iOS Future Scope & Conversion Guide for RecordX

This document explains how the current Android app can be converted into an iOS app.

1. Reusable Components (That can be used directly on iOS)

The libraries we are using are cross-platform and will also work on iOS:

React Native Core: UI components (View, Text, Image, etc.) will remain the same.

react-native-vision-camera: Camera functionality will work on iOS with the same code.

react-native-video: Video playback is supported on iOS.

react-native-fs: File system access will work on iOS as well (only paths need to be changed).

@react-native-camera-roll/camera-roll: Gallery access is supported on iOS.

@react-native-picker/picker: The UI picker will work on iOS with a native look.

react-native-safe-area-context: Handling for iPhone notch and dynamic island is already covered.

2. Required Changes (That need to be done)

To make the app fully functional on iOS, the following changes are required:

A. Custom Native Module (VideoMerger)

Current Status: The `VideoMerger` module is currently written only in Java for Android (`android/app/src/main/java/com/basicapp/VideoMerger.java`).

Action Required: An equivalent module for iOS needs to be written using Swift or Objective-C (using the AVFoundation framework to merge videos).

B. Permissions (Info.plist)

On Android we use `AndroidManifest.xml`; on iOS we need to add permissions in `Info.plist`:

`NSCameraUsageDescription`: For camera access.

`NSMicrophoneUsageDescription`: For audio recording.

`NSPhotoLibraryUsageDescription`: For saving videos to the gallery.

`NSPhotoLibraryAddUsageDescription`: For adding items to the gallery.

C. File Paths

Current Code: `RNFS.ExternalStorageDirectoryPath` is being used, which is Android-only.

Change: For iOS, we need to use `RNFS.DocumentDirectoryPath` or `RNFS.LibraryDirectoryPath`.

D. Platform Specific Code

In `App.tsx`, usage of `PermissionsAndroid` should be inside a condition for Android only (it is already under `Platform.OS === 'android'`, but this should be verified).

E. For Telemetry Data Access

1. Native Module (Major Change)

- **Android**: Currently we are using TelemetryModule.kt (Kotlin).
- **iOS**: You will need to create a new module TelemetryModule.swift (Swift) or in Objective-C.

2. APIs (These are different on iOS)

- **CPU**: On Android we read /proc/self/stat. On iOS we need to use Mach kernel APIs (task_info, thread_info).
- **Memory**: On Android we use ActivityManager. On iOS we use mach_task_basic_info.
- **GPU**: On iOS, getting real-time GPU usage is very difficult (Apple does not allow direct access). We may need to skip GPU data or use **MetricKit** (which gives aggregated reports after ~24 hours).

3. Latency Logic: The Date.now() based latency calculation will work the same way on iOS.

3. Steps to Build for iOS

1. Mac Machine: macOS is required to build an iOS app.

2. Install Dependencies: Go to the `ios` folder and run `pod install`.

3. Xcode Setup: Open the project in Xcode and configure Signing & Capabilities (Apple Developer Account required).

4. Run: `npx react-native run-ios`.