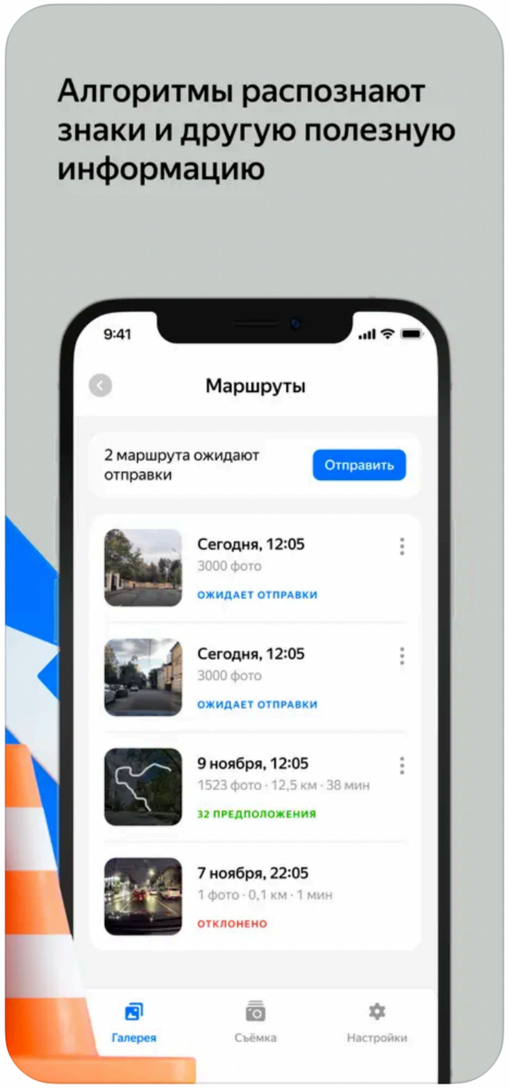
Яндекс

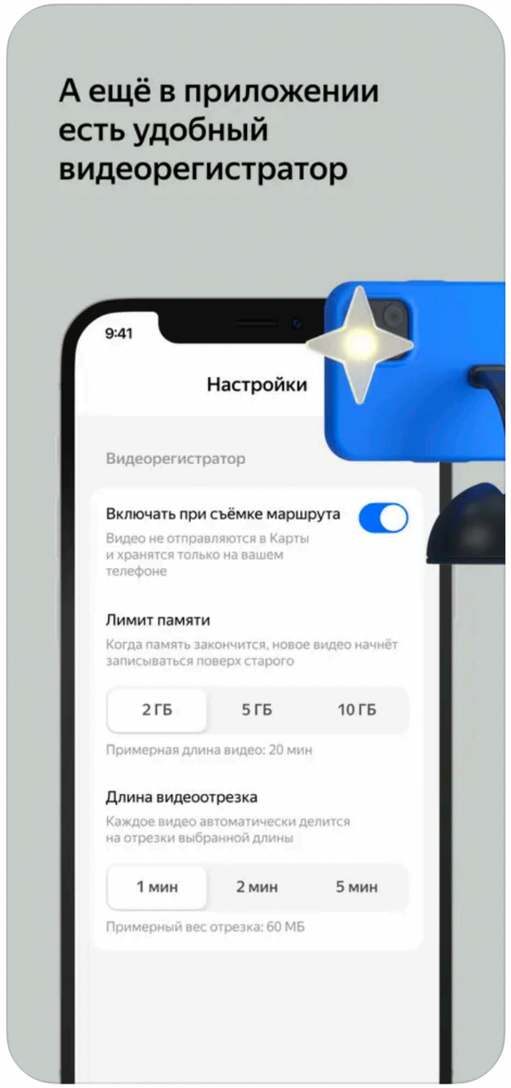
Мультиплатформенный Redux и SwiftUI / Jetpack Compose

Юрий Потапов

Команда разработки мобильных Яндекс.Карт

Народный Картограф





Цели

- Ускорить разработку приложений iOS и Android
- Вынести всю бизнес логику в мультиплатформу
- Минимизировать платформенный код
- Стандартизировать написание UI компонентов для двух платформ
- Проверить готовность SwiftUI и Jetpack Compose к использованию в проектах

Декларативный UI
Что это и зачем?

Что такое декларативное программирование?

〉 Декларативное программирование — это парадигма, противопоставляемая императивной. Описываем не «как», а «что» мы хотим получить

〉 Декларативный стиль более высокоуровневый, чем императивный и позволяет писать гораздо меньше кода в среднем

Императивный код
(как?)

```
result = 0
for element in array {
    result += element
}
```

Декларативный код
(что?)

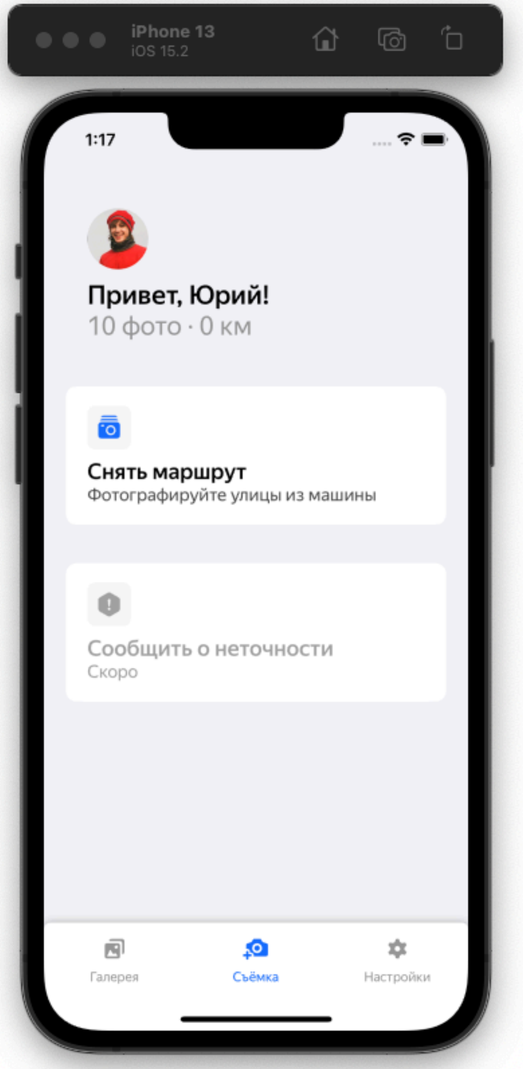
```
result = array.reduce(0, +)
```

Что такое декларативный UI?

Декларативные UI Фреймворки имеют следующий принцип работы:

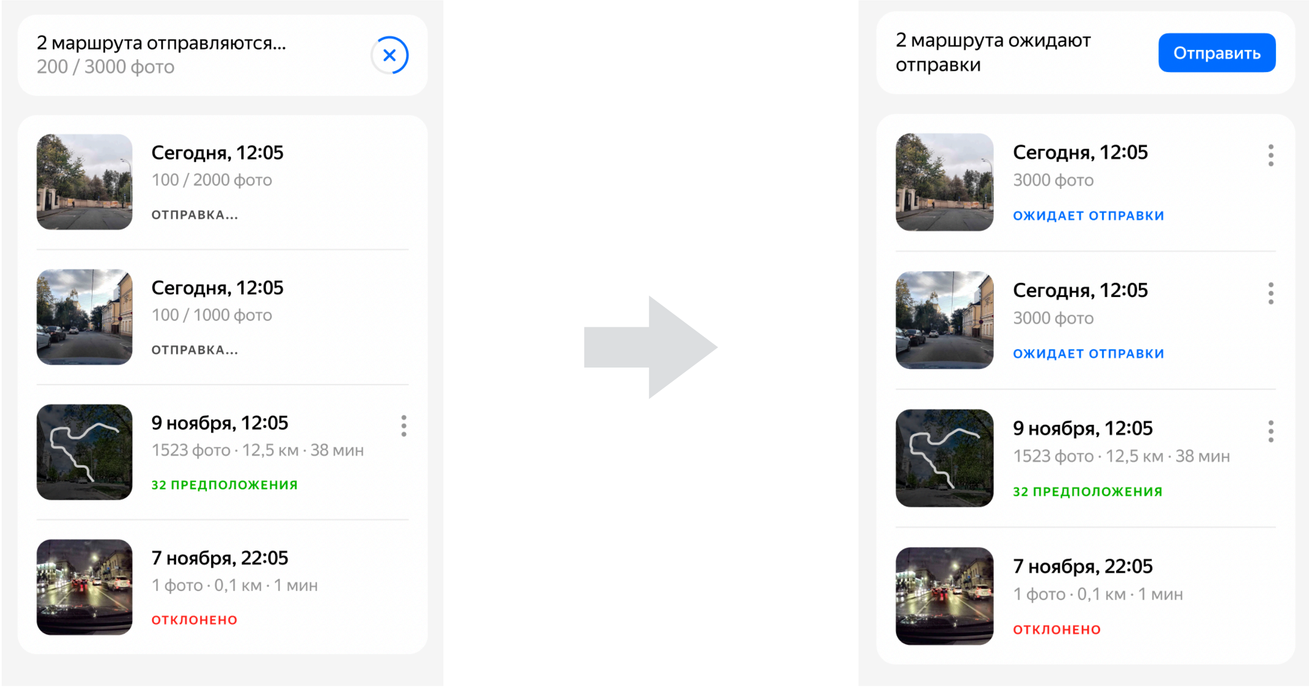
Программист описывает структуру в виде дерева.

Элементы дерева (View) могут быть примитивными (Text, Image) или являться композицией других элементов



Что такое декларативный UI?

Внешний вид этих View может зависеть от каких-то данных, получаемых снаружи.
А точнее, от потока этих данных.
Новые данные порождают необходимость перерисовать то, что есть на экране.



Зачем декларативный UI?

〉 Позволяет избежать расхождения в синхронизации данных модели и того, что отображено на экране

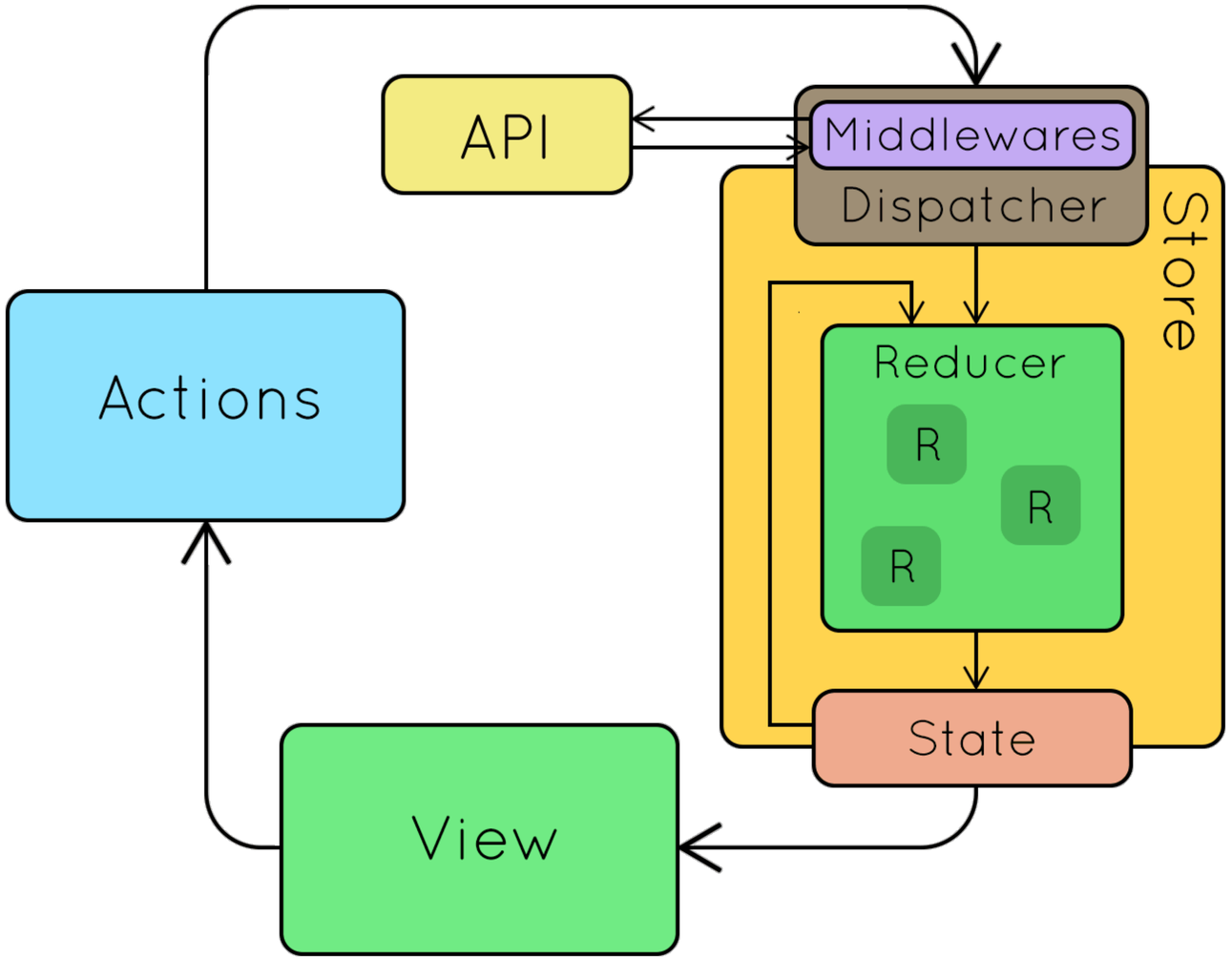
〉 Хорошо читаем

〉 Apple и Google разрабатывают собственные фреймворки декларативного UI - SwiftUI и Jetpack Compose и вкладывают в их развитие много ресурсов

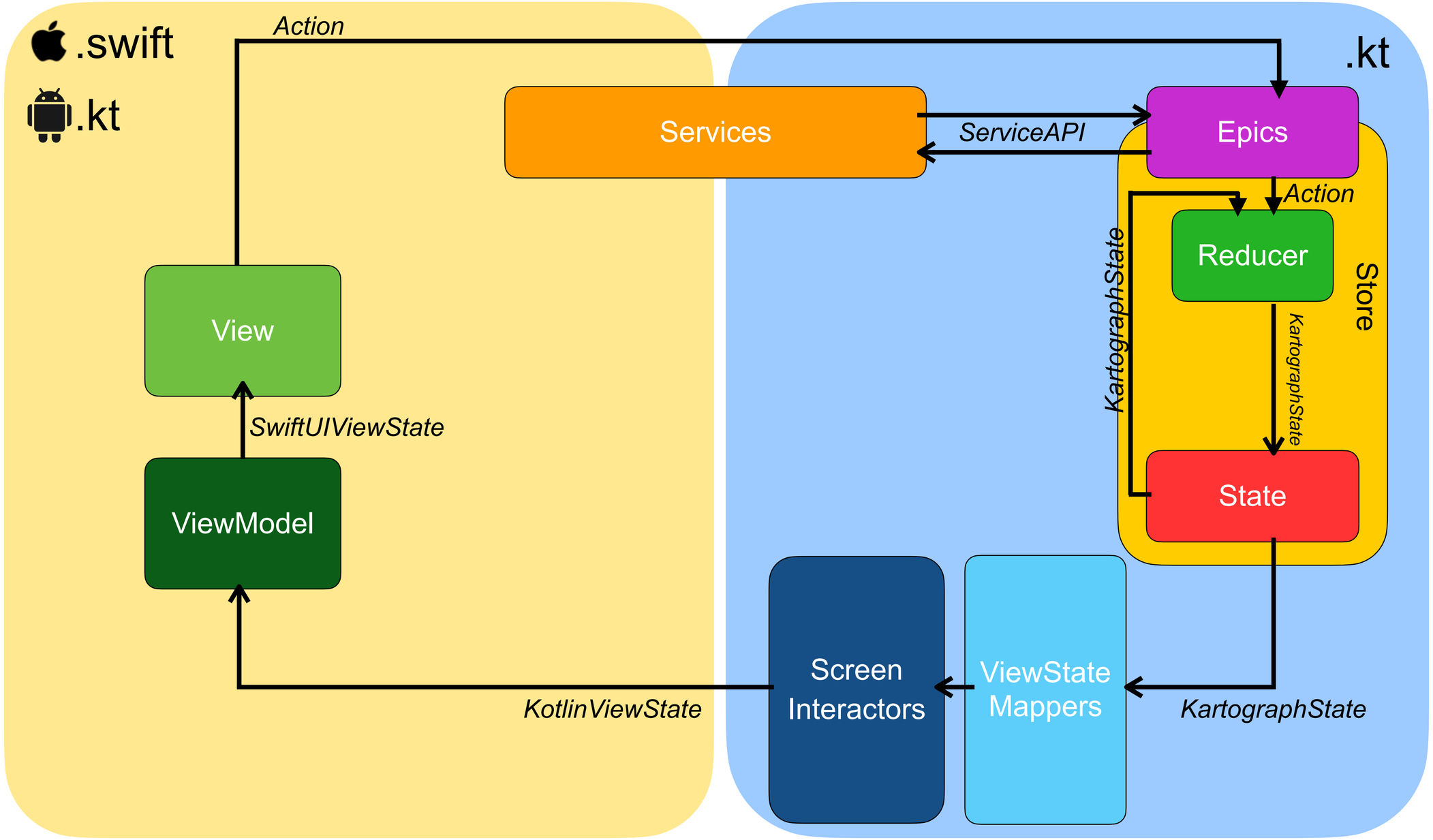
〉 SwiftUI и Jetpack Compose настолько похожи и по философии, и по форме, что разработчики могут стать еще более универсальными и “мультиплатформенными”

