

iOS App Design Exploration

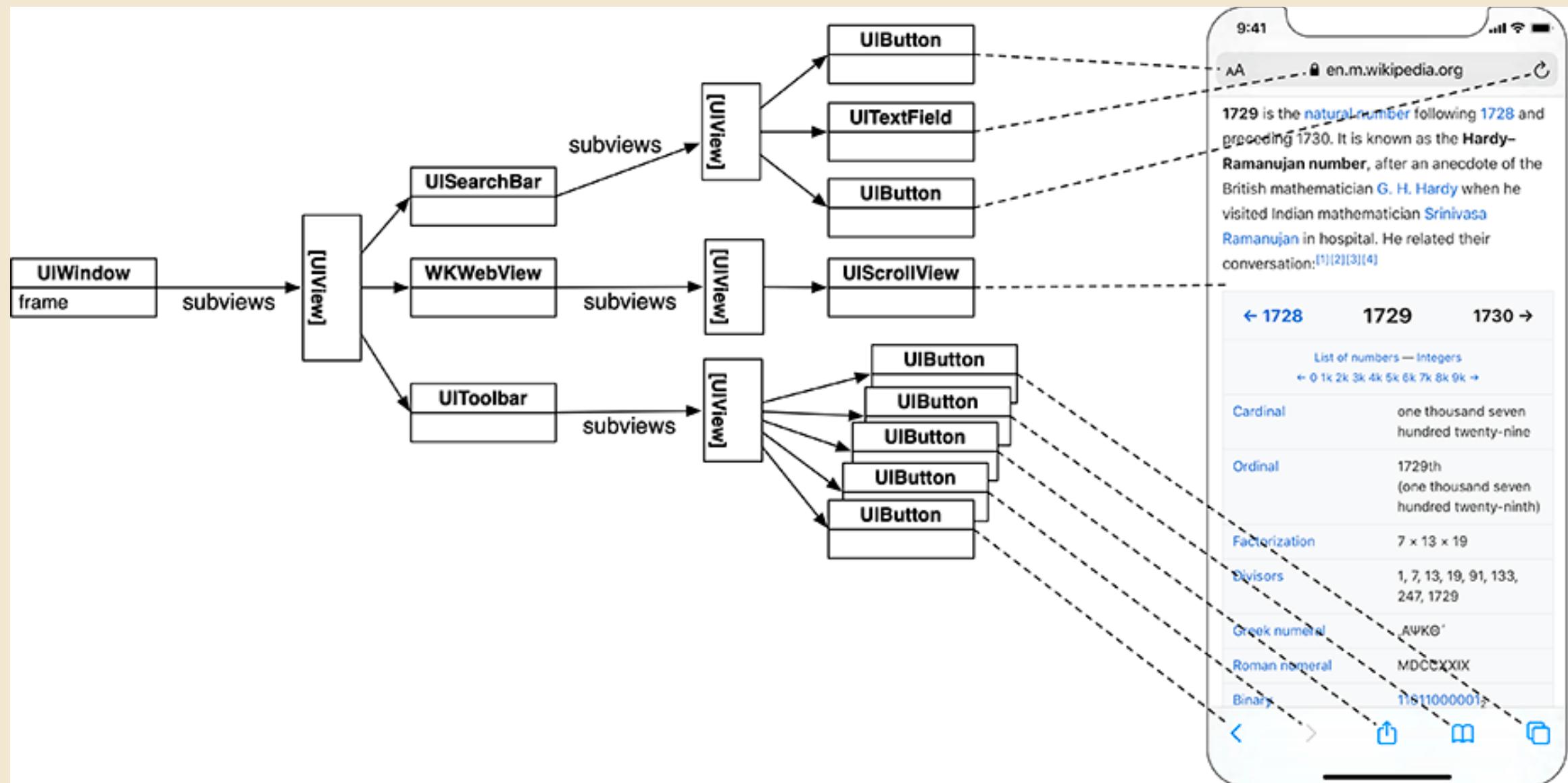
Steve Lai

2020/05/21 & 2020/05/28

Outline

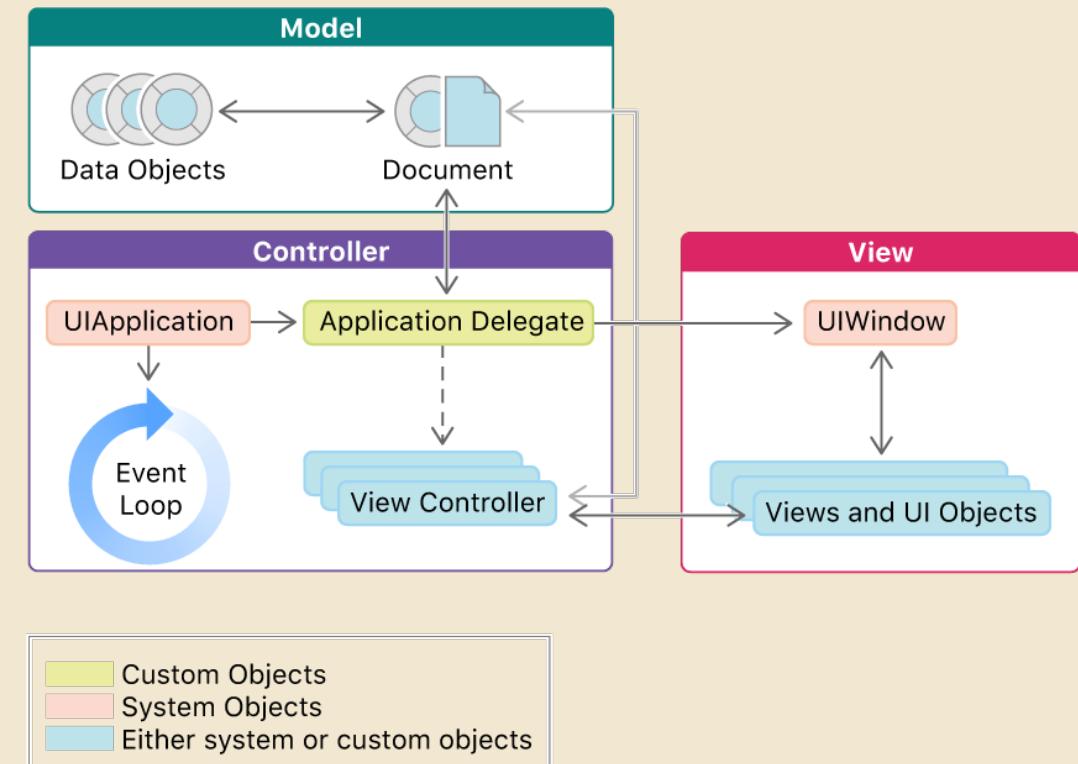
- iOS and Swift Overview
- UI/UX and design principle
- UI Views and Bars
- Case Study
- Advanced Topics
 - Delegate
 - ARC (Automatic Reference Counting)
- Implementation

The View Hierarchy



UIKit

- 對程式設計而言，UIKit 是主要處理相關的套件
 - UIKit 提供了
 - 建立和管理 APP 的主要部分
 - 建構 window 和 view 的架構
 - 管理和使用者的互動
 - 接收和回應事件
 - main run loop



Some Features of Swift

Variables/Constants

```
var myVariable = 42  
let myConstant = 12  
var typedVariable: Type  
let typedConstant: Type
```

Structures

```
struct MyStruct {  
}
```

IF Statement

```
if condition {  
    //do X  
} elseif otherCondition {  
    //do Y  
} else {  
    //do Z  
}
```

Functions

```
func aFunction(input: Type) -> Type {  
    //Do Something  
    return theOutput  
}
```

Classes

```
class myClass: SuperClass {  
}
```

Data Types

Int	23
Float	2.3
Double	3.1415926
Bool	true/false
String	"abc"
Array	[1,2,3]
Dictionary	[key: value]

Switch Statement

```
switch someVariable {  
    case 1:  
        //do X  
    case 2:  
        //do X  
    default:  
        //do X  
}
```

Loops

```
for variable in low...high {  
}  
  
for item in array {  
}  
  
while condition {  
}
```



www.appbrewery.co

User Experience (UX)

- Apple 將良好的使用者經驗帶到Apple的產品中
- 在廣告與品牌行銷上非常注重於使用者如何使用產品，以及使用者感受



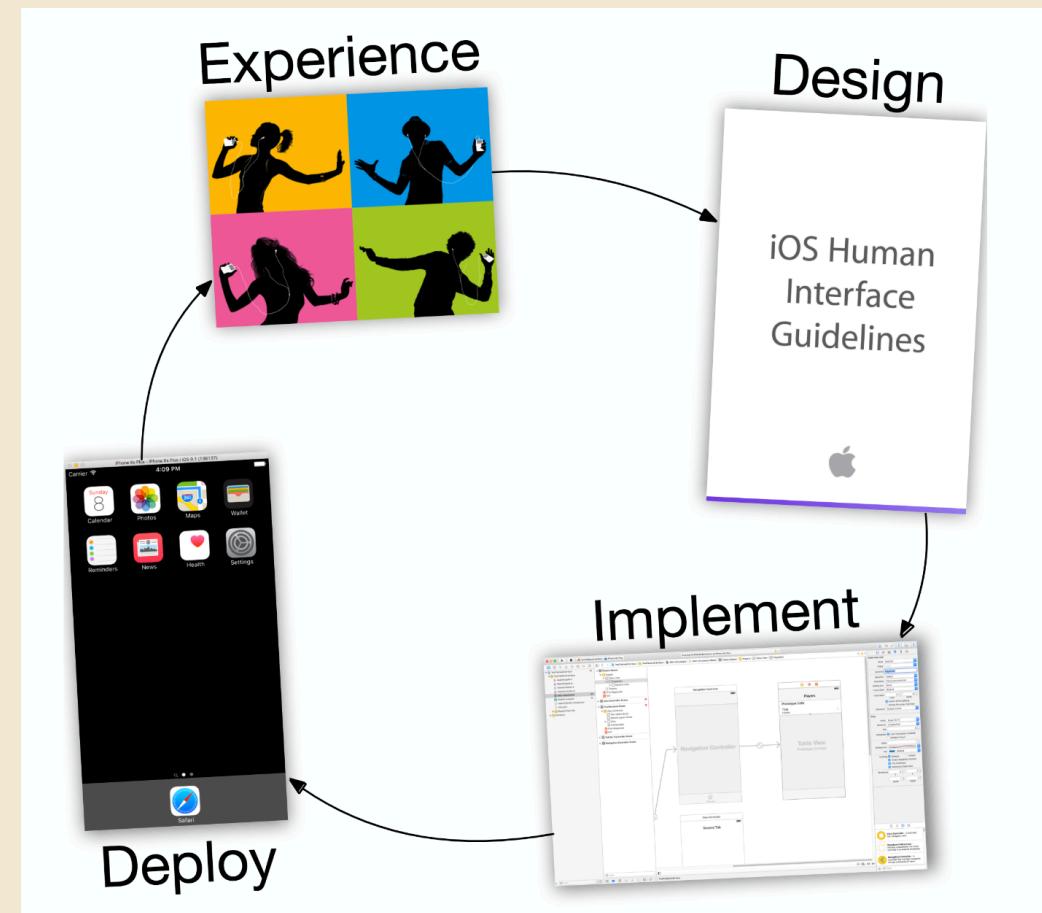
iOS Design Themes

- Three primary themes differentiate iOS from other platforms
- Clarity (清楚明瞭)
 - 每個尺寸的文字都清晰可辨
 - 圖標精確清晰，裝飾恰當
 - 強化功能的重點
- Deference (尊重內容)
 - 流暢的使用介面與互動可以幫助使用者理解內容，而不會互相競爭
 - 內容通常會佔滿畫面，並利用半透明與模糊效果
 - 利用漸層和陰影讓介面明亮輕巧，並確保突顯出內容
- Depth
 - 利用視覺層和逼真的動作傳達層次感
 - 觸控和發現，提高並且在不丟失上下文的情況下訪問功能和附加內容

Design Principles

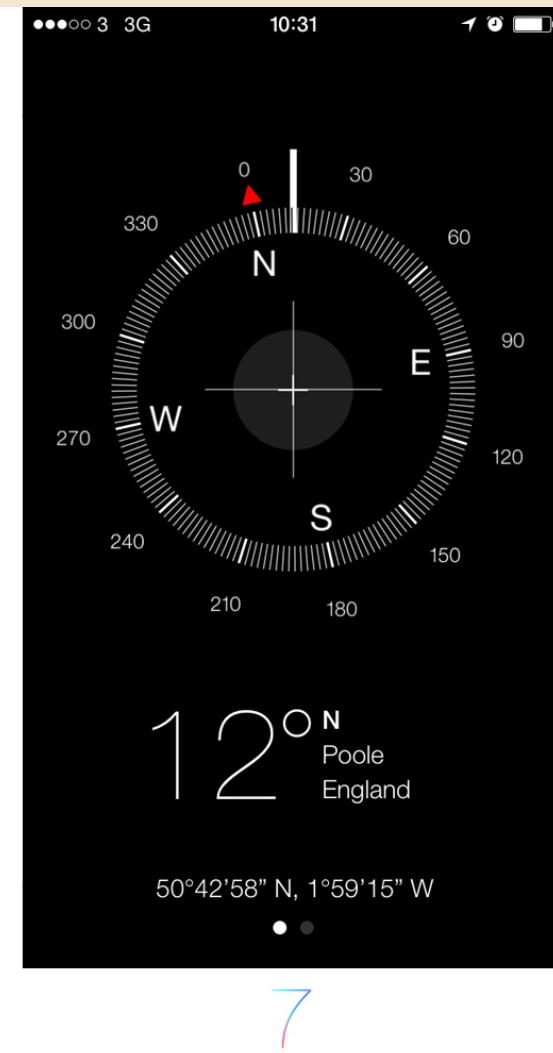
- Aesthetic integrity (設計美學完整性)
 - 外觀與功能的整合程度
 - 利用醒目且清楚的圖形讓使用者專注在功能上
 - 助於預測使用者行為
- Consistency (一致性)
 - 使用系統所提供之為使用者所熟悉的圖形、介面、文字格式與一致性的術語
 - 提供使用者預期的功能與使用方式
- Direct Manipulation (直接操作)
 - 使用者可以透過畫面直接的操作並看見結果，且能增進理解
 - 例如：觸碰螢幕或是旋轉螢幕，並立刻看到所改變的結果
- Feedback (回饋)
 - 回饋讓使用者知道抱持瞭解狀況
 - 透過內建的聲音或動畫讓使用者感知操作後的回饋
 - 例如進度表傳達長時間的運行
- Metaphors (隱喻)
 - 透過與系統或是真實世界一樣的隱喻讓使用者瞭解功能，並能夠快速學會
 - 例如左右滑動、切換開關等
- User Control
 - 讓使用者擁有掌控權
 - 比起直接接管或是決策，建議採取的動作或是警告危險後過會是更好的

