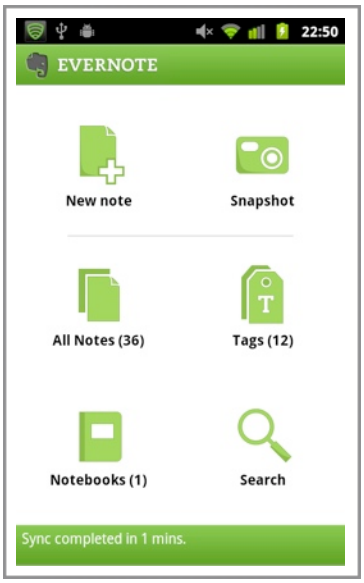
Android best practices in 30 minutes

UI patterns

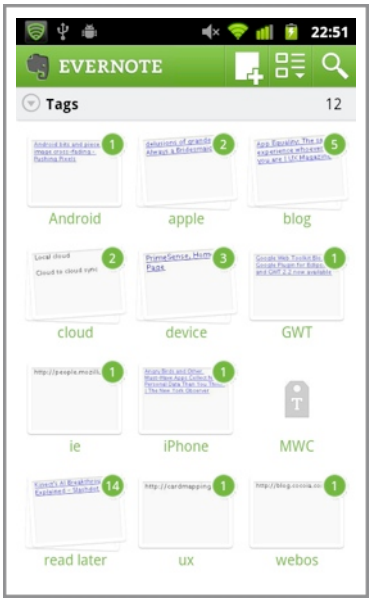
There are few established UI design patterns that can make an app instantly look Androidish.



Dashboard

Landing page of the application should **visually clear** and provide **easy access to main tasks** user want to do with the application.

Apache licensed layout class for implementing dashboard
https://gist.github.com/882650



Action bar

Action bar is a branded top bar of the application that provides easy access to relevant actions on the screen and a shortcut to application's home screen. The action bar can also be used to indicate use's location in the application.

ActionBarSherlock can be used to implement action bar that supports Honeycomb and pre-Honeycomb
https://github.com/JakeWharton/ActionBarSherlock
(Apache license)

Quick actions

Quick action menu provides a unified way to present actions that can be performed on an object. It consists of three components.
1)**Click target** that user can identify as the object that he or she can interact with. Taping this target will open the quick menu pop-up menu. A very good example of this is the contact icon in various default Android apps.

2) **Pop-up action container** that also indicates which screen object the presented actions are related to. This container should not cover the screen object in question.

3) **Action icons** representing all available actions. Only actions that make sense on the current object should be presented. If all actions don't fit the screen he action bar can be scrolled by dragging.

GreenDroid project can be used to implement quick actions:
https://github.com/cyrilmottier/GreenDroid
(Apache license)

Few examples of Android share intent selection

Activity stack handling

To make apps work consistently and the way users expect it is very important to design activity stack management from the very beginning.

Following two pictures demonstrate the importance of activity stack design. There are many ways of solving the problem and this solution is not intended to be the only / best one.

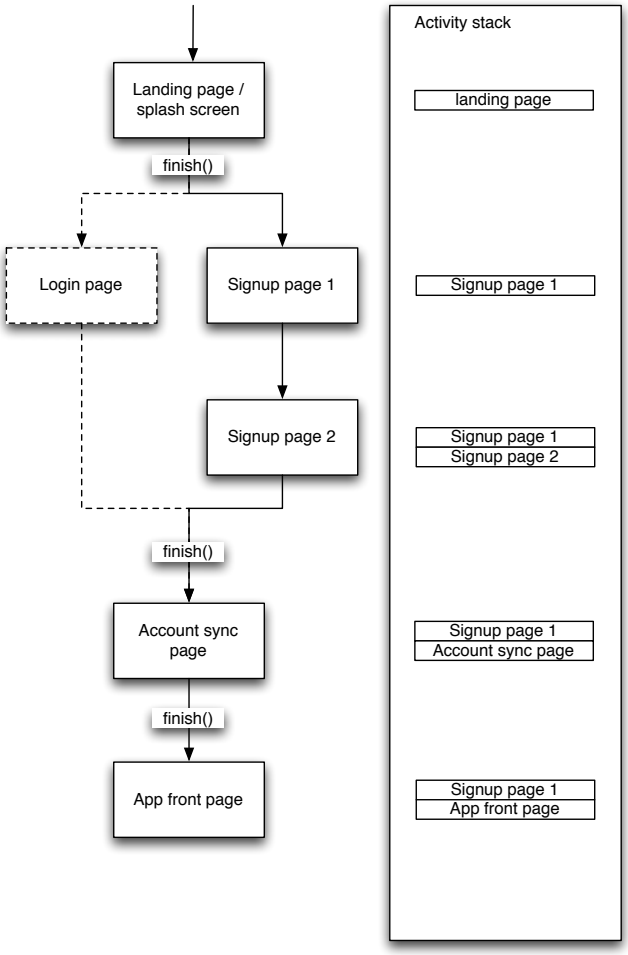


Figure: example of activity stack problem. Pressing back in app's front page will not exit the app but show signup page 1.

