Урок 3

Паттерны навигации в iOS-приложениях

Всё о том, как переходить от контроллера к контроллеру

Введение

Мы уже изучали переходы **segue,** а также пробовали обходиться без них. Также мы знакомы с **UINavigationController** и **UITableController**. То есть мы знаем всё, чтобы успешно совершать переход от экрана к экрану, но пока не знаем, как применять технологии более эффективно. А этому как раз и посвящён курс.

В iOS есть три способа отобразить новый экран:

1. Воспользоваться **segue,** подкорректировав переход при помощи специальных методов в коде.
2. Создать контроллер в коде и вызвать один из двух методов – **show** или **present**.
3. Заменить **rootViewController** у **UIWindow:** это подменит самый первый контролер в приложении. Также можно создать дополнительный **UIWindow** и отобразить его поверх текущего.

Паттернов переходов немного больше. Как их использовать?

1. Не использовать паттерны, а просто совершать переходы когда захочется.
2. Использовать **router,** отдельный класс, связанный с **UIViewController,** в который вынесена вся логика перехода.
3. Использовать **Wireframe**, отдельный класс, отвечающий за сборку и переходы между контроллерами. Этот паттерн мы рассматривать не будем, так как он почти не используется.
4. Использовать координаторы – несколько классов, полностью инкапсулирующих логику переходов.

Не все эти способы и паттерны можно использовать вместе. **Segue** плохо сочетается с **Wireframe** и координаторами.

На протяжении этого урока мы будем работать с четырьмя контроллерами:

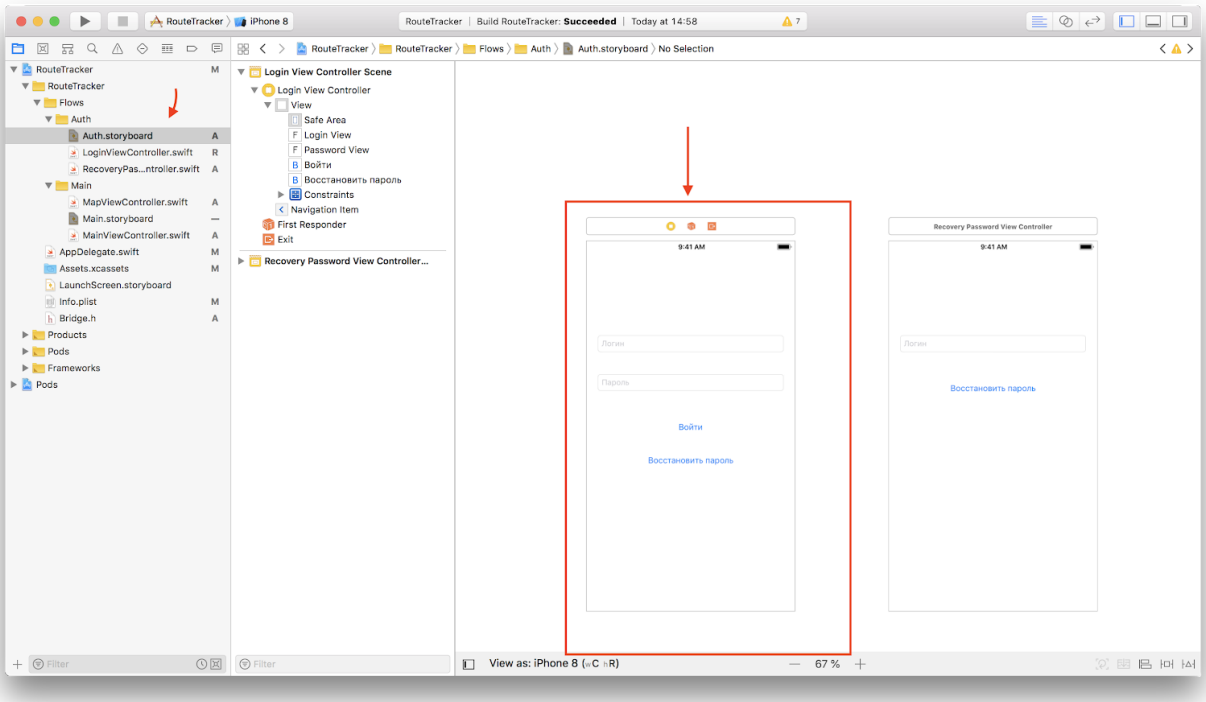
- вход;
- восстановление пароля;
- главный экран;
- карта.

Работая с ними, мы познакомимся со всеми подходами.

Контроллеры

Подготовим необходимые контроллеры. Они будут очень простыми.

Создадим первый контроллер авторизации. Он будет располагаться в отдельном **storyboard,** так как это отдельный сценарий авторизации.



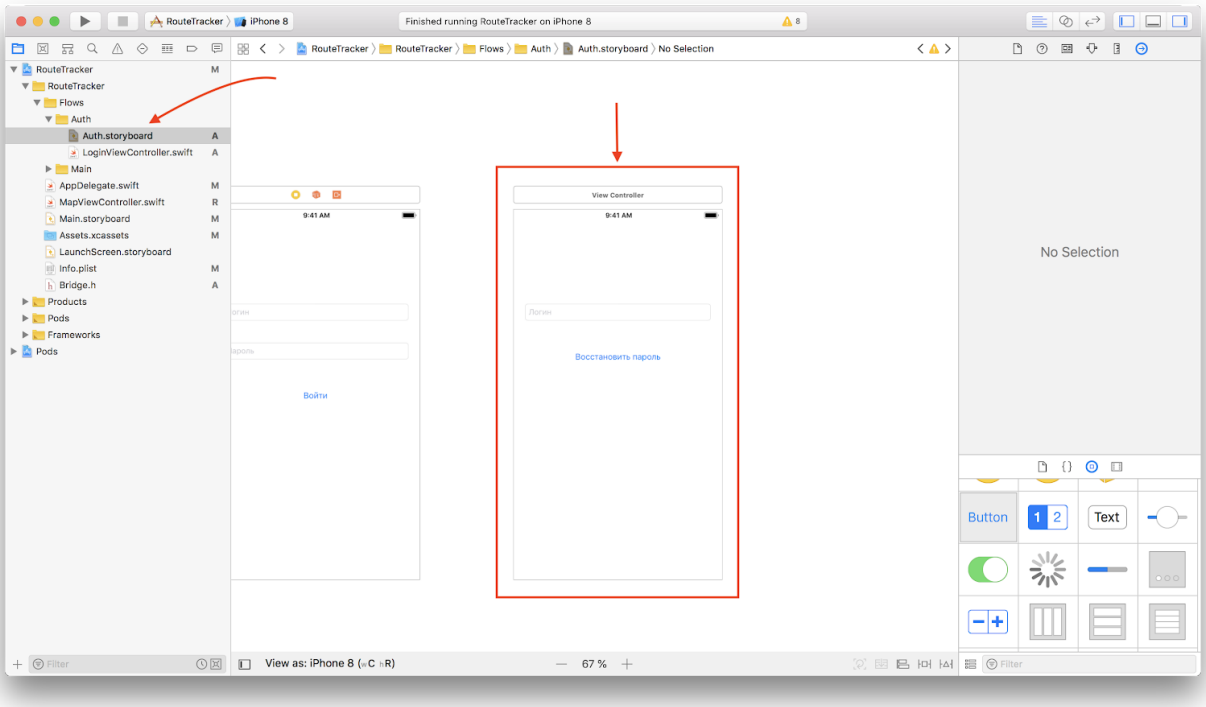
А вот его код:

```
import UIKit

final class LoginViewController: UIViewController {

    enum Constants {
        static let login = "admin"
        static let password = "123456"
    }

    @IBOutlet weak var loginView: UITextField!
    @IBOutlet weak var passwordView: UITextField!


    @IBAction func login(_ sender: Any) {
        guard
            let login = loginView.text,
            let password = passwordView.text,
            login == Constants.login && password == Constants.password
        else {
            return
        }

        print("Логин")
    }

    @IBAction func recovery(_ sender: Any) {

    }
}
```

У нас протянуты аутлеты для полей ввода и экшены для нажатия на кнопки. По нажатию на «Вход»

мы проверяем, что данные верны, а при нажатии на «Восстановление» мы должны попасть на экран восстановления.

Экран восстановления пароля:

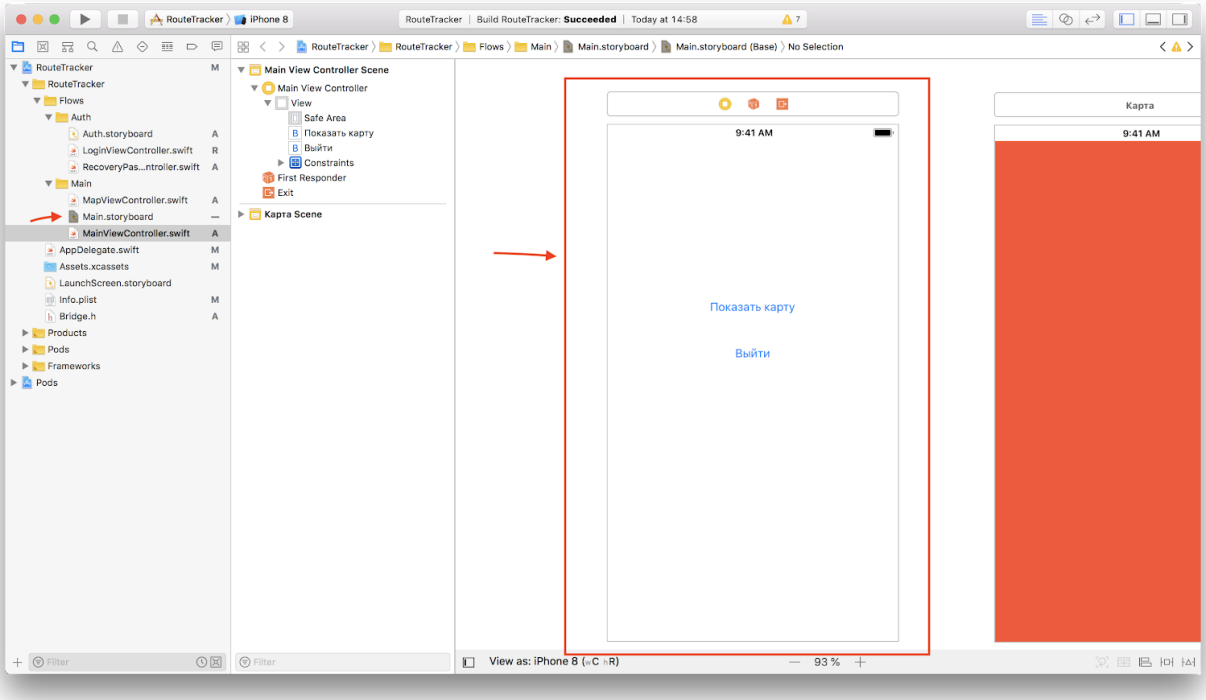


И его код:

```
import UIKit

final class RecoveryPasswordViewController: UIViewController {

    @IBOutlet weak var loginView: UITextField!

    @IBAction func recovery(_ sender: Any) {
        guard
            let login = loginView.text,
            login == LoginViewController.Constants.login else {

                return
        }

        showPassword()
    }

    private func showPassword() {
            let alert = UIAlertController(title: "Пароль", message: "123456",
preferredStyle: .alert)

        let ok = UIAlertAction(title: "OK", style: .cancel)
        alert.addAction(ok)
        present(alert, animated: true)
    }
}
```

Мы видим один аутлет для поля ввода логина и кнопку, которая проверяет, что логин верный, и показывает всплывающее окно с паролем.

