправить (приводит к отпрвке пароля на
    let didTapSendButton: ControlEvent<Void>

    /// Нажатие на крестик в NavBar'е
    let didTapCloseButton: ControlEvent<Void>

    /// Нажатие в ZeroScreen'е на кнопку "закрыть" после успешной
    let didTapCloseAfterChangingSuccess: ControlEvent<Void>

    /// Событие,означающее что сообщение об ошибке было закрыто
    let errorMessageWasClosed: ControlEvent<Void>
}
```

?

?

```swift
protocol ChangePasswordPresentable: Presentable {
    var listener: ChangePasswordPresentableListener? { get set }

    func showSuccess(viewModel: ZeroScreenConfig)
    func showActionSheetFor(error: Error?)
}
```

```swift
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Раньше

Сейчас

```
protocol ChangePasswordPresentableListener: AnyObject {
    var isLoading: Observable<Bool> { get }

    func send(password: String) !

    func closeChangePassword()

    func passwordChanged()
}
```

```
struct ChangePasswordViewOutput {
    let newPassword: Observable<String>

    /// Нажатие на анопку отправить (приводит к отпрвке пароля на
    let didTapSendButton: ControlEvent<Void>

    /// Нажатие на крестик в NavBar'e
    let didTapCloseButton: ControlEvent<Void>

    /// Нажатие в ZeroScreen'e на кнопку "закрыть" после успешной
    let didTapCloseAfterChangingSuccess: ControlEvent<Void>

    /// Событие,означающее что сообщение об ошибке было закрыто
    let errorMessageWasClosed: ControlEvent<Void>
}
```

?

?

```
protocol ChangePasswordPresentable: Presentable {
    var listener: ChangePasswordPresentableListener? { get set }

    func showSuccess(viewModel: ZeroScreenConfig)
    func showActionSheetFor(error: Error?)
}
```

```
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Раньше

Сейчас

```
protocol ChangePasswordPresentableListener: AnyObject {
    var isLoading: Observable<Bool> { get }

    func send(password: String)

    func closeChangePassword()

    func passwordChanged()
}
```

```
struct ChangePasswordViewOutput {
    let newPassword: Observable<String>

    /// Нажатие на кнопку отправить (приводит к отпрвке пароля на
    let didTapSendButton: ControlEvent<Void>

    /// Нажатие на крестик в NavBar'е
    let didTapCloseButton: ControlEvent<Void>

    /// Нажатие в ZeroScreen'е на кнопку "закрыть" после успешной
    let didTapCloseAfterChangingSuccess: ControlEvent<Void>

    /// Событие,означающее что сообщение об ошибке было закрыто
    let errorMessageWasClosed: ControlEvent<Void>
}
```

?

?

```
protocol ChangePasswordPresentable: Presentable {
    var listener: ChangePasswordPresentableListener? { get set }

    func showSuccess(viewModel: ZeroScreenConfig)
    func showActionSheetFor(error: Error?)
}
```

```
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Раньше

Сейчас

```
protocol ChangePasswordPresentableListener: AnyObject {
    var isLoading: Observable<Bool> { get }

    func send(password: String) !

    func closeChangePassword()

    func passwordChanged()
}
```

```
struct ChangePasswordViewOutput {
    let newPassword: Observable<String>

    /// Нажатие на анопку отправить (приводит к отпрвке пароля на
    let didTapSendButton: ControlEvent<Void>

    /// Нажатие на крестик в NavBar'е
    let didTapCloseButton: ControlEvent<Void>

    /// Нажатие в ZeroScreen'е на кнопку "закрыть" после успешной
    let didTapCloseAfterChangingSuccess: ControlEvent<Void>

    /// Событие,означающее что сообщение об ошибке было закрыто
    let errorMessageWasClosed: ControlEvent<Void>
}
```

?

?

```
protocol ChangePasswordPresentable: Presentable {
    var listener: ChangePasswordPresentableListener? { get set }

    func showSuccess(viewModel: ZeroScreenConfig)
    func showActionSheetFor(error: Error?)
}
```

```
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Раньше

Сейчас

Presenter

```
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

?

View

```
private func bindUI() {
    textField.rx.text
        .map { ($0?.count ?? 0) >= 6 }
        .bind(to: sendButton.rx.isEnabled)
        .disposed(by: bag)
}
```

?

Presenter

```
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Раньше | **Сейчас**

View

```
private func bindUI() {
    textField.rx.text
        .map { ($0?.count ?? 0) >= 6 }
        .bind(to: sendButton.rx.isEnabled)
        .disposed(by: bag)
}
```

1. Логика валидации находится внутри ViewController'а

Presenter

```
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Раньше | **Сейчас**

View

```
private func bindUI() {
    textField.rx.text
        .map { ($0?.count ?? 0) >= 6 }
        .bind(to: sendButton.rx.isEnabled)
        .disposed(by: bag)
}
```

1. Логика валидации находится внутри ViewController'а
2. Кликабельность кнопки никак не зависит от состояния экрана

Presenter

```
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Раньше

Сейчас

View

```
private func bindUI() {
    textField.rx.text
        .map { ($0?.count ?? 0) >= 6 }
        .bind(to: sendButton.rx.isEnabled)
        .disposed(by: bag)
}
```

1. Логика валидации находится внутри ViewController'а
2. Кликабельность кнопки никак не зависит от состояния экрана
3. View сама отвечает на вопрос «Что нужно показать»

Presenter

```
struct ChangePasswordPresenterOutput {
    let isLoadingIndicatorVisible: Driver<Bool>

    let isSendButtonEnabled: Driver<Bool>

    let showSuccess: Signal<Void>

    let showPasswordChangeError: Signal<String>
}
```

Раньше

Сейчас

View

```
private func bindUI() {
    textField.rx.text
        .map { ($0?.count ?? 0) >= 6 }
        .bind(to: sendButton.rx.isEnabled)
        .disposed(by: bag)
}
```

1. Логика валидации находится внутри ViewController'а
2. Кликабельность кнопки никак не зависит от состояния экрана
3. View сама отвечает на вопрос «Что нужно показать»

Presenter

```
func transform(_ input: Observable<ChangePasswordInteractorState>) -> ChangePasswordPresenterOutput {
    let state = input

    let isSendButtonEnabled = state
        .map { state -> Bool in
            switch state {
            case .userInput(_, let isValid): return isValid
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let isLoadingIndicatorVisible = state
        .map { state -> Bool in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let showPasswordChangeError = state
        .map { state -> String? in
            switch state {
            case .loadingError(let error): return error.localizedDescription
            default: return nil
            }
        }
        .asSignal(onErrorJustReturn: nil).compactMap()

    let showSuccess = state
        .map { state -> Void? in
            switch state {
            case .passwordChanged: return Void()
            default: return nil
            }
        }.compactMap().asSignal(onErrorJustReturn: Void())

    return ChangePasswordPresenterOutput(isLoadingIndicatorVisible: isLoadingIndicatorVisible,
                                         isSendButtonEnabled: isSendButtonEnabled,
                                         showSuccess: showSuccess,
                                         showPasswordChangeError: showPasswordChangeError)
}
```

Раньше

Сейчас

View

```
private func bindUI() {
    textField.rx.text
        .map { ($0?.count ?? 0) >= 6 }
        .bind(to: sendButton.rx.isEnabled)
        .disposed(by: bag)
}
```

1. Логика валидации находится внутри ViewController'а
2. Кликабельность кнопки никак не зависит от состояния экрана
3. View сама отвечает на вопрос «Что нужно показать»

Presenter

```
func transform(_ input: Observable<ChangePasswordInteractorState>) -> ChangePasswordPresenterOutput {
    let state = input

    let isSendButtonEnabled = state
        .map { state -> Bool in
            switch state {
            case .userInput(_, let isValid): return isValid
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let isLoadingIndicatorVisible = state
        .map { state -> Bool in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let showPasswordChangeError = state
        .map { state -> String? in
            switch state {
            case .loadingError(let error): return error.localizedDescription
            default: return nil
            }
        }
        .asSignal(onErrorJustReturn: nil).compactMap()

    let showSuccess = state
        .map { state -> Void? in
            switch state {
            case .passwordChanged: return Void()
            default: return nil
            }
        }.compactMap().asSignal(onErrorJustReturn: Void())

    return ChangePasswordPresenterOutput(isLoadingIndicatorVisible: isLoadingIndicatorVisible,
                                         isSendButtonEnabled: isSendButtonEnabled,
                                         showSuccess: showSuccess,
                                         showPasswordChangeError: showPasswordChangeError)
}
```

Раньше

Сейчас

Presenter

```
func transform(_ input: Observable<ChangePasswordInteractorState>) -> ChangePasswordPresenterOutput {
    let state = input

    let isSendButtonEnabled = state
        .map { state -> Bool in
            switch state {
            case .userInput(_, let isValid): return isValid
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let isLoadingIndicatorVisible = state
        .map { state -> Bool in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let showPasswordChangeError = state
        .map { state -> String? in
            switch state {
            case .loadingError(let error): return error.localizedDescription
            default: return nil
            }
        }
        .asSignal(onErrorJustReturn: nil).compactMap()

    let showSuccess = state
        .map { state -> Void? in
            switch state {
            case .passwordChanged: return Void()
            default: return nil
            }
        }.compactMap().asSignal(onErrorJustReturn: Void())

    return ChangePasswordPresenterOutput(isLoadingIndicatorVisible: isLoadingIndicatorVisible,
                                         isSendButtonEnabled: isSendButtonEnabled,
                                         showSuccess: showSuccess,
                                         showPasswordChangeError: showPasswordChangeError)
}
```

Сейчас

1. Логика валидации теперь находится в Interactor’е

Presenter

```
func transform(_ input: Observable<ChangePasswordInteractorState>) -> ChangePasswordPresenterOutput {
    let state = input

    let isSendButtonEnabled = state
        .map { state -> Bool in
            switch state {
            case .userInput(_, let isValid): return isValid
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let isLoadingIndicatorVisible = state
        .map { state -> Bool in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let showPasswordChangeError = state
        .map { state -> String? in
            switch state {
            case .loadingError(let error): return error.localizedDescription
            default: return nil
            }
        }
        .asSignal(onErrorJustReturn: nil).compactMap()

    let showSuccess = state
        .map { state -> Void? in
            switch state {
            case .passwordChanged: return Void()
            default: return nil
            }
        }.compactMap().asSignal(onErrorJustReturn: Void())

    return ChangePasswordPresenterOutput(isLoadingIndicatorVisible: isLoadingIndicatorVisible,
                                         isSendButtonEnabled: isSendButtonEnabled,
                                         showSuccess: showSuccess,
                                         showPasswordChangeError: showPasswordChangeError)
}
```

Сейчас

1. Логика валидации теперь находится в Interactor’е
2. Кликабельность кнопки зависит от состояния экрана

Presenter

```
func transform(_ input: Observable<ChangePasswordInteractorState>) -> ChangePasswordPresenterOutput {
    let state = input

    let isSendButtonEnabled = state
        .map { state -> Bool in
            switch state {
            case .userInput(_, let isValid): return isValid
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let isLoadingIndicatorVisible = state
        .map { state -> Bool in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let showPasswordChangeError = state
        .map { state -> String? in
            switch state {
            case .loadingError(let error): return error.localizedDescription
            default: return nil
            }
        }
        .asSignal(onErrorJustReturn: nil).compactMap()

    let showSuccess = state
        .map { state -> Void? in
            switch state {
            case .passwordChanged: return Void()
            default: return nil
            }
        }.compactMap().asSignal(onErrorJustReturn: Void())

    return ChangePasswordPresenterOutput(isLoadingIndicatorVisible: isLoadingIndicatorVisible,
                                         isSendButtonEnabled: isSendButtonEnabled,
                                         showSuccess: showSuccess,
                                         showPasswordChangeError: showPasswordChangeError)
}
```

Сейчас

1. Логика валидации теперь находится в Interactor’е
2. Кликабельность кнопки зависит от состояния экрана
3. На вопрос «Что нужно показать» отвечает Presenter

Presenter

```
func transform(_ input: Observable<ChangePasswordInteractorState>) -> ChangePasswordPresenterOutput {
    let state = input

    let isSendButtonEnabled = state
        .map { state -> Bool in
            switch state {
            case .userInput(_, let isValid): return isValid
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let isLoadingIndicatorVisible = state
        .map { state -> Bool in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let showPasswordChangeError = state
        .map { state -> String? in
            switch state {
            case .loadingError(let error): return error.localizedDescription
            default: return nil
            }
        }
        .asSignal(onErrorJustReturn: nil).compactMap()

    let showSuccess = state
        .map { state -> Void? in
            switch state {
            case .passwordChanged: return Void()
            default: return nil
            }
        }.compactMap().asSignal(onErrorJustReturn: Void())

    return ChangePasswordPresenterOutput(isLoadingIndicatorVisible: isLoadingIndicatorVisible,
                                         isSendButtonEnabled: isSendButtonEnabled,
                                         showSuccess: showSuccess,
                                         showPasswordChangeError: showPasswordChangeError)
}
```

Сейчас

Interactor

```
func send(password: String) {
    analyticalService.logEvent(screen: AnalyticalSpace.Auth.changePass,
                               event: AnalyticalEvent.Tap.Auth.buttonApply)
    _isLoading.onNext(true)

    authService.changePassword(newPassword: password) { [weak self] result in
        guard let strongSelf = self else { return }
        switch result {
        case .success(let model):
            strongSelf.obtainContractInfo(newPassword: model.password)
            strongSelf.changePassSuccessEvent()
        case .failure(let error):
            strongSelf._isLoading.onNext(false)
            strongSelf.failChangePassEvent(error: error)
        }
    }
}
```

```
private func obtainContractInfo(newPassword: String) {
    authService.obtainContractInfo { [weak self] result in

        guard let strongSelf = self else { return }

        strongSelf._isLoading.onNext(false)

        switch result {
        case .success(let contractInfo):
            strongSelf.showSuccess()

        case .failure(let error):
            strongSelf.presenter.showActionSheetFor(error: error)
        }
    }
}
```

Presenter

```
func transform(_ input: Observable<ChangePasswordInteractorState>) -> ChangePasswordPresenterOutput {
    let state = input

    let isSendButtonEnabled = state
        .map { state -> Bool in
            switch state {
            case .userInput(_, let isValid): return isValid
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let isLoadingIndicatorVisible = state
        .map { state -> Bool in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let showPasswordChangeError = state
        .map { state -> String? in
            switch state {
            case .loadingError(let error): return error.localizedDescription
            default: return nil
            }
        }
        .asSignal(onErrorJustReturn: nil).compactMap()

    let showSuccess = state
        .map { state -> Void? in
            switch state {
            case .passwordChanged: return Void()
            default: return nil
            }
        }.compactMap().asSignal(onErrorJustReturn: Void())

    return ChangePasswordPresenterOutput(isLoadingIndicatorVisible: isLoadingIndicatorVisible,
                                         isSendButtonEnabled: isSendButtonEnabled,
                                         showSuccess: showSuccess,
                                         showPasswordChangeError: showPasswordChangeError)
}
```

Раньше

Сейчас

Interactor

```
func send(password: String) {
    analyticalService.logEvent(screen: AnalyticalSpace.Auth.changePass,
                               event: AnalyticalEvent.Tap.Auth.buttonApply)
    _isLoading.onNext(true)

    authService.changePassword(newPassword: password) { [weak self] result in
        guard let strongSelf = self else { return }
        switch result {
        case .success(let model):
            strongSelf.obtainContractInfo(newPassword: model.password)
            strongSelf.changePassSuccessEvent()
        case .failure(let error):
            strongSelf._isLoading.onNext(false)
            strongSelf.presenter.showActionSheetFor(error: error)
            strongSelf.failChangePassEvent(error: error)
        }
    }
}
```