Как всё устроено

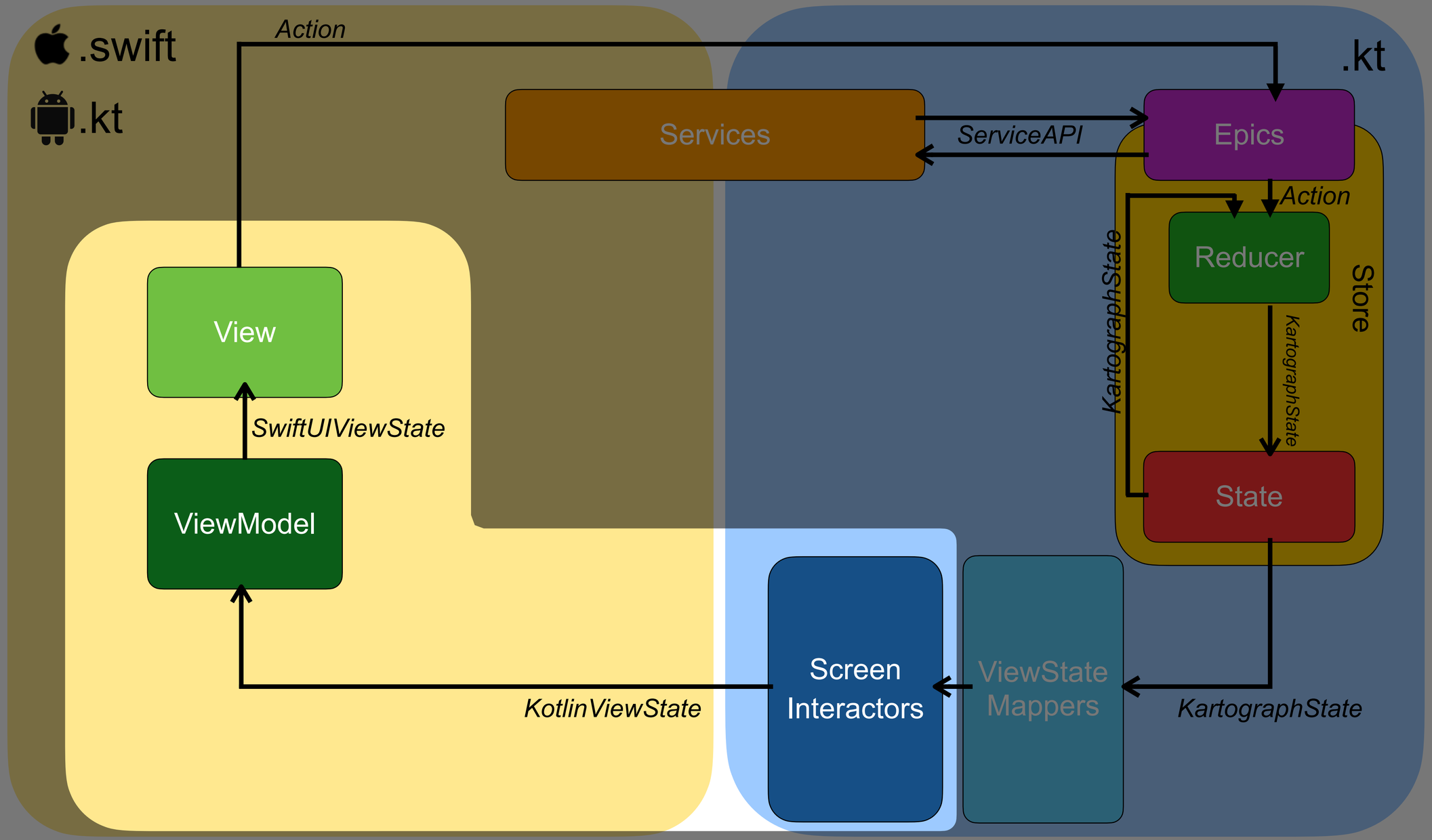
Redux


API
Middlewares
Dispatcher
Store
Actions
Reducer
R
R
R
State
View

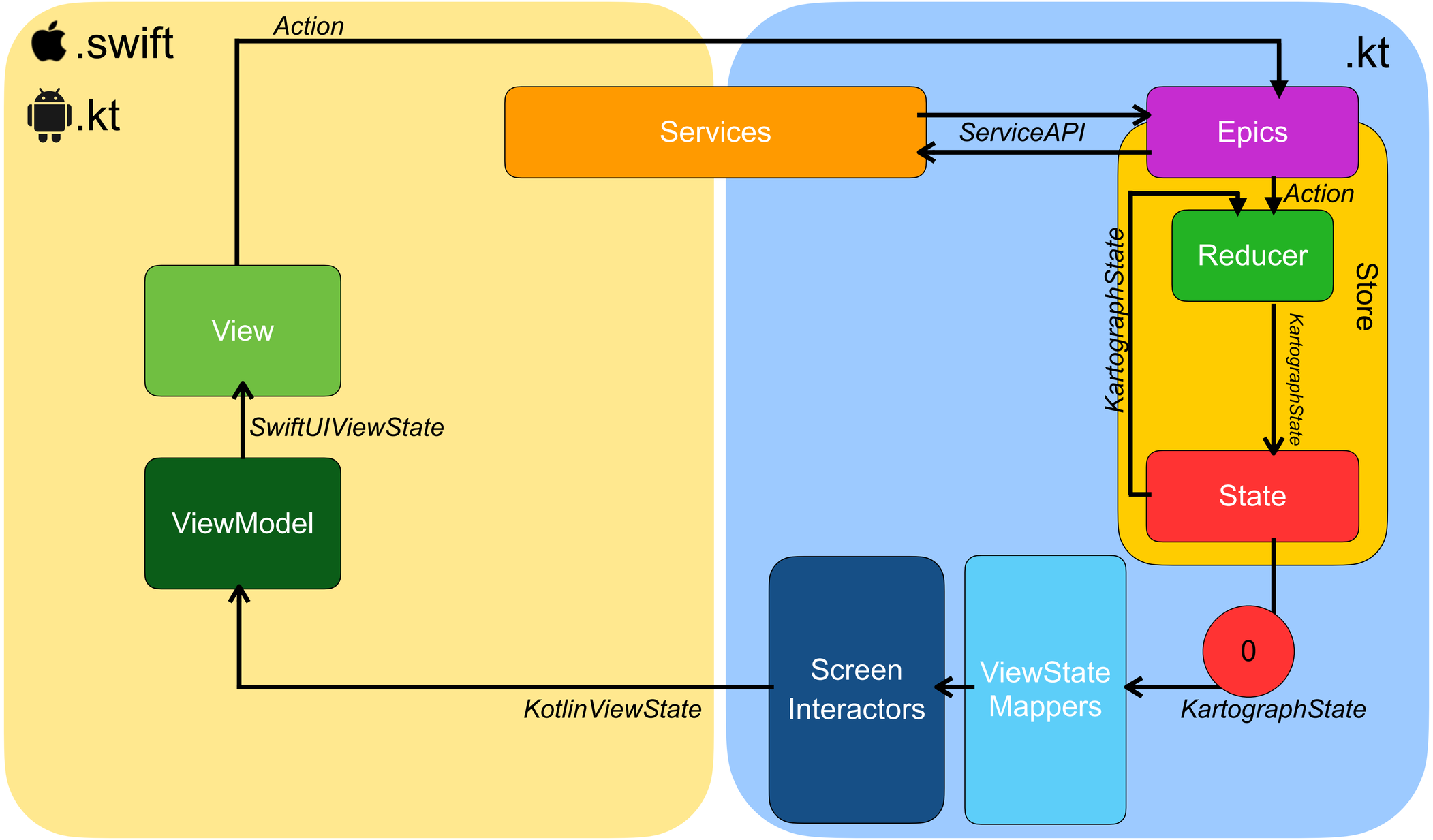
Kartograph Redux



Kartograph Redux



Kartograph Redux



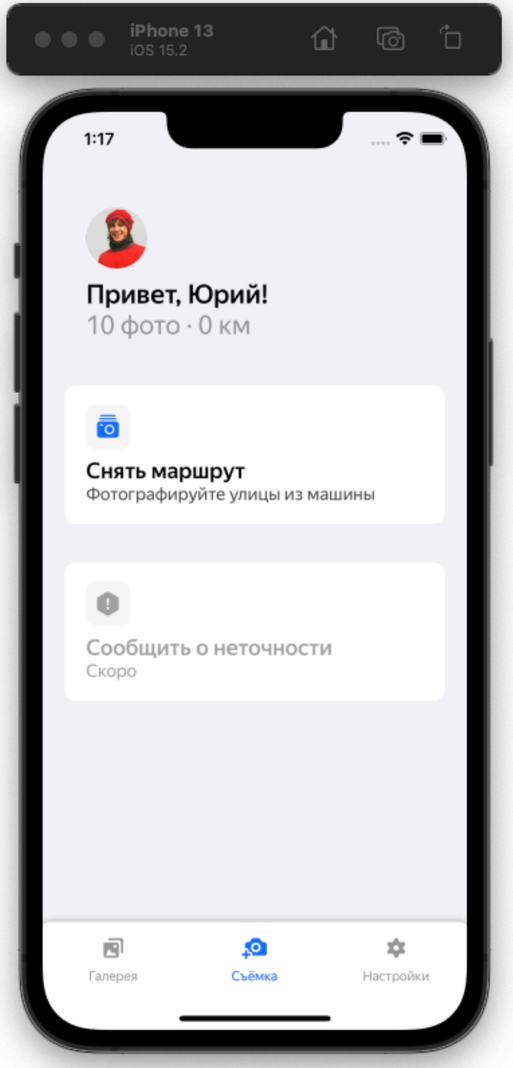
ViewState mappers

KartographState

```
internal data class KartographState(
    val navigationState: NavigationState,
    val authState: KartographAuthState,
    val captureState: CaptureState,
    val settingsState: SettingsState,
    val ridesState: RidesState,
    ...
)
```

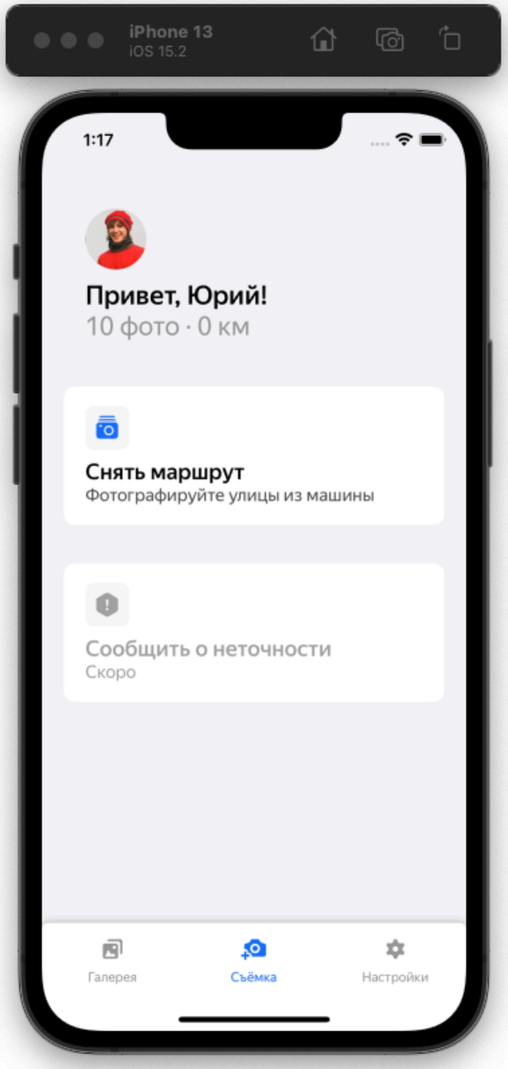
KartographState

```
internal data class KartographState(
    val navigationState: NavigationState,
    val authState: KartographAuthState,
    val captureState: CaptureState,
    val settingsState: SettingsState,
    val ridesState: RidesState,
    ...
)
```

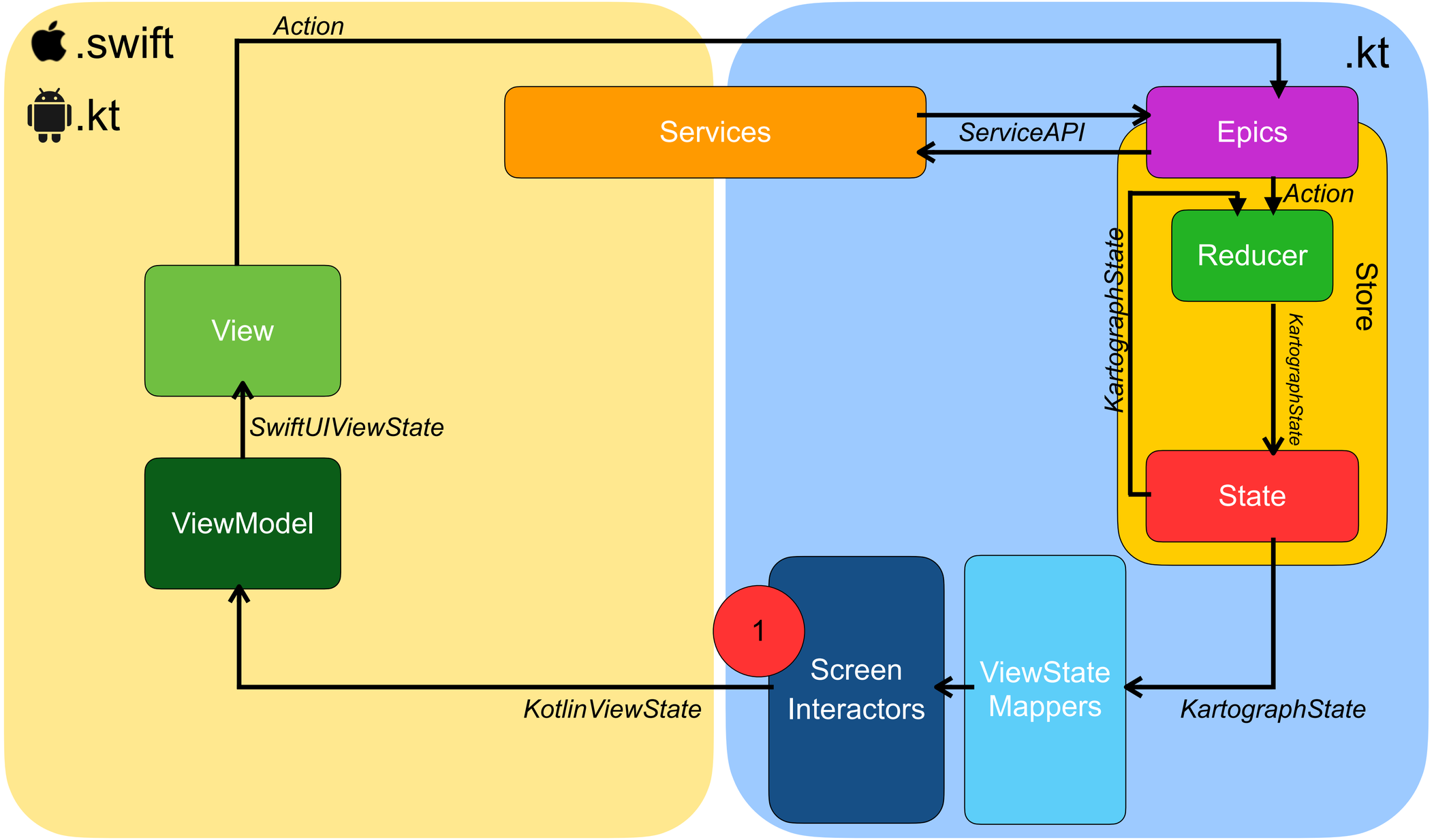


KartographViewStateMapper.kt

```
internal class MainScreenViewStateMapper(
    private val store: Store<KartographState>
) : ScreenViewStateMapper<MainScreenViewState> {

    override fun viewStates(): Flow<MainScreenViewState> {
        return store.states()
            .mapNotNull { kartographState ->
                val headerState = makeHeaderState(
                    kartographState.authState,
                    kartographState.ridesState
                )
                val featureButtonsStates = makeFeatureButtonsState()
                return@mapNotNull MainScreenViewState(headerState, featureButtonsStates)
            }
            .distinctUntilChanged()
    }
```



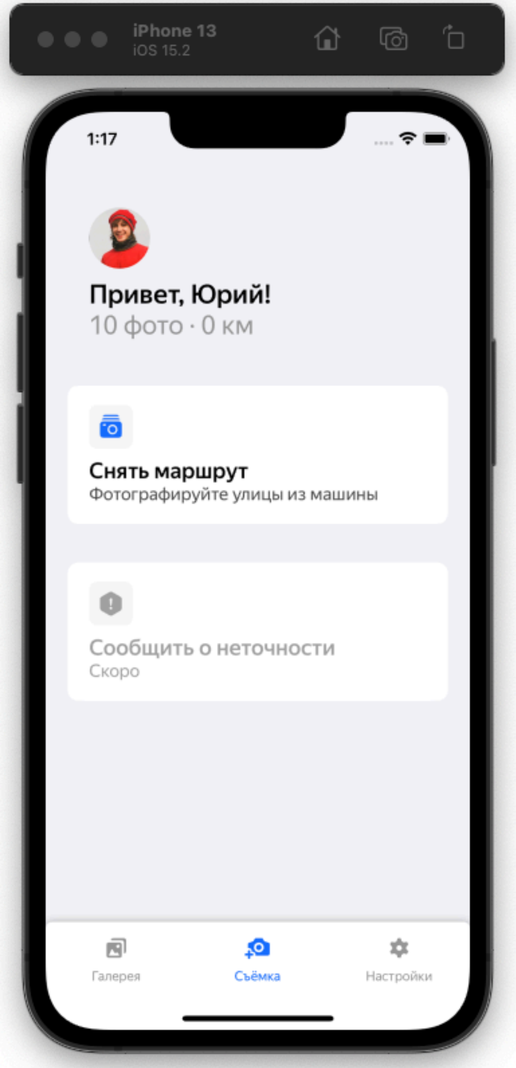
ы


.swift
.kt
Action
.kt
Services
ServiceAPI
Epics
Action
Reducer
Store
KartographState
KartographState
State
View
SwiftUIViewState
ViewModel
1
Screen
Interactors
ViewState
Mappers
KotlinViewState
KartographState

Мультиплатформенные интерфейсы

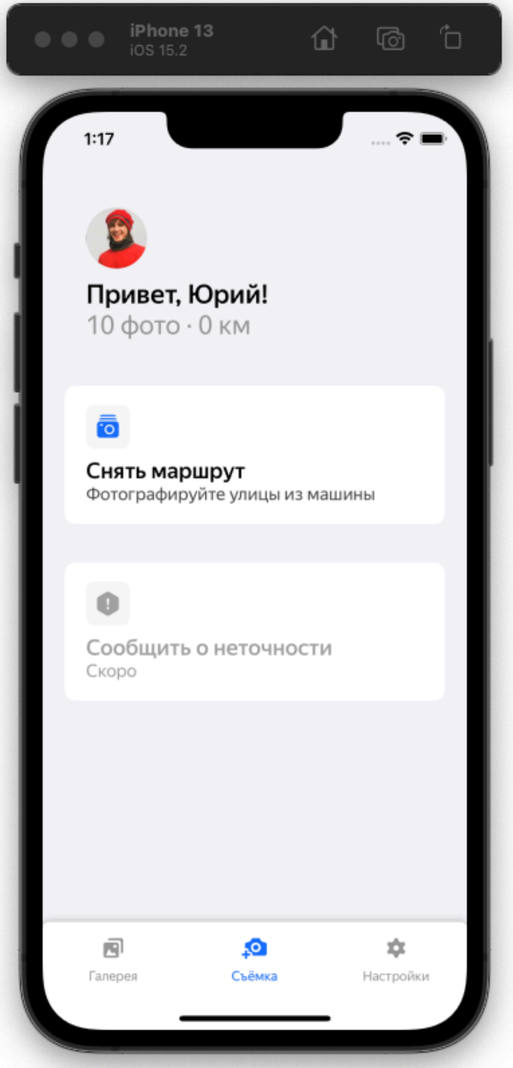
ScreenInteractor.kt

```
interface MainScreenInteractor {
    fun viewStates(): PlatformFlow<MainScreenViewState>
    fun dispatch(mainScreenAction: MainScreenAction)
}
```



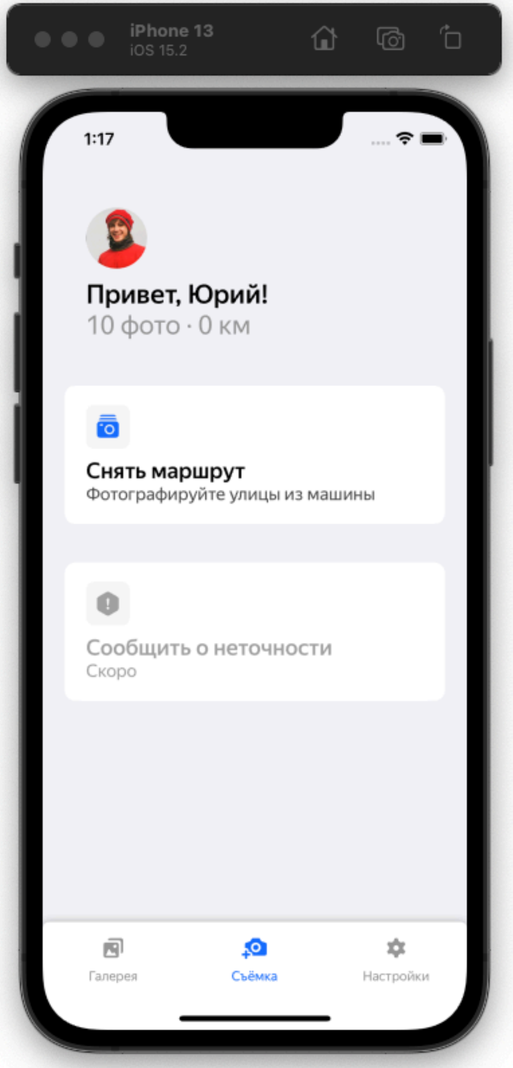
ScreenInteractor.kt

```
interface MainScreenInteractor {
    fun viewStates(): PlatformFlow<MainScreenViewState>
    fun dispatch(mainScreenAction: MainScreenAction)
}
```