Теперь перейдём к главному сценарию. Добавим простой контроллер с двумя кнопками для перехода к карте и выхода.

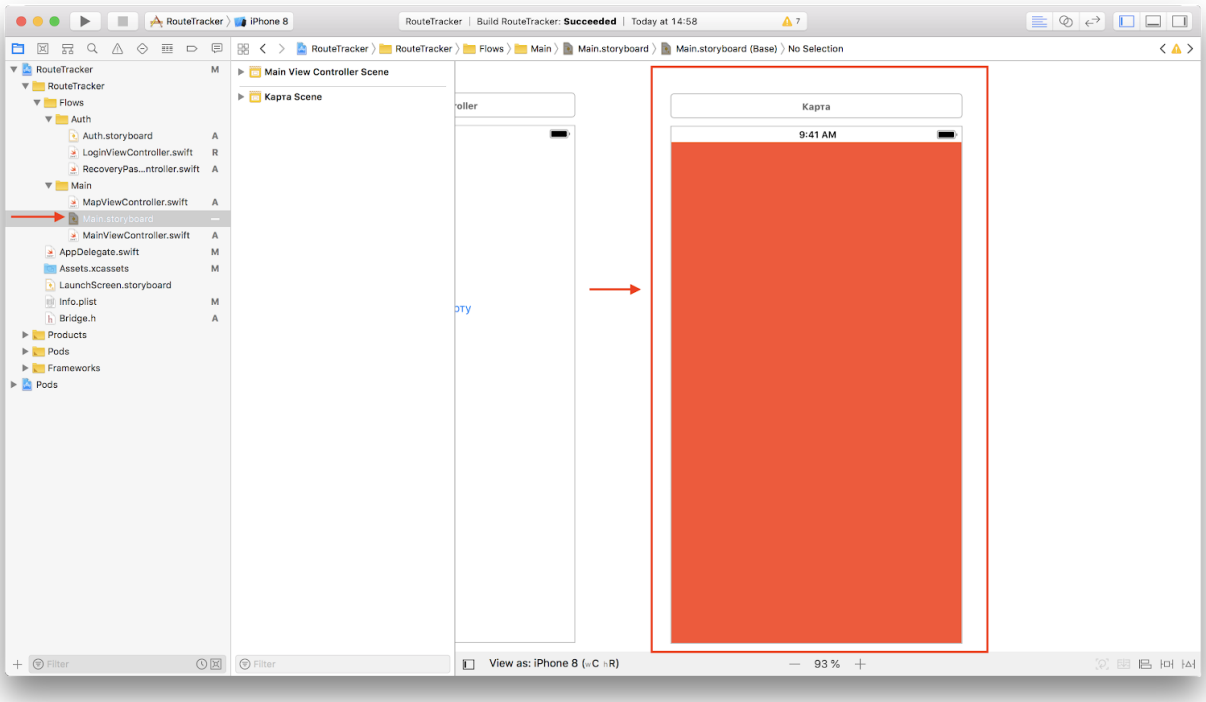


И его код:

```
import UIKit

final class MainViewController: UIViewController {

    @IBAction func showMap(_ sender: Any) {
    }

    @IBAction func logout(_ sender: Any) {
    }

}
```

Здесь два экшена от кнопок, оба пока ничего не делают.

И последний контроллер, уже знакомый вам – контроллер карты.



И его код:

```
import UIKit
import GoogleMaps
import CoreLocation

class MapViewController: UIViewController {

// Центр Москвы
       let coordinate = CLLocationCoordinate2D(latitude: 59.939095, longitude:
30.315868)
    var locationManager: CLLocationManager?
    var route: GMSPolyline?
    var routePath: GMSMutablePath?

    @IBOutlet weak var mapView: GMSMapView!

    override func viewDidLoad() {
        super.viewDidLoad()
        configureMap()
        configureLocationManager()
    }

    func configureMap() {
// Создаём камеру с использованием координат и уровнем увеличения
        let camera = GMSCameraPosition.camera(withTarget: coordinate, zoom: 17)
// Устанавливаем камеру для карты
        mapView.camera = camera
    }

    func configureLocationManager() {
        locationManager = CLLocationManager()
        locationManager?.delegate = self
        locationManager?.allowsBackgroundLocationUpdates = true
```

```
        locationManager?.pausesLocationUpdatesAutomatically = false
        locationManager?.desiredAccuracy = kCLLocationAccuracyNearestTenMeters
        locationManager?.startMonitoringSignificantLocationChanges()
        locationManager?.requestAlwaysAuthorization()
    }

    @IBAction func updateLocation(_ sender: Any) {
// Отвязываем от карты старую линию
        route?.map = nil
// Заменяем старую линию новой
        route = GMSPolyline()
// Заменяем старый путь новым, пока пустым (без точек)
        routePath = GMSMutablePath()
// Добавляем новую линию на карту
        route?.map = mapView
// Запускаем отслеживание или продолжаем, если оно уже запущено
        locationManager?.startUpdatingLocation()
    }

    @IBAction func currentLocation(_ sender: Any) {

        locationManager?.requestLocation()
    }
}

extension MapViewController: CLLocationManagerDelegate {
        func locationManager(_ manager: CLLocationManager, didUpdateLocations
locations: [CLLocation]) {
// Берём последнюю точку из полученного набора
        guard let location = locations.last else { return }
// Добавляем её в маршрут
        routePath?.add(location.coordinate)
// Обновляем путь у линии маршрута путём повторного присвоения
        route?.path = routePath
// Чтобы наблюдать за движением, установим камеру на только что добавленную
// точку
         let position = GMSCameraPosition.camera(withTarget: location.coordinate,
zoom: 17)
        mapView.animate(to: position)
    }

     func locationManager(_ manager: CLLocationManager, didFailWithError error:
Error) {
        print(error)
    }
}
```

Код последнего контроллера не важен в данном уроке. Он полностью идентичен тому, что мы написали на прошлом. Он практически не принимает участия в навигации.

Переходы

Теперь у нас есть контроллеры, и мы можем сформулировать задачу, которую будем решать.

1. При запуске приложение должно проверять, авторизован пользователь или нет.
    a. Если авторизован, показывается главный сценарий.

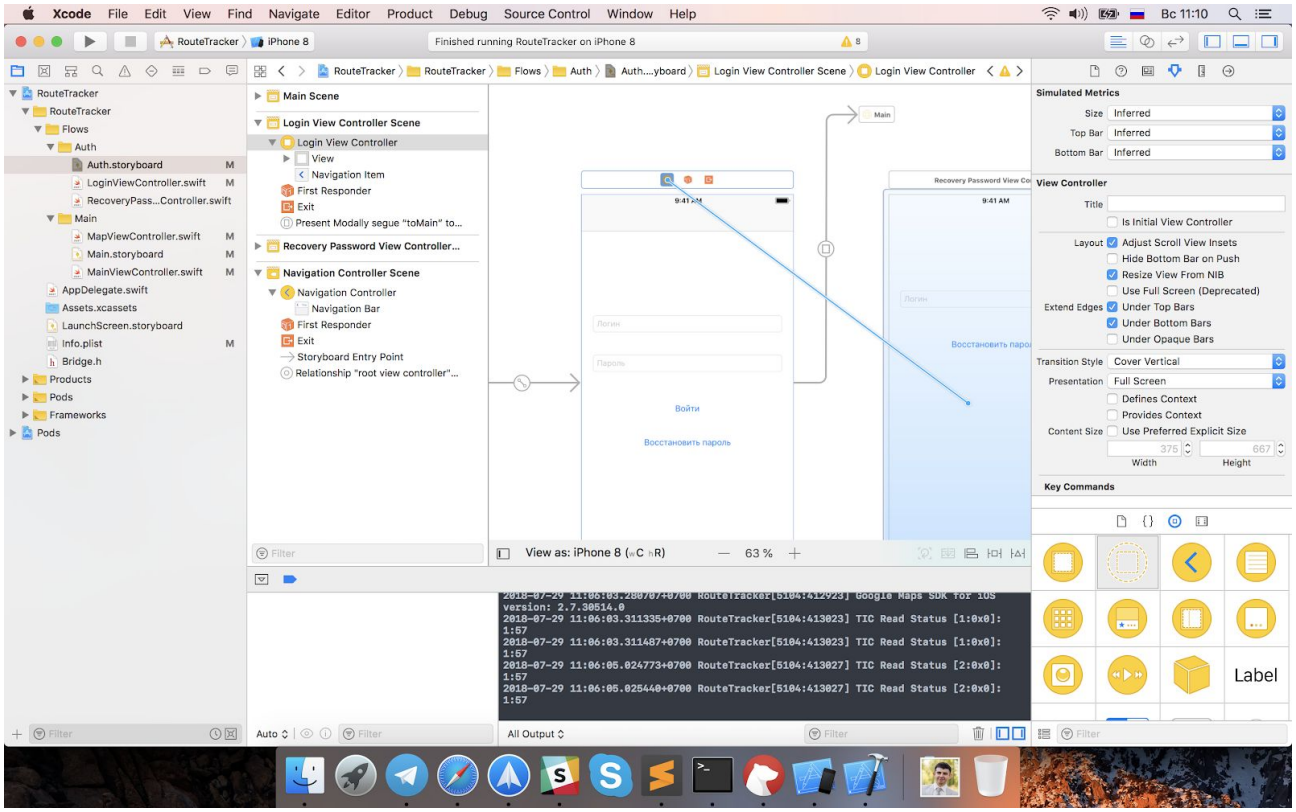
    b. Если не авторизован, показывается сценарий входа.
2. На сценарии входа:
    a. При вводе верного логина и пароля мы должны перейти к главному сценарию.
    b. При нажатии на кнопку восстановления мы должны перейти на экран восстановления пароля.
    c. На экране восстановления мы должны иметь возможность вернуться назад на экран входа.
3. На главном сценарии.
    a. На главном экране мы не должны иметь возможности просто вернуться назад к логину.
    b. Но при нажатии на кнопку «Выйти» мы должны разлогиниться и выйти на экран входа.
    c. При нажатии на кнопку карты мы переходим к карте.