iOS Design Concepts



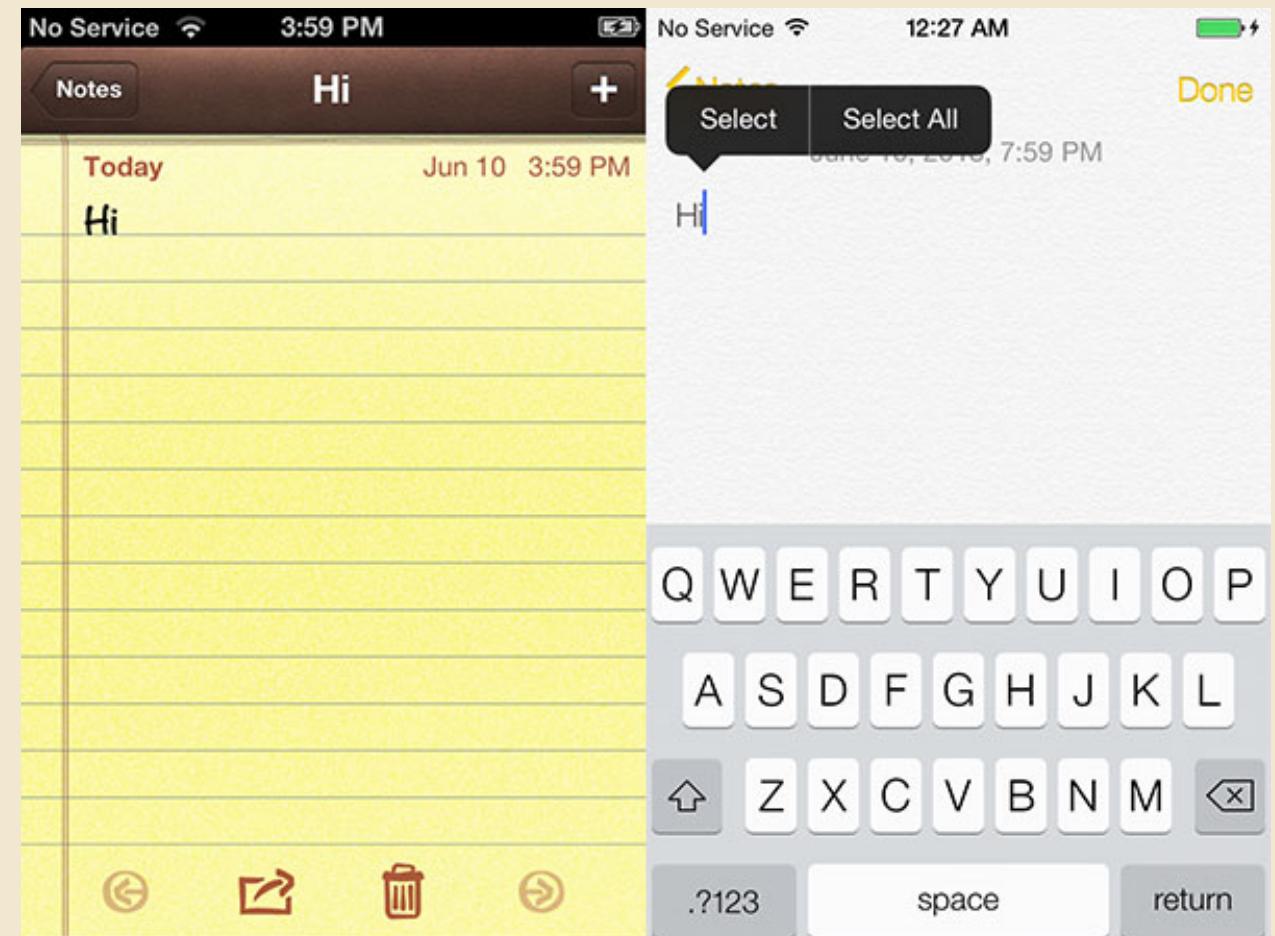
Apple design concepts

- 時常改變設計的策略
 - 寫實設計到平面設計



Apple design concepts

- 時常改變設計的策略
 - 寫實設計到平面設計



Apple design concepts

- 時常改變設計的策略
 - 寫實設計到平面設計



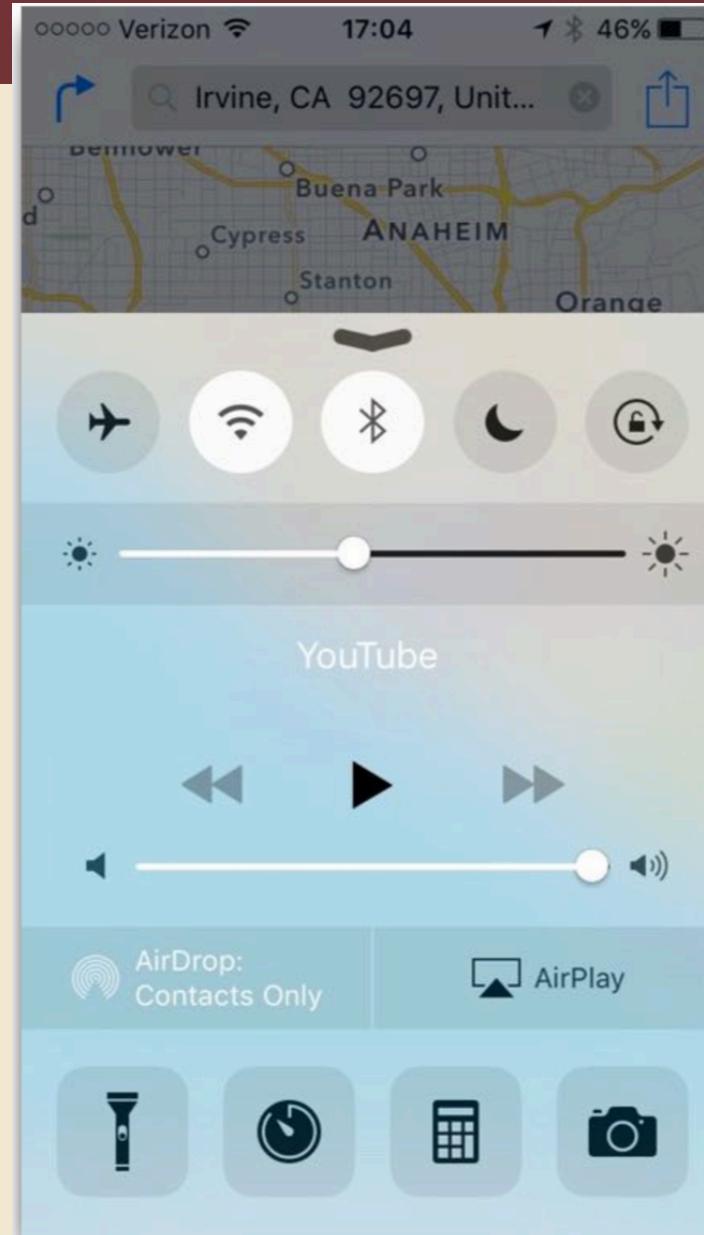
Defer to Content

- 使用整個螢幕
- 減少或消除擬真
- 利用半透明



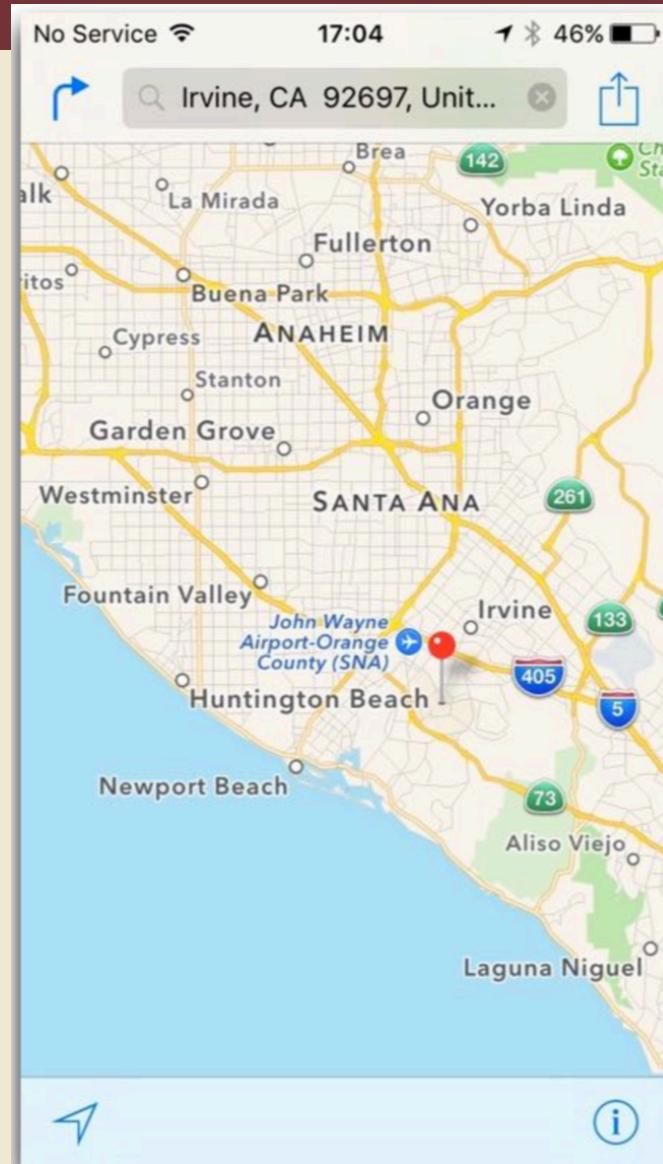
Defer to Content

- 使用整個螢幕
- 減少或消除擬真
- 利用半透明



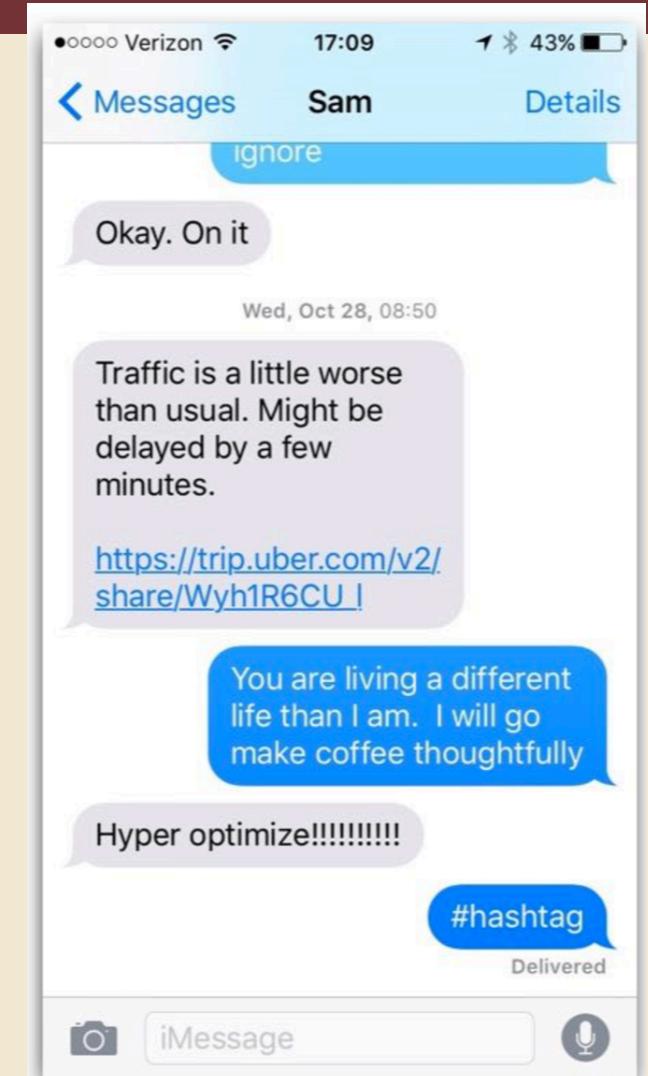
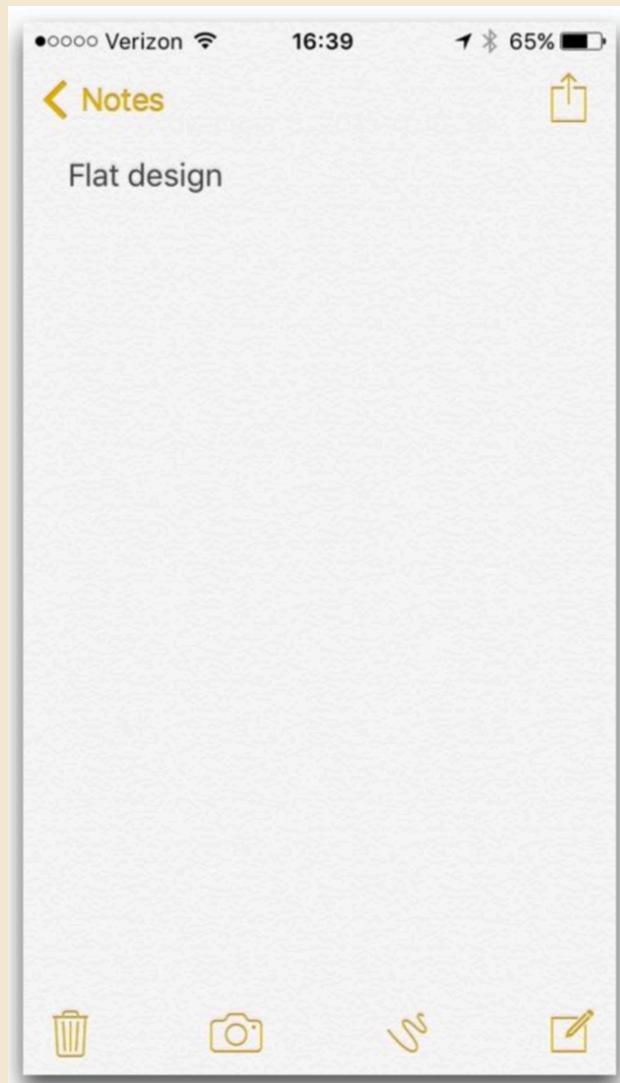
Defer to Content

- 使用整個螢幕
- 減少或消除擬真
- 利用半透明

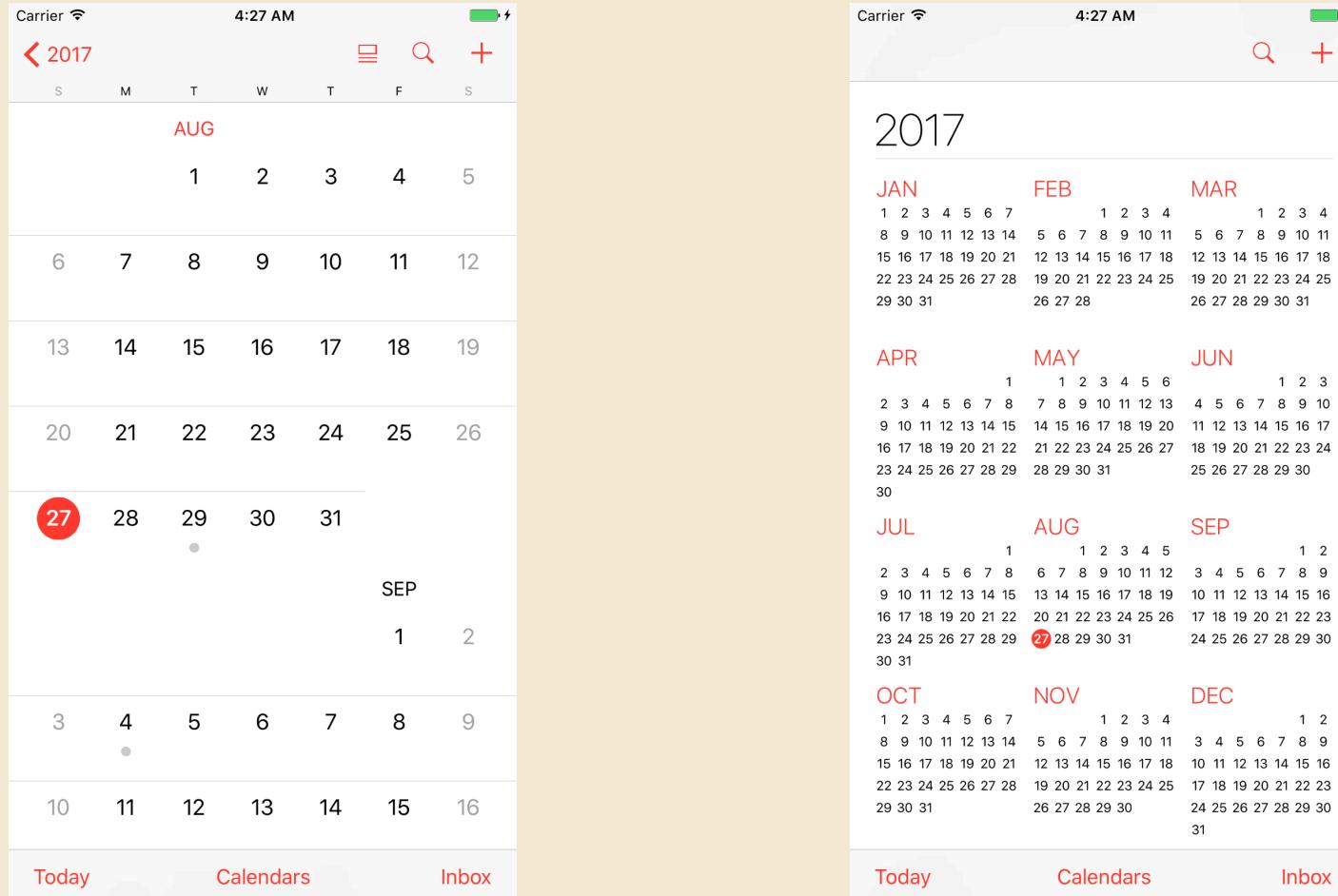


Promote Content Clarity

- 利用空白減少干擾
- 利用顏色
 - 相同作用
 - 標明同樣目的
- 利用無邊框按鈕

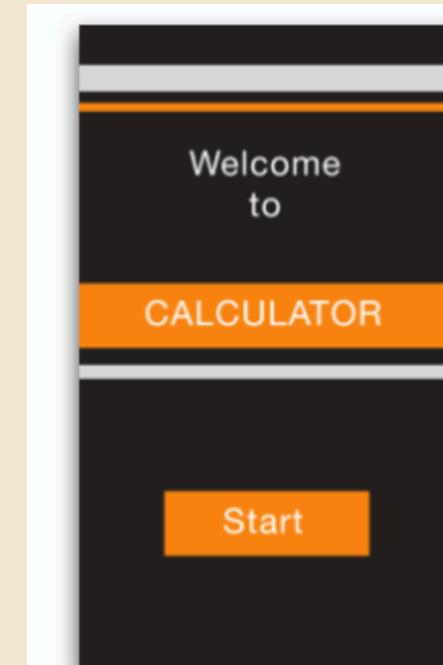


Use Depth to Reflect Content

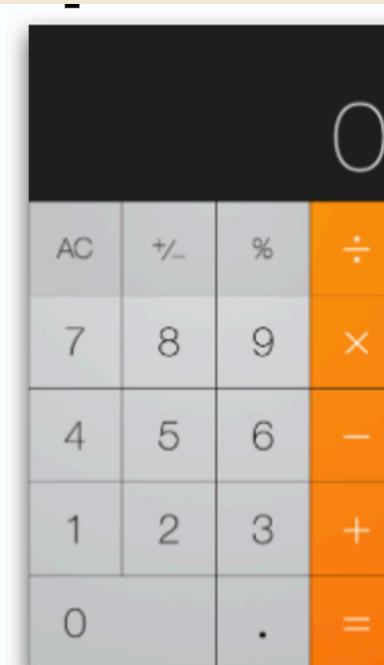


Start Instantly

- 使用者在開啟APP第一時間就開始打分數
- 不用使用無謂的開始畫面
- 直接進入APP功能畫面
- 進入之後不要求使用者重新開啟



Waste of Time



Start Instantly

Start Instantly

- 避免要求使用者填入可以自動從系統中取得的資訊
- 建立預設的設定
- 避免使用者到iOS設定APP設定此APP相關功能
- 避免在一開始就讓使用者閱讀某些協定或條款
 - 會嚇到且干擾使用者
 - 將這些協定放到App store

Start Instantly

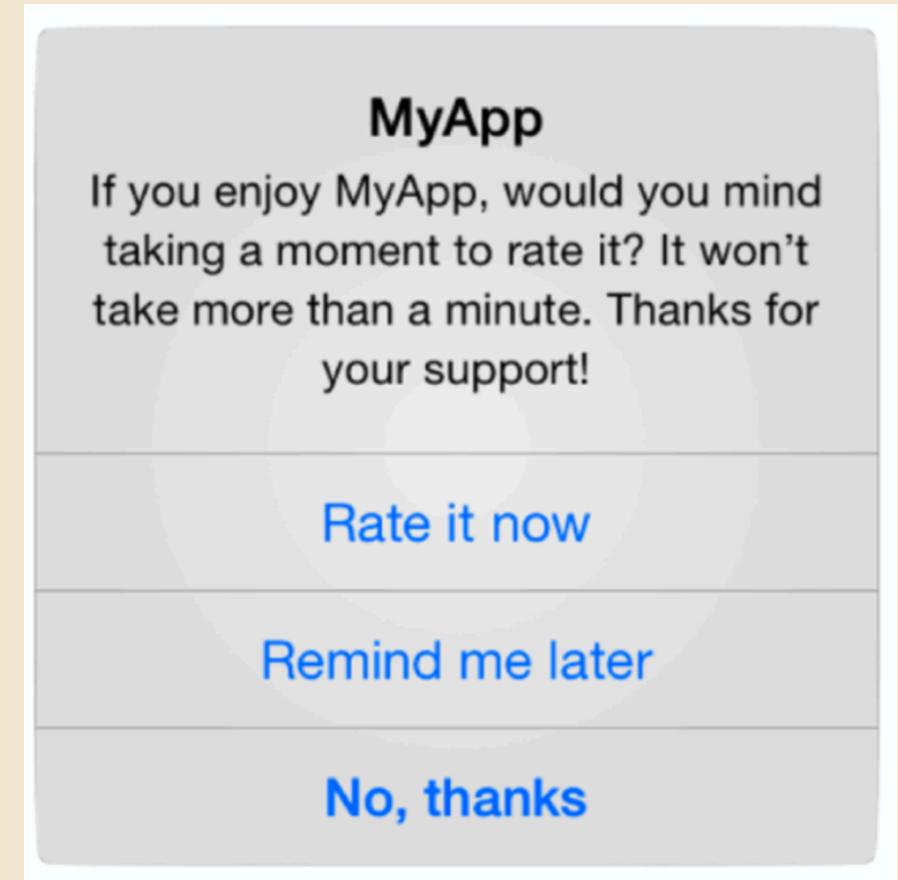
- 等待使用者輸入
 - 沒有必要之前不要求使用者登入
 - 沒有必要之前不詢問系統權限
- 解釋相關權限要求

Start Instantly

- 小心使用導覽
 - 是否太複雜
 - 是否圖示不清楚
 - 無法快速瞭解目的
- 如果一定需要導覽，也可能需要思考是否重新設計
- 如果非需要導覽不可
 - 簡單，基本
 - 夠就好
 - 可以跳過
 - 不要出現太多文字

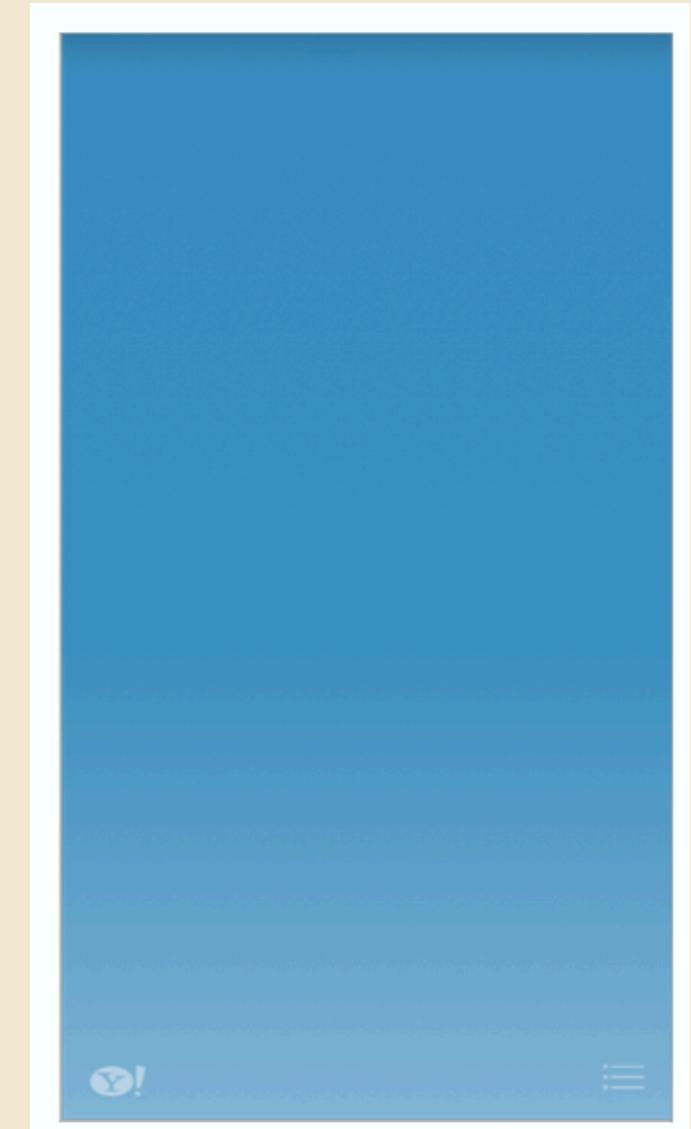
Start Instantly

- 要求評分
 - 好的評分可以增加能見度與使用率
- 不要再一開始就要求評分
- 可以再使用過某些功能後再要求



Launching

- 越快越好
- 進入畫面不應該包含
 - 關於的視窗
 - 登入或開始畫面
 - logo或品牌廣告



Restarting

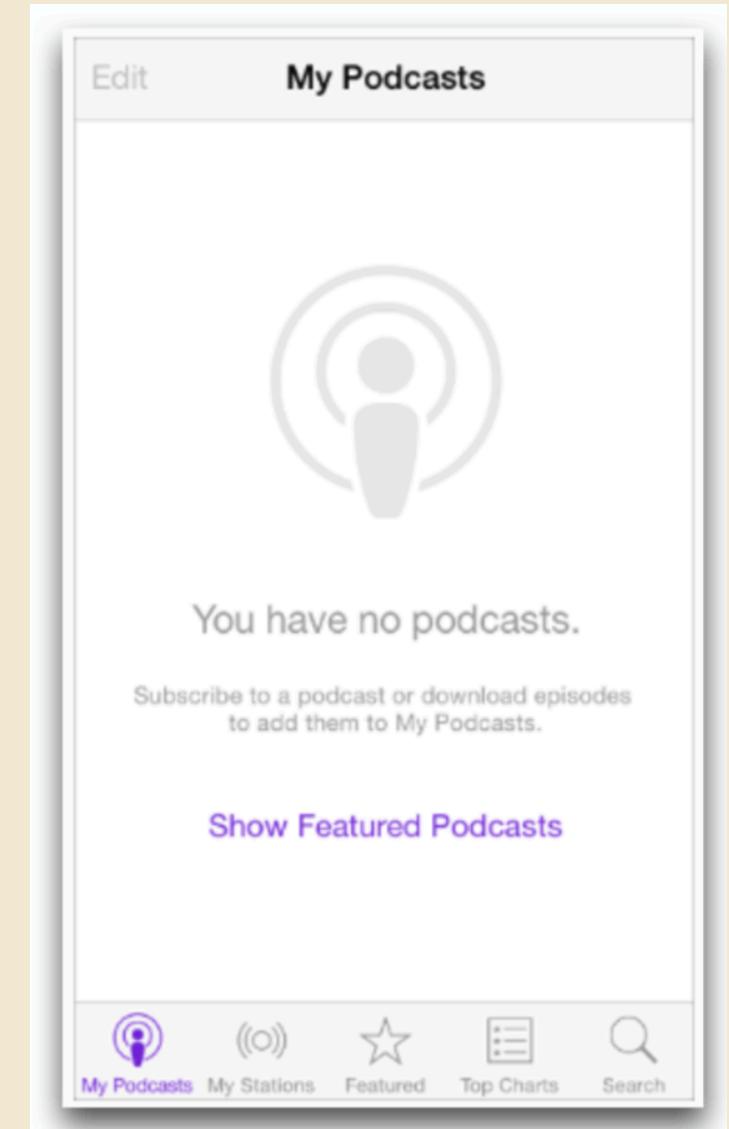
- 重新開始在使用者上次離開的狀態
 - 需要將狀態儲存
- 避免使用中重新啟動
 - 看起來像是當掉

Stopping

- iOS APP沒有離開或關閉的選項
 - 使用者可以利用home等方式離開
- APP自動關閉看起來也像當掉
- 使用者可能隨時會關閉
 - 週期性的儲存狀態

User Central

- 讓使用者決定
- 如果有錯誤
 - 告訴他們原因
 - 告訴他們如何修理
 - 等他們自己處理
- 如果有功能無法使用
 - 告訴他們原因
 - 怎樣解決



Familiarity

- 利用使用者所熟悉的
 - 控制方式
 - 表達方式
- 並找真的使用者測試
- 新的方法不一定會比較好
 - 必須確定真的是好的，有用的

Visibility

- 當使用者知道有選項時，使用者才會知道可以做什麼
- Navigation 選項
 - 連結看起來是可以按的
 - 可以拖拉的物件看起來可以拖拉
 - flat design比較難達成，但是是值得努力的
 - 利用動畫等方式

Discoverability

- 在按鈕上顯示文字，讓使用者知道功能
- 利用選單讓使用者知道所有動作與功能
- 避免不必要或是必需要學習的手勢動作

Consistency

- APP功能的一致性
 - 在各個頁面中，同一種顏色代表同一種功能或意思
 - 同一種字型、大小、樣式...
 - 用同一種圖示代表同一件意思
- 使用標準用法
 - 當SDK改變，無須重新設計

