Nivelir — новый фреймворк навигации в iOS

Тимур Шафигуллин

Навигация в iOS

это сложно

Сложно потому что

1. Навигация с цепочкой переходов
2. Проверка текущего открытого экрана
3. Передача данных между экранами

Есть куча краевых кейсов

1. Одинаковая навигация с нескольких экранов
2. Переход по ссылке или push-уведомлению (deep linking)
3. Обработка условий при навигации (проверка прав, авторизация, и т.п)

Какой фреймворк выбрать?!

Тимур Шафигуллин

iOS Разработчик в Core команде hh.ru

Разрабатываю iOS приложения с 2017 года

Знаю, как построить сложную навигацию и не сойти с ума

@timbaev @timbaev @timbaev

История навигации в hh.ru

Как открывались наши экраны?

1. Форк Marshroute от Avito
2. enum Destination
3. Координаторы

Как открывались наши экраны?

1. **Форк Marshroute от Avito**
2. enum Destination
3. Координаторы

Как открывались наши экраны?

1. Форк Marshroute от Avito
2. enum Destination
3. Координаторы

```swift
public enum Destination: Equatable {
    case vacancies(searchConfiguration: VacancySearchConfiguration)
    case geoSearch
    case latestSearch
    case favorites
    case autosearchList
    case autosearch(savedSearch: HHSavedSearch)
    case resumes
    case resume(data: ResumeDestinationData)
    case resumeFix(data: ResumeDestinationData)
    case suitableVacancies(resumeId: String?)
    case similarVacancies(vacancyID: String)
    case resumeViews(resumeId: String?)
    case createResume(force: Bool)
    case negotiations
    case invitations
    case negotiation(negotiationId: String)
    case blacklistedVacancies
    case countrySwitch
    case vacancy(vacancyId: String, trackingParams: [String: AnyHashable]?)
    case employer(employerId: String)
    case authorization
    case logout(isUserInitiated: Bool)
    case searchTab
    case account
    case notificationSettings
    case supportChat(supportChatPushNotification: SupportChatPushNotification?)
    case jobSearchMap(searchConfiguration: VacancySearchConfiguration)
    case resumeWebFilling(resumeId: String)
    case articles
    case suitableVacanciesMain(resumeID: String)
    case vacancyTinder(resumeID: String?)
    case chat(chatID: String)
    case chatsTab
    case searchHistory
    case partTimeVacancies(partTimeJobs: Set<PartTimeJob>)

    // Web url deeplinks
    case safariWebView(url: URL)
    case browser(url: URL, showAlert: Bool)
    case webView(url: URL, title: String)
    case employersRating(isLast: Bool)
}
```

Как открывались наши экраны?

1. Форк Marshroute от Avito
2. enum Destination
3. Координаторы

```
static NSString *const kHHApplicationBackurlsHostForCustomScheme = @"backurlsHostForCustomScheme";
static NSString *const kHHApplicationUrlLogin                    = @"login";
static NSString *const kHHApplicationUriApplicant                = @"applicant";
static NSString *const kHHApplicationUriEmployer                 = @"employer";
static NSString *const kHHApplicationUriAccount                  = @"account";
static NSString *const kHHApplicationUriSearch                   = @"search";
static NSString *const kHHApplicationUriVacancy                  = @"vacancy";
static NSString *const kHHApplicationUriVacancies                = @"vacancies";
static NSString *const kHHApplicationUriResume                   = @"resume";
static NSString *const kHHApplicationUriResumeCreate             = @"create";
static NSString *const kHHApplicationUriResumeCompletion         = @"completion";
static NSString *const kHHApplicationUriResumes                  = @"resumes";
static NSString *const kHHApplicationUriResumeHash               = @"resumeHash";
static NSString *const kHHApplicationUriHistory                  = @"history";
static NSString *const kHHApplicationUriNew                      = @"new";
static NSString *const kHHApplicationUriResumeView               = @"resumeview";
static NSString *const kHHApplicationUriNegotiations             = @"negotiations";
static NSString *const kHHApplicationUriNegotiationsTopic        = @"topic";
static NSString *const kHHApplicationUriNegotiationsTopicId      = @"topicId";
static NSString *const kHHApplicationUriNegotiationsChatId       = @"chatId";
static NSString *const kHHApplicationUriParameterCode            = @"code";
static NSString *const kHHApplicationUriParameterError           = @"error";
static NSString *const kHHApplicationUriParameterWidget          = @"widget";
static NSString *const kHHApplicationUriParameterText            = @"text";
static NSString *const kHHApplicationUriFavoriteVacancies        = @"favorite_vacancies";
static NSString *const kHHApplicationUriNearby                   = @"nearby";
static NSString *const kHHApplicationUriBlacklistedVacancies     = @"blacklist";
static NSString *const kHHApplicationUriCountrySwitch            = @"area_switcher";
static NSString *const kHHApplicationUriSuitableResumes          = @"suitable_resumes";
static NSString *const kHHApplicationUriGifts                    = @"gifts";
static NSString *const kHHApplicationUriLastSearch               = @"last_search";
static NSString *const kHHApplicationUriSuitable                 = @"suitable";
static NSString *const kHHApplicationUriSubscribe                = @"subscribe";
static NSString *const kHHApplicationUriVacancyResponses         = @"vacancyresponses";
static NSString *const kHHApplicationUriVacancyId                = @"vacancyId";
static NSString *const kHHApplicationUriTopicIdT                 = @"t";
static NSString *const kHHApplicationUriItem                     = @"item";
static NSString *const kHHApplicationUriMakeApplicantBackurl     = @"makeApplicantBackurl";
static NSString *const kHHApplicationUriCaptchaSuccessBackurl    = @"captchaSuccessBackurl";
static NSString *const kHHApplicationUriAutosearch               = @"autosearch";
static NSString *const kHHApplicationUriJobSearchMap             = @"vacancies_nearby";
static NSString *const kHHApplicationUriDocuments                = @"documents";
static NSString *const kHHApplicationUriAnalyticsSource          = @"analytics_source";
static NSString *const kHHApplicationUriOAuth                    = @"oauth";
static NSString *const kHHApplicationUriAuthorize                = @"authorize";
static NSString *const kHHApplicationUriArticles                 = @"articles";
static NSString *const kHHApplicationUriResumeFix                = @"fix";
static NSString *const kHHApplicationUriSuitableVacancies        = @"suitable_vacancies";
static NSString *const kHHApplicationUriVacancyTinder            = @"tinderlike_suitable";
static NSString *const kHHApplicationUriSearchHistory            = @"search_history";
static NSString *const kHHApplicationURLTypeEmployersRating      = @"employers_rating";
```

Как открывались наши экраны?

1. Форк Marshroute от Avito
2. enum Destination
3. Координаторы

Как открывались наши экраны?

1. Форк Marshroute от Avito
2. enum Destination
3. Координаторы

Неудобно


Фреймворки
навигации

Фреймворков навигации — много

Навскидку вот

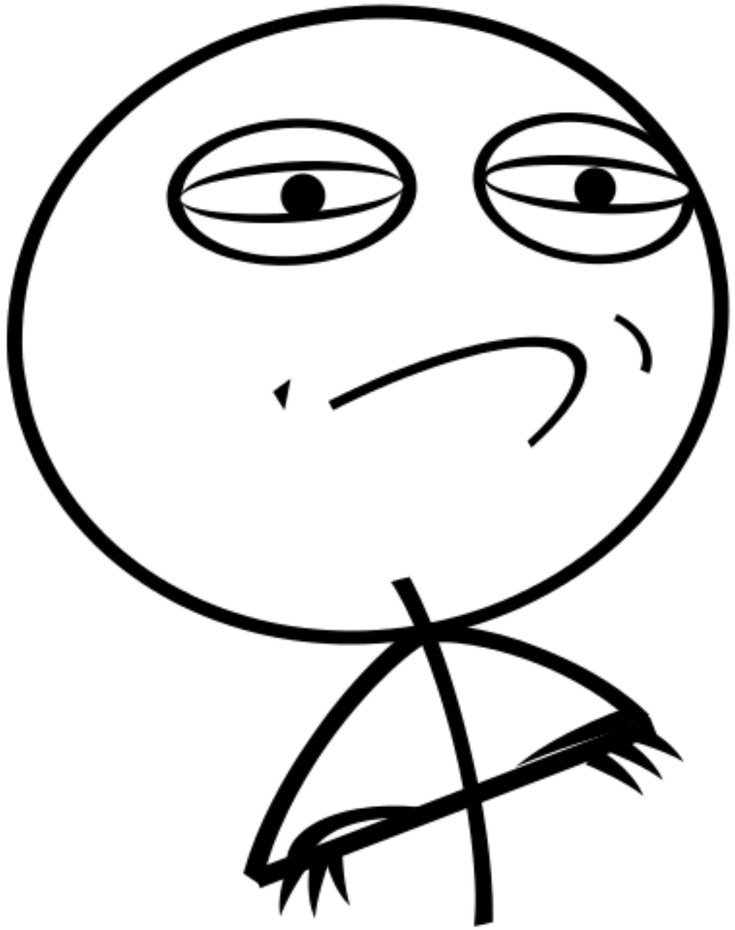
1. Стандартный средства для навигации из UIKit
2. Avito Marshroute
3. Badoo
4. Route Composer
5. Какой-нибудь ещё

Но если присмотреться...

1. Стандартный средства для навигации из UIKit
2. Avito Marshroute
3. **Badoo**
4. **Route Composer**
5. Какой-нибудь ещё

Как оценить фреймворк?

Критерии фреймворков



Мы подумали

и придумали аж 3 критерия

Критерии выбора фреймворков

01

Удобство работы

02

Граф навигации

03

Масштабируемость

	Фреймворк/Критерий	Badoo	route-composer
Удобство работы	Локальная навигация	❌	❌
	Цепочки открытия	❌	✅
	Поиск открытого экрана	✅/❌	✅
	Удобный DSL	✅	❌
	Строгость типизации	❌	✅/❌
	Кастомные анимации	❌	✅/❌
Граф навигации	Обработка ошибок	❌	✅
	Интерцепторы	❌	✅
	Deep links	✅	✅
Масшта-бируемость	Многомодульность	✅	✅
	Постепенная миграция	✅	❌

	Фреймворк/Критерий	Badoo	route-composer	Nivelir
Удобство работы	Локальная навигация	❌	❌	✅
	Цепочки открытия	❌	✅	✅
	Поиск открытого экрана	✅/❌	✅/❌	✅
	Удобный DSL	✅	❌	✅
	Строгость типизации	❌	✅/❌	✅
	Кастомные анимации	❌	✅/❌	✅
Граф навигации	Обработка ошибок	❌	✅	✅
	Интерцепторы	❌	✅	✅
	Deep links	✅	✅	✅
Масшта-бируемость	Многомодульность	✅	✅	✅
	Постепенная миграция	✅	❌	✅

Timbaev 7 апреля в 12:49

Обзор решений для навигации в iOS

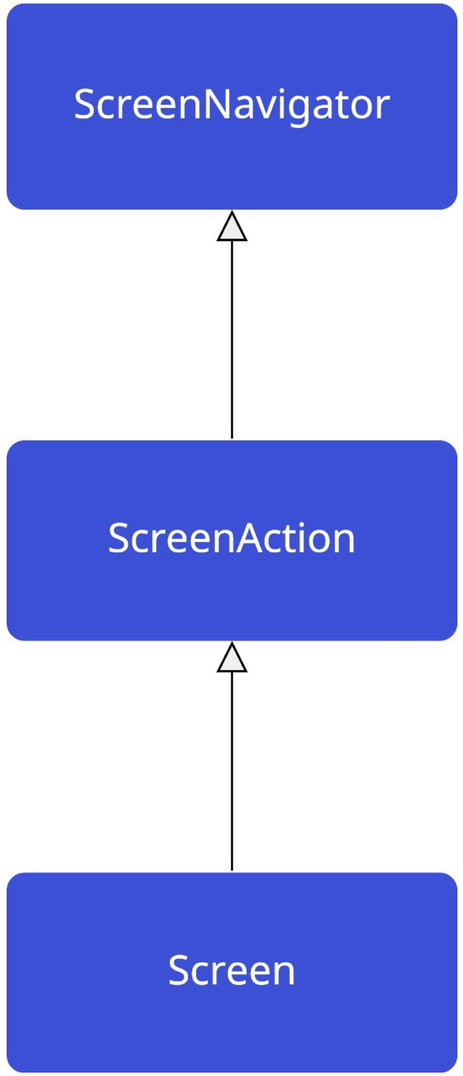
Блог компании HeadHunter, Разработка под iOS*, Разработка мобильных приложений*, Swift*

Всем привет! Меня зовут Тимур, я – iOS разработчик в hh.ru. В этой статье поговорим о фреймворкинге навигации в iOS. Я расскажу кулстори о популярных и не очень решениях и их преимуществах, а еще о том, как мы искали фреймворк мечты среди этой смертной любви. Поехали!