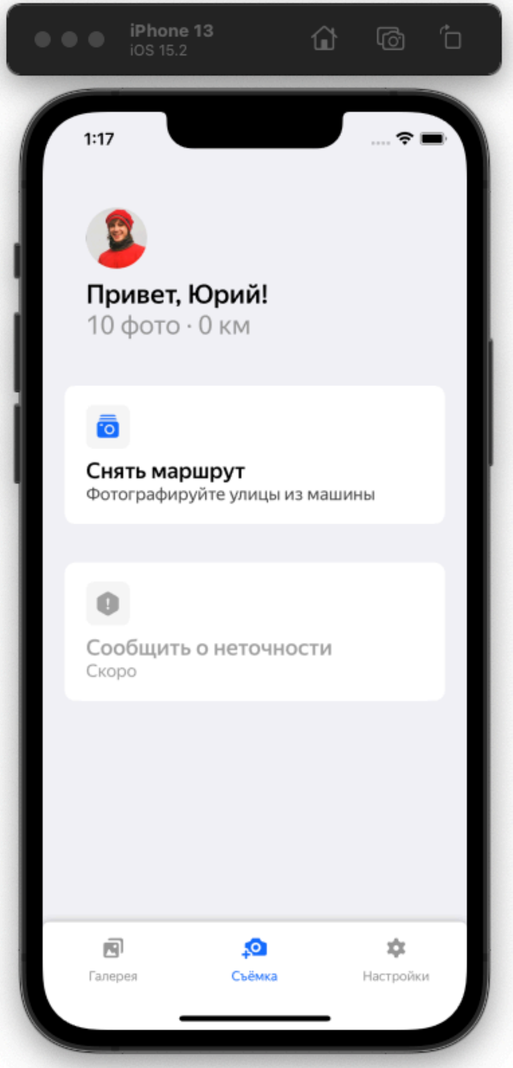
ScreenInteractor.kt

```
interface MainScreenInteractor {
    fun viewStates(): PlatformFlow<MainScreenViewState>
    fun dispatch(mainScreenAction: MainScreenAction)
}
```

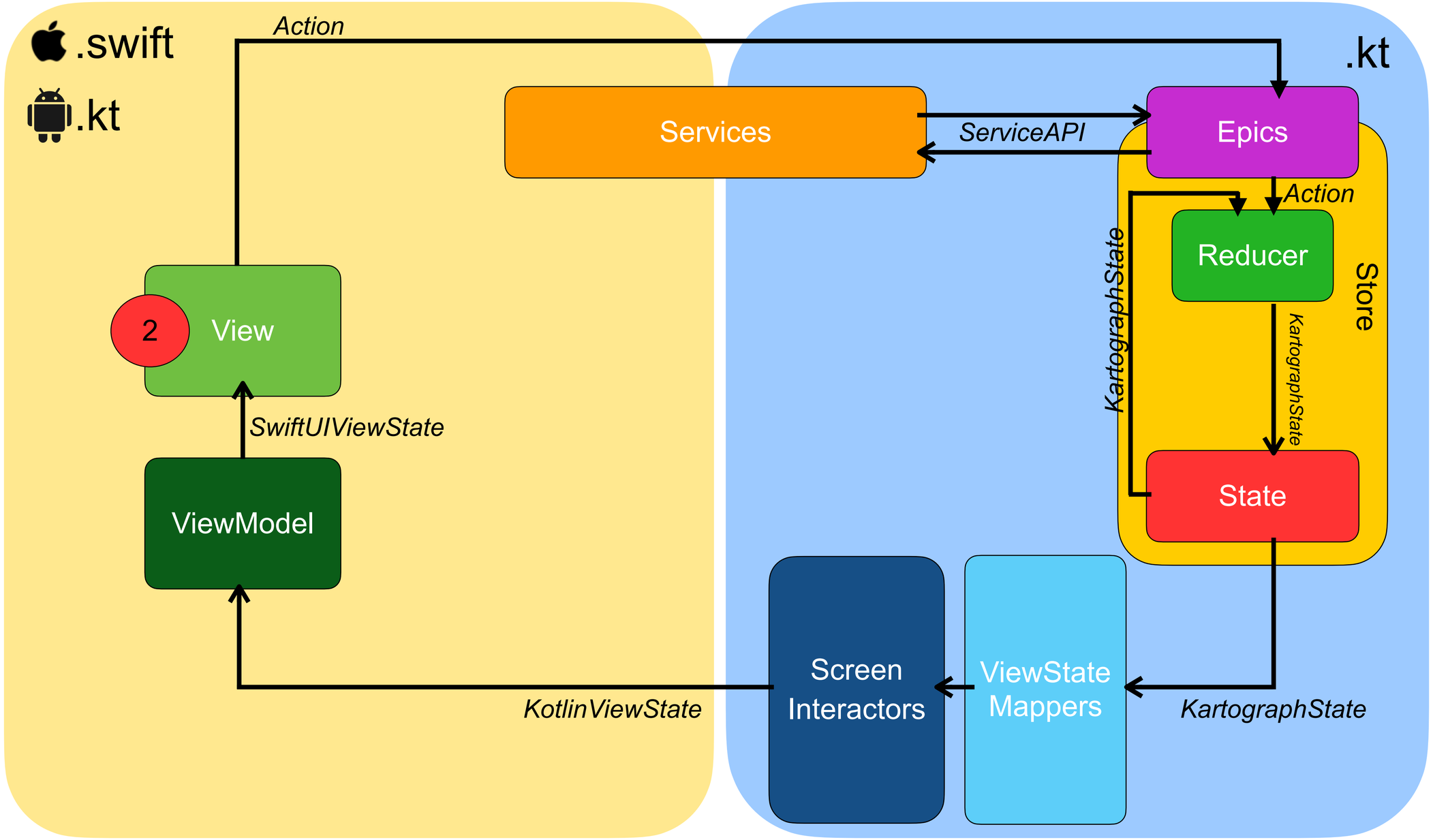


ScreenInteractor.kt

```
interface MainScreenInteractor {
    fun viewStates(): PlatformFlow<MainScreenViewState>
    fun dispatch(mainScreenAction: MainScreenAction)
}
```



Kartograph Redux



Declarative UI Frameworks

SwiftUI

```swift
struct SomeView: View {

    let title: String = "Hello, world!"

    var body: some View {
        Text(title)
    }

}
```

Compose

```kotlin
@Composable
fun SomeView(title: String = "Hello, world!") {
    Text(text = title)
}
```

Ключевые отличия

SwiftUI

- На основе UIKit
- UI Зависит от версии ОС
- Системная закрытая библиотека

Compose

- Собственный рендеринг
- Независимость от ОС
- Открытый исходный код

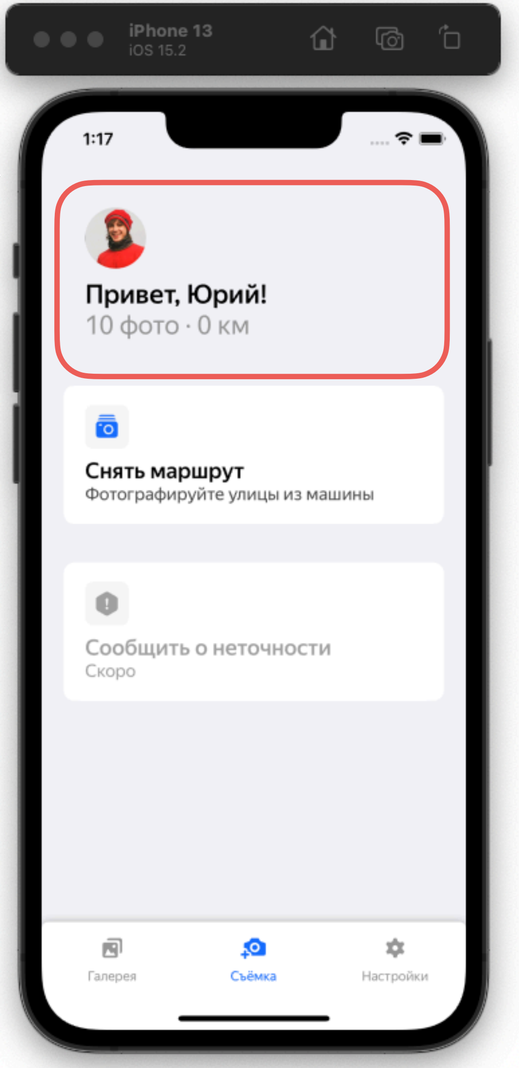
SwiftUI  / Jetpack Compose 

SwiftUI

View

```swift
struct MainScreenHeader: View {
    let avatar: UIImage
    let title: String
    let subtitle: String

    var body: some View {
        VStack(alignment: .leading) {
            Image(uiImage: avatar)
                .resizable()
                .frame(width: 56, height: 56)
                .clipShape(Circle())
            Text(title)
                .font(Fonts.medium(size: 24).font)
                .foregroundColor(RawColors.text.primary.color)

            Text(subtitle)
                .font(Fonts.regular(size: 24).font)
                .foregroundColor(RawColors.text.secondary.color)
        }
    }
}
```

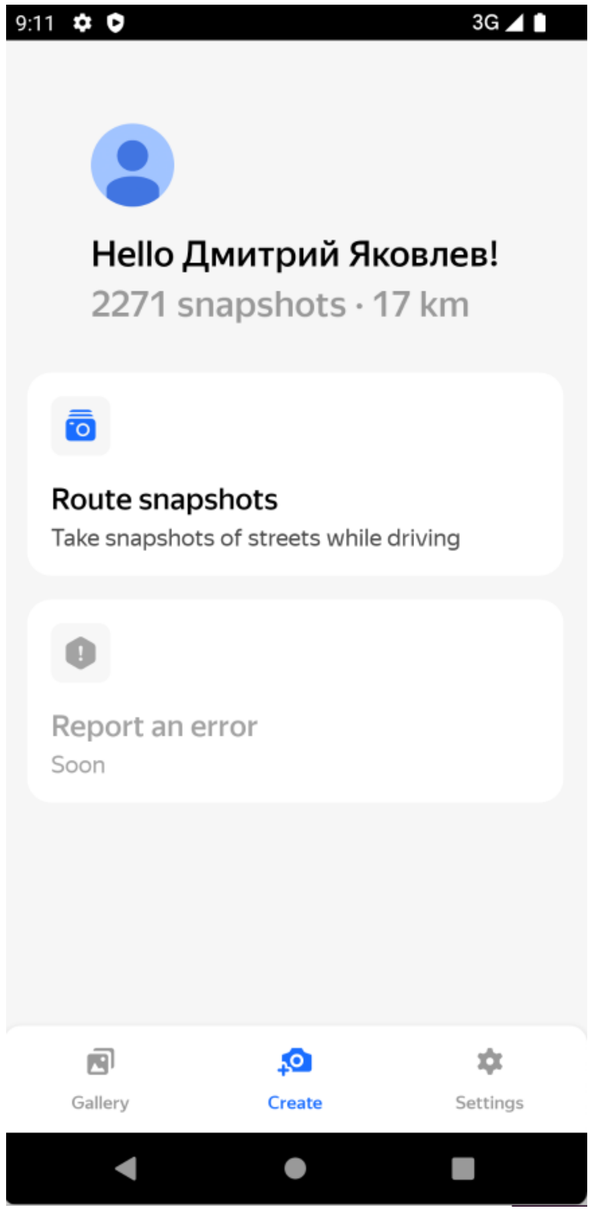
RenderView



Jetpack Compose

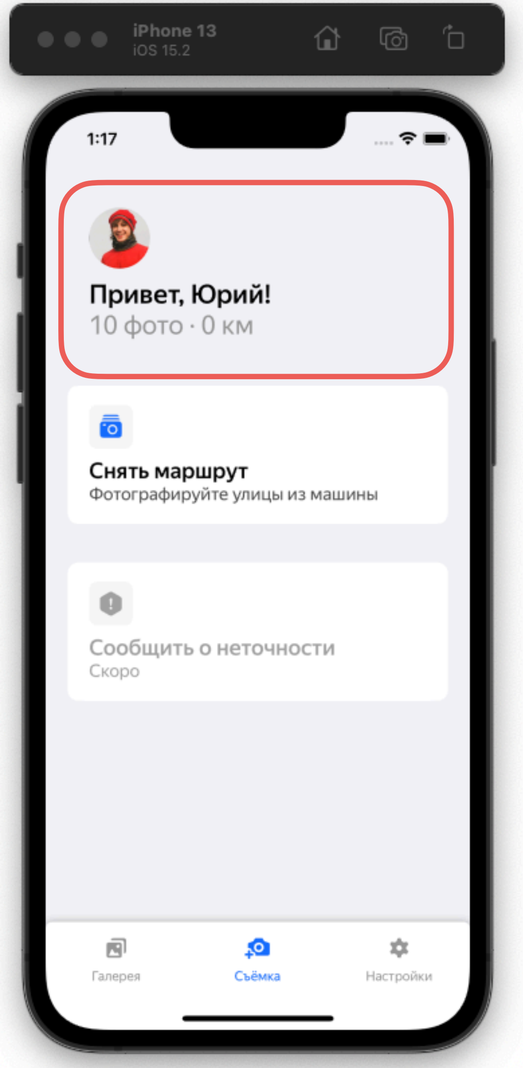
```
@Composable
internal fun MainScreenLayout(
    state: MainScreenViewState,
    onCaptureClick: () -> Unit,
    onReportClick: () -> Unit
) {
```

```
@Composable
internal fun Header(state: MainScreenHeaderViewState) {
```



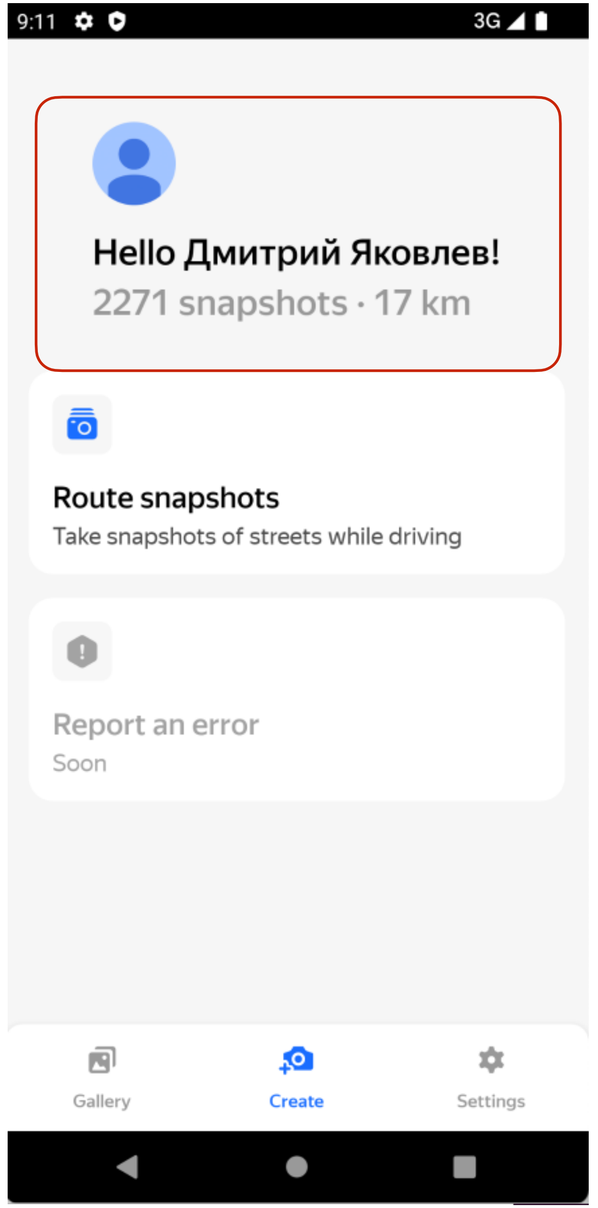
SwiftUI stateless View

```swift
struct MainScreenHeader: View {
    let avatar: UIImage
    let title: String
    let subtitle: String

    var body: some View {
        VStack(alignment: .leading) {
            Image(uiImage: avatar)
                .resizable()
                .frame(width: 56, height: 56)
                .clipShape(Circle())
            Text(title)
                .font(Fonts.medium(size: 24).font)
                .foregroundColor(RawColors.text.primary.color)

            Text(subtitle)
                .font(Fonts.regular(size: 24).font)
                .foregroundColor(RawColors.text.secondary.color)
        }
    }
}
```



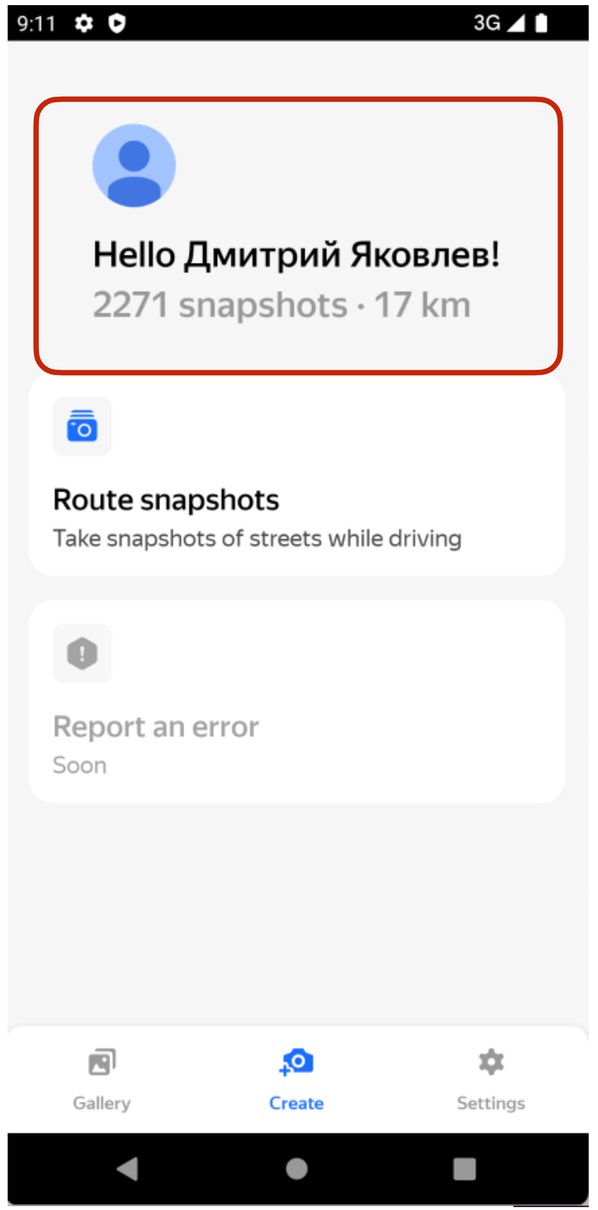
Compose stateless View

```
@Composable
internal fun Header(state: MainScreenHeaderViewState) {
    Column(
        modifier = Modifier.padding(start = 32.dp, end = 32.dp),
        horizontalAlignment = Alignment.Start,
        verticalArrangement = Arrangement.Top,
    ) { this: ColumnScope
        Spacer(modifier = Modifier.size(56.dp))
        AvatarImage(state.avatarState)
        Spacer(modifier = Modifier.size(16.dp))
        Text(
            text = state.title,
            style = MapsText.size24Medium,
            color = MapsColors.textPrimary,
        )
        Spacer(modifier = Modifier.size(4.dp))
        Text(
            text = state.subtitle,
            style = MapsText.size24Medium,
            color = MapsColors.textSecondary,
        )
    }
}
```



