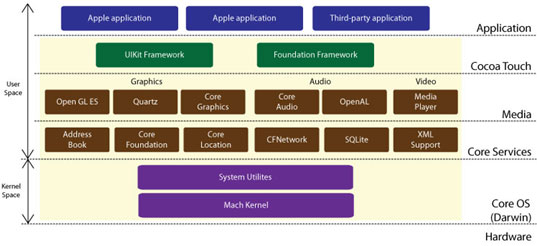
About the iOS Technologies

* iOS is an operating system from Apple Inc that runs on iPad, iPhone, and iPod touch devices.
* This operating system manages the device hardware as well as provides the technologies required to implement native apps to iPad, iPhone, iPod touch and MAC OS.
* This operating system also provides various system apps, such as Phone, Mail, and Safari that provide standard system services to the user.
* The iOS Software Development Kit (SDK)contains the tools and interfaces needed to develop, install, run, and test native apps that appear on an iOS device’s Home screen.
* Native apps are built using the iOS system frameworks and Objective-C language and run directly on iOS.
* Unlike web apps, the native apps are installed physically in user’s device and this application always available to the user, even when the device is in Airplane mode.

**Note:** It is possible to create web apps using a combination of HTML, cascading style sheets (CSS), and JavaScript code and these web Apps are run in Safari browser.

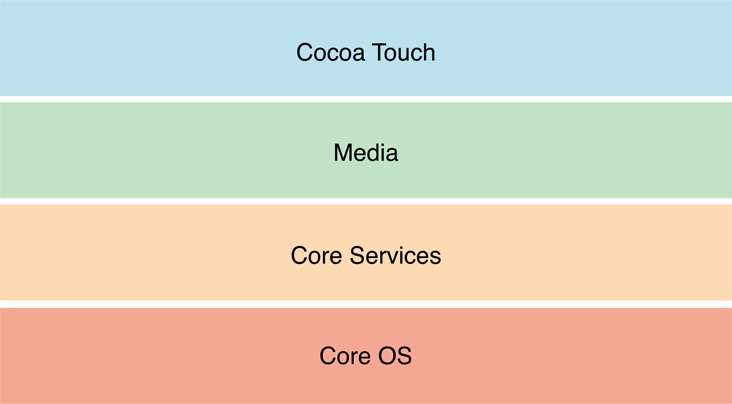
### The iOS Architecture Is Layered



**Figure 1: iOS Architecture**

**Courtesy: www.blog.inf.ed.ac.uk**

* As shown in the highest level, iOS acts as an intermediary between the hardware and the apps.
* Apps will not have the ability to talk to the hardware directly instead they communicate to the hardware through a set of well-defined system interfaces.
* The presence of interfaces makes it easy to write apps that work consistently on devices with different hardware capabilities.
* The implementation of iOS technologies can be viewed as a set of layers, which are shown in Figure 2.
* The Lower layers contain fundamental services and technologies.
* The Higher-level layers build upon the lower layers and provide more sophisticated services and technologies.

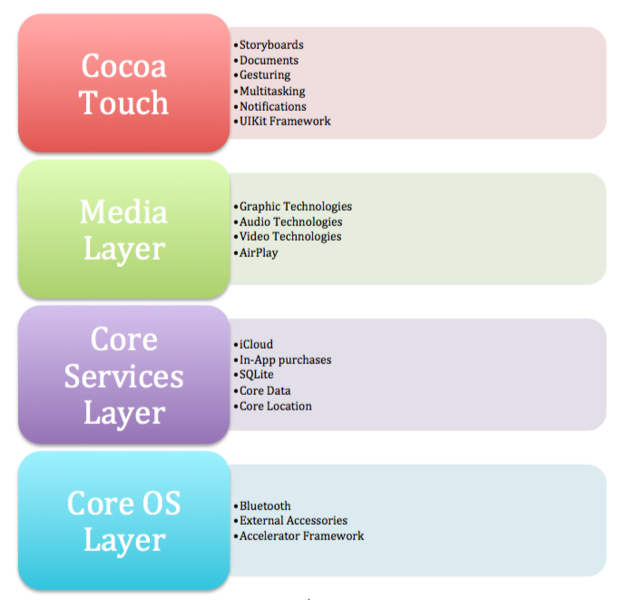


**Figure 2: Layers of iOS**

**Courtesy: Apple Documentation**

* While writing the applications its recommended to use high-level frameworks instead of low-level frameworks.
* The higher-level frameworks are there to provide object-oriented abstractions for lower-level constructs.
* By using high-level frameworks we can avoid of writing our own code, instead we can use the high-level frameworks to avoid writing code and it also facilitates us to write complex features, such as sockets and threads.
* We can use low-level frameworks also only when if we not find these features in high-level.

The Layers of iOS



**Figure 3: All layers of iOS**

**Courtesy: www.** **wordpress.com**

**The above Figure 3 shows different layers in an iOS. It consists of following Layers:**

* As we know iOS is an operating system for the iPhone’s (an Apple Inc Product).
* It is derived from core OS X technologies (an OS for MAC pc).
* The iOS includes the OS X Kernel, BSD Sockets for Networking and objective-C and C/C++ compilers for native performance. iOS 9.0 is the latest release of iOS on June 8, 2015

Cocoa Touch Layer

* The Cocoa Touch layer consists of fundamental frameworks for building iOS apps.
* These frameworks define the appearance of the iOS app.
* This framework also provide the basic app infrastructure and support for fundamental technologies like multitasking, touch-based input, push notifications, and many high-level system services.
* While designing the apps the developer first investigate the technologies in this layer to see if they already exists according to our needs and requirements.

## High-Level Features

The key technologies that are available in the Cocoa Touch layer are described below.

### App Extensions

* The iOS 8 provides custom facility to extend the existing Apps by providing the frameworks.
* For example, we might supply an app extension that helps users post content to his/her social sharing website.
* After users install and enable this extension, they can choose it when they tap the Share button in their current app.
* The custom sharing extension provides the code that accepts, validates, and posts the users content.
* The system lists the extension in the sharing menu and instantiates it when the user chooses it.
* By using Xcode, We create an app extension by adding a preconfigured app extension target to an app.
* After a user installs an app that contains an extension, the extension can be enabled by the user in the Settings app.
* When the user is running other apps, the system makes the enabled extension available in the appropriate system UI, such as the Share menu.

**iOS supports app extensions for the following areas, which are known as extension points:**

* Share. Share content with social websites or other entities.
* Action. Perform a simple task with the current content.
* Widget. Provide a quick update or enable a brief task in the Today view of Notification Center.
* Photo editing. Perform edits to a photo or video within the Photos app.
* Document provider. Provide a document storage location that can be accessed by other apps. Apps that use a document picker view controller can open files managed by the Document Provider or move files into the Document Provider.
* Custom keyboard. Provide a custom keyboard that the user can choose in place of the system keyboard for all apps on the device.

