The Building for Billions Playbook

A Companion to The Secrets to App Success on Google Play

Edition 1.0

Introduction

Today, hundreds of millions of users are coming online for the first time and, for the majority, their first online experience will be on a smartphone. This represents a huge opportunity to grow your Android app or game business.

Most of these users are in emerging markets. Many will be using low cost or second hand phones and their data access may be limited by network coverage or cost. There will also be a wide range of cultural, language, and educational backgrounds you'll need to consider.

Helping you understand what needs to be done to grasp this opportunity is the subject of this guide. It takes you through the design, development, go to market, and in market activities you should consider to best meet the needs of millions of users.

Table of Contents

Useful resources

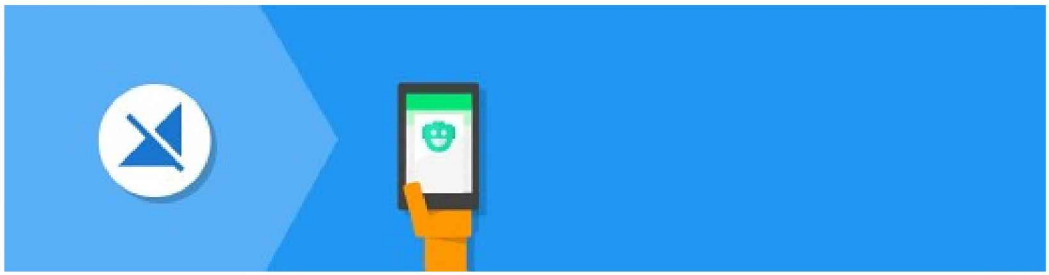
Keep up to date with our developer resources, join our communities, and get support from the Help Center.

Develop your app to account for varying devices and data uses

In this section

Many of the next billion users will be using second hand or low-cost smartphones. These devices are likely to have smaller, lower resolution screens and less memory. Access to power may be limited, so users will need to go longer between battery charges. Users may also have limited access to data, as connections may be slow or intermittent. This section looks at how to code an app to allow for these conditions.

Make sure your app is still usable on slow and intermittent data connections



Manage how your app uses data so it provides essential information first: to maintain good functionality and responsive behavior even over an intermittent 2G network.

Why it works:

Your app's ability to transfer information in a timely manner is dependent on the network connection. Detecting the quality of the network and adjusting the way your app uses it can help your app provide a great user experience.

How to do it:

Use ConnectivityManager functions `isActiveNetworkMetered`, `getActiveNetworkInfo`, and `getNetworkCapabilities` as well as TelephonyManager `getNetworkType` and `getDataState` to detect the quality and availability of the data network. Also monitor ConnectivityManager `CONNECTIVITY_CHANGE` for changes in network status.

Best practices:

- Fetch and display text before images and other rich media.
- Listen for changes in the network availability or quality and adjust downloads accordingly by scaling up or down the number and size of requests. For example, scale down by downloading lower resolution media or exclude media from the download entirely.
- When network quality is high, particularly on unmetered networks, consider prefetching data the user will want, so it can be available if the network quality drops. But remember, many phones will have limited memory too.
- Optimize images and avoid background services where possible.
- Give users control over when and what connection they use, particularly for large downloads or syncing.

Get started:

Managing Network Usage

Make your app useful even when it's offline

When operating in an area with limited or no connectivity, ensure your app is able to offer basic functionality and display its essential content.

Why it works:

In emerging markets network coverage may be patchy and, even when there is coverage, relatively slow. It's therefore common for devices to lose data network connectivity. Building your app so it continues to provide useful features and content in limited data network or intermittent data connection environments will ensure users remain engaged.

Best practices:

- Tell the user that they have lost connectivity only when it matters, such as when a message cannot be sent due to lack of connectivity.
- Create an offline-first architecture using `GcmNetworkManager` (to schedule synchronization tasks) and `ContentProviders` (to manage and access data).
- Use a local database (such as SQLite or SharedPreferences) for long lived data and a bounded disk cache (such as

`DiskLruCache`) for transitory data.

- Use an architecture that separates fetches from the network from the processes that present the user interface.
- Queue outbound updates from the app and send them automatically as soon as network connectivity is restored.
- Cache content or perform updates when network connectivity is good. Base caching on the user's likely demand for content or data in the future.
- Data that doesn't typically change should only be requested once over the network and cached for future use.

Get started:

Managing Network Usage

Help users control the amount and cost of their data use

Provide users with the ability to control the amount of data your app downloads, and the networks it uses for different types of upload and download.

Why it works:

In most emerging markets users will pay for mobile data as they use it. Public Wi-Fi hotspots will often meter data too. Users who are comfortable with your app's data use and associated costs will become more engaged. It's important to give users the ability to control how data is used, particularly to reduce the use of expensive data connections.

