Session 1

Exploring iPhone SDK

MODULE OBJECTIVE

At the end of this module, you will be able to:

- Describe the iPhone OS architecture and its various versions

SESSION OBJECTIVES

At the end of this session, you will be able to:

- Illustrate the initial stages of app development in the iPhone OS using a suitable example
- Identify the components of the iPhone OS as well as the features of the iPhone simulator
- Describe the architecture and the various versions of the iPhone OS

INTRODUCTION

You are probably well aware of the popularity and admiration Apple phones and devices enjoy worldwide. The Apple App Store was officially launched in July 10, 2008. According to Apple, since then, there have been more than 1 million apps written and made available in the App Store with more than 10 billion downloads. Having an app is like what it was to have a web page in the 1990s—everyone wants one. The developer industry around building apps has practically exploded on the scene. That is probably also the reason you have signed up for this course—you want to learn how to build an app!

This course provides you the necessary skills and knowhow required to develop efficient iOS apps. The first session introduces you to the architecture and versions of iOS phones. As you delve deeper into this subject, you will realize that the best way to learn how to build an app is to actually build an app!

GETTING STARTED WITH AN APP

Every app starts with an idea. You may have your own idea for an app, or perhaps your boss has an idea and you are in charge of making it a reality. For example, you are at a bar or a concert hall and a music band is playing, which you have never heard of and you want to remember the track. Another case could relate to a time when you may be out with friends and are talking about music, and someone mentions a music group or band, which you would like to remember. It would be great to have a place where you can keep all this information about music bands together so that you can go back later when you have time to learn more about them.

Introducing Bands

In this course, you will build an app named **Bands**. It will be a simple app that you can use to take quick notes about your favorite music bands or groups. It will not win an Apple Design Award, but that is not its purpose. It is meant to be a conduit to teach you what it is like to take an idea for an app and translate it into reality, teaching you the skills along the way that you can apply when building your own apps. The figures in this section show what the **Bands** app will look like when you are done.

Figure 1 shows the first screen of the **Bands** app. It is a list of various music bands that are added to the app, alphabetically sorted and indexed. When you tap on a band, you will be able to see its details, as shown in **Figure 2**. Here, you can add notes and pictures of the band, as well as set some properties about the band.

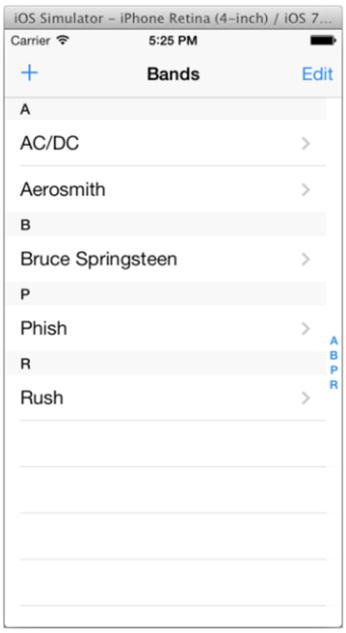

Figure 1: First Screen of Bands App

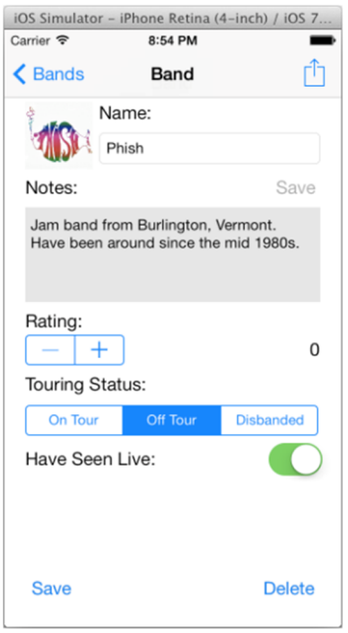

Figure 2: Detail Screen

From the band details, you can perform various actions, such as sharing the band, searching the web, finding local record stores, and searching iTunes, as shown in **Figure 3**. You will also be able to share your bands with your friends on social networking sites, as shown in **Figure 4**.