```
private func obtainContractInfo(newPassword: String) {
    authService.obtainContractInfo { [weak self] result in

        guard let strongSelf = self else { return }

        strongSelf._isLoading.onNext(false)

        switch result {
        case .success(let contractInfo):
            strongSelf.showSuccess()

        case .failure(let error):
            strongSelf.presenter.showActionSheetFor(error: error)
        }
    }
}
```

Раньше

Presenter

```
func transform(_ input: Observable<ChangePasswordInteractorState>) -> ChangePasswordPresenterOutput {
    let state = input

    let isSendButtonEnabled = state
        .map { state -> Bool in
            switch state {
            case .userInput(_, let isValid): return isValid
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let isLoadingIndicatorVisible = state
        .map { state -> Bool in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }.asDriver(onErrorJustReturn: false)

    let showPasswordChangeError = state
        .map { state -> String? in
            switch state {
            case .loadingError(let error): return error.localizedDescription
            default: return nil
            }
        }
        .asSignal(onErrorJustReturn: nil).compactMap()

    let showSuccess = state
        .map { state -> Void? in
            switch state {
            case .passwordChanged: return Void()
            default: return nil
            }
        }.compactMap().asSignal(onErrorJustReturn: Void())

    return ChangePasswordPresenterOutput(isLoadingIndicatorVisible: isLoadingIndicatorVisible,
                                         isSendButtonEnabled: isSendButtonEnabled,
                                         showSuccess: showSuccess,
                                         showPasswordChangeError: showPasswordChangeError)
}
```

Сейчас

View

```
extension ChangePasswordViewController: BindableView {
    func getOutput() -> ChangePasswordViewOutput {

        return ChangePasswordViewOutput(newPassword: textField.rx.text.orEmpty.asObservable(),
                                        didTapSendButton: sendButton.rx.tap,
                                        didTapCloseButton: ControlEvent(events: didTapCloseButton),
                                        didTapCloseAfterChangingSuccess: ControlEvent(events: didTapCloseAfterSuccess),
                                        errorMessageWasClosed: ControlEvent(events: errorMessageWasClosed))
    }

    func bindWith(_ input: ChangePasswordPresenterOutput) {
        input.isLoadingIndicatorVisible.drive(onNext: { [weak self] visible in
            if visible {
                self?.showActivityController()
            } else {
                self?.hideActivityController()
            }
        }).disposed(by: disposeBag)

        input.isSendButtonEnabled.drive(sendButton.rx.isEnabled).disposed(by: disposeBag)

        input.showPasswordChangeError.emit(onNext: { [weak self] message in
            self?.showError(message: message)
        }).disposed(by: disposeBag)

        input.showSuccess.emit(onNext: { [weak self] in
            self?.showSuccess()
        }).disposed(by: disposeBag)
    }
}
```

Цели

- [ ] Составление диаграмм состояний экранов
- [ ] Получение более полного и точного описания требований на этапе моделирования
- [ ] Минимизировать проблему непредвиденного поведения экранов
- [ ] Сделать код более устойчивым к внесению правок и доработок
- [ ] Минимизировать нарушение SRP принципа
- [ ] Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- [ ] Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- [ ] Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

Цели

- ☑ Составление диаграмм состояний экранов
- ☐ Получение более полного и точного описания требований на этапе моделирования
- ☐ Минимизировать проблему непредвиденного поведения экранов
- ☐ Сделать код более устойчивым к внесению правок и доработок
- ☐ Минимизировать нарушение SRP принципа
- ☐ Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- ☐ Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- ☐ Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

Цели

- ☑ Составление диаграмм состояний экранов
- ☑ Получение более полного и точного описания требований на этапе моделирования
- ☐ Минимизировать проблему непредвиденного поведения экранов
- ☐ Сделать код более устойчивым к внесению правок и доработок
- ☐ Минимизировать нарушение SRP принципа
- ☐ Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- ☐ Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- ☐ Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

Цели

- ☑ Составление диаграмм состояний экранов
- ☑ Получение более полного и точного описания требований на этапе моделирования
- ☑ Минимизировать проблему непредвиденного поведения экранов
- ☐ Сделать код более устойчивым к внесению правок и доработок
- ☐ Минимизировать нарушение SRP принципа
- ☐ Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- ☐ Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- ☐ Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

Цели

- ☑ Составление диаграмм состояний экранов
- ☑ Получение более полного и точного описания требований на этапе моделирования
- ☑ Минимизировать проблему непредвиденного поведения экранов
- ☑ Сделать код более устойчивым к внесению правок и доработок
- ☐ Минимизировать нарушение SRP принципа
- ☐ Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- ☐ Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- ☐ Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

Цели

- ☑ Составление диаграмм состояний экранов
- ☑ Получение более полного и точного описания требований на этапе моделирования
- ☑ Минимизировать проблему непредвиденного поведения экранов
- ☑ Сделать код более устойчивым к внесению правок и доработок
- ☐ Минимизировать нарушение SRP принципа
- ☐ Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- ☐ Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- ☑ Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

Interactor

```
private let authService: AuthorizationService
private let credentials: CredentialSupplier

private let _state = BehaviorRelay<ChangePasswordInteractorState>(value: .userInput(password: "", isValid: false))

private let didChangePassword = PublishRelay<Void>()
private let passwordChangingRequestFailed = PublishRelay<Error>()

private let disposeBag = DisposeBag()
```

```
private extension ChangePasswordInteractor {
    /// Метод осуществляет смену пароля. Принимает на вход провалидированный пароль.
    private func performChangePasswordActions(newPassword: String) {
        authService.changePassword(newPassword: newPassword) { [weak self] result in
            guard let strongSelf = self else { return }
            switch result {
            case .success(let model): strongSelf.loadContractInfo(newPassword: model.password)
            case .failure(let error): strongSelf.passwordChangingRequestFailed.accept(error)
            }
        }
    }

    private func loadContractInfo(newPassword: String) {
        authService.obtainContractInfo { [weak self] result in
            guard let strongSelf = self else { return }

            switch result {
            case .success(let contractInfo):
                strongSelf.credentials.save(contractInfo: contractInfo.contract, password: newPassword)
                strongSelf.didChangePassword.accept(Void())
            case .failure(let error):
                strongSelf.passwordChangingRequestFailed.accept(error)
            }
        }
    }
}
```

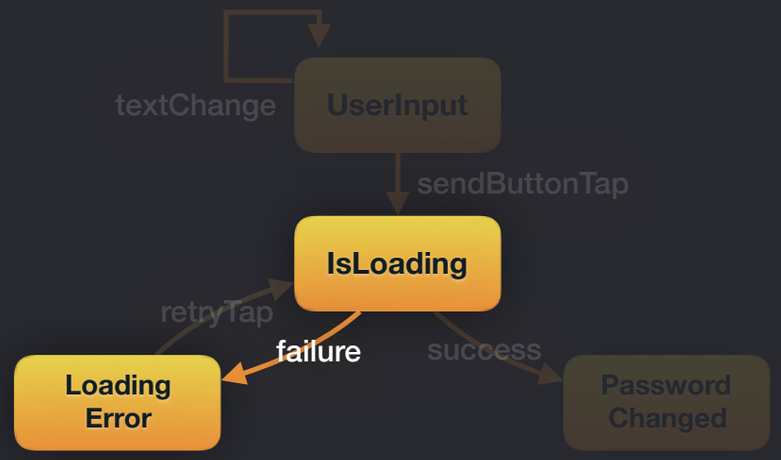
1

2

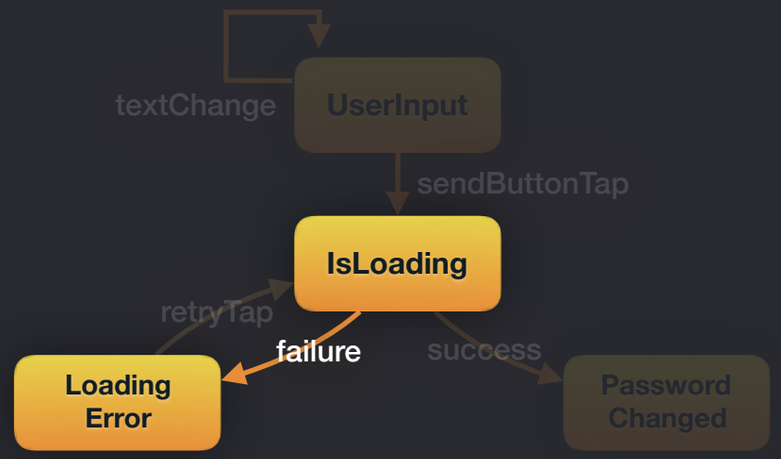
Важно!

- В конкретном состоянии Interactor реагирует на конкретные события, и игнорируирует все остальные.
  На практике, чаще всего в конкретном состоянии он реагирует на какое-то 1 событие. Иногда их может быть несколько.

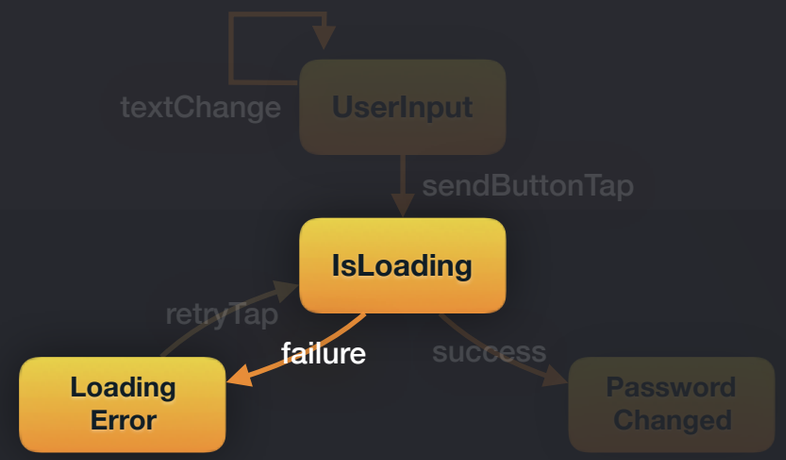
```swift
func transform(_ input: ChangePasswordViewOutput) -> Observable<ChangePasswordInteractorState> {
    let readonlyState = _state.asObservable()

    let newPassword = input.newPassword
        .skip(1) // Исходное значение textField'а пропускаем
        .map { $0.trimmingWhitespacesAndNewlines() }
        .distinctUntilChanged()
        .share()

    StateTransform.transiotionToUserInput(state: readonlyState,
                                          newPassword: newPassword,
                                          errorMessageWasClosed: input.errorMessageWasClosed,
                                          bindoTo: _state,
                                          disposedBy: disposeBag)

    let loadingEntryAction: (String) -> Void = { [weak self] newPassword in
        self?.performChangePasswordActions(newPassword: newPassword)
    }

    StateTransform.transiotionToLoading(readonlyState: readonlyState,
                                        didTapSendButton: input.didTapSendButton,
                                        bindoTo: _state,
                                        disposedBy: disposeBag,
                                        stateEntryAction: loadingEntryAction)

    StateTransform.fromLoadingToLoadingError(state: readonlyState,
                                             passwordChangingRequestFailed: passwordChangingRequestFailed.asObservable(),
                                             bindoTo: _state,
                                             disposedBy: disposeBag)

    StateTransform.fromLoadingToPasswordChanged(state: readonlyState,
                                                didChangePassword: didChangePassword.asObservable(),
                                                bindoTo: _state,
                                                disposedBy: disposeBag)

    bindListner(readonlyState: readonlyState,
                didTapCloseButton: input.didTapCloseButton,
                didTapCloseAfterChangingSuccess: input.didTapCloseAfterChangingSuccess)

    return readonlyState
}
```



```swift
func transform(_ input: ChangePasswordViewOutput) -> Observable<ChangePasswordInteractorState> {
    let readonlyState = _state.asObservable()

    let newPassword = input.newPassword
        .skip(1) // Исходное значение textField'а пропускаем
        .map { $0.trimmingWhitespacesAndNewlines() }
        .distinctUntilChanged()
        .share()

    StateTransform.transiotionToUserInput(state: readonlyState,
                                          newPassword: newPassword,
                                          errorMessageWasClosed: input.errorMessageWasClosed,
                                          bindoTo: _state,
                                          disposedBy: disposeBag)

    let loadingEntryAction: (String) -> Void = { [weak self] newPassword in
        self?.performChangePasswordActions(newPassword: newPassword)
    }

    StateTransform.transiotionToLoading(readonlyState: readonlyState,
                                        didTapSendButton: input.didTapSendButton,
                                        bindoTo: _state,
                                        disposedBy: disposeBag,
                                        stateEntryAction: loadingEntryAction)

    StateTransform.fromLoadingToLoadingError(state: readonlyState,
                                             passwordChangingRequestFailed: passwordChangingRequestFailed.asObservable(),
                                             bindoTo: _state,
                                             disposedBy: disposeBag)

    StateTransform.fromLoadingToPasswordChanged(state: readonlyState,
                                                didChangePassword: didChangePassword.asObservable(),
                                                bindoTo: _state,
                                                disposedBy: disposeBag)

    bindListner(readonlyState: readonlyState,
                didTapCloseButton: input.didTapCloseButton,
                didTapCloseAfterChangingSuccess: input.didTapCloseAfterChangingSuccess)

    return readonlyState
}
```



```swift
/// - Переход из состояния 'loading' в 'loadingError' (когда загрузка заврешилась ошибкой)
static func fromLoadingToLoadingError(state readonlyState: Observable<ChangePasswordInteractorState>,
                                      passwordChangingRequestFailed: Observable<Error>,
                                      bindoTo _state: BehaviorRelay<ChangePasswordInteractorState>,
                                      disposedBy disposeBag: DisposeBag) {
    let fromLoadingToLoadingError = passwordChangingRequestFailed
        .withLatestFrom(readonlyState, resultSelector: latestFromBothValues())
        .filter { _, state in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }
        .map { error, _ in State.loadingError(error) }

    fromLoadingToLoadingError.bind(to: _state).disposed(by: disposeBag)
}
```



Стейт состояния при событии ошибки ответа сервера:

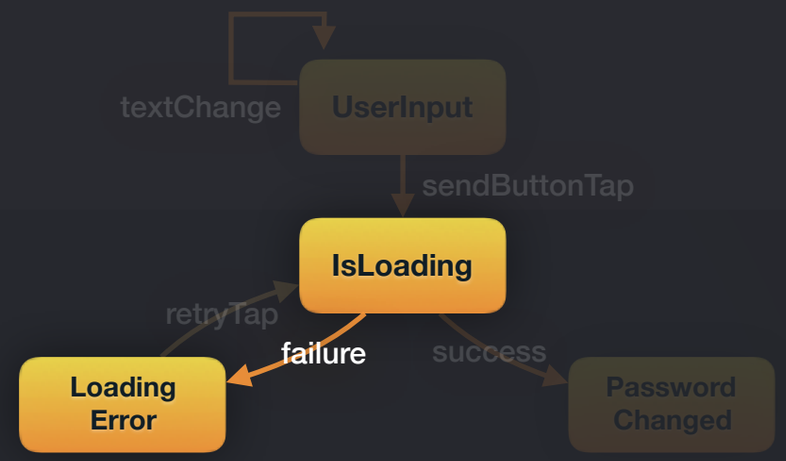
1. При инвертировании события захватываем текущее состояние

2. Событие допускается только в состоянии isLoading. Проверяем это с помощью оператора filter

3. Создаём состояние loadingError

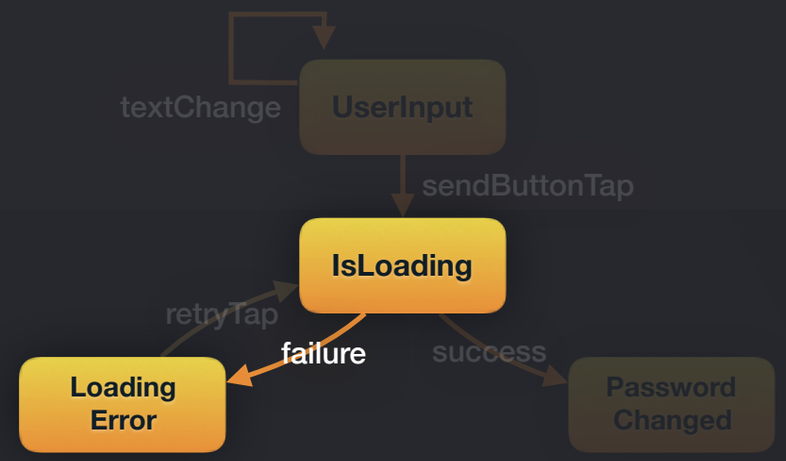
4. Обновляем текущее состояние

