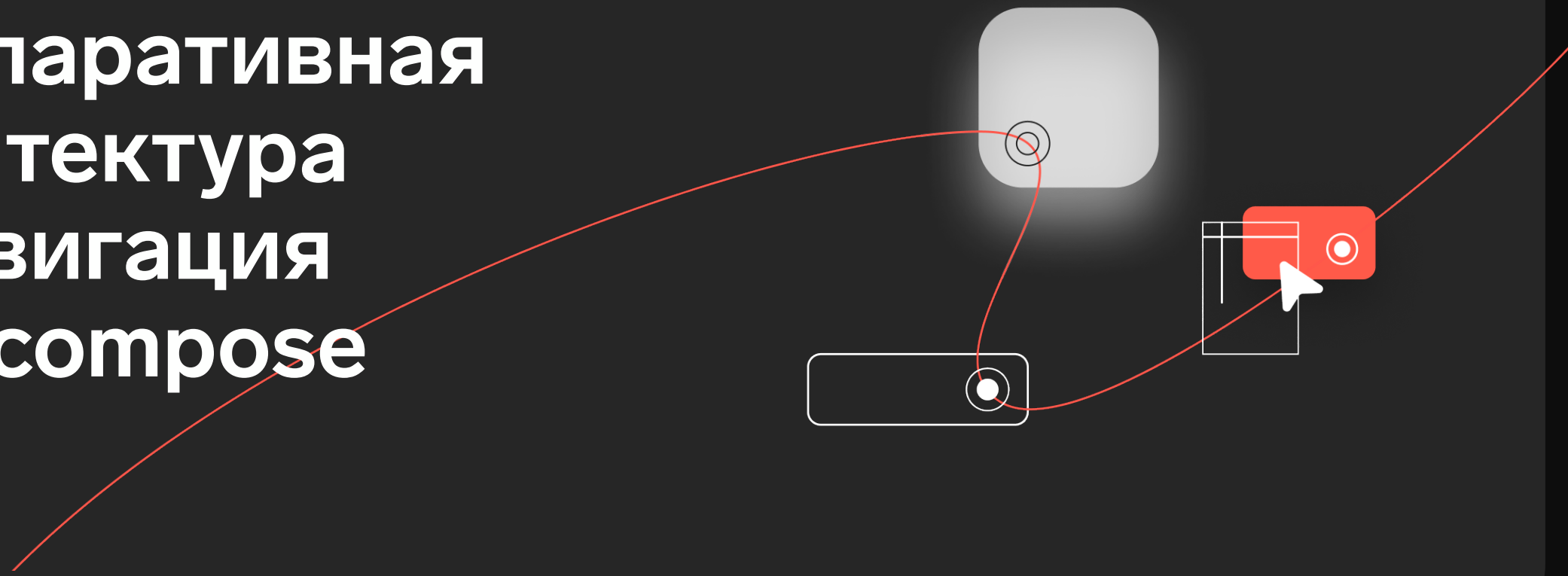
Декларативная архитектура и навигация с Decompose



Контур

Алексей Панов

Ведущий инженер-программист

Обо мне

- В Android разработке с 2015 года
- Фанат Kotlin и новых технологий

ajiekcx

Алексей Панов

Контур


Подпись


Толк


Диадок


Меркурий


Коннект


ОФД


Экстерн


Маркировка


Школа

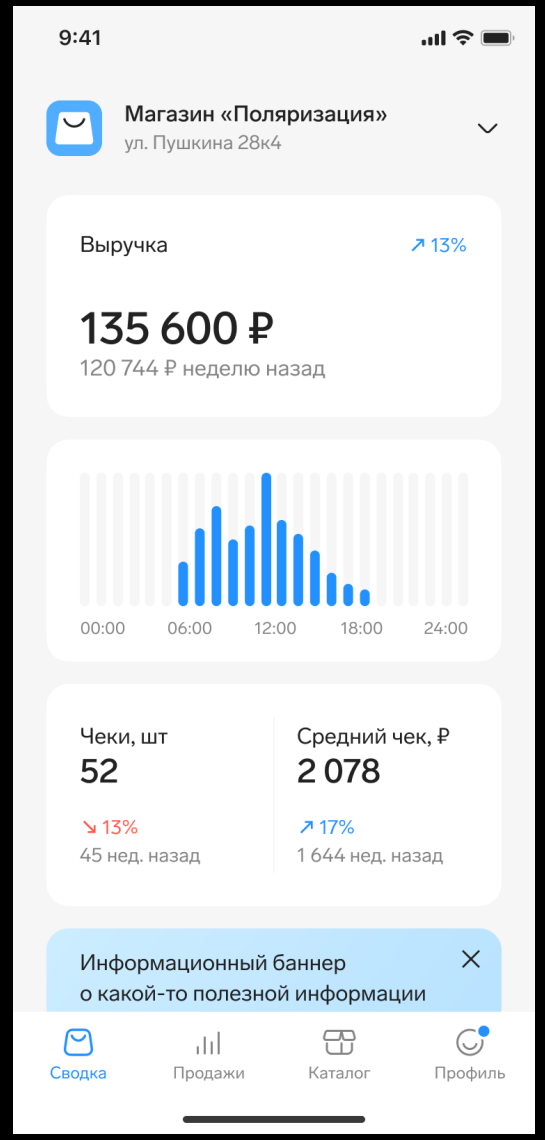

Маркет

Контур.Маркет

Приложение для автоматизации малого и среднего бизнеса

Стек технологий:

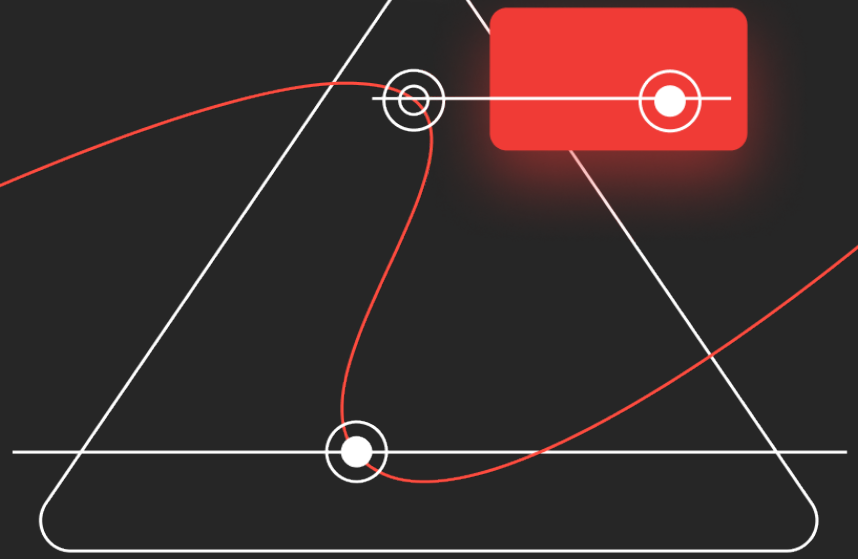
- KMM
- Decompose
- SwiftUI
- Jetpack Compose



План доклада

- Декларативная архитектура и навигация
- Основы Decompose
- Особенности Jetpack Compose и SwiftUI
- Интеграция с императивными фичами
- Многомодульность с Decompose

Декларативная архитектура



Архитектура

Подходы

Декларативный

Что сделать?

SQL

LISP

Императивный

Как сделать?

Kotlin

Swift

Подходы

Декларативный

Что сделать?

SQL

LISP

Model View Intent

Императивный

Как сделать?

Kotlin

Swift

Model View Presenter

Декларативный UI

В декларативном — описываем экран

```
@Composable
fun Greeting(name: String) {
    Text( text: "Hello $name")
}
```

В императивном — конструируем вьюшки

```
val layout = FrameLayout( context: this)
val textView = TextView( context: this)
textView.text = "Hello world"
layout.addView(textView)
```

В декларативной архитектуре UI опционален

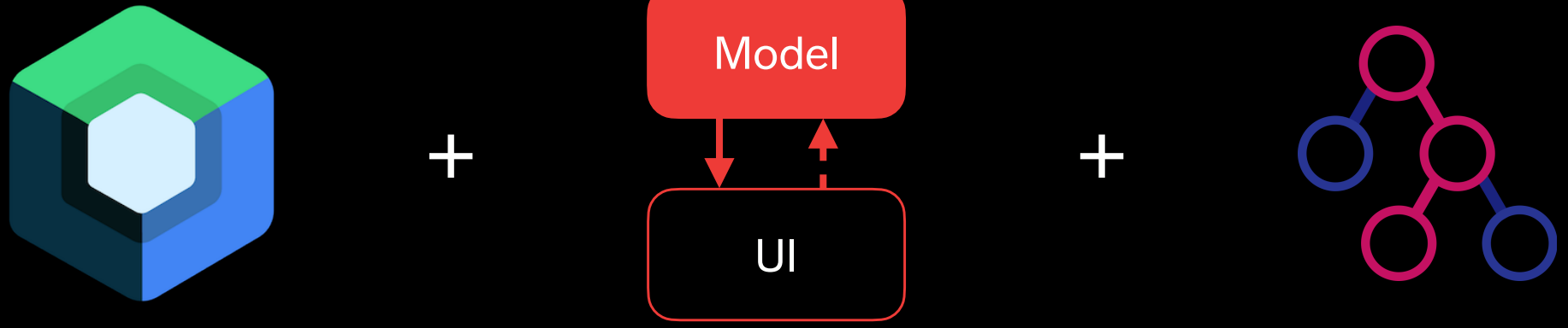
Но если UI есть — он должен быть декларативный

Декларативная архитектура

- Это архитектура, в которой большинство компонентов системы следуют декларативной парадигме

Декларативная архитектура

- Это архитектура, в которой большинство компонентов системы следуют декларативной парадигме



Декларативный UI + UDF + Декларативная навигация

Навигация

Декларативная

- Происходит в бизнес логике
- Можно установить произвольное состояние всего графа навигации

Императивная

- Происходит в UI
- Нельзя установить произвольное состояние

Минусы императивной навигации

- UI отвечает за создание компонентов бизнес логики

Минусы императивной навигации

- UI отвечает за создание компонентов бизнес логики
- Внедрение зависимостей происходит в UI

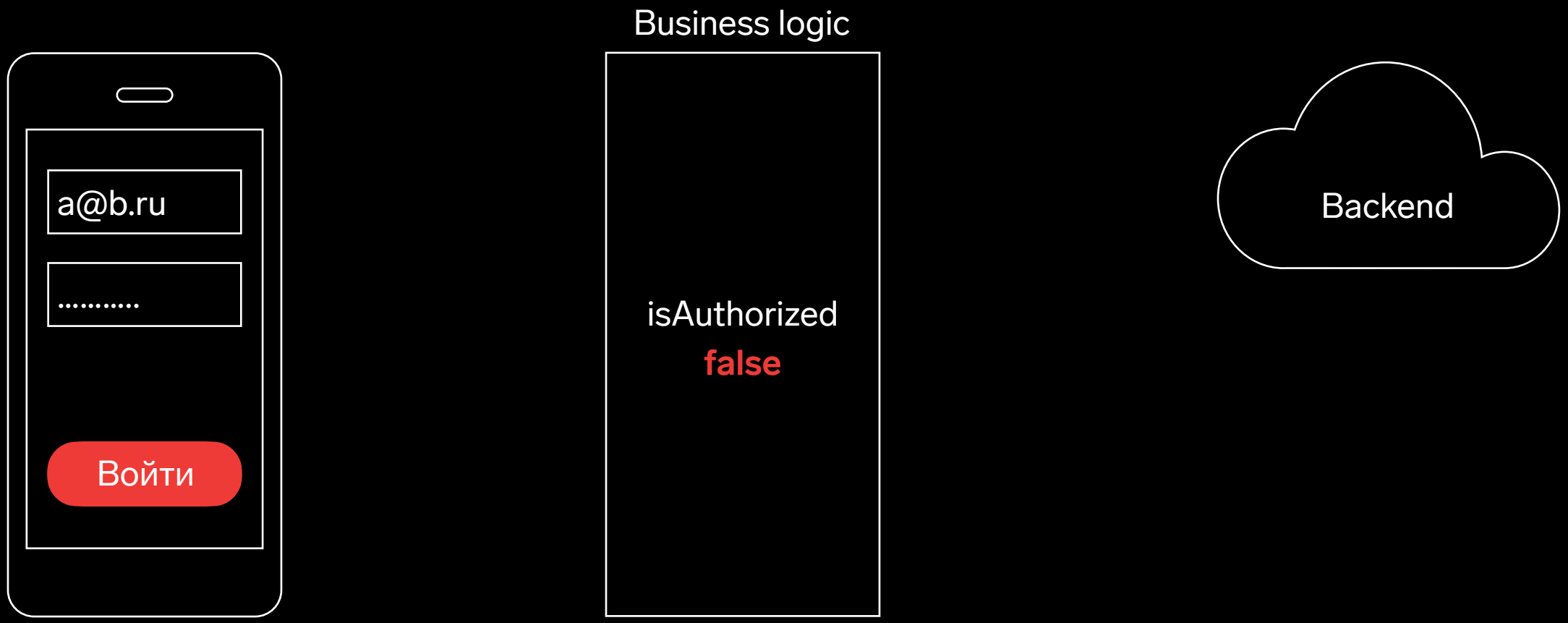
Минусы императивной навигации

- UI отвечает за создание компонентов бизнес логики
- Внедрение зависимостей происходит в UI
- Сильная завязка на конкретный UI

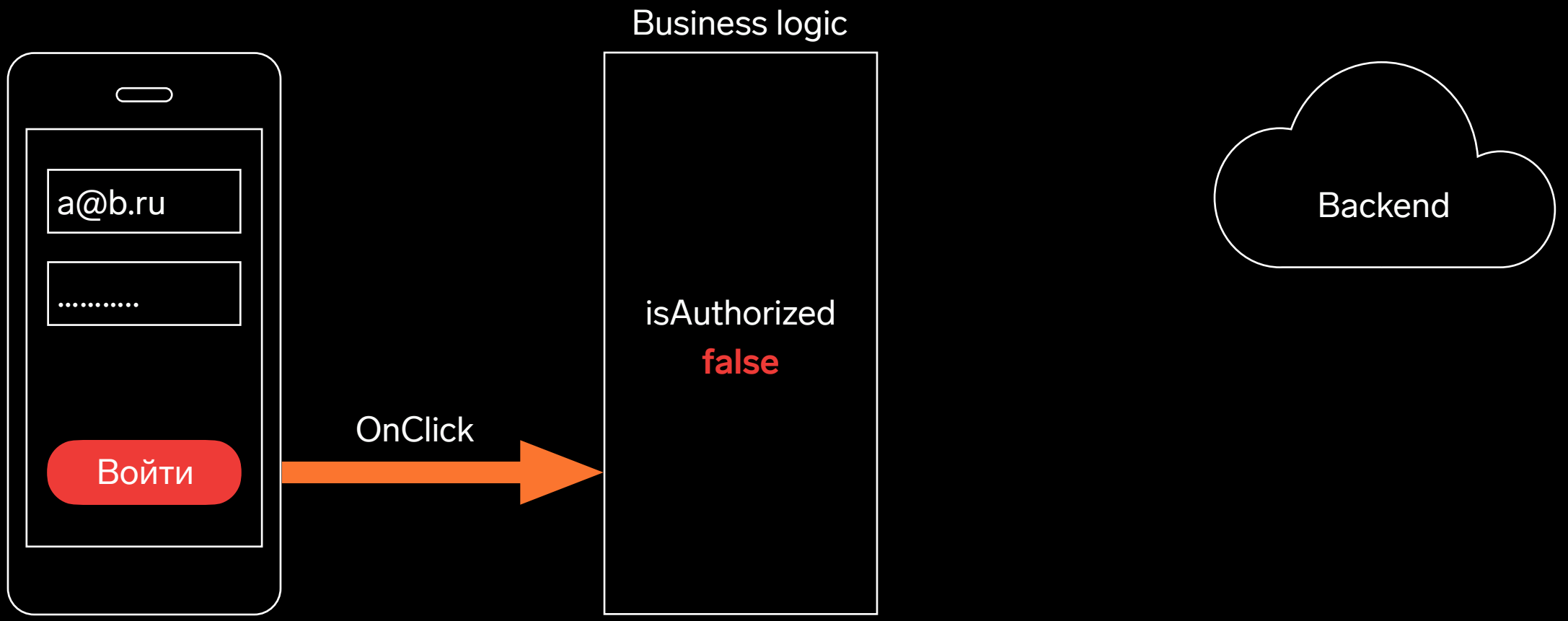
Минусы императивной навигации

- UI отвечает за создание компонентов бизнес логики
- Внедрение зависимостей происходит в UI
- Сильная завязка на конкретный UI
- Состояния бизнес логики и навигации могут разойтись

