



الجامعة التكنولوجية
قسم علوم الحاسوب

Mobile application design

3^{ed} Class, SW branch, Second Semester.
(2025-2024)

By: Teaba Wala aldeen Khairi.

University of technology, Computer science department.

E-Mail: teaba.w.khairi@uotechnology.edu.iq

Requirements for the course

- Google doc
- Google slide & using Google meet

To pass the course the student must do:

- 1 Home works
- 2 Reports.....
- 3 Group project.....
- 5 Final exam.....
- 6 Mid exam.....

Mobile application design / 3rd Class – second course

INTRODUCTION

- Introduction to mobile applications
- Embedded systems
- Market and business drivers for mobile applications
- Publishing and delivery of mobile applications
- Requirements gathering and validation for mobile applications

BASIC DESIGN

- Basics of embedded systems design
- Embedded OS
- Design constraints for mobile applications, both hardware and software related
- Architecting mobile applications
- User interfaces for mobile applications
 - touch events and gestures
- Achieving quality constraints
- performance, usability, security, availability and modifiability.

ADVANCED DESIGN

- Designing applications with multimedia and web access capabilities
- Integration with GPS and social media networking applications
- Accessing applications hosted in a cloud computing environment
- Design patterns for mobile applications.

TECHNOLOGY I - ANDROID

- Establishing the development environment
- Android architecture
- Activities and views
- Interacting with UI
- Persisting data using SQLite
- Packaging and deployment
- Interaction with server side applications
- Using Google Maps, GPS and Wifi
- Integration with social media applications.

TECHNOLOGY II - IOS

- Introduction to Objective C
- iOS features
- UI implementation
- Touch frameworks
- Data persistence using Core Data and SQLite
- Location aware applications using Core Location and Map Kit
- Integrating calendar and address book with social media application
- Using Wifi
- iPhone marketplace.

References

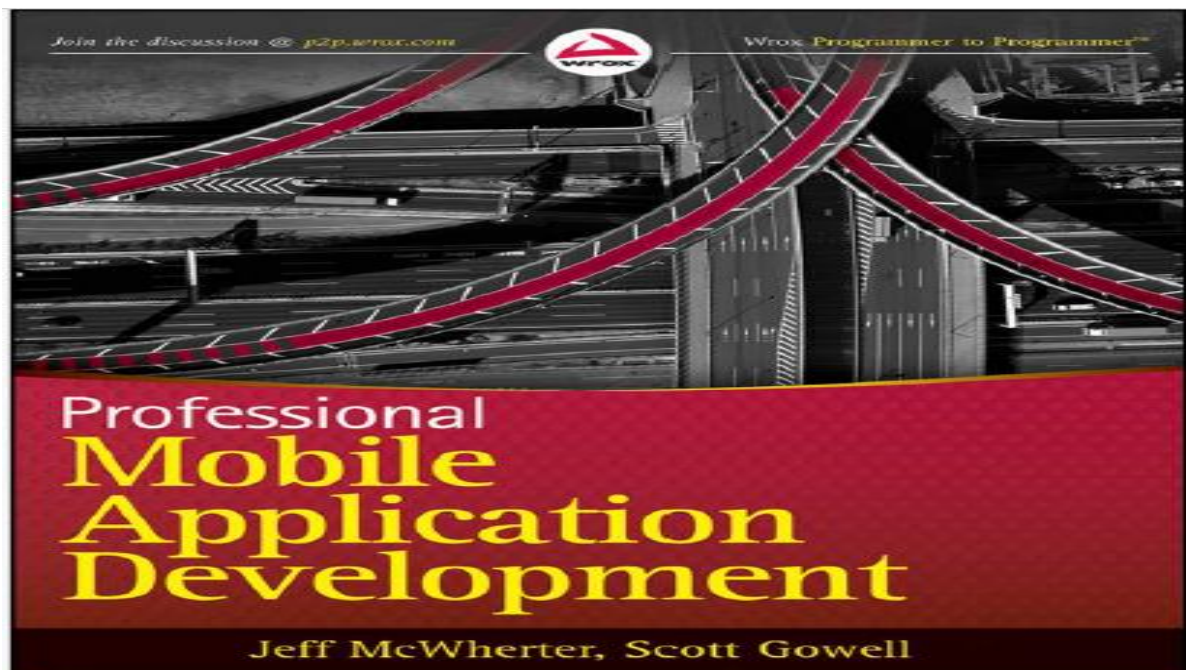
1. Jeff McWherter and Scott Gowell, "Professional Mobile Application Development", Wrox, 2012
2. Charlie Collins, Michael Galpin and Matthias Kappler, "Android in Practice", DreamTech, 2012
3. James Dovey and Ash Furrow, "Beginning Objective C", Apress, 2012
4. David Mark, Jack Nutting, Jeff LaMarche and Frederic Olsson, "Beginning iOS 6 Development: Exploring the iOS SDK", Apress, 2013



Yotube channel

<https://www.youtube.com/channel/UC8oLRPZj2gL0wRIO1UW>

[ZZMw](#)



Lecture 1

Introduction to Mobile Technology

1. What does make mobile Technology very important?

Mobile technology is receiving significant attention in the business and IT worlds. The technology represents a dramatic change in technological capacity that has enabled potential economic advantage for those able to take advantage of it. Mobile technology is the basis of innovations in reaching customers, and in redesigning business processes and software products that lead to the creation of many small businesses.

Mobile application development is the process to making software for smartphones and digital assistants, most commonly for Android and iOS. The software can be preinstalled on the device, downloaded from a mobile app store or accessed through a mobile web browser. The programming and markup languages used for this kind of software development include Java, Swift, C# and HTML5.

Embedded systems, also known as embedded computers, are small-form-factor computers that power specific tasks. They may function as standalone devices or as part of larger systems, hence the term "embedded," and are often used in applications with size, weight, power, and cost constraints.

1.1. The widespread of the use of Mobile Devices

- 1) Mobile devices add a host of new possibilities for business and personal software because they are truly the first mobile computing platforms.
- 2) The capability to be made aware of its current environment through built-in sensors. Mobile devices have sensors designed to capture where they are, here they're going, and the environment around them.

- 3) Sensors can identify their present location to within a few meters and capture their current heading, orientation, and acceleration. Additionally, they can recognize how close they are to another object through a proximity sensor. These devices also have the capability to capture information about the ambient environment, including light levels, temperature, pressure, and magnetic field.
- 4) The capability to communicate with other computing devices through a variety of mechanisms. A laptop can communicate using Wi-Fi and Bluetooth. However, mobile devices also have these communication capabilities; they can communicate via cellular signals and using Near Field Communication (NFC).
- 5) Mobile devices have most of the same features, such as being able to display and manipulate data. Some of these features have enhanced usability because they are on a device that is easily moved.
- 6) Mobile devices are also computers.

1.2. Advantages of Mobile Applications for Organizations to Reach Customers

- 1) Smartphone users almost always have their device within reach.
- 2) Having a mobile app can also support brand loyalty and awareness.
- 3) One area where mobile devices enable a strong potential for disruption of the assumptions made about a business process is the payment industry, where a lot of companies are innovating to provide consumers and businesses the capability to make and receive payments.
- 4) The final, and potentially most important, advantage of an app is that it can take full advantage of the device's hardware and software capabilities to provide the customer with capabilities that make your products an easy option for them.

1.3. Changing Business Process

Businesses are paying significant attention to mobile because these qualities suggest that the technology may have implications for strategic and tactical

advantage, or, as demonstrated with the banking app, become competitive necessities. Information technology was applied to portions of the existing process to make it faster or increase accuracy. As businesses became more adept with the technology and the technology became more capable, it was recognized that the full potential of the technology was not being realized, and companies began rethinking entire processes to take advantage of the technology. Mobile technology is likely to follow a similar path in application to business processes.

Ways You can use mobile apps for your business

To achieve success while investing in a mobile app for your business, you have to pay close attention to your approach.

- Depending on your branding needs, there are multiple ways you can choose to go mobile.
- Though the primary goal of your app is to interact with your customer base, the goal of the interaction often differs from one company to another.



1.4. Making Money and marketing

A final reason that mobile is important is that many people see the potential to start businesses and make money. The Google Play Store and the Apple App Store provide the app developer access to the market of app purchasers.

The developer does not have to worry about product distribution, returns, or payment collection. The store does all this and conveniently deposits the proceeds into the developer's bank account. Additionally, smartphone users automatically go to these stores to get new apps or browse for apps that might interest them. One final and very big reason for the strong focus on app development is that Google and Apple either support or provide the development environments needed to create apps for their stores. Taken together, this creates significant potential for individuals or small businesses to make money in the app market. The mobile apps market covers the following areas:

1. Mobile apps market sizing.
2. Mobile apps market forecast.
3. Mobile apps market industry analysis.

Apps make money for their producer through several approaches. Apps can be sold for a onetime fee, like other products. Consumers buy the app through the appropriate store and it is theirs for use whenever they like. The more apps the developers sell, the more money they make. Ad supported apps make money by including an advertisement on a small portion of the screen. Anytime a user clicks an ad, the developer makes money. Both Google and Apple provide developers access to the code to display ads and a service to provide the ads and track the clicks. In contrast to a paid app, the only time the developer gets paid is if an ad is clicked (Apple's ad service also pays per view of the ad, but the amount is significantly less than a click). The amount of money generated by a single click is very small, so to make much money it is important to get a lot of users of the app. A third approach to making money is to provide for in-app purchases. With this model, the user gets the app for free but needs to make a purchase to get additional features. For example, a developer might provide a game for free but require a purchase for more advanced

levels of the game. Another approach is subscription based. The app provides functionality that requires access to the developer's data or other services. To use the service, users buy a monthly or annual subscription



The benefits of mobile apps

- 1. Build a stronger brand Creating a mobile app can help your business build a stronger brand. ...
- 2. Connect with customers better Mobile apps allow you to connect with customers while they're on the go. ...
- 3. Provide value to customers ...
- 4. Improve customer engagement ...
- 5. Personalize a marketing channel ...

WHAT IS FLUTTER?

Flutter allows you to build beautiful native apps on iOS and Android from a single codebase

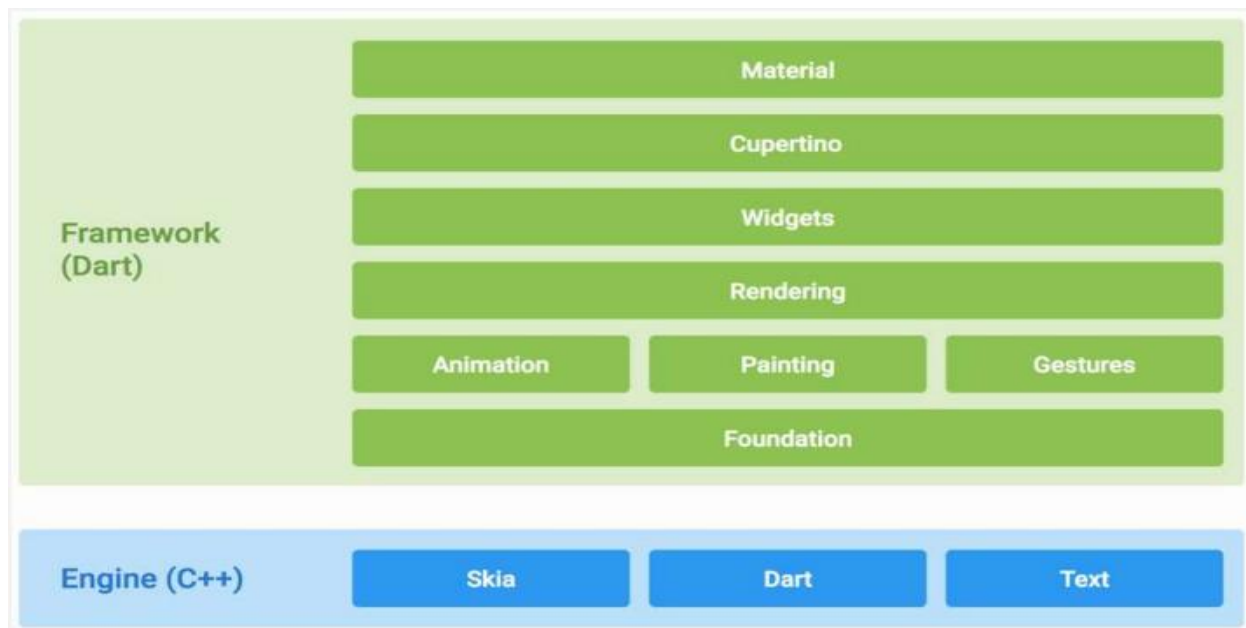
- Open-source mobile app SDK
- Developed by Google

- Building high-performance apps for iOS and Android, from a single codebase

WHY USE FLUTTER?

Flutter makes it easy and fast to build beautiful mobile apps.

- Reactive framework
- Material and Cupertino widgets
- Hot reload
- Dart language and core libs
- Interop with mobile SDKs
- Android Studio/IntelliJ official IDE
- Debugger, Format



DART LANGUAGE

Productive. Syntax must be clear and concise, tooling simple.

- Fast. Runtime performance and startup must be great and predictable even on small mobile devices.
- Portable. Client developers have to think about three platforms today: iOS, Android, and Web. The language needs to work well on all of them.

