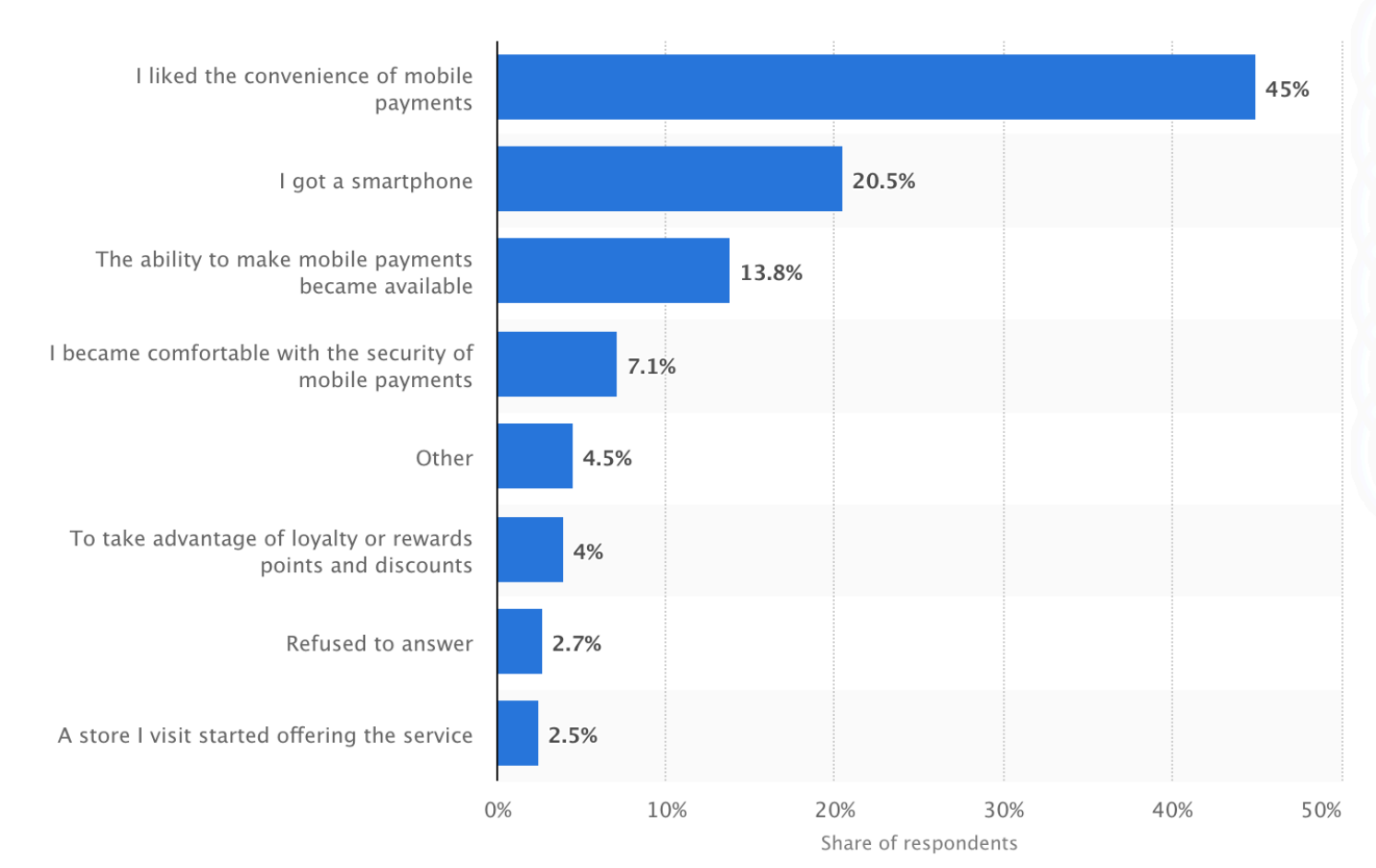
App User Experience Design

Dr. Steve Lai

2020/10/16

Mobile Banking

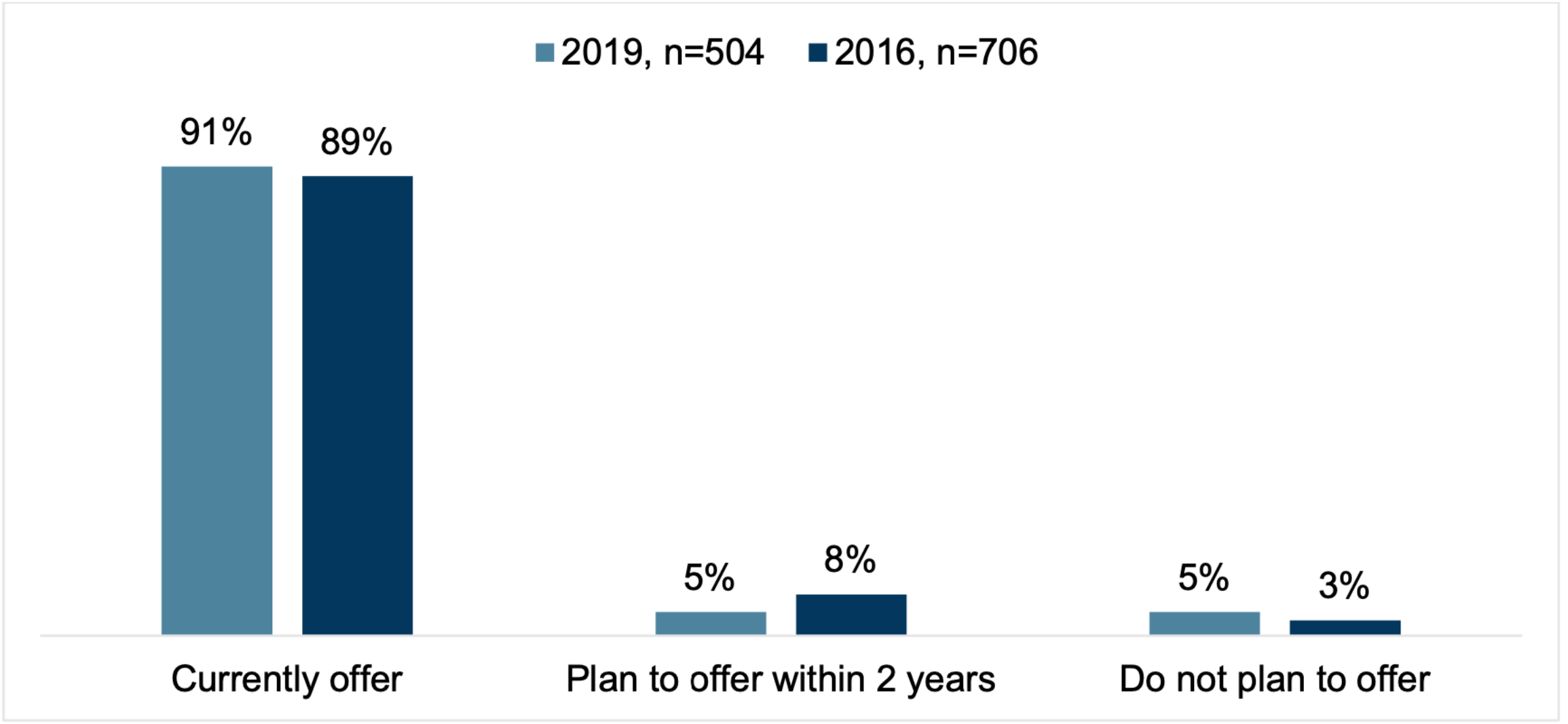
