2-iOS SDK

Il sistema operativo iOS

iOS è il sistema operativo per dispositivi mobili di Apple, spedito con dispositivi iPhone, iPod Touch e (vecchi) iPad. Il primo rilascio fù nel 2007, la versione corrente è iOS 16, e il corso copre fino ad iOS 7, rilasciato nel settebre 2014. Le applicazioni iOS eseguono su un sistema UNIX-based e supporta completamente threads, sockets, ecc...

Gestione della memoria in iOS

iOS utilizza un sistema di memoria virtuale, ogni programma la i propri indirizzi a spazi vituali; la memoria disponibile è limitata alla memoria fisica.

> Nota
> iOS non supporta il packaging del disco quindi, quando la memoria è piena, il sistema libera della memoria virtuale occupata.
> Le notifiche di memoria piena vengono mandate alla applicazioni, così possono liberare la memoria

Multi-threading in iOS

Dalla varsione 4, iOS permette alle applicazioni di eseguire in background quando non sono visibili sullo schermo, la maggior parte di queste applicazioni risiede nella memoria ma non esegue nessun tipo di codice. Questo perché il sistema sospende rapidamente le applicazioni in background per preservare la duarata della batteria.
In alcuni casi le applicazione potrebbero chiedere al sistema operativo il permesso di eseguire in background.

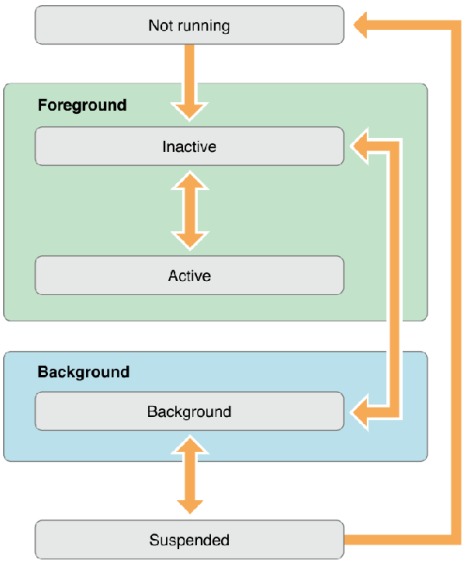
> Nota
> L'esecuzione di un applicazione in background richiede una gestione appropriata degli stati dell'applicazione

App sandbox

Per ragioni di sicurezza, iOS chiude nella fase di installazione, ogni applicazione (anche impostazioni e dati) in una sandbox. In quasto modo limita l'applicazione nell'accedere ai file, alle impostazini, alle risorse di rete, all'hardware, ecc...
Il sistema installa ogni applicazione nella propria cartella sandbox, che può essere vista come home per l'applicazione e i suoi dati.

> Nota
> Le sandbox prevengono solo che le applicazione infette infettino altre applicazioni o altre parti del sistema

Il ciclo di vita delle applicazioni



Quando un applicazione è lanciata questa vine messa dalla fase *not-running* alle fasi di *active* o *background*. iOS crea un processo e il thread principale per l'applicazione, successivamente vengono invocate tutte le funzioni principali su questo thread principale. Questo avvenimento si chiama "main event loop".
Il main event loop riceve eventi dal sistema operativo, questi eventi sono generati dalle azioni dell'utente (es. relativi alla UI)

Il ciclo di vita di un applicazione:

1. Lancio dell'applicazione
2. Inizializzazione dell'applicazione
3. Caricamento della root del view controller, questa corrisponde alla root delle UI dell'applicazione
4. Attesa di un evento
5. Gestione dell'evento, dopo questa torna al punto 4
6. Terminazione dell'applicazione

> Nota
> L'event loop è formato dei punti 4 e 5.

> Nota
> La distruzione di un oggetto, ovvero quando avviene la chiamata di `dealloc` avviene quando l'event loop completa la sua esecuzione

UIApplication and UIApplicationDelegate

In iOS le applicazioni sono un istanza di una classe chiamata `UIApplication`, che è gestita dal sistema operativo e non viene mai toccata direttamente.
Ogni applicazione ha un oggetto delegato (`delegate` object) conforme al protocollo `UIApplicationDelegate` che riceve messaggi quando un applicazione cambia il suo stato.