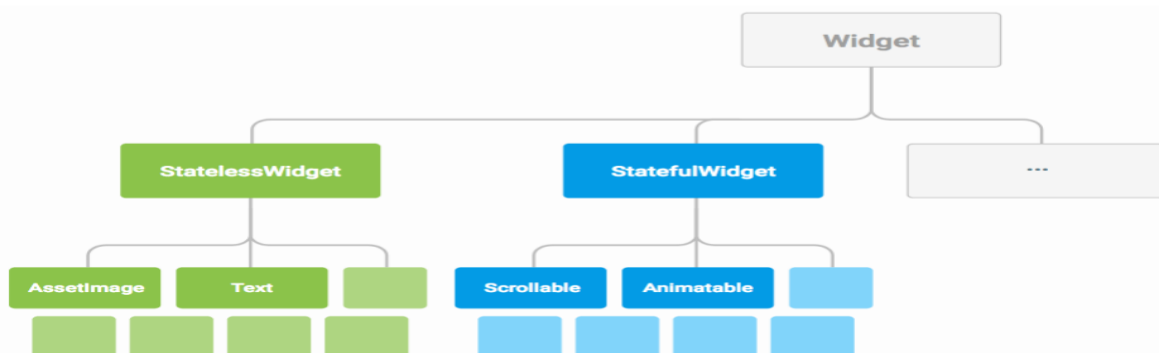
- Approachable. The language can't stray too far from the familiar if it wishes to be relevant for millions of developers.
- Reactive. A reactive style of programming should be supported by the language.



Widgets are the basic building blocks. Each widget is an immutable declaration of part of the user interface. Unlike other frameworks that separate views, view controllers, layouts, and other properties, Flutter has a consistent, unified object model: the widget.

A widget can define:

- a structural element (like a button or menu)
- a stylistic element (like a font or color scheme)
- an aspect of layout (like padding)



Example

```
main() {  
  runApp( new MaterialApp  
    ( title: 'Flutter Demo', theme: new ThemeData  
      ( primarySwatch: Colors.green, ),  
    home: new Scaffold( appBar: new AppBar  
      ( title: new Text("Flutter Demo"),  
    ),  
    body: new Text("Hello World!"), ), ), )
```

Stateless Widget

```
new Text( 'Hello! How are you?', textAlign: TextAlign.center, overflow: TextOverflow.ellipsis, style:  
new TextStyle(fontWeight: FontWeight.bold), )
```

In short...

Variables & constants:

```
var name = 'Voyager I';  
var year = 1977;  
final bar = const [];  
const baz = const [];
```

Getters and setters

```
class Spacecraft {  
  DateTime launchDate ;  
  int get launchYear => launchDate?.year ;  
}
```

Functions:

```
int fibonacci(int n) {  
  if (n == 0 || n == 1) return n;  
  return fibonacci(n - 1) + fibonacci(n - 2);  
}
```

Lambda (fat-arrow) expressions:

```
flybyObjects.where((name) => name.contains('flower')).forEach(print);
```

Chapter 2

App Design Issues and Considerations

2.1 Intro

App development for mobile devices is, in many ways, similar to development for other platforms. However, in other ways, development requires attention to items that are not even present in traditional development.

2.2 App Design

Designing for the specific device your app will run on is extremely important! Applications that work well on a traditional computer may be complete disasters if ported to a mobile platform without redesigning the logic to fit the device's capabilities. The capabilities of the device enable you to design an application that can do different things than an application on a traditional computer. Apps are cheap and easy to obtain. A well-designed app can be a delight to use. A poorly designed app will not be used for long, if at all. The operating system, device size, and mobility all impact design and must be accounted for.

2.3 Operating System Design Issues

The primary technical difference between mobile device operating systems and operating systems used on laptop and desktop computers is that the mobile operating system is not a true multitasking system. On mobile devices, only one **app can be active at a time.**

When another app is started, or the app is interrupted by another app (for example, a phone call),

- the app that was running gets put in the background.
- It remains in the background until the user specifically accesses it again.
- If it remains in the background too long, or if available memory gets too low, the operating system may kill it.

This back-and-forth between different states is called the app's *life cycle*. Both Android and iOS apps have a life cycle. The life cycle is based on the user's

interaction with the app and the operating system's need for memory and processing resources.

As users interact with the device, they may switch between apps or different views within a single app. When this happens the app goes through different states, requiring the developer to handle this switch so that users don't lose data or get unnecessarily interrupted in the task they were performing.

This makes understanding, and designing for, the app life cycle extremely important to the successful app developer.

Mobile App. Development Challenges :

- Competitive, fluid vendor landscape (Apple, Android consortium incl. Amazon, RIM, HP) means apps need to be multi-platform for wide adoption
- No “standard” device (iOS, Windows Phone devices?)
- Low bandwidth input (in most cases; what about tablets?)
- Limited screen size (tablets?)
- Unreliability in connectivity and device (network access, power, ambient light, noise, at least for now)
- Integration tradeoffs with cloud and enterprise services

Application Development Support :

- Object-Oriented Languages
 - iOS: Swift
 - Android: Java/Kotlin
- Cross-platform frameworks:
 - Flutter, Xamarin, Titanium, PhoneGap, ...
 - Scripting languages (JavaScript, Ruby)
- C and C++ (native code)
- Integrated into mobile app dev. Frameworks

Framework Capabilities, Add-Ons

1- Built-in Services:

- GUI
- OS services (file I/O, threads, device mgmt.)

- Graphics
- Device access (GPS, camera, media players, sensors)
- Networking
- Standard language libraries (Java, Kotlin)

2- Add-ons:

- Google Play services (e.g. Google Maps, etc.)
- Database support (SQLite)
- Chromium WebView

2.4 Android Life Cycle

To understand the Android life cycle it is useful to first understand the states that an Android app user experiences.

When users touch an app's icon, the app is started and becomes visible to the users.

While the app is visible, the users can interact with it. This is considered the Resumed or running state.

As the users interact with the app, they may be interrupted with a pop-up window, or they may be distracted and not touch the screen for a period of time. If the users stop interacting for a period of time, the app will fade but still be partially visible. In either of these two cases the app enters the Paused state. If the users close the pop-up or touch the screen, the app becomes fully visible again, and the app again enters the Resumed state.

If users don't touch the screen for a longer period of time and the screen goes black, or the user starts another app so that the original app is no longer visible, the app enters the Stopped state.

Types of Android Applications

• **Foreground:**

- application that's useful only when it's in the foreground and is effectively
- suspended when it's not visible

- **Background :**

- application with limited interaction
- spends most of its lifetime hidden

- **Intermittent Applications:**

- well-designed applications
- applications that expect limited interactivity but do most of their work in the background.

- **Widgets and Live Wallpapers :**

- applications are represented only as a homescreen Widget or as a Live Wallpaper.

The Activity Lifecycle

- Android runtime manages Activities
- Activities have a “lifecycle” consisting of states: from creation until death
- Standard (lifecycle) methods on the activity are invoked at each state change (can test by rotating device)

Activity States

- Created: Born to run
- Active: Working 9 to 5
- Paused: I'm about to break
- Resumed: Back to work
- Stopped: Obscured by clouds, vulnerable

Activity Transitions

- Created \Rightarrow Active
- Active \Leftrightarrow Paused
- Paused \Rightarrow Stopped \Rightarrow Active
- Stopped \Rightarrow Killed

Lifecycle Methods

- onCreate(Bundle savedInstanceState): create views, (re) initialize state
- onStart(): Restore *transient* state; one-time processing
- onResume(): Session-specific processing, restore *transient* state
- onPause(): Save persistent data, release resources, **quickly!** Last method guaranteed to be called.
- onStop(): Called optionally by runtime
- onDestroy(): If finish() is called, or object is being *temporarily* destroyed. Distinguish via isFinishing().

If users *turn on the screen or use the Back button to get back to the app*, the app again enters the *Resumed* state. An app can remain in the *Stopped* state for quite some time.

However, if the *device is rebooted or a user runs a number of other apps before coming back to the original app*, that app can be *Destroyed* by the operating system to free up resources for other apps that the user is actually interacting with.

To design an app that functions well given this pattern of use, developers must understand what happens as the app enters and leaves these states, as well as what they should design the app to do in those instances. This requires understanding the Android life cycle.

The Android life cycle (Figure 1) begins when a user touches an app's icon. This action causes the onCreate method in the app's initial activity to execute. This method

includes code to load the screen (called a *layout*) associated with the initial activity to load. The developer needs to place code in this method that initializes variables and layout objects to the settings required for the user to begin interacting with the app.

After the activity has been created, the `onStart` method is executed. This method does not have to be implemented but is useful if the app requires certain settings to be the same for every time the app starts, whether it is an initial start after the activity is created or restarted after the activity is brought back from a stopped (but not destroyed) state.

After the activity has started, the `onResume` method is executed. This method also does not have to be implemented but is very useful to return the app to the running state that the app was in before it paused. This includes turning on system services used by the app (for example the GPS or the camera), restarting animations, and any other settings needed to allow users to pick up where they left off.

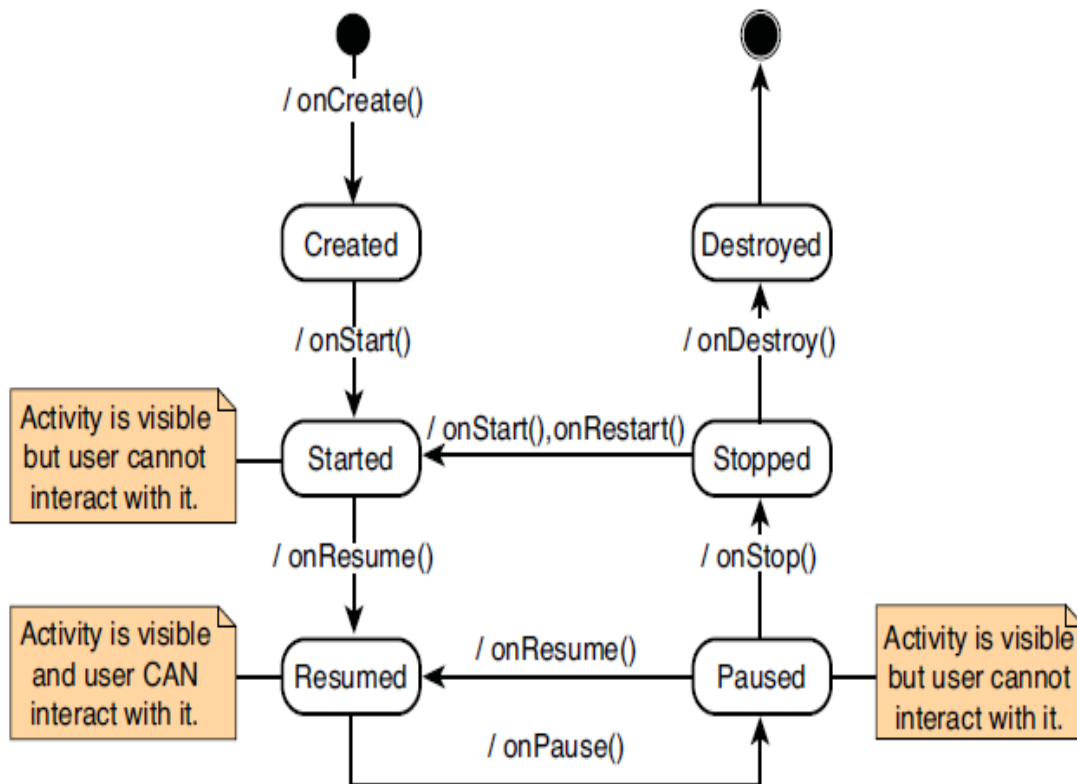


Figure 1 Android life cycle.

When a user stops directly interacting with the app, the path to destruction begins. None of the methods executed on the path to destruction have to be implemented. However, they often serve a useful purpose and should be considered. The first method executed is `onPause`. This method should be used to stop services that the app is using, to stop animations, or to store important state information so that users can start using the app exactly as they left it. If the app is about to become invisible, the `onStop` method will be executed. This method should make sure important data is permanently stored so that as system resources are consumed by other apps, they are not lost.

Finally, if not restarted, the `onDestroy` method will be executed just before the operating system takes away all the app's resources. This is your last chance to capture important data before all is lost.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        Toast.makeText(this@MainActivity, "this is the on create ", Toast.LENGTH_SHORT).show()

    }

    public override fun onStart() {
        Toast.makeText(this@MainActivity, "this is the on start ", Toast.LENGTH_SHORT).show()
        super.onStart()
    }

    public override fun onRestart() {
        super.onRestart()
        Toast.makeText(this@MainActivity, "this is the on restart ", Toast.LENGTH_SHORT).show()
    }

    public override fun onResume() {
        Toast.makeText(this@MainActivity, "this is the on resume ", Toast.LENGTH_SHORT).show()

