



## Урок 8

# AVFoundation

Работа с камерой и микрофоном телефона

[Камера](#)

[Микрофон](#)

[Практическое задание](#)

[Дополнительные материалы](#)

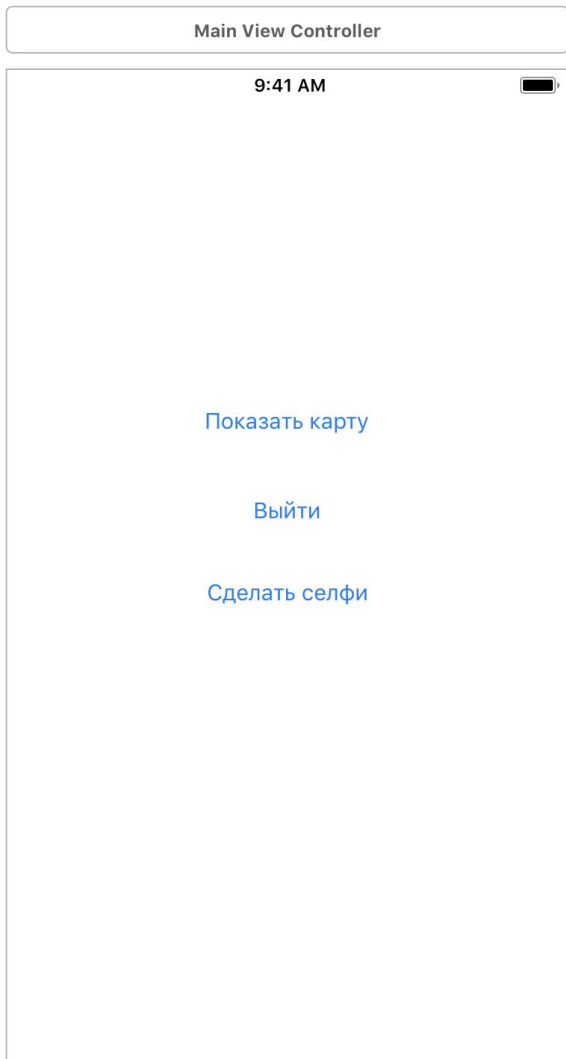
[Используемая литература](#)

# Камера

Часто в приложениях нам необходимо получить снимок с камеры. Для этого у нас есть два пути:

- Воспользоваться стандартным компонентом. Он прост, но с задачей справляется.
- Написать своё решение. Это не намного сложнее, но позволяет полностью настроить внешний вид камеры и её работу.

Начнем со стандартного решения – **UIImagePickerController**. Это целый комплекс контроллеров и инструментов для работы со снимками, скрывающихся за очень простым интерфейсом. Работать с ним тоже очень просто. Давайте добавим в приложение кнопку для получения селфи.



Протянем **IBOutlet** в контроллер – **@IBAction func takePicture(\_ sender: Any)**.

И напишем логику:

```
@IBAction func takePicture(_ sender: Any) {  
    // Проверка, поддерживает ли устройство камеру  
    guard UIImagePickerController.isSourceTypeAvailable(.camera) else {  
        return  
    }  
    // Создаём контроллер и настраиваем его
```

```

        let imagePickerController = UIImagePickerController()
// Источник изображений: камера
        imagePickerController.sourceType = .camera
// Изображение можно редактировать
        imagePickerController.allowsEditing = true
        imagePickerController.delegate = self

// Показываем контроллер
        present(imagePickerController, animated: true)
    }

```

Здесь мы создаём и настраиваем **UIImagePickerController**. Прежде чем приступить к настройке, проверяем, что у девайса, на котором мы запускаем приложение, есть камера. Если её нет, нет и смысла пытаться сделать снимок. Далее мы устанавливаем в качестве источника изображений камеру. В качестве альтернативы мы можем выбрать галерею изображений. Ставим флаг, позволяющий редактировать фото. Это значит, что после выбора изображения появится экран, где мы сможем обрезать часть снимка. Установим делегата и презентуем контроллер. Так как **UIImagePickerController** – просто контроллер, мы можем его презентовать, но не добавлять в стек навигации к **UINavigationController**, это запрещено.