Вот такой нехитрый сценарий, но при этом мы можем попробовать все подходы.

Вариант 1. Беспорядочные segue

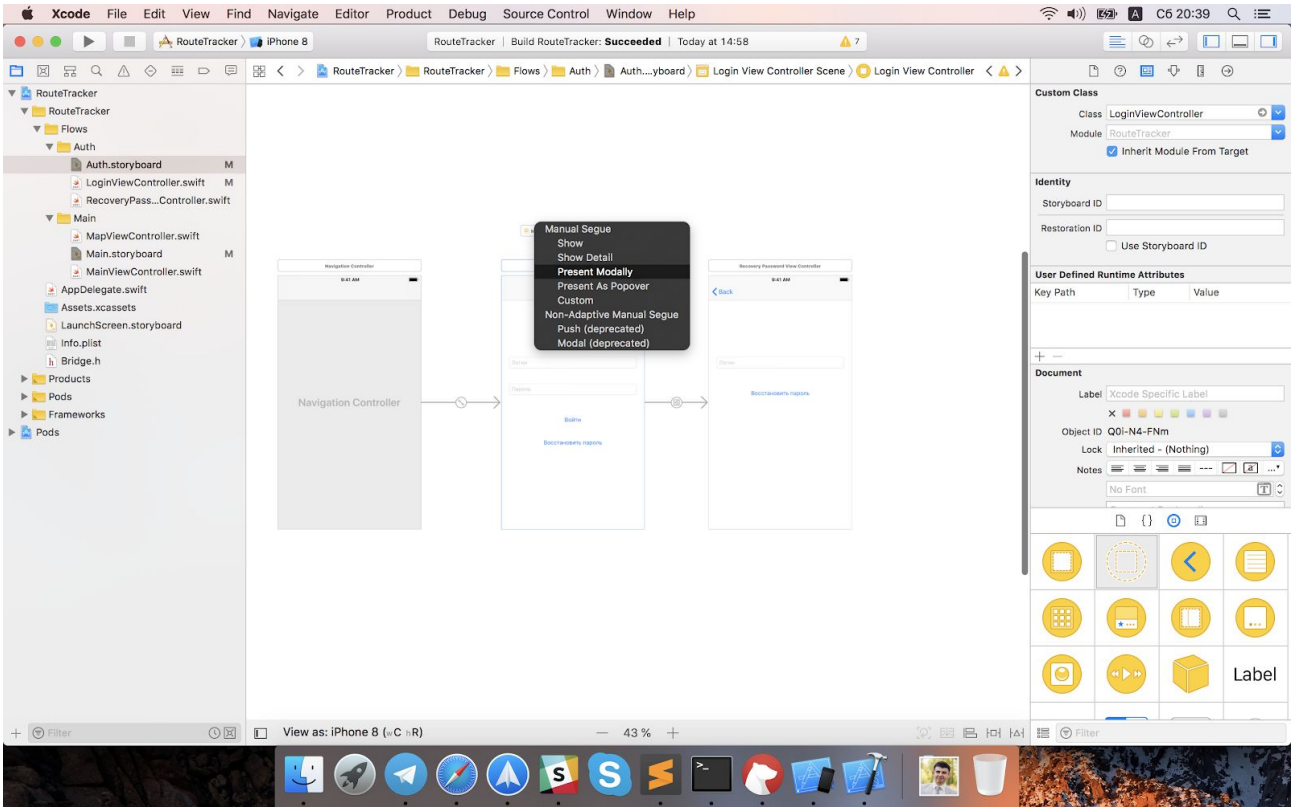
Итак, первое, что мы попробуем, – беспорядочные segue. Добавим **UINavigaionController.**

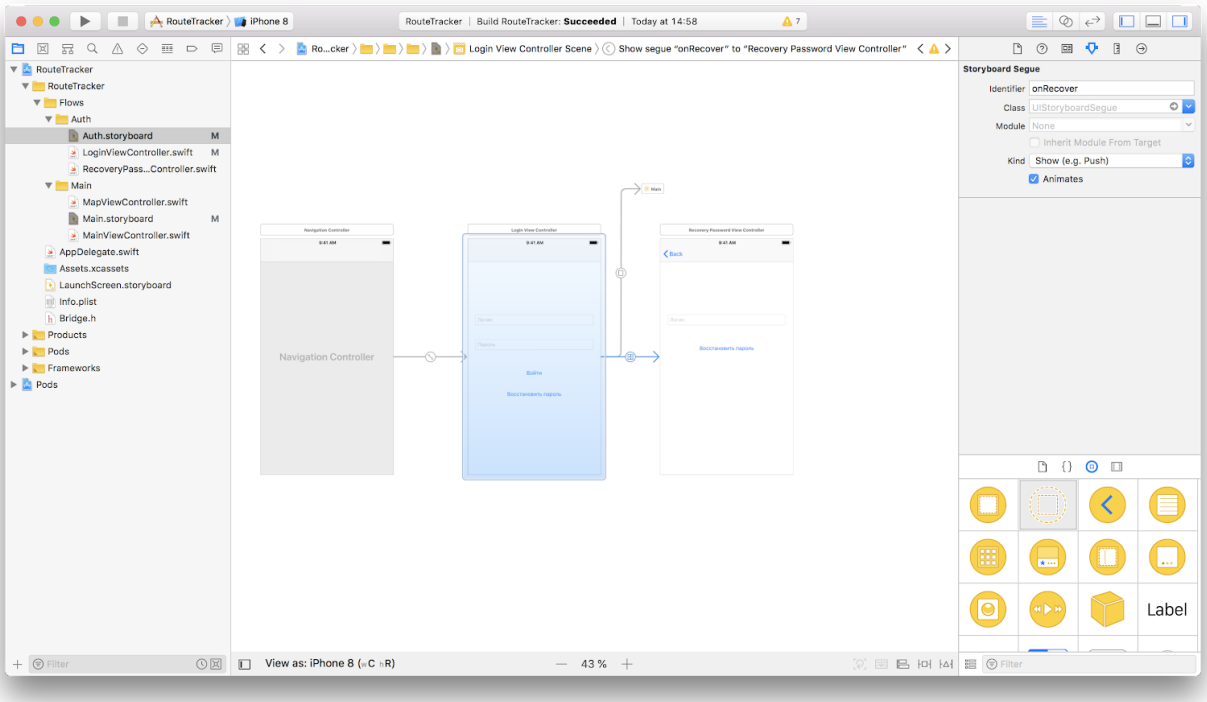
Самая важная деталь: мы будем перетягивать **segue** не от кнопки или другого элемента, а от самого контроллера, а точнее от жёлтого квадратика вверху контроллера.

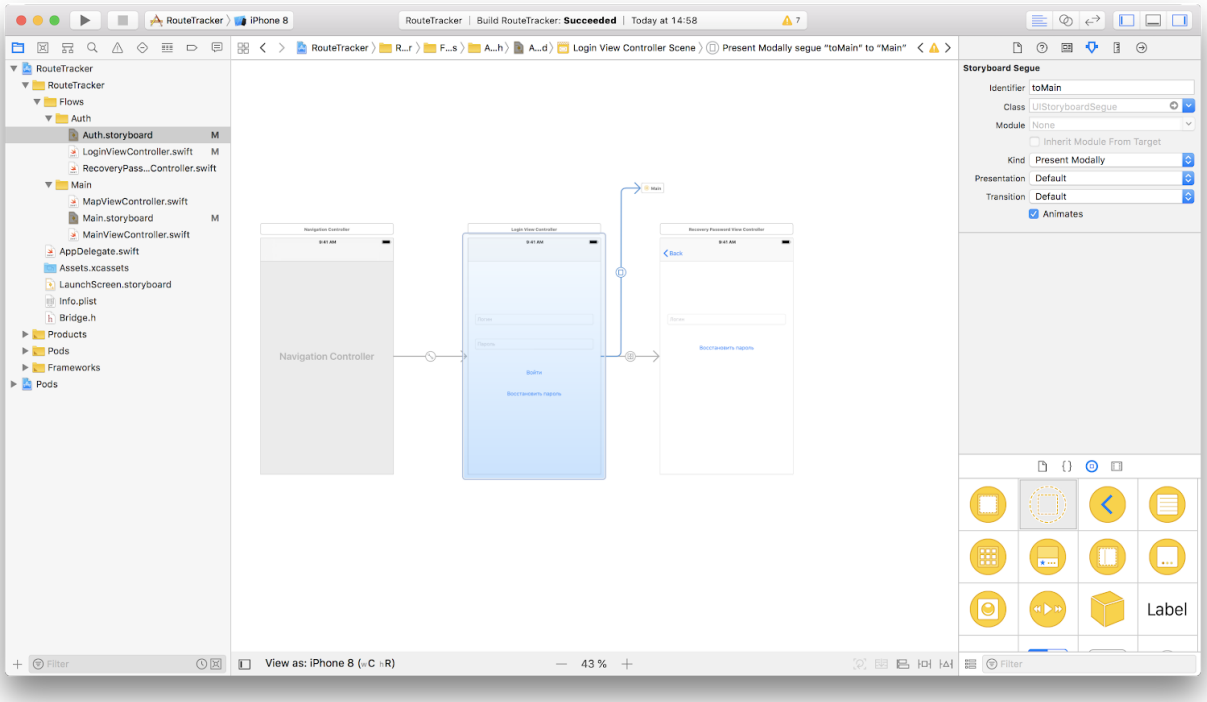


Перечислим действия в сценарии авторизации:

1. Добавим **UINavigationController.**
2. Добавим **segue** к контроллеру восстановления с типом **show** и идентификатором **onRecover.**
3. Добавим ссылку на главный **storyboard.**
4. Добавим **segue** к сториборду с типом **present** и идентификатором **toMain.**


Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help
RouteTracker | Build RouteTracker: Succeeded | Today at 14:58
RouteTracker
RouteTracker
Flows
Auth
Auth.storyboard
LoginViewController.swift
RecoveryPass...Controller.swift
Main
MapViewController.swift
Main.storyboard
MainViewController.swift
AppDelegate.swift
Assets.xcassets
LaunchScreen.storyboard
Info.plist
Bridge.h
Products
Pods
Frameworks
Pods
Navigation Controller
Manual Segue
Show
Show Detail
Present Modally
Present As Popover
Custom
Non-Adaptive Manual Segue
Push (deprecated)
Modal (deprecated)
Custom Class
Class LoginViewController
Module RouteTracker
Inherit Module From Target
Identity
Storyboard ID
Restoration ID
Use Storyboard ID
User Defined Runtime Attributes
Key Path
Type
Value
Document
Label
Object ID Q0l-N4-FNm
Lock Inherited - (Nothing)
Notes
No Font
Label
View as: iPhone 8
43 %





Код контроллера авторизации теперь выглядит вот так.