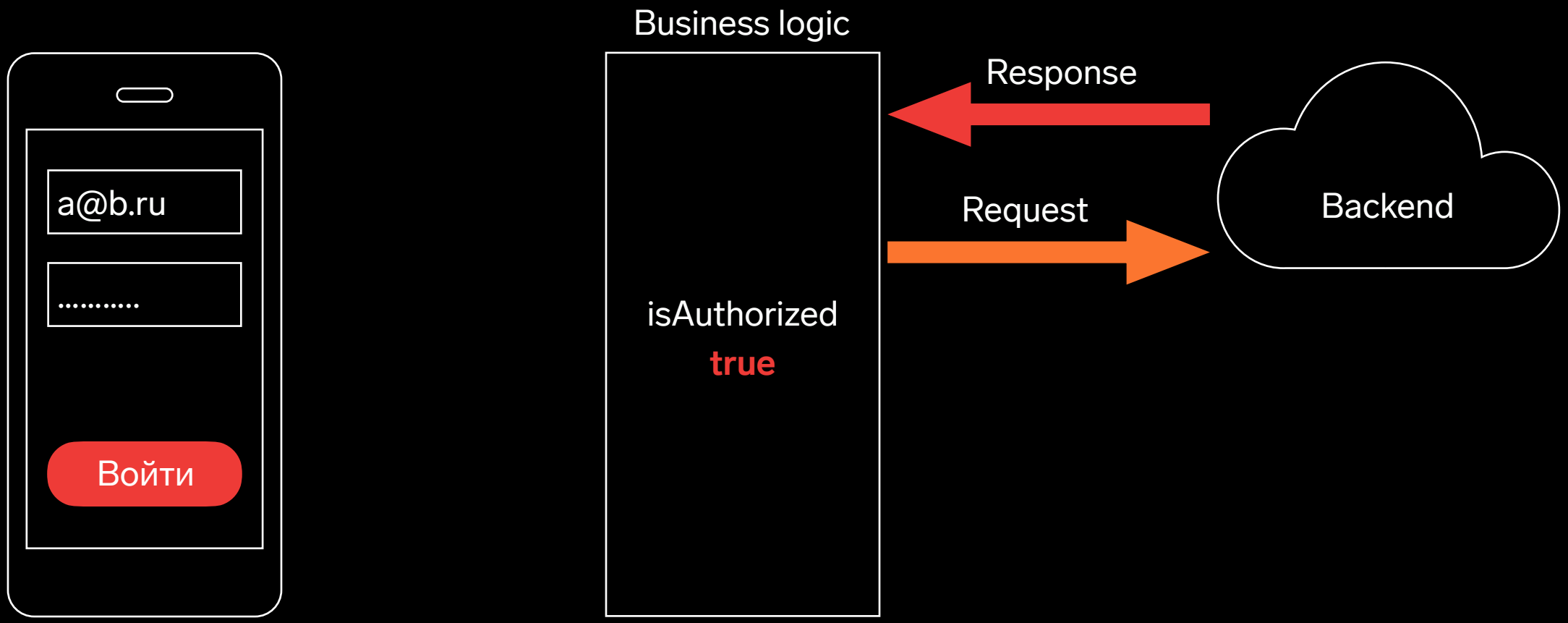
Минусы императивной навигации



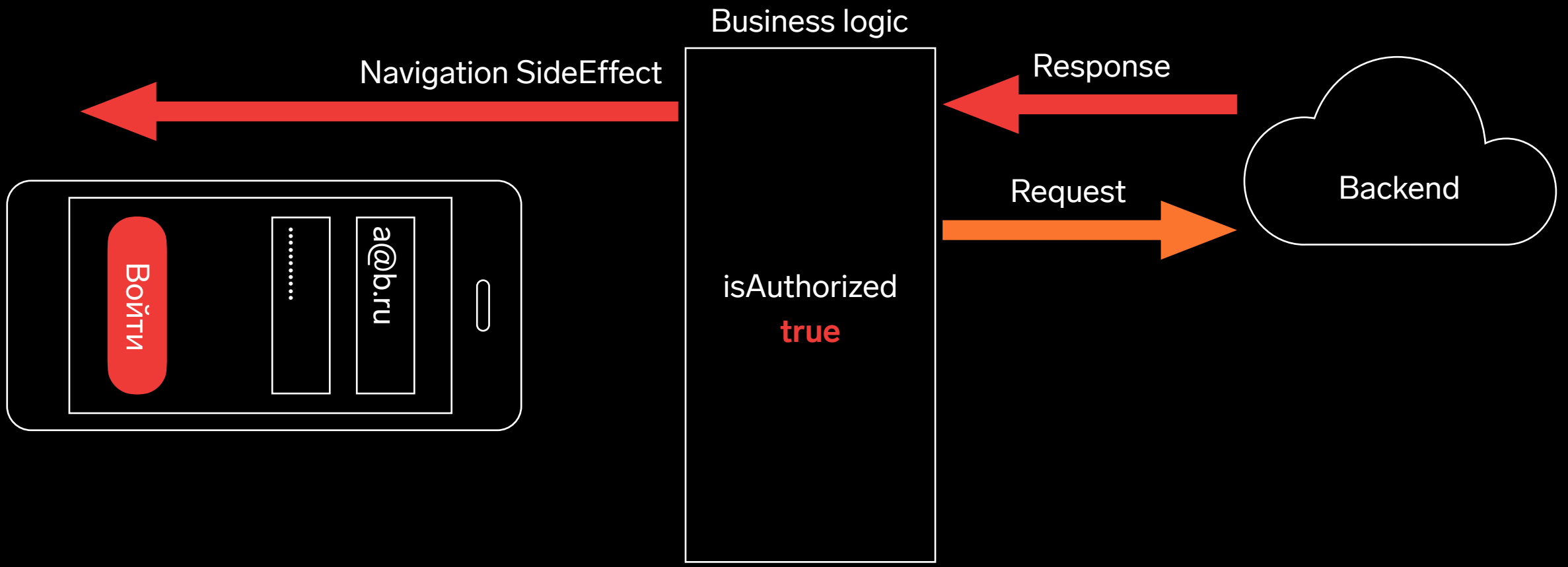
Минусы императивной навигации



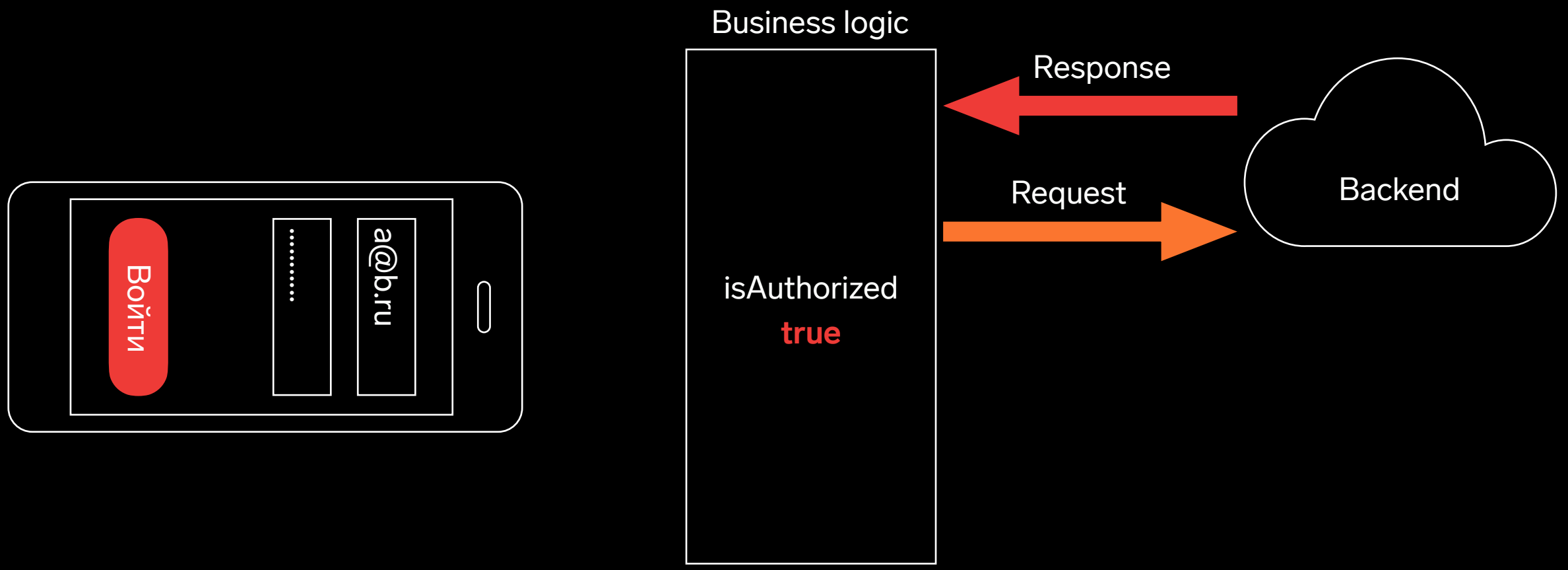
Минусы императивной навигации



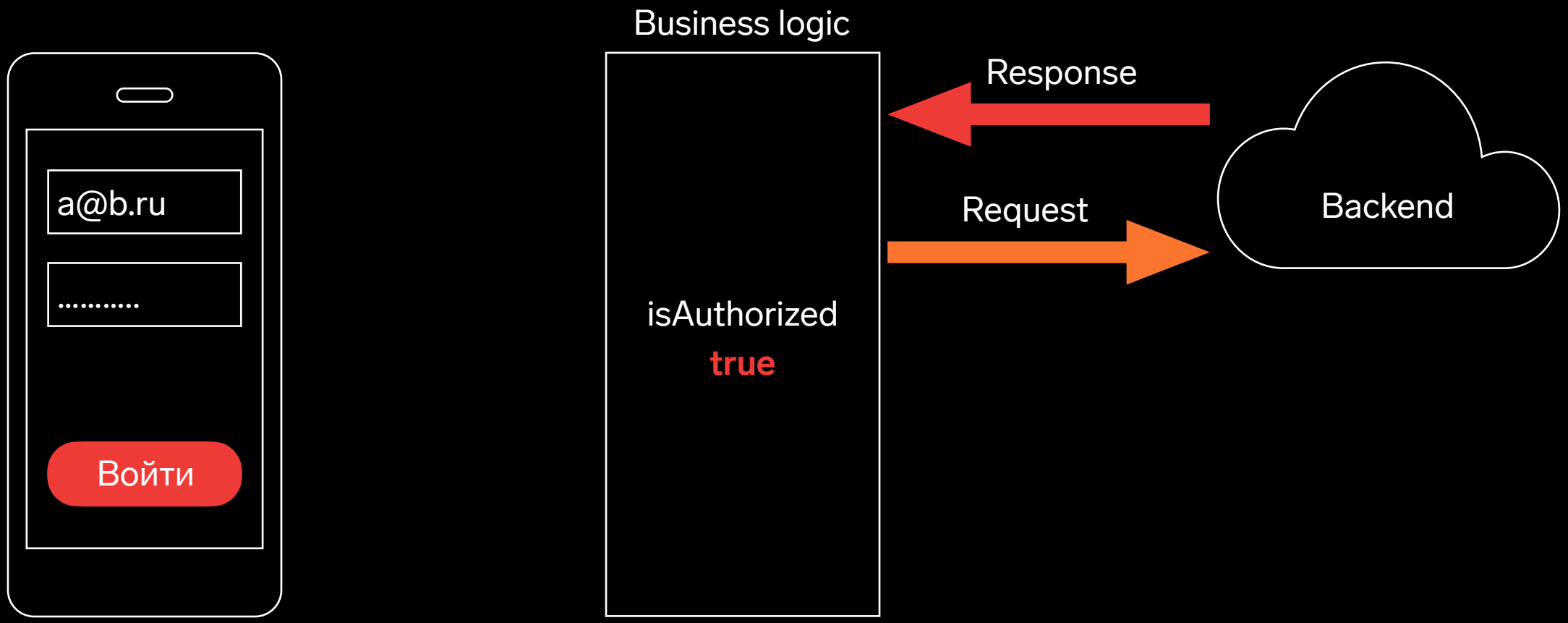
Минусы императивной навигации



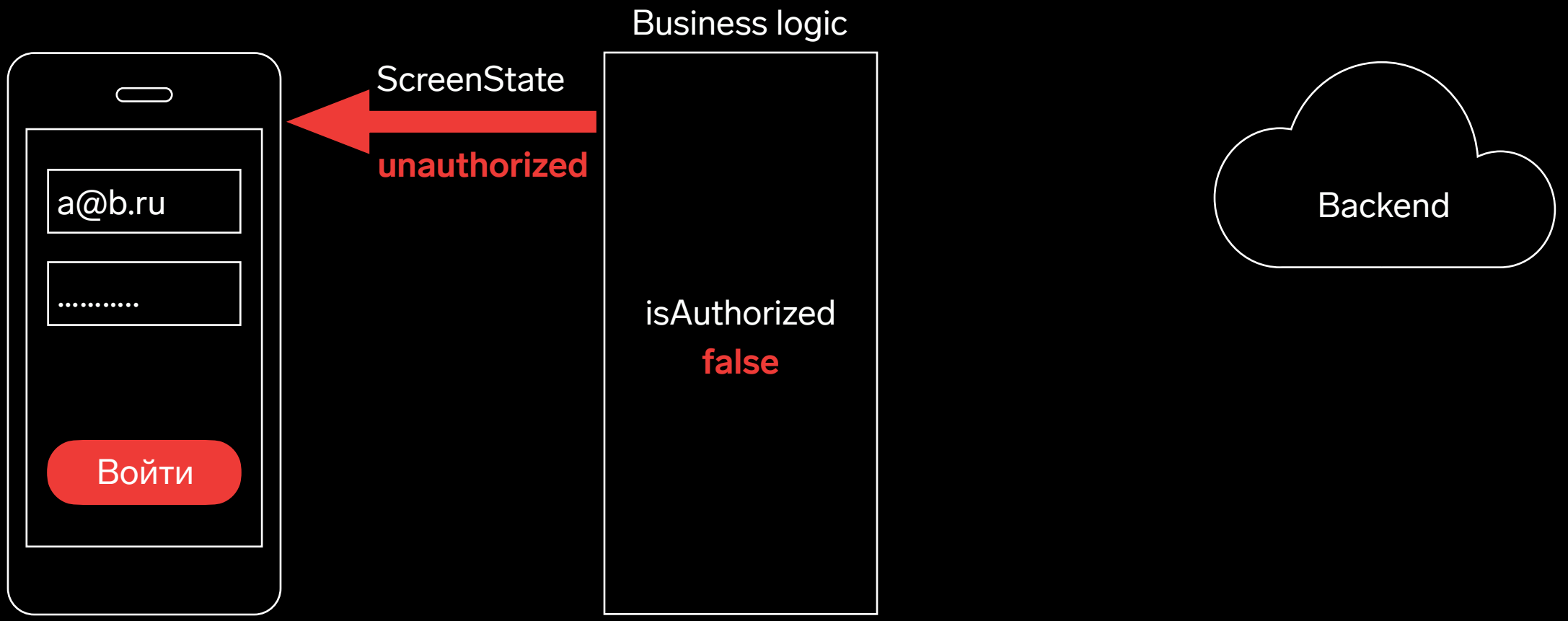
Минусы императивной навигации



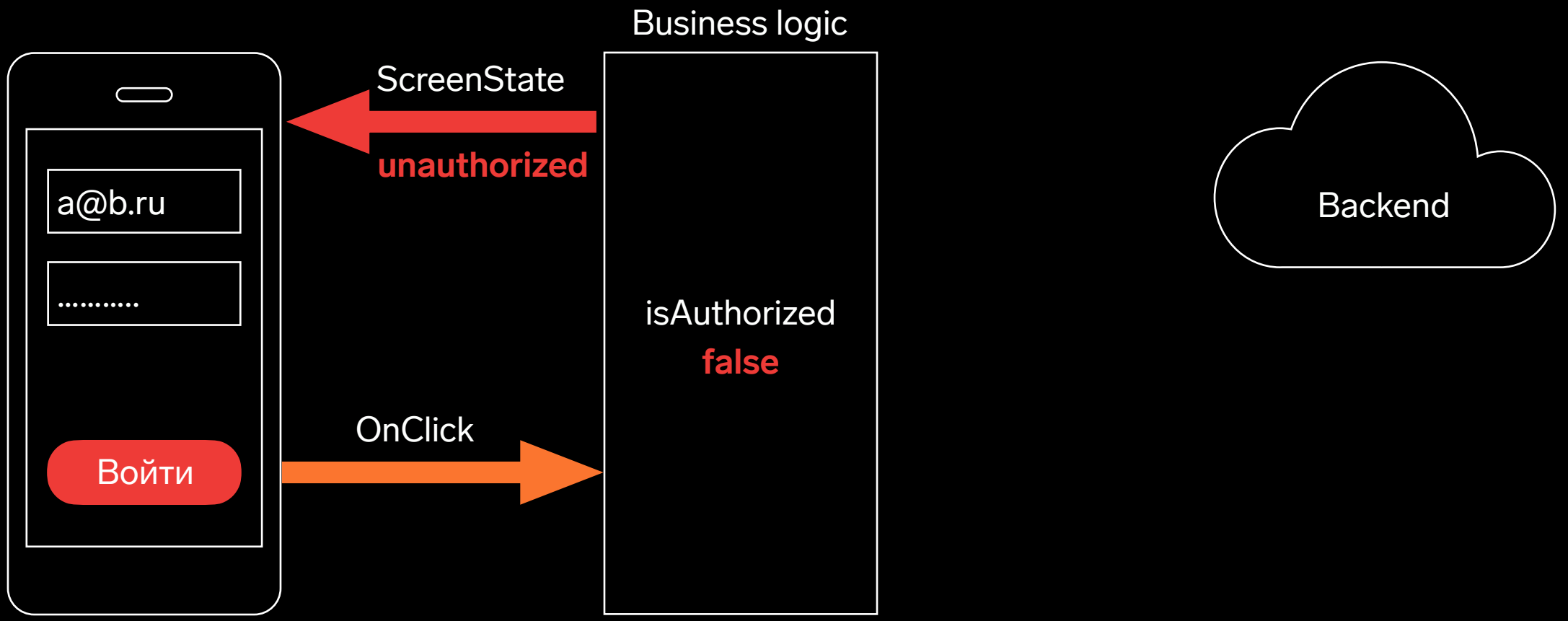
Минусы императивной навигации



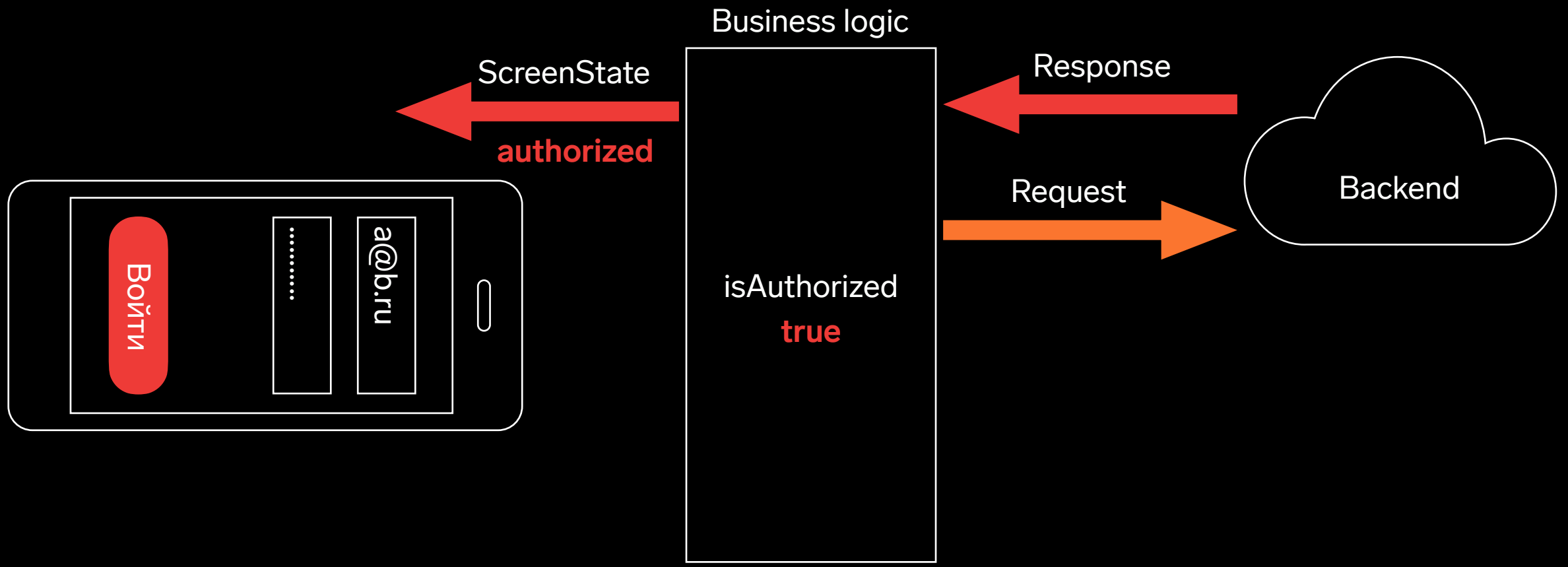
Декларативная навигация



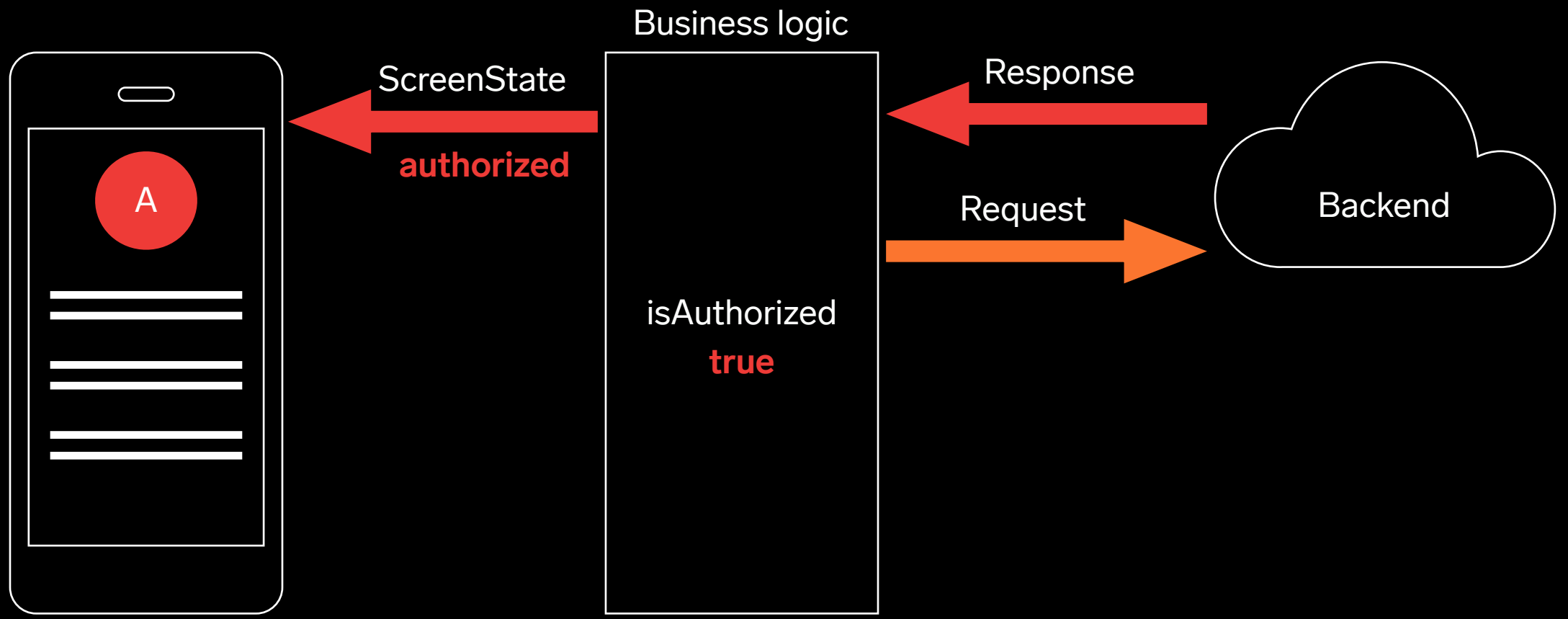
Декларативная навигация



Декларативная навигация



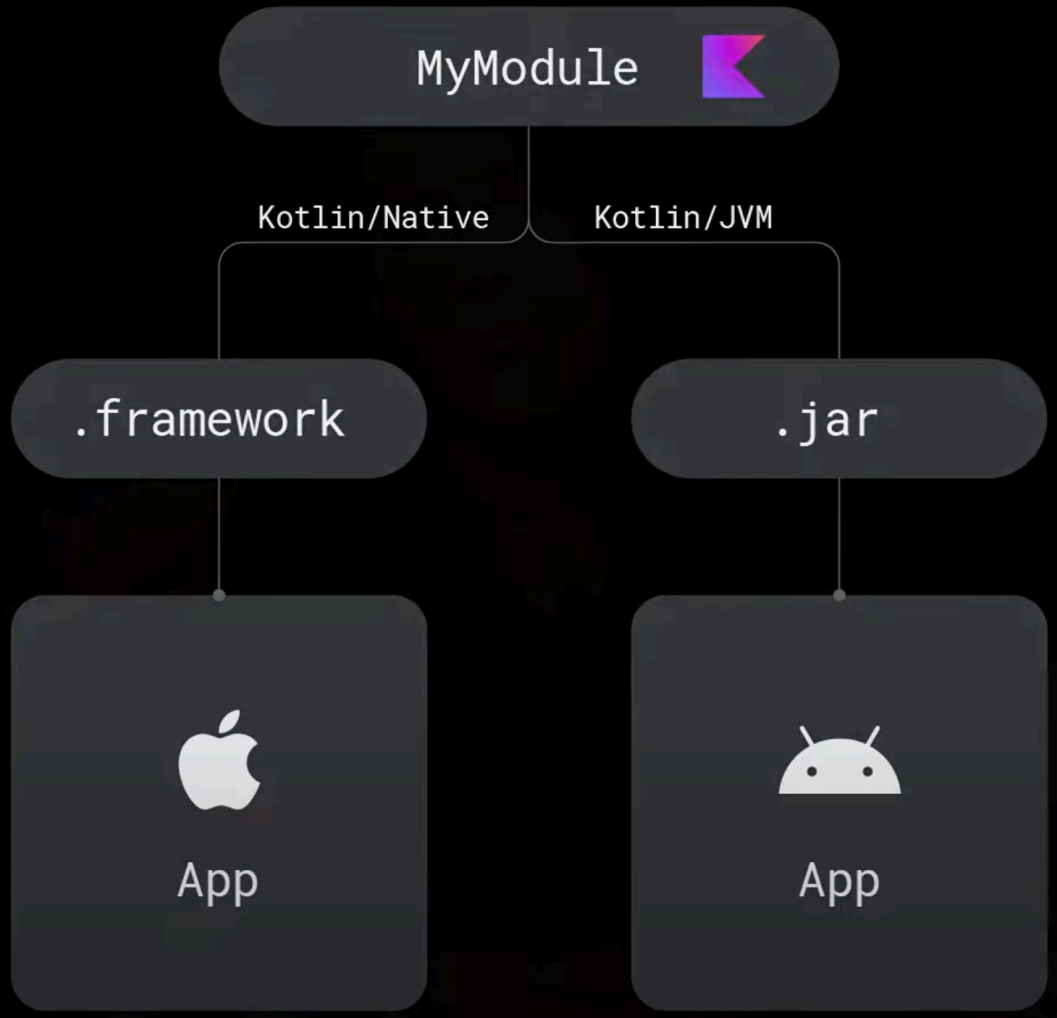
Декларативная навигация




Создал StateFlow и отслеживаю состояние в UI, изи
Никаких либ!
Бэкстек
Мультистек
Диплинки
...
ЖЦ Android
Смерть процесса
...
As Crônicas de Wesley

Декларативная архитектура — независимость от платформы

Kotlin Multiplatform Mobile (KMM)



Требования к библиотеке для навигации

- Должна следовать декларативной парадигме

Требования к библиотеке для навигации

- Должна следовать декларативной парадигме
- Поддерживать основные способы навигации

Требования к библиотеке для навигации

- Должна следовать декларативной парадигме
- Поддерживать основные способы навигации