### Handoff

* The Handoff feature allows the user to run same application in different devices like iPhone to Mac etc.
* Handoff is a feature in OS X and iOS that extends the user experience of continuity across devices.
* The Handoff feature enables users to begin an activity on one device, then switch to another device and resume the same activity on the another device.
* For example, a user who is browsing a long article in Safari moves to an iOS device that's signed into the same Apple ID and the same webpage automatically opens in Safari on iOS, with the same scroll position as on the original device. Handoff makes this experience as seamless (discontinuous) as possible.
* To make this Handoff, an app adopts a small API in Foundation.
* Each ongoing activity in one app is represented by in user activity object that contains the data needed to resume an activity on another device to make it as discontinues process.
* When the user chooses to resume that activity, the object is sent to the resuming device.
* Each user activity object has a delegate object that is invoked to refresh the activity state at opportune times, such as just before the user activity object is sent between the devices.
* If continuing an activity requires more data than is easily transferred by the user activity object, the resuming app has the option to open a stream to the originating app.
* Document-based apps automatically support activity continuation for users working with iCloud-based documents.

### Document Picker

* This document picker view controller feature allows the users to access to files outside the application’s sandbox.
* This is a simple mechanism for sharing documents between apps.
* This feature also enables more complex workflows, because the users can edit a single document with multiple apps.
* The document picker lets us to access files from a number of document providers.
* For example, the iCloud document provider grants access to documents stored inside another app’s iCloud container. Third-party developers can provide additional document providers by using the Storage Provider extension.

### AirDrop

* The AirDrop feature lets users to share photos, documents, URLs, and other kinds of data with nearby devices.
* It supports for sending files from one iOS device to other iOS devices by using AirDrop, it is built into the existing [UIActivityViewController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIActivityViewController_Class/index.html#//apple_ref/occ/cl/UIActivityViewController) class.
* This class displays different options for sharing the content to another device that we specify. If you are not yet using this class, we should consider it to adding it to our interface.

**To receive files that are sent using AirDrop, The app must do the following:**

* First declare the support for the appropriate document types in Xcode. (Xcode will adds the appropriate keys to the app’s Info.plist file.)
* The system uses this information to determine whether your app can open a given file.
* To implement this in [application: openURL: sourceApplication: annotation:](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html#//apple_ref/occ/intfm/UIApplicationDelegate/application:openURL:sourceApplication:annotation:) method in app delegate. (The system calls this method when a new file is received.)
* Files sent to the apps are placed in the Documents/Inbox directory of your app’s home directory. If you plan to modify the file, you must move it out of this directory before doing so. (The system allows your app to read and delete files in this directory only.)
* Files stored in this directory are encrypted using data protection, so you must be prepared for the file to be inaccessible if the device is currently locked.

### TextKit

* This TextKit provides a full-featured, high-level set of classes for handling text and fine typography.
* By using TextKit technique, we can lay out styled text into paragraphs, columns, and pages; we can flow text around arbitrary regions such as graphics; and you can use it to manage multiple fonts.
* If we were considering the using of Core Text to implement text rendering, we should consider TextKit instead.
* TextKit is integrated with all UIKit text-based controls to enable apps to create, edit, display, and store text more easily—and with less code than was previously possible in iOS.
* TextKit comprises new UIKit classes, along with extensions to existing classes, including the following:
* The [NSAttributedString](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSAttributedString_Class/index.html#//apple_ref/occ/cl/NSAttributedString) class has been extended to support new attributes.
* The [NSLayoutManager](https://developer.apple.com/library/ios/documentation/UIKit/Reference/NSLayoutManager_Class_TextKit/index.html#//apple_ref/occ/cl/NSLayoutManager) class generates glyphs and lays out text.
* The [NSTextContainer](https://developer.apple.com/library/ios/documentation/UIKit/Reference/NSTextContainer_Class_TextKit/index.html#//apple_ref/occ/cl/NSTextContainer) class defines a region where text is laid out.
* The [NSTextStorage](https://developer.apple.com/library/ios/documentation/UIKit/Reference/NSTextStorage_Class_TextKit/index.html#//apple_ref/occ/cl/NSTextStorage) class defines the fundamental interface for managing text-based content.

### UIKit Dynamics

* The Apps can now specify dynamic behaviors for [UIView](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIView_Class/index.html#//apple_ref/occ/cl/UIView) objects and for other objects that conform to the [UIDynamicItem](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIDynamicItem_Protocol/index.html#//apple_ref/occ/intf/UIDynamicItem) protocol. (Objects that conform to this protocol are called dynamic items.)
* The Dynamic behaviors offer a way to improve the user experience for the app by incorporating real-world behavior and characteristics into your app’s user interface.

**UIKit dynamics supports the following types of behaviors:**

* The [**UIAttachmentBehavior**](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIAttachmentBehavior_Class/index.html#//apple_ref/occ/cl/UIAttachmentBehavior) object specifies a connection between two dynamic items or between an item and a point. When one item (or point) moves, the attached item also moves. The connection is not completely static, though. An attachment behavior has damping and oscillation properties that determine how the behavior changes over time.
* The [**UICollisionBehavior**](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UICollisionBehavior_Class/index.html#//apple_ref/occ/cl/UICollisionBehavior) object lets dynamic items participate in collisions with each other and with the behavior’s specified boundaries. The behavior also lets those items respond appropriately to collisions.
* The [**UIGravityBehavior**](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIGravityBehavior_Class/index.html#//apple_ref/occ/cl/UIGravityBehavior) object specifies a gravity vector for its dynamic items. Dynamic items accelerate in the vector’s direction until they collide with other appropriately configured items or with a boundary.
* The [**UIPushBehavior**](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPushBehavior_Class/index.html#//apple_ref/occ/cl/UIPushBehavior) object specifies a continuous or instantaneous force vector for its dynamic items.
* The [**UISnapBehavior**](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UISnapBehavior_Class/index.html#//apple_ref/occ/cl/UISnapBehavior) object specifies a snap point for a dynamic item. The item snaps to the point with a configured effect. For example, a dynamic item can snap to the point as if it were attached to a spring.
* Dynamic behaviors become active when we add them to an animator object, which is an instance of the [**UIDynamicAnimator**](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIDynamicAnimator_Class/index.html#//apple_ref/occ/cl/UIDynamicAnimator) class. The animator provides the context in which dynamic behaviors execute. A given dynamic item can have multiple behaviors, but all of those behaviors must be animated by the same animator object.

