**IOS Architecture**

Apple describes the set of frameworks and technologies that are currently implemented within the iOS operating system as a series of layers. Each of these layers is made up of a variety of different frameworks that can be used and incorporated into your applications.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, native apps are installed physically on a device and are therefore always available to the user, even when the device is in Airplane mode. They reside next to other system apps, and both the app and any user data is synced to the user’s computer through iTunes.

The implementation of iOS technologies can be viewed as a set of layers, which is shown below.

|  |
| --- |
| **Cocoa Touch** |
| **Media** |
| **Core Service** |
| **Core OS** |

**Cocoa Touch Layer**

Cocoa Touch Layer contains the key frame work for building ios apps. It also provides basic app infrastructure such as multitasking, push notification, and touch based input and many high level services for the app.

The following are the key technologies available in the cocoa touch layer.

* **App Extension:** An app extension lets you extend custom functionality and content beyond your app and make it available to users while they’re using other apps or the system. You create an app extension to enable a specific task; after users get your extension, they can use it to perform that task in a variety of contexts.

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

* [Share](https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/ShareSheet.html#//apple_ref/doc/uid/TP40014214-CH12-SW1) - an extension that enables your app to share content with users on social networks and other sharing services.
* [Action](https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/Services.html#//apple_ref/doc/uid/TP40014214-CH13-SW1) - an extension which allows creating custom action buttons in the Action sheet to let users view or transform content originating in a host app.
* [Photo Editing](https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/Photos.html#//apple_ref/doc/uid/TP40014214-CH17-SW1) - an extension that lets users edit a photo or a video within the Photos app.
* [Document Provider](https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/FileProvider.html#//apple_ref/doc/uid/TP40014214-CH18-SW1) - an extension used for allowing other apps to access the documents managed by your app.
* [Custom Keyboard](https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/Keyboard.html#//apple_ref/doc/uid/TP40014214-CH16-SW1) - an extension that replaces the system keyboard.
* **Handoff:** Handoff lets users start an activity on one device and seamlessly resume the activity on another device. Provide continuity for users with multiple devices by supporting Handoff in your apps and websites. 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 as possible.
* **Document Picker:** The document picker feature lets users select documents from outside your app’s sandbox. These include documents stored in iCloud Drive and documents provided by a third-party extension. Users can open these documents directly, editing them in place. This access simplifies sharing documents between apps and enables more complex workflows. For example, users can easily edit a single document using multiple apps.
* **AirDrop:** AirDrop lets users share photos, documents, URLs, and other kinds of data with nearby devices. Support for sending files to other iOS devices using AirDrop 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 that you specify. If you are not yet using this class, you should consider adding it to your interface.
* **TextKit:** Text Kit is a set of classes and protocols in the UIKit framework providing high-quality typographical services that enable apps to store, lay out, and display text with all the characteristics of fine typesetting, such as kerning, ligatures, line breaking, and justification. Text Kit is built on top of Core Text, so it provides the same speed and power. UITextView is fully integrated with Text Kit; it provides editing and display capabilities that enable users to input text, specify formatting attributes, and view the results. TextKit is a full-featured, high-level set of classes for handling text and fine typography. Using TextKit, you can lay out styled text into paragraphs, columns, and pages; you can flow text around arbitrary regions such as graphics; and you can use it to manage multiple fonts. If you were considering using Core Text to implement text rendering, you 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.
* **UIKit Dynamics:** It is a full physics engine integrated into UIKit. It allows you to create interfaces that feel real by adding behaviors such as gravity, attachments (springs) and forces. You define the physical traits that you would like your interface elements to adopt, and the dynamics engine takes care of the rest.

UIKit dynamics supports the following types of behaviors:

* **UIGravityBehavior** – Provides gravitational behavior to your views.
* **UICollisionBehavior** – Provides collision detection.
* **UISnapBehavior** – Causes a view to move to and snap to a specific point on the interface as if connected to it.
* **UIAttachmentBehavior** – Specifies a dynamic connection between two dynamic items, or between a dynamic item and an anchor point.
* **UIPushBehavior** – Applies a continuous or instantaneous force to one or more dynamic items, causing those items to change position accordingly.

### Multitasking: 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 the time 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. Apps that do need specific types of work can ask the system for background execution time.

