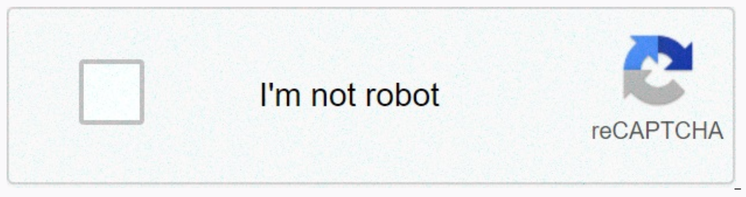

Continue

Generate apk android studio react native

An Android Package Kit (APK) is the package file format used by Android OS for the distribution and installation of mobile apps. It is similar to the .exe you have on Windows OS, a .apk file is for Android. What can I do with it? A troubleshooting file .apk allows you to install and test your app before you publish it to app stores. Mind you, this is not yet ready for release and there are quite a few things you need to do to before you can publish. Nevertheless, it will be useful for initial distribution and testing. You must enable troubleshooting settings on your phone to run this apk. Premise: react-native version > 0.57 How to generate one in 3 steps? Step 1: Go to the root of the project in the terminal and run the following command: respond-native bundle --platform android --dev fake --entry-file index.js --bundle-output android/app/src/main/assets/index.android.bundle --assets-dest android/app/src/main/res Step 2: Go to Android library: CD Step 3: Now in this android, android, android folder run this command ./gradlew assembleDebug There! you can find apk file in the following way: yourProject/Android/app/build/output/apk/debug/app-debug.apk Now you have your .apk file generated, install it on your Android phone and enjoy! Thanks. When you start developing a mobile application using React Native, the main problem that comes to your mind when your boss asks you to give something to test for QAs or I want to see it on my Android phone.... is, How should I build an APK with React Native? Well, don't worry! I'm going to guide you through to solve your problem and create an APK successfully. First of all, I think you noticed a folder of Android implementations like Android in your React Native project. The next sy most important is to create a private signing key. Create a private signing keyTo create a signing key, you can use keytool . If you are using Windows, keytool can be found at C:\Program Files\Java\jdkx.x.x_x\bin . Open a command prompt with Run as administrator. Run the following code, which generates a keystore valid for 10000 days under the file name my-key.keystore.keytool -v-keystore my-keystore -a.k.a. my-key-alias -keyalg RSA-keysize 2048-validity 10000Alias is a name that you will use later when signing your app, remember to take note of the alias. Don't ever commit your keystore file to your version control and keep it private and safeNow you're done with the first step of creating an APK now! Add Keystore to your projectNow, place the created keystore file under the android/app folder in your original workbookNext, add the following lines to android\app\build.gradleAdd the following lines to gradle.propertiesNow you are ready to build your APK. Generating APKPlace your terminal library for Android using,Then following commandFor windows, windows, assembleReleaseFor Linux, ./gradlew assembleReleaseAssembleRelease in gradle will perform bundling JavaScript needed to run your app on APK.gradle.properties should not include org.gradle.configureondemand = true . If you allow this in your gradle.properties, this will not bundle JS and assets to your APKNow APK creation process is done! You can find the generated APK on android/app/build/output/apk/app-release.apkHey, Cheer up!!! Now your biggest problem is solved! :)Rea0, If your boss's demands are strengthened,Reduce the size of APKOki, now it's something your boss would ask for absolutely,So here you go.... You need to activate Proguard, which has some magical powers (well, not really magic, but it removes bytecode and dependencies that your app doesn't use) to reduce the size of your APK a little. Add the following line under buildTypes in android\app\build.gradleGood Luck with the release of your app! Fingers crossed!! :) Android requires all apps to digitally sign with a certificate before they can be installed. To distribute your Android application through the Google Play store, it must be signed with a release key, which must then be used for all future updates. Since 2017, Google Play is able to manage signing releases automatically thanks to App Signing by Google Play functionality. But before your application binary is uploaded to Google Play it must be signed with an upload key. The documentation for signing your applications in the Android developer documentation describes the topic in detail. This guide covers the process in brief, as well as a list of the steps required to package the JavaScript package. Generating an upload key#You can generate a private signing key using keytool. On Windows keytool must be run from C:\Programs\Java\jdkx.x.x_x\bin.$ keytool -genkeypair -v -keystore my-upload-keystore -a.k.a. my-key-a.k.a.,keyalg RSA -keysize 2048-validity 10000Dis command you for passwords for keystore and key and for Distinguished Name fields for your key. Then generates keystore as a file called my-upload-key.keystore.The keystore contains a single key, valid for 10000 days. The alias is a name that you'll need later when you sign your app, so be sure to note the alias. On mac, if you are not sure where your JDK bin folder is, then execute the following command to find it:It will print the folder of JDK, which will look something like this:/Library/Java/Java-VirtualEmkinner/jdkX.X.X_XXX.jdk/Content/HomeNavigate to this folder using the $ CD /your jdk/path command and use keyto ol command with sudo permission as shown below.