Member-only story

How to integrate Native modules in React Native Apps ? Explained

Gopesh Jangid · Follow
4 min read · May 7, 2023

Listen Share More

React Native is a popular framework for developing cross-platform mobile applications. It allows developers to build high-quality mobile apps with the use of JavaScript and React. Native modules are an essential part of React Native development that helps developers to add native functionality to the app. In this article, we will discuss how to integrate native modules in React Native apps with examples.

What are Native Modules?

Native modules are a set of code that allows React Native to interact with the native environment of the device. They are written in the native language of the platform such as Objective-C, Swift for iOS, and Java, Kotlin for Android. Native modules are used when the functionality that is required is not available in React Native APIs or libraries.

Integrating Native Modules in React Native Apps

Integrating native modules in React Native apps is a simple process. The following are the steps involved:

Step 1: Create a new Native Module

To create a new native module, we need to create a new module with the required functionality. We can create a native module using the following command:

```
react-native create-library <library-name>
```

This will create a new directory with the specified library name. Inside the directory, there will be an iOS and an Android subdirectory, which contains the code for the native modules for each platform.

Step 2: Implement Native Code

Next, we need to implement the native code for the native module. We can implement the native code for iOS and Android separately. To implement the native code, we need to follow the platform-specific instructions.

For iOS, we need to implement the native code in Objective-C or Swift. We can find the instructions for implementing the native code for iOS in the official documentation of React Native.

For Android, we need to implement the native code in Java or Kotlin. We can find the instructions for implementing the native code for Android in the official documentation of React Native.

Step 3: Export Native Code

After implementing the native code, we need to export it so that it can be used in React Native. We can export the native code using the following command:

```
module.exports = NativeModules.<module-name>;
```

This will export the module with the specified name. We can then use this module in our React Native app.

Step 4: Use Native Module in React Native App

To use the native module in our React Native app, we need to import the native module into our JavaScript code. We can import the native module using the following command:

```
import { <module-name> } from '<library-name>';
```

This will import the native module with the specified name from the specified library. We can then use this module in our JavaScript code.

Example of using Native Module in React Native App

Let's take an example of integrating the native module to get the battery level of the device.

Step 1: Create a new Native Module

We can create a new native module using the following command:

```
react-native create-library BatteryModule
```

This will create a new directory named BatteryModule with the required files.

Step 2: Implement Native Code

Next, we need to implement the native code for the battery module. We can implement the native code for iOS and Android separately. To implement the native code, we need to follow the platform-specific instructions.

For iOS, we can implement the native code in Objective-C or Swift. We can use the following code to get the battery level on iOS:

```
#import "BatteryModule.h"
#import <UIKit/UIKit.h>

@implementation BatteryModule

RCT_EXPORT_MODULE();
```

```
RCT_EXPORT_METHOD(getBatteryLevel:(RCTPromiseResolveBlock)resolve
                  rejecter:(RCTPromiseRejectBlock)reject)
{
  UIDevice *device = UIDevice.currentDevice;
  device.batteryMonitoring
/ Check if battery monitoring is available
if (device.batteryMonitoringEnabled == NO) {
device.batteryMonitoringEnabled = YES;
}

// Get battery level
float batteryLevel = [device batteryLevel];

if (batteryLevel >= 0) {
resolve(@(batteryLevel));
} else {
NSError *error = [NSError errorWithDomain:@"BatteryModule" code:1 userInfo:nil
reject(@"Battery Level Error", @"Could not get battery level", error);
}
}
```

For Android, we can implement the native code in Java or Kotlin. We can use the following code to get the battery level on Android:

```
package com.batterymodule;

import android.os.BatteryManager;
import android.content.Context;
import com.facebook.react.bridge.Promise;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.ReactMethod;

public class BatteryModule extends ReactContextBaseJavaModule {

  private final ReactApplicationContext reactContext;

  public BatteryModule(ReactApplicationContext reactContext) {
    super(reactContext);
    this.reactContext = reactContext;
  }

  @Override
  public String getName() {
    return "BatteryModule";
  }
```

```
    @ReactMethod
    public void getBatteryLevel(Promise promise) {
      try {
        Intent batteryIntent = this.reactContext.registerReceiver(null, new Inte
        int level = batteryIntent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
        int scale = batteryIntent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
        float batteryLevel = level / (float)scale;
        promise.resolve(batteryLevel);
      } catch (Exception e) {
        promise.reject("Battery Level Error", e.getMessage());
      }
    }
  }
```

Step 3: Export Native Code

After implementing the native code, we need to export it so that it can be used in React Native. We can export the native code using the following command:

```
module.exports = NativeModules.BatteryModule;
```

This will export the module with the name “BatteryModule”.

Step 4: Use Native Module in React Native App

To use the battery module in our React Native app, we need to import the native module into our JavaScript code. We can import the native module using the following command:

```
import { BatteryModule } from 'BatteryModule';
```

This will import the battery module from the “BatteryModule” library.

We can then use the “BatteryModule” in our React Native app as follows:

```
import React, { useState, useEffect } from 'react';
import { View, Text } from 'react-native';
import { BatteryModule } from 'BatteryModule';

const App = () => {
  const [batteryLevel, setBatteryLevel] = useState(null);

  useEffect(() => {
    BatteryModule.getBatteryLevel()
      .then(level => setBatteryLevel(level))
      .catch(error => console.log(error));
  }, []);

  return (
    <View>
      <Text>Battery Level: {batteryLevel}</Text>
    </View>
  );
};

export default App;
```

This will display the battery level of the device in our React Native app.

In conclusion, integrating native modules in React Native apps allows developers to add native functionality to their apps. It is a simple process that involves creating a new native module, implementing the native code, exporting the native code, and using the native module in our React Native app. With the use of native modules, developers can build high-quality mobile apps with a native look and feel.

if you have learned something new today. And don't forget to ***follow*** *and* ***and send claps for appriciations.***

Please follow me

@ Gopesh Jangid

Thank you !!!

React Native React Native Modules React Native Application Reactlearning

React Native Learning