Support Expertise

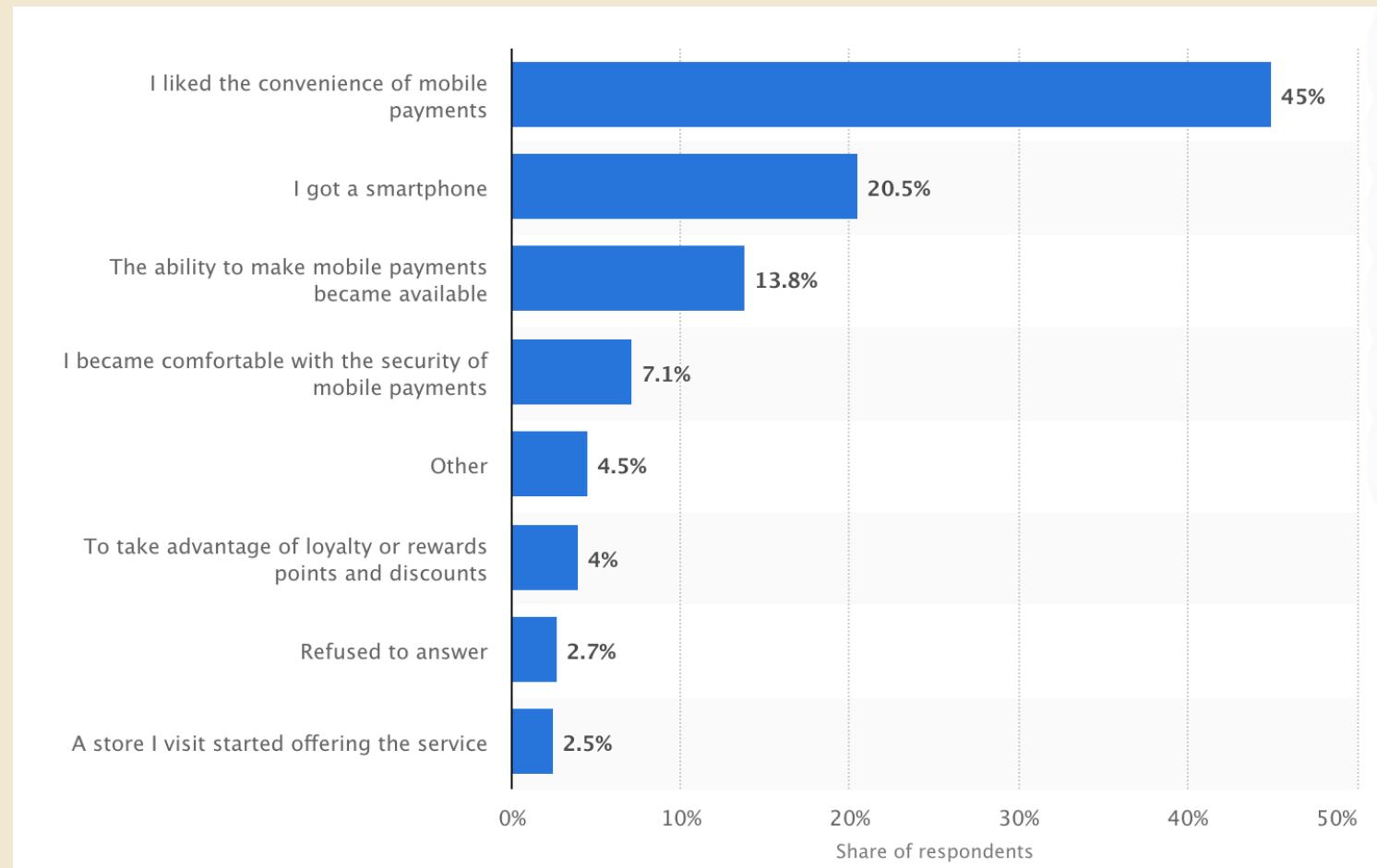
- 當使用者越來越熟悉APP，他應該要很能夠很順利的使用、學習、發現應用
 - 使用捷徑
 - 進階的用法
 - 特定的手勢
 - 為了加速達成功能

Feedback

- APP以及其物件應該要能夠立即有反應
 - 物件移動
 - 按鈕或是字型改變
 - 存取狀態
- 否則使用者會覺得疑惑甚至會覺得APP無法運作

Mobile Banking

- 根據 Juniper research 調查，在2019年有17.5億人使用手機操作銀行服務
- 有45%人覺得方便是主要的原因

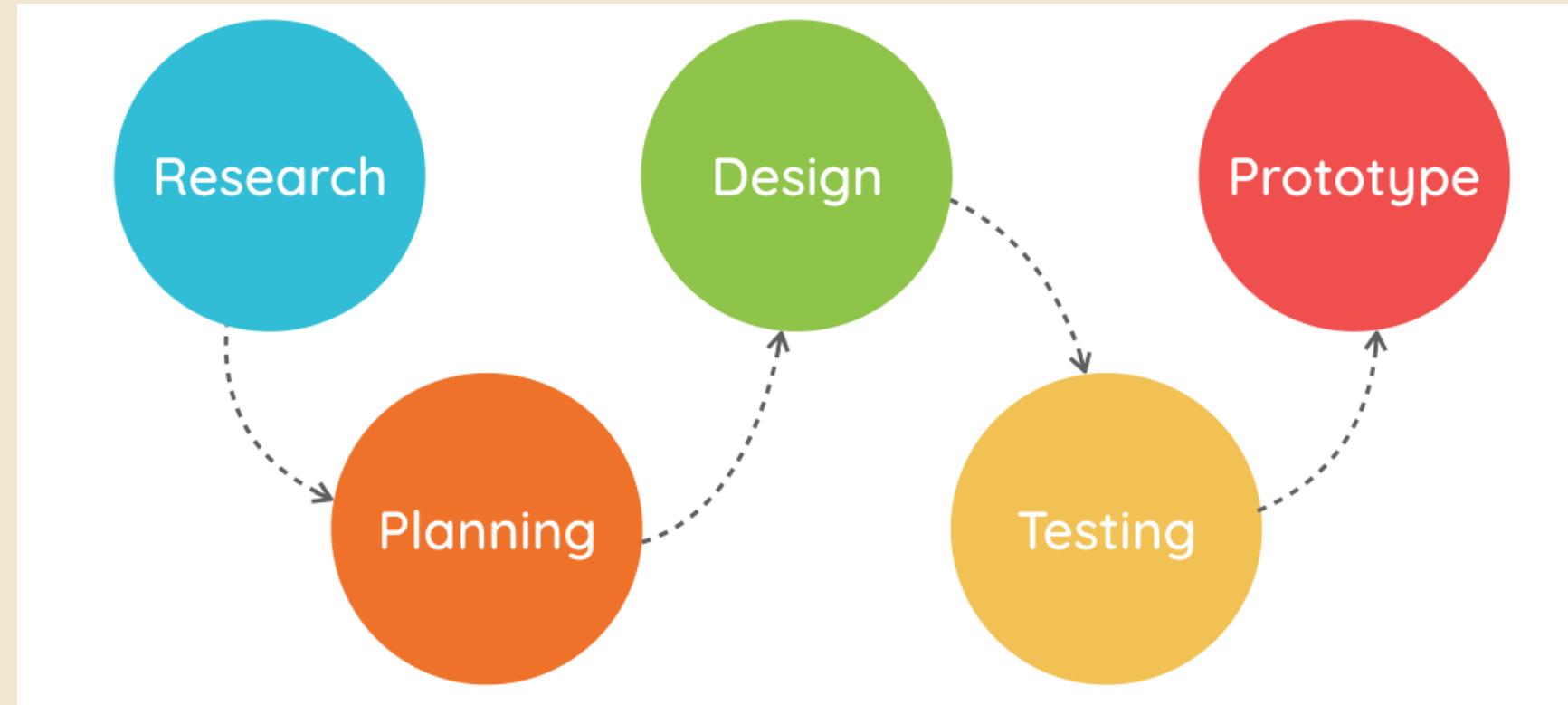


Mobile Banking App Development: Core Features

- Security
 - 確保與網頁應用程式一樣的安全等級
 - 密碼強度檢查
 - 使用數位簽章
- Simplicity
 - 盡可能減少功能
 - 直覺的操作介面
 - 客制化體驗
 - 個人化體驗
 - 降低每一步驟反應時間

Stages of the Mobile Banking Application Development

- 領域研究
- 計畫
- 設計 (UI/UX)
- 測試
- 市場調查與上線
- 維護與支援



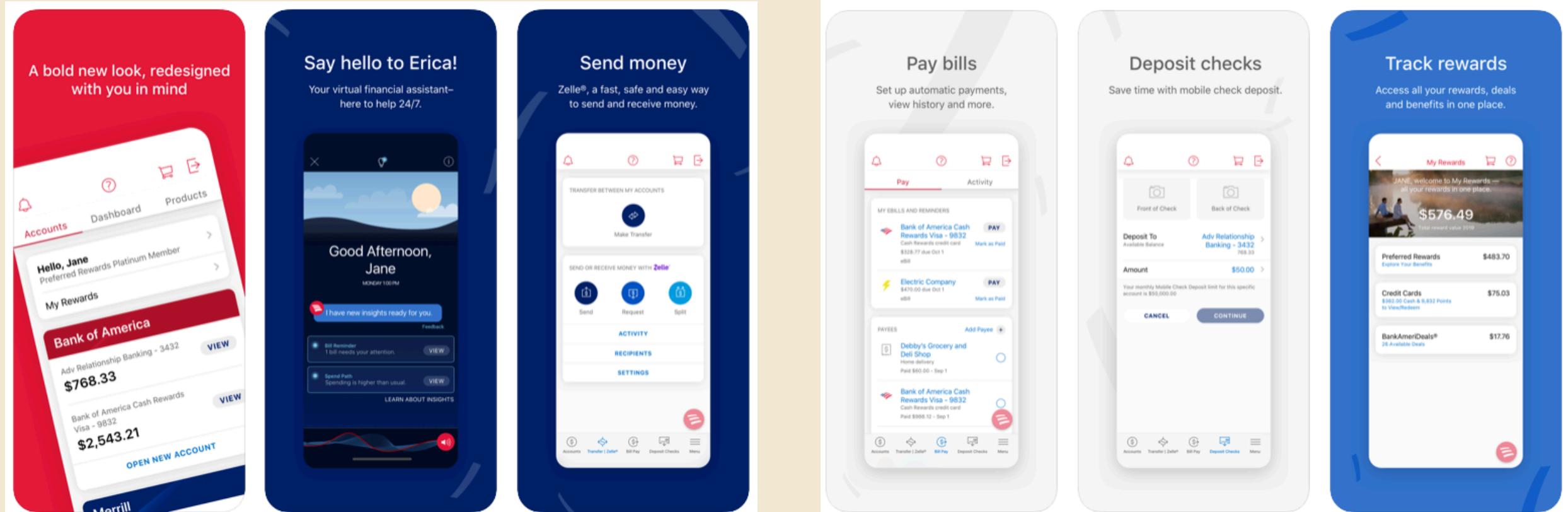
<https://blog.prototypyr.io/monkiri-ui-case-study-71fcb100f1a1>

<https://www.sam-solutions.com/blog/the-guide-to-developing-a-banking-app/>

Challenges When Building a Fintech App

- 別一次使用者太多資訊與功能
 - 如何折衷的呈現資料
- 呈現有趣的樣貌
- 使用者所使用的語言
 - 減少使用術語，使用一般大眾熟知的語言
 - 根據不同層級客戶客制化
- 讓使用者感到安全
 - 解釋為何需要相關資料
 - 資料收集的用意
- 幫助使用者做正確的決定
 - 提供理性的資訊

Case Study: App of Bank of America



<https://apps.apple.com/tw/app/bank-america-mobile-banking/id284847138#?platform=iphone>

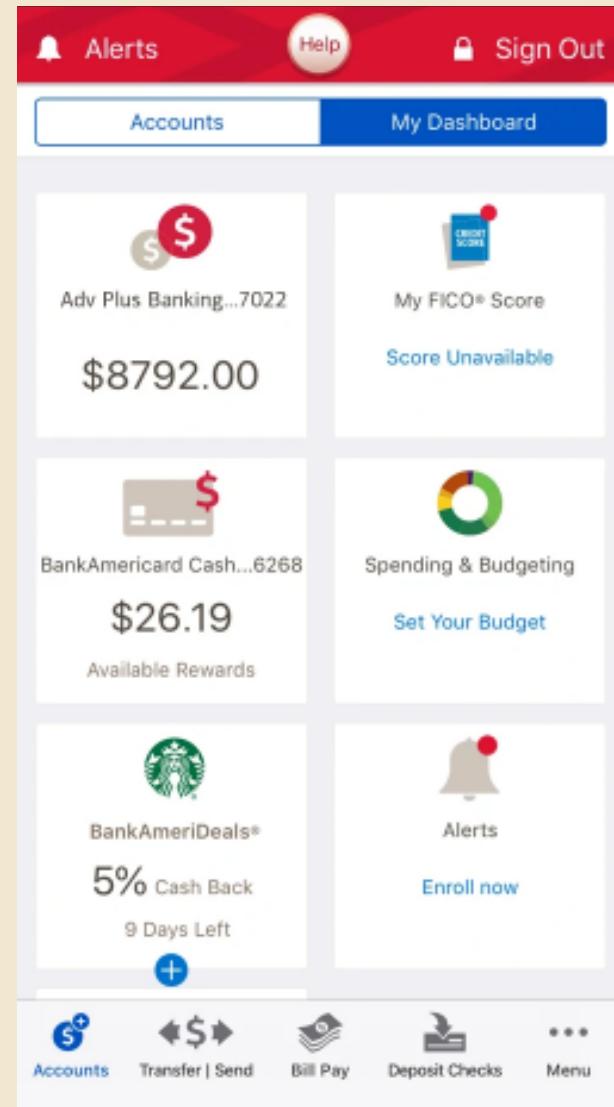
Copyright © 2020 Steve Lai. All rights reserved.

Core Features of App of Bank of America

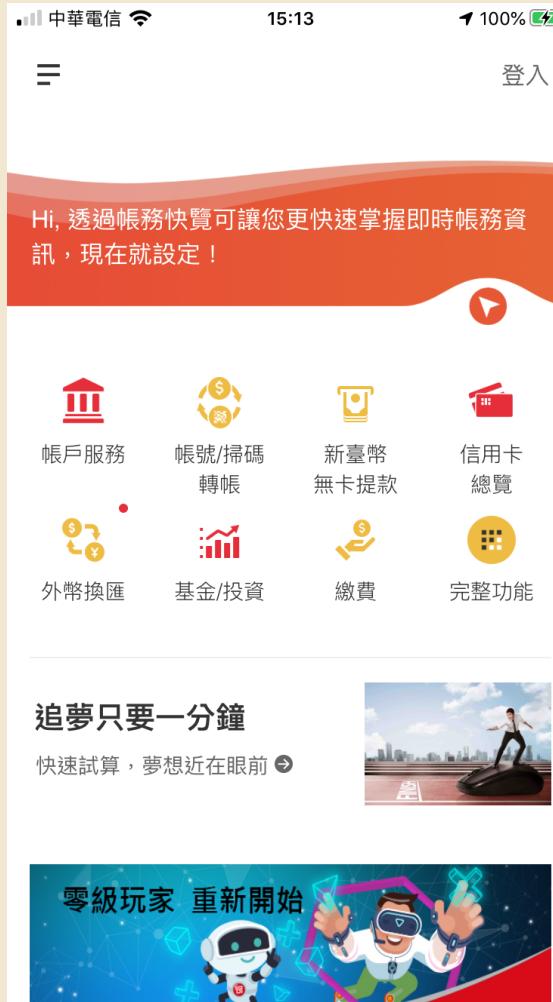
- Control of balances and activities
- Activation or replacement of cards
- Money transfer with Zelle
- eBills payment
- Mobile check deposit
- Virtual financial assistant Erica
- ATMs and financial centers finder
- Cashback with BankAmeriDeals
- Online ID or passcode change
- Fingerprint sign-in
- Security guarantee

The Advantages of App of Bank of America

- Simple UI
 - 第一印象
 - 簡單、明確且友善的介面
 - 可以透過簡單的按鈕找到需要的功能
- Easy Account Management
- Mobile Transfer & Payment
- Check Deposit
- Security

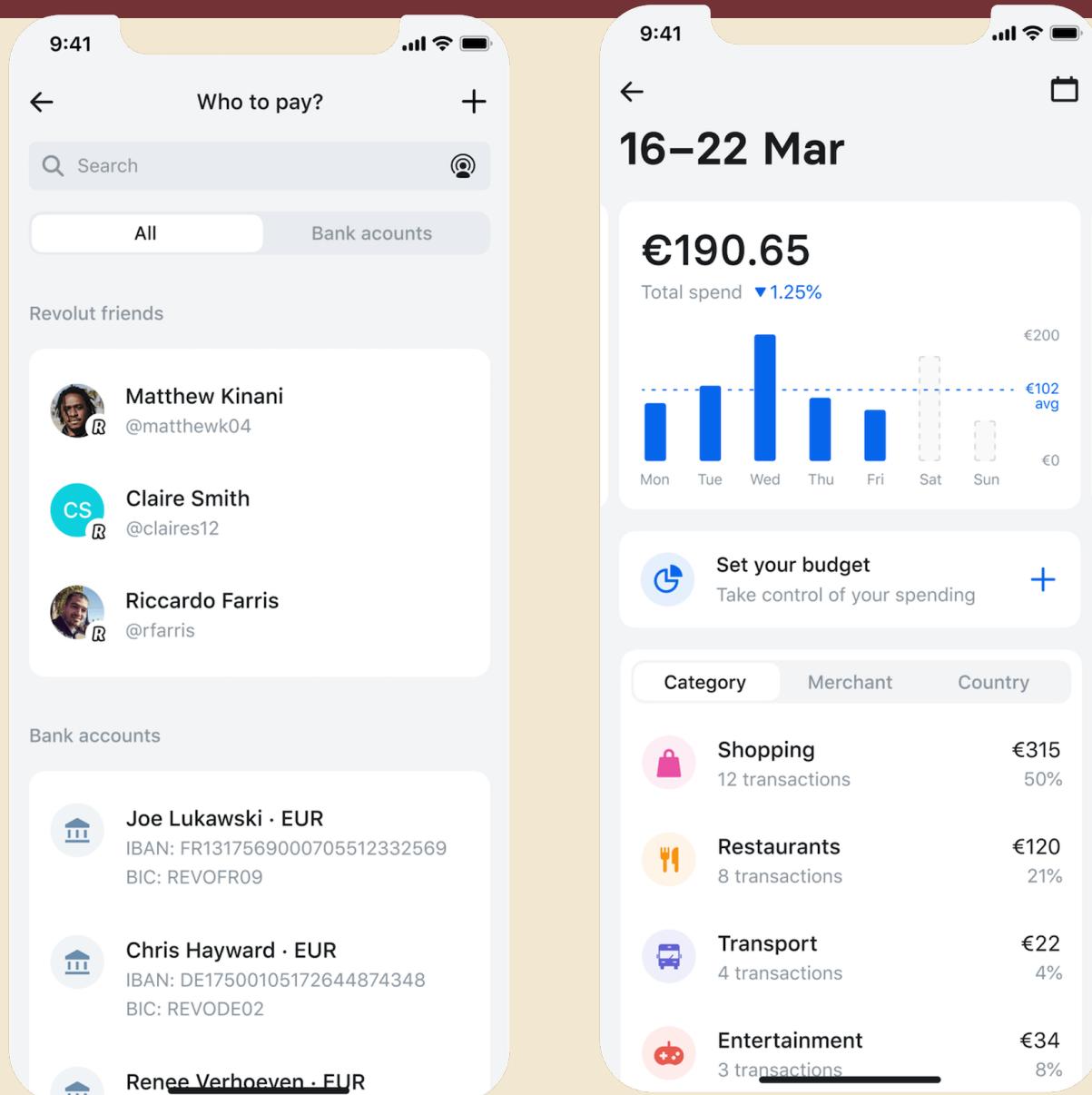


Case Study: App of Taiwan's Banks



Great Patterns and Examples

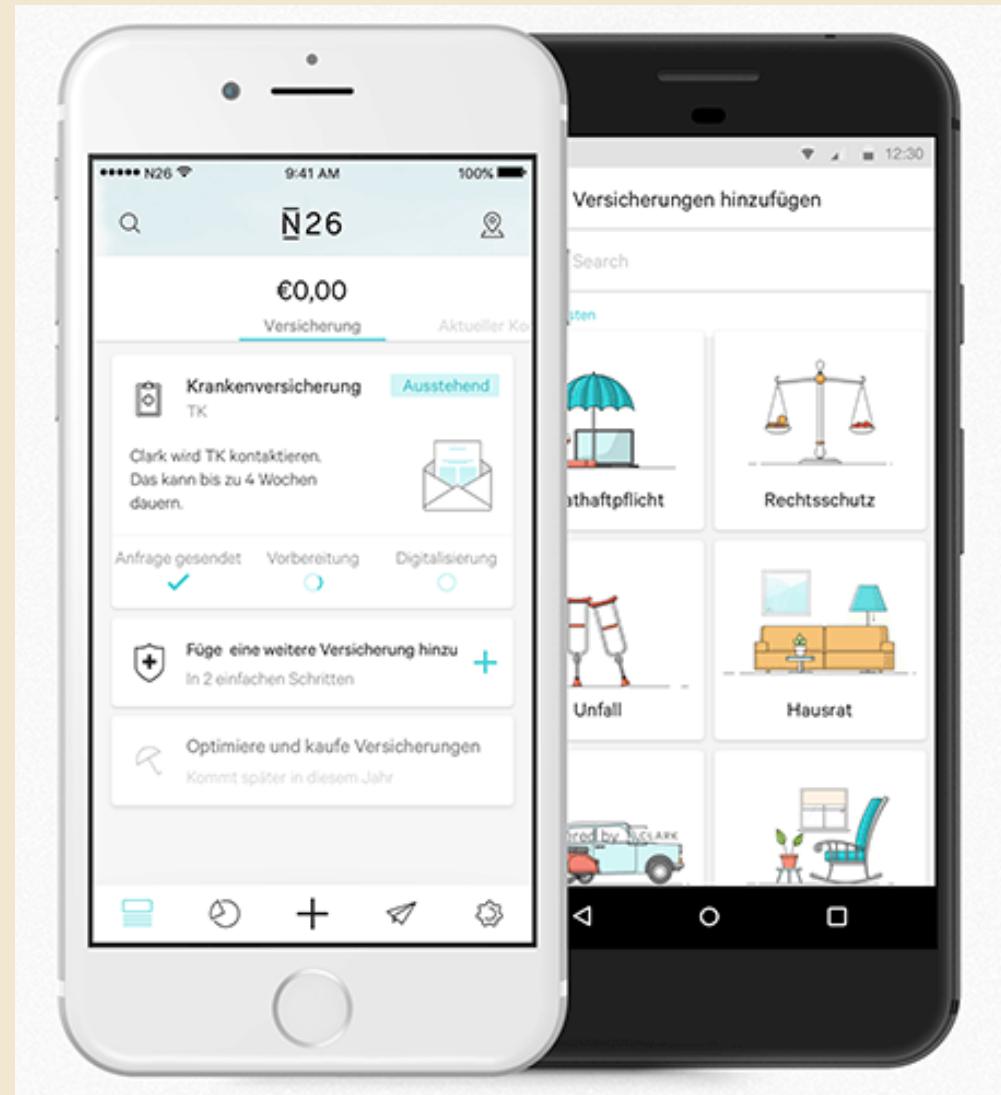
- Revolut
 - Heuristics and trendy aesthetics