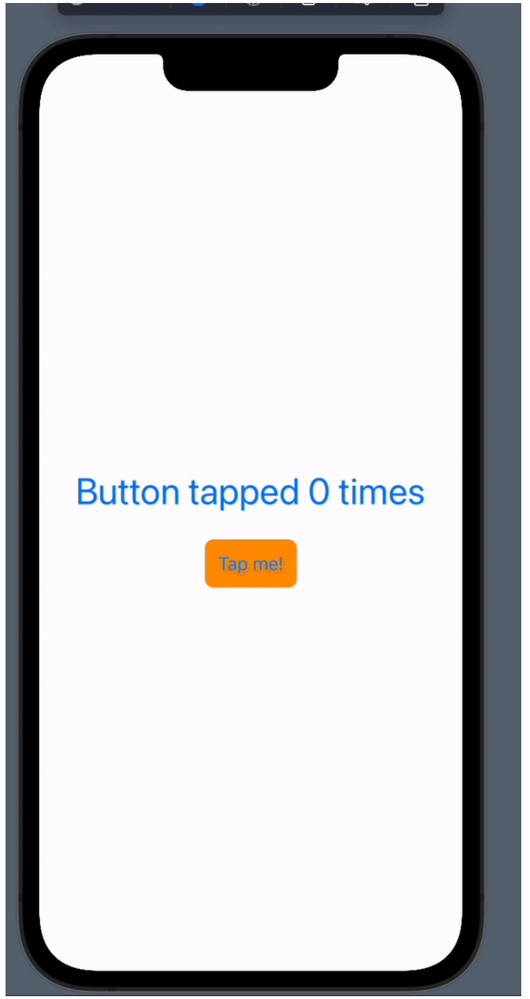
Compose stateless View

```
@Composable
internal fun Header(state: MainScreenHeaderViewState) {
    Column(
        modifier = Modifier.padding(start = 32.dp, end = 32.dp),
        horizontalAlignment = Alignment.Start,
        verticalArrangement = Arrangement.Top,
    ) { this: ColumnScope
        Spacer(modifier = Modifier.size(56.dp))
        AvatarImage(state.avatarState)
        Spacer(modifier = Modifier.size(16.dp))
        Text(
            text = state.title,
            style = MapsText.size24Medium,
            color = MapsColors.textPrimary,
        )
        Spacer(modifier = Modifier.size(4.dp))
        Text(
            text = state.subtitle,
            style = MapsText.size24Medium,
            color = MapsColors.textSecondary,
        )
    }
}
```



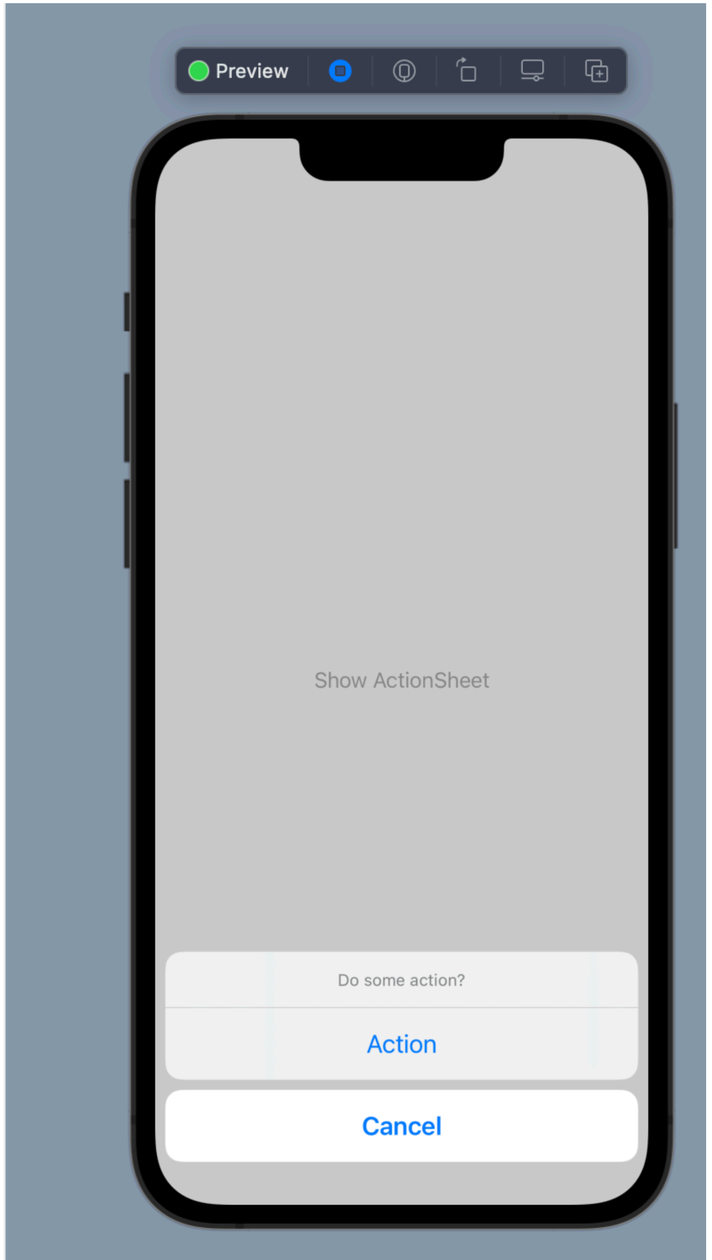
SwiftUI stateful View

```swift
struct CounterView: View {
    @State var buttonTaps: Int = 0

    var body: some View {
        VStack {
            Text("Button tapped \(buttonTaps) times")
                .font(.largeTitle)
                .foregroundColor(.blue)
                .padding()
            Button(action: { self.buttonTaps += 1}) {
                Text("Tap me!")
                    .padding(12)
                    .background(Color.orange)
                    .cornerRadius(8)
            }
        }
    }
}
```



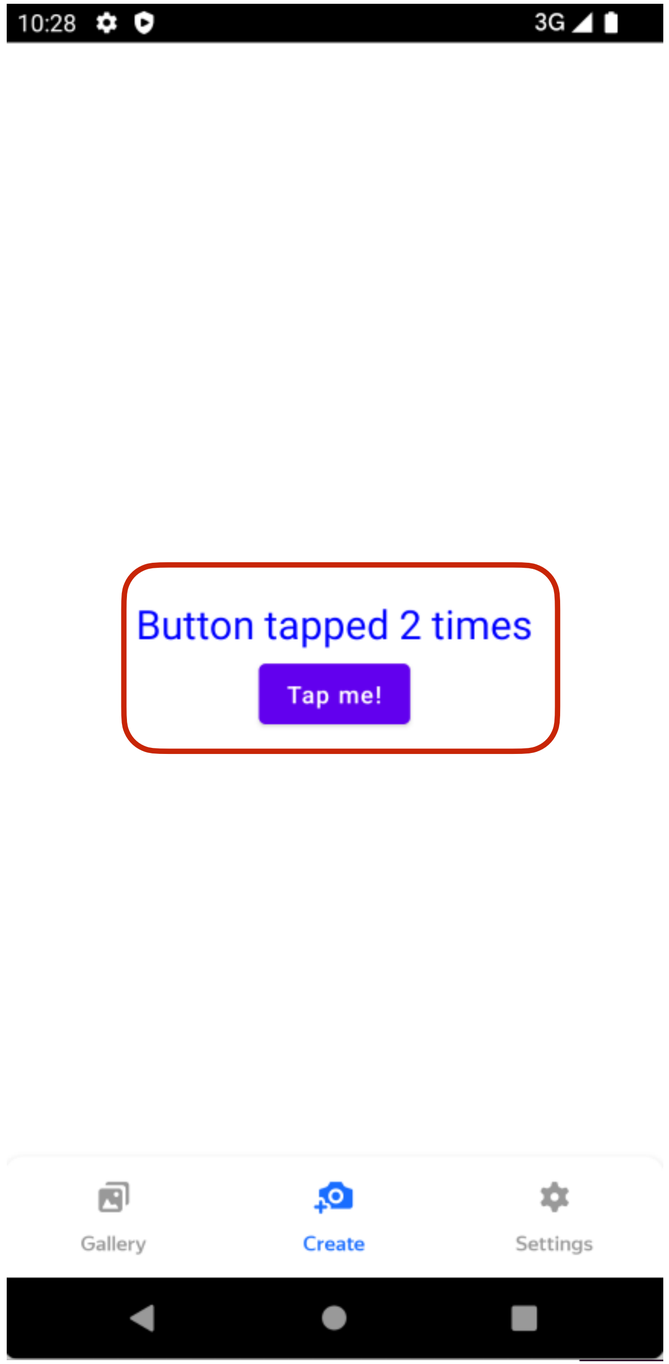
SwiftUI stateful View

```swift
import Combine
import SwiftUI
import KartographUI

struct ContentView: View {

    var body: some View {
        ActionsButtonView()
    }

}

struct ActionsButtonView: View {
    @State private var actionsPresented = false

    var body: some View {
        Button(
            action: { actionsPresented = true },
            label: { Text("Show ActionSheet") }
        )
        .actionSheet(isPresented: $actionsPresented) {
            ActionSheet(
                title: Text("Do some action?"),
                buttons: [
                    .default(Text("Action")) {
                        // do something
                    },
                    .cancel(Text("Cancel"))
                ]
            )
        }
    }
}
```



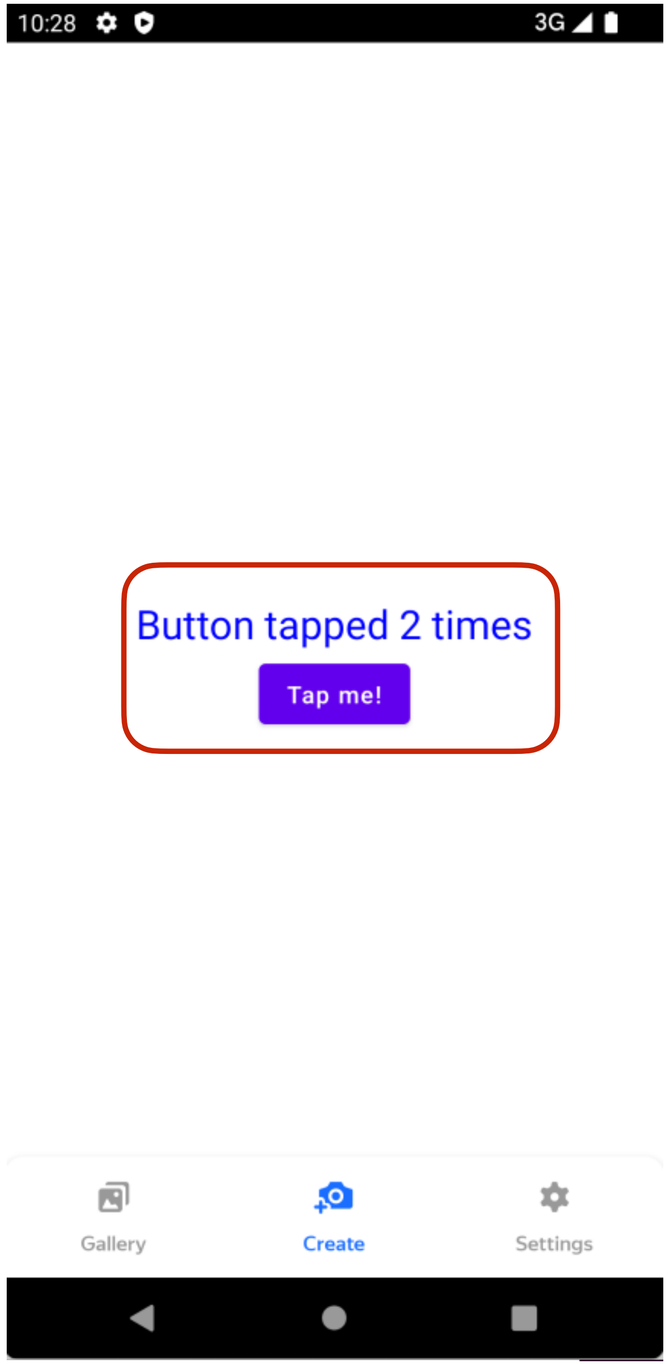
Compose stateful View

```
@Composable
fun CounterView() {
    var buttonTaps by remember { mutableStateOf( value: 0) }
    Box { this: BoxScope
        Column(modifier = Modifier
                .padding(16.dp)
                .align(Alignment.Center),
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            Text(
                text = "Button tapped $buttonTaps times",
                color = Color.Blue,
                modifier = Modifier.padding(bottom = 8.dp),
                style = MaterialTheme.typography.h5
            )
            Button(
                onClick = { buttonTaps++ }) { this: RowScope
                Text(
                    text = "Tap me!",
                    color = Color.White,
                )
            }
        }
    }
}
```



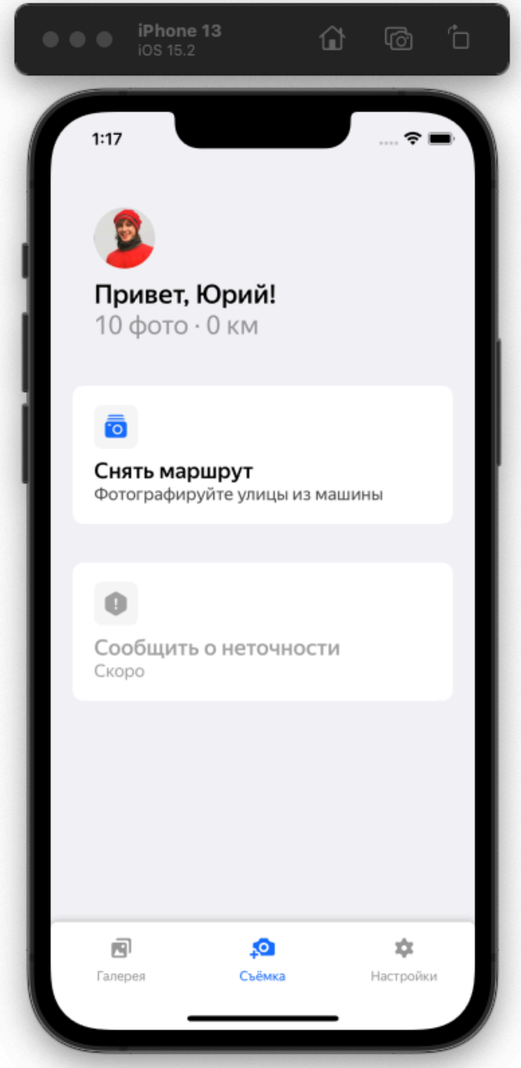
Compose stateful View

```
@Composable
fun CounterView() {
    var buttonTaps by remember { mutableStateOf( value: 0) }
    Box { this: BoxScope
        Column(modifier = Modifier
                .padding(16.dp)
                .align(Alignment.Center),
            horizontalAlignment = Alignment.CenterHorizontally
        ) { this: ColumnScope
            Text(
                text = "Button tapped $buttonTaps times",
                color = Color.Blue,
                modifier = Modifier.padding(bottom = 8.dp),
                style = MaterialTheme.typography.h5
            )
            Button(
                onClick = { buttonTaps++ }) { this: RowScope
                Text(
                    text = "Tap me!",
                    color = Color.White,
                )
            }
        }
    }
}
```



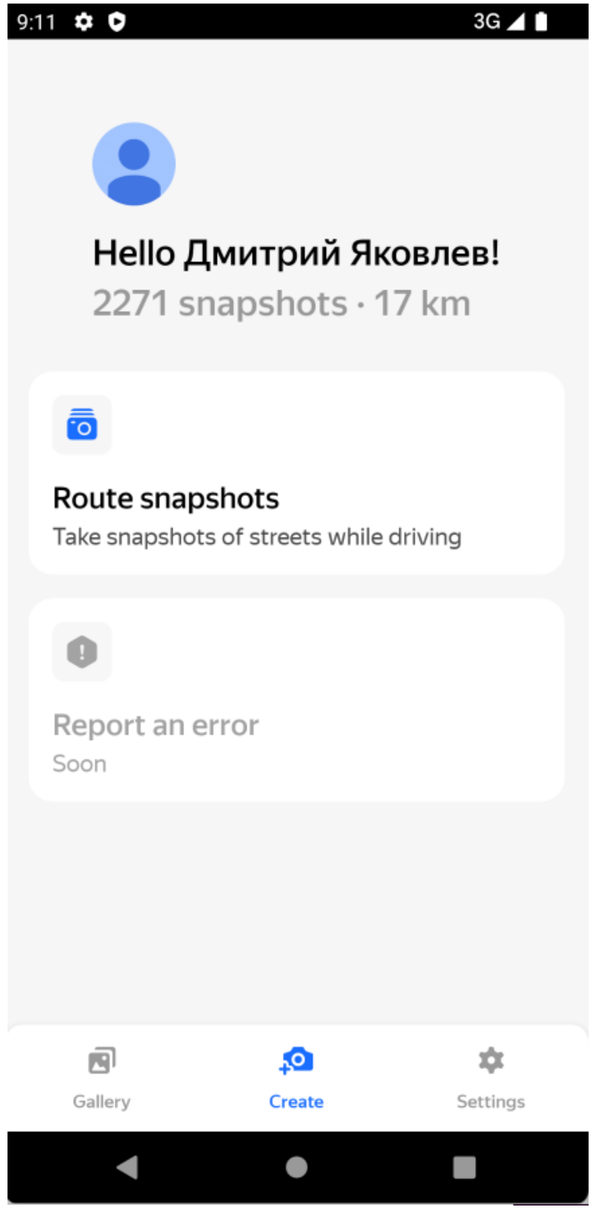
SwiftUI ViewState based view

```swift
    public typealias ViewModel = ViewStateModel<MainScreenViewState, MainScreenActionsHandler>

    public init(model: ViewModel) {
        self.model = model
    }

    @ObservedObject var model: ViewModel

    public var body: some View {
        List {
            MainScreenHeader(
                avatar: model.viewState.avatar,
                title: model.viewState.title,
                subtitle: model.viewState.subtitle
            )
            .listRowBackground(Color.clear)

            ForEach(model.viewState.features) { feature in
                Button(action: { model.actionsHandler.onFeature(id: feature.id) }) {
                    MainScreenFeatureView(feature: feature)
                }
                .disabled(!feature.isEnabled)
                .opacity(feature.isEnabled ? 1.0 : 0.9)
            }
        }
    }
}
```