Теперь реализуем методы делегата.

```

extension MainViewController: UINavigationControllerDelegate,
UIImagePickerControllerDelegate {

    func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
// Если нажали на кнопку Отмена, то UIImagePickerController надо закрыть
        picker.dismiss(animated: true)
    }

    func imagePickerController(
        _ picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
        [String: Any]) {
// Мы получили медиа от контроллера
// Изображение надо достать из словаря info
        let image = extractImage(from: info)
        print(image!)
// Закрываем UIImagePickerController
        picker.dismiss(animated: true)
    }

// Метод, извлекающий изображение
    private func extractImage(from info: [String: Any]) -> UIImage? {
// Пытаемся извлечь отредактированное изображение
        if let image = info[UIImagePickerControllerEditedImage] as? UIImage {
            return image
// Пытаемся извлечь оригинальное
        } else if let image = info[UIImagePickerControllerOriginalImage] as?
        UIImage {
            return image
        } else {
// Если изображение не получено, возвращаем nil
            return nil
        }
    }
}

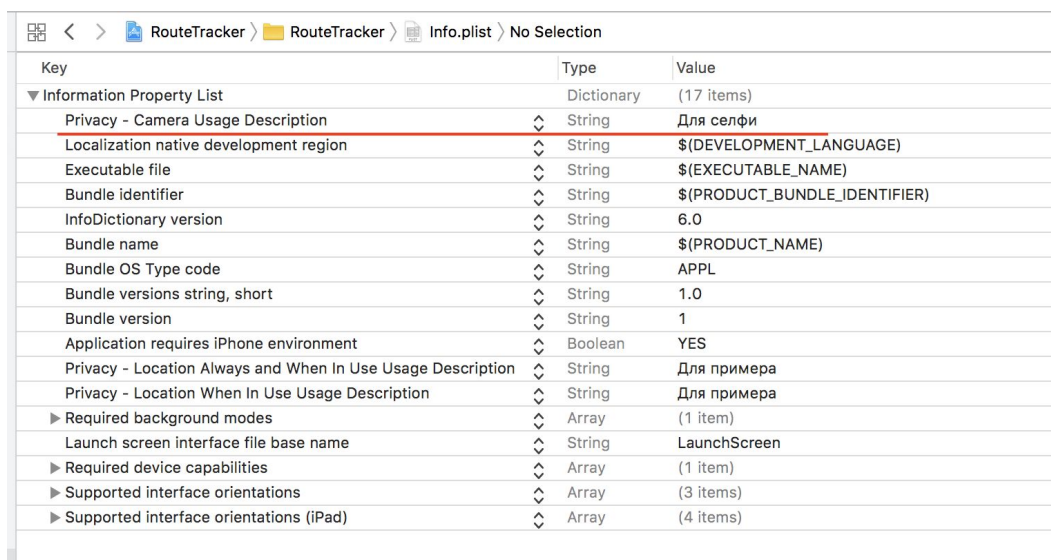
```

У контроллера два метода: **func imagePickerControllerDidCancel(\_ picker: UIImagePickerController)**

и `func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [String: Any])`.

Первый вызывается, когда пользователь нажимает кнопку **Отмена**, второй – когда мы получаем снимок или видео (да, контроллер можно настроить для получения видео). К сожалению, мы получаем не само изображение, а словарь с разной информацией, поэтому изображение надо извлечь. Для этого мы и написали метод `func extractImage(from info: [String: Any]) -> UIImage?`. Он принимает словарь и ищет изображение. Если юзер редактировал изображение, оно будет под ключом `UIImagePickerControllerEditedImage`, если не редактировал – под ключом `UIImagePickerControllerOriginalImage`.

