



React Native - Native Modules

Android with Java

Introduction

- This project introduces native modules for react native and android using java. This simple project will read the battery status from android code and use it from react native. We will study how to communicate our javascript code with the android platform and vice versa.
- The scope of this project will be gradually increasing to cover more native's proofs of concepts and get more understanding of how Native Modules works.
- In the same way we use Java for this project, it's possible to use Kotlin. Feel free to experiment with it.

- The repository of this article can be found here: <https://github.com/osielmesa/BatteryManager> and every section has its own branch in it.

Create and register the native module in Android

Create a react native project in the location you want using react native CLI:

```
npx react-native@latest init BatteryManager
```

Open the android folder in Android Studio

Let's create our first custom module:

Create a java class under your android package (ex. java/com.batterymanager). Let's call it BatteryModule.

In order to use this class from our JS code we need to import several packages at the top of the file:

```
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.bridge.ReactContext;
import com.facebook.react.bridge.ReactContextBaseJavaModule;
import com.facebook.react.bridge.ReactMethod;
```

We need to extend our java class from ReactContextBaseJavaModule and implement the getName method that will return the name of our module:

```
public class BatteryModule extends ReactContextBaseJavaModule {

    BatteryModule(ReactApplicationContext context) {
        super(context);
    }

    @NonNull
    @Override
    public String getName() {
        //This is the name we are going to use from JS to access this module
        return "BatteryModule";
    }
}
```

```
}  
}
```

Now we are going to create a function that will be called from JS passing two parameters. This function will log in Android native those two parameters (tag and description). When we want to expose a function to JS we need to use the `@ReactMethod` decorator:

```
@ReactMethod  
public void nativeLogger(String tag, String description) {  
    Log.d(tag, description);  
}
```

This is how our `BatteryModule` class looks so far:

```
package com.batterymanager;  
  
import androidx.annotation.NonNull;  
  
import com.facebook.react.bridge.NativeModule;  
import com.facebook.react.bridge.ReactApplicationContext;  
import com.facebook.react.bridge.ReactContext;  
import com.facebook.react.bridge.ReactContextBaseJavaModule;  
import com.facebook.react.bridge.ReactMethod;  
import android.util.Log;  
  
public class BatteryModule extends ReactContextBaseJavaModule {  
  
    // This is the constructor of the class  
    BatteryModule(ReactApplicationContext context) {  
        super(context);  
    }  
  
    @NonNull  
    @Override  
    public String getName() {  
        //This is the name we are going to use from javascript to access this module  
        return "BatteryModule";  
    }  
  
    /**  
     * Android logger function  
     * Using @ReactMethod decorator is how we expose a function to react native  
     * @param tag The tag to use in the log  
     * @param description The description to use in the log  
     */  
}
```

```

    @ReactMethod
    public void nativeLogger(String tag, String description) {
        Log.d(tag, description);
    }
}

```

Now let's register our **BatteryModule** class with React Native:

For this we will need to create a class that will extend from `ReactPackage` class. This class will register our modules and let react-native knows about them. Let's create a java class inside our package and name it `AppPackage`:

```

package com.batterymanager;

import com.facebook.react.ReactPackage;
import com.facebook.react.bridge.NativeModule;
import com.facebook.react.bridge.ReactApplicationContext;
import com.facebook.react.uimanager.ViewManager;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class AppPackage implements ReactPackage {

    @Override
    public List<ViewManager> createViewManagers(ReactApplicationContext
reactContext) {
        return Collections.emptyList();
    }

    @Override
    public List<NativeModule> createNativeModules(ReactApplicationContext
reactContext) {
        List<NativeModule> modules = new ArrayList<>();

        modules.add(new BatteryModule(reactContext));


        return modules;
    }
}

```

In the method `createNativeModules` we create and return a list of `NativeModule` that will contain all our modules, for now we only have a `BatteryModule`. Keep in mind this list has to be filled with new instances of our modules.

To finish registering all our modules we need to add our `AppPackage` class to the list of packages returned in the function `getPackages` of the already existing class `MainApplication`, in our case it should look like this:

```
@Override
protected List<ReactPackage> getPackages() {
    @SuppressWarnings("UnnecessaryLocalVariable")
    List<ReactPackage> packages = new PackageList(this).getPackages();
    // Packages that cannot be autolinked yet can be added manually here, for example:
    // packages.add(new MyReactNativePackage());
    packages.add(new AppPackage()); // This is our register class instance
    return packages;
}
```

After this step we can go ahead and build our project using the green hammer icon in Android Studio: 

Congratulations! This is our first native android module for react-native. For now it's very simple, just a native logger function but it proves how to create a module, register it and expose methods to react native. You can find how the project looks so far by checking out this branch: [1/Create and register the native module in Android](#). Now let's continue by testing how to use our logger method from react native.

Use the native module from react-native

It's time to test. Let's log something in the android native console. The goal is to send data (tag and description) from JS side to Android side and log it.