- Корректно работать с особенностями платформы

Требования к библиотеке для навигации

- Должна следовать декларативной парадигме
- Поддерживать основные способы навигации
- Корректно работать с особенностями платформы
- Быть мультиплатформенной

Decompose

Мультиплатформенная библиотека для навигации и компонентизации от Аркадия Иванова

- Независимость от UI
- Правильная инъекция зависимостей через конструктор
- Общая логика навигации
- Компоненты с жизненным циклом
- Сохранение состояния при смерти процесса в Android и не только

Google

Аналоги Decompose

Все Карты Картинки Видео Новости Ещё Инструменты

Результатов: примерно 0 (0,15 сек.)

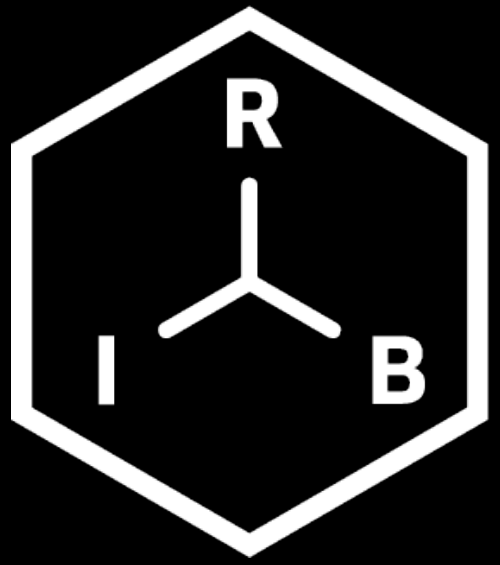
По запросу **Аналоги Decompose** ничего не найдено.

Рекомендации:

- Убедитесь, что все слова написаны без ошибок.
- Попробуйте использовать другие ключевые слова.
- Попробуйте использовать более популярные ключевые слова.

Старший брат Decompose

RIBs (Router Interactor Builder) — архитектурный фреймворк от Uber
для управления состоянием и навигацией в приложениях
для Android и iOS



CROSS-PLATFORM
MOBILE ARCHITECTURE

Возможности Decompose и RIBs

	Иерархическая структура	Сохранение состояния	Независимость от UI	Мультиплатформенность	Тулинг
Decompose	✅	✅	✅	✅	❌
RIBs	✅	❌	✅	❌	✅