### Multitasking

* Multitasking means running more than one task simultaneously.
* In iOS the battery life is an important consideration for users of iOS devices and the multitasking model in iOS is designed to maximize battery life while giving apps time when they need to do critical work.
* When the user presses the Home button, the foreground app shifts to a background execution context. If the app has no more work to do, it is suspended from active execution and put into a "freeze-dried” state, where it remains in memory but does not execute any code.
* In iOS Apps that do need specific types of work can ask the system for background execution time. For example:
* An app can request a finite amount of time to complete some important task.
* An app that supports specific services (such as audio playback) can request time to provide those services.
* An app can use local notifications to generate user alerts at designated times, whether or not the app is running.
* An app can download content periodically from the network.
* An app can download content in response to a push notification.

### Auto Layout

* In iOS Auto layout feature helps us to build dynamic interfaces with very less amount of code.
* By using Auto Layout, we define the rules for how to lay out the elements in the user interface.
* These rules express a larger class of relationships and are more understandable to use than the springs and struts model that was used previously. For example, you can specify that a button always be 20 points from the left edge of its parent view.

The entities used in Auto Layout are Objective-C objects called constraints. Constraints provide several benefits:

* They support localization through the swapping of strings alone, instead of requiring you to update your layouts.
* They support mirroring of user interface elements for right-to-left languages, such as Hebrew and Arabic.
* They promote a better separation of responsibilities between objects in the view and controller layers.

A view object usually has values for its standard size, its positioning within its superview, and its positioning relative to its sibling views. A view controller can override these values if something nonstandard is required.

### Storyboards

* The Storyboards concepts are the recommended way to design the app’s user interface.
* Storyboards let us to design our entire user interface in one place so that we can see all of the views and view controllers and understand how they work together.
* An important part of storyboards is the ability to define nonstop transitions from one view controller to another.
* These transitions allow us to capture the flow of the user interface in addition to the content.
* We can define these transitions in picture format, in Xcode, or initiate them programmatically.
* We can use a single storyboard file to store all of the app’s view controllers and views, or we can use multiple view storyboards to organize portions of your interface.
* During App build time, Xcode takes the contents of the storyboard file and divides it into discrete pieces that can be loaded individually for better performance.
* The app never needs to manipulate these pieces directly. The UIKit framework provides convenience classes for accessing the contents of a storyboard from our code.

### UI State Preservation

* State preservation allows the Apps to be run continuously if it faces some problems like memory, if it faces problems regarding on memory it closes the background application programs.
* When an app moves from the foreground to the background, it can preserve the state of its views and view controllers.
* During its next launch cycle, it can use that preserved state information to restore its views and view controllers to their previous configurations and make it appear as if the app had never quit.

### Apple Push Notification Service

* The Push Notification service provides alert users about new information, even when your app is not running.
* Using this service, you can push text notifications, add a badge to your app icon, or trigger audible alerts on user devices at any time.
* By using these messages users can come to know that they should open their app to receive the related information.
* In iOS 7, we can even push silent notifications to let their app to know that new content is available to download.
* According to design standpoint, there are two parts to making push notifications to work for iOS apps. First, the app must request the delivery of notifications and process the notification data once it is delivered. Second, we need to provide a server-side process to generate the notifications in the first place.
* This process lives on their own local server and works with Apple Push Notification Service to trigger the notifications.

### Local Notifications

* These features of Local notifications cover the existing push notification mechanism by giving apps a way to generate the notifications locally instead of relying on an external server.
* The Apps running in the background can use local notifications as a way to get a user’s attention to the App when important are events happening.
* For example, a navigation app running in the background can use local notifications to alert the user when it is time to make a turn.
* Apps can also schedule the delivery of local notifications for a future date and time and have those notifications delivered even if the app is not running.
* The advantage of local notifications is that they are independent of our app.
* After a notification is scheduled, the system manages the delivery of it. The app does not even have to be running when the notification is delivered.

### Gesture Recognizers

* The feature of Gesture recognizers is to detect common types of gestures, such as swipes and pinches in the app’s views.
* Because they use the same heuristics as the system for detecting gestures, gesture recognizers offer a consistent behavior for your apps.
* To use one, you attach the gesture recognizer to the view and give it an action method to perform when the gesture occurs.
* The gesture recognizer does the difficult work of tracking the raw touch events and determining when they constitute the intended gesture.
* All gesture recognizers are based on the [UIGestureRecognizer](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIGestureRecognizer_Class/index.html#//apple_ref/occ/cl/UIGestureRecognizer) class, which defines the basic behavior.
* UIKit supplies standard gesture recognizer subclasses to detect taps, pinches, pans, swipes, rotations.
* We can also tailor the behavior of most gesture recognizers to the app’s needs.
* For example, you can tell a tap gesture recognizer to detect a specific number of taps before calling your action method.

### Standard System View Controllers

* The Many system frameworks define view controllers for standard system interfaces.
* As a developer first we have to check the view that is going to use is already provided in view controllers rather than create our own.
* Apple Inc recommended using these view controllers in custom apps to present a consistent user experience.
* Whenever we need to perform one of the following tasks, you should use a view controller from the corresponding framework:
* **Display or edit contact information.** Apple Inc recommended using the view controllers in the Address Book UI framework.
* **Create or edit calendar events**. Apple Inc recommended using the view controllers in the EventKit UI framework.
* **Compose an email or SMS message.** Apple Inc recommended using the view controllers in the Message UI framework.
* **Open or preview the contents of a file.** Apple Inc recommended using the [UIDocumentInteractionController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIDocumentInteractionController_class/index.html#//apple_ref/occ/cl/UIDocumentInteractionController) class in the UIKit framework.
* **Take a picture or choose a photo from the user’s photo library.** Apple Inc recommended using the [UIImagePickerController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIImagePickerController_Class/index.html#//apple_ref/occ/cl/UIImagePickerController) class in the UIKit framework.
* **Shoot a video clip.** Apple Inc recommended using the [UIImagePickerController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIImagePickerController_Class/index.html#//apple_ref/occ/cl/UIImagePickerController) class in the UIKit framework.

## Cocoa Touch Frameworks

**The Cocoa Touch layer and the services they offer the following frameworks.**

### Address Book UI Framework

* The Address Book UI framework feature in an Objective-C programming interface that is used to display standard system interfaces for creating new contacts and for editing and selecting existing contacts.
* This framework feature simplifies the work needed to display contact information in the app and also makes sure that the app uses the same interfaces as other apps, thus ensuring consistency across the platform.

### EventKit UI Framework

* The EventKit UI framework feature provides view controllers for presenting the standard system interfaces for viewing and editing calendar-related events.
* This framework builds upon the event-related data in the EventKit framework.

### iAd Framework

* The iAd framework feature lets us to deliver banner-based advertisements from our app.
* The Advertisements are incorporated into standard views that we integrate into our user interface and present them when we want.
* The views themselves work with Apple’s iAd Service to automatically handle all the work associated with loading and presenting rich media ads and responding to taps in those ads.