<https://www.justinmind.com/blog/banking-app-design/>

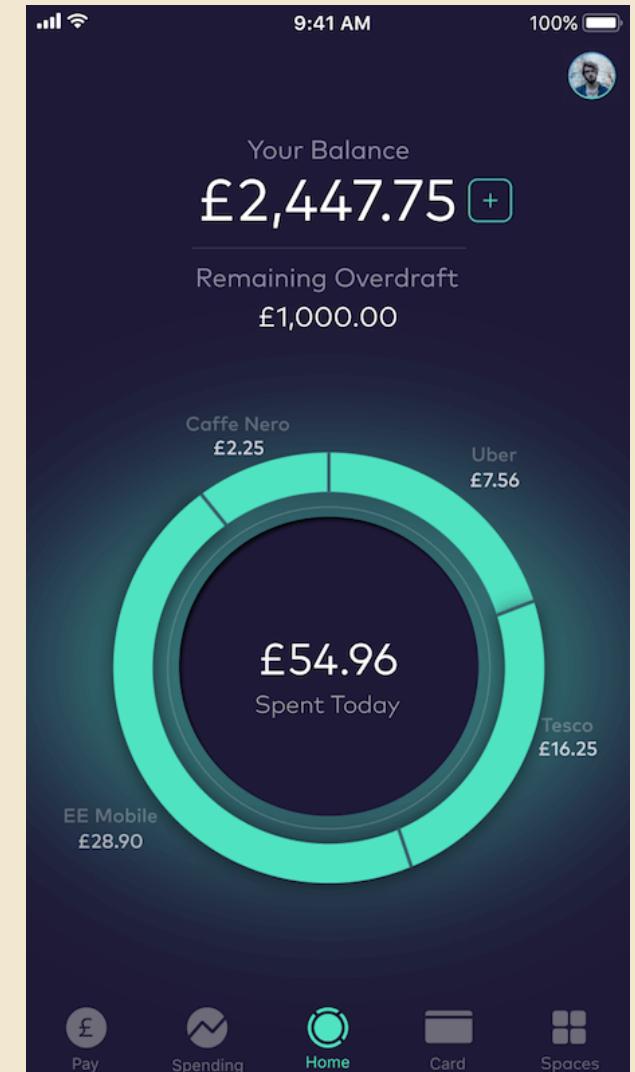
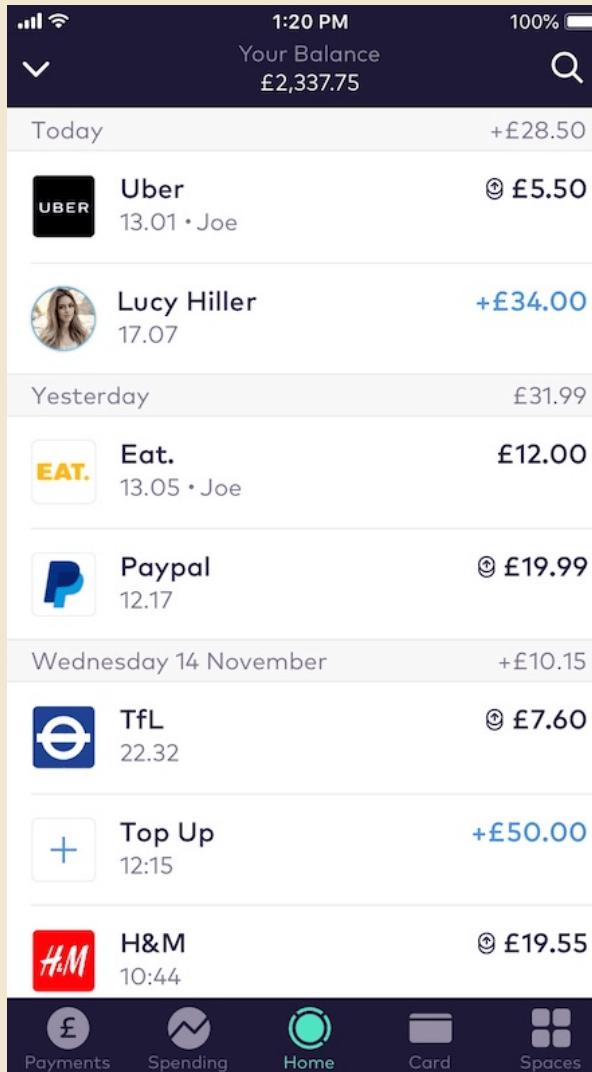
Great Patterns and Examples

- Cards instead of lists: N26



Great Patterns and Examples

- Landing pages with statistics: Starling Bank



Great Patterns and Examples

- Monese

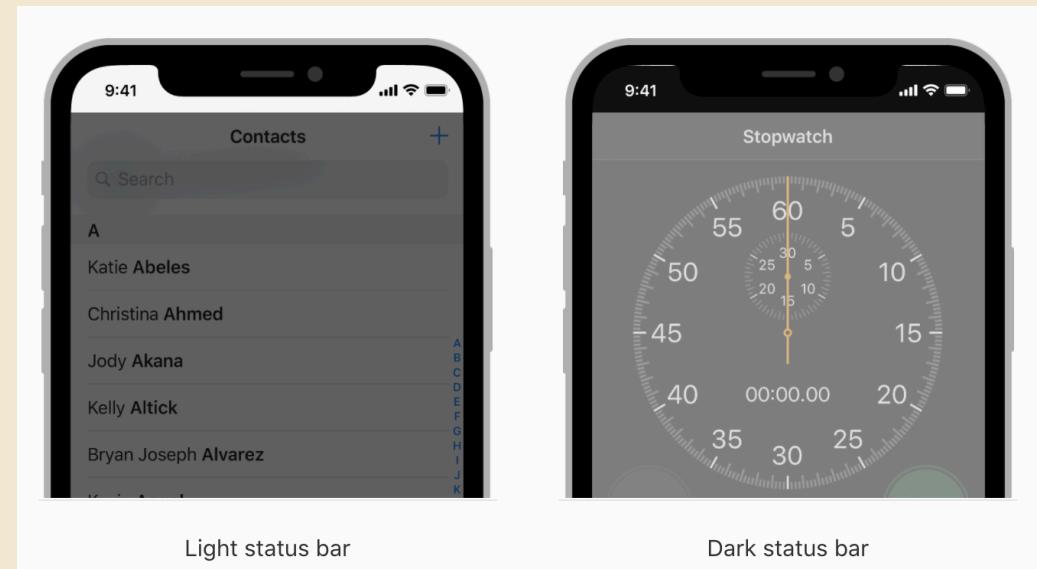
The image displays two screenshots from the Monese mobile application. The left screenshot shows the 'Choose account' screen, which lists four accounts: 'UK' (selected, showing £2,935.00), 'Avios' (linked to British Airways), 'PayPal' (linked to myemail@gmail.com), and 'Eurozone'. An 'Add account' button is at the bottom right. The right screenshot shows the 'Payments' screen, featuring tabs for 'Pay', 'Request', and 'Recurring' (selected). It lists 'RECENTLY PAID TO' and 'READY TO PAY' sections, each with four entries (Aurelio Offutt, Faiza Ainsworth, Matthew, Camille Mack) showing their names, profile icons, currencies (GBP or EUR), and transaction details. An 'Add new' button is located at the bottom right.

Bars

- Status Bars
- Navigation Bars
- Toolbars
- Tab Bars
- Search Bars
- Scope Bar

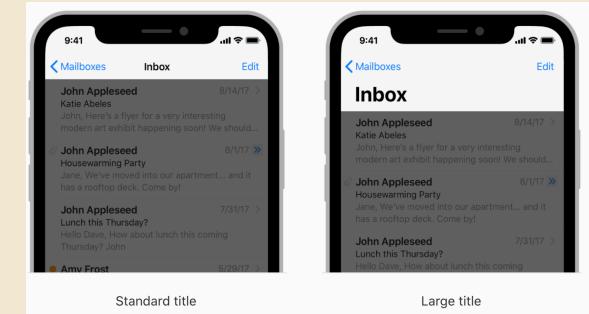
Status Bars

- Use the system-provided status bar
- Set globally in UIApplication
 - requires info.plist
- Set per view in view controllers
 - preferredStatusBarStyle
- Status have light or dark styles
- Obscure content under the status bar
- Consider temporarily hiding the status bar when displaying full-screen media
- Avoid permanently hiding the status bar



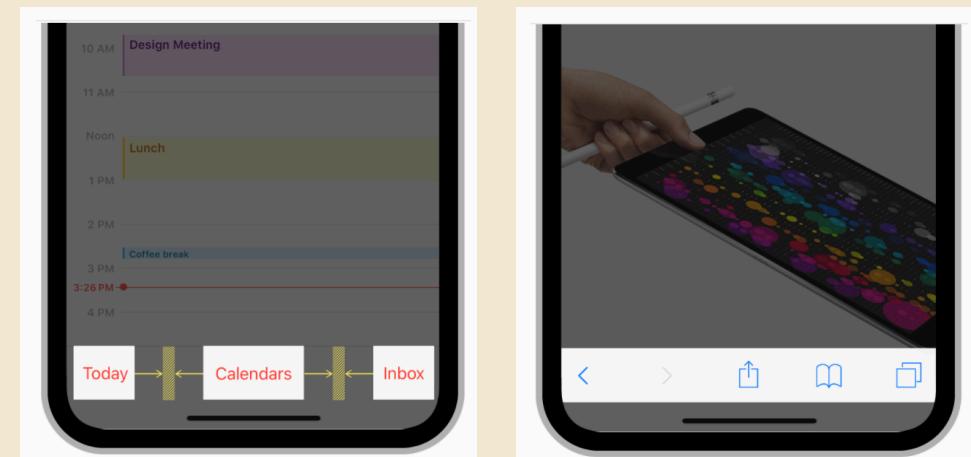
Navigation Bars

- Appears at the top of an app screen, below the status bar, and enables navigation through a series of hierarchical screens.
 - Back button, often labeled with the title of the previous screen, appears on the left side of the bar
 - The right side of a navigation bar contains a control, like an Edit or a Done button, for managing the content within the active view
- Tips
 - Consider temporarily hiding the navigation bar when displaying full-screen content
 - Avoid crowding a navigation bar with too many controls
 - Use the standard back button
 - Don't include multi-segment breadcrumb paths
 - Give text-titled buttons enough room
 - Consider using a segmented control in a navigation bar to flatten your app's information hierarchy



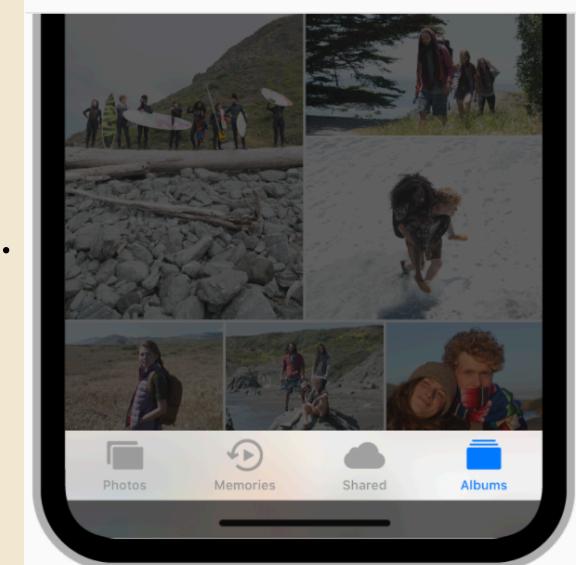
Toolbars

- A toolbar appears at the bottom of an app screen and contains buttons for performing actions relevant to the current view or content within it
- Toolbars are translucent, may have a background tint, and often hide when people are unlikely to need them
- Tips
 - Provide relevant toolbar buttons
 - Consider whether icons or text-titled buttons are right for your app
 - Avoid using a segmented control in a toolbar
 - Give text-titled buttons enough room



Tab Bars

- A tab bar appears at the bottom of an app screen and provides the ability to quickly switch between different sections of an app
- Tab bars are translucent, may have a background tint, maintain the same height in all screen orientations, and are hidden when a keyboard is displayed.
- A tab bar may contain any number of tabs, but the number of visible tabs varies based on the device size and orientation.
- If some tabs can't be displayed due to limited horizontal space, the final visible tab becomes a More tab, which reveals the additional tabs in a list on a separate screen.
- Tips
 - In general, use a tab bar to organize information at the app level.
 - Use a tab bar strictly for navigation.
 - Avoid having too many tabs.
 - Don't hide a tab bar when people navigate to different areas in your app.
 - Don't remove or disable a tab when its function is unavailable.
 - Always switch contexts in the attached view.
 - Make sure tab bar icons are visually consistent and balanced.
 - Use badging to communicate unobtrusively.



System Icons

- In iOS 13 or later, prefer using SF Symbols to represent tasks and types of content in your app.
- If your app is running in iOS 12 or earlier, follow the guidance.

<https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/system-icons/>



Toolbar Icons

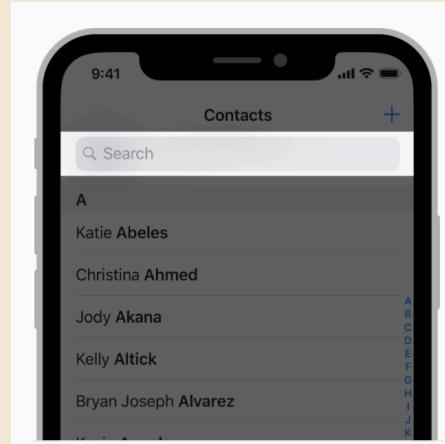
Icon	Name
↑	Action (Share)
+	Add
Bookmark icon	Bookmarks
Camera icon	Camera
Cancel icon	Cancel
Compose icon	Compose
Done icon	Done
Edit icon	Edit
Fast Forward icon	Fast Forward
Organize icon	Organize
Pause icon	Pause

Tab Bar Icons

Icon	Name
Bookmarks icon	Bookmarks
Contacts icon	Contacts
Downloads icon	Downloads
Favorites icon	Favorites
Featured icon	Featured
History icon	History
More icon	More
Most Recent icon	Most Recent
Most Viewed icon	Most Viewed
Search icon	Search
Top Rated icon	Top Rated

Search Bars and Scope Bar

- A search bar allows people to search through a large collection of values by typing text into a field.
- A search bar can be displayed alone, or in a navigation bar or content view
- Tips
 - Use a search bar instead of a text field to implement search.
 - **Enable the Clear button.**
 - **Enable the Cancel button when appropriate.**
 - **If necessary, provide hints and context in a search bar.**
 - **Consider providing helpful shortcuts and other content below a search bar**
- A scope bar can be added to a search bar to let people refine the scope of a search.

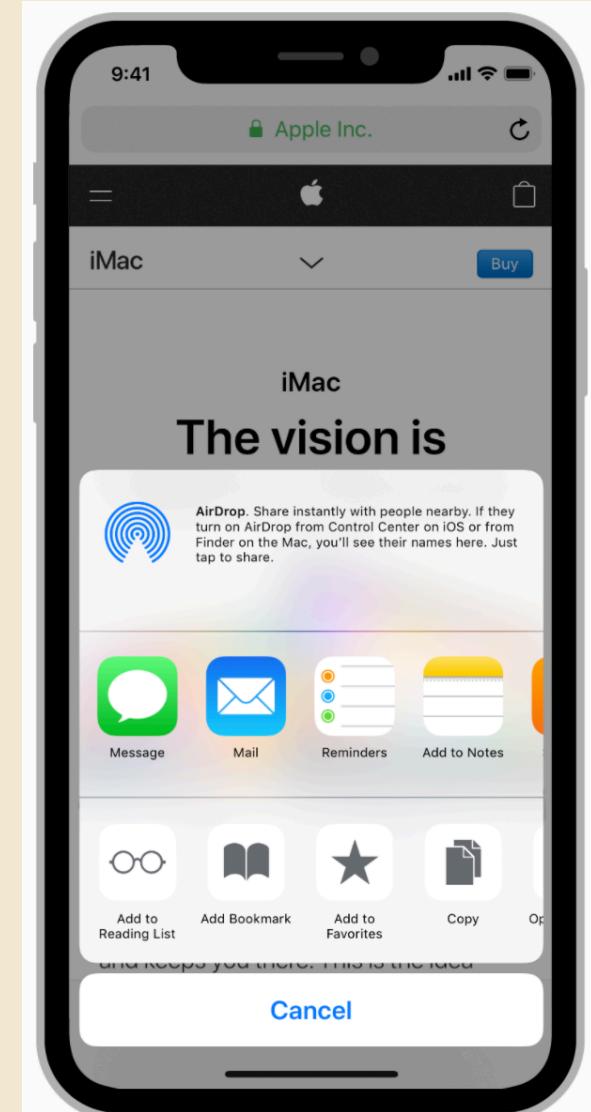
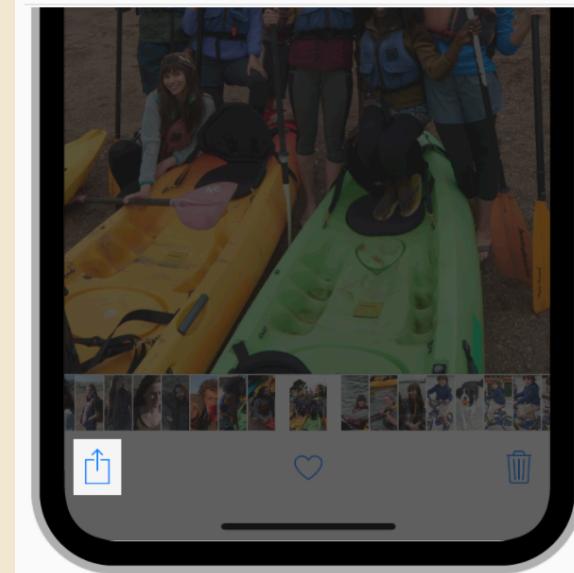


Content View

- Activity/ Activity View Controller
- Collection View
- Container View Controller
- Image View
- Map View
- Page View Controller
- Popover
- Scroll View
- Spilt View
- Table View
- Text View
- Web View

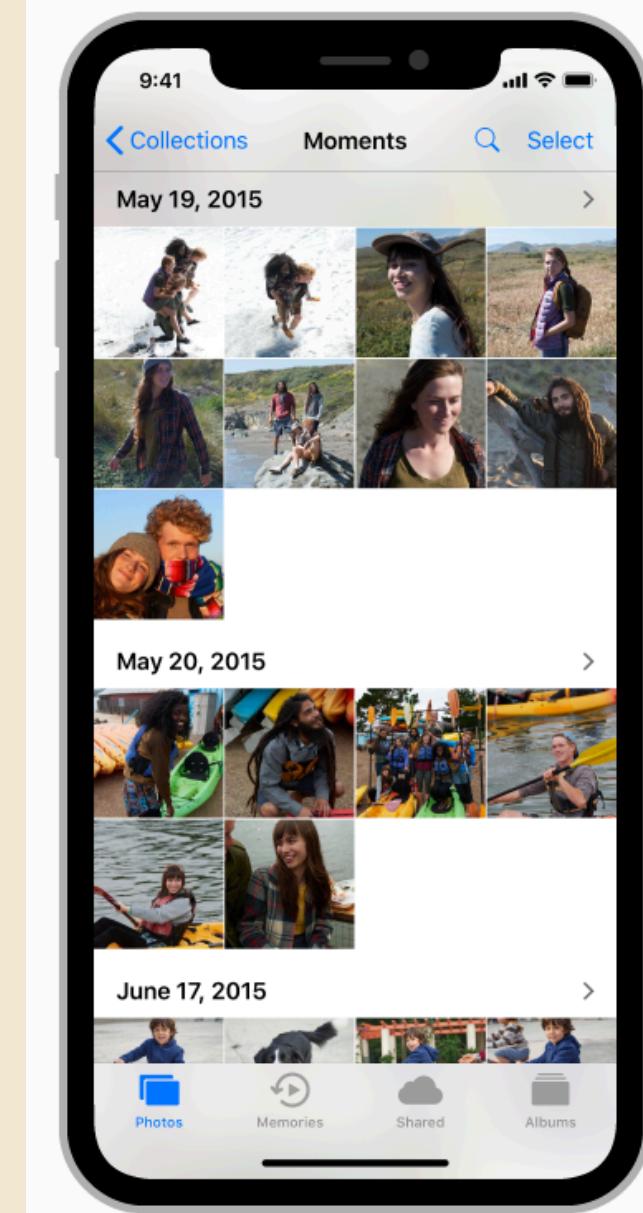
Activity/ Activity View Controller

- 提供對應的內容相容的功能
 - APP向iOS註冊能夠處理某種內容
 - 提供icon
 - APP必須處理這種功能



Collection View

- 客制化的方式顯示物件
- table or list 的其他方式
- 可以制定顯示物件的大小

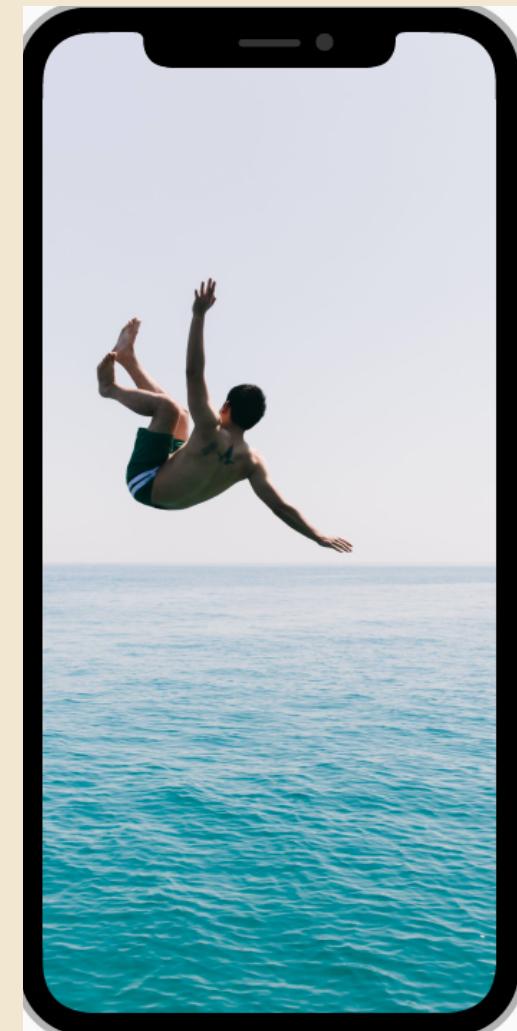


Container View Controller

- A base class for
 - Tab Bar View Controllers
 - Navigation View Controllers
 - Split View Controllers