Best practices:

- If your app's data consumption could be significant, provide users with an onboarding process that covers data use. With this process, help users find the best balance between data used and the quality and timeliness of content presented by your app.
- Offer users settings to control data syncing, pre-fetching, and network use behavior. Examples of the controls you might include are: quality options for streamed audio or video, options

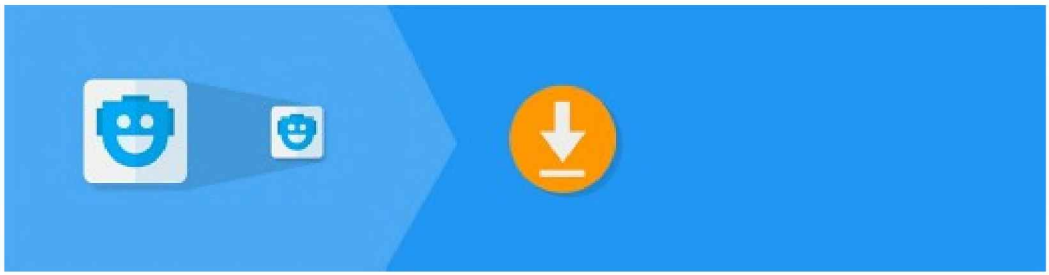
to take lower resolution images or skip them entirely, or defining content that is fetched over Wi-Fi and cached for later access.

- Provide a network preferences screen (by including an Activity that supports action `ACTION_MANAGE_NETWORK_USAGE`) as a way to navigate to the app's network settings from outside the app. It's typically invoked from the settings or data usage screens, and can be used from within the app too.

Get started:

Managing Network Usage

Encourage installs and retention by keeping your APK small



Look to reduce the size of your app's APK file, particularly by reducing the size of bundled graphics and code. Also minimize its storage footprint after installation and allow users to move it to a memory card.

Why it works:

Emerging market users can be reluctant to download apps with large APKs, because of the impact large APKs have on data use and storage space on devices. As users may be using phones with limited storage capacity, apps using too much local storage are likely to be removed.

Best practices:

- Keep your APK size small: under 10 megabytes.
- Optimize the graphic resources bundled with your APK. Use vector images where possible (the Android Support Library provides the support-vector-drawable and animated-vector-drawable libraries to provide backward compatible support for

vector images). For raster images use WebP format rather than PNG in APKs. WebP is supported in Android 4.0 and above. Avoid using large background images.

- If you've many large images across multiple screen densities, consider splitting your APK by density. By targeting builds by specific densities, users with low density devices won't have to incur the penalty of unused large images.
- Use a tool such as ProGuard to reduce the size of your compiled code.
- Use external libraries with care: ensure that they've been optimized for mobile use.
- Enable resource shrinking at build time by setting `minifyEnabled=true` and `shrinkResources=true` in `build.gradle`.
- Selectively include Play Services APIs into your APK.
- Allow your app to be installed to external storage using the `android:installLocation` flag in your AndroidManifest. Where possible, also ensure that any data your app writes is stored to external storage.

Get started:

Optimizing your APK

Use memory efficiently to accommodate devices with limited memory

Look at all the ways in which your app can use memory — such as user data, caches, and processes — and critically examine their use to look for ways of reducing your app's overall memory footprint.

Why it works:

The phones in use in emerging markets may offer users 512MB of memory or less. Apps that take up a significant portion of this memory are likely to be uninstalled, as users look to get the most out of their devices. Minimizing memory use can help improve your retention rates.

Best practices:

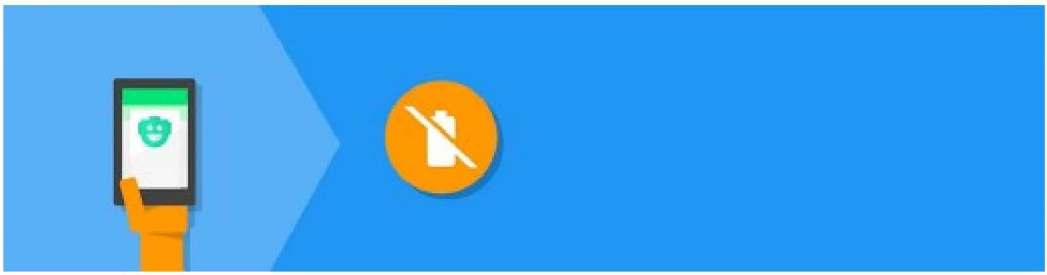
- Use `ActivityManager` methods such as `isLowRamDevice` and `getMemoryClass` to determine the nature of available memory at runtime. Use this information to adapt your app to best use the available memory.
- Use the Android Studio memory benchmarking and profiling tools to measure memory use at runtime.