- 根據 Juniper research 調查，在2019年有17.5億人使用手機操作銀行服務
- 有45%人覺得方便是主要的原因



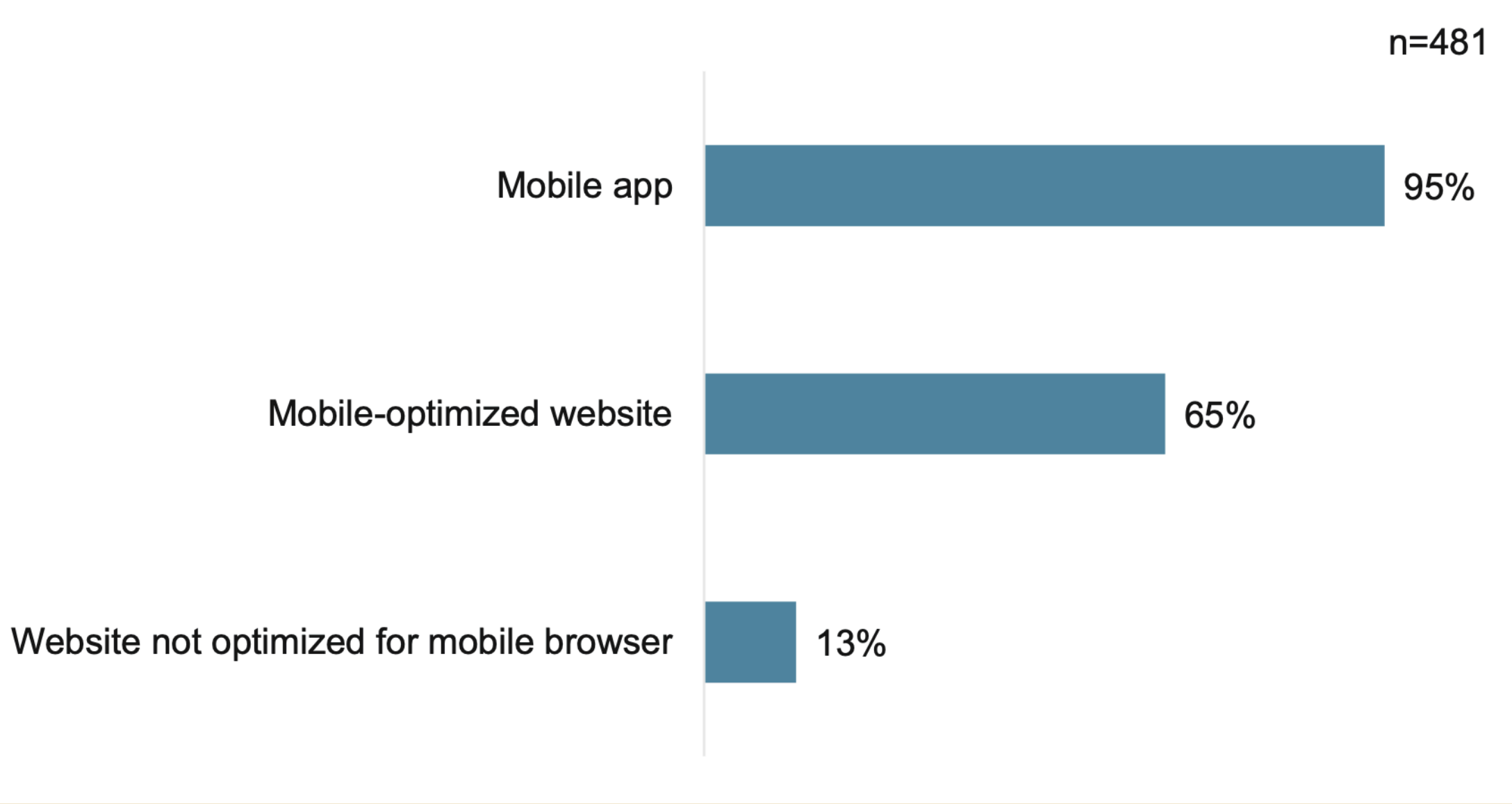
Financial Institutions across the U.S. Participate in the Mobile Landscape Transformation