Decompose

ComponentContext

ChildStack

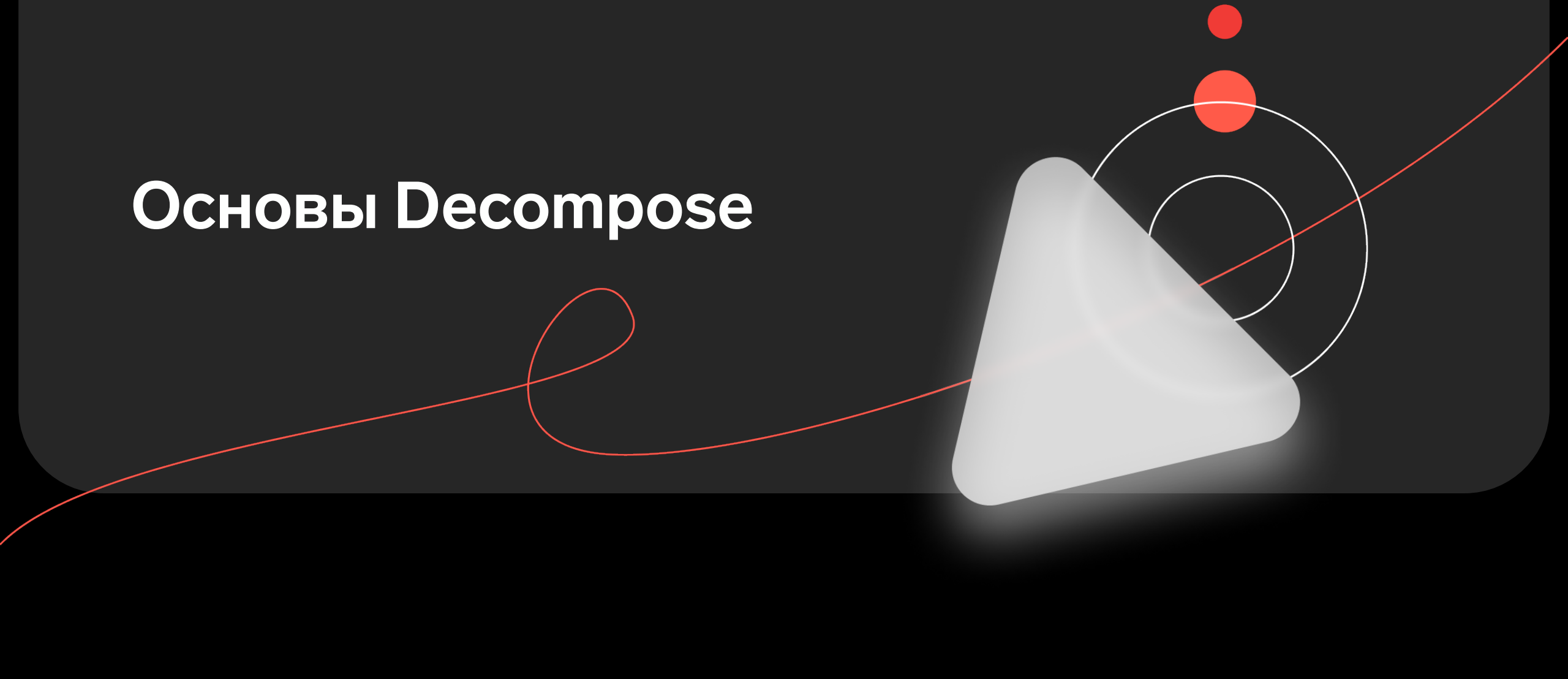
RIBs

Builder Component

Interactor Router

Presenter View

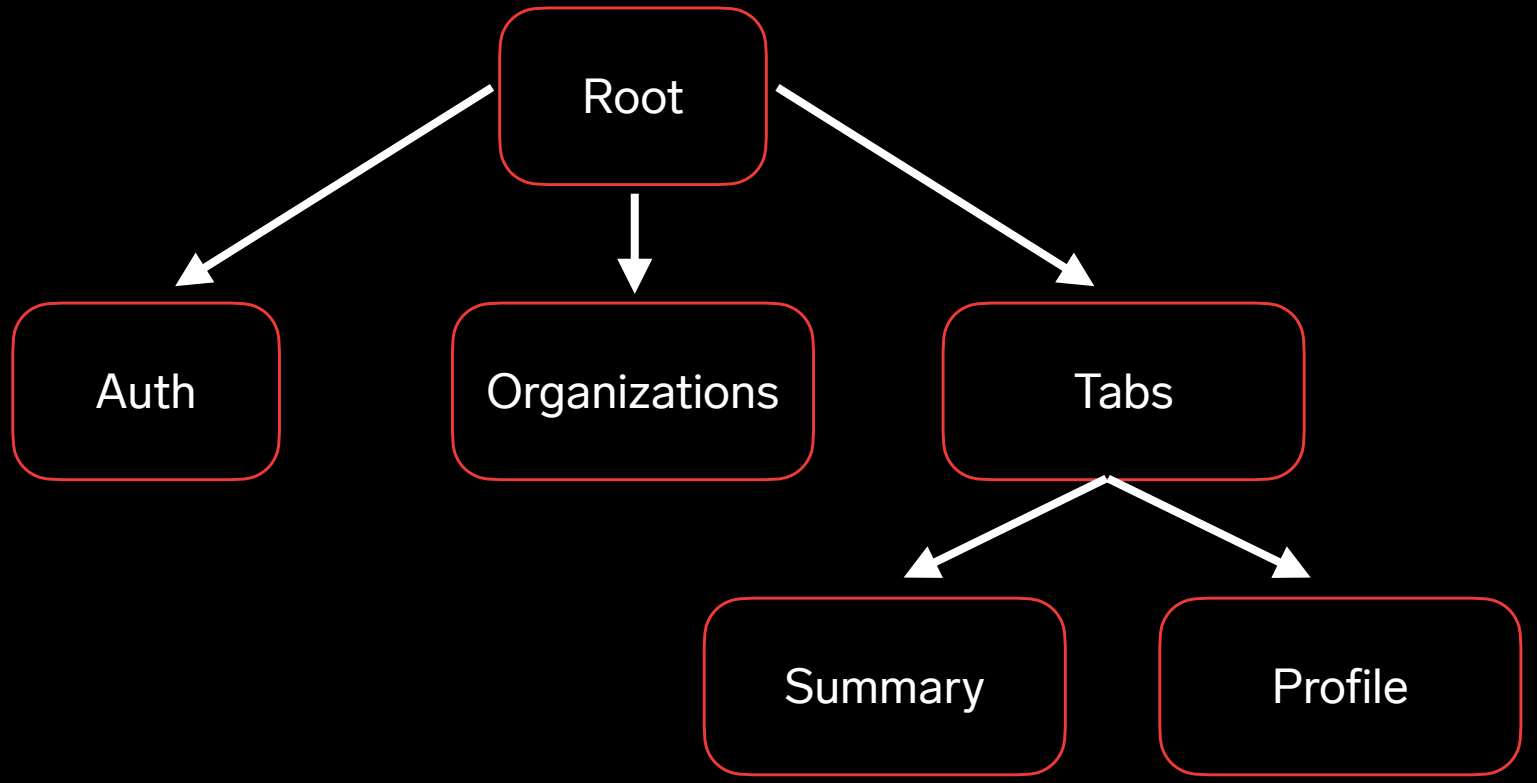
Основы Decompose



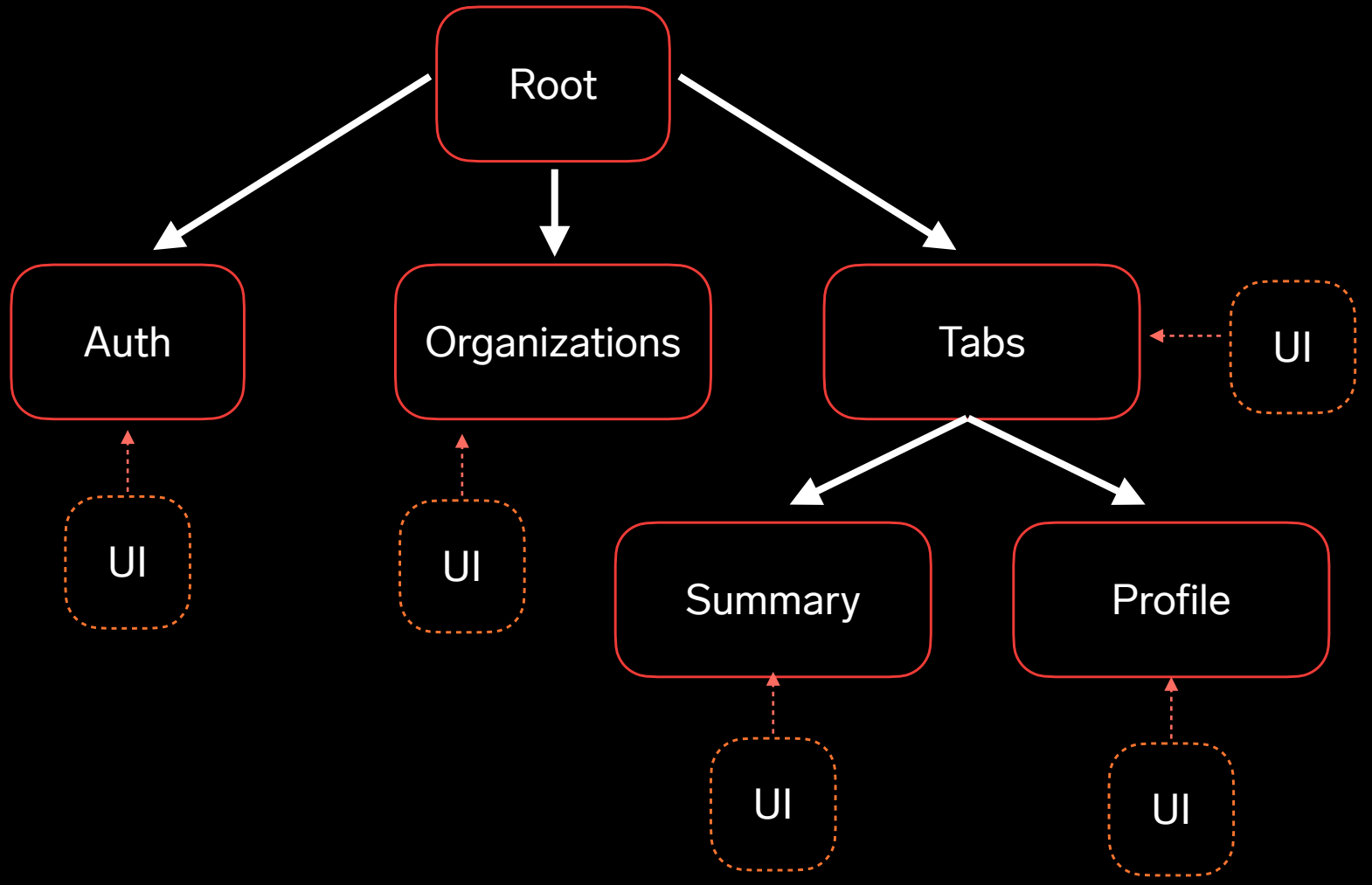
Component в Decompose

- Инкапсулирует логику
- Имеет свой жизненный цикл
- Может содержать дочерние компоненты

Иерархическая структура компонентов



Иерархическая структура компонентов



Интерфейс Component context

- LifecycleOwner — отслеживание ЖЦ компонента
- StateKeeperOwner — сохранение данных при смерти процесса
- InstanceKeeperOwner — сохранение объекта при изменении конфигурации
- BackHandlerOwner — обработка нажатия кнопки «Назад»

Создание корневого компонента

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val root = RootComponent(defaultComponentContext())
    }
}
```

Создание корневого компонента

```
final class RootHolder {
    let lifecycle: LifecycleRegistry
    let root: Root

    init(rootDependencies: RootDependencies) {
        lifecycle = LifecycleRegistryKt.LifecycleRegistry()

        root = RootComponent(
            dependencies: rootDependencies,
            componentContext: DefaultComponentContext(lifecycle: lifecycle)
        )

        lifecycle.onCreate()
    }

    deinit {
        lifecycle.onDestroy()
    }
}
```

Создание корневого компонента

```
struct ContentView: View {
    let rootHolder = RootHolder()

    @Environment(\.scenePhase)
    var scenePhase: ScenePhase

    var body: some View {
        RootView(rootHolder.root)
          .onChange(of: scenePhase) { newPhase in
            switch newPhase {
            case .background: LifecycleRegistryExtKt.stop(rootHolder.lifecycle)
            case .inactive: LifecycleRegistryExtKt.pause(rootHolder.lifecycle)
            case .active: LifecycleRegistryExtKt.resume(rootHolder.lifecycle)
            @unknown default: break
            }
        }
    }
}
```

Имплементация контекста

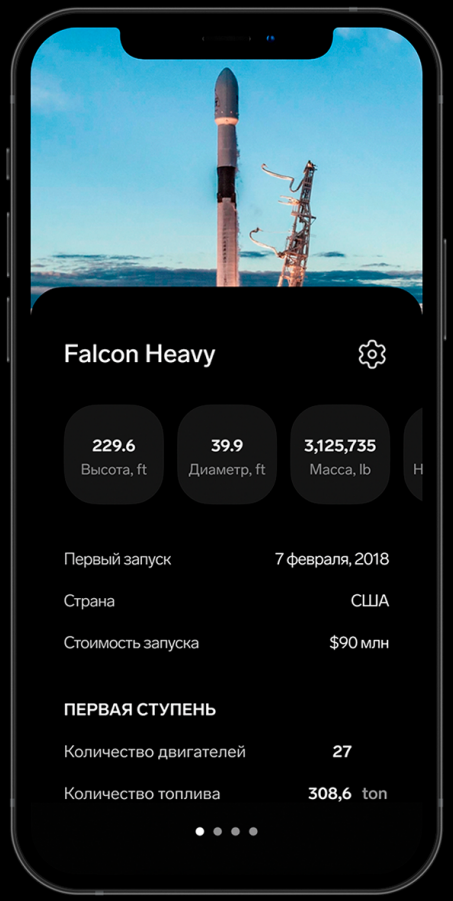
```
interface Root

class RootComponent(
    componentContext: ComponentContext
) : Root, ComponentContext by componentContext {
    // The rest of the code
}
```

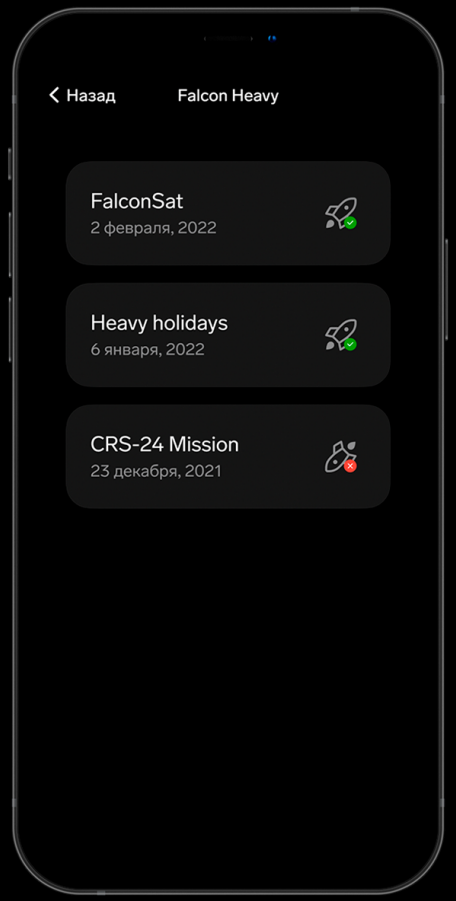
Имплементация контекста

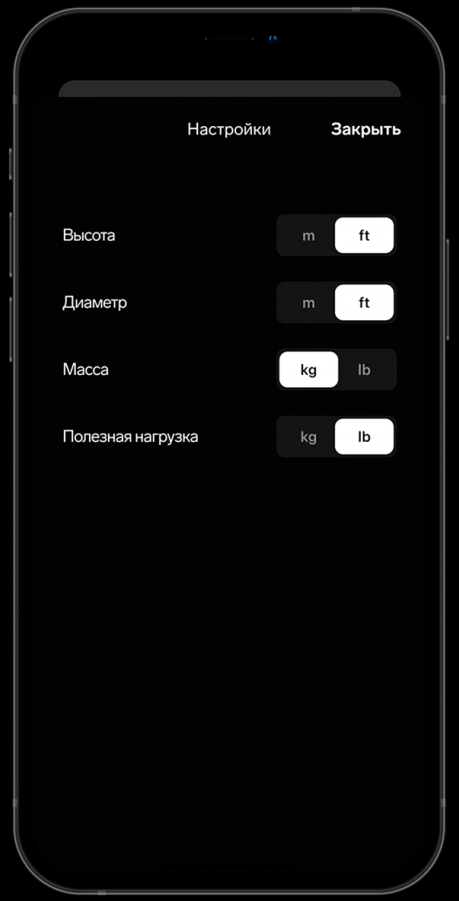
```
interface Root

