Используем State Machine на SwiftUI и Combine

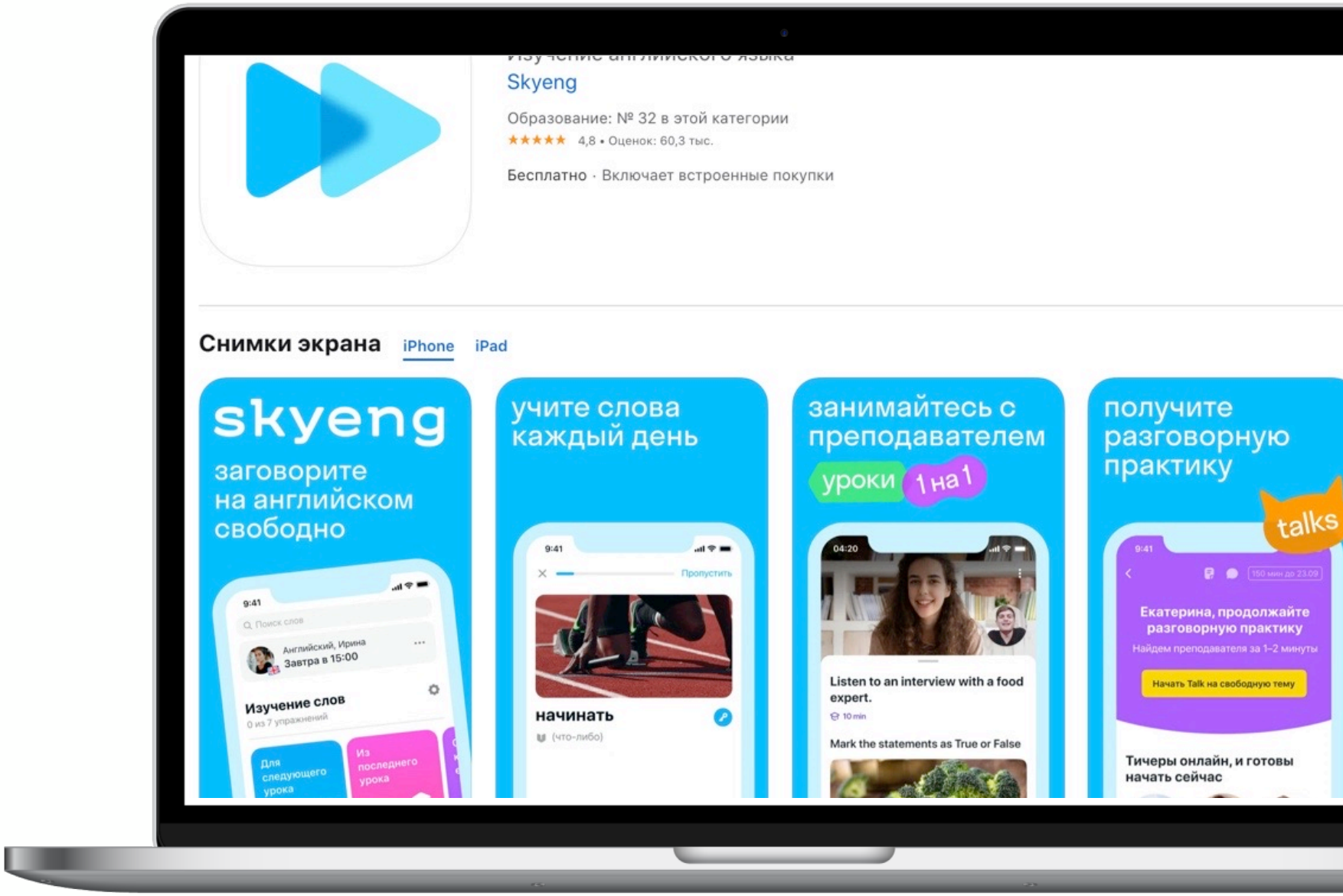
быстрый, простой и удобный способ создания приложений

skyeng

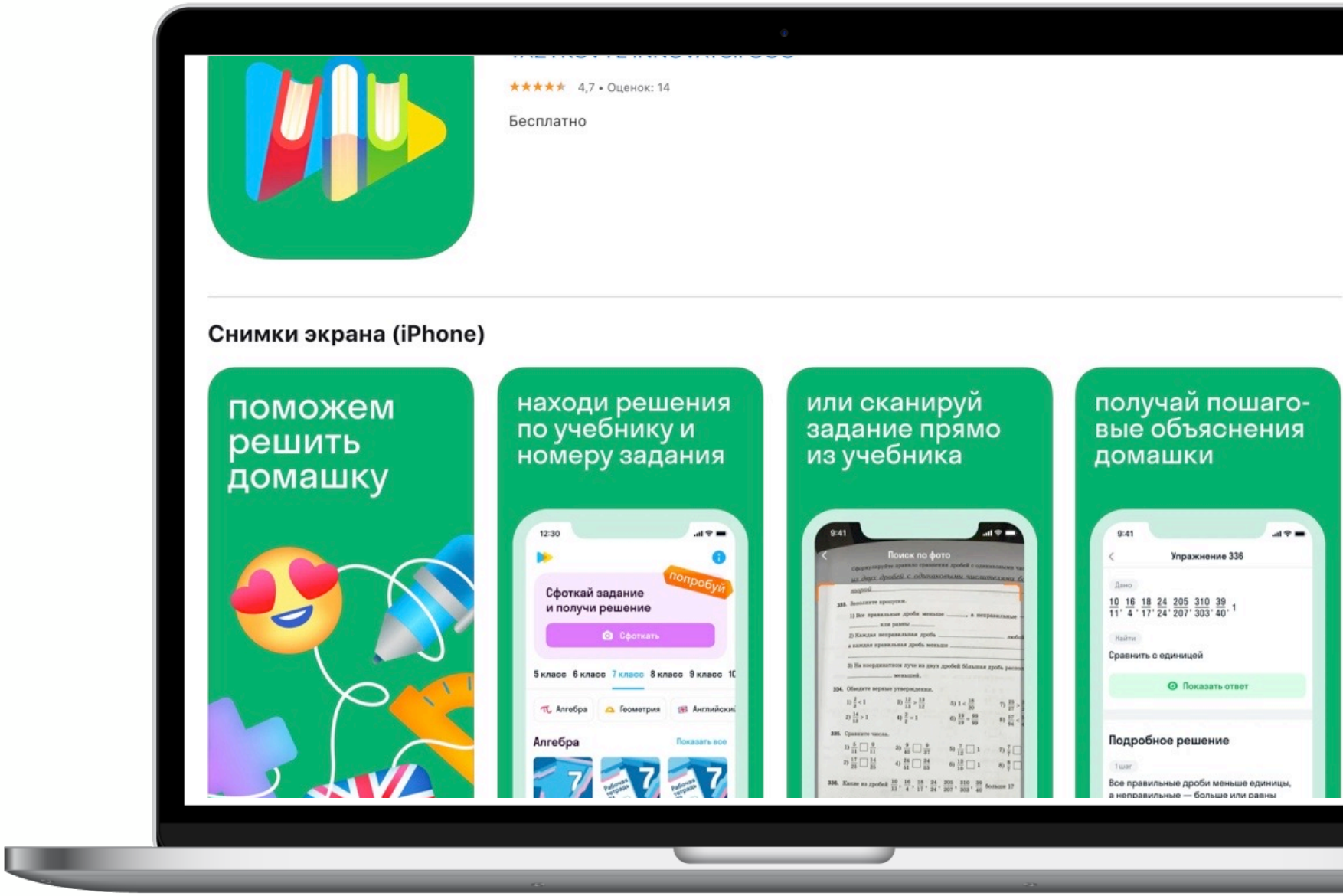
План

- Про мобильную разработку Skyeng
- Мой путь к SwiftUI, Combine, State Machine
- Коротко про MVVM и почему он хорошо подходит для приложений на SwiftUI + Combine
- Что такое State Machine, какие проблемы решает этот подход, как документировать состояния
- Примеры и детали реализации MVVM с State Machine на SwiftUI и Combine
- Интересные моменты SwiftUI, с которыми столкнулся
- Навигация в SwiftUI, координаторы
- Материалы и библиотеки