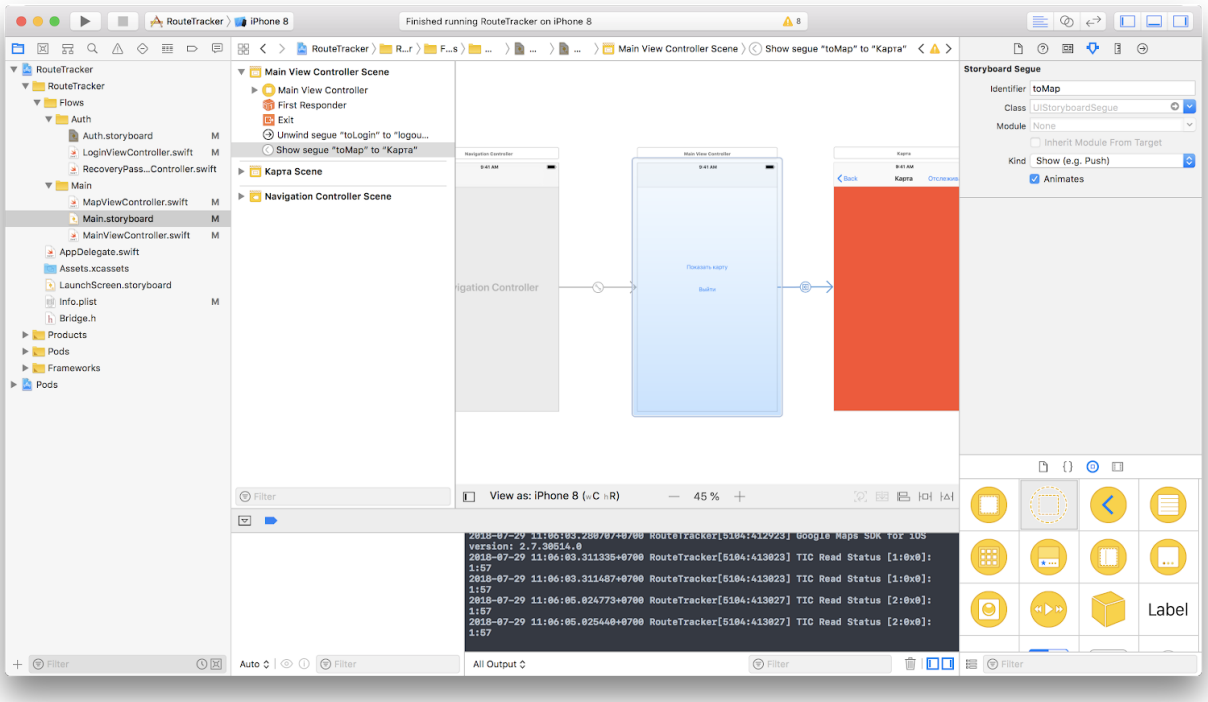
```
import UIKit

final class LoginViewController: UIViewController {

    enum Constants {
        static let login = "admin"
        static let password = "123456"
    }
```

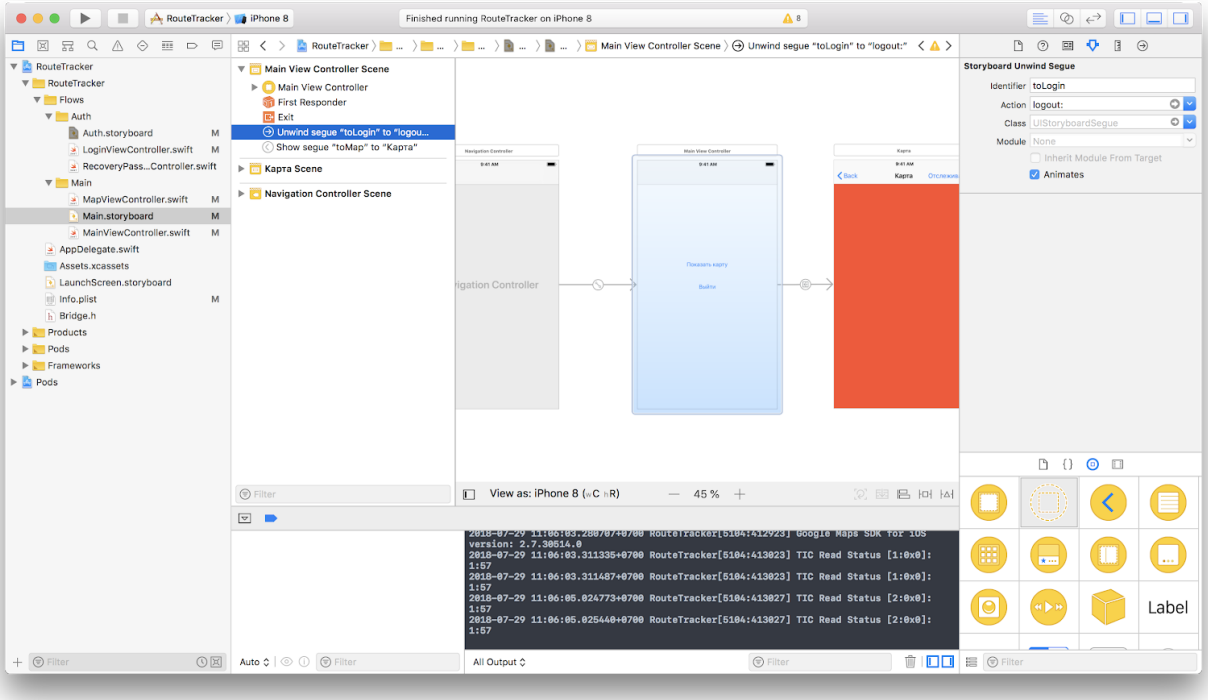
```
    @IBOutlet weak var loginView: UITextField!
    @IBOutlet weak var passwordView: UITextField!

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

// Показав контроллер авторизации, проверяем: если мы авторизованы,
// сразу переходим к основному сценарию
        if UserDefaults.standard.bool(forKey: "isLogin") {
            performSegue(withIdentifier: "toMain", sender: self)
        }

    }


    @IBAction func login(_ sender: Any) {
        guard
            let login = loginView.text,
            let password = passwordView.text,
            login == Constants.login && password == Constants.password
        else {
            return
        }
// Сохраним флаг, показывающий, что мы авторизованы
        UserDefaults.standard.set(true, forKey: "isLogin")
// Перейдём к главному сценарию
        performSegue(withIdentifier: "toMain", sender: sender)
    }

    @IBAction func recovery(_ sender: Any) {
        performSegue(withIdentifier: "onRecover", sender: sender)
    }

// Unwind segue для выхода автоматически удаляет флаг авторизации
    @IBAction func logout(_ segue: UIStoryboardSegue) {
        UserDefaults.standard.set(false, forKey: "isLogin")
    }
}
```

Мы добавили вызовы переходов по их идентификаторам. Также при авторизации мы сохраняем в **UserDefaults** флаг, показывающий, что мы авторизованы. При последующих запусках мы будем проверять этот флаг и в случае авторизации сразу переходить к главному сценарию. Напоследок мы добавили **Unwind segue** для выхода. Контроллер для восстановления при этом не изменится.

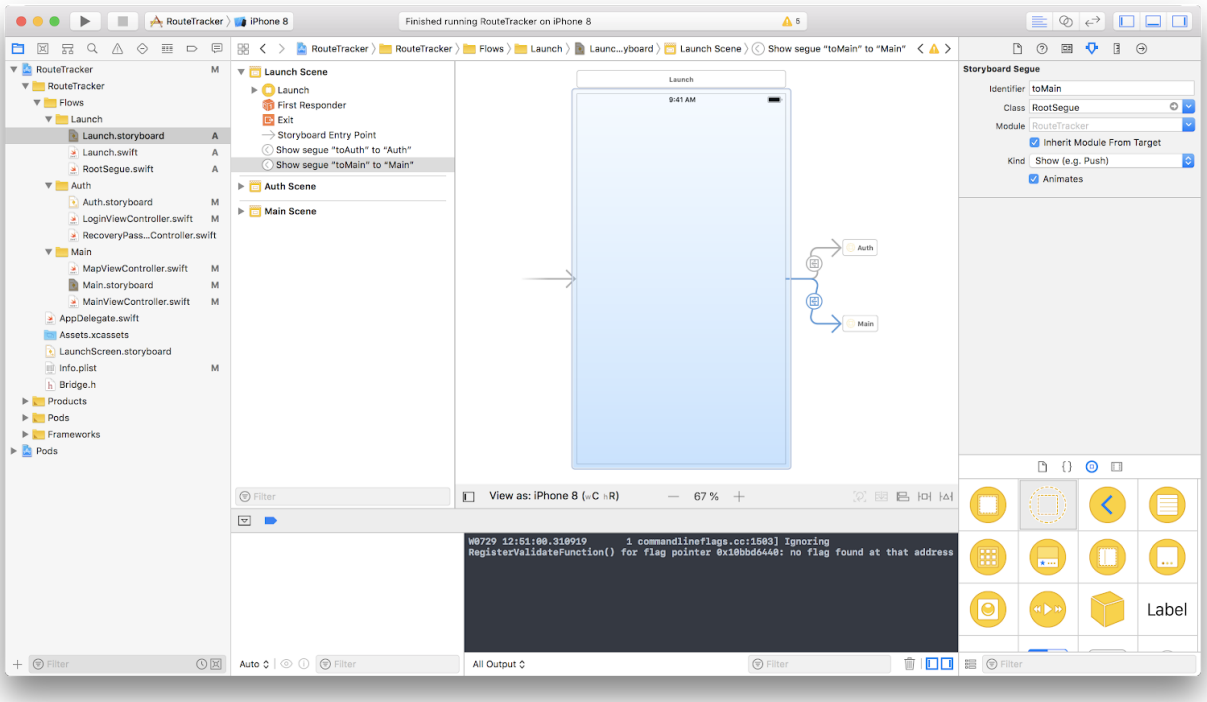
На главном сценарии мы сделаем следующее:

