# iOS + React Native SDK Integration Documentation

## Prerequisites

### Tools and Software

1. **Xcode**: Download and install Xcode from the Mac App Store.
2. **Node.js and npm**: Install from the Node.js website.
   `brew install node`
3. **React Native CLI**: Install globally using npm.
   `npm install -g react-native-cli`
4. **CocoaPods**: Install using RubyGems.
   `sudo gem install cocoapods`

## Step 1: Project Setup

### Native iOS App Setup

1. **Create a new iOS project** in Xcode or use an existing project.
2. **Configure the necessary permissions** in the `Info.plist` file to ensure the app has the required access:

   Ex:

```
<key>NSCameraUsageDescription</key>

<string>We need access to your camera to capture the secondary ID</string>

<key>NSFaceIDUsageDescription</key>

<string>We need access to Face ID for authentication purposes</string>

<key>NSFaceIDAuthenticationReason</key>

<string>Use Face ID to authenticate</string>
```

3. **Organize your project structure**:
    - Create directories for your native iOS code and React Native code to maintain a clean and organized project structure.

### React Native App Setup

1. **Initialize a new React Native project** within your iOS project directory:
   `npx react-native init YourReactNativeProject`
2. **Ensure the React Native project is fully functional** by running it independently:

   ```
   cd YourReactNativeProject

   npx react-native run-ios
   ```

## Step 2: Podfile Configuration

**Create or update your Podfile** to include the necessary dependencies:

```
require 'json'


platform :ios, '13.4'

install! 'cocoapods', :deterministic_uuids => false


target 'YourNativeApp' do

  use_expo_modules!

  config = use_native_modules!


  use_react_native!(

    :path => config[:reactNativePath],
```

```
    :hermes_enabled => podfile_properties['expo.jsEngine'] == nil ||
podfile_properties['expo.jsEngine'] == 'hermes',

    :app_path => "#{Pod::Config.instance.installation_root}/..",

    :privacy_file_aggregation_enabled =>
podfile_properties['apple.privacyManifestAggregationEnabled'] !=
'false',

  )


  post_install do |installer|

    react_native_post_install(installer)

  end

end
```

**Install the dependencies** by running the following command in the ios directory:

```
cd ios

pod install
```

## Step 3: AppDelegate Configuration

**Modify your AppDelegate.h** to declare the necessary properties and import headers:

```
#import <RCTAppDelegate.h>

#import <UIKit/UIKit.h>


@interface AppDelegate : RCTAppDelegate
```

```
@property (nonatomic, strong) UIWindow *window;

@end
```

**Update your AppDelegate.m to set up the React Native bridge and handle the JavaScript bundle:**

```
#import "AppDelegate.h"

#import <React/RCTBundleURLProvider.h>

#import <React/RCTLinkingManager.h>


@implementation AppDelegate


- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions

{

  self.moduleName = @"main";

  self.initialProps = @{};


  self.window = [[UIWindow alloc] initWithFrame:[UIScreen
mainScreen].bounds];

  UIViewController *rootViewController = [UIViewController new];

  self.window.rootViewController = rootViewController;

  [self.window makeKeyAndVisible];


  return [super application:application
didFinishLaunchingWithOptions:launchOptions];
```

```
}

- (NSURL *)sourceURLForBridge:(RCTBridge *)bridge
{
  return [self bundleURL];
}

- (NSURL *)bundleURL
{
  #if DEBUG
  return [[RCTBundleURLProvider sharedSettings]
jsBundleURLForBundleRoot:@"index"];
  #else
  return [[NSBundle mainBundle] URLForResource:@"main"
withExtension:@"jsbundle"];
  #endif
}

// Linking API
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url
options:(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options {
  return [super application:application openURL:url options:options]
|| [RCTLinkingManager application:application openURL:url
options:options];
}
```

```
// Universal Links

- (BOOL)application:(UIApplication *)application
continueUserActivity:(nonnull NSUserActivity *)userActivity
restorationHandler:(nonnull void
(^)(NSArray<id<UIUserActivityRestoring>> *
_Nullable))restorationHandler {

  BOOL result = [RCTLinkingManager application:application
continueUserActivity:userActivity
restorationHandler:restorationHandler];

  return [super application:application
continueUserActivity:userActivity
restorationHandler:restorationHandler] || result;

}


// Explicitly define remote notification delegates to ensure
compatibility with some third-party libraries

- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken

{

  return [super application:application
didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];

}


- (void)application:(UIApplication *)application
didFailToRegisterForRemoteNotificationsWithError:(NSError *)error

{
```

```
  return [super application:application
didFailToRegisterForRemoteNotificationsWithError:error];

}


- (void)application:(UIApplication *)application
didReceiveRemoteNotification:(NSDictionary *)userInfo
fetchCompletionHandler:(void
(^)(UIBackgroundFetchResult))completionHandler

{

  return [super application:application
didReceiveRemoteNotification:userInfo
fetchCompletionHandler:completionHandler];

}


@end
```