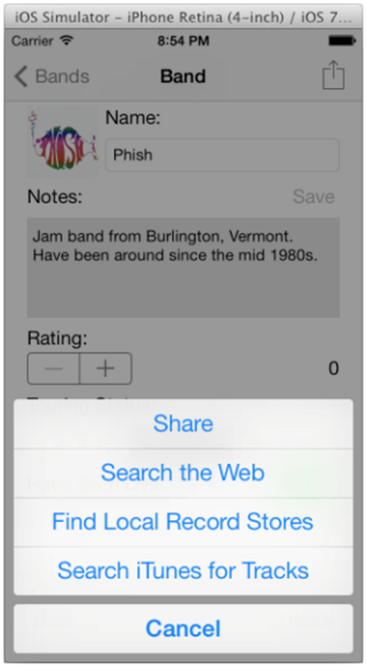

Figure 3: Actions Menu

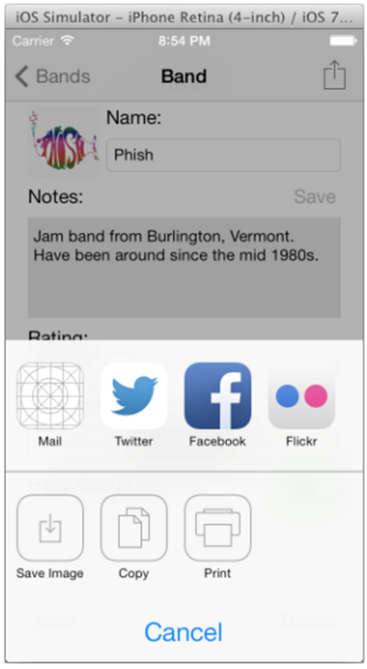

Figure 4: Share via Social Network

You will be able to search the web for more information about a band, as shown in **Figure 5**. You will be able to find local record stores and even search and preview tracks of a band, as shown in **Figure 6** and **Figure 7**, respectively.

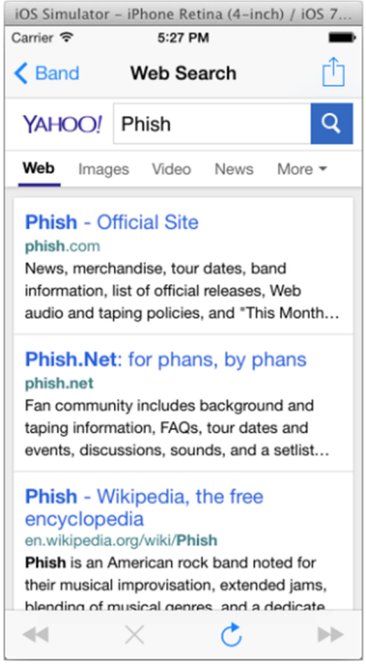

Figure 5: Web Search Option

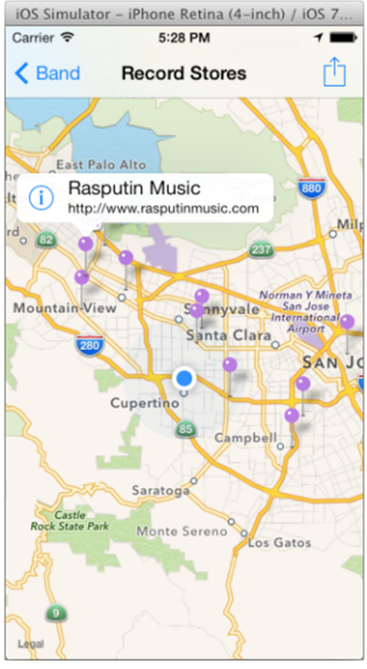

Figure 6: Search Record Stores

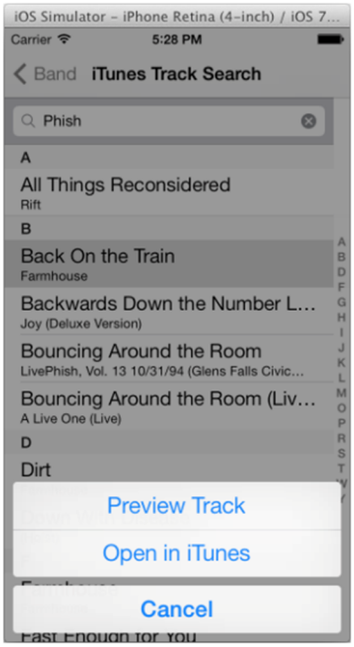

Figure 7: Preview Track and iTunes

Now that you have seen what the **Bands** app can become, it is time to start!

