var player: AVPlayer?

func setupPlayer() {

if let videoURL = URL(string: "https://example.com/video.mp4") {

player = AVPlayer(url: videoURL)

}

}

func presentPlayerViewController() {

let playerViewController = AVPlayerViewController()

playerViewController.player = player

present(playerViewController, animated: true) {

playerViewController.player!.play()

}

}

// Call setupPlayer and presentPlayerViewController from your ViewController

NotificationCenter.default.addObserver(forName: .AVPlayerItemDidPlayToEndTime, object: player.currentItem, queue: .main) { \_ in

player.seek(to: CMTime.zero)

player.play()

}

let playerLayer = AVPlayerLayer(player: player)

playerLayer.frame = myView.bounds

myView.layer.addSublayer(playerLayer)

player.addObserver(self, forKeyPath: "status", options: [.new, .old], context: nil)

override func observeValue(forKeyPath keyPath: String?, of object: Any?, change: [NSKeyValueChangeKey : Any]?, context: UnsafeMutableRawPointer?) {

if keyPath == "status" {

if player.status == .readyToPlay {

// The player is ready

} else if player.status == .failed {

// Handle error

}

}

}

class AudioEngineManager {

var audioEngine: AVAudioEngine

var playerNode: AVAudioPlayerNode

var mixerNode: AVAudioMixerNode

var reverbNode: AVAudioUnitReverb

init() {

// Initialize the audio engine components

audioEngine = AVAudioEngine()

playerNode = AVAudioPlayerNode()

mixerNode = audioEngine.mainMixerNode

reverbNode = AVAudioUnitReverb()

// Attach nodes to the engine

audioEngine.attach(playerNode)

audioEngine.attach(reverbNode)

// Connect nodes

audioEngine.connect(playerNode, to: reverbNode, format: nil)

audioEngine.connect(reverbNode, to: mixerNode, format: nil)

// Setup reverb effect

reverbNode.loadFactoryPreset(.mediumHall)

reverbNode.wetDryMix = 50

// Configure the audio session

do {

try AVAudioSession.sharedInstance().setCategory(.playAndRecord,

mode: .default)

try AVAudioSession.sharedInstance().setActive(true)

} catch {

print("Audio session configuration error: \(error)")

}

// Register for audio interruption notifications

NotificationCenter.default.addObserver(self,

selector: #selector(handleAudioSessionInterruption),

name: AVAudioSession.interruptionNotification,

object: AVAudioSession.sharedInstance())

}

@objc func handleAudioSessionInterruption(notification: Notification) {

// Handle audio session interruption

}

func playAudioFile() {

// Load and play an audio file

guard let fileURL = Bundle.main.url(forResource: "audio",

withExtension: "mp3") else { return }

do {

let audioFile = try AVAudioFile(forReading: fileURL)

playerNode.scheduleFile(audioFile, at: nil, completionHandler: nil)

if !audioEngine.isRunning {

try audioEngine.start()

}

playerNode.play()

} catch {

print("Error playing audio file: \(error)")

}

}

}

var captureSession = AVCaptureSession()

captureSession.sessionPreset = .high

let movieOutput = AVCaptureMovieFileOutput()

if captureSession.canAddOutput(movieOutput) {

captureSession.addOutput(movieOutput)

} else {

fatalError("Could not add movie output to capture session.")

}

let composition = AVMutableComposition()

guard let compositionTrack = composition.addMutableTrack(withMediaType: .video, preferredTrackID: kCMPersistentTrackID\_Invalid) else {

fatalError("Unable to create composition track.")

