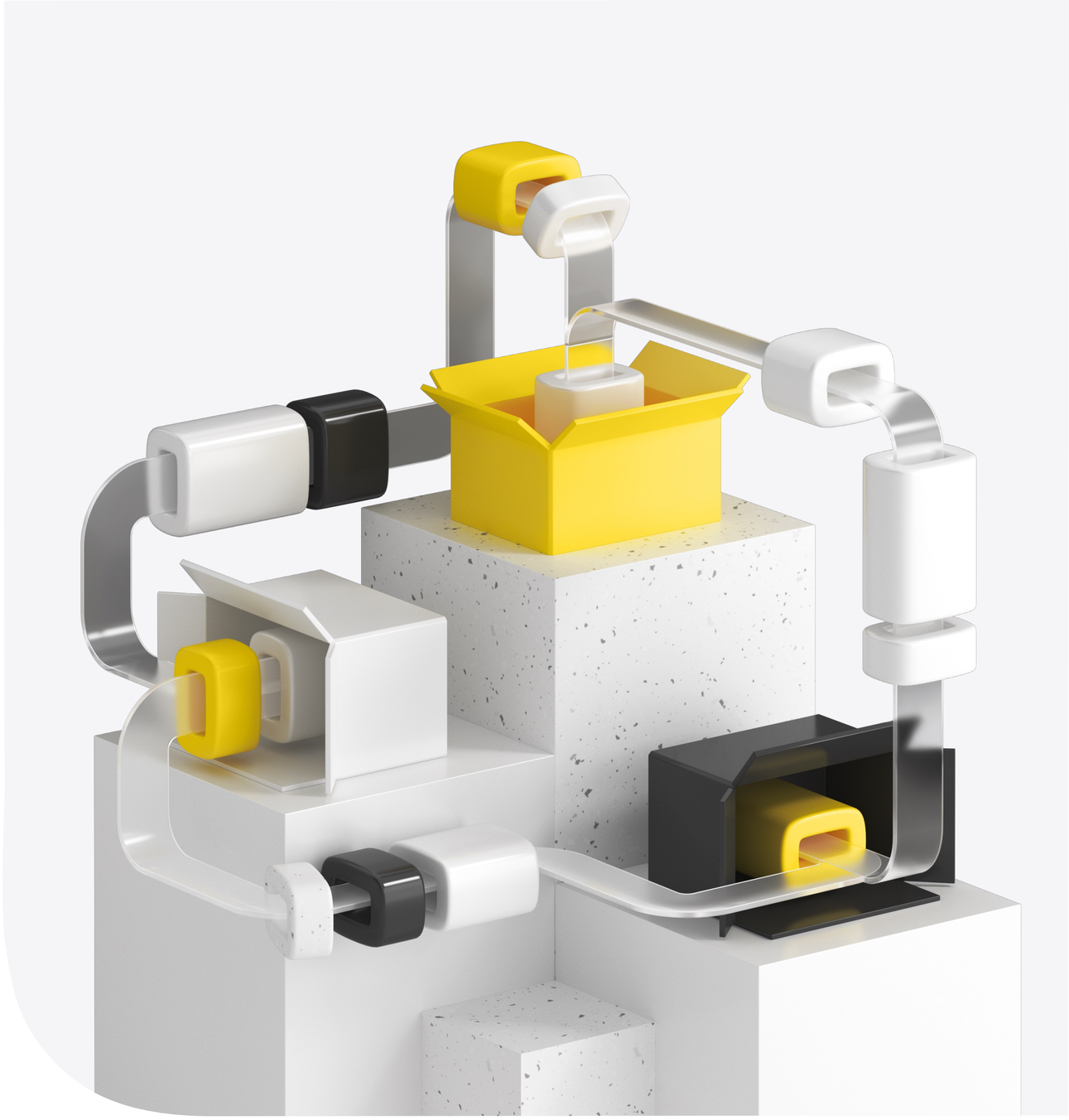
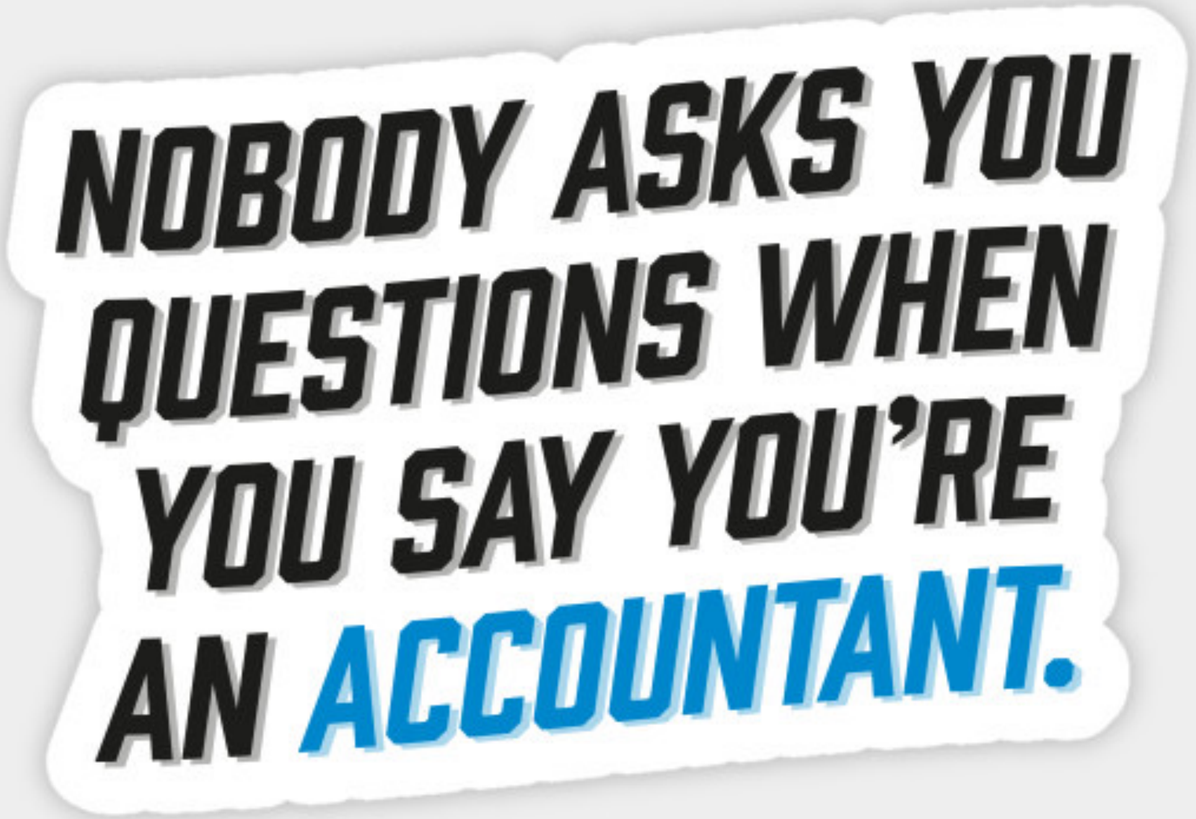
IT's TINKOFF

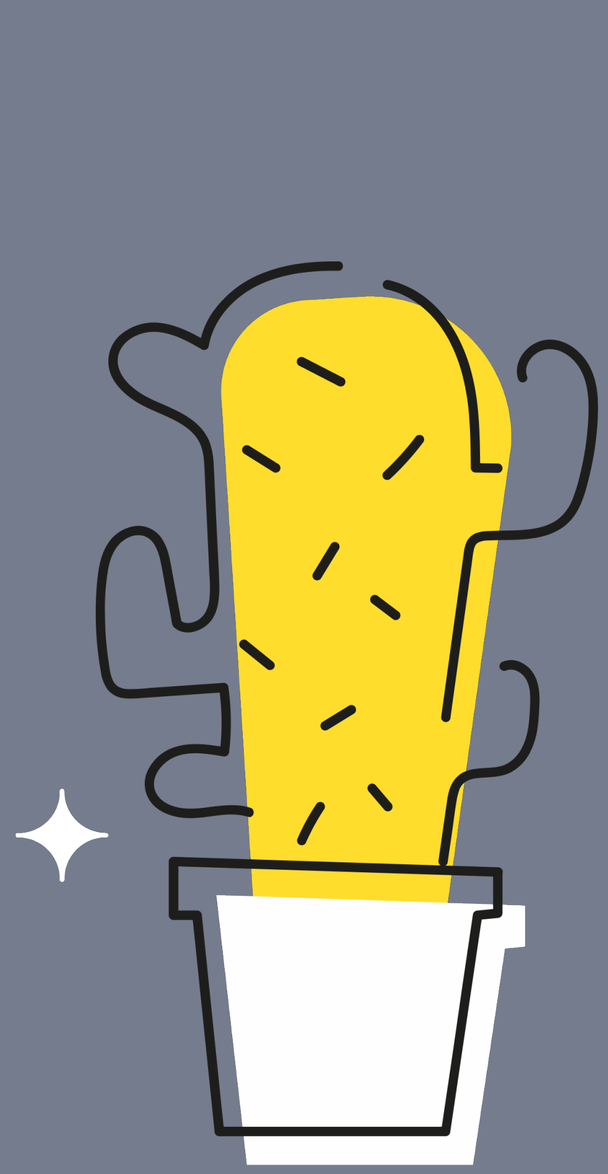
Навигация в модульном проекте

Проблемы навигации
и декларативный подход к решению




NOBODY ASKS YOU
QUESTIONS WHEN
YOU SAY YOU'RE
AN ACCOUNTANT.

- Подходы к навигации в iOS
- Проблемы FlowCoordinator
- Какие задачи
- Возможное решение



Сегодня

Сначала был UIViewController

- Навигация внутри UIViewController
- Монолитное приложение
- Массивные VC

Попытка выделить роутинг в MVC/MVP/MVVM/VIPER

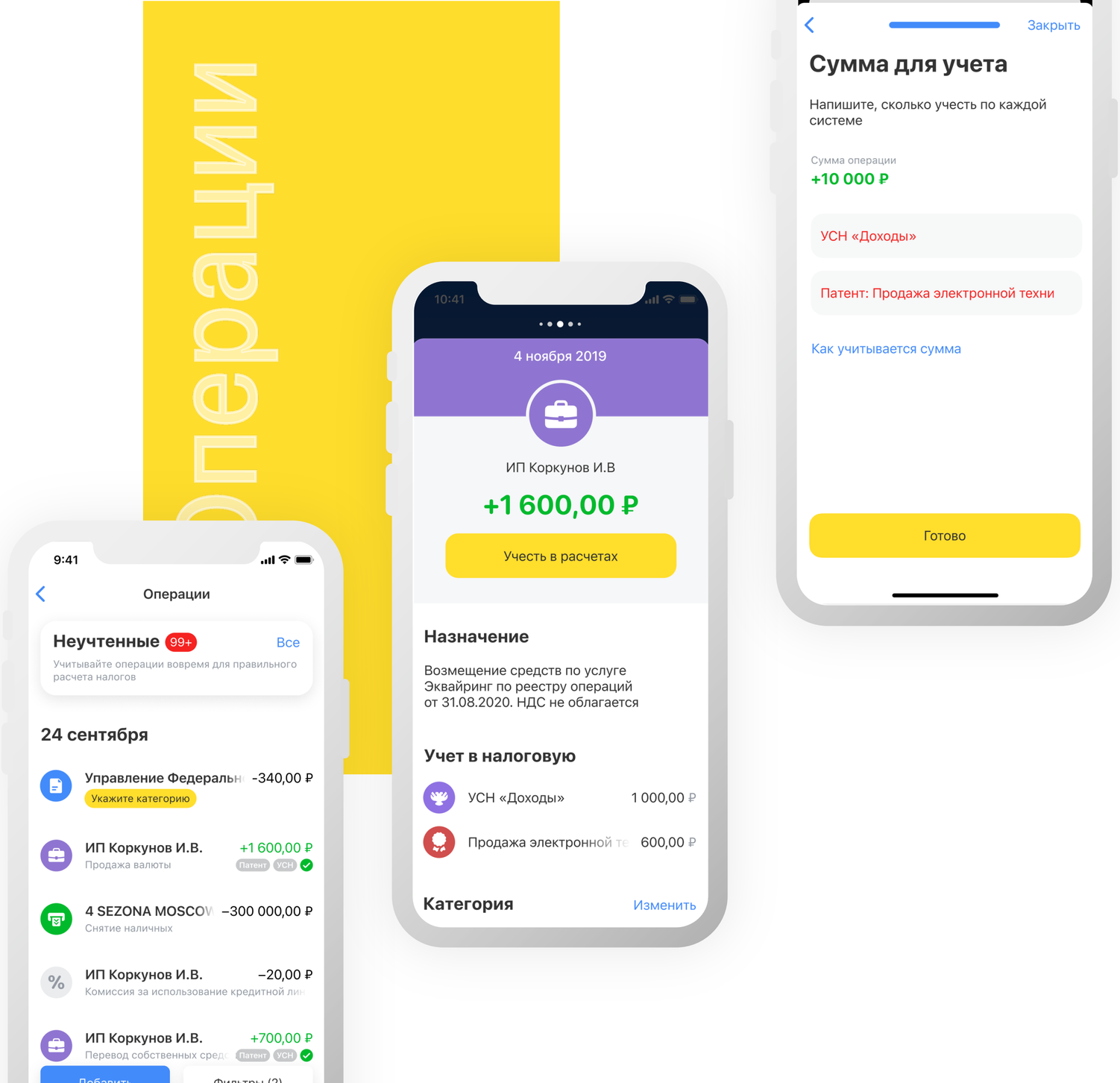
01

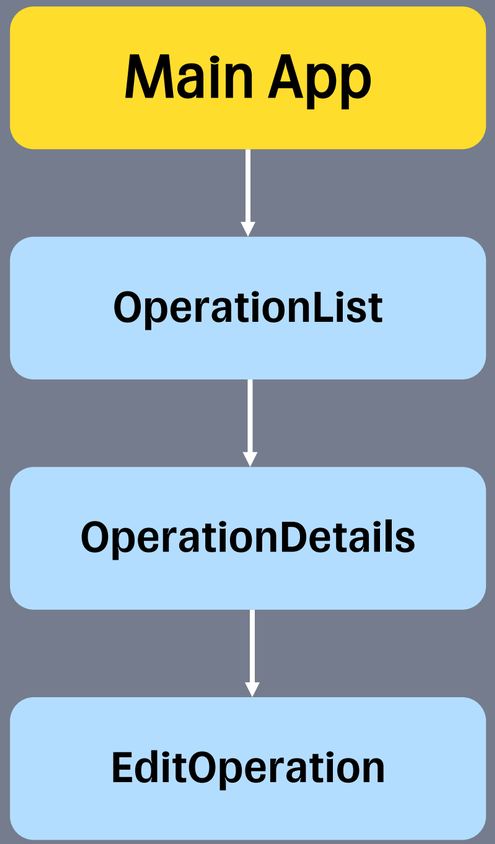
Появилась абстракция
со своей ответственностью

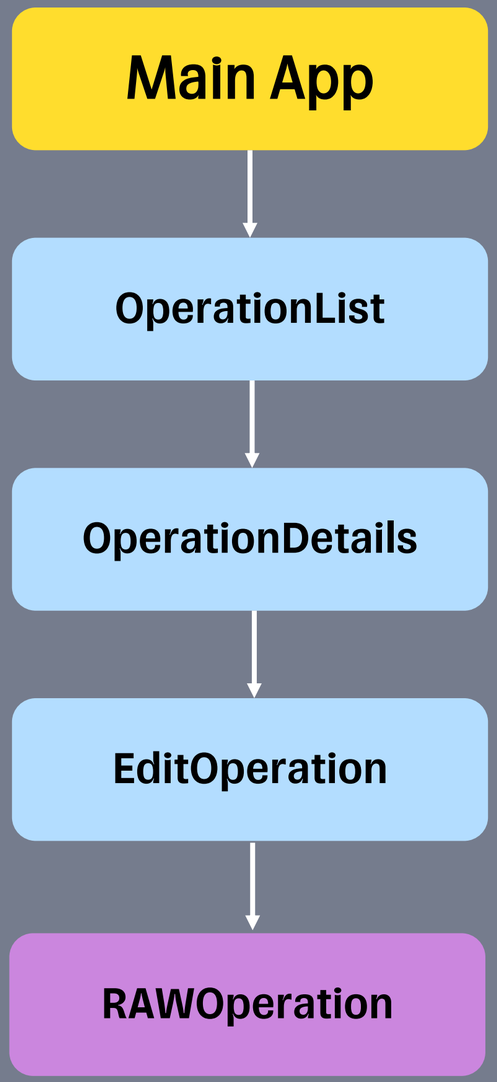
02

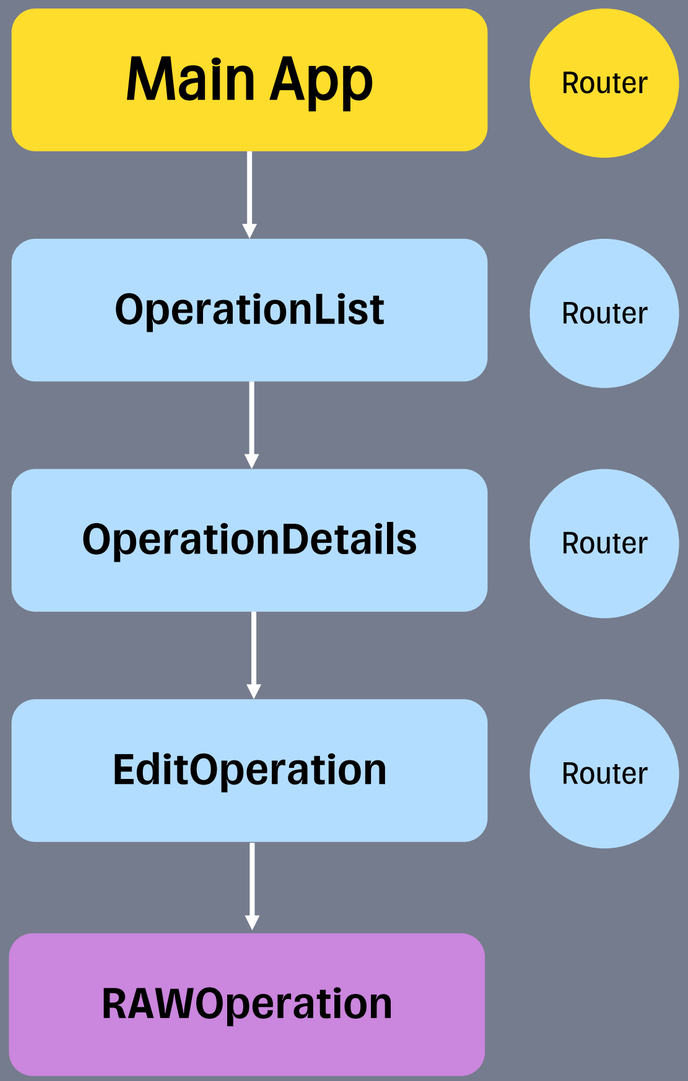
Начали распиливать
монолит

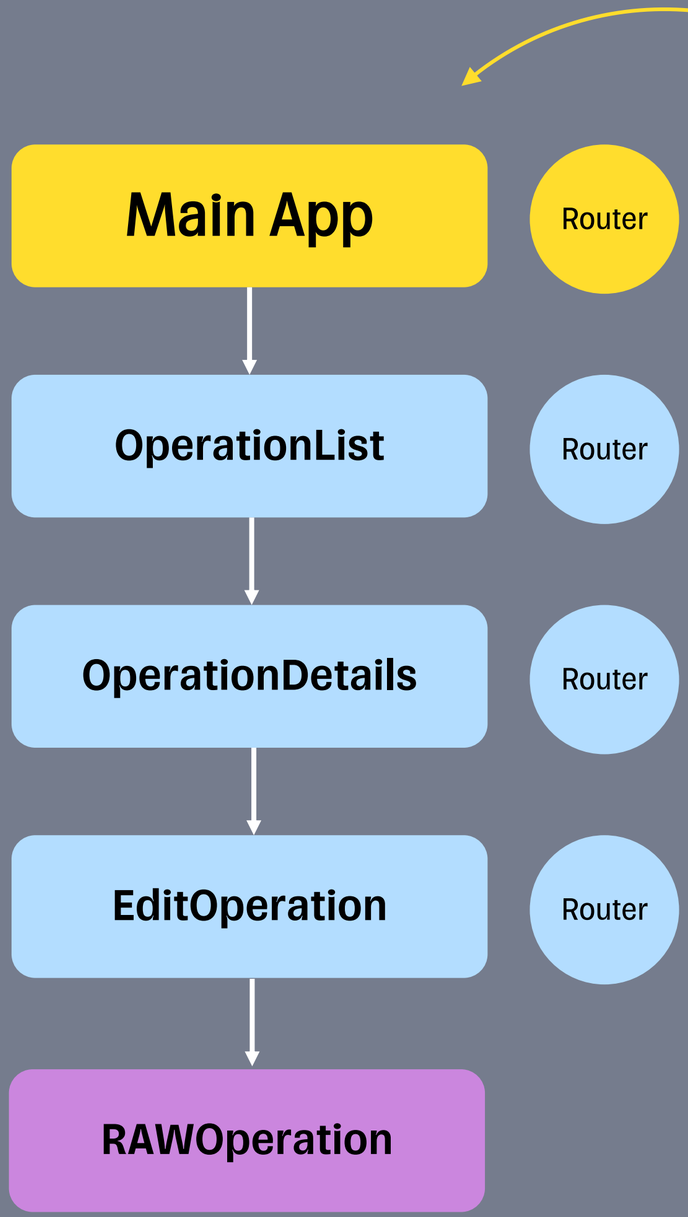
Абстрактный пример Модулей и навигации




Main App
OperationList
OperationDetails
EditOperation


Main App
OperationList
OperationDetails
EditOperation
RAWOperation


Main App
Router
OperationList
Router
OperationDetails
Router
EditOperation
Router
RAWOperation