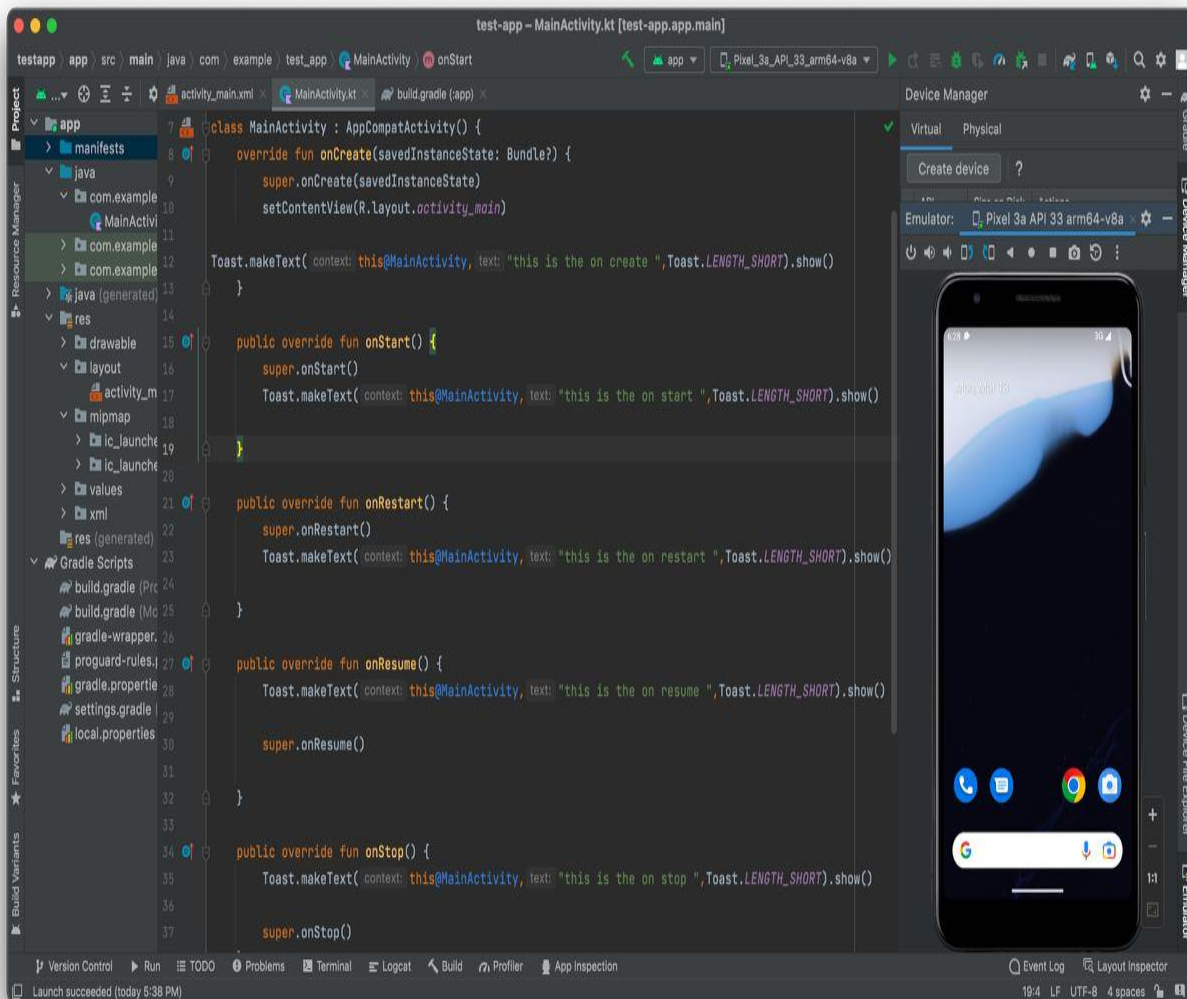
        super.onResume()
    }

    public override fun onStop() {
        Toast.makeText(this@MainActivity, "this is the on stop ", Toast.LENGTH_SHORT).show()

        super.onStop()
    }

    public override fun onDestroy() {
        Toast.makeText(this@MainActivity, "this is the on destroy ", Toast.LENGTH_SHORT).show()