- Benchmark the memory footprint in each version of your app as this can help catch unintended memory footprint growth.
- Use lower resolution images on low memory devices.
- Release memory when your user interface becomes hidden.
- Release memory as memory becomes tight.
- Use the Memory Monitor Tool to find out whether undesirable garbage collection (GC) event patterns might be causing performance problems.
- Run Heap Viewer to identify object types that get or stay allocated unexpectedly or unnecessarily.
- Use Allocation Tracker to identify inefficient memory use.
- Use services sparingly as they consume memory.

Get started:

Managing your app's memory

Avoid battery draining features so users can go longer between charges



Avoid processes or features that drain battery, particularly those background processes that don't immediately contribute to the user experience.

Why it works:

Low cost phones tend to use lower capacity batteries and their users may not have regular access to power sources they can use to charge their phone. Apps that reduce the time between charges are likely to be uninstalled, so eliminating processes or features that can reduce battery life can help improve your retention rates.

Best practices:

- Send push messages from your server using Google Cloud Messaging (GCM) to trigger periodic or event specific activity in your app. This avoids long running persistent network connections and is efficient as GCM uses a low power, persistent connection to delivers messages.
- Use `GcmNetworkManager` or other techniques and batch

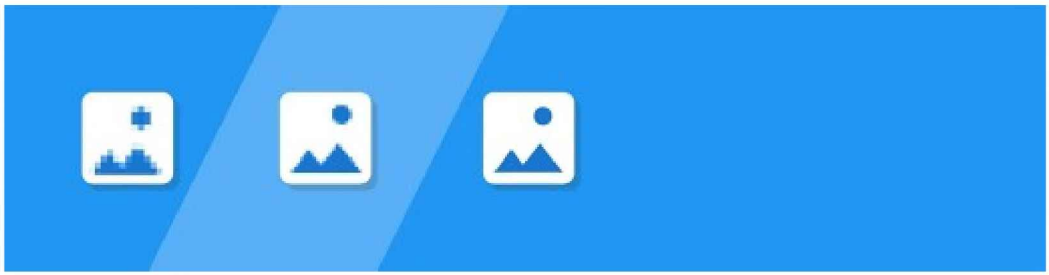
network requests to reduce the number of power hungry radio system activations. With `GcmNetworkManager` you can also schedule background task to start when the device is charging and connected to an unmetered network.

- Don't ask the device to remain awake using `Wakelock` and honor the user's sleep settings.
- Benchmark your app's battery use with Batterystats, using Battery Historian to convert the data collected into an HTML visualization.

Get started:

Optimizing Battery Life

Save data and battery by using images of the right size for a crisp display



Make sure that images sent to devices match the size at which they'll be rendered. Use image compression that balances quality with size.

Why it works:

Oversized images will consume data unnecessarily and take longer to download than correctly sized images. Once on the device, oversized images consume memory, processor, and battery as they are scaled to fit. By sending the right sized image you save your user data traffic charges and deliver a more responsive UI.

Best practices:

- Ensure your app always requests the right sized images from your server, and that your server provides those images.
- Use WebP to deliver images to Android 4.0 and above (lossless WebP is supported from Android 4.2) in the app's APK and online. WebP delivers smaller file sizes than PNG and JPG with at least the same image quality.

- Consider making image size requests based on network type or network quality. For very poor connections, consider not downloading images altogether. Use dynamic placeholders such as pre-computed palette values or low-resolution thumbnails to improve the user experience while images are being fetched.
- Use an image loading library, such as Glide or Picasso, to handle image fetching and caching. They will also appropriately size images and offer other features, such as transitions.

Get started:

WebP

Ensure your app looks good on a variety of screen sizes

Make sure your app works well on a variety of screens: offering crisp graphics and appropriate layouts on low resolution and small screens.

Why it works:

Many devices in use in emerging markets will have smaller, lower resolution screens compared to the common screen sizes and densities. Ensuring your app offers a good UX on smaller screens will increase your potential audience.

Best practices:

- Use density independent pixels (dp) units, not pixel (px) units, when defining app layouts. This ensures that the physical size of your user interface will be consistent regardless of device.
- Follow the material design guidelines on metrics and keylines to ensure you have layouts that can scale across screen densities.
- Ensure that your app layouts work well on small and medium screen sizes and be selective about which UI elements are visible: focus on showing the user the essential information first.

- Ensure that your graphics and text work well on low density (ldpi and mdpi) screens. Provide bitmaps that scale correctly.
- Test your graphics on ldpi and mdpi screen densities and layouts on small and medium screen sizes.

Get started:

Supporting Multiple Screens

Bonus tip:

Devices with lower density or smaller screens tend to have lower hardware specifications. To maximize the performance of your app on these devices consider reducing the use of or removing processor intensive graphics effects, such as animations or transitions.