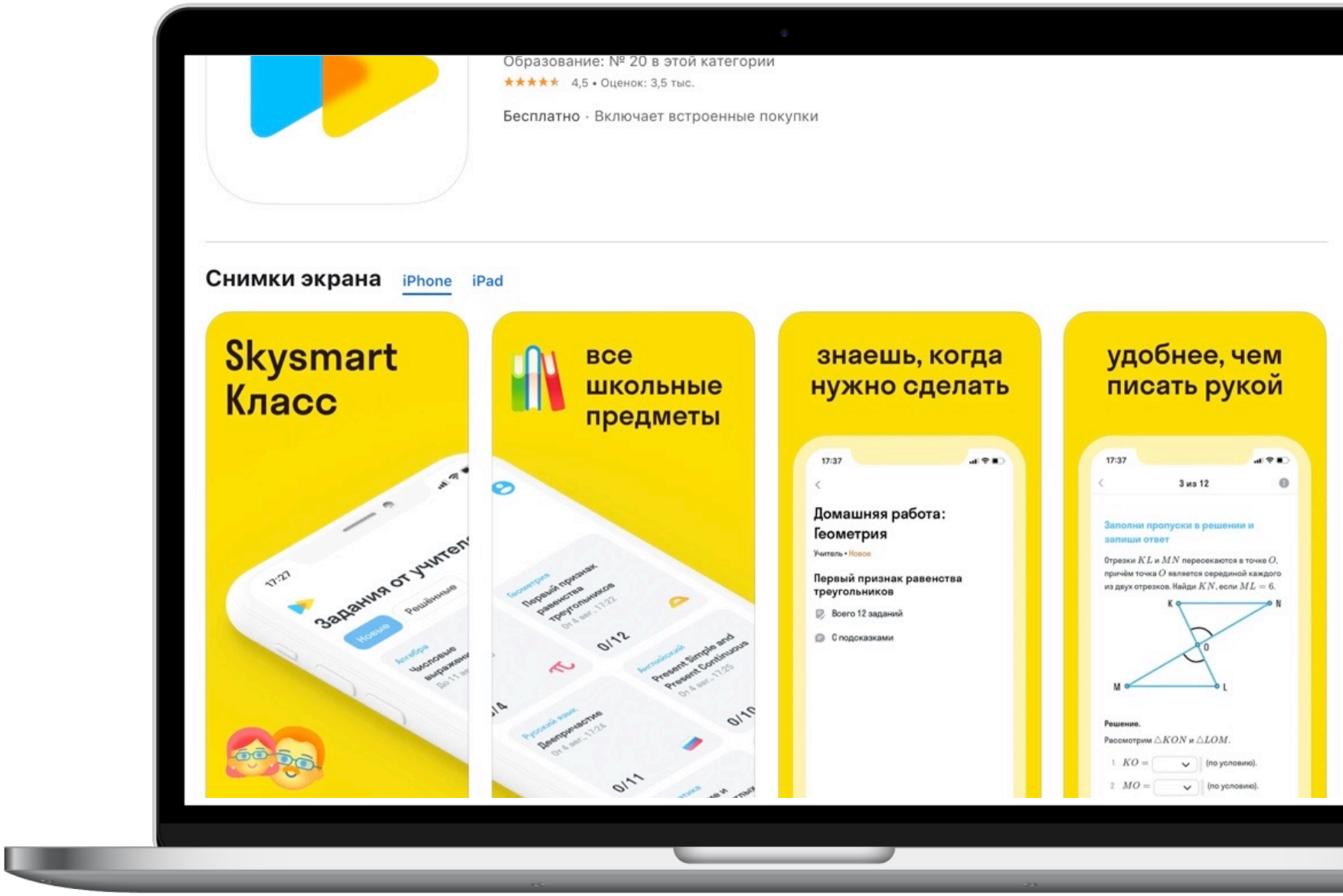
Немного про мобильную разработку в Skyeng



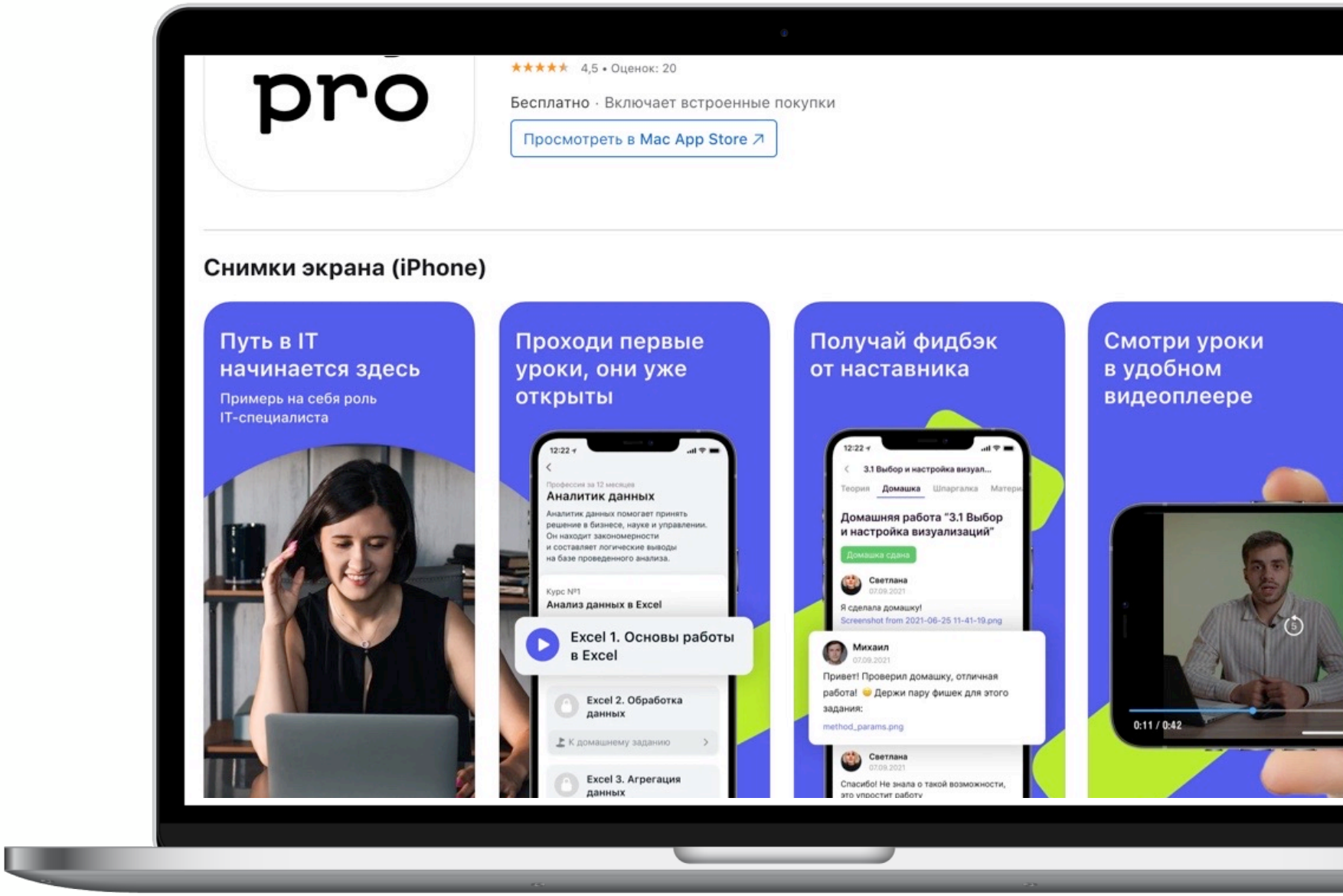
Немного про мобильную разработку в Skyeng