**Cocoa Touch Frame Work:**

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

**Address Book UI Framework:**

* Address Book UI is an iOS framework for displaying, selecting, editing, and creating contacts in a user’s Address Book. Similar to the Message UI framework, Address Book UI contains a number of controllers that can be presented modally, to provide common system functionality in a uniform interface.
* To use the framework, add both AddressBook.framework and AddressBookUI.framework to our project,under the “Link Binary With Libraries” phase.
* The Address Book technology for iOS provides a way to store people’s contact information and other personal information in a centralized database, and to share this information between applications. The technology has several parts:
* The Address Book framework provides access to the contact information.
* The Address Book UI framework provides the user interface to display the information.
* The Address Book database stores the information.
* The Contacts application provides a way for users to access their contact information.

**EventKit UI Framework:**

* To work with reminder and calendar events, we need to link against **EventKit**. We will also need a persistent store to save reminder items. Conveniently, EventKit provides this : **EKEventStore**. An **EKEventStore** allows us to fetch, create, edit, and delete events from a user’s Calendar database.
* Both reminders and calendar data are stored in the Calendar database. Ideally, we will have only one event store for your entire app, and you will instantiate it once. That’s because an **EKEventStore** object requires a relatively large amount of time to initialize and release.
* The Event Kit UI framework provides the classes needed to create, edit, and display events using a view controller. It provides several configurable view controller classes.

**GameKit Framework:**

Game Kit provides three separate pieces of functionality:

* Game Center offers a centralized game service that connects players to each other. Game Center implements many different features:
* Friends allow players to create anonymous online personas. Users connect to Game Center and interact with other players through analias. Players can set status messages as well as mark other players as friends.
* Multiplayer allows your game to create network matches that connect players through Game Center. Players can invite their friends or be connected to anonymous players. Most importantly, players can receive invitations to join a match even when your game is not running. Your game is running on each device and the instances of your game exchange match and voice data with each other.
* Turn-Based Gaming provides store-and-forward network match infrastructure where the match is played out over a series of discrete turns. This kind of match can be played without requiring all of the players to be connected to Game Center simultaneously.
* Leaderboards allow your game to store and fetch player scores from Game Center.
* Achievements provide the ability to track a player’s accomplishments in your game.
* Challenges allow a player to challenge other players to complete an achievement or to beat a leaderboard score.

**iAd Framework:**

* iAd allows our application to earn revenue by displaying advertisements to the user. Our application dedicates a portion of its user interface to display advertisements and in turn us to receive revenue when users view or click those advertisements.
* While we are developing our application, iAd sends test advertisements to help us to verify our implementation is correct. To receive live advertisements from iAd in a release application, we need to integrate the iAd Framework in our application and submit our binary using iTunes Connect.

**MapKit Framework:**

* MapKit is a really neat API available on the iPhone that makes it easy to display maps, jump to coordinates, plot locations, and even draw routes and other shapes on top
* The Map Kit framework provides an interface for embedding maps directly into your own windows and views. This framework also provides support for annotating the map, adding overlays, and performing reverse-geocoding lookups to determine placemark information for a given map coordinate.

**Message UI Framework:**

* The Message UI framework provides specialized view controllers for presenting standard composition interfaces for email and SMS (Short Messaging Service) text messages. Use these interfaces to add message delivery capabilities without requiring the user to leave the app.
* To display a composition interface, present the corresponding view controller modally from our app. Once presented, the user has the option to customize the contents before sending or canceling the message.

**Notification Center Framework:**

* The Notification Center framework helps us to create and manage extensions—typically called widgets—in the Today view. The framework provides API we can use to specify whether a widget has content to display and to customize aspects of its appearance and behavior on both platforms.
* In OS X, the Notification Center framework provides ways to customize the editing and searching experience in a widget.

**PushKit Framework:**

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

**UIKit Framework:**

* The UIKit framework (UIKit.framework) provides the crucial infrastructure needed to construct and manage iOS apps. This framework provides the window and view architecture needed to manage an app’s user interface, the event handling infrastructure needed to respond to user input, and the app model needed to drive the main run loop and interact with the system.

**Media Layer**

* The Media layer contains the graphics, audio, and video technologies we use to implement multimedia experiences in our apps. The technologies in this layer make it easy for us to build apps that look and sound great.
* The Media layer is distributed into several components that are specifically classified for graphical, audio or visual support frameworks. The graphical frameworks allow developers to create apps that provide graphical interfaces, animations, image input/output (I/O) readability and access to native visual elements of the device. The audio framework enables the playing, recording and integrating of audio within developed apps.

