Parsa Taleb

3032263716

An Approach to Analyze Environmental Noise Pollution Via Smartphone Mass Data Collection

Bay area in general, and the city of Berkeley in particular, is suffering from extreme noise pollution. Nearby highways, three major airports, and, worse than all, busier-then-ever railroad with 24/7 loud whistling are exhausting the residents. Recent research shows that noise-pollution can be seriously fatal and behind a number of heart diseases.

A noise-map of the city of Berkeley is not currently in hand. It is believed that in many areas people are living in a very high level of noise-pollution well above the maximum thresholds that international standards allow. We are working with the department of environmental health - city of Berkeley to scientifically measure and map the noise level, and suggest ways to reduce the noise pollution while keeping the business going.

At the Theoretical and Applied Fluid Dynamics (TAF) Lab, receiving advice from Professor Reza Alam, a mobile application, from scratch, was developed to detect and analyze noise pollution in City of Berkeley. The app, commercially ready to download, records a session to accurately log surrounding frequency, amplitude, musical note, FFT, DFT, PSD, as well as user's geographical location data. After local analysis to detect and extract environmental noise, it saves tabulated data and a copy of audio file of the session in local storage area, and provides users with an option to quickly share between two devices. Once the device is connected to the internet, it uploads the data to an online server in background (without user interruption), and downloads a processed noise pollution 3D map depicting the live and historical level of noise in different parts of city of Berkeley. A JavaScript online server was also created to process mass live user data and compress it into a downloadable file, so that the app can render a noise pollution level map. The app is ready, submitted to the Apple app store for a technical review before becoming available to download for public. It is planned to publish a paper and apply for research funding next semester. This project has potential of public and private funding by map companies, city governments, or medical research groups to study the effect of noise pollution.