        super.onDestroy()
    }
}
```



2.5 iOS Life Cycle

The life cycle for iOS is similar to Android's. However, iOS uses both an app life cycle and a screen (called *view*) life cycle to accomplish essentially the same things. As with Android, the life cycle (Figure 2) begins when the user taps an app's icon. The `application:didFinishLaunchingWithOptions:` method is similar to an activity's `onCreate` method. However, in iOS this method is used to set up the operating environment for the complete app, not just a single activity.

The `applicationWillResignActive:` method is executed when the app is interrupted, similar to when the `onPause` method is executed in Android.

Finally, when the app is no longer visible, the `applicationDidEnterBackground:` method is executed. As with Android, code in these methods should be used to turn off services and save important data for the user before it's potentially lost.

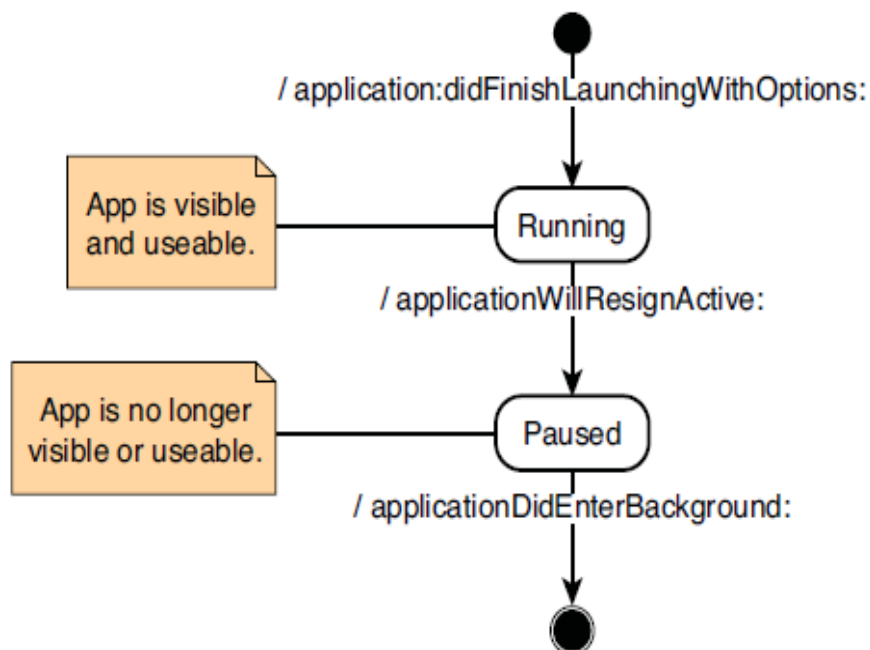


Figure 2 iOS App life cycle.

Unlike Android, iOS has a separate life cycle for displayed screens (called `ViewControllers`). The view life cycle (Figure 3) begins after the application has finished loading or the user goes to a different page in the app. After the view is loaded into memory, the `viewDidLoad:` method executes. This method is executed only once if the view stays in memory. You should write code in this method to set the initial state of the view. After the view has loaded into memory, just before the view is visible to the user, the `viewWillAppear:` method is executed.

Code in this method should be used to load any data into the views that will be visible to the user and turn on services that the user needs to interact with the app. This method executes every time the view reappears on the device.

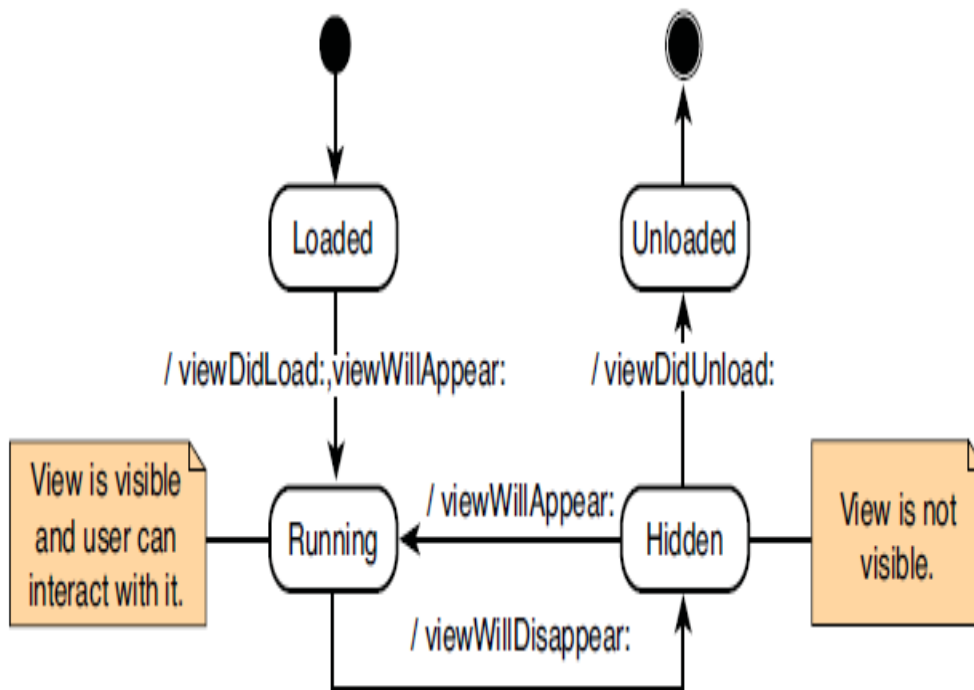


Figure 3 iOS View life cycle.

Just like in Android, if an app is interrupted, or the user does not interact with the device for a period of time, or the user moves to another view in the app, the view is pushed into the background.

2.6 Screen Size and Orientation Issues

The most obvious difference between mobile and traditional application design is the amount of real estate you have to work with. The mobile device has significantly less area to design the interaction that your users can experience with your app. Poor interface design is the easiest way to get bad reviews for your app.

Mobile devices are also used in different situations than traditional computing devices are. App users are often multitasking (walking, talking with friends, and so on). The app design must **allow users to switch to your app and do what they want to do right away, before they are distracted again.** If users can't easily figure out how to use the app, no amount of help will satisfy them. This is no different from traditional development.

However, the very **limited screen real estate makes it a significant challenge.** In addition, the focus among app developers has been on very good user interface design, so the competition is fierce for apps that work really well.

In response to the limited screen size, **both iOS and Android have the capability to scroll to interface elements not on the screen.** Scrolling can be both horizontal and vertical. However, both scrolling capabilities should be used judiciously **حكمة**, especially horizontal scrolling. Scrolling down a list has become a natural action on **both traditional computers and mobile devices.**

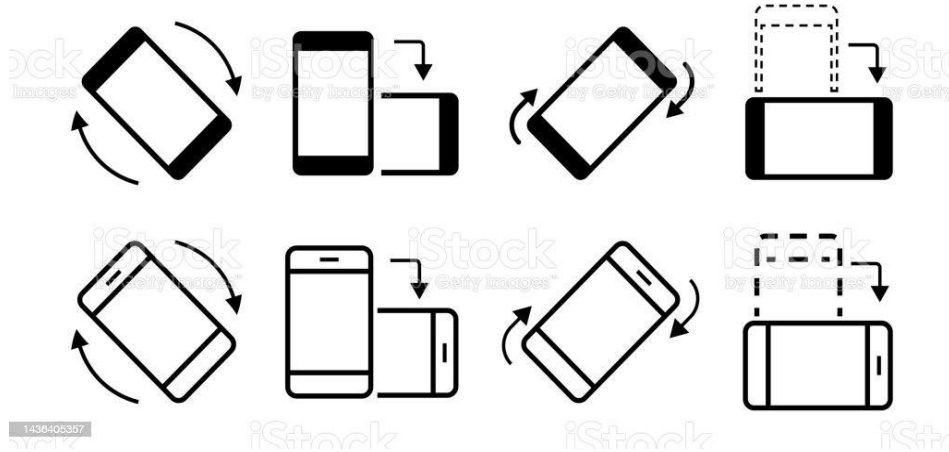
However, horizontal scrolling has not. Horizontal scrolling should be reserved for use for elements that start on the main screen and extend off the screen. Users won't naturally think to horizontally scroll to look for items they can't find on the main screen. Even **vertical scrolling should be limited.** Lists are obvious choices for vertical scrolling, but other types of interface elements should be limited. Additionally, **when scrolling, you must also fix certain elements so that the user can perform needed operations without scrolling back through the entire contents of the screen.**



The obvious answer to the limited screen size is to carefully plan the user's interaction with your app. Screens should focus on one, or a very limited and coherent, set of tasks that the user can or would want to do. Navigation should be planned and designed so that it is obvious to the user how to proceed to the next task. If a task requires multiple steps, those steps should be designed as distinct screens, and the user should be guided through the screens needed to complete the whole task.

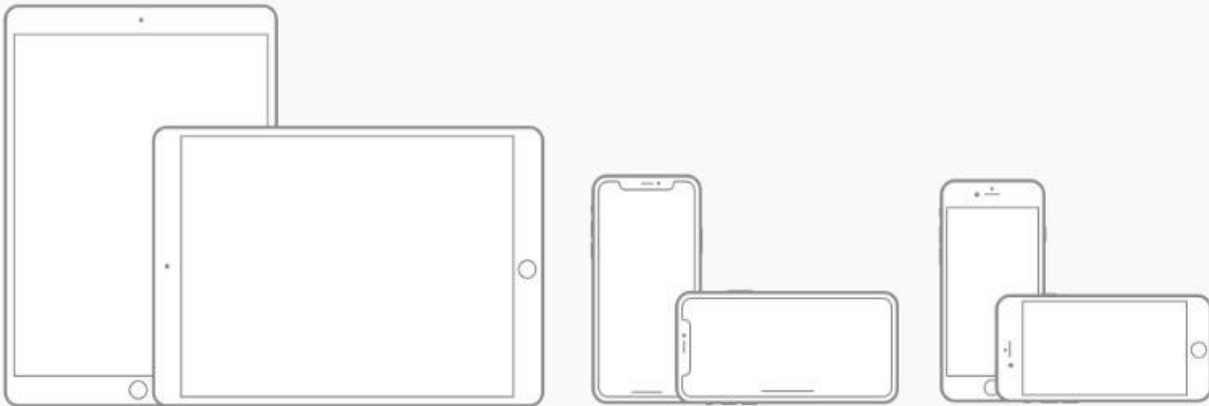
Although screen size is a nontrivial design issue, the fact that by default, a screen's orientation can change as the user turns the device also presents design issues. When the user turns the device from vertical to horizontal orientation, the layout or view reorganizes to that orientation. This significantly changes the amount of vertical and horizontal real estate for your interface. Interface elements that were obvious to the user in the vertical orientation may become inaccessible in the horizontal orientation. Again, this can be a very frustrating experience for your user, unless you carefully plan for the layout in both orientations, and thoroughly test it as well. Scrolling can be implemented to alleviate some of the problems associated with orientation change. However, simply adding scrolling may not solve the user experience issues. **If you cannot make it work in an alternative orientation, as a last resort you can code the app to work in only one orientation.**

- i. The solutions to these screen size and orientation issues are planning and design.
- ii. What does the user want to do with your app?
- iii. What can your user do with your app?
- iv. What are the logical steps needed to accomplish those tasks given the device limitations?
- v. These are the types of questions you must answer to design a successful app.

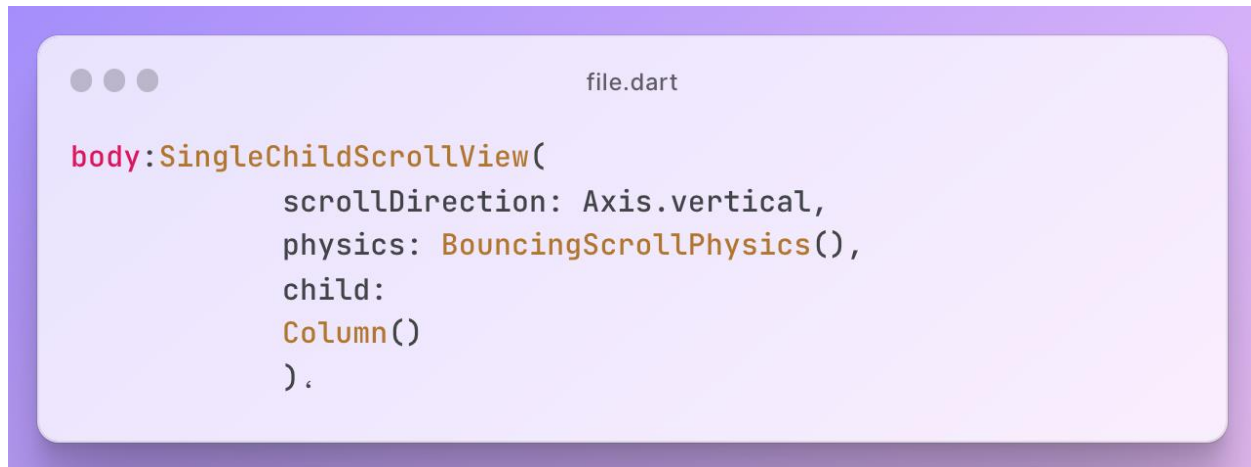


Device Screen Sizes and Orientations

iOS devices come in a variety of different screen sizes and can be used in either portrait or landscape orientation.



| Device | Portrait dimensions | Landscape dimensions |
|------------------|---------------------|----------------------|
| 12.9" iPad Pro | 2048px × 2732px | 2732px × 2048px |
| 10.5" iPad Pro | 1668px × 2224px | 2224px × 1668px |
| 9.7" iPad | 1536px × 2048px | 2048px × 1536px |
| 7.9" iPad mini 4 | 1536px × 2048px | 2048px × 1536px |
| iPhone X | 1125px × 2436px | 2436px × 1125px |
| iPhone 8 Plus | 1242px × 2208px | 2208px × 1242px |
| iPhone 8 | 750px × 1334px | 1334px × 750px |
| iPhone 7 Plus | 1242px × 2208px | 2208px × 1242px |
| iPhone 7 | 750px × 1334px | 1334px × 750px |
| iPhone 6s Plus | 1242px × 2208px | 2208px × 1242px |
| iPhone 6s | 750px × 1334px | 1334px × 750px |
| iPhone SE | 640px × 1136px | 1136px × 640px |

A screenshot of a code editor window titled 'file.dart'. The code defines a SingleChildScrollView widget with vertical scrolling, bouncing physics, and a Column child.

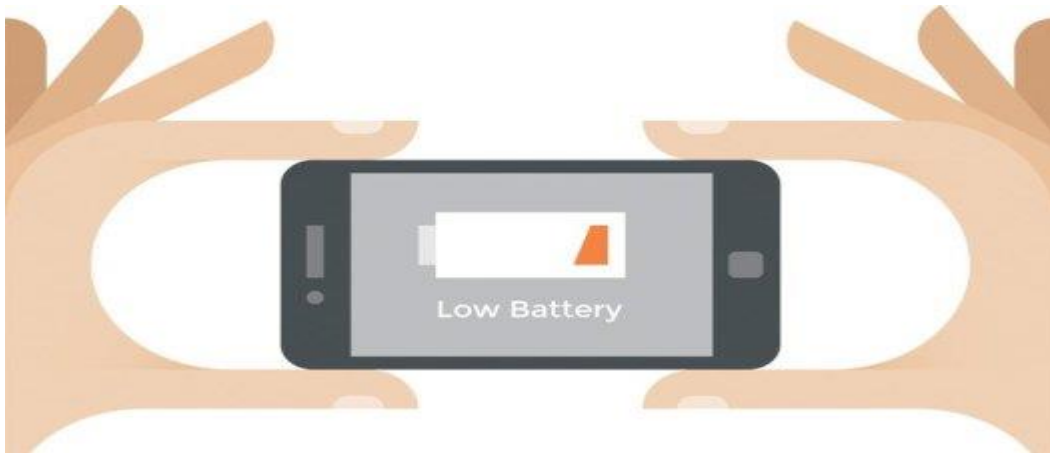
```
body:SingleChildScrollView(  
    scrollDirection: Axis.vertical,  
    physics: BouncingScrollPhysics(),  
    child:  
    Column()  
).
```

2.8 Battery Issues

Mobile devices are just that mobile. This means that they are not always connected to a power source. They rely on batteries for their power, and batteries can be drained. Your job as a developer is to not drain those batteries unnecessarily. **This is not just a courtesy issue. If every time your app is used the user's device quickly becomes a brick, it will be noticed. An app that quickly drains power will not get used, will get bad reviews, and eventually will not get downloaded at all.**

The app's life cycle plays an important role here. If the app is interrupted, all device access or use should be suspended immediately. When the app is about to become active, turn on as late as possible only those device capabilities needed. For example, in the previously mentioned app that uses weather data, the weather retrieval is started as one app activity becomes active. If the weather data is successfully retrieved, it is time stamped. The next time the activity becomes active, the weather data will be retrieved only if it is outdated, thus saving battery power. Some battery issues are beyond your control. You cannot make an app that extends battery power. However, you can definitely make an app that significantly reduces battery life. Be sure to plan for battery use when designing your app. Here are some common reasons why your mobile battery may be draining quickly:

1. Too many push notifications and alerts
2. Too many apps running location services
3. Too many apps running in the background
4. The screen is too bright
5. The screen is staying on too long before going to sleep



2.9 Hardware Issues

A very cool aspect of mobile computing is the set of hardware components available on the device. Many devices have the capability to locate the device within a few meters using the GPS, have sensors that can capture device **orientation**, have lights that can be turned on and off, have cameras, and have other hardware components that allow the device to interact with the environment. Access to these components can make fun and useful apps.

A final important issue with the use of hardware devices is accuracy. There are several aspects of this issue.