ADDITIONAL KNOWHOW

The idea of mobile computing has been around since the late 1970s. The first real mobile computer was the Psion Organiser, which was released in 1984, followed by the Psion Organiser II in 1986. These early mobile computers looked like calculators.

Mobile computing began to pick up speed in the 1990s–that was when the Personal Digital Assistant (PDA) began to catch on. The phrase Personal Digital Assistant was first used by a Chief Executive Officer (CEO) of Apple, but not the one you may be thinking of. **John Sculley** became the CEO of **Apple** after **Steve Jobs** was forced out. By most accounts, the PDA was not a success and was discontinued in 1998.

Through the rest of the 1990s and early 2000s, mobile computing continued to evolve. There were many popular PDAs such as the **Palm Pilot** as well as the devices running Windows Mobile. They had their users, but they did not have an excited developer base. Smartphones were also coming into their own during this time. They combined the features of a PDA with the capability to make phone calls. **Palm** and **Windows Mobile** along with the **BlackBerry** dominated those early days. That changed in 2007 when Apple announced the iPhone.

The original iPhone went on sale from June 2007. It was an instant hit and went on to become an iconic device. Using an iPhone felt like you were using the future. Though touch screens had been around for more than a decade, the iPhone made you feel like you were interacting with real objects—not just touching buttons on a screen. The original release of the iPhone had one big limitation, though—there was no Software Development Kit (SDK) that developers could use to write native applications for it.

Instead, Steve Jobs recommended to the developers that they should write web applications that feel like native applications. At that time the technology world was in the middle of the **Web 2.0** craze, and therefore the idea did not seem too far-fetched, but it just did not fly with the developers.

By September 2007, the idea of "jail breaking" your iPhone had begun to catch on. Hackers had not only figured out how to unlock the digital security that Apple had in place, but also how to write and run their own applications on the device. However, you had to be pretty brave to do this, because jail breaking voided the warranty of an expensive phone, and writing bad software could easily render the device useless.

In October 2007, Apple reversed the course and announced that an SDK for writing native third-party applications for the iPhone was in its development stage. In March 2008, it was released to developers along with an innovative way to distribute these new applications, which was called the **App Store**. That was when the term **app** became the preferred term for these third-party applications. It was also when the rush to this new development platform, now called the **iPhone OS**, began.

Naming an App

Once you have conceived the idea of an app, the next step would be to give it a name. The name should be catchy and memorable, and must adequately describe what the app does. The name also has to be unique within the App Store. For this purpose, you can search the App Store for names you like and see if there is already an app with that name.

You also need to keep in mind how the app name will look on an actual iPhone or iPad. Typically, you have about 12 characters before the app name is truncated. **Figure 8** shows what a long name looks like. In contrast, native Apple apps are great examples of short names that describe the apps well, as shown in **Figure 9**.