TAF LAB

THEORETICAL & APPLIED FLUID DYNAMICS LABRATORY

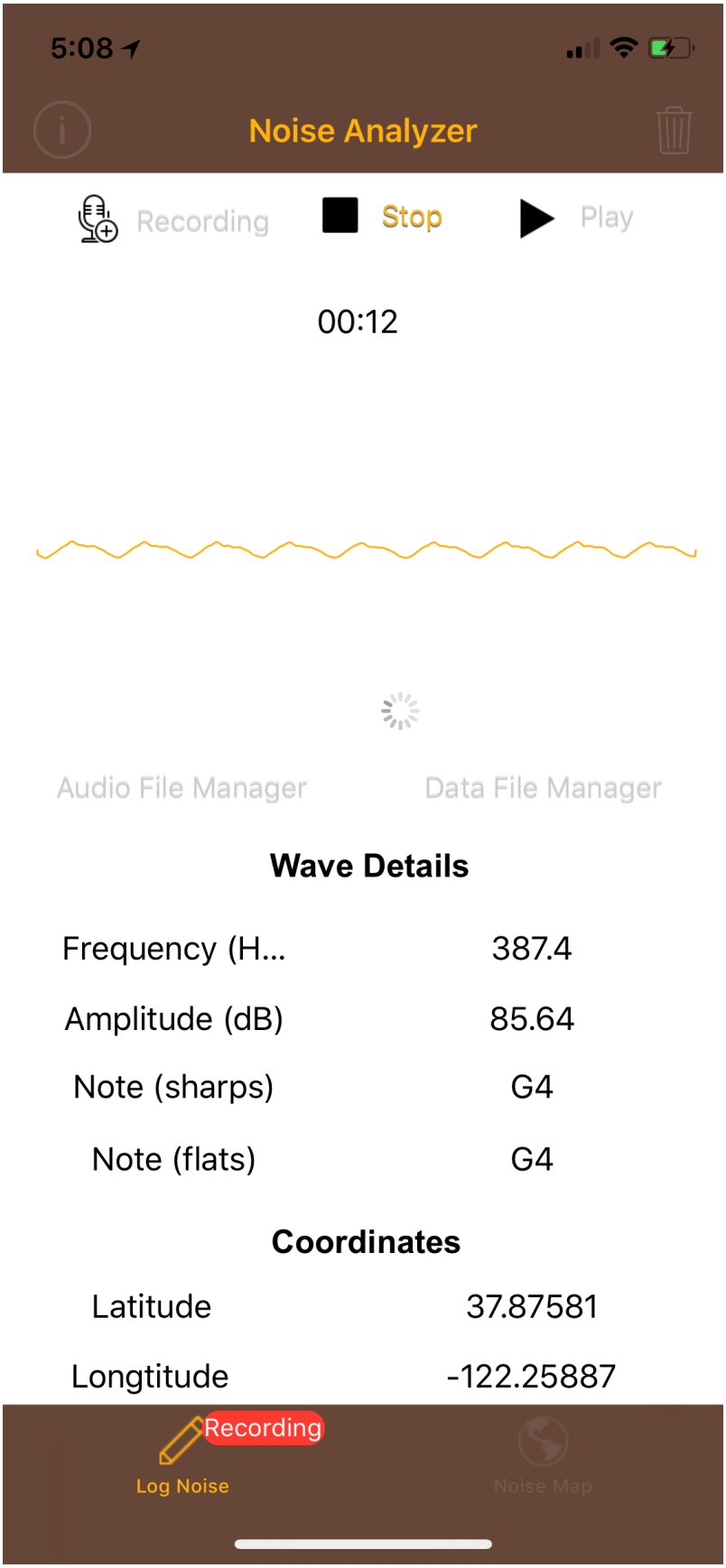
SOUND MAP

VER 0.2

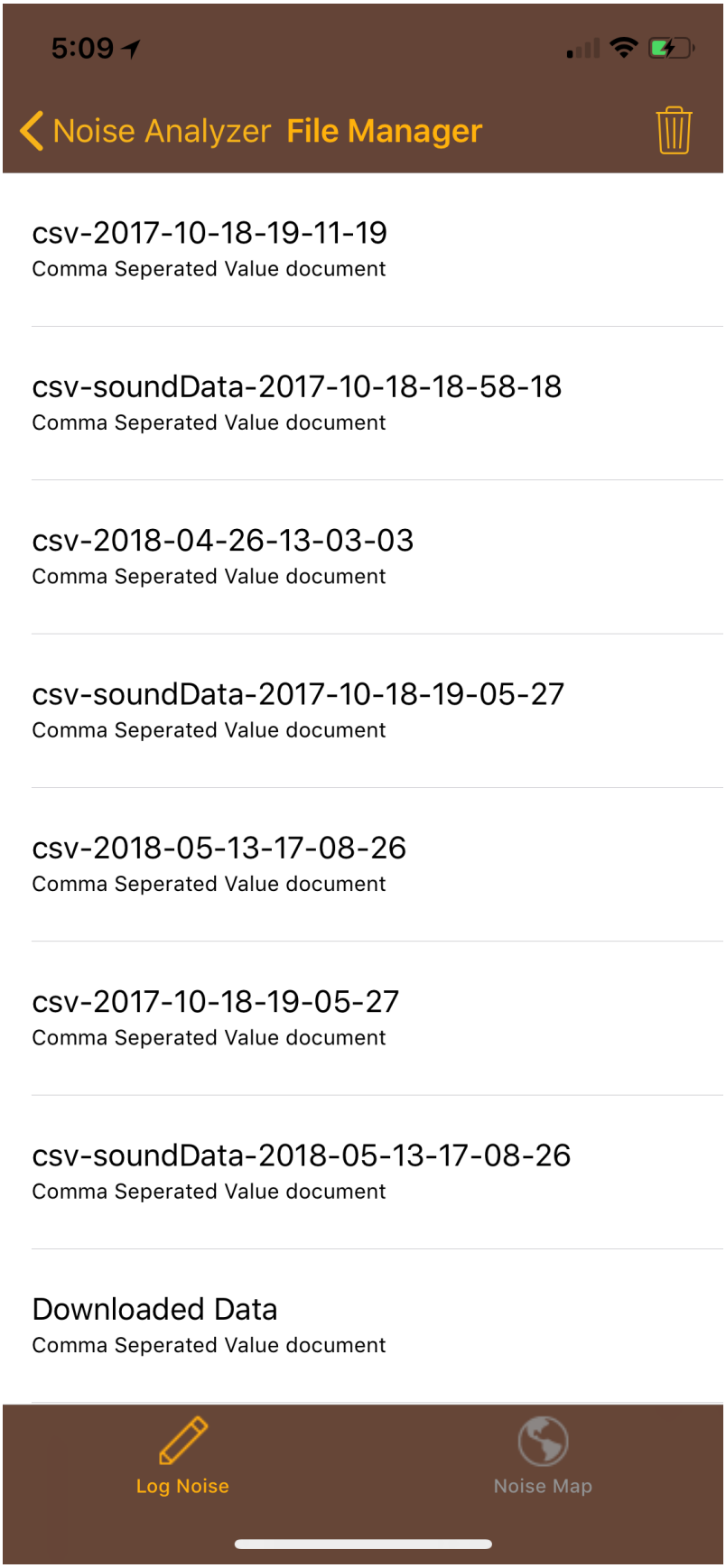
DESIGNED BY PARSA

Berkeley

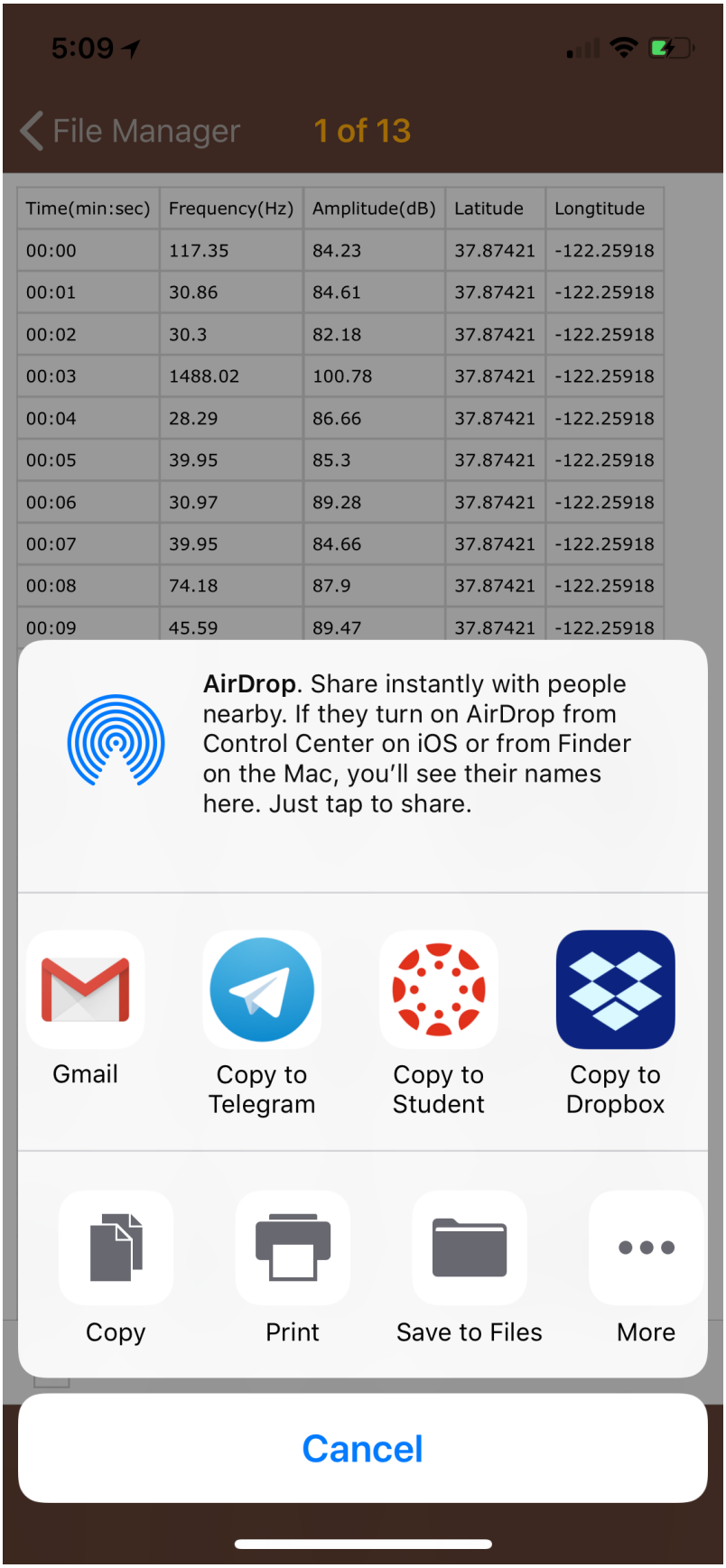
UNIVERSITY OF CALIFORNIA

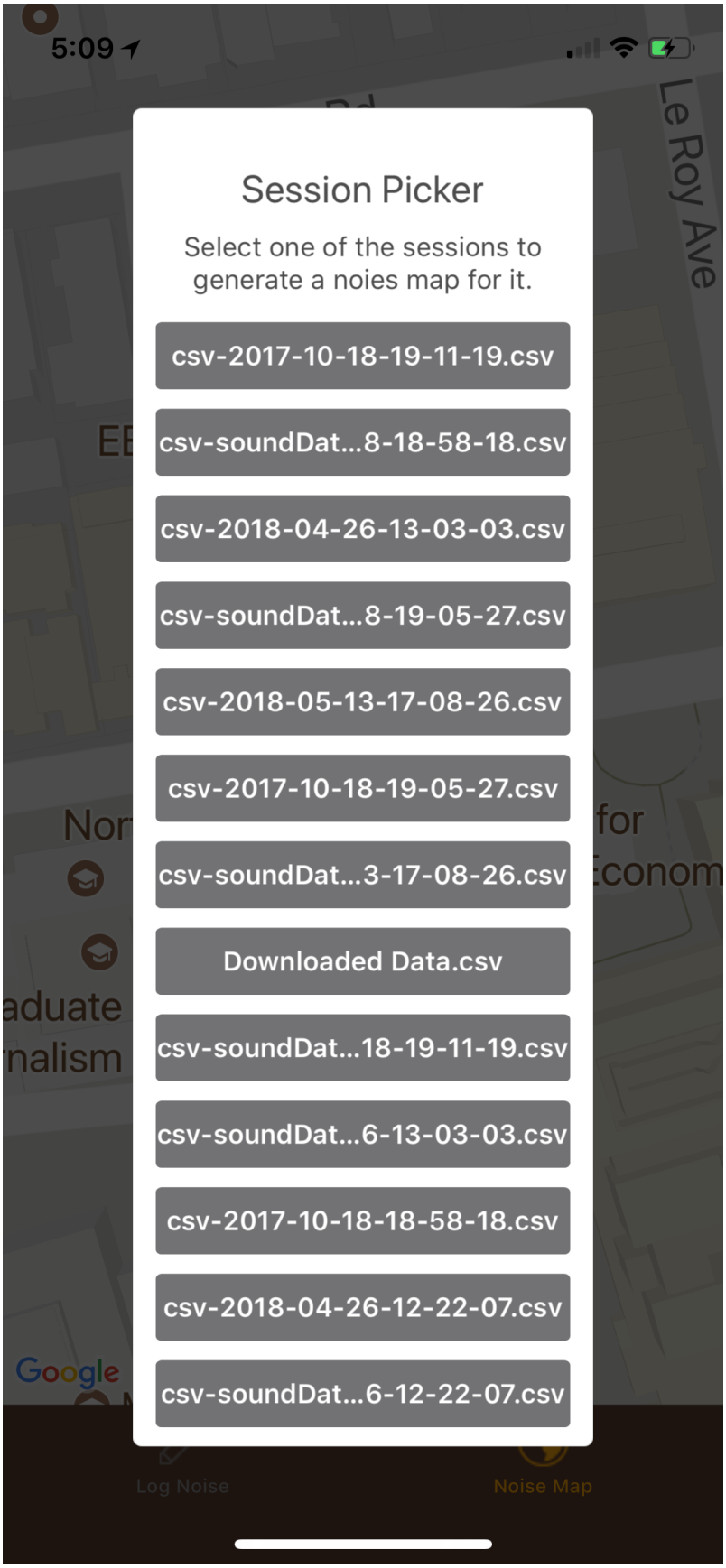

5:08
Noise Analyzer
Recording
Stop
Play
00:12
Audio File Manager
Data File Manager
Wave Details
Frequency (H...
387.4
Amplitude (dB)
85.64
Note (sharps)
G4
Note (flats)
G4
Coordinates
Latitude
37.87581
Longtitude
-122.25887
Recording
Log Noise
Noise Map

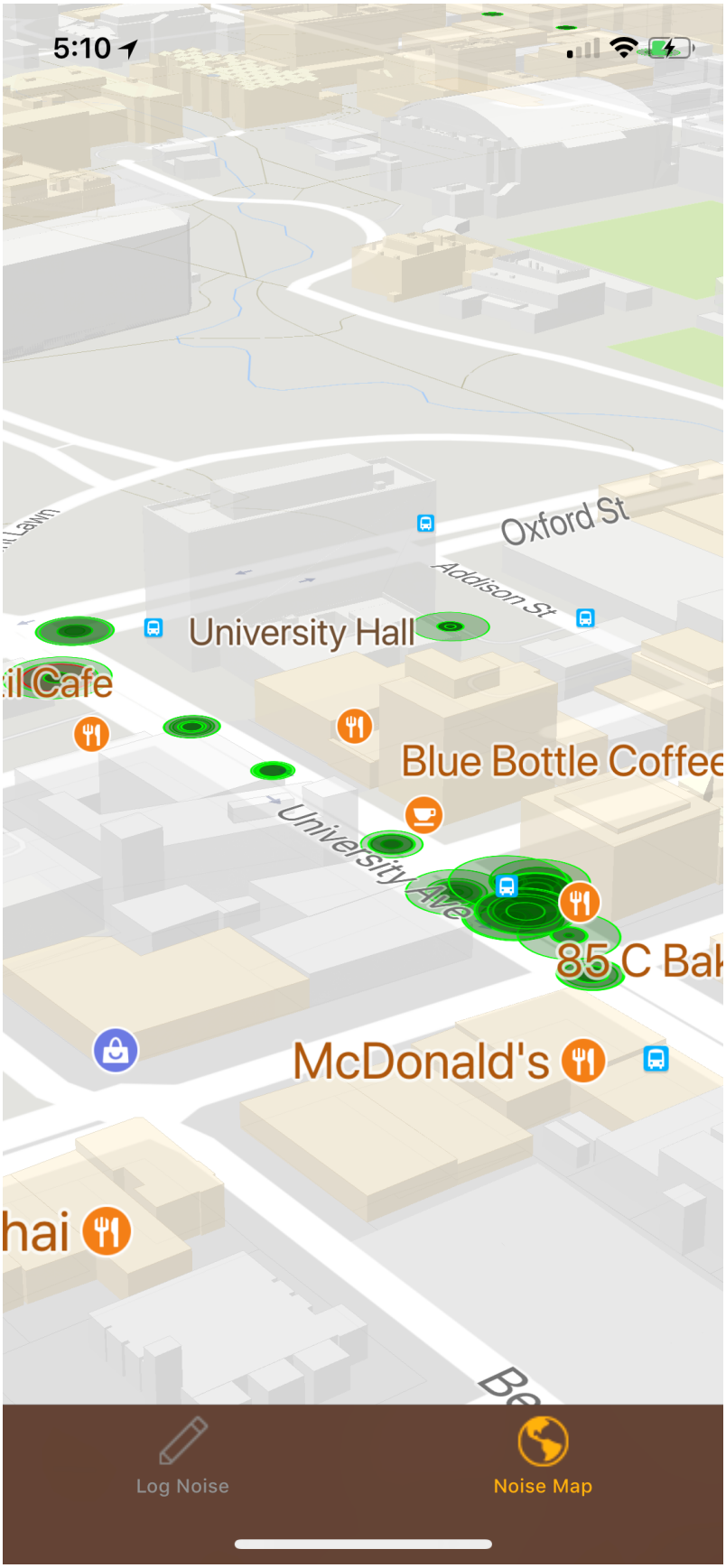

5:08
Noise Analyzer
Audio File Manager
AudioFile-2018-05-13-17-08-26.m4a
AudioFile-2017-10-18-19-11-19.m4a
AudioFile-2018-04-26-13-03-03.m4a
AudioFile-2017-10-18-18-58-18.m4a
AudioFile-2017-10-18-19-05-27.m4a
AudioFile-2018-04-26-12-22-07.m4a
Log Noise
Noise Map