$ sudo keytool -genkey -v -keystore my-upload-key.keystore -a.k.a. my-key-alias -keyalg RSA -keysize 2048 -validity 10000Note: Remember to keep the keystore file private. If you have lost the upload key or it is you should follow these instructions. Set up Gradle variables#Place the my-upload-key.keystore file under the android/app folder in your workbook. Edit the ~/.gradle/gradle.properties or android/gradle.properties file, and add the following (replace ***** with the correct keystore password, alias and key password), MYAPP_UPLOAD_STORE_FILE=my-upload key.keystoreMYAPP_UPLOAD_KEY_ALIAS=my-key-aliasMYAPP_UPLOAD_STORE_PASSWORD=*****MYAPP_UPLOAD_KEY_PASSWORD=****** These will be global Gradle variables that we can later use in our Gradle config to sign our app. Security note: If you're not keen on saving your passwords in plain text and you're running OSX, you can also save your credentials to the Keychain Access app. Then you can skip the last two rows to ~/.gradle/gradle.properties.Adding signing config to your app's Gradle config#The last configuration step to do is to configure publishing builds to be signed using the upload key. Edit the android/app/build.gradle file in the workbook and add signing conflict if (project.hasProperty('MYAPP_UPLOAD_STORE_FILE')) { storeFile file (MYAPP_UPLOAD_STORE_FILE) storePassword MYAPP_UPLOAD_STORE_PASSWORD keyAlias MYAPP_UPLOAD_KEY_ALIAS keyPassword MYAPP_UPLOAD_KEY_PASSWORD signingConfig signingConfigs... releaseGenerating release APK#Run the following in a terminal:$ ./gradlew bundleReleaseGradles bundleRelease will bring together all the JavaScript needed to run your app into the Android App Bundle (AAB). If you need to change the way the javascript package and/or the resources that can be pulled are unified (for example, if you changed the default file/folder name or the overall structure of the project), look at android/app/build.gradle to see how you can update it to reflect these changes. Note: Make sure gradle.properties do not include org.gradle.configureondemand = true, as that will make the release build skip bundling JS and assets in the app binary. The generated AAB can be found under Android/app/build/output/bundle/release/app.aab, and is ready to be uploaded to Google Play.Note: For Google Play to accept AAB format, the App Signing of Google Play must be configured for your application on the Google Play Console. If you're updating an existing app that doesn't use the Google Play App, see our download section to learn how to perform this configuration change. Test the release of your app#Before uploading the release build to the Play Store, be sure to test it thoroughly. First, remove any previous version of the app you've already installed. Install it on the device using the following project root command:$ npx react-native run-android --variant=releaseNote that --variant=release is only available if set up sign as described above. You can end all running bundler instances as all your code is bundled into APK's assets. Publishing to other stores#By default, the generated APK has the original code for both x86 and ARMv7a CPU architectures. This makes it easier to share APCs running on almost all Android devices. But this has the disadvantage that there will be some unused native code on any device, leading to unnecessarily larger APKs.You can create an APK for each CPU by changing the following line in android/app/build.gradle:- abiFilters armeabi-v7a, x86 - def enableSeparateBuildPerCPUArchitecture = false+ def enableSeparateBuildPerCPUArchitecture = trueUpload both of these files to markets that support device targeting, such as Google Play and Amazon AppStore, and users will automatically get the appropriate APK. If you wish to upload to other markets, such as APKFiles, which does not support multiple APKs for a single app, change the following line as well as creating standard universal APK with binaries for both CPUs.- universalApk false // If true, also generate a universal APK + universalApk true / / If true, also generate a universal APKEnabling Proguard to reduce the size of APK (optional)#Proguard is a tool that can slightly reduce the size of APK. It does this by removing parts of the React Native Java bytecode (and its dependencies) that your app doesn't use. IMPORTANT: Be sure to test your app thoroughly if you've enabled Proguard. Proguard often requires configuration specific to each original library you use. See app/proguard-rules.pro.To enable Proguard, edit android/app/build.gradle: * Run Proguard to shrink the Java bytecode in release builds.def enableProguardIn ReactReleaseBuilds = trueMigrating old Android Native apps to use App Signing by Google Play#If you aremigrating from previous version of React Native chances are your app does not use App use by Google Signing Feature. We recommend that you enable it to take advantage of things like automatic app sharing. In order to migrating from the old way of signing you need to start by generating new upload key and then replace the release signing config in android/app/build.gradle to use the upload key instead of the release one (see section on adding signing config to gradle). Once that's done, follow the instructions from the Google Play Help site to send your original release key to Google Play. Play.