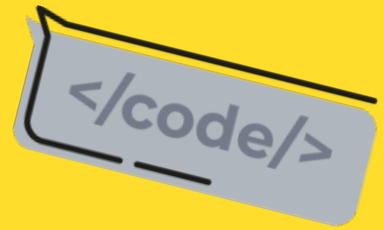



```
class Router: OperationListRouter, OperationDetailsRouter, EditOperationRouter {

    func navigateToOperation(id: String) {

    }

    func navigateToEditOperation(id: String) {

    }
}
```

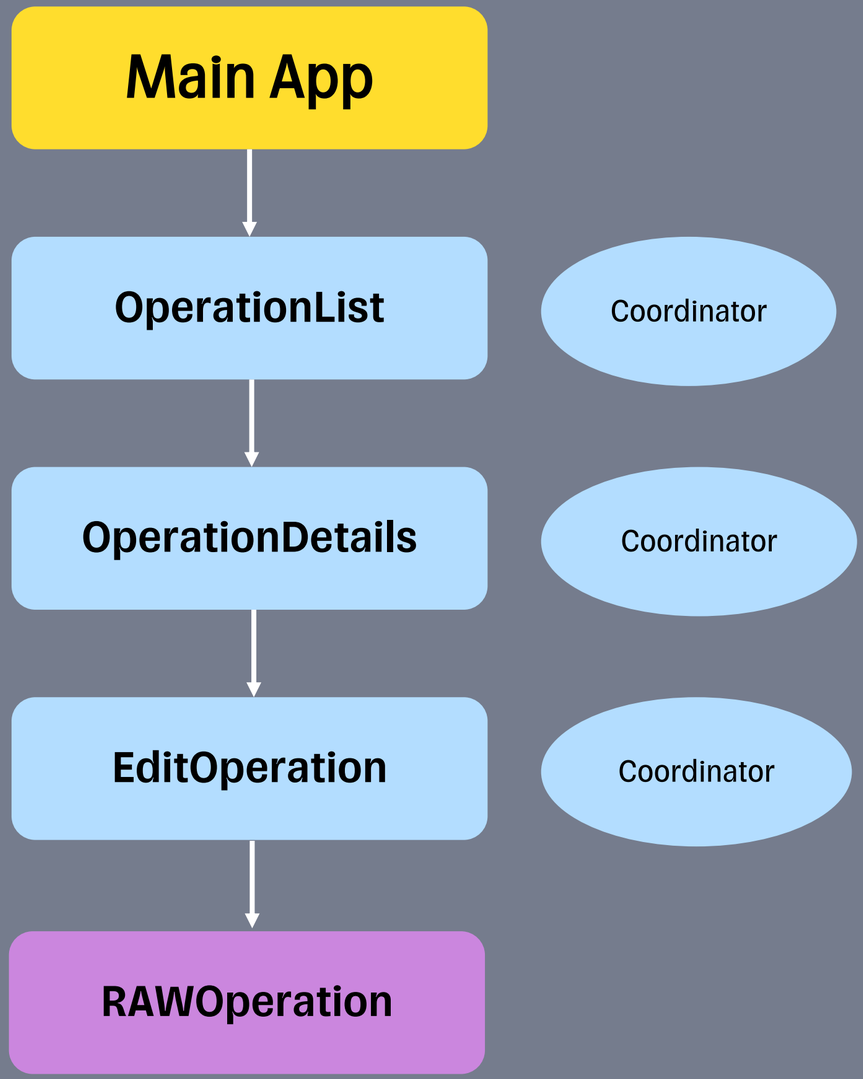
Попытка выделить роутинг в MVC/MVP/MVVM/VIPER

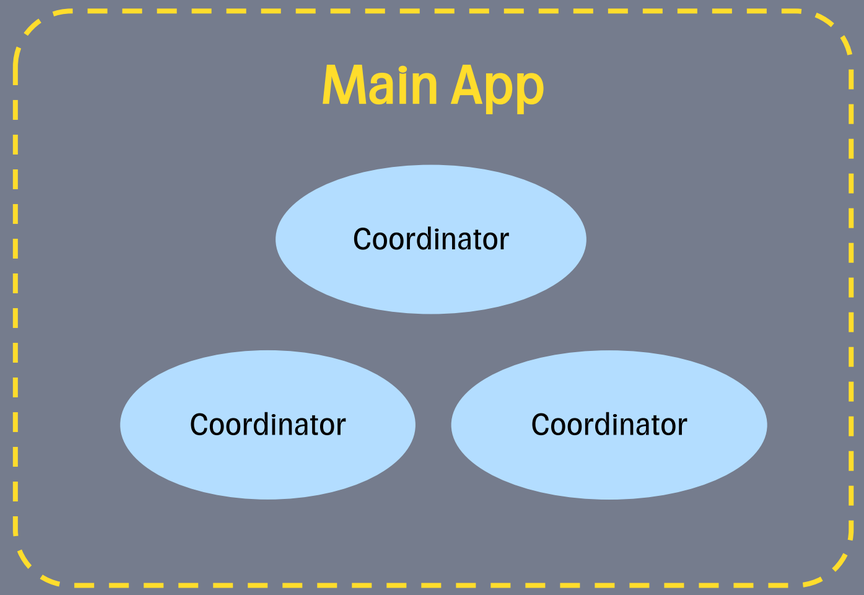
- Большая связность модулей
- Главный роутер в основном приложении контролирует слишком много
- Много копирования одного и того же кода

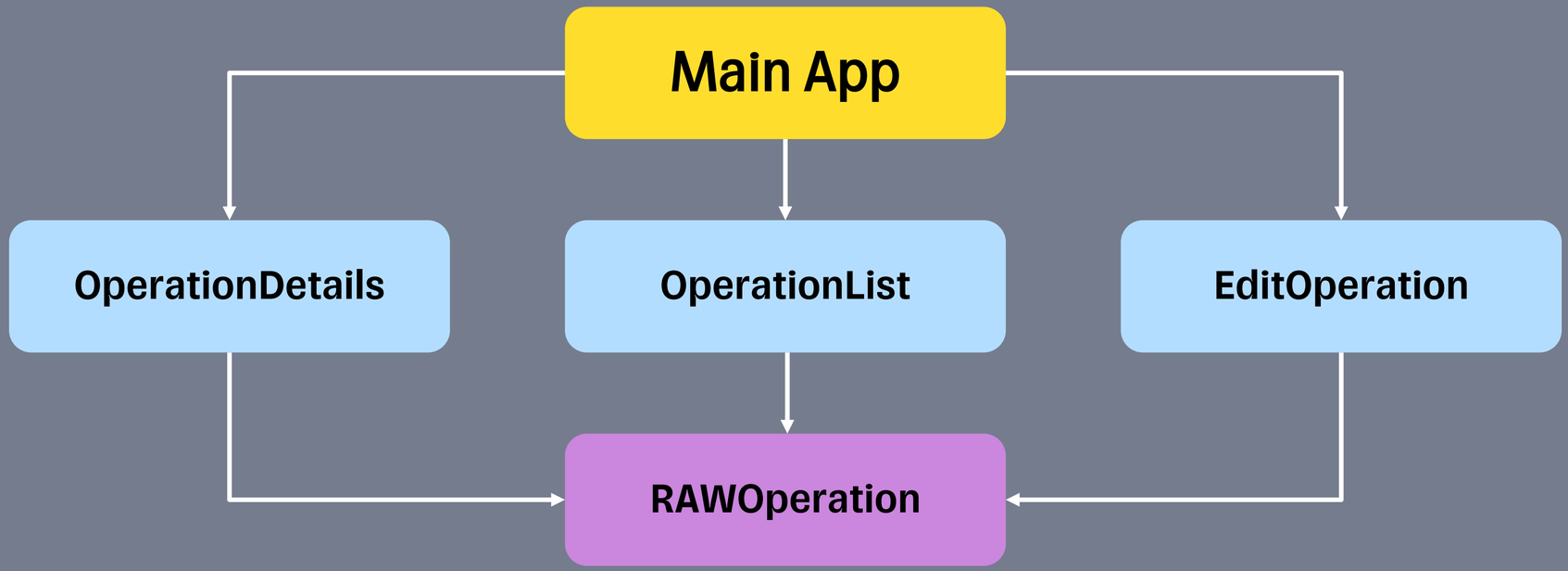
Эпоха FlowCoordinator



- Декомпозиция роутера на составляющие
- Появляется абстракция
- Появляется цепочка ответственностей
- Позволяет избавиться от явных связей


Main App
OperationList
Coordinator
OperationDetails
Coordinator
EditOperation
Coordinator
RAWOperation


Main App
Coordinator
Coordinator
Coordinator


Main App
OperationDetails
OperationList
EditOperation
RAWOperation

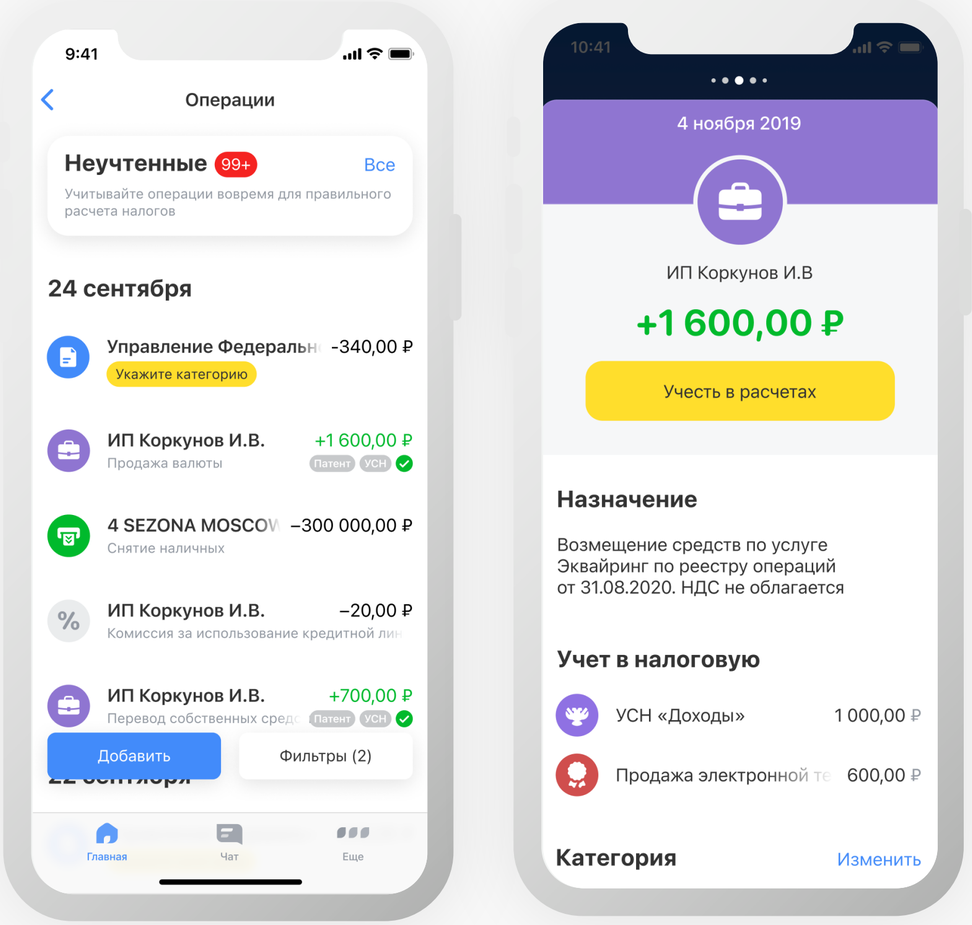
Все еще много проблем

- Нужно хранить
все переходы
что знать кто top
- Жизненный цикл завязан
на ViewController
- Костыли для нативных
переходов, backSwipe,
iPad

Все еще много проблем

- Достаточно объемные координаторы в основном проекте
- Реализация диплинков на координаторах на грани с магией
- Контролировать доступы на этапе координатора требует дополнительной абстракции

Задача навигации



01 Инкапсуляция логики внутри навигации модуля и вне без связности модулей

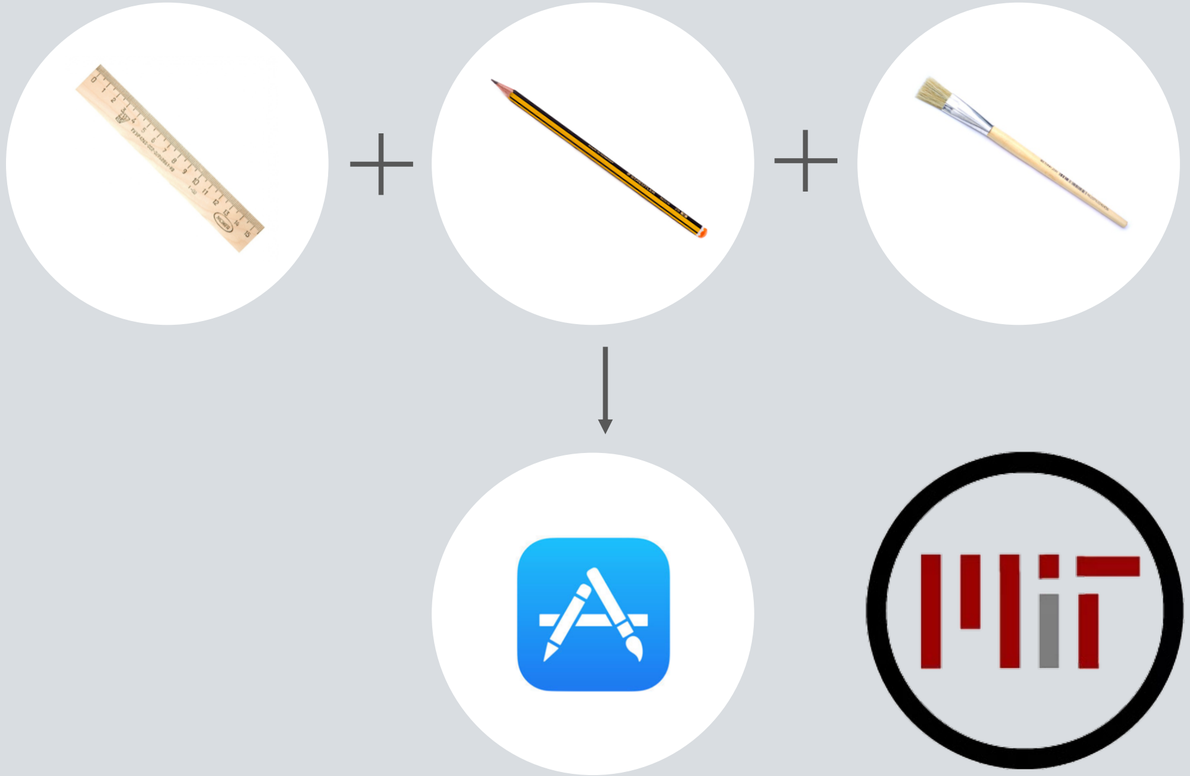
02 Удобная обработка диплинков

03 Ограничение доступа (авторизация, проверка ролей)

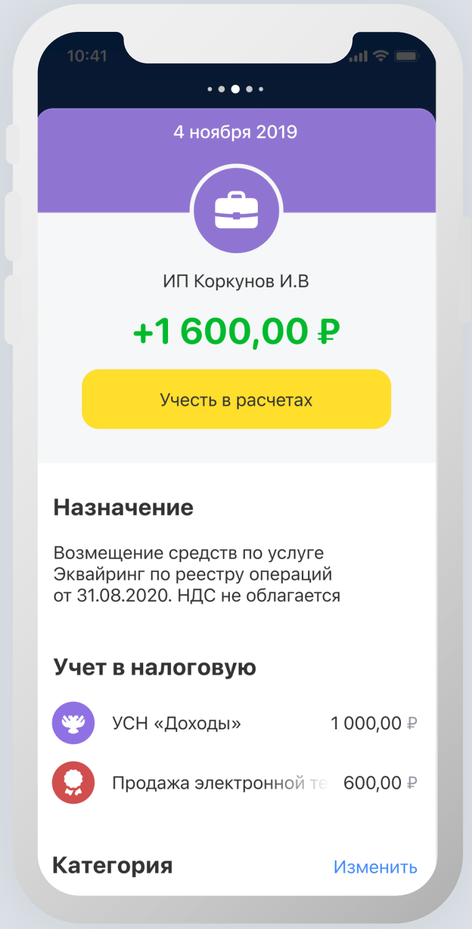
04 Настройка отображения модуля поверх другого модуля