class RootComponent(
    componentContext: ComponentContext
) : Root, ComponentContext by componentContext {
    // The rest of the code
}
```

Один контекст для одного компонента!

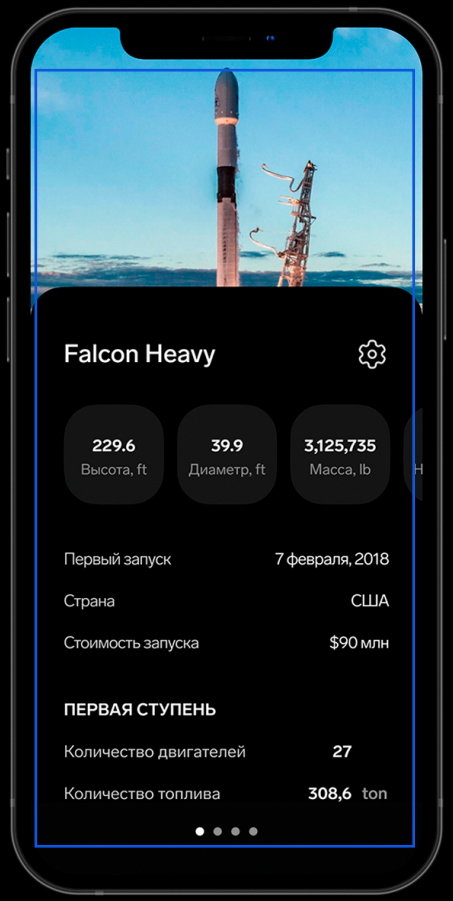
Пример

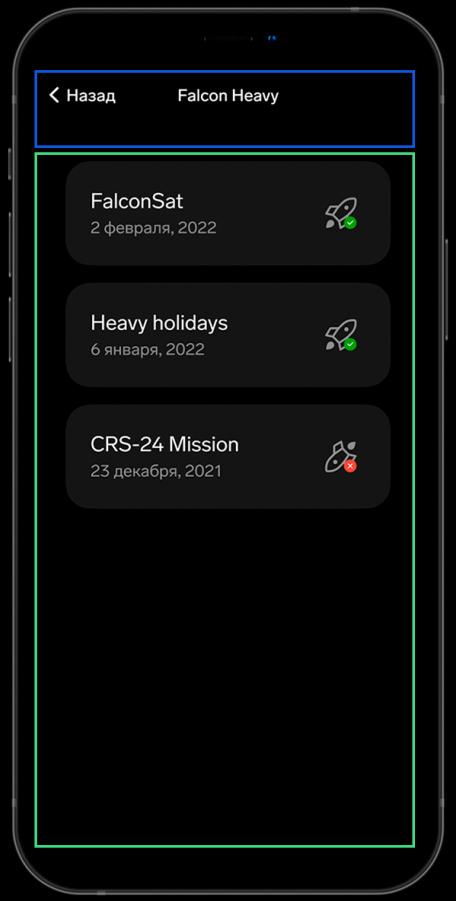

Falcon Heavy
229.6
Высота, ft
39.9
Диаметр, ft
3,125,735
Масса, lb
Первый запуск
7 февраля, 2018
Страна
США
Стоимость запуска
$90 млн
ПЕРВАЯ СТУПЕНЬ
Количество двигателей
27
Количество топлива
308,6 ton


Назад
Falcon Heavy
FalconSat
2 февраля, 2022
Heavy holidays
6 января, 2022
CRS-24 Mission
23 декабря, 2021


Настройки
Закрыть
Высота
m
ft
Диаметр
m
ft
Масса
kg
lb
Полезная нагрузка
kg
lb

Пример

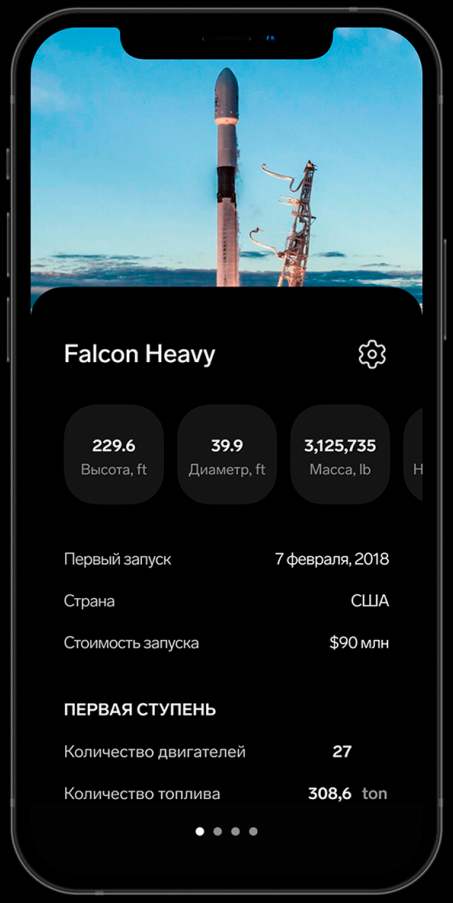




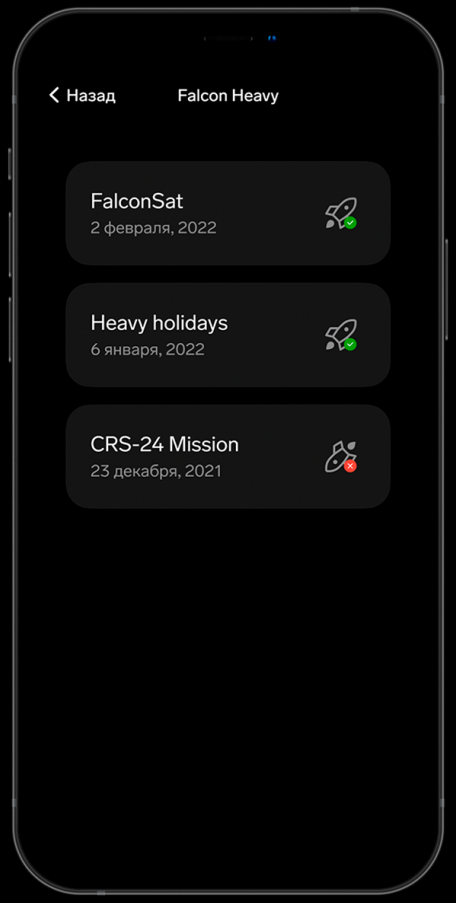
SwiftUI

Compose Multiplatform

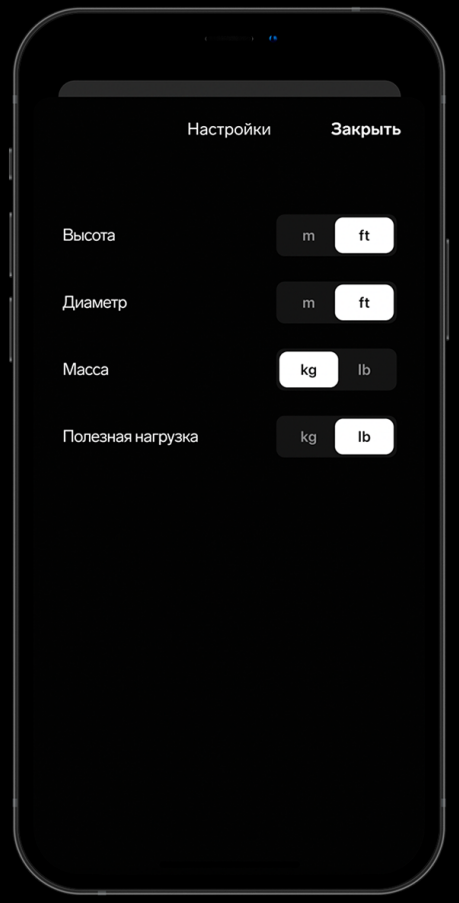
Пример



RocketsComponent

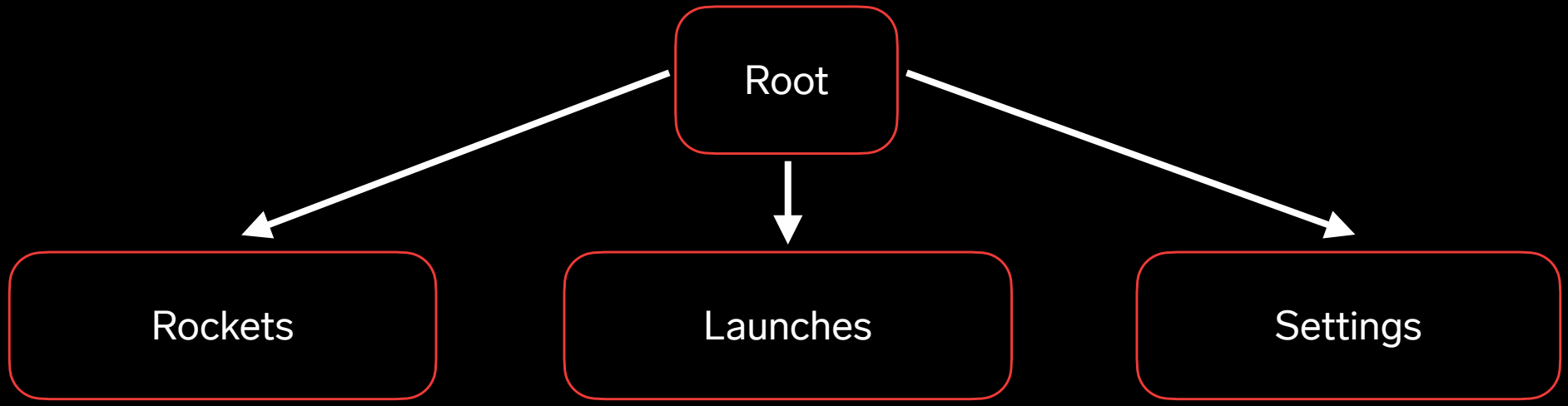


LaunchesComponent

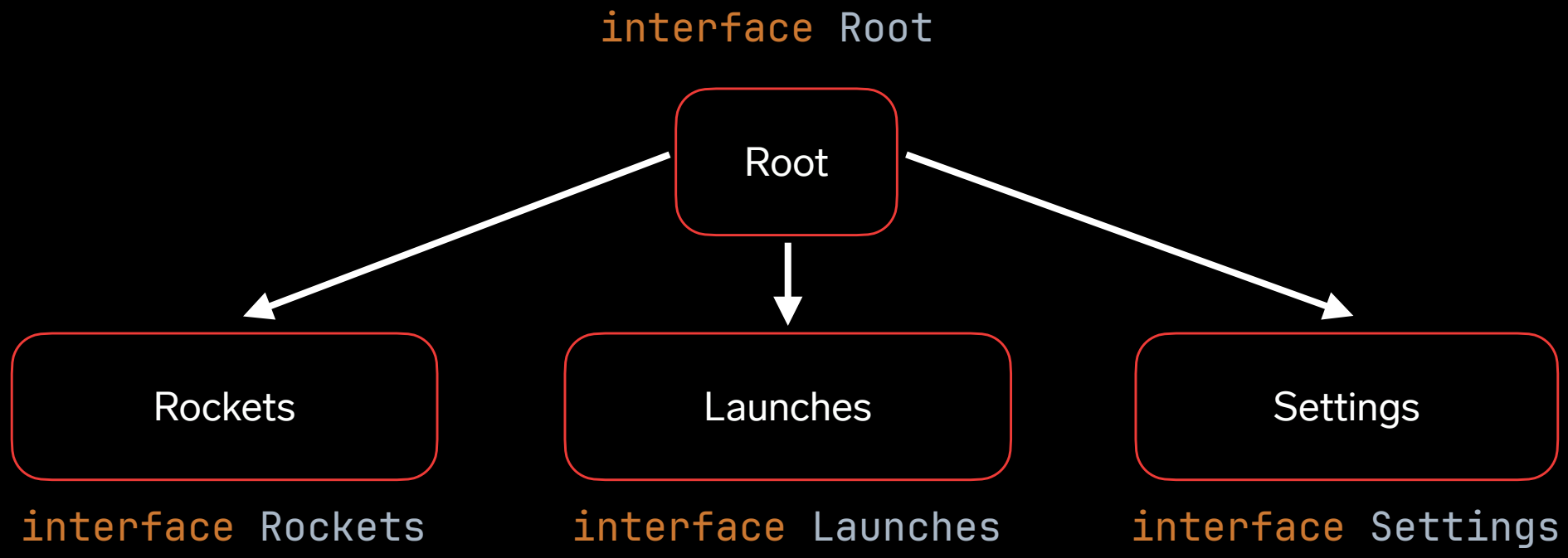


SettingsComponent

Пример



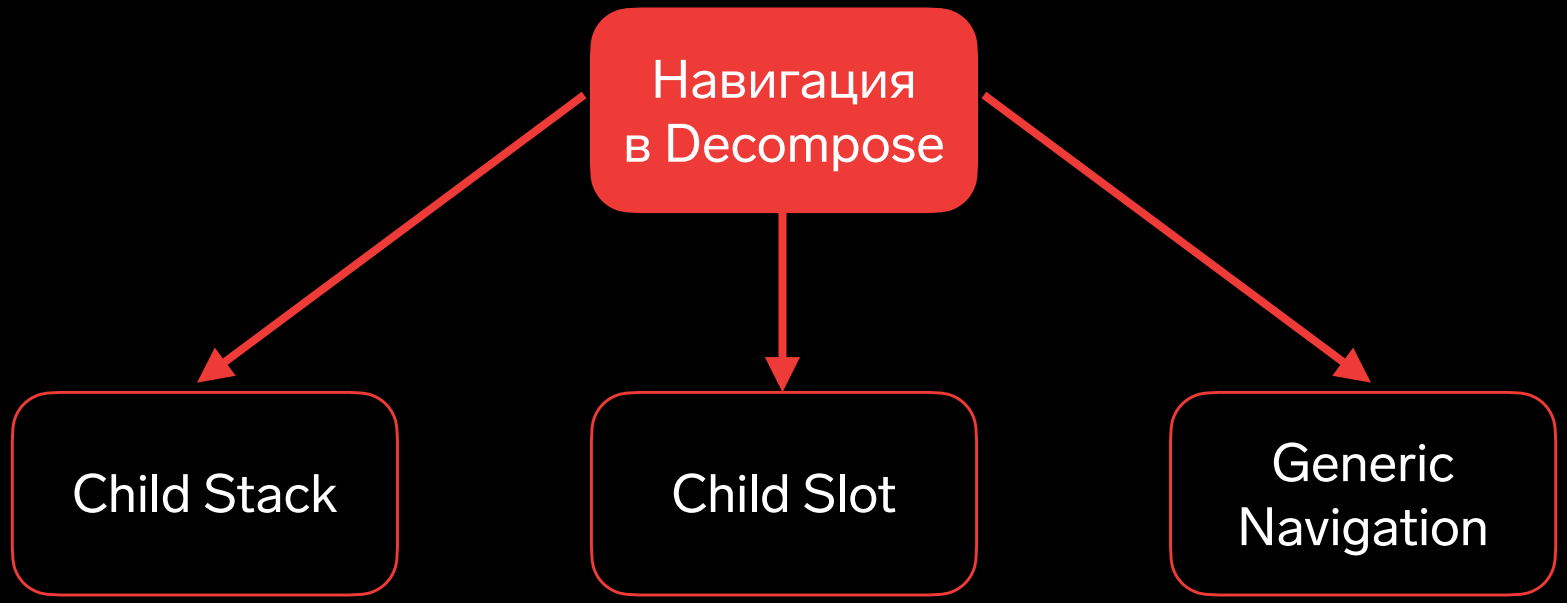
Пример

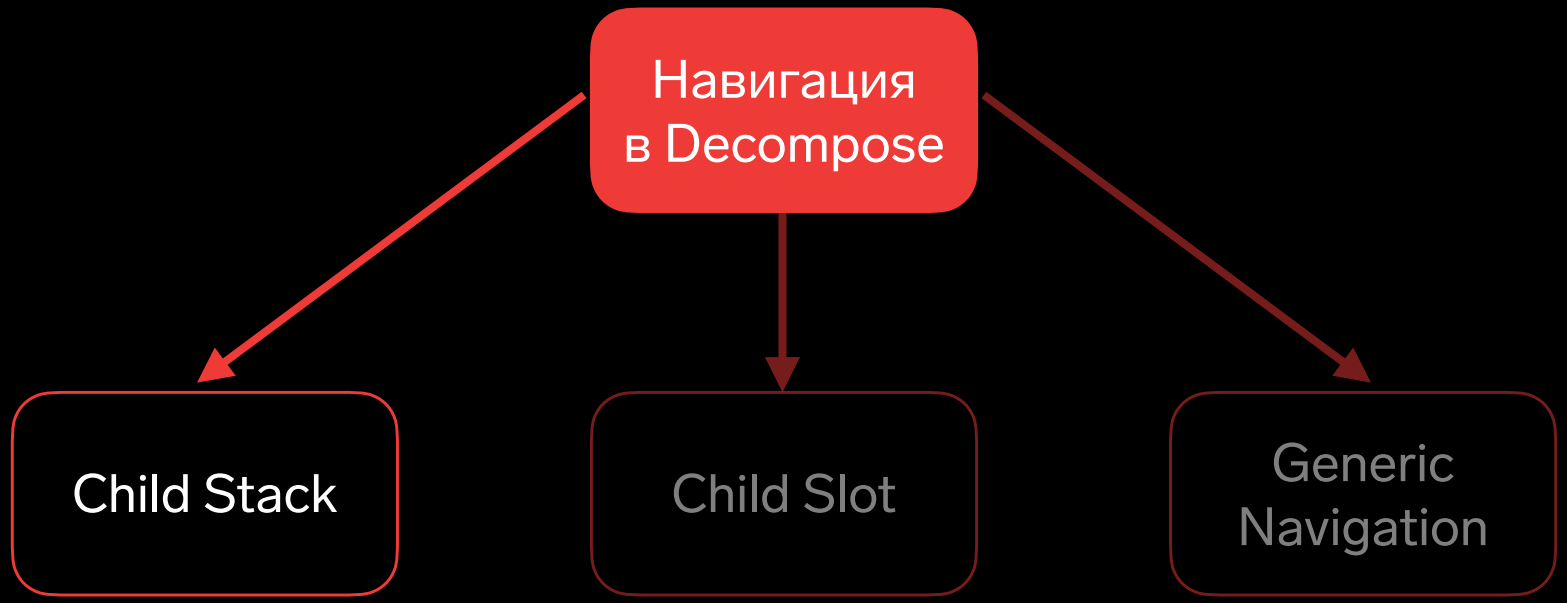

interface Root
Root
Rockets
Launches
Settings
interface Rockets
interface Launches
interface Settings

Пример