**Graphic Technology:**

* A graphical user interface (GUI) is an interface through which a user interacts with electronic devices such as computers, hand-held devices and other appliances. This interface uses icons, menus and other visual indicator (graphics) representations to display information and related user controls, unlike text-based interfaces, where data and commands are in text. GUIl representations are manipulated by a pointing device such as a mouse, trackball, stylus, or a finger on a touch screen.
* iOS provides built-in support for apps running on either Retina displays or standard-resolution displays. For vector-based drawing, the system frameworks automatically use the extra pixels of a Retina display to improve the crispness of your content. And if you use images in your app, UIKit provides support for loading high-resolution variants of your existing images automatically.

## Audio Technologies:

iOS offers a rich set of tools for working with sound in your application. These tools are arranged into frameworks according to the features they provide, as follows.

* Use the Media Player framework to play songs, audio books, or audio podcasts from a user’s iPod library. For details, see [Media Player Framework Reference](https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MediaPlayer_Framework/index.html#//apple_ref/doc/uid/TP40006952), [iPod Library Access Programming Guide](https://developer.apple.com/library/ios/documentation/Audio/Conceptual/iPodLibraryAccess_Guide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40008765), and the [AddMusic](https://developer.apple.com/library/ios/samplecode/AddMusic/Introduction/Intro.html" \l "//apple_ref/doc/uid/DTS40008845" \t "_self) sample code project.
* Use the AV Foundation framework to play and record audio using a simple Objective-C interface. For details, see [AV Foundation Framework Reference](https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVFoundationFramework/index.html#//apple_ref/doc/uid/TP40008072) and the [avTouch](https://developer.apple.com/library/ios/samplecode/avTouch/Introduction/Intro.html" \l "//apple_ref/doc/uid/DTS40008636" \t "_self) sample code project.
* Use the Audio Toolbox framework to play audio with synchronization capabilities, access packets of incoming audio, parse audio streams, convert audio formats, and record audio with access to individual packets. For details, see [Audio Toolbox Framework Reference](https://developer.apple.com/library/ios/documentation/MusicAudio/Reference/CAAudioTooboxRef/index.html#//apple_ref/doc/uid/TP40002089) and the [SpeakHere](https://developer.apple.com/library/ios/samplecode/SpeakHere/Introduction/Intro.html" \l "//apple_ref/doc/uid/DTS40007802" \t "_self) sample code project.
* Use the Audio Unit framework to connect to and use audio processing plug-ins. For details, see [Audio Unit Hosting Guide for iOS](https://developer.apple.com/library/ios/documentation/MusicAudio/Conceptual/AudioUnitHostingGuide_iOS/Introduction/Introduction.html#//apple_ref/doc/uid/TP40009492).
* Use the OpenAL framework to provide positional audio playback in games and other applications. iOS supports OpenAL 1.1. For information on OpenAL, see the [OpenAL](http://openal.org/) website, OpenAL FAQ for iPhone OS, and the oalTouch sample code project.

## Video Technologies:

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

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

* 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
* 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
* 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
* Numerous audio formats, including the ones listed in [Audio Technologies](https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/MediaLayer/MediaLayer.html#//apple_ref/doc/uid/TP40007898-CH9-SW2)

## AirPlay

* With AirPlay, we can stream music, photos, and videos to our Apple TV, or stream music to our AirPort Express or AirPlay-enabled speakers. And with AirPlay Mirroring, we can display your iOS screen on our Apple TV.
* AirPlay lets our app stream audio and video content to Apple TV and stream audio content to third-party AirPlay speakers and receivers. AirPlay support is built into numerous frameworks—UIKit framework, Media Player framework, AV Foundation framework, and the Core Audio family of frameworks—so in most cases we do not need to do anything special to support it. Any content we play using these 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.
* To extend the content displayed by an iOS device, create a second window object and assign it to any [UIScreen](https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIScreen_Class/index.html" \l "//apple_ref/occ/cl/UIScreen" \t "_self) objects that are connected to the device through AirPlay. Use this technique when the content we display on the attached screen is different than the content displayed on the iOS device.
* The playback classes of the Media Player framework automatically support AirPlay. We can also display Now Playing content on a connected Apple TV using AirPlay.
* Use the [AVPlayer](https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVPlayer_Class/index.html" \l "//apple_ref/occ/cl/AVPlayer" \t "_self) class in AV Foundation to manage our app’s audio and video content. This class supports streaming its content via AirPlay when enabled by the user.