Image VIew

- 可以顯示影像
- 可以控制大小
- 可以控制畫面長寬的比例

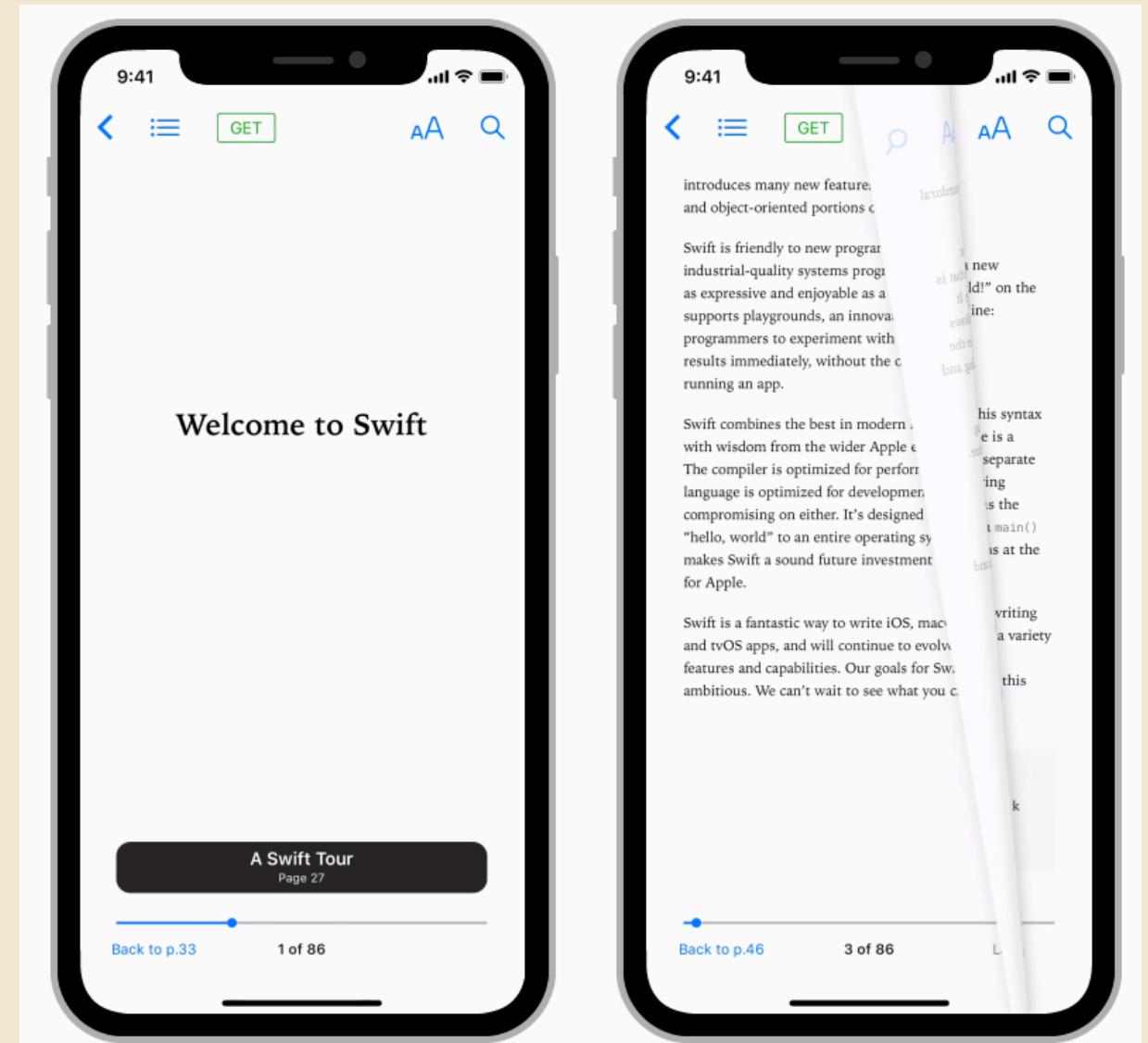


Map View

- 顯示地圖的畫面
 - 必須搭配MapKit
- 可以顯示不同方式的顯示
 - 一般地圖
 - 衛星地圖
 - 混合地圖
- 可以標記地點

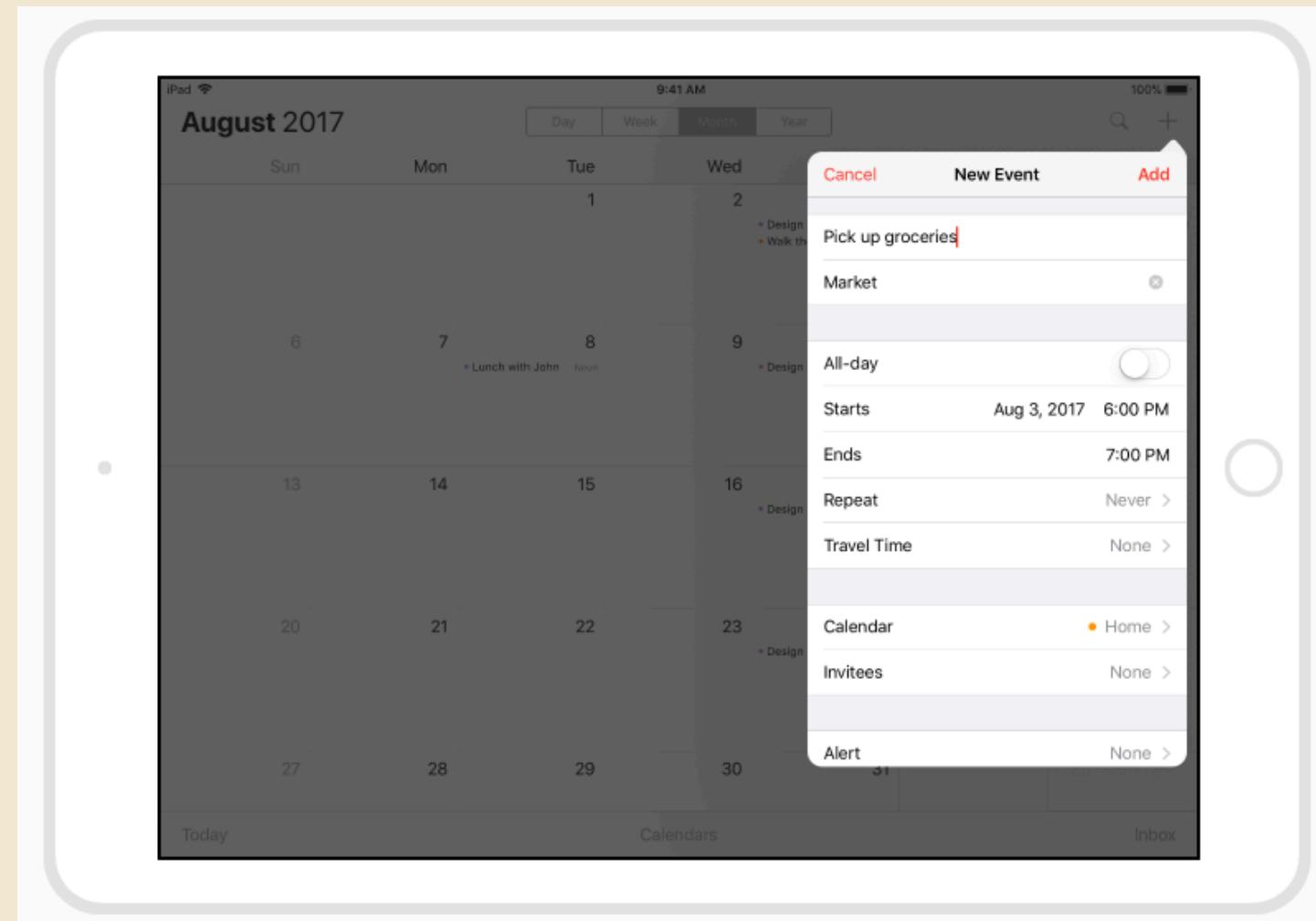
Page View Controller

- 提供線性的畫面顯示
- 不能跳到其他畫面



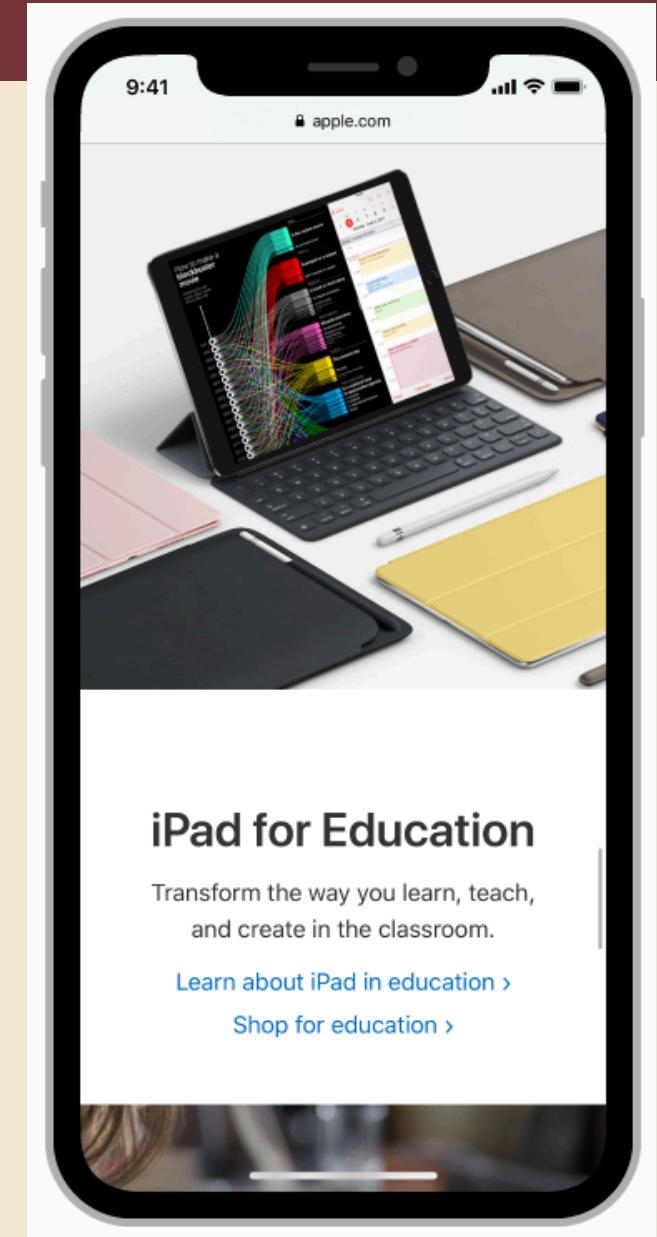
Popover

- 避免在iPhone中使用



ScrollView

- 當畫面比螢幕還要大時，會使用
- 可以讓他像翻頁一樣使用



Spilt View

- 可以同時顯示兩種不同畫面
 - 如果有足夠的空間 iPad
- 一個是主要另一個次要

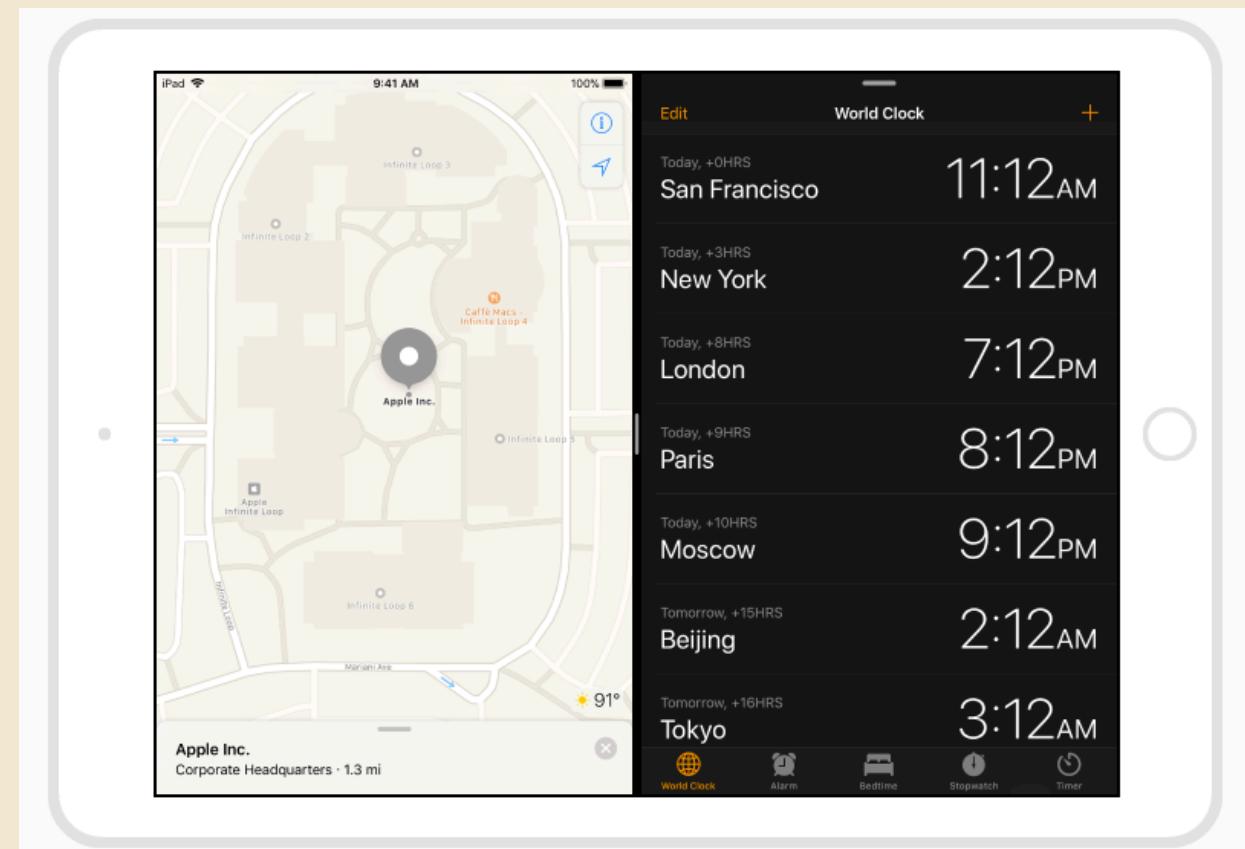
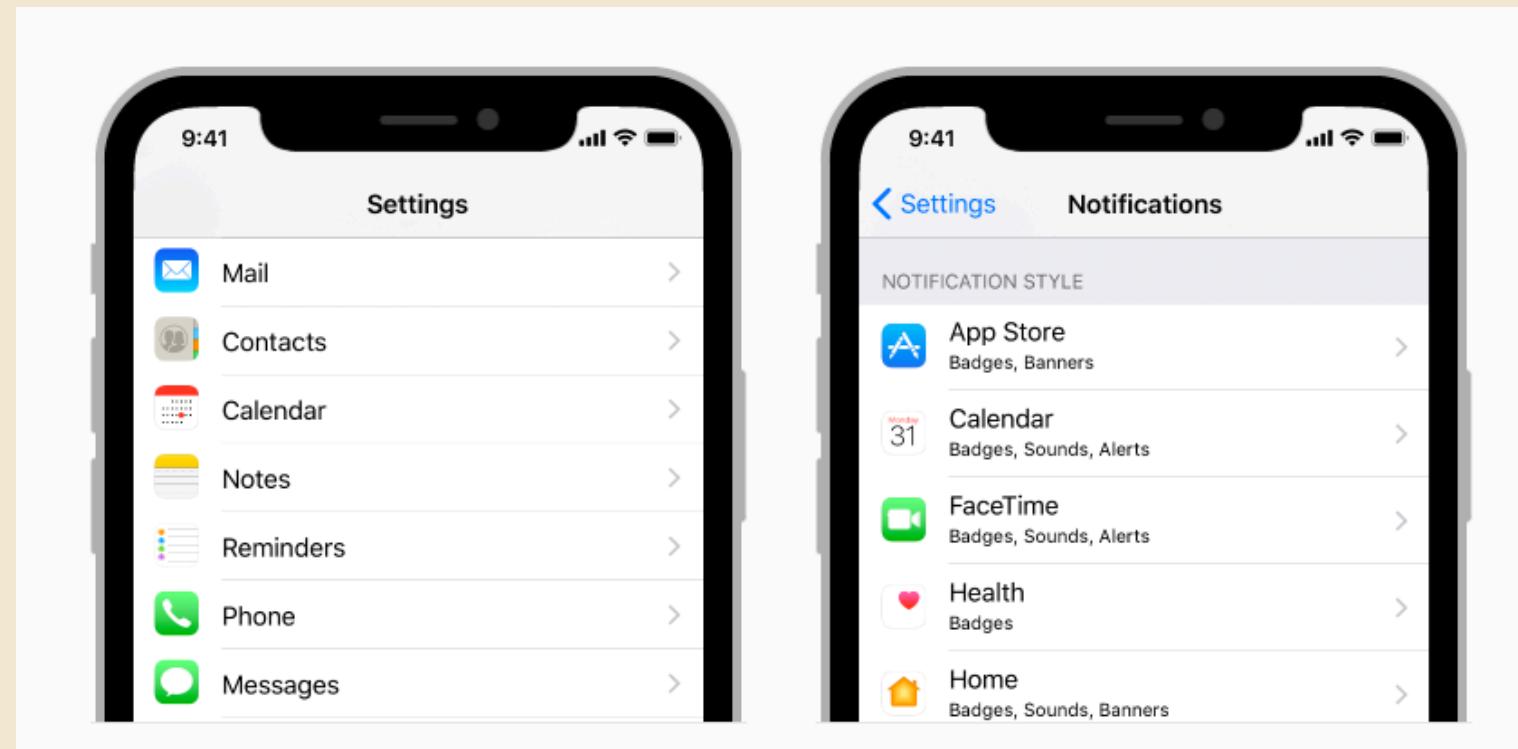


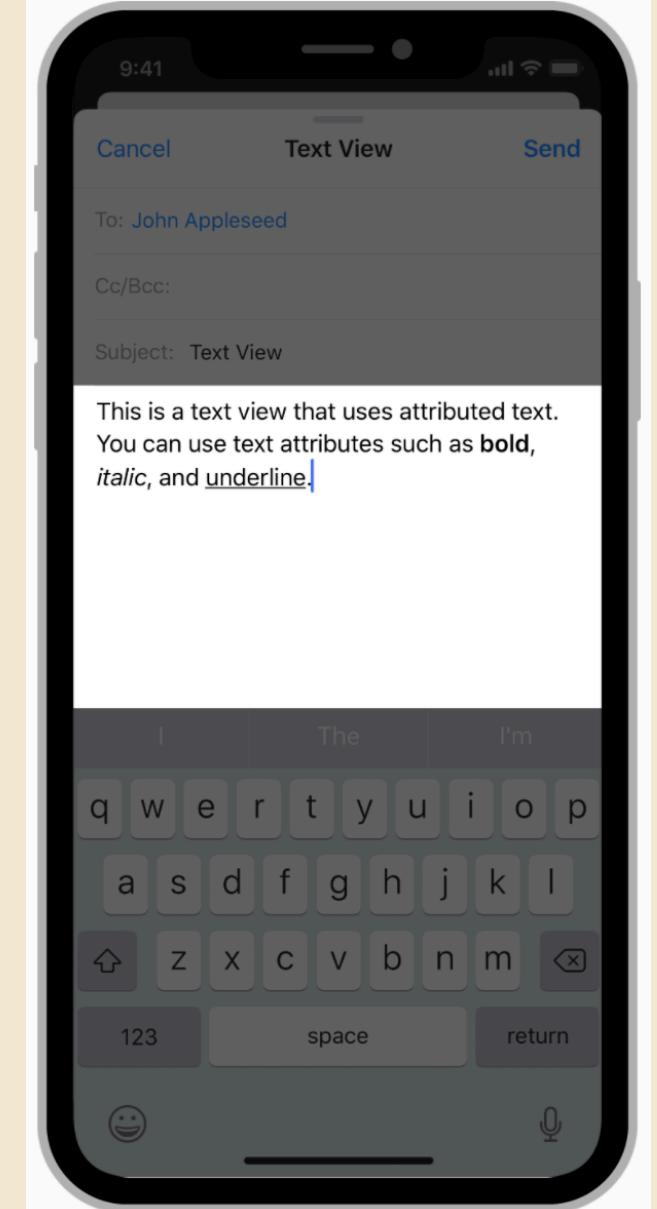
Table View

- 用橫列的方式顯示內容
- 可以用Section分類
- 提供內建的Controller
- 幾乎是使用最多的view



Text View

- UITextView
 - 可以有各種size
 - 可以scrolling
 - 可以改變字型、顏色、對齊...



Web View

- WKWebView
- 顯示網頁
- 可以提供客制化的行為

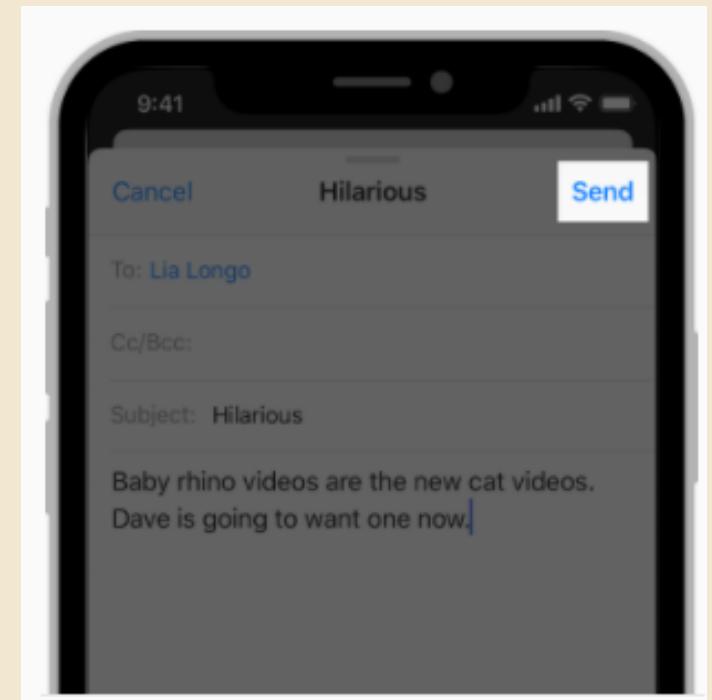


Controls

- Buttons
- Context menus
- Edit Menus
- Labels
- Page controls
- Pickers
- Progress indicators
- Refresh content controls
- Segmented controls
- Sliders
- Steppers
- Switches
- Text field

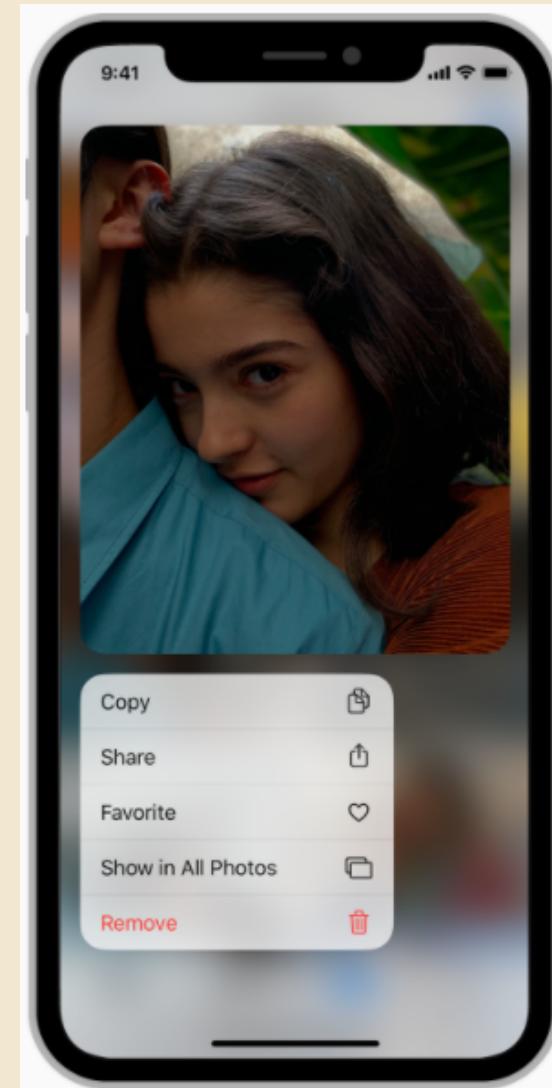
Buttons

- Use verbs in titles
- Use title-case for titles
- Keep titles short.
- Consider adding a border or a background only when necessary.