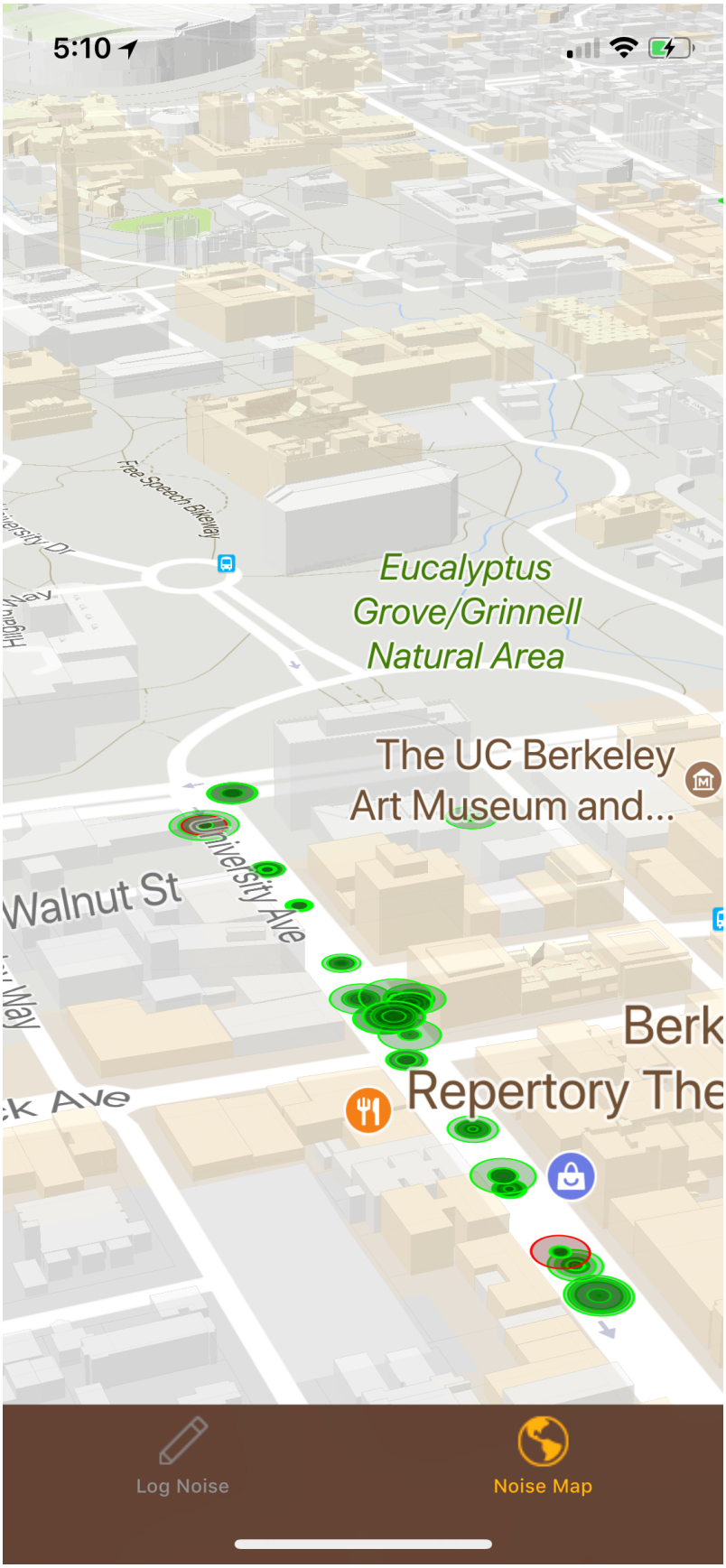

5:09
Noise Analyzer File Manager
csv-2017-10-18-19-11-19
Comma Seperated Value document
csv-soundData-2017-10-18-18-58-18
Comma Seperated Value document
csv-2018-04-26-13-03-03
Comma Seperated Value document
csv-soundData-2017-10-18-19-05-27
Comma Seperated Value document
csv-2018-05-13-17-08-26
Comma Seperated Value document
csv-2017-10-18-19-05-27
Comma Seperated Value document
csv-soundData-2018-05-13-17-08-26
Comma Seperated Value document
Downloaded Data
Comma Seperated Value document
Log Noise
Noise Map

Time(min:sec)	Frequency(Hz)	Amplitude(dB)	Latitude	Longtitude
00:00	117.35	84.23	37.87421	-122.25918
00:01	30.86	84.61	37.87421	-122.25918
00:02	30.3	82.18	37.87421	-122.25918
00:03	1488.02	100.78	37.87421	-122.25918
00:04	28.29	86.66	37.87421	-122.25918
00:05	39.95	85.3	37.87421	-122.25918
00:06	30.97	89.28	37.87421	-122.25918
00:07	39.95	84.66	37.87421	-122.25918
00:08	74.18	87.9	37.87421	-122.25918
00:09	45.59	89.47	37.87421	-122.25918
00:10	106.99	88.56	37.87421	-122.25918
00:11	385.7	91.16	37.87421	-122.25918
00:12	381.68	91.01	37.87421	-122.25918
00:13	55.7	88.79	37.87421	-122.25918
00:14	36.43	90.91	37.87421	-122.25918
00:15	44.01	89.01	37.87421	-122.25918
00:16	74.04	90.41	37.87421	-122.25918
00:17	268.69	89.49	37.87421	-122.25918
00:18	30.38	88.99	37.87421	-122.25918
00:19	97.1	88.65	37.87421	-122.25918
00:20	62.04	89.0	37.87421	-122.25918
00:21	60.91	89.44	37.87421	-122.25918
00:22	78.39	90.88	37.87421	-122.25918
00:23	109.1	90.14	37.87421	-122.25918
00:24	78.48	90.07	37.87421	-122.25918
00:25	98.66	88.8	37.87421	-122.25918


5:09
File Manager
1 of 13
Time(min:sec) Frequency(Hz) Amplitude(dB) Latitude Longtitude
00:00 117.35 84.23 37.87421 -122.25918
00:01 30.86 84.61 37.87421 -122.25918
00:02 30.3 82.18 37.87421 -122.25918
00:03 1488.02 100.78 37.87421 -122.25918
00:04 28.29 86.66 37.87421 -122.25918
00:05 39.95 85.3 37.87421 -122.25918
00:06 30.97 89.28 37.87421 -122.25918
00:07 39.95 84.66 37.87421 -122.25918
00:08 74.18 87.9 37.87421 -122.25918
00:09 45.59 89.47 37.87421 -122.25918
AirDrop. Share instantly with people nearby. If they turn on AirDrop from Control Center on iOS or from Finder on the Mac, you'll see their names here. Just tap to share.
Gmail
Copy to Telegram
Copy to Student
Copy to Dropbox
Copy
Print
Save to Files
More
Cancel


5:09
Le Roy Ave
Session Picker
Select one of the sessions to generate a noies map for it.
csv-2017-10-18-19-11-19.csv
csv-soundDat...8-18-58-18.csv
csv-2018-04-26-13-03-03.csv
csv-soundDat...8-19-05-27.csv
csv-2018-05-13-17-08-26.csv
csv-2017-10-18-19-05-27.csv
csv-soundDat...3-17-08-26.csv
Downloaded Data.csv
csv-soundDat...18-19-11-19.csv
csv-soundDat...6-13-03-03.csv
csv-2017-10-18-18-58-18.csv
csv-2018-04-26-12-22-07.csv
csv-soundDat...6-12-22-07.csv
Google
Log Noise
Noise Map


5:10
Oxford St
Addison St
University Hall
Cafe
Blue Bottle Coffee
University Ave
85 C Bak
McDonald's
hai
Log Noise
Noise Map