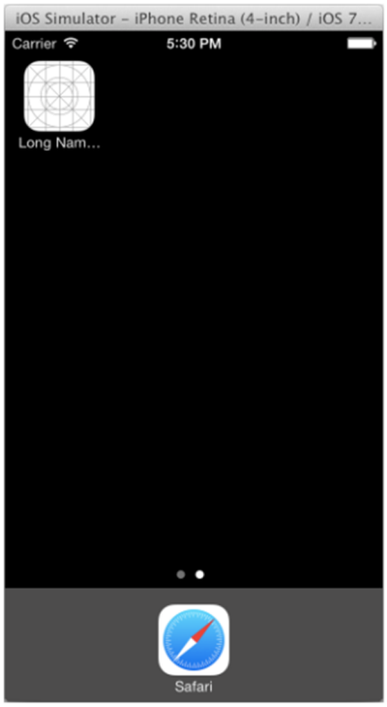

Figure 8: Example of a Long App Name

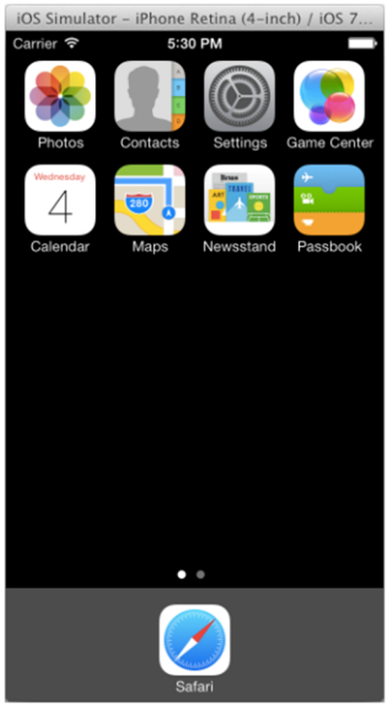

Figure 9: Examples of Short App Names

Searching the available apps in the App Store does not guarantee whether a name is available. The only true way to know is when you attempt to submit an app through iTunes Connect.

Scoping the App

After naming an app, you should scope what the app will do. Your app should be useful, but you also need to keep in mind what is realistic. Having too many features not only takes a lot of development time, but might also make the app difficult to use for the first-time users. On the other hand, having very few features might make your app less useful. The key is to find the right balance of features.

For this purpose, you can list all the things you would like your app to do. For example, the following list includes some ideas that the **Bands** app can incorporate:

1. Add any number of music bands.
2. Take notes about a music band.
3. Take pictures of a music band.
4. Tell your friends about a music band.
5. Search the web for a music band.
6. Find places to buy merchandise related to a music band.
7. Plot all the tour dates of a music band.
8. Retrieve and read reviews from publications and critics of a music band.
9. Have an associated multimedia library for the music of the band.
10. Preview the tracks of a music band.

Once you have created a list of the possible ideas for an app, you need to identify the most valuable and practical ideas to implement. Consider the functions and features that are used in Apple's native apps. If your app is duplicating them, it can cause your app to be rejected when you submit it for approval.

Therefore, from the above list, you can remove the following ideas:

7. **Plot All the Tour Dates of a Music Band:** From where will you get this information? Having a user to manually enter these dates will be cumbersome and will in all probability be seldom used. So this idea can be shelved.
8. **Retrieve and Read Reviews from Publications and Critics of a Music Band:** The same issue applies here, that is, from where will you get the data? In addition, copyright issues are involved. Therefore, this idea can also be dismissed.
9. **Have an Associated Multimedia Library for the Music of the Band:** This task is aptly done by Apple's Music app. Therefore, this idea can also be dismissed.

The rest are all great ideas that can make **Bands** a useful app.

Defining the Features

With a manageable and useful list of ideas, you can now define the features of the app. Defining the features and what it takes to create them helps you to evaluate the time and effort it will take for developing the app. It also helps you to ascertain the end goal of the process, which is a fully functional app.

This course selects the features of the **Bands** app depending on the lessons they can provide for building your own apps. These features include:

1. **Adding a Band:** The **Bands** app should enable users to add a music band. This includes adding the band's name, notes about the band, as well as an optional picture. The app should also enable users to rate a band. The users should be able to record if the band is on tour, off tour, or disbanded. They should be able to mark if they have seen the band's performance live. Access to the camera or a photo library is needed for the band picture. **A simple user interface** that lists all these features in one place and allows the users to edit them seems like a logical choice.
2. **Saving Multiple Bands:** The **Bands** app should be able to save multiple music bands. This means that the app should have some sort of **persistent storage** as well as a way to view all the bands and find them quickly.
3. **Sharing Bands:** When users find a music band they like, they will want to tell others. The **Bands** app should be able to **send emails and text messages**, which are pre-formatted with information the users add about a band. Having the ability to share a band through **social media tools**, such as Facebook, Twitter, and Flickr, would also be useful.
4. **Searching for a Band on the Web:** The users should be able to search the web for more information about music bands in which they are interested. Therefore, the app should have an **in-built light-weight browser**, so that users do not need to use Mobile Safari. The app should also be user-friendly and do the initial search for the users.