05 Обработка нативных «особенностей» UIKit

RouteComposer



Создание экрана

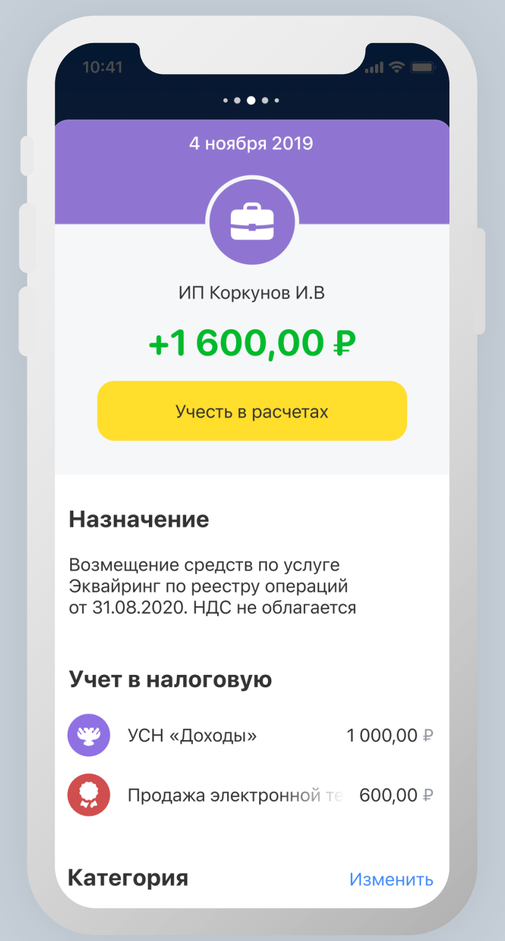


01 У всех экранов разные конструкторы

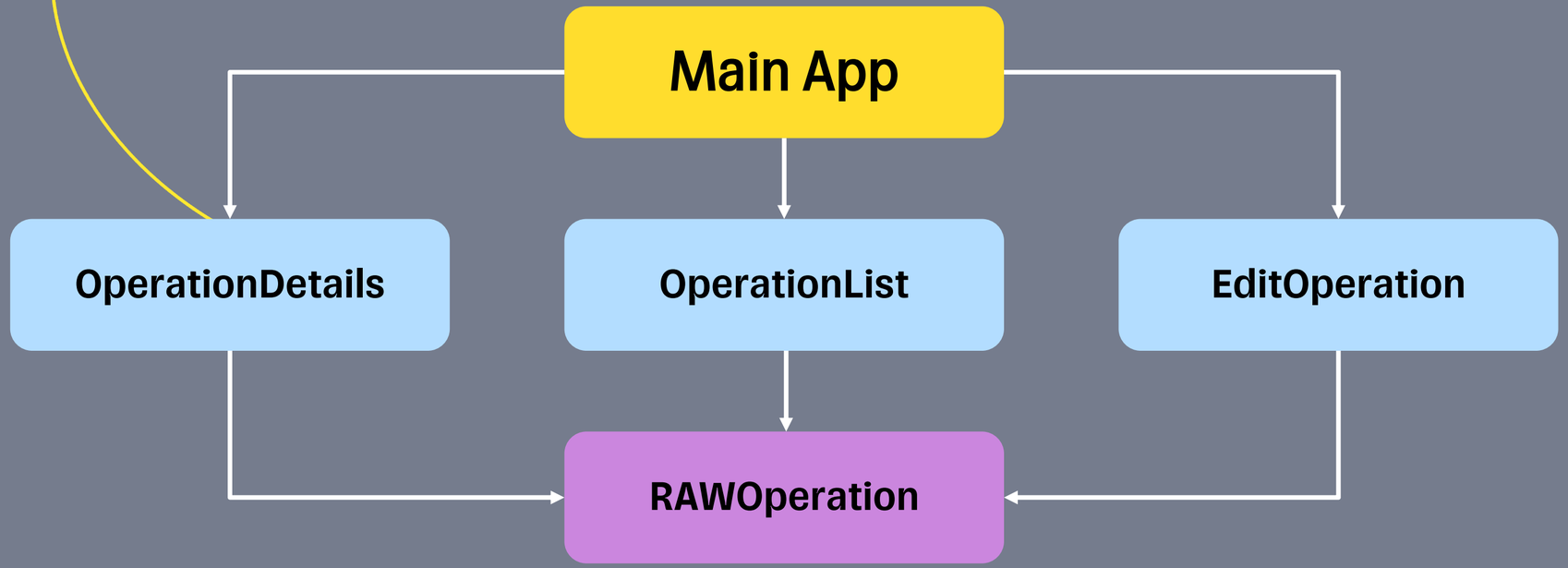
02 Они могут быть созданы из кода, storyboard, SwiftUI View

03 У экранов разные входные параметры

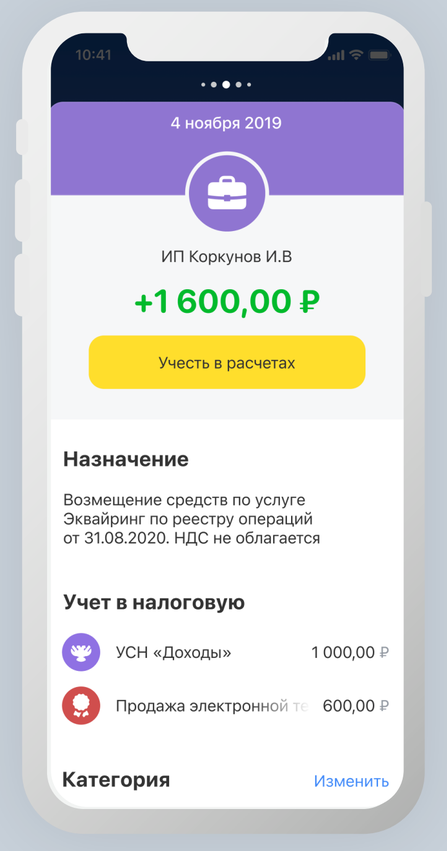
04 Создание нужно покрыть тестами




```
StepAssembly(
    finder: NilFinder(),
    factory: OperationDetailsFactory()
)
.using(GeneralAction.presentModally())
.from(GeneralStep.current())
.assemble()
```



Фабрика



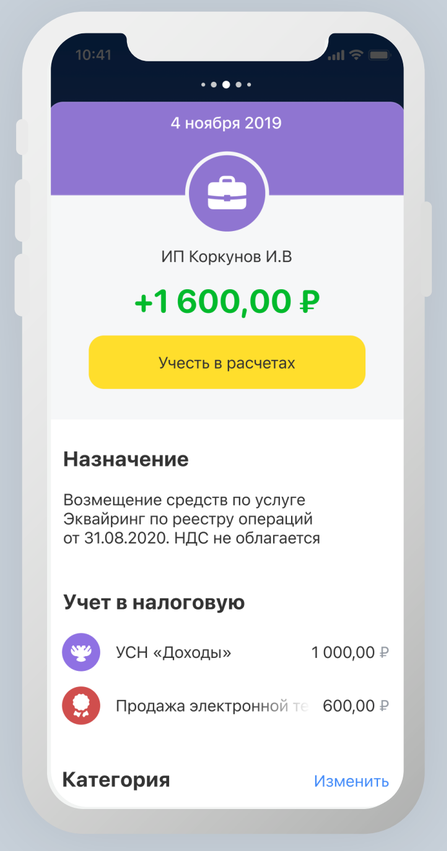
OperationDetails

```
StepAssembly(
    finder: NilFinder(),
    factory: OperationDetailsFactory()
)
.using(GeneralAction.presentModally())
.from(GeneralStep.current())
.assemble()
```

Route Composer

```
public protocol Factory {

    associatedtype ViewController
    associatedtype Context

    func build(with context: Context) throws → ViewController
}
```

Фабрика

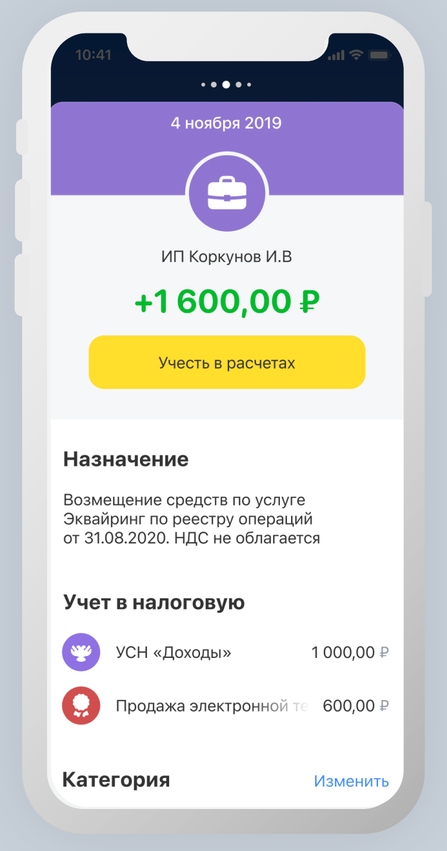


OperationDetails

```
StepAssembly(
    finder: NilFinder(),
    factory: OperationDetailsFactory()
)
.using(GeneralAction.presentModally())
.from(GeneralStep.current())
.assemble()
```

```
struct OperationDetailsFactory: Factory {

    struct Context {
        let operationId: Operation.OperationID
    }

    func build(with context: Context) throws → some UIViewController {
        OperationDetailsViewController(operationId: context.operationId)
    }
}
```

Фабрика

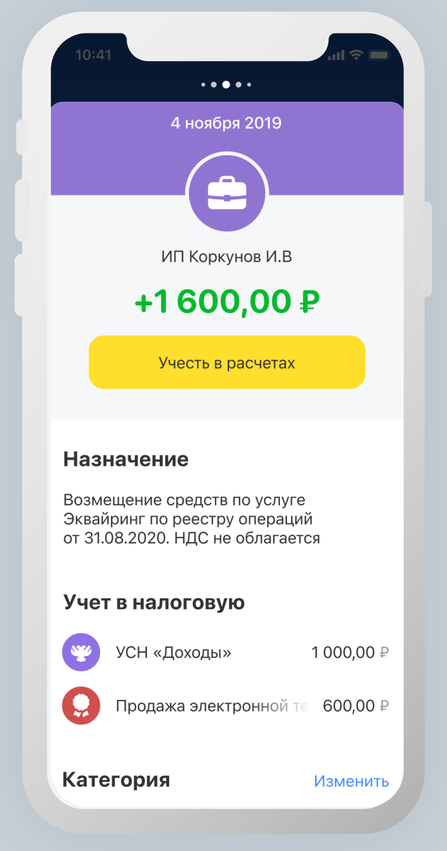


OperationDetails

```
StepAssembly(
    finder: NilFinder(),
    factory: OperationDetailsFactory()
)
.using(GeneralAction.presentModally())
.from(GeneralStep.current())
.assemble()
```

```
struct OperationDetailsFactory: Factory {

    struct Context {
        let operationId: Operation.OperationID
    }

    func build(with context: Context) throws → some UIViewController {
        OperationDetailsViewController(operationId: context.operationId)
    }
}
```

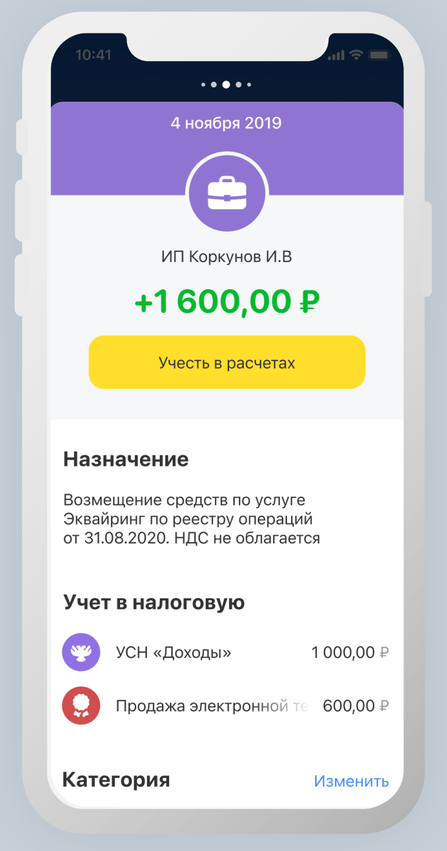
Файндер




```
typealias OperationId = String

var operaionDetailsScreen: DestinationStep<OperationDetailsViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: OperationDetailsFactory()
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```

```
public protocol Finder {

    associatedtype ViewController: UIViewController

    associatedtype Context

    func findViewController(with context: Context) throws -> ViewController?

}
```

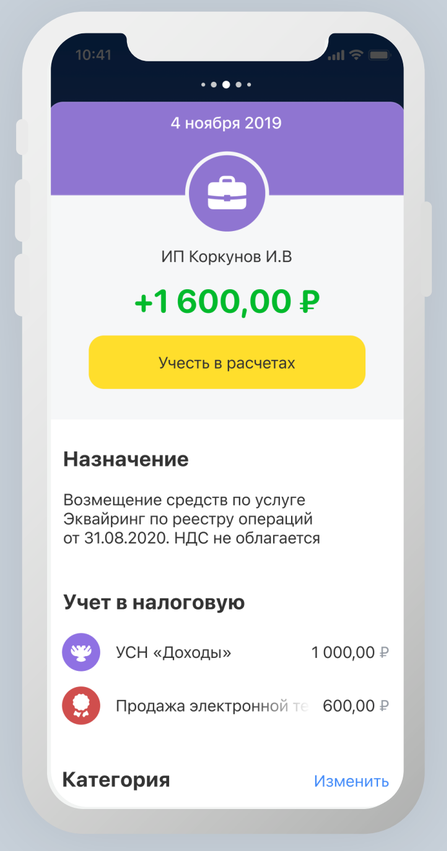
Файндер




```
var operaionDetailsScreen: DestinationStep<OperationDetailsViewController, OperationId> {
    StepAssembly(
        finder: ClassFinder<OperationDetailsViewController, OperationId>(),
        factory: OperationDetailsFactory()
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```

```
public protocol Finder {

    associatedtype ViewController: UIViewController

    associatedtype Context

    func findViewController(with context: Context) throws -> ViewController?

}
```

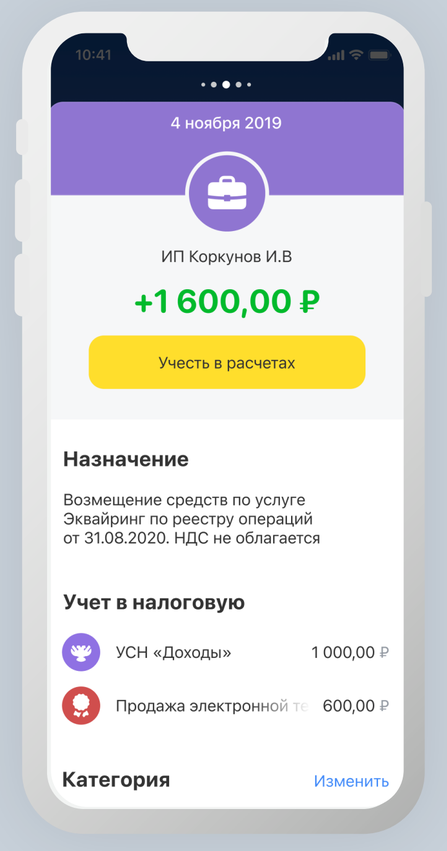
Файндер




```
var operaionDetailsScreen: DestinationStep<OperationDetailsViewController, OperationId> {
    StepAssembly(
        finder: OperationDetailsFinder(),
        factory: OperationDetailsFactory()
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```

```
class OperationDetailsFinder: StackIteratingFinder {

    typealias ViewController = OperationDetailsViewController

    typealias Context = OperationId

    func isTarget(_ viewController: OperationDetailsViewController,
                  with operationId: OperationId) -> Bool
    {
        viewController.operationId == operationId
    }
```

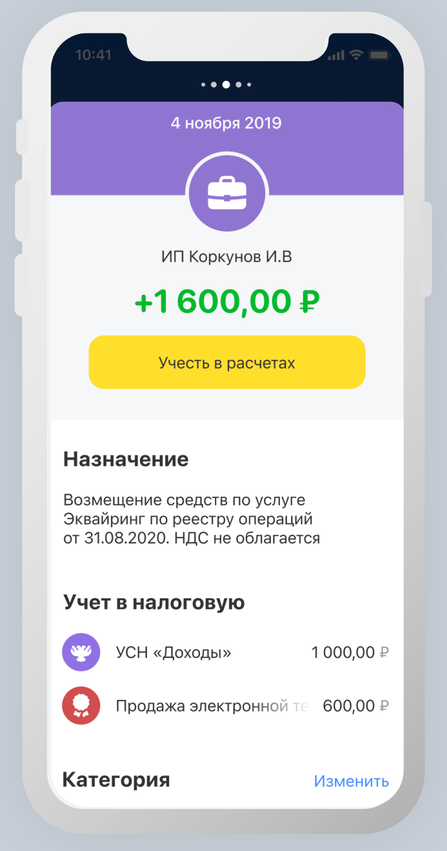
Файндер




```
var operaionDetailsScreen: DestinationStep<OperationDetailsViewController, OperationId> {
    StepAssembly(
        finder: OperationDetailsFinder(),
        factory: OperationDetailsFactory()
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```

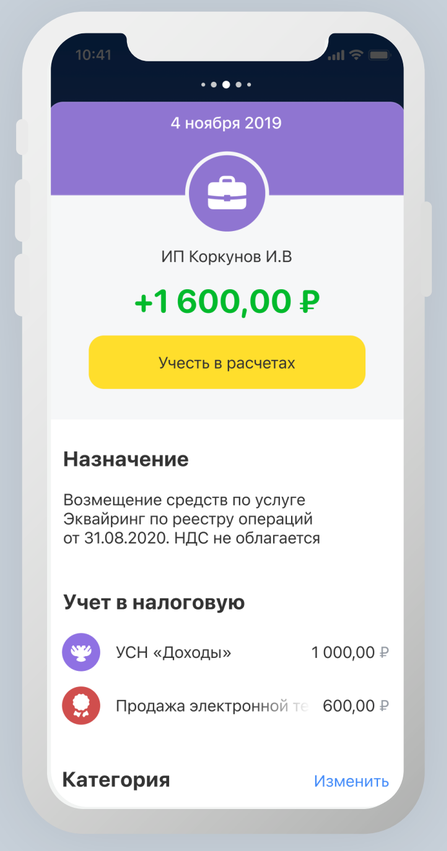
```
class OperationDetailsFinder: StackIteratingFinder {

    typealias ViewController = OperationDetailsViewController

    typealias Context = OperationId

    func isTarget(_ viewController: OperationDetailsViewController,
                  with operationId: OperationId) -> Bool
    {
        viewController.operationId = operationId
        return true
    }
```

Assembly



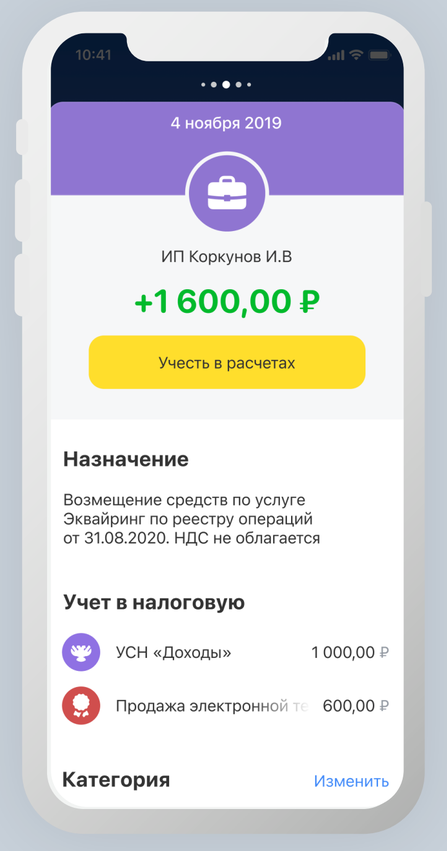

```
StepAssembly(
    finder: NilFinder(),
    factory: OperationDetailsFactory()
)
.using(GeneralAction.presentModally())
.from(GeneralStep.current())
.assemble()
```

