Integration with an Android Fragment

The guide for Integration with Existing Apps details how to integrate a full-screen React Native app into an existing Android app as an Activity. To use React Native components within Fragments in an existing app requires some additional setup. The benefit of this is that it allows for a native app to integrate React Native components alongside native fragments in an Activity.

1. Add React Native to your app

Follow the guide for Integration with Existing Apps until the Code integration section. Continue to follow Step 1. Create an `index.android.js` file and Step 2. Add your React Native code from this section.

2. Integrating your App with a React Native Fragment

You can render your React Native component into a Fragment instead of a full screen React Native Activity. The component may be termed a "screen" or "fragment" and it will function in the same manner as an Android fragment, likely containing child components. These components can be placed in a `/fragments` folder and the child components used to compose the fragment can be placed in a `/components` folder.

You will need to implement the `ReactApplication` interface in your main Application Java/Kotlin class. If you have created a new project from Android Studio with a default activity, you will need to create a new class (e.g. `MyReactApplication.java` or `MyReactApplication.kt`). If it is an existing class you can find this main class in your `AndroidManifest.xml` file. Under the `<application />` tag you should see a property `android:name` e.g. `android:name=".MyReactApplication"`. This value is the class you want to implement and provide the required methods to.

Ensure your main Application class implements ReactApplication:

Java Kotlin

```
public class MyReactApplication extends Application implements ReactApplication {...}
```

Override the required methods `getUseDeveloperSupport`, `getPackages` and `getReactNativeHost`:

Java Kotlin

```
public class MyReactApplication extends Application implements ReactApplication {
    @Override
    public void onCreate() {
        super.onCreate();
        SoLoader.init(this, false);
    }

    private final ReactNativeHost mReactNativeHost = new DefaultReactNativeHost(this) {
        @Override
        public boolean getUseDeveloperSupport() {
            return BuildConfig.DEBUG;
        }

        protected List<ReactPackage> getPackages() {
            List<ReactPackage> packages = new PackageList(this).getPackages();
            // Packages that cannot be autolinked yet can be added manually here
            return packages;
        }
    };

    @Override
    public ReactNativeHost getReactNativeHost() {
        return mReactNativeHost;
    }
}
```

If you are using Android Studio, use Alt + Enter to add all missing imports in your class. Alternatively these are the required imports to include manually:

Java Kotlin

```
import android.app.Application;
```

```
import com.facebook.react.PackageList;
import com.facebook.react.ReactApplication;
import com.facebook.react.ReactNativeHost;
import com.facebook.react.ReactPackage;
import com.facebook.react.defaults.DefaultReactNativeHost;
import com.facebook.soloader.SoLoader;

import java.util.List;
```

Perform a "Sync Project files with Gradle" operation.

Step 3. Add a FrameLayout for the React Native Fragment

You will now add your React Native Fragment to an Activity. For a new project this Activity will be `MainActivity` but it could be any Activity and more fragments can be added to additional Activities as you integrate more React Native components into your app.

First add the React Native Fragment to your Activity's layout. For example `main_activity.xml` in the `res/layouts` folder.

Add a `<FrameLayout>` with an id, width and height. This is the layout you will find and render your React Native Fragment into.

```
<FrameLayout
    android:id="@+id/reactNativeFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Step 4. Add a React Native Fragment to the FrameLayout

To add your React Native Fragment to your layout you need to have an Activity. As mentioned in a new project this will be `MainActivity`. In this Activity add a button and an event listener. On button click you will render your React Native Fragment.

Modify your Activity layout to add the button:

```
<Button
    android:layout_margin="10dp"
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Show react fragment" />
```

Now in your Activity class (e.g. `MainActivity.java` or `MainActivity.kt`) you need to add an `OnClickListener` for the button, instantiate your `ReactFragment` and add it to the frame layout.

Add the button field to the top of your Activity:

Java Kotlin

```
private Button mButton;
```

Update your Activity's `onCreate` method as follows:

Java Kotlin

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);

    mButton = findViewById(R.id.button);
    mButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Fragment reactNativeFragment = new ReactFragment.Builder()
                    .setComponentName("HelloWorld")
                    .setLaunchOptions(getLaunchOptions("test message"))
                    .build();

            getSupportFragmentManager()
                    .beginTransaction()
                    .add(R.id.reactNativeFragment, reactNativeFragment)
                    .commit();
```

```
        }
    });
}
```

In the code above `Fragment reactNativeFragment = new ReactFragment.Builder()` creates the ReactFragment and `getSupportFragmentManager().beginTransaction().add()` adds the Fragment to the Frame Layout.

If you are using a starter kit for React Native, replace the "HelloWorld" string with the one in your `index.js` or `index.android.js` file (it's the first argument to the AppRegistry.registerComponent() method).

Add the `getLaunchOptions` method which will allow you to pass props through to your component. This is optional and you can remove `setLaunchOptions` if you don't need to pass any props.

Java Kotlin

```java
private Bundle getLaunchOptions(String message) {
    Bundle initialProperties = new Bundle();
    initialProperties.putString("message", message);
    return initialProperties;
}
```

Add all missing imports in your Activity class. Be careful to use your package's BuildConfig and not the one from the facebook package! Alternatively these are the required imports to include manually:

Java Kotlin

```java
import android.app.Application;

import com.facebook.react.ReactApplication;
import com.facebook.react.ReactNativeHost;
import com.facebook.react.ReactPackage;
import com.facebook.react.shell.MainReactPackage;
import com.facebook.soloader.SoLoader;
```

Perform a "Sync Project files with Gradle" operation.

Step 5. Test your integration

Make sure you run `yarn` to install your react-native dependencies and run `yarn native` to start the metro bundler. Run your android app in Android Studio and it should load the JavaScript code from the development server and display it in your React Native Fragment in the Activity.

Step 6. Additional setup - Native modules

You may need to call out to existing Java/Kotlin code from your react component. Native modules allow you to call out to native code and run methods in your native app. Follow the setup here native-modules-android

*Last updated on **Jun 21, 2023***