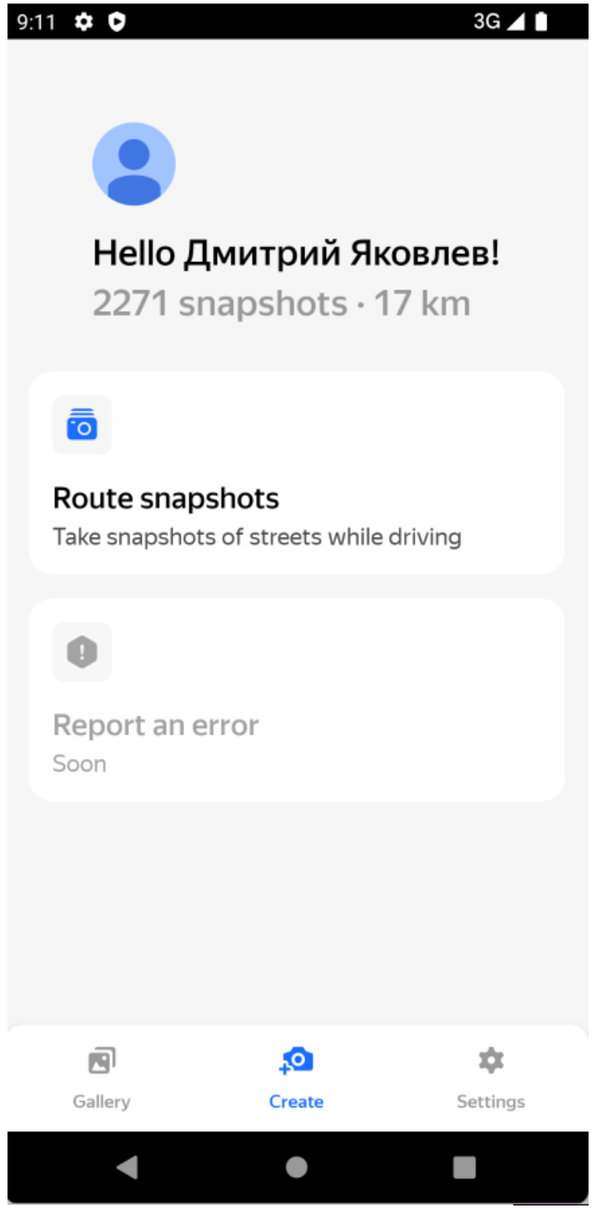
Compose ViewState based View

```
internal object MainScreen {

    @Immutable
    data class State(
        val viewState: MainScreenViewState,
    )

    @Composable
    fun View(state: State, dispatch: (action: UserAction) -> Unit) {
        val viewState = state.viewState
        Column(
            modifier = Modifier
                .fillMaxSize()
                .background(MapsColors.bgAdditional)
                .recomposeHighlighter(),
        ) { this: ColumnScope
            MainScreenLayout(viewState, dispatch)
        }
    }

    @Composable
    private fun MainScreenLayout(
        state: MainScreenViewState,
        dispatch: (action: UserAction) -> Unit
    ) {
        //Actual Layout
    }
}
```



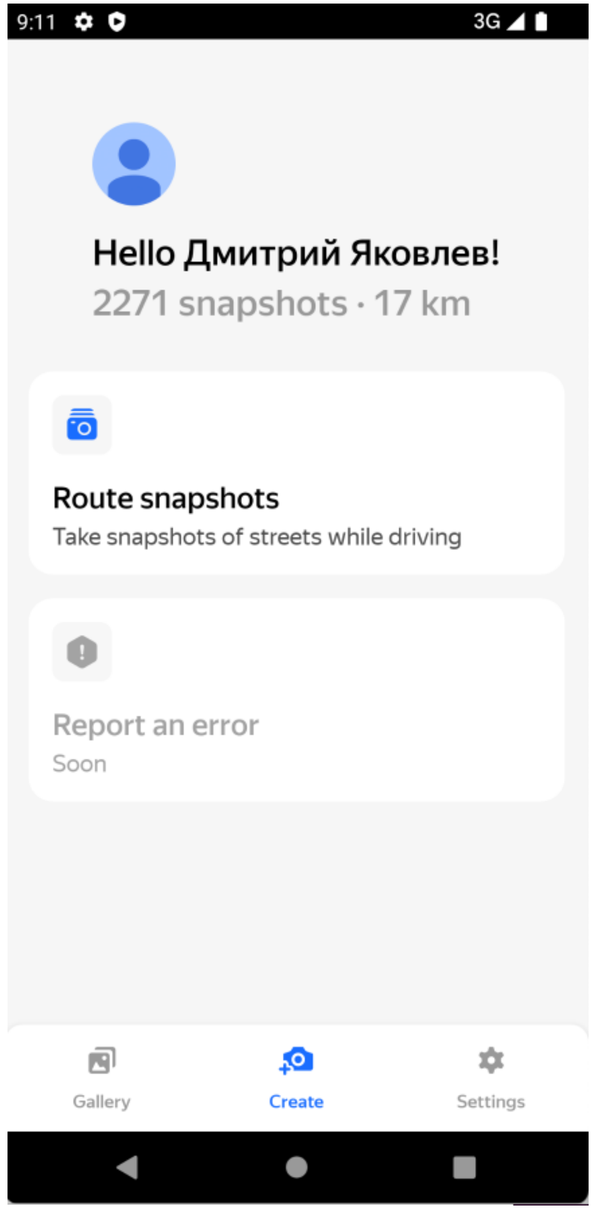
Compose ViewState based View

```
internal object MainScreen {

    @Immutable
    data class State(
        val viewState: MainScreenViewState,
    )

    @Composable
    fun View(state: State, dispatch: (action: UserAction) -> Unit) {
        val viewState = state.viewState
        Column(
            modifier = Modifier
                .fillMaxSize()
                .background(MapsColors.bgAdditional)
                .recomposeHighlighter(),
        ) { this: ColumnScope
            MainScreenLayout(viewState, dispatch)
        }
    }

    @Composable
    private fun MainScreenLayout(
        state: MainScreenViewState,
        dispatch: (action: UserAction) -> Unit
    ) {
        //Actual Layout
    }
}
```

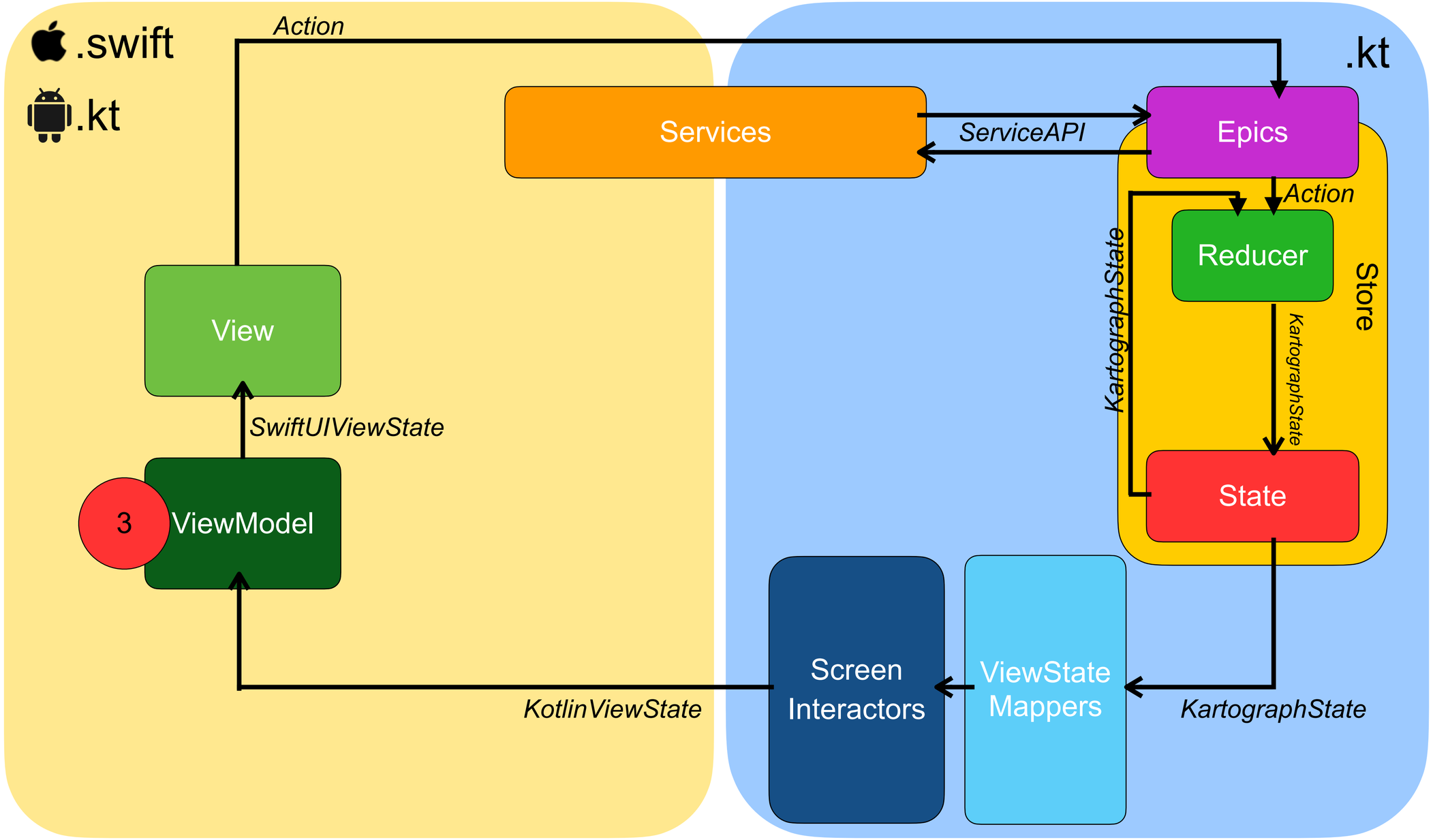


Compose ViewState based View

```
internal object MainScreen {

    @Immutable
    data class State(
        val viewState: MainScreenViewState,
    )

    @Composable
    fun View(state: State, dispatch: (action: UserAction) -> Unit) {
        val viewState = state.viewState
        Column(
            modifier = Modifier
                .fillMaxSize()
                .background(MapsColors.bgAdditional)
                .recomposeHighlighter(),
        ) { this: ColumnScope
            MainScreenLayout(viewState, dispatch)
        }
    }

    @Composable
    private fun MainScreenLayout(
        state: MainScreenViewState,
        dispatch: (action: UserAction) -> Unit
    ) {
        //Actual Layout
    }
}
```



Kartograph Redux



ViewStateModel.swift

```swift
public final class ViewStateModel<ViewState, ActionsHandler>: ObservableObject {

    @Published public var viewState: ViewState
    public let actionsHandler: ActionsHandler

    public init<PublisherT: Publisher>(
        _ initialViewState: ViewState,
        updates: PublisherT,
        actionsHandler: ActionsHandler
    ) where PublisherT.Output == ViewState {
        self.actionsHandler = actionsHandler
        self.viewState = initialViewState
        canceller = updates.sink{ [weak self] in
            self?.viewState = $0
        }
    }

    private var canceller: AnyCancellable?
}
```

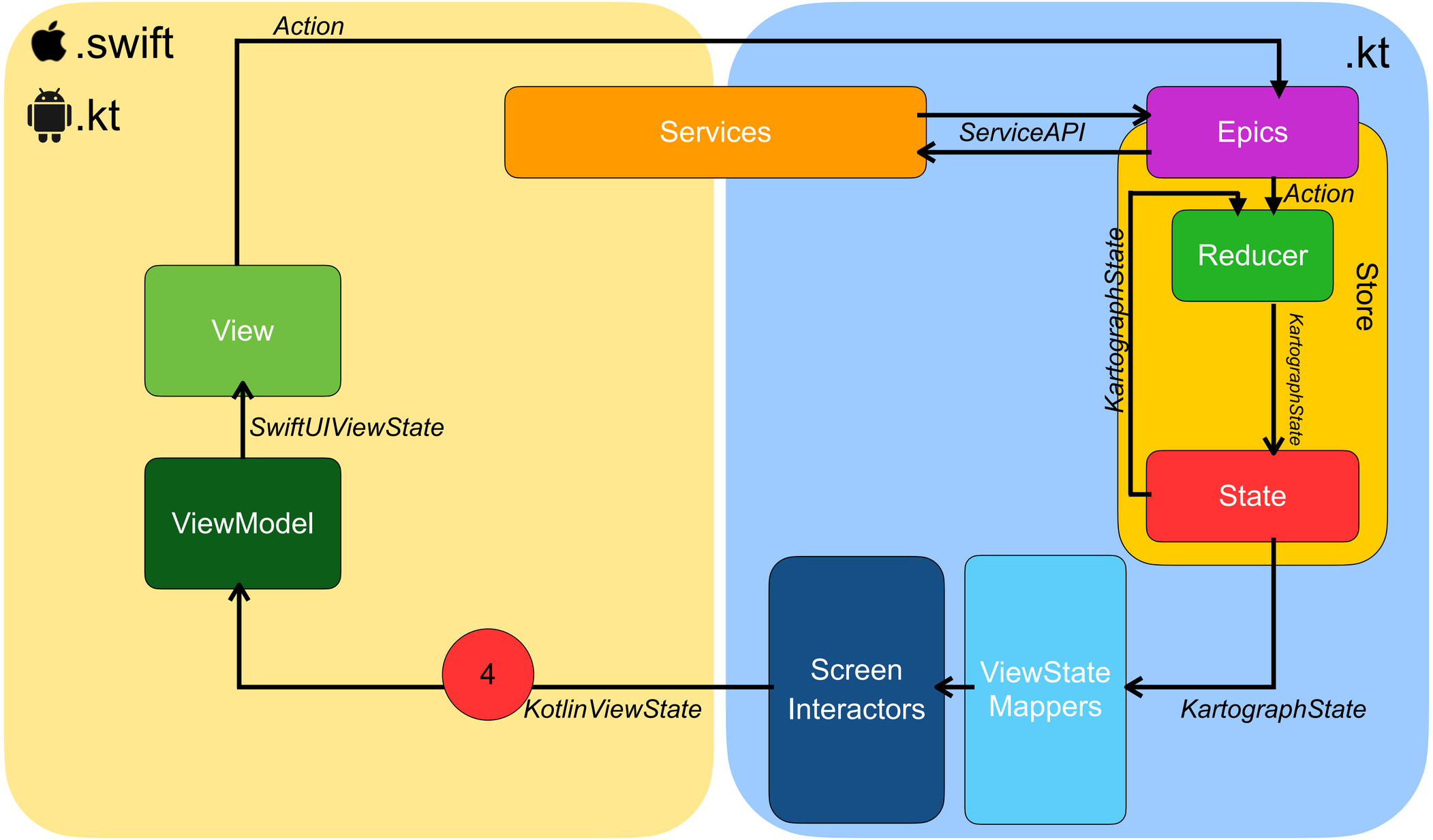
ViewStateModel.swift

```swift
public final class ViewStateModel<ViewState, ActionsHandler>: ObservableObject {

    @Published public var viewState: ViewState
    public let actionsHandler: ActionsHandler

    public init<PublisherT: Publisher>(
        _ initialViewState: ViewState,
        updates: PublisherT,
        actionsHandler: ActionsHandler
    ) where PublisherT.Output == ViewState {
        self.actionsHandler = actionsHandler
        self.viewState = initialViewState
        canceller = updates.sink{ [weak self] in
            self?.viewState = $0
        }
    }

    private var canceller: AnyCancellable?
}
```

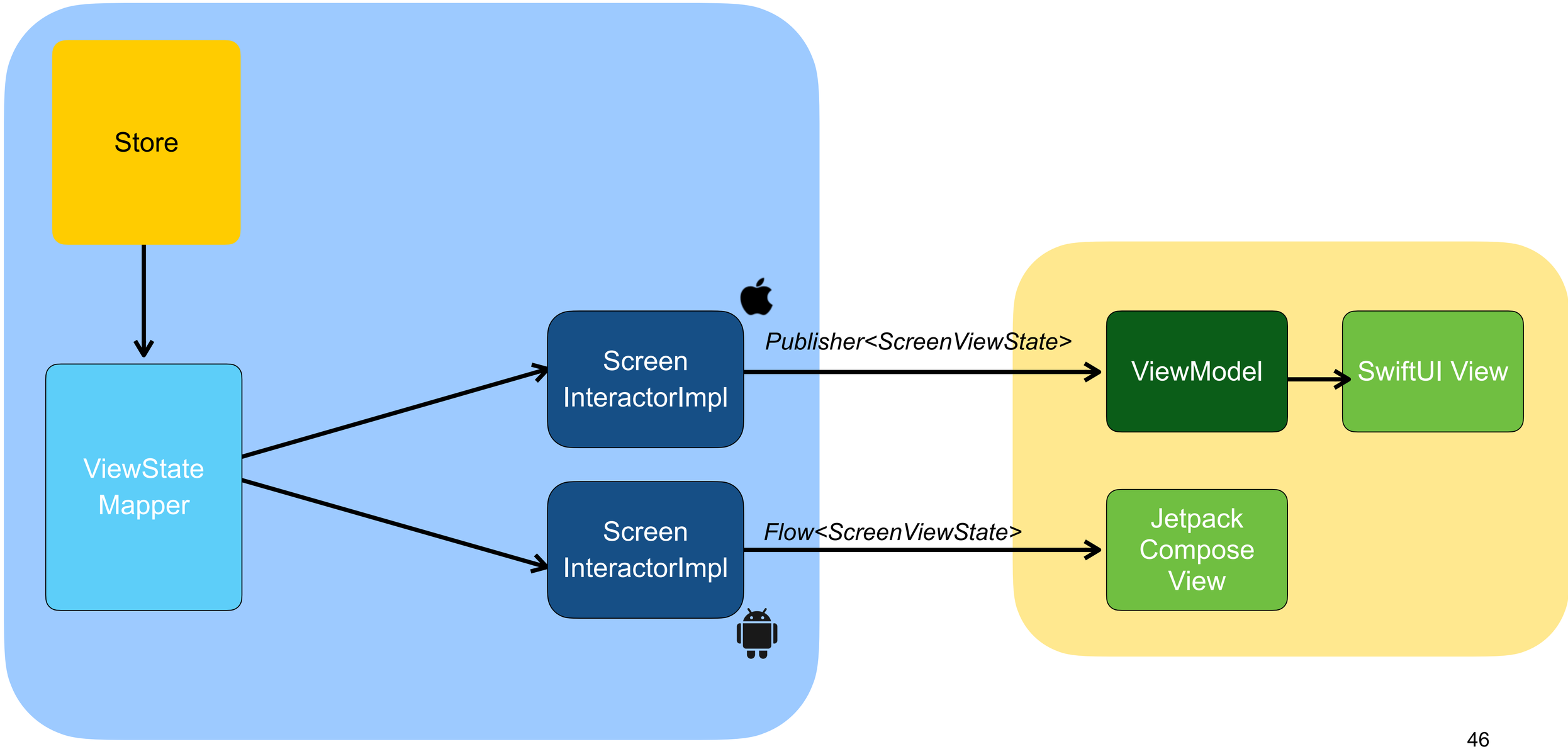
ViewStateModel.swift

```swift
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
public protocol ObservableObject : AnyObject {

    /// The type of publisher that emits before the object has changed.
    associatedtype ObjectWillChangePublisher : Publisher = ObservableObjectPublisher where
        Self.ObjectWillChangePublisher.Failure == Never

    /// A publisher that emits before the object has changed.
    var objectWillChange: Self.ObjectWillChangePublisher { get }
}
```

Kartograph Redux



Main screen


Store
ViewState Mapper
Screen InteractorImpl
Screen InteractorImpl
Publisher<ScreenViewState>
Flow<ScreenViewState>
ViewModel
SwiftUI View
Jetpack Compose View