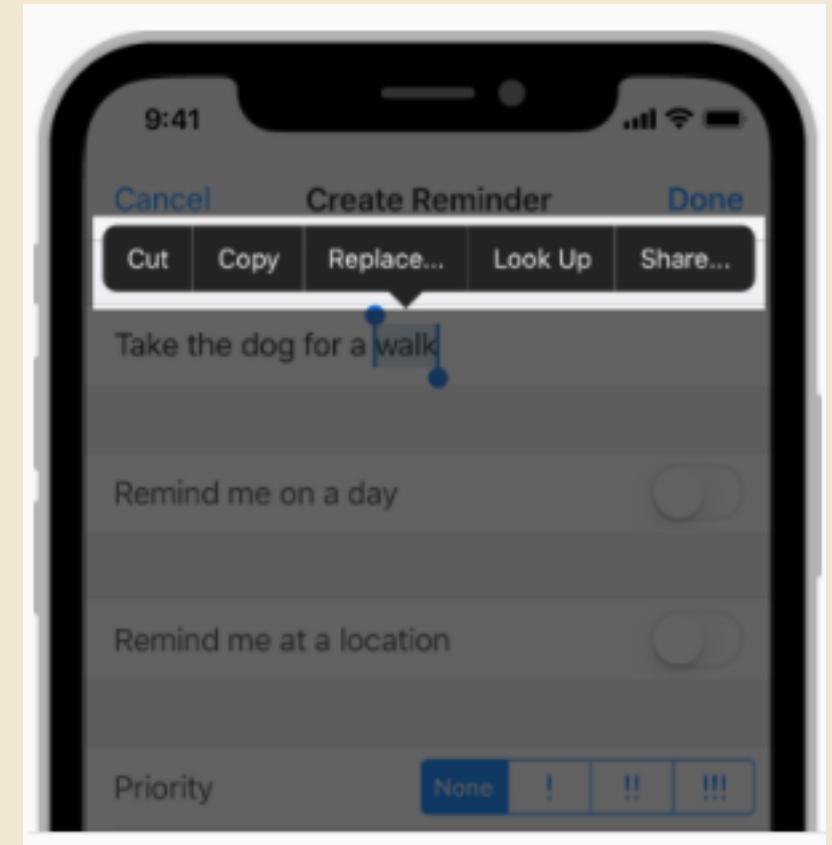
Context Menus

- In iOS 13 and later, you can use context menus to give people access to additional functionality related to onscreen items without cluttering the interface.



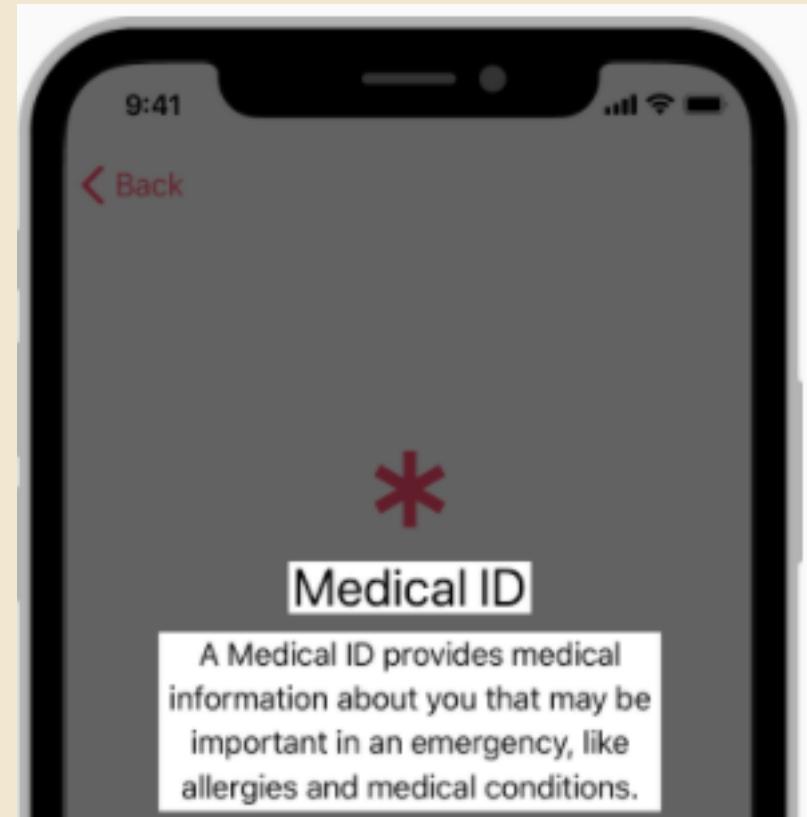
Edit Menus

- sdfsd



Labels

- A label describes an onscreen interface element or provides a short message
- Keep labels legible



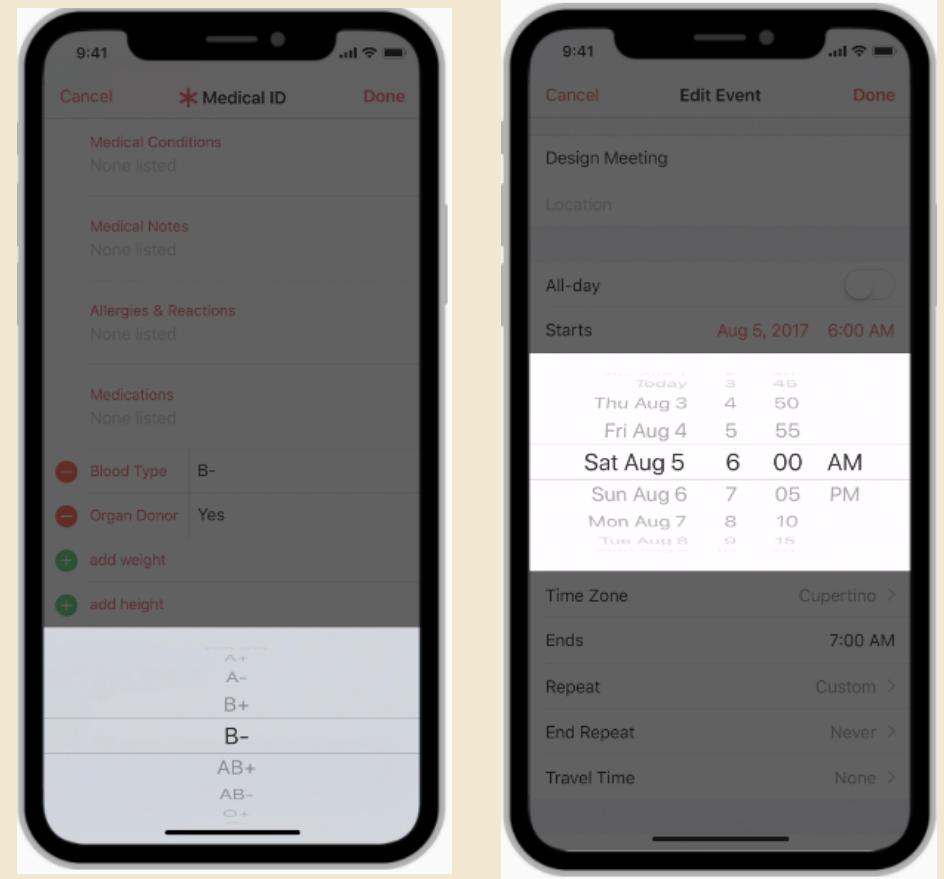
Page Controls

- A page control shows the position of the current page in a flat list of pages.
- It appears as a series of small indicator dots, representing the available pages in the order they were opened.
- Tips
 - Don't use a page control with hierarchical pages.
 - Don't display too many pages
 - 10 ~ 20 is too many to read
 - Center page controls at the bottom of the screen



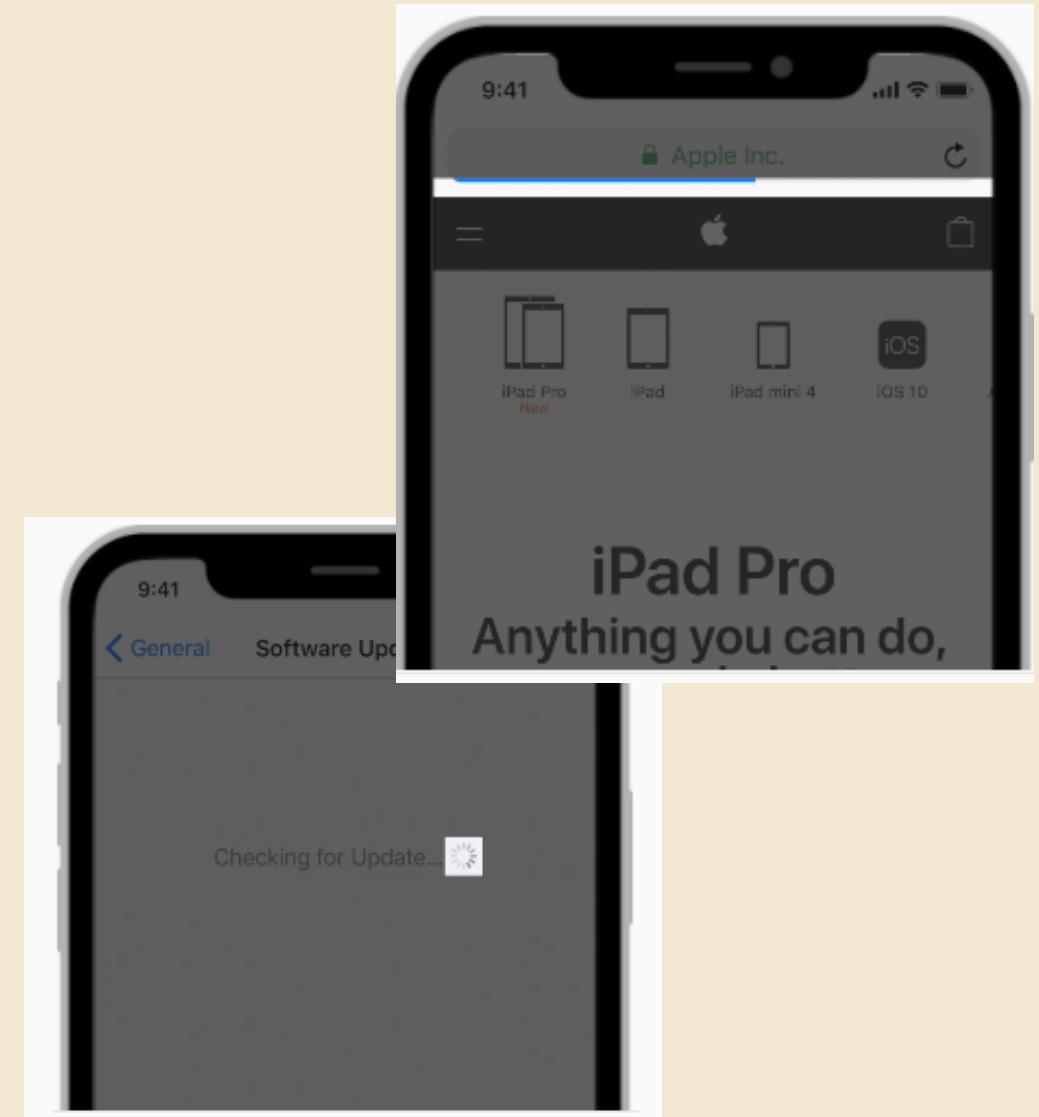
Pickers and Date Pickers

- Pickers tips
 - Use predictable and logically ordered values
 - Avoid switching screens to show a picker
 - Use a table instead of a picker for large value lists
- Date Pickers
 - Consider providing less granularity when specifying minutes



Progress Indicators

- Network Activity Indicators
 - Before iOS 12

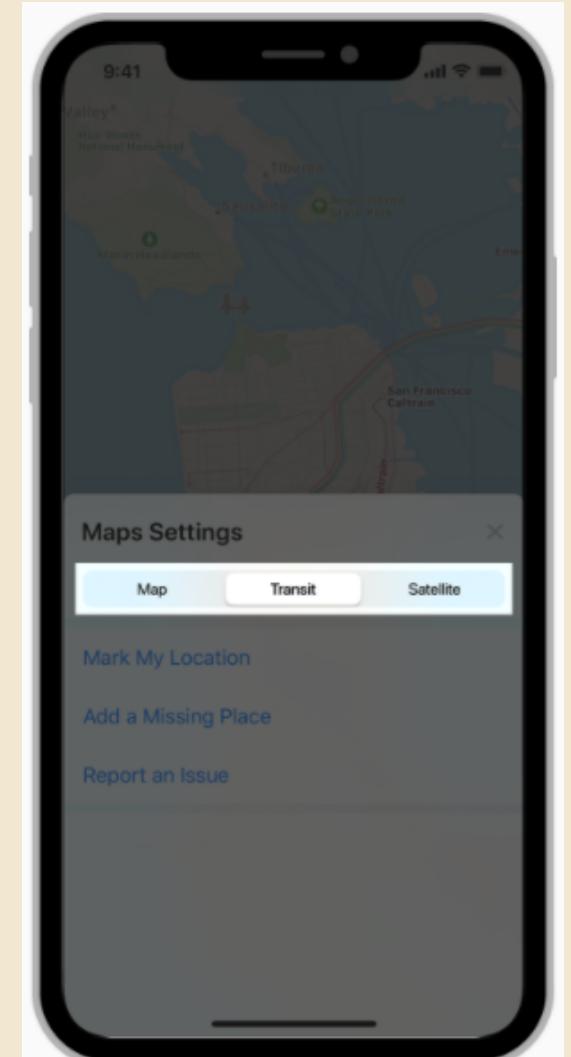


Refresh Content Controls

- A refresh control is manually initiated to immediately reload content, typically in a table view, without waiting for the next automatic content update to occur.

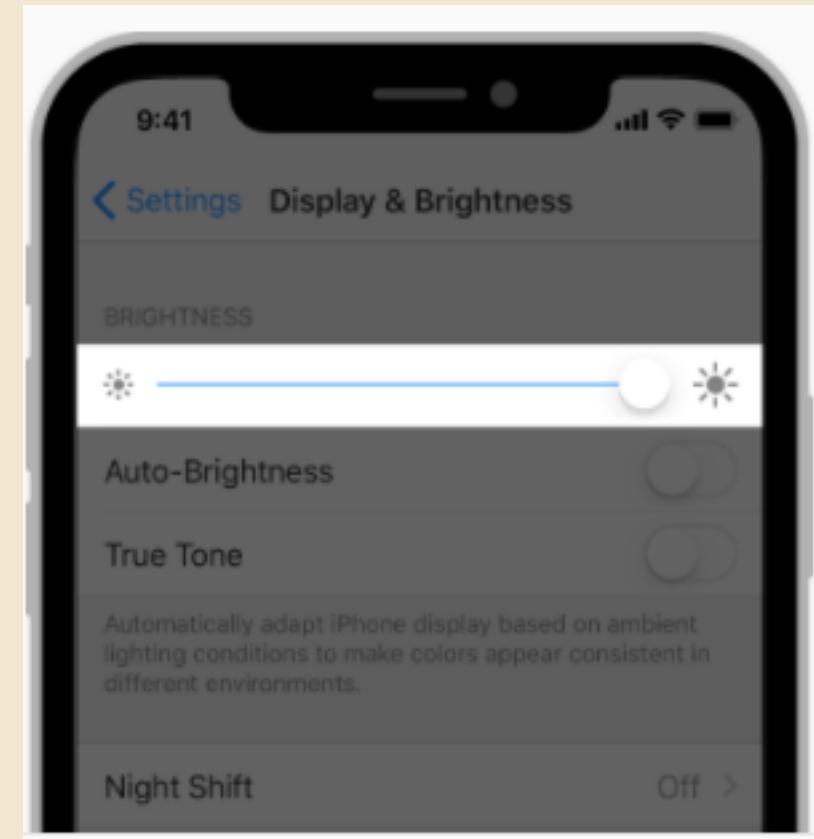
Segmented Controls

- A segmented control is a linear set of two or more segments, each of which functions as a mutually exclusive button.



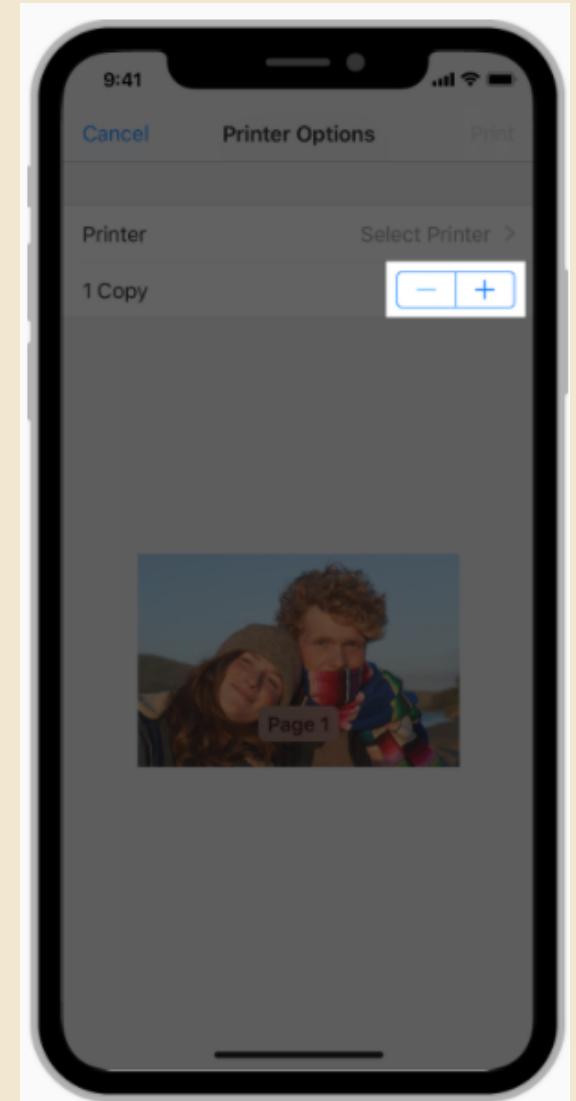
Sliders

- Customize a slider's appearance if it adds value
- Don't use a slider to adjust audio volume.



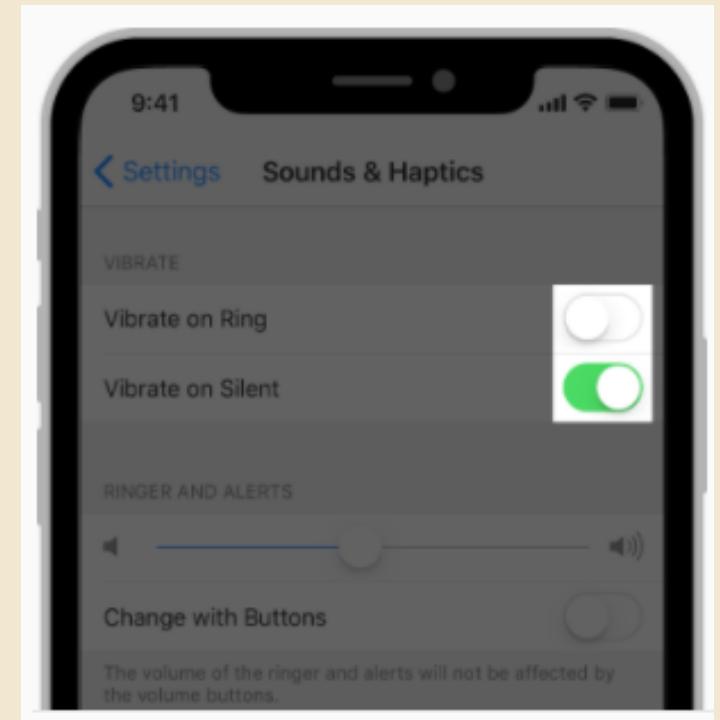
Steppers

- A stepper is a two-segment control used to increase or decrease an incremental value. By default, one segment of a stepper displays a plus symbol and the other displays a minus symbol.



Switches

- A switch is a visual toggle between two mutually exclusive states — on and off.

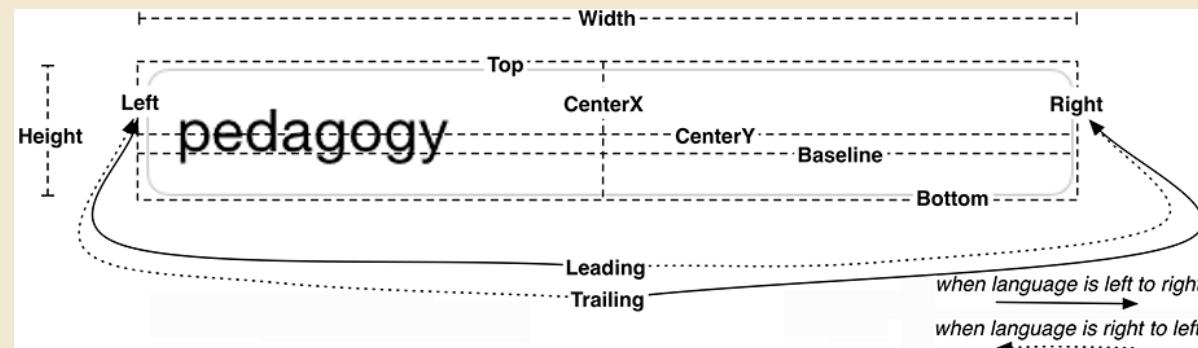


Text Fields

- A text field is a single-line, fixed-height field, often with rounded corners, that automatically brings up a keyboard when the user taps it.
- Tips
 - Show a hint in a text field to help communicate purpose.
 - Display a Clear button in the right end of a text field when appropriate.
 - Use secure text fields when appropriate
 - Use images and buttons to provide clarity and functionality in text fields
 - Show the appropriate keyboard type

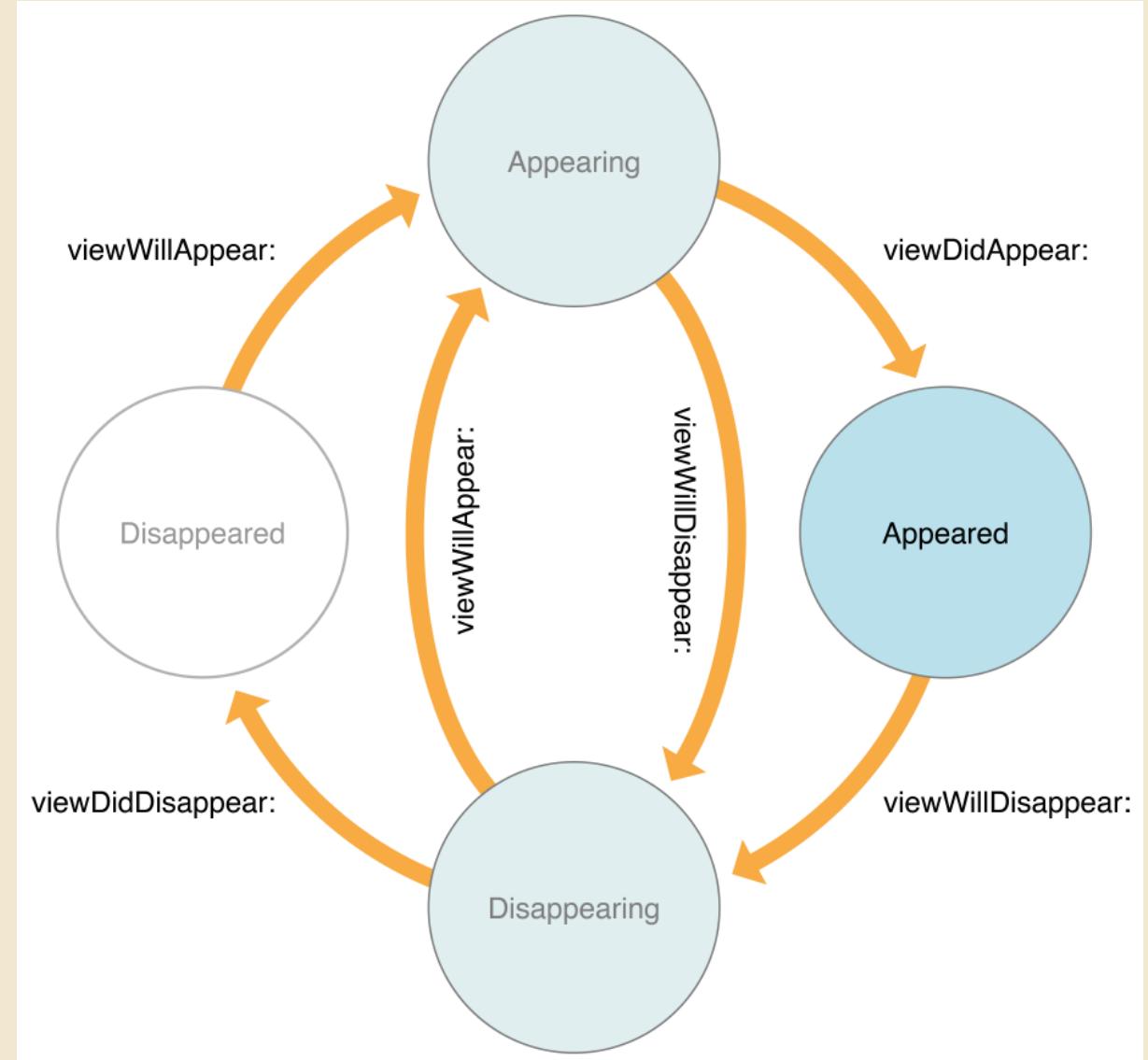
The Auto Layout System

- Width, height
 - Determine the alignment rectangle's size
- Top, bottom, left, right
 - Determine the spacing between the given edge of the alignment rectangle and the alignment rectangle of another view.
- CenterX, centerY
 - Determine the center point of the alignment rectangle
- FirstBaseline, lastBaseline
 - The same as the bottom attribute for most, but not all, views
- Leading, Trailing
 - language-specific attributes
- Apple recommends that you add constraints using Interface Builder.
- If your views are created and configured programmatically, then you can add constraints in code



View Controller Lifecycles

- viewDidLoad()
- viewWillAppears()
- viewDidAppear()
- viewWillDisappear()
- viewDidDisappear()



iOS User Interfaces: Storyboards vs. NIBs vs. Custom Code vs. SwiftUI

- Storyboards
 - A visual tool for laying out multiple application views and the transitions between them
 - Pros
 - Performance, prototypes
 - Cons
 - Reusability, data flow
- NIBs (or XIBs)
 - Each NIB file corresponds to a single view element and can be laid out in the Interface Builder, making it a visual tool as well
 - Pros
 - Reusability, Performance
 - Cons
 - Reusability, data flow

iOS User Interfaces: Storyboards vs. NIBs vs. Custom Code vs. SwiftUI (Cont.)