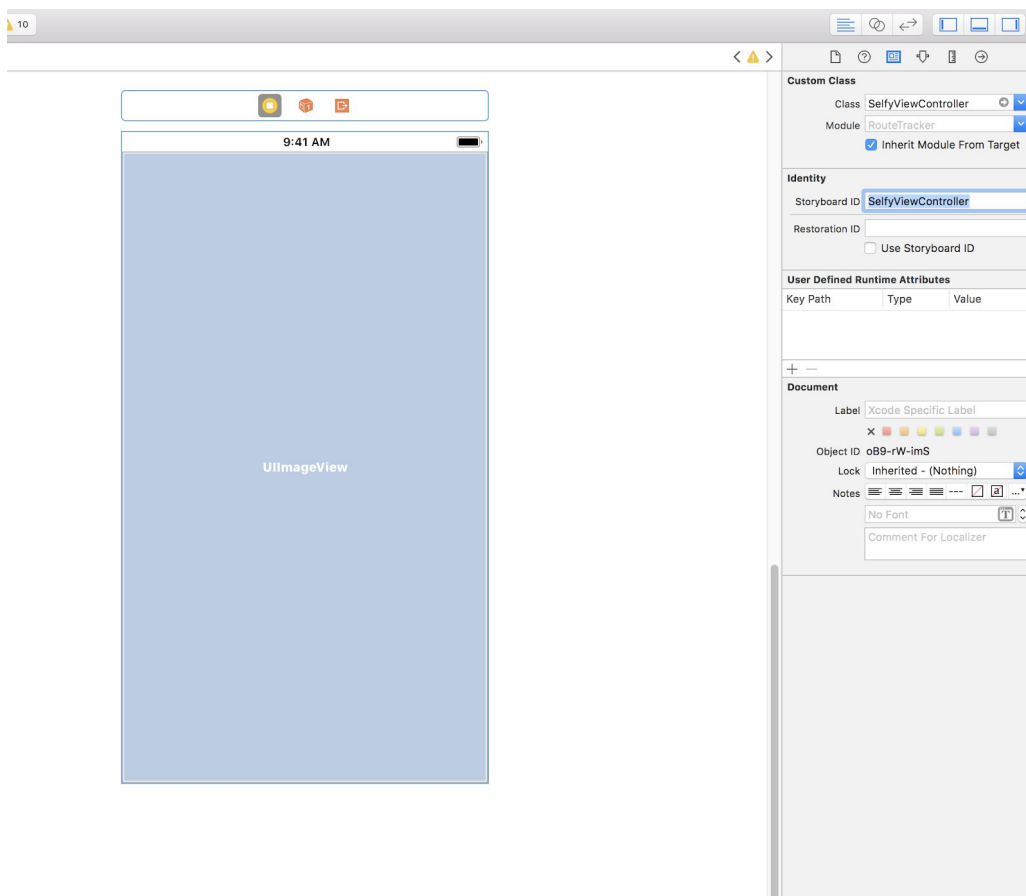
Последний шаг для работы с камерой – добавление в `Plist.info` информации о том, зачем нам доступ.



Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
Privacy - Camera Usage Description	String	Для селфи
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES
Privacy - Location Always and When In Use Usage Description	String	Для примера
Privacy - Location When In Use Usage Description	String	Для примера
▶ Required background modes	Array	(1 item)
Launch screen interface file base name	String	LaunchScreen
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
▶ Supported interface orientations (iPad)	Array	(4 items)

Теперь можно запустить приложение. После нажатия на кнопку оно выдаст запрос на доступ к камере, мы его примем и увидим интерфейс камеры. Сделаем снимок, после чего сможем его обрезать, и он упадёт в приложение.

Добавим контроллер для показа селфи.



**SelfyViewController** содержит только **UIImageView** на весь экран, где и будет показано изображение. Протянем от него **IBOutlet**, добавим свойство для хранения изображения и напишем показ этого изображения.

```
class SelfyViewController: UIViewController {
    @IBOutlet weak var imageView: UIImageView!
    var image: UIImage?

    override func viewDidLoad() {
        super.viewDidLoad()
        imageView.image = image
    }
}
```

Осталось настроить навигацию. Добавим замыкание **var onTakePicture: ((UIImage) -> Void)?** в **MainViewController**. Таким образом мы сможем настроить действие после получения фото. Перепишем метод делегата по получению фото.

```
func imagePickerController(
    _ picker: UIImagePickerController, didFinishPickingMediaWithInfo info:
    [String: Any]) {
    // Закрываем UIImagePickerController
    picker.dismiss(animated: true) { [weak self] in
    // После того, как он закроется, извлечём изображение
        guard let image = self?.extractImage(from: info) else { return }
    // Если оно будет извлечено, выполним действие на его получение
        self?.onTakePicture?(image)
    }
}
```

Теперь мы сразу закрываем контроллер. Когда он закроется, мы извлечём изображение и выполним замыкание.