- Results from the *2019 Federal Reserve Mobile Financial Services Survey of Financial Institutions*
- There were 5,303 banks and 5,308 credit unions in the U.S. as of June 2019.
- The 2019 MFS Survey collected data from 504 FIs – 337 banks and 167 credit unions – representing 6 percent of all banks and 3 percent of all credit unions nationally.

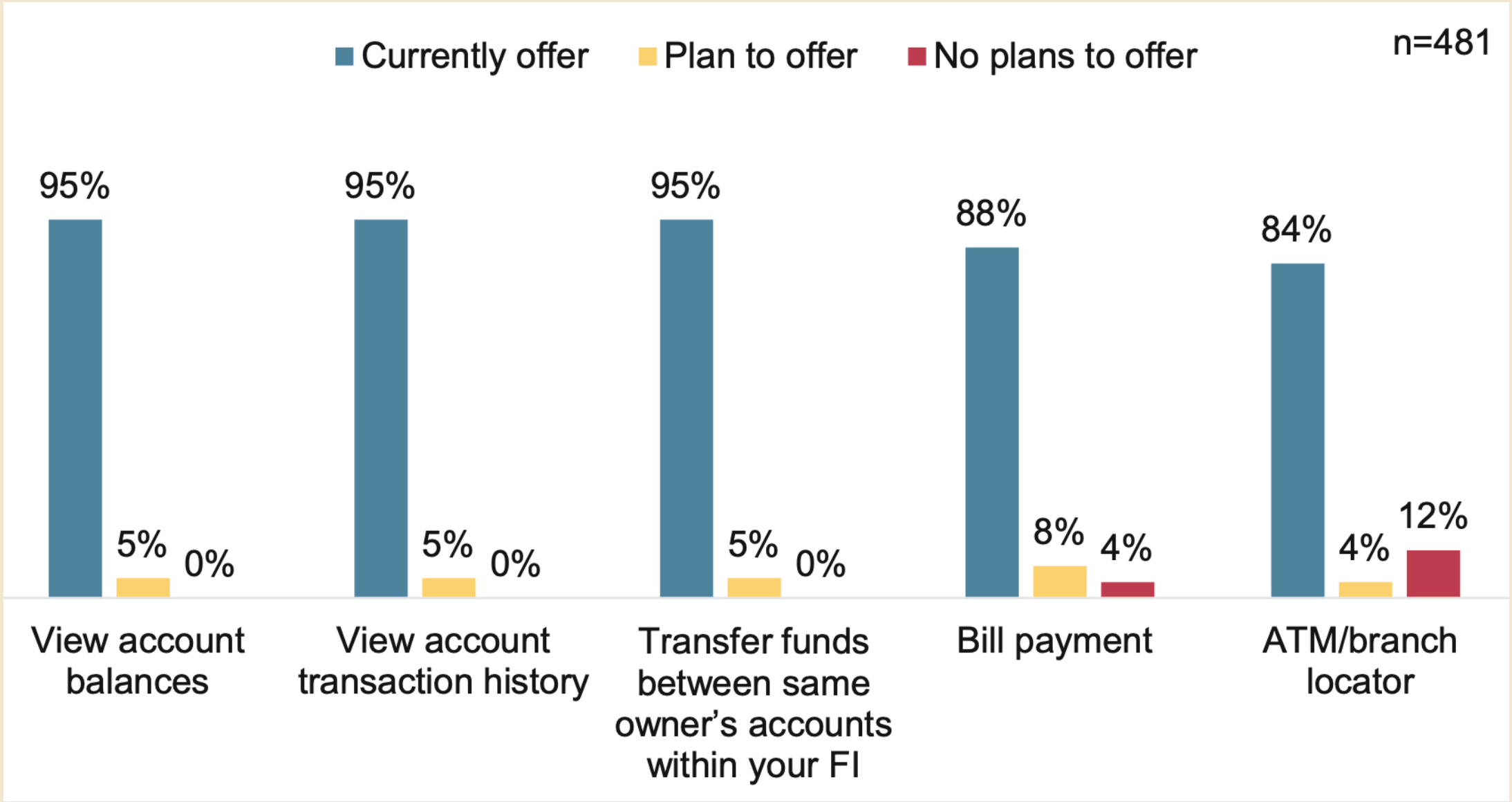
Do You Currently Offer or Plan to Offer Mobile Banking to Retail Customers?