```swift
/// - Переход из состояния 'loading' в 'loadingError' (когда загрузка заврешилась ошибкой)
static func fromLoadingToLoadingError(state readonlyState: Observable<ChangePasswordInteractorState>,
                                      passwordChangingRequestFailed: Observable<Error>,
                                      bindoTo _state: BehaviorRelay<ChangePasswordInteractorState>,
                                      disposedBy disposeBag: DisposeBag) {
    let fromLoadingToLoadingError = passwordChangingRequestFailed
        .withLatestFrom(readonlyState, resultSelector: latestFromBothValues())
        .filter { _, state in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }
        .map { error, _ in State.loadingError(error) }

    fromLoadingToLoadingError.bind(to: _state).disposed(by: disposeBag)
}
```



Смена состояния при ошибке ответа сервера:

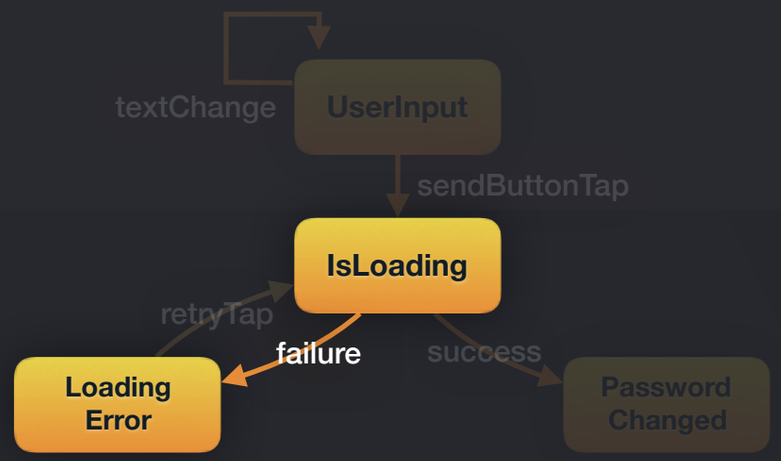
```swift
/// - Переход из состояния 'loading' в 'loadingError' (когда загрузка заврешилась ошибкой)
static func fromLoadingToLoadingError(state readonlyState: Observable<ChangePasswordInteractorState>,
                                      passwordChangingRequestFailed: Observable<Error>,
                                      bindoTo _state: BehaviorRelay<ChangePasswordInteractorState>,
                                      disposedBy disposeBag: DisposeBag) {
    let fromLoadingToLoadingError = passwordChangingRequestFailed
        .withLatestFrom(readonlyState, resultSelector: latestFromBothValues())
        .filter { _, state in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }
        .map { error, _ in State.loadingError(error) }

    fromLoadingToLoadingError.bind(to: _state).disposed(by: disposeBag)
}
```



Смена состояния при ошибке ответа сервера:

1. При наступлении события захватываем текущее состояние.
Используем для этого оператор withLatestFrom

```swift
/// - Переход из состояния 'loading' в 'loadingError' (когда загрузка заврешилась ошибкой)
static func fromLoadingToLoadingError(state readonlyState: Observable<ChangePasswordInteractorState>,
                                      passwordChangingRequestFailed: Observable<Error>,
                                      bindoTo _state: BehaviorRelay<ChangePasswordInteractorState>,
                                      disposedBy disposeBag: DisposeBag) {
    let fromLoadingToLoadingError = passwordChangingRequestFailed
        .withLatestFrom(readonlyState, resultSelector: latestFromBothValues())
        .filter { _, state in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }
        .map { error, _ in State.loadingError(error) }

    fromLoadingToLoadingError.bind(to: _state).disposed(by: disposeBag)
}
```

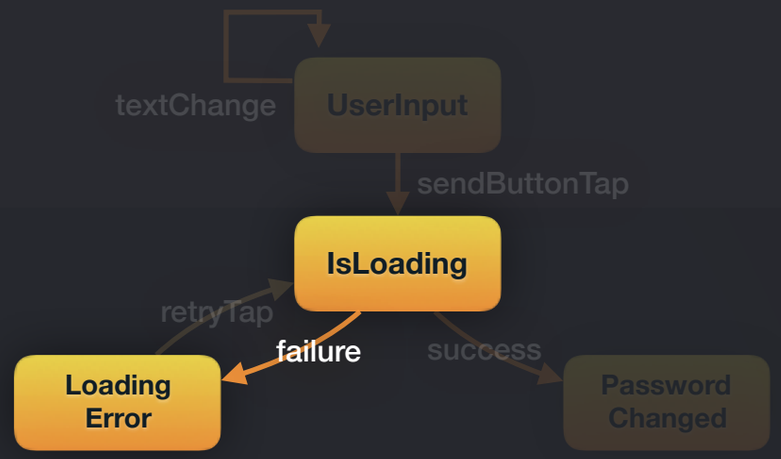


Смена состояния при ошибке ответа сервера:

1. При наступлении события захватываем текущее состояние.
Используем для этого оператор withLatestFrom

2. Событие допускается только в состоянии isLoading. В ином случае игнорируем его.
Проверяем это с помощью оператора filter.

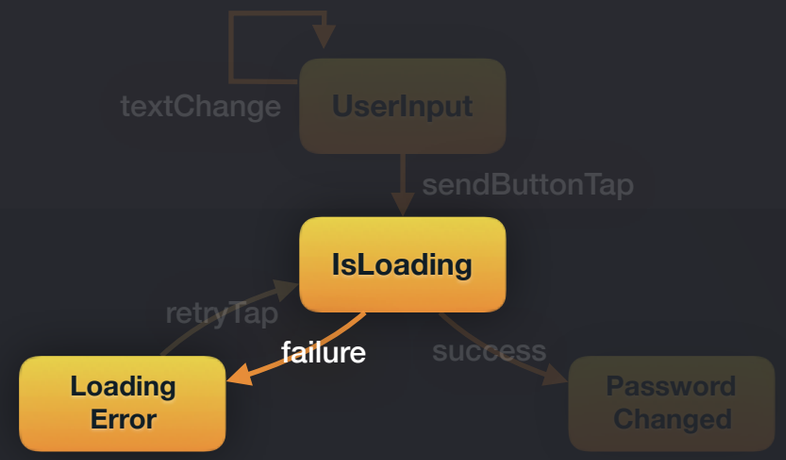
```swift
/// - Переход из состояния 'loading' в 'loadingError' (когда загрузка заврешилась ошибкой)
static func fromLoadingToLoadingError(state readonlyState: Observable<ChangePasswordInteractorState>,
                                      passwordChangingRequestFailed: Observable<Error>,
                                      bindoTo _state: BehaviorRelay<ChangePasswordInteractorState>,
                                      disposedBy disposeBag: DisposeBag) {
    let fromLoadingToLoadingError = passwordChangingRequestFailed
        .withLatestFrom(readonlyState, resultSelector: latestFromBothValues())
        .filter { _, state in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }
        .map { error, _ in State.loadingError(error) }

    fromLoadingToLoadingError.bind(to: _state).disposed(by: disposeBag)
}
```



Смена состояния при ошибке ответа сервера:

1. *При наступлении события захватываем текущее состояние. Используем для этого оператор withLatestFrom*
2. *Событие допускается только в состоянии isLoading. В ином случае игнорируем его. Проверяем это с помощью оператора filter.*
3. *Создаём состояние loadingError. Используем оператор map*

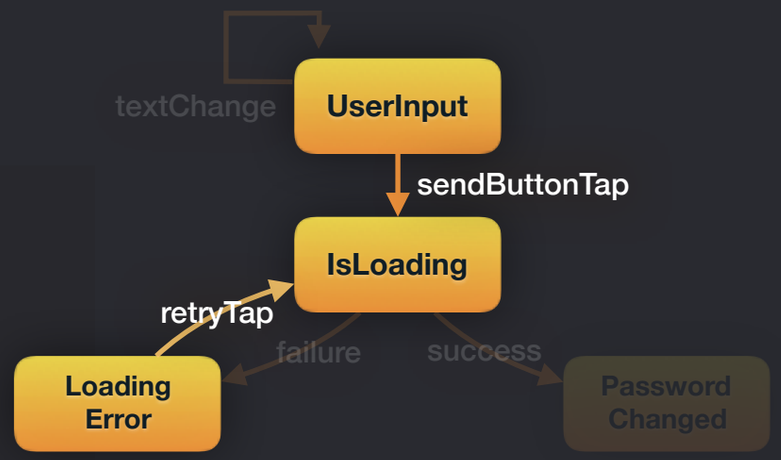
```swift
/// - Переход из состояния 'loading' в 'loadingError' (когда загрузка заврешилась ошибкой)
static func fromLoadingToLoadingError(state readonlyState: Observable<ChangePasswordInteractorState>,
                                      passwordChangingRequestFailed: Observable<Error>,
                                      bindoTo _state: BehaviorRelay<ChangePasswordInteractorState>,
                                      disposedBy disposeBag: DisposeBag) {
    let fromLoadingToLoadingError = passwordChangingRequestFailed
        .withLatestFrom(readonlyState, resultSelector: latestFromBothValues())
        .filter { _, state in
            switch state {
            case .isLoading: return true
            default: return false
            }
        }
        .map { error, _ in State.loadingError(error) }

    fromLoadingToLoadingError.bind(to: _state).disposed(by: disposeBag)
}
```



Смена состояния при ошибке ответа сервера:

1. *При наступлении события захватываем текущее состояние. Используем для этого оператор withLatestFrom*
2. *Событие допускается только в состоянии isLoading. В ином случае игнорируем его. Проверяем это с помощью оператора filter.*
3. *Создаём состояние loadingError. Используем оператор map*
4. *Обновляем текущее состояние. Используем для этого bind(to:)*

```swift
/** Переход из состояния 'userInput' в 'loading'. При входе в состояние 'loading' запускается
  деятельность (отправка запроса на сервер) */
static func transiotionToLoading(readonlyState: Observable<ChangePasswordInteractorState>,
                                 didTapSendButton: ControlEvent<Void>,
                                 bindoTo _state: BehaviorRelay<ChangePasswordInteractorState>,
                                 disposedBy disposeBag: DisposeBag,
                                 stateEntryAction: @escaping (_ newPassword: String) -> Void) {
    let fromUserInputToLoading = didTapSendButton
        .withLatestFrom(readonlyState)
        .map { state -> String? in
            // Запрос на смену пароля можно сделать только в состоянии userInput и если строка валидна
            switch state {
            case let .userInput(password, isValid): return isValid ? password : nil
            default: return nil
            }
        }
        .compactMap()
        .share()
    /* Если не делать share(), то сначала нужно вызвать stateEntryAction(newPassword), и только
       после этого .bind(to: _state). Если в обратном порядке, то без оператора share() замыкание
       tateEntryAction(newPassword) вызвано не будет. */

        fromUserInputToLoading
            .map { _ in State.isLoading }
            .bind(to: _state).disposed(by: disposeBag)

        fromUserInputToLoading.subscribe(onNext: { newPassword in
            // Запускаем деятельность при входе в состояние 'isLoading'
            stateEntryAction(newPassword)
        }).disposed(by: disposeBag)
    }
}
```



Аналитика

```
final class ChangePasswordAnalytics: RibBaseAnalytics {
    private let disposeBag = DisposeBag()

    init(viewOutput: ChangePasswordViewOutput,
         interactorOutput: Observable<ChangePasswordInteractorState>,
         presenterOutput: ChangePasswordPresenterOutput,
         lifeCycleObservable: ViewControllerLifeCycleObservable) {
        super.init(lifeCycleObservable: lifeCycleObservable)

        bindEvents(interactorState: interactorOutput, viewOutput: viewOutput)
    }

    private func bindEvents(interactorState: Observable<ChangePasswordInteractorState>,
                            viewOutput: ChangePasswordViewOutput) {

        interactorState.subscribe(onNext: { [weak self] state in
            switch state {
            case .userInput, .isLoading:
                break
            case .loadingError(let error):
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changeFail,
                                                 error: error)
            case .passwordChanged:
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changePassSuccess)
            }
        }).disposed(by: disposeBag)

        viewOutput.didTapSendButton.subscribe(onNext: { [weak self] in
            self?.analyticalService.logEvent(screen: AnalyticalSpace.Auth.changePass,
                                             event: AnalyticalEvent.Tap.Auth.buttonApply)
        }).disposed(by: disposeBag)
    }
```

Аналитика

```
final class ChangePasswordAnalytics: RibBaseAnalytics {
    private let disposeBag = DisposeBag()

    init(viewOutput: ChangePasswordViewOutput,
         interactorOutput: Observable<ChangePasswordInteractorState>,
         presenterOutput: ChangePasswordPresenterOutput,
         lifeCycleObservable: ViewControllerLifeCycleObservable) {
        super.init(lifeCycleObservable: lifeCycleObservable)

        bindEvents(interactorState: interactorOutput, viewOutput: viewOutput)
    }

    private func bindEvents(interactorState: Observable<ChangePasswordInteractorState>,
                            viewOutput: ChangePasswordViewOutput) {

        interactorState.subscribe(onNext: { [weak self] state in
            switch state {
            case .userInput, .isLoading:
                break
            case .loadingError(let error):
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changeFail,
                                                 error: error)
            case .passwordChanged:
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changePassSuccess)
            }
        }).disposed(by: disposeBag)

        viewOutput.didTapSendButton.subscribe(onNext: { [weak self] in
            self?.analyticalService.logEvent(screen: AnalyticalSpace.Auth.changePass,
                                             event: AnalyticalEvent.Tap.Auth.buttonApply)
        }).disposed(by: disposeBag)
    }
```

Аналитика