Теперь настроим координатор, чтобы добавить переходы:

```
private func funcShowSelfyModule(image: UIImage) {
    let controller = UIStoryboard(name: "Main", bundle: nil)
        .instantiateViewController(SelfyViewController.self)

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

Это метод для показа селфи-контроллера. Он принимает изображение аргументом, создает экземпляр контроллера и устанавливает ему изображение.

Осталось настроить контроллер с меню:

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

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

    controller.onLogout = { [weak self] in
        self?.onFinishFlow?()
    }
    // Добавляем настройку замыкания для показа фото
    controller.onTakePicture = { [weak self] image in
        self?.funcShowSelfyModule(image: image)
    }

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

Здесь мы добавили настройку замыкания **onTakePicture**. Теперь мы можем увидеть сделанное фото.

Напишем примитивный контроллер для получения изображений с камеры. Ниже приведен огромный код контроллера. Чтобы понять все детали того, что там происходит, внимательно прочтите комментарии в коде и пояснения после него.

```
import UIKit
// Подключаем фреймворк для работы с медиа
import AVFoundation

class PhotoViewController: UIViewController {
    // Замыкание, которое будет вызвано, когда мы получим изображение
    var onTakePicture: ((UIImage) -> Void)?

    // Сессия видеозахвата
    var captureSession: AVCaptureSession?
    // Объект, отвечающий за получение фотоданных
```

```

    var cameraOutput: AVCapturePhotoOutput?
    // Слой, который отображает данные с камеры в реальном времени
    var previewLayer: AVCaptureVideoPreviewLayer?

    // Переменная, которая запомнит, в какой ориентации был телефон
    // во время снимка
    var deviceOrientationOnCapture: UIDeviceOrientation?

    override func viewDidLoad() {
        super.viewDidLoad()

        // При изменении вьюшек меняем размер слоя, иначе он будет некрасиво
        // смотреться при повороте
        previewLayer?.frame = view.layer.bounds
    }

    // Отслеживаем повороты устройства
    override func viewWillTransition(to size: CGSize, with coordinator:
    UINavigationControllerTransitionCoordinator) {

        // Поддерживает ли наш слой изменение ориентации
        if previewLayer?.connection?.isVideoOrientationSupported == true {
            // Устанавливаем текущую ориентацию, иначе изображение будет перевёрнутым
            previewLayer?.connection?.videoOrientation =
            UIDevice.current.orientation.getAVCaptureVideoOrientationFromDevice()
        }
    }

    override func viewDidAppear() {
        super.viewDidAppear()

        // Создаём объект, отвечающий за получение фотоданных
        let cameraOutput = makeCameraOutput()
        self.cameraOutput = cameraOutput

        // Создаём сессию видеозахвата
        guard let captureSession = makeCameraSession(cameraOutput: cameraOutput)
        else { return }
        self.captureSession = captureSession

        // Настраиваем слой, который отображает данные с камеры в реальном времени
        configurePreviewLayer(captureSession, cameraOutput)
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        // Как только экран становится виден, запускаем сессию видеозахвата
        // В этот момент мы подключаемся к камере и считываем с неё изображение
        // в реальном времени
        captureSession?.startRunning()
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        // Как только экран закрывается, останавливаем сессию видеозахвата
        captureSession?.stopRunning()
    }

    @IBAction func takePicture(_ sender: Any) {
        // Создаём настройки получения снимка

```

```

// Снимок будет получен в формате jpeg
    let settings = AVCapturePhotoSettings(format: [AVVideoCodecKey:
AVVideoCodecType.jpeg])
// Делаем снимок (получаем текущую картинку в формате, заданном выше)
    cameraOutput?.capturePhoto(with: settings, delegate: self)
}