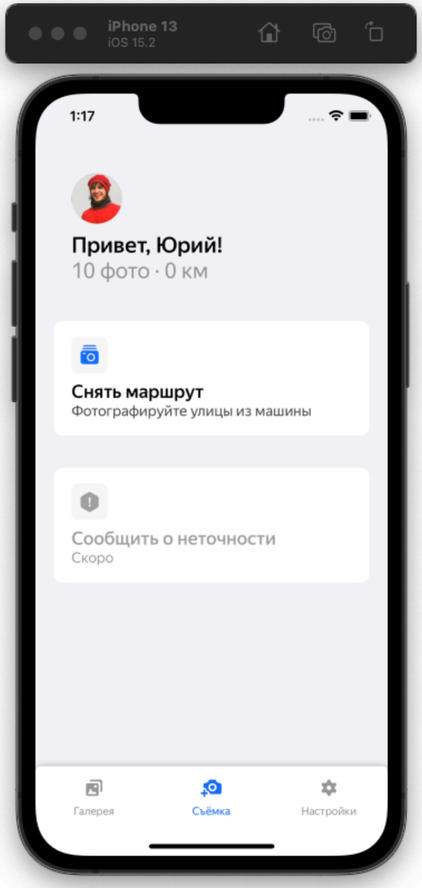
SwiftUI ViewState based view

.kt

```kotlin
data class MainScreenViewState(val header: MainScreenHeaderViewState, val features: List<MainScreenFeatureViewState>)

data class MainScreenHeaderViewState(val avatarState: AvatarState, val title: String, val subtitle: String)

sealed class AvatarState {
    object Placeholder : AvatarState()
    data class Image(val image: RuntimeImage) : AvatarState()
}

data class MainScreenFeatureViewState(val type: MainScreenFeatureType, val title: String, val subtitle: String, val isEnabled: Boolean)

enum class MainScreenFeatureType {
    MIRRORS, FEEDBACK
}
```

.swift

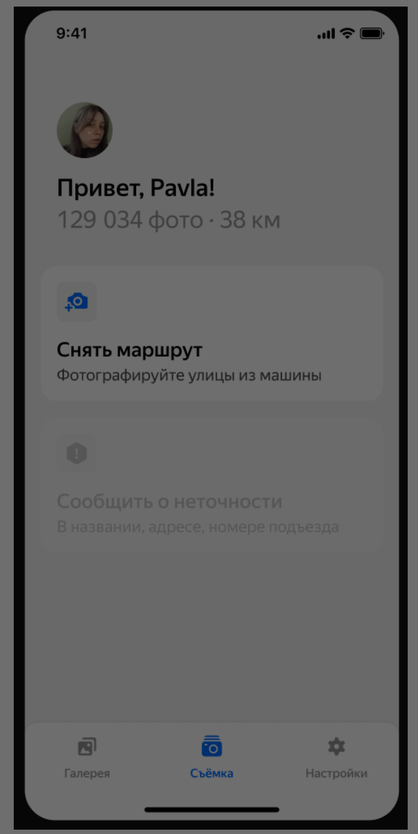
```swift
public struct MainScreenFeature: Identifiable {
    public var id: String { title }

    let icon: UIImage
    let title: String
    let subtitle: String
    let isEnabled: Bool
}

public struct MainScreenViewState {
    let avatar: UIImage
    let title: String
    let subtitle: String
    let features: [MainScreenFeature]
}
```



SwiftUI ViewState based view

```
extension MainScreen.ViewModel {
    convenience init(kotlinInteractor: MainScreenInteractor) {
        let actionsHandler = MainScreenHandlerImpl(interactor: kotlinInteractor)
        let updates = toCombine(kotlinInteractor.viewStates()).map { $0.toNative() }
        self.init(MainScreenViewState.initial, updates: updates, actionsHandler: actionsHandler)
    }
}
```

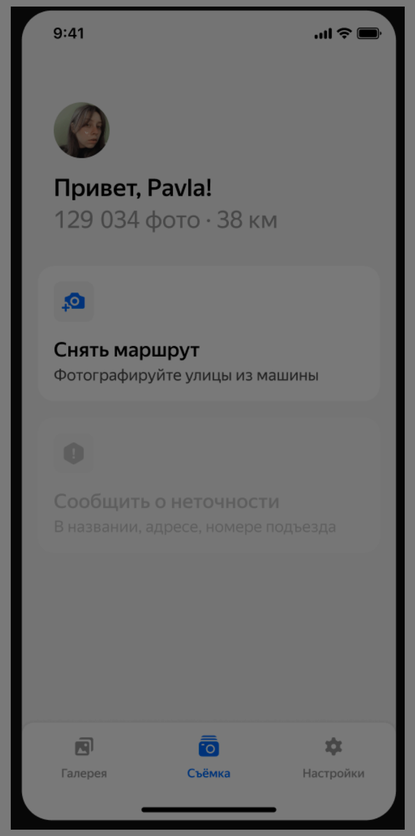
```
private extension KotlinNative.MainScreenViewState {

    func toNative() -> KartographUI.MainScreenViewState {
        let avatar = (header.avatarState as? AvatarState.Image)?.image ?? RawImages.settingsUserpic_72.day
        let features: [MainScreenFeature] = self.features.map { $0.toNative() }
        return KartographUI.MainScreenViewState(
            avatar: avatar,
            title: header.title,
            subtitle: header.subtitle,
            features: features
        )
    }
}
```



SwiftUI ViewState based view

```
extension MainScreen.ViewModel {
    convenience init(kotlinInteractor: MainScreenInteractor) {
        let actionsHandler = MainScreenHandlerImpl(interactor: kotlinInteractor)
        let updates = toCombine(kotlinInteractor.viewStates()).map { $0.toNative() }
        self.init(MainScreenViewState.initial, updates: updates, actionsHandler: actionsHandler)
    }
}
```

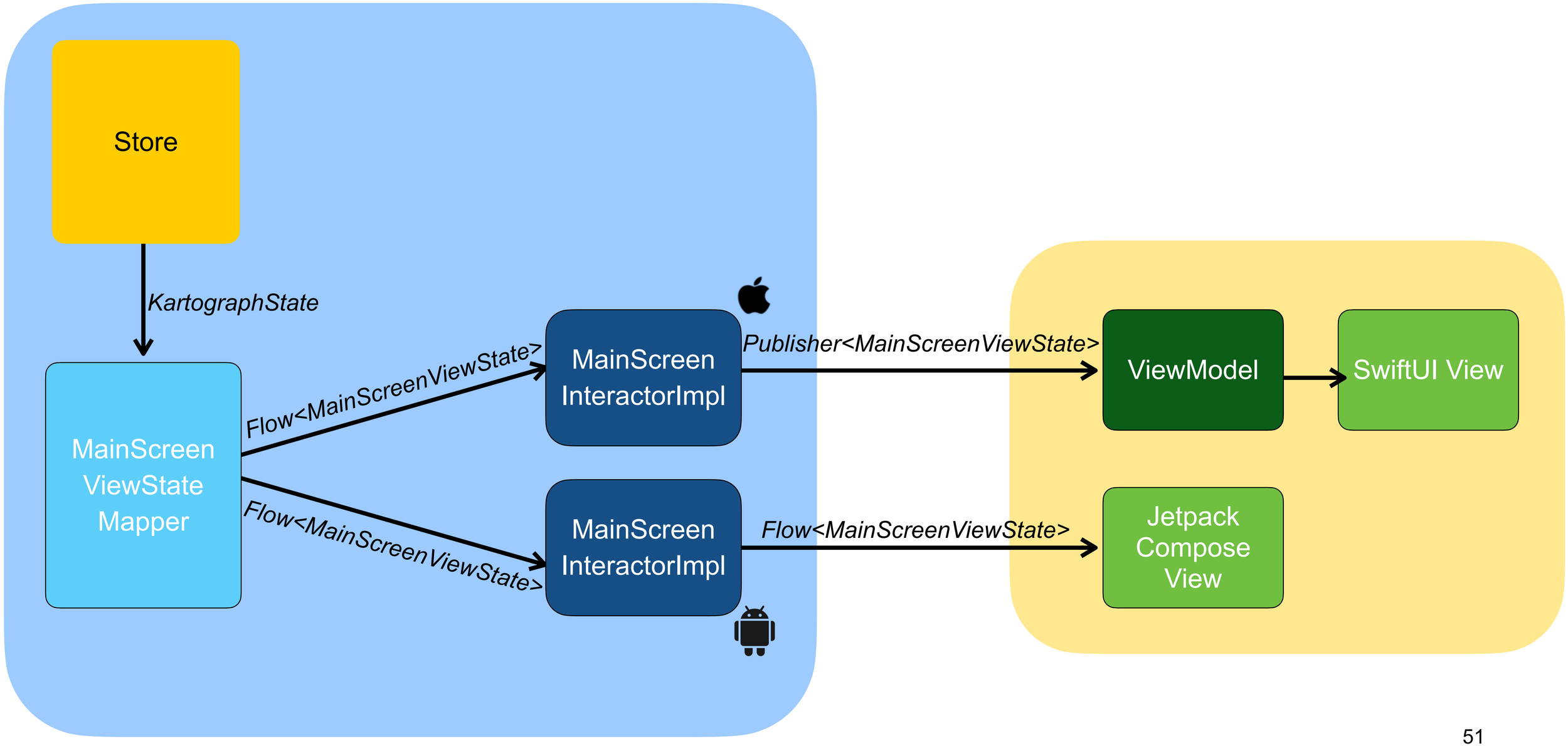
```
private extension KotlinNative.MainScreenViewState {

    func toNative() -> KartographUI.MainScreenViewState {
        let avatar = (header.avatarState as? AvatarState.Image)?.image ?? RawImages.settingsUserpic_72.day
        let features: [MainScreenFeature] = self.features.map { $0.toNative() }
        return KartographUI.MainScreenViewState(
            avatar: avatar,
            title: header.title,
            subtitle: header.subtitle,
            features: features
        )
    }
}
```



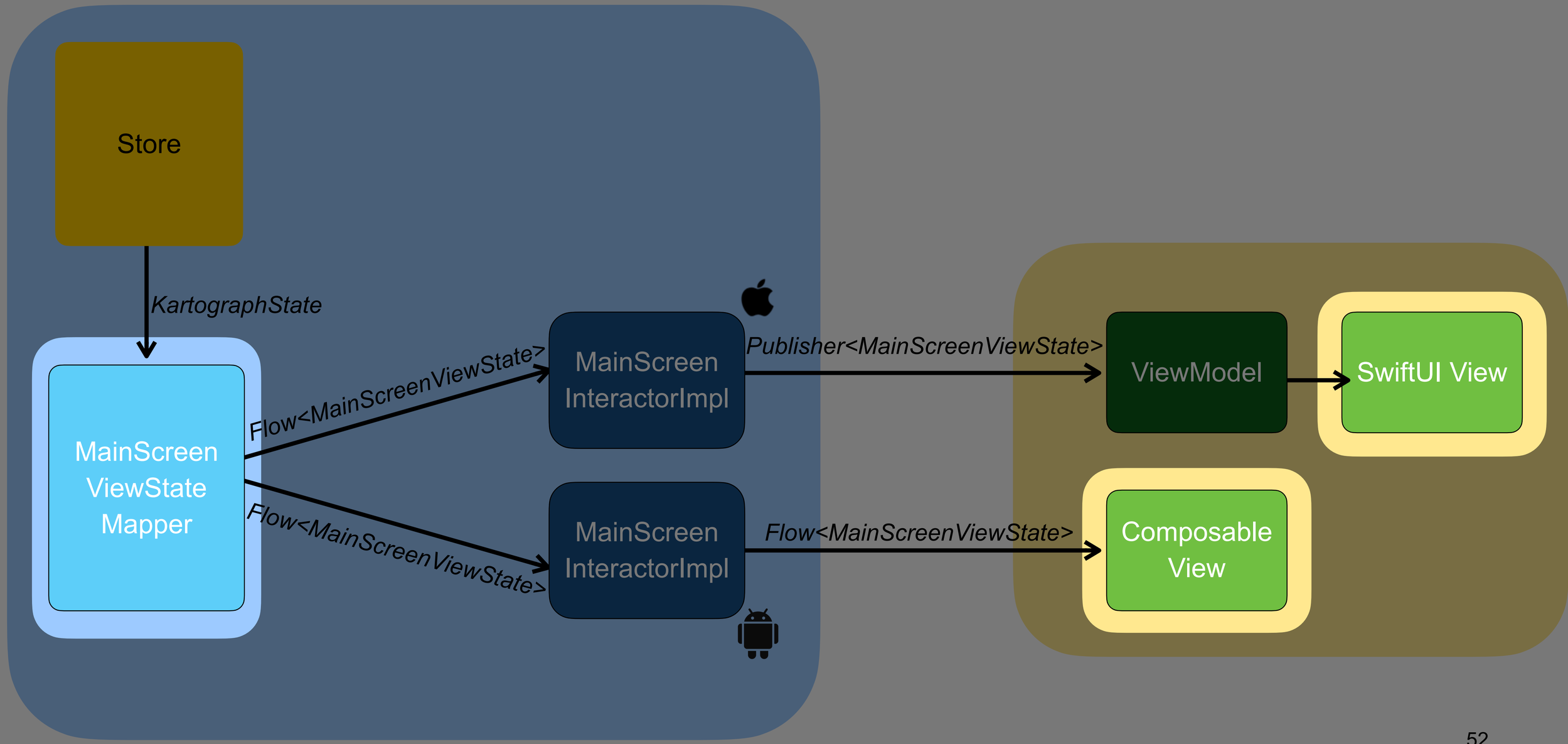
Подписка в Jetpack Compose

```
private val composeView by bind<ComposeView>(R.id.compose_view)


composeView.setContent {
    val state by interactor.viewStates()
        .collectAsState(
            initial = interactor.currentState(),
            context = scope.coroutineContext
        )
    MapsDefaultTheme {
        MainScreenLayout(
            state = state,
            dispatch = interactor::dispatch,
        )
    }
}
```

Main screen


Store
KartographState
MainScreen ViewState Mapper
Flow<MainScreenViewState>
Flow<MainScreenViewState>
MainScreen InteractorImpl
MainScreen InteractorImpl
Publisher<MainScreenViewState>
Flow<MainScreenViewState>
ViewModel
SwiftUI View
Jetpack Compose View

Main screen


Store
KartographState
MainScreen ViewState Mapper
Flow<MainScreenViewState>
Flow<MainScreenViewState>
MainScreen InteractorImpl
MainScreen InteractorImpl
Publisher<MainScreenViewState>
Flow<MainScreenViewState>
ViewModel
SwiftUI View
Composable View

SwiftUI в UIKit

Использование SwiftUI View в UIKit

```
case .permissionRationale:
    let vm = FullscreenNotificationScreenView.ViewModel(permissionsInteractor:
        kartographUiComponent.permissionScreenInteractor)
    viewController = UIHostingController(rootView: FullscreenNotificationScreenView(model: vm))
case .settings:
    let vm = try await SettingsScreen.ViewModel(kotlinInteractor:
        kartographUiComponent.settingsScreenInteractor)
    viewController = UIHostingController(rootView: SettingsScreen(model: vm))
```

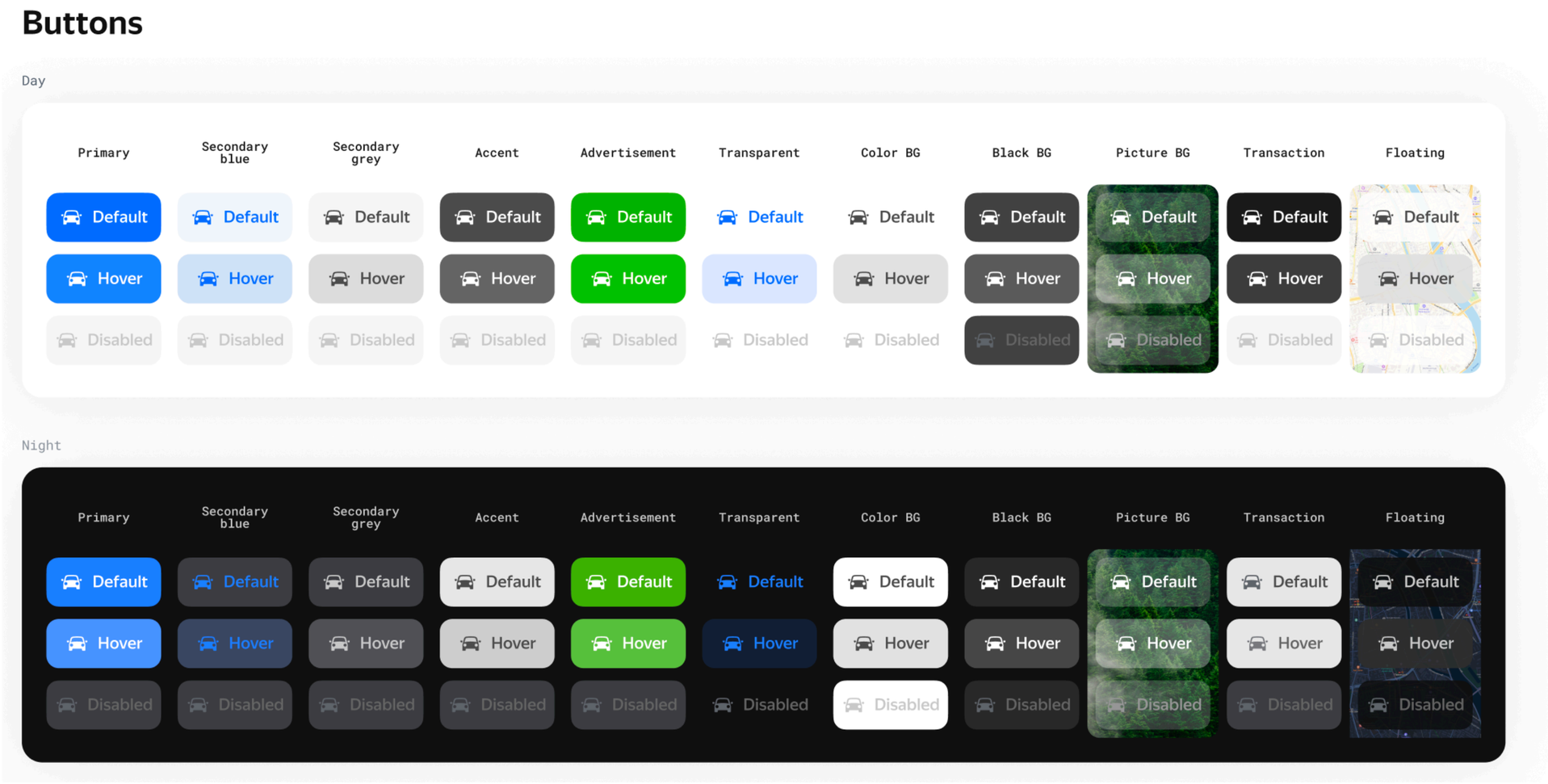
Composable в AndroidView

Точка входа в Composable

```
private val composeView by bind<ComposeView>(R.id.compose_view)


composeView.setContent {
    val state by interactor.viewStates()
        .collectAsState(
            initial = interactor.currentState(),
            context = scope.coroutineContext
        )
    MapsDefaultTheme {
        MainScreenLayout(
            state = state,
            dispatch = interactor::dispatch,
        )
    }
}
```

UIKit в SwiftUI

Дизайн-система



CommonViewWrapper

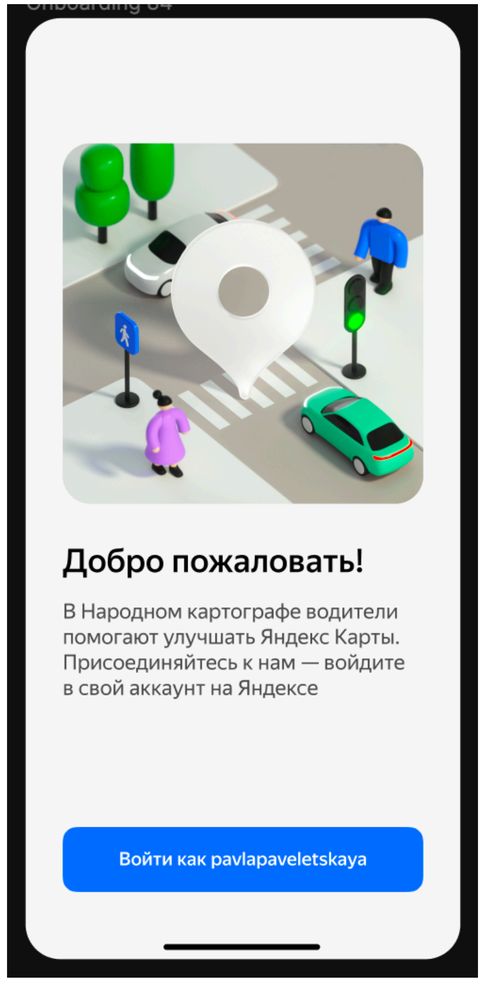
```swift
import SwiftUI
import YandexMapsUI

struct CommonViewWrapper<Model: CommonViewModel>: UIViewRepresentable {
    let model: Model

    func makeUIView(context: Context) -> CommonView {
        return model.makeView()
    }

    func updateUIView(_ uiView: CommonView, context: Context) {
        uiView.bind(to: model)
    }
}