**Media Layer Framework:**

The following data describes the media layer frameworks and the services they offer.

**Assets Library Framework:**

* The assets framework used to access the user photos and video’s stored in the devices managed by the photo application.
* Instances of the Photos framework model classes ([PHAsset](https://developer.apple.com/library/ios/documentation/Photos/Reference/PHAsset_Class/index.html" \l "//apple_ref/occ/cl/PHAsset), [PHAssetCollection](https://developer.apple.com/library/ios/documentation/Photos/Reference/PHAssetCollection_Class/index.html" \l "//apple_ref/occ/cl/PHAssetCollection), and[PHCollectionList](https://developer.apple.com/library/ios/documentation/Photos/Reference/PHCollectionList_Class/index.html#//apple_ref/occ/cl/PHCollectionList)) represent the entities a user works with in the Photos app: assets (images or videos), collections of assets (such as albums or moments), and lists of collections (such as album folders or moment clusters). These objects, also called photo entities, are read-only, immutable, and contain only metadata.
* You work with assets and collections by fetching the photo entities you’re interested in and then using those objects to fetch the data you need to work with. To make changes to photo entities, you create change request objects and explicitly commit them to the shared[PHPhotoLibrary](https://developer.apple.com/library/ios/documentation/Photos/Reference/PHPhotoLibrary_Class/index.html#//apple_ref/occ/cl/PHPhotoLibrary) object. This architecture makes it easy, safe, and efficient to work with the same assets from multiple threads or multiple apps and app extensions.

**AV Foundation Framework:**

* AVFoundation is one of several frameworks that we can use to play and create time-based audiovisual media.
* It provides an Objective-C interface we use to work on a detailed level with time-based audiovisual data.
* For example, We can use it to examine, create, edit, or reencode media files. We can also get input streams from devices and manipulate video during realtime capture and playback.
* The primary class that the AV Foundation framework uses to represent media is [AVAsset](https://developer.apple.com/library/ios/documentation/AVFoundation/Reference/AVAsset_Class/index.html" \l "//apple_ref/occ/cl/AVAsset" \t "_self).
* The design of the framework is largely guided by this representation. Understanding its structure will help us to understand how the framework works.
* An AVAsset instance is an aggregated representation of a collection of one or more pieces of media data (audio and video tracks).
* It provides information about the collection as a whole, such as its title, duration, natural presentation size, and so on. AVAsset is not tied to particular data format.

**Core Audio:**

* Core Audio provides software interfaces for implementing audio features in applications you create for iOS .
* In iOS, Core Audio capabilities include recording, playback, sound effects, positioning, format conversion, and file stream parsing will be done.
* Audio Queue Services provides a straightforward, low overhead way to record and play audio in iOS.
* It is the recommended technology to use for adding basic recording or playback features to our iOS.
* A built-in equalizer and mixer that we can use in our applications
* Automatic access to audio input and output hardware.
* APIs to let us manage the audio aspects of our application in the context of a device that can take phone calls
* Optimizations to extend battery life without impacting audio quality.

**CoreAudioKit Framework:**

* The CoreAudioKit framework (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 is a C-based API that is based on the Quartz advanced drawing engine.
* It provides low-level, lightweight 2D rendering with unmatched output fidelity.
* We use this framework to handle path-based drawing, transformations, color management, offscreen rendering, patterns, gradients and shadings, image data management, image creation, masking, and PDF document creation, display, and parsing.
* Quartz 2D is an advanced, two-dimensional drawing engine available for iOS application development.
* Quartz 2D provides low-level, lightweight 2D rendering with unmatched output fidelity regardless of display or printing device.
* The Quartz 2D API is easy to use and provides access to powerful features such as transparency layers, path-based drawing, offscreen rendering, advanced color management, anti-aliased rendering, and PDF document creation, display, and parsing.

**Core Image Framework:**

* Core Image is an image processing and analysis technology designed to provide near real-time processing for still and video images. In iOS and OS X we can use Core Image classes to.
* Process images using the many built-in image filters.
* Chain together filters and then archive them for later use.
* Detect features (such as faces and eyes) in still and video images, and track faces in video images.
* Analyze images to get a set of auto adjustment filters.
* Create custom filters for use in our app.