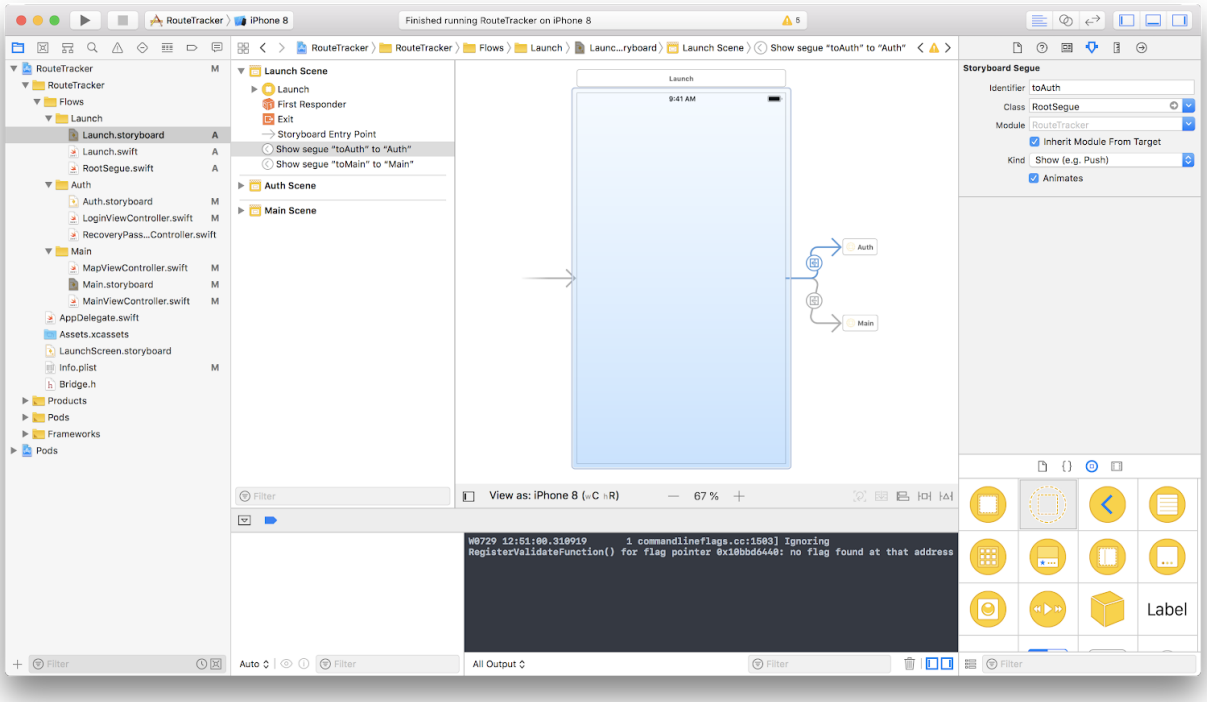
1. Добавим **UINavigationController.**
2. Добавим **segue** к контроллеру с картой с типом **show** и идентификатором **toMap.**
3. Добавим **Unwind Segue** для выхода с идентификатором **toLogin.**





Запустим приложение. Всё работает. Но если мы, будучи авторизованными, откроем приложение, то сначала увидим, как оно запустится, а затем автоматически перейдём к главному сценарию. Это нужно исправить, но как выбрать экран, на который мы попадём? Если использовать **segue**, придётся создать начальный экран, который будет решать, что загружать дальше, и менять главный контролер. Для этого создадим отдельный сценарий и контроллер в нём.





Как видите, это отдельный **Storyboard**: **Launch**, в нём единственный контроллер **Launch**. Также две ссылки на другие **Storyboard**: **Auth** и **Main**. Обратите внимание (скриншоты выше) на идентификаторы и класс **Segue**. Класс значится как **RootSegue**, его нет в стандартной библиотеке. Вот его код:

```
import UIKit

class RootSegue: UIStoryboardSegue {

    override func perform() {
```

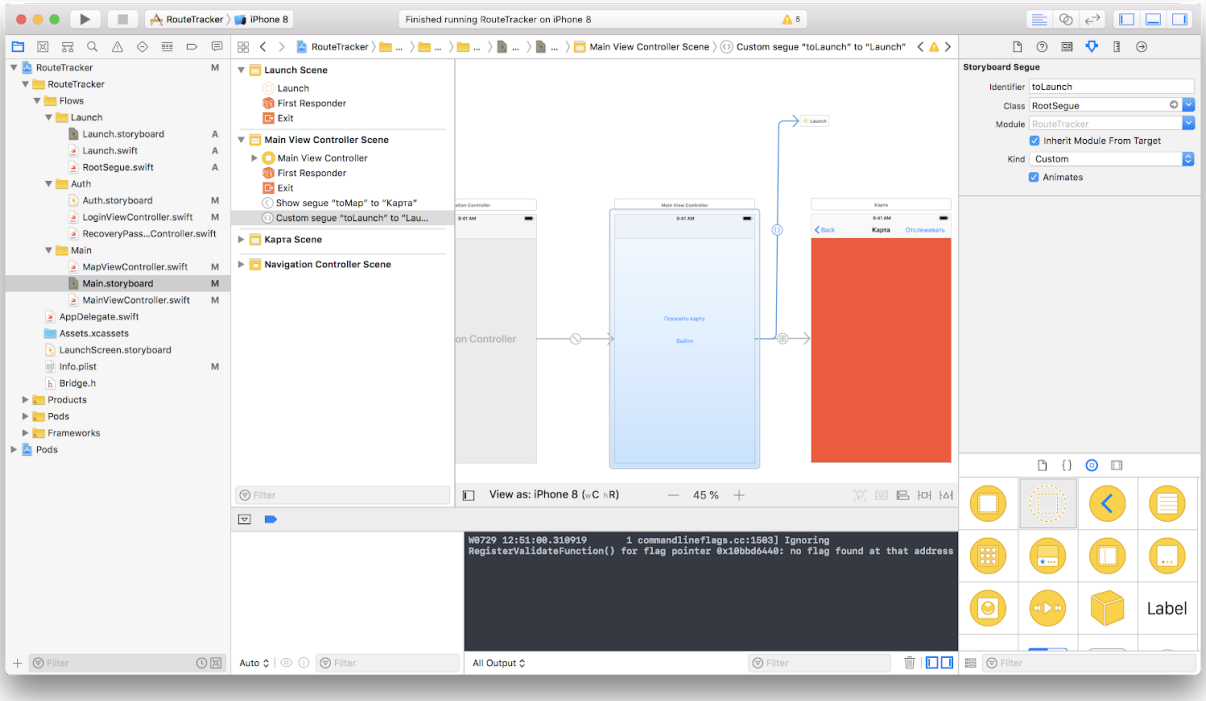
```
        UIApplication.shared.keyWindow?.rootViewController = destination
    }

}
```

Самое главное происходит в методе **perform:** мы получаем доступ к свойству **rootViewController** у текущего **UIWindow** и заменяем его на тот контроллер, на который мы переходим. В результате мы заменяем все контроллеры на новые.

Давайте посмотрим на код **Launch** контроллера:

```
class Launch: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        if UserDefaults.standard.bool(forKey: "isLogin") {
            performSegue(withIdentifier: "toMain", sender: self)
        } else {
            performSegue(withIdentifier: "toAuth", sender: self)
        }
    }

}
```

Сюда переместилась проверка, авторизованы мы или нет. По завершении проверки мы вызываем соответствующий переход. В результате в **Login**-контроллере этой проверки не будет, мы её удалили. Также мы удалили **UnwindSegue**, так как не сможем вернуться на контроллер, которого нет в стеке. Вместо этого в главном сценарии сделаем такой же **RootSegue**-переход на стартовый экран.



Обработчик кнопки **Logout** выглядит так.

```
    @IBAction func logout(_ sender: Any) {
        UserDefaults.standard.set(false, forKey: "isLogin")
```

```
        performSegue(withIdentifier: "toLaunch", sender: sender)
    }
```

Теперь у нас простые переходы и при этом нет лишней анимации.

Вариант 2. Segue и Router

В нашем примере все переходы довольно простые и их не так много, поэтому беспорядочные **segue** не вызывают проблем. Но если у вас количество переходов с экрана увеличивается, да еще и нужно передавать данные, то возрастает и количество кода. В такой ситуации логика переходов выносится в роутер. Роутеры можно написать по-разному: кто-то переносит код в экстеншены, кто-то – в отдельный класс. Мы выберем второй вариант.

Нам потребуется базовый класс и три роутера для всех контроллеров, с которых мы осуществляем переходы.

Базовый роутер выглядит вот так.

```
class BaseRouter: NSObject {
    @IBOutlet weak var controller: UIViewController!

// Метод, принимающий идентификатор сигвея и замыкание, которое будет вызвано
// при переходе. Метод является дженериком
    func perform<Controller: UIViewController>(
        segue: String,
        performAction: ((Controller) -> Void)? = nil) {

// Оборачиваем замыкание в другое замыкание, принимающее обобщённый
// контроллер и пытаемся привести его к типу требуемого контроллера
        let performAction = performAction.map { action in
            { (controller: UIViewController) in
// Выбрасывает исключение в режиме дебага, если тип контроллера неверный
                guard let controller = controller as? Controller else {
                    assertionFailure("Ожидался \(Controller.self)")
                    return
                }
                action(controller)
            }
        }

        controller?.performSegue(withIdentifier: segue, sender: performAction)
    }

    func prepare(
        for segue: UIStoryboardSegue,
        sender: Any?) {

        guard let action = sender as? ((UIViewController) -> Void) else { return
}
        action(segue.destination)
    }

}
```

Он имеет свойство **controlle** для доступа к контроллеру и вызова у него метода **performSegue,** и два метода. Как они работают, лучше прочитать в комментариях, но важно понять общий принцип.