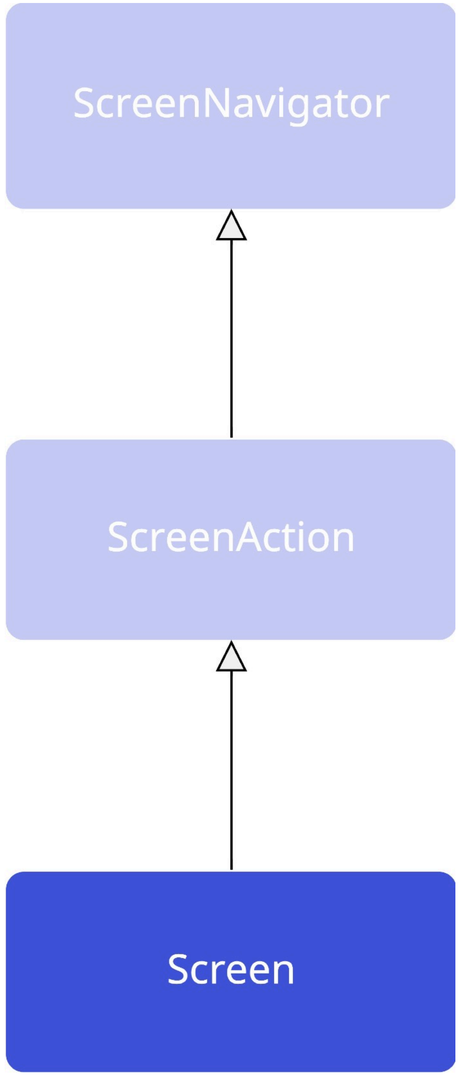
Читать далее

Основные сущности

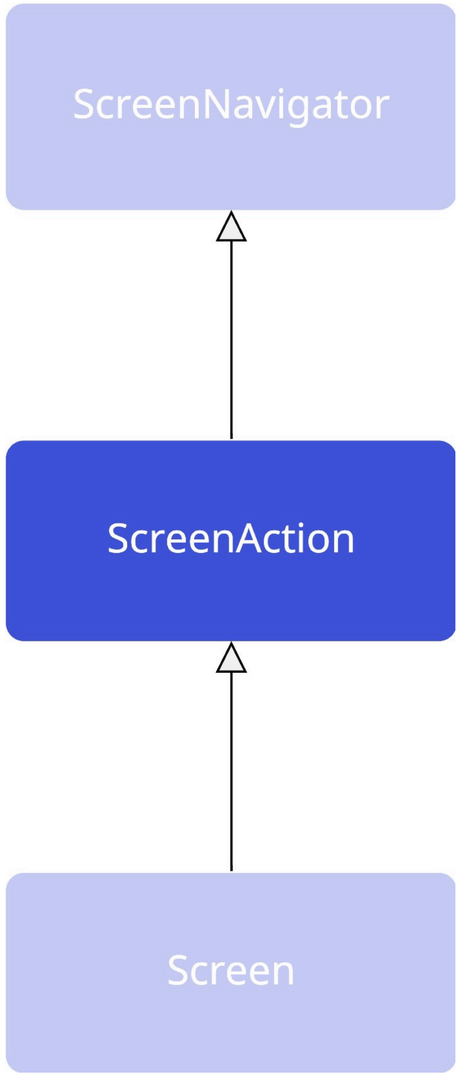
Основные сущности



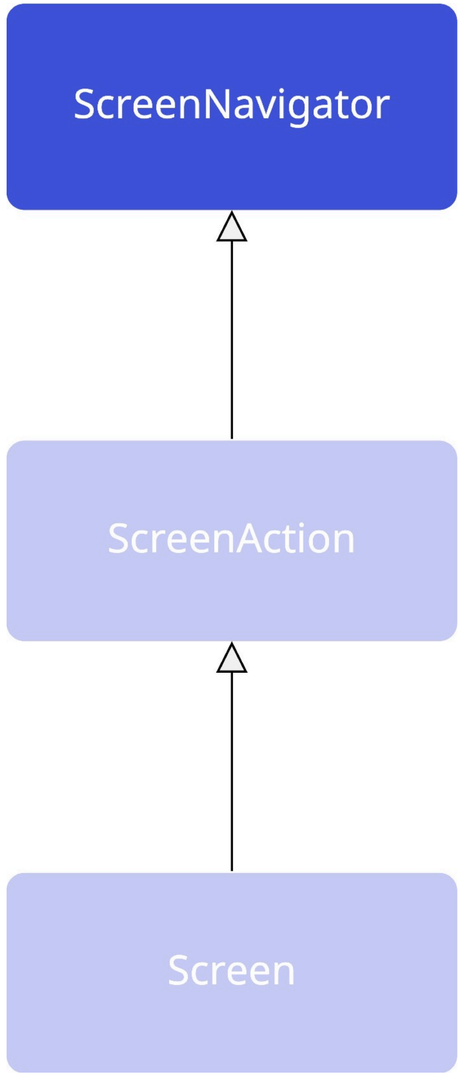
Screen



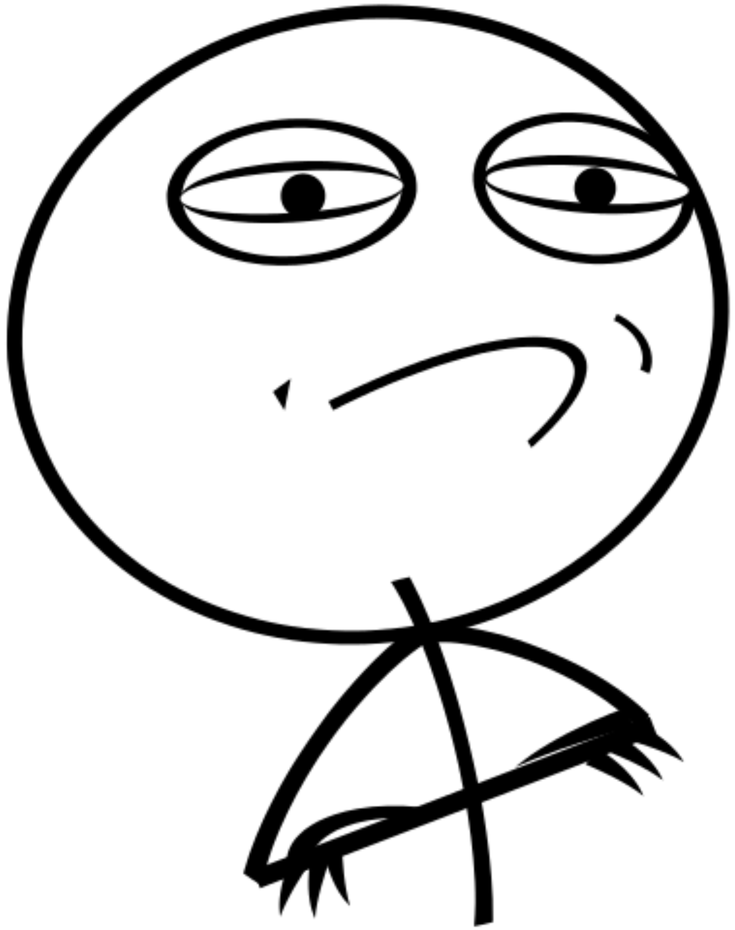
ScreenAction



ScreenNavigator




Удобство
работы



Удобство

локальной навигации

Что имеется в виду?

Удобство локальной навигации

1. Легкая и удобная
2. Без больших конструкций
3. Удобная работа с контроллерами из UIKit

Локальная навигация

```
private func showChat(id: Int) {
    screenNavigator.navigate(from: stack) { route in
        route.push(
            ChatScreen(
                roomID: roomID,
                chatID: id
            )
        )
    }
}
```

Локальная навигация

1. UIAlertController
2. UIDocumentInteractionController
3. UIImagePickerController
4. SKStoreProductViewController
5. И другие

ActionSheet

```
private func pickPhotoImage(sender: UIView) {
    let actionSheet = ActionSheet(
        anchor: .center(of: sender),
        actions: [
            ActionSheetAction(title: "Take Photo") {
                self.pickPhotoImageFromCamera()
            },
            ActionSheetAction(title: "Choose Photo") {
                self.pickPhotoImageFromPhotoLibrary()
            },
            .cancel(title: "Cancel")
        ]
    )

    screenNavigator.navigate(from: self) { route in
        route.showActionSheet(actionSheet)
    }
}
```

ActionSheet – создание

```
private func pickPhotoImage(sender: UIView) {
    let actionSheet = ActionSheet(
        anchor: .center(of: sender),
        actions: [
            ActionSheetAction(title: "Take Photo") {
                self.pickPhotoImageFromCamera()
            },
            ActionSheetAction(title: "Choose Photo") {
                self.pickPhotoImageFromPhotoLibrary()
            },
            .cancel(title: "Cancel")
        ]
    )

    screenNavigator.navigate(from: self) { route in
        route.showActionSheet(actionSheet)
    }
}
```

ActionSheet – якорь

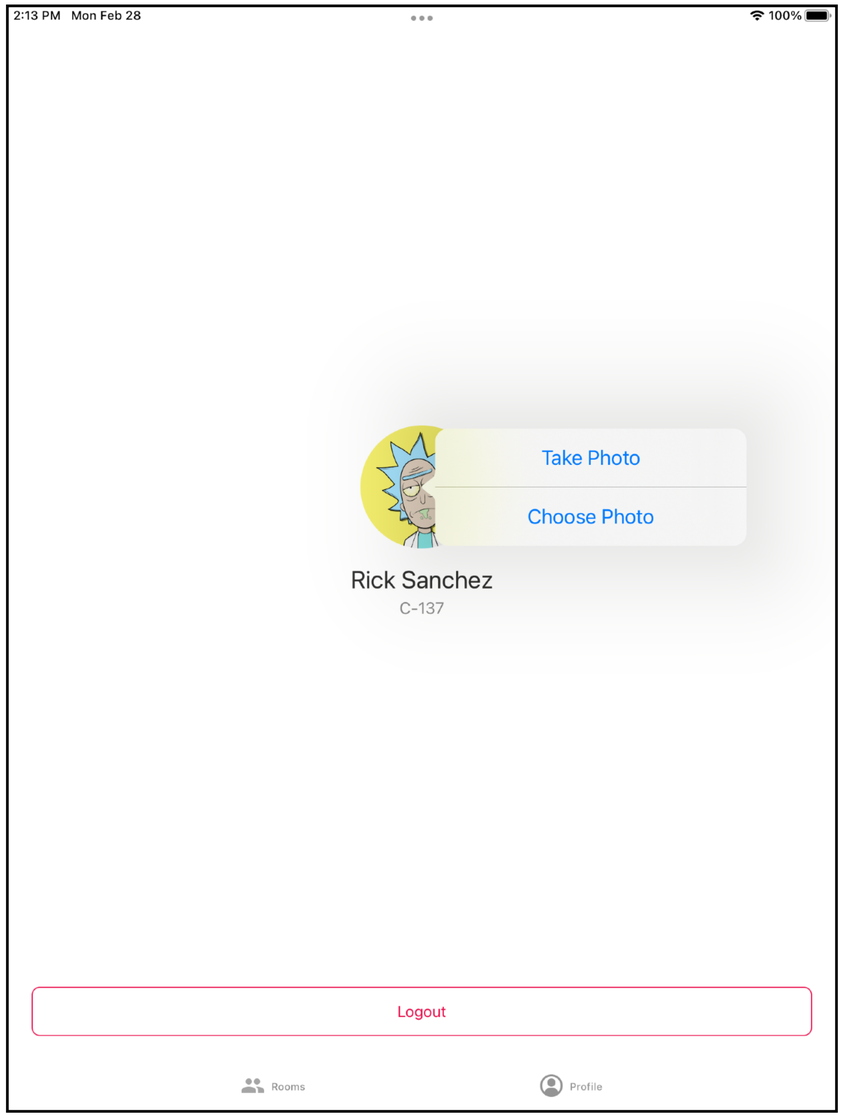
```
private func pickPhotoImage(sender: UIView) {
    let actionSheet = ActionSheet(
        anchor: .center(of: sender),
        actions: [
            ActionSheetAction(title: "Take Photo") {
                self.pickPhotoImageFromCamera()
            },
            ActionSheetAction(title: "Choose Photo") {
                self.pickPhotoImageFromPhotoLibrary()
            },
            .cancel(title: "Cancel")
        ]
    )

    screenNavigator.navigate(from: self) { route in
        route.showActionSheet(actionSheet)
    }
}
```

ActionSheet – действия

```
private func pickPhotoImage(sender: UIView) {
    let actionSheet = ActionSheet(
        anchor: .center(of: sender),
        actions: [
            ActionSheetAction(title: "Take Photo") {
                self.pickPhotoImageFromCamera()
            },
            ActionSheetAction(title: "Choose Photo") {
                self.pickPhotoImageFromPhotoLibrary()
            },
            .cancel(title: "Cancel")
        ]
    )

    screenNavigator.navigate(from: self) { route in
        route.showActionSheet(actionSheet)
    }
}
```

ActionSheet – показ

```
private func pickPhotoImage(sender: UIView) {
    let actionSheet = ActionSheet(
        anchor: .center(of: sender),
        actions: [
            ActionSheetAction(title: "Take Photo") {
                self.pickPhotoImageFromCamera()
            },
            ActionSheetAction(title: "Choose Photo") {
                self.pickPhotoImageFromPhotoLibrary()
            },
            .cancel(title: "Cancel")
        ]
    )

    screenNavigator.navigate(from: self) { route in
        route.showActionSheet(actionSheet)
    }
}
```



ActionSheet – UIKit

```
private func pickPhotoImage(sender: UIView) {
    let actionSheet = UIAlertController(title: nil, message: nil, preferredStyle: .actionSheet)

    if let popoverController = actionSheet.popoverPresentationController {
        popoverController.sourceRect = CGRect(
            origin: CGPoint(x: sender.bounds.minX, y: sender.bounds.midY),
            size: .zero
        )

        popoverController.sourceView = sender
    }

    actionSheet.addAction(
        UIAlertAction(title: "Take Photo", style: .default, handler: { action in
            self.pickPhotoImageFromCamera()
        })
    )

    actionSheet.addAction(
        UIAlertAction(title: "Choose Photo", style: .default, handler: { action in
            self.pickPhotoImageFromPhotoLibrary()
        })
    )

    actionSheet.addAction(
        UIAlertAction(title: "Cancel", style: .cancel)
    )

    present(actionSheet, animated: true)
}
```

MediaPicker

```
private func pickPhotoImageFromPhotoLibrary() {
    let mediaPicker = MediaPicker { result in
        self.screenNavigator.navigate(from: self) { $0.dismiss() }

        if let result = result {
            self.profileView?.photoImage = result.editedImage ?? result.originalImage
        }
    }
}
```

MediaPicker – создание

```
private func pickPhotoImageFromPhotoLibrary() {
    let mediaPicker = MediaPicker { result in
        self.screenNavigator.navigate(from: self) { $0.dismiss() }

        if let result = result {
            self.profileView?.photoImage = result.editedImage ?? result.originalImage
        }
    }
}
```

MediaPicker – обработка результата

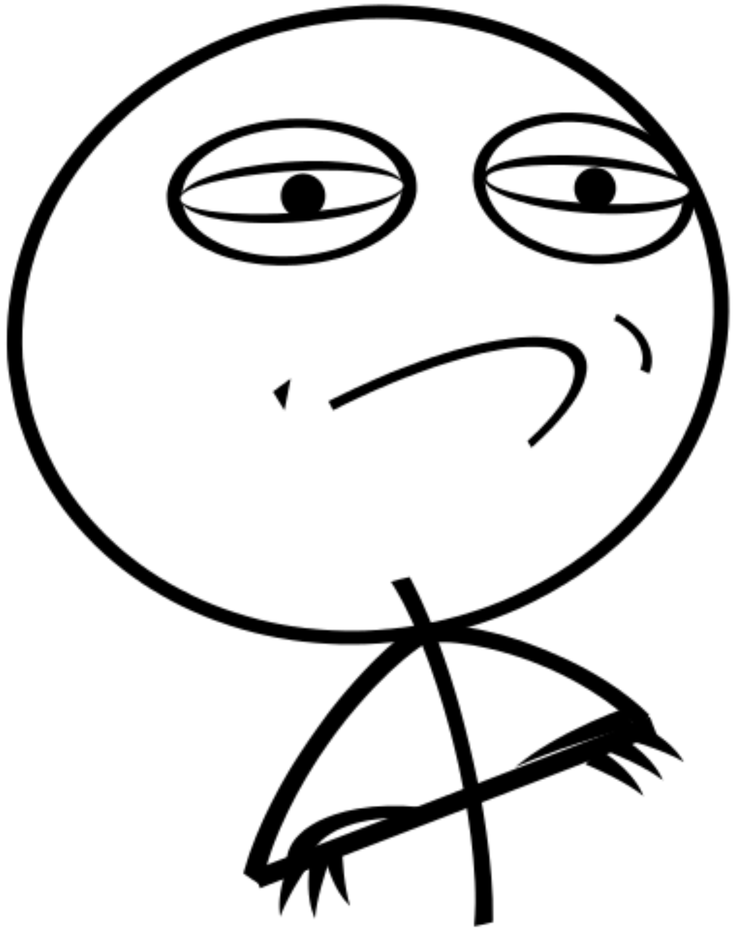
```
private func pickPhotoImageFromPhotoLibrary() {
    let mediaPicker = MediaPicker { result in
        self.screenNavigator.navigate(from: self) { $0.dismiss() }

        if let result = result {
            self.profileView?.photoImage = result.editedImage ?? result.originalImage
        }
    }
}
```

MediaPicker – показ

```
private func pickPhotoImageFromPhotoLibrary() {
    ...
    screenNavigator.navigate(from: self) { route in
        route.showMediaPicker(mediaPicker)
    }
}
```

MediaPicker – UIKit

1. Запросить разрешение
2. Проверить на доступность источник
3. Создать UIImagePickerController
4. Обработать результат в UIImagePickerControllerDelegate
5. ≈ 73 строк кода



Удобство

Цепочки открытия

Что имеется в виду?