```
final class ChangePasswordAnalytics: RibBaseAnalytics {
    private let disposeBag = DisposeBag()

    init(viewOutput: ChangePasswordViewOutput,
         interactorOutput: Observable<ChangePasswordInteractorState>,
         presenterOutput: ChangePasswordPresenterOutput,
         lifeCycleObservable: ViewControllerLifeCycleObservable) {
        super.init(lifeCycleObservable: lifeCycleObservable)

        bindEvents(interactorState: interactorOutput, viewOutput: viewOutput)
    }

    private func bindEvents(interactorState: Observable<ChangePasswordInteractorState>,
                            viewOutput: ChangePasswordViewOutput) {

        interactorState.subscribe(onNext: { [weak self] state in
            switch state {
            case .userInput, .isLoading:
                break
            case .loadingError(let error):
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changeFail,
                                                 error: error)
            case .passwordChanged:
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changePassSuccess)
            }
        }).disposed(by: disposeBag)

        viewOutput.didTapSendButton.subscribe(onNext: { [weak self] in
            self?.analyticalService.logEvent(screen: AnalyticalSpace.Auth.changePass,
                                             event: AnalyticalEvent.Tap.Auth.buttonApply)
        }).disposed(by: disposeBag)
    }
```

Аналитика

```
final class ChangePasswordAnalytics: RibBaseAnalytics {
    private let disposeBag = DisposeBag()

    init(viewOutput: ChangePasswordViewOutput,
         interactorOutput: Observable<ChangePasswordInteractorState>,
         presenterOutput: ChangePasswordPresenterOutput,
         lifeCycleObservable: ViewControllerLifeCycleObservable) {
        super.init(lifeCycleObservable: lifeCycleObservable)

        bindEvents(interactorState: interactorOutput, viewOutput: viewOutput)
    }

    private func bindEvents(interactorState: Observable<ChangePasswordInteractorState>,
                            viewOutput: ChangePasswordViewOutput) {
        interactorState.subscribe(onNext: { [weak self] state in
            switch state {
            case .userInput, .isLoading:
                break
            case .loadingError(let error):
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changeFail,
                                                 error: error)
            case .passwordChanged:
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changePassSuccess)
            }
        }).disposed(by: disposeBag)

        viewOutput.didTapSendButton.subscribe(onNext: { [weak self] in
            self?.analyticalService.logEvent(screen: AnalyticalSpace.Auth.changePass,
                                             event: AnalyticalEvent.Tap.Auth.buttonApply)
        }).disposed(by: disposeBag)
    }
```

Аналитика

```
final class ChangePasswordAnalytics: RibBaseAnalytics {
    private let disposeBag = DisposeBag()

    init(viewOutput: ChangePasswordViewOutput,
         interactorOutput: Observable<ChangePasswordInteractorState>,
         presenterOutput: ChangePasswordPresenterOutput,
         lifeCycleObservable: ViewControllerLifeCycleObservable) {
        super.init(lifeCycleObservable: lifeCycleObservable)

        bindEvents(interactorState: interactorOutput, viewOutput: viewOutput)
    }

    private func bindEvents(interactorState: Observable<ChangePasswordInteractorState>,
                            viewOutput: ChangePasswordViewOutput) {

        interactorState.subscribe(onNext: { [weak self] state in
            switch state {
            case .userInput, .isLoading:
                break
            case .loadingError(let error):
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changeFail,
                                                 error: error)
            case .passwordChanged:
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changePassSuccess)
            }
        }).disposed(by: disposeBag)

        viewOutput.didTapSendButton.subscribe(onNext: { [weak self] in
            self?.analyticalService.logEvent(screen: AnalyticalSpace.Auth.changePass,
                                             event: AnalyticalEvent.Tap.Auth.buttonApply)
        }).disposed(by: disposeBag)
    }
```

Аналитика

```
final class ChangePasswordAnalytics: RibBaseAnalytics {
    private let disposeBag = DisposeBag()

    init(viewOutput: ChangePasswordViewOutput,
         interactorOutput: Observable<ChangePasswordInteractorState>,
         presenterOutput: ChangePasswordPresenterOutput,
         lifeCycleObservable: ViewControllerLifeCycleObservable) {
        super.init(lifeCycleObservable: lifeCycleObservable)

        bindEvents(interactorState: interactorOutput, viewOutput: viewOutput)
    }

    private func bindEvents(interactorState: Observable<ChangePasswordInteractorState>,
                            viewOutput: ChangePasswordViewOutput) {

        interactorState.subscribe(onNext: { [weak self] state in
            switch state {
            case .userInput, .isLoading:
                break
            case .loadingError(let error):
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changeFail,
                                                 error: error)
            case .passwordChanged:
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changePassSuccess)
            }
        }).disposed(by: disposeBag)

        viewOutput.didTapSendButton.subscribe(onNext: { [weak self] in
            self?.analyticalService.logEvent(screen: AnalyticalSpace.Auth.changePass,
                                             event: AnalyticalEvent.Tap.Auth.buttonApply)
        }).disposed(by: disposeBag)
    }
```

Аналитика

```
final class ChangePasswordAnalytics: RibBaseAnalytics {
    private let disposeBag = DisposeBag()

    init(viewOutput: ChangePasswordViewOutput,
         interactorOutput: Observable<ChangePasswordInteractorState>,
         presenterOutput: ChangePasswordPresenterOutput,
         lifeCycleObservable: ViewControllerLifeCycleObservable) {
        super.init(lifeCycleObservable: lifeCycleObservable)

        bindEvents(interactorState: interactorOutput, viewOutput: viewOutput)
    }

    private func bindEvents(interactorState: Observable<ChangePasswordInteractorState>,
                            viewOutput: ChangePasswordViewOutput) {

        interactorState.subscribe(onNext: { [weak self] state in
            switch state {
            case .userInput, .isLoading:
                break
            case .loadingError(let error):
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changeFail,
                                                 error: error)
            case .passwordChanged:
                self?.analyticalService.logEvent(prefixScreen: AnalyticalSpace.Auth.none,
                                                 screen: AnalyticalSpace.Auth.changePass,
                                                 event: AnalyticalEvent.Event.Auth.changePassSuccess)
            }
        }).disposed(by: disposeBag)

        viewOutput.didTapSendButton.subscribe(onNext: { [weak self] in
            self?.analyticalService.logEvent(screen: AnalyticalSpace.Auth.changePass,
                                             event: AnalyticalEvent.Tap.Auth.buttonApply)
        }).disposed(by: disposeBag)
    }
}
```

Аналитика ScrollView

Отправка событий при просмотре рекламных банеров

Аналитика ScrollView

```
struct CashbackOffersViewOutput {
  let didTapRetry: ControlEvent<Void>
  let didTapSearch: ControlEvent<Void>
  let didSelectPartner: ControlEvent<UInt64>
  let didSelectCategory: ControlEvent<UInt64>

  /// Используется только классом, в котором находится вся аналитика экрана
  let analytics: Analytics