- **First**, the accuracy of the component can differ among manufacturers. Consider what the minimal level of accuracy is needed for effective use of your app, and

design for that. Be sure to give the user options if the required level of accuracy is not available.

- **Second**, accuracy often takes time. For example, to find the location of the device within a few hundred meters is often very quick. However, accuracy of a few feet often takes much more time. What is the required level of accuracy for your app's functionality? What can the user do if the device cannot achieve this? How quick does the acquisition of location need to be? All are important considerations when you are designing the app. A very good design strategy when you need better accuracy is to keep the user informed of progress. The Google Maps app provides an example of this. When finding your location on the map, the app first shows a big blue circle that gets progressively smaller as the accuracy improves.
- **Finally**, how the device returns data to the app may impact the level of accuracy your app can access. Again, using GPS as an example, the number of digits reported for the latitude and longitude coordinates dictate the level of accuracy of those coordinates. In some cases, the number of digits reported can differ. This is primarily an issue for the Android platform because it can differ among versions of the Android OS.

Overheating is a very common problem in smartphones, and there are many things that could be causing it on your device. These include hot weather, faulty manufacturing, the use of incompatible chargers and cables, internal damage, and more. You might be aware that overheating damages battery health and reduces its overall capacity which eventually leads to quick battery drain. In rare cases, it can even cause your phone to start swelling or injure you.

Follow these tips to avoid overheating:

1. Avoid using your phone while it's charging.
2. Avoid using your phone in very hot weather.

3. Use chargers and cables designed by the manufacturer.
4. Don't play graphics-intensive games your phone can't handle.
5. Get the battery replaced with a new one.

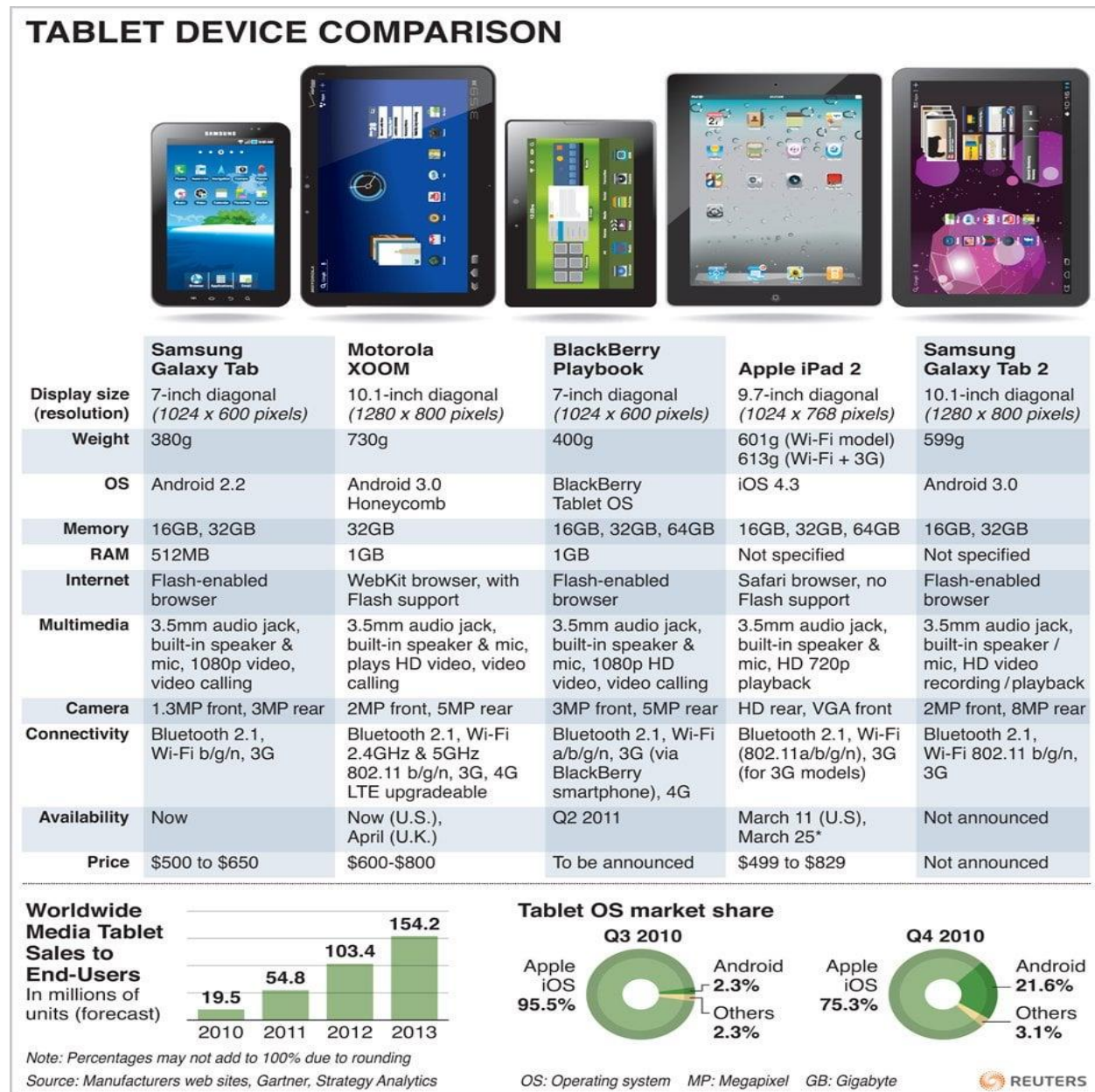
There can be many reasons why your phone is *freezing or crashing apps* upon launch. For instance, if the device is really old, it's obvious that its hardware is failing; by that point, you should probably just buy a new Android phone.

Apps can also crash if you've not given them the permissions necessary to operate properly; for instance, a navigation app can't work without access to your location. Your phone can also freeze due to a lack of internal storage, in which case, you should free up some storage space to make sure the OS has enough room to run smoothly. For more help, check out these solutions to fix crashing apps.



2.10 Device Differences

Android devices (phones and tablets) and iOS phones and tablets each have a unique set of hardware and software capabilities that make the way the user interacts with the device different for each. Again, to fully capture the device's capabilities and not degrade the user experience, you must design for those unique characteristics.



2.11 Android

Android devices originally used four hardware buttons (Figure 4) to support the user's use of the device. These buttons were the Home button, the Menu button, the Search button, and the Back button. The user could press any of these buttons at any time during use of your app, which would impact the functioning of your app. The **Home** and **Back** buttons worked independently of your code, whereas the **Menu** and

Search buttons provided functionality only if your app was specifically coded to use these buttons.



Figure 4 Android hardware buttons.

However, more recent Android devices have replaced these buttons with virtual buttons at the bottom of the screen and an action bar at the top of the screen (Figures 5 and 6, respectively). The Back and Home virtual buttons remain the same in both form and function. However, the Menu and Search buttons were eliminated, and a Recents virtual button was added. The Recents button shows the user's recently used apps.

The action bar displays the app's icon and title and the menu. Menu items will be displayed with an icon (if defined). If there are too many menu items to be displayed with an icon, the extra menu items are accessible through the three vertical dots on the right side of the menu bar. If an app is targeting older versions of Android as well as newer, the action bar presents only the three dots at the far right. When pressed, these dots perform the same function as the Menu button.



Figure 5 Android virtual buttons.



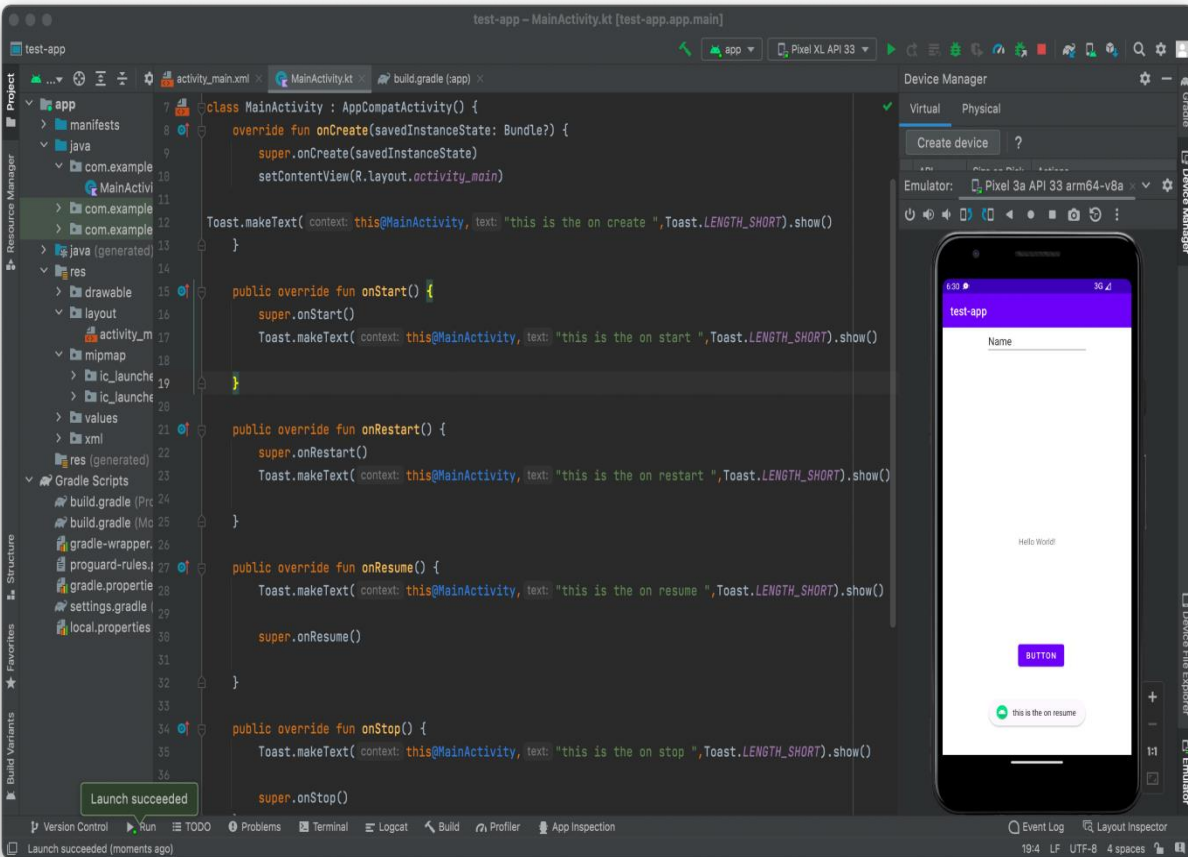
Figure 6 Android action bar.

The Home button immediately moves your app to the Stopped state. This causes the `onPause` and `onStop` methods to execute. It will **not destroy** the app unless it needs the system resources. This means you must pay attention to these events even though you may not be anticipating this behavior when your app is in use.

The Back button immediately goes back one action or activity. This can have several implications for your code. *For example*, if the user is looking at an activity and presses the Back button, the visible activity will be immediately moved to the Stopped state (causing the `onPause` and `onStop` methods to execute). It will move the previous activity into the Running state. This will cause the `onStart` and `onResume` methods to execute for that activity. If your activity has displayed a pop-up, the activity is currently in the Paused state (because it is partially visible). Pressing the Back button will hide the pop-up and cause the `onResume` method to execute, and your activity will be placed in the running state. If the soft keyboard is displayed, your app is also in the Paused state. Pressing the Back button will hide the keyboard and again put your activity in the Running state.

If the user presses the Menu button, your app will not do anything unless you have specifically programmed it to have a menu. Menus can be useful ways to provide the user with access to functionality that is not used as the normal course of events in using your app; thus, you don't want to waste valuable screen real estate to provide access to that functionality.

Finally, the Search button also does nothing unless you code it. You can use this button to allow the user to search for information within your app. The hardware/virtual buttons provided by Android devices either have an impact on your app or can be used to extend the functionality of your app. In either case it is important to plan for the impact of these buttons when designing your app.



2.12 iOS

The primary hardware button of concern on iOS devices is the Home button. This button immediately moves any app presently running to the background. The `viewWillDisappear:`, `applicationWillResignActive:`, and `applicationDidEnterBackground :` methods will all be called. Plan your app so that this action will not cause problems.

Both Android and iOS have a button that puts the device to sleep or reboots it. This action also must be handled. Fortunately, the same methods that put the app in the background for other actions are executed so, typically, no additional programming is required to prepare for this.

2.13 Introducing Example App

To learn both Android and iOS design and development, let us consider building the same app on each platform. Building the same app on both platforms is useful for understanding differences and similarities between the platforms. The app is called *MyContactList*. Building a contact list app is a good way to learn mobile development for two reasons. First, its purpose and function is generally understood, so a significant part of any application development effort (understanding the functional requirements) does not need to be explained. Second, a contact list app requires utilizing many basic and advanced features of mobile development; therefore, it is very useful in providing a context for learning these concepts.

The MyContactList app consists of four different screens. Each screen is used to illustrate basic app development concepts you will use in almost any subsequent app you develop. Additionally, you'll learn how to navigate between screens in an app.

2.14 Contact Screen

The contact screen shown in Figure 7 is used to enter, edit, and save information about people in your contact list. While developing this screen, you learn some of the fundamental concepts of mobile user interface design and data entry. Later on, you use this screen as a way to learn how to create and store data in a database on a mobile platform. Finally, the contact screen shows you how to integrate hardware capabilities into an application by using the device's camera to capture a contact's picture and to make a phone call by tapping the contact's phone number.

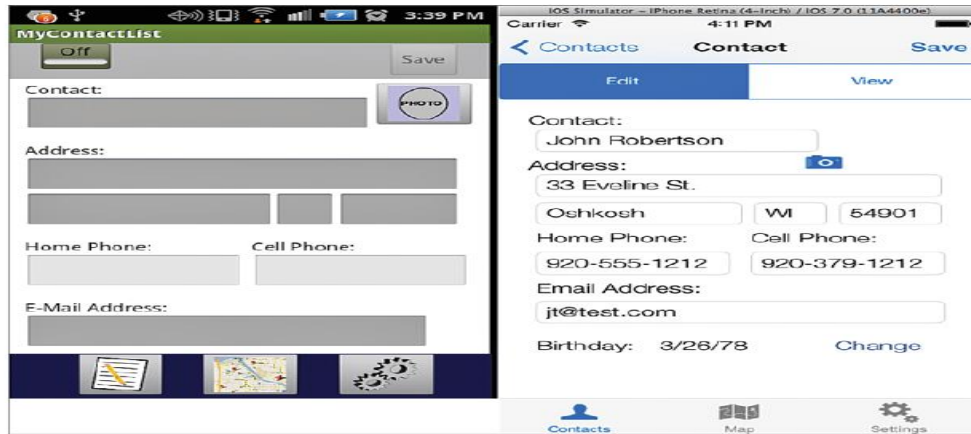


Figure 7 The Contact screen.

2.16 Contact List Screen

The contact list (Figure 8) is used to search for basic contact information and allow selection of a contact for further action (for example, editing and deleting). Lists are very important components of many apps on both Android and iOS. Developing this screen teaches you how integrate them into any future app. This screen also demonstrates how to access information provided by hardware components of the device.

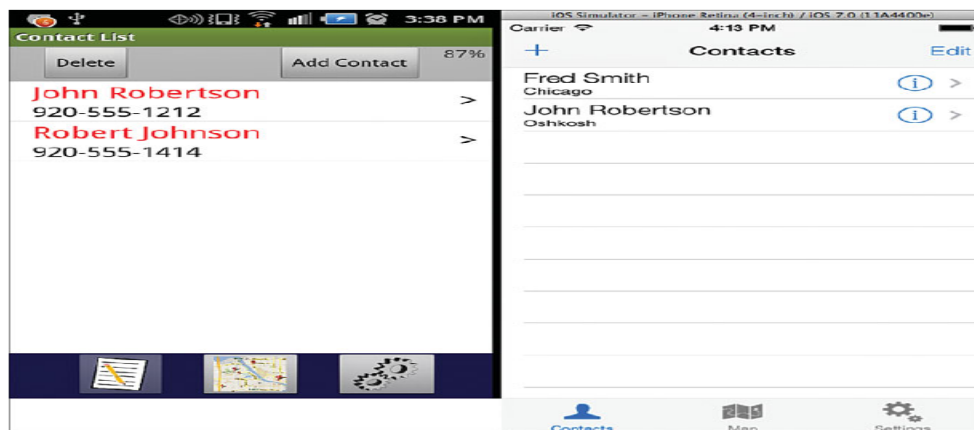
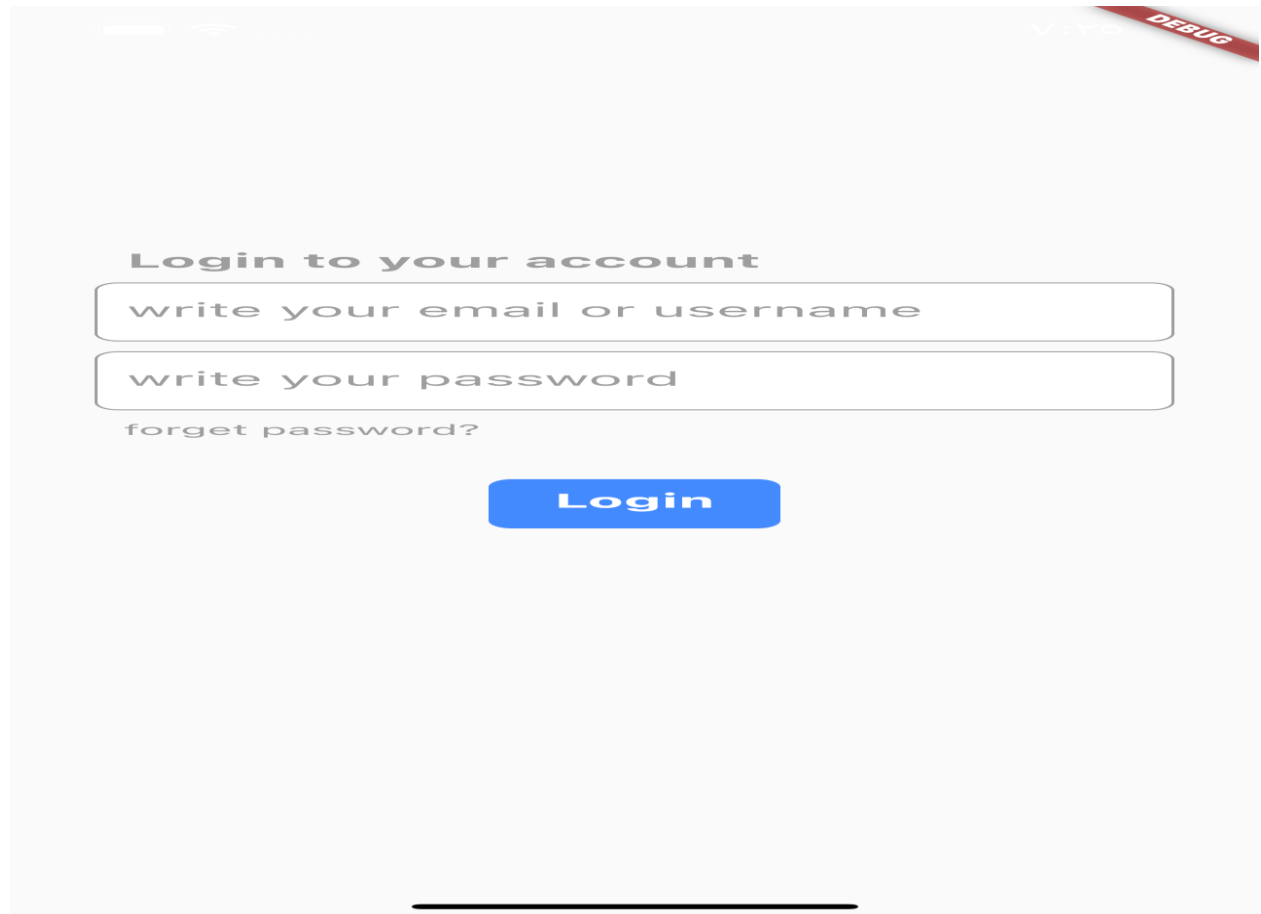


Figure 8 The Contact List screen.

Another example



