STT and TTS in iOS

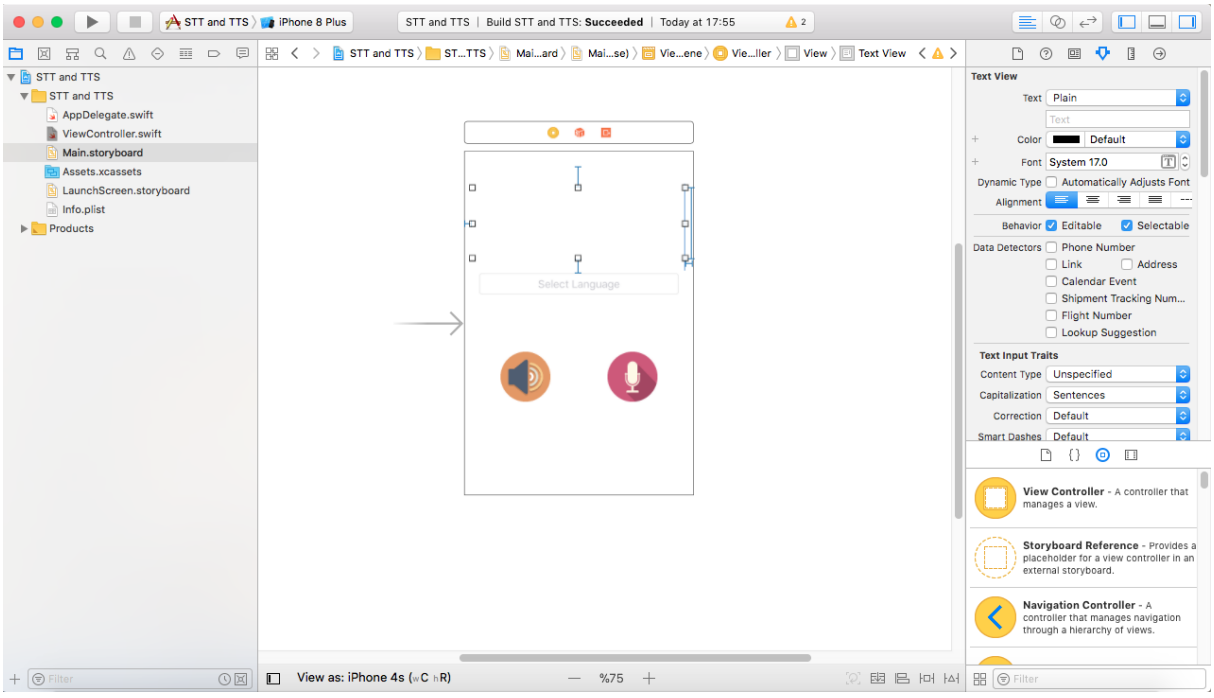
At 2016, Apple introduced Speech Framework, which is a useful API for speech recognition. There are other useful speech recognition frameworks available, however, they are either expensive or not very effective. However, the Speech framework

On the other hand, TTS can be used by Apple since iOS 7, which is very easy to use. To make things more clear, iOS 7 introduced with a new class name AVSpeechSynthesizer which is powerful part of AVFoundation framework.

This documentation aims to explain about the Apple's Speech and AVSpeechSynthesizer frameworks by building a STT and TTS application prototype by suing swift 4 and deployment version of 9.

1. Designing the UI

To be able to focus on STT and TTS frameworks design has done in the interface builder. To do so, I used a UITextView where we will write the text to be spoken by TTS and also will show what has spoken for STT. Under that, there is a UITextField which will be used for the language selection for both TTS and STT. Other than that, there are two buttons which is for TTS and STT.



2. Implementation of code

2.1 Speech to Text

First of all, Speech framework is only available for iOS 10 and later versions. However, we can check the iOS version of the device and if it supports iOS 10 and above app will work Speech framework otherwise it wont. Speech framework does not work in offline mode so that the application needs to have an internet connection. Also speech framework provides

Inside the ViewController class, we import the necessary frameworks

```
import UIKit
import Speech
import AVFoundation


@available(iOS 10.0, *)
class ViewController: UIViewController,SFSpeechRecognizerDelegate,UITextViewDelegate,
UITextFieldDelegate,UIPickerViewDelegate,UIPickerViewDataSource{
```

As mentioned in the beginning, to check for availability of the device, on top of the class, we have to add,

```
@available(iOS 10.0, *)
```

Later, as we see in above code preview, we have to adopt

```
SFSpeechRecognizerDelegate
```

protocol.