Немного про мобильную разработку в Skyeng



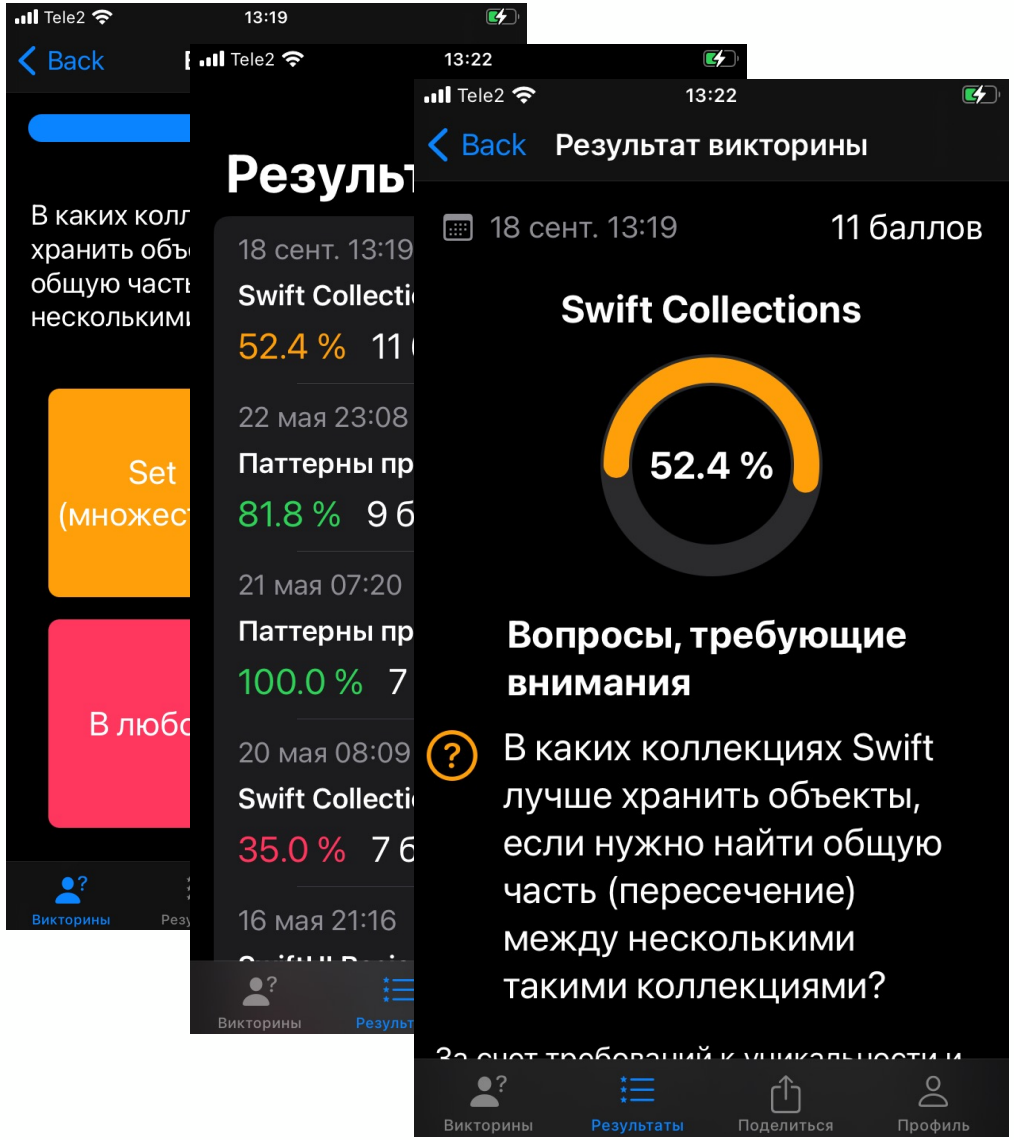
Немного про мобильную разработку в Skyeng



С чего все начиналось?

- В компании, где я работал, нужно было сделать приложение - образовательный инструмент с определенным набором фич
- При этом хотелось проверить новый iOS стек SwiftUI и Combine и понять, как можно начать его постепенно интегрировать в нашу кодовую базу

Приложение для викторин



MVVM

skyeng

Если коротко, то MVVM это

View

- Показывает UI, может и с анимацией
- Передает пользовательский ввод в ViewModel

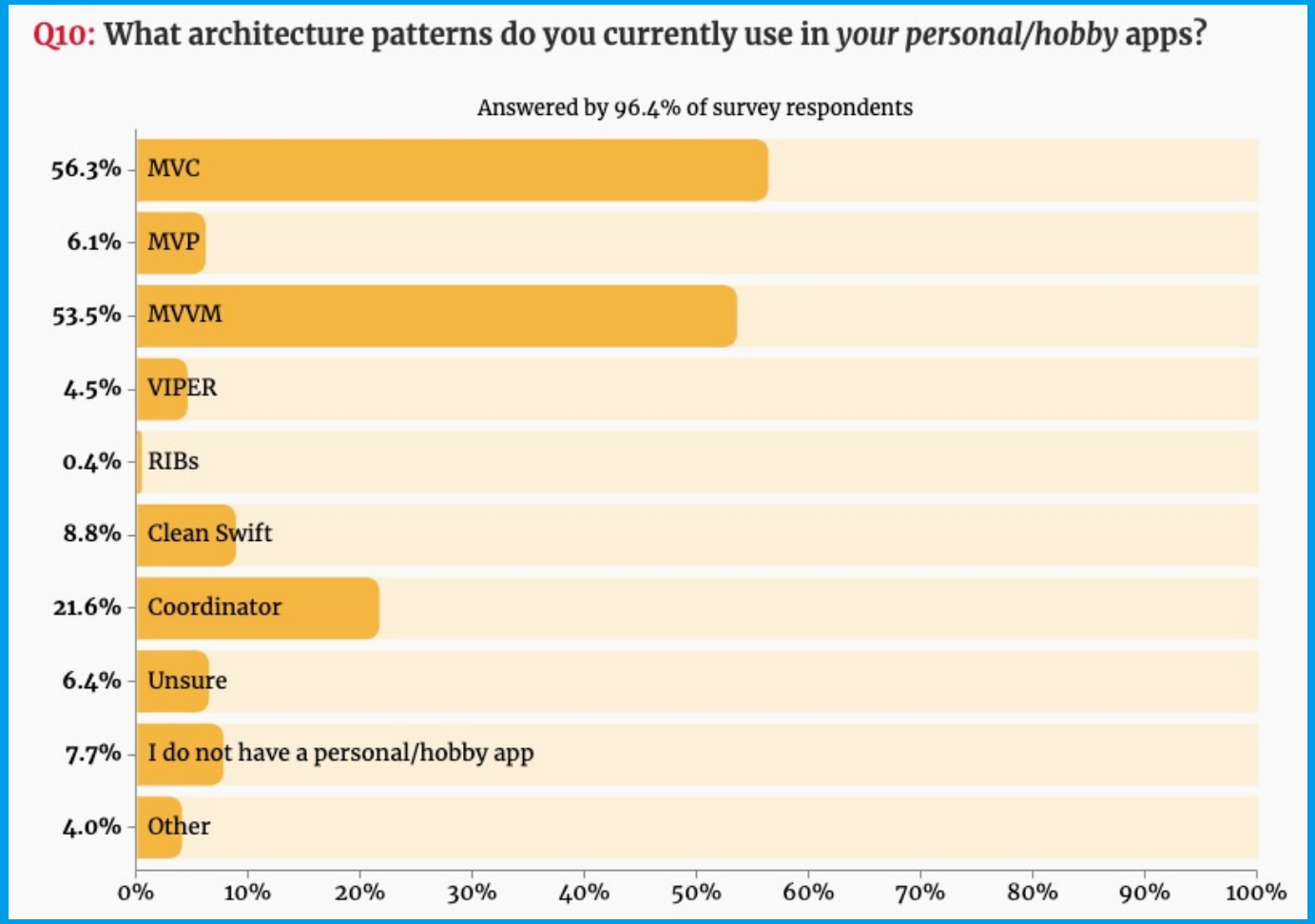
ViewModel

- Управляет поведением UI и хранит состояние
- Транслирует действия пользователя в правила бизнес логики
- Готовит данные модели для показа пользователю
- Может ничего не знать про UI Framework (UIKit, SwiftUI)