5:10
Free Speech Bikeway
Eucalyptus
Grove/Grinnell
Natural Area
The UC Berkeley
Art Museum and...
Walnut St
University Ave
Berk
Repertory The
Log Noise
Noise Map

```swift
//  SoundMap
//
//  Created by parsataleb on 12/19/16.
//
//
import UIKit
import AVFoundation
import AudioKit
import AudioKitUI
import MapKit
import CoreLocation

var dataRoll = NSMutableString()


class RecorderViewController: UIViewController, CLLocationManagerDelegate  {

    var recorder: AVAudioRecorder!

    var player:AVAudioPlayer!

    @IBOutlet var recordButton: UIButton!

    @IBOutlet var stopButton: UIButton!

    @IBOutlet var playButton: UIButton!

    @IBOutlet var statusLabel: UILabel!

    var meterTimer:Timer!

    var soundFileURL:URL!

    @IBOutlet weak var frequencyLabel: UILabel!

    @IBOutlet weak var amplitudeLabel: UILabel!

    @IBOutlet weak var latitude: UILabel!

    @IBOutlet weak var longtitude: UILabel!



    var locationManager:CLLocationManager!

    var timer = Timer()
    let microphone = AKMicrophone()
    var tracker: AKFrequencyTracker?
    var silence: AKBooster?
    var debugIncrement = 0
```

```
var currentFileName = ""


@IBOutlet weak var audioInputPlot: EZAudioPlot!

//let ezFile = EZAudioFile(url: URL(fileURLWithPath: "/Users/Asrap/Library/
 Developer/CoreSimulator/Devices/E3941AA8-BED6-4207-A11F-C83A9BAD10FD/data/
 Containers/Data/Application/94406D69-4ACF-4E99-BC11-9D569F796458/
 Documents/AudioFile-2017-01-07-23-59-21.m4a"))
//let data = ezFile?.getWaveformData()



let noteFrequencies:[Double] =
 [16.35,17.32,18.35,19.45,20.6,21.83,23.12,24.5,25.96,27.5,29.14,30.87]



let noteNamesWithSharps = ["C",
 "C♯","D","D♯","E","F","F♯","G","G♯","A","A♯","B"]

let noteNamesWithFlats = ["C",
 "D♭","D","E♭","E","F","G♭","G","A♭","A","B♭","B"]

@IBOutlet weak var noteNameWithSharpsLabel: UILabel!

@IBOutlet weak var noteNameWithFlatsLabel: UILabel!

@IBOutlet weak var dirtyDancer: UIActivityIndicatorView!



let Gold = UIColor(displayP3Red: 255/255, green: 183/255, blue: 67/255,
 alpha: 1)

var sessionsTimeDateLabel: String = ""

override func viewDidLoad() {
    super.viewDidLoad()



    self.tabBarController?.tabBar.tintColor = Gold
    tracker = AKFrequencyTracker(microphone)
    silence = AKBooster(tracker!, gain: 0)



    stopButton.isEnabled = false
    playButton.isEnabled = false
```

```
        setSessionPlayback()
        askForNotifications()
        checkHeadphones()
        setupPlot()

    }

    override func viewWillAppear(_ animated: Bool) {

        super.viewWillAppear(animated)


        determineMyCurrentLocation()


    }


    override func viewDidAppear(_ animated: Bool) {




    }

    func determineMyCurrentLocation() {
        locationManager = CLLocationManager()
        locationManager.delegate = self
        locationManager.desiredAccuracy = kCLLocationAccuracyBestForNavigation

        //kCLLocationAccuracyBest

        locationManager.requestAlwaysAuthorization()

        if CLLocationManager.locationServicesEnabled() {
            locationManager.startUpdatingLocation()
            //locationManager.startUpdatingHeading()
        }
    }

    func locationManager(_ manager: CLLocationManager, didUpdateLocations
     locations: [CLLocation]) {
        let userLocation:CLLocation = locations[0] as CLLocation

        // Call stopUpdatingLocation() to stop listening for location updates,
        // other wise this function will be called every time when user
         location changes.

        // manager.stopUpdatingLocation()

        //print("user latitude = \(userLocation.coordinate.latitude)")
```

```
        //print("user longitude = \(userLocation.coordinate.longitude)")

        let roundLat = Double(userLocation.coordinate.latitude).roundTo(places:
         5)
        let roundLong =
         Double(userLocation.coordinate.longitude).roundTo(places: 5)

        latitude.text = "\(roundLat)"
        longtitude.text = "\(roundLong)"

    }

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


    @objc func measureFrequency() {


        determineMyCurrentLocation()

        //print("Timer Fired \(debugIncrement += 1)")
        //print("microphone is on: \(microphone.isStarted)")


        if let frequency = tracker?.frequency {

            let roundFreq = Double(frequency).roundTo(places: 2)

            //print("get frequency: \(frequency)")
            frequencyLabel.text = "\(roundFreq)"


            if  recordButton.currentTitle == "Recording"
            {


                if let myTime = statusLabel.text {
                    dataRoll.append("\(myTime),\(roundFreq),")
                }
            }

        } else {
            frequencyLabel.text = "Problem"
        }
        //

        var myFrequency = Double((tracker?.frequency)!)
```

```
while (myFrequency > noteFrequencies[noteFrequencies.count-1]) {
    myFrequency = myFrequency / 2.0
}
while (myFrequency < noteFrequencies[0]) {
    myFrequency = myFrequency * 2.0
}



var minDistance: Double = 10000.0
var index = 0

for  k in 0 ... noteFrequencies.count-1{

    let distance = fabs(noteFrequencies[k] - myFrequency)
    if (distance < minDistance){
        index = k
        minDistance = distance
    }
}



let octave = Int(log2f(Float((tracker?.frequency)!) /
 Float(myFrequency)))
noteNameWithSharpsLabel.text = "\(noteNamesWithSharps[index])\(octave)"
noteNameWithFlatsLabel.text = "\(noteNamesWithFlats[index])\(octave)"

//
```

```
    if let amplitude = tracker?.amplitude {


        let roundAmp = Double(10 * (log(Double(amplitude)) -
         log(Double(0.00001)))).roundTo(places: 2)

        amplitudeLabel.text = "\(roundAmp)"


        let roundLat = Double(latitude.text!)?.roundTo(places: 5)
        let roundLong = Double(longtitude.text!)?.roundTo(places: 5)

        if let lat = roundLat {
            if let long = roundLong {



                if  recordButton.currentTitle  == "Recording"
                {

                    dataRoll.append("\(roundAmp),\(lat),\(long)\n")
                    //print(dataRoll)

                }
            }
        }


    } else {
        amplitudeLabel.text = "Problem"
    }
}


//  override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
//       let destViewController = segue.destination as! Tabulated
//      destViewController.tableLabelText = sender as! String
// }




@objc func updateAudioMeter(_ timer:Timer) {

    if recorder.isRecording {
        let min = Int(recorder.currentTime / 60)
        let sec = Int(recorder.currentTime.truncatingRemainder(dividingBy:
         60))
        let s = String(format: "%02d:%02d", min, sec)
        statusLabel.text = s
```

```
            recorder.updateMeters()
            // if you want to draw some graphics...
            //var apc0 = recorder.averagePowerForChannel(0)
            //var peak0 = recorder.peakPowerForChannel(0)
        }
    }


    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        recorder = nil
        player = nil


    }


    @IBAction func deleteAll(_ sender: Any) {
        deleteAllRecordings()
    }

    @IBOutlet weak var infoOutlet: UIBarButtonItem!

    @IBOutlet weak var deleteAllOutlet: UIBarButtonItem!


    @IBOutlet weak var AudioFileManagerOutlet: UIButton!

    @IBOutlet weak var DataFileManagerOutlet: UIButton!


    @IBAction func record(_ sender: UIButton) {

        if player != nil && player.isPlaying {
            player.stop()
        }

        if recorder == nil {
            //print("recording. recorder nil")
            recordButton.setTitle("Recording", for:UIControlState())
            //
            let tabBarControllerItems = self.tabBarController?.tabBar.items

            if let tabArray = tabBarControllerItems {
                let tabBarItem1 = tabArray[0]
                let tabBarItem2 = tabArray[1]


                tabBarItem2.isEnabled = false
                tabBarItem1.badgeValue = "Recording"
                dirtyDancer.startAnimating()
```

```
            infoOutlet.isEnabled = false
            deleteAllOutlet.isEnabled = false
            AudioFileManagerOutlet.isEnabled = false
            DataFileManagerOutlet.isEnabled = false



        }


        timer = Timer.scheduledTimer(timeInterval: 1, target: self,
         selector: #selector(RecorderViewController.measureFrequency),
         userInfo: nil, repeats: true)
        AudioKit.output = silence
        do {
            try AudioKit.start()
        } catch {
            AKLog("AudioKit did not start!")
        }
        microphone.start()
        tracker?.start()


        //

        dataRoll.setString("")

         dataRoll.append("Time(min:sec),Frequency(Hz),Amplitude(dB),Latitud
         e,Longtitude\n")


        recordButton.isEnabled = false
        playButton.isEnabled = false
        stopButton.isEnabled = true
        recordWithPermission(true)
        return
    }

    if recorder != nil && recorder.isRecording {
        print("pausing")
        recorder.pause()
        recordButton.setTitle("Continue", for:UIControlState())

    } else {
        print("recording")
        recordButton.setTitle("Recording", for:UIControlState())

        let tabBarControllerItems = self.tabBarController?.tabBar.items

        if let tabArray = tabBarControllerItems {
            let tabBarItem1 = tabArray[0]
```

```
            let tabBarItem2 = tabArray[1]


            tabBarItem2.isEnabled = false
            tabBarItem1.badgeValue = "Recording"
            dirtyDancer.startAnimating()
            infoOutlet.isEnabled = false
            deleteAllOutlet.isEnabled = false
            AudioFileManagerOutlet.isEnabled = false
            DataFileManagerOutlet.isEnabled = false


        }

        timer = Timer.scheduledTimer(timeInterval: 1, target: self,
         selector: #selector(RecorderViewController.measureFrequency),
         userInfo: nil, repeats: true)
        AudioKit.output = silence
        do {
            try AudioKit.start()
        } catch {
            AKLog("AudioKit did not start!")
        }
        microphone.start()
        tracker?.start()



        dataRoll.setString("")

         dataRoll.append("Time(min:sec),Frequency(Hz),Amplitude(dB),Latitud
         e,Longtitude\n")

        recordButton.isEnabled = false
        playButton.isEnabled = false
        stopButton.isEnabled = true
        //            recorder.record()
        recordWithPermission(false)
    }
}

@IBAction func stop(_ sender: UIButton) {
    print("✅Stop was tapped.")

    recorder?.stop()
    player?.stop()

    meterTimer.invalidate()

    recordButton.setTitle("Record", for:UIControlState())
```

```
        let tabBarControllerItems = self.tabBarController?.tabBar.items

        if let tabArray = tabBarControllerItems {
            let tabBarItem1 = tabArray[0]
            let tabBarItem2 = tabArray[1]

            tabBarItem2.isEnabled = true
            tabBarItem1.badgeValue = nil
            dirtyDancer.stopAnimating()
            infoOutlet.isEnabled = true
            deleteAllOutlet.isEnabled = true
            AudioFileManagerOutlet.isEnabled = true
            DataFileManagerOutlet.isEnabled = true


        }


        timer.invalidate()
        do {
            try AudioKit.stop()
        } catch {
            AKLog("AudioKit did not stop!")
        }
        microphone.stop()
        tracker?.stop()

        frequencyLabel.text = "-"
        amplitudeLabel.text = "-"
        noteNameWithSharpsLabel.text = "-"
        noteNameWithFlatsLabel.text = "-"


        let session = AVAudioSession.sharedInstance()
        do {
            try session.setActive(false)
            playButton.isEnabled = true
            stopButton.isEnabled = false
            recordButton.isEnabled = true
        } catch let error as NSError {
            print("could not make session inactive")
            print(error.localizedDescription)
        }

        //recorder = nil
    }

    @IBAction func play(_ sender: UIButton) {
        setSessionPlayback()
        play()
    }
```

```
func play() {

    var url:URL?
    if self.recorder != nil {
        url = self.recorder.url
    } else {
        url = self.soundFileURL!
    }
    print("playing \(String(describing: url))")

    do {
        self.player = try AVAudioPlayer(contentsOf: url!)
        stopButton.isEnabled = true
        player.delegate = self
        player.prepareToPlay()
        player.volume = 1.0
        player.play()
    } catch let error as NSError {
        self.player = nil
        print(error.localizedDescription)
    }

}


func setupRecorder() {
    let format = DateFormatter()
    format.dateFormat="yyyy-MM-dd-HH-mm-ss"

    sessionsTimeDateLabel = "\(format.string(from: Date()))"

    currentFileName = "AudioFile-\(format.string(from: Date())).m4a"
    //print(currentFileName)

    let documentsDirectory =
     FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)
     [0]
    self.soundFileURL =
     documentsDirectory.appendingPathComponent(currentFileName)

    //print("✅writing to soundfile url: '\(soundFileURL!)'")

    if FileManager.default.fileExists(atPath: soundFileURL.absoluteString)
     {
        // probably won't happen. want to do something about it?
        print("soundfile \(soundFileURL.absoluteString) exists")
    }


    //start of testing
```

```
var result = true
let myAudioSession = AVAudioSession.sharedInstance()
result = ((try?
 myAudioSession.setCategory(AVAudioSessionCategoryPlayAndRecord)) !=