> Nota
> Una transazione di stato è seguita anche dalla corrispondente chiamata del metodo dell'applicazione delegata all'oggetto

> Nota
> Questi metodi sono un modo di rispondere ai cambiamenti di stato in modo appropriato

I metodi del protocollo UIApplicationDelegate

A seguire i metodi del protocollo:

- `application:willFinishLaunchingWithOptions:`, questo metodo corrisponde alla prima possibilità di esecuzione quando l'aplicazione viene lanciata
- `application:didFinishLaunchingWithOptions:`, questo metodo permette du eseguire ogni inizializzazione prima che l'applicazione venga mostrata
- `applicationDidBecomeActive:`, questo metodo permette all'applicazione di sapere se è prossima ad andare in foreground, è da utilizzare per ogni preparazione fatta all'ultimo
- `applicationWillResignActive:`, questo metodo permette all'applicazione di sapere se sta transitando via dalla fase di foreground, questo metodo va utilizzato per mettere l'applicazione in una fase "dormiente"
- `applicationDidEnterBackground:`, questo metodo permette di sapere se l'applicazione sta eseguendo in background e che potrebbe essere sospesa in un qualsiasi momento

- `applicationWillEnterForeground:`, questo metodo permette di sapere se l'applicazione stia transitando dal background al foreground, senza essere ancora attiva
- `applicationWillTerminate:`, questo metodo permette di capire se l'applicazione sta per essere terminata, non viene chiamata se l'applicazione viene messa nello stato di sospensione

iOS layers

iOS presenta una struttura a strati (layers). Il sistema operatico agisce da intermediario tra le applicazioni e l'hardware sottostante, mediante l'utilizzo di interfacce di sistema ben definite.
Gli strati più bassi contengono i servizi e le metodi fondamentali, quelli più alti invece forniscono servizi e metodi più soffisticati.

> Nota
> Le metodi in iOS sono raccolte come fremeworks, specialmente `Foundation and UIKit frameworks`
> Un framework è una cartella che contiene librerie e risorse, neccessarie per supportare le librerie, condivise dinamicamente

Tutti i layers
Applicazioni/utente
Cocoa Touch
Media
Core Services
Core OS
Hardware

I layers di iOS

Layer	**Particolarità**	**Frameworks principali**	
Cocoa Touch	Framework per la creazione di applicazioni iOS, multitasking, touch-based input, notifiche push, e molti altri servizi ad alto lo	- `UIKit Framework`, costruzione e gestione dell'interfaccia utente per le applicazioni iOS - `Map Kit Framework`, mappe scorrevoli incorporabili all'interno dell'UI delle applicazioni - `Game KitFramework`, supporto per il Game Center - `Address BookFramework`, interfacce di sistema standard per la gestione dei contatti - `MessageUI Framework`, interfaccia per scrivere email o messaggi SMS - `Event Kit Framework` interfacce di sistema standard per la gestione degli eventi del calendario	
Media	metodi per grafica, audio e video	- `AV Foundation Framework`, riproduzione, registrazione e gestione dei contenuti audio/video - `Media PlayerFramework`, supporto ad alto livello per la riproduzione di contenuti audio/video - `Core Audio Framework`, supporto nativo e a bassolivello per la gestione degli audio - `COre GraphicsFramework`, supporto per il path-based drawing, antianalised rendering, gradienti, immagini, colori - `Quartz Core Framework`, visione efficente delle animazione mediante il `Core Animetion interfaces` - `OpenGL ES Framework`, strumenti per il supporto al disegno 2D e 3D	
Core Services	Servizi di sistema fondamentali per le applicazioni	- `CFNetwork Framework`, BSD sockets, connessioni TLS/SSL, DNS resolution, connessioni HTTP/HTTPS - `Core Data Framework` - `Core Foundation Framework`, collections (librerie del C), strings, date and time, threads - `Core Location Framework`, provvede la localizzazione e l'intestazione delle informazioni alle Applicazioni - `Foundation Framework`, "impacchetta" `COre Foundation` per i tipi di Objective-C - `System Configuration Framework`, Connettività e raggiungibilità	

Layer	Particolarità	Frameworks principali	
Core OS	Low-level features	- `Accellerate Framework`, matematica di vettori e matrici, getione dei segnali digitali, gestione dei grandi numeri, processamento delle immagini - `Core Bluetooth Frameworks` - `Security Frameworks`, supporto alla crittografia simmetrica, ai codici di autenticazione dei messaggi (HMACs) e digest - `System`, kernel enviroment, drivers e l'interfaccia UNIX a basso livello del sistema operativo. Supporto per la concorrenza (POSIX thread e Grand Central Dispach), Networking (BSD sockets), accesso al File-system, I/O standard, Bonjour e servizi DNS	

> Nota
> `HMACs` è una funzione hash criptografica che permette di garantire sia l'integrità sia l'autenticazione dei messaggi