Удобство цепочек открытия экранов

1. **Возможность совмещать шаги навигации**

Цепочки открытия

Последовательные

```
func openSupportScreen() {
    navigator.navigate { route in
        route
            .top(.stack)
            .dismiss()
            .push(screens.supportScreen())
    }
}
```

Цепочки открытия

Вложенные

```
func showRootVacancyScreen() {
    navigator.navigate { route in
        route
            .first(.tabs)
            .selectTab(of: UINavigationController.self, with: .index(0)) { route in
                route.popToRoot()
            }
    }
}
```

Цепочки открытия

Возрастающая вложенность

```
func showRootVacancyScreen() {
    screenNavigator.navigate { route in
        route
            .first(.tabs)
            .selectTab(of: UINavigationController.self, with: .index(0)) { route in
                route.push(screens.chatScreen(roomID: roomID, chatID: 1)) { route in
                    route.present(screens.chatScreen(roomID: roomID, chatID: 2))
                }
            }
    }
}
```

Цепочки открытия

Отдельные роуты

```
func showRootVacancyScreen() {
    let presentChatRoute = ScreenModalRoute
        .initial
        .present(screens.chatScreen(roomID: roomID, chatID: 2))

    let pushChatRoute = ScreenStackRoute
        .initial
        .push(screens.chatScreen(roomID: roomID, chatID: 1), route: presentChatRoute)

    navigator.navigate { route in
        route
            .first(.tabs)
            .selectTab(of: UINavigationController.self, with: .index(0), route: pushChatRoute)
    }
}
```

Цепочки открытия

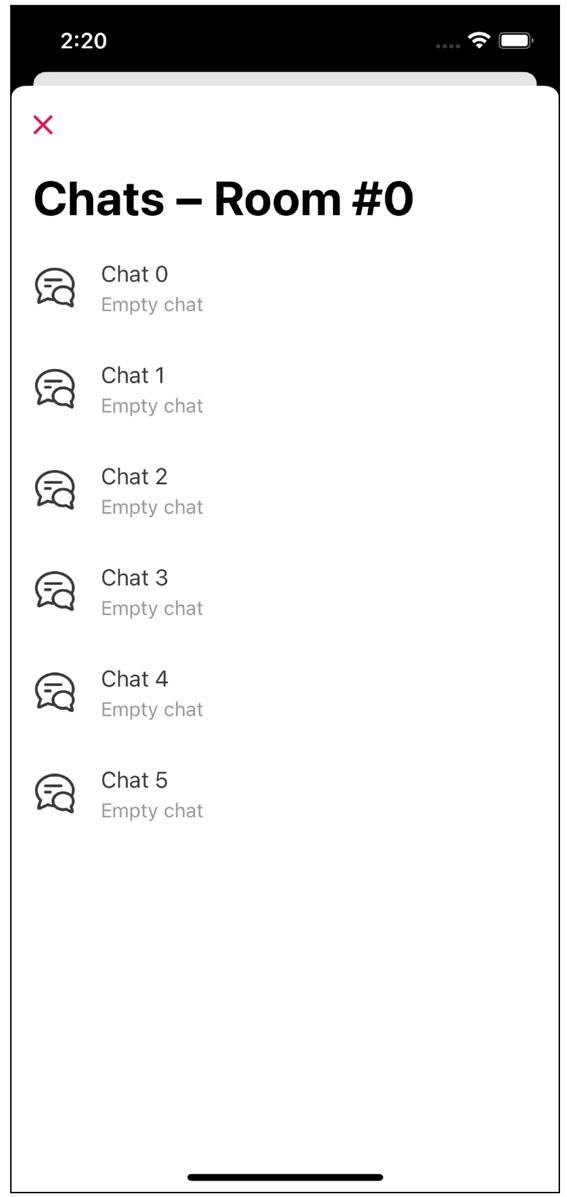
Роуты

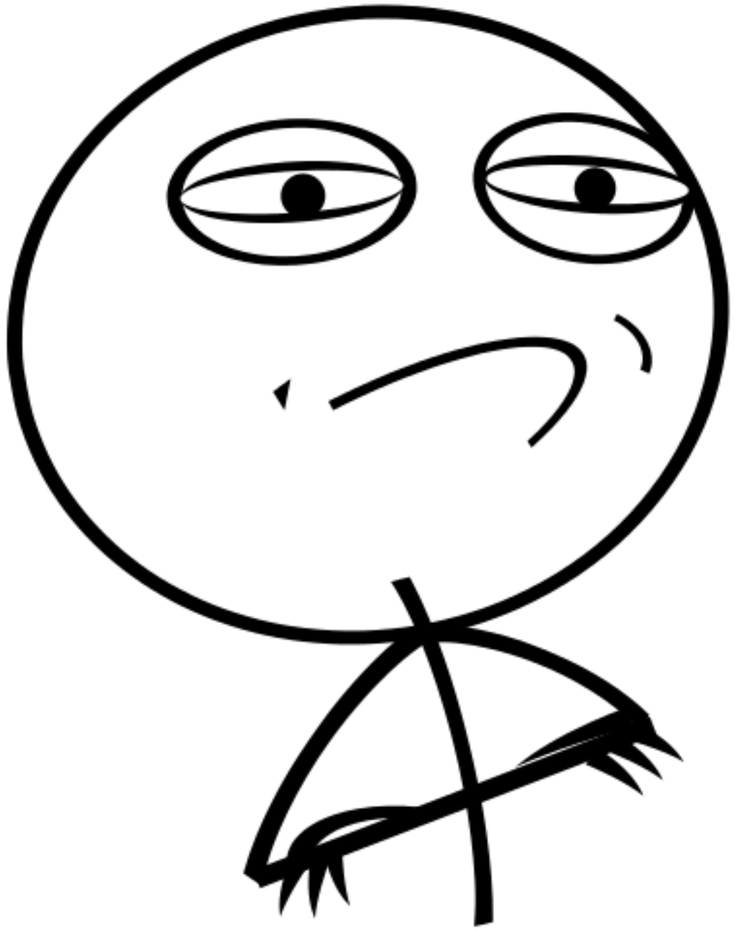
```
public typealias ScreenModalRoute = ScreenRootRoute<UIViewController>

public typealias ScreenStackRoute = ScreenRootRoute<UINavigationController>

public typealias ScreenTabsRoute = ScreenRootRoute<UITabBarController>

public typealias ScreenWindowRoute = ScreenRootRoute<UIWindow>
```

Цепочки открытия

Изменение стека

```
private func showChainNavigation() {

    screenNavigator.navigate(from: stack) { route in

        route

            .push(screens.chatScreen(roomID: roomID, chatID: 1))

            .pop()

            .push(screens.chatScreen(roomID: roomID, chatID: 2))

            .push(screens.chatScreen(roomID: roomID, chatID: 3)) { route in

                route.present(screens.chatScreen(roomID: roomID, chatID: 4))

            }

    }

}
```





Удобство

Поиск открытого экрана

Что имеется в виду?

Удобство поиска открытого экрана

1. Не требуется совершать навигацию
2. Проверка данных на найденном экране
3. Обновление данных

Поиск открытого экрана

```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .resolve()
}
```

Поиск открытого экрана

```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .resolve()
}
```

Поиск открытого экрана

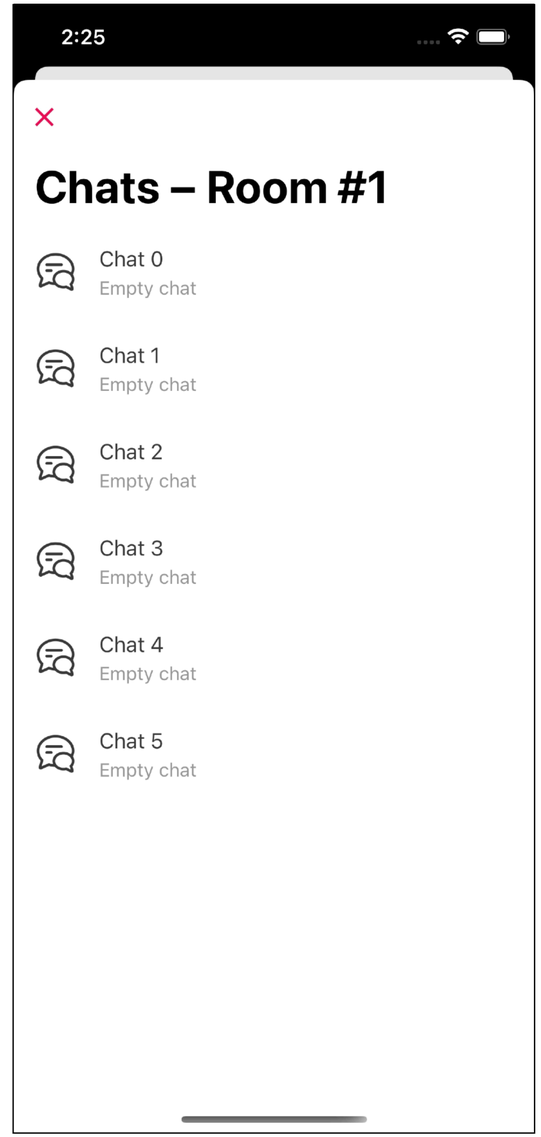
```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .resolve()
}
```

Поиск открытого экрана

```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .resolve()
}
```

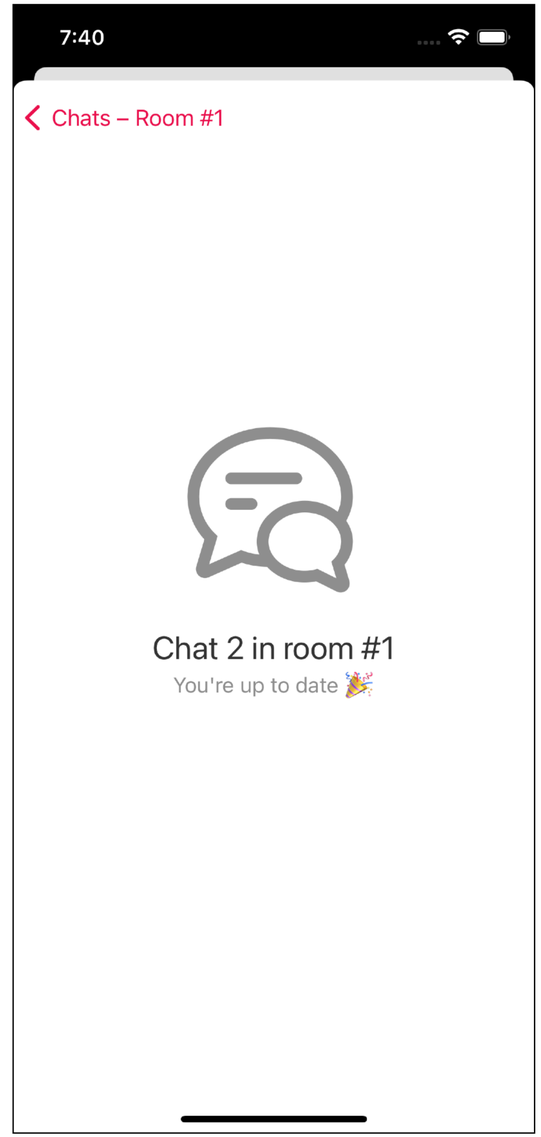
Поиск открытого экрана

```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .resolve()
}
```



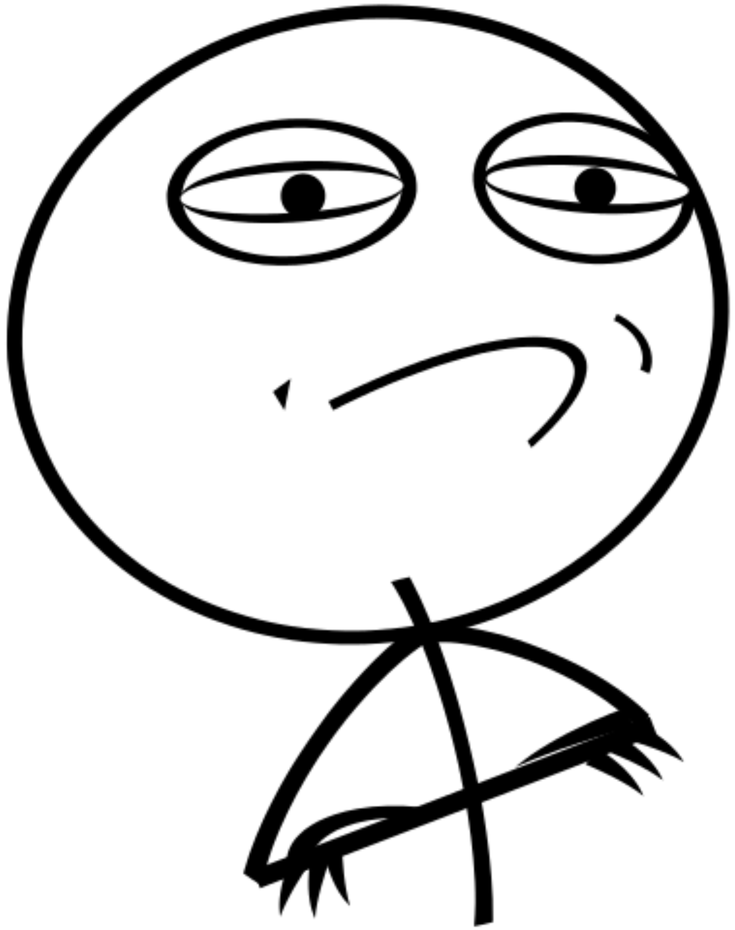
Поиск открытого экрана

```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .resolve()
}
```



Поиск открытого экрана

```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .resolve()
}
```



Удобство

Удобный DSL

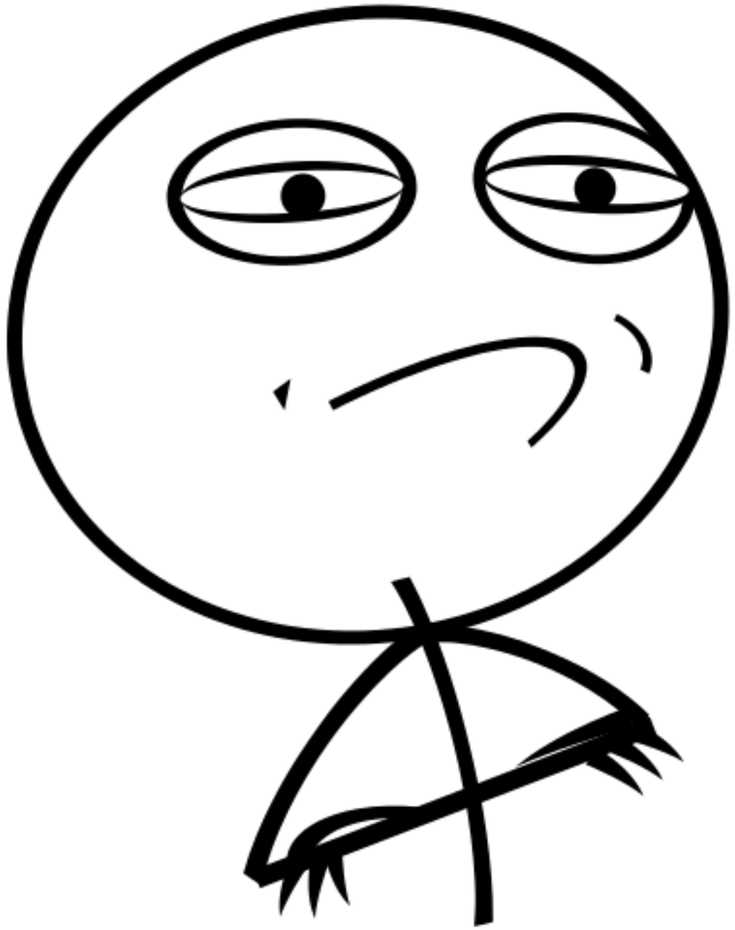
Что имеется в виду?