- Custom Code (Programmatically)
 - No GUI tools, but rather, handling all custom positioning, animation, etc. programmatically
 - Pros
 - Under the hood, merge conflicts, performance, reusability
 - Cons
 - Prototyping, refactoring
- SwiftUI
 - New and constructive method of creating interfaces across all Apple platforms and be able to create user interfaces with one set of tools and APIs
 - The new and improved Swift syntax allows you to write the code with greater ease, and it will work seamlessly with all of the Xcode design tools
 - Pros
 - Declarative syntax, declarative syntax, native on all Apple platforms
 - Cons
 - After iOS 13, no custom views extensions so far

Comparisons: SwiftUI vs. Interface Builder vs. Storyboards

- Editing
 - SwiftUI has the advantage here because Interface Builder has a lot of XML code, which is not easy to read or edit.
 - Storyboards do not well for editing because it has a tendency to get bigger and bigger, making it difficult to control any source changes.
 - SwiftUI editing is quite intuitive and straightforward.
- Creating functionality
 - IB and Storyboard is that they do not know a whole lot about Swift and vice-versa.
 - For example, with IB, it is possible to Ctrl-drag something right into the code and connects it to functionality. However, if you later decide to delete this code, IB will still compile it and call code that does not even exist. The Storyboard has similar issues when creating view controllers and other functions.
- Integrating with Swift
 - Before Swift, its product was centered around Objective-C.
 - However, Swift has a lot of advantages over Objective-C, such as value types, protocol extensions, and many other things.
 - The new SwiftUI design was created to enjoy all benefits Swift has to offer, whereas IB and Storyboard are all designed around Objective-C.

When to Use

- Before iOS 13
 - Use storyboards
 - Short lifespan
 - Small or medium app size
 - one developers
 - Tips
 - Multiple storyboards
 - Split storyboards according logical modules
 - Use nibs to design reusable views whenever possible
 - Avoid storyboards
 - Large app size
 - 2 or more developers
 - Tips
 - Use a hybrid approach
 - work on a single storyboard at one time
- After iOS 13
 - Can just use SwiftUI

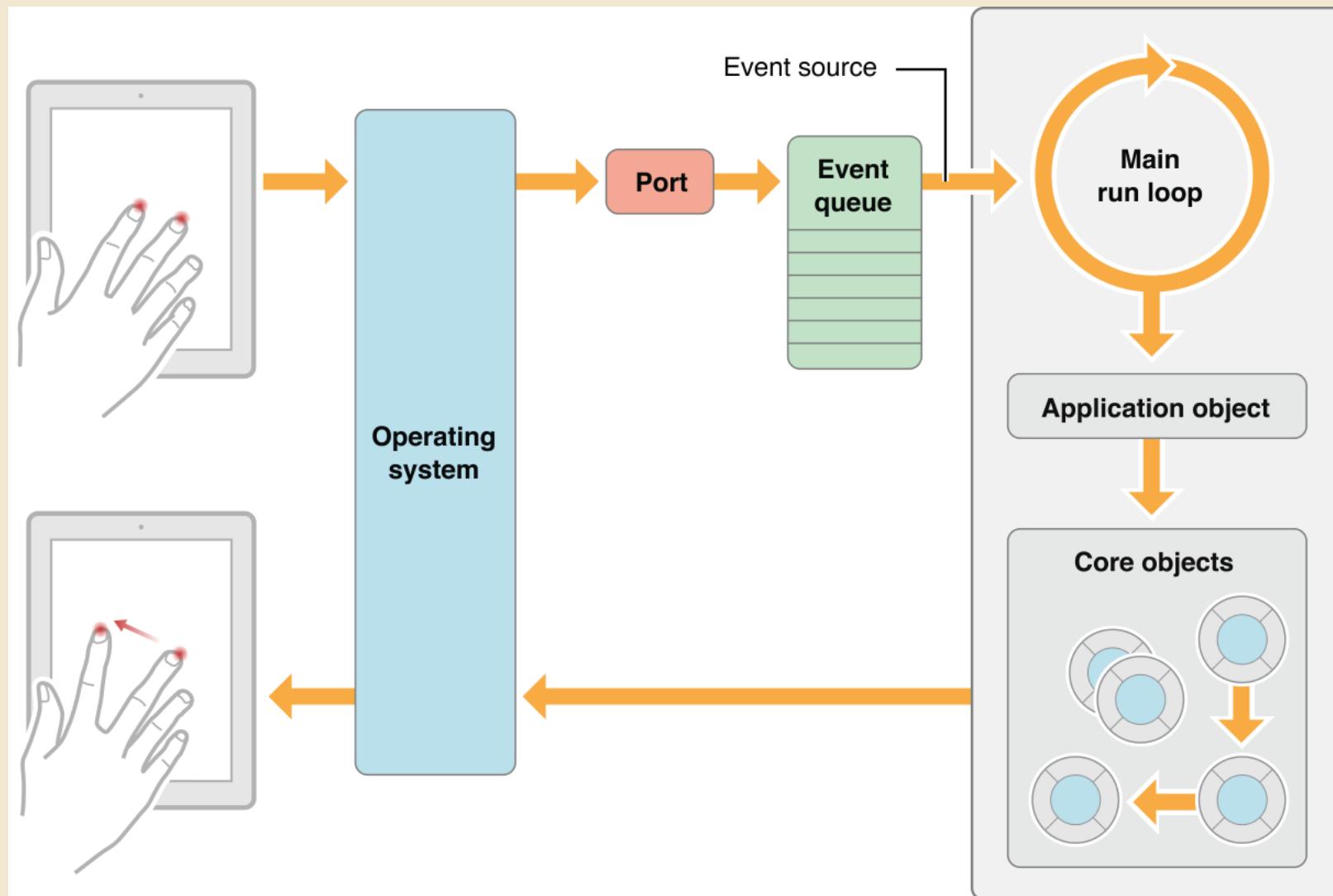
delegate

- delegate是一個包含某種動作的物件
- 當某個事件發生時會對應發生的，會接著執行delegate中的物件

main.m

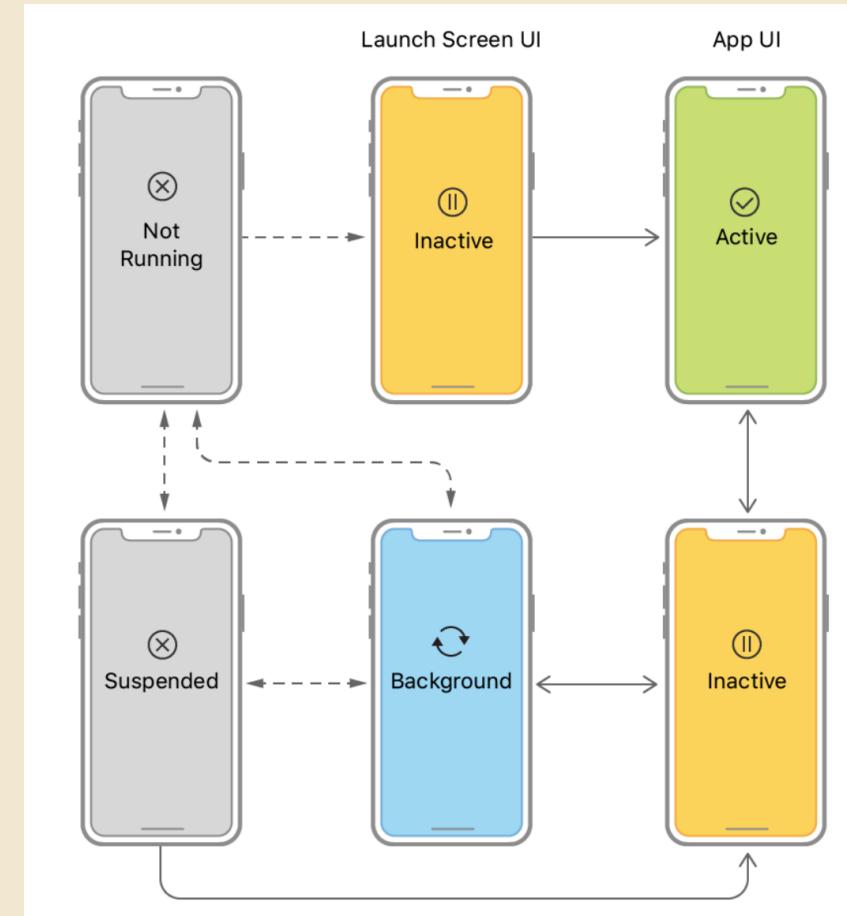
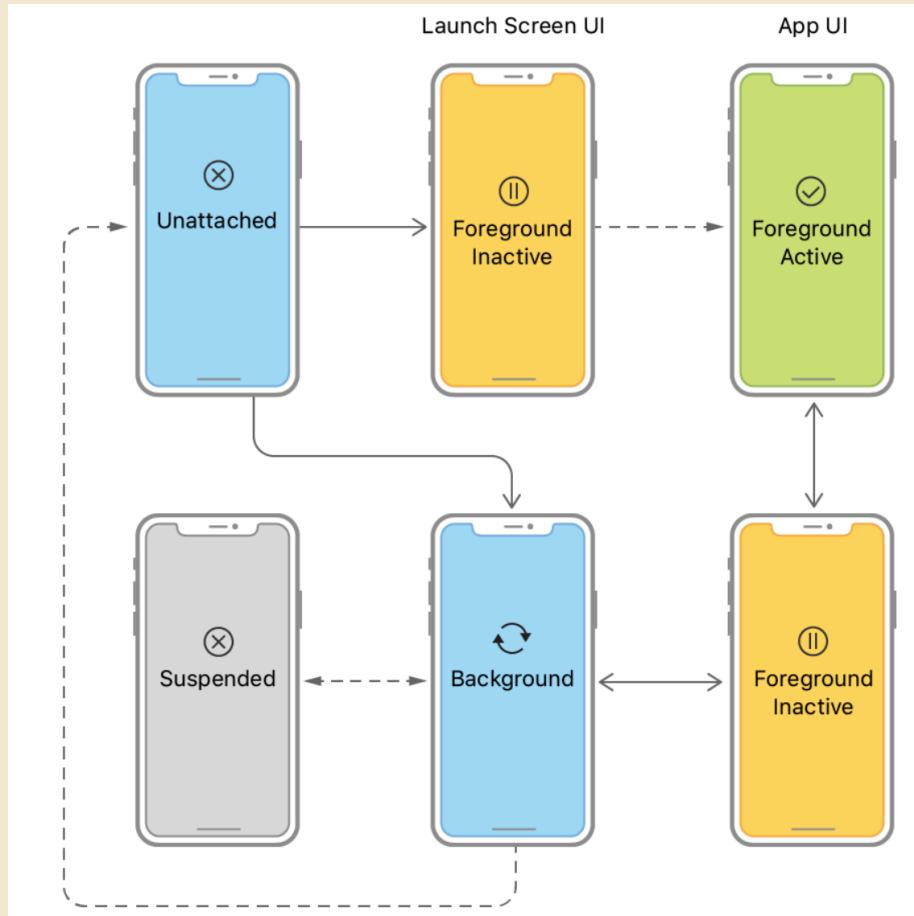
```
int main(int argc, char * argv[]) {
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass
([AppDelegate class]));
    }
}
```

- `@autoreleasepool`是指使用ARC的意思
- `main`會在`autoreleasepool`中呼叫`UIApplicationMain`
- `UIApplicationMain`會為app產生兩個用重要的元件
 - `UIApplication`的物件, 稱為*application*
 - 控制app event loop和app的行為
 - `AppDelegate`的物件, 稱為*app delegate*
 - 產生app window且提供狀態轉變時的回應(設計師可以自行控制的部分)



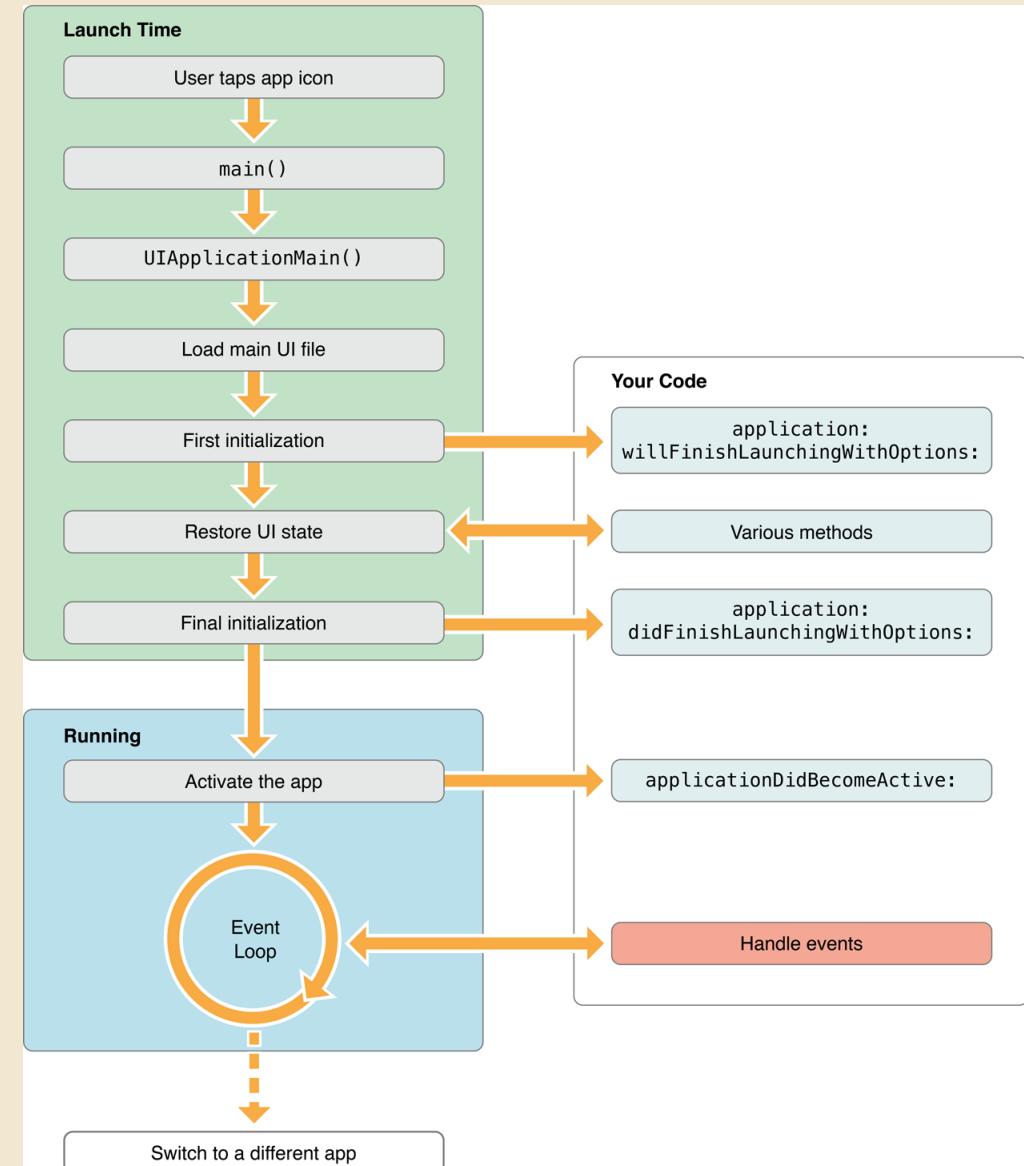
App Life Cycle

Respond to Scene-Based Life-Cycle Events (in iOS 13 and later)



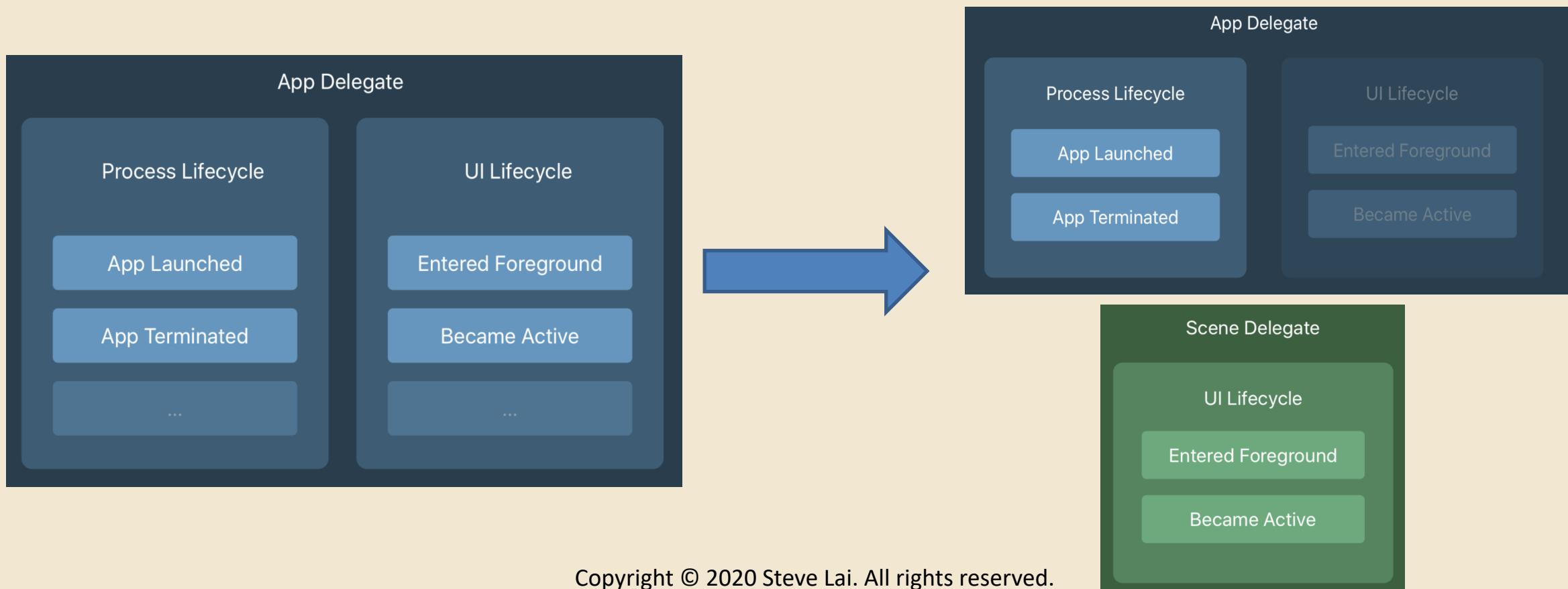
AppDelegate

- 在 AppDelegate 對應的 Method 中標明在狀態改變時要作的動作
- application:willFinishLaunchingWithOptions:—App 開始執行時可以做的動作
- application:didFinishLaunchingWithOptions:—在畫面呈現在使用者之前最後的動作
- applicationDidBecomeActive:—App 轉換到前端執行時
- applicationWillResignActive:—App 要從前端轉到後端執行時
- applicationDidEnterBackground:—App 在後端執行時
- applicationWillEnterForeground:—App 要從後端進入前端執行時
- applicationWillTerminate:—App 即將要結束時



SceneDelegate

- App could have multiple windows (scenes)
- In iOS 13 in Xcode 11, **SceneDelegate** class take over some UI life cycle functions which is implemented by **AppDelegate** class before