```
class RocketsComponent(
    componentContext: ComponentContext,
    private val navigateLaunches: (String) → Unit,
    private val navigateSettings: () → Unit
) : Rockets, ComponentContext by componentContext {
    override fun onLaunchesClick(rocketId: String) {
        navigateLaunches(rocketId)
    }
    override fun onSettingsClick() { navigateSettings() }
}

class LaunchesComponent(
    private val rocketId: String,
    componentContext: ComponentContext
) : Launches, ComponentContext by componentContext
```

Навигация в Decompose


Навигация
в Decompose
Child Stack
Child Slot
Generic
Navigation


Навигация
в Decompose
Child Stack
Child Slot
Generic
Navigation

3 столпа навигации

1

Конфигурация — является ключом компонента и может содержать аргументы

```
sealed interface ScreenConfig : Parcelable {
    @Parcelize
    object Rockets : ScreenConfig

    @Parcelize
    data class Launches(
        val rocketId: String
    ): ScreenConfig
}
```

3 столпа навигации

2

Navigation — интерфейс с командами для навигации

```
interface StackNavigation<C : Any> :
StackNavigator<C>,
StackNavigationSource<C>
```

3 столпа навигации

3

ChildStack содержит активный компонент и бэкстек

```
data class ChildStack<out C : Any, out T : Any>(
    val active: Child.Created<C, T>,
    val backStack: List<Child<C, T>> = emptyList(),
)
```

Пример корневого компонента

```
interface Root {
    val childStack: Value<ChildStack<*, Child>>

    sealed class Child {
        class RocketsChild(val component: Rockets) : Child()
        class LaunchesChild(val component: Launches) : Child()
    }
}
```

Создание Child Stack

```
private val navigation = StackNavigation<ScreenConfig>()

override val childStack: Value<ChildStack<*, Root.Child>> = childStack(
    source = navigation,
    handleBackButton = true,
    initialStack = { listOf(ScreenConfig.Rockets) },
    childFactory = ::child,
)

private fun child(
    config: ScreenConfig,
    componentContext: ComponentContext
): Root.Child {
    return when (config) {
        is ScreenConfig.Rockets -> {...}
        is ScreenConfig.Launches -> {...}
    }
}
```

RootComponent.kt

Создание Child Stack

```
private val navigation = StackNavigation<ScreenConfig>()

override val childStack: Value<ChildStack<*, Root.Child>> = childStack(
    source = navigation,
    handleBackButton = true,
    initialStack = { listOf(ScreenConfig.Rockets) },
    childFactory = ::child,
)

private fun child(
    config: ScreenConfig,
    componentContext: ComponentContext
): Root.Child {
    return when (config) {
        is ScreenConfig.Rockets → {...}
        is ScreenConfig.Launches → {...}
    }
}
```

RootComponent.kt

Создание Child Stack

```
private fun child(
    config: ScreenConfig,
    componentContext: ComponentContext
): Root.Child {
    return when (config) {
        is ScreenConfig.Rockets → {
            Root.Child.RocketsChild(
                RocketsComponent(
                    componentContext = componentContext,
                    navigateLaunches = { rocketId →
                        navigation.push(ScreenConfig.Launches(rocketId))
                    },
                    navigateSettings = {},
                )
            )
        }
        is ScreenConfig.Launches → {
            Root.Child.LaunchesChild(
                LaunchesComponent(config.rocketId, componentContext)
```

RootComponent.kt

Создание Child Stack

```
private val navigation = StackNavigation<ScreenConfig>()

override val childStack: Value<ChildStack<*, Root.Child>> = childStack(
    source = navigation,
    handleBackButton = true,
    initialStack = { listOf(ScreenConfig.Rockets) },
    childFactory = ::child,
)

private fun child(
    config: ScreenConfig,
    componentContext: ComponentContext
): Root.Child {
    return when (config) {
        is ScreenConfig.Rockets -> {...}
        is ScreenConfig.Launches -> {...}
    }
}
```

RootComponent.kt

Встроенный observable тип

```
abstract class Value<out T : Any> {

    abstract val value: T

    abstract fun subscribe(observer: (T) → Unit)

    abstract fun unsubscribe(observer: (T) → Unit)
}
```

Интеграция с Jetpack Compose

```
@Composable
fun RootScreen(root: Root) {
    val childStack by root.childStack.subscribeAsState()

    Children(stack = childStack) {
        when (val child = it.instance) {
            is Root.Child.RocketsChild →
                RocketsScreen(component = child.component)
            is Root.Child.LaunchesChild →
                LaunchesScreen(component = child.component)
        }
    }
}
```

Интеграция с Jetpack Compose

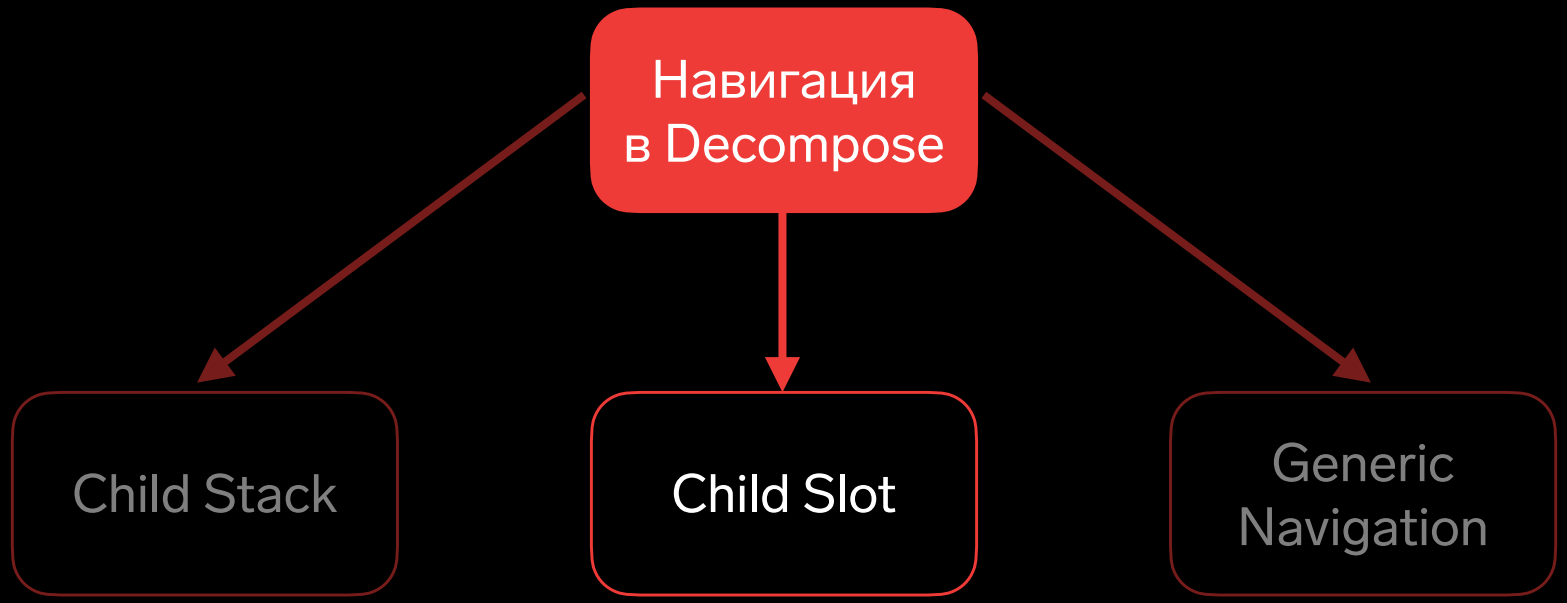
```
@Composable
fun RootScreen(root: Root) {
    val childStack by root.childStack.subscribeAsState()

    Children(
        stack = childStack,
        animation = stackAnimation { from, to, direction →
            if (direction.isFront) {
                slide() + fade()
            } else {
                scale(frontFactor = 1F, backFactor = 0.7F) + fade()
            }
        }
    ) {…}
}
```

Интеграция со SwiftUI

```
public class ObservableValue<T : AnyObject> : ObservableObject {
    private let observableValue: Value<T>

    @Published
    var value: T

    private var observer: ((T) -> Void)?

    init(_ value: Value<T>) {
        observableValue = value
        self.value = observableValue.value
        observer = { [weak self] value in self?.value = value }
        observableValue.subscribe(observer: observer!)
    }

    deinit {
        observableValue.unsubscribe(observer: self.observer!)
    }
}
```

Интеграция со SwiftUI

```
struct RootView: View {
    private let root: Root

    @ObservedObject
    private var childStack: ObservableValue<ChildStack<AnyObject, RootChild>>

    private var activeChild: RootChild { childStack.value.active.instance }

    init(_ root: Root) {
        self.root = root
        childStack = ObservableValue(root.childStack)
    }

    var body: some View {
        ChildView(child: activeChild)
    }
}
```


Навигация
в Decompose
Child Stack
Child Slot
Generic
Navigation


Child
Slot
Child
Overlay

Child Slot

- Содержит один или ноль активных компонентов
- Подходит для модального отображения информации
- А также для срытия/отображения частей экрана

```
data class ChildSlot<out C : Any, out T : Any>(
    val child: Child.Created<C, T>? = null,
)
```

Пример с ChildSlot

```
private val slotNavigation = SlotNavigation<SlotConfig>()

override val childSlot: Value<ChildSlot<*, Root.SlotChild>>
    = childSlot(
        source = slotNavigation,
        handleBackButton = true,
        childFactory = ::child
    )

private fun child(
    config: SlotConfig,
    componentContext: ComponentContext
): Root.SlotChild {
    return when (config) {
        is SlotConfig.Settings -> {...}
    }
}
```

Пример корневого компонента

```
interface Root {
    val childStack: Value<ChildStack<*, Child>>

    val childSlot: Value<ChildSlot<*, SlotChild>>

    fun dismissSlotChild()

    sealed class SlotChild {
        class SettingsChild(val component: Settings) : SlotChild()
    }

    sealed class Child {
        class RocketsChild(val component: Rockets) : Child()
        class LaunchesChild(val component: Launches) : Child()
    }
}
```

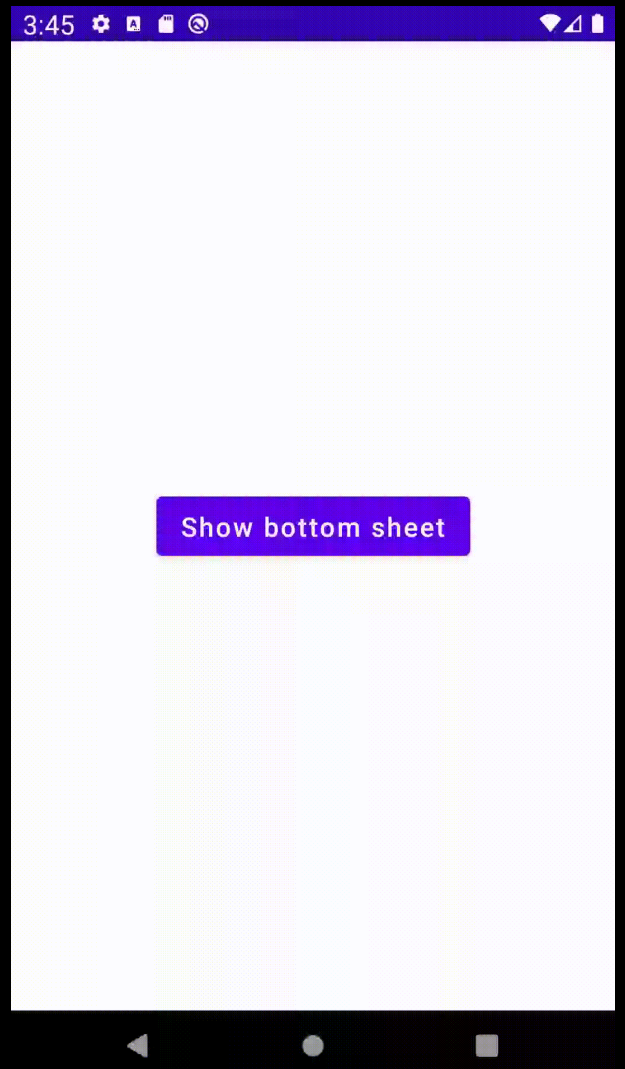
Bottom Sheet в SwiftUI

```
extension View {
    @ViewBuilder
    func sheet<T, Content>(
        item: T?,
        onDismiss: @escaping () -> Void,
        @ViewBuilder content: @escaping (T) -> Content
    ) -> some View where Content: View {
        sheet(
            isPresented: Binding(get: { item != nil }, set: {_,_ in }),
            onDismiss: onDismiss,
            content: { content(item!) }
        )
    }
}
```

Bottom Sheet в SwiftUI

```
struct RootView: View {
    ...
    var body: some View {
        ChildView()
        .sheet(
            item: childOverlay.value.child?.instance,
            onDismiss: { root.dismissOverlay() },
            content: { child in
              ...
            }
        )
    }
}
```

Bottom Sheet в Jetpack Compose



Декларативный Bottom Sheet

```
@Composable
fun <C : Any, T : Any> rememberSlotModalBottomSheetState(
    slot: Value<ChildSlot<C, T>>,
    onDismiss: () → Unit,
    skipHalfExpanded: Boolean = false,
    sheetContent: @Composable (child: Child.Created<C, T>) → Unit,
): SlotModalBottomSheetState {
    ...
}
```

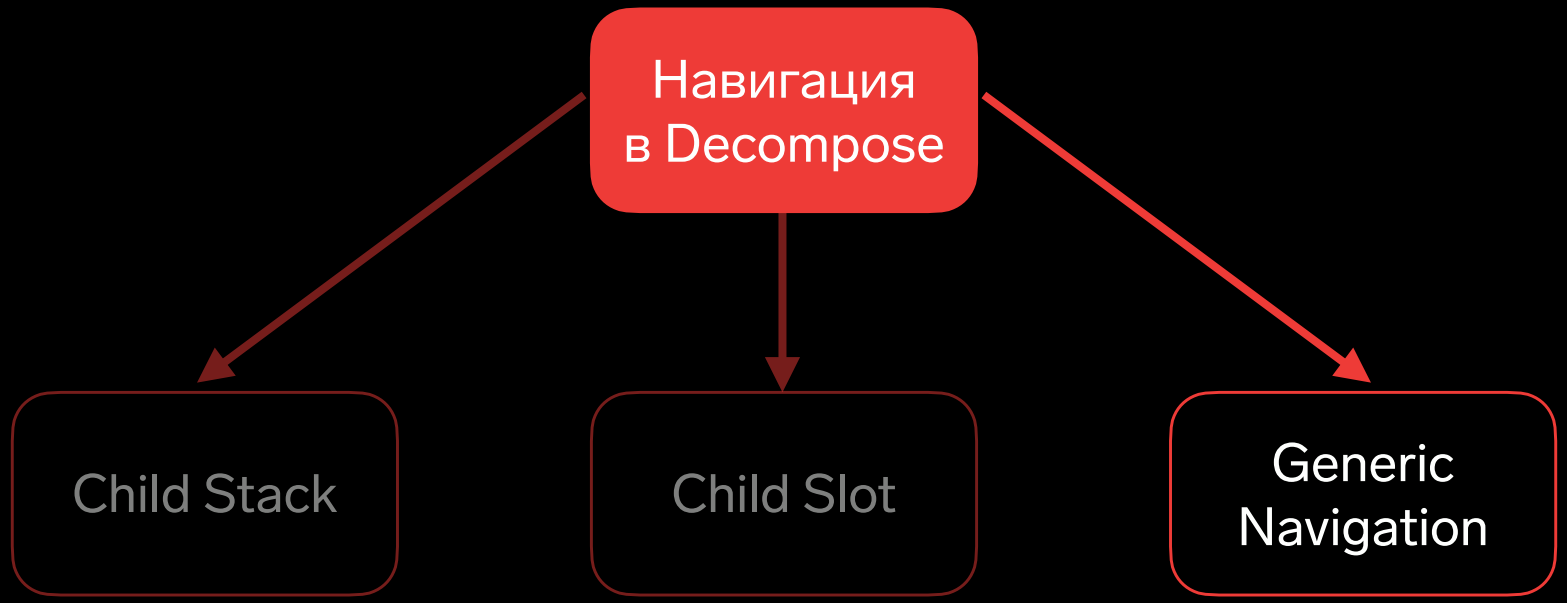
https://github.com/arkivanov/Decompose/issues/261

Bottom Sheet в Jetpack Compose