  struct Analytics {
    /// Событие отправляется, если рекламная картинка была на экране дольше 3 секунд
    let promoImageWasShown: ControlEvent<UInt64>
    /// В качестве параметра передаём id рекламного видео, которое посмотрел пользователь
    let promoVideoWasShown: ControlEvent<UInt64>
  }
}
```

Цели

- ☑ Составление диаграмм состояний экранов
- ☑ Получение более полного и точного описания требований на этапе моделирования
- ☑ Минимизировать проблему непредвиденного поведения экранов
- ☑ Сделать код более устойчивым к внесению правок и доработок
- ☐ Минимизировать нарушение SRP принципа
- ☐ Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- ☐ Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- ☑ Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

Цели

- ☑ Составление диаграмм состояний экранов
- ☑ Получение более полного и точного описания требований на этапе моделирования
- ☑ Минимизировать проблему непредвиденного поведения экранов
- ☑ Сделать код более устойчивым к внесению правок и доработок
- ☑ Минимизировать нарушение SRP принципа
- ☐ Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- ☐ Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- ☑ Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

Цели

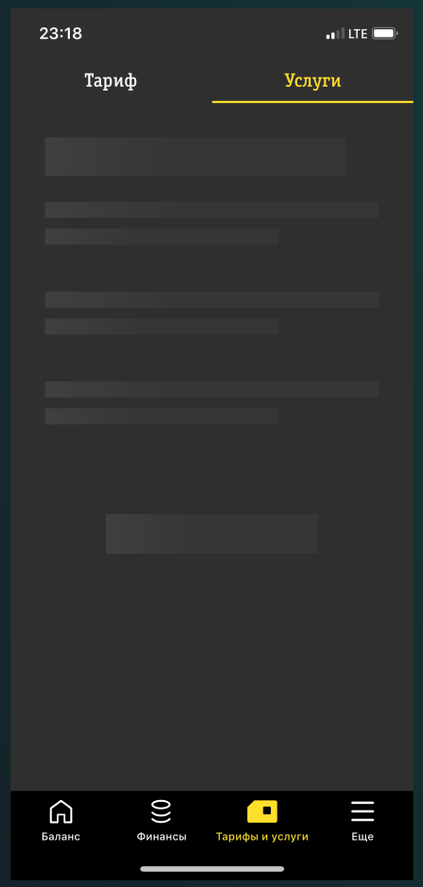
- ☑ Составление диаграмм состояний экранов
- ☑ Получение более полного и точного описания требований на этапе моделирования
- ☑ Минимизировать проблему непредвиденного поведения экранов
- ☑ Сделать код более устойчивым к внесению правок и доработок
- ☑ Минимизировать нарушение SRP принципа
- ☑ Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- ☐ Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- ☑ Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

Цели

- ☑ Составление диаграмм состояний экранов
- ☑ Получение более полного и точного описания требований на этапе моделирования