### MapKit Framework

* The MapKit framework feature provides a scrollable map that we can used to incorporate into the app’s user interface.
* Not only to just displaying a map, we can use the framework interfaces to customize the map’s content and appearance.
* We can flag points of interest using annotations, and we can use custom overlays to design differently of our own content with the map content
* For example, we might use an overlay to draw a bus route, or use annotations to highlight nearby shops and restaurants.
* In addition to displaying the maps, the MapKit framework integrates with the Maps app and Apple’s map servers to facilitate directions.
* From this Maps app, users can delegate the providing of directions to any app that supports directions.
* Apps that provide specialized types of directions, such as subway routes, can register to provide those directions when asked.
* The Apps which implemented this framework can also request walking and driving directions from Apple servers and merge that route information with their custom directions to provide a complete point-to-point experience for the user.

### Message UI Framework

* The Message UI framework feature provides support for composing email or SMS messages from the app.
* The composition support consists of a view controller interface that we present in the app.
* We can populate the fields of this view controller to set the recipients, subject, body content, and any attachments that the user want to include with the message.
* After presenting the view controller, the user then has the option of editing the message before sending it.

### Notification Center Framework

* The Notification Center framework feature provides support for creating widgets that display information in Notification Center.

### PushKit Framework

* The PushKit framework feature provides registration support for VoIP apps (like Skype, Viber etc).
* This framework replaces the previous APIs for registering VoIP apps.
* Instead of keeping a continuous connection open, and thus emptying the device’s battery, the app can use this framework to receive push notifications when there is an incoming call.

### Twitter Framework

* This Twitter framework has been replaced by using the Social framework.
* The Social framework supports a UI for generating tweets and support for creating URLs to access the Twitter service.

### UIKit Framework

* The UIKit framework provides final infrastructure for implementing graphical, event-driven apps in iOS
* UIKit framework including the following:
* Basic app management and infrastructure, including the app’s main run loop
* User interface management, including support for storyboards and nib files
* A view controller model to encapsulate the contents of the user interface
* Objects representing the standard system views and controls
* Support for handling touch- and motion-based events
* Support for a document model that includes iCloud integration; see [Document-Based App Programming Guide for iOS](https://developer.apple.com/library/ios/documentation/DataManagement/Conceptual/DocumentBasedAppPGiOS/Introduction/Introduction.html#//apple_ref/doc/uid/TP40011149)
* Graphics and windowing support, including support for external displays
* Multitasking support
* Printing support
* Support for customizing the appearance of standard UIKit controls
* Support for text and web content
* Cut, copy, and paste support
* Support for animating user-interface content
* Integration with other apps on the system through URL schemes and framework interfaces
* Accessibility support for disabled users
* Support for the Apple Push Notification service;
* Local notification scheduling and delivery
* PDF creation
* Support for using custom input views that behave like the system keyboard
* Support for creating custom text views that interact with the system keyboard
* Support for sharing content through email, Twitter, Facebook, and other services
* In addition to providing the fundamental code for building your app, UIKit also incorporates support for some device-specific features, such as the following:
* The built-in camera (where present)
* The user’s photo library
* Device name and model information
* Battery state information
* Proximity sensor information
* Remote control information from attached headsets

Media Layer

* This Media layer contains the graphics, audio, and video technologies that is used to implement multimedia experiences in mobile apps.
* The technologies are available in this layer makes easy to build apps that look and sound great by providing the graphics, audio, and video technologies to the app.

## Graphics Technologies

* Graphics is most important thing in all application to attract the people towards App.
* Graphics Technology in iOS provides numerous technologies to help put high quality graphics, your custom art and graphics onscreen.
* This can be possible by using UIKit view architecture to make it easy to deliver content.
* By using Graphics Technology we can use high quality graphics interface or we can build our own custom views by using any of the following technologies.

|  |  |
| --- | --- |
| **Technology** | **Description** |
| UIKit graphics | UIKit defines high-level support for drawing images and Bezier paths and for animating the content of your views |
| Core Graphics framework | Core Graphics (also known as Quartz) is the native drawing engine for iOS apps and provides support for custom 2D vector- and image-based rendering. |
| Core Animation | Core Animation (part of the Quartz Core framework) is a foundational technology that optimizes the animation experience of your apps. |
| Core Image | Core Image provides advanced support for manipulating video and still images in a nondestructive manner. |
| OpenGL ES and GLKit | OpenGL ES handles advanced 2D and 3D rendering using hardware-accelerated interfaces |
| Metal | Metal provides extremely low-overhead access to the A7 GPU, enabling un believable high performance for your sophisticated graphics to translate and computation tasks. |
| TextKit and Core Text | TextKit is a family of UIKit classes used to perform fine typography (Style of appearing of text) and text management. |
| Image I/O | Image I/O provides interfaces for reading and writing most image formats. |
| Photos Library | The Photos and PhotosUI frameworks provide access to a user’s photos, videos, and media. |

## Audio Technologies

* The iOS audio technologies work with the hardware to provide a rich audio experience for the users.
* This experience includes the ability to play and record high-quality audio, to handle MIDI content, and to work with a device’s built-in sounds.
* The Following are the Audio technologies are available in iOS.

|  |  |
| --- | --- |
| **Technology** | **Description** |
| Media Player framework | This high-level framework provides easy access to a user’s iTunes library and support for playing tracks and playlists. |
| AV Foundation | AV Foundation is an Objective-C interface for managing the recording and playback of audio and video. |
| OpenAL | OpenAL is an industry-standard technology for delivering positional audio |
| Core Audio | Core Audio is a set of frameworks that provide both simple and sophisticated interfaces for the recording and playback of audio and MIDI content |

iOS supports many industry-standard and Apple-specific audio formats, including the following:

* AAC
* Apple Lossless (ALAC)
* A-law
* IMA/ADPCM (IMA4)
* Linear PCM
* µ-law
* DVI/Intel IMA ADPCM
* Microsoft GSM 6.10
* AES3-2003

## Video Technologies

* The iOS video technologies provide support for managing static video content in app or playing back streaming content from the Internet.
* For devices with the appropriate recording hardware, you can also record video and incorporate it into your app.
* The Following are the Video technologies are available in iOS.

|  |  |
| --- | --- |
| **Technology** | **Description** |
| [UIImagePickerController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIImagePickerController_Class/index.html#//apple_ref/occ/cl/UIImagePickerController) | The [UIImagePickerController](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIImagePickerController_Class/index.html#//apple_ref/occ/cl/UIImagePickerController) class is a UIKit view controller for choosing user media files |
| AVKit | The AVKit framework provides a set of simple-to-use interfaces for presenting video. |
| AV Foundation | AV Foundation provides advanced video playback and recording capabilities |
| Core Media | The Core Media framework defines the low-level data types and interfaces for manipulating media |