1.

## Step 4: Creating a SwiftUI View to Display React Native Components

**Create a Swift file** for integrating the React Native view:

```
import SwiftUI

import React


struct ReactNativeView: UIViewControllerRepresentable {

  func makeUIViewController(context: Context) -> UIViewController {

    let jsCodeLocation: URL
```

```
    #if DEBUG

    jsCodeLocation = URL(string:
"http://localhost:8081/index.bundle?platform=ios")!

    #else

    jsCodeLocation = Bundle.main.url(forResource: "main",
withExtension: "jsbundle")!

    #endif


    let rootView = RCTRootView(

      bundleURL: jsCodeLocation,

      moduleName: "YourReactNativeModule",

      initialProperties: nil,

      launchOptions: nil

    )


    let viewController = UIViewController()

    viewController.view = rootView

    return viewController

  }


  func updateUIViewController(_ uiViewController: UIViewController,
context: Context) {}

}
```

## Step 5: Integrating the React Native View into SwiftUI

**Integrate the React Native view into your SwiftUI view**:

```
struct ContentView: View {
  @State private var showReactNativeView = false

  var body: some View {
    VStack {
      Button(action: {
        showReactNativeView = true
      }) {
        Text("Launch React Native View")
      }
      .sheet(isPresented: $showReactNativeView) {
        ReactNativeView()
      }
    }
  }
}
```

## Running the Project

### Using Metro Bundler

1. **Start Metro Bundler:**
    - Open your terminal and navigate to the root of your project folder.

Run the following command to start the Metro Bundler:
`yarn start`

2. **Run the Project in Xcode Simulator:**
    - Open Xcode and load your project.
    - Select the desired simulator device from the top bar.
    - Click the 'Run' button to build and run the project.
    - The app should now run in the simulator, connecting to the Metro Bundler for loading JavaScript.

### Using Release Bundle

1. **Create a Release Bundle:**
    - Open your terminal and navigate to the root of your project folder.

Run the following command to bundle the JavaScript for release:
`npx react-native bundle --platform ios --dev false --entry-file index.js --bundle-output ios/main.jsbundle --assets-dest ios`

2. **Configure Xcode for Release Bundle:**
    - Open Xcode and load your project.
    - Open the `ContentView.swift` file (or the relevant file where you configure the React Native view).

Modify the `jsCodeLocation` to point to the release bundle instead of the Metro Bundler URL:

```
#if DEBUG

let jsCodeLocation = URL(string:
"http://localhost:8081/index.bundle?platform=ios")!

#else
```

```
let jsCodeLocation = Bundle.main.url(forResource: "main",
withExtension: "jsbundle")!

#endif
```

    - 

3. **Run the Project in Xcode Simulator:**
    - Select the desired simulator device from the top bar in Xcode.
    - Click the 'Run' button (or use `Cmd + R`) to build and run the project.
    - The app should now load the JavaScript from the bundled `main.jsbundle` file instead of the Metro Bundler.