After that, we have the UI elements and languageOption array for the picker view.

```
  @IBOutlet var textView: UITextView!
  @IBOutlet var languageTextField: UITextField!
  @IBOutlet weak var sttButton: UIButton!

  var languageOption = ["...","en-US","pl","tr","de"]
```

User Authorization

Before using the speech framework for speech recognition, you have to first ask for users' permission because the recognition doesn't happen just locally on the iOS Device Apple's servers. All the voice data is transmitted to Apple's backend for processing. Therefore, it is mandatory to get user's permission. To do that, we create a speechRecognizer instance with controlling the version of iOS 10.

```
  private var speechRecognizer :SFSpeechRecognizer? = {
    let iosVersion = NSString(string: UIDevice.current.systemVersion).doubleValue
    return (iosVersion >= 10 ? SFSpeechRecognizer()! : nil)
  }()
```

Above code represents the speech recognizer object creating by controlling the version of iOS 10. According to that, if the device uses iOS 10 and above the code will pass and if not it will return nil.
To continue with user authentication, we create a new function called

prepareSpeechRecognizer()

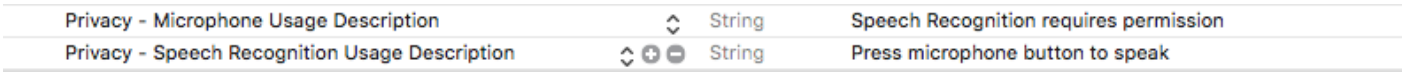
and call it in ViewDidLoad() . Implementation of the function is as below,

```
// Preperation of the speech recognizer
    func prepareSpeechRecognizer(){

        sttButton.isEnabled = false //2
        speechRecognizer?.delegate = self  //3

        if #available(iOS 10, *) {
            SFSpeechRecognizer.requestAuthorization { (authStatus) in // 4

                var isButtonEnabled = false

                switch authStatus { // 5
                case .authorized:
                    isButtonEnabled = true

                case .denied:
                    isButtonEnabled = false
                    print("User denied access to speech recognition")

                case .restricted:
                    isButtonEnabled = false
                    print("Speech recognition restricted on this device")

                case .notDetermined:
                    isButtonEnabled = false
                    print("Speech recognition not yet authorized")
                }

                OperationQueue.main.addOperation() {
                    self.sttButton.isEnabled = isButtonEnabled
                }
            }
        }
    }
```

According to to above function,

1. After we create the SFSpeechRecognizer instance with version control,
2. By default, we disable the sttButton, until the speech recognizer is activated.
3. Then set speech recognizer delegate to self which in this case our ViewController.
4. After that, we must request the authorization of Speech Recognition by calling SFSpeechRecognizer.requestAuthorization
5. Finally, check the status of the verification. If it's authorized, enable the sttButton. If not, print the error message and disable the microphone button.

After these steps, we have to provide authorization message into info.plist file to prevent the app crash. To do that, open the info.plist file in the Xcode and enter the keys as picture below.



Handling the speech Recognition

After we implement the user authorization, we have to implement speech recognition. To do that under the speechRecognition variable we add,

```
private var recognitionRequest: SFSpeechAudioBufferRecognitionRequest?
private var recognitionTask: SFSpeechRecognitionTask?
private let audioEngine = AVAudioEngine()
```

SFSpeechAudioBufferRecognitionRequest : handles the speech recognition request. Provides an audio input to the recognizer.

SFSpeechRecognitionTask : Gives the result of the recognition request. Allows us to cancel or stop the task.

AVAudioEngine : Is responsible to provide your audio input.