**OS supports many industry-standard video formats and compression standards, including the following:**

* The H.264 video, up to 1.5 Mbps, 640 by 480 pixels, 30 frames per second, Low-Complexity version of the H.264 Baseline Profile with AAC-LC audio up to 160 Kbps, 48 kHz, stereo audio in .m4v, .mp4, and .mov file formats
* The H.264 video, up to 768 Kbps, 320 by 240 pixels, 30 frames per second, Baseline Profile up to Level 1.3 with AAC-LC audio up to 160 Kbps, 48 kHz, stereo audio in .m4v, .mp4, and .mov file formats
* The MPEG-4 video, up to 2.5 Mbps, 640 by 480 pixels, 30 frames per second, Simple Profile with AAC-LC audio up to 160 Kbps, 48 kHz, stereo audio in .m4v, .mp4, and .mov file formats

## AirPlay

* The AirPlay lets us app to interact audio and video stream content to Apple TV and stream audio content to third-party AirPlay speakers and receivers.
* AirPlay support is built into many frameworks like —UIKit framework, Media Player framework, AV Foundation framework, and the Core Audio family of frameworks.
* Any content you play using the above frameworks is automatically made eligible for AirPlay distribution.
* When the user chooses to play your content using AirPlay, it is routed automatically by the system.

## Media Layer Frameworks

The following sections describe the frameworks of the Media layer and the services they offer.

### Assets Library Framework

* The Assets Library framework provides access to the photos and videos managed by the Photos app on a user’s device
* Use this framework to access items in the user’s saved photos album or in any albums imported onto the device. You can also save new photos and videos back to the user’s saved photos album.

### AV Foundation Framework

* The AV Foundation framework provides a set of Objective-C classes for playing, recording, and managing audio and video content.
* Use this framework when you want to integrate media capabilities seamlessly into your app’s user interface.
* You can also use it for more advanced media handling. For example, you use this framework to play multiple sounds simultaneously and control numerous aspects of the playback and recording process.

**The services offered by this framework include:**

* Audio session management, including support for declaring app’s audio capabilities to the system
* Management of your app’s media assets
* Support for editing media content
* The ability to capture audio and video
* The ability to play back audio and video
* Track management
* Metadata management for media items
* Stereophonic panning
* Precise synchronization between sounds
* An Objective-C interface for determining details about sound files, such as the data format, sample rate, and number of channels
* Support for streaming content over AirPlay

### AVKit Framework

* The AVKit framework controls the existing objects in AV Foundation to manage the presentation of video on a device.
* It is intended as a replacement for the Media Player framework when you need to display video content.

### Core Audio

* The Core Audio is a family of frameworks and provides native support for handling audio.
* This frameworks support the generation, recording, mixing, and playing of audio in apps.
* We can also use these interfaces to work with MIDI content and to stream audio and MIDI content to other apps.

|  |  |
| --- | --- |
| **Framework** | **Services** |
| CoreAudio.framework | Defines the audio data types used throughout Core Audio. |
| AudioToolbox.framework | Provides playback and recording services for audio files and streams. |
| AudioUnit.framework | Provides services for using the built-in audio units, which are audio processing modules. |
| CoreMIDI.framework | Provides a standard way to communicate with MIDI devices, including hardware keyboards and synthesizers |
| MediaToolbox.framework | Provides access to the audio tape interfaces. |

### CoreAudioKit Framework

* The CoreAudioKit framework provides standard views for managing connections between apps that support inter-app audio
* One view provides a switcher that displays the icons of other connected apps and the other view displays the transport controls that the user can use to manipulate the audio provided by the host app.

### Core Graphics Framework

* The Core Graphics framework contains the interfaces for the Quartz 2D drawing API.
* It supports path-based drawing, anti aliased rendering, gradients, images, colors, coordinate-space transformations, and PDF document creation, display, and parsing.

### Core Image Framework

* The Core Image framework provides a powerful set of built-in filters for manipulating video and still images.
* You can use the built-in filters for everything from touching up and correcting photos to face, feature, and QR code detection.
* The advantage of these filters is that they operate in a nondestructive manner, leaving your original images unchanged. Because the filters are optimized for the underlying hardware, they are fast and efficient

### Core Text Framework

* The Core Text framework offers a simple, high-performance C-based interface for written text and handling fonts.
* This framework is for apps that do not use TextKit but that still want the kind of advanced text handling capabilities found in word processor apps.
* The framework provides a sophisticated text layout engine, including the ability to wrap text around other content. It also supports advanced text styling using multiple fonts and rendering attributes

### Core Video Framework

* The Core Video framework provides buffer and buffer-pool support for the Core Media framework. Most apps never need to use this framework directly.

### Image I/O Framework

* The Image I/O framework provides interfaces for importing and exporting image data and image metadata.
* This framework makes use of the Core Graphics data types and functions and supports all of the standard image types available in iOS.
* We can also use this framework to access Exif and IPTC metadata properties for images.

### Media Accessibility Framework

* The Media Accessibility framework manages the presentation of closed-caption content in media files
* This framework works in conjunction with new settings that let the user enable the display of closed captions.

### Media Player Framework

The Media Player framework provides high-level support for playing audio and video content from the app. We can use this framework to do the following:

* Play video to a user’s screen or to another device over AirPlay. You can play this video full screen or in a resizable view.
* Access the user’s iTunes music library. You can play music tracks and playlists, search for songs, and present a media picker interface to the user.
* Configure and manage movie playback.
* Display Now Playing information in the lock screen and App Switcher. You can also display this information on an Apple TV when content is delivered via AirPlay.
* Detect when video is being streamed over AirPlay.

### Photos Framework

* The Photos framework provides the APIs for working with photo and video assets, including iCloud Photos assets that are managed by the Photos app.
* Key features include a thread-safe architecture for fetching and caching thumbnails and full-sized assets, requesting changes to assets, observing changes made by other apps, and resemble editing of asset content.

### Photos UI Framework

* The Photos UI framework lets you create app extensions for editing image and video assets in the Photos app.

### Quartz Core Framework

* The Quartz Core framework contains the Core Animation interfaces.
* Core Animation is an advanced compositing technology that makes it easy to create view-based animations that are fast and efficient

Core Services Layer

* The Core Services layer contains basic system services for apps.
* Key among these services is the Core Foundation and Foundation frameworks, which define the basic types that all apps use.
* This layer also contains individual technologies to support features such as location, iCloud, social media, and networking.

## High-Level Features

The following sections describe some of the high-level features available in the Core Services layer.

### Peer-to-Peer Services

* The Multipeer Connectivity framework provides peer-to-peer connectivity over Bluetooth.
* You can use peer-to-peer connectivity to initiate communication sessions with nearby devices.
* We can use this in Games as well as Apps.

### iCloud Storage

* The iCloud storage lets app to write user documents and data to a central location.
* The users can then access the iCloud items from all of their devices like computers and iOS devices.
* Making a user’s documents in everyplace using iCloud means that users can view or edit those documents from any device without having to sync or transfer files explicitly.
* Storing documents in a user’s iCloud account also provides a layer of safety for users.
* Even if a user loses a device, the documents on that device are not lost if they are in iCloud storage.

