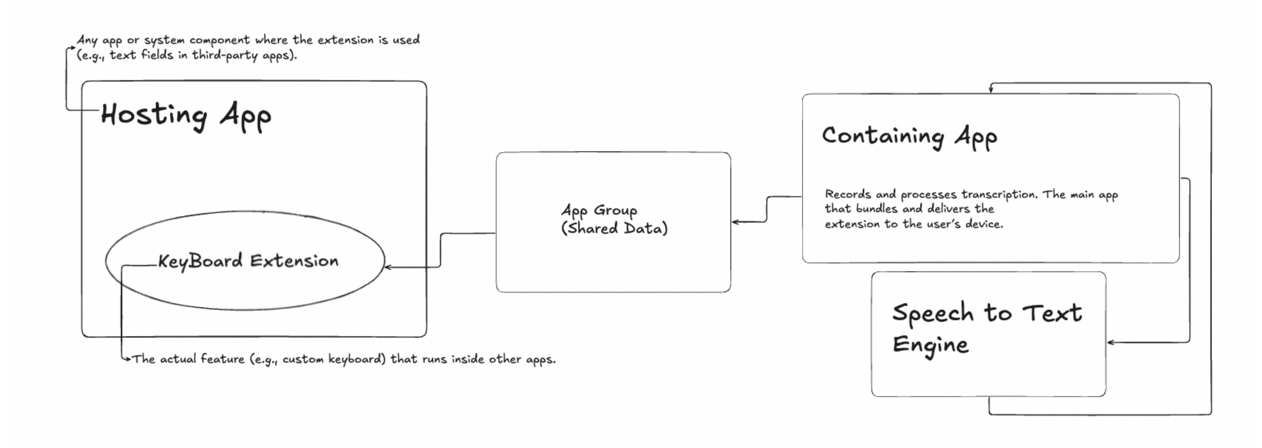
# Building a Custom iOS Keyboard Extension with Push-to-Talk Transcription

## Introduction

For my assignment, I explored the development of a custom iOS keyboard extension that allows users to press a microphone button and insert transcribed speech into any text field. On the surface, this looks simple: tap mic → record → transcribe → insert text. However, while working on it, I quickly ran into the **sandbox restrictions** Apple imposes on keyboard extensions.

This case study documents the approaches I tried, the limitations I faced, and the workaround I finally adopted. The result images are attached in the document at the end.

## Approach Discussion


Any app or system component where the extension is used (e.g., text fields in third-party apps).
Hosting App
KeyBoard Extension
The actual feature (e.g., custom keyboard) that runs inside other apps.
App Group (Shared Data)
Containing App
Records and processes transcription. The main app that bundles and delivers the extension to the user's device.
Speech to Text Engine

## Initial Approach: Direct Recording in the Keyboard Extension

My first instinct was straightforward:

- Add a mic button to the keyboard layout.
- Use `AVAudioSession` and `AVAudioRecorder` inside the keyboard extension to capture audio.

- Run speech-to-text on the audio and insert the result.

### Result

This approach failed almost immediately. Every attempt to activate the audio session inside the keyboard returned the error:

```
Session activation failed (Code: 560557684)
```

After digging deeper, I learned that Apple deliberately blocks microphone usage in third-party keyboards. The reason is obvious in hindsight — if custom keyboards could record audio at will, it would create a massive privacy risk.

### Limitation

**Microphone access is not allowed inside keyboard extensions.**

## Second Approach: Delegating Recording to the Host App

Realizing that the keyboard itself could not record, I shifted the responsibility to the host app (the main container app).

### Setup

- The keyboard extension and the host app shared an **App Group container**.
- When the user tapped the mic button on the keyboard, the extension wrote a “recording requested” flag to share `UserDefaults`.
- I then attempted to have the host app pick this up and start recording in the background.

### Result

This didn’t work as expected either. IOS does not let normal apps run microphone recording silently in the background, especially when triggered by another process. Unless the host app is in the foreground and has the mic entitlement active, the system simply blocks it.