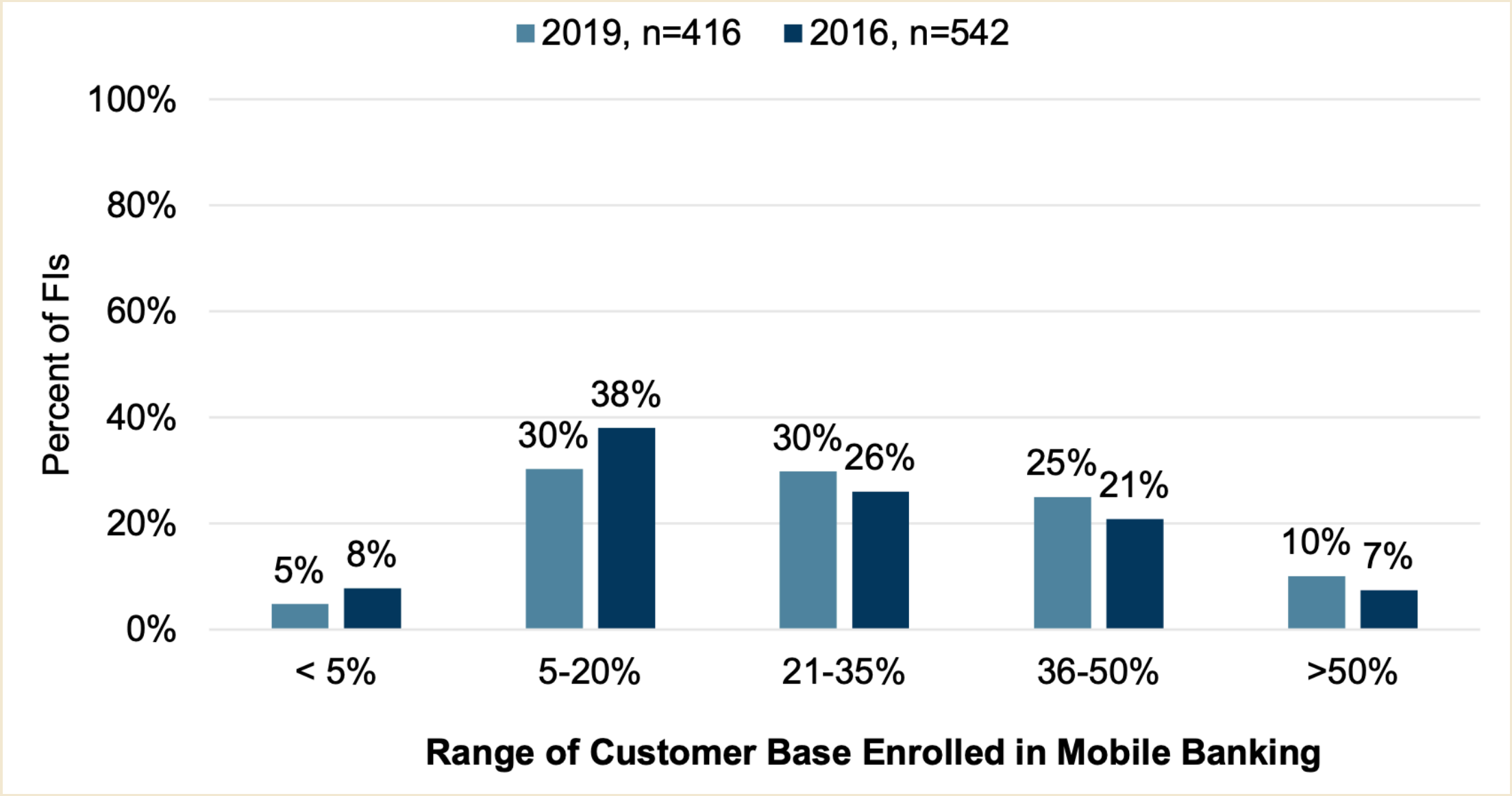
How Do You Offer Retail Customers Access to Your Mobile Banking Services?



Which Mobile Banking Features Do You Currently Offer or Plan to Offer?

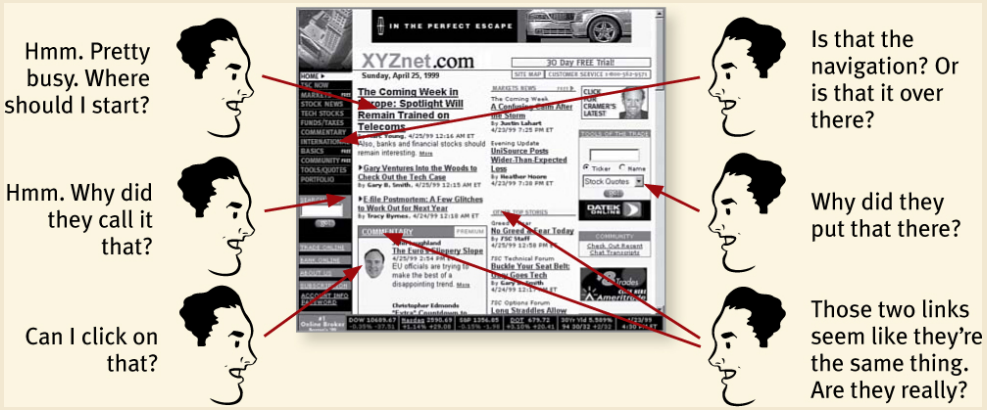


What Percentage Of Your Retail Customers Are ENROLLED In Your Mobile Banking Services?