There are two ways that apps can take advantage of iCloud storage, each of which has a different intended usage:

* **iCloud document storage.**Use this feature to store user documents and data in the user’s iCloud account.
* **iCloud key-value data storage.**Use this feature to share small amounts of data among instances of your app.
* **CloudKit storage**. Use this feature when you want to create publicly shared content or when you want to manage the transfer of data yourself.
* Most apps use iCloud document storage to share documents from a user’s iCloud account.

### Data Protection

* Protecting of Data from others is important thing in all applications , iOS provides Data protection by using encryption technique.
* Data protection allows apps that work with sensitive user data to take advantage of the built-in encryption available on some devices.
* When the app says a specific file as protected, the system stores that file on disk in an encrypted format.
* While the device is locked, the contents of the file are inaccessible to both app and to any potential hacker.
* When the user device is unlocked by the user, a decryption key is created to allow the app to access the file.

### File-Sharing Support

* The File-sharing support lets apps to make user data files available in iTunes 9.1 and later.
* An app that declares its support for file sharing makes the contents of its Documents directory are available to the user.
* The user can then add or remove files from this directory as needed from iTunes.
* This feature does not allow the app to share files with other apps on the same device.

**To enable file sharing for your app, do the following:**

1. Add the UIFileSharingEnabled key to your app’s Info.plist file, and set the value of the key to [YES](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/ObjCRuntimeRef/index.html#//apple_ref/doc/c_ref/YES).
2. Put whatever files you want to share in your app’s Documents directory.
3. When the device is plugged into the user’s computer, iTunes displays a File Sharing section in the Apps tab of the selected device.
4. The user can add files to this directory or move files to the desktop

### In-App Purchase

* In-App Purchase gives you the ability to sell app-specific content and services and iTunes content from inside the app.
* This feature is implemented using the StoreKit framework, which provides the infrastructure needed to process financial transactions using the user’s iTunes account.
* The app handles the overall user experience and the presentation of the content or services available for purchase.
* For downloadable content, we can host the content ourselves or let Apple’s servers host it for us.

### SQLite

* The SQLite it’s a library.
* The SQLite it’s a library lets us to embed a lightweight SQL database into the app without running a separate remote database server process.
* From app, we can create local database files and manage the tables and records in those files.
* The library is designed for general-purpose use but is still optimized to provide fast access to database records.

### XML Support

* The Foundation framework provides the [NSXMLParser](https://developer.apple.com/library/ios/documentation/Cocoa/Reference/Foundation/Classes/NSXMLParser_Class/index.html#//apple_ref/occ/cl/NSXMLParser) class for retrieving elements from an XML document.
* In Additional support for manipulating XML content is provided by the libxml2 library.
* This open source library lets you to parse or write arbitrary XML data quickly and transform XML content to HTML.

## Core Services Frameworks

The following frameworks describe below are the Core Services layer and the services they offer are.

### Accounts Framework

* The Accounts framework provides a single sign-on model for certain user accounts.
* Single sign-on improves the user experience by eliminating the need to prompt the user separately for multiple accounts.
* It also simplifies the development model for you by managing the account authorization process for your app.
* You can use this framework in conjunction with the Social framework.

### Address Book Framework

* The Address Book framework provides programmatic access to a user’s contacts database.
* If the app uses this contact information, we can use this framework to access and modify that information.
* For example, a chat program might use this framework to retrieve the list of possible contacts with which to initiate a chat session and display those contacts in a custom view.

### Ad Support Framework

* This is used for advertising purposes.
* The Ad Support framework provides access to an identifier that apps can use for advertising purposes.
* This framework also provides a flag that indicates whether the user has option out of ad tracking.
* Apps are required to read and honor the opt-out flag before trying to access the advertising identifier.

### CFNetwork Framework

* This can be used for working with network protocols.
* The CFNetwork framework is a set of high-performance C-based interfaces that use object-oriented abstractions for working with network protocols.
* These abstractions give you detailed control over the protocol stack and make it easy to use lower-level constructs such as BSD sockets.
* You can use this framework to simplify tasks such as communicating with FTP and HTTP servers or resolving DNS hosts. With the CFNetwork framework, you can:
* Use BSD sockets
* Create encrypted connections using SSL or TLS
* Resolve DNS hosts
* Work with HTTP servers, authenticating HTTP servers, and HTTPS servers
* Work with FTP servers
* Publish, resolve, and browse Bonjour services

CFNetwork is based, both physically and theoretically, on BSD sockets.

### CloudKit Framework

* CloudKit provides a conduit for moving data between the app and iCloud.
* Unlike other iCloud technologies where data transfers happen transparently, CloudKit gives you control over when transfers occur.
* You can use CloudKit to manage all types of data.
* Apps that use CloudKit directly can store data in a repository that is shared by all users.
* This public repository is tied to the app itself and is available even on devices without a registered iCloud account.
* As the app developer, we can manage the data in this container directly and see any changes made by users through the CloudKit dashboard.

### Core Data Framework

* The Core Data significantly reduces the amount of code that we have to write for the app.
* The Core Data framework is a technology for managing the data model of a Model-View-Controller app.
* Core Data is intended for use in apps in which the data model is already highly structured.
* Instead of defining data structures programmatically, we can use the graphical tools in Xcode to build a schema representing your data model.
* At runtime, instances of the data-model entities are created, managed, and made available through the Core Data framework.

**The Core Data also provides the following features:**

* Storage of object data in a SQLite database for optimal performance
* An [NSFetchedResultsController](https://developer.apple.com/library/ios/documentation/CoreData/Reference/NSFetchedResultsController_Class/index.html#//apple_ref/occ/cl/NSFetchedResultsController) class to manage results for table views
* Management of undo/redo beyond basic text editing
* Support for the validation of property values
* Support for propagating changes and ensuring that the relationships between objects remain consistent
* Support for grouping, filtering, and organizing data in memory

### Core Foundation Framework

* The Core Foundation framework is a set of C-based interfaces that provide basic data management and service features for iOS apps.

This framework includes support for the following:

* Collection data types (arrays, sets, and so on)
* Bundles
* String management
* Date and time management
* Raw data block management
* Preferences management
* URL and stream manipulation
* Threads and run loops
* Port and socket communication
* The Core Foundation framework is closely related to the Foundation framework, which provides Objective-C interfaces for the same basic features.
* When we need to mix Foundation objects and Core Foundation types, we can take advantage of the “toll-free bridging” that exists between the two frameworks.
* Toll-free bridging means that we can use some Core Foundation and Foundation types interchangeably in the methods and functions of either framework.
* This support is available for many of the data types, including the collection and string data types.