**Core Text Framework:**

* Core Text is for apps that need a low-level text-handling technology correlating with the Core Graphics framework (Quartz).
* If we work directly with Quartz and you need to draw some text, use Core Text.
* for example, we have our own page layout engine—we have some text and we know where it needs to go in our view—we can use Core Text to generate the glyphs and position them relative to each other with all the features of fine typesetting, such as kerning, ligatures, line-breaking, hyphenation, and justification.

**Image I/O Framework:**

* imageProperties Reference defines constants that represent characteristics of images used by the Image I/O framework.
* The Image I/O framework (ImageIO.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 Player Framework:**

* The Media Player framework provides facilities for playing movie, music, audio podcast, and audio book files.
* This framework also gives application access to the iPod library, letting us find and play audio-based media items synced from iTunes on the desktop. iPod library access is read-only.
* This framework’s [MPVolumeView](https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MPVolumeView_Class/index.html" \l "//apple_ref/occ/cl/MPVolumeView) class lets us present a control to let the user adjust system audio output volume and choose among the available output routes, such as for sending audio to an AirPlay-enabled device.
* The [MPMoviePlayerController](https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MPMoviePlayerController_Class/index.html" \l "//apple_ref/occ/cl/MPMoviePlayerController) and [MPTimedMetadata](https://developer.apple.com/library/ios/documentation/MediaPlayer/Reference/MPTimedMetadata_Class/index.html" \l "//apple_ref/occ/cl/MPTimedMetadata) classes let user play streamed video content and respond to time-based metadata contained in the stream.

**Metal Framework:**

* Metal provides extremely low-overhead access to the A7 GPU enabling incredibly high performance for our sophisticated graphics rendering and computational tasks.
* Metal eliminates many performance bottlenecks—such as costly state validation—that are found in traditional graphics APIs.
* Metal is explicitly designed to move all expensive state translation and compilation operations out of the critical path of your most performance sensitive rendering code.
* Metal provides precompiled shaders, state objects, and explicit command scheduling to ensure our application achieves the highest possible performance and efficiency for our GPU graphics and compute tasks.
* This design philosophy extends to the tools used to build our app. When our app is built, Xcode compiles Metal shaders in the project into a default library, eliminating most of the runtime cost of preparing those shaders.
* The Metal Framework provides extremely low-overhead access to the GPU, enabling incredibly high performance for our sophisticated graphics rendering and computational tasks.
* Metal eliminates many performance bottlenecks—such as costly state validation—that are found in traditional graphics APIs.
* Metal moves all expensive state translation and compilation operations out of the critical path of our most performance sensitive rendering code.
* It provides precompiled shaders, state objects, and explicit command scheduling to ensure that your application achieves the highest possible performance and efficiency.

**OpenAL Framework:**

* OpenAL is a cross-platform 3D audio API appropriate for use with gaming applications and many other types of audio applications.
* The library models a collection of audio sources moving in a 3D spaces that are heard by a single listener somewhere in that space.
* The basic OpenAL objects are a listener, a source, and a Buffer. There can be a large number of Buffers, which contain audio data.
* Each buffer can be attached to one or more Sources, which represent points in 3D space which are emitting audio.
* There is always one listener object (per audio context), which represents the position where the sources are heard – rendering is done from the perspective of the listener.

**OpenGL ES Framework:**

* The OpenGL ES framework (OpenGLES.framework) provides tools for drawing 2D and 3D content.
* It is a C-based framework that works closely with the device hardware to provide fine-grained graphics control and high frame rates for full-screen immersive apps such as games.
* We use the OpenGL framework in conjunction with the EAGL interfaces, which provide the interface between our OpenGL ES drawing calls and the native window objects in UIKit.
* TheOpen Graphics Library (OpenGL)is used for visualizing 2D and 3D data.
* It is a multipurpose open-standard graphics library that supports applications for 2D and 3D digital content creation, mechanical and architectural design, virtual prototyping, flight simulation, video games, and more.
* we use OpenGL to configure a 3D graphics pipeline and submit data to it. Vertices are transformed and lit, assembled into primitives, and rasterized to create a 2D image.
* OpenGL is designed to translate function calls into graphics commands that can be sent to underlying graphics hardware.