```

main.dart

import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: SafeArea(
          child: Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: [
                SizedBox(
                  height: 200,
                ),
                Container(
                  alignment: Alignment.centerLeft,
                  padding: const EdgeInsets.only(left: 40, bottom: 10),
                  child: Text(
                    ('Login to your account'),
                    style: TextStyle(
                      fontSize: 20,
                      color: Colors.grey,
                      fontWeight: FontWeight.bold),
                  ),
                ),
                Container(
                  padding: const EdgeInsets.only(left: 10, top: 15),
                  decoration: BoxDecoration(
                    color: Colors.white,
                    borderRadius: BorderRadius.circular(8),
                    border: Border.all(color: Colors.grey)),
                  width: 370,
                  height: 60,
                  child: Text(
                    ('write your email or username'),
                    style: TextStyle(
                      fontSize: 20,
                      color: Colors.grey,
                    ),
                  ),
                ),
                Padding(padding: const EdgeInsets.only(top: 10)),
                Container(
                  padding: const EdgeInsets.only(left: 10, top: 15),
                  decoration: BoxDecoration(
                    color: Colors.white,
                    borderRadius: BorderRadius.circular(8),
                    border: Border.all(color: Colors.grey)),
                  width: 370,
                  height: 60,
                  child: Text(
                    ('write your password'),
                    style: TextStyle(
                      fontSize: 20,
                      color: Colors.grey,
                    ),
                  ),
                ),
                Container(
                  alignment: Alignment.centerLeft,
                  width: 350,
                  height: 40,
                  child: Text(
                    ('forget password?'),
                    style: TextStyle(color: Colors.grey, fontSize: 15),
                  ),
                ),
                Padding(padding: const EdgeInsets.only(top: 30)),
                Container(
                  padding: const EdgeInsets.only(top: 10),
                  decoration: BoxDecoration(
                    color: Colors.blueAccent,
                    borderRadius: BorderRadius.circular(8)),
                  width: 100,
                  height: 50,
                  child: Text(
                    ('Login'),
                    textAlign: TextAlign.center,
                    style: TextStyle(
                      fontSize: 20,
                      color: Colors.white,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}

```

2.17 Map Screen

The map screen (Figure 9) is used to display the recorded location of a single contact or all your contacts on a map with a pin. The screen also demonstrates how to display the device's present location on the map and how to switch between different map views. The usefulness and importance of maps on mobile devices needs no further explanation. Through the development of this screen you learn how to integrate mapping into your apps. Additionally, the screen will be used to demonstrate another approach to accessing sensor information.

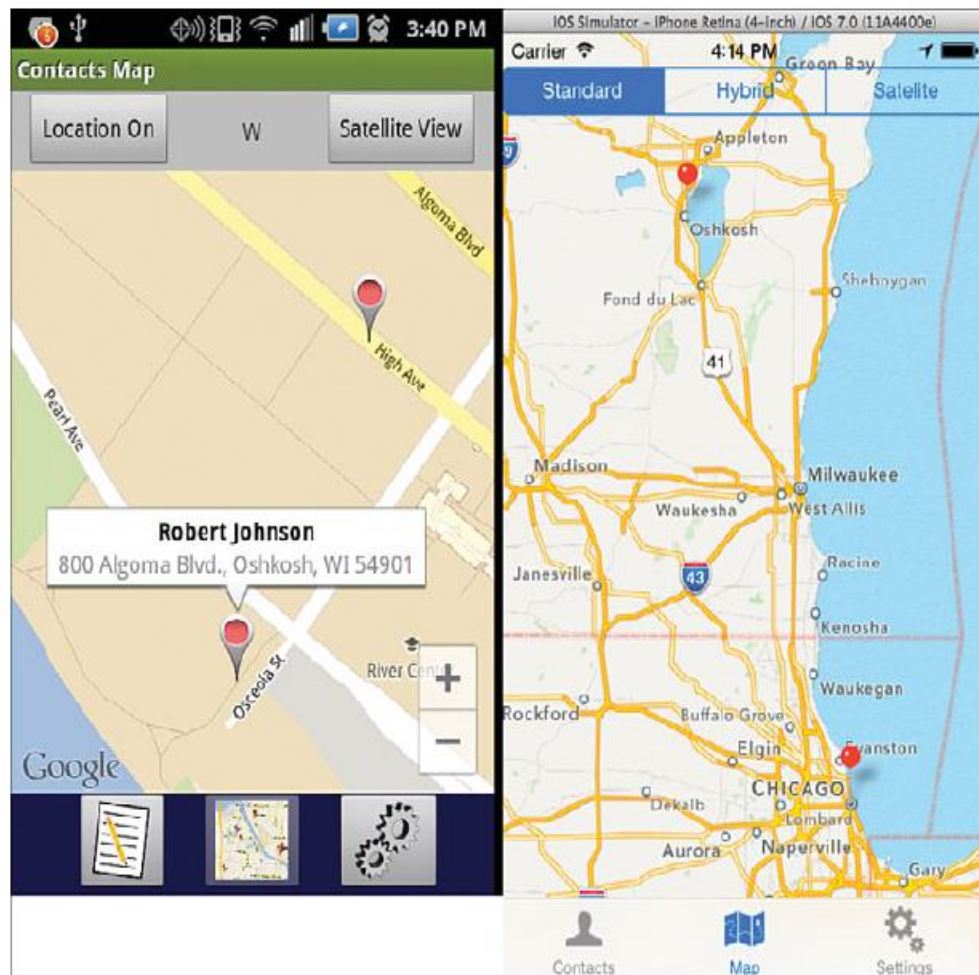


Figure 9 The Map screen.

2.18 Settings Screen

The Settings screen (Figure 10) is used to set the sort order for the contacts in the Contact List. In developing this screen, you learn to use a method of data persistence designed for capturing and storing individual pieces of data. This type of data persistence is often used to capture user preferences for an app. You also learn to use a different type of display widget (view).

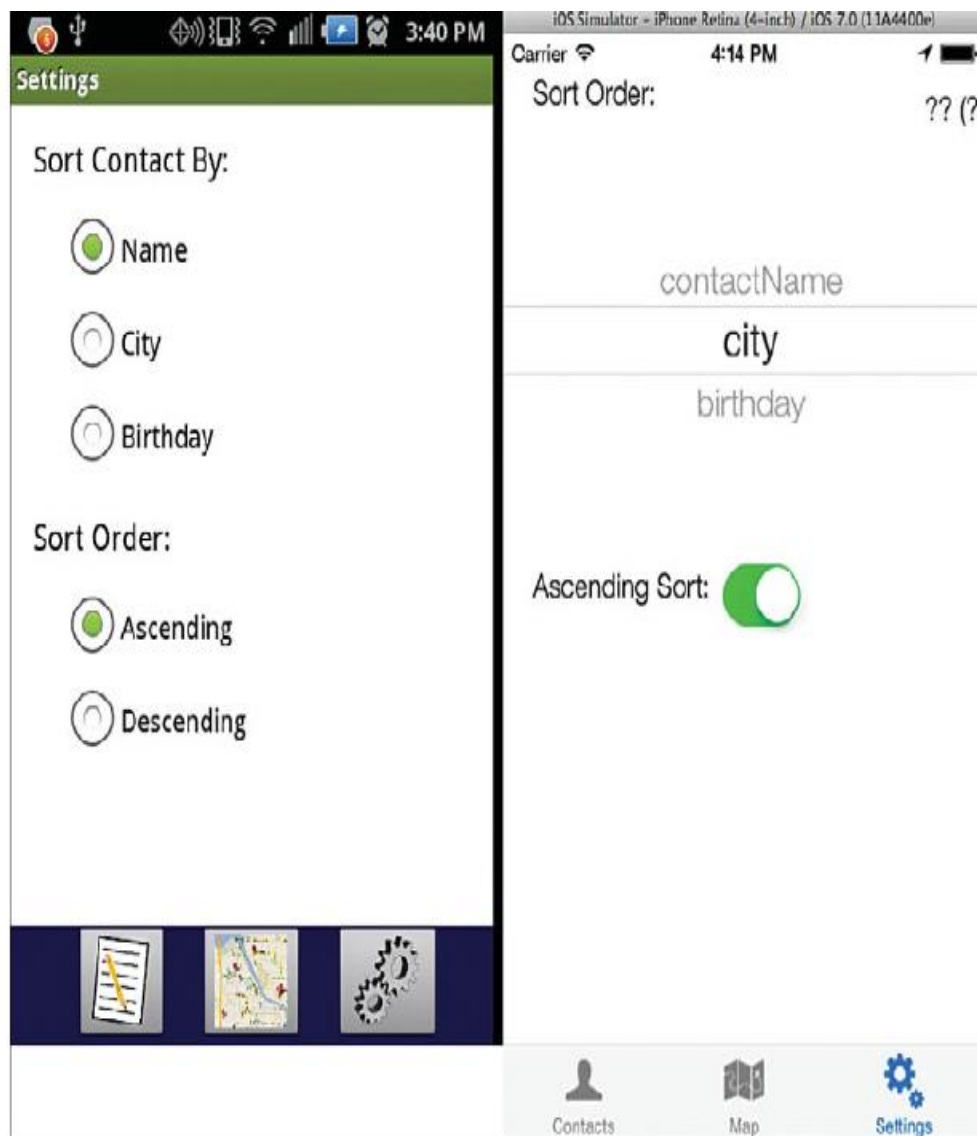
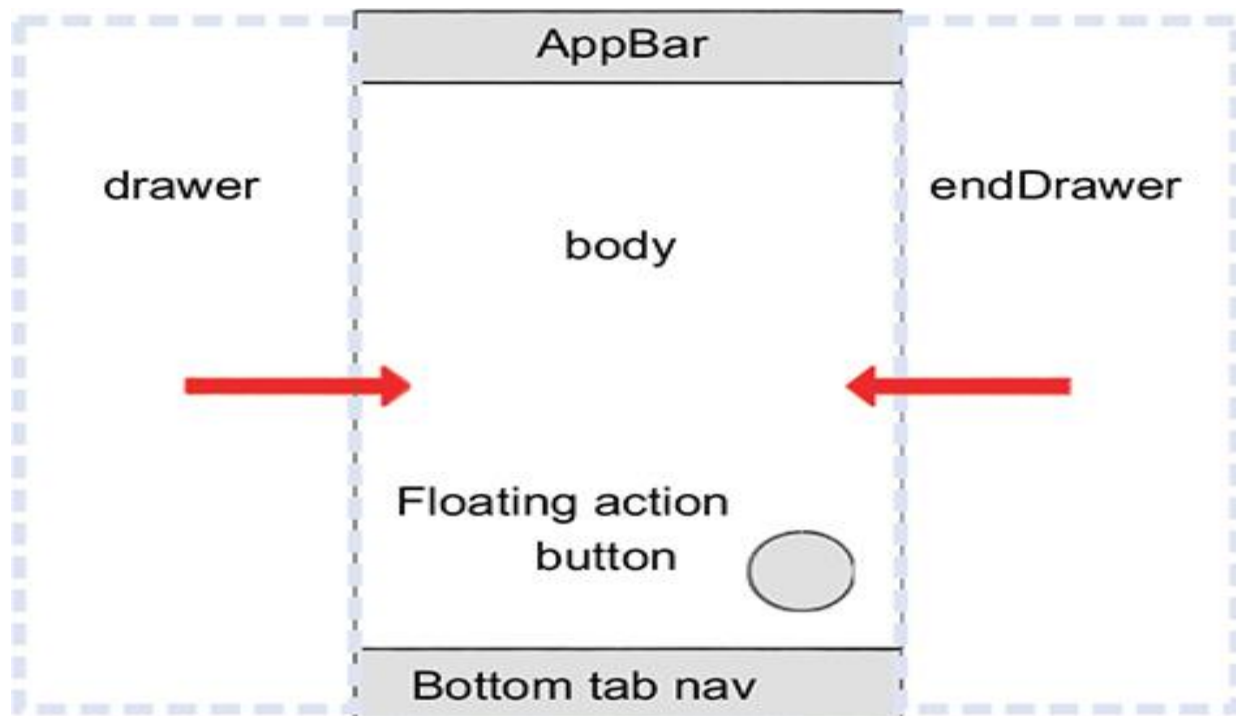
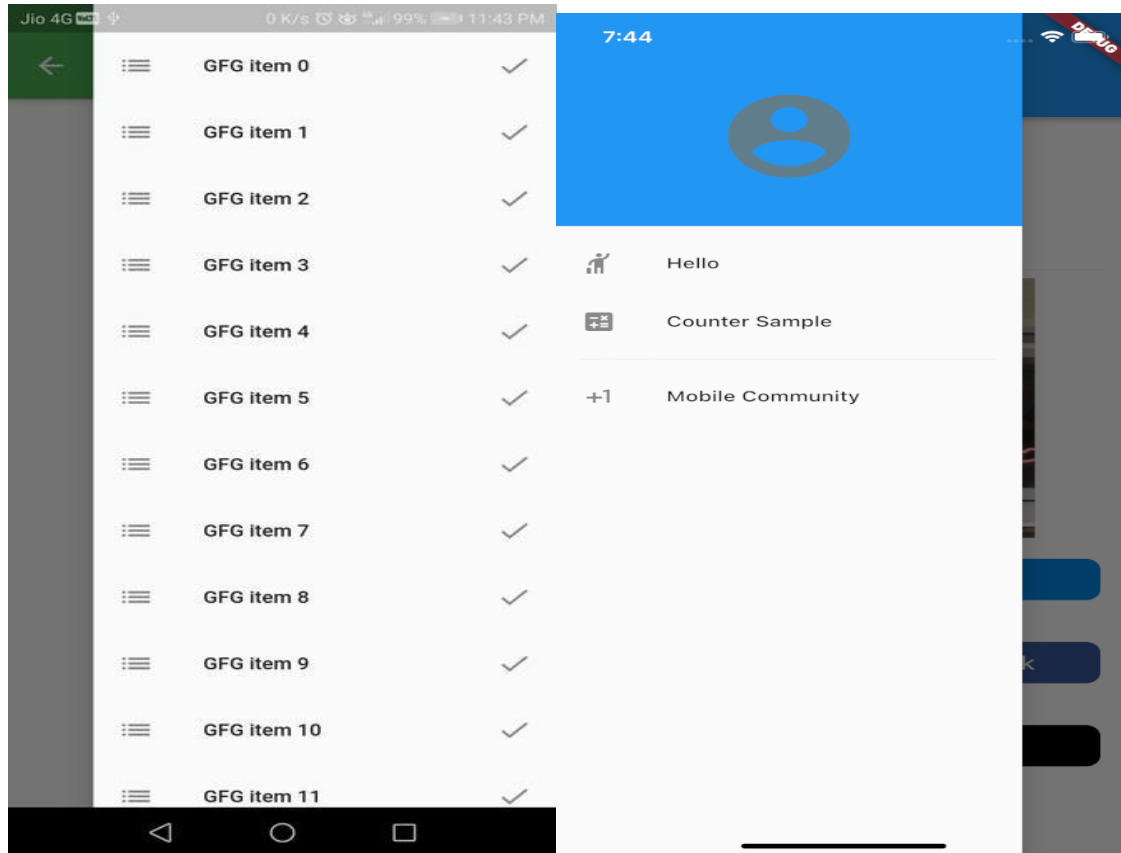


Figure 10 The setting screen .

Extra info

- The difference between the drawer and endDrawer is Drawer by default opens from left-to-right and the endDrawer by default opens from right-to-left. However, we can use the `textDirection` property to change the direction
- The Scaffold is a widget in Flutter used to implements the basic material design visual layout structure in our app
- <https://flutterawesome.com/>





To use this plugin, add `url_launcher` as a [dependency in your pubspec.yaml file](#).