Model

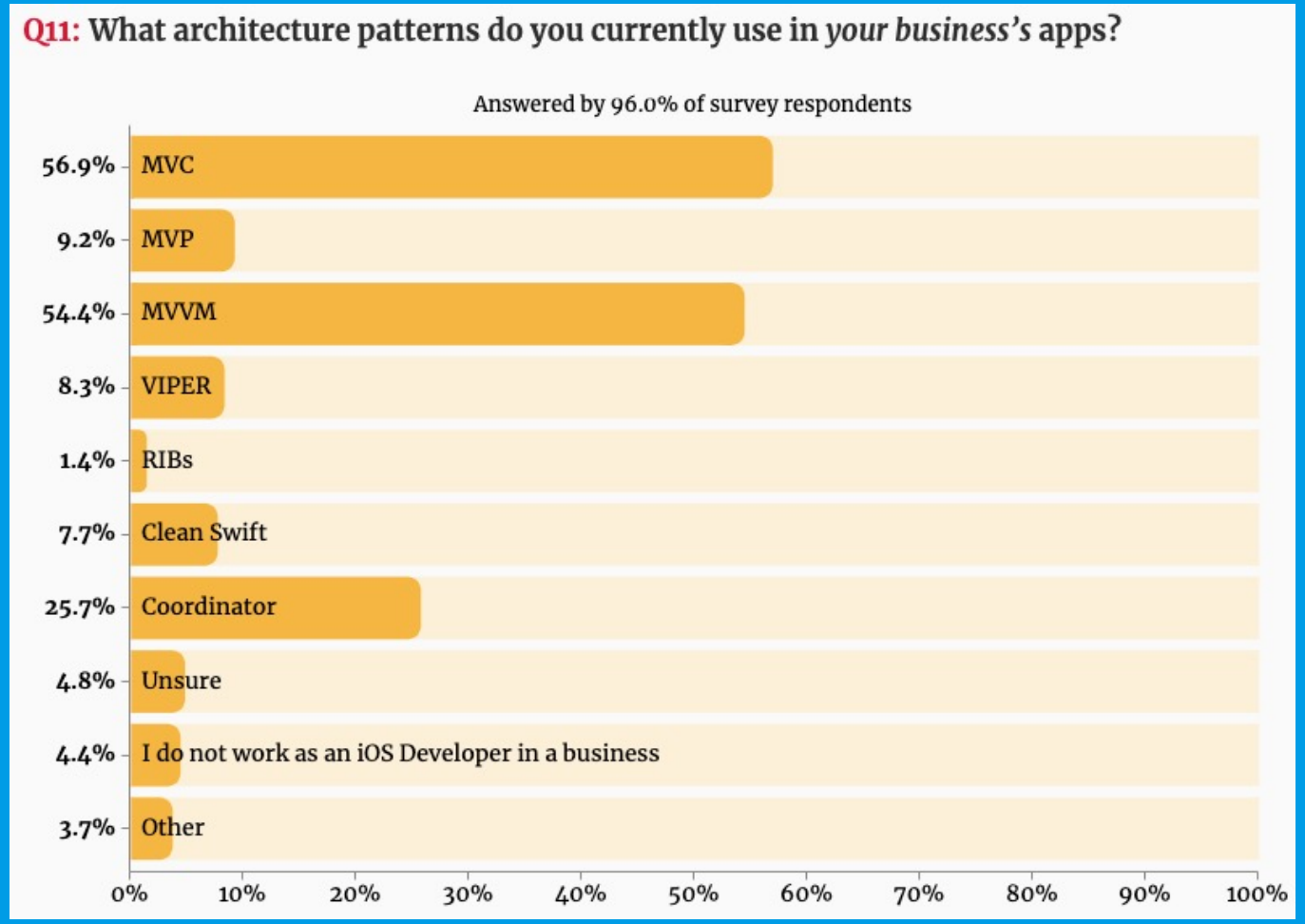
- Бизнес логика
- Данные

MVVM и iOS



данные опроса iOS Dev Survey 2020

MVVM и iOS



данные опроса iOS Dev Survey 2020

Недостаток MVVM в iOS:

Отсутствие нативного инструмента для работы с биндингами **приводило** к непростому выбору

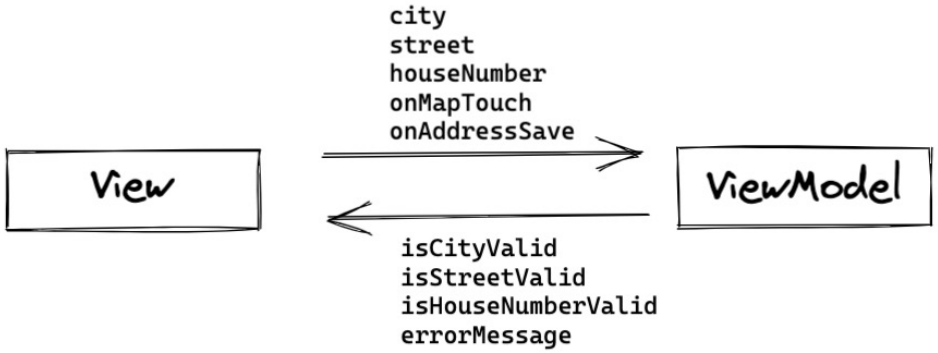
- Использовать callback вместо биндингов
- Создавать свои обертки поверх KVO или использовать готовые, но малоизвестные: SwiftBond, RZDataBinding и т.д.
- Добавлять в проект крупную внешнюю зависимость (RxSwift, ReactiveCocoa), а это
  - повышает порог входа в проект
  - сужает выбор кандидатов при найме (хотят и могут в Rx)

Привет, SwiftUI и Combine!
Часть проблем позади!

Пример

Проблема

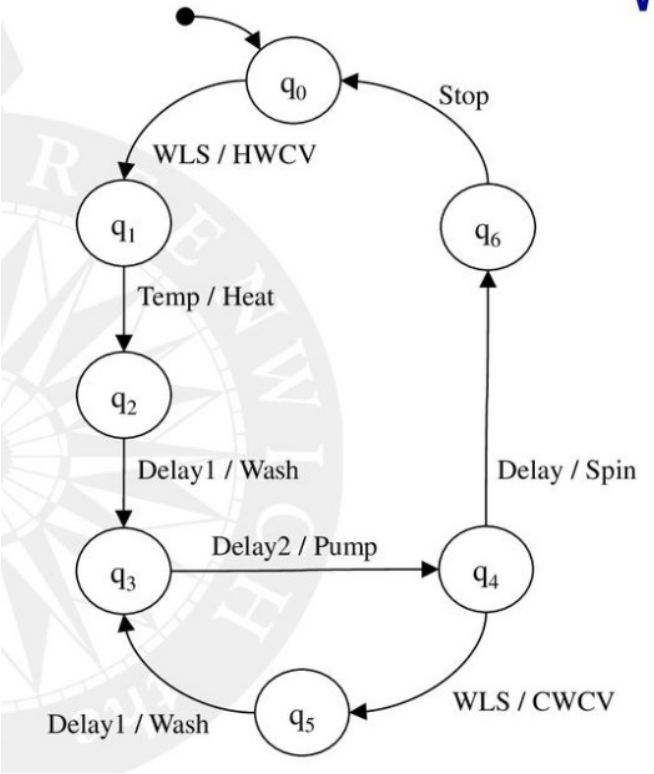
- Большое число состояний, в котором может оказаться UI, часть из которых вообще не нужна
- Добавление каждого нового состояния еще больше увеличивает сложность
- Их сложнее не только обрабатывать, но и проверять



Идея

Если проблема в том, что состояний много, может ограничить их количество и оставить только те, что нужны?

State Machine живет в каждом доме



Washing Machine FSM

State	
q_0	Cold and Empty
q_1	Full with Hot Water
q_2	Water Heated to Correct Temperature
q_3	Drum Agitating
q_4	Pumping
q_5	Full with Cold Water
q_6	Spinning

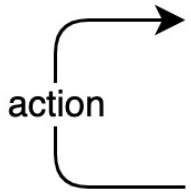
Для решения используем State Machine (Finite State Machine)

- State Machine может быть только в одном из наперёд заданных состояний
- Смена состояния происходит как реакция на какие-то внешние действия
- Чтобы определить FSM нужны: список состояний, начальное состояние и действия для перехода в каждое состояние

Как описать State Machine?