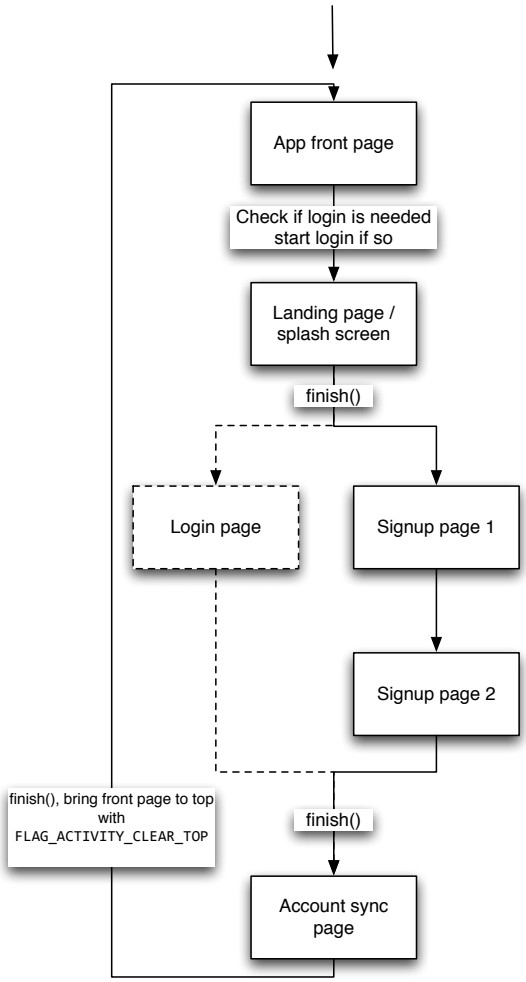
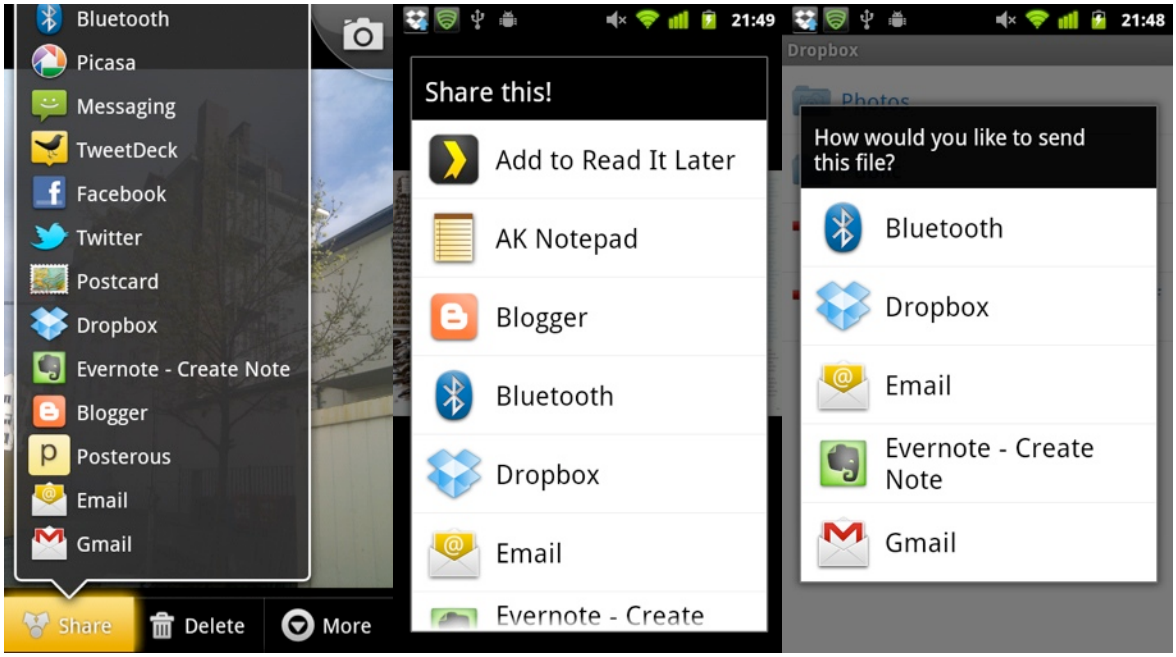


Figure: example of activity stack design that solves the issue.

Android intents

Android intents API enable apps to extend and benefit from other apps' functionality. It is an open ended integration to all other applications without having to know anything about application functionality.

Instead of implementing twitter, Facebook etc. sharing an app can simply declare that it wants to share text, image or URL (among others). The Android framework will then figure out which of the installed apps can handle that type of sharing request. It is possible to get the list of available applications from code and decide to do something with that information or simply let the Android system to present the list to user and let user to pic one.