 nil)
if !result {
    print("setCategory failed")
}
result = ((try? myAudioSession.setActive(true)) != nil)
if !result {
    print("setActive failed")
}
// Get the set of available inputs. If there are no audio accessories
 attached, there will be
// only one available input -- the built in microphone.
let inputs = myAudioSession.availableInputs
// Locate the Port corresponding to the built-in microphone.
var builtInMicPort: AVAudioSessionPortDescription? = nil
for port: AVAudioSessionPortDescription in inputs! {
    if (port.portType == AVAudioSessionPortBuiltInMic) {
        builtInMicPort = port
        break
    }
}
// Print out a description of the data sources for the built-in
 microphone
print("There are \(String(describing:
 (builtInMicPort?.dataSources?.count))) data sources for port :\"\
 (String(describing: builtInMicPort))\"")
print("\(String(describing: builtInMicPort?.dataSources))")
// loop over the built-in mic's data sources and attempt to locate the
 front microphone
var frontDataSource: AVAudioSessionDataSourceDescription? = nil
for source: AVAudioSessionDataSourceDescription in
 (builtInMicPort?.dataSources)! {
    if source.orientation?.isEqual(AVAudioSessionOrientationFront) ??
     false {
        frontDataSource = source
        break
    }
}
// end data source iteration
if frontDataSource != nil {
    print("Currently selected source is \"\(String(describing:
     builtInMicPort?.selectedDataSource?.dataSourceName))\" for port
     \"\(String(describing: builtInMicPort?.portName))\"")
    print("Attempting to select source \"\(String(describing:
     frontDataSource))\" on port \"\(String(describing:
     builtInMicPort?.portName))\"")
    // Set a preference for the front data source.
```

```
            result = ((try?
             builtInMicPort?.setPreferredDataSource(frontDataSource)) != nil)
            if !result {
                // an error occurred. Handle it!
                print("setPreferredDataSource failed")
            }
        }
        // Make sure the built-in mic is selected for input. This will be a no-
         op if the built-in mic is
        // already the current input Port.

        result = ((try? myAudioSession.setPreferredInput(builtInMicPort)) !=
         nil)
        if !result {
            // an error occurred. Handle it!
            print("setPreferredInput failed")
        }



        //end of testing



        let recordSettings:[String : AnyObject] = [
            AVFormatIDKey:              NSNumber(value:
             kAudioFormatAppleLossless),
            AVEncoderAudioQualityKey :
             NSNumber(value:AVAudioQuality.max.rawValue),
            AVEncoderBitRateKey :       NSNumber(value:320000),
            AVNumberOfChannelsKey:      NSNumber(value:2),
            AVSampleRateKey :           NSNumber(value:44100.0)
        ]


        do {
            recorder = try AVAudioRecorder(url: soundFileURL, settings:
             recordSettings)
            recorder.delegate = self
            recorder.isMeteringEnabled = true
            recorder.prepareToRecord() // creates/overwrites the file at
             soundFileURL

        } catch let error as NSError {
            recorder = nil
            print(error.localizedDescription)
        }

    }

    func recordWithPermission(_ setup:Bool) {
```

```
        let session:AVAudioSession = AVAudioSession.sharedInstance()
        // ios 8 and later

        //print(String(describing:
         (AVAudioSession.sharedInstance().availableInputs)!))

        //print(String(describing:
         (AVAudioSession.sharedInstance().availableInputs?[0].dataSources)!))

        //var port = AVAudioSession.sharedInstance().availableInputs[0] as?
         AVAudioSessionPortDescription

        if (session.responds(to:
         #selector(AVAudioSession.requestRecordPermission(_:)))) {
            AVAudioSession.sharedInstance().requestRecordPermission({(granted:
             Bool)-> Void in
                if granted {
                    print("✅Permission to record granted")
                    self.setSessionPlayAndRecord()
                    if setup {
                        self.setupRecorder()
                    }
                    self.recorder.record()
                    self.meterTimer = Timer.scheduledTimer(timeInterval: 0.1,
                                                           target:self,

                                                   selector:#selector(Recorde
                                                   rViewController.updateAudi
                                                   oMeter(_:)),
                                                           userInfo:nil,
                                                           repeats:true)
                } else {
                    print("Permission to record not granted")
                }
            })
        } else {
            print("requestRecordPermission unrecognized")
        }
    }

    func setSessionPlayback() {
        let session:AVAudioSession = AVAudioSession.sharedInstance()

        do {
            try session.setCategory(AVAudioSessionCategoryPlayback)
        } catch let error as NSError {
            print("could not set session category")
            print(error.localizedDescription)
        }
        do {
            try session.setActive(true)
```

```
        } catch let error as NSError {
            print("could not make session active")
            print(error.localizedDescription)
        }
    }

    func setSessionPlayAndRecord() {
        let session = AVAudioSession.sharedInstance()
        do {
            try session.setCategory(AVAudioSessionCategoryPlayAndRecord)
            //try session.setCategory(AVAudioSessionCategoryPlayAndRecord,
             mode: AVAudioSessionModeDefault, options:
             AVAudioSessionCategoryOptions.defaultToSpeaker)

        } catch let error as NSError {
            print("could not set session category")
            print(error.localizedDescription)
        }
        do {
            try session.setActive(true)
        } catch let error as NSError {
            print("could not make session active")
            print(error.localizedDescription)
        }
    }

    func deleteAllRecordings() {



        let alert = UIAlertController(title: "Files",
                                      message: "Are you really sure that you
                                       want to delete all the data and audio
                                       files?",
                                      preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "Yes, Delete All",
         style: .default, handler: {action in

            //


            //
            let docsDir =

                 NSSearchPathForDirectoriesInDomains(.documentDirectory, .userD
                 omainMask, true)[0]

            let fileManager = FileManager.default

            do {
```

```
        let files = try fileManager.contentsOfDirectory(atPath:
         docsDir)
        var recordings = files.filter( { (name: String) -> Bool in
            return name.hasSuffix("m4a")
        })
        for i in 0 ..< recordings.count {
            let path = docsDir + "/" + recordings[i]

            print("removing \(path)")
            do {
                try fileManager.removeItem(atPath: path)
            } catch let error as NSError {
                NSLog("could not remove \(path)")
                print(error.localizedDescription)
            }
        }

    } catch let error as NSError {
        print("could not get contents of directory at \(docsDir)")
        print(error.localizedDescription)
    }

    //

    let docsDir2 =

         NSSearchPathForDirectoriesInDomains(.documentDirectory, .userD
         omainMask, true)[0]

    let fileManager2 = FileManager.default

    do {
        let files2 = try fileManager2.contentsOfDirectory(atPath:
         docsDir2)
        var recordings = files2.filter( { (name: String) -> Bool in
            return name.hasSuffix("csv")
        })
        for i in 0 ..< recordings.count {
            let path2 = docsDir2 + "/" + recordings[i]
            let mapDataNotRemoved = docsDir2 + "/" + "Downloaded
             Data.csv"

            print("removing \(path2)")
            do {

                if path2 == mapDataNotRemoved {
                    print("deleting but mapData")

                }else{
```

```
                        try fileManager2.removeItem(atPath: path2)

                    }
                } catch let error as NSError {
                    NSLog("could not remove \(path2)")
                    print(error.localizedDescription)
                }
            }

        } catch let error as NSError {
            print("could not get contents of directory at \(docsDir2)")
            print(error.localizedDescription)
        }



        //


        //

        self.recorder = nil
    }))
    alert.addAction(UIAlertAction(title: "No", style: .default, handler:
     {action in
        self.recorder = nil
    }))
    self.present(alert, animated:true, completion:nil)









}

func askForNotifications() {

    NotificationCenter.default.addObserver(self,

                                           selector:#selector(RecorderView
                                           Controller.background(_:)),

                                           name:NSNotification.Name.UIAppl
                                           icationWillResignActive,
                                          object:nil)
```

```
        NotificationCenter.default.addObserver(self,
                                               selector:#selector(RecorderView
                                               Controller.foreground(_:)),
                                               name:NSNotification.Name.UIAppl
                                               icationWillEnterForeground,
                                               object:nil)

        NotificationCenter.default.addObserver(self,
                                               selector:#selector(RecorderView
                                               Controller.routeChange(_:)),
                                               name:NSNotification.Name.AVAudi
                                               oSessionRouteChange,
                                               object:nil)
    }

    @objc func background(_ notification:Notification) {
        print("❗App in background.")
    }

    @objc func foreground(_ notification:Notification) {
        print("❗App in foreground.")
    }


    @objc func routeChange(_ notification:Notification) {
        //print("routeChange \(String(describing: (notification as
         NSNotification).userInfo))")

        if let userInfo = (notification as NSNotification).userInfo {
            //print("userInfo \(userInfo)")
            if let reason = userInfo[AVAudioSessionRouteChangeReasonKey] as?
             UInt {
                //print("reason \(reason)")
                switch AVAudioSessionRouteChangeReason(rawValue: reason)! {
                case AVAudioSessionRouteChangeReason.newDeviceAvailable:
                    print("✅New Device Available.")
                    print("❗did you plug in headphones?")
                    checkHeadphones()
                case AVAudioSessionRouteChangeReason.oldDeviceUnavailable:
                    print("✅Old Device Unavailable.")
                    print("❗did you unplug headphones?")
                    checkHeadphones()
                case AVAudioSessionRouteChangeReason.categoryChange:
```

```
                print("✅Session activated/deactiavated (Category Change)")
            case AVAudioSessionRouteChangeReason.override:
                print("✅Override")
            case AVAudioSessionRouteChangeReason.wakeFromSleep:
                print("✅Wake From Sleep")
            case AVAudioSessionRouteChangeReason.unknown:
                print("✅Unknown")
            case
             AVAudioSessionRouteChangeReason.noSuitableRouteForCategory:
                print("✅No Suitable Route For Category")
            case AVAudioSessionRouteChangeReason.routeConfigurationChange:
                print("✅Route Configuration Change")

            }
        }
    }

    // this cast fails. that's why I do that goofy thing above.
    //        if let reason = userInfo[AVAudioSessionRouteChangeReasonKey]
     as? AVAudioSessionRouteChangeReason {
    //        }

    /*
     AVAudioSessionRouteChangeReasonUnknown = 0,
     AVAudioSessionRouteChangeReasonNewDeviceAvailable = 1,
     AVAudioSessionRouteChangeReasonOldDeviceUnavailable = 2,
     AVAudioSessionRouteChangeReasonCategoryChange = 3,
     AVAudioSessionRouteChangeReasonOverride = 4,
     AVAudioSessionRouteChangeReasonWakeFromSleep = 6,
     AVAudioSessionRouteChangeReasonNoSuitableRouteForCategory = 7,
     AVAudioSessionRouteChangeReasonRouteConfigurationChange
      NS_ENUM_AVAILABLE_IOS(7_0) = 8

     routeChange Optional([AVAudioSessionRouteChangeReasonKey: 1,
      AVAudioSessionRouteChangePreviousRouteKey:
      <AVAudioSessionRouteDescription: 0x17557350,
     inputs = (
     "<AVAudioSessionPortDescription: 0x17557760, type = MicrophoneBuiltIn;
      name = iPhone Microphone; UID = Built-In Microphone;
      selectedDataSource = Bottom>"
     );
     outputs = (