**Photos UI Framework:**

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

**Quartz Core Framework:**

* The Quartz Core framework (QuartzCore.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.
* The compositing engine takes advantage of the underlying hardware to manipulate your view’s contents efficiently and in real time.
* Specify the start and end points of the animation, and let Core Animation do the rest. And because Core Animation is built in to the underlying UIView architecture, it is always available.

**SceneKit Framework:**

* SceneKit is an Objective-C framework for building simple games and rich app user interfaces with 3D graphics, combining a high-performance rendering engine with a high-level, descriptive API.
* SceneKit has been available since OS X v10.8 and is now available in iOS for the first time. Lower-level APIs (such as OpenGL ES) require us to implement the rendering algorithms that display a scene in precise detail.
* By contrast, SceneKit let us describe our scene in terms of its content—geometry, materials, lights, and cameras—then animate it by describing changes to those objects.
* SceneKit’s 3D physics engine enlivens our app or game by simulating gravity, forces, rigid body collisions, and joints.
* Add high-level behaviors that make it easy to use wheeled vehicles such as cars in a scene, and add physics fields that apply radial gravity, electromagnetism, or turbulence to objects within an area of effect.
* Use OpenGL ES to render additional content into a scene, or provide GLSL shaders that replace or augment SceneKit’s rendering. We can also add shader-based post-processing techniques to SceneKit’s rendering, such as color grading or screen space ambient occlusion

**SpriteKit Framework:**

* The SpriteKit framework (SpriteKit.framework) provides a hardware-accelerated animation system for 2D and 2.5D games.
* SpriteKit provides the infrastructure that most games need, including a graphics rendering and animation system, sound playback support, and a physics simulation engine.
* Using SpriteKit frees we from creating these things ourself and let us focus on the design of our content and the high-level interactions for that content.
* Content in a SpriteKit app is organized into scenes.
* A scene can include textured objects, video, path-based shapes, Core Image filters, and other special effects. SpriteKit takes those objects and determines the most efficient way to render them onscreen.
* When it comes time to animate the content in your scenes, We can use SpriteKit to specify explicit actions we want to perform or use the physics simulation engine to define physical behaviors (such as gravity, attraction, or repulsion) for our objects.
* In addition to the SpriteKit framework, there are Xcode tools for creating particle emitter effects and texture atlases.
* We can use the Xcode tools to manage app assets and update SpriteKit scenes quickly.

**Core Services Layer**

* The Core Services layer contains fundamental system services for apps.
* Key among these services are the Core Foundation and Foundation frameworks, which define the basic types that all apps use. T
* his layer also contains individual technologies to support features such as location, iCloud, social media, and networking.
* The following section describes the high level features available in Core Services Layer they are.

**iCloud Storage:**

* iCloud is a free service that lets users access their personal content on all their devices—wirelessly and automatically via Apple ID.
* iCloud does this by combining network-based storage with dedicated APIs, supported by full integration with the operating system.
* Apple provides server infrastructure, backup, and user accounts, so we can focus on building great iCloud-enabled apps.
* There are three iCloud storage services: key-value storage, document storage, and CloudKit.
* key-value and document storage is to eliminate explicit synchronization between devices.
* A user never needs to think about syncing, and your app never interacts directly with iCloud servers.
* CloudKit allows us to store app and user data as records in a public database, that is shared between users of our app, or a private database accessible only by the current user.

**Peer-to-Peer Services:**

* The peer Connectivity framework provides support for discovering services provided by nearby iOS devices using infrastructure Wi-Fi networks.
* peer-to-peer Wi-Fi, and Bluetooth personal area networks and subsequently communicating with those services by sending message-based data, streaming data, and resources.
* Peer IDs ([MCPeerID](https://developer.apple.com/library/ios/documentation/MultipeerConnectivity/Reference/MCPeerID_class/index.html" \l "//apple_ref/occ/cl/MCPeerID)) uniquely identify an app running on a device to nearby peers.
* Session objects maintain a set of peer ID objects that represent the peers connected to the session.
* Advertiser objects also use a single local peer object to provide information that identifies the device and its user to other nearby devices.

**Block Objects:**

* Blocks are objects that encapsulate a unit of work—or, in less abstract terms, a segment of code—that can be executed at any time.
* They are essentially portable and anonymous functions that one can pass in as arguments of methods and functions or that can be returned from methods and functions.
* Blocks themselves have a typed argument list and may have inferred or declared returned type.
* We may also assign a block to a variable and then call it just as we would a function.
* The caret symbol (^) is used as a syntactic marker for blocks.