```

CommonViewWrapper

```
11  struct CommonViewWrapper<Model: CommonViewModel>: UIViewRepresentable {
12      let model: Model
13
14      func makeUIView(context: Context) -> CommonView {
15          return model.makeView()
16      }
17
18      func updateUIView(_ uiView: CommonView, context: Context) {
19          uiView.bind(to: model)
20      }
21  }
22
```

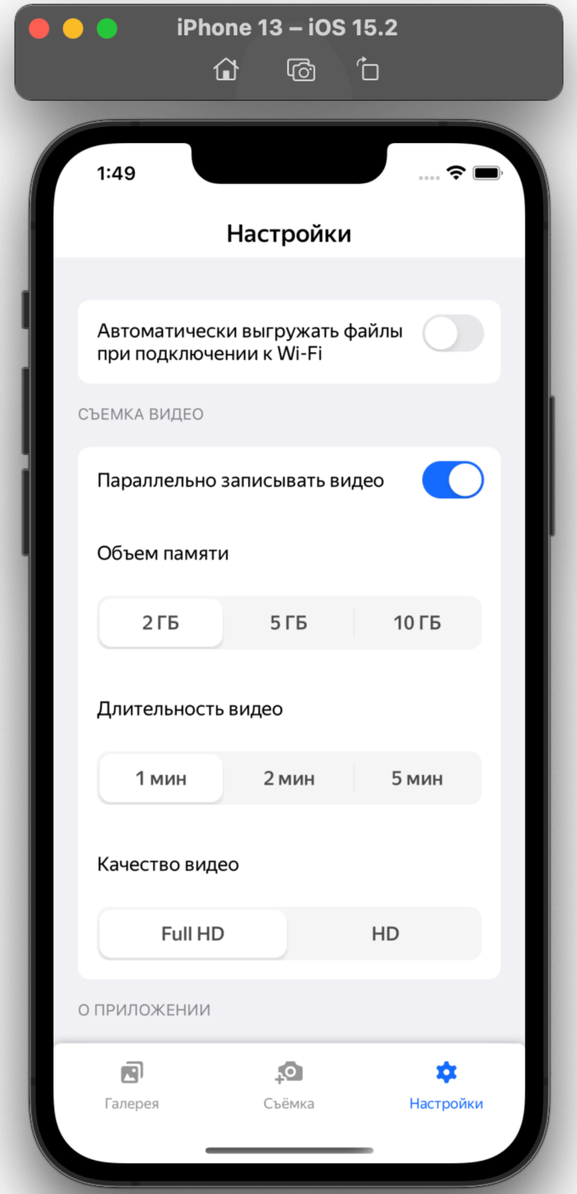
CommonViewWrapper

```
29 public struct LoginView<Model: LoginViewModel>: View {
30     @ObservedObject public var model: Model
31
32     public init(model: Model) {
33         self.model = model
34
35         loginButtonModel = GeneralButtonViewModel.makeLarge(
36             style: .primary, text: model.loginButtonText)
37         loginButtonModel.onTap = model.onLogIn
38     }
39
40     public var body: some View {
41         VStack(alignment: .center) {
42             Image(uiImage: KartographImages.login)
43                 .padding(.bottom, 32)
44             VStack(alignment: .leading, spacing: 16) {
45                 Text(model.title)
46                     .font(Fonts.medium(size: 28).font)
47                     .foregroundColor(RawColors.text.primary.color)
48                 Text(model.subtitle)
49                     .font(Fonts.regular(size: 18).font)
50                     .foregroundColor(RawColors.text.primaryVariant.color)
51             }
52             CommonViewWrapper(model: loginButtonModel)
53         }
54         .padding(.horizontal, 32)
55     }
56
57     private let loginButtonModel: GeneralButtonViewModel
58 }
59
```



List - экран настроек

```
List {
    Section {
        CommonViewWrapper(model: model.autoUpload)
    }

    Section(header: Text("Съемка видео")) {
        CommonViewWrapper(model: model.captureVideo)
        CommonViewWrapper(model: model.memoryMaxUsage)
        CommonViewWrapper(model: model.duration)
        CommonViewWrapper(model: model.quality)
    }

    if model.isDebugPanelAvailable {
         Section(header: Text("Debug")) {
              Button(action: model.onDebug, label: { debug })
        }
    }

    Section {
        Button(action: model.onLogout, label: { logout })
    }
}
```



AndroidView в Composable

AndroidView в Compose

```
@Composable
fun <T : View> AndroidView(
    factory: (Context) -> T,
    modifier: Modifier = Modifier,
    update: (T) -> Unit = NoOpUpdate
) {
```

AndroidView в Compose

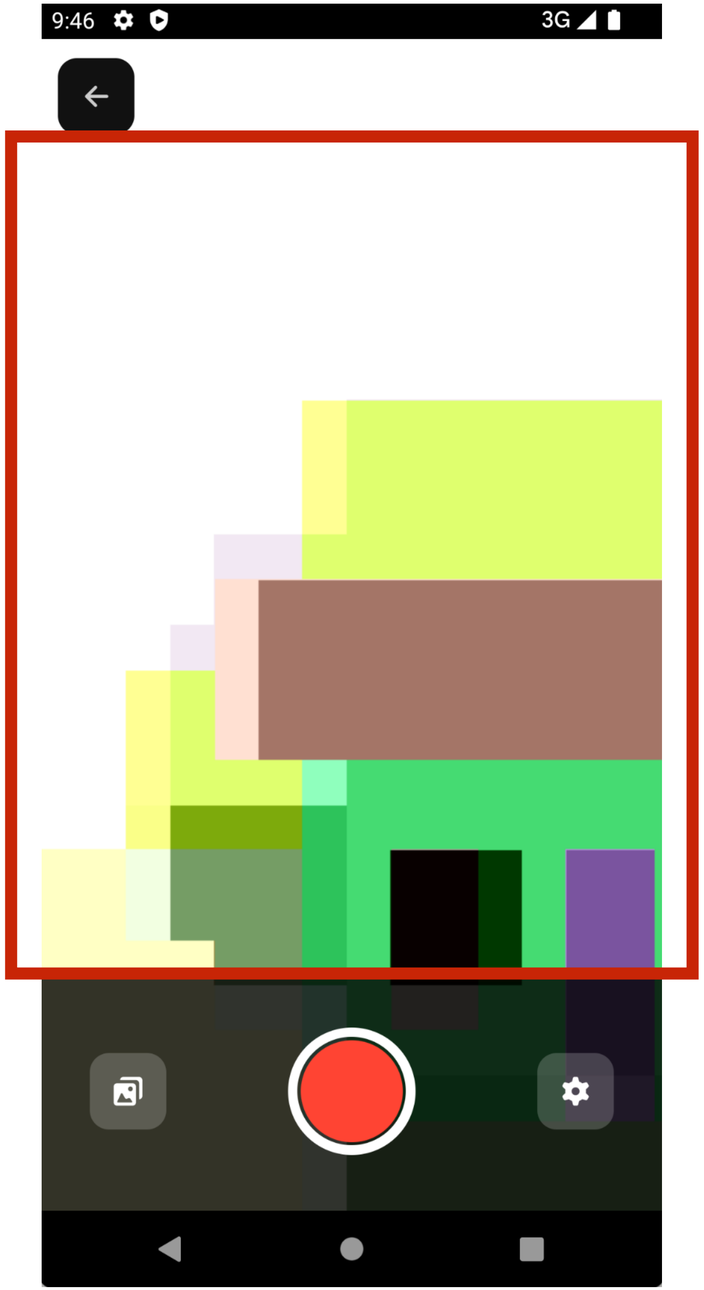
```
@Composable
fun <T : View> AndroidView(
    factory: (Context) -> T,
    modifier: Modifier = Modifier,
    update: (T) -> Unit = NoOpUpdate
) {
```

AndroidView в Compose

```
@Composable
fun <T : View> AndroidView(
    factory: (Context) -> T,
    modifier: Modifier = Modifier,
    update: (T) -> Unit = NoOpUpdate
) {
```

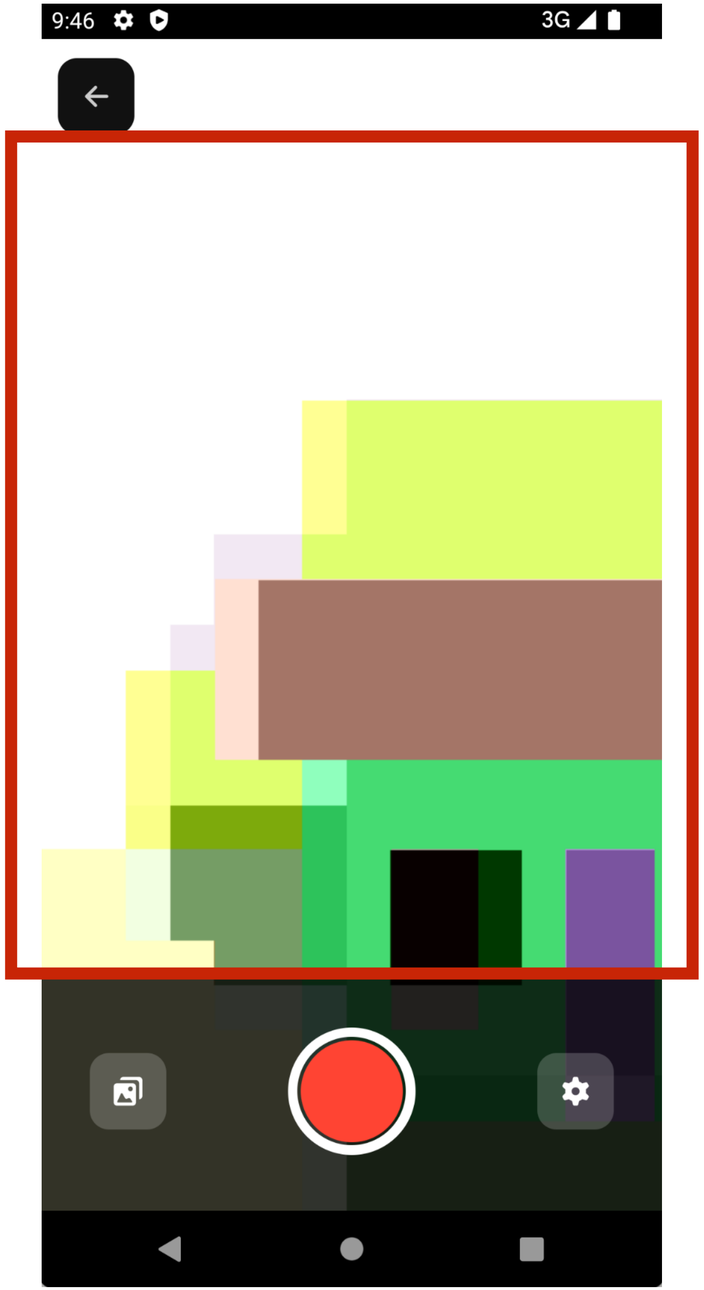
AndroidView в Compose

```
@Composable
internal fun CameraPreview(
    cameraProviderFuture: ListenableFuture<ProcessCameraProvider>,
    cameraReadyCallback: (PreviewView) -> Unit
) {
    AndroidView(
        factory = { context ->
            PreviewView(context).apply {  this: PreviewView
                layoutParams = FrameLayout.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT,
                    ViewGroup.LayoutParams.MATCH_PARENT
                )
                scaleType = PreviewView.ScaleType.FILL_START
                implementationMode = PreviewView.ImplementationMode.COMPATIBLE
                post {
                    cameraProviderFuture.addListener(
                        { cameraReadyCallback(this) },
                        ContextCompat.getMainExecutor(context)
                    )
                }
            }
        }
    )
}
```



AndroidView в Compose

```
@Composable
internal fun CameraPreview(
    cameraProviderFuture: ListenableFuture<ProcessCameraProvider>,
    cameraReadyCallback: (PreviewView) -> Unit
) {
    AndroidView(
        factory = { context ->
            PreviewView(context).apply { this: PreviewView
                layoutParams = FrameLayout.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT,
                    ViewGroup.LayoutParams.MATCH_PARENT
                )
                scaleType = PreviewView.ScaleType.FILL_START
                implementationMode = PreviewView.ImplementationMode.COMPATIBLE
                post {
                    cameraProviderFuture.addListener(
                        { cameraReadyCallback(this) },
                        ContextCompat.getMainExecutor(context)
                    )
                }
            }
        }
    )
}
```



Тонкости

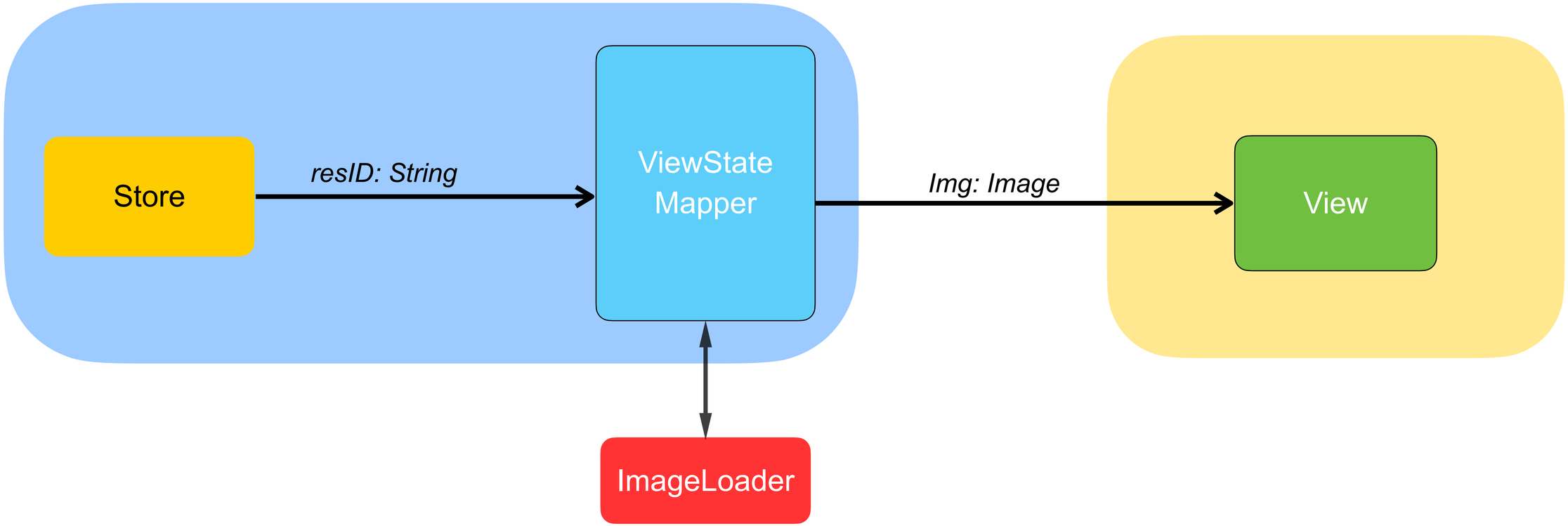
Previews

Pods 〉 Development Pods 〉 KartographUI 〉 Screens 〉 AppOnboarding 〉 AppOnboardingScreen 〉 No Selection

```
182 }
183
184 // MARK: - Previews
185
186 private extension AppOnboardingScreenModel {
187     static var mock: AppOnboardingScreenModel {
188         let pageInfos: [OnboardingPageInfo] = [
189             OnboardingPageInfo(
190                 image: RawImages.kartographOnboarding_01.day,
191                 text: "Народный Картограф это сервис, где водители улучшают карты
                        и навигацию"
192             ),
193             OnboardingPageInfo(
194                 image: RawImages.kartographOnboarding_02.day,
195                 text: "Снимайте дорогу из машины, номера и лица людей будут
                        размыты"
196             ),
197             OnboardingPageInfo(
198                 image: RawImages.kartographOnboarding_03.day,
199                 text: "Нейросеть распознает на снимках разметку и знаки и
                        переносит их на карты"
200             ),
201             OnboardingPageInfo(
202                 image: RawImages.kartographOnboarding_04.day,
203                 text: "А еще вы можете пользоваться приложением как
                        видеорегистратором"
204             )
205         ]
206
207         return AppOnboardingScreenModel(pageInfos: pageInfos, buttonText:
                "Начать", onButton: {})
208     }
209 }
210
211 struct AppOnboardingScreen_Previews: PreviewProvider {
212     static var previews: some View {
213         AppOnboardingScreen(model: AppOnboardingScreenModel.mock)
214     }
215 }
216
```