Assembly



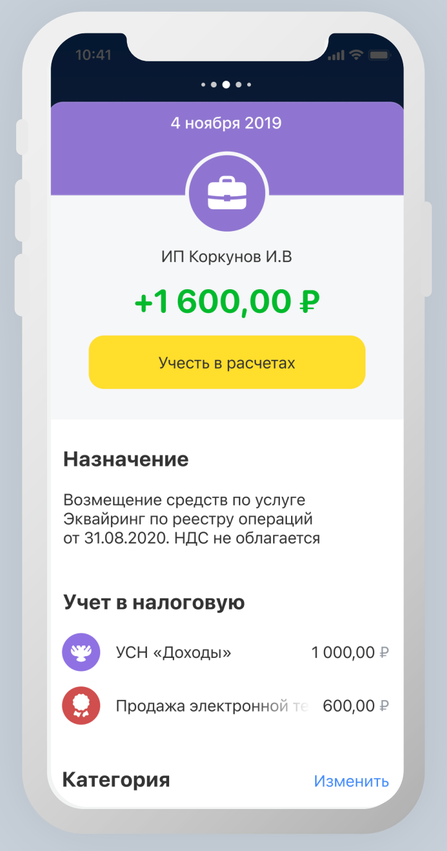

```
StepAssembly(
    finder: NilFinder(),
    factory: OperationDetailsFactory()
)
.using(GeneralAction.presentModally())
.from(GeneralStep.current())
.assemble()
```

Assembly




```
StepAssembly(
    finder: NilFinder(),
    factory: OperationDetailsFactory()
)
.using(GeneralAction.presentModally())
.from(GeneralStep.current())
.assemble()
```

Action



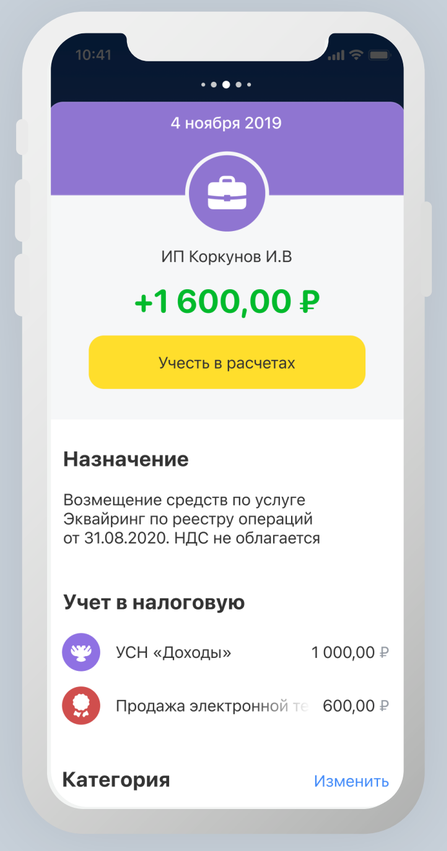
OperationDetails

```
StepAssembly(
    finder: NilFinder(),
    factory: OperationDetailsFactory()
)
.using(GeneralAction.presentModally())
.from(GeneralStep.current())
.assemble()
```

Route Composer

```
public struct PresentModallyAction: Action {

    public func perform(with viewController: UIViewController,
                        on existingController: UIViewController,
                        animated: Bool,
                        completion: @escaping (_: RoutingResult) -> Void) {
        existingController.present(viewController, animated: animated, completion: {
            completion(.success)
        })
    }

}
```

Step



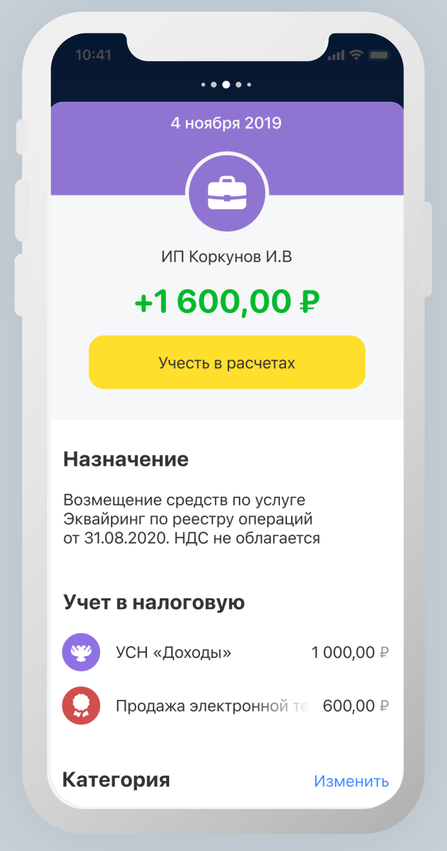
OperationDetails

```
StepAssembly(
    finder: NilFinder(),
    factory: OperationDetailsFactory()
)
.using(GeneralAction.presentModally())
.from(GeneralStep.current())
.assemble()
```

Route Composer

```
struct CurrentViewControllerStep: RoutingStep, PerformableStep {

    func perform(with context: AnyContext) throws -> PerformableStepResult {
        guard let viewController = windowProvider.window?.topmostViewController else {
            throw RoutingError.compositionFailed(.init("Topmost view controller was not found."))
        }
        return .success(viewController)
    }
```

DestinationStep



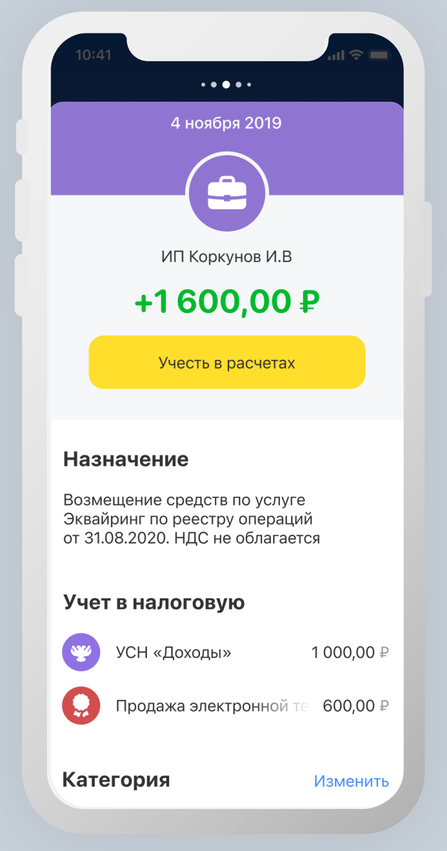
OperationDetails

```
static var operationDetails: DestinationStep
    StepAssembly(
        finder: NilFinder(),
        factory: OperationDetailsFactory()
    )
    .using(GeneralAction.presentModally())
    .from(GeneralStep.current())
    .assemble()
}
```

Route Composer

```
public func assemble() -> DestinationStep<ViewController, Context> {
    DestinationStep(chain(previousSteps))
}
```

DestinationStep

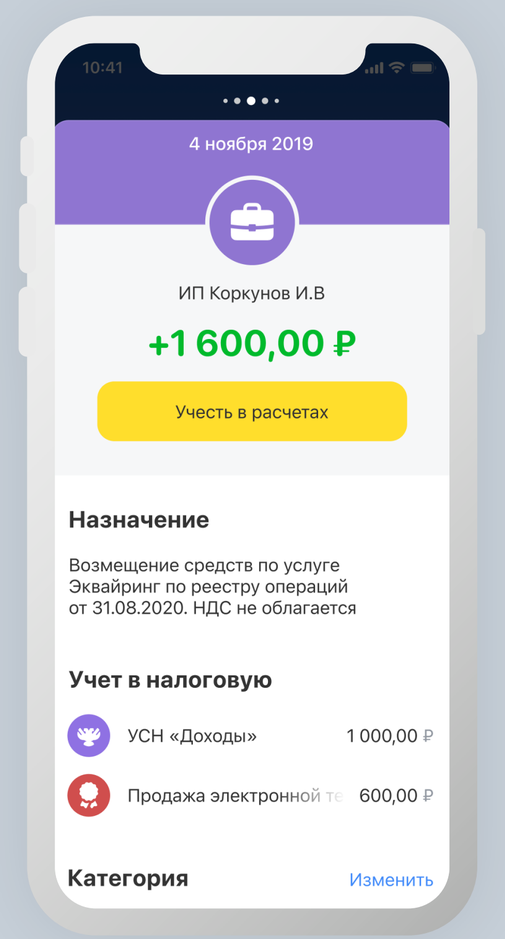


OperationDetails

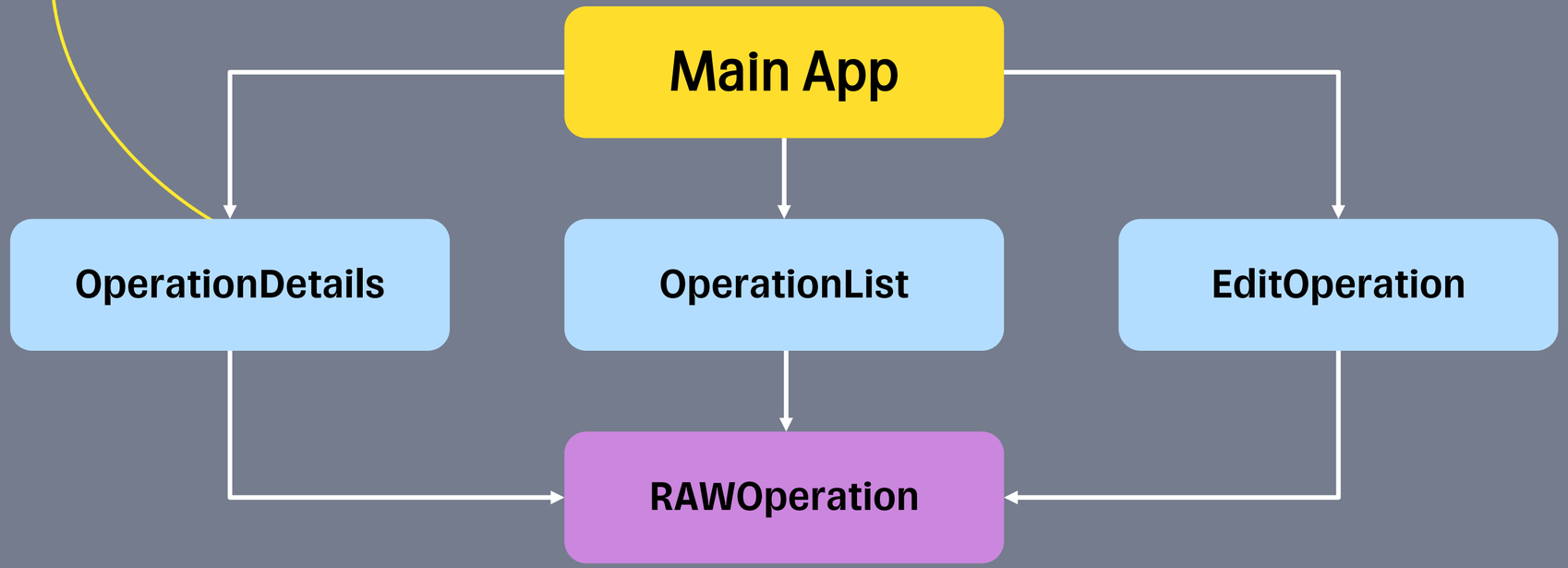
```
static var operationDetails: DestinationStep
    StepAssembly(
        finder: NilFinder(),
        factory: OperationDetailsFactory()
    )
    .using(GeneralAction.presentModally())
    .from(GeneralStep.current())
    .assemble()
}
```

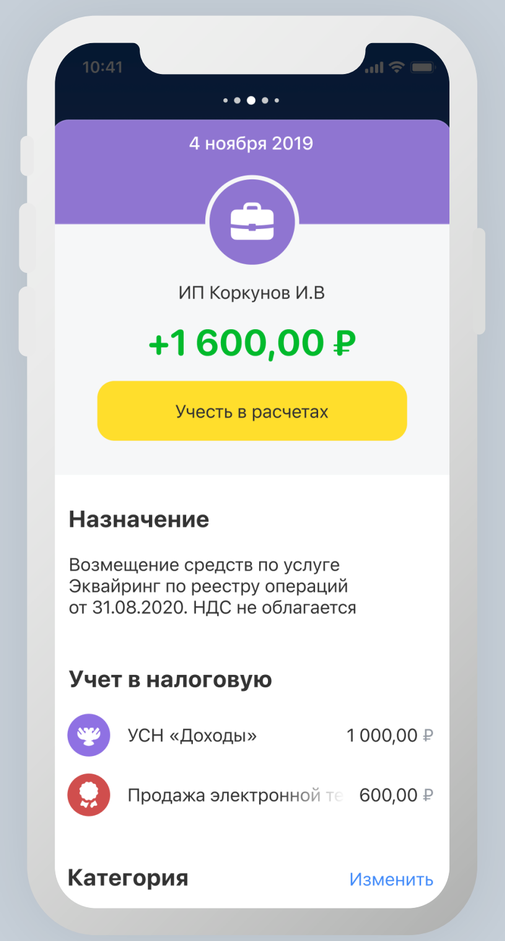
Route Composer

```
public func assemble() -> DestinationStep<ViewController, Context> {
    DestinationStep(chain(previousSteps))
}
```

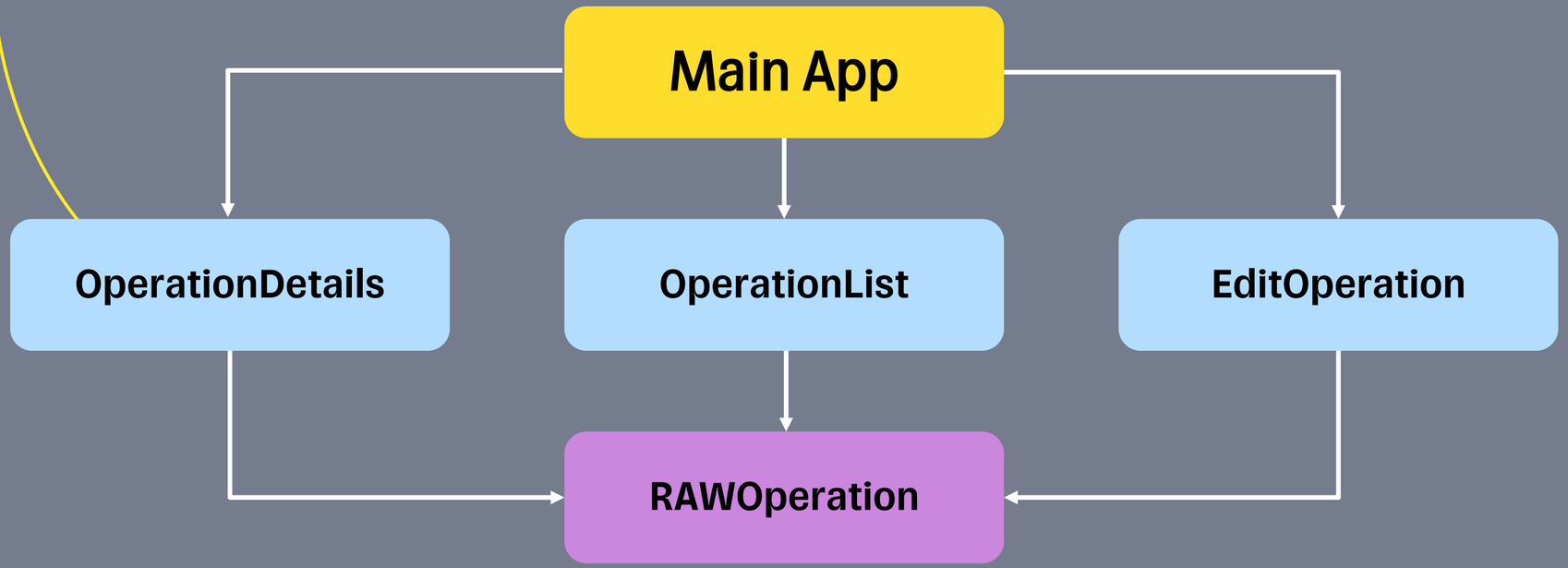


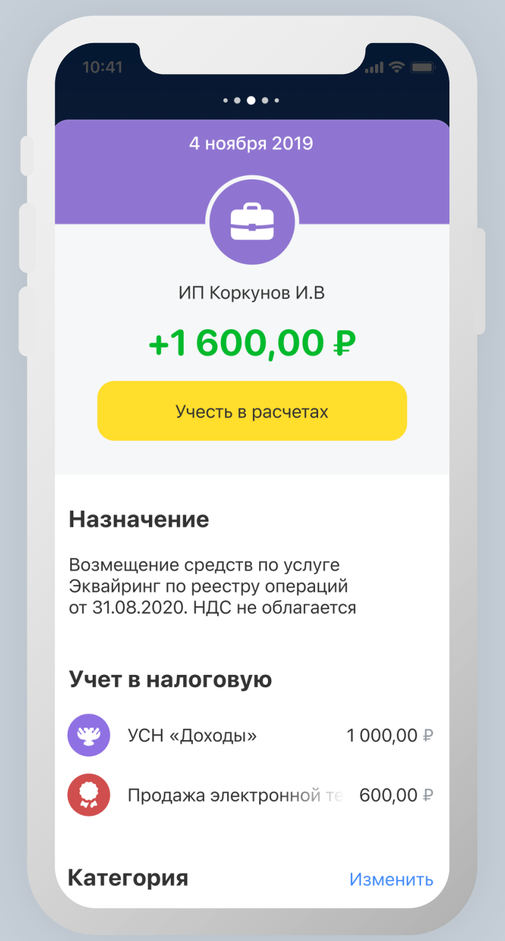

```
static var operationDetails: DestinationStep
    StepAssembly(
        finder: NilFinder(),
        factory: OperationDetailsFactory()
    )
    .using(GeneralAction.presentModally())
    .from(GeneralStep.current())
    .assemble()
}
```