Give all your users an app that feels fast and responsive

Tune your app's layout and UI to offer all users a similar perception of performance and response.

Why it works:

User perception of acceptable app performance doesn't diminish simply because they may be using a low cost or pre-owned device. By optimizing your UI and using appropriate visual cues and UI feedback, you can provide users with an experience that feels fast and responsive regardless of their device's constraints.

Best practices:

- During app startup provide a placeholder UI (where your app presents a temporary user interface quickly) or a branded launch screen, to reduce the perception of load time compared to a blank canvas.
- Avoid empty states such as lists with no items. Offer a non-interactive image and a text tagline, starter content, nearest matches, or educational content instead.
- Provide responsive interactions on all touchable elements, this

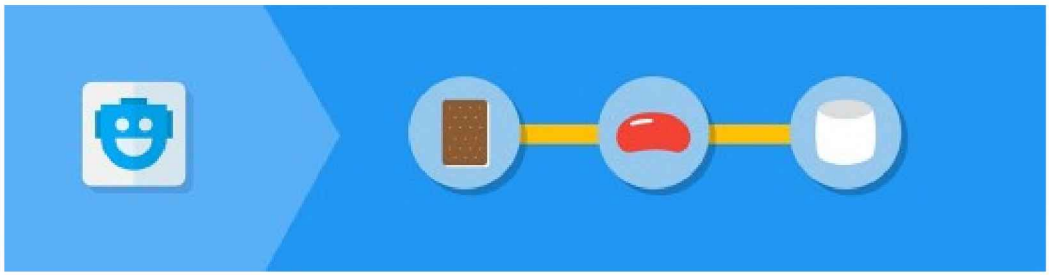
gives users the impression that the app has responded to their request immediately, even if the underlying process takes a moment to complete.

- Keep visual indicators of progress and activity simple, with minimal visual changes. Avoid blocking dialogs with progress indicators: apps that are unresponsive when performing background activity feel slow and reduce user satisfaction.
- Simplify or remove animations on low cost devices to reduce the demands on the device's CPU and GPU.
- Debug GPU overdraw, where pixels are drawn more than once per pass, and look to minimize it to deliver a smooth framerate.
- Make sure your app matches the screen refresh rate of 60 frames per second and profile your app using on-device tools to confirm.

Get started:

Keeping Your App Responsive

Reach a wide audience with an app that runs on popular Android versions



To reach the widest audience in emerging markets consider offering backward compatibility to Jelly Bean and where practical to Ice Cream Sandwich.

Why it works:

The age profile of devices in emerging markets will generally be older and, as a consequence, the versions of Android they are running include a higher percentage of earlier versions. Therefore, apps that can run across a broad range of Android versions are likely to work for a larger audience.

Best practices:

- Use the Google Play services APIs to access the best of Google, largely independent of Android platform version.
- Take advantage of Android Support Library packages to add backward-compatible versions of Android framework APIs to apps you want to run in devices.

- Always target the latest version of Android (by setting `targetSdkVersion`). This ensures that your app inherits the latest runtime behavior.
- Use `minSdkVersion` to set the backwards compatibility level of the app. Use 16 when offering compatibility to Jelly Bean and 14 to go back to Ice Cream Sandwich. Once done the Android build tools will report the incorrect use of new APIs that might not be available in older versions of the platform.

Get started:

Support Library Features

Plan ahead for local users and languages

In this section

The needs and aspirations of the next billion users will share common ground with your existing users, but also encompass unique and novel requirements. Creating an app, or retargeting an existing app, for these users will require a deep understanding of local challenges and opportunities.

01 Design for the widest audience by understanding local users

02 To simplify localization have translation in mind when designing your app

03 Translate your app with the App Translation Service in the Developer Console

04 Test your app in each language to ensure a successful launch

Design for the widest audience by understanding local users

Research new markets thoroughly to ensure you understand the social, cultural, educational, and language factors that affect your potential audience. Also find as much information as you can on the device mix and network coverage and speed found in the market.

Why it works:

By understanding the unique challenges and opportunities faced by users in new markets, as well as their preferences and needs, you'll be able to better determine what changes your app might need to serve those users and help formulate a better go to market plan. This will make you more likely to succeed in entering new markets.

Questions to ask:

- What are the levels of numeracy and literacy?
- What are the local content trends, preferences, and sensibilities: what is the right tone and language?
- Are there any device features that are more common or unique to the market, such as dual and triple SIM phones?

- What is the buying power of local users and how does that affect the prices I can charge for my app, subscriptions, or in-app products?
- Does the market prefer prices that end in .00, .99 or others?
- Do any of my existing users know these markets and would they be willing to offer advice or feedback?