Удобный DSL

1. С описанием навигации нет трудностей
2. API навигации удобно-читаемо
3. Автодополнение

Удобный DSL

```
52
53      private func showAutocompletion() {
54          
55      }
56
57
58
59
60
61
62
63
64
```



Удобство

Строгость типизации

Что имеется в виду?

Удобство строгости типизации

1. Описание навигации строго типизировано
2. Передача данных между экранами

Строгость типизации

```
private let container: UIViewController?


private func showChainNavigation() {

    screenNavigator.navigate(from: container) { route in

        route.push(screens.chatScreen(roomID: roomID, chatID: 1)) // ❌ Referencing instance method 'push(_:animation:separated:)' on 'ScreenThenable' requires that 'UIViewController' inherit from 'UINavigationController'

    }

}
```

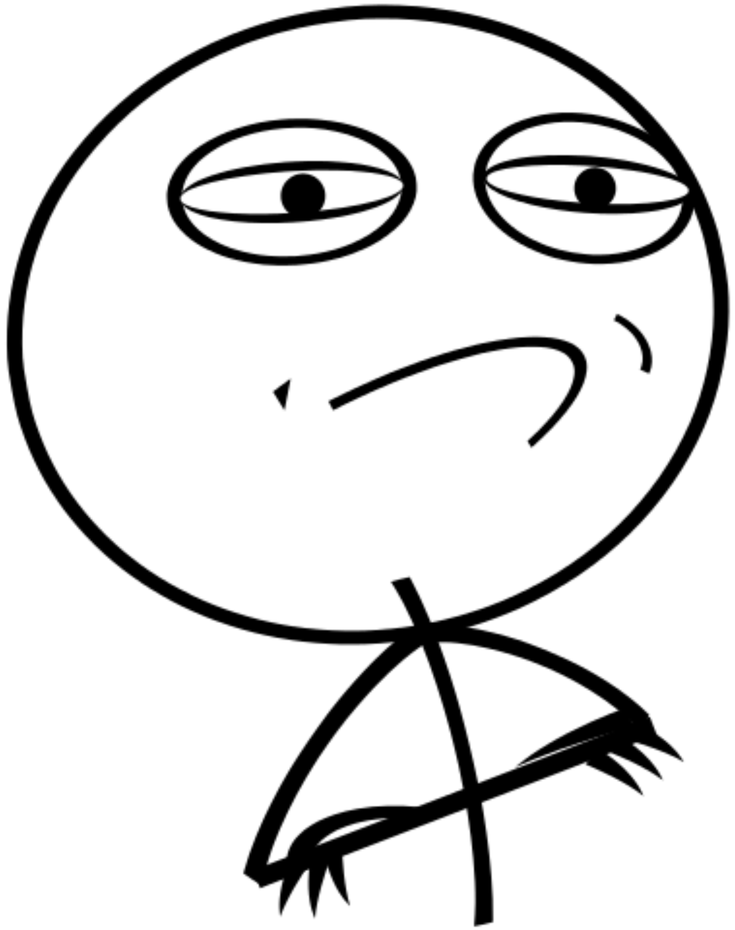
Строгость типизации

```
private let container: UIViewController?

private func showChainNavigation() {
    screenNavigator.navigate(from: container?.stack) { route in
        route.push(screens.chatScreen(roomID: roomID, chatID: 1)) // ✅
    }
}
```

Строгость типизации

Передача данных

```
private let container: UIViewController?

private func showChainNavigation() {
    screenNavigator.navigate(from: container?.stack) { route in
        route.push(screens.chatScreen(roomID: roomID, chatID: 1))
    }
}
```



Удобство

Кастомные анимации

Что имеется в виду?

Удобство кастомных анимаций

1. Смена корневого экрана
2. Изменение стека
3. Переключение таба
4. Модальный показ

Что можно анимировать?

1. **Смена корневого экрана**

```
protocol ScreenRootCustomAnimation
```

2. **Изменение стека**

```
protocol ScreenStackCustomAnimation
```

3. **Переключение таба**

```
protocol ScreenTabCustomAnimation
```

4. **Модальный показ**

```
protocol UIViewControllerTransitioningDelegate
```

Смена корневого экрана

```
public struct ScreenRootTransitionAnimation: ScreenRootCustomAnimation {

    public let duration: TimeInterval

    public let options: UIView.AnimationOptions


    public func animate(

        container: UIWindow,

        from root: UIViewController?,

        to newRoot: UIViewController,

        completion: @escaping () -> Void

    ) {

        UIView.transition(

            with: container,

            duration: duration,

            options: options,

            animations: { },

            completion: { _ in

                completion()

            }

        )

    }

}
```

Смена корневого экрана

```
public struct ScreenRootTransitionAnimation: ScreenRootCustomAnimation {

    public let duration: TimeInterval

    public let options: UIView.AnimationOptions


    public func animate(

        container: UIWindow,

        from root: UIViewController?,

        to newRoot: UIViewController,

        completion: @escaping () -> Void

    ) {

        UIView.transition(

            with: container,

            duration: duration,

            options: options,

            animations: { },

            completion: { _ in

                completion()

            }

        )

    }

}
```

Смена корневого экрана

```
public struct ScreenRootTransitionAnimation: ScreenRootCustomAnimation {
    public let duration: TimeInterval
    public let options: UIView.AnimationOptions

    public func animate(
        container: UIWindow,
        from root: UIViewController?,
        to newRoot: UIViewController,
        completion: @escaping () -> Void
    ) {
        UIView.transition(
            with: container,
            duration: duration,
            options: options,
            animations: { },
            completion: { _ in
                completion()
            }
        )
    }
}
```

Смена корневого экрана

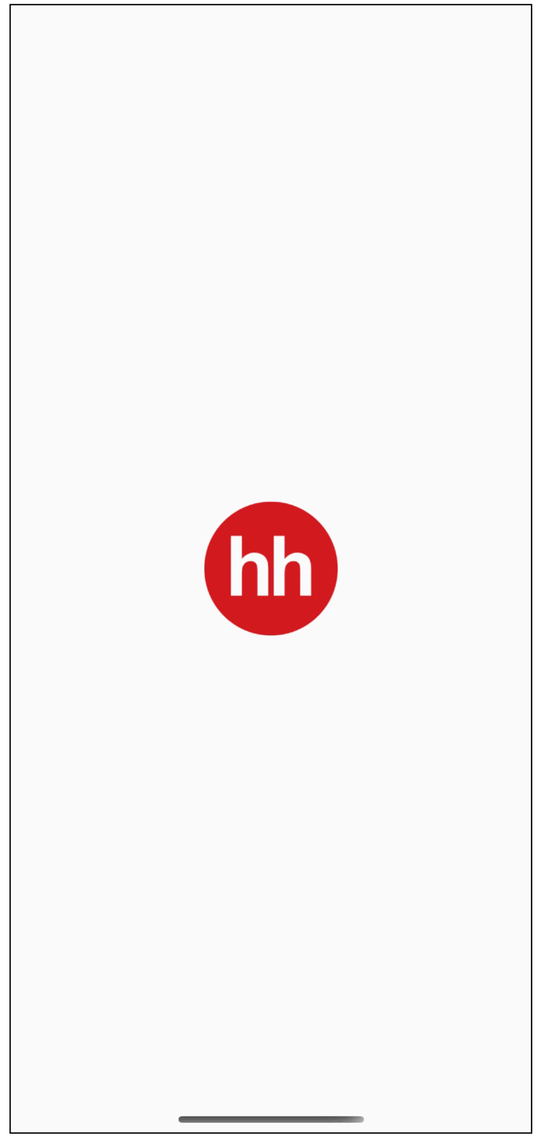
```
extension ScreenRootAnimation {

    public static let crossDissolve = Self.custom(
        ScreenRootTransitionAnimation(
            duration: 0.3,
            options: .transitionCrossDissolve
        )
    )
}
```

Смена корневого экрана

```
func openHomeScreen() {
    navigator.navigate { route in
        route.setRoot(to: screens.homeScreen(), animation: .crossDissolve)
    }
}
```

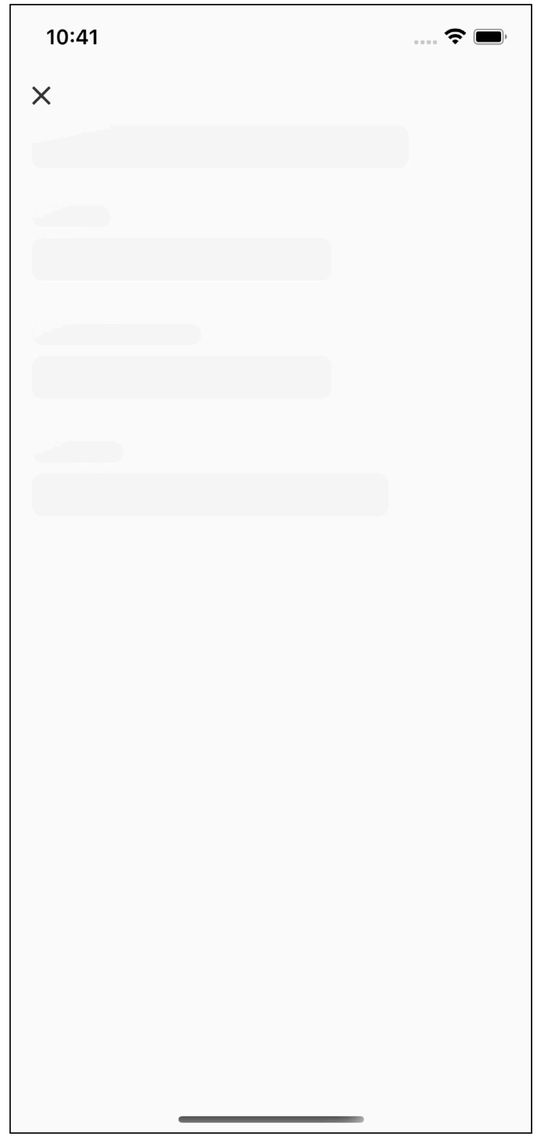
Смена корневого экрана

```
func showOnboardingRootScreen() {
    navigator.navigate { route in
        route.setRoot(to: screens.onboardingRootScreen(), animation: .onboarding)
    }
}
```



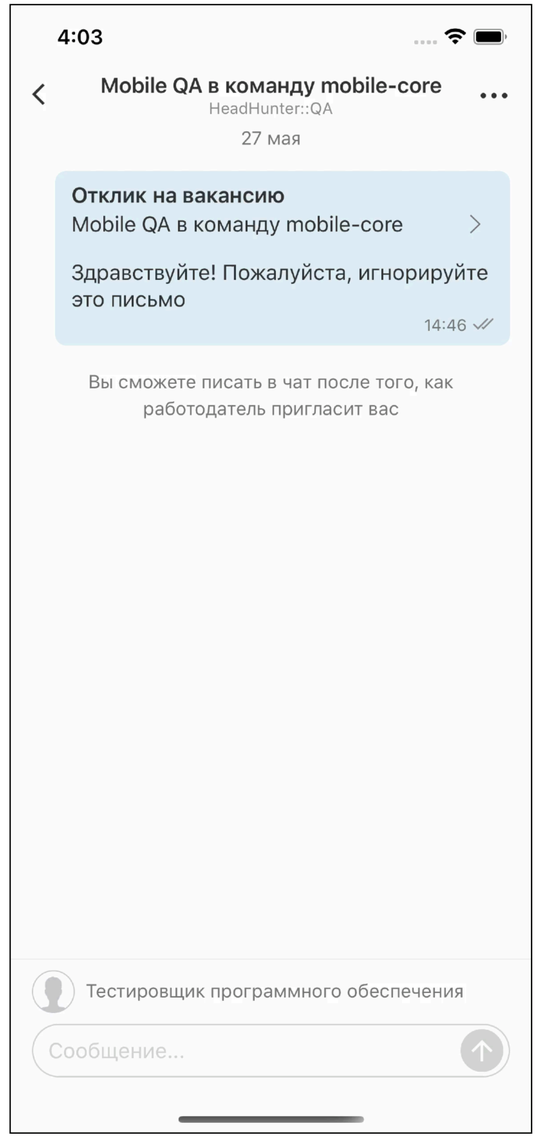
Изменение стека навигации

```
let container: UINavigationController

func showResumeEditScreen() {
    navigator.navigate(from: container) { route in
        route.replace(with: screens.resumeEditScreen(), animation: .crossDissolve)
    }
}
```



Модальный показ

```
func showFloatingActionSheet() {
    navigator.navigate(from: container) { route in
        route.present(
            screens
                .floatingActionSheetScreen()
                .withModalTransitioningDelegate(CardModalTransitionAnimator())
        )
    }
}
```



Модальный показ

```
func showFloatingActionSheet() {
    navigator.navigate(from: container) { route in
        route.present(
            screens
                .floatingActionSheetScreen()
                .withModalTransitioningDelegate(CardModalTransitionAnimator())
        )
    }
}
```