Then we create a new function called startRecording().This function is called when the Start Recording button is tapped. Its main function is to start up the speech recognition and start listening to your microphone.

```
// start recording and get the best transcription
func startRecording() {

    // check if recognition task working. If so, cancel the task and the reognition.
    if recognitionTask != nil {
        recognitionTask?.cancel()
        recognitionTask = nil
    }

    //Create an AVAudioSession to prepare for the audio recording. Here we set the category of the session as recording, the mode as measurement, and activate it. Setting these properties may throw an exception, so it has to be in a catch clause.
    let audioSession = AVAudioSession.sharedInstance()
    do {
        try audioSession.setCategory(AVAudioSessionCategoryRecord)
        try audioSession.setMode(AVAudioSessionModeMeasurement)
        try audioSession.setActive(true, with: .notifyOthersOnDeactivation)
    } catch {
        print("audioSession properties weren't set because of an error.")
    }

    //Instantiate the recognitionRequest. Here we create the SFSpeechAudioBufferRecognitionRequest object. Later, we use it to pass our audio data to Apple's servers.
        recognitionRequest = SFSpeechAudioBufferRecognitionRequest()

    //Give audioEngine(the device) an audio input for recording
     let inputNode = audioEngine.inputNode

    //Check if the recognitionRequest object is instantiated and is not nil.
```

```
        guard let recognitionRequest = recognitionRequest else {
            fatalError("Unable to create an SFSpeechAudioBufferRecognitionRequest object")
        }

        //Tell recognitionRequest to report partial results of speech recognition as the user speaks.
        recognitionRequest.shouldReportPartialResults = true

        //Start the recognition by calling the recognitionTask method of our speechRecognizer. This function has a completion handler. This completion handler will be called every time the recognition engine has received input, has refined its current recognition, or has been canceled or stopped, and will return a final transcript.
        recognitionTask = speechRecognizer?.recognitionTask(with: recognitionRequest, resultHandler: { (result, error) in

            //Define a boolean to determine if the recognition is final.
            var isFinal = false

            // If the result isn't nil, set the textView.text property as our result's best transcription. Then if the result is the final result, set isFinal to true.
            if result != nil {
                self.textView.text = result?.bestTranscription.formattedString
                isFinal = (result?.isFinal)!
            }

            //If there is no error or the result is final, stop the audioEngine (audio input) and stop the recognitionRequest and recognitionTask. At the same time, we enable the Start Recording button(sttButton).
            if error != nil || isFinal {
                self.audioEngine.stop()
                inputNode.removeTap(onBus: 0)

                self.recognitionRequest = nil
                self.recognitionTask = nil

                self.sttButton.isEnabled = true
            }
        })

        //Add an audio input to the recognitionRequest. Note that it is ok to add the audio input after starting the recognitionTask. The Speech Framework will start recognizing as soon as an audio input has been added.
        let recordingFormat = inputNode.outputFormat(forBus: 0)
        inputNode.installTap(onBus: 0, bufferSize: 1024, format: recordingFormat) { (buffer, when) in
            self.recognitionRequest?.append(buffer)
        }

        //Prepare and start the audioEngine.
        audioEngine.prepare()

        do {
            try audioEngine.start()
        } catch {
            print("audioEngine couldn't start because of an error.")
        }
    }
```

Triggering Speech Recognition

Speech recognition should be available when creating a speech recognition task, so we have to add a delegate method to ViewController. If speech recognition is unavailable or changes its status, the sttButton.enable property should be set. For this situation, we implement the availabilityDidChange method of the SFSpeechRecognizerDelegate protocol. Implementation of the method is below

```
        public  func speechRecognizer(_ speechRecognizer: SFSpeechRecognizer, availabilityDidChange available: Bool)
{
    if available {
        sttButton.isEnabled = true
    } else {
        sttButton.isEnabled = false
    }
  }
```

This method will be called when the availability changes. If speech recognition is available, the record button will also be enabled.

Last thing we have to do is create the sttAction methods which will be working after we tap the recording button(sttButton).

```
@IBAction func sttAction(_ sender: Any) {
        if self.audioEngine.isRunning {
            self.audioEngine.stop()
            self.recognitionRequest?.endAudio()
            self.sttButton.isEnabled = false
            self.sttButton.setImage(UIImage(named:"microphone_icon"), for: .normal)
    } else {
            self.speechRecognizer = SFSpeechRecognizer(locale: Locale.init(identifier: self.languageTextField.text!))
            self.startRecording()
            self.sttButton.setImage(UIImage(named:"microphoneStop_icon"), for: .normal)
        }
  }
```

In this function, we must check whether our audioEngine is running. If it is running, the app should stop the audioEngine , terminate the input audio to our recognitionRequest, disable our sttButton, and set the button's image to microphone_icon.png

If the audioEngine is working, we set the preferred language to be spoken locale by taking the instance from user with pickerView and call startRecording() method and set the image of the button to microphone_stop.png

2.2 Text to Speech

While STT is working only in online mode, TTS is working from the device inside so it is offline. To prepare the the TTS, inside the ttsAction method we put the following code.

```
@IBAction func ttsAction(_ sender: Any) {
    let utterance = AVSpeechUtterance(string: textView.text) // 1
    utterance.voice = AVSpeechSynthesisVoice(language: languageTextField.text) // 2
    utterance.rate = 0.5 // 3

    let synthesizer = AVSpeechSynthesizer() // 4
    synthesizer.speak(utterance) // 5
  }
```

1. Create an instance of AVSpeechUtterance. AVSpeechUtterance, provides the chunk of text to be spoken along with the parameters that effect its speech which we provide the string from the textField.

2. Give utterance object the language by AVSpeechSynthesisVoice providing from pickerView.
3. Define the rate of utterance object.
4. Create a AVSpeechSynthesiser instance.
5. Trigger the speak method of synthesizer.