```
val sheetState = rememberSlotModalBottomSheetState(
    overlay = root.childSlot,
    onDismiss = root::dismissSlotChild,
    skipHalfExpanded = true
) { child →
    when (val instance = child.instance) {
        is Root.SlotChild.SettingsChild → {
            SettingsModalScreen(component = instance.component)
        }
    }
}
```

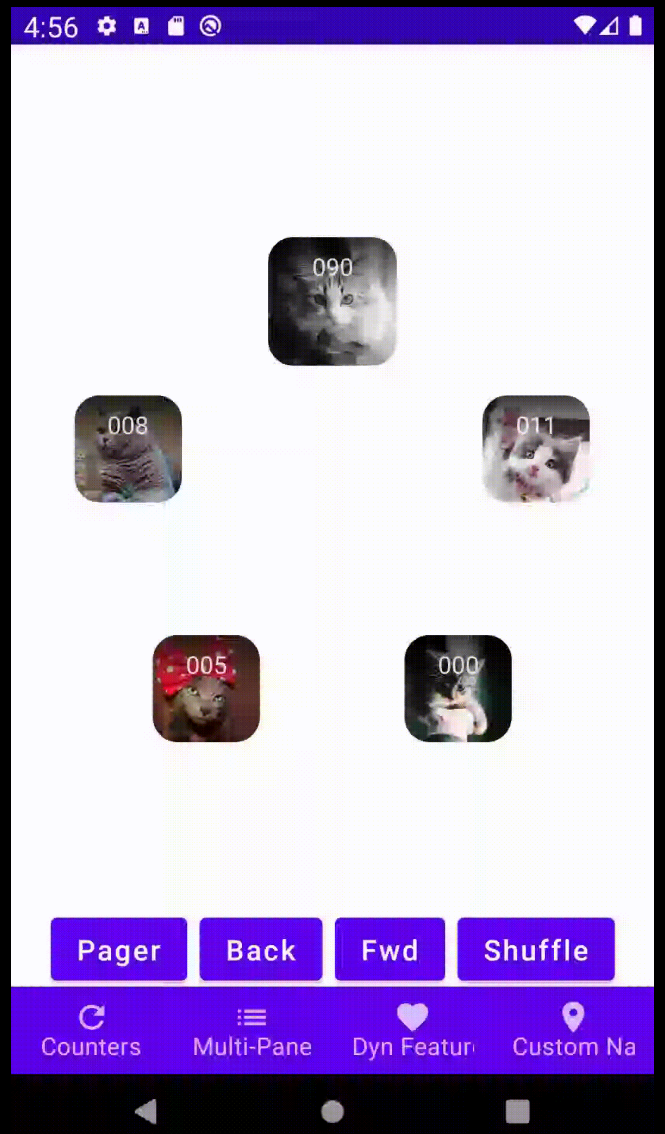
Навигируемся

```
private fun child(
  config: ScreenConfig,
  componentContext: ComponentContext
): Root.Child {
   return when (config) {
       is ScreenConfig.Rockets -> {
           Root.Child.Rockets(
               component = RocketsComponent(componentContext),
               navigateLaunches = { rocketId ->
                   navigation.push(ScreenConfig.Launches(rocketId))
               },
               navigateSettings = {
                   slotNavigation.activate(SlotConfig.Settings)
               }
           )
       }
       ...
```


Навигация
в Decompose
Child Stack
Child Slot
Generic
Navigation

Generic Navigation

https://github.com/arkivanov/Decompose



Преимущества Decompose

Поддержка Parcelable

```
@Parcelize
data class ChequesConfig(
    val chequeTotalSum: Double,
    val chequeCountInfo: ChequeCountInfo,
    val cheques: List<ChequeInfo>
) : SummaryConfig
```

Поддержка ViewModel

```
class RootComponent(
    componentContext: ComponentContext
) : Root, ComponentContext by componentContext {
    private val viewModel = instanceKeeper.getOrCreate {
        RootViewModel()
    }

    private class RootViewModel : InstanceKeeper.Instance {
        override fun onDestroy() {
            // Аналог onCleared во ViewModel
        }
    }
  ...
}
```

Вложенная навигация в Decompose

- Нет различий от глобальной навигации
- Можно делать любую вложенность
- Можно привязать ViewModel к любому компоненту

Сложная навигация

- Deeplink и начальный стек экранов

```
private val navigation = StackNavigation<Config>()

private val stack = childStack(
    source = navigation,
    initialStack = { getInitialStack(deepLink) },
    childFactory = ::child,
)
```

Сложная навигация

- Deeplink и начальный стек экранов
- Полный контроль над стеком навигации

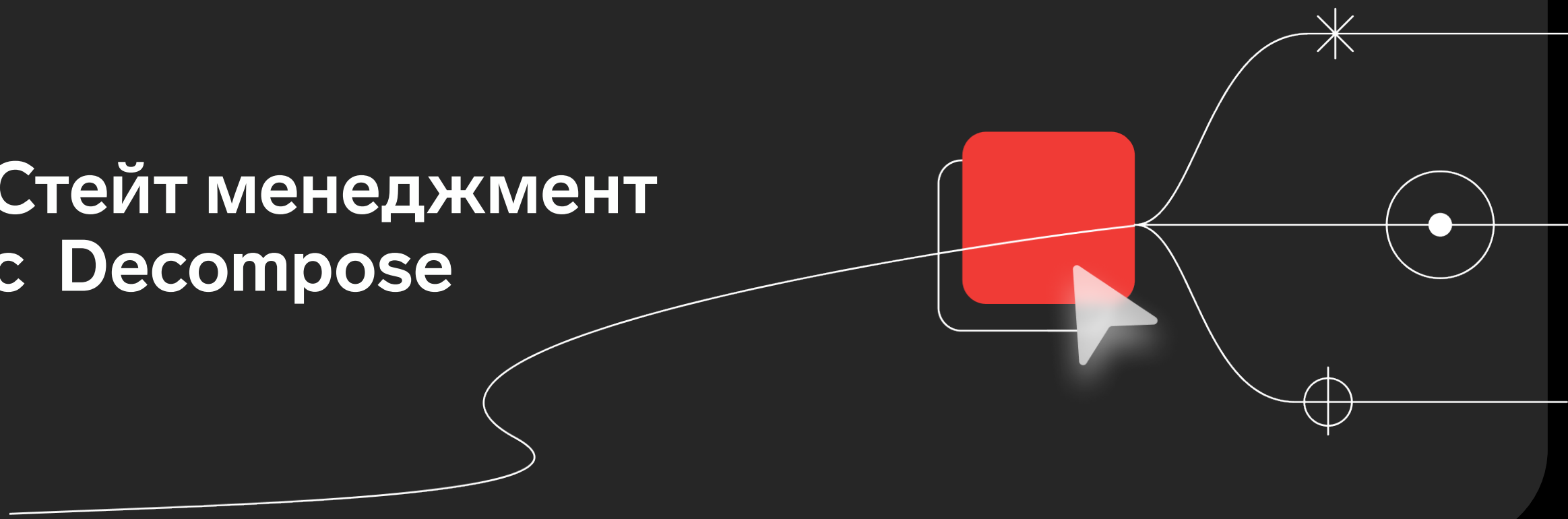
```
navigation.navigate { currentStack →
    currentStack + listOf(
        ScreenConfig.Launches("1"),
        ScreenConfig.Launches("2")
    )
}
```

Сложная навигация

- Deeplink и начальный стек экранов
- Полный контроль над стеком навигации
- Мультистек в одну строку

```
navigation.bringToFront(Config.Profile)
```

Стейт менеджмент с Decompose



Decompose никак не ограничивает выбор паттерна для управления состоянием

Но лучше, если это будет UDF

Пример компонента

```
interface Rockets {
    val state: AnyStateFlow<RocketsUiState>

    fun onLaunchesClick(rocketId: String)

    fun onSettingsClick()
}
```

Пример компонента

```
interface Rockets {
    val state: AnyStateFlow<RocketsUiState>

    fun onLaunchesClick(rocketId: String)

    fun onSettingsClick()
}
```

https://habr.com/ru/post/596497/

Пример компонента

```
class RocketsComponent(
    componentContext: ComponentContext,
    private val navigateLaunches: (String) -> Unit,
    private val navigateSettings: () -> Unit,
) : Rockets, ComponentContext by componentContext {

    private val feature = instanceKeeper.getOrCreate {
        RocketsFeature()
    }

    override val state: AnyStateFlow<RocketsUiState>
        get() = feature.uiState.wrapToAny()
    ...
}
```

MVI и State Machine — визуализация и анализ диаграммы состояний с помощью VisualFSM

Василий Рылов

Контур

Mobius
2022 Spring

Недекларативные сюрпризы от Apple

1. Двухсторонний Binding

- Некоторые вьюшки требуют передачу биндинга, а не значения

```
Toggle(
    "Сравнить с прошлой неделей",
    isOn: Binding(
        get: { state.compareWithPreviousWeek },
        set: { _, _ in component.toggleCompareWithPreviousWeek() }
    )
)
```

2. TabView

- Автоматически переключает вьюшки по нажатию на таб

```
TabView {
    SummaryView()
        .tabItem {
            Label("Summary", systemImage: "")
        }
    ProfileView()
        .tabItem {
            Label("Profile", systemImage: "")
        }
}
```

https://github.com/arkivanov/Decompose/discussions/212

3. Pull-to-refresh

- Требует явно дожидаться завершения загрузки данных через механизм async await
- Нельзя показать без действия пользователя

```
extension View {
    public func refreshable(
        action: @escaping @Sendable () async -> Void
    ) -> some View
}
```

4. Анимации через State

- Лишаемся возможности анимировать вьюшку через State

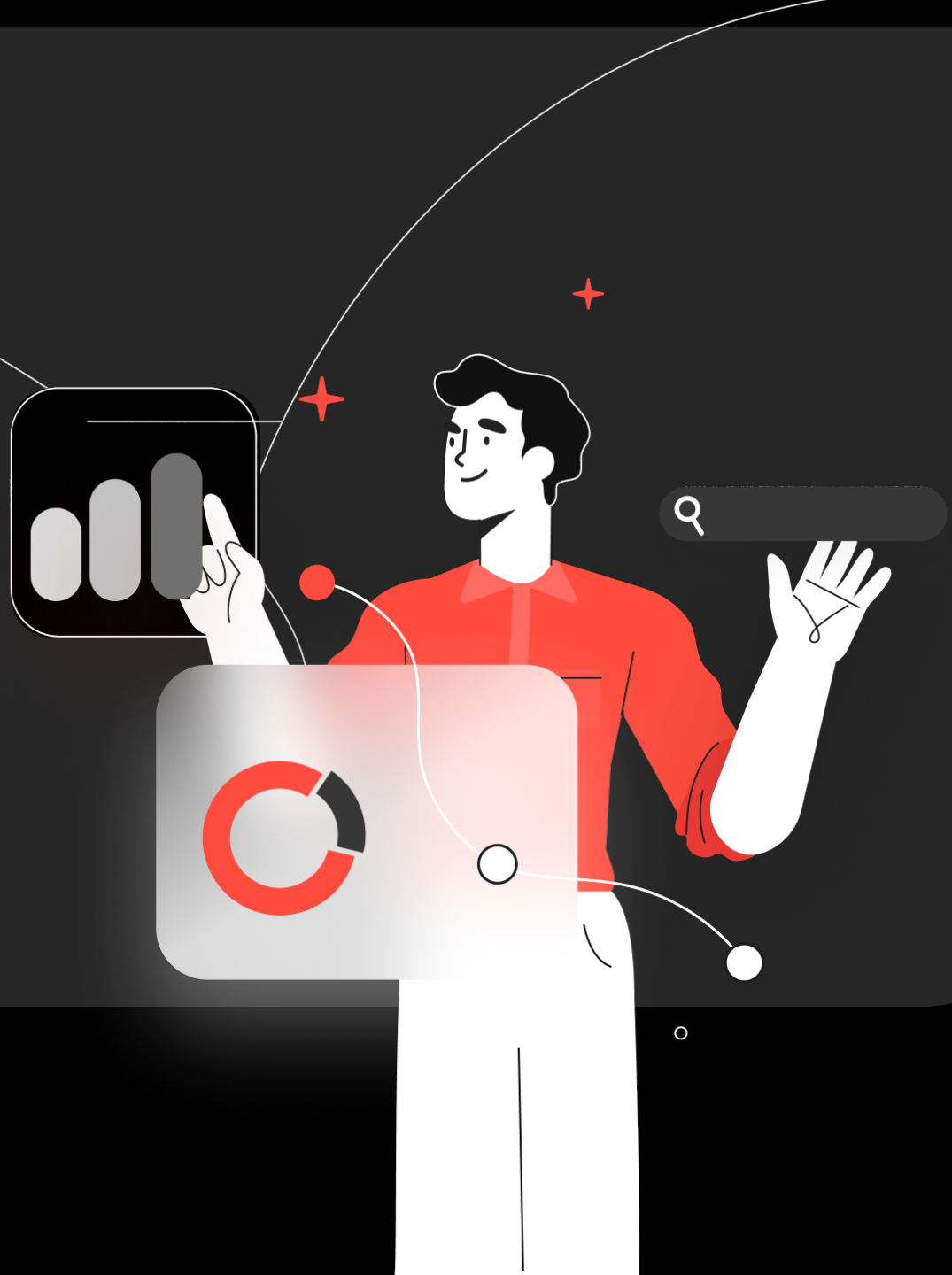
```
@State private var showDetail = false

var body: some View {
    VStack {
        Button("Show details") {
            withAnimation {
                showDetail.toggle()
            }
        }

        if showDetail {
            Text("Details")
        }
    }
}
```

5. Встроенная навигация

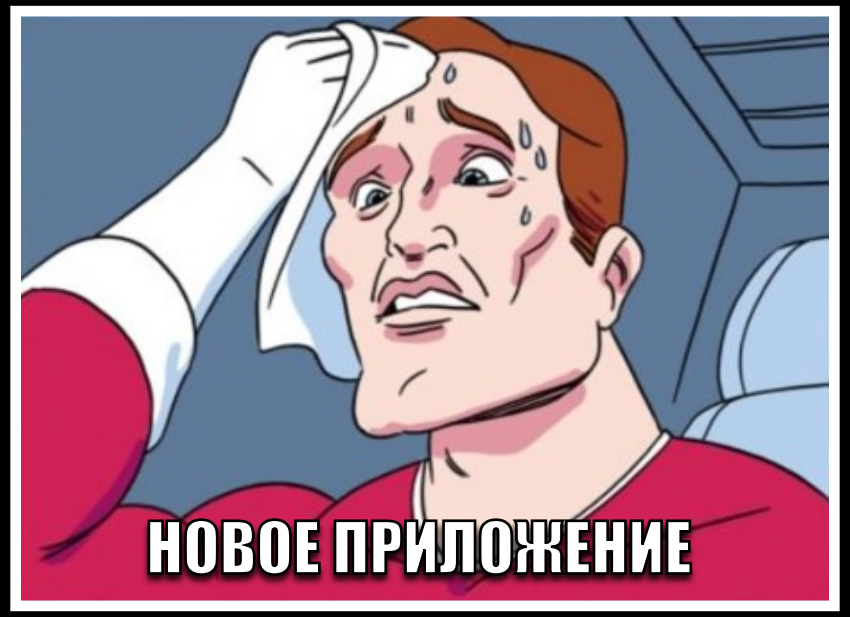
- SwiftUI сильно завязан на стандартную навигацию
- Toolbar можно показать только внутри NavigationView
- NavigationStack поможет вернуть жесты и анимации переходов