     "<AVAudioSessionPortDescription: 0x17557f20, type = Speaker; name =
      Speaker; UID = Built-In Speaker; selectedDataSource = (null)>"
     )>])
     routeChange Optional([AVAudioSessionRouteChangeReasonKey: 2,
      AVAudioSessionRouteChangePreviousRouteKey:
      <AVAudioSessionRouteDescription: 0x175562f0,
     inputs = (
```

```
        "<AVAudioSessionPortDescription: 0x1750c560, type = MicrophoneBuiltIn;
         name = iPhone Microphone; UID = Built-In Microphone;
         selectedDataSource = Bottom>"
        );
        outputs = (
        "<AVAudioSessionPortDescription: 0x17557de0, type = Headphones; name =
         Headphones; UID = Wired Headphones; selectedDataSource = (null)>"
        )>])
        */
    }

    func checkHeadphones() {
        // check NewDeviceAvailable and OldDeviceUnavailable for them being
         plugged in/unplugged
        let currentRoute = AVAudioSession.sharedInstance().currentRoute
        if currentRoute.outputs.count > 0 {
            for description in currentRoute.outputs {
                if description.portType == AVAudioSessionPortHeadphones {
                    print("✅headphones are plugged in")
                    break
                } else {
                    print("✅headphones are unplugged")
                }
            }
        } else {
            print("checking headphones requires a connection to a device")
        }
    }







}

// MARK: AVAudioRecorderDelegate
extension RecorderViewController : AVAudioRecorderDelegate {

    func audioRecorderDidFinishRecording(_ recorder: AVAudioRecorder,
                                         successfully flag: Bool) {
        print("✅Finished recording \(flag).")
        stopButton.isEnabled = false
        playButton.isEnabled = true
        recordButton.setTitle("Record", for:UIControlState())

        let tabBarControllerItems = self.tabBarController?.tabBar.items

        if let tabArray = tabBarControllerItems {
```

```
        let tabBarItem1 = tabArray[0]
        let tabBarItem2 = tabArray[1]

        tabBarItem2.isEnabled = true
        tabBarItem1.badgeValue = nil
        dirtyDancer.stopAnimating()
        infoOutlet.isEnabled = true
        deleteAllOutlet.isEnabled = true
        AudioFileManagerOutlet.isEnabled = true
        DataFileManagerOutlet.isEnabled = true


    }


    timer.invalidate()
    do {
        try AudioKit.stop()
    } catch {
        AKLog("AudioKit did not stop!")
    }
    microphone.stop()
    tracker?.stop()


    frequencyLabel.text = "-"
    amplitudeLabel.text = "-"
    noteNameWithSharpsLabel.text = "-"
    noteNameWithFlatsLabel.text = "-"


    // iOS8 and later
    let alert = UIAlertController(title: "Recorder",
                                  message: "Finished Recording",
                                  preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "Keep", style: .default, handler:
     {action in
        print("✅User decided to keep the session.")


        let currentTimeAndDate = self.sessionsTimeDateLabel
        //

        let contentsOfFile = "\(dataRoll)"

        if let dir = FileManager.default.urls(for: .documentDirectory,
         in: .userDomainMask).first {

            //let format = DateFormatter()
            //format.dateFormat="yyyy-MM-dd-HH-mm-ss"
```

```
        let myFileName = "csv-\(currentTimeAndDate).csv"


        let path = dir.appendingPathComponent(myFileName)

        //writing
        do {
            try contentsOfFile.write(to: path, atomically: false,
             encoding: String.Encoding.utf8)


        }
        catch {/* error handling here */}


    }


    //            "csv-\(currentTimeAndDate).csv"

    let myAKAudioFile = try! AKAudioFile(readFileName: "\
     (self.currentFileName)", baseDir: .documents)

    let newAudio = myAKAudioFile.floatChannelData?[0]


    //44100 samples per second
    print("❗️New session's number of samples (@sample rate of 44100
     samples/sec) \(String(describing: (newAudio?.count)!))")


    var contentsOfFile2 =  String(describing: newAudio!)
    //            print(contentsOfFile2)

    contentsOfFile2.remove(at: contentsOfFile2.startIndex)
    contentsOfFile2.remove(at: contentsOfFile2.index(before:
     contentsOfFile2.endIndex))


    if let dir = FileManager.default.urls(for: .documentDirectory,
     in: .userDomainMask).first {

        let myFileName2 = "csv-soundData-\(currentTimeAndDate).csv"
        let path2 = dir.appendingPathComponent(myFileName2)

        //writing
        do {
            try contentsOfFile2.write(to: path2, atomically: false,
             encoding: String.Encoding.utf8)
        }
```

```
                catch {/* error handling here */}


            }

            //

            self.recorder = nil
        }))
        alert.addAction(UIAlertAction(title: "Delete", style: .default,
         handler: {action in
            print("✅User decided to delete the session")
            self.recorder.deleteRecording()
        }))
        self.present(alert, animated:true, completion:nil)
    }


    func audioRecorderEncodeErrorDidOccur(_ recorder: AVAudioRecorder,
                                          error: Error?) {

        if let e = error {
            print("\(e.localizedDescription)")
        }
    }

    func setupPlot() {
        let plot = AKNodeOutputPlot(microphone, frame: audioInputPlot.bounds)

        plot.plotType = .buffer
        plot.shouldFill = false
        plot.shouldMirror = false
        plot.shouldCenterYAxis = true
        plot.color = Gold




        let documentsUrl = FileManager.default.urls(for: .documentDirectory,
         in: .userDomainMask).first!
        //
        let audioFilePath =
         documentsUrl.appendingPathComponent("AudioFile-2017-03-04-13-38-45.m4a
         ")
        //
        //        let audioFile = EZAudioFile(url: URL(fileURLWithPath:
         audioFilePath.path))
        //        let waveformData = audioFile!.getWaveformData()
        //
        //        plot.updateBuffer(waveformData?.buffers[0], withBufferSize:
         (waveformData?.bufferSize)!)
```

```
        //        print(waveformData ?? "no WaveFormData")
        //

        //var audioFile: EZAudioFile!
        //print("😜😜😜😜😜😜😜😜😜😜😜😜😜😜😜😜😜😜😜😜😜😜")
        //print(audioFilePath.path)

        if FileManager.default.fileExists(atPath: audioFilePath.path) {

            print("✅The audio File IS AVAILABLE")


        }else {

            print("✅The audio File IS NOT AVAILABLE")

        }


        //        audioFile = EZAudioFile(url: URL(fileURLWithPath:
         audioFilePath.path))
        //
        //        audioFile.getWaveformData()
        //




        audioInputPlot.addSubview(plot)

    }









}

// MARK: AVAudioPlayerDelegate
extension RecorderViewController : AVAudioPlayerDelegate {
    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully
     flag: Bool) {
        print("finished playing \(flag)")
        recordButton.isEnabled = true
        stopButton.isEnabled = false
    }
```

```
    func audioPlayerDecodeErrorDidOccur(_ player: AVAudioPlayer, error: Error?)
     {
        if let e = error {
            print("\(e.localizedDescription)")
        }

    }
}

extension Double {
    /// Rounds the double to decimal places valu
    func roundTo(places:Int) -> Double {
        let divisor = pow(10.0, Double(places))
        return (self * divisor).rounded() / divisor
    }



}
```

```swift
//
//  NoiseMapViewController.swift
//  SoundMap
//
//  Created by parsataleb on 12/28/16.
//
//

import UIKit
import GoogleMaps
import CSwiftV
import SCLAlertView

class NoiseMapViewController: UIViewController {



    var fileNames = [String]()
    var fileURLs = [NSURL]()
    var mapIndex = 0


    override func viewDidLoad() {
        super.viewDidLoad()




        // Do any additional setup after loading the view.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    /*
     // MARK: - Navigation

     // In a storyboard-based application, you will often want to do a little
      preparation before navigation
     override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
     // Get the new view controller using segue.destinationViewController.
     // Pass the selected object to the new view controller.
     }
     */
    override func loadView() {
        // Create a GMSCameraPosition that tells the map to display the
        // coordinate -33.86,151.20 at zoom level 6.
        //
```

```
let camera = GMSCameraPosition.camera(withLatitude: 37.875205,
 longitude: -122.259121, zoom: 18.5)
let mapView = GMSMapView.map(withFrame: CGRect.zero, camera: camera)
view = mapView

//
//        let circleCenter1 = CLLocationCoordinate2D(latitude:
 37.875205, longitude: -122.259121)
//        let circ1 = GMSCircle(position: circleCenter1, radius: 10)
//        circ1.fillColor = UIColor(red: 0.35, green: 0, blue: 0,
 alpha: 0.30)
//        circ1.strokeColor = .red
//        circ1.strokeWidth = 1
//        circ1.map = mapView
//
//        let position1 = CLLocationCoordinate2D(latitude: 37.875205,
 longitude: -122.259121)
//        let marker1 = GMSMarker(position: position1)
//        marker1.title = "Really noisy here!"
//        marker1.opacity = 0
//        marker1.map = mapView
//
//
//        let circleCenter2 = CLLocationCoordinate2D(latitude:
 37.875294, longitude: -122.258609)
//        let circ2 = GMSCircle(position: circleCenter2, radius: 5)
//
//        circ2.fillColor = UIColor(red: 0, green: 0.35, blue: 0,
 alpha: 0.30)
//        circ2.strokeColor = .green
//        circ2.strokeWidth = 1
//        circ2.map = mapView
//
//        let position2 = CLLocationCoordinate2D(latitude: 37.875294,
 longitude: -122.258609)
//        let marker2 = GMSMarker(position: position2)
//        marker2.title = "Very boring here!"
//        marker2.opacity = 0
//        marker2.map = mapView

//print("🔵 myFileURL are \(fileURLs)")
//print("🔴 myFileURL 2 are \(fileURLs[2])🔴")
```

```
}

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    prepareFileURLs()
    mapPickerAlert()
}

func prepareFileURLs() {

    //
    let documentsUrl =  FileManager.default.urls(for: .documentDirectory,
     in: .userDomainMask).first!

    print("💐document URL is:\(documentsUrl)💐")

    do {
        // Get the directory contents urls (including subfolders urls)
        let directoryContents = try
         FileManager.default.contentsOfDirectory(at: documentsUrl,
         includingPropertiesForKeys: nil, options: [])
        // print("****content",directoryContents)


        let csvFiles = directoryContents.filter{ $0.pathExtension ==
         "csv" }
        //print("csv urls:",csvFiles)
        //fileURLs.append(csvFiles as NSURL)

        fileNames = csvFiles.map{ $0.lastPathComponent }

    } catch let error as NSError {
        print(error.localizedDescription)
    }


    //
    for file in fileNames {
        let fileParts = file.components(separatedBy: ",")

        let fileURL = documentsUrl.appendingPathComponent("\
         (fileParts.joined(separator: "," ))")

        //print("Here0 \(fileParts)")
        //print("Hereeeeee1", fileURLs)


        if FileManager.default.fileExists(atPath: fileURL.path) {

            fileURLs.append(fileURL as NSURL)
```

```
            //print("Hereeeeee2", fileURLs)

        }

    }


}

func generateMap() {
    do {
        let inputString = try String(contentsOf: fileURLs[mapIndex] as URL,
         encoding: String.Encoding.utf8)


        //print("💐Master is \(fileURLs[mapIndex])💐")

        let noiseData = CSwiftV(with: inputString)

        let sampleEntries = noiseData.rows

        //print("💐sample Entries are \(sampleEntries)💐and")
        //  print(Double("-122.40823") as Any)
        // sampleEntries[2][3] = "37.78746"
        // sampleEntries[3][3] = "37.78756"
        // sampleEntries[4][3] = "37.78766"

        let camera = GMSCameraPosition.camera(withLatitude:  Double("\
         (sampleEntries[1][3])") ?? 0, longitude: Double("\
         (sampleEntries[1][4])") ?? 0, zoom: 18.5)
        let mapView = GMSMapView.map(withFrame: CGRect.zero, camera:
         camera)
        view = mapView


        for entry in sampleEntries {
            //print("🔴Found another \(entry[3])🔴")



            let circleCenter = CLLocationCoordinate2D(latitude: Double("\
             (entry[3])") ?? 0, longitude: Double("\(entry[4])") ?? 0)
            //default radius 10

            var myRadius = Double(0)


            let frequency = Double("\(entry[1])") ?? 0
```

```
            if (frequency) > 500 {

                myRadius = 8
                let circ = GMSCircle(position: circleCenter, radius:
                 myRadius)

                circ.fillColor = UIColor(red: 0.35, green: 0, blue: 0,
                 alpha: 0.30)
                circ.strokeColor = .red

                circ.strokeWidth = 1
                circ.map = mapView

            } else {

                myRadius = frequency / 40


                let circ = GMSCircle(position: circleCenter, radius:
                 myRadius)

                circ.fillColor = UIColor(red: 0.00, green: 0.35, blue: 0,
                 alpha: 0.30)
                circ.strokeColor = .green


                circ.strokeWidth = 1
                circ.map = mapView

            }




            let position = CLLocationCoordinate2D(latitude: Double("\
             (entry[3])") ?? 0, longitude: Double("\(entry[4])") ?? 0)