// Ищем среди датчиков устройства обычную переднюю камеру
func cameraDeviceFind() -> AVCaptureDevice? {
// Настройки поиска
    let discoverySession = AVCaptureDevice.DiscoverySession(
        deviceTypes: [.builtInWideAngleCamera], // Обычная передняя камера
        mediaType: .video, // Тип устройства - камера
        position: .front // Передняя камера
    )
    let devices = discoverySession.devices // Все найденные устройства
// Первое найденное устройство типа «передняя камера»
    let device = devices
        .first(where: { $0.hasMediaType(AVMediaType.video) && $0.position ==
        .front })
    return device
}

func makeCameraOutput() -> AVCapturePhotoOutput {
// Создаём объект, который будет получать данные из сессии видеозахвата
// в момент получения снимка
    let cameraOutput = AVCapturePhotoOutput()
// Здесь мы можем указать различные настройки,
// в документации их описано намного больше
// Высокое разрешение
    cameraOutput.isHighResolutionCaptureEnabled = true
// Живые фото отключены
    cameraOutput.isLivePhotoCaptureEnabled = false
    return cameraOutput
}

func makeCameraSession(cameraOutput: AVCapturePhotoOutput) ->
AVCaptureSession? {
// Создаём сессию видеозахвата
    let captureSession = AVCaptureSession()

    guard
// Если мы нашли переднюю камеру
        let device = cameraDeviceFind(),
// Если смогли создать из неё устройство получения данных
        let input = try? AVCaptureDeviceInput(device: device),
// Если можем добавить в сессию в качестве источника данных
        captureSession.canAddInput(input) else {
        return nil
    }

// то добавляем в сессию в качестве источника данных
    captureSession.addInput(input)
// Устанавливаем предустановленный режим съёмки фото
    captureSession.sessionPreset = AVCaptureSession.Preset.photo
// Добавляем объект, отвечающий за получение фотоданных
// в качестве приёмника данных
    captureSession.addOutput(cameraOutput)

    return captureSession
}

```



```

    }

    func configurePreviewLayer(_ captureSession: AVCaptureSession,
cameraOutput: AVCapturePhotoOutput) {
// Создаём слой, который отображает данные с камеры в реальном времени
    let previewLayer = AVCaptureVideoPreviewLayer(session: captureSession)
// Добавляем его на корневую вьюшку экрана
    view.layer.insertSublayer(previewLayer, at: 0)
// Устанавливаем размеры слоя равными размерам текущей вьюшки
    previewLayer.frame = view.layer.bounds

    self.previewLayer = previewLayer
}

}

extension PhotoViewController: AVCapturePhotoCaptureDelegate {
    func photoOutput(_ output: AVCapturePhotoOutput, didFinishProcessingPhoto
photo: AVCapturePhoto, error: Error?) {
// Преобразуем фотоданные снимка в изображение
        guard
// Если мы получили фотоданные
            let imageData = photo.fileDataRepresentation(),
// Если смогли сделать из них изображение
            let uiImage = UIImage(data: imageData),
// Если извлекли кор изображения
            let cgImage = uiImage.cgImage,
// Если есть запомненная ориентация
            let deviceOrientationOnCapture = self.deviceOrientationOnCapture
        else {
            return
        }

// то создаём изображение из кор изображения, при этом разворачиваем его,
// чтобы оно не оказалось перевёрнутым
        let image = UIImage(
            cgImage: cgImage,
            scale: 1.0,
            orientation:
deviceOrientationOnCapture.getUIImageOrientationFromDevice()
        )

// Вызываем замыкание, передав в него изображение
        onTakePicture?(image)
    }

    func photoOutput(_ output: AVCapturePhotoOutput, willBeginCaptureFor
resolvedSettings: AVCaptureResolvedPhotoSettings) {
// Перед началом извлечения фотоданных запоминаем текущую ориентацию
устройства
        deviceOrientationOnCapture = UIDevice.current.orientation
    }
}

fileprivate extension UIDeviceOrientation {
// Преобразование текущей ориентации экрана в ориентацию изображения
    func getUIImageOrientationFromDevice() -> UIImageOrientation {
        let orientation: UIImageOrientation

```