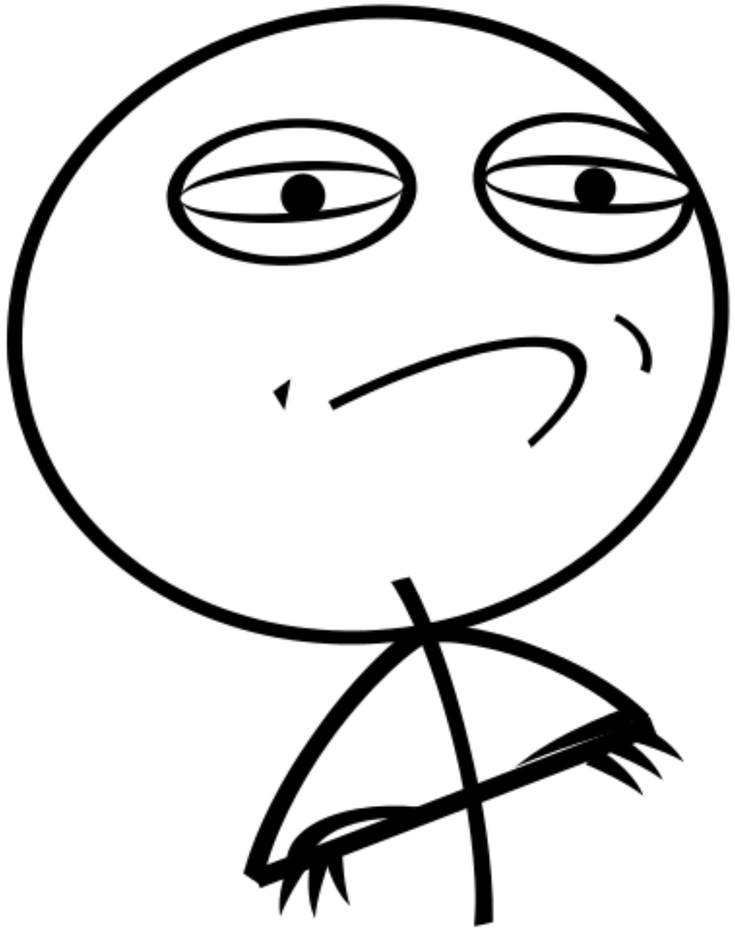
Модальный показ – UIKit

```
class ApplicantChatViewController: UIViewController {

    private let animator = CardModalTransitionAnimator()

    func showFloatingActionSheet() {
        let controller = FloatingActionSheetViewController()

        controller.transitioningDelegate = animator
        controller.modalPresentationStyle = .custom

        present(controller, animated: true)
    }
}
```

Модальный показ – UIKit

```
class ApplicantChatViewController: UIViewController {

    private let animator = CardModalTransitionAnimator()

    func showFloatingActionSheet() {
        let controller = FloatingActionSheetViewController()

        controller.transitioningDelegate = animator
        controller.modalPresentationStyle = .custom

        present(controller, animated: true)
    }
}
```

Граф навигации



Граф навигации

Обработка ошибок

Что имеется в виду?

Граф навигации — обработка ошибок

1. Не найден нужный экран
2. Отмена авторизации
3. Другие условия

Обработка ошибок

Не найден нужный экран

```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .resolve()
}
```

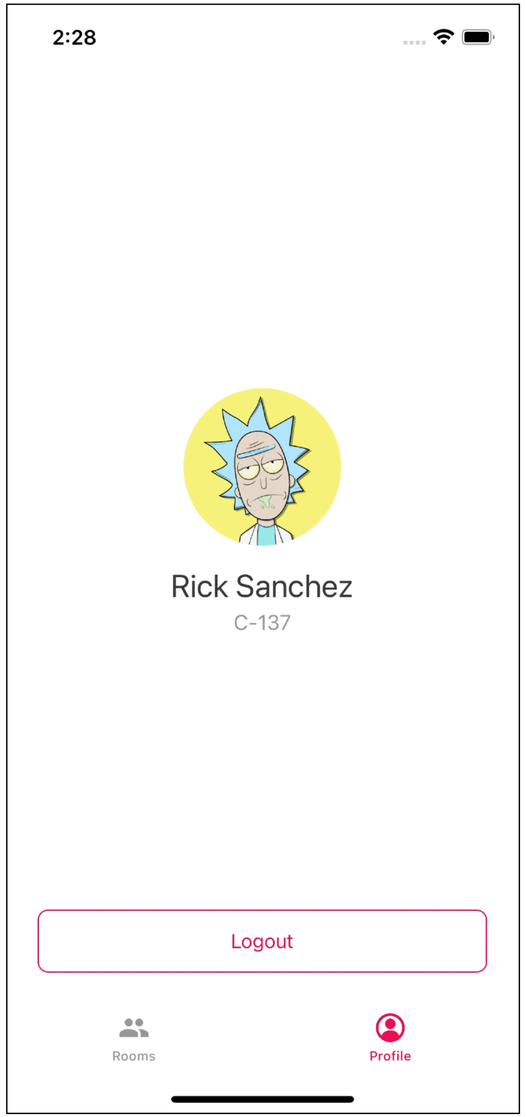
Обработка ошибок

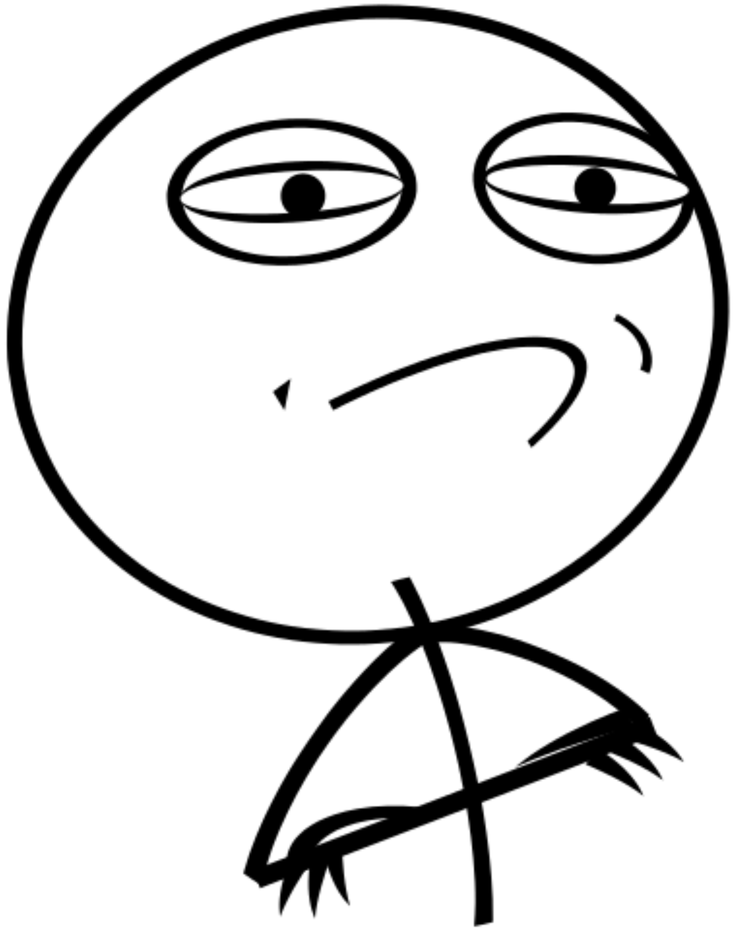
Не найден нужный экран

```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .fallback(
            to: showChatListRoute(roomID: roomID)
                .top(.stack)
                .push(screen)
        )
}
```

Обработка ошибок по типу

```
let mediaPicker = MediaPicker(source: .camera) { ... }

screenNavigator.navigate(from: self) { route in
    route
        .showMediaPicker(mediaPicker)
        .fallback { error, route in
            switch error {
            case is MediaPickerSourceAccessDeniedError:
                return route.showAlert(.cameraPermissionRequired)

            case is UnavailableMediaPickerSourceError:
                return route.showAlert(.unavailableMediaSource)

            ...
            }
        }
}
```





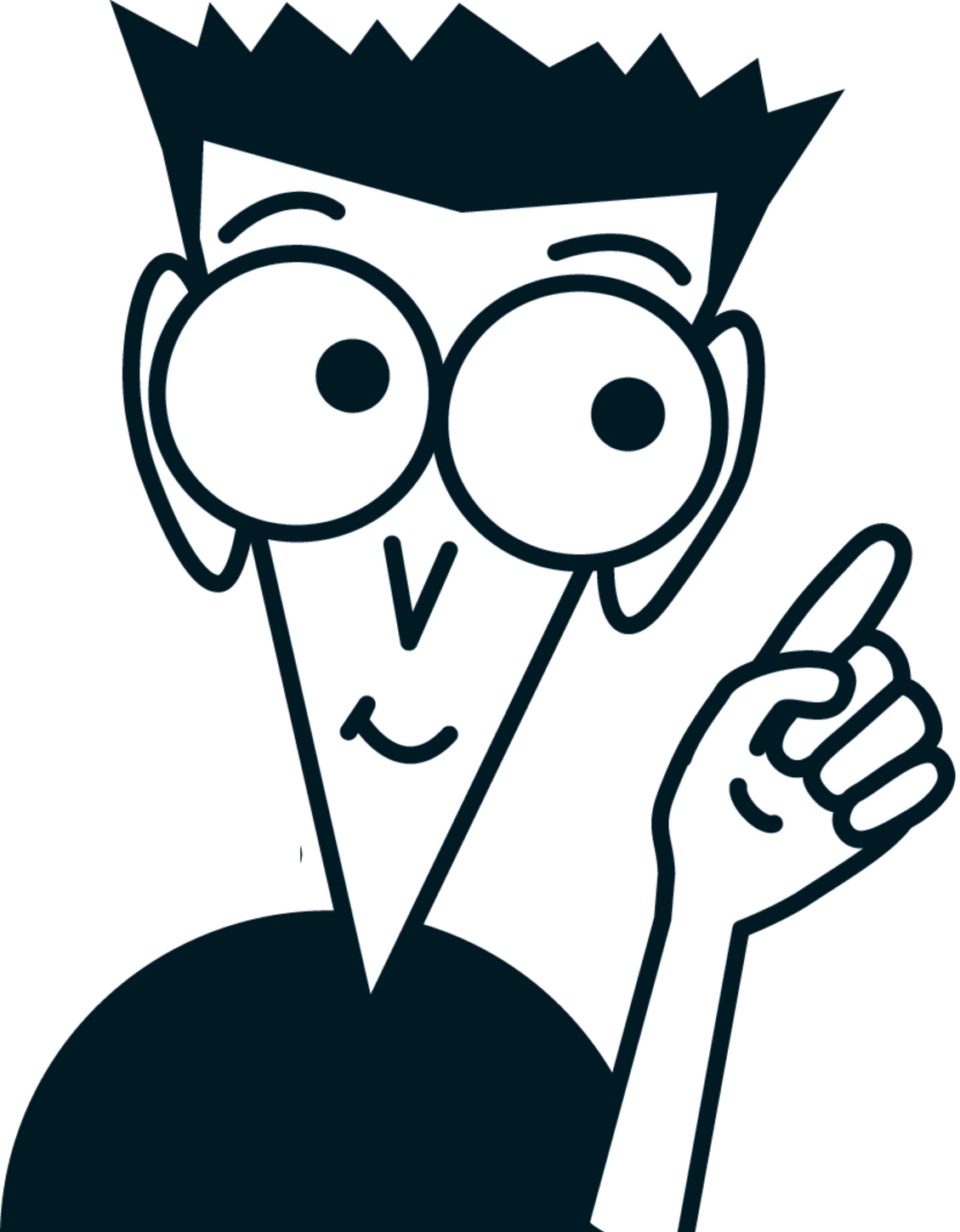
Граф навигации

интерсепторы

Что имеется в виду?

Граф навигации — интерсепторы

1. Запрос авторизации
2. Различные условия



Интерсепторы = действие навигации

Интерсепторы

Авторизация

```
struct ScreenAuthorizeAction<Container: UIViewController>: ScreenAction {

    typealias Output = Container

    let services: ScreenAuthorizeActionServices
    let screens: ScreenAuthorizeActionScreens

    init(...) { ... }

    func perform(
        container: Container,
        navigator: ScreenNavigator,
        completion: @escaping Completion
    ) { ... }
}
```

```
struct ScreenAuthorizeAction<Container: UIViewController>: ScreenAction {

    func perform(
        container: Container,
        navigator: ScreenNavigator,
        completion: @escaping Completion
    ) {
        navigator.logInfo("Checking authorization")

        if services.authorizationService().isAuthorized {
            completion(.success(container))
        } else {
            navigator.navigate(
                to: screens.showAuthorizationRoute { isAuthorized in
                    if isAuthorized {
                        completion(.success(container))
                    } else {
                        completion(.failure(ScreenCanceledError(for: self)))
                    }
                }
            )
        }
    }
}
```

```
struct ScreenAuthorizeAction<Container: UIViewController>: ScreenAction {

    func perform(
        container: Container,
        navigator: ScreenNavigator,
        completion: @escaping Completion
    ) {
        navigator.logInfo("Checking authorization")

        if services.authorizationService().isAuthorized {
            completion(.success(container))
        } else {
            navigator.navigate(
                to: screens.showAuthorizationRoute { isAuthorized in
                    if isAuthorized {
                        completion(.success(container))
                    } else {
                        completion(.failure(ScreenCanceledError(for: self)))
                    }
                }
            )
        }
    }
}
```

```
struct ScreenAuthorizeAction<Container: UIViewController>: ScreenAction {

    func perform(
        container: Container,
        navigator: ScreenNavigator,
        completion: @escaping Completion
    ) {
        navigator.logInfo("Checking authorization")

        if services.authorizationService().isAuthorized {
            completion(.success(container))
        } else {
            navigator.navigate(
                to: screens.showAuthorizationRoute { isAuthorized in
                    if isAuthorized {
                        completion(.success(container))
                    } else {
                        completion(.failure(ScreenCanceledError(for: self)))
                    }
                }
            )
        }
    }
}
```

```
struct ScreenAuthorizeAction<Container: UIViewController>: ScreenAction {

    func perform(
        container: Container,
        navigator: ScreenNavigator,
        completion: @escaping Completion
    ) {
        navigator.logInfo("Checking authorization")

        if services.authorizationService().isAuthorized {
            completion(.success(container))
        } else {
            navigator.navigate(
                to: screens.showAuthorizationRoute { isAuthorized in
                    if isAuthorized {
                        completion(.success(container))
                    } else {
                        completion(.failure(ScreenCanceledError(for: self)))
                    }
                }
            )
        }
    }
}
```