"DON'T MAKE ME THINK!"

- I should be able to "get it"—what it is and how to use it—without expending any effort thinking about it.
- Well, self-evident enough, for instance, that your next door neighbor, who has no interest in the subject of your site and who barely knows how to use the Back button, could look at your Home page and say, "Oh, it's a _____." (With any luck, she'll say, "Oh, it's a _____. Great!" But that's another subject.)
- When I'm looking at a page that doesn't make me think, all the thought balloons over my head say things like "OK, there's the_____. And that's a _____. And there's the thing that I want."



Users Shouldn't Spend Their Time Thinking About

- I could list dozens of things that users shouldn't spend their time thinking about, like
  - Where am I?
  - Where should I begin?
  - Where did they put _____?
  - What are the most important things on this page?
  - Why did they call it that?
  - Is that an ad or part of the site?
- Sometimes, though, particularly if you're doing something original or groundbreaking or something that's inherently complicated, you have to settle for *self-explanatory*.
  - On a self-explanatory page, it takes a *little* thought to "get it"—but only a little.
  - The appearance of things (like size, color, and layout), their well-chosen names, and the *small* amounts of carefully crafted text should all work together to create a sense of nearly effortless understanding.
- Here's the rule: If you can't make something self-evident, you at least need to make it self-explanatory.

How We *Really* Use The Web(or APP)

- What they actually do most of the time (if we're lucky) is glance at each new page, scan some of the text, and click on the first link that catches their interest or vaguely resembles the thing they're looking for. There are almost always large parts of the page that they don't even look at.
- FACT OF LIFE #1: WE DON'T READ PAGES. WE SCAN THEM.
  - We're usually on a mission.
  - We know we don't need to read everything.
  - We're good at it.

Fact Of Life #2

- FACT OF LIFE #2: WE DON'T MAKE OPTIMAL CHOICES. WE SATISFICE.
- In reality, though, most of the time we *don't* choose the best option—we choose the *first reasonable option*, a strategy known as satisficing. As soon as we find a link that seems like it might lead to what we're looking for, there's a very good chance that we'll click it.
  - We're usually in a hurry.
    - Optimizing is hard, and it takes a long time. Satisficing is more efficient.
  - There's not much of a penalty for guessing wrong.
    - Unlike firefighting, the penalty for guessing wrong on a Web site is usually only a click or two of the Back button, making satisficing an effective strategy.
  - Weighing options may not improve our chances.
  - Guessing is more fun.
    - It's less work than weighing options, and if you guess right, it's faster. And it introduces an element of chance—the pleasant possibility of running into something surprising and good.

Fact Of Life #2

- FACT OF LIFE #3: WE DON'T FIGURE OUT HOW THINGS WORK. WE MUDDLE THROUGH.
- And the fact is, we get things done that way. I've seen lots of people use software, Web sites, and consumer products effectively in ways that are nothing like what the designers intended.
  - It's not important to us.
  - If we find something that works, we stick to it.

Billboard Design 101

- Take advantage of conventions
- Create effective visual hierarchies
- Break pages up into clearly defined areas
- Make it obvious what's clickable
- Eliminate distractions
- Format content to support scanning

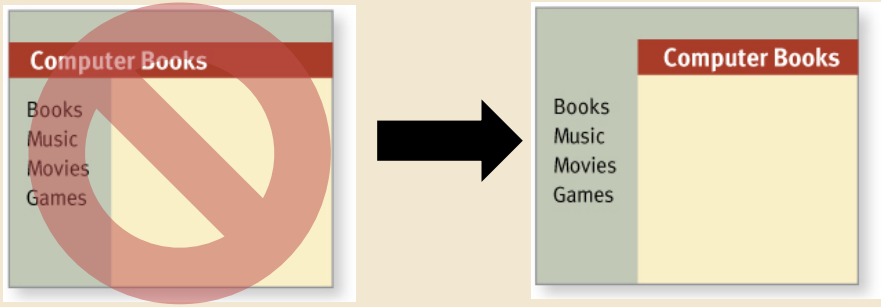
Conventions Are Your Friends

- One of the best ways to make almost anything easier to grasp in a hurry is to follow the existing conventions—the widely used or standardized design patterns. For example,
  - Stop signs.
  - Controls in cars.

- In the past twenty years, many conventions for Web pages have evolved. As users, we've come to have a lot of expectations about
  - Where things will be located on a page.
    - For example, users expect the logo identifying the site to be in the top-left corner.
  - How things work.
    - For example, almost all sites that sell products use the metaphor of a shopping cart and a very similar series of forms for specifying things like your method of payment, your shipping address, and so on.
  - How things look.
    - Many elements have a standardized appearance, like the icon that tells you it's a link to a video, the search icon, and the social networking sharing options.
- Innovate when you know you have a better idea, but take advantage of conventions when you don't.
- Consistency *is* always a good thing to strive for within your site or app.
- If you can make something *significantly* clearer by making it *slightly* inconsistent, choose in favor of clarity.