Paid vs. lite version

Google's Android market doesn't support demo versions of apps. Refund period is only 15 minutes. This means that Android users can be resistant to buying apps especially if they're expensive (more than $5). Developers generally provide an ad supported, feature or time limited lite version of their app to let users try the app first. Because there's not support for for this it has to be done by uploading another app to the market.

True multi-tasking & memory constrains

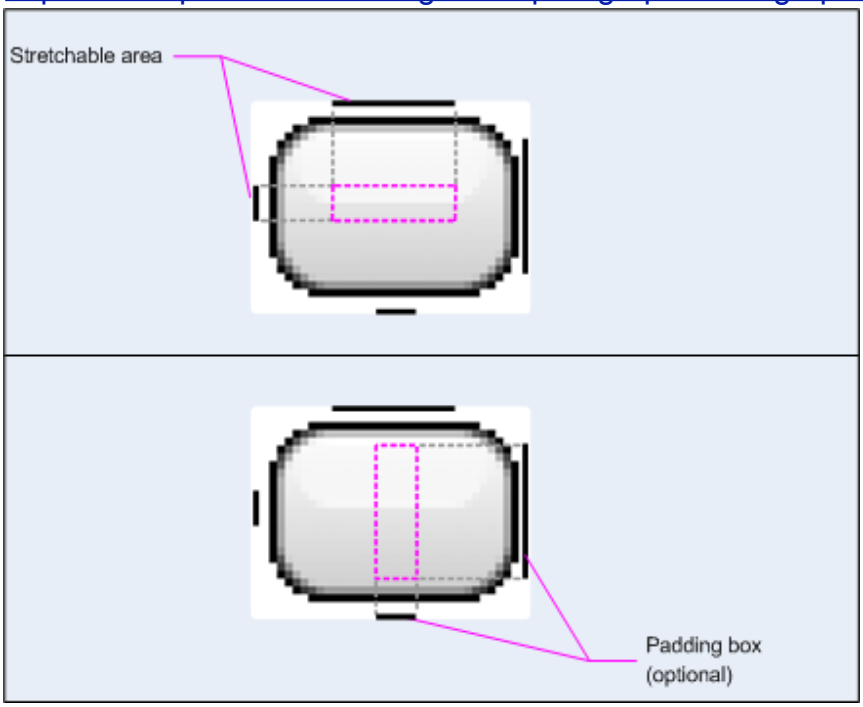
Android is a true multitasking environment. As most handsets are memory constrained it means that developers should be careful with their memory consumption and background functionality. Android OS can kill an app that is on background any time to free memory for more urgent apps and tasks. This has to be taken into account when designing the app. Basically, the app has to save its state when it is killed by the OS and restore it on relaunch.

Visuals

Android devices come in many sizes and forms. The APIs provide good tools to support all these devices. Here are few tips about the most important ones.

9-patch images

9-patch / 9-slice images are very useful.
See more info here:
http://developer.android.com/guide/topics/graphics/2d-graphics.html#nine-patch



Screen density

Different screens have different pixel sizes. From this follows that same image on one screen can appear to be different physical size than on another. On touch based UIs it is important that the buttons etc interaction points are same physical size or users won't be able to manipulate them as intended.

This is a fairly large issue and I'm not going to try to explain it here in depth. Basic solution to this problem is that we need to create more than one set of graphical assets. The assets are placed into corresponding folders in the project folder structure where Android OS will automatically select the correct ones to use when the app is run on a device.

See more:
http://developer.android.com/guide/practices/screens_support.html

Density independent pixels & scaled pixels

When creating Android layout definitions we need to take into account the different screen densities (see above). All measurement definitions in layouts can always be given in DPs (density independent pixels). Android OS automatically converts them to pixels when app

is run on devices. By using DPs we can be sure that our layouts end up being same physical size on different devices.

Scaled pixels are used when defining font sizes. A scaled pixel works similarly than DPs but they are also scaled by user's font preferences. Ie. if user has set a larger font to his / her device SPs are automatically scaled up.

Android icons

Android comes with set of default icons that are available to all apps. They're already available in different densities. Using them can save a lot of time.

http://since2006.com/android/2.1-drawables.php

Quick notes

Back button

All Android phones must have a back button and users use it. Therefore Android app UI should not have soft back button.

Don't hide status bar

Android status bar is the access point to events and long running background tasks. It is technically possible to hide it from an app but doing so will cause the app to much less useful in real phone use.

Use notifications for long running background tasks

When an app does something in the background (downloads something etc.) always use a notification bar icon to indicate it and to provide access corresponding application screen so user can cancel the task if needed.

Android versions

Project should always target lowest possible Android versions that supports all required features to maximize the user base. Android 2.1+ is a very safe bet as more than 90% of devices are running 2.1 or better. 2.1 also has very mature APIs.

If we, however, need newer features there are tricks like lazy class loading to support newer features without breaking support for older Android versions.