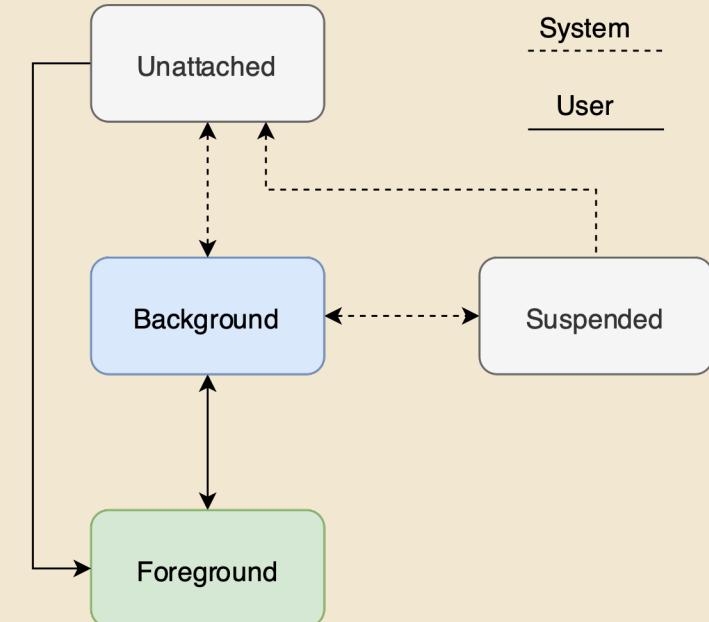


ScenDelegate



SceneDelegate

- Unattached: Being attached and detached
 - `scene(_:willConnectTo:options:)`
 - `sceneDidDisconnect(_:)`
- Background
 - Move to foreground-inactive: `sceneWillResignActive(_:)`
 - Move to background: `sceneDidEnterBackground(_:)`
 - If not processing any tasks: `sceneDidDisconnect(_:)`
- Foreground
 - to the *foreground*
 - The scene is about to become visible onscreen: `sceneWillEnterForeground(_:)`
 - The scene is ready to respond to user events: `sceneDidBecomeActive(_:)`
 - to the *background*
 - The scene is about to stop responding to user events: `sceneWillResignActive(_:)`
 - The scene is no longer visible onscreen: `sceneDidEnterBackground(_:)`



ARC

- Automatic Reference Counting (自動指向計算)
- iOS的專案中預設將ARC開啟

原始的作法

- 記憶體遺漏 (leak memory)
- 問題
 - 如果收到一個function回傳的指標，誰要負責控制此變數
 - function負責？因為他產生的物件
 - 也許還有其他function產生物件指向他
 - ...
- 要避免記憶體遺漏 (leak memory)是一個浩大的工程

Reference Counting

- 需要有一個計算的變數
 - 每當物件產生時產生對應變數
 - 如有初始有指標指向該物件，初始值為1
 - 每當有指標指向該物件，該變數加1
 - 每當有指標不再指向該物件，該變數減1
 - 當該變數為0時，回收該物件記憶體空間

Reference Counting

- 優點
 - 不用煩惱誰該負責
 - 當該變數為0時再處理
- 缺點
 - 程式設計師依然要記得處理該變數
 - 如果忘記依然會出現記憶體遺漏

- ARC為了解決此問題
- 每當產生物件並有指標指向或變動不指向該物件
 - ARC自動產生對應的程式碼處理計算現在指向的指標個數
- 當該變數為0
 - ARC自動回收該物件

- 將指標設定指向空 (nil, null)，代表不指向該物件
- 雖然有ARC，但記憶體管理依然重要
 - 程式設計師還是必須注意可能產生的問題

Function Implementation

- Cocopods and Firebase

Python Financial Data Analysis

Steve Lai

2020/06/04

Outline

- Python Requirements
- Data Analysis
- Data Visualization
 - Financial Graphics
 - Financial Time Series
- Financial Models
- Machine Learning
- Case Studies
 - House Prices: Advanced Regression Techniques
 - Home Credit Default Risk

Python Requirements

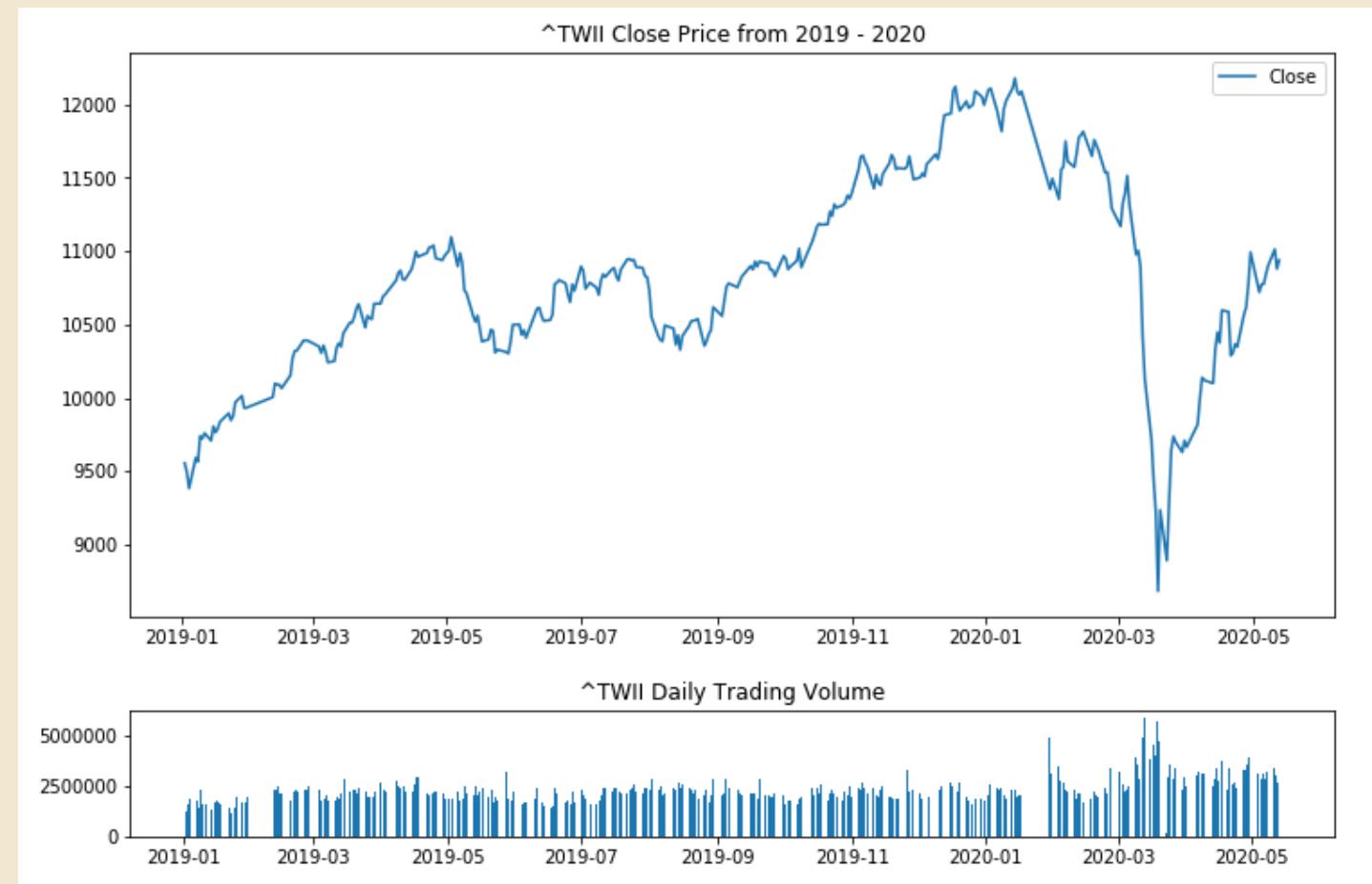
- Environment
 - Jupyter
 - Pycharm
- Python 3.X
 - PEP (Python Enhancement Proposal) 8 style guidelines
- Numpy
- Pandas
- Matplotlib
- Others
 - Scipy
 - Science kit learn
 - Deep learning framework

Data Analysis

- Also be known as knowledge mining from data, knowledge extraction or **knowledge discovery from data (KDD)**
 - identifying the goal
 - creating a target data set
 - data cleaning and preprocessing
 - data transformation
 - choosing the data mining method
 - choosing the data mining algorithms
 - data mining
 - interpreting mined patterns
 - acting on the discovered knowledge
- Step:
 1. Data cleaning
 2. Data integration
 3. Data selection
 4. Data transformation
 5. Data mining
 6. Pattern evaluation

Sample Data Source

- TWSE
- TAIFEX
- Quandl
 - Need sign up an account
- pandas_datareader



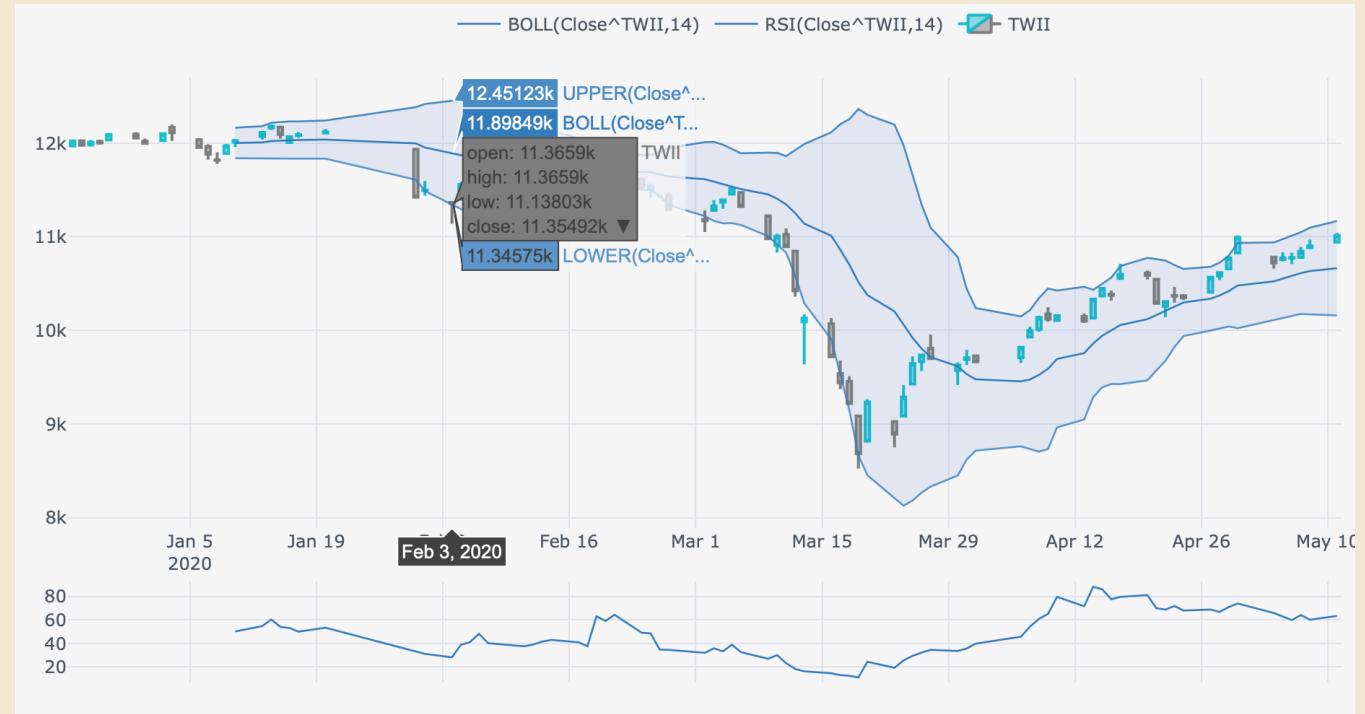
Data Visualization

- Requirement packages
 - Matplotlib
 - Seaborn
 - Plotly
- Graphic Types
 - plot
 - bar
 - histogram
 - scatter
 - pie
 - box
 - polygon



Financial Graphics

- Requirement packages
 - Cufflinks
 - chart_studio
 - mplfinance
- Graphics Types
 - Candlestick chart
 - Bollinger bands
 - Relative strength indicator
 - Moving average convergence divergence



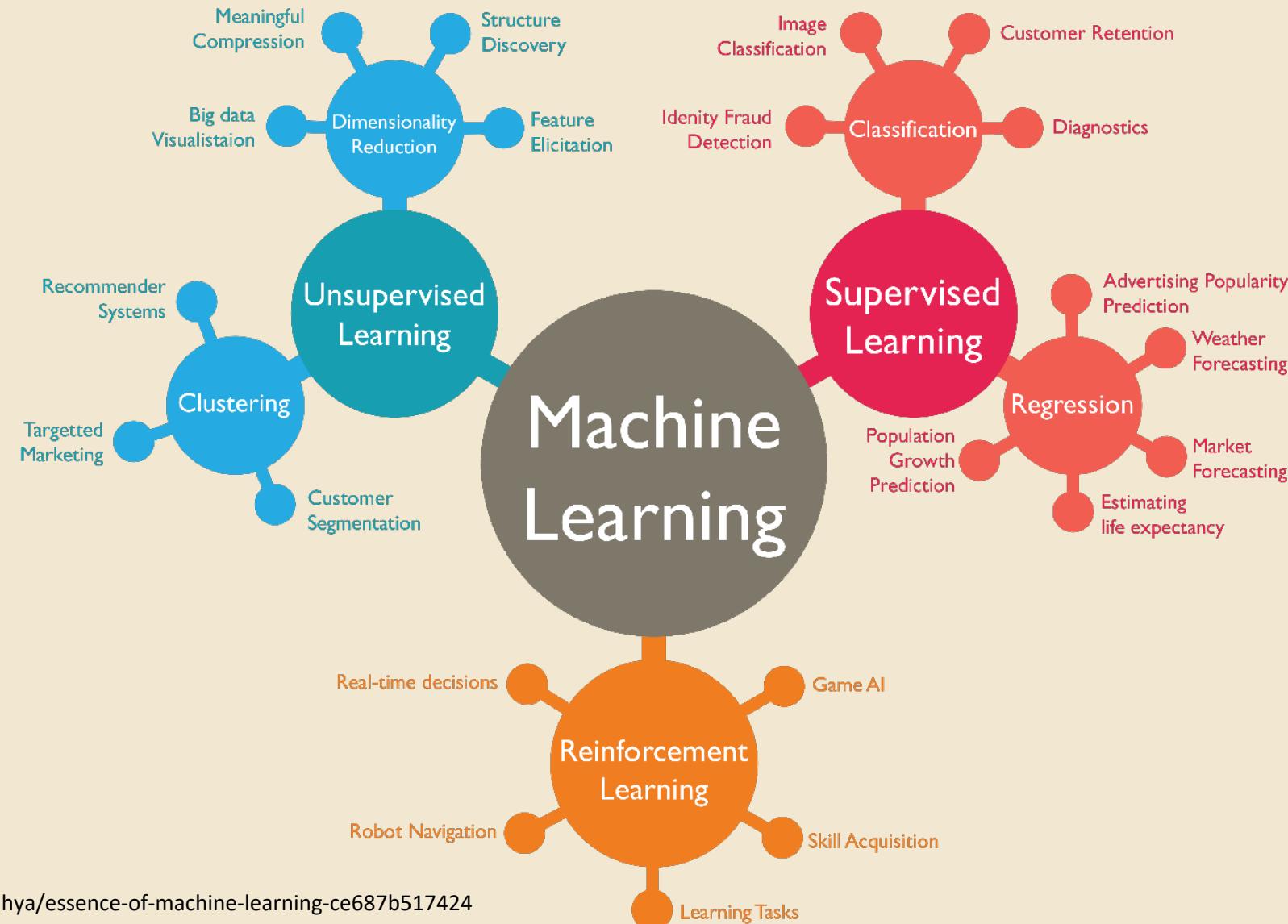
Financial Time Series

- Summary Statistics
 - `df.describe()`
 - mean, min, max, std, quartile
- Time difference
 - `df.diff()`
 - `df.pct_change()`
- Time resample
- Long/Short simple moving averages
 - `df.rolling(window=5, min_periods=5).mean()`
 - `df.rolling(window=30, min_periods=30).mean()`
- Exponential moving averages
 - `df.ewm(span=5).mean()`
 - `df.ewm(span=30).mean()`
- Correlation Analysis
 - `df.pct_change().corr()`

Financial Models

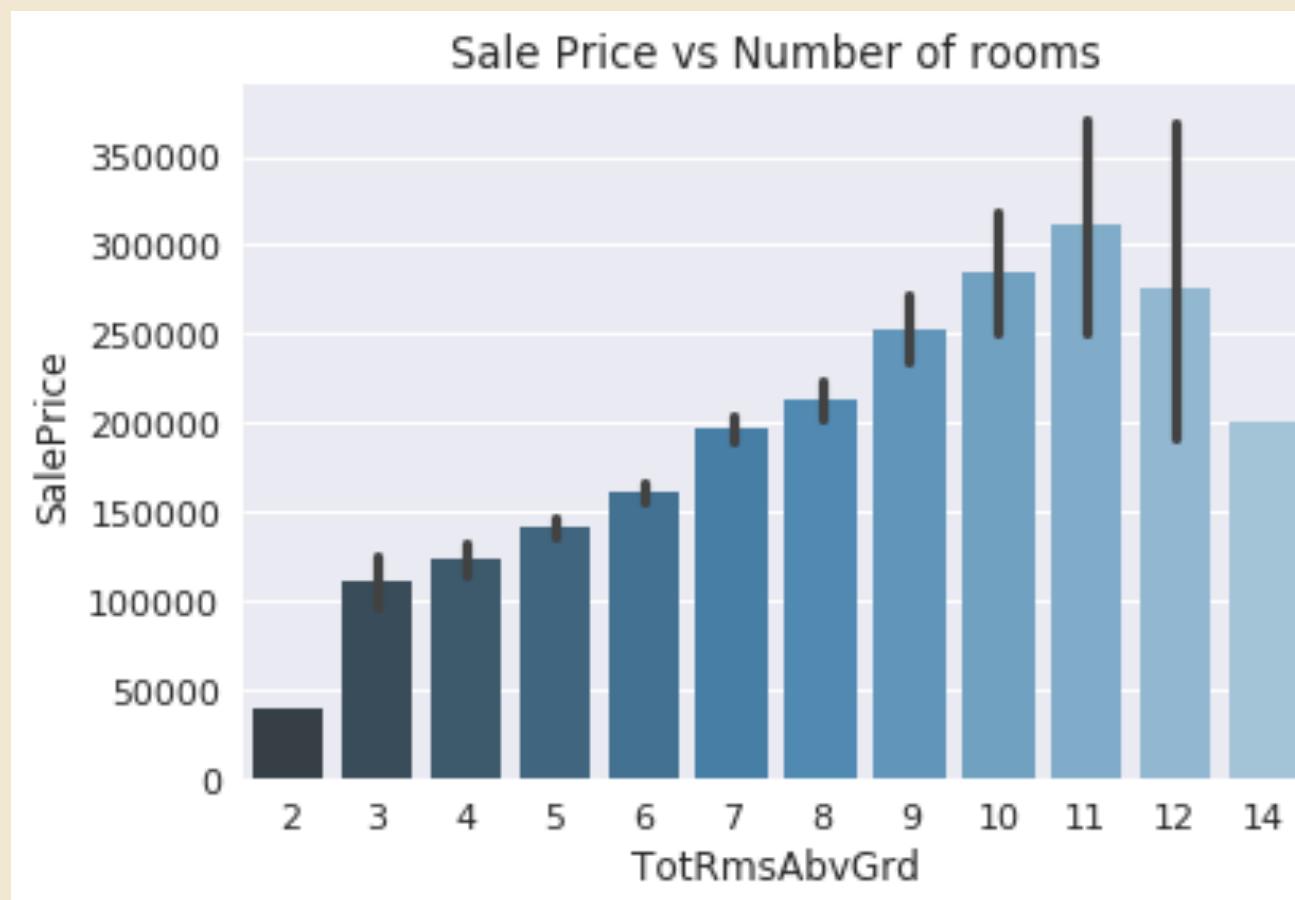
- Linear models
- Nonlinear models
 - Implied volatility model
 - Markov switching model
 - Threshold model
 - Smooth transition model

Machine Learning



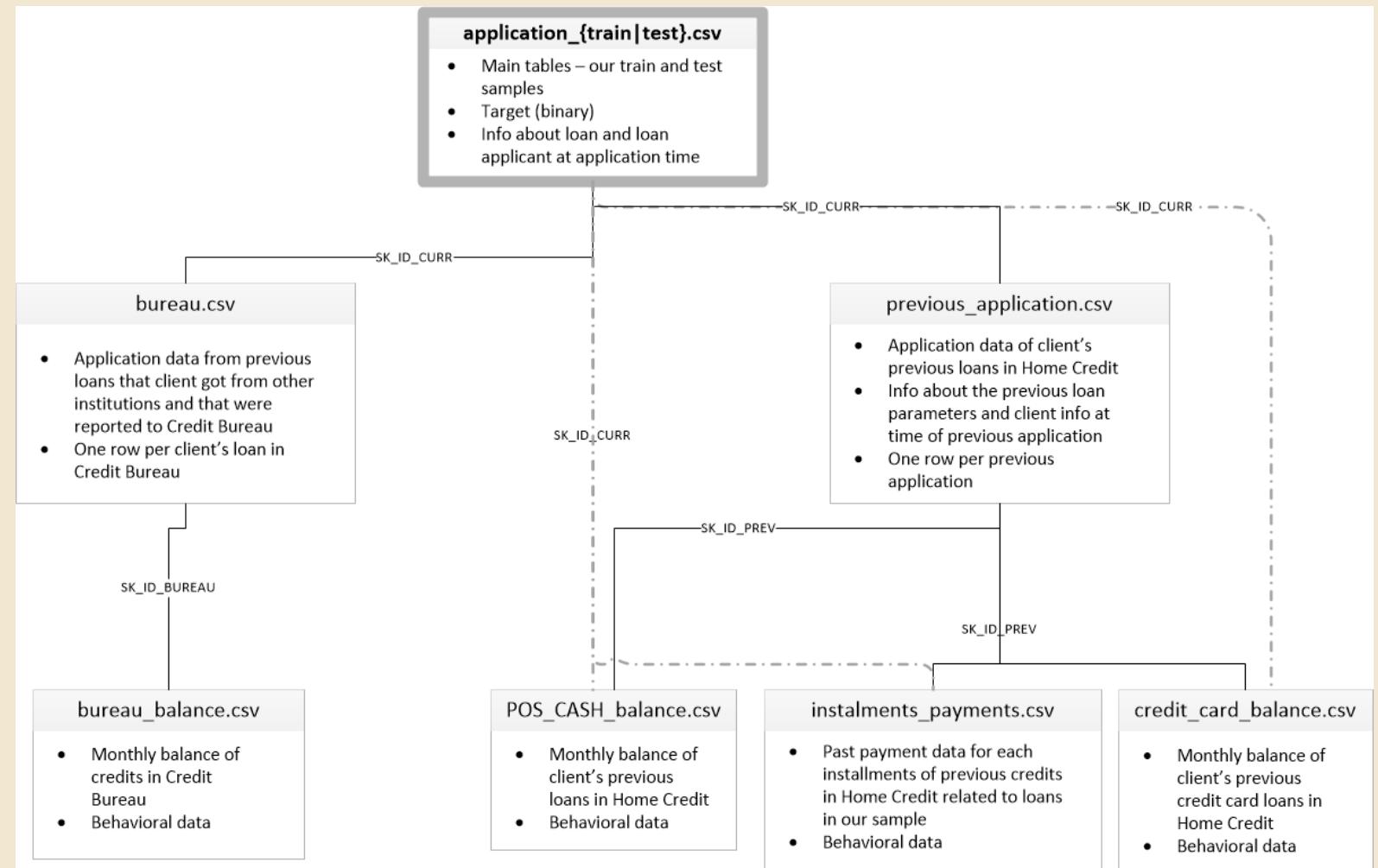
Case Study: House Prices

- Kaggle
 - House Prices: Advanced Regression Techniques



Case Study: Home Credit Default Risk

- Kaggle
 - Home Credit Default Risk



Code Reference

- https://github.com/lzrong0203/fin_ios_python