}

// Assume clip1 and clip2 are AVURLAssets of video files

try compositionTrack.insertTimeRange(CMTimeRangeMake(start: .zero, duration: clip1.duration), of: clip1.tracks(withMediaType: .video)[0], at: .zero)

try compositionTrack.insertTimeRange(CMTimeRangeMake(start: .zero, duration: clip2.duration), of: clip2.tracks(withMediaType: .video)[0], at: clip1.duration)

let asset = AVAsset(url: videoFileURL)

let metadata = asset.metadata

let newMetadataItem = AVMutableMetadataItem()

newMetadataItem.key = AVMetadataKey.commonKeyDescription as NSString

newMetadataItem.keySpace = AVMetadataKeySpace.common

newMetadataItem.value = "New description" as NSString

let exportSession = AVAssetExportSession(asset: asset, presetName: AVAssetExportPresetHighestQuality)!

exportSession.metadata = [newMetadataItem]

let captureSession = AVCaptureSession()

captureSession.sessionPreset = .high

let videoDevice = AVCaptureDevice.default(.builtInWideAngleCamera, for: .video, position: .front)

let audioDevice = AVCaptureDevice.default(.builtInMicrophone, for: .audio, position: .unspecified)

do {

let videoInput = try AVCaptureDeviceInput(device: videoDevice!)

let audioInput = try AVCaptureDeviceInput(device: audioDevice!)

captureSession.addInput(videoInput)

captureSession.addInput(audioInput)

} catch {

print("Error configuring capture session: \(error)")

}

let videoOutput = AVCaptureVideoDataOutput()

videoOutput.setSampleBufferDelegate(self, queue: DispatchQueue(label: "videoQueue"))

captureSession.addOutput(videoOutput)

let audioOutput = AVCaptureAudioDataOutput()

audioOutput.setSampleBufferDelegate(self, queue: DispatchQueue(label: "audioQueue"))

captureSession.addOutput(audioOutput)

func captureOutput(\_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer, from connection: AVCaptureConnection) {

// Process the sampleBuffer (a frame of video) here

}

let filter = CIFilter(name: "CISepiaTone")!

let context = CIContext()

func applyFilter(to sampleBuffer: CMSampleBuffer) {

guard let pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer) else {

return

}

let ciImage = CIImage(cvPixelBuffer: pixelBuffer)

filter.setValue(ciImage, forKey: kCIInputImageKey)

if let outputImage = filter.outputImage {

context.render(outputImage, to: pixelBuffer)

}

}

let audioEngine = AVAudioEngine()

let inputNode = audioEngine.inputNode

let format = inputNode.inputFormat(forBus: 0)

inputNode.installTap(onBus: 0, bufferSize: 1024, format: format) { (buffer, time) -> Void in

// Process the audio buffer here

}

audioEngine.prepare()

do {

try audioEngine.start()

} catch {

print("AudioEngine didn't start: \(error)")

}

SFSpeechRecognizer.requestAuthorization { authStatus in

// Handle the authorization status

}

guard let speechRecognizer = SFSpeechRecognizer(locale: Locale(identifier: "en-US")) else {

// Handle the failure of creating a speech recognizer for the given locale

return

}

speechRecognizer.delegate = self

let recognitionRequest = SFSpeechAudioBufferRecognitionRequest()

let audioEngine = AVAudioEngine()

do {

let audioSession = AVAudioSession.sharedInstance()

try audioSession.setCategory(.record, mode: .measurement, options: .duckOthers)

try audioSession.setActive(true, options: .notifyOthersOnDeactivation)

let inputNode = audioEngine.inputNode

let recordingFormat = inputNode.outputFormat(forBus: 0)

inputNode.installTap(onBus: 0, bufferSize: 1024, format: recordingFormat) { buffer, \_ in

recognitionRequest.append(buffer)

}

audioEngine.prepare()

try audioEngine.start()

let recognitionTask = speechRecognizer.recognitionTask(with: recognitionRequest) { result, error in // Handle the result or error }

} catch { // Handle the error }

if let result = result { // Update the UI with the transcription

let transcribedText = result.bestTranscription.formattedString

}

audioEngine.stop()

recognitionRequest.endAudio()

recognitionTask?.cancel()