У нас два стандартных метода: **performSegue** вызывает переход без возможности добавить свой обработчик, **prepare(for:sender:)** вызывается уже перед переходом и позволяет добавить логику – например, передать параметр вызываемому контроллеру.

Эти два метода работают, устраняют проблему и позволяют вызвать переход и сконфигурировать вызываемый контроллер одновременно. Это достигается путём хитрого замыкания, которое передаётся через аргумент sender.

Теперь напишем роутеры для всех контроллеров, которые работают с **Segue**, и перепишем контроллеры, чтобы они работали с роутерами.

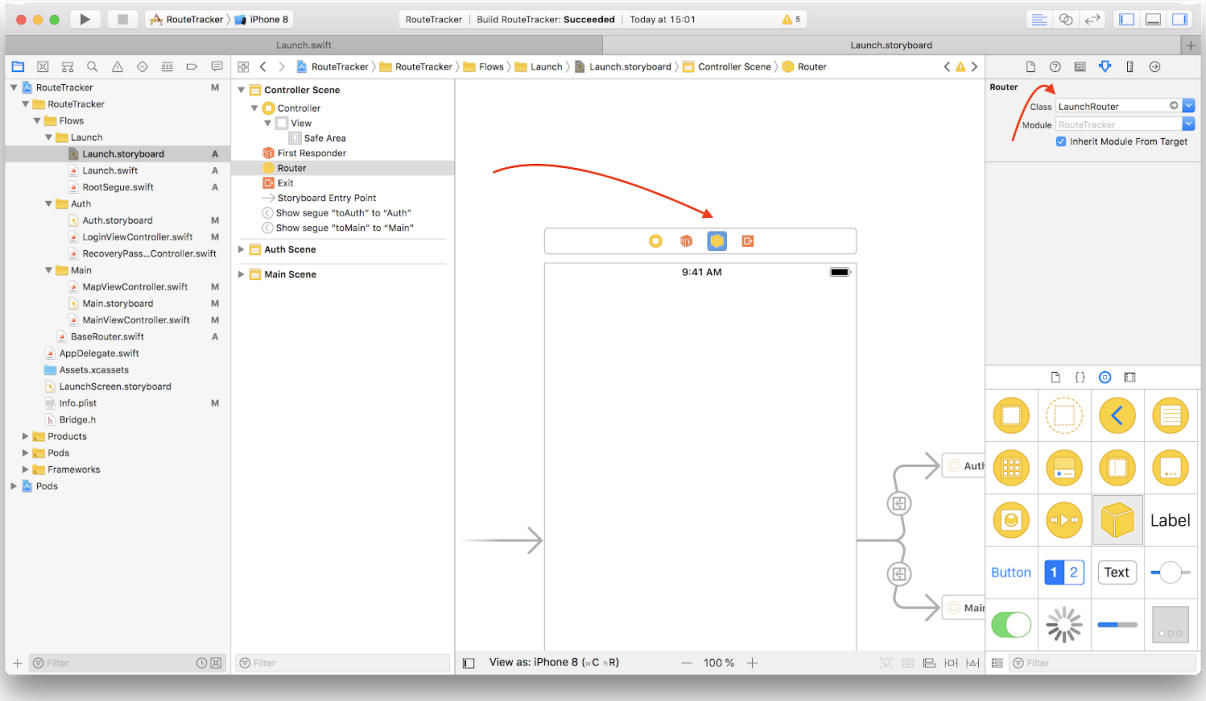
```
import UIKit

class Launch: UIViewController {

    @IBOutlet weak var router: LaunchRouter!

    override func viewDidLoad() {
        super.viewDidLoad()
        if UserDefaults.standard.bool(forKey: "isLogin") {
            router.toMain()
        } else {
            router.toAuth()
        }
    }

}


final class LaunchRouter: BaseRouter {

    func toMain() {
        perform(segue: "toMain")
    }

    func toAuth() {
        perform(segue: "toAuth")
    }

}
```

```
import UIKit

final class LoginViewController: UIViewController {

    enum Constants {
        static let login = "admin"
        static let password = "123456"
    }

    @IBOutlet weak var loginView: UITextField!
    @IBOutlet weak var passwordView: UITextField!
    @IBOutlet weak var router: LoginRouter!


    @IBAction func login(_ sender: Any) {
        guard
            let login = loginView.text,
```

```
            let password = passwordView.text,
            login == Constants.login && password == Constants.password
        else {
            return
        }
// Сохраним флаг, показывающий, что мы авторизованы
        UserDefaults.standard.set(true, forKey: "isLogin")
        router.toMain()
    }

    @IBAction func recovery(_ sender: Any) {
        router.onRecover()
    }
}

final class LoginRouter: BaseRouter {
    func toMain() {
        perform(segue: "toMain")
    }

    func onRecover() {
        perform(segue: "onRecover")
    }
}
```

Прежде чем разобрать последний контроллер, давайте посмотрим, какие именно изменения необходимо сделать в **storyboard,** чтобы все заработало.

Надо добавить объект в контроллер и установить для него аутлетом контроллер, а сам объект аутлетом подключить в контроллер.



Чтобы увидеть мощь роутеров, добавим бесполезное свойство **usselesExampleVariable** в контроллер карты. Мы будем устанавливать его при переходе.

```
<...>
class MapViewController: UIViewController {

    var usselesExampleVariable = ""
    <...>
}
```

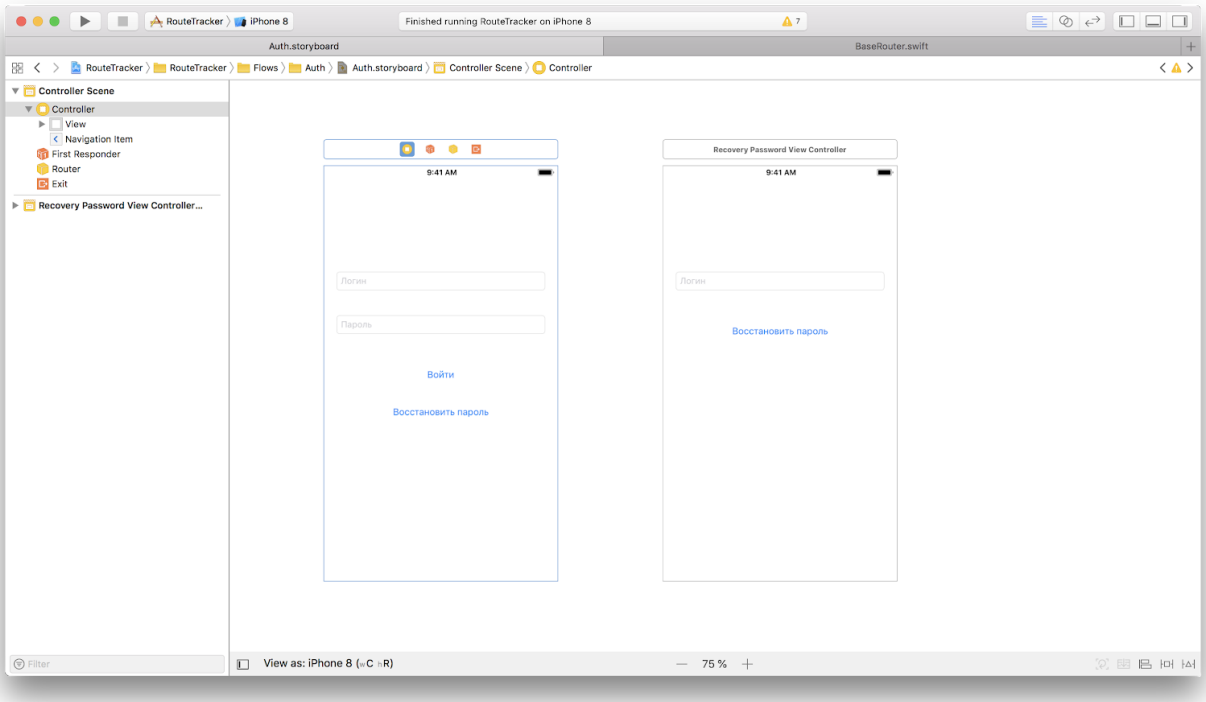
```
import UIKit

final class MainViewController: UIViewController {

    @IBOutlet weak var router: MainRouter!

    @IBAction func showMap(_ sender: Any) {
        router.toMap(usseles: "пример")
    }

    @IBAction func logout(_ sender: Any) {
        UserDefaults.standard.set(false, forKey: "isLogin")
        router.toLaunch()
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        router.prepare(for: segue, sender: sender)
    }
}

final class MainRouter: BaseRouter {
    func toMap(usseles: String) {
        perform(segue: "toMap") { (controller: MapViewController) in
            controller.usselesExampleVariable = usseles
        }
    }

    func toLaunch() {
        perform(segue: "toLaunch")
    }
}
```