Create Effective Visual Hierarchies

- Each page should have a clear visual hierarchy.
  - Which things are most important, which things are similar, and which things are part of other things.
- The more important something is, the more prominent it is.
  - The most important elements are either larger, bolder, in a distinctive color, set off by more white space, or nearer the top of the page—or some combination of the above.
  - Things that are related logically are related visually.
  - Things are "nested" visually to show what's part of what.

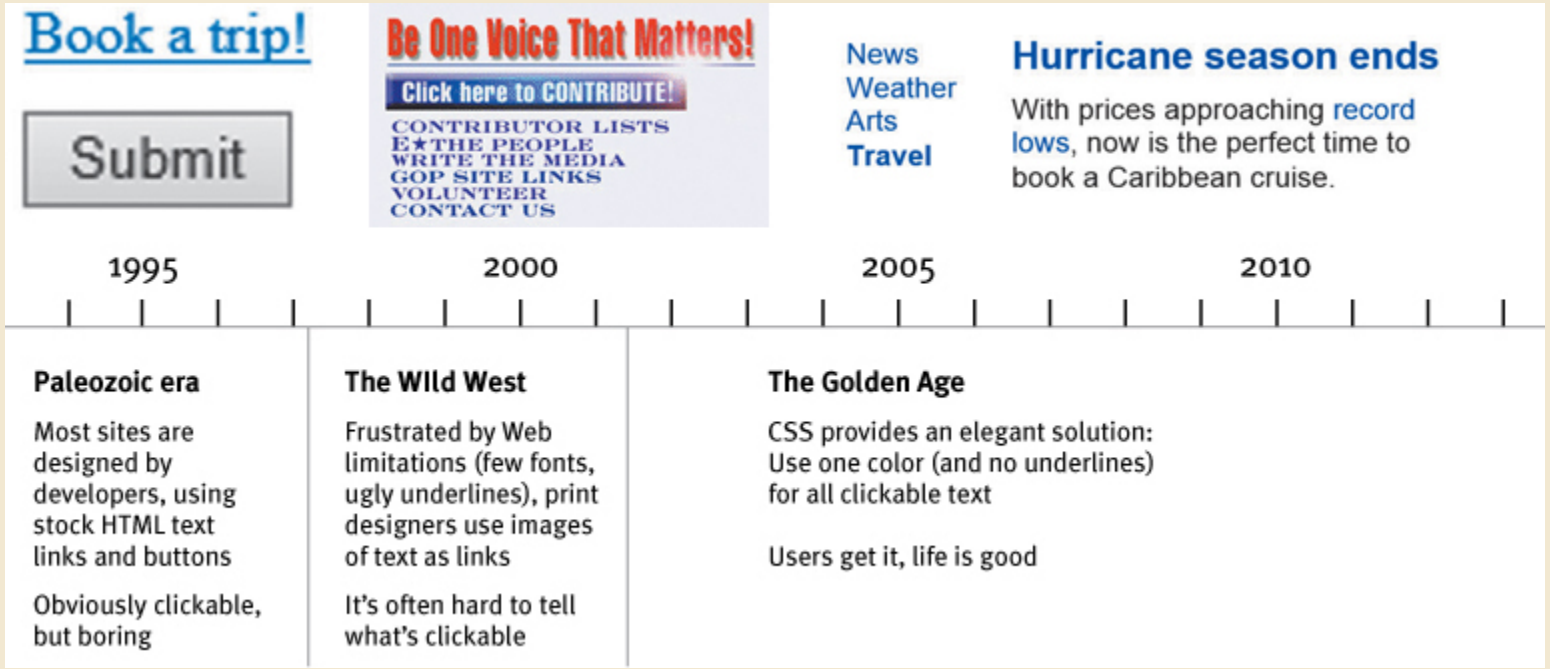


Break Up Pages Into Clearly Defined Areas

- Dividing the page into clearly defined areas is important because it allows users to decide quickly which areas of the page to focus on and which areas they can safely ignore.
- We're constantly parsing our environment (like the handles on doors) for these clues (to decide whether to pull or push).

Make It Obvious What's Clickable

- Since a large part of what people are doing on the Web is looking for the next thing to click, it's important to make it easy to tell what's clickable.