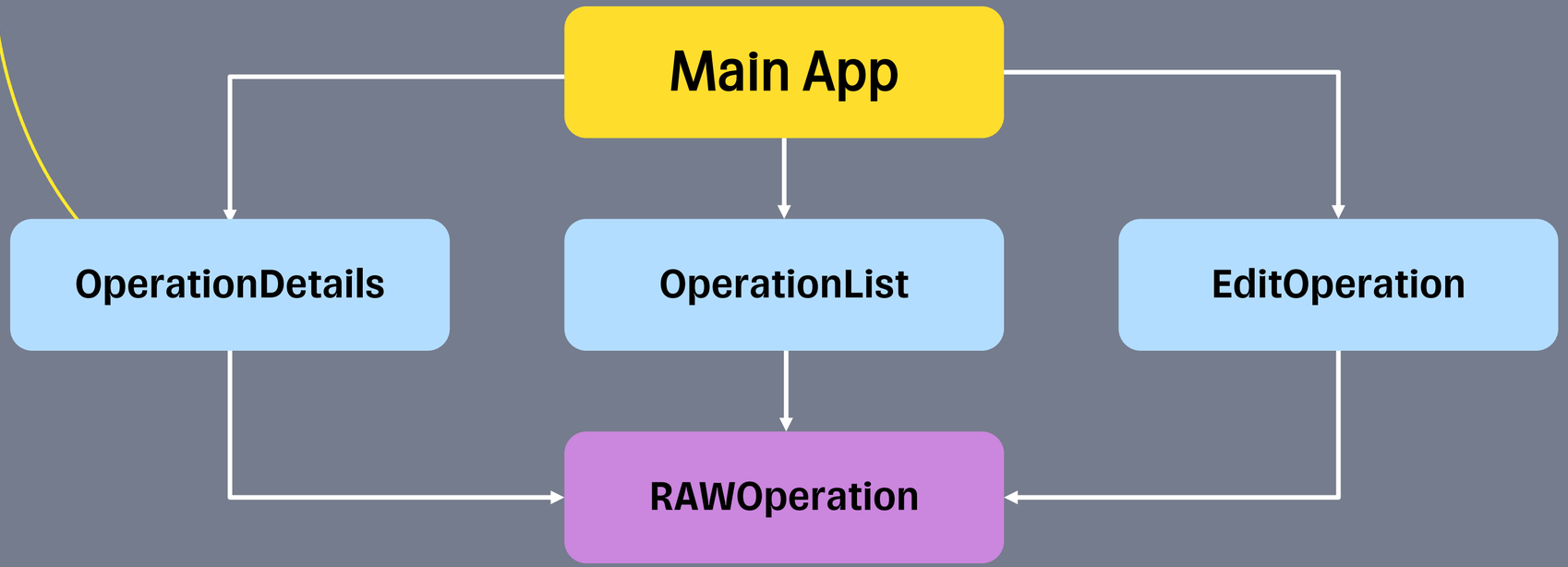


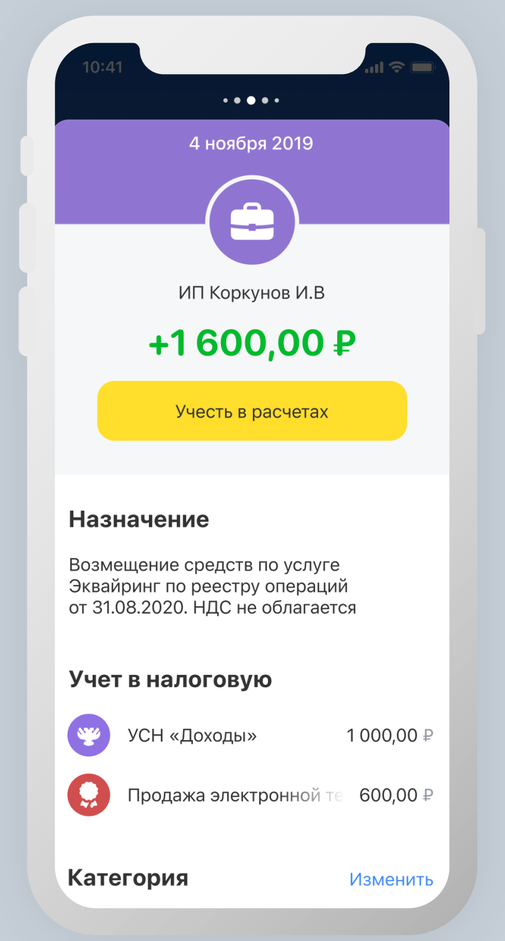

```
class OperationDetailsModuleRoutes: DestinationsProvider {

    func operationDetails(operationId: String) -> AnyDestination {
        return Destination(to: operationDetailsScreen, with: operationId)
            .unwrapped()
    }
```



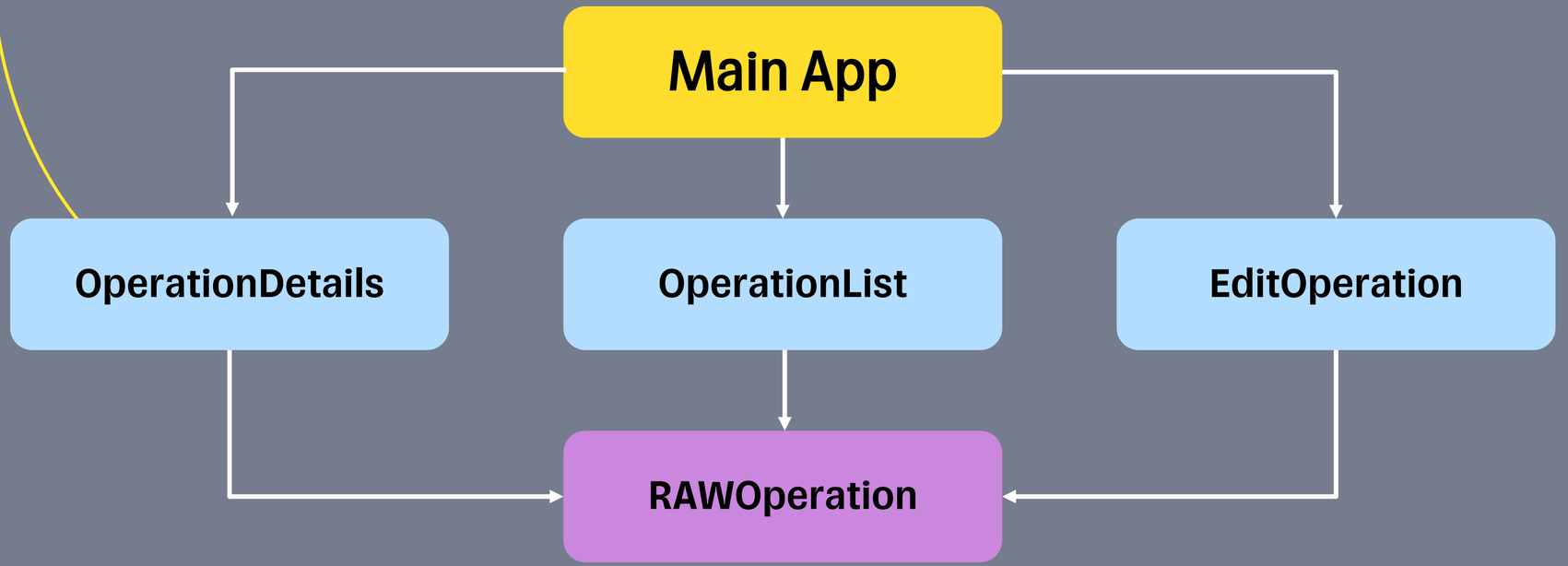


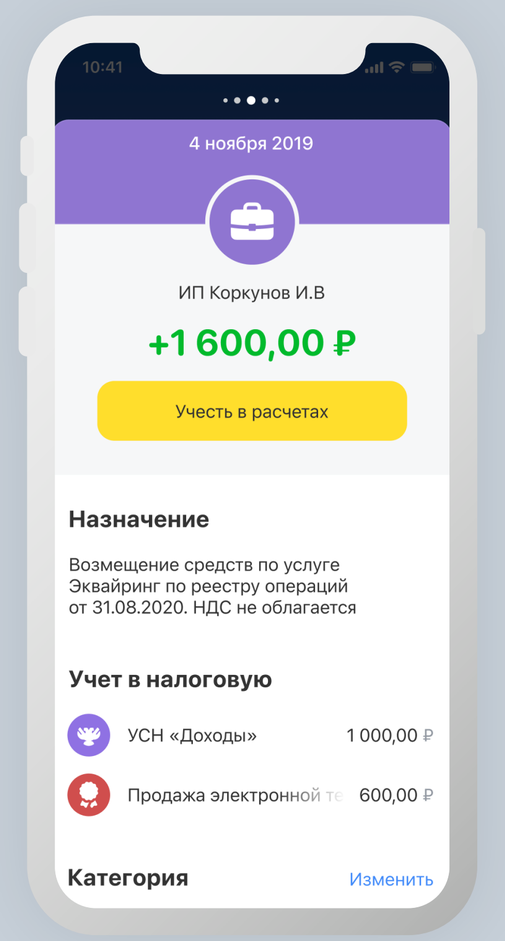

```
class OperationDetailsModuleRoutes: DestinationsProvider {

    func operationDetails(operationId: String) -> AnyDestination {
        return Destination(to: operationDetailsScreen, with: operationId)
            .unwrapped()
    }
```



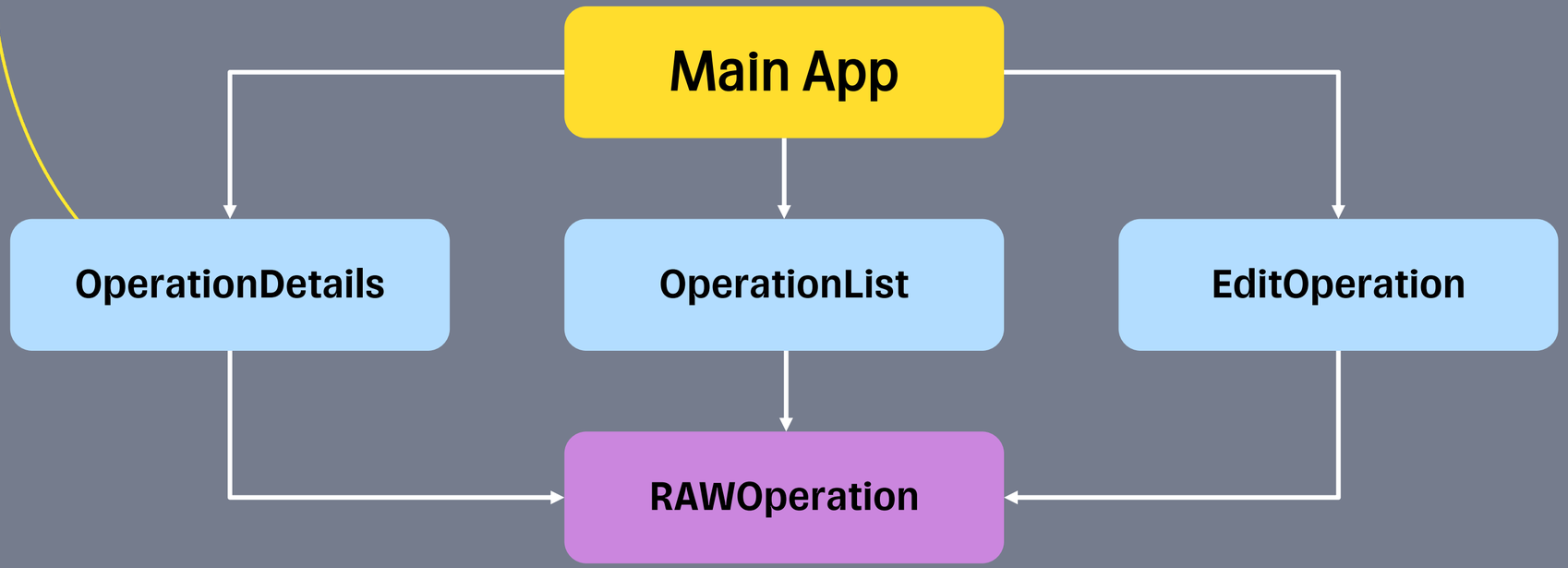


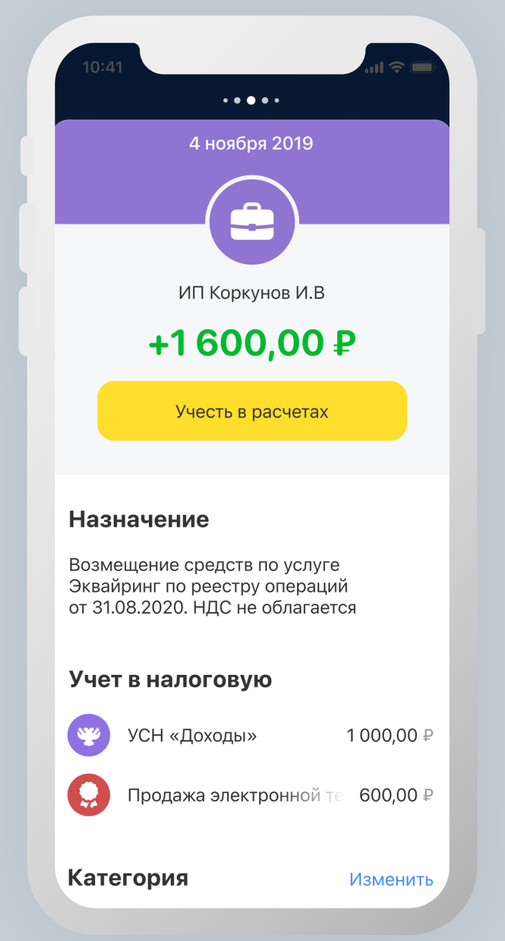

```
class OperationDetailsModuleRoutes: DestinationsProvider {

    func operationDetails(operationId: String) -> AnyDestination {
        return Destination(to: operationDetailsScreen, with: operationId)
            .unwrapped()
    }
```



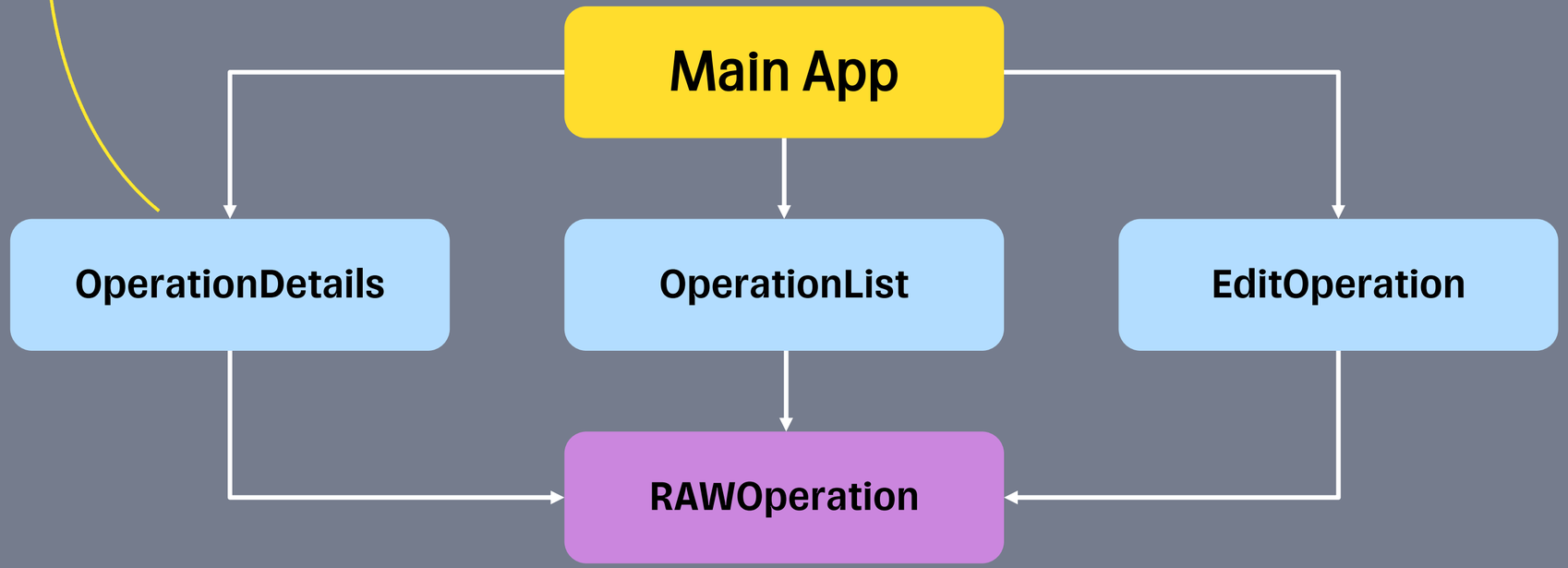


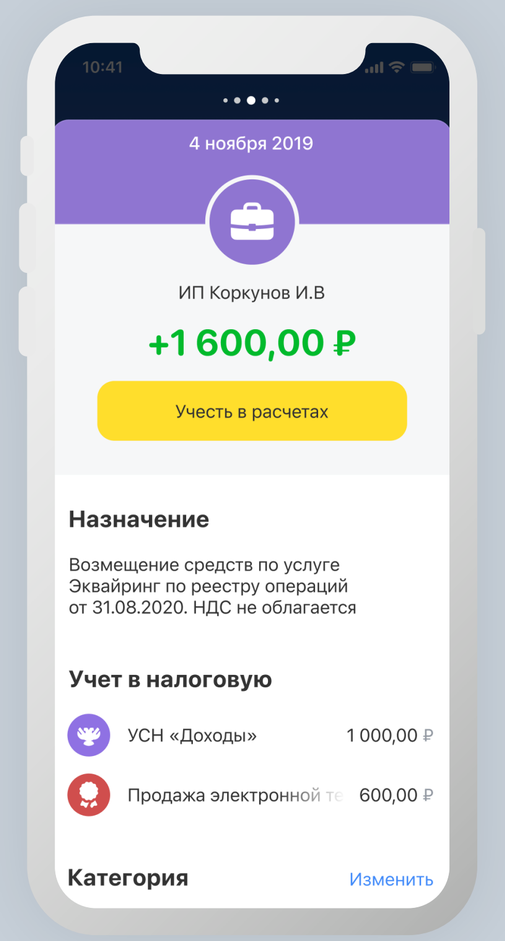

```
class OperationDetailsModuleRoutes: DestinationsProvider {

    func operationDetails(operationId: String) -> AnyDestination {
        return Destination(to: operationDetailsScreen, with: operationId)
            .unwrapped()
    }
```



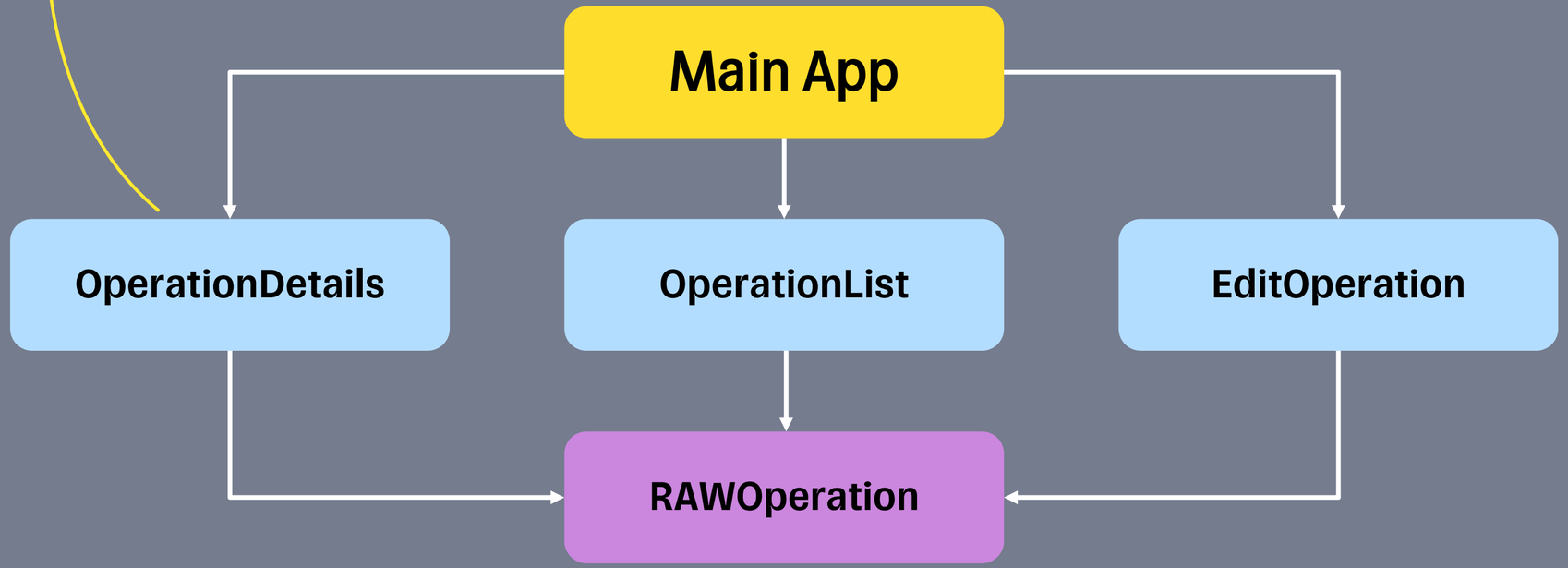


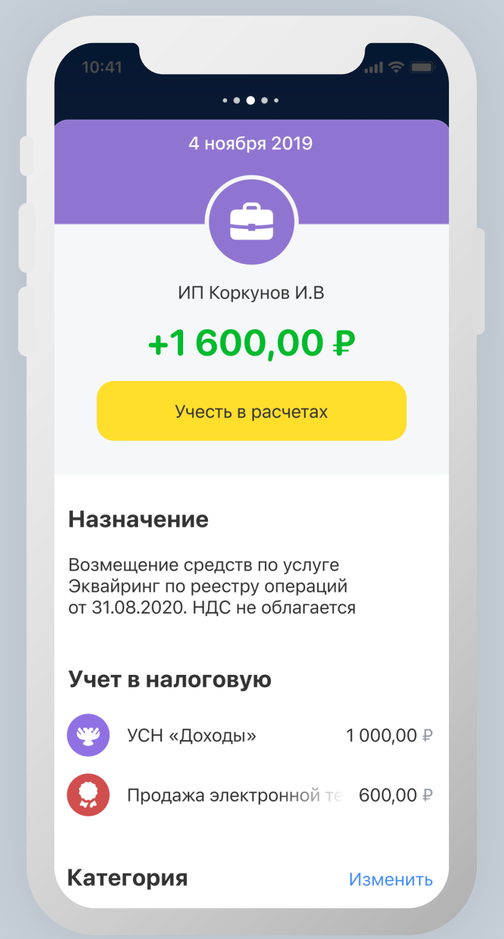

```
class OperationDetailsModuleRoutes: DestinationsProvider {

    func operationDetails(operationId: String) -> AnyDestination {
        return Destination(to: operationDetailsScreen, with: operationId)
            .unwrapped()
    }

    func destination(from url: URL) -> AnyDestination? {
        guard let operationId = operationId(url: url) else {
            return nil
        }
        return operationDetails(operationId: operationId)
    }
```