5. **Finding Local Record Stores:** If users want to buy tickets for a local show or a poster for a band they like, they might need to find a record store. The **Bands** app should be able to show **a map with pins**, marking all the record stores that are located close to the user's location.
6. **Searching for Tracks:** The users might also like to find new tracks of a band and preview them. The **Bands** app should enable users to **sample new tracks**, and if they like them, they should be able to purchase them through the iTunes store.

Creating a Development Plan

This course describes the development plan for the **Bands** app in subsequent sessions.

To develop an app, you should be familiar with **Objective-C** and **Cocoa Touch**, the language and framework used to create all iOS apps.

Next, you need to learn about Xcode, the **integrated development environment** (IDE) you will use to build the **Bands** app. From there, you will implement each feature of the app, from the easiest to the most complicated.

Finally, when the app's features are complete, you will learn how to deliver your app to **beta testers** before the ultimate goal of getting your app into the App Store.

When an app is uninstalled in a device, the corresponding file and folder in the **Applications** folder are deleted automatically.

COMPONENTS OF THE iPHONE SDK

Now, let's understand the tools and the environment you will need to begin working on your app.

The **iPhone SDK** includes a suite of development tools to help you develop applications for your iPhone and iPod Touch. It includes the following components:

- **Xcode:** It is the IDE that allows you to manage, edit, and debug your projects.
- **Dashcode:** It is the IDE that allows you to develop the web-based iPhone applications and dashboard widgets. **Dashcode** is beyond the scope of this course.
- **iPhone Simulator:** It is a software simulator used to simulate an iPhone on your Mac.
- **Interface Builder:** It provides a visual editor for designing a user interface for an iPhone application.
- **Instrument:** It is an analysis tool to help you optimize your application in real-time.

Xcode can be used to develop both iOS and OS X applications. OS X developers have a separate paid-for program that gives access to the OS X App Store. However, you do not need special hardware to test OS X Apps. Therefore, you can create and test OS X apps with Xcode before you sign up as an OS X developer.

Features of the iPhone Simulator

The iPhone simulator simulates various features of a real iPhone or iPod touch device. You can test the following features on the iPhone simulator:

- Screen rotation: left, top, and right
- Support for gestures:
  - Tap
  - Double tap
  - Touch and hold
  - Swipe

- Flick
- Drag
- Pinch

- Low-memory warning simulations

The iPhone simulator, being a software simulator for the real device, also has its limitations. The following features are not available on the iPhone simulator:

- Obtaining location data (the iPhone simulator returns only a fixed coordinate, such as Latitude 37.3317 North and Longitude 122.0307 West)
- Making phone calls
- Accessing the accelerometer
- Sending and receiving Short Message Service (SMS) messages
- Installing applications from the App Store
- Camera
- Microphone
- Several features of OpenGL ES

Despite these limitations, the iPhone simulator is a useful tool for testing your apps.

It should be noted that the speed of the iPhone simulator is more tightly coupled to the performance of your Mac, as opposed to how the actual device performs. Therefore, it is important that you test your application on a real device rather than rely exclusively on the iPhone simulator for testing.

ADDITIONAL KNOWHOW

Apple offers a separate Enterprise program for large corporations that want to develop apps in-house and distribute them internally. For more details, you can refer to *http://developer.apple.com/programs/enterprise.*

ARCHITECTURE OF THE iPHONE OS

Although this course does not explore the innards of the iPhone OS, it is important to understand some of the important points of the iPhone OS architecture. **Figure 10** shows the different abstraction layers that make up the Mac OS X and the iPhone OS.