Keep The Noise Down To A Dull Roar

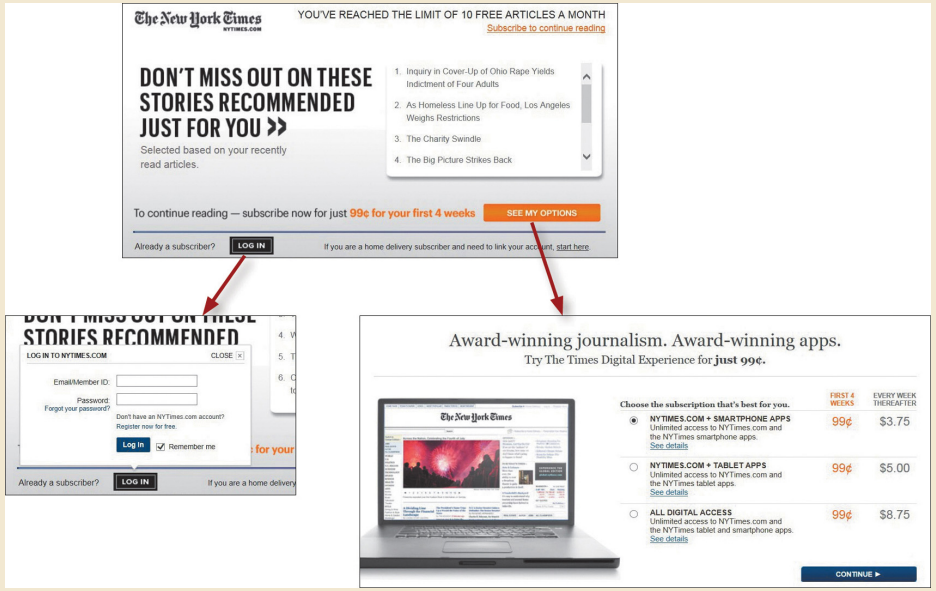
- There are really three different kinds of noise:
  - Shouting
    - The truth is, *everything* can't be important.
    - Shouting is usually the result of a failure to make tough decisions about which elements are really the most important and then create a visual hierarchy that guides users to them first.
  - Disorganization
    - This is a sure sign that the designer doesn't understand the importance of using grids to align the elements on a page.
  - Clutter
    - We've all seen pages—especially Home pages—that just have too much *stuff*.

Format Text To Support Scanning

- Much of the time—perhaps most of the time—that users spend on your Web pages is spent scanning the text in search of something.
  - Use plenty of headings.
  - Keep paragraphs short.
  - Use bulleted lists.
  - Highlight key terms.

Why Users Like Mindless Choices

- How *hard* each click is
  - the amount of thought required and the amount of uncertainty about whether I'm making the right choice.
- Links that clearly and unambiguously identify their target give off a strong scent that assures users that clicking them will bring them nearer to their "prey."
  - Ambiguous or poorly worded links do not.
- **Brief:** The smallest amount of information that will help me
- **Timely:** Placed so I encounter it exactly when I need it
- **Unavoidable:** Formatted in a way that ensures that I'll notice it



The Art Of Not Writing For The Web

- Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences, for the same reason that a drawing should have no unnecessary lines and a machine no unnecessary parts.
- Getting rid of all those words that no one is going to read has several beneficial effects:
  - It reduces the noise level of the page.
  - It makes the useful content more prominent.
  - It makes the pages shorter, allowing users to see more of each page at a glance without scrolling.
- You can—and should—eliminate as much *happy talk* as possible.
- Your objective should always be to eliminate instructions entirely by making everything self-explanatory, or as close to it as possible.
  - When instructions are absolutely necessary, cut them back to the bare minimum.

What's The Difference?

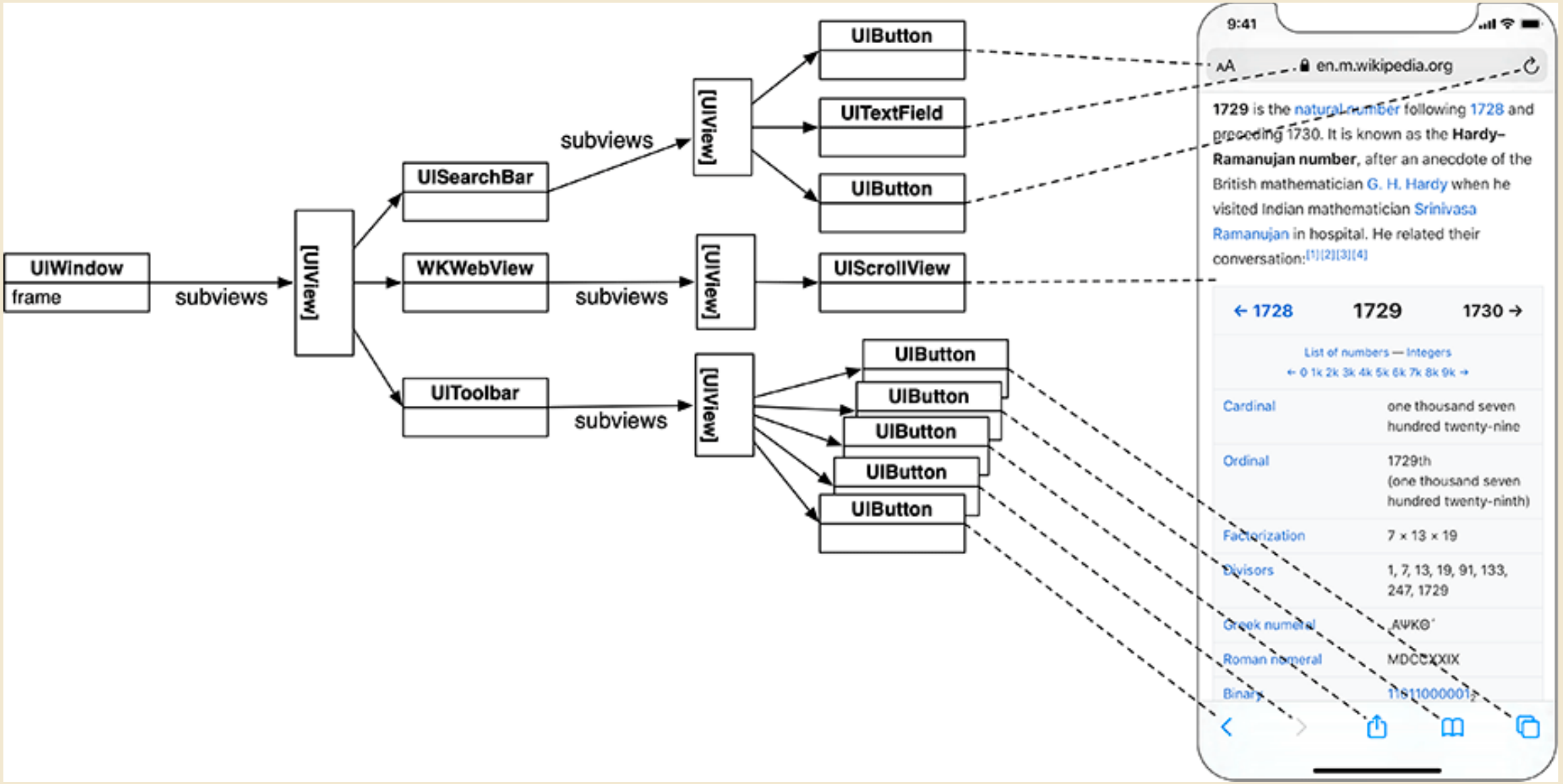
- What's the difference between computer and mobile?
  - The basic principles are still the same.
  - If anything, people are moving faster and reading even less on small screens.
- It's all about tradeoffs
  - Constraints
    - things you *have to* do and things you *can't* do
  - Tradeoffs
    - the less-than-ideal choices you make to live within the constraints
- Whenever you're designing, you're dealing with constraints. And where there are constraints, there are tradeoffs to be made.