### Core Location Framework

* The Core Location framework provides location and heading information to the apps.
* For location information, the framework uses the onboard GPS, cell, or Wi-Fi radios to find the user’s current longitude and latitude.
* We can incorporate this technology into our own apps to provide position-based information to the user.
* For example, we might have a service that searches for nearby restaurants, shops, or facilities, and base that search on the user’s current location.

**Core Location also provides the following capabilities:**

* Access to compass-based heading information on iOS devices that include a magnetometer
* Support for region monitoring based on a geographic location or Bluetooth beacon
* Support for low-power location-monitoring using cell towers
* Collaboration with MapKit to improve the quality of location data in specific situations, such as when driving

### Core Media Framework

* The Core Media framework provides the low-level media types used by the AV Foundation framework.
* Most apps never need to use this framework, but it is still provided for those few developers who need more accurate control over the creation and presentation of audio and video content.

### Core Motion Framework

* The Core Motion framework provides a single set of interfaces for accessing all motion-based data available on a device.
* This framework supports accessing both raw and processed accelerometer data using a new set of block-based interfaces.
* For devices with a built-in gyroscope, you can retrieve the raw gyro data as well as processed data reflecting the attitude and rotation rates of the device.
* We can use both the accelerometer and the gyro-based data for games or other apps that use motion as input or as a way to enhance the overall user experience.
* For devices with step-counting hardware, we can access that data and use it to track fitness-related activities.

### Core Telephony Framework

* The Core Telephony framework provides interfaces for interacting with phone-based information on devices that have a cellular radio.
* Apps can use this framework to get information about a user’s cellular service provider.
* Apps interested in cellular call events (such as VoIP apps) can also be notified when those events occur.

### EventKit Framework

* The EventKit framework provides an interface for accessing calendar events on a user’s device.

**We can use this framework to do the following:**

* Get existing events and reminders from the user’s calendar
* Add events to the user’s calendar
* Create reminders for the user and have them appear in the Reminders app
* Configure alarms for calendar events, including setting rules for when those alarms should be triggered

### Foundation Framework

* The Foundation framework provides Objective-C wrappers to many of the features found in the Core Foundation framework, which is described in [Core Foundation Framework](https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/CoreServicesLayer/CoreServicesLayer.html#//apple_ref/doc/uid/TP40007898-CH10-SW1).

**The Foundation framework provides support for the following features:**

* Collection data types (arrays, sets, and so on)
* Bundles
* String management
* Date and time management
* Raw data block management
* Preferences management
* URL and stream manipulation
* Threads and run loops
* Bonjour
* Communication port management
* Internationalization
* Regular expression matching
* Cache support

### HealthKit Framework

* The HealthKit framework is a new framework for managing a user’s health-related information.
* With the proliferation of apps and devices for tracking health and fitness information, it's difficult for users to get a clear picture of how they are doing.
* HealthKit makes it easy for apps to share health-related information, whether that information comes from devices connected to an iOS device or is entered manually by the user.
* The user’s health information is stored in a centralized and secure location.
* The user can then see all of that data displayed in the Health app.
* When the app implements support for HealthKit, it gets access to health-related information for the user and can provide information about the user, without needing to implement support for specific fitness-tracking devices.
* The user can decide which data should be shared with his app.
* Once data is shared with the app, the app can register to be notified when that data changes; we have fine-grained control over when your app is notified.
* For example, you could request that your app be notified whenever the user takes his or her blood pressure, or be notified only when a measurement shows that the user’s blood pressure reaches a specific reading.

### HomeKit Framework

* HomeKit is a new framework for controlling the devices in home.
* New devices are being introducing for the home to offering more connectivity between mobile and devices.
* HomeKit framework provides a standardized way to communicate with those devices.
* By using this framework in App user can communicate with devices that users have in their homes.
* Using this kind of app, users can discover devices in their home and configure them.
* User can also create actions to control those devices.
* The user can group actions together and trigger them using the App called Siri.
* Once a configuration is created, users can invite other people to share access to it. For example, a user might temporarily offer access to a house guest.
* Use the HomeKit Accessory Simulator to test the communication of your HomeKit app with a device.

### JavaScript Core Framework

* The JavaScript Core framework provides Objective-C wrapper classes for many standard JavaScript objects.
* Use this framework to evaluate JavaScript code and to parse JSON data.

### Mobile Core Services Framework

* The Mobile Core Services framework defines the low-level types used in uniform type identifiers (UTIs).
* UTI provides a unique identifier for a particular file type, data type, directory or bundle type, and so on

### Multipeer Connectivity Framework

* The Multipeer Connectivity framework supports to find of nearby devices and the direct communication with those devices without requiring Internet connectivity.
* By using this framework we can create Multipeer sessions easily and to support reliable in-order data transmission and real-time data transmission.
* Because of this framework, the app can communicate with nearby devices and can exchange unlimited data.
* This framework provides programmatic and UI-based options for discovering and managing network services.
* This App can integrate the [MCBrowserViewController](https://developer.apple.com/library/ios/documentation/MultipeerConnectivity/Reference/MCBrowserViewController_class/index.html#//apple_ref/occ/cl/MCBrowserViewController) class into their UI to display a list of peer devices for the user to choose from.
* Alternatively, we can use the [MCNearbyServiceBrowser](https://developer.apple.com/library/ios/documentation/MultipeerConnectivity/Reference/MCNearbyServiceBrowserClassRef/index.html#//apple_ref/occ/cl/MCNearbyServiceBrowser) class to look for and manage peer devices programmatically.

### NewsstandKit Framework

* The Newsstand app is available for iPhone provides a central place for users to read magazines and newspapers.
* Publishers who want to deliver their magazine and newspaper content through Newsstand can create their own iOS apps using the NewsstandKit framework which lets you initiate background downloads of new magazine and newspaper issues.
* After you start a download, the system handles the download operation and notifies your app when the new content is available.

### PassKit Framework

* The Passbook app provides a place to store coupons for users like boarding passes, event tickets, and discount cards for businesses.
* Instead of carrying a physical representation of the above mentioned items, users can now store them into their iOS device and use them the same way as before they are using.
* The PassKit framework provides the Objective-C interfaces you need to integrate support for these items into your apps
* We can use this framework in combination with web interfaces and file format information to create and manage the passes that company offers.
* Passes are created by the company’s web service and delivered to their user’s device via email, Safari, or their custom app.
* The pass itself, using a special file format like cryptographically signed before being delivered to the user.
* The file format identifies relevant information about the service being offered so that the user knows what the service is for.
* It might also contain a bar code or other information that we can then use to validate the card so this can be redeemed or used by many companies.

### Quick Look Framework

* The Quick Look framework is used for previewing the contents of App directly.
* The main primary reason of this framework is for apps that, download files from the network or that otherwise work with files from unknown sources.
* After obtaining the file, we can use the view controller provided by this framework to display the contents of that file directly in the user interface.

### Safari Services Framework

* This Safari Services framework provides support for programmatically adding URLs to the user’s Safari reading list.