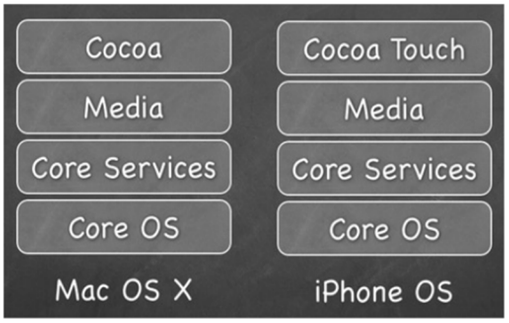

Figure 10: Abstraction Layers of Mac OS X and iPhone OS

The iPhone OS is architecturally very similar to the Mac OS X, except that the topmost layer is Cocoa Touch for iPhone instead of Cocoa Framework.

Now, let's discuss each abstraction layer in **iPhone OS**.

Core OS Layer

In both MAC and iPhone, the bottom layer is the **Core OS**, which is the foundation of their operating system. It is responsible for memory management, file systems, networking, and other OS tasks. It also interacts directly with the hardware.

The Core OS layer consists of the following components:

- OS X Kernel
- BSD
- Security
- Keychain
- File System
- Mach
- Sockets
- Power Management
- Certificates
- Bonjour

Core Services Layer

The **Core Services layer** provides fundamental access to iPhone OS services. It provides an abstraction layer over the services provided in the Core OS layer.

The Core Services layer consists of the following components:

- Collections
- Networking
- SQLite
- Net Services
- Preferences
- Address Book
- File Access
- Core Location
- Threading
- URL Utilities

Media Layer

The **Media layer** provides multimedia services that you can use in your iPhone applications. This layer consists of the following components:

- Core Audio
- Audio Mixing
- Video Playback
- PDF
- Core Animation
- OpenGL
- Audio Recording
- JPG, PNG, TIFF
- Quartz
- OpenGL ES

Cocoa Touch Layer

The **Cocoa Touch layer** provides an abstraction layer to expose the various libraries for programming the iPhone and iPod Touch, such as:

- Multi-touch events
- Accelerometer
- Localization
- Web Views
- Image Picker
- Multi-touch controls
- View Hierarchy
- Alerts
- People Picker
- Controllers

Apple delivers most of its system interfaces in special packages called **frameworks**. A **framework** is a software library that provides specific functionalities, such as:

- Access to the centralized database
- A user interface (UI) to display the stored contacts
- Low-level C application programming interfaces (APIs) for audio recording and playback, and managing hardware
- An interface for the iPhone OS-supplied audio processing
- Access to network services and configurations
- Generalized solution for object graph management
- Abstraction for common data types and Unicode strings
- Facilitation for playing movies and audio files

The iPhone SDK consists of the following frameworks, as shown in **Table 1**, grouped by functionalities:

Table 1: iPhone SDK Frameworks

FRAMEWORK NAME	DESCRIPTION
AddressBook.framework	Provides access to the centralized database for storing a user's contacts.
AddressBookUI.framework	Provides the UI to display the contacts stored in the Address Book database.
AudioToolbox.framework	Provides low-level C APIs for audio recording and playback, as well as managing the audio hardware.
AudioUnit.framework	Provides the interface for iPhone OS-supplied audio processing plug-ins in your application.
AVFoundation.framework	Provides low-level C APIs for audio recording and playback, as well as for managing the audio hardware.
CFNetwork.framework	Provides access to network services and configurations, such as HTTP, FTP, and Bonjour services.
CoreAudio.framework	Declares data types and constants used by other Core Audio interfaces.
CoreData.framework	Provides a generalized solution for object graph management in your application.
CoreFoundation.framework	Provides abstraction for common data types, Unicode strings, XML, URL resource, and so on.
CoreGraphics.framework	Provides C-based APIs for 2D rendering; based on the Quartz drawing engine.
CoreLocation.framework	Provides location-based information using a combination of GPS, cell ID, and Wi-Fi networks.
ExternalAccessory.framework	Provides a way to communicate with accessories.
Foundation.framework	Provides the foundation classes for Objective C, such as `NSObject`, basic data types, operating system services, and so on.
GameKit.framework	Provides networking capabilities to games; commonly used for peer-to-peer connectivity and in-game voice feature.
IOKit.framework	Provides capabilities for driver development.
MapKit.framework	Provides an embedded map interface for your application.
MediaPlayer.framework	Provides facilities for playing movies and audio files.
MessageUI.framework	Provides a view-controller–based interface for composing e-mail messages.
MobileCoreServices.framework	Provides access to standard types and constants.
OpenAL.framework	Provides an implementation of the OpenAL specification.
OpenGLES.framework	Provides a compact and efficient subset of the OpenGL API for 2D and 3D drawing.
QuartzCore.framework	Provides ability to configure animations and effects and then render those effects in hardware.