Best practices:

- Focus on creating an interface that doesn't rely on the written word to be usable.
- Use fewer words and minimize non-numerical input.
- Provide graphical cues with audio and voice support.
- Keep the interaction simple: avoid scrolling menus; use tappable, browsable interfaces.
- Use auto-complete and curated lists, avoid searching and filtering.
- Use material design components to provide users with a familiar interaction paradigm.
- Build features by thinking mobile first. For example, don't rely on familiarity with email and web passwords: offer phone number-based user registration.
- Dual or triple SIM phones are common, add appropriate call or messaging features.
- Spending power will be weaker than you might be used to, so a good onboarding flow is vital. Provide app and subscription trials so the user can try before they buy.

Get started:

Material design

To simplify localization have translation in mind when designing your app

Make sure that your app is designed to be easily localized by accommodating the variations you'll find in different languages: allow for spacing, density, order, emphasis, and wording variations. Also make sure that date, time, and similar are internationalized and display according to the phone's settings.

Why it works:

Designing your app with the nuances of localization in mind will save you time and money when you come to expanding into emerging markets. It'll also ensure a positive, mistake-free experience for your users.

Best practices:

- Extract UI strings from your app code and keep them in an external file. Android makes this easy with a resources directory in each Android project.
- Design a single set of flexible layouts. For example, build in 30 percent extra space in UI elements to accommodate other

languages.

- Use alternative layouts for localizations with caution, as they tend to increase maintenance effort — even though Android makes it easy to declare sets of layouts and other resources for specific languages, locales, screen sizes, and more.
- Support Right to Left layouts and text using full native support for features such as layout mirroring in Android 4.2 and later.
- Use system-provided formats for dates, times, numbers, and currencies so your app automatically matches the user's selection.
- Include a full set of default resources, such as layouts, drawables, and strings. Include them in the default resource directories without any language or locale qualifiers.

Get started:

Design for localization

Bonus tip:

Consider whether you'll use a single APK or separate APKs for different markets. Using a single APK is recommended. However, multiple APKs could be useful if you're going to significantly change your app's content to meet the needs of local markets. For emerging markets you might want to create APKs that contain graphics optimized for small screens or a separate lite version that allows for poor data access.

Translate your app with the App Translation Service in the Developer Console

Obtain a quality translation of the text used in your app as well as your Google Play store listing, making sure that text in images is translated too. Consider using the Google Play App Translation Service to access pre-qualified translation resources.

Why it works:

Translating your app and listing text into the native language of any target market significantly improves the download numbers and user engagement with your app. And a good quality translation is important, because a poor translation will often be taken as a sign that the app itself is of poor quality.

How to do it:

- Ensure all your strings are defined in `strings.xml`, remove any redundant strings, and add additional information to help with translation.
- In the Developer Console, select the app you want to translate.

- Find the app translation service at the bottom of the APK section.
- Select your translator and target languages.
- Pay for the service.
- Manage the translation directly with your chosen translator.

Best practices:

- If selecting a translator yourself, make sure that they have experience translating apps and games as literal translation of your app's text strings might not convey the correct meaning. This can be a particular issue when translating games, because the language in games often has subtle but important nuances that may be obvious only to a gamer.
- Remember to include the content of your Play store listing and text used in images as part of your translation request. When using the Google Play App Translation Service add these text strings to the `strings.xml` file sent for translation.

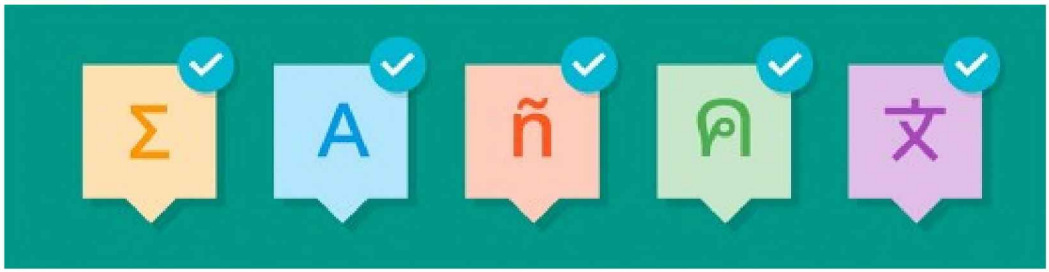
Get started:

Translate UI strings and other resources

Bonus tip:

You can also use the App Translation Service in the Developer Console to get translations of your in-app product details and universal app campaign text.

Test your app in each language to ensure a successful launch



Thoroughly test your app with local users prior to general release and before investing in other activities. In particular, test to ensure your app is capable of achieving good levels of engagement before investing in acquisition activities.

Why it works:

Regardless of the quality of your translation service and how well aligned your app is to your local market research, mistakes and misunderstandings can happen. Testing your app with the largest representative set of users you can manage will help you identify and eliminate issues, issues that could hamper your apps entry into its target market.