> Nota
> Con il termine `digest` si intende una funzione hash criptografica

> Nota
> Bonjour è una metodo di rete Apple che permette ai dispositivi di rilevare i dispositivi Apple vicini

> Nota
> DNS vuol dire Domain Name System (sistema di nomi di dominio), ed indica un sistema utilizzato per assegnare nomi agli host (nodi della rete). In oltre indica anche il protocollo che regola il funzionamento del servizio, i programmi che lo implementano, i server su cui questi vengono elaborati, l'insieme di server cooperanti per fornire un ervizio più intelligente.
> Per saperne di più Wikipedia (https://it.wikipedia.org/wiki/Domain_Name_System)

> Nota
> La DNS resolution è il processo di traduzione degli indirizzi IP in domini

> Nota
> Le BSD sockets sono delle API per la gestione di Internet Sockets e Unix domain sockets utilizzati per l'IPC (inter-process comunication)

iOS SDK

L'iOS Software Development Kit continete gli strumenti e le interfacce necessarie per sviluppare, installare, eseguire e testare app native.

- Tool: Xcode
- Linguaggio: Objective-C, più C/C++
- Librerie: iOS frameworks
- Documentazione: iOS Developer Library

Xcode

Xcode è un ambiente di sviluppo usato epr creare, testare, debuggare e ritoccare applicazioni; esso contiene tutti gli strumenti neccassire per creare applicazioni:

- Costruttore dell'interfaccia
- Debugger
- Instruments, utilizzato per analizzare il comportamento delle applicazioni, ad esempio per monitorare l'allocazione della memoria
- Simulatore iOS
  È utilizzato per scrivere codice che può essere eseguito sul simulatore o su un iDevice connesso.

Model-View-Controller

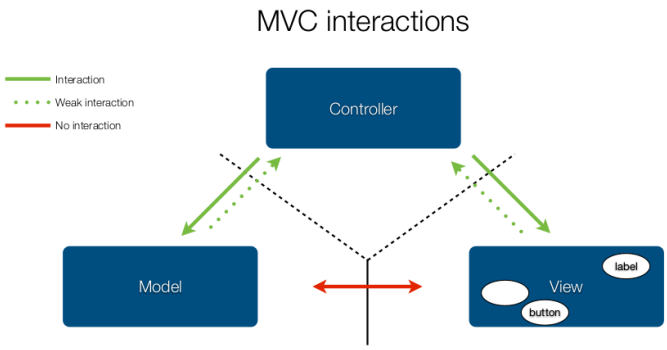
Tutte le applicazioni iOS sono create utilizzando il pattern MVC, che è utiizzato per organizzare le parti del codice in modo pulito ed ordinato in campi separati, secondo le loro responsabilità e particolarità di ognuno di esse. L'organizzazione è estremamente importante perché provvede un modo di creare applicazioni che sono semplici da scrivere, mantenere e debuggare.

> Nota
> La iOS SDK è creata in modo da guidare gli sviluppatori a creare applicazioni utilizzando la MVC,.
> Capire e rispettare la MVC è il 90% del lavoro quando si sviluppa per iOS

Il codice di un applicazione può appertenere ad un modello, ad una vista o ad un controller.

- Il modello è la rappresentazione dei dati che sarà usata nell'applicazione. Un modello è indipendente dalla vista fiché non saprò come i dati sarranno mostrati.
- La vista è l'interfaccia utente che mostra i contenuti dell'aplicazione. Una vista è indipendente dal modello fiché questa contiene un mucchio di elementi grafici utilizzabili in tante applicazioni.
- Il controller è il cervello dell'applicazione, gestisce come i dati del modello devono essere rappresentati nella vista. Un controller è fortemente dipendete da un modello e da una vista, perché deve sapere quai dati dovrà gestire e con quali elementi grafici dovrà interagire.

Iterazioni MVC



- Il modello ha un iterazione diretta con la vista e non con il controller, dato che è solo responsabile di tenere i dati (es. database)
- Il controller:
  - ha un iterazione diretta con il modello perché ha necessità di leggere e memoriazzare dati
  - ha un iterazione diretta con la vista perché deve aggiornare gli elementi della UI
  - mantiene un riferimento agli elementi dell'UI anche utilizza, chiamati outlets (IBOutlet)
- La vista non ha un interazione diretta né con il controller né con il modello, perché il suo lavoro è solo quello di rappresentare gli elementi dell'UI sullo schermo

> Nota
> Anche se la vista non interagisce direattamente con il controller, perché non sa della sua esistenza, deve notificarlo su determinati eventi (es. pressione di un bottone). L'interazione tra questi avviene in modo cieco, attraverso actions (IBAction).
> Il controller può registrare nella vista che è il bersaglio, così quando avviene un azione la vista la manda direttamente al bersaglio.