начало

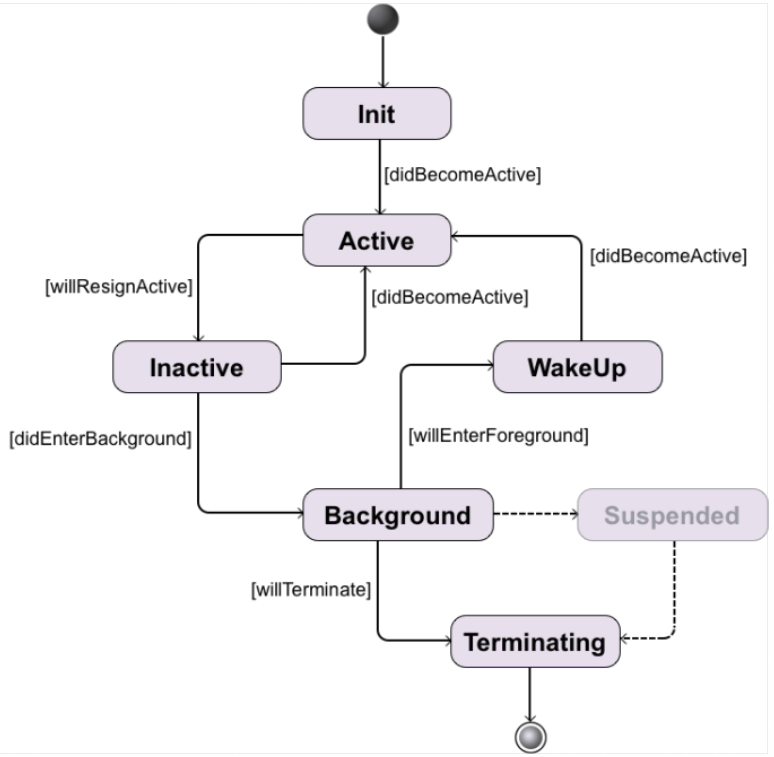
состояние

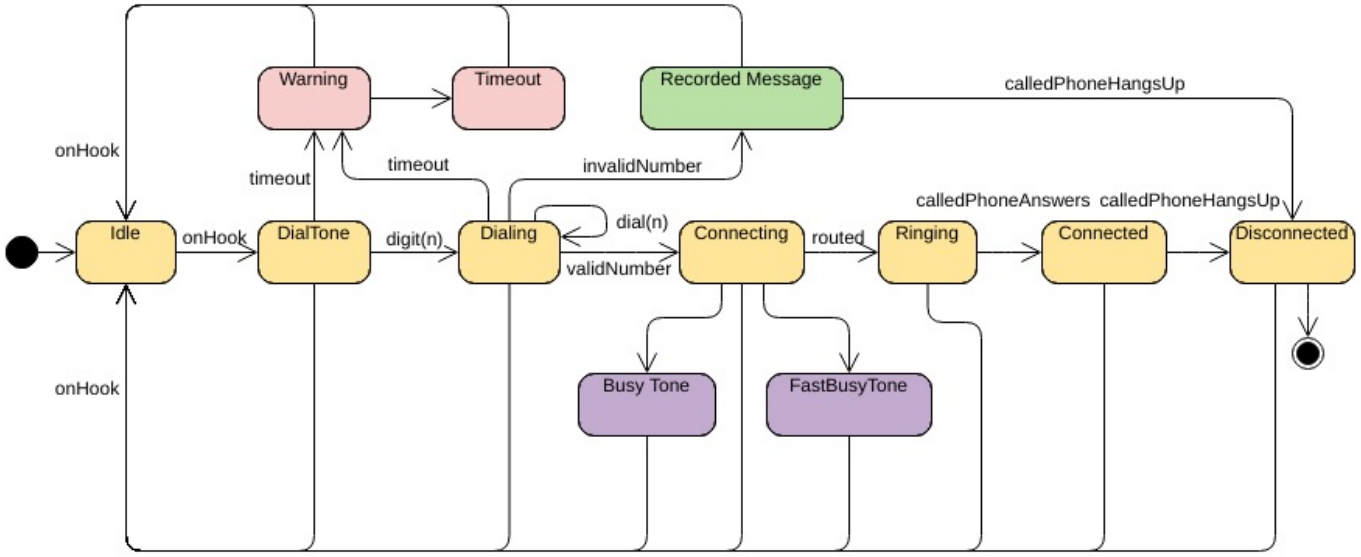


действия для смены состояний

конец

Примеры State Machine

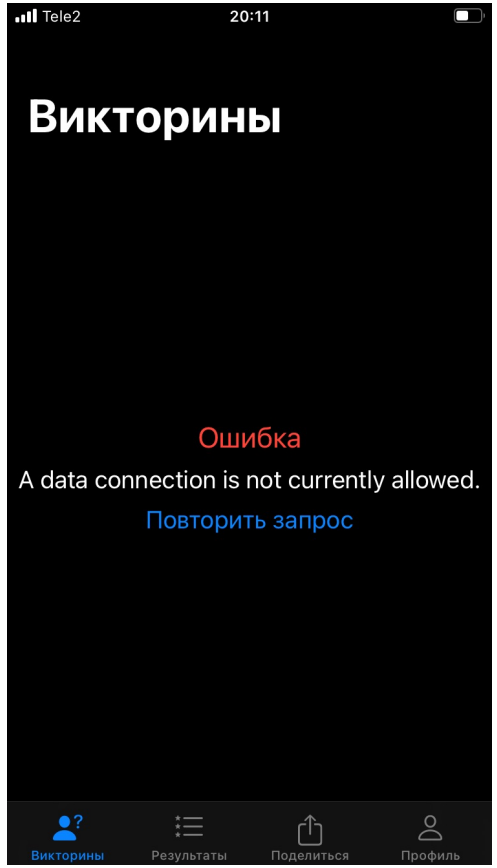


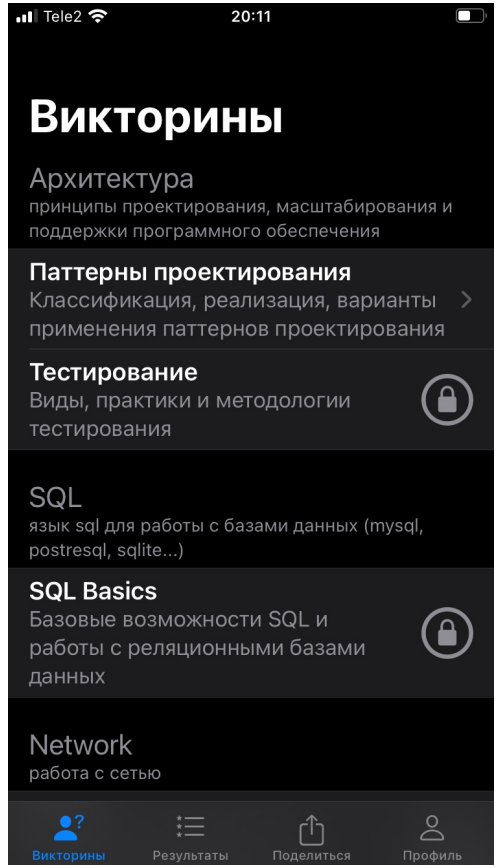


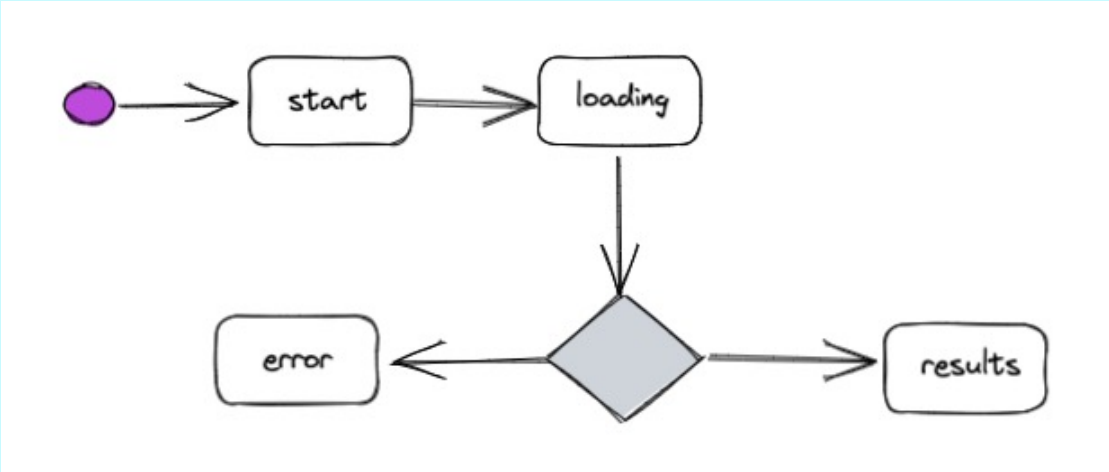
Примеры реализации

skyeng

Пример 1: список викторин








```swift
class QuizesListViewModel: ObservableObject {

    @Published private(set) var state = State.start
    ...

    enum State {
        case start
        case loading
        case results([Quiz])
        case error(Error)
    }
    ...

    enum Event {
        case onAppear
        case onQuizesLoaded([Quiz])
        case onQuizSelect(Int)
        case onFailedToLoadQuizes(Error)
    }
}
```

```swift
extension QuizesListViewModel {

    static func reduce(_ state: State, _ event: Event) -> State {

        switch state {

          case .start:
              switch event {
                case .onAppear:
                    return .loading
                default:
                    return state
              }

          case .loading:
              switch event {
                case .onFailedToLoadQuizes(let error):
                    return .error(error)
                case .onQuizesLoaded(let quizes):
                    return .loaded(quizes)
                default:
                    return state
              }

          case .loaded:
              return state

          case .error:
              return state
        }
    }

}
```