- ☑ Минимизировать проблему непредвиденного поведения экранов
- ☑ Сделать код более устойчивым к внесению правок и доработок
- ☑ Минимизировать нарушение SRP принципа
- ☑ Стандартизировать и упростить работу с VIPER / VIP / RIB модулями
- ☑ Декларативный стиль вместо императивного: фокус на том «что» делает экран, а не «как»
- ☑ Более чистый код: убрать проксирование вызовов, излишние опционалы, уменьшить кол-во повторяющихся конструкций кода

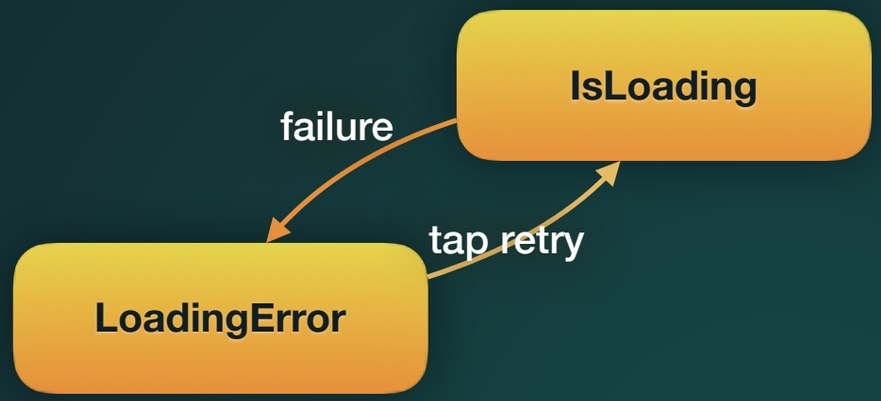
Вложенное состояние

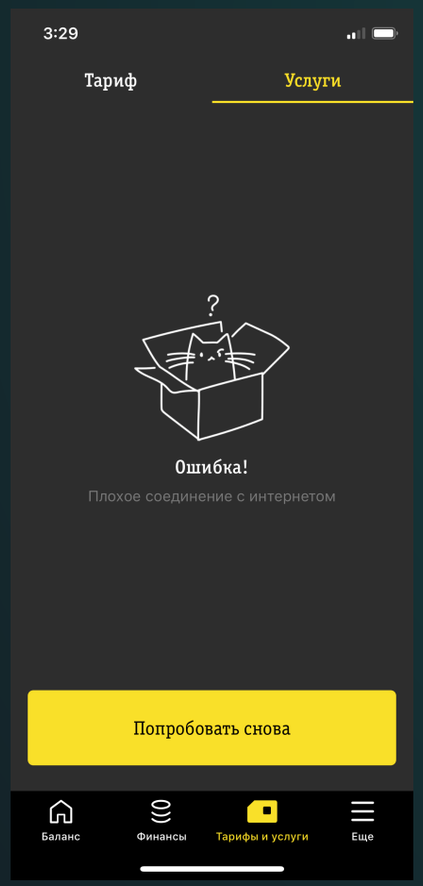
IsLoading



Подгрузка секции с данными

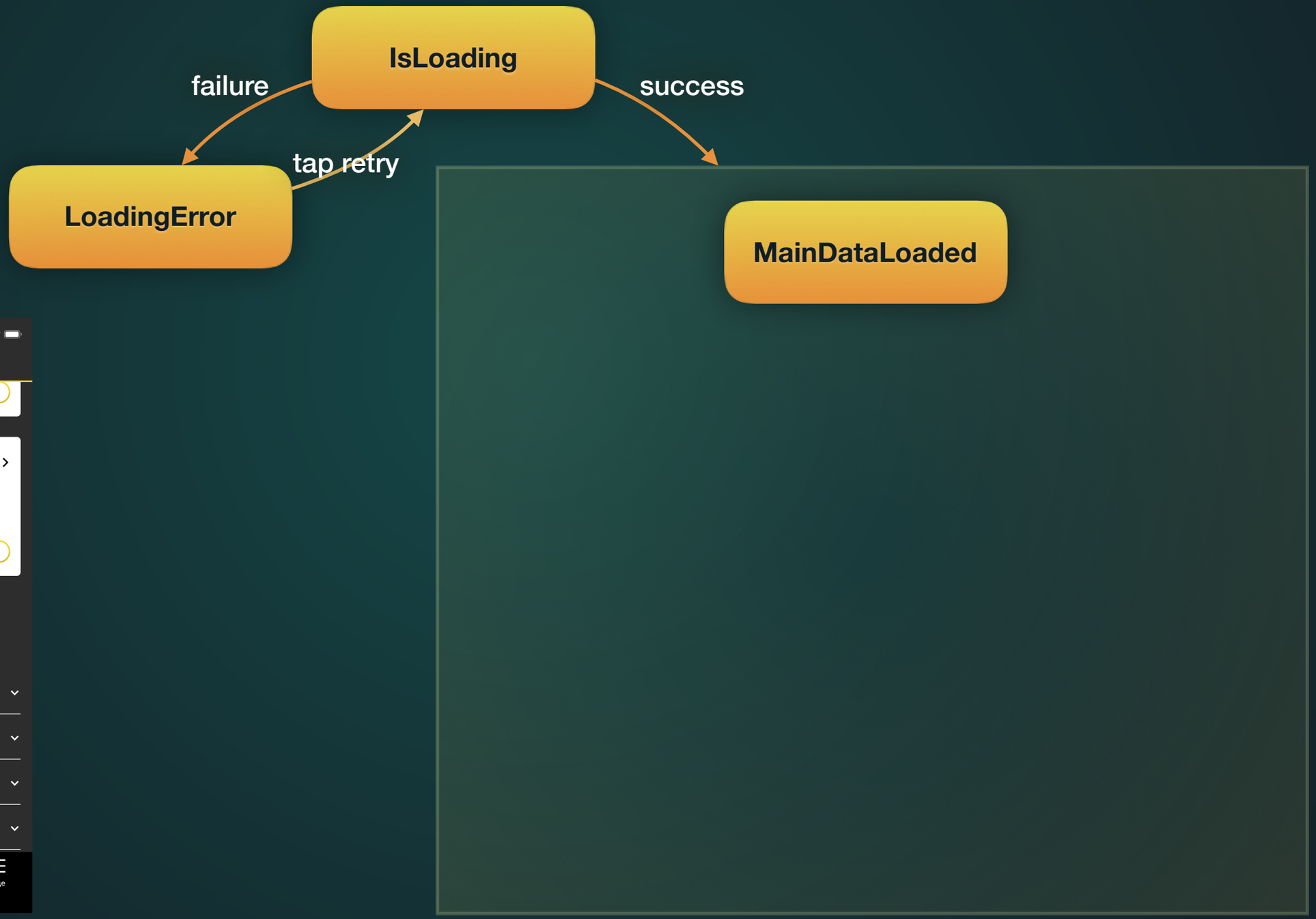
Вложенное состояние

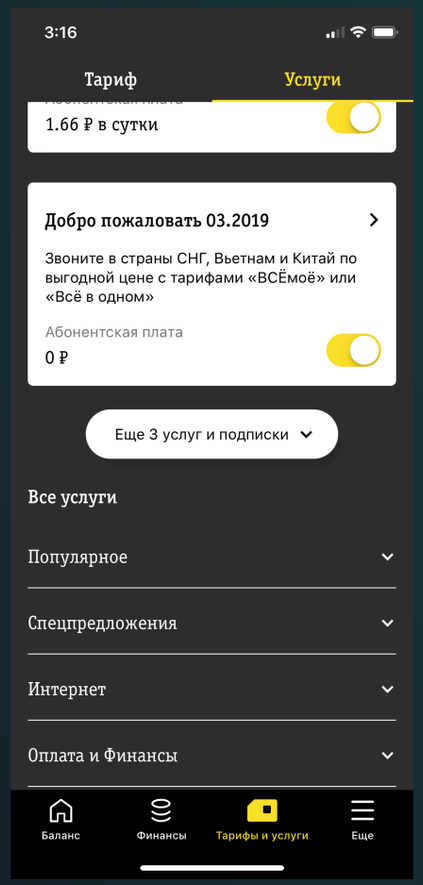




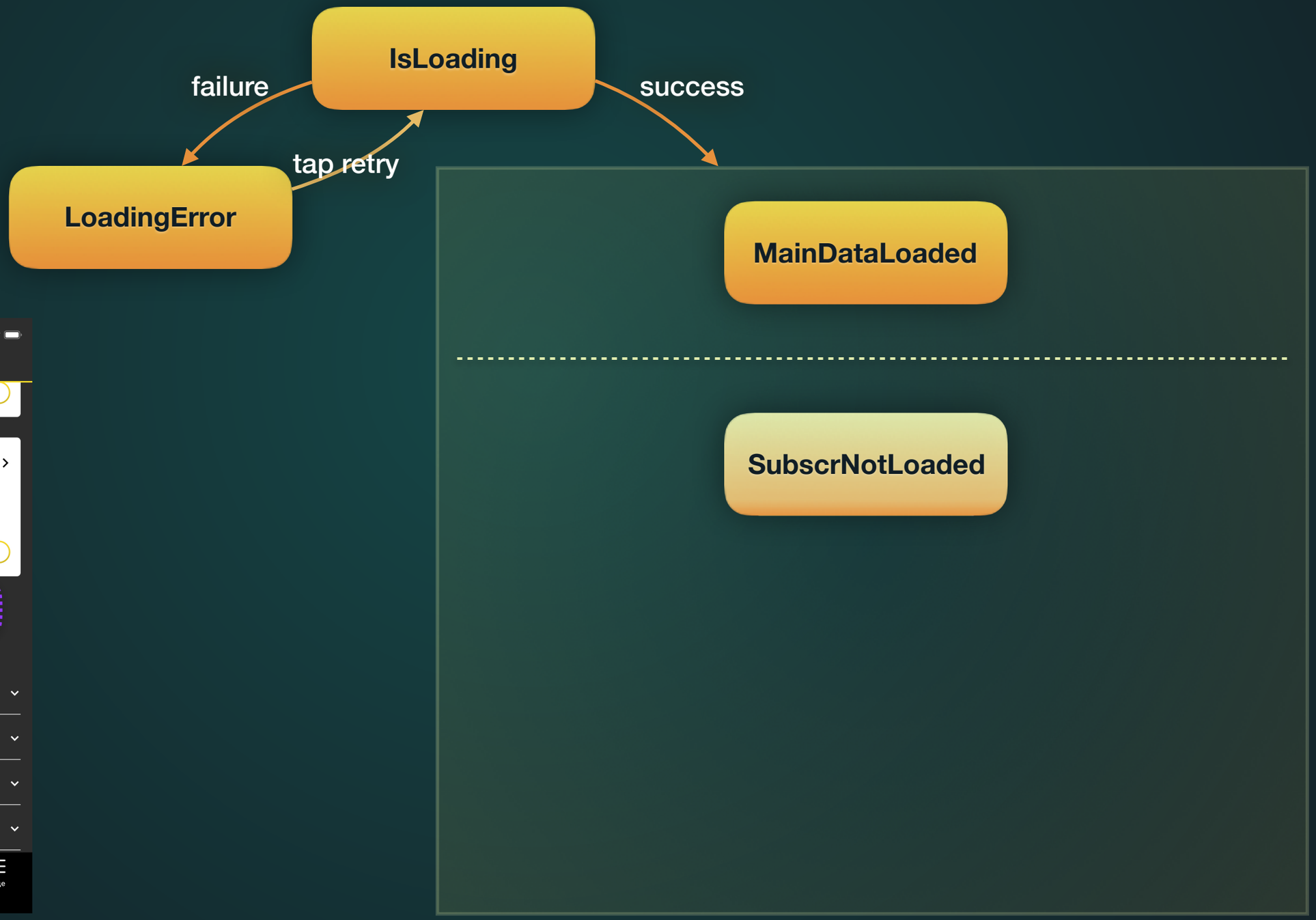
Подгрузка секции с данными

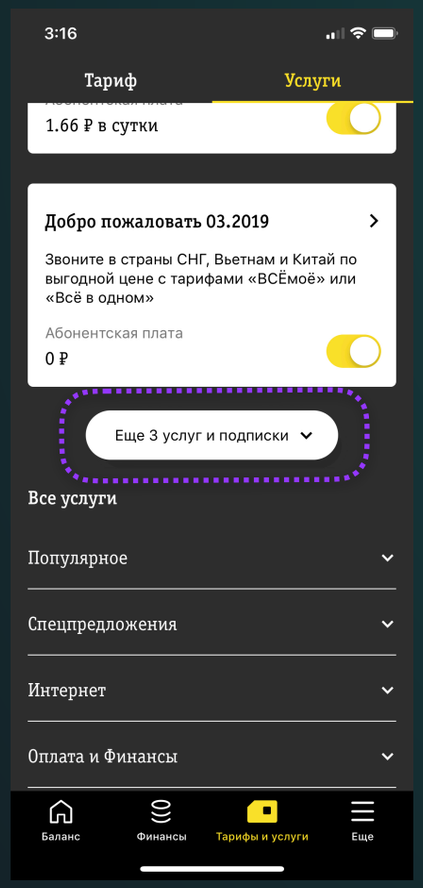
Вложенное состояние



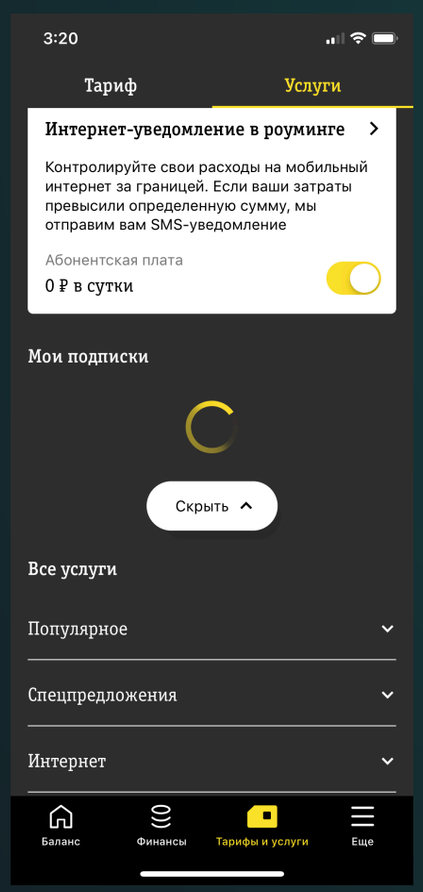


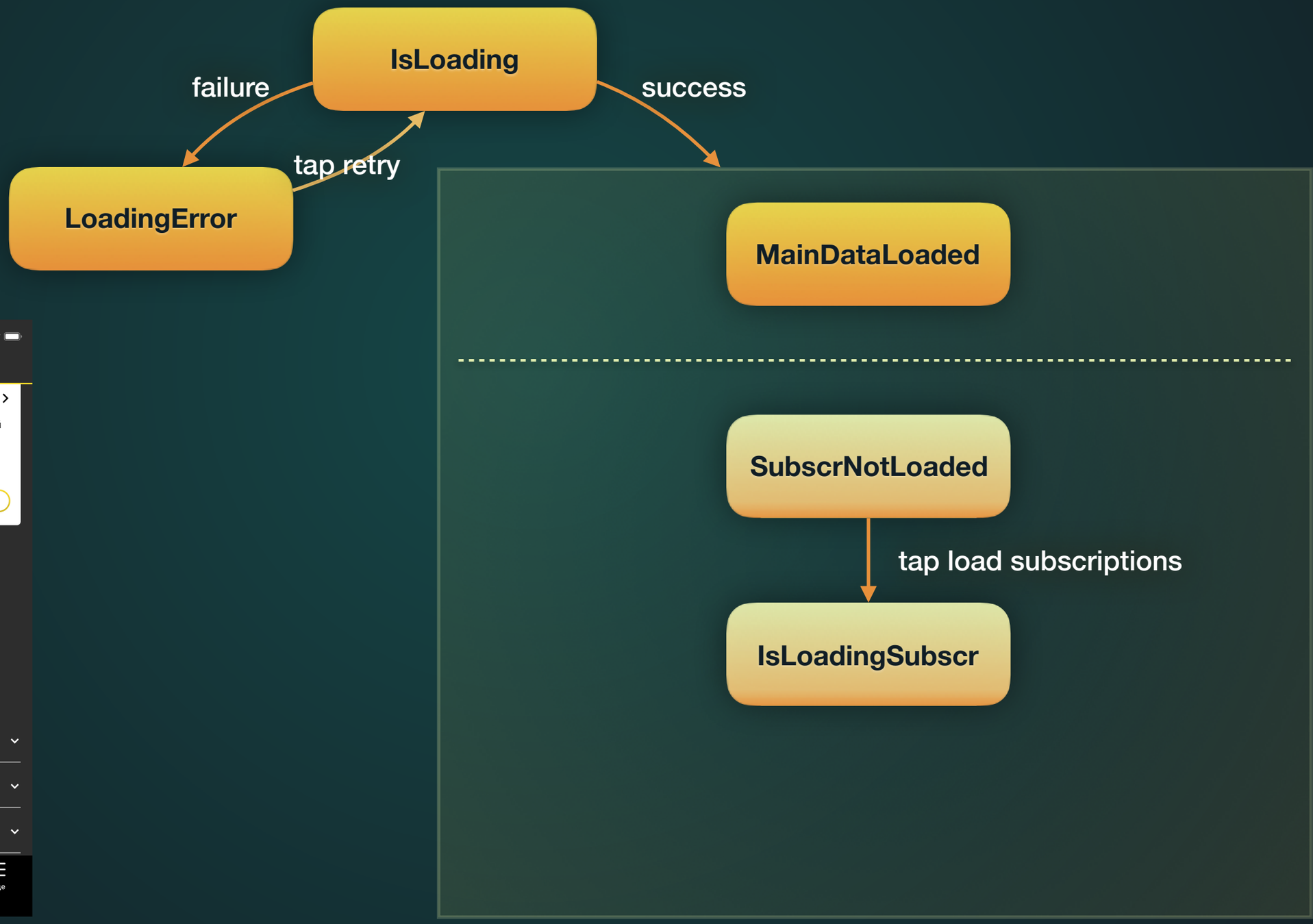
Вложенное состояние



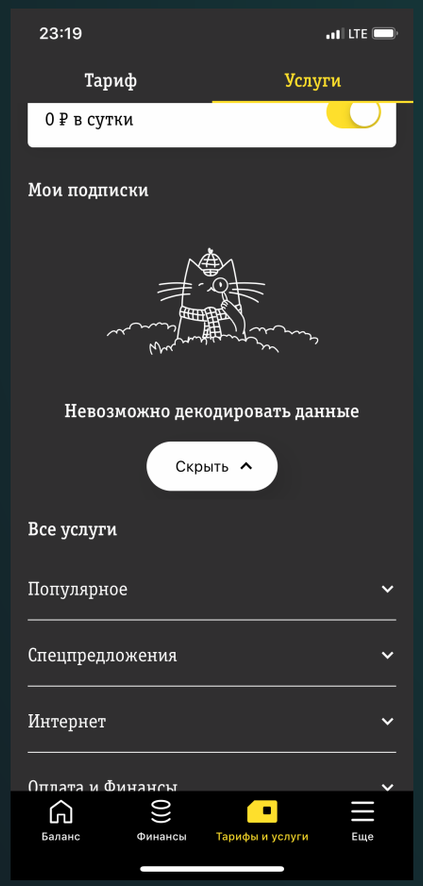


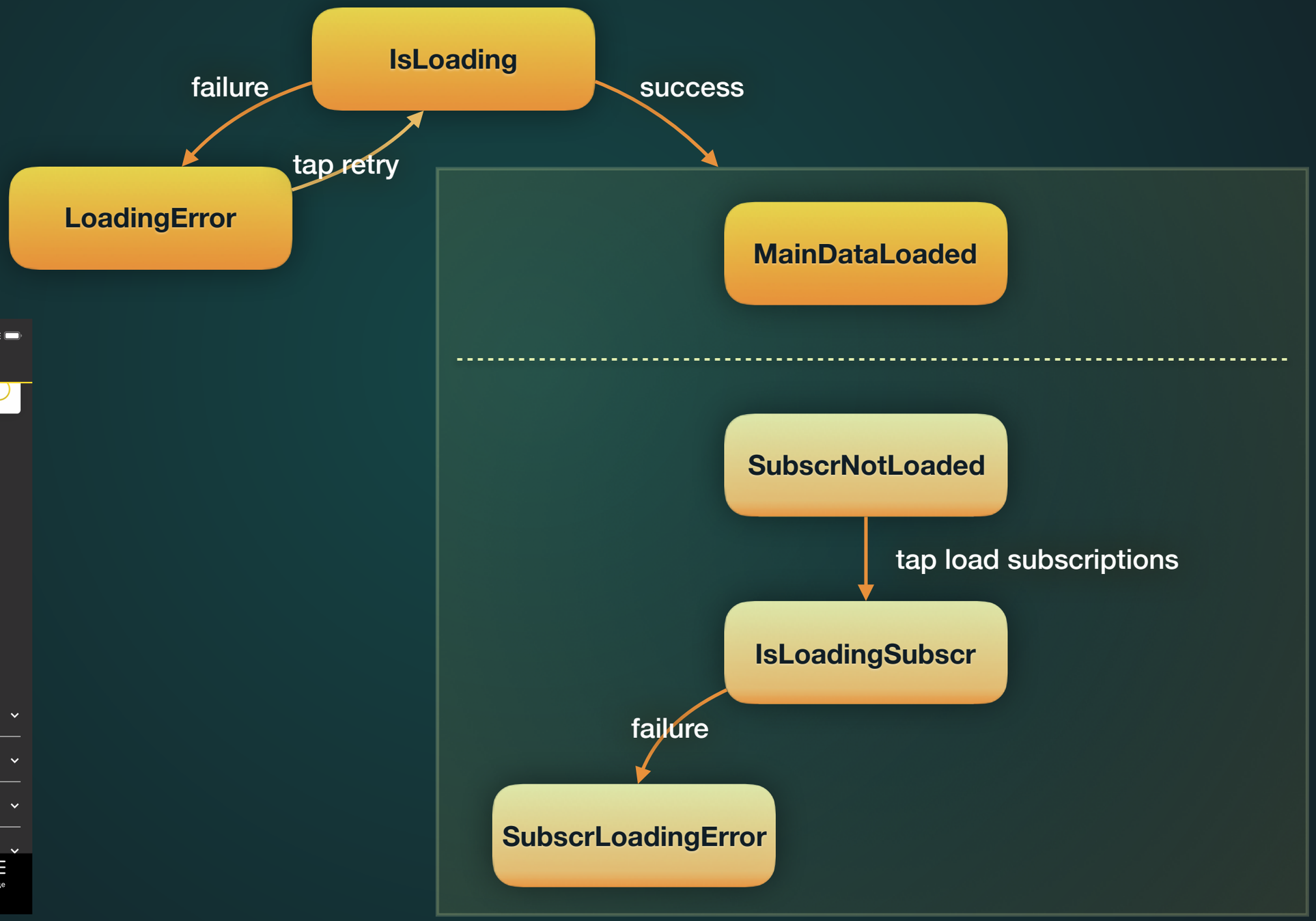
Вложенное состояние



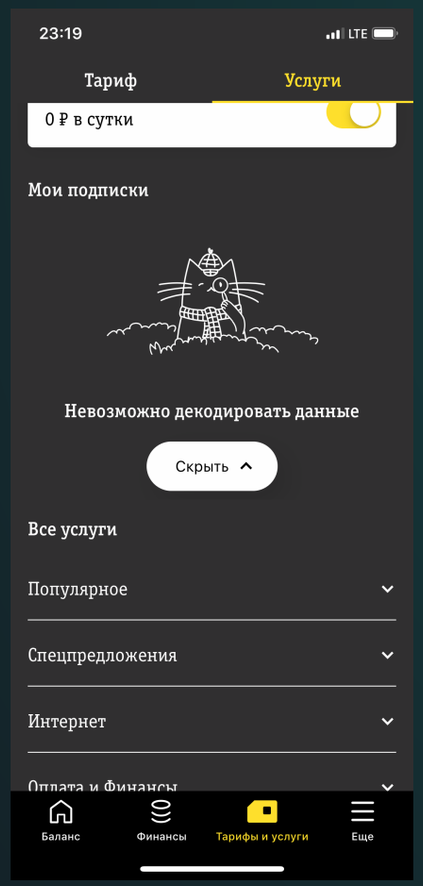


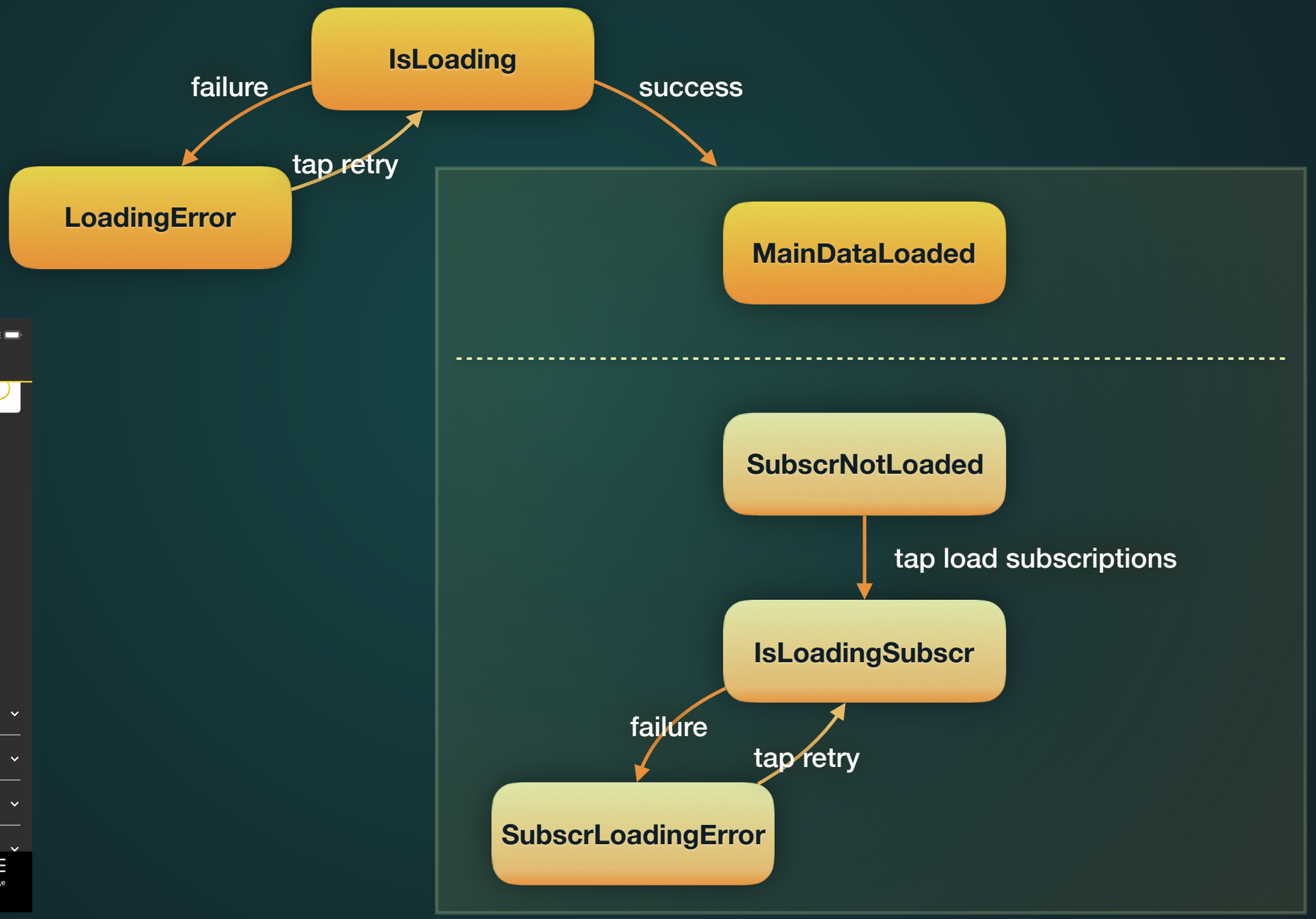
Вложенное состояние



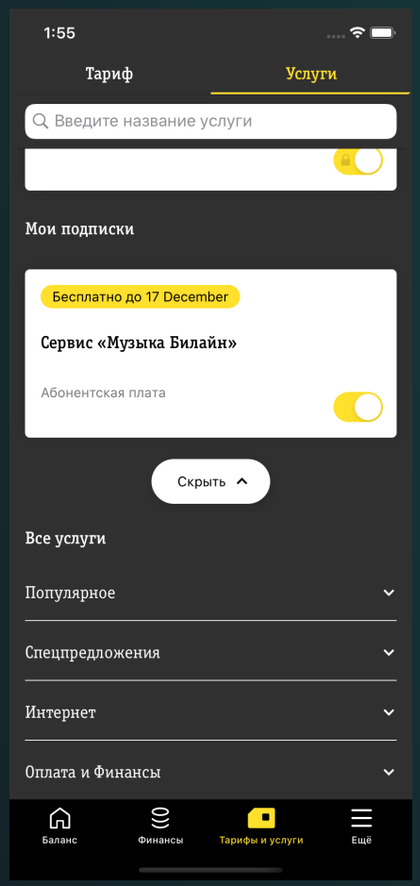


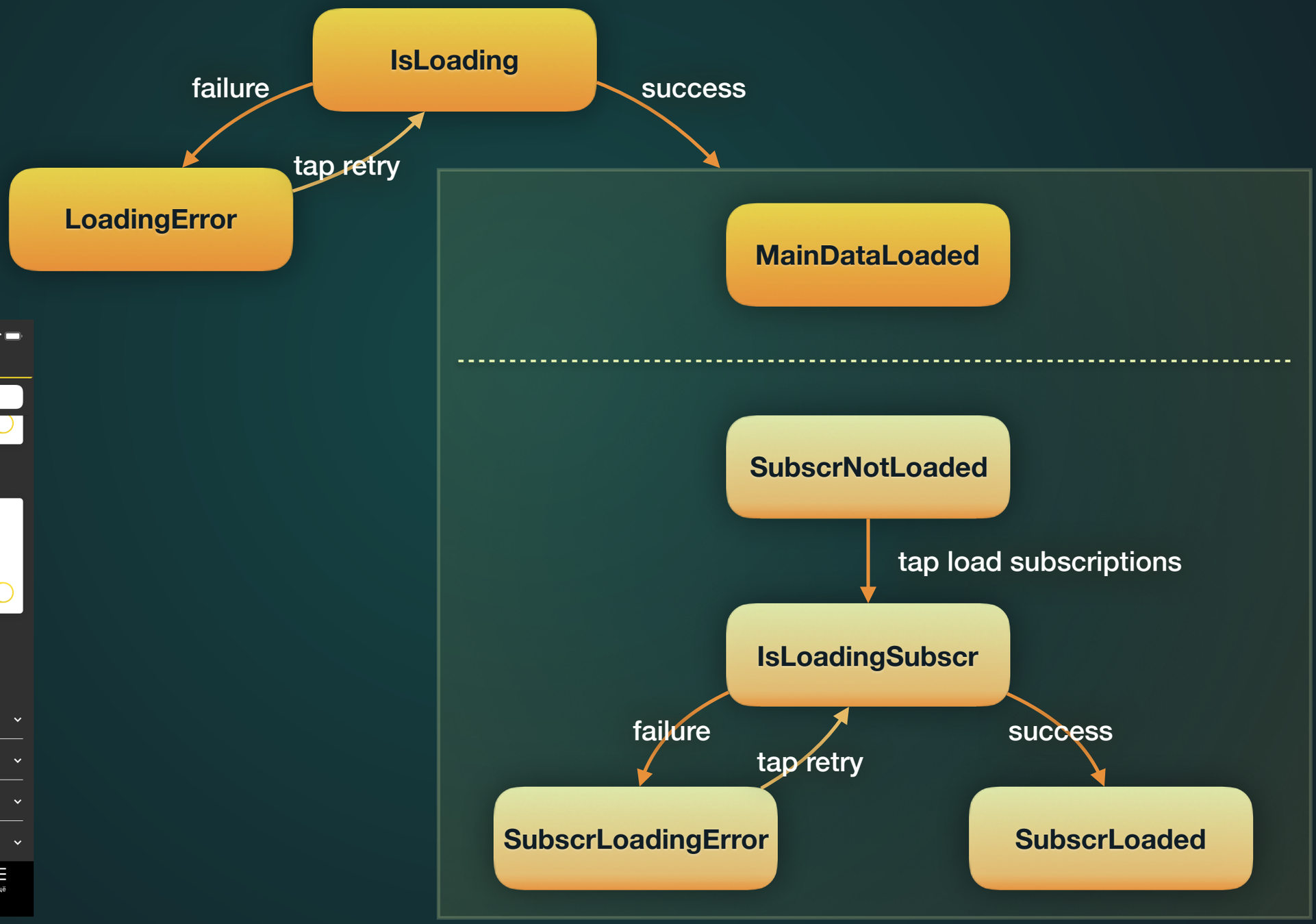
Вложенное состояние





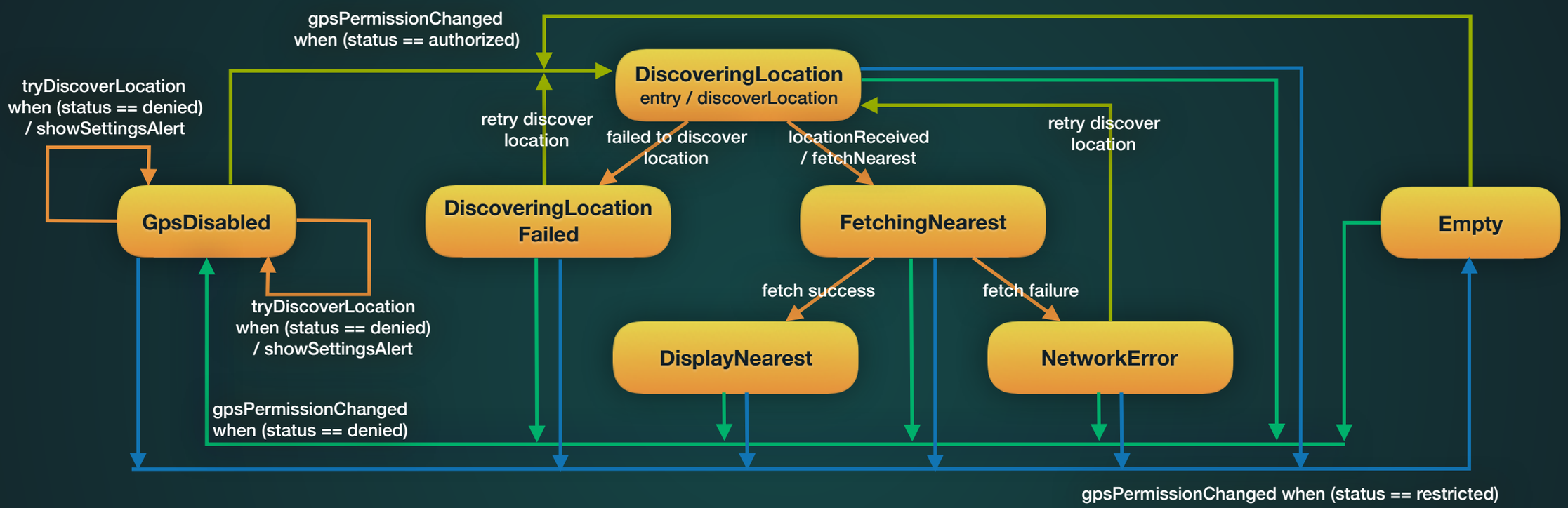
Вложенное состояние





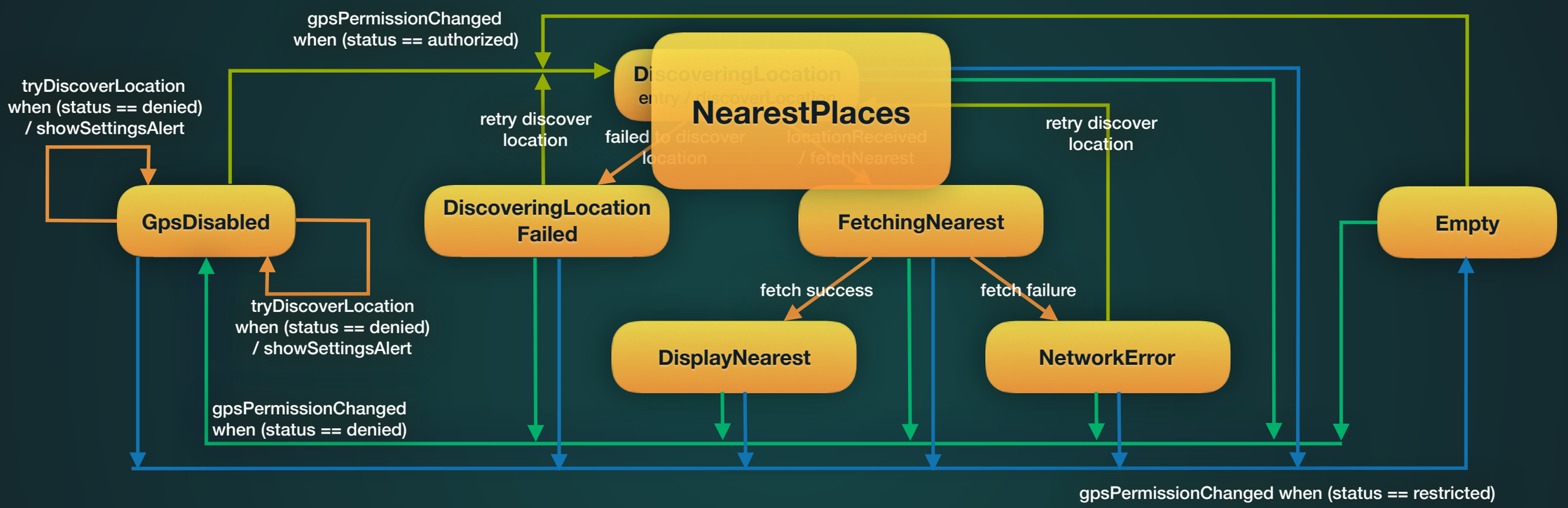
Hardcore

Hardcore



Загрузка ближайших <...>

Hardcore



Загрузка ближайших <...>

Hardcore

NearestPlaces

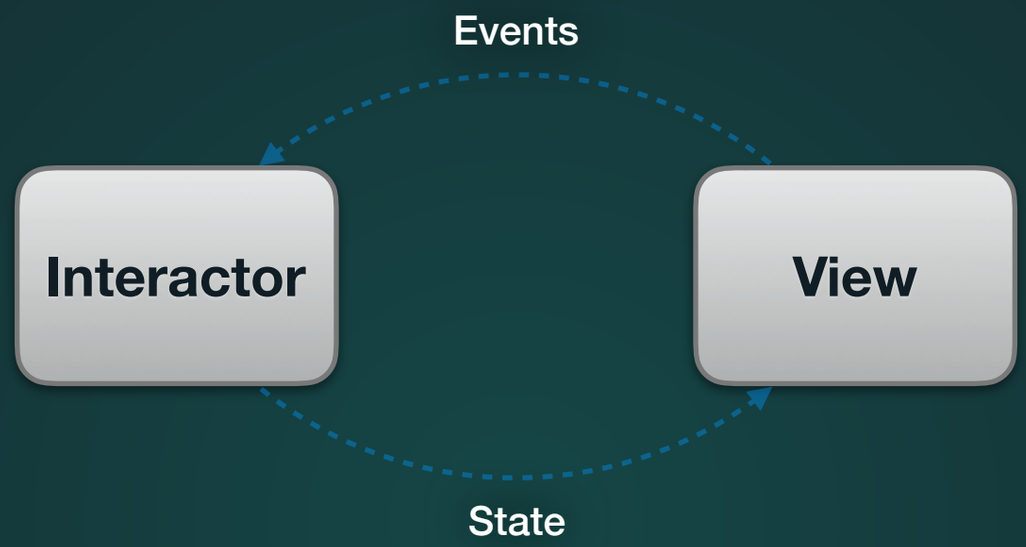
Загрузка ближайших <...>

Hardcore

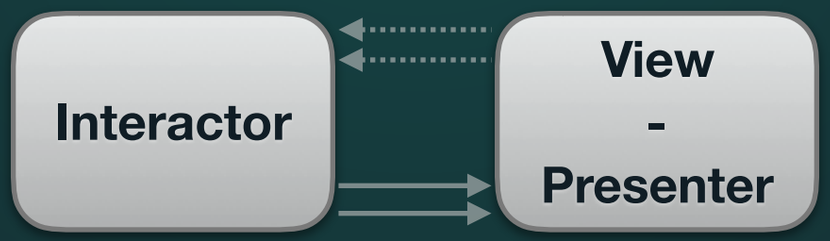
Загрузка ближайших <...>

WebView

Presenter не имеет смысла – экран с WebView

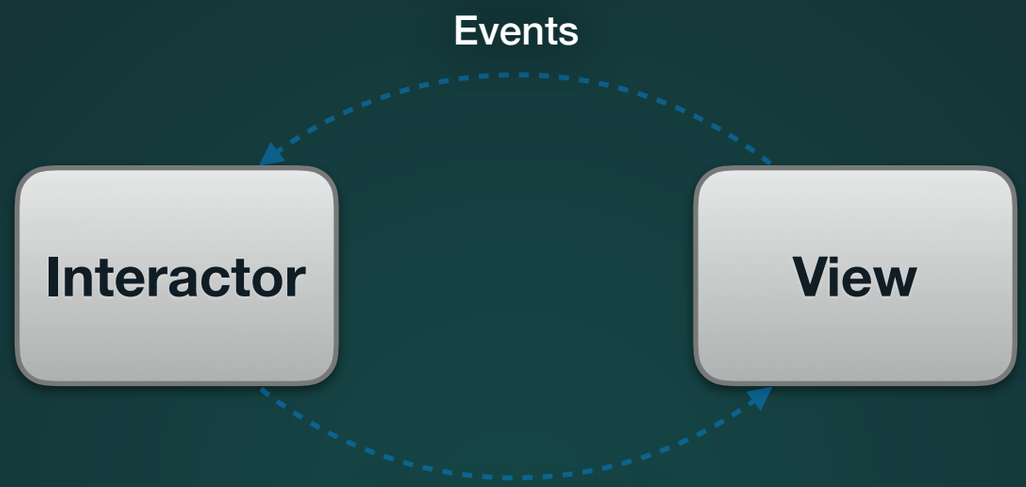


Несоблюдение SRP принципа

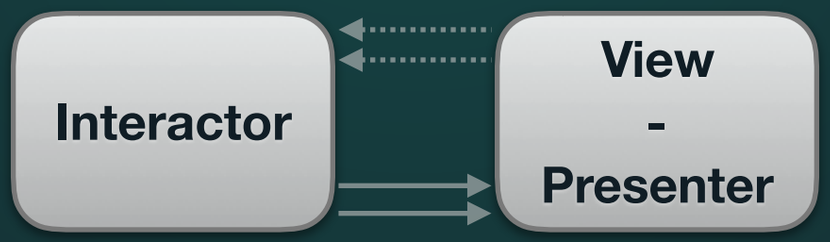


WebView

State не имеет смысла –
экран с WebView



Несоблюдение SRP принципа



Примеры

Interactor