> Nota
> Questo modello d'interazione permette di avere una vista completamente indipendente dal controller

> Nota
> Alcune viste interagiscono con il controller per essere sincronizzati, quando un certo evento dovrebbe avvenire/avviene/è avvenuto, la vista deve informare il controller così può eseguire alcune operazioni.
> Questo avvenimento si chiama delegazione, il controller è il delegato, questo vuol dire che la vista passa la responsabilità al controller per compiere determinate task.
> Altre viste necessitano del controller per sapere quali dati mostrare, per questo motivo il controller diventa un data source per la vista.

> Nota importante
> Le iterazioni avvengono mediante l'utilizzo di protocolli

> Nota
> L'interazione tra modello e controller permette al modello di comunicare al controller (che aggiornerà la vista) che i dati hanno subito un cambiamento. Questa comunicazione avviene mediante notifiche o KVO (key-value observing)

View Controller

Gli oggetti del view controller forniscono le infrastrutture per gestire contenuti e coordinare se questi sono mostrati o nascosti. Avendo differenti classi permette di dividere porzioni dell'interfaccia utente, l'implementazione dell'interfaccia utente viene divisa in unità più piccole e gestibili.

> Nota
> Il view controller è una sottoclasse della classe `UIViewController`, e deve eseguire determinate task che sono specificate nella sua super-classe

Interfaccia utente

L'interfaccia utente è composta da:

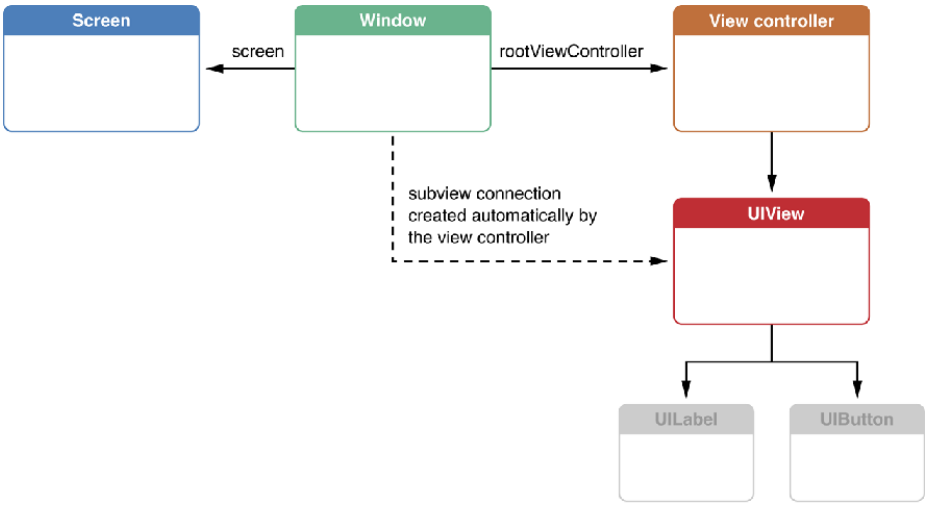
- Schermo, `UIScreen` identifica lo schermo fisico conneso al dispositivo
- Finestra, `UIWindow` fornisce un supporto allo schermo per rappresentare (disegnare sullo schermo)
- Vista, le classi di `UIView` si occupano di eseguire la rappresentazione. Questi ogegtti sono attaccati alla finestra e rappresentanto i loro contenuti quando la finestra richiede di farlo.

Una vista rappresenta l'elemento dell'interfaccia utente, ognuna copre una specifica area, all'interno di quest'area mostra i contenuti o risponde agli eventi dell'utente. Le viste possono essere annidate in una vista gerarchica, le sotto-viste sono posizionate e disegnano in modo relativo alla loro super-view.

> Nota
> Le viste possono animare i valori delle loro proprietà, questa cosa è cruciale eprché permette all'utente di capire i cambiamenti dell'interfaccia.

> Nota
> Le applicazioni complicate sono composte da molte viste che possono essere raggruppare in viste gerarchiche.

> Nota
> La vista che risponde alle interazioni con l'utente sono chiamate controls (`UIControl`), es. `UISliders` e `UIButton`



> Nota importante
> Ogni vista è controllata da un solo view controller, che ha una proprietà `view`, la quale indica che il view controller possiede quella vista.