**Data Protection:**

* Data protection allows apps that work with sensitive user data to take advantage of the built-in encryption available on some devices.
* When our app designates 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 our app and to any potential intruders.
* However, when the device is unlocked by the user, a decryption key is created to allow our app to access the file.
* Other levels of data protection are also available for us to use.
* Implementing data protection requires us to be considerate in how we create and manage the data we want to protect.
* Apps must be designed to secure the data at creation time and to be prepared for access changes when the user locks and unlocks the device.

**File-Sharing Support:**

* File-sharing support lets apps 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 available to the user.
* The user can then move files in and out of this directory as needed from iTunes.
* This feature does not allow our app to share files with other apps on the same device; that behavior requires the pasteboard or a document interaction controller object.

### In-App Purchase:

* In-App Purchase gives you the ability to vend app-specific content and services and iTunes content from inside our app.
* This feature is implemented using the StoreKit framework, which provides the infrastructure needed to process financial transactions using the user’s iTunes account.
* our 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 our self or let Apple’s servers host it for us.

### 

### SQLite

* The SQLite library lets you embed a lightweight SQL database into our app without running a separate remote database server process.
* From our 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.

## Core Services Frameworks

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

### Accounts Framework:

### The Accounts framework provides access to user accounts stored in the Accounts database.

### An account stores the login credentials of a particular service, such as Twitter, that can be used as authentication for the service.

### By implementing the Accounts framework into our app, we do not need to be responsible for storing account logins our self. Instead, the user can grant access for your app to use their account login credentials

* The Accounts framework (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 us by managing the account authorization process for our app. we use this framework in conjunction with the Social framework.

**Address Book Framework:**

* The Address Book framework provides access to a centralized contacts database, called the Address Book database, that stores a user’s contacts.
* Applications such as Mail and Messages use this database to present information about known and unknown persons.
* The Address Book framework (AddressBook.framework) provides programmatic access to a user’s contacts database.
* If our app uses 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.

**CFNetwork Framework**

* The CFNetwork framework (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.
* We can use this framework to simplify tasks such as communicating with FTP and HTTP servers or resolving DNS hosts. With the CFNetwork framework, we 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.

### Foundation Framework:

The Foundation framework (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 suppor

**HealthKit Framework:**

* HealthKit (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 our 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 decides which data should be shared with our app.

**HomeKit Framework:**

* HomeKit (HomeKit.framework) is a new framework for communicating with and controlling connected devices in a user’s home.
* New devices being introduced for the home are offering more connectivity and a better user experience.
* HomeKit provides a standardized way to communicate with those devices.
* our app can use HomeKit to communicate with devices that users have in their homes.
* Using our app, users can discover devices in their home and configure them.
* They can also create actions to control those devices.
* The user can group actions together and trigger them using Siri.
* Once a configuration is created, users can invite other people to share access to it.

**WebKit Framework:**

* The WebKit framework (WebKit.framework) lets display HTML content in our 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.

**StoreKit Framework:**

* The StoreKit framework (StoreKit.framework) provides support for the purchasing of content and services from within our iOS apps, a feature known as *In*-App Purchase.
* For example, we can use this feature to allow the user to unlock additional app features. Or if we are a game developer, we can use it to offer additional game levels.
* In both cases, the StoreKit framework handles the financial aspects of the transaction, processing payment requests through the user’s iTunes Store account and providing our app with information about the purchase.
* StoreKit 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 the downloading (or unlocking) of the appropriate content.
* This division of labor gives you control over the user experience for purchasing content.

**PassKit Framework:**

* The Passbook app provides users with a place to store coupons, boarding passes, event tickets, and discount cards for businesses.
* Instead of carrying a physical representation of these items, users can now store them on their iOS device and use them the same way as before.
* The PassKit framework (PassKit.framework) provides the Objective-C interfaces we need to integrate support for these items into our apps.
* We use this framework in combination with web interfaces and file format information to create and manage the passes your company offers.
* Passes are created by our company’s web service and delivered to the user’s device via email, Safari, or your custom app.
* The pass itself, using a special file format, is cryptographically signed before being delivered.
* 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 that it can be redeemed or used.
* our app handles the other aspects of the transaction, including the presentation of a purchasing interface and the downloading (or unlocking) of the appropriate content.
* This division of labor gives you control over the user experience for purchasing content.