```
NavigationView {
    List {
        NavigationLink("Purple", destination: ColorDetail(color: .purple))
        NavigationLink("Pink", destination: ColorDetail(color: .pink))
    }
    .navigationTitle("Colors")
}
```

Работаем с Legacy




Фрагменты
Compose


НОВОЕ ПРИЛОЖЕНИЕ

Императивная глобальная навигация

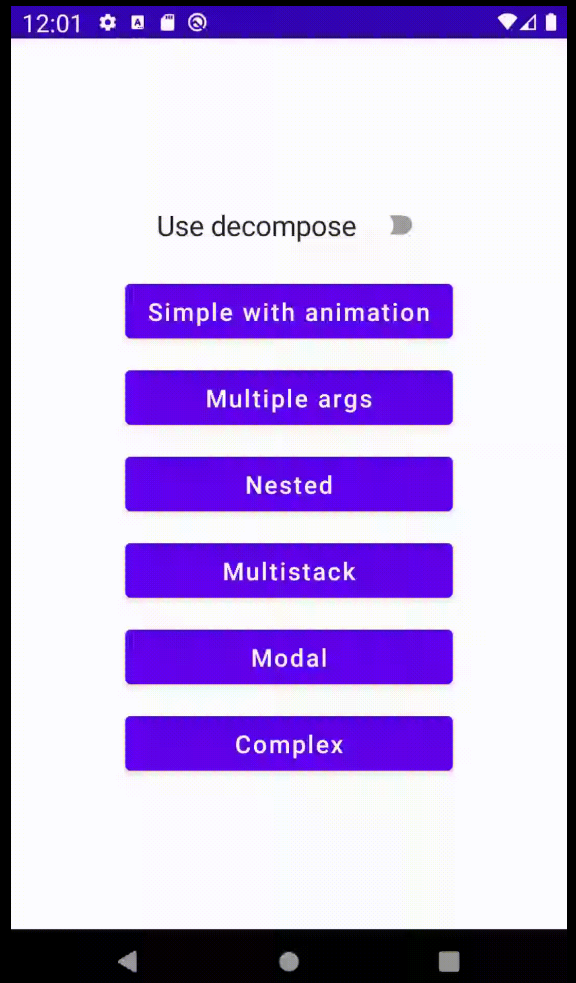
- \+ Облегчает взаимодействие со старыми модулями
- − UI знает как создавать каждый компонент
- − Внедрение зависимостей происходит в UI
- − Состояния бизнес логики и навигации могут разойтись

Постепенная миграция на Decompose

1. Переписываем верстку экрана на Compose
2. Адаптируем бизнес логику, избавляясь от платформенных зависимостей
3. Создаем корневой компонент во фрагменте
4. Избавляемся от фрагментов

Decompose + Jetpack Navigation

https://github.com/AJIEKCX/navigation-talk



Декларативная глобальная навигация

- − Сложно интегрироваться с императивными фичами
- \+ UI создает только рутовый компонент
- \+ Внедрение зависимостей происходит в бизнес логике
- \+ Один источник правды о состоянии навигации

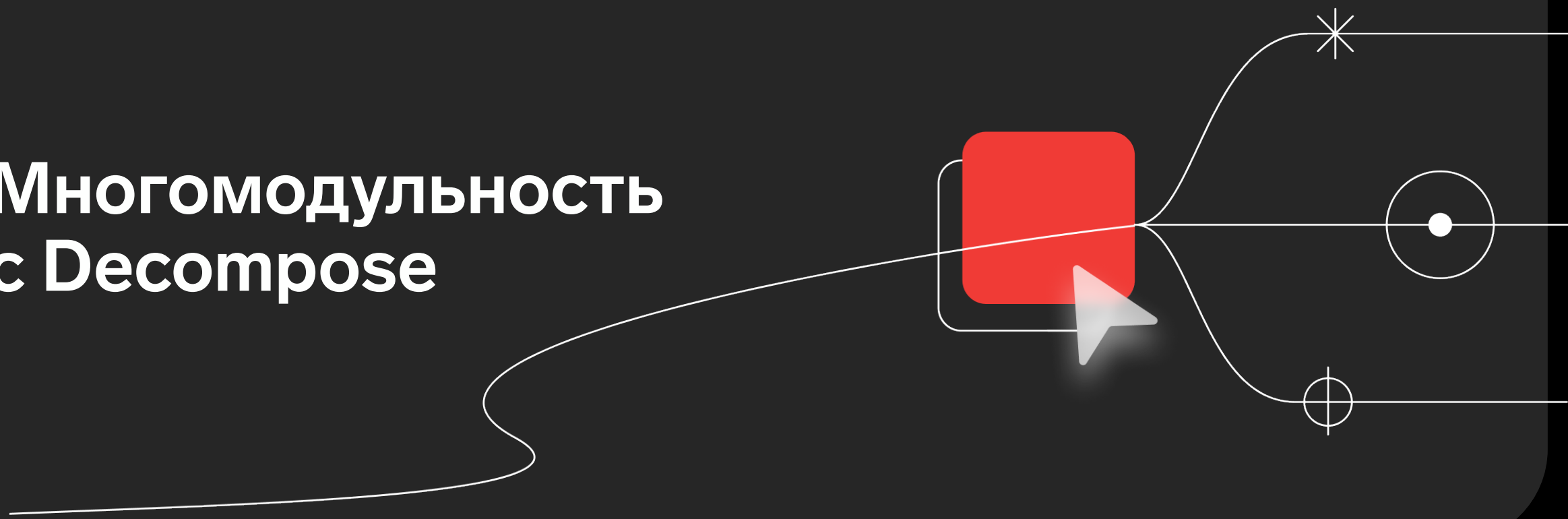
Интеграция с legacy фичами в iOS

```
struct ChatView: UIViewControllerRepresentable {
    func makeUIViewController(context: Context) -> some UIViewController {
      ...
    }

    func updateUIViewController(
        _ uiViewController: UIViewControllerType,
        context: Context
    ) {
      ...
    }
}
```

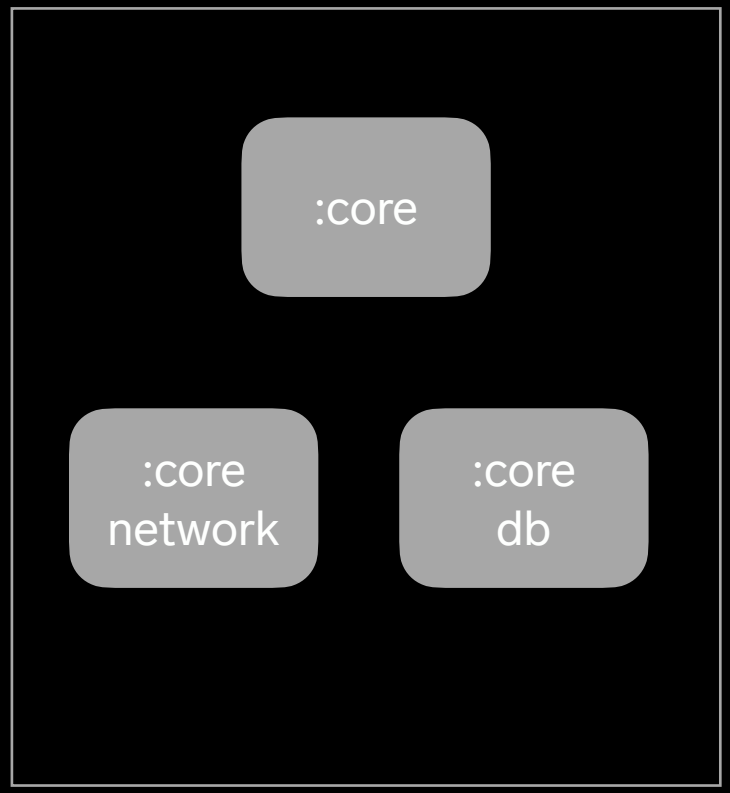
Интеграция с legacy фичами в Android

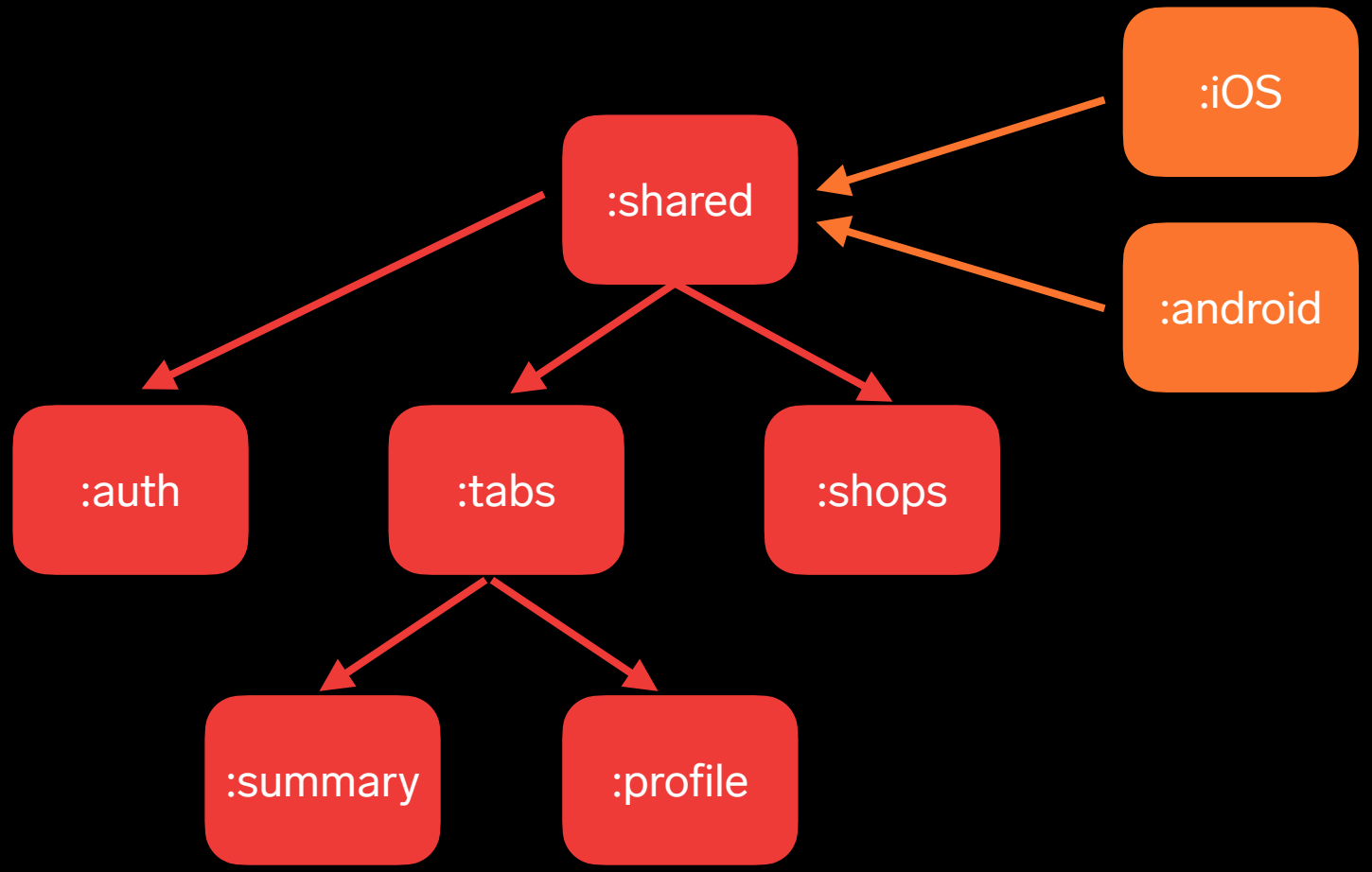
```
class ChatActivity : FragmentActivity(R.layout.activity_chat)

<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:name="ru.kontur.chat.ChatFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Многомодульность с Decompose



Многомодульность с Decompose



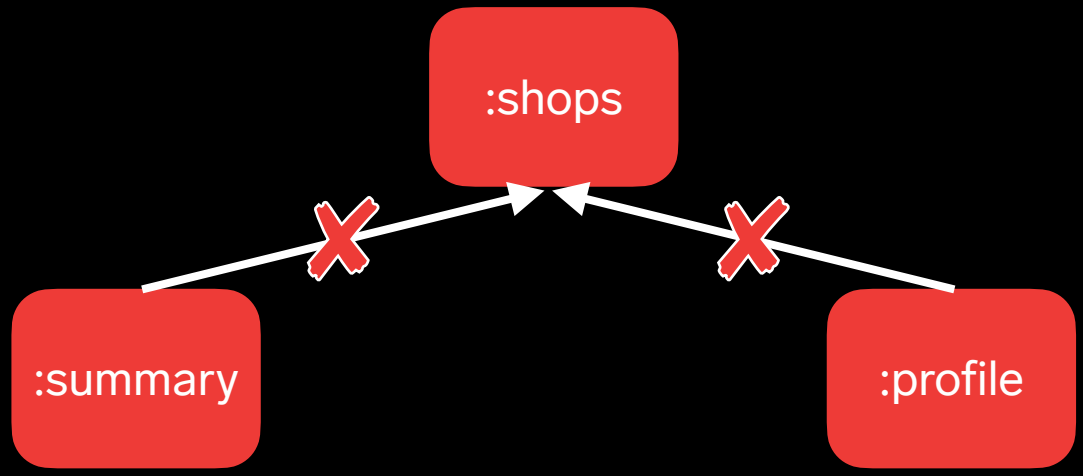


Ограничения в KMM

- Лучше развертывать только один модуль в iOS framework
- Зависимости должны быть объявлены как api в iosMain
- Зависимости должны быть экспортированы в framework

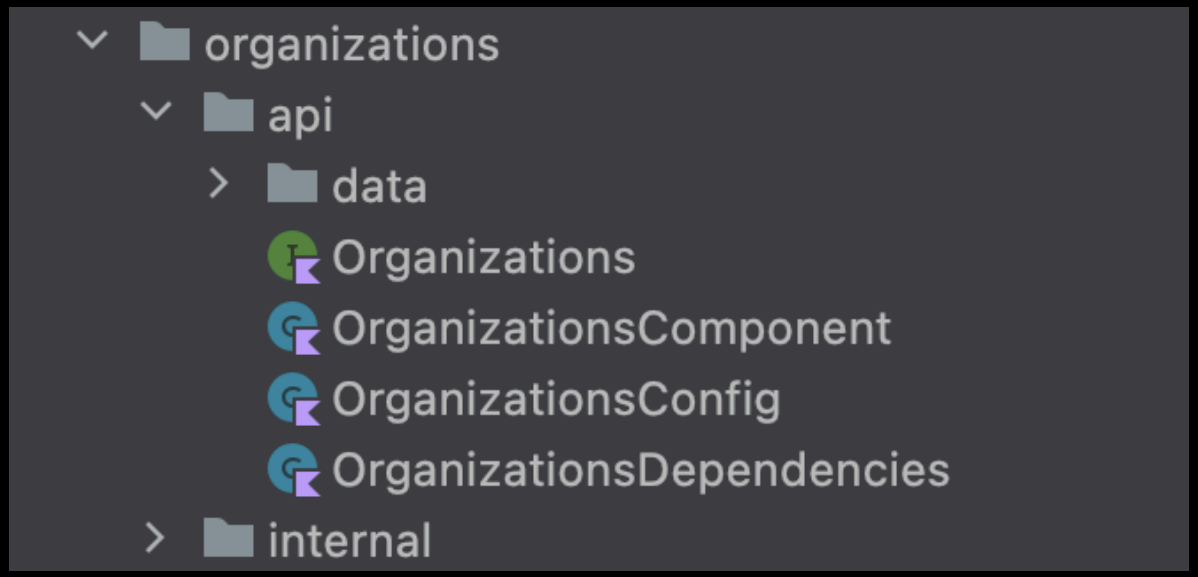
Зависимость фичей друг на друга

- Родительский компонент знает о потомках, но не наоборот
- Большие фичи не должны зависеть друг от друга
- Взаимодействие между фичами происходит через общий родительский компонент



Структура фича модуля

- Деление модуля на api и internal пакеты
- Явное определение зависимостей



Подводим итоги

Декларативная архитектура

- Способствует лучшему разделению логики приложения

Декларативная архитектура

- Способствует лучшему разделению логики приложения
- Определяет единый источник правды о состоянии навигации

Декларативная архитектура

- Способствует лучшему разделению логики приложения
- Определяет единый источник правды о состоянии навигации
- Следует принципу единой ответственности

Декларативная архитектура

- Способствует лучшему разделению логики приложения
- Определяет единый источник правды о состоянии навигации
- Следует принципу единой ответственности
- Позволяет достичь максимального покрытия тестами

Decompose

Качество

Стабильная библиотека
с хорошей документацией
и примерами

Decompose

Качество

Стабильная библиотека
с хорошей документацией
и примерами

Простота

Имеет четкие границы применения,
не навязывает свою архитектуру

Decompose

Качество

Стабильная библиотека
с хорошей документацией
и примерами

Скорость

Значительно ускоряет разработку
за счет максимального
переиспользования кода

Простота

Имеет четкие границы применения,
не навязывает свою архитектуру

Decompose

Качество

Стабильная библиотека
с хорошей документацией
и примерами

Простота

Имеет четкие границы применения,
не навязывает свою архитектуру

Скорость

Значительно ускоряет разработку
за счет максимального
переиспользования кода

Гибкость

Позволяет сделать навигацию
любой сложности

Пример

https://github.com/AJIEKCX/SpaceXRockets

Спасибо за внимание!
Вопросы?

Алексей Панов
Ведущий-инженер программист

Контур

ajiekcx