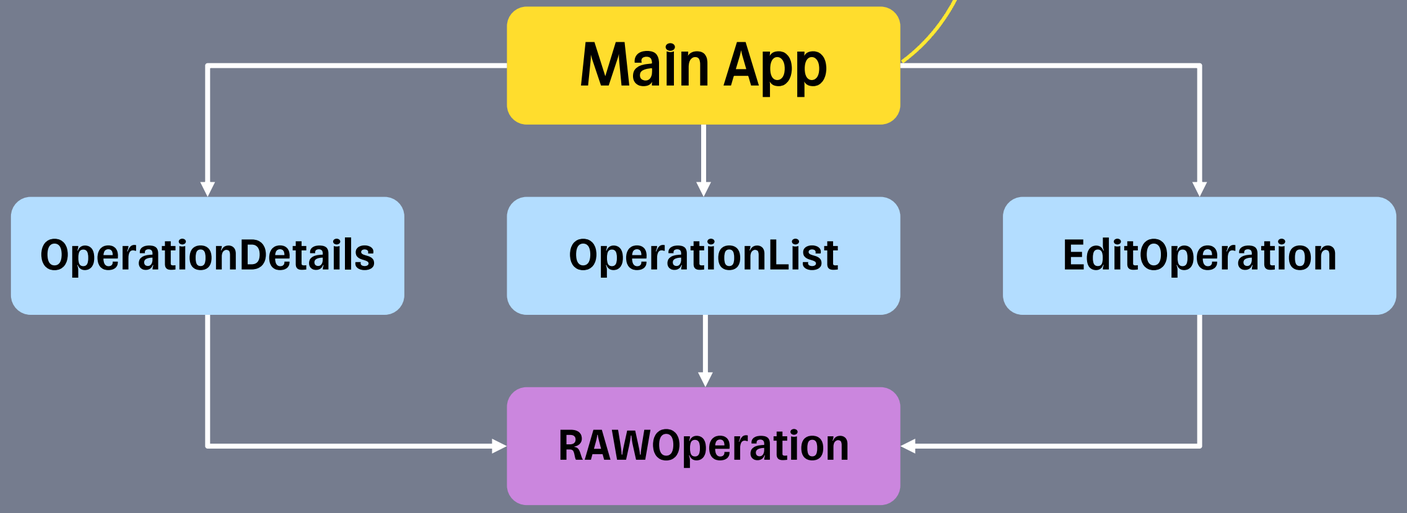


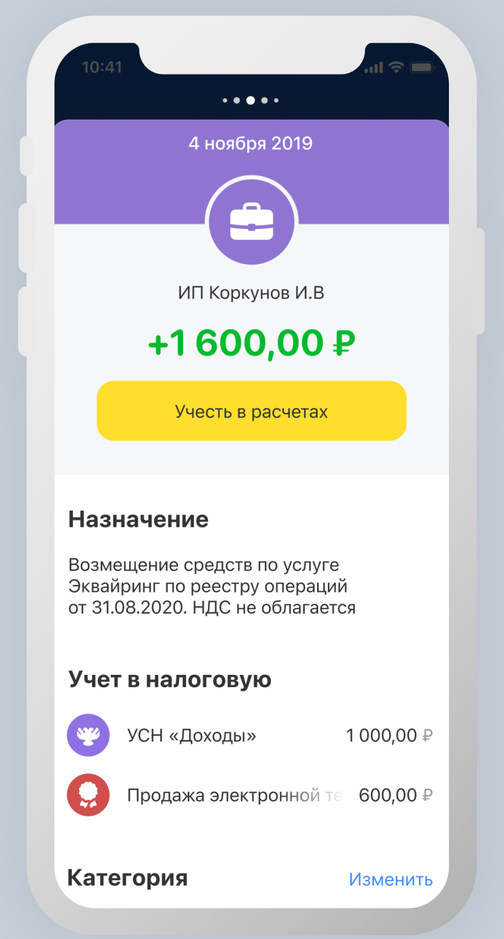

```
class OperationDetailsModuleRoutes: DestinationsProvider {

    func operationDetails(operationId: String) -> AnyDestination {
        return Destination(to: operationDetailsScreen, with: operationId)
            .unwrapped()
    }

    func destination(from url: URL) -> AnyDestination? {
        guard let operationId = operationId(url: url) else {
            return nil
        }
        return operationDetails(operationId: operationId)
    }
```



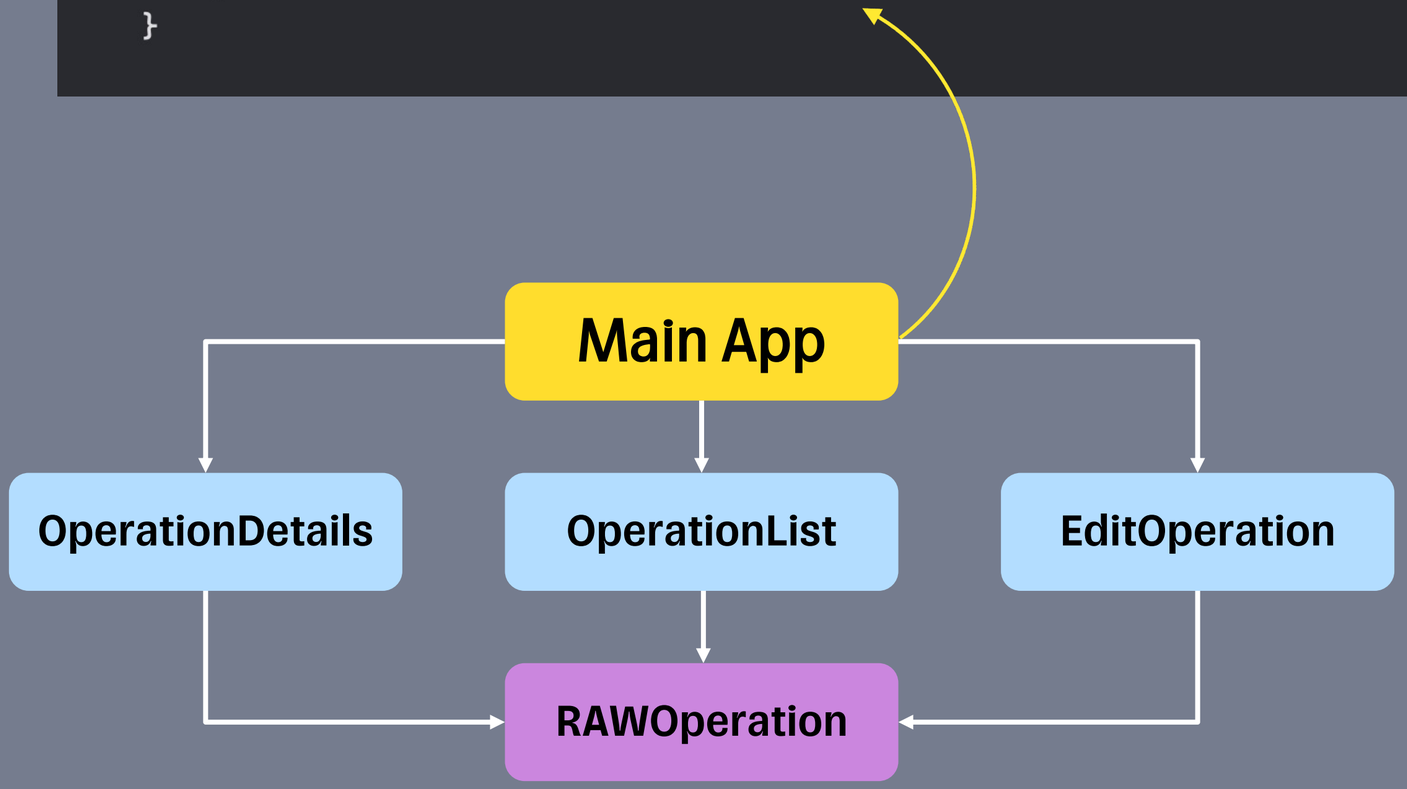


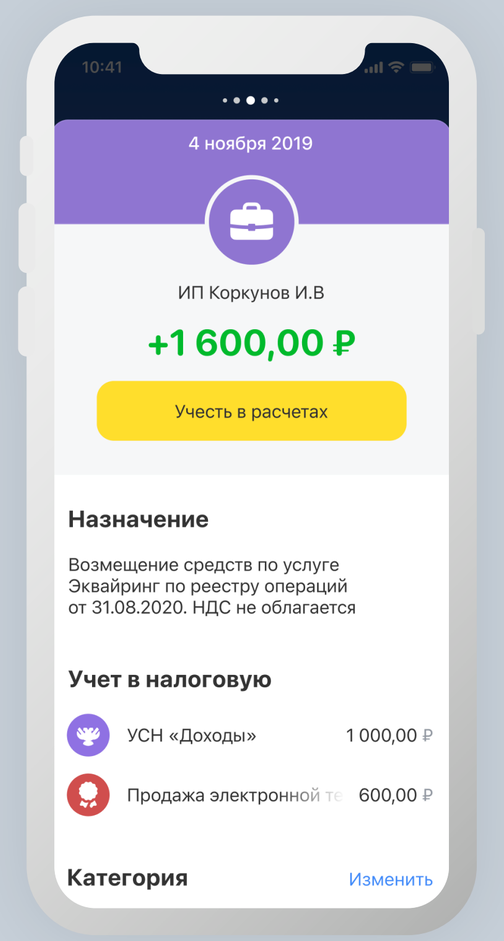

```
struct ExampleUniversalLinksManager {

    private let operationDetailsRoutes = OperationDetailsModuleRoutes()

    func openOperationDetails(operationId: OperationId) throws {
        try router.navigate(to: operationDetailsRoutes
            .operationDetails(operationId: operationId)
        )
    }
```



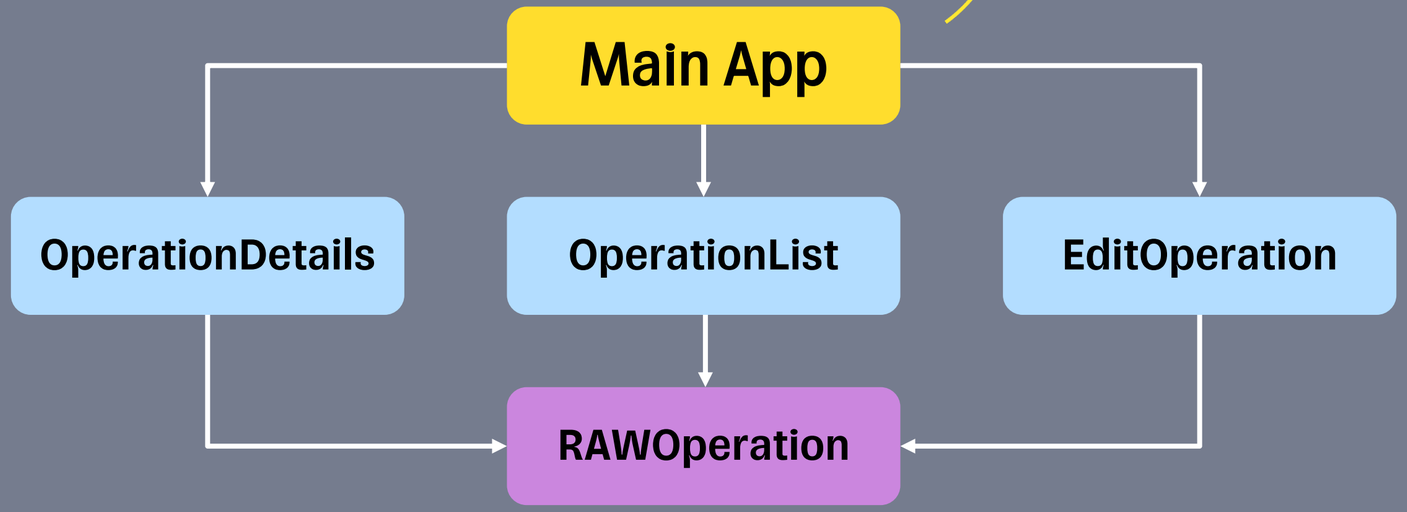


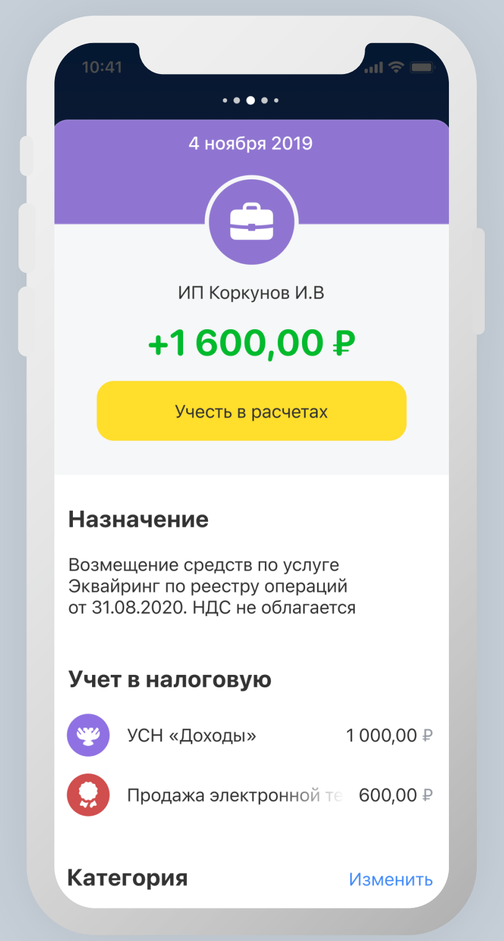

```
struct ExampleUniversalLinksManager {

    private let operationDetailsRoutes = OperationDetailsModuleRoutes()

    func openOperationDetails(operationId: OperationId) throws {
        try router.navigate(to: operationDetailsRoutes
            .operationDetails(operationId: operationId)
        )
    }
```



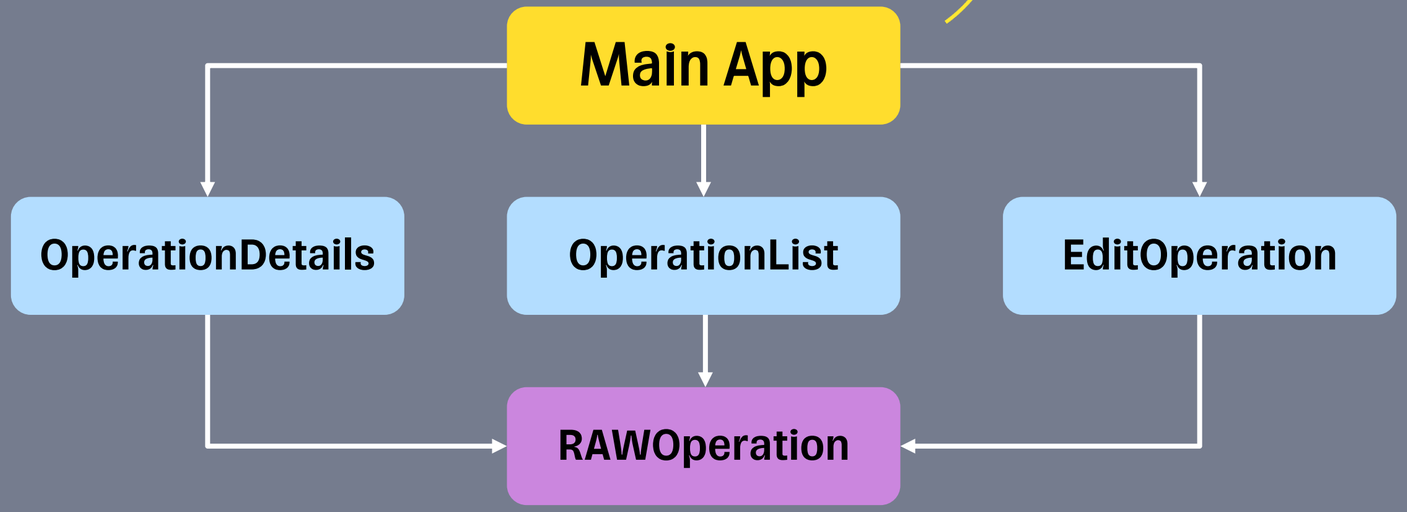


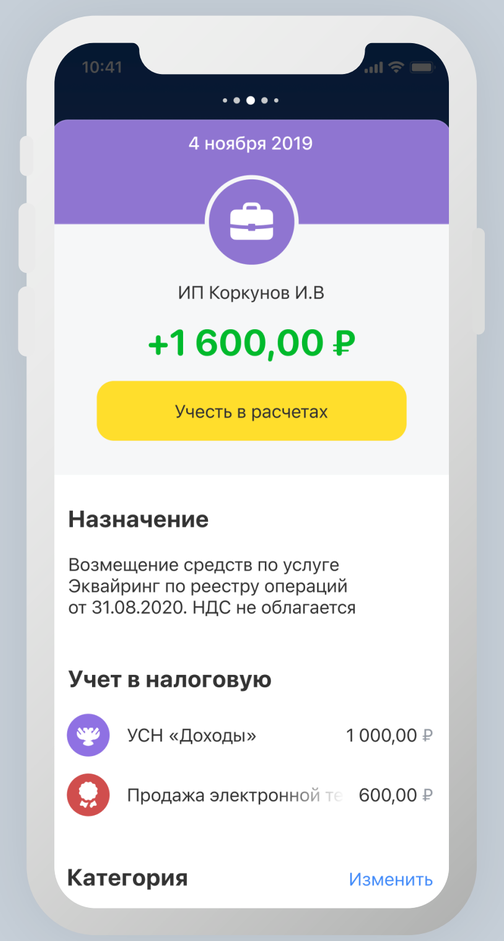

```swift
struct ExampleUniversalLinksManager {

    private let operationDetailsRoutes = OperationDetailsModuleRoutes()
    private let editOperation = EditOperationDetailsModuleRoutes()

    private var destinations: [DestinationsProvider] {
        [operationDetailsRoutes, editOperation]
    }

    func destination(for url: URL) -> AnyDestination? {
        guard let translator = destinations.first(where: { $0.destination(from: url) != nil }) else {
            return nil
        }

        return translator.destination(from: url)
    }
```