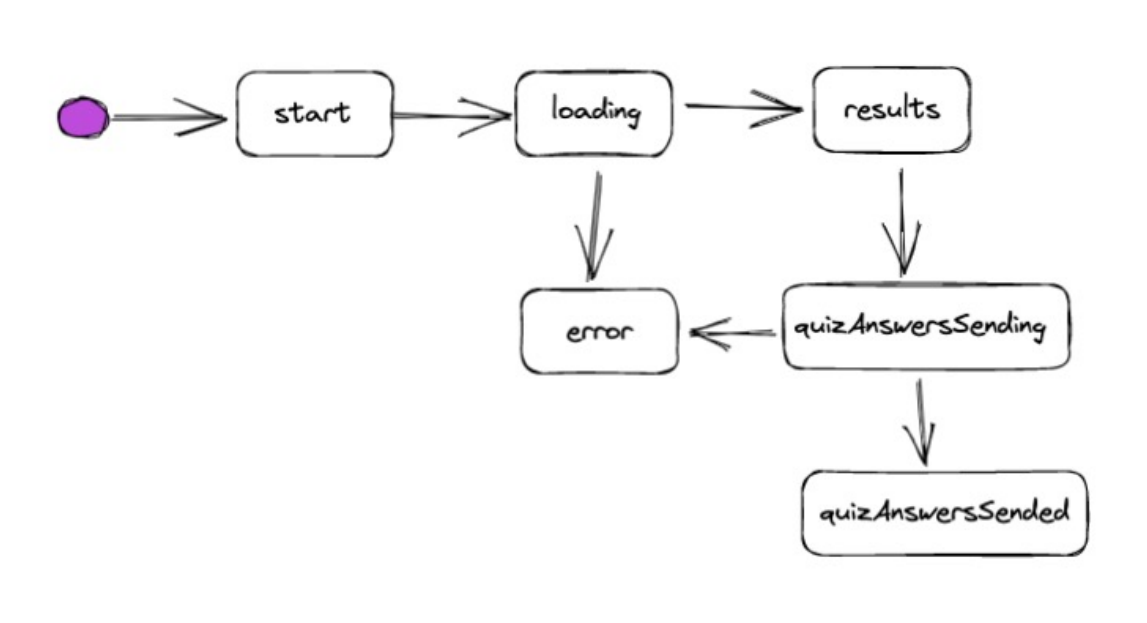
skyeng

Пример 2: экран викторины

- Викторина состоит из набора вопросов
- В каждом вопросе есть 4 варианта ответа
- У каждого вопроса есть свой лимит времени на ответ
- При выборе ответа на вопрос или по истечении времени на ответ, открывается следующий вопрос
- Если вопросы закончились, ответы отправляются на сервер, пользователю предлагают перейти в результаты

```swift
enum QuizViewModelState {
    case start, loading, results
    case noQuestionsLeft
    case quizAnswersSending
    case quizAnswersSended
    case error(_ errorMessage: String)
}
```

```swift
enum QuizViewModelEvent {
    case onAppear
    case quizFetchingStarted
    case quizFetchingCompleted(_ withQuiz: Quiz)
    case questionAnswered(quiestionId: Int, answerId: Int)
    case quiestionTimeLimitExceeded(questionId: Int)
    case quizAnswersSendingStarted
    case quizAnswersSendingCompleted
    case onDisappear
    case error(_ errorMessage: String)
}
```



skyeng

```swift
class QuizViewModel: ObservableObject {

    @Published private(set) var state: QuizViewModelState = .start
    .....

    static func reduce(_ state: State, _ event: Event) -> State

      switch event {

        case .onAppear:
            fetchQuiz()
            return .start

        case .quizFetchingStarted:
            return .loading

        .....
      }

}
```

```swift
struct QuizView: View {

    @ObservedObject var quizViewModel: QuizViewModel

    ....

    var body: some View {
        VStack() {
            contentView()
        }
        .onAppear {
            quizViewModel.handleStateForEvent(.onAppear)
        }
        .onDisappear {
            quizViewModel.handleStateForEvent(.onDisappear)
        }
    }

    @ViewBuilder func contentView() -> some View {

        switch quizViewModel.state {

        case .loading:
            LoadingView()

        case .start:
            EmptyView()

        .....

    }

}
```

skyeng

Как менять View в зависимости от State?

```
class ViewModel: ObservableObject {

    @Published var state = State.start

    enum ViewState {
        case start
        ........
        case emptyState
    }
}


struct ContentView: View {

  @ObservedObject var viewModel: ViewModel

    var body: some View {

        VStack {

          //TODO: выбрать способ переключать View в зависимости от state

        }
    }
}
```

skyeng

Как менять View в зависимости от State?

```
struct ContentView: View {

    @ObservedObject var viewModel: ViewModel

    var body: some View {

        VStack {

          contentView

        }
    }

    private var contentView: some View {

        switch viewModel.state {

        case .loading:
            return Text("Loading")

        case .error:
            return Text("Error")

        default:
            return Text("....")
        }
    }
}
```

skyeng

Как менять View в зависимости от State?

```
struct ContentView: View {

    @ObservedObject var viewModel: ViewModel

    var body: some View {

        VStack {

          contentView

        }
    }

    private var contentView: some View {

        switch viewModel.state {

        case .loading:
            return Text("Loading")

        case .error:
            return Button("error detected, reload") {
                // some action to reload data
            }

        default:
            return Text("....")
        }
    }
}
```

Ошибка компиляции

Function declares an opaque return type, but the return statements in its body do not have matching underlying types

skyeng

Как менять View в зависимости от State?

```
struct ContentView: View {

    @ObservedObject var viewModel: ViewModel

    var body: some View {

        VStack {

          contentView

        }
    }

    private var contentView: some View {

        switch viewModel.state {

        case .loading:
            return AnyView(Text("Loading"))

        case .error:
            return AnyView(Button("error detected, reload") {
                // some action to reload data
            })

        default:
            return AnyView(Text("...."))
        }
    }
}
```

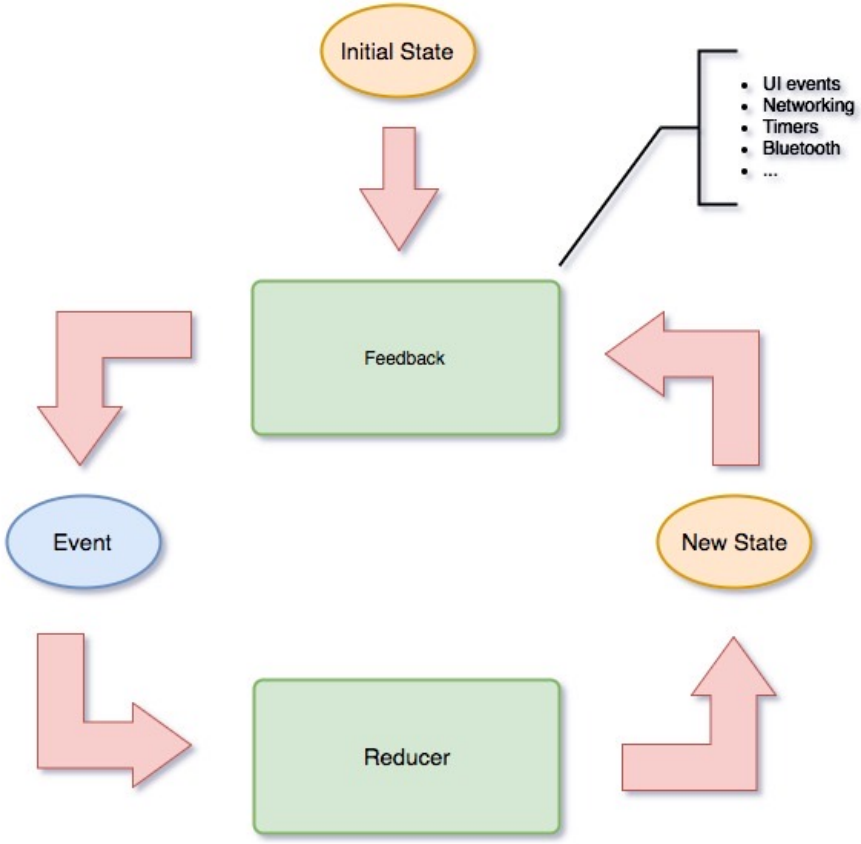
skyeng

Как менять View в зависимости от State?

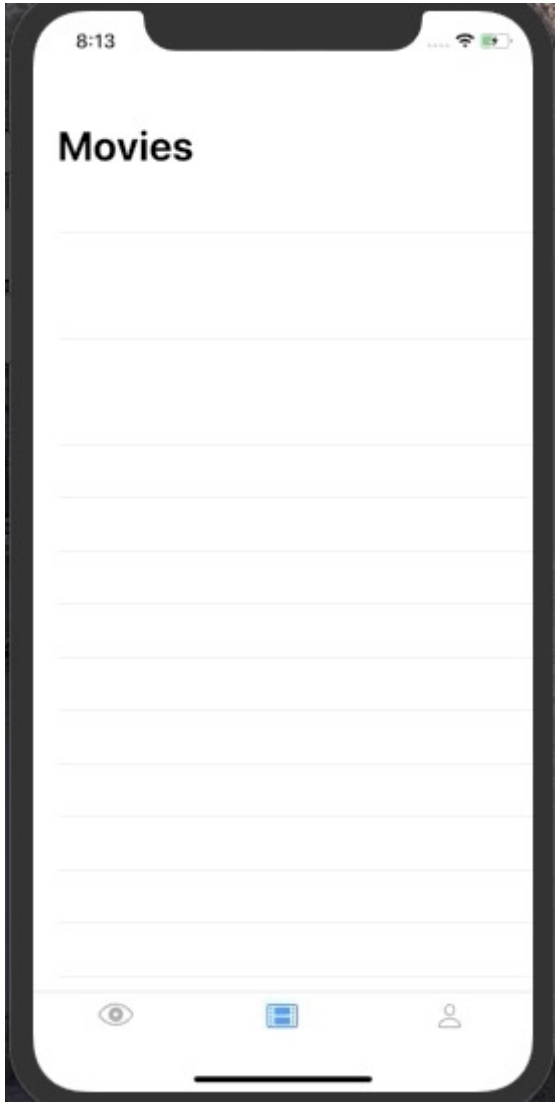
```
struct ContentView: View {

    @ObservedObject var viewModel: ViewModel

    var body: some View {

        VStack {

          contentView

        }
    }

    @ViewBuilder private var contentView: some View {

        switch viewModel.state {

        case .loading:
            Text("Loading")

        case .error:
            Button("error detected, reload") {
                // some action to reload data
            }

        default:
            Text("....")
        }
    }
}
```