            let marker = GMSMarker(position: position)
            marker.title = "\(entry[2])Hz - \(entry[2])dB"
            marker.snippet = "Time:\(entry[0]) Lat:\(entry[3]) Long:\
             (entry[4])"
            marker.opacity = 0
            marker.map = mapView



        }

    }
    catch {/* error handling here */}
```

```
    }

    func mapPickerAlert() {


        let appearance = SCLAlertView.SCLAppearance(
            showCloseButton: false,

            showCircularIcon: false
            //SCLAnimationStyle
        )


        let alertView = SCLAlertView(appearance: appearance)


        var myindex = 0
        for file in fileNames {
            let fileParts = file.components(separatedBy: ",")
            let buttonName = "\(fileParts.joined(separator: "," ))"


            alertView.addButton(buttonName) {
                self.prepareFileURLs()
                myindex = self.fileNames.index(of: buttonName)!

                print("😡 \(buttonName) button \(myindex) tapped")
                self.mapIndex = myindex
                self.generateMap()

            }
        }



        alertView.showNotice("Session Picker", subTitle: "Select one of the
         sessions to generate a noies map for
         it.",animationStyle:SCLAnimationStyle.noAnimation)



    }


}
```

```swift
//
import UIKit
import QuickLook
import SCLAlertView


let quickLookController = QLPreviewController()

class FileListViewController: UIViewController, UITableViewDelegate,
 UITableViewDataSource, QLPreviewControllerDataSource,
 QLPreviewControllerDelegate{


    @IBOutlet weak var tblFileList: UITableView!
    var fileNames = [String]()
    var fileURLs = [NSURL]()



    override func viewDidLoad() {
        super.viewDidLoad()


        //        if let dir =
         FileManager.default.urls(for: .documentDirectory,
         in: .userDomainMask).first {
        //            print("\(dir)")
        //        }

        prepareFileURLs()

        configureTableView()
        navigationItem.title = "File Manager"
        quickLookController.dataSource = self



    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    // MARK: Custom Methods

    func configureTableView() {
        tblFileList.delegate = self
        tblFileList.dataSource = self
```

```
        tblFileList.register(UINib(nibName: "FileListCell", bundle: nil),
         forCellReuseIdentifier: "idCellFile")
        tblFileList.reloadData()
    }


    //
    //Bundle.main.url(forResource: fileParts[0], withExtension: fileParts[1])
    func prepareFileURLs() {

        //
        let documentsUrl =  FileManager.default.urls(for: .documentDirectory,
         in: .userDomainMask).first!

        do {
            // Get the directory contents urls (including subfolders urls)
            let directoryContents = try
             FileManager.default.contentsOfDirectory(at: documentsUrl,
             includingPropertiesForKeys: nil, options: [])
            // print("****content",directoryContents)


            let csvFiles = directoryContents.filter{ $0.pathExtension ==
             "csv" }
            //print("csv urls:",csvFiles)
            //fileURLs.append(csvFiles as NSURL)

            fileNames = csvFiles.map{ $0.lastPathComponent }

        } catch let error as NSError {
            print(error.localizedDescription)
        }


        //
        for file in fileNames {
            let fileParts = file.components(separatedBy: ",")

            let fileURL = documentsUrl.appendingPathComponent("\
             (fileParts.joined(separator: "," ))")

            //print("Here0 \(fileParts)")
            //print("Hereeeeee1", fileURLs)


            if FileManager.default.fileExists(atPath: fileURL.path) {

                fileURLs.append(fileURL as NSURL)

                //print("Hereeeeee2", fileURLs)
```

```
        }

    }


}

// MARK: UITableView Methods

func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}
func numberOfPreviewItems(in controller: QLPreviewController) -> Int {
    return fileURLs.count
}
func extractAndBreakFilenameInComponents(fileURL: NSURL) -> (fileName:
 String, fileExtension: String) {
    // Break the NSURL path into its components and create a new array with
     those components.
    let fileURLParts = fileURL.path!.components(separatedBy: "/")

    // Get the file name from the last position of the array above.
    let fileName = fileURLParts.last

    // Break the file name into its components based on the period symbol
     (".").
    let filenameParts = fileName?.components(separatedBy: ".")

    // Return a tuple.
    return (filenameParts![0], filenameParts![1])
}

func tableView(_ tableView: UITableView, numberOfRowsInSection section:
 Int) -> Int {




    return fileURLs.count
}


func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath)
 -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "idCellFile",
     for: indexPath as IndexPath)

    let currentFileParts = extractAndBreakFilenameInComponents(fileURL:
     fileURLs[indexPath.row])

    cell.textLabel?.text = currentFileParts.fileName
```

```
        cell.detailTextLabel?.text =
         getFileTypeFromFileExtension(fileExtension:
         currentFileParts.fileExtension)



        return cell
    }


    func tableView(_ tableView: UITableView, heightForRowAt indexPath:
     IndexPath) -> CGFloat {
        return 80.0
    }


    func previewController(_ controller: QLPreviewController, previewItemAt
     index: Int) -> QLPreviewItem {
        return fileURLs[index]
    }


    func getFileTypeFromFileExtension(fileExtension: String) -> String {
        var fileType = ""

        switch fileExtension {
        case "docx":
            fileType = "Microsoft Word document"

        case "pages":
            fileType = "Pages document"

        case "jpeg":
            fileType = "Image document"

        case "key":
            fileType = "Keynote document"

        case "pdf":
            fileType = "PDF document"

        case "csv":
            fileType = "Comma Seperated Value document"

        case "m4a":
            fileType = "m4a document"

        default:
            fileType = "Text document"
```

```
    }

    return fileType
}



func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
 IndexPath) {
    if QLPreviewController.canPreview(fileURLs[indexPath.row]) {
        quickLookController.currentPreviewItemIndex = indexPath.row
        navigationController?.pushViewController(quickLookController,
         animated: true)
    }
}



func previewControllerWillDismiss(_ controller: QLPreviewController) {
    print("The Preview Controller will be dismissed.")
}
func previewControllerDidDismiss(_ controller: QLPreviewController) {
    tblFileList.deselectRow(at: tblFileList.indexPathForSelectedRow!,
     animated: true)
    print("The Preview Controller has been dismissed.")
}
func previewController(_ controller: QLPreviewController, shouldOpen url:
 URL, for item: QLPreviewItem) -> Bool {
    if item as! NSURL == fileURLs[0] {
        return true
    }
    else {
        print("Will not open URL \(url.absoluteString)")
    }

    return false
}



@IBAction func DeleteEntry(_ sender: Any) {



    let appearance = SCLAlertView.SCLAppearance(
        showCloseButton: false,

        showCircularIcon: false
        //SCLAnimationStyle
```

```
        )


        let alertView = SCLAlertView(appearance: appearance)


        for file in fileNames {
            let fileParts = file.components(separatedBy: ",")
            let buttonName = "\(fileParts.joined(separator: "," ))"


            if buttonName == "Downloaded Data.csv"{
                print("not deleting")
            }else{
                alertView.addButton(buttonName) {
                    self.prepareFileURLs()

                    //print("😡 \(buttonName) button tapped")

                    let docsDir2 =
                        NSSearchPathForDirectoriesInDomains(.documentDirectory,
                         .userDomainMask, true)[0]

                    let fileManager2 = FileManager.default

                    do {

                        let path2 = docsDir2 + "/" + buttonName

                        print("removing \(path2)")
                        do {


                            try fileManager2.removeItem(atPath: path2)


                        } catch let error as NSError {
                            NSLog("could not remove \(path2)")
                            print(error.localizedDescription)
                        }


                    }


                }
            }
```

```
        }



        alertView.showNotice("Delete Entry", subTitle: "Select one of the files
         to delete, closing in 5 secs...",closeButtonTitle:
         "close",animationStyle:SCLAnimationStyle.noAnimation)



    }


}
```

```swift
import UIKit
import AVFoundation
import SCLAlertView

let reuseIdentifier = "recordingCell"

class RecordingsCollectionViewController: UICollectionViewController {

    var recordings = [URL]()
    var player:AVAudioPlayer!
    var fileNames = [String]()
    var fileURLs = [NSURL]()


    override func viewDidLoad() {
        super.viewDidLoad()

        // Uncomment the following line to preserve selection between
         presentations
        // self.clearsSelectionOnViewWillAppear = false

        // set the recordings array
        listRecordings()

        let recognizer = UILongPressGestureRecognizer(target: self, action:
         #selector(RecordingsCollectionViewController.longPress(_:)))
        recognizer.minimumPressDuration = 0.5 //seconds
        recognizer.delegate = self
        recognizer.delaysTouchesBegan = true
        self.collectionView?.addGestureRecognizer(recognizer)

        let doubleTap = UITapGestureRecognizer(target:self,
         action:#selector(RecordingsCollectionViewController.doubleTap(_:)))
        doubleTap.numberOfTapsRequired = 2
        doubleTap.numberOfTouchesRequired = 1
        doubleTap.delaysTouchesBegan = true
        self.collectionView?.addGestureRecognizer(doubleTap)
    }

    /**
     Get the cell with which you interacted.
     */
    //    func getCell(rec:UIGestureRecognizer) -> UICollectionViewCell {
    //        var cell:UICollectionViewCell!
    //
    //        let p = rec.locationInView(self.collectionView)
    //        let indexPath = self.collectionView?.indexPathForItemAtPoint(p)
    //        if indexPath == nil {
    //            NSLog("couldn't find index path");
    //        } else {
```

```
//            cell =
 self.collectionView?.cellForItemAtIndexPath(indexPath!)
//            NSLog("found cell at \(indexPath!.row)")
//        }
//        return cell
//    }

@objc func doubleTap(_ rec:UITapGestureRecognizer) {
    if rec.state != .ended {
        return
    }

    let p = rec.location(in: self.collectionView)
    if let indexPath = self.collectionView?.indexPathForItem(at: p) {
        askToRename((indexPath as NSIndexPath).row)
    }

}

@objc func longPress(_ rec:UILongPressGestureRecognizer) {
    if rec.state != .ended {
        return
    }
    let p = rec.location(in: self.collectionView)
    if let indexPath = self.collectionView?.indexPathForItem(at: p) {
        askToDelete((indexPath as NSIndexPath).row)
    }

}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
}

/*
 // MARK: - Navigation

 // In a storyboard-based application, you will often want to do a little
  preparation before navigation
 override func prepareForSegue(segue: UIStoryboardSegue, sender:
  AnyObject!) {
 // Get the new view controller using [segue destinationViewController].
 // Pass the selected object to the new view controller.
 }
 */

// MARK: UICollectionViewDataSource

override func numberOfSections(in collectionView: UICollectionView) -> Int
 {
```

```
        return 1
    }


    override func collectionView(_ collectionView: UICollectionView,
     numberOfItemsInSection section: Int) -> Int {
        return self.recordings.count
    }

    override func collectionView(_ collectionView: UICollectionView,
     cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {

        let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
         reuseIdentifier, for: indexPath) as! RecordingCollectionViewCell

        cell.label.text = recordings[(indexPath as
         NSIndexPath).row].lastPathComponent

        return cell
    }

    // MARK: UICollectionViewDelegate

    override func collectionView(_ collectionView: UICollectionView,
     shouldHighlightItemAt indexPath: IndexPath) -> Bool {
        return true
    }


    override func collectionView(_ collectionView: UICollectionView,
     shouldSelectItemAt indexPath: IndexPath) -> Bool {
        return true
    }

    override func collectionView(_ collectionView: UICollectionView,
     didSelectItemAt indexPath: IndexPath) {

        print("selected \(recordings[(indexPath as
         NSIndexPath).row].lastPathComponent)")

        //var cell = collectionView.cellForItemAtIndexPath(indexPath)
        play(recordings[(indexPath as NSIndexPath).row])

    }

    func play(_ url:URL) {
        print("playing \(url)")

        do {
            self.player = try AVAudioPlayer(contentsOf: url)
            player.prepareToPlay()
```

```
            player.volume = 1.0
            player.play()
        } catch let error as NSError {
            self.player = nil
            print(error.localizedDescription)