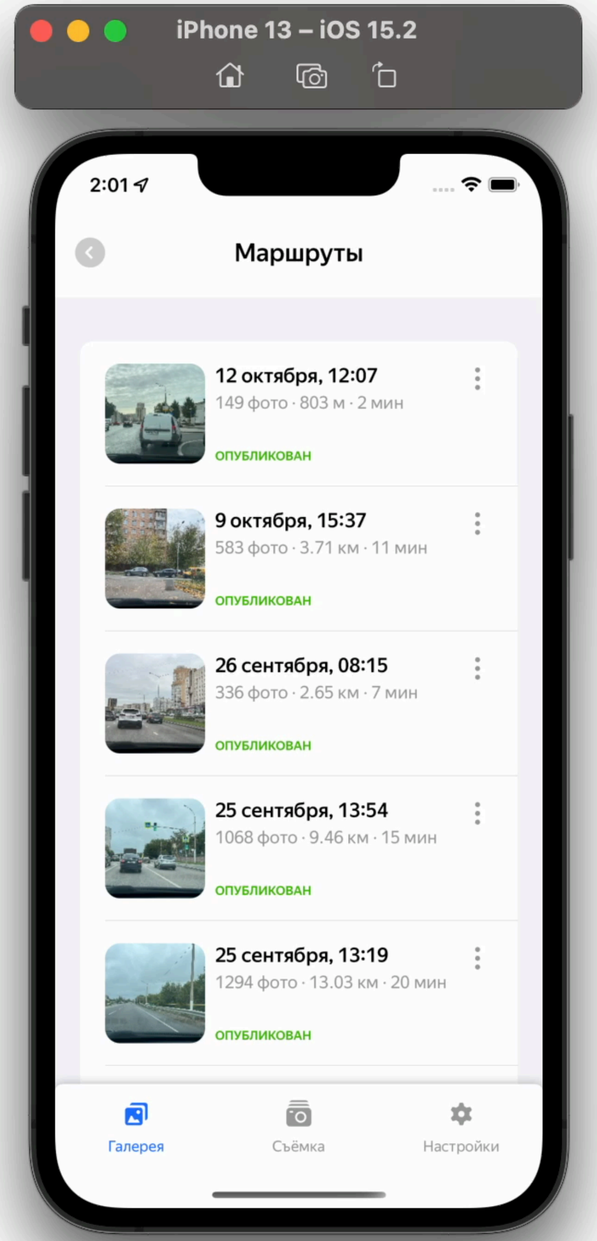
Загрузка асинхронных данных

〉 Сначала сделали загрузку в мультиплатформе - ❌



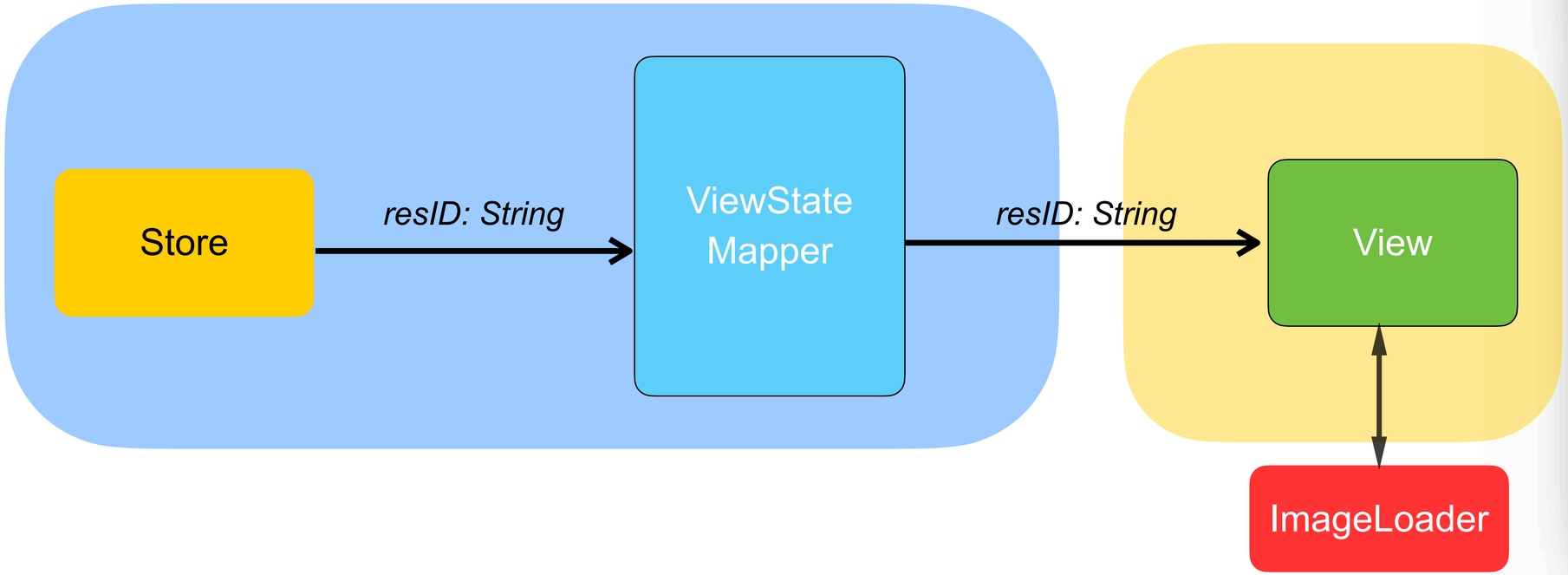
Загрузка асинхронных данных

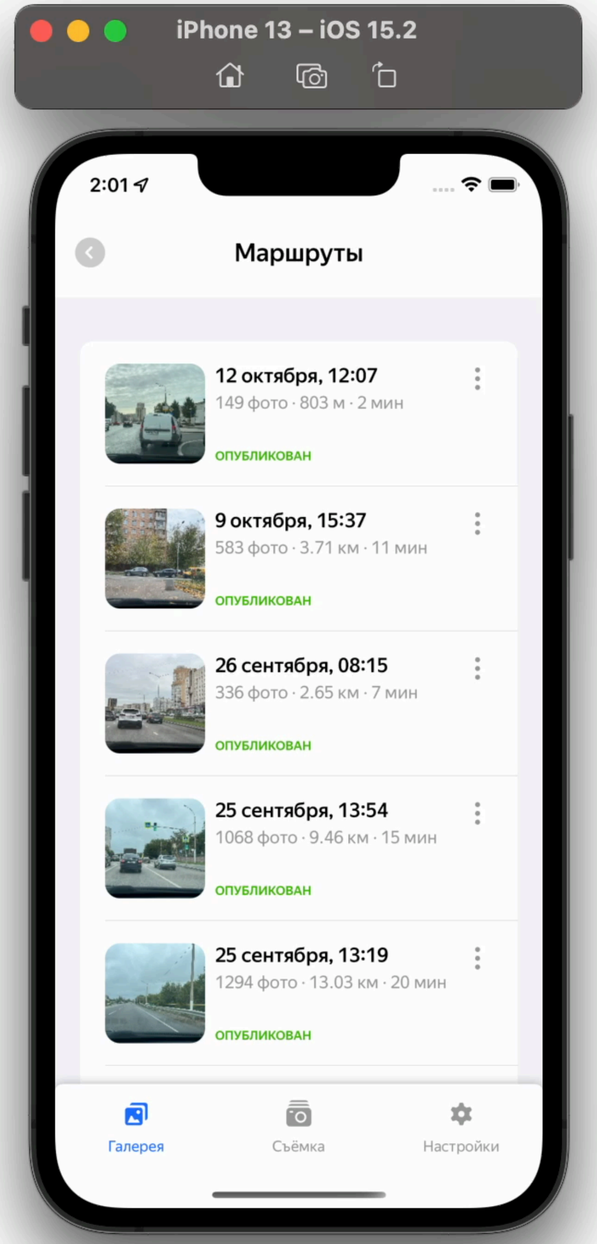
- Сложно контролировать загрузки
- Потери в производительности, если много загрузок на одном экране



Загрузка асинхронных данных

〉 Загрузка данных инициируется и останавливается из Вью - ✅

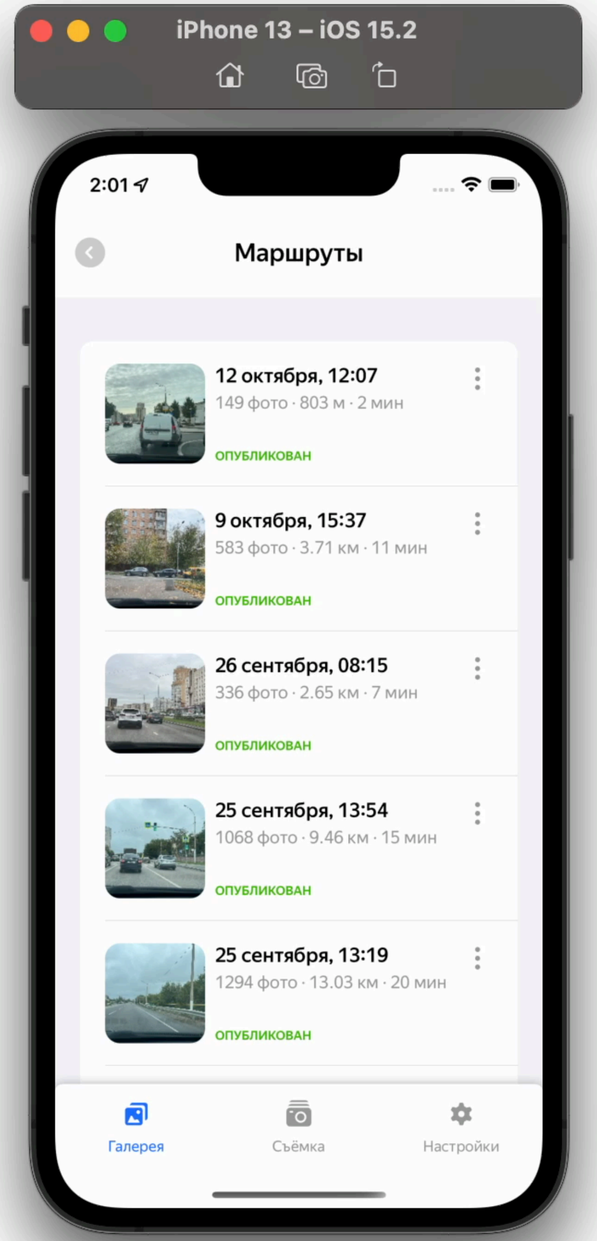




Загрузка асинхронных данных

〉 Загрузка данных инициируется и останавливается из Вью - ✅

```
public var body: some View {
    content(loader.asyncImagePhase)
        .onAppear { loader.loadImage() }
        .onDisappear { loader.cancelDownload() }
}
```



SwiftUI Binding and Redux

〉Можно использовать в Redux стандартные Read/Write UI компоненты, такие как Toggle, если использовать кастомный Binding:

```
let binding = Binding<Bool>(
    get: { viewState.autoUploadValue },
    set: { _ in model.dispatch(action: .setAutoUpload(!viewState.autoUploadValue)) }
)

let toggle = Toggle(viewState.autoUploadTitle, isOn: binding)
```

Лямбда-функции в Jetpack Compose

〉Лучше не создавать лямбды, а использовать ссылки на них (где возможно)

```
SettingsScreen.View(
    state = SettingsScreen.State(state),
    dispatch = interactor::dispatch
)
```

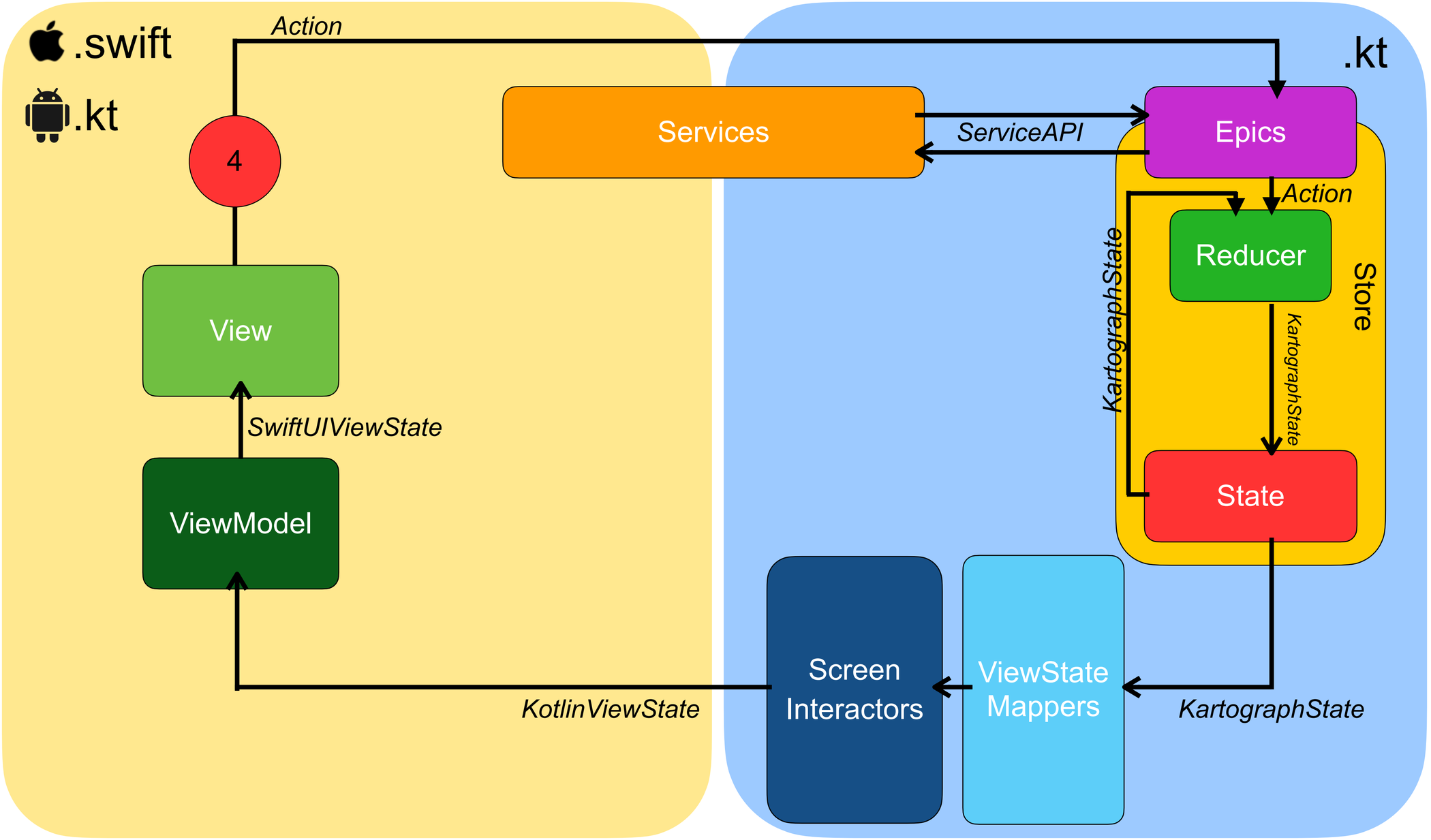
```
TabsScreen.View(
    state = state,
    onGalleryTabClick = {
        interactor.dispatch(TabsScreenAction.SelectGallery)
    },
```

Данные из другого модуля в Jetpack Compose

〉Оборачиваем данные из другого модуля и помечаем аннотацией @Immutable

```
@Immutable
data class State(
    val viewState: MainScreenViewState,
)
```

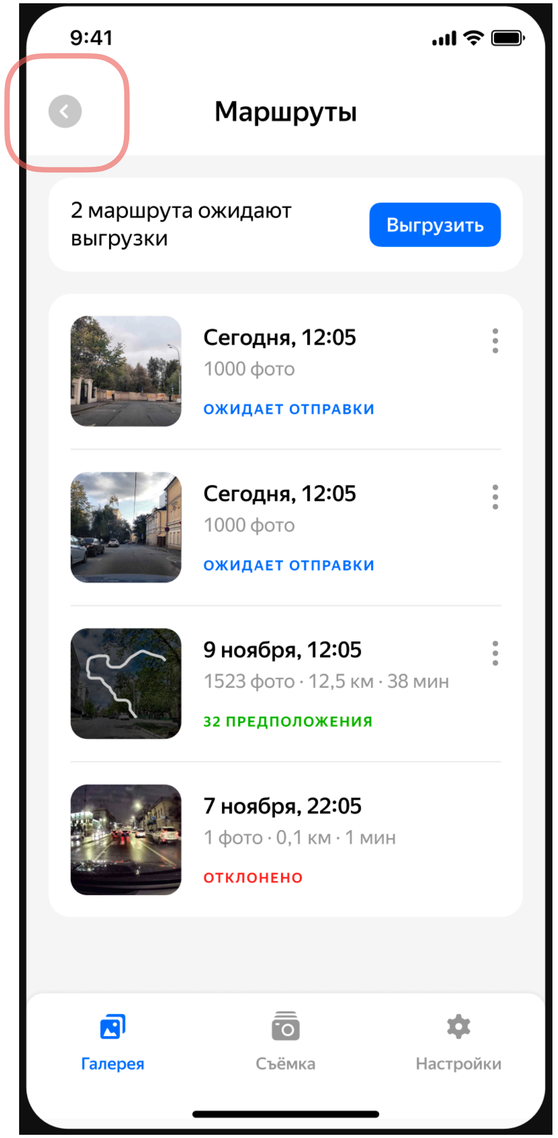
Kartograph Redux



Отправка событий

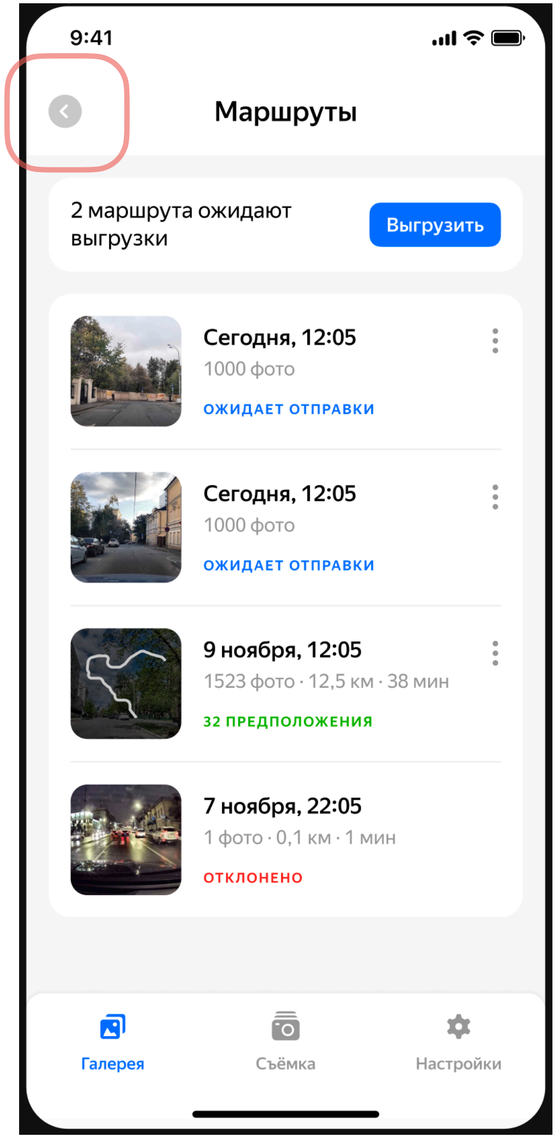
Actions

```
class GalleryActionsHandlerImpl: GalleryActionsHandler {

    init(interactor: GalleryScreenInteractor) {
        self.interactor = interactor
    }

    func onBack() {
        interactor.dispatch(GalleryScreenAction.GalleryGoBack())
    }

    private let interactor: GalleryScreenInteractor
}
```



Actions

```
class GalleryActionsHandlerImpl: GalleryActionsHandler {

    init(interactor: GalleryScreenInteractor) {
        self.interactor = interactor
    }

    func onBack() {
        interactor.dispatch(GalleryScreenAction.GalleryGoBack())
    }

    private let interactor: GalleryScreenInteractor
}

. . .

Button(
    action: model.actionsHandler.onBack,
    label: { BackButtonView() }
)
```



Автологирование

Логирование

〉 События разделены на 3 вложенные категории:
- все события
- логируемые
- пользовательские

〉 Все события - наследники одного sealed класса.

〉 Все логируемые события - сериализуемые.

```
// Base actions
interface KartographAction : Action, AutoParcelable
@Serializable
sealed class Loggable : KartographAction
@Serializable
sealed class UserAction : Loggable()
```

Пользовательские события

〉 В интерактарах используем только UserAction: так события точно залогируются

```
interface GalleryScreenInteractor {
    fun viewStates(): PlatformFlow<GalleryScreenViewState>
    fun dispatch(galleryAction: UserAction)
}
```

Технические события

〉 Пришлось придумать новые события

Изменение навигационного стека

```
@Serializable
data class LogNavigationState(
    val screens: List<ScreenId>,
    val tab: TabId? = null,
    val dialog: DialogId? = null
) : Loggable()
```

Итог по автологированию

- Изменения в коде -> изменения в логировании, не надо поддерживать
- Сразу на обеих платформах
- Точно знаем, что все пользовательские события будут залогированы
- Под технические данные можно явно добавлять события
- Аналитики легко могут распарсить данные

Выводы по SwiftUI / Compose

Выводы

Плюсы:

- Быстрое описание UI
- Критически меньше кода на платформах
- Унификация кода iOS/Android.
- Отображение внутри UIKit / AndroidView и наоборот
- Превью
- Анимации SwiftUI

Выводы

Точки роста:

〉 Данные из мультиплатформы приходится дублировать
〉 Превью в Compose - иногда ломается
〉 Анимации в Compose - нужно исследовать / ждать новых версий библиотеки. Иногда неожидаемо работают не так
〉 Не каждый экран получится написать

Яндекс

Спасибо за внимание!

potap-yur@yandex-team.ru
https://github.com/yury-potapov