Интерсепторы

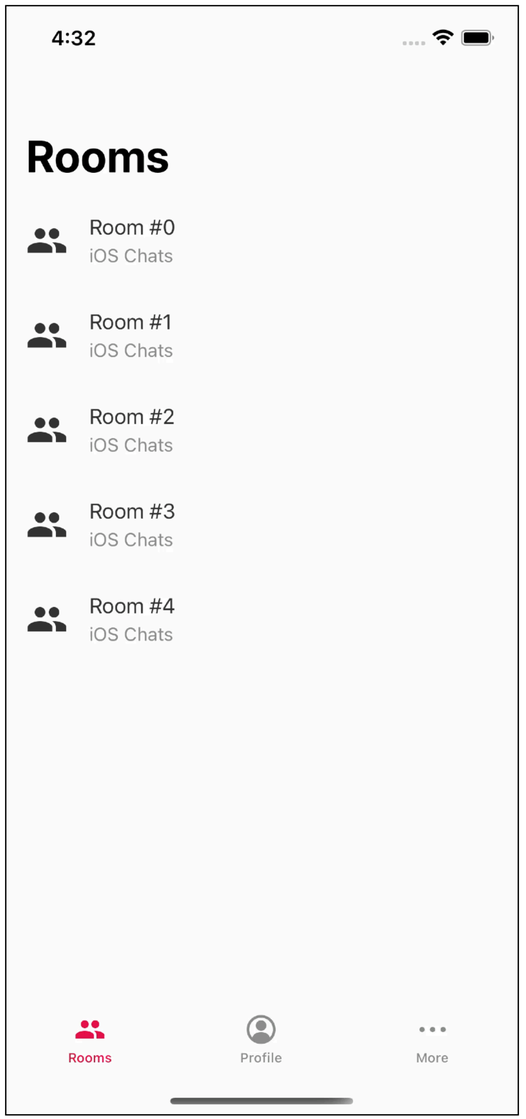
Extension

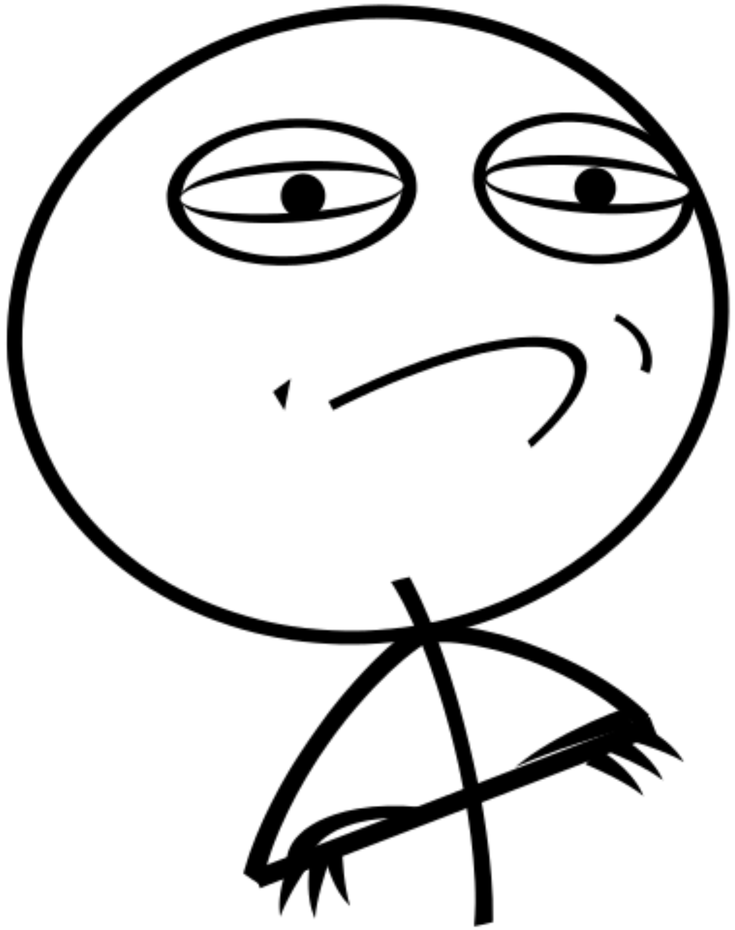
```
extension ScreenThenable where Current: UIViewController {

    func authorize(
        services: ScreenAuthorizeActionServices,
        screens: ScreenAuthorizeActionScreens
    ) -> Self {
        then(
            ScreenAuthorizeAction<Current>(
                services: services,
                screens: screens
            )
        )
    }
}
```

Интерсепторы

Использование

```
navigator.navigate(from: self) { route in

    route

        .authorize(services: services, screens: screens)

        .present(screens.chatListScreen(roomID: roomID).withStackContainer())

}
```





Граф навигации

Deep links

Что имеется в виду?

Граф навигации — deep links

1. «Глубокая» навигация по ссылке или push-уведомлению
2. Возможность прописать навигацию независимо от состояния

Переход в чат

`nivelir://chat?room_id=1&chat_id=1`

↓

Открытие домашнего экрана с UITabBarController

↓

Переключение на таб со списком комнат

↓

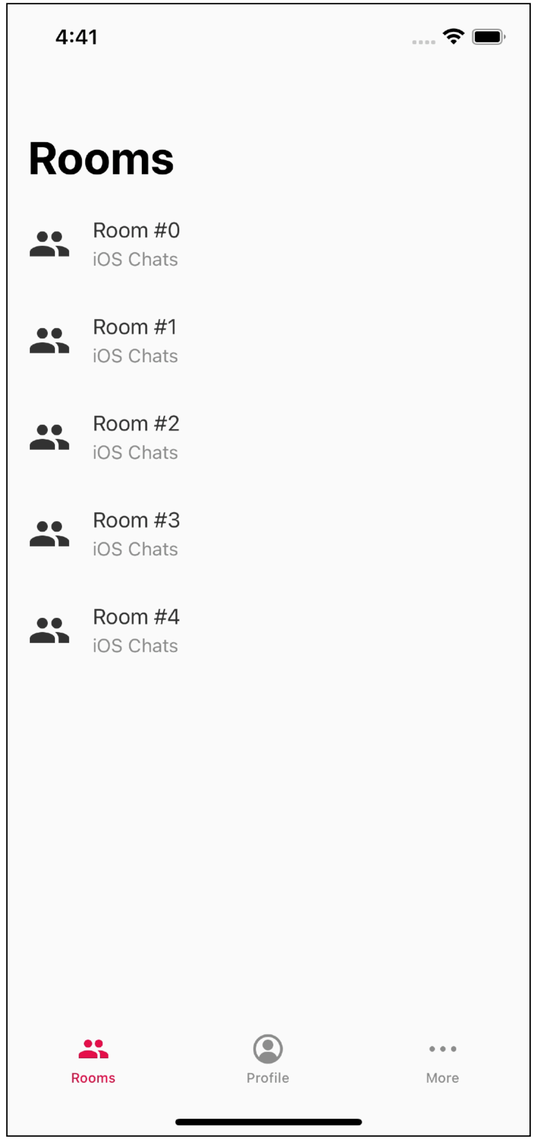
Проверка авторизации

↓

Открытие списка чатов

↓

Открытие чата



Сборка ScreenRoute

nivelir://chat?room_id=1&chat_id=1

↓

Открытие домашнего экрана с UITabBarController

↓

Переключение на таб со списком комнат

↓

Проверка авторизации

↓

Открытие списка чатов

↓

Открытие чата

```
func showHomeRoute() → ScreenWindowRoute {

    ScreenWindowRoute()

        .setRoot(to: homeScreen())

        .makeKeyAndVisible()

}
```

Сборка ScreenRoute

nivelir://chat?room_id=1&chat_id=1

↓

Открытие домашнего экрана с UITabBarController

↓

Переключение на таб со списком комнат

↓

Проверка авторизации

↓

Открытие списка чатов

↓

Открытие чата

```
func showRoomListRoute() → ScreenWindowRoute {

    ScreenWindowRoute()

        .last(.container(key: roomListScreen().key))

        .makeVisible()

        .fallback(to: showHomeRoute())

}
```

Сборка ScreenRoute

nivelir://chat?room_id=1&chat_id=1

↓

Открытие домашнего экрана с UITabBarController

↓

Переключение на таб со списком комнат

↓

Проверка авторизации

↓

Открытие списка чатов

↓

Открытие чата

```
func showChatListRoute(roomID: Int) → ScreenWindowRoute {

    let screen = chatListScreen(roomID: roomID)


    return ScreenWindowRoute()

        .last(.container(key: screen.key))

        .makeVisible()

        .fallback(

            to: showRoomListRoute()

                .top(.container)

                .authorize(services: services, screens: self)

                .present(screen.withStackContainer())

        )
}
```

Сборка ScreenRoute

nivelir://chat?room_id=1&chat_id=1

↓

Открытие домашнего экрана с UITabBarController

↓

Переключение на таб со списком комнат

↓

Проверка авторизации

↓

Открытие списка чатов

↓

Открытие чата

```
func showChatRoute(roomID: Int, chatID: Int) → ScreenWindowRoute {
    let screen = chatScreen(roomID: roomID, chatID: chatID)

    return ScreenWindowRoute()
        .last(.container(key: screen.key))
        .makeVisible()
        .refresh()
        .fallback(
            to: showChatListRoute(roomID: roomID)
                .top(.stack)
                .push(screen)
        )
}
```

Deeplinks

Deeplink для чатов

```
struct ChatDeeplink: Deeplink {

    let roomID: Int

    let chatID: Int

    func navigate(

        screens: Screens,

        navigator: ScreenNavigator,

        handler: DeeplinkHandler

    ) throws {

        navigator.navigate(

            to: screens.showChatRoute(

                roomID: roomID,

                chatID: chatID

            )

        )

    }

}
```

Deeplinks

Данные для диплинка

```
struct ChatDeeplinkPayload: Decodable {

    enum CodingKeys: String, CodingKey {
        case roomID = "room_id"
        case chatID = "chat_id"
    }

    let roomID: Int
    let chatID: Int
}
```

Deeplinks

Поддержка URL

```
extension ChatDeeplink: URLDeeplink {

    static func url(
        scheme: String?,
        host: String?,
        path: [String],
        query: ChatDeeplinkPayload?,
        context: Any
    ) throws -> ChatDeeplink? {
        guard let payload = query, scheme == "nivelir", host == "chat" else {
            return nil
        }

        return Self(roomID: payload.roomID, chatID: payload.chatID)
    }
}
```

```
nivelir://chat?room_id=1&chat_id=1
```

Deeplinks

Поддержка URL

```
extension ChatDeeplink: URLDeeplink {

    static func url(
        scheme: String?,
        host: String?,
        path: [String],
        query: ChatDeeplinkPayload?,
        context: Any
    ) throws -> ChatDeeplink? {
        guard let payload = query, scheme == "nivelir", host == "chat" else {
            return nil
        }

        return Self(roomID: payload.roomID, chatID: payload.chatID)
    }
}
```

```
nivelir://chat?room_id=1&chat_id=1
```

Deeplinks

Поддержка URL

```
func scene(_ scene: UIScene, openURLContexts URLContexts: Set<UIOpenURLContext>) {
    guard let url = URLContexts.first?.url else {
        return
    }

    serviceFactory
      .deeplinkManager()
      .handleURLIfPossible(url, context: serviceFactory)
}
```

Deeplinks

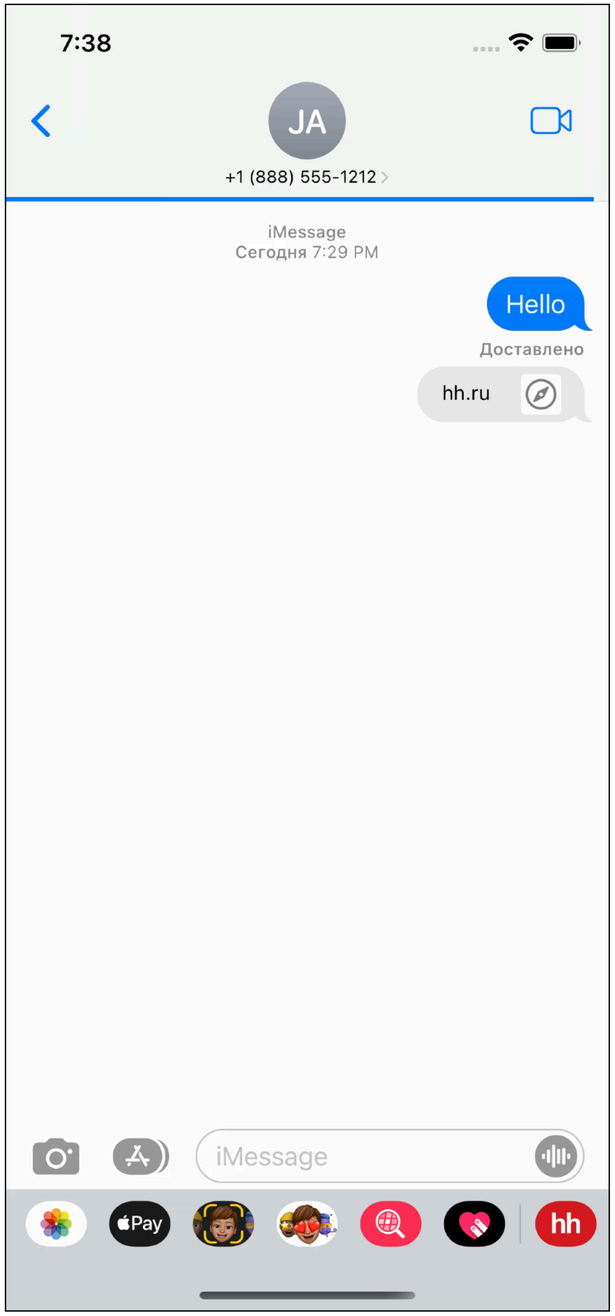
Менеджер

```
DeeplinkManager(
    deeplinkTypes: [
        ChatDeeplink.self
    ],
    navigator: screenNavigator()
)
```

