AVFoundation

Recording Audio

Step 0: Import `AVFoundation` at the top of the class that needs access to recording methods, and make sure it also conforms to `AVAudioRecorderDelegate`

Step 1: Add a UIButton with an IBAction for recording audio. Do it as an IBOutlet as well so that when the user taps it, it can change from “Record” to “Stop Recording” or something similar

`@IBAction func recordButtonPressed(_ sender: Any) {}`

`@IBOutlet weak var recordButton: UIButton!`

Step 2: There are a few properties that we might want to include as a part of recording audio:

`var recorder: AVAudioRecorder?` - this is `AVFoundation`'s way of recording audio (optional in a class so we don’t have to init if we don’t want to yet)

`var recordedURL: URL?` - this is probably the best way of saving the recorded audio somewhere in the app’s memory

`var id: String?` - this is a good way to identify each unique recording if you plan on doing multiple

```
var isRecording: Bool {
  return recorder?.isRecording ?? false }
```

-this is a good way to evaluate whether or not the recorder is currently doing work or not

Step 3: Let’s make a method so that we can save the URL to a specific place when the recording has finished.

```
func saveRecordedAudio() -> URL {
  let fileManager = FileManager.default
  let documentsDirectory = try! fileManager.url(for: .documentsDirectory, 
in: .userDomainMask, appropriateFor: nil, create: true)
  return 
documentsDirectory.appendingPathComponent(id).appendingPathExtension(“caf”)
}
```

Step 4: Now we can implement the actual audio recording method:

```
func record() {


}
```

Make sure that we aren't already recording with the Bool property we set earlier, and if we are we can tell the recorder to stop:

```
func record() {
  guard !isRecording else {
      recorder?.stop()
      return
  }
}
```

In a do try catch, we can attempt to record the audio and save it via the URL method we already made:

```
func record() {
  guard !isRecording else {
      recorder?.stop()
      return
  }


  do {
            let audioFormat = AVAudioFormat(standardFormatWithSampleRate:
44100.0, channels: 2)!
            recorder = try AVAudioRecorder(url: saveRecordedAudio(), format:
audioFormat)
            recorder?.prepareToRecord()
            recorder?.delegate = self
            recorder?.record()
        } catch {
            NSLog("Unable to start recording: \(error)")
        }
}
```

Step 5: Now that we have this method, be sure to call it when the user taps the recordButton:

```
@IBAction func recordButtonPressed(_ sender: Any) {
  record()
}
```

Step 6: Last step is to use a delegate method to perform the work of setting our optional recordedURL property to the recording once the recording has finished and to reset the recorder. `audioRecorderDidFinishRecording` is your pal here:

```
func audioRecorderDidFinishRecording(_ recorder: AVAudioRecorder, successfully
flag: Bool) {
        recordingURL = recorder.url
        self.recorder = nil
    }
```

*you'll want to implement some logic so that the buttons change their titles appropriately when they are tapped so the user gets good visual feedback

---

Playing Audio

Step 0: most of the steps will follow the same format as a recorder. Conform your class to `AVAudioPlayerDelegate` so it can use the delegate methods.

Step 1: Add a UIButton with an IBAction for playing audio. Do it as an IBOutlet as well so that when the user taps it, it can change from "Play" to "Stop" or something similar
`@IBAction func playButtonPressed(_ sender: Any) {}`
`@IBOutlet weak var playButton: UIButton!`

Step 2: Create a property for the player just like we did for the recorder `var player: AVAudioPlayer?`. Do the same with a Bool to check if we're playing or not.

```
var isPlaying: Bool {
  return player?.isPlaying ?? false }
```

Step 3: Now we can implement the actual play method:

```
func play() {


}
```

Make sure we have a recording URL or else we won't have anything to play

```
func play() {
  guard let url = recordedURL else {
      return
  }
}
```

Make sure that we aren't already playing with the Bool property we set earlier, and if we are we can tell the player to stop:

```
func play() {
  guard let url = recordedURL else {
      return
  }
  guard !isPlaying else {
      player?.stop()
      return
  }
}
```

In a do try catch, we can attempt to play the audio via the URL method we already saved it in:

```
func play() {
  guard let url = recordedURL else {
      return
```

```
    }
    guard !isPlaying else {
        player?.stop()
        return
    }

    do {
        player = try! AVAudioPlayer(contentsOf: url)
        player?.delegate = self
        player?.play
    } catch {
        NSLog("Unable to start playing: \(error)")
    }
}
```

Step 4: Be sure to call this play method when the play button is tapped

Step 5: Last step is to use a delegate method to perform the work of resetting the player. `audioPlayerDidFinishPlaying` is your pal here:

```
func audioPlayerDidFinishPlayer(_ recorder: AVAudioPlayer, successfully flag:
Bool) {
        self.player = nil
    }
```

*again, it's a good idea to make some logic to change the button's title depending on if it's playing or not.

If all goes well, you should have the audio recorded and playing successfully! Be sure to handle the permissions and privacy issues in the Xcode project's info folder!