```

        switch self {
        case .portrait,
            .faceUp:
            orientation = .right
        case .portraitUpsideDown,
            .faceDown:
            orientation = .left
        case .landscapeLeft:
            orientation = .down
        case .landscapeRight:
            orientation = .up
        case .unknown:
            orientation = .down
        }
        return orientation
    }
}

// Преобразование текущей ориентации экрана в ориентацию видеозахвата
func getAVCaptureVideoOrientationFromDevice() -> AVCaptureVideoOrientation {
    let orientation: AVCaptureVideoOrientation
    switch self {
    case .portrait,
        .faceUp:
        orientation = .portrait
    case .portraitUpsideDown,
        .faceDown:
        orientation = .portraitUpsideDown
    case .landscapeLeft:
        orientation = .landscapeRight
    case .landscapeRight:
        orientation = .landscapeLeft
    case .unknown:
        orientation = .landscapeLeft
    }
    return orientation
}
}

```

Листинг вышел очень объёмным, потому что при ручной работе с камерой нам доступно много параметров, которые мы можем менять. За счет этого мы можем очень гибко конфигурировать камеру. Давайте пройдемся по объектам, которые мы использовали.

**AVCaptureSession** – сессия для захвата видео. Сама по себе не создает снимков, но она, как клей, связывает все остальные компоненты вместе. У неё мы настраиваем устройство, которое будет служить источником данных, объект-обработчик данных. Настраивая внутренние параметры, создаём оптимальный режим для работы с фото.

**AVCaptureDevice** – устройство, которое будет использовано в качестве источника данных. Это может быть камера или микрофон. В телефоне, как правило, несколько таких устройств, и нужно определиться, с каким из них мы будем работать.

**AVCaptureVideoPreviewLayer** – простой слой, который можно добавить в любую **UIView**. Его назначение – выводить поток данных, который проходит через сессию в текущий момент. Именно его видит пользователь. Его можно модифицировать, как любой другой **CALayer**.

**AVCapturePhotoOutput** – объект, который работает приёмником данных из сессии в момент получения снимка. Именно он отвечает за то, как эти данные будут обработаны. В нём мы можем указывать качество, формат и ещё много других настроек. Полный их перечень можно посмотреть в документации. Он – наследник класса **AVCaptureOutput** и служит только для получения

изображений. Но есть и другие наследники, способные работать с другими форматами. Например, **AVCaptureVideoDataOutput** извлекает видео.

**AVCapturePhotoCaptureDelegate** – сущность, получающая информацию о процессе извлечения фотографии. Этот процесс требует времени. Делегат будет уведомлён обо всех этапах. Мы реализовали оба. **willBeginCaptureFor** вызывается непосредственно перед извлечением, в нём мы запоминаем ориентацию экрана. **didFinishProcessingPhoto** вызывается, когда данные уже извлечены. На основе данных уже можно сразу получить **UIImage**, но у нас написано больше кода. Дело в том, что мы получаем фото в неверной ориентации, ведь ориентация камеры, не меняется, если мы перевернём телефон. А значит, нам надо посмотреть, в каком положении находился телефон в момент съёмки, и повернуть изображение так, чтобы ориентация телефона совпала с ориентацией фотографии.

Зачем может понадобиться писать столько кода, когда есть более простое решение? Дело в гибкости: если вам понадобится сделать кастомный интерфейс или настройки камеры, у вас не получится просто воспользоваться **UIImagePickerController**.

```
final class MainViewController: UIViewController {

    <...>
    var onPhoto: (() -> Void)?