FRAMEWORK NAME	DESCRIPTION
Security.framework	Provides the ability to secure your data and control access to software.
StoreKit.framework	Provides in-app purchase support for applications.
SystemConfiguration.framework	Provides the ability to determine network availability and state on device.
UIKit.framework	Provides the fundamental objects for managing an application's UI.

VERSIONS OF iPHONE OS

The "iPhone OS" was officially named iOS in 2010. Its first appearance on the original iPhone included many of the default apps you still see today, such as the Safari browser, Mail app, Calendar, Notes, Clock, Photos, and Camera. But it was the second version that the operating system really became the game-changing device you know today.

Table 2 describes iOS's history with the first iOS version to the most recent iOS update:

Table 2: iOS Versions

Version	Added
1.0	Initial release of iPhone
1.1	Additional features and bug fixes for 1.0
2.0	Released with iPhone 3G; comes with App Store
2.1	Additional features and bug fixes for 2.0
2.2	Additional features and bug fixes for 2.1
3.0	Third major release of iPhone OS
4.0	Support iPhone 3G and iPod Touch (second generation, MB & MC model)
5.0	Released for iPhone 3GS, iPhone 4 (GSM and CDMA), iPhone 4S, iPod Touch (third and fourth generation)
5.1	Supported for the iPad (first generation) and iPod Touch (third generation)
6.0	Support with iTunes and over-the-air updates
6.1	Supported for the iPhone 3GS and iPod Touch (fourth generation)
7.0	Support for older devices, specifically the iPhone 3GS and iPod Touch (fourth generation)
8.0	Support for older devices, specifically iPhone 6 Plus or later and iPod Touch (fifth generation)

For a detailed description of the features and drawbacks in each release of iOS (including iOS 8), check the following link: *http://en.wikipedia.org/wiki/IPhone_OS_version_history.*

Currently, there are more than seven iOS versions in the market. Some applications are not supported by the latest iOS because they were designed as per the latest iOS available at the time of their development. Therefore, while developing any application, you should ensure that it is supported by the future versions of iOS.

During the lab hour of this session, you will identify and play with the various APIs of the iOS 8 environment in an Apple device.

Cheat Sheet

- All apps start with an idea. After thinking about an idea, you should give a name to the app. The name should be unique within the App Store. You should also check the look and feel of the app on an actual iPhone or iPad and limit the number of characters to 12.
- After naming the app, you should scope the features that your app should provide. Your app should be relevant and realistic.
- You should define the selected features and identify their purpose to evaluate the time and effort required for developing an app.
- The iPhone SDK includes the following components:
  - Xcode
  - Dashcode
  - iPhone simulator
  - Interface Builder
  - Instruments
- The iPhone simulator simulates various features of a real iPhone or iPod touch device. However, it has its own set of limitations. It does not allow you to obtain location data, make phone calls, access the accelerometer, and send and receive SMS.
- The iPhone OS architecture is composed of the following layers:
  - Core OS layer
  - Core Services layer
  - Media layer
  - Cocoa Touch layer
- The "iPhone OS" was officially named iOS in 2010. The current version of iOS is 8, which supports iPhone 6 Plus or later and iPod Touch (fifth generation) devices.