skyeng

Готовимся к более сложным задачам

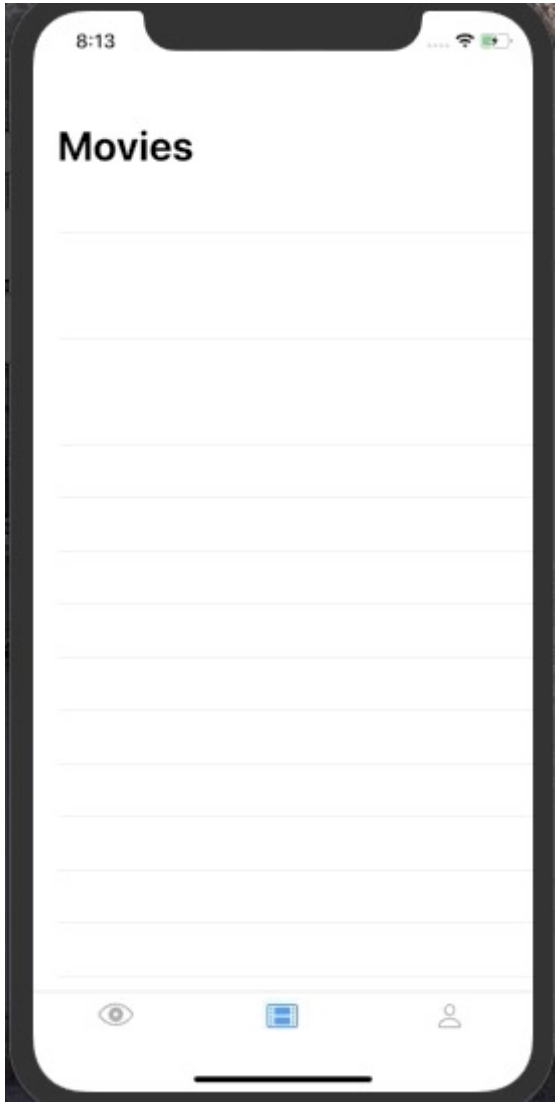


Готовимся к более сложным задачам. Пример




```
enum Event {
    case didLoad(Results)
    case didFail(Error)
    case fetchNext
}

struct State: Builder {
    var batch: Results
    var movies: [Movie]
    var status: Status
}
enum Status {
    case idle
    case loading
    case failed(Error)
}

struct MoviesView: View {
    typealias State = MoviesViewModel.State
    typealias Event = MoviesViewModel.Event
    let context: Context<State, Event>

    var body: some View {
        List {
            ForEach(context.movies.identified(by: \.id)) { movie in
                MovieCell(movie: movie).onAppear {
                // When we reach the end of the list
                // we send `fetchNext` event
                    if self.context.movies.last == movie {
                        self.context.send(event: .fetchNext)
                    }
                }
            }
        }
    }
}
```

Готовимся к более сложным задачам. Пример




```
static func reducer(state: inout State, event: Event) {
    switch event {
    case .didLoad(let batch):
        state.movies += batch.results
        state.status = .idle
        state.batch = batch
    case .didFail(let error):
        state.status = .failed(error)
    case .retry:
        state.status = .loading
    case .fetchNext:
        state.status = .loading
    }
}

static var feedback: Feedback<State, Event> {
    return Feedback(lensing: { $0.nextPage }) { page in
        URLSession.shared
            .fetchMovies(page: page)
            .map(Event.didLoad)
            .replaceError(replace: Event.didFail)
            .receive(on: DispatchQueue.main)
    }
}
```


Добавим навигацию

```
struct QuizzesListView: View {

    var body: some View {
        NavigationView {
            VStack {
                Text("Available quizzes")

                NavigationLink(destination: SingleQuizView(quiz: "SwiftUI")) {
                    Text("SwiftUI Quiz")
                }

                NavigationLink(destination: SingleQuizView(quiz: "Combine")) {
                    Text("Combine Quiz")
                }
            }
            .navigationTitle("Quizzes Lists")
        }
    }
}
```

skyeng

Навигация с ленивой (lazy) инициализацией

```
struct LazyView<Content: View>: View {

    let build: () -> Content

    init(_ build: @autoclosure @escaping () -> Content) {
        self.build = build
    }

    var body: Content {
        build()
    }
}


struct QuizzesListView: View {
  var body: some View {
    .....

    NavigationLink(destination: LazyView(SingleQuizView(quiz: "Combine"))) {
      Text("Combine Quiz")
    }
    .....
  }
}
```

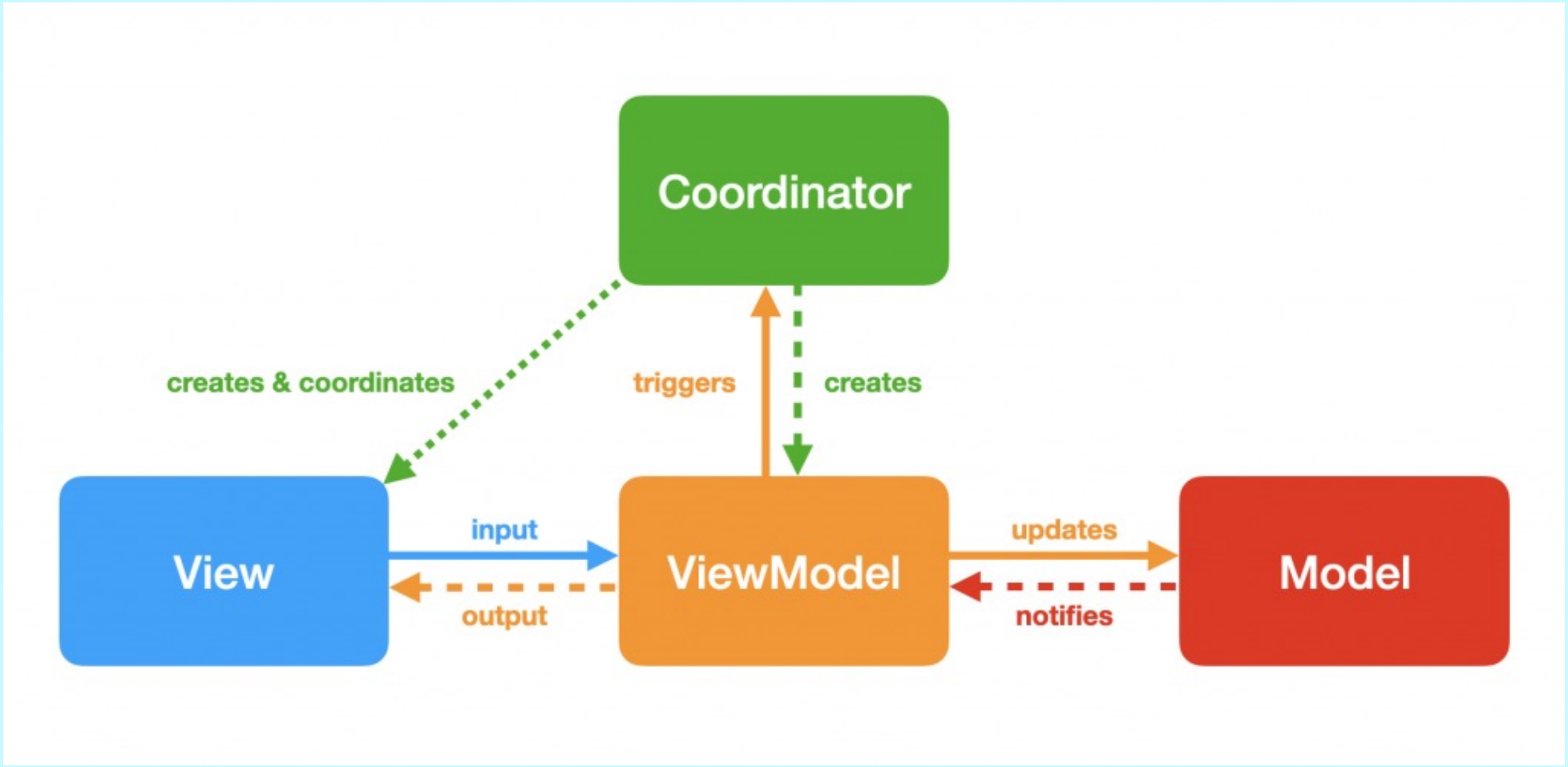
skyeng

Добавим навигацию

```swift
struct QuizzesListView: View {

    @State private var selection: String? = nil

    var body: some View {
        NavigationView {

            VStack {

                NavigationLink(destination: SingleQuizView(quiz: "SwiftUI"),
                               tag: "SwiftUI",
                               selection: $selection) { EmptyView() }

                NavigationLink(destination: SingleQuizView(quiz: "Combine"),
                               tag: "Combine",
                               selection: $selection) { EmptyView() }

                Button("SwiftUI Quiz") {
                    self.selection = "SwiftUI"
                }

                Button("Combine Quiz") {
                    self.selection = "Combine"
                }
            }
            .navigationTitle("Quizzes List")
        }
    }
}
```