Deeplinks

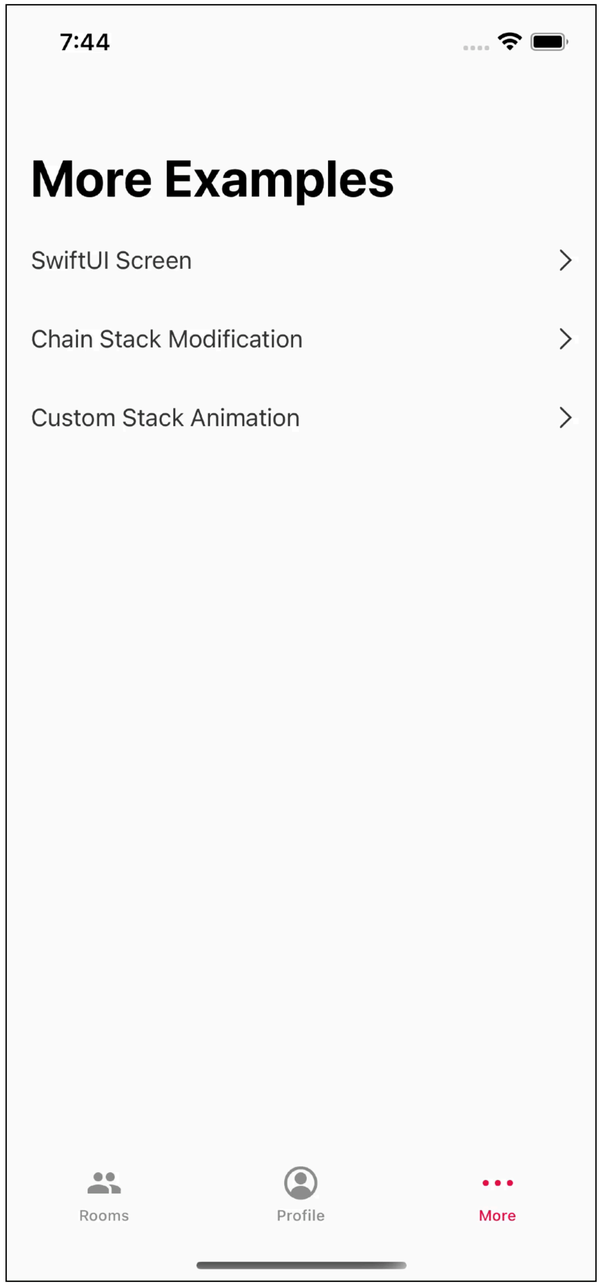
Менеджер – активация

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

    deeplinkManager.activate(screens: screens)
}
```



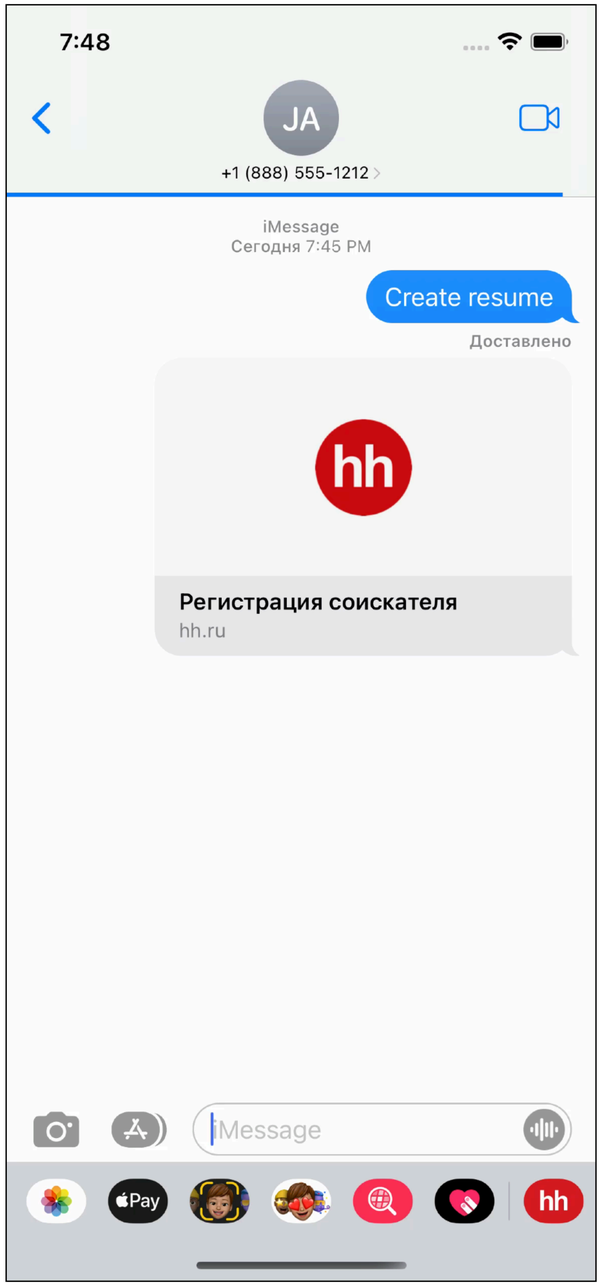
Deeplinks

Проверяем 🚀



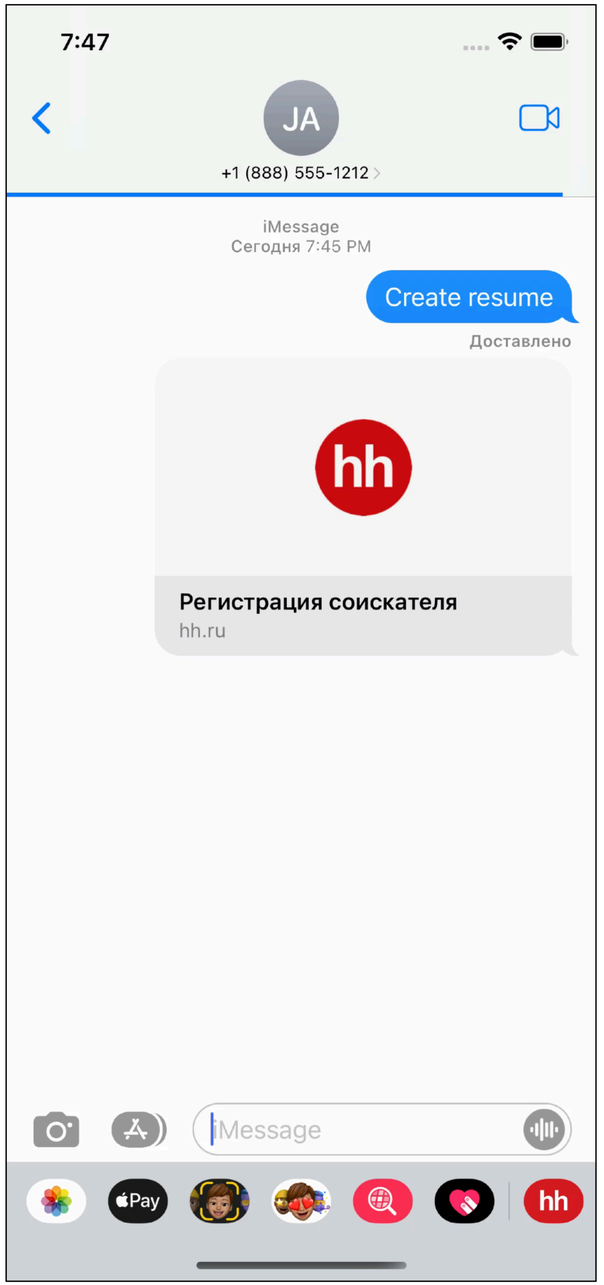
Deeplinks

Бизнес-логика



Deeplinks

Бизнес-логика



Deeplinks

Бизнес-логика

```
public struct CreateResumeDeeplink: Deeplink {

    private let context: ResumeServiceFactory

    public func navigate(
        screens: ResumeScreenFactory,
        navigator: ScreenNavigator,
        handler: DeeplinkHandler
    ) throws {
        ...
    }
}
```

```
let route = screens
    .showResumeTabRoute()
    .top(.stack)
    .popToRoot()
    .top(.container, route: screens.authorizeRoute())
    .try(
        action: CheckResumeCountAction(
            resumeListService: context.resumeListService(),
            screens: screens
        ),
        done: { resumeCount, route in
            handleResumeCount(resumeCount, route: route)
        }
    )
    .fallback { error, route in
        if error is ScreenCanceledError {
            return route
        }
        return route.top(.container).showAlert(.error(error.mapToHHSDKErrorModel()))
    }

navigator.navigate(to: route)
```

```
let route = screens
    .showResumeTabRoute()
    .top(.stack)
    .popToRoot()
    .top(.container, route: screens.authorizeRoute())
    .try(
        action: CheckResumeCountAction(
            resumeListService: context.resumeListService(),
            screens: screens
        ),
        done: { resumeCount, route in
            handleResumeCount(resumeCount, route: route)
        }
    )
    .fallback { error, route in
        if error is ScreenCanceledError {
            return route
        }
        return route.top(.container).showAlert(.error(error.mapToHHSDKErrorModel()))
    }

navigator.navigate(to: route)
```

```
struct CheckResumeCountAction<Container: ScreenContainer>: ScreenAction {
    typealias Output = Int
    func perform(container: Container, avigator: ScreenNavigator, completion: @escaping Completion) {
        navigator.navigate(to: ScreenWindowRoute().showLoadingHUD())
        resumeListService
            .resumes()
            .handleEvents(
                receiveCompletion: { _ in
                    navigator.navigate(to: ScreenWindowRoute().hideHUD())
                }
            )
            .sink(
                receiveValue: { resumes in
                    completion(.success(resumes.resumesPage.items.count))
                },
                receiveFailure: { error in
                    completion(.failure(error))
                }
            ).store(in: cancellableBag)
    }
}
```

```
struct CheckResumeCountAction<Container: ScreenContainer>: ScreenAction {
    typealias Output = Int
    func perform(container: Container, avigator: ScreenNavigator, completion: @escaping Completion) {
        navigator.navigate(to: ScreenWindowRoute().showLoadingHUD())
        resumeListService
            .resumes()
            .handleEvents(
                receiveCompletion: { _ in
                    navigator.navigate(to: ScreenWindowRoute().hideHUD())
                }
            )
            .sink(
                receiveValue: { resumes in
                    completion(.success(resumes.resumesPage.items.count))
                },
                receiveFailure: { error in
                    completion(.failure(error))
                }
            ).store(in: cancellableBag)
    }
}
```

```
struct CheckResumeCountAction<Container: ScreenContainer>: ScreenAction {
    typealias Output = Int
    func perform(container: Container, avigator: ScreenNavigator, completion: @escaping Completion) {
        navigator.navigate(to: ScreenWindowRoute().showLoadingHUD())
        resumeListService
            .resumes()
            .handleEvents(
                receiveCompletion: { _ in
                    navigator.navigate(to: ScreenWindowRoute().hideHUD())
                }
            )
            .sink(
                receiveValue: { resumes in
                    completion(.success(resumes.resumesPage.items.count))
                },
                receiveFailure: { error in
                    completion(.failure(error))
                }
            ).store(in: cancellableBag)
    }
}
```

```
struct CheckResumeCountAction<Container: ScreenContainer>: ScreenAction {
    typealias Output = Int
    func perform(container: Container, avigator: ScreenNavigator, completion: @escaping Completion) {
        navigator.navigate(to: ScreenWindowRoute().showLoadingHUD())
        resumeListService
            .resumes()
            .handleEvents(
                receiveCompletion: { _ in
                    navigator.navigate(to: ScreenWindowRoute().hideHUD())
                }
            )
            .sink(
                receiveValue: { resumes in
                    completion(.success(resumes.resumesPage.items.count))
                },
                receiveFailure: { error in
                    completion(.failure(error))
                }
            ).store(in: cancellableBag)
    }
}
```

```
struct CheckResumeCountAction<Container: ScreenContainer>: ScreenAction {
    typealias Output = Int
    func perform(container: Container, avigator: ScreenNavigator, completion: @escaping Completion) {
        navigator.navigate(to: ScreenWindowRoute().showLoadingHUD())
        resumeListService
            .resumes()
            .handleEvents(
                receiveCompletion: { _ in
                    navigator.navigate(to: ScreenWindowRoute().hideHUD())
                }
            )
            .sink(
                receiveValue: { resumes in
                    completion(.success(resumes.resumesPage.items.count))
                },
                receiveFailure: { error in
                    completion(.failure(error))
                }
            ).store(in: cancellableBag)
    }
}
```

```
struct CheckResumeCountAction<Container: ScreenContainer>: ScreenAction {
    typealias Output = Int
    func perform(container: Container, avigator: ScreenNavigator, completion: @escaping Completion) {
        navigator.navigate(to: ScreenWindowRoute().showLoadingHUD())
        resumeListService
            .resumes()
            .handleEvents(
                receiveCompletion: { _ in
                    navigator.navigate(to: ScreenWindowRoute().hideHUD())
                }
            )
            .sink(
                receiveValue: { resumes in
                    completion(.success(resumes.resumesPage.items.count))
                },
                receiveFailure: { error in
                    completion(.failure(error))
                }
            ).store(in: cancellableBag)
    }
}
```

```
let route = screens
    .showResumeTabRoute()
    .top(.stack)
    .popToRoot()
    .top(.container, route: screens.authorizeRoute())
    .try(
        action: CheckResumeCountAction(
            resumeListService: context.resumeListService(),
            screens: screens
        ),
        done: { resumeCount, route in
            handleResumeCount(resumeCount, route: route)
        }
    )
    .fallback { error, route in
        if error is ScreenCanceledError {
            return route
        }
        return route.top(.container).showAlert(.error(error.mapToHHSDKErrorModel()))
    }

navigator.navigate(to: route)
```

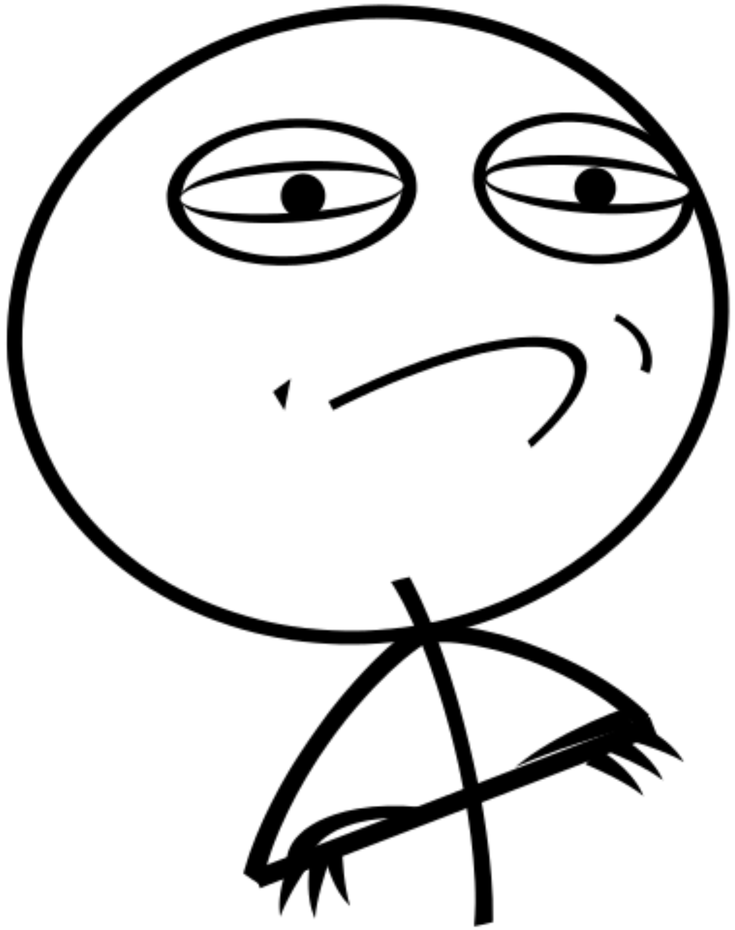
```
let route = screens
    .showResumeTabRoute()
    .top(.stack)
    .popToRoot()
    .top(.container, route: screens.authorizeRoute())
    .try(
        action: CheckResumeCountAction(
            resumeListService: context.resumeListService(),
            screens: screens
        ),
        done: { resumeCount, route in
            handleResumeCount(resumeCount, route: route)
        }
    )
    .fallback { error, route in
        if error is ScreenCanceledError {
            return route
        }
        return route.top(.container).showAlert(.error(error.mapToHHSDKErrorModel()))
    }

navigator.navigate(to: route)
```

```
let route = screens
    .showResumeTabRoute()
    .top(.stack)
    .popToRoot()
    .top(.container, route: screens.authorizeRoute())
    .try(
        action: CheckResumeCountAction(
            resumeListService: context.resumeListService(),
            screens: screens
        ),
        done: { resumeCount, route in
            handleResumeCount(resumeCount, route: route)
        }
    )
    .fallback { error, route in
        if error is ScreenCanceledError {
            return route
        }
        return route.top(.container).showAlert(.error(error.mapToHHSDKErrorModel()))
    }

navigator.navigate(to: route)
```

Поддерживаемые типы

1. URL – `URLDeeplink`
2. Push Notifications – `NotificationDeeplink`
3. Shortcuts – `ShortcutDeeplink`


Масштабируемость



Масштабируемость

Многомодульность

Что имеется в виду?

Масштабируемость — многомодульность

1. Навигация между фиче-модулями
2. Шаринг навигации через DI

Многомодульность

DI