### Social Framework

* This Social framework provides a simple interface for accessing the user’s social media accounts like Facebook and Twitter.
* This framework covers the Twitter framework and adds support for other social accounts, including Facebook, Sina Weibo, and others.
* All Apps can use this framework to post status updates and images to the user’s account.
* This framework works with the Accounts framework to provide a single sign-on model for the user and to ensure that access to the user’s account is approved.

### StoreKit Framework

* The StoreKit framework provides support for the purchasing of content and services from within our iOS apps this feature is known as In-App Purchase
* For example, by using this feature we can give a feature to allow the user to unlock additional app features that are available in App.
* If we are developing a game, we can use it to offer additional game levels.
* In both cases, the StoreKit framework handles the financial aspects of the transaction and processing payment requests through the user’s iTunes Store account and providing the app with information about the purchase.
* The StoreKit framework focuses on the financial aspects of a transaction, ensuring that transactions occur securely and correctly.
* Our app handles the other aspects of the transaction, including the presentation of a purchasing interface and downloading or unlocking of the appropriate content.
* This division of labor gives you control over the user experience for purchasing content.
* When we decided to go through these kinds of applications like purchasing interface you for the user we can make use of this framework.
* We can also decide the delivery mechanism that works best for our app.

### System Configuration Framework

* The System Configuration framework is used to provide the reach ability interfaces like we can use to determine the network configuration of a device.
* We can use this framework to determine whether a Wi-Fi or cellular connection is in use and whether a particular host server can be accessed or not.

### WebKit Framework

* The WebKit framework is used to display the HTML contents in App.
* In addition to displaying HTML, we can provide basic editing support so that users can replace text and manipulate document text and attributes, including CSS properties.
* WebKit also supports creating and editing content at the DOM level of an HTML document.
* For example, you could extract the list of links on a page, modify them, and replace them prior to displaying the document in a web view.

Core OS Layer

* We use the Core OS Layer where we need to explicitly deal with security or communicating with an external hardware accessories in such situations we do so using this frameworks layer.
* The Core OS layer contains technologies of low-level features.

**The following are the Core OS Layer frameworks**

## Accelerate Framework

* The Accelerate framework contains the interfaces for performing digital signal processing (DSP), linear algebra, and image-processing calculations.
* The advantage of using this framework over writing your own versions of these interfaces is that they are optimized for all of the hardware configurations present in iOS devices.
* By using this framework can write our code once and it can be assured that it runs efficiently on all devices.

## Core Bluetooth Framework

* The Core Bluetooth framework allows for developers to interact specifically with Bluetooth low energy (LE) accessories.
* The Objective-C interfaces of this framework allows us to do the following:
* Scan for Bluetooth accessories and connect and disconnect to ones you find
* We can send documents from app by turning the iOS device to search a peripheral for other Bluetooth devices.
* Broadcast iBeacon (It’s a kind of protocol from Apple) information from the iOS device.
* It save the state of the Bluetooth connections and restore those connections when the app is subsequently launched
* It can notify when changes are available to the Bluetooth peripherals

## External Accessory Framework

* The External Accessory framework provides support for communicating with hardware devices or accessories attached to an iOS-based device.
* Accessories can be connected to the device through the 30-pin dock connector of a device or wirelessly by using Bluetooth.
* The External Accessory framework provides a way to get information about each available accessory and to initiate communications sessions.
* After that it is possible to freely manipulate the accessory directly by using any commands it supports.

## Generic Security Services Framework

* The Generic Security Services framework provides a standard set of security-related services to the iOS apps.
* The basic interfaces of this framework are specified in IETF [RFC 2743](http://www.ietf.org/rfc/rfc2743.txt) and [RFC 4401](http://tools.ietf.org/html/rfc4401)
* In addition to offering the standard interfaces, iOS includes some additions for managing credentials that are not specified by the standard but that are required by many apps.

## Local Authentication Framework

* The Local Authentication Framework lets us to use Touch ID to authenticate the user.
* In some apps may need the secure access to all of their content, and in some others Apps might not need the secure, in that certain pieces of information or options.
* In either case, we can require the user to authenticate before proceeding.
* By using this framework we can give alert to the user and we can authenticate the valid user.

## Network Extension Framework

* The Network Extension framework provides support for configuring and controlling Virtual Private Network (VPN) tunnels.
* By using this framework we can create VPN configurations.
* We can then start VPN tunnels manually or supply on-demand rules to start the VPN tunnel in response to specific events.

## Security Framework

* This framework provides interfaces for managing certificates, public and private keys, and trust policies
* In addition to the built-in security features, iOS also provides this explicit Security framework that we can use to guarantee the security of the data our app manages.
* This framework supports the generation of cryptographically secure pseudorandom (duplicate) numbers.
* It also supports the storage of certificates and cryptographic keys in the keychain, which is a secure repository for sensitive user data.
* The Common Crypto library provides additional support for symmetric encryption; hash-based message authentication codes (HMACs), and digests. The digests feature provides functions that are essentially compatible with those in the OpenSSL library, which is not available in iOS.
* It is possible for us to share keychain items among multiple apps that you create.
* Sharing items makes it easier for apps in the same suite to interoperate smoothly.
* For example, we could use this feature to share user passwords or other elements that might otherwise require you to prompt the user from each app separately.
* To share data between apps, we must configure the Xcode project of each app with the proper entitlements

## System

* The system level includes the kernel environment, drivers, and low-level UNIX interfaces of the operating system.
* The kernel itself, based on Mach, is responsible for every aspect of the operating system.
* It manages the virtual memory system, threads, file system, network, and inter process communication.
* The drivers at this layer also provide the interface between the available hardware and system frameworks.
* For security purposes, access to the kernel and drivers is restricted to a limited set of system frameworks and apps.
* iOS provides a set of interfaces for accessing many low-level features of the operating system.
* The app accesses these features through the LibSystem library.
* The interfaces are C based and provide support for the following:
* Concurrency (POSIX threads and Grand Central Dispatch)
* Networking (BSD sockets)
* File-system access
* Standard I/O
* Bonjour and DNS services
* Locale information
* Memory allocation
* Math computations

64-Bit Support

* The iOS was initially designed to support binary files on devices using a 32-bit architecture.
* In iOS 7 support was introduced for compiling, linking, and debugging binaries on a 64-bit architecture.
* All system libraries and frameworks are 64-bit ready, meaning that they can be used in both 32-bit and 64-bit apps.
* When compiled using the 64-bit runtime, apps may run faster because of the availability of extra processor resources in 64-bit architecture mode.

References:

1. <https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html>
2. <https://tilakgondi.wordpress.com/2015/01/14/ios-architecture/>
3. https://blog.inf.ed.ac.uk/sapm/2014/02/26/response-to-article-architectural-patterns-for-mobile-application-development-by-s1014475/