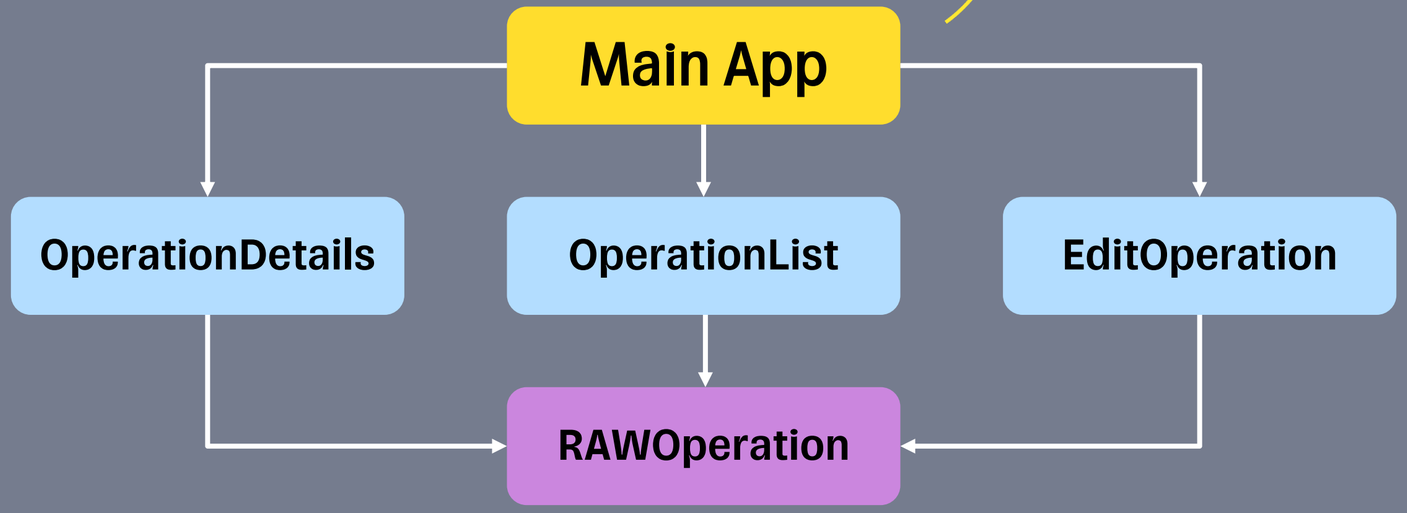



```swift
struct ExampleUniversalLinksManager {

    private let operationDetailsRoutes = OperationDetailsModuleRoutes()
    private let editOperation = EditOperationDetailsModuleRoutes()

    private var destinations: [DestinationsProvider] {
        [operationDetailsRoutes, editOperation]
    }

    func destination(for url: URL) -> AnyDestination? {
        guard let translator = destinations.first(where: { $0.destination(from: url) != nil }) else {
            return nil
        }

        return translator.destination(from: url)
    }
```






```swift
struct ExampleUniversalLinksManager {

    private let operationDetailsRoutes = OperationDetailsModuleRoutes()
    private let editOperation = EditOperationDetailsModuleRoutes()

    private var destinations: [DestinationsProvider] {
        [operationDetailsRoutes, editOperation]
    }

    func destination(for url: URL) -> AnyDestination? {
        guard let translator = destinations.first(where: { $0.destination(from: url) != nil }) else {
            return nil
        }

        return translator.destination(from: url)
    }
```



Задача навигации

- Инкапсуляция логики навигации внутри модуля и вне без связности модулей
- Удобная обработка диплинков
- Ограничение доступа (авторизация, проверка ролей)
- Настройка отображения модуля поверх другого модуля

Задача навигации

- **Инкапсуляция логики навигации внутри модуля и вне без связности модулей**
- Удобная обработка диплинков
- Ограничение доступа (авторизация, проверка ролей)
- Настройка отображения модуля поверх другого модуля

Задача навигации

- **Инкапсуляция логики навигации внутри модуля и вне без связности модулей**
- **Удобная обработка диплинков**
- Ограничение доступа (авторизация, проверка ролей)
- Настройка отображения модуля поверх другого модуля

Interceptor

```
var operaionDetailsScreen: DestinationStep<OperationDetailsViewController, OperationId> {
    StepAssembly(
        finder: OperationDetailsFinder(),
        factory: OperationDetailsFactory()
    )
    .adding(LoginInterceptor())
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```

```
public protocol RoutingInterceptor {

    associatedtype Context
    func perform(with context: Context, completion: @escaping (_: RoutingResult) -> Void)
}
```

Interceptor

```
var operaionDetailsScreen: DestinationStep<OperationDetailsViewController, OperationId> {
    StepAssembly(
        finder: OperationDetailsFinder(),
        factory: OperationDetailsFactory()
    )
    .adding(LoginInterceptor())
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```

```
public protocol RoutingInterceptor {

    associatedtype Context
    func perform(with context: Context, completion: @escaping (_: RoutingResult) -> Void)
}
```

ContextTask

```
var editOperaionCategoryScreen: DestinationStep<EditOperationViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: EditOperationFactory()
    )
    .adding(AnalyticsContextTask<EditOperationViewController, OperationId>())
    .adding(AnalyticsPostRoutingTask<EditOperationViewController, OperationId>())
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```

```
struct AnalyticsContextTask<VC: UIViewController, C>: ContextTask {
    func perform(on viewController: VC, with context: C) throws {
        print("ContextTask \(String(describing: viewController))")
    }
}
```

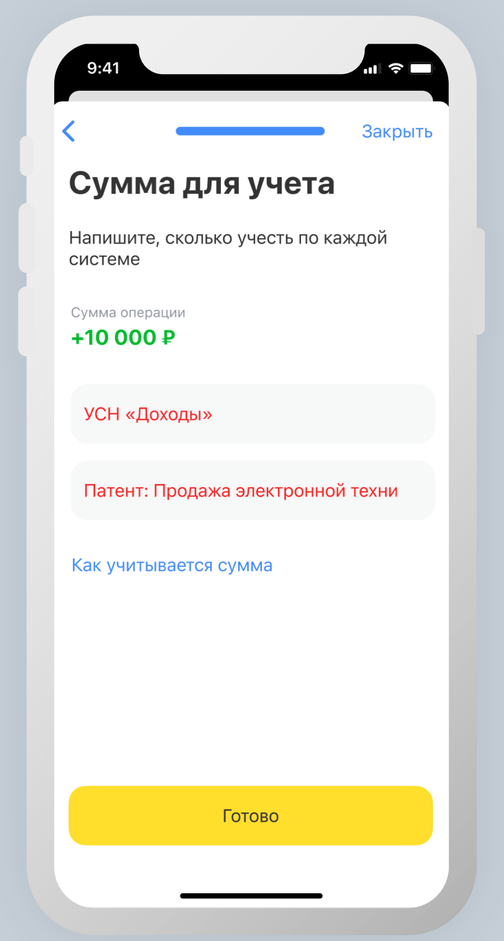
ContextTask

```
var editOperaionCategoryScreen: DestinationStep<EditOperationViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: EditOperationFactory()
    )
    .adding(AnalyticsContextTask<EditOperationViewController, OperationId>())
    .adding(AnalyticsPostRoutingTask<EditOperationViewController, OperationId>())
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```

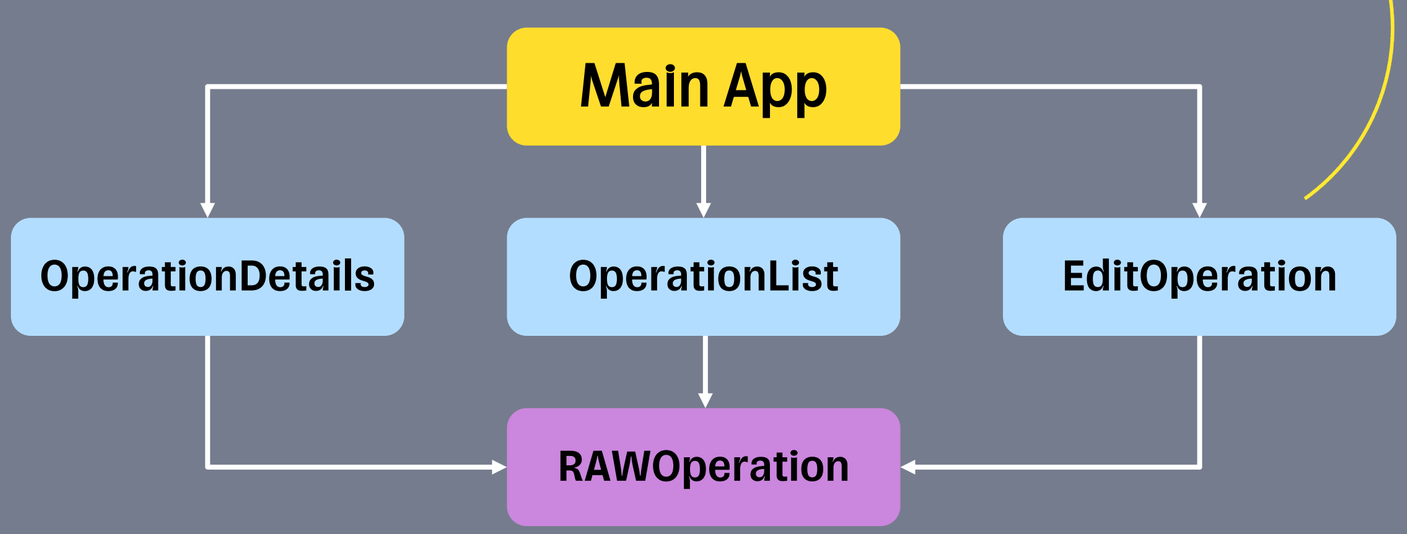
```
struct AnalyticsContextTask<VC: UIViewController, C>: ContextTask {
    func perform(on viewController: VC, with context: C) throws {
        print("ContextTask \(String(describing: viewController))")
    }
}
```

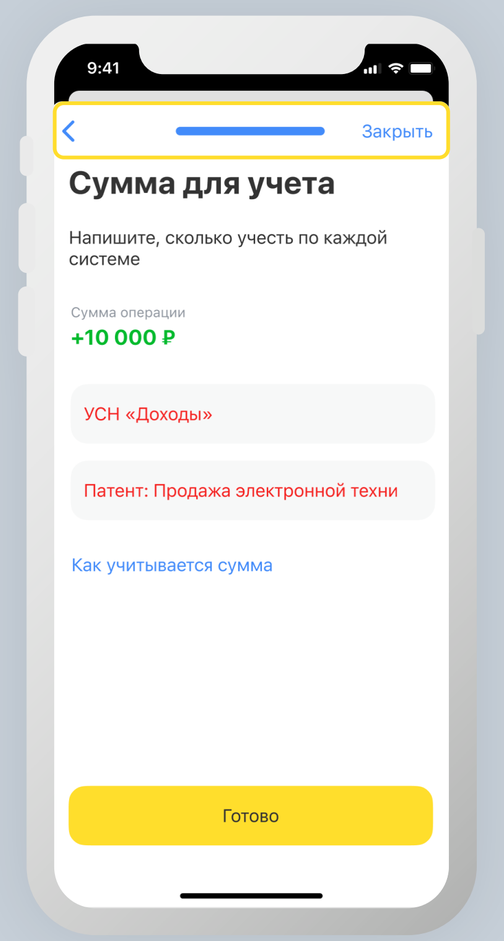
Задача навигации

- **Инкапсуляция логики навигации внутри модуля и вне без связности модулей**
- **Удобная обработка диплинков**
- **Ограничение доступа (авторизация, проверка ролей)**
- Настройка отображения модуля поверх другого модуля

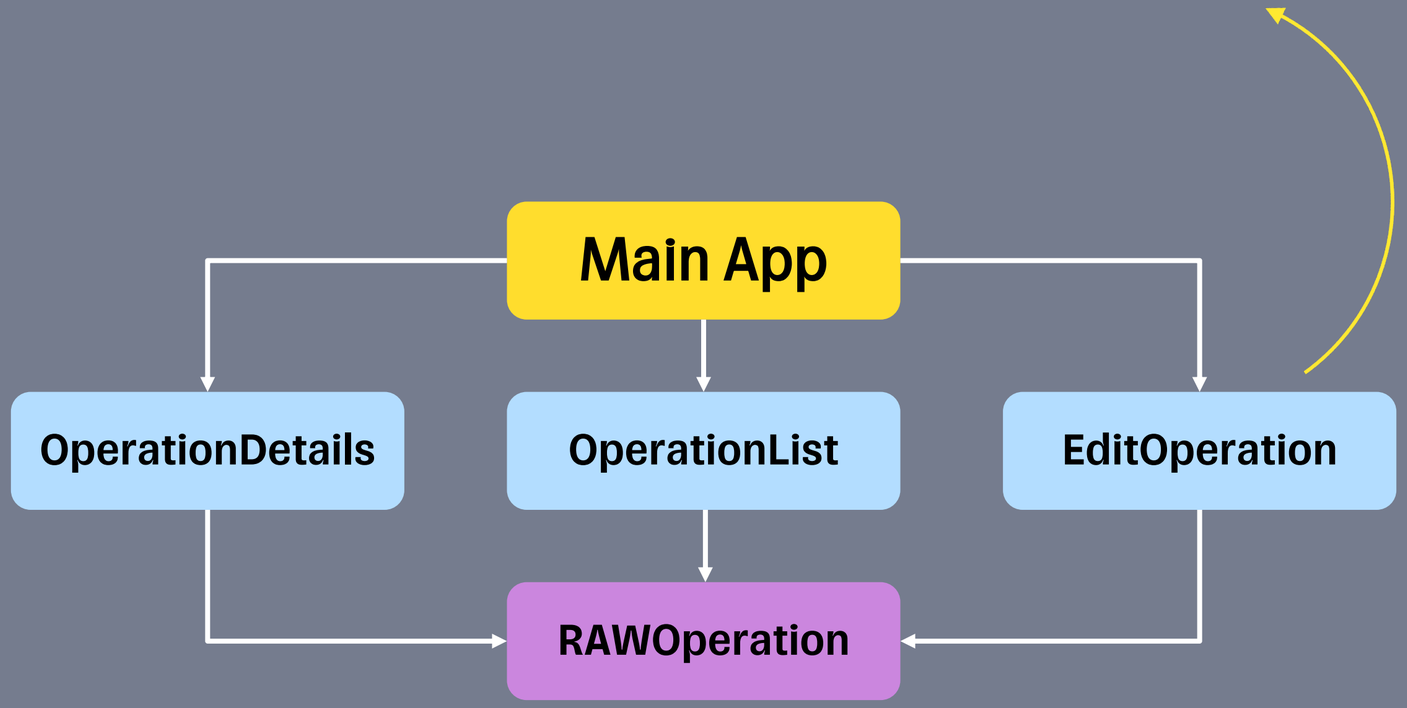


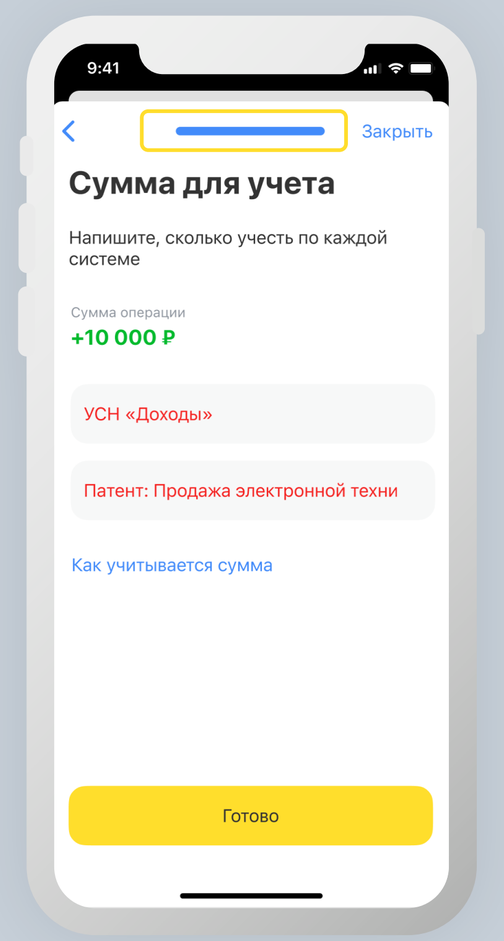

```swift
var editOperaionScreen: DestinationStep<EditOperationViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: EditOperationFactory()
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```



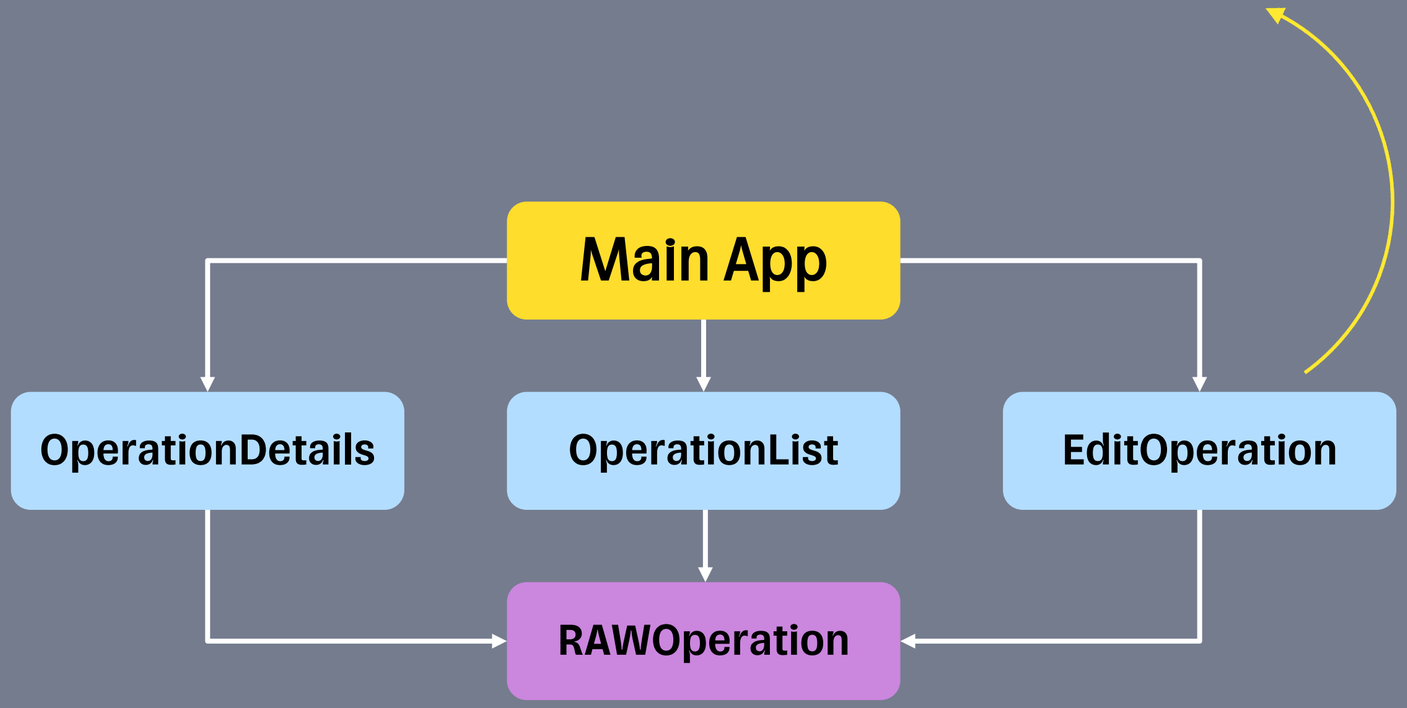


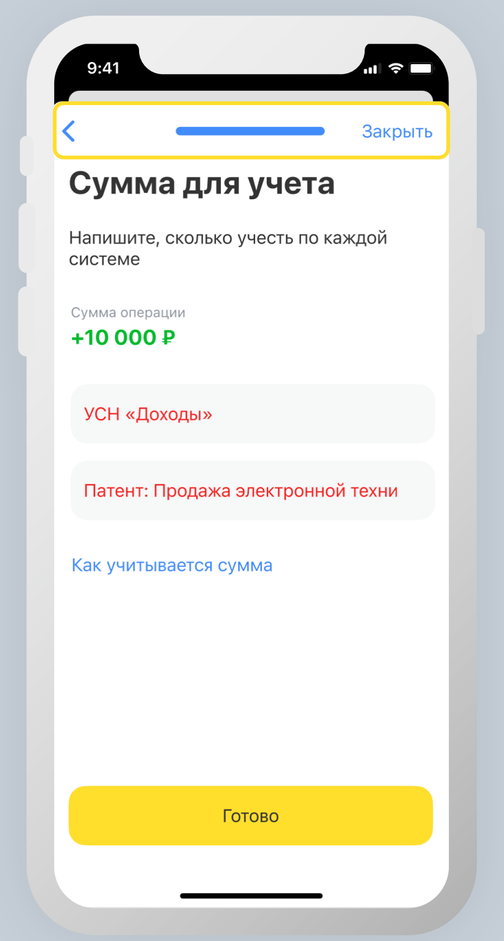