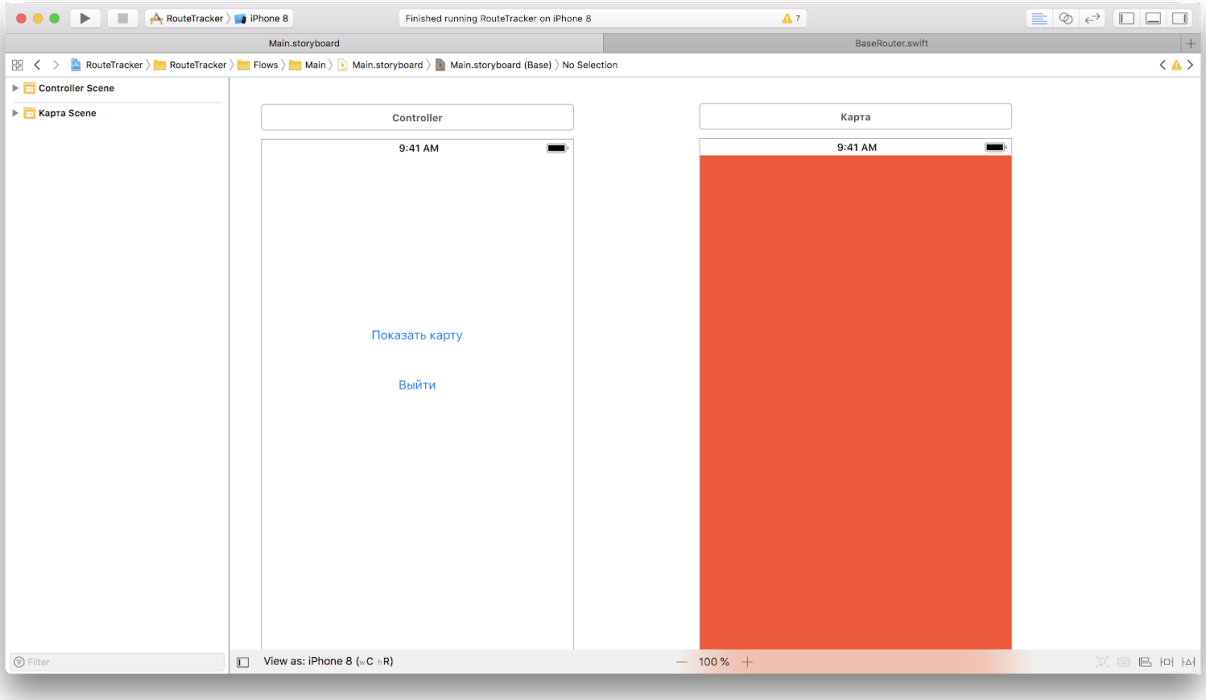
Здесь при переходе на экран карты используется замыкание и конфигурируемый контроллер.

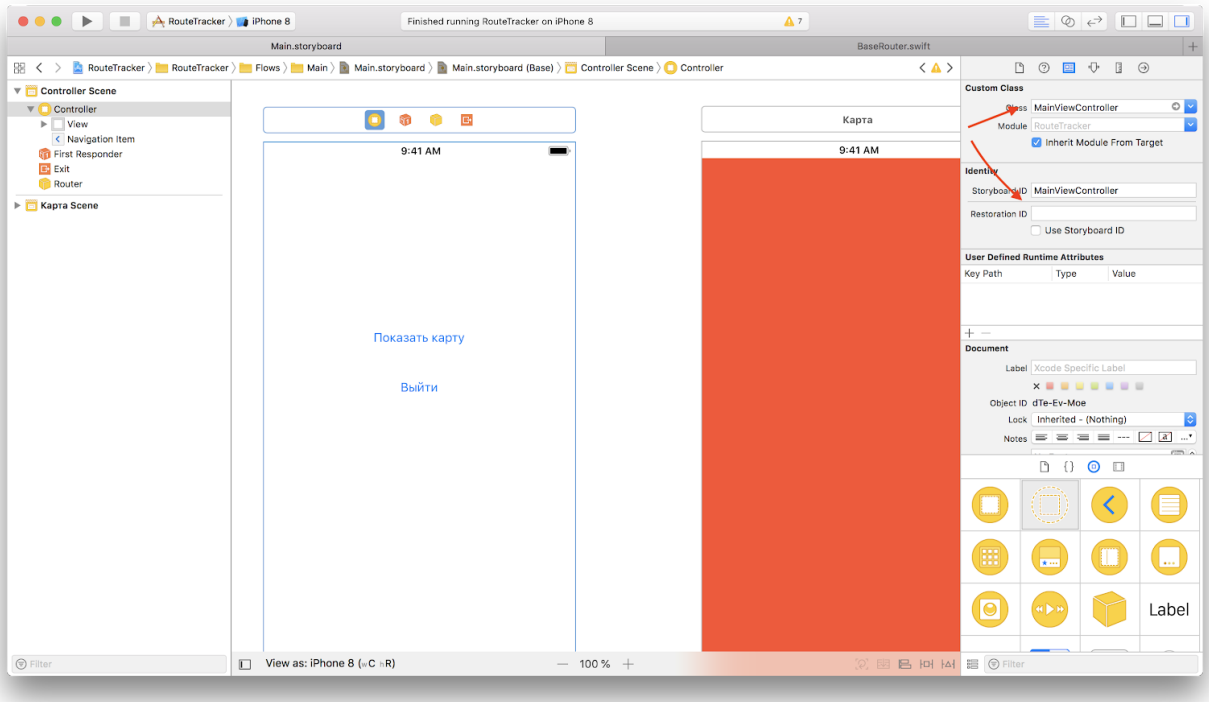
Вариант 3. Ручное создание и Router

Удалим все **Segue** и перепишем роутеры на ручное создание контроллеров. Вы также увидите, что инкапсуляция логики в роутерах позволяет не менять сами контроллеры.

Какие изменения нас ждут? Во-первых, нам не нужен больше сценарий **Launch,** просто удалим его. Во-вторых, уберём в настройках проекта сториборд, с которого будем загружаться. В-третьих, удалим все **segue** и **UINavigationController** со **storyboard.** В-четвертых, каждому контроллеру в поле **Storyboard ID** пропишем идентификатор, аналогичный его классу.


RouteTracker
iPhone 8
Finished running RouteTracker on iPhone 8
Auth.storyboard
BaseRouter.swift
Controller Scene
Controller
View
Navigation Item
First Responder
Router
Exit
Recovery Password View Controller...
9:41 AM
Логин
Пароль
Войти
Восстановить пароль
Recovery Password View Controller
9:41 AM
Логин
Восстановить пароль
View as: iPhone 8
75 %


RouteTracker
iPhone 8
Finished running RouteTracker on iPhone 8
Main.storyboard
BaseRouter.swift
Controller Scene
Карта Scene
Controller
9:41 AM
Показать карту
Выйти
Карта
9:41 AM
View as: iPhone 8
100 %



Теперь добавим хелпер, чтобы инстанцировать контроллеры из **storyboard:**

```
import UIKit

// Протокол для объектов, имеющих идентификатор в сториборде
protocol StoryboardIdentifiable {
    static var storyboardIdentifier: String { get }
}

// Расширение UIViewController,
// которое даёт совместимость с протоколом StoryboardIdentifiable
extension UIViewController: StoryboardIdentifiable { }

// Расширение протокола StoryboardIdentifiable для UIViewController,
// создающее идентификатор в сториборде, равный названию класса контроллера
extension StoryboardIdentifiable where Self: UIViewController {

    static var storyboardIdentifier: String {
        return String(describing: self)
    }
}

extension UIStoryboard {

    func instantiateViewController<T: UIViewController>(_: T.Type) -> T {
                                        guard     let     viewController     =
self.instantiateViewController(withIdentifier:   T.storyboardIdentifier)   as?   T
```

```
else {
                                fatalError("View  controller  с  идентификатором
\(T.storyboardIdentifier) не найден")
        }

        return viewController
    }

    func instantiateViewController<T: UIViewController>() -> T {
                                        guard    let    viewController    =
self.instantiateViewController(withIdentifier:  T.storyboardIdentifier)  as?  T
else {
                                fatalError("View  controller  с  идентификатором
\(T.storyboardIdentifier) не найден")
        }

        return viewController
    }
}
```

Затем перепишем базовый роутер:

```
import Foundation

class BaseRouter: NSObject {
    @IBOutlet weak var controller: UIViewController!

    func show(_ controller: UIViewController) {
        self.controller.show(controller, sender: nil)
    }

    func present(_ controller: UIViewController) {
        self.controller.present(controller, animated: true)
    }

    func setAsRoot(_ controller: UIViewController) {
        UIApplication.shared.keyWindow?.rootViewController = controller
    }

}
```

Как вы видите, вместо вызова **segue** мы перешли на вызов методов контроллера.

Добавим код запуска пользовательского интерфейса в **AppDelegate:**

```
import UIKit
import GoogleMaps

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        GMSServices.provideAPIKey("AIzaSyAuW-cBcK81Fq22yEn92y_fdBeGL8n6qq0")
```

```
        let controller: UIViewController
        if UserDefaults.standard.bool(forKey: "isLogin") {
            controller = UIStoryboard(name: "Main", bundle: nil)
                .instantiateViewController(MainViewController.self)
        } else {
            controller = UIStoryboard(name: "Auth", bundle: nil)
                .instantiateViewController(LoginViewController.self)
        }
        window = UIWindow()

         window?.rootViewController = UINavigationController(rootViewController:
controller)

        window?.makeKeyAndVisible()

        return true
    }

}
```

Обратите внимание, что мы не просто получаем контроллеры, но и оборачиваем их в **UINavigationController**.

Последний штрих: изменим роутеры. Контроллеры трогать не надо, так как контракт мы не меняли.

```
final class LoginRouter: BaseRouter {
    func toMain() {
        let controller = UIStoryboard(name: "Main", bundle: nil)
            .instantiateViewController(MainViewController.self)

        setAsRoot(UINavigationController(rootViewController: controller))
    }

    func onRecover() {
        let controller = UIStoryboard(name: "Auth", bundle: nil)
            .instantiateViewController(RecoveryPasswordViewController.self)

        show(controller)
    }
}
```

```
final class MainRouter: BaseRouter {
    func toMap(usseles: String) {
        let controller = UIStoryboard(name: "Main", bundle: nil)
            .instantiateViewController(MapViewController.self)

        controller.usselesExampleVariable = usseles

        show(controller)
    }

    func toLaunch() {
        let controller = UIStoryboard(name: "Auth", bundle: nil)
            .instantiateViewController(LoginViewController.self)
        setAsRoot(UINavigationController(rootViewController: controller))
```

```
    }
}
```

Готово. Можно запустить приложение и убедиться, что этот вариант тоже работает.

Вариант 4. Координаторы

Координаторы – тяжёлая артиллерия в арсенале навигации. Для большинства проектов они избыточны. Их стоит использовать, если в вашем проекте запутанная навигация, то есть порядок контроллеров не постоянен, а зависит от внешних факторов, если контроллеров просто много или они используются в разных сценариях. Но когда вы поймёте, как работают координаторы, использовать их станет довольно легко, и это можно будет делать в любом проекте.