        } catch {
            print("AVAudioPlayer init failed")
        }

    }



    /*
     // Uncomment these methods to specify if an action menu should be
      displayed for the specified item, and react to actions performed on the
      item
     func collectionView(collectionView: UICollectionView!,
      shouldShowMenuForItemAtIndexPath indexPath: NSIndexPath!) -> Bool {
     return false
     }

     func collectionView(collectionView: UICollectionView!, canPerformAction
      action: String!, forItemAtIndexPath indexPath: NSIndexPath!, withSender
      sender: AnyObject!) -> Bool {
     return false
     }

     func collectionView(collectionView: UICollectionView!, performAction
      action: String!, forItemAtIndexPath indexPath: NSIndexPath!, withSender
      sender: AnyObject!) {

     }
     */



    func listRecordings() {

        let documentsDirectory =
         FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)
         [0]
        do {
            let urls = try FileManager.default.contentsOfDirectory(at:
             documentsDirectory, includingPropertiesForKeys: nil, options:
             FileManager.DirectoryEnumerationOptions.skipsHiddenFiles)
            self.recordings = urls.filter( { (name: URL) -> Bool in
                return name.lastPathComponent.hasSuffix("m4a")
            })

        } catch let error as NSError {
```

```
            print(error.localizedDescription)
        } catch {
            print("something went wrong listing recordings")
        }

    }

    func askToDelete(_ row:Int) {
        let alert = UIAlertController(title: "Delete",
                                      message: "Delete Recording \
                                       (recordings[row].lastPathComponent)?",
            preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "Yes", style: .default, handler:
         {action in
            print("yes was tapped \(self.recordings[row])")
            self.deleteRecording(self.recordings[row])
        }))
        alert.addAction(UIAlertAction(title: "No", style: .default, handler:
         {action in
            print("no was tapped")
        }))
        self.present(alert, animated:true, completion:nil)
    }

    func askToRename(_ row:Int) {
        let recording = self.recordings[row]

        let alert = UIAlertController(title: "Rename",
                                      message: "Rename Recording \
                                       (recording.lastPathComponent)?",
            preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "Yes", style: .default, handler:
         {[unowned alert] action in
            print("yes was tapped \(self.recordings[row])")
            if let textFields = alert.textFields{
                let tfa = textFields as [UITextField]
                let text = tfa[0].text
                let url = URL(fileURLWithPath: text!)
                self.renameRecording(recording, to: url)
            }
        }))
        alert.addAction(UIAlertAction(title: "No", style: .default, handler:
         {action in
            print("no was tapped")
        }))
        alert.addTextField(configurationHandler: {textfield in
            textfield.placeholder = "Enter a filename"
            textfield.text = "\(recording.lastPathComponent)"
        })
        self.present(alert, animated:true, completion:nil)
    }
```

```
func renameRecording(_ from:URL, to:URL) {
    let documentsDirectory =
     FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)
     [0]
    let toURL =
     documentsDirectory.appendingPathComponent(to.lastPathComponent)

    print("renaming file \(from.absoluteString) to \(to) url \(toURL)")
    let fileManager = FileManager.default
    fileManager.delegate = self
    do {
        try FileManager.default.moveItem(at: from, to: toURL)
    } catch let error as NSError {
        print(error.localizedDescription)
    } catch {
        print("error renaming recording")
    }
    DispatchQueue.main.async(execute: {
        self.listRecordings()
        self.collectionView?.reloadData()
    })

}


func deleteRecording(_ url:URL) {

    print("removing file at \(url.absoluteString)")
    let fileManager = FileManager.default

    do {
        try fileManager.removeItem(at: url)
    } catch let error as NSError {
        print(error.localizedDescription)
    } catch {
        print("error deleting recording")
    }

    DispatchQueue.main.async(execute: {
        self.listRecordings()
        self.collectionView?.reloadData()
    })
}

override func motionBegan(_ motion: UIEventSubtype, with event: UIEvent?) {
    print("😡Device was shaken!")

    prepareFileURLs()
```

```
let appearance = SCLAlertView.SCLAppearance(
    showCloseButton: false,

    showCircularIcon: false
    //SCLAnimationStyle
)


let alertView = SCLAlertView(appearance: appearance)


for file in fileNames {
    let fileParts = file.components(separatedBy: ",")
    let buttonName = "\(fileParts.joined(separator: "," ))"



    alertView.addButton(buttonName) {
        self.prepareFileURLs()



        let docsDir2 =

             NSSearchPathForDirectoriesInDomains(.documentDirectory, .u
             serDomainMask, true)[0]

        do {

            let path2 = docsDir2 + "/" + buttonName

            //print("removing \(path2)")

            //    try fileManager2.removeItem(atPath: path2)

            let shareObject = UIActivityViewController(activityItems:
             [path2], applicationActivities: nil)
            self.present(shareObject, animated: true, completion: nil)




        }


    }
```

```
        }


        alertView.showNotice("Share Entry", subTitle: "Select one of the files
         to share.",animationStyle:SCLAnimationStyle.noAnimation)









    }

}

extension RecordingsCollectionViewController: FileManagerDelegate {

    func fileManager(_ fileManager: FileManager, shouldMoveItemAt srcURL: URL,
     to dstURL: URL) -> Bool {

        print("should move \(srcURL) to \(dstURL)")
        return true
    }


    func prepareFileURLs() {

        //
        let documentsUrl =  FileManager.default.urls(for: .documentDirectory,
         in: .userDomainMask).first!

        do {
            // Get the directory contents urls (including subfolders urls)
            let directoryContents = try
             FileManager.default.contentsOfDirectory(at: documentsUrl,
             includingPropertiesForKeys: nil, options: [])
            // print("****content",directoryContents)


            let csvFiles = directoryContents.filter{ $0.pathExtension ==
             "m4a" }
            //print("csv urls:",csvFiles)
            //fileURLs.append(csvFiles as NSURL)
```

```
            fileNames = csvFiles.map{ $0.lastPathComponent }

        } catch let error as NSError {
            print(error.localizedDescription)
        }


        //
        for file in fileNames {
            let fileParts = file.components(separatedBy: ",")

            let fileURL = documentsUrl.appendingPathComponent("\
             (fileParts.joined(separator: "," ))")

            //print("Here0 \(fileParts)")
            //print("Hereeeeee1", fileURLs)


            if FileManager.default.fileExists(atPath: fileURL.path) {

                fileURLs.append(fileURL as NSURL)

                //print("Hereeeeee2", fileURLs)

            }

        }


    }



}

extension RecordingsCollectionViewController : UIGestureRecognizerDelegate {

}
```

```swift
import UIKit

class RecordingCollectionViewCell: UICollectionViewCell {

    @IBOutlet var label: UILabel!

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        let view = UIView(frame:self.frame)
        view.autoresizingMask = [.flexibleWidth, .flexibleHeight]
        view.backgroundColor = UIColor(red: 0.2, green: 0.6, blue: 1.0, alpha:
         1.0)
        view.layer.borderColor = UIColor.white.cgColor
        view.layer.borderWidth = 4
        self.selectedBackgroundView = view
    }

}
```

```swift
//
//  AppDelegate.swift
//  SoundMap
//
//  Created by parsataleb on 12/19/16.
//
//bugs:
//session picker not working properly
//File Manager delete Not worrking properly
//horizontal view
//warnings
//offline mode doesn't download the file and deletes it

//demo: new file with 3d view
//audio file manger matches MP3 File Manager

import UIKit
import GoogleMaps

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?


    func application(_ application: UIApplication,
     didFinishLaunchingWithOptions launchOptions:
     [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.

        let Gold = UIColor(displayP3Red: 255/255, green: 183/255, blue: 67/255,
         alpha: 1)
        UINavigationBar.appearance().titleTextAttributes =
         [NSAttributedStringKey.foregroundColor: Gold]

        GMSServices.provideAPIKey("AIzaSyAAbaSK5O7-oDltMDoOctMuVYN68Gx5zcA")

        downloadMapDataToDocFolderAsFirstEntry()


        return true
    }

    func applicationWillResignActive(_ application: UIApplication) {
        // Sent when the application is about to move from active to inactive
         state. This can occur for certain types of temporary interruptions
         (such as an incoming phone call or SMS message) or when the user quits
         the application and it begins the transition to the background state.
        // Use this method to pause ongoing tasks, disable timers, and
         invalidate graphics rendering callbacks. Games should use this method
         to pause the game.
```

```
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        // Use this method to release shared resources, save user data,
         invalidate timers, and store enough application state information to
         restore your application to its current state in case it is terminated
         later.
        // If your application supports background execution, this method is
         called instead of applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(_ application: UIApplication) {
        // Called as part of the transition from the background to the active
         state; here you can undo many of the changes made on entering the
         background.
    }

    func applicationDidBecomeActive(_ application: UIApplication) {
        // Restart any tasks that were paused (or not yet started) while the
         application was inactive. If the application was previously in the
         background, optionally refresh the user interface.

    }

    func applicationWillTerminate(_ application: UIApplication) {
        // Called when the application is about to terminate. Save data if
         appropriate. See also applicationDidEnterBackground:.
    }
    func downloadMapDataToDocFolderAsFirstEntry() {


        let documentsUrl = FileManager.default.urls(for: .documentDirectory,
         in: .userDomainMask).first!


        //getting rid of the old file


        let originPath = documentsUrl.appendingPathComponent("pbJDEc")
        let destinationPath = documentsUrl.appendingPathComponent("Downloaded
         Data.csv")



        //print("😜dest path...")
        //print(destinationPath)
        //print("😜origin path...")
        //print(originPath)

        if FileManager.default.fileExists(atPath: originPath.path) {
            do{
```

```
            try FileManager.default.removeItem(at: originPath)
            print("✅old temp file deleted.")
        }catch{
            print("❌error deleting temp file.")
        }


    }else {
        print("✅Old temp file is not available")
    }


    if FileManager.default.fileExists(atPath: destinationPath.path) {

        do {
            try FileManager.default.removeItem(at: destinationPath)
            print("✅Old downloadedd data file deleted.")
        }catch{
            print("❌Could not delete old downloaded data")
        }


    }else {
        print("❌FILE 2 is not available")
    }


    //downloading csv from google drive to document folder

    if let mapDataURL = URL(string: "https://goo.gl/pbJDEc") {
        // create your document folder url
        // your destination file url
        let destination =
         documentsUrl.appendingPathComponent(mapDataURL.lastPathComponent)
        //print(destination)

        //  just download the data from your url
        URLSession.shared.downloadTask(with: mapDataURL, completionHandler:
         { (location, response, error) in
            // after downloading your data you need to save it to your
             destination url
            guard
                let httpURLResponse = response as? HTTPURLResponse,
                 httpURLResponse.statusCode == 200,
                let mimeType = response?.mimeType,
                 mimeType.hasPrefix("text/csv"),
                let location = location, error == nil
                else { return }
            do {
```

```
                    //moving from temp to doc folder
                    try FileManager.default.moveItem(at: location, to:
                     destination)
                    print("✅Downloaded temp file saved.")

                    //renaming the file




                    try FileManager.default.moveItem(at: originPath, to:
                     destinationPath)


                    print("✅Downloaded temp file renamed.")




                } catch let error as NSError {
                    print(error.localizedDescription)
                }
            }).resume()


        }
    }


}
```