### Limitation

**Background audio recording is heavily restricted.**

## Optimized Approach: Foreground Delegation via App Switch

Finally, I arrived at the most realistic workaround:

1. The user taps the mic button on the keyboard.
2. Keyboard extension triggers the host app via a custom URL scheme or `requestSceneSessionActivation`.
3. The host app comes to the foreground and provides a proper push-to-talk UI.
    - Press & hold = start recording.
    - Release = stop and transcribe.
4. The transcribed text is saved into the shared App Group container.
5. When the user switches back to the keyboard, it reads the saved transcription and inserts it into the text field.

### Why this works

- The host app has full mic access and can manage `AVAudioSession` properly.
- The keyboard stays within Apple's restrictions while still being able to insert the transcription.

### Limitation

- The user has to switch out to the main app briefly, which interrupts the typing flow.
- However, this is the best compromise given iOS's strict sandboxing.

## Transcription Process Discussion:

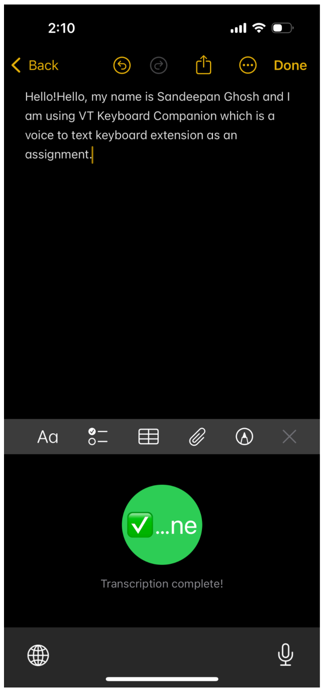
- The audio file path is subsequently transmitted to TranscriptionClient.uploadAudio(filePath:).
- TranscriptionClient validates the file (confirming existence and non-empty status).
- The audio is then uploaded to the Groq Whisper API (utilizing the **whisper-large-v3** model).
- The API response (a JSON object containing "text") is parsed.
- The resulting data is stored in UserDefaults (within the App Group) for shared access with the keyboard.
- Notifications are disseminated to facilitate UI updates (keyboard/application) and text insertion.

```
curl https://api.groq.com/openai/v1/audio/transcriptions \
  -H "Authorization: bearer ${GROQ_API_KEY}" \
  -F "file=@./sample_audio.m4a" \
  -F model=whisper-large-v3 \
  -F prompt="Specify context or spelling" \
  -F language="en" \
  -F temperature=0 \
  -F response_format=json
```

## User Flow:

1. User switches to VTKeyboard in any app
2. User taps microphone button in keyboard
3. Keyboard extension signals main app via App Group
4. Main app opens and automatically starts recording
5. Audio is transcribed using Groq AI
6. Transcribed text is sent back to keyboard
7. Text is automatically inserted into the active field input cursor.

## Results:

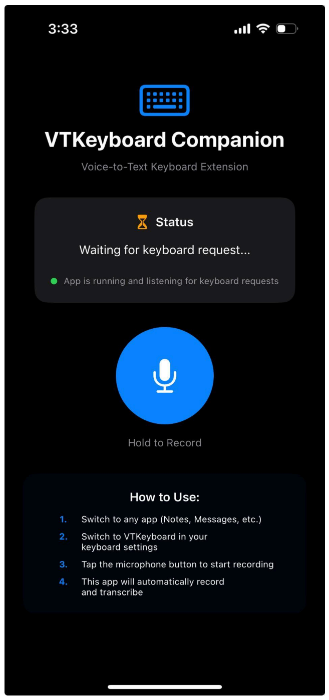


2:10

Back · Done

Hello!Hello, my name is Sandeepan Ghosh and I am using VT Keyboard Companion which is a voice to text keyboard extension as an assignment.

Transcription complete!



3:33

VTKeyboard Companion

Voice-to-Text Keyboard Extension

Status

Waiting for keyboard request...

● App is running and listening for keyboard requests

Hold to Record

How to Use:

1. Switch to any app (Notes, Messages, etc.)
2. Switch to VTKeyboard in your keyboard settings
3. Tap the microphone button to start recording
4. This app will automatically record and transcribe