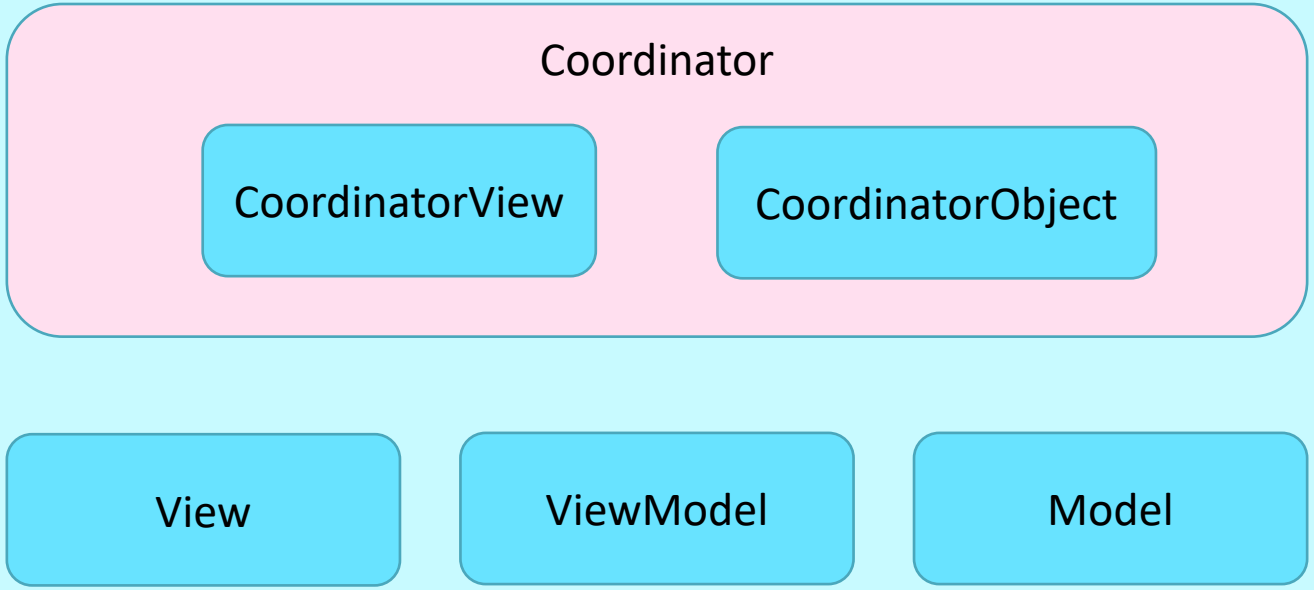
skyeng

Добавим навигацию: MVVM + координатор



skyeng

Добавим навигацию: координатор



skyeng

Добавим навигацию: координатор

```swift
class CoordinatorObject: ObservableObject {

    @Published var listViewModel: ListViewModel!
    @Published var detailViewModel: DetailViewModel?

    init() {
        self.listViewModel = ListViewModel(coordinator: self)
    }

    func open(_ item: ListItem) {
        self.detailViewModel = DetailViewModel(item: item, coordinator: self)
    }
}


struct CoordinatorView: View {

    @ObservedObject var object: CoordinatorObject

    var body: some View {
        NavigationView {
            ListView(viewModel: object.listViewModel)
                .navigation(item: object.detailViewModel) { DetailView(viewModel: $0) }
        }
    }

}
```

skyeng

Добавим навигацию: координатор

```
struct ListView: View {
    @ObservedObject var viewModel: ListViewModel

    var body: some View {
        List(viewModel.items) { item in
            Cell(item)
                .onNavigation { viewModel.open(item) }
        }
    }

}

class ListViewModel: ObservableObject, Identifiable {

    @Published var items = [ListItem]()

    private unowned let coordinator: CoordinatorObject

    init(coordinator: CoordinatorObject) {
        self.coordinator = coordinator
    }

    func open(_ item: ListItem) {
        coordinator.open(item)
    }

}
```

skyeng

Добавим навигацию: координатор

```
enum CoordinatorTab {
    case quizzes
    case results
    case profile
}

class CoordinatorObject: ObservableObject {

    @Published var tab = CoordinatorTab.results
    @Published var quizzesViewModel: QuizzesViewModel!
    @Published var resultsViewModel: ResultsViewModel!
    @Published var profileViewModel: ProfileViewModel!

    init() {
        self.quizzesViewModel = QuizzesViewModel(coordinator: self)
        self.resultsViewModel = ResultsViewModel(coordinator: self)
        self.profileViewModel = ProfileViewModel(coordinator: self)
    }

    func switchToQuizzesTab() {
        self.tab = .quizzes
    }

}
```

```
struct CoordinatorView: View {

    @ObservedObject var object: CoordinatorObject

    var body: some View {
        TabView(selection: $object.tab) {

            QuizzesView(viewModel: object.quizzesViewModel)
                .tabItem { /* ... */ }
                .tag(CoordinatorTab.quizzes)

            ResultsView(viewModel: object.resultsViewModel)
                .tag(CoordinatorTab.results)
                .tabItem { /* ... */ }

            ProfileView(viewModel: object.profileViewModel)
                .tabItem { /* ... */ }
                .tag(CoordinatorTab.profile)
        }
    }
}
```

skyeng

Добавим навигацию: координатор

```
class CoordinatorObject: ObservableObject {

    @Published var sheetViewModel: SheetViewModel?

    func openSheet(_ info: SheetInformation) {
        self.sheetViewModel = SheetViewModel(info: info)
    }

}

struct CoordinatorView: View {

    @ObservedObject var object: CoordinatorObject

    var body: some View {
        Text("Quizzes list")
            .sheet(item: $object.sheetViewModel) {
                SomeSheet(viewModel: $0)
            }
    }

}
```

skyeng

Добавим навигацию: координатор

```swift
final class UnauthenticatedCoordinator: NavigationCoordinatable {

    let stack = NavigationStack(initial: \UnauthenticatedCoordinator.start)

    @Root var start = makeStart
    @Route(.modal) var forgotPassword = makeForgotPassword
    @Route(.push) var registration = makeRegistration

    func makeRegistration() -> RegistrationCoordinator {
        return RegistrationCoordinator()
    }

    @ViewBuilder func makeForgotPassword() -> some View {
        ForgotPasswordScreen()
    }

    @ViewBuilder func makeStart() -> some View {
        LoginScreen()
    }
}
```

```swift
struct QuizzesApp: App {

    var body: some Scene {
        WindowGroup {
            MainCoordinator().view()
        }
    }
}

.......

struct QuizzesScreen: View {

    @EnvironmentObject var quizzesRouter: QuizzesCoordinator.Router

    var body: some View {
        List {
          /* ... */
        }
        .navigationBarItems(
            trailing: Button("Open quizz") {
                    todosRouter.route(to: \.openQuiz)
                }
            )
        )
    }
}
```

skyeng

Добавим навигацию: координатор

```swift
// First way to access router from ViewModel
struct QuizzesScreen: View {
    @StateObject var viewModel = QuizzesViewModel()
    @EnvironmentObject var quizzesRouter: QuizzesCoordinator.Router

    var body: some View {
        List {
          /* ... */
        }
        .onAppear {
            viewModel.router = quizzesRouter
        }
    }
}

// Second way to access router from ViewModel
class LoginScreenViewModel: ObservableObject {

    var mainRouter: MainCoordinator.Router? = RouterStore.shared.retrieve()

    func loginButtonPressed() {
        mainRouter?.root(\.authenticated)
    }
}
```

skyeng

Подведем итоги

Вспомнили про MVVM для iOS

Поговорили о имеющихся сложностях применения MVVM

Рассмотрели подход с использованием State Machine

Посмотрели примеры реализации State Machine на SwiftUI и Combine

Рассмотрели как можно организовать координатор

Материалы

- iOS Dev Survey
- «Data Flow Through SwiftUI» WWDC
- CombineFeedback library
- «Building a state-driven app in SwiftUI using state machines» by Peter Ringset
- «Modern MVVM iOS App Architecture with Combine and SwiftUI» by Vadim Bulavin
- « Avoiding SwiftUI's AnyView » by John Sundell
- « How to Use the Coordinator Pattern in SwiftUI» QuickBird
- Stinsen library

Спасибо

Slava Slutsker
Head of mobile platform

website: slutsker.pro
twitter: @SlavaSlutsker
LinkedIn: Slava Slutsker

skyeng