```
enum QuizMoreScoreInteractorState {
    case dataLoaded(_ quizScore: Int)
    case isLoading
    case loadingError(Error)
}
```

```
public enum ProfileInteractorState {
    case isLoading
    case dataLoaded(ProfileInfo)
    case loadingError(Error)
}

struct ProfileInteractorOutput {
    let state: Observable<ProfileInteractorState>
    let showLogoutDialog: Observable<Void>
}
```

```
public enum CardListInteractorState {
    case isLoading
    case dataLoaded(cards: [PaymentCard], isB2b: Bool)
    case loadingError(Error)
}

struct CardListInteractorOutput {
    let state: Observable<CardListInteractorState>

    let isCardInfoLoaderVisible: Observable<Bool>
    let showCardInfoError: Observable<String>
}
```

Presenter

```
struct QuizMoreScorePresenterOutput {
    let loadingIndicatorVisible: Driver<Bool>
    let info: Info
    let showError: Signal<SubscrZeroScreenViewModel?>
    let infoViewVisible: Driver<Bool>
}
```

```
struct ProfilePresenterOutput {
    let contentVisible: Driver<Bool>
    let loadingIndicatorVisible: Driver<Bool>
    let showError: Signal<String?>
    let confirmedEmail: Driver<Bool>
    let profileInfo: Driver<ProfileVM>
    let showLogoutDialog: Signal<Void>
}
```

```
struct CardListPresenterOutput {
    let cardsList: Driver<CardsList>

    let isSkeletonLoaderVisible: Driver<Bool>
    let isCardInfoLoaderVisible: Driver<Bool>
    let showError: Signal<String>

}
```

View

```
struct QuizMoreScoreViewControllerOutput {
    let viewWillAppear: ControlEvent<Void>
    let didTapRetryButton: ControlEvent<Void>
    let didTapCloseScreenButton: ControlEvent<Void>
    let didTapMoreScoreButton: ControlEvent<Void>
}
```

```
struct ProfileViewOutput {
    let retryTap: ControlEvent<Void>
    let logoutButtonTap: ControlEvent<Void>
    let logoutActionSheetItemTap: ControlEvent<Void>
    let paymentFeedbackTap: ControlEvent<Void>
    let addEmailTap: ControlEvent<Void>
    let manageEmailTap: ControlEvent<Void>
    let blockingNumberTap: ControlEvent<Bool>
    let notificationTap: ControlEvent<Void>
}
```

```
struct CardListViewOutput {
    let addCardButtonTap: ControlEvent<Void>
    let selectVirtualCard: ControlEvent<Void>
    let selectCard: ControlEvent<PaymentCardViewModel>
    let pullToRefresh: ControlEvent<Void>
}
```

Ещё примеры

Interactor

```
enum MainQuizInteractorState {
    case isLoading
    case dataLoaded(mainQuiz: MainQuiz)
    case loadingError(Error)
}
struct MainQuizInteractorOutput {
    let mainState: Observable<MainQuizInteractorState>
    let subscriptionState: Observable<QuizSubscriptionState?>
    let showError: Observable<Error?>
}
```

Presenter

```
struct MainQuizPresenterOutput {
    let infoViewVisible: Driver<Bool>

    let initialLoadingIndicatorVisible: Driver<Bool>
    let hideRefreshControl: Signal<Void>

    let info: Info
    let showError: Signal<SubscrZeroScreenViewModel?>
    let showSubscriptionError: Signal<String?>
}
```

View

```
struct MainQuizViewControllerOutput {
    let didTapRetryButton: ControlEvent<Void>
    let didTapCloseButton: ControlEvent<Void>
    let pullToRefresh: ControlEvent<Void>
    let didTapSwitch: ControlEvent<Bool>

    let prizeChance: PrizeChanceViewOutput
    let replenishAccount: PointsForActionViewOutput
    let checkInBill: PointsForActionViewOutput
    let selectTariff: PointsForActionViewOutput
    let plugInAutopayment: PointsForActionViewOutput
    let quiz: PointsForPlayViewOutput
    let shake: PointsForPlayViewOutput
}
```

```
enum ContactsInteractorState {
    case empty

    case notDetermined
    case denied

    case isLoading

    case showContacts([CNContact])
    case error(Error)
    case restricted
}

struct ContactsInteractorOutput {
    let state: Observable<ContactsInteractorState>
    let searchText: Observable<String>
}
```

```
struct ContactsPresenterOutput {
    let loadingIndicatorVisible: Driver<Bool>
    let contentVisible: Driver<Bool>

    let showError: Signal<(kind: ErrorKind, message: String)?>

    let contacts: Driver<[ContactsPresenterOutput.ContactVM]>
}
```

```
struct ContactsViewOutput {
    let openSettingsTap: ControlEvent<Void>

    /// Выбранный номер телефона
    let didSelectContact: ControlEvent<String>

    let searchText: ControlEvent<String>
}
```

Тесты

Presenter: повторная загрузка данных

Тесты

```
func testErrorRetrying() {
  // MARK: arrange

  let time0: TestTime = 0
  let time1: TestTime = 60
  let time2: TestTime = 200
  let time3: TestTime = 250

  let loadingError = JsonMappingError(code: .nilData)

  let states: [Recorded<Event<CashbackPartnerCategoryInteractorState>>] = [
    .next(time0, State.isLoading),                          Загрузка
    .next(time1, State.loadingError(loadingError)),         Ошибка
    .next(time2, State.isLoading),                          Загрузка
    .next(time3, State.dataLoaded(partnersInfo)),           Успех
  ]

  let interactorState = scheduler.createColdObservable(states).asObservable()

  let intercatorOutput = CashbackPartnerCategoryInteractorOutput(state: interactorState,
                                                                 partnerDescription: Observable.empty())

  let presenter = CashbackPartnerCategoryPresenter()

  let presenterOutput = presenter.transform(intercatorOutput)
```

1. Мокируем состояния Interactor’а

Тесты

```
func testErrorRetrying() {
  // MARK: arrange

  let time0: TestTime = 0
  let time1: TestTime = 60
  let time2: TestTime = 200
  let time3: TestTime = 250

  let loadingError = JsonMappingError(code: .nilData)

  let states: [Recorded<Event<CashbackPartnerCategoryInteractorState>>] = [
    .next(time0, State.isLoading),                          Загрузка
    .next(time1, State.loadingError(loadingError)),         Ошибка
    .next(time2, State.isLoading),                          Загрузка
    .next(time3, State.dataLoaded(partnersInfo)),           Успех
  ]

  let interactorState = scheduler.createColdObservable(states).asObservable()

  let intercatorOutput = CashbackPartnerCategoryInteractorOutput(state: interactorState,
                                                                 partnerDescription: Observable.empty())

  let presenter = CashbackPartnerCategoryPresenter()

  let presenterOutput = presenter.transform(intercatorOutput)
```

1. Мокируем состояния Interactor'а

2. Биндим IntercatorOutput и Presenter

Тесты

```
let loadingIndicatorVisible = scheduler.createObserver(Bool.self)
let contentVisible = scheduler.createObserver(Bool.self)
let showError = scheduler.createObserver(String?.self)

let tags = scheduler.createObserver([FilterTag].self)
let partners = scheduler.createObserver([CBPartnerModel].self)
// let partnerDescription - в этом тесте не проверяем

presenterOutput.loadingIndicatorVisible.drive(loadingIndicatorVisible).disposed(by: disposeBag)
presenterOutput.contentVisible.drive(contentVisible).disposed(by: disposeBag)
presenterOutput.showError.emit(to: showError).disposed(by: disposeBag)
presenterOutput.tags.drive(tags).disposed(by: disposeBag)
presenterOutput.partners.drive(partners).disposed(by: disposeBag)

                                                    1. Создаём тестовые Observer'ы
// MARK: start

scheduler.start()

// MARK: assert

XCTAssertEqual(loadingIndicatorVisible.events,
               [.next(time0, true), .next(time1, false), .next(time2, true), .next(time3, false)])

XCTAssertEqual(contentVisible.events,
               [.next(time0, false), .next(time3, true)])

XCTAssertEqual(showError.events, [.next(time0, nil), .next(time1, loadingError.localizedDescription), .next(time2, nil)])

XCTAssertEqual(tags.events, [.next(time3, [tag]), .completed(time3)])

XCTAssertEqual(partners.events, [.next(time0, []), .next(time3, [partner])])

}
```

Тесты

```
let loadingIndicatorVisible = scheduler.createObserver(Bool.self)
let contentVisible = scheduler.createObserver(Bool.self)
let showError = scheduler.createObserver(String?.self)

let tags = scheduler.createObserver([FilterTag].self)
let partners = scheduler.createObserver([CBPartnerModel].self)
// let partnerDescription - в этом тесте не проверяем

presenterOutput.loadingIndicatorVisible.drive(loadingIndicatorVisible).disposed(by: disposeBag)
presenterOutput.contentVisible.drive(contentVisible).disposed(by: disposeBag)
presenterOutput.showError.emit(to: showError).disposed(by: disposeBag)
presenterOutput.tags.drive(tags).disposed(by: disposeBag)
presenterOutput.partners.drive(partners).disposed(by: disposeBag)