Using the IDE you prefer, let's open the react-native project and position yourself on the file App.tsx. In this file let's import NativeModules and Button from react-native:

```
import { SafeAreaView, ScrollView, StatusBar, useColorScheme, View, NativeModules, Button } from 'react-native';
```

The idea is to create a function that will invoke our module when the button is pressed. Let's create that with some basic style:

```
/**
 * Sample React Native App
 * https://github.com/facebook/react-native
 *
 * @format
 */

import React from 'react';
import { SafeAreaView, StatusBar, useColorScheme, View, NativeModules, Button,
StyleSheet } from 'react-native';
function App(): JSX.Element {
  const isDarkMode = useColorScheme() === 'dark';

  const onPress = () => {
    console.log('We will invoke the native module here!');
  };

  return (
    <SafeAreaView style={styles.container}>
      <StatusBar barStyle={isDarkMode ? 'light-content' : 'dark-content'} />
      <View>
        <Button title="Click to log data in android console!" color="#303F9F"
onPress={onPress} />
      </View>
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
});

export default App;
```

Now let's add a variable to our App component that using destructuring will contain the reference to our BatteryModule. With this variable we will be able to access our exposed methods in the module:

```
const { BatteryModule } = NativeModules; // here we use the name returned in our
getName method of BatteryModule java class
```

It's time to call our method from react-native in the onPress function:

```
const onPress = () => {
  BatteryModule.nativeLogger('RN to Android', 'This is a log coming from JS');
};
```

Our final App.tsx file:

```
/**
 * Sample React Native App
 * https://github.com/facebook/react-native
 *
 * @format
 */

import React from 'react';
import { SafeAreaView, StatusBar, useColorScheme, View, NativeModules, Button,
StyleSheet } from 'react-native';
function App(): JSX.Element {
  const isDarkMode = useColorScheme() === 'dark';
  const { BatteryModule } = NativeModules; // here we use the name returned in our
getName method of BatteryModule java class
  const onPress = () => {
    BatteryModule.nativeLogger('RN to Android', 'This is a log coming from JS');
  };

  return (
    <SafeAreaView style={styles.container}>
      <StatusBar barStyle={isDarkMode ? 'light-content' : 'dark-content'} />
      <View>
        <Button title="Click to log data in android console!" color="#303F9F"
onPress={onPress} />
      </View>
    </SafeAreaView>
  );
}
```

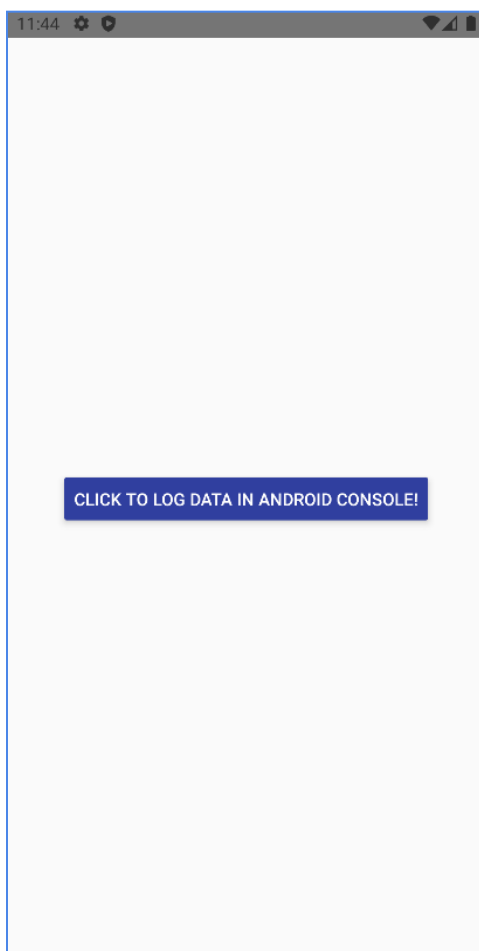
```
}  
  
const styles = StyleSheet.create({  
  container: {  
    flex: 1,  
    justifyContent: 'center',  
    alignItems: 'center',  
  },  
});  
  
export default App;
```

Now let's check in android logcat our native log

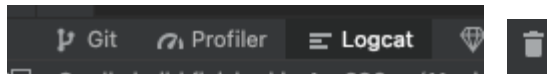
If you didn't so far build the project in Android Studio using the green hammer.

Run your react-native project on a simulator or physical device as usual:

yarn start and in a separate console **yarn android**:



Head to Android Studio and open the Logcat tab at the bottom and clear the console in the Logcat top left trash icon:



Then hit the button in the App and voila:



The log should be visible with the data you sent from the JS side.

The road so far: We create, register and expose a method in a native module using java. We import this Native module in react-native side and send data from JS to Android specifically to our exposed method as parameters. The current status can be found in this branch: [2/Use_the_native_module_from_react-native](#)

What's next? We are going to increase our proof of concept by sending data from Android to the JS side using a callback. I know we still do not have anything yet related to the battery status on the device but we will get there.