```
import 'package:flutter/material.dart';
import 'package:url_launcher/url_launcher.dart';

final Uri _url = Uri.parse('https://flutter.dev');

void main() => runApp(
  const MaterialApp(
    home: Material(
      child: Center(
        child: ElevatedButton(
          onPressed: _launchUrl,
```

```
        child: Text('Show Flutter homepage'),  
      ),  
    ),  
  ),  
),  
);
```

```
Future<void> _launchUrl() async {  
  if (!await launchUrl(_url)) {  
    throw Exception('Could not launch $_url');  
  }  
}
```

3.1 Monetizing Apps

The market for apps is big and getting bigger. Vision Mobile has estimated the mobile app economy to be growing up to \$6.1 trillion in 2021. When you create and test an app, you need your efforts to pay off. What is the best way to do this? What options do you have to make money from apps? What do you have to do to start? In this chapter, you explore the various avenues for monetizing apps.



3.2 App Monetization Strategies

Making money from apps is possible but difficult. There are hundreds of thousands of apps available. The revenue generated from an individual app is typically very small, and the app stores take a 30% cut of all the revenue that your app generates.

- How do you get people to pay you to use yours?
- What are the different ways you can collect money for the use of your apps?

- How can you get a lot of people to pay you to use the app so that the revenue generated is enough to fund further development or expansion?

These are some of the questions that many developers are asking.

The infographic is titled "ULTIMATE GUIDE TO APP MONETIZATION STRATEGY". It features a cartoon character with glasses and a red shirt holding a large smartphone. Three coins with dollar signs are floating next to the phone. The background is a teal and orange geometric pattern.

WHY IS APP MONETIZATION ESSENTIAL?

- Mobile apps have become a preferred digital channel for customers and have boosted the app usage growth year-over-year. The app growth has induced change in user behavior, making it imperative for apps to capitalize on in order to drive more sales and revenue.

DEFINE YOUR MONETIZATION STRATEGY

| WHAT | WHO | GOALS |
|--|--|---|
| <ul style="list-style-type: none"> Determine the value your app provides to users Realize what your users will pay for | <ul style="list-style-type: none"> Understand who your target audience is How will you incorporate their preferences | <ul style="list-style-type: none"> Strategize how to drive users for your app How to generate revenue |

THE STATE OF APP STORE REVENUE IN 2017

3.3. Paid Apps

The simplest approach to monetizing an app is to charge for the download. The price is advertised in the app store and the user decides, based on your description of the app, whether to buy it. If the user buys the app, you get the money. No need to worry about getting clicks or designing features to be purchased. The problem is getting enough customers to generate significant income. One approach to solving

this problem is to raise the price of the app. However, as the price of the app goes up, the number of downloads goes down. As mentioned before, the market is very price sensitive. You will need a very enticing description of your app to get downloads. Additionally, after users have purchased the app, future updates are free. You would need to add a new app to charge again.

A significant problem for all strategies, which is exacerbated with paid apps, is getting the potential customer to find your app among the thousands of apps in the app stores. If the user searches the app store using keywords, and your app is displayed along with a number of free apps, the user will often not even read your app's description, focusing only on the free ones.

To attempt to remedy this, you need to advertise. Advertising is a double-edged sword. It costs money! Even a limited Google ad campaign where you pay Google to display an ad whenever someone searches on a set of keywords costs hundreds of dollars. It takes a lot of \$.99 app sales to cover this cost.

An up-and-coming approach to the paid app approach is to build apps for business use. If your app solves a business problem, a business will be happy to pay the cost. This approach does require marketing. It is unlikely that businesses will search the Play Store for a solution to their business problems. This approach requires a much larger commitment than is typical for many app developers. It requires a business plan, establishing a target market, and directly contacting the market with information about your product. It also requires a potentially significant monetary investment.



3.4 Ad Supported Apps

The app market is very price sensitive. Free apps get downloaded at a much greater rate than any paid app. However, a totally free app is hard to make money with. That is why although many apps appear to be free, they often have a way to make money built within the app. The most common approach to making money from a free app is to embed ads within the app screens. Ads take up screen real-estate, so you will have to plan and code the user interface with this in mind.



3.5 In-App Purchases

Making enough money from ad-supported apps to support a business is difficult. You need a very popular app to generate a significant amount of money. Another approach for monetization is to release a free app that is supported by in-app purchases. The basic theory for an in-app purchase monetization strategy is that you generate downloads with the free app, get the users hooked, and then allow them to add features by advertising the feature in the app.

The sale is made during use of the app. This is a very popular approach among app developers.

Another advantage over the basic paid app is that in-app purchasing opens up the possibility of a regular revenue stream from the same user, instead of relying on a single purchase up front.

In-app purchasing is one way to establish a freemium business model, where most of your users use the free version, but a small percentage becomes heavily invested in your app and service and convert to paying for the service, thus underwriting the free experience for everyone else.

You can use several approaches to create products for in-app purchase. Most are dependent on the type of app and its designed use. Game apps often limit the user to only a few game levels and then provide an in-app purchase option for moving to the next level. Other games have consumables that can be purchased during game use. For example, a number of lives are purchased and used up. If users want to continue playing, they have to buy more. Other apps include features that can be unlocked with an in-app purchase. Another approach is to limit the amount of data that can be saved or the number of times the app can be used before the user needs to pay for it. Finally, if your app includes access to content, that content could be purchased on a subscription basis. For example, services that provide weather information cost the developer money, depending on the number of times the weather service is accessed. If the user wants that information in the app, the developer could charge a subscription fee to cover the cost and generate revenue for themselves. Often an in-app purchase strategy is combined with an ad-supported strategy. The free version includes ads that are eliminated as a bonus for an in-app purchase.



3.6 Owning Your Own Business

If you are going to sell apps, you should have a business. This is not an absolute requirement. The Play and App Stores will let you sell apps as an individual. However, it is good practice because you can cleanly separate your individual life, income, and assets from your business income and assets. This is important for both tax and liability purposes. The vehicle for setting up your own business is in most cases a Limited Liability Corporation, or LLC.

3.7 Other Income Possibilities

Making money from selling apps or advertising in apps is possible. However, your income is very dependent on how successful your app is in terms of number of downloads. You should create, publish, and attempt to monetize an app in at least one of the markets. This will give the knowledge to make money in other ways from app development. Although the authors of this book have several apps available in both markets, we make far more money doing training and consulting than we do from the monetization of our apps.

One of the surest ways to make money from app development is to get paid to do it for someone else. This way, you get paid whether the app sells or not. App developers are a very hot commodity in the business world. The average salary for an app developer employed by an organization is around \$100K/year. You have to sell a lot of \$.99 apps each year to make this kind of money. If you establish your ability by publishing apps for both iOS and Android, you can greatly increase your marketability. This is true if you are willing to work for a corporation or if you want to work for yourself as an independent developer.

You can approach independent development as a consultant or a contract developer.

3.8 Choosing a Platform

Should you develop for Android or iOS? Both? There are advantages and disadvantages to all three options.

- You will have to consider your app audience, market size, how committed you are to making a successful app, and what your goals are in publishing an app.
- Android has the biggest market share in terms of devices sold. Therefore, it also has the biggest potential for app sales. However, much of the market growth is outside the United States.
- Your app would have to have universal appeal to take advantage of the growth.
- A second problem with the Android smartphone market is that although the market is big, many of the owners of these phones primarily use them as dumb phones. They are not a likely market for your app.
- Finally, Android app users are much less willing to pay for an app than their iPhone compatriots are.

Studies have shown that each iOS user spends about three times as much on apps as an Android user. On the other hand, some industry segments you might target may have a tendency to use Android devices at a much greater rate than iOS devices.

- Android has an advantage in that the barrier to entry is lower for the casual developer than it is for iOS. The fee to be a developer is a one-time fee, and all the tools are free. You can develop Android apps on either Windows or Mac computers.
- Finally, publishing an app is relatively easy. If you want to publish apps for fun, bragging rights, or to demonstrate your credentials as a developer, Android is a good choice.

- Although the iOS market is smaller than the Android market, people get iPhones because they want to use the apps. They are much more likely to look for apps to use, and iPhone users are also much more willing to pay for an app. This makes selling an app using the simple paid approach as well as in-app purchase discussed earlier much more feasible.
- On the other hand, Apple charges a higher annual fee to be a developer, the publishing process is more complicated, and is in no way guaranteed.
- Finally, you must have a Mac to use Xcode to create your app. You cannot create an iOS app on a Windows machine.

Why not publish on both? If you are trying to make money from apps, this is probably the way to go. You get access to a larger market and thus more potential revenue generation.

Additionally, after you've gone through the effort of designing a user experience, user interface, and logic to create an app for one market, you can leverage that effort by creating an app for the other environment. You will have to completely recode the app, but coding is much easier if you already know what you want the app to do. Additionally, many of the control structures are almost identical on the two platforms (for example, a for loop) so much of the code structure can be copied directly between the code files. You will have to learn both development environments in-depth, but you've got a good start from what you have learned by completing this module. **The primary problem with targeting both markets is that you will have to keep your app current on both platforms. This means more work just to keep the lights on.**

You can make money from mobile apps in several ways. Revenue can be generated through monetization of the app itself using ads, in-app purchase, or charging for the download. If you choose to generate revenue from your apps, you should create a Limited Liability Corporation for both tax and liability purposes. You can also make money by coding apps for other people or organizations as a consultant or freelancer.