    <...>
    @IBAction func takePicture(_ sender: Any) {
        onPhoto?()
    }
}
```

В главном контроллере добавилось замыкание для перехода к контроллеру камеры:

```
private func showPhotoModule() {
    let controller = UIStoryboard(name: "Main", bundle: nil)
        .instantiateViewController(PhotoViewController.self)
    controller.onTakePicture = { [weak self] image in
        self?.showSelfyModule(image: image)
    }
    rootController?.pushViewController(controller, animated: true)
}
```

```
private func showMainModule() {
    <...>
    // Добавляем настройку замыкания для показа фото
    controller.onPhoto = { [weak self] in
        self?.showPhotoModule()
    }
    <...>
}
```

В координаторе мы добавили метод **showPhotoModule** для показа нового контроллера, который вызвали при наступлении события **onTakePicture** у главного экрана.

# Микрофон

Напоследок рассмотрим, как записать аудио с микрофона.

```
// Класс, позволяющий записать аудио с микрофона
var recorder: AVAudioRecorder?
func recordNewAudio() {
// URL директории с документами для хранения аудиозаписи
let documentsURL = FileManager.default.urls(for: .cachesDirectory, in:
.userDomainMask)[0]
// URL файла, в который будет записываться аудио
let url =
documentsURL.appendingPathComponent("newRecord_\(Date().timeIntervalSince1970).m
4a")
do {
// Переключаем главную аудиосессию в режим записи
try
AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayAndRecord)

// Создаём экземпляр AVAudioRecorder для записи в файл
recorder = try AVAudioRecorder(url: url, settings: [
    AVFormatIDKey: kAudioFormatMPEG4AAC, // Формат, в который мы
// пишем, в данном
// случае - aac
    AVSampleRateKey: 44100.0, // Битрейт записи
    AVNumberOfChannelsKey: 2 // Количество каналов,
// 2 - стерео
])

} catch {
// Выводим ошибку, если не вышло создать AVAudioRecorder
print(error.localizedDescription)
}
recorder?.delegate = self
// Запускаем запись
recorder?.prepareToRecord()
recorder?.record()
}

func stopRecord() {
// Если идёт запись
if recorder?.isRecording == true {
// Останавливаем её
recorder?.stop()
}
}

extension PhotoViewController: AVAudioRecorderDelegate {
func audioRecorderDidFinishRecording(_ recorder: AVAudioRecorder,
successfully flag: Bool) {
// Если запись закончилась, переключаем главную аудиосессию
// в режим проигрывания аудио
try?
AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayback)
}

func audioRecorderBeginInterruption(_ recorder: AVAudioRecorder) {
// Если что-то прервало запись, удалим AVAudioRecorder
```

```
        self.recorder = nil
// И переключаем главную аудиосессию в режим проигрывания аудио
AVAudioSession.sharedInstance().setCategory(AVAudioSessionCategoryPlayback)
    }
}
```

Код, приведённый выше, можно разместить в любом классе, главное – произвести импорт **import AVFoundation** и переименовать класс, к которому мы пишем расширение для имплементации **AVAudioRecorderDelegate**.

Так можно получить звук с помощью класса **AVAudioRecorder**. Конечно, мы могли бы воспользоваться методом, как и в случае с камерой, через сессии и девайсы, но **AVAudioRecorder** проще и при этом достаточно гибок для большинства ситуаций. Нам надо создать его экземпляр, указать формат аудиозаписи, url файла и запустить запись. Важно не забыть изменить режим аудиосессии, так как она одна на всё приложение и мы не можем одновременно проигрывать и записывать звук, поэтому мы переходим в режим записи прежде чем начать записывать, и возвращаемся в режим проигрывания после того, как завершим запись, иначе она прервётся, например, при входящем звонке.

## Практическое задание

1. Добавить в приложение возможность сделать аватарку и разместить её на маркере в текущей точке маршрута при отслеживании движения.
2. Для получения фото используйте **UIImagePickerController**.
3. Полученное изображение сохраните на диск.
4. Если пользователь переснимет аватар, новый файл должен заменять старый.
5. На карту добавьте маркер, который будет отображаться в текущей точке маршрута.
6. Как только текущая точка маршрута изменится, меняйте положение маркера.

## Дополнительные материалы

1. [Раздел документации, посвящённый работе с AVFoundation](#).

## Используемая литература

1. [Раздел документации, посвящённый работе с AVFoundation](#).