Принцип работы координаторов основан на том, что контроллер не знает ничего о других контроллерах и не решает, когда и куда ему переходить: он имеет свойства-замыкания, которые вызываются, если необходимы внешние действия, а устанавливает эти свойства координатор в зависимости от контекста.

Для подготовки к использованию координаторов нам необходимо будет удалить все роутеры, включая базовый: они сейчас не нужны. Не забудьте удалить роутеры в **storyboard** (там они встроены как объекты), иначе получите ошибку на стадии выполнения. После удаления роутеров мы изменим контроллеры, добавим в них свойства с замыканиями, которые будут выполняться вместо переходов.

```
final class LoginViewController: UIViewController {

    enum Constants {
        static let login = "admin"
        static let password = "123456"
    }

    @IBOutlet weak var loginView: UITextField!
    @IBOutlet weak var passwordView: UITextField!

    var onLogin: (() -> Void)?
    var onRecover: (() -> Void)?


    @IBAction func login(_ sender: Any) {
        guard
            let login = loginView.text,
            let password = passwordView.text,
            login == Constants.login && password == Constants.password
        else {
            return
        }
        UserDefaults.standard.set(true, forKey: "isLogin")
        onLogin?()
    }

    @IBAction func recovery(_ sender: Any) {
        onRecover?()
    }
}
```

```
final class MainViewController: UIViewController {

    var onMap: ((String) -> Void)?
    var onLogout: (() -> Void)?

    @IBAction func showMap(_ sender: Any) {
        onMap?("пример")
    }

    @IBAction func logout(_ sender: Any) {
        UserDefaults.standard.set(false, forKey: "isLogin")
        onLogout?()
    }
}
```

Обратите внимание на свойства **onLogin**, **onRecover**, **onMap**, **onLogout**. Все они – замыкания, которые выполняются при определённых действиях – как правило, при нажатиях на кнопки. Но при этом в контроллерах не описано их поведение, то есть в них может быть любое действие, всё что угодно, что удовлетворяет типу замыкания.

Теперь приступим к координаторам. Напишем базовый координатор.

```
import Foundation

// Абстрактный класс-координатор
class BaseCoordinator {

    var childCoordinators: [BaseCoordinator] = []

    func start() {
// Переопределить в наследниках
    }

    func addDependency(_ coordinator: BaseCoordinator) {
        for element in childCoordinators where element === coordinator {
            return
        }
        childCoordinators.append(coordinator)
    }

    func removeDependency(_ coordinator: BaseCoordinator?) {
        guard
            childCoordinators.isEmpty == false,
            let coordinator = coordinator
            else { return }

         for (index, element) in childCoordinators.reversed().enumerated() where
element === coordinator {
            childCoordinators.remove(at: index)
            break
        }
    }

    func setAsRoot(_ controller: UIViewController) {
        UIApplication.shared.keyWindow?.rootViewController = controller
    }
}
```

Этот базовый класс не имеет никакого отношения к переходам, он содержит только странную на

первый взгляд логику – массив того же типа, что и сам класс, и методы добавления элементов в массив и их удаления. Всё дело в том, как используются координаторы. Они не хранятся в контроллерах, они существуют отдельно, можно даже сказать, поверх контроллеров. Каждый координатор – отдельный сценарий. Если мы, находясь в одном сценарии, переходим в подсценарий, мы создаём новый координатор. Таких образом, координаторы образуют граф, или дерево, где каждый сценарий – ветка. Но кто будет хранить в памяти все эти координаторы? Первый хранится в **AppDelegate**, но если координатор создается для показа подсценария, он и должен хранить на него ссылку.

Давайте напишем первый координатор:

```
import Foundation

final class ApplicationCoordinator: BaseCoordinator {

    override func start() {
        if UserDefaults.standard.bool(forKey: "isLogin") {
            toMain()
        } else {
            toAuth()
        }
    }

    private func toMain() {
// Создаём координатор главного сценария
        let coordinator = MainCoordinator()
// Устанавливаем ему поведение на завершение
// Так как подсценарий завершился, держать его в памяти больше не нужно
            self?.removeDependency(coordinator)
// Заново запустим главный координатор, чтобы выбрать следующий сценарий
            self?.start()
        }
// Сохраним ссылку на дочерний координатор, чтобы он не выгружался из памяти
        addDependency(coordinator)
// Запустим сценарий дочернего координатора
        coordinator.start()
    }

    private func toAuth() {
// Создаём координатор сценария авторизации
        let coordinator = AuthCoordinator()
// Устанавливаем ему поведение на завершение
        coordinator.onFinishFlow = { [weak self, weak coordinator] in
// Так как подсценарий завершился, держать его в памяти больше не нужно
            self?.removeDependency(coordinator)
// Заново запустим главный координатор, чтобы выбрать выбрать следующий
// сценарий
            self?.start()
        }
// Сохраним ссылку на дочерний координатор, чтобы он не выгружался из памяти
        addDependency(coordinator)
// Запустим сценарий дочернего координатора
        coordinator.start()
    }

}
```

И сразу, чтобы было понятно, код из **AppDelegate:**

```
import UIKit
import GoogleMaps

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    var coordinator: ApplicationCoordinator?


     func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        GMSServices.provideAPIKey("AIzaSyAuW-cBcK81Fq22yEn92y_fdBeGL8n6qq0")

        window = UIWindow()
        window?.makeKeyAndVisible()


        coordinator = ApplicationCoordinator()
        coordinator?.start()

        return true
    }
```

Теперь здесь нет кода установки первого контроллера. Мы подготавливаем и показываем пустое **UIWindow.** Также мы создаём базовый координатор и запускаем его.

При старте базовый координатор проверяет, авторизованы мы или нет, и переходит к соответствующему подсценарию. Методы запуска подсценариев – **toMain** и **toAuth.** В примере они очень похожи. Описывать их отдельно от кода очень сложно, поэтому самое время посмотреть на код ещё раз и перечитать комментарии. Единственное, что для вас пока непонятно, – что за свойства **onFinishFlow.** Это замыкания в дочерних координаторах, которые будет вызываться, когда те закончат свою работу.

Осталось рассмотреть координаторы каждого из сценариев.

```
final class AuthCoordinator: BaseCoordinator {

    var rootController: UINavigationController?
    var onFinishFlow: (() -> Void)?

    override func start() {
        showLoginModule()
    }

    private func showLoginModule() {
        let controller = UIStoryboard(name: "Auth", bundle: nil)
            .instantiateViewController(LoginViewController.self)

        controller.onRecover = { [weak self] in
            self?.showRecoverModule()
        }

        controller.onLogin = { [weak self] in
            self?.onFinishFlow?()
        }

               let  rootController  =  UINavigationController(rootViewController:
```

```
controller)
        setAsRoot(rootController)
        self.rootController = rootController
    }

    private func showRecoverModule() {
        let controller = UIStoryboard(name: "Auth", bundle: nil)
            .instantiateViewController(RecoveryPasswordViewController.self)
        rootController?.pushViewController(controller, animated: true)
    }

}
```

При запуске координатора мы показываем модуль (часто контроллеры называют модулями, подразумевая не сам контроллер, а его семантический смысл) входа. При этом мы получаем экземпляр контроллера и устанавливаем его замыкания – те самые, которые мы определили в контроллерах.

На **onRecover** мы показываем модуль восстановления пароля. На **onLogin** мы вызываем **onFinishFlow** у координатора. Это логично, так как при успешном входе процесс авторизации закончен, а значит, и сценарий завершился.

Настроенный контроллер мы заворачиваем в **UINavigationController** и устанавливаем как базовый для **UIWindow.** Получившийся **UINavigationController** мы сохраняем в свойстве координатора, чтобы иметь возможность добавлять в него другие контроллеры.

Показ контроллера восстановления пароля уже проще, так как у него нет параметров: мы просто создаём его экземпляр и добавляем в **UINavigationController.**

```
final class MainCoordinator: BaseCoordinator {

    var rootController: UINavigationController?
    var onFinishFlow: (() -> Void)?

    override func start() {
        showMainModule()
    }

    private func showMainModule() {
        let controller = UIStoryboard(name: "Main", bundle: nil)
            .instantiateViewController(MainViewController.self)

        controller.onMap = { [weak self] usseles in
            self?.showMapModule(usseles: usseles)
        }

        controller.onLogount = { [weak self] in
            self?.onFinishFlow?()
        }

            let rootController = UINavigationController(rootViewController:
controller)
        setAsRoot(rootController)
        self.rootController = rootController
    }

    private func showMapModule(usseles: String) {
        let controller = UIStoryboard(name: "Main", bundle: nil)
            .instantiateViewController(MapViewController.self)
```

```
        controller.usselesExampleVariable = usseles

        rootController?.pushViewController(controller, animated: true)
    }

}
```

Координатор главного сценария работает по аналогичному принципу, только он ещё и извлекает строковое значение из главного контроллера и передаёт его в контроллер карты.

Практическое задание

1. Добавьте в ваше приложение класс **User** со свойствами **login** и **password.**
2. Сделайте класс **User** совместимым с **Realm.**
3. Сделайте поле **login** ключевым полем **primaryKey.**
4. Добавьте контроллер для входа, где можно ввести логин и пароль, а также воспользоваться кнопками «Вход» и «Регистрация».
5. При нажатии на кнопку «Вход» ищите пользователя в базе данных по его логину, затем проверьте пароль. Если данные верны, авторизуйте пользователя.
6. На экране авторизации добавьте поля для ввода логина и пароля, а также кнопку «Зарегистрироваться».
7. При нажатии на кнопку регистрации создайте пользователя и запишите в базу данных.
8. Прежде чем делать запись, поищите в базе пользователя с таким логином. Если он существует, измените ему пароль (да, это нелогично с точки зрения здравого смысла, но для обучения – хороший вариант).
9. Настройте навигацию в приложении понравившимся вам способом. Не стоит выбирать координаторы только потому, что они самые навороченные. Помните, что для вашего приложения они избыточны, но если вам хочется, можете выбрать и их для учебного проекта, это не будет ошибкой. Подумайте, достаточно ли хорошо вы их поняли, чтобы успеть завершить ДЗ.

Дополнительные материалы

1. Основные практики обеспечения безопасности iOS-приложений.
2. 10 самых опасных угроз для веб-приложений по версии Owasp в 2017 г.

Используемая литература

1. Сайт библиотеки ReactiveX.
2. Репозиторий библиотеки ReactiveX.