// MARK: start

scheduler.start()

// MARK: assert

XCTAssertEqual(loadingIndicatorVisible.events,
               [.next(time0, true), .next(time1, false), .next(time2, true), .next(time3, false)])

XCTAssertEqual(contentVisible.events,
               [.next(time0, false), .next(time3, true)])

XCTAssertEqual(showError.events, [.next(time0, nil), .next(time1, loadingError.localizedDescription), .next(time2, nil)])

XCTAssertEqual(tags.events, [.next(time3, [tag]), .completed(time3)])

XCTAssertEqual(partners.events, [.next(time0, []), .next(time3, [partner])])

}
```

1. Создаём тестовые Observer’ы

2. Биндим PresenterOutput и тестовые Observer’ы

Тесты

```
let loadingIndicatorVisible = scheduler.createObserver(Bool.self)
let contentVisible = scheduler.createObserver(Bool.self)
let showError = scheduler.createObserver(String?.self)

let tags = scheduler.createObserver([FilterTag].self)
let partners = scheduler.createObserver([CBPartnerModel].self)
// let partnerDescription - в этом тесте не проверяем

presenterOutput.loadingIndicatorVisible.drive(loadingIndicatorVisible).disposed(by: disposeBag)
presenterOutput.contentVisible.drive(contentVisible).disposed(by: disposeBag)
presenterOutput.showError.emit(to: showError).disposed(by: disposeBag)
presenterOutput.tags.drive(tags).disposed(by: disposeBag)
presenterOutput.partners.drive(partners).disposed(by: disposeBag)

// MARK: start

scheduler.start()

// MARK: assert

XCTAssertEqual(loadingIndicatorVisible.events,
               [.next(time0, true), .next(time1, false), .next(time2, true), .next(time3, false)])

XCTAssertEqual(contentVisible.events,
               [.next(time0, false), .next(time3, true)])

XCTAssertEqual(showError.events, [.next(time0, nil), .next(time1, loadingError.localizedDescription), .next(time2, nil)])

XCTAssertEqual(tags.events, [.next(time3, [tag]), .completed(time3)])

XCTAssertEqual(partners.events, [.next(time0, []), .next(time3, [partner])])

}
```

1. Создаём тестовые Observer'ы

2. Биндим PresenterOutput и тестовые Observer'ы

3. Запускаем тест

Тесты

```
let loadingIndicatorVisible = scheduler.createObserver(Bool.self)
let contentVisible = scheduler.createObserver(Bool.self)
let showError = scheduler.createObserver(String?.self)

let tags = scheduler.createObserver([FilterTag].self)
let partners = scheduler.createObserver([CBPartnerModel].self)
// let partnerDescription - в этом тесте не проверяем

presenterOutput.loadingIndicatorVisible.drive(loadingIndicatorVisible).disposed(by: disposeBag)
presenterOutput.contentVisible.drive(contentVisible).disposed(by: disposeBag)
presenterOutput.showError.emit(to: showError).disposed(by: disposeBag)
presenterOutput.tags.drive(tags).disposed(by: disposeBag)
presenterOutput.partners.drive(partners).disposed(by: disposeBag)

// MARK: start

scheduler.start()

// MARK: assert

XCTAssertEqual(loadingIndicatorVisible.events,
               [.next(time0, true), .next(time1, false), .next(time2, true), .next(time3, false)])

XCTAssertEqual(contentVisible.events,
               [.next(time0, false), .next(time3, true)])

XCTAssertEqual(showError.events, [.next(time0, nil), .next(time1, loadingError.localizedDescription), .next(time2, nil)])

XCTAssertEqual(tags.events, [.next(time3, [tag]), .completed(time3)])

XCTAssertEqual(partners.events, [.next(time0, []), .next(time3, [partner])])
}
```

1. Создаём тестовые Observer'ы
2. Биндим PresenterOutput и тестовые Observer'ы
3. Запускаем тест
4. Проверяем результат через XCTAssertEqual

Тесты

```
let loadingIndicatorVisible = scheduler.createObserver(Bool.self)
let contentVisible = scheduler.createObserver(Bool.self)
let showError = scheduler.createObserver(String?.self)

let tags = scheduler.createObserver([FilterTag].self)
let partners = scheduler.createObserver([CBPartnerModel].self)
// let partnerDescription - в этом тесте не проверяем

presenterOutput.loadingIndicatorVisible.drive(loadingIndicatorVisible).disposed(by: disposeBag)
presenterOutput.contentVisible.drive(contentVisible).disposed(by: disposeBag)
presenterOutput.showError.emit(to: showError).disposed(by: disposeBag)
presenterOutput.tags.drive(tags).disposed(by: disposeBag)
presenterOutput.partners.drive(partners).disposed(by: disposeBag)

// MARK: start

scheduler.start()

// MARK: assert

XCTAssertEqual(loadingIndicatorVisible.events,
               [.next(time0, true), .next(time1, false), .next(time2, true), .next(time3, false)])

                        [.isLoading, .loadingError(loadingError), .isLoading, .dataLoaded(partnersInfo)]


    func testErrorRetrying() {
```

1. Создаём тестовые Observer'ы
2. Биндим PresenterOutput и тестовые Observer'ы
3. Запускаем тест
4. Проверяем результат через XCTAssertEqual

Правила

Правила

- В Output’ах используем только Observable. Observer’ы и Subject’ы категорически исключены

Правила

- В Output’ах используем только Observable. Observer’ы и Subject’ы категорически исключены
- Immutable структуры данных

Правила

- В Output’ах используем только Observable. Observer’ы и Subject’ы категорически исключены
- Immutable структуры данных
- Не используем Rx для общения экранов друг с другом

Правила

- В Output’ах используем только Observable. Observer’ы и Subject’ы категорически исключены
- Immutable структуры данных
- Не используем Rx для общения экранов друг с другом
- Не используем Rx в сетевом слое

Правила

- В Output’ах используем только Observable. Observer’ы и Subject’ы категорически исключены
- Immutable структуры данных
- Не используем Rx для общения экранов друг с другом
- Не используем Rx в сетевом слое
- Используя Rx, мы не пытаемся удивить друг друга своим мастерством

Уточнение требований

Уточнение требований

Примеры двусмысленных формулировок:

Если данные не загрузились, то ничего не показываем.

Если пришёл ответ сервера, то открываем следующий экран.

Отображаем данные на экране по мере их подгрузки.

При скроллинге списка заранее подгружаем данные.

Уточнение требований

Примеры двусмысленных формулировок:

– Мам, что мне делать?

– Главное не делай глупостей.

Уточнение требований

Автоматическая авторизация

Уточнение требований

Автоматическая авторизация

Исходная постановка: после разлогина, неважно ручного или из-за ошибки токена, xbr срабатывать не должен.

Уточнение требований

Автоматическая авторизация

Исходная постановка: после разлогина, неважно ручного или из-за ошибки токена, xbr срабатывать не должен.

Чтобы это требование стало однозначным, на этапе анализа мы его переформулировали:

Уточнение требований

Автоматическая авторизация

Исходная постановка: после разлогина, неважно ручного или из-за ошибки токена, xbr срабатывать не должен.

Чтобы это требование стало однозначным, на этапе анализа мы его переформулировали:

XBR срабатывает 1 раз за жизненный цикл приложения. Это может произойти в 2 сценариях:

Уточнение требований

Автоматическая авторизация

Исходная постановка: после разлогина, неважно ручного или из-за ошибки токена, xbr срабатывать не должен.

Чтобы это требование стало однозначным, на этапе анализа мы его переформулировали:

XBR срабатывает 1 раз за жизненный цикл приложения. Это может произойти в 2 сценариях:

1. При запуске приложения в случае, если отсутствуют сохранённые Credentials и у пользователя включен мобильный интернет

Уточнение требований

Автоматическая авторизация

Исходная постановка: после разлогина, неважно ручного или из-за ошибки токена, xbr срабатывать не должен.

Чтобы это требование стало однозначным, на этапе анализа мы его переформулировали:

XBR срабатывает 1 раз за жизненный цикл приложения. Это может произойти в 2 сценариях:

1. При запуске приложения в случае, если отсутствуют сохранённые Credentials и у пользователя включен мобильный интернет

2. Если при запуске приложения отсутствовали сохранённые Credentials и был выключен мобильный интернет. В этом случае, если пользователь ещё не авторизовался самостоятельно через логин и пароль, и включил мобильный интернет, то запускается xbr.

С чего начать?

- Сделать 1 несложный экран с 3 состояниями. Наиболее комфортно будет сделать новый экран. Например экран баланса, профиля или списка контактов.

- Почитать главу 5 и 6.

- Спроектировать на бумаге экран посложнее и попробовать реализовать его.

Итоги

Итоги

- Устойчивый к изменениям код

Итоги

- Устойчивый к изменениям код
- Меньше багов и более легкая их починка

Итоги

- Устойчивый к изменениям код
- Меньше багов и более легкая их починка
- Чистый код. Разработчикам легче понимать код друг друга

Итоги

- Устойчивый к изменениям код
- Меньше багов и более легкая их починка
- Чистый код. Разработчикам легче понимать код друг друга
- Снижение стоимости сопровождения

Итоги

- Устойчивый к изменениям код
- Меньше багов и более легкая их починка
- Чистый код. Разработчикам легче понимать код друг друга
- Снижение стоимости сопровождения
- Можно с высокой степенью детализации восстановить диаграмму состояний экрана по коду.

Итоги

- Устойчивый к изменениям код
- Меньше багов и более легкая их починка
- Чистый код. Разработчикам легче понимать код друг друга
- Снижение стоимости сопровождения
- Можно с высокой степенью детализации восстановить диаграмму состояний экрана по коду.
- *Общее проектирование для iOS / Android

Итоги

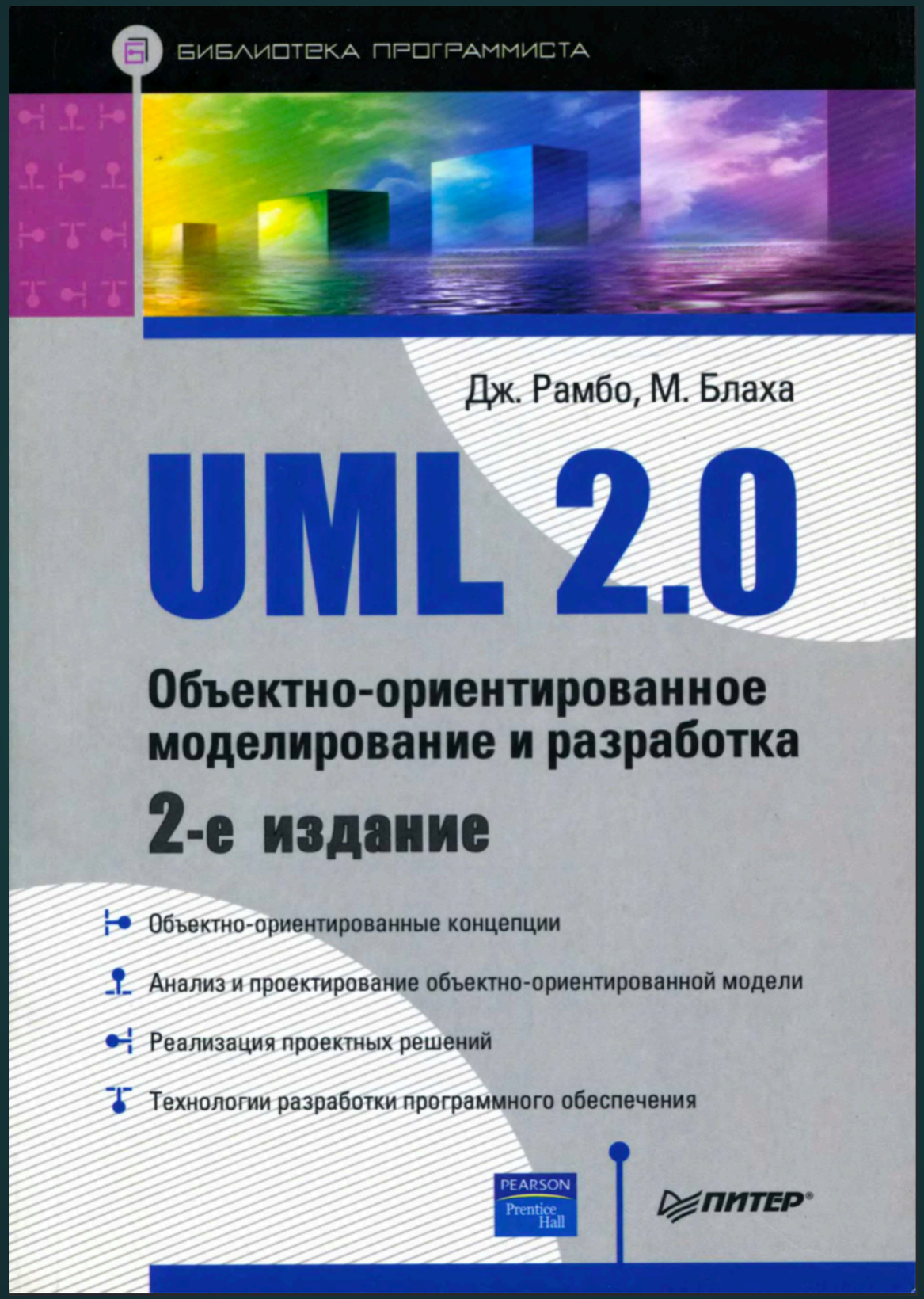

Стоимость сопровождения
200
180
160
140
120
100
80
60
40
20
0
1
2
3
5
8
13
21
34
Кол-во состояний
Ordinary
Stateful

Итоги


Стоимость сопровождения
200
180
160
140
120
100
80
60
40
20
0
1
2
3
5
8
13
21
24
34
current max
Кол-во состояний
Ordinary
Stateful

Итоги


Стоимость сопровождения
50
45
40
35
30
25
20
15
10
5
0
3
5
8
13
Кол-во состояний
Ordinary
Stateful


БИБЛИОТЕКА ПРОГРАММИСТА
Дж. Рамбо, М. Блаха
UML 2.0
Объектно-ориентированное моделирование и разработка
2-е издание
Объектно-ориентированные концепции
Анализ и проектирование объектно-ориентированной модели
Реализация проектных решений
Технологии разработки программного обеспечения
PEARSON Prentice Hall
ПИТЕР

Ссылки

Дмитрий Тримонов: Rx в современной мобильной разработке под iOS
https://www.youtube.com/watch?v=TBOjtw9g0Nw

Шпаргалка по тестированию требований к мобильным приложениям
https://habr.com/company/mobileup/blog/336992/

https://github.com/iDmitriyy/StatefulScreenExample (демо проект)

https://t.me/iDmitriyy

https://medium.com/@dy.ignatyev

Ссылки

Дмитрий Тримонов: Rx в современной мобильной разработке под iOS
https://www.youtube.com/watch?v=TBOjtw9g0Nw

Шпаргалка по тестированию требований к мобильным приложениям
https://habr.com/company/mobileup/blog/336992/

https://github.com/iDmitriyy/StatefulScreenExample (демо проект)

https://t.me/iDmitriyy

https://medium.com/@dy.ignatyev