```
struct Screens {

    func homeScreen() -> AnyTabsScreen {
        HomeScreen(
            services: services,
            screens: self
        ).eraseToAnyScreen()
    }

    func roomListScreen() -> AnyModalScreen {
        RoomListScreen(
            services: services,
            screens: self
        ).eraseToAnyScreen()
    }
}
```

Многомодульность

DI

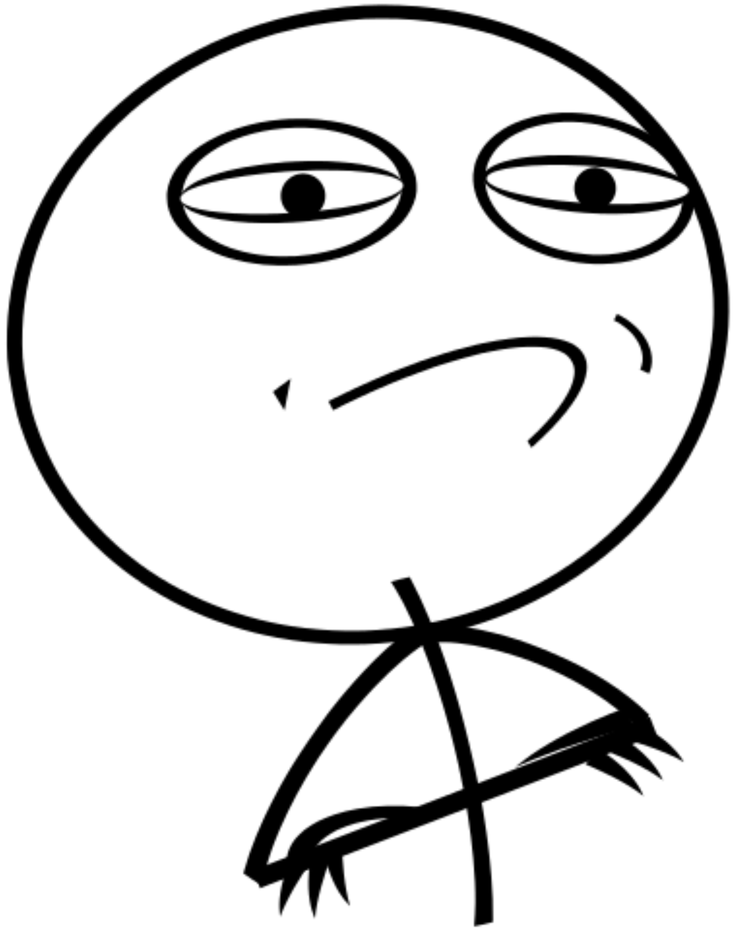
```
struct Screens {

    func homeScreen() -> AnyTabsScreen {
        HomeScreen(
            services: services,
            screens: self
        ).eraseToAnyScreen()
    }

    func roomListScreen() -> AnyModalScreen {
        RoomListScreen(
            services: services,
            screens: self
        ).eraseToAnyScreen()
    }
}
```

```
public typealias AnyModalScreen = AnyScreen<UIViewController>

public typealias AnyStackScreen = AnyScreen<UINavigationController>

public typealias AnyTabsScreen = AnyScreen<UITabBarController>
```

Многомодульность

DI

```
struct Screens {

    func showHomeRoute() → ScreenWindowRoute {
        ScreenWindowRoute()
            .setRoot(to: homeScreen(), animation: .crossDissolve)
            .makeKeyAndVisible()
    }

    func showRoomListRoute() → ScreenWindowRoute {
        ScreenWindowRoute()
            .last(.container(key: roomListScreen().key))
            .makeVisible()
            .fallback(to: showHomeRoute())
    }
}
```



Масштабируемость

Постепенная миграция

Что имеется в виду?

Масштабируемость — постепенная миграция

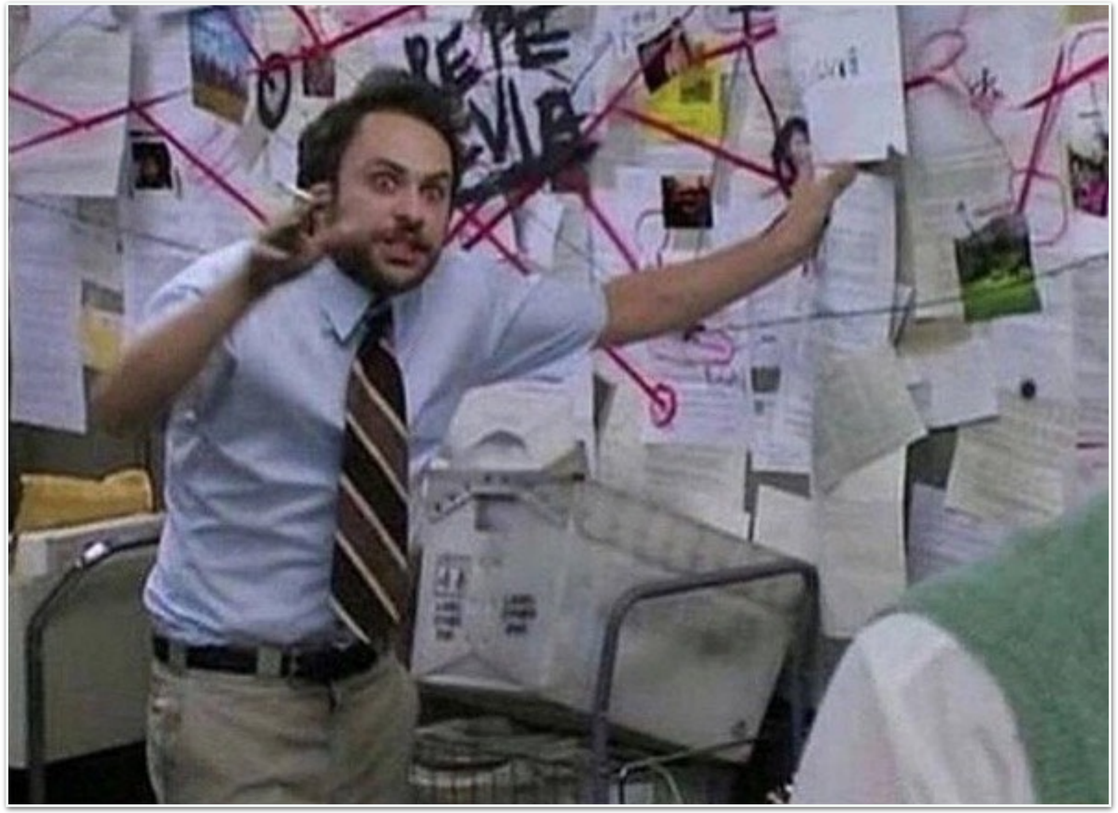
1. Плавный переход к фрейворку
2. Не требуется делать фича-фриз для миграции

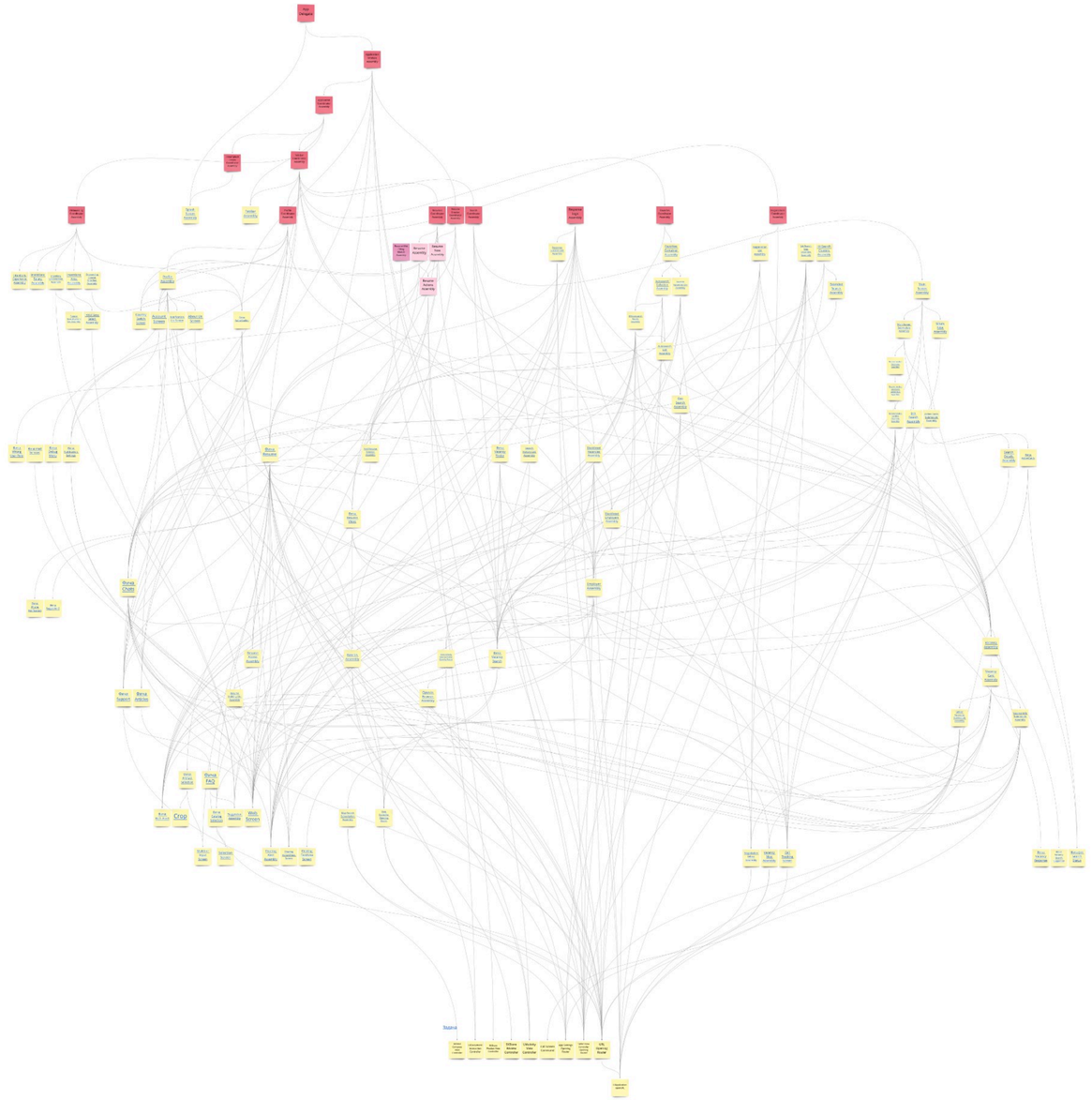
Как мигрировать?

1. Начинайте от листовых экранов
2. `UIViewController → Screen`
3. Личный опыт

Как мигрировать?

1 **Начинайте от листовых экранов**





Как мигрировать?

2 UIViewController → Screen

```
struct LegacyAssembly {

    func assembly(roomID: Int, chatID: Int) → ChatViewController {

        ChatViewController(roomID: roomID, chatID: chatID)

    }

}


extension ChatViewController: Screen { }


let screen = LegacyAssembly()

    .assembly(roomID: 1, chatID: 1)

    .eraseToAnyModalScreen()
```

Как мигрировать?

3 **Личный опыт**

Два подхода в проекте

↓

Большую часть фабрик (≈100) переписали
за неделю силой 6 разработчиков

↓

Перевели все 43 диплинка за 2 квартала

↓

Вся навигация в приложениях через Nivelir

Больше возможностей

Media Picker

Document Preview

Screen Observers

Store Product

Store Review

Call

Mail

Nivelir

Sharing

DSL

Action Sheet

Logger

Screen Decorator

Deeplink Manager

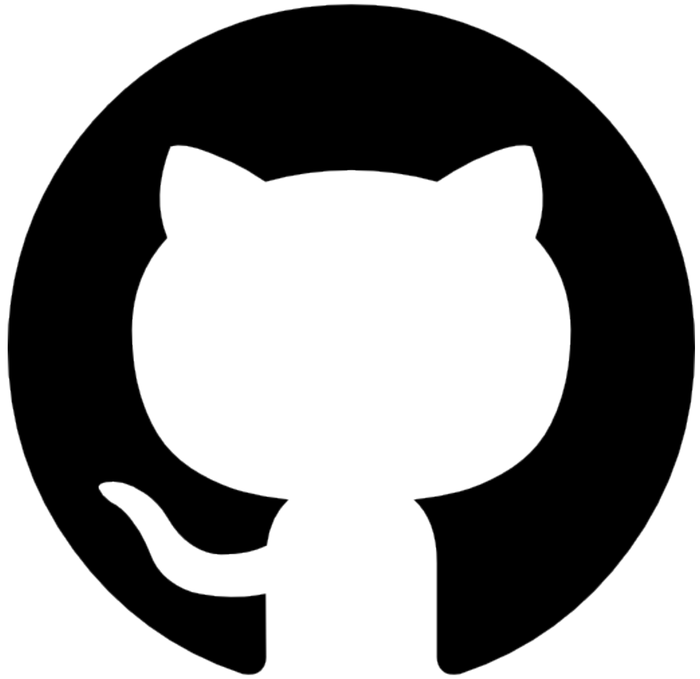
Custom Animations

HUD

Alert


Какие
Итоги

Фреймворк/Критерий	Badoo	route-composer	Nivelir
Локальная навигация	❌	❌	✅
Цепочки открытия	❌	✅	✅
Поиск открытого экрана	✅/❌	✅	✅
Удобный DSL	✅	❌	✅
Строгость типизации	❌	✅/❌	✅
Кастомные анимации	❌	✅/❌	✅
Обработка ошибок	❌	✅	✅
Интерцепторы	❌	✅	✅
Deep links	✅	✅	✅
Многомодульность	✅	✅	✅
Постепенная миграция	✅	❌	✅



Примеры кода

На GitHub

Поддержка в чате

Telegram

Спросите меня о чём-нибудь ;)

1. Навигация в iOS
2. История навигации в hh.ru
3. Критерии для фреймворка навигации
4. Возможности Nivelir

Ссылка на слайды