How to do it:

- Decide whether you want to run alpha and beta tests, or just beta tests.
- Collect email addresses from your testers (to run a closed test), create a Google+ community or Google Group and invite testers to join (to run a closed test) or choose to run an open test (which you can limit to a specific number of testers to make it

manageable).

- Select your app in the Developer Console.
- Upload your alpha or beta APK to the relevant tab, choose the testing method, add a feedback email address, and publish your APK on Google Play.
- Share the opt-in URL with your testers.
- Review their feedback, update the app, and iterate until you're ready to release your app.

Best practices:

- Run tests with local users or native-language speakers.
- Look out for clipped text, overlapping text, and poor line wrapping.
- Check for incorrect word breaks or punctuation.
- Validate alphabetical sorting to ensure the order is as expected.
- Make sure all layouts and text directions are correct.
- Watch for untranslated text; check for the resources directory being marked with an incorrect language qualifier.
- Test for default resources.

Get started:

Set up alpha/beta tests

Bonus tip:

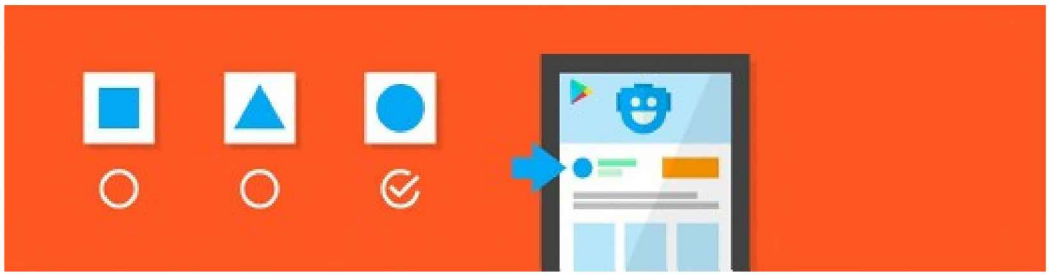
Use Cloud Test Lab to test your app on popular, physical Android devices, across multiple languages, screen orientations, and Android API versions.

Grow and engage audiences in new markets

In this section

Finding an audience among the next billion users starts with a compelling Play store listing. After that you'll need to make careful and creative use of the tools available to find the right connection with your specific audience, starting with their feedback. This section looks at these and the range of engagement, promotion, and monetization options that can work well, as well as the analytics tools and reports you'll use to measure success.

Create a compelling Play Store listing and optimize it with experiments



Don't simply translate your app's Google Play store listing — make it compelling for local users with the right app icon, description, screenshots, graphics, and video. Use Store Listing Experiments to find the best combination of text and images for local users that maximizes your installs.

Why it works:

Unless users know exactly what they're looking for — an app by a specific developer, for example — they tend to make their download decision based first on the app icon, then other aspects of the store listing such as the screenshots, description, and video. Using Store Listing Experiments takes the guesswork out of finding the best Play Store page content for your app.

Best practices:

- Familiarize yourself with the Google Play Policy guidelines & practices to know what you can and cannot include in your listing, particularly the policies on spam.

- Create a unique app icon that will appeal to local users. Consider working with a professional graphic designer or use the Android Asset Studio tools.
- Make sure screenshots show off localized versions of the best and most important features of your app. Use a consistent and clean status bar across all screenshots.
- Give readers a clear impression of your app's UI by adding at least one unaltered screenshot for smartphone, 7-inch tablet, and 10-inch tablet versions. Include portrait and landscape mode shots too.
- Make sure your featured image sums up the unique features of your app in a way that local users can relate to.
- Keep your app's description crisp and to the point; make sure the first sentence sums up the key and unique features of the app. Avoid overuse of keywords, focus on readability and fast comprehension: grab the user's attention, and keep it.
- Include a variety of words in your app description that represent the core features of your app, so users can find it when searching.
- Include a short video (under 2 minutes) that explains the user flow. A simple screen recording can be all you need. Use a voice over by a native speaker or use suitably translated subtitles.
- Remember to localize the text in all your screenshots and videos.

How to do it:

- In the Developer Console, click All Applications and select an app.
- On the left menu open **Store Listing** then **Experiments**.
- Give your experiment a name, set the percentage of readers who

should see the experiment and set up the content variants you want to test — you can test up to three variants of your page's graphics and text, either as a global test of graphics (app icon, feature graphic, screenshots, and video) or text and graphics for any localization of your store listing. You can even experiment with the order of your screenshots.

- Start your experiment and keep an eye on the banner to follow progress.
- Once the test has completed, open **Results** and choose to update your store listing with the content of the winning experiment.

Get started:

Create a Great Listing

Bonus tip:

With Google Tag Manager and Google Analytics, you can run A/B tests on in-app elements without the need to update your app.