Finally, money can be made by getting hired as an app developer in a larger company. This is a much more reliable source of income than any of the other approaches.

- What kind of app do you want to make?
- Who is the target audience? How will you monetize it?
- Why would somebody want your app?
- How difficult would it be to make?

Becoming a mobile app developer allows you to be part of this creative process and bring your ideas to life. Mobile app development provides you with the opportunity to work with different software development kits and programming languages, such as Java and Python, enabling you to master the art of creating hybrid mobile applications for multiple platforms, including iOS and Android operating systems. In addition, testing the application on target devices helps developers identify technical problems before the application is available to the public, improving the user experience you can use these methods to choose your platform :

1. Determine Your App's Purpose and Target Audience
2. Your Budget and Timeline.
3. Evaluate the Platform's Features and Capabilities.
4. Look for Strong Community Support and Resources.
5. Choose a Platform That Aligns with Your Development Skills and Experience.

Chapter 4

Publishing Apps

4.1 Introduction

Creating an app can be a challenging exercise, but for most developers this is not enough. The finished app has to be available for someone to use to complete the process. Apps are made available to their audience by publishing them. The manner of publication is dependent on the target audience and the platform. The two primary audiences are typically the employees of an organization for which the app was developed or the public. Many aspects of publishing are the same for both audiences. However, some important differences exist that the developer needs to be aware of. Publishing Android and iOS apps have many similarities and some significant differences.

4.2 App Distribution through the App/Play Stores

Both Apple and Google provide a marketplace where developers can sell their apps. To publish an app in either of these stores, a developer must configure the app and conform to the requirements established by their sponsors. A significant amount of the work required to publish the app should be done during development. Both Google and Apple have requirements and guidelines for apps that should be incorporated during development. You should read these guidelines prior to development so that you are not doing a lot of rework just to get the app published. Apple is especially meticulous about these rules. Every app is reviewed before it is published, and if your app does not conform, it will be rejected. Google will publish an application that does not meet its store

requirements, but will remove the app from the store later if it finds that the app violates its rules. الفرق بين كوكل وابل بالنسبة للنشر مهم.

When you are confident your app meets publication requirements of the market you are targeting, the process of publishing requires several steps in either market and is greatly facilitated by preparing prior to beginning the process. The preparation is similar for both stores.

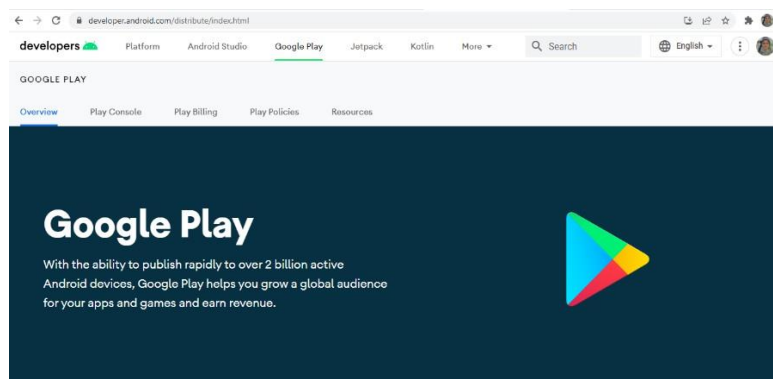
- 1- *You will need to prepare an icon for your app.* Android requires an app icon sized to 512 x 512, whereas iOS requires the icon to be 1024 x 1024 pixels.
- 2- *Both platforms require at least one screenshot of your app for each targeted screen size.* You can provide up to eight screenshots for each targeted Android device and up to four screenshots for each targeted iOS device.
- 3- *For iOS apps, you also have to supply a launch image that matches the resolution for the devices that the app will run on.* The launch image is displayed while the app is loading and is typically a blank image of what the first screen looks like or a splash screen.
- 4- *In addition to preparing screenshots, you should also prepare a description of your app.* This narrative is presented to potential customers, so you should be as clear as possible in describing exactly what the app is designed to do and why it is advantageous for the customer to buy the app.
- 5- *For iOS apps, you will also have to provide a description of any specific conditions or requirements that the tester will need to know to adequately review your app.*
- 6- *Determining what price you want to charge for the app. In Android, you enter this price. In iOS you will be prompted to select from a set of pricing tiers.*

- 7- Determine what general category best describes the app (for example, game, sports, tool, and so on) because you will have to indicate this during the publishing process.
- 8- Determine in which countries your app should be made available.

After you have completed the previous steps, you are ready to publish. Although there are many similarities, each store has its own process and requirements. (والإبل) You may want to look at the specific description online, to get access to these resources for iOS, go to <http://developer.apple.com> . You will need to have a developer ID to access these resources.



To access Android page online resources go to <http://developer.android.com/distribute/index.html>. You do not have to sign in to access this information.



4.3 Android Play Store Distribution

The Google Play Store does not provide any copy protection for your app. Instead, Google provides a Licensing Library, which allows developers to add copy protection to their apps. The Licensing Library needs to be added to the workspace just like the Google Play Services library was. After the library is added, search the developer site for detailed instructions for adding licensing to your app. When you add licensing to an app, each time the user opens the app, the Play Store is queried to determine whether the user bought the app. If the user did not buy the app, the developer can have the app close or take whatever action is deemed appropriate. If you do not use licensing, your app can be copied to other devices quite easily.




After you have added licensing (if you want to), the app needs to be compiled into a signed Android application package (APK) file. Prior to compiling the app, you need to go through the code and remove any logging operations, all debug breakpoints, and address as many of the warnings identified by Eclipse as you think necessary. To create a signed APK (which is required by the Play Store) you need to set a private key. Fortunately, you can do this using the

Export Wizard at the same time you are creating the APK.

To publish apps, you need to sign up as a Google developer. The cost is a one-time fee of \$25, which must be paid from a Google Wallet account. If you are

Android has had sustained investment in enterprise



| Lollipop | Marshmallow | Nougat | Oreo | Pie | Q |
|-----------------------|---|------------------------|--|----------------------------------|--|
| Work profile for BYOD | APIs for Dedicated Devices (single use) | Work profile passwords | Zero-touch enrollment | Shared device/ multi-user | Personal/work calendar sync |
| Fully managed device | Runtime permissions | File-based encryption | Work profiles on company-owned devices | Lock-task mode for multiple apps | Improved tools for work profile provisioning |
| Managed Google Play | Improved cert support | Always-on VPN | Employee privacy disclosures | In-app profile switching | Manual system updates |
| | Data usage tracking | Turn off work mode | | | Enterprise Wi-Fi provisioning |

going to sell your apps, you will need this account for payments. After you have set up all your accounts, you are ready to publish!

tutorial- <https://www.raywenderlich.com/1231-android-app-distribution-from-zero-to-google-play-store>

4.4 iOS App Store Distribution

As a developer, unlike in Android, you don't have to worry about setting up any licensing.

- 1- The first step is to set up your Distribution Certificate and Distribution Provisioning Profile.
- 2- The next step is to set up an entry for your app in iTunes Connect. You will have to use your developer ID to sign in. iTunes Connect is the website that is used to manage many aspects of your app, including seeing reports on how the app is performing.

In iTunes Connect you provide information about the app, pricing, and screenshots prior to being able to upload the app.

- 3- You need to upload your App . When it is complete, you will get a success message.
- 4- Your app is now waiting to be reviewed by Apple.

The app review process is very thorough and may take several weeks to complete.

- 5- After the app is accepted, it will be available for purchase. If it is rejected, Apple will inform you of the reason and give you a chance to correct the problems.

Generally, your first app takes the longest to get reviewed. Future apps typically get a less-thorough review because you have demonstrated that you can conform to the App Store requirements.

4.5 App Distribution for the Enterprise

App distribution within an organization differs between iOS and Android. Generally, distribution is easier because you are not required to conform to the specifications of the Play or App stores.

4.6 Android Enterprise Distribution

Android Enterprise is a Google-led initiative to enable the use of Android devices and apps in the workplace. The program offers APIs and other tools for developers to integrate support for Android into their enterprise mobility management (EMM) solutions. This site provides developers with an overview of the program and the background information required to start building an Android Enterprise solution. Distributing Android apps within an organization is very easy.

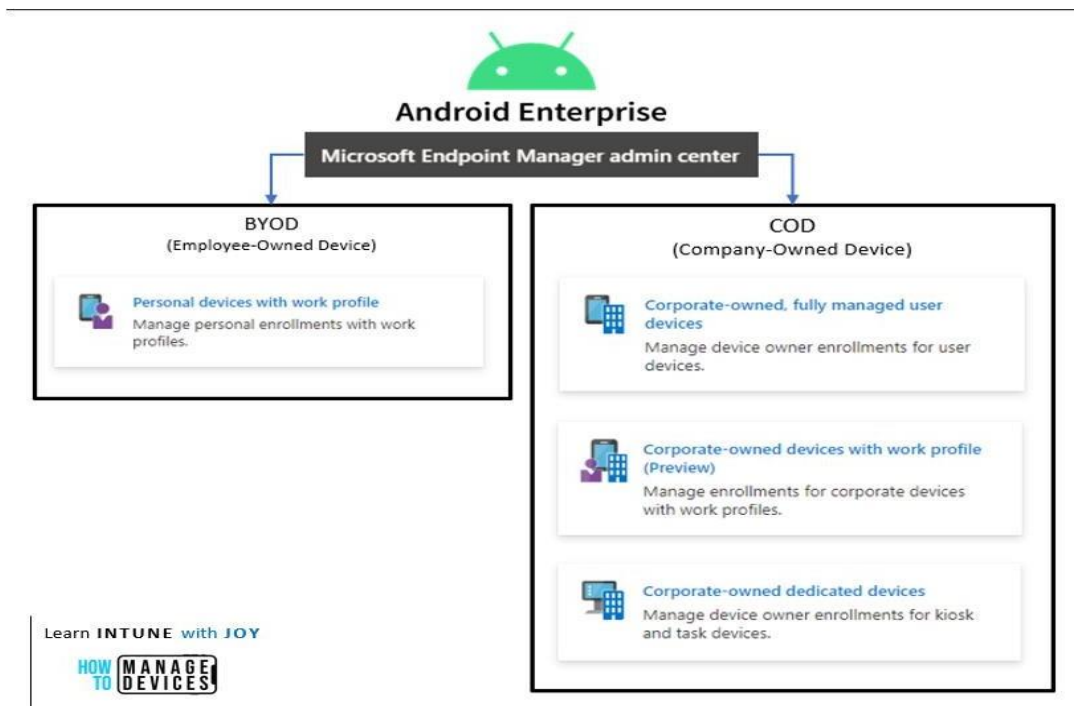
- 1- You prepare an APK just like you did for the Play Store.
- 2- Then you give it to your users.

The easiest way to do this is by sending users an email with the APK attached. If users open the email on an Android device, the device automatically asks if they want to install it. However, for this to occur, users must have set their device to accept apps from unknown sources. To do this, go to the Settings app on the Android device and check the box next to Unknown Sources. Unfortunately, this item is not located in exactly the same spot in Settings on

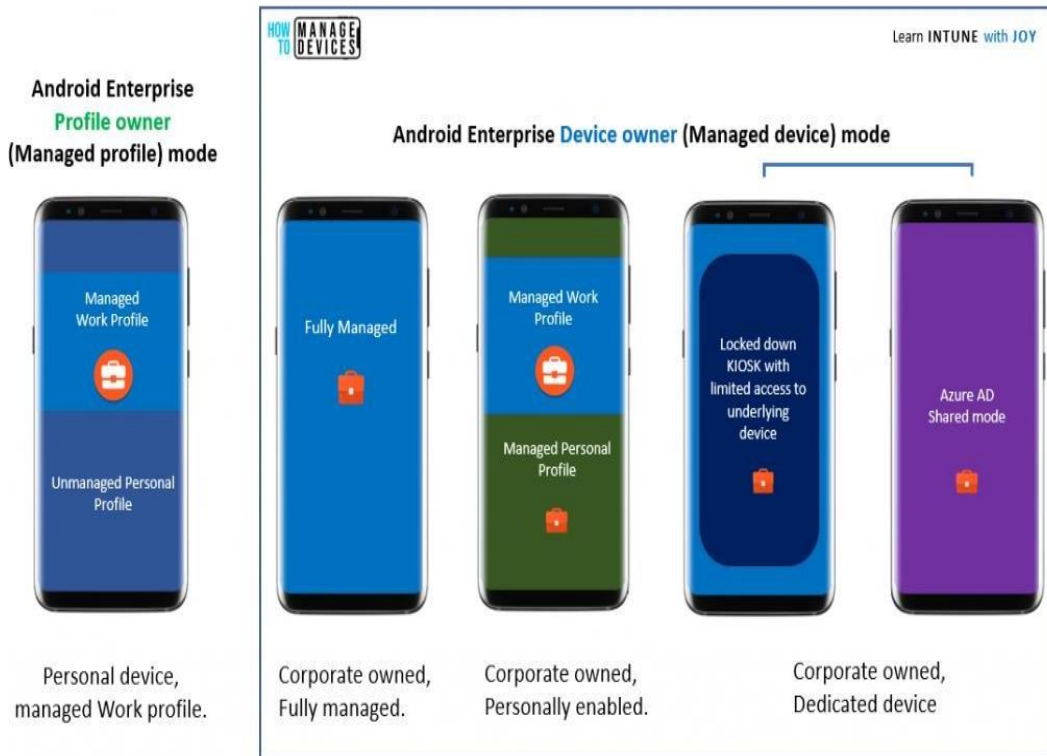
every Android device. Some common locations are under Applications or Security.

Another approach would be to set up an internal website to distribute the app. Again, accepting apps from unknown sources must be checked. Although these are the two most common approaches, because Android is open, you can choose whatever method works for your organization.

Many organizations also implement Mobile Device Management (MDM) solutions to manage their mobile devices and app distribution. These systems can help organizations implement security controls on both devices owned by employees as well as devices owned by the enterprise.



Different modes of Android Enterprise management available with MEM Intune



A visual representation of the different modes of Android Enterprise management available with MEM Intune in terms of endpoints



An overall view of the Android Enterprise management with MEM Intune.

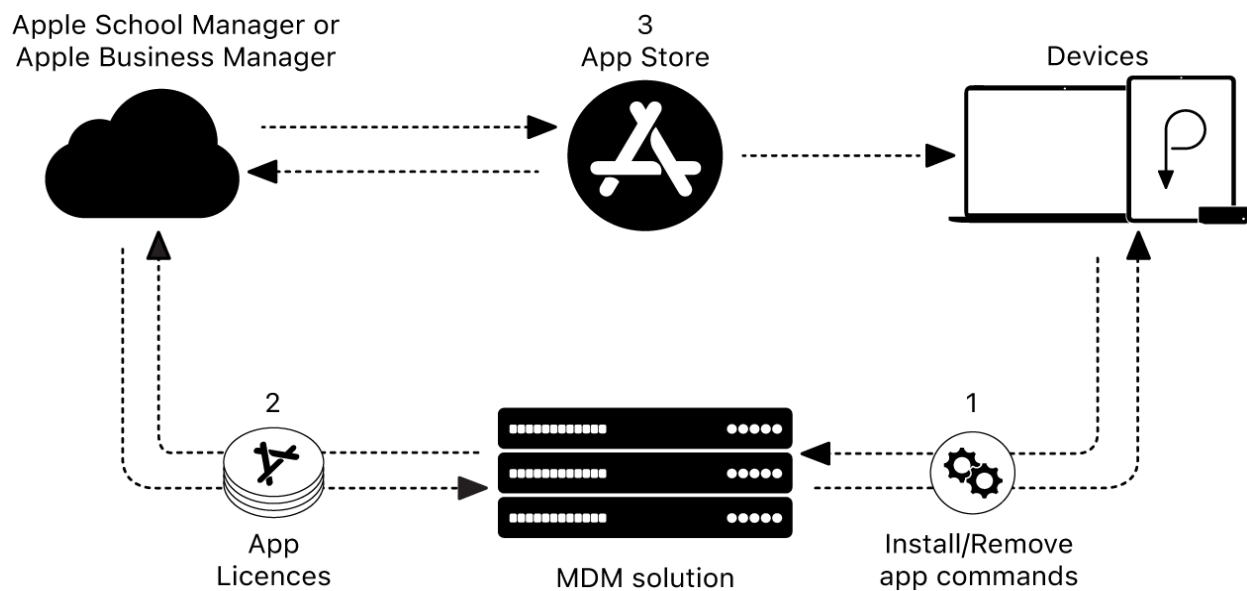
4.7 iOS Enterprise Distribution

Distribution of iOS apps within the organization is a bit more complicated in iOS than it is in Android.

The first step is to get an iOS Enterprise Developer license. The cost of the license is \$299 per year but allows unlimited distribution of apps within the organization. You cannot sell apps in the App Store with this license. Organizations that want to do both internal and public development need both an Enterprise Developer license and an iOS Developer license.

Distributing within the organization requires setting up both an enterprise

distribution certificate and an enterprise distribution provisioning profile. These are then packaged with the app. There is no need to use iTunes Connect with in-house apps. However, the provisioning profile expires after a year. Prior to that time, a new profile must be created, packaged with the app, and redistributed, or the app will stop working. After an app is compiled with the appropriate certificate and profile, it can be distributed through iTunes, or wirelessly from a secure server.



To distribute an iOS enterprise app, you can :

- 1- Create an archive of your app, or have a teammate send you an archived app.
- 2- Distribute your internal app using your company's authorized software distribution mechanism.
- 3- Protect the distribution of the app file because it can be installed on any iOS device.
- 4- Upload the .ipa file and a manifest.plist file to a website somewhere.
- 5- Put a hyperlink that connects to the .plist file.