```swift
var editOperaionScreen: DestinationStep<EditOperationViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: EditOperationFactory()
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```



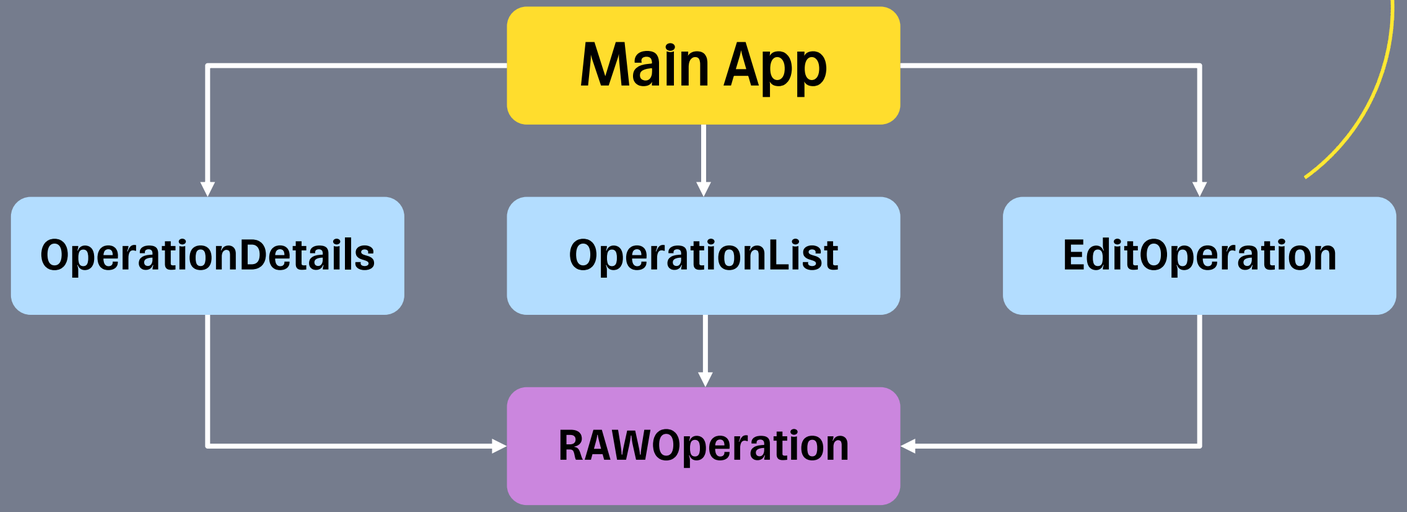


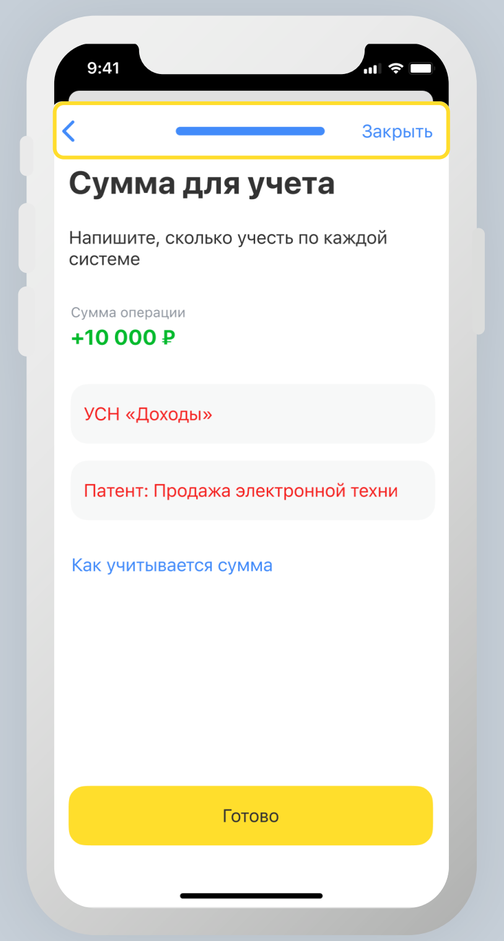

```
var editOperaionScreen: DestinationStep<EditOperationViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: EditOperationFactory()
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```



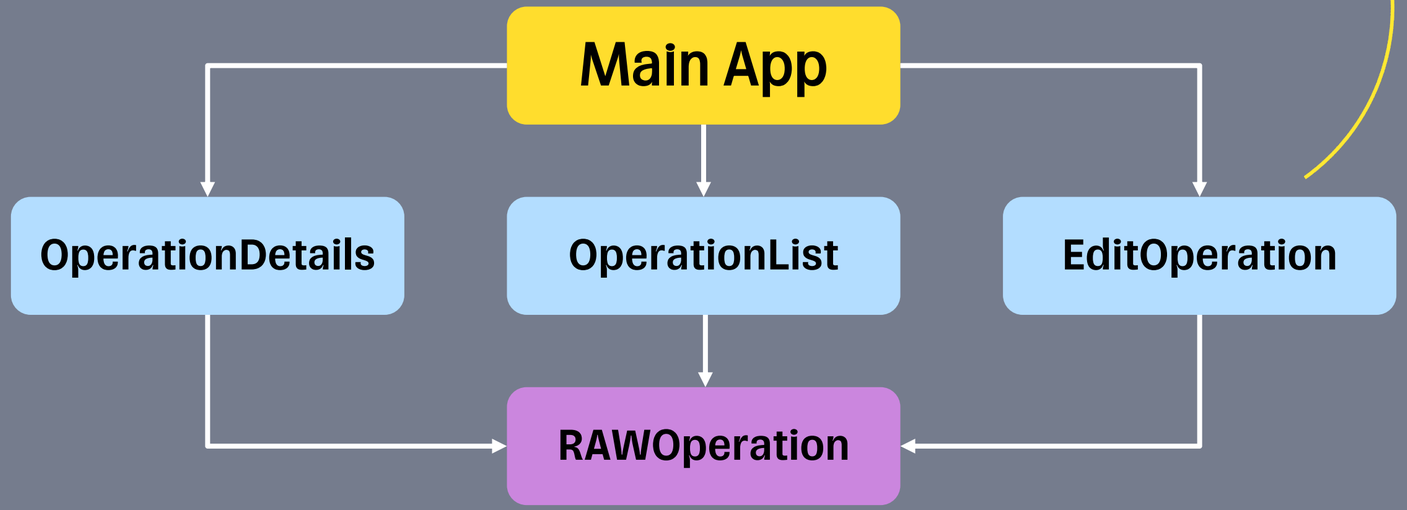


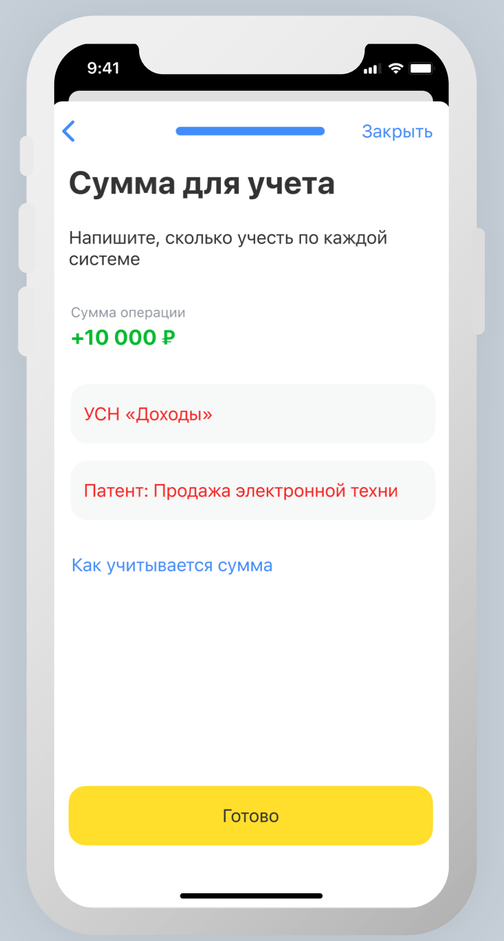

```
var editOperaionScreen: DestinationStep<EditOperationViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: EditOperationFactory()
    )
    .using(ProgressNavigationController.push())
    .from(SingleContainerStep(
        finder: NilFinder(),
        factory: ProgressNavigationControllerFactory())
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```



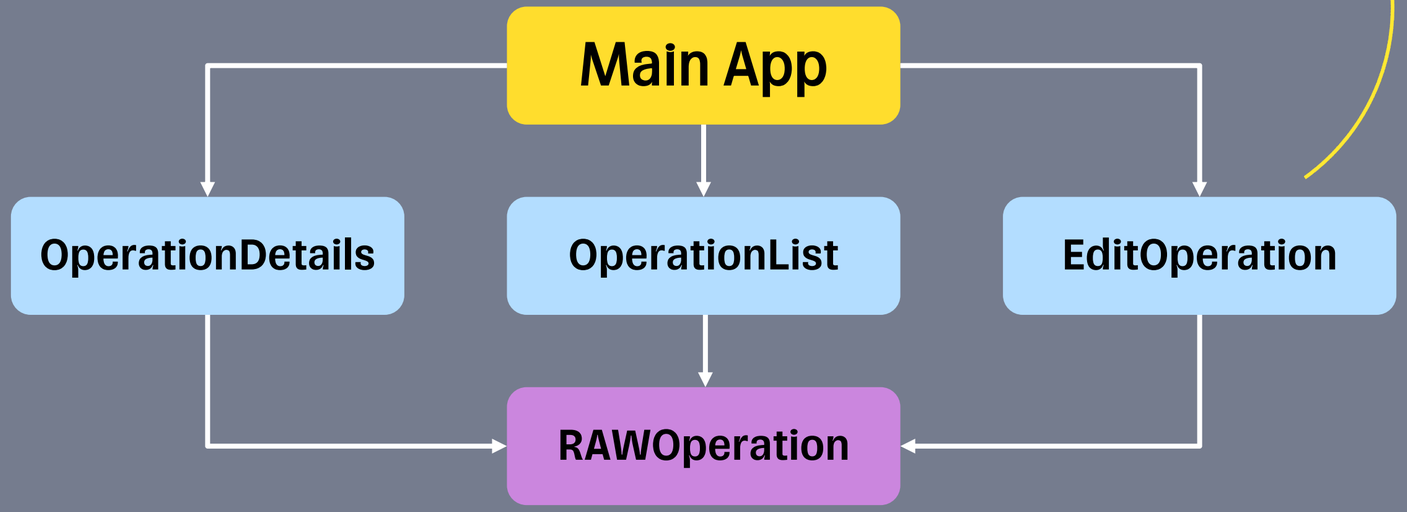


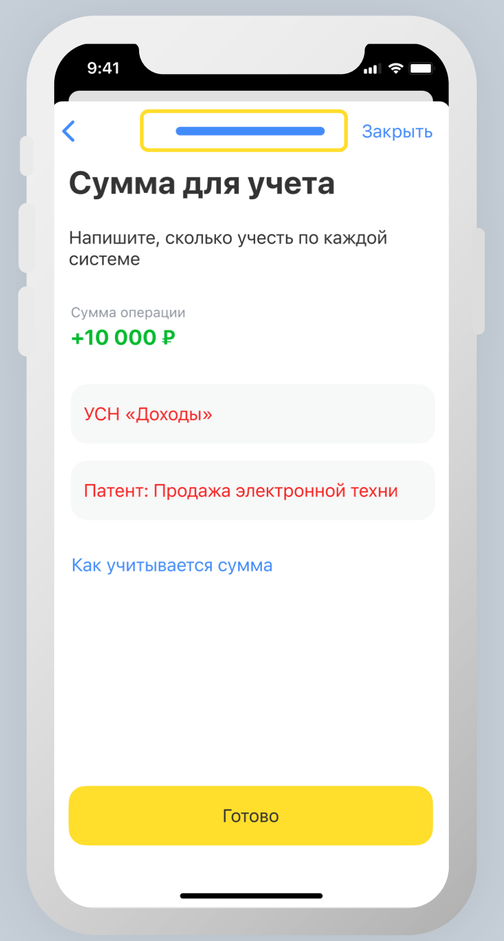

```
var editOperaionScreen: DestinationStep<EditOperationViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: EditOperationFactory()
    )
    .using(ProgressNavigationController.push())
    .from(SingleContainerStep(
        finder: NilFinder(),
        factory: ProgressNavigationControllerFactory())
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```



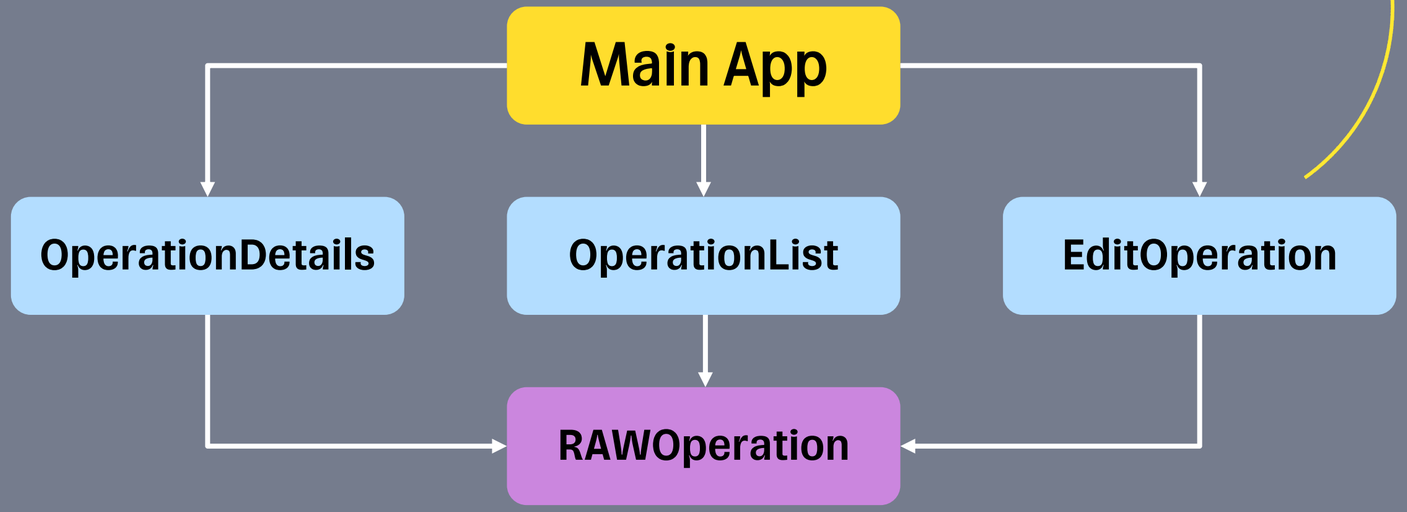



```
var editOperaionScreen: DestinationStep<EditOperationViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: EditOperationFactory()
    )
    .using(ProgressNavigationController.push())
    .from(SingleContainerStep(
        finder: NilFinder(),
        factory: ProgressNavigationControllerFactory())
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(GeneralStep.current())
    .assemble()
}
```






```
var editOperaionScreen: DestinationStep<EditOperationViewController, OperationId> {
    StepAssembly(
        finder: NilFinder(),
        factory: EditOperationFactory()
    )
    .using(ProgressNavigationController.push())
    .from(SingleContainerStep(
        finder: NilFinder(),
        factory: ProgressNavigationControllerFactory())
    )
    .using(GeneralAction.presentModally(presentationStyle: .popover))
    .from(editOperaionCategoryScreen)
    .assemble()
}
```



Задача навигации

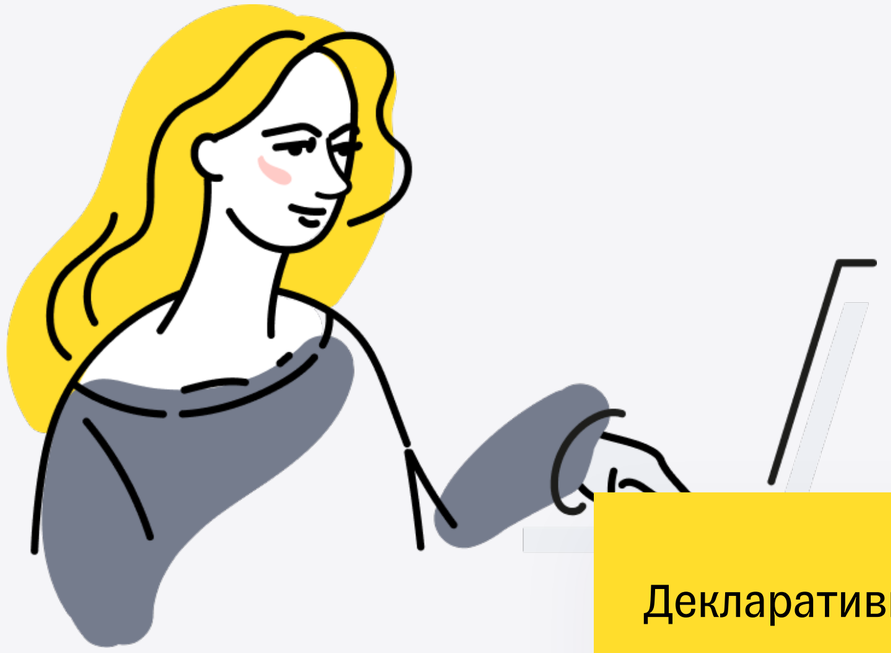
- Инкапсуляция логики навигации внутри модуля и вне без связности модулей
- Удобная обработка диплинков
- Ограничение доступа (авторизация, проверка ролей)
- Настройка отображения модуля поверх другого модуля

Минусы

Усложнение уровня входа

Нет полного контроля до переписывания целого FLOW

Очередная зависимость



Выводы

Декларативная навигация в iOS приложении возможна

Декларативный подход позволяет контролировать навигацию твоего кода

Сложность интеграции в экосистему

IT's TINKOFF

Спасибо за внимание!