Gather feedback and iterate on your app with beta tests and reviews

Use beta testing as well as ratings and reviews as opportunities to listen to your users and gather feedback. Use their feedback to regularly update your app: resolve issues and add new features.

Why it works:

Apps that are updated and improved regularly tend to maintain user engagement. User feedback is often a valuable source of suggestions for improvements or details of issues that you might not have found. Staged rollouts can help you to identify crashes and ANRs before you roll out an update to your larger user base. Responding to feedback, particularly app reviews in Google Play, engages your users and makes them feel valued, helping build loyalty. And don't ignore negative reviews or feedback, reply indicating how and when you'll address the feedback. Users are more likely to leave a good rating and share your app as the result of a positive interaction.

How to do it:

- To set up alpha or beta tests in the Developer Console click on **All Applications** and then select the application you'd like to test.

Upload the alpha or beta version of the app's APK and publish it on Google Play. Inform your testers that the app is available on Google Play and provide them with a feedback channel.

- To read your app's reviews, in the Developer Console, click on **All Applications**, then select the application you'd like to view, and choose **Ratings & Reviews**. To reply to a review, click **Reply to this review**. The user is sent an email when you reply, including an option to update their review or contact you by email. You can edit your reply later too if, for example, the user updates their review or rating.

- To use staged rollouts in the Developer Console click on **All Applications** and then select the application you'd like to rollout. Choose the percentage of users you want to receive your app in the first rollout, then save and publish your updated app. Monitor crash reports and user feedback, correct any issues and publish a new version of the app if necessary. Repeat with more users until you're happy the app can be released to all your users.

Best practices:

- To understand your app's reviews better, you can apply filters to see them by rating, written language, app version, and/or device. You can also export your reviews in bulk to conduct your own sentiment analysis.

- Run an open test when you want to encourage feedback from the broadest community of readers. Alternatively use closed testing from an email list where you want to test with minimum visibility and already have a group of known testers in mind. Balance between these options by creating a testing community in Google+ or as a Google Group. Here you can control who'll be testing, but also have the opportunity to recruit new testers and encourage discussion among your testers.

- For staged rollouts start with a relatively small percentage of users, perhaps 10 to 20 percent, for the first rollout and give

them 12 to 24 hours to use the app. If this goes well try a larger group, perhaps 20 to 40 percent, for 6 to 12 hours. Then try a final step of 60 to 80 percent for a few hours or go straight to a full release.

Get started:

Set up alpha/beta tests

Bonus tip:

You can see breakdowns of ratings and reviews by dimensions such as device, country, Android version, and so on. Analyze your ratings and reviews to gather valuable feedback about how to improve your app for different groups of users.

Create an engaging app that keeps users coming back

Make use of the Android and Google features that can support user engagement with your app.

Why it works:

Use Android and Google features to gain access to tried and tested tools and technologies that have been proven to enhance user engagement with apps.

Engagement tools:

- Use Android intents to make the features of your app available when users want to complete specific actions.
- Extend your app's interface with rich notifications. Deliver your users relevant and timely details from your app when they aren't actively using or seeking information from it.
- Use targeted topic messaging delivered from Google Cloud Messaging to communicate with specific segments of your audience and maintain or refresh their interest in your app.
- Set up App Indexing so that users can return to your app when it

appears in their search results.

- Allow users to login with Google sign-in, eliminating the need to enter their details and remember a new password. You can then give users options to use their Google account features, such as saving content to Drive or adding an event to their calendar.
- Where you already have your own user accounts use Smart Lock for Passwords to save user credentials and automatically sign-in users from all the devices they own.
- Take advantage of Google Play game services to add features such as leaderboards, multiplayer, quests, and more to your games.
- Use the Nearby API to add support for novel interactions between users or enable local multiplayer games.

Get started:

Engage & Retain Users

Bonus tip:

Focus on creating an engaging app before you consider investing in user acquisition efforts. Use alpha and beta testing, trials, or limited releases to help assess and refine engagement. With a highly engaging app you'll achieve a better return on your investment in acquisition.

Promote your app, drive installs, and optimize your marketing

Find the right mix of acquisition tools to grow your local audience.

Why it works:

Finding new users among the next billion will present similar challenges to finding users in any market. The promotion and marketing tools from Android and Google are tried and trusted means of connecting with new users. They allow you to focus on finding the right mix and messaging, instead of forcing you to code your own solutions.

Promotion tools:

- Create universal app campaigns to reach users from the Google Search Network, YouTube and Google Display Network.
- Set up App Indexing to ensure that your app is surfaced when users search for information it contains. Once surfaced the user can install your app directly from their search results.
- Consult the User Acquisition page in the Developer Console to

find out how users are finding and installing your app, then use this information to refine your acquisition strategy.

- Offer App Invites so that users can easily share your app with family, friends, and colleagues.
- Nurture a community of users on social networks to spread the word about your app.
- Create an AdWords re-engagement campaign to bring back users who have your app installed.
- Offer an app install banner on your mobile website so users can install your app directly, without searching for it on the Play store.
- When users discover your app videos on YouTube, bring them to your Play Store listing with a merchandise card.

Get started:

Get Users

Mix the right business models and earn more revenue

Find the right mix of monetization options from paid apps, ads, subscriptions, and in-app products.

Best practices:

- Consider releasing a free version of your app to reach a wider audience.
- With Google Play In-App Billing, you can offer paid apps, in-app purchases, or subscriptions.
- With subscriptions, users can opt in to weekly, monthly, quarterly, 6-monthly, annual, and seasonal subscriptions. You can also offer renewal, free trials, upgrading and downgrading, and many other choices to your users.
- Localize your prices and experiment — a simple exchange rate conversion may not be appropriate. Also consider preferences for ending prices in .00, .99, or others.
- Provide a good onboarding flow, which demonstrates value to the user before requiring them to make a purchase.
- When planning promotions understand local buying cycles: link

promotional activity with local holidays and events.

- Use AdMob to display ads from millions of advertisers in your app.
- Try combining AdMob ads and in-app purchases – this combination has been found to be a very effective way of optimizing app revenue.

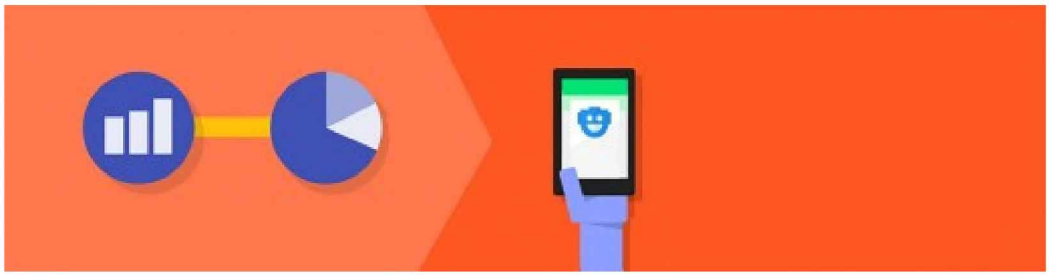
Get started:

Earn

Bonus tip:

Take advantage of the lowest minimum prices Google Play offers in different markets to make your price points attractive to local users.

Measure what matters and analyze results to keep improving your app



Use the tools and reports offered by the Google Play Developer Console, Google Play game services player analytics, and Google Analytics to gain insights into your audience. Use these insights to improve your app and how you promote it.

Why it works:

The analytics reports available from Google Play and Google Analytics help you understand how people discover your app, how effective your marketing campaigns are in driving discovery, and how people navigate through different parts of your app. These reports help you understand the performance of your acquisition channels so you can fine tune them to deliver the best return on investment. You can also dig into how users are interacting within your app to find and remove roadblocks and optimize conversions.

Reports and insights:

- In the Google Play Developer Console use the Reports section to get details of Installs and uninstalls, upgrades, ratings, crashes, and “Application Not Responding” (ANR) errors. There are also

Financial reports that show where your app revenue comes from.

- From the Developer Console, you can also access the Google Play game services statistics, where you'll find insights on your players, their engagement with your games, and your financials.
- Assess the general health of your app and follow data trends with the Google Analytics App Overview. It provides a summary of the most relevant data from all the Mobile App Analytics reports.
- The Google Analytics' Audience Reporting section has a wealth of data about your users' characteristics: what app versions they're using, what devices they're on, where they're from, and what they're interested in. Among these, the Active Users report highlights how users come back over time.
- The Behavior Flow report in Google Analytics shows how users interact with your app allowing you to: identify roadblocks that are preventing users from progressing through your app, check that users are following conversion flows, and more.

Get started:

Analyze

Section 4

Additional resources:

Find success on Google Play
developers.android.com/distribute

Get news and tips in your inbox
g.co/play/developernews

Android Developers Blog
android-developers.blogspot.com

+AndroidDevelopers on Google+
google.com/+androiddevelopers

Android Developers on YouTube
youtube.com/androiddevelopers

@AndroidDev on Twitter
twitter.com/androiddev

Google Play Developer Help Center
g.co/play/developerhelpcenter

Please give us your feedback!

Google Play is committed to helping app and game developers find success with the next billion users. If you've incorporated any best practices, based on the learning from this guide, into your app please let us know at the link below. And, please give us your feedback on the content of this guide, so we can continue to improve our products and services to help you grow your app or game business on Google Play:

goo.gl/pnLuVw