App Design Exploration

Outline

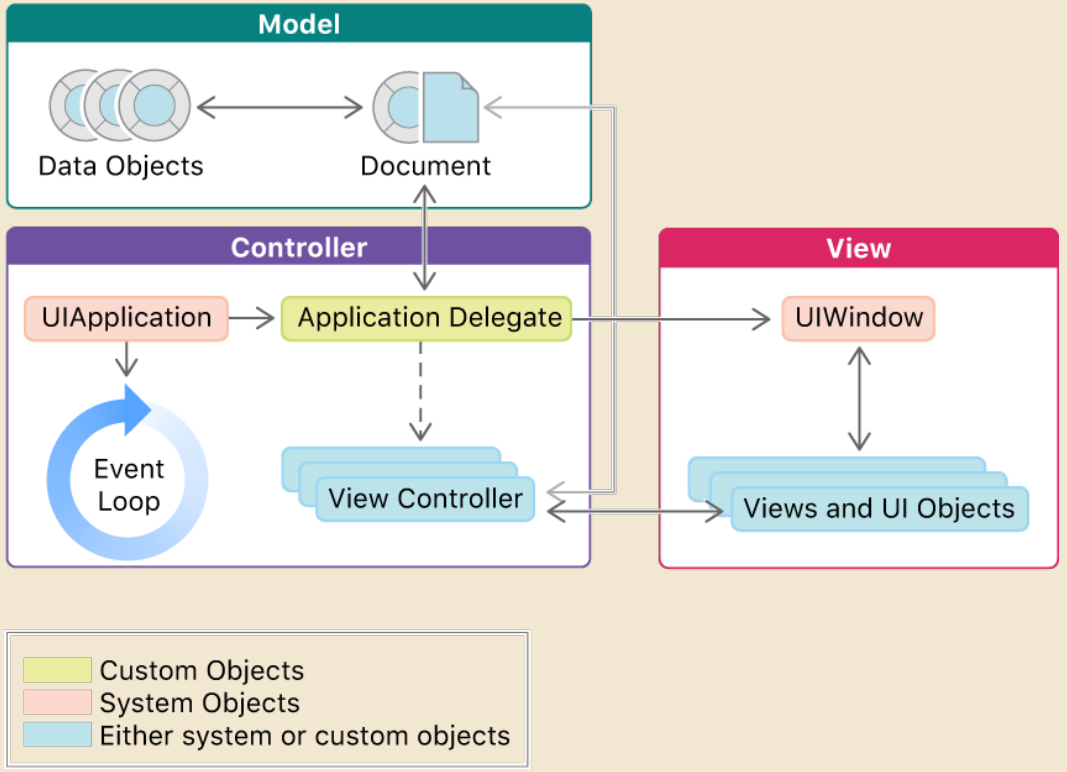
- iOS and Swift Overview
- UI/UX and design principle
- UI Views and Bars
- Case Study
- Advanced Topics
  - Delegate
  - ARC (Automatic Reference Counting)
- Implementation

The View Hierarchy



UI Kit

- 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
```

Functions

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

Data Types

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

Structures

```
struct MyStruct {
}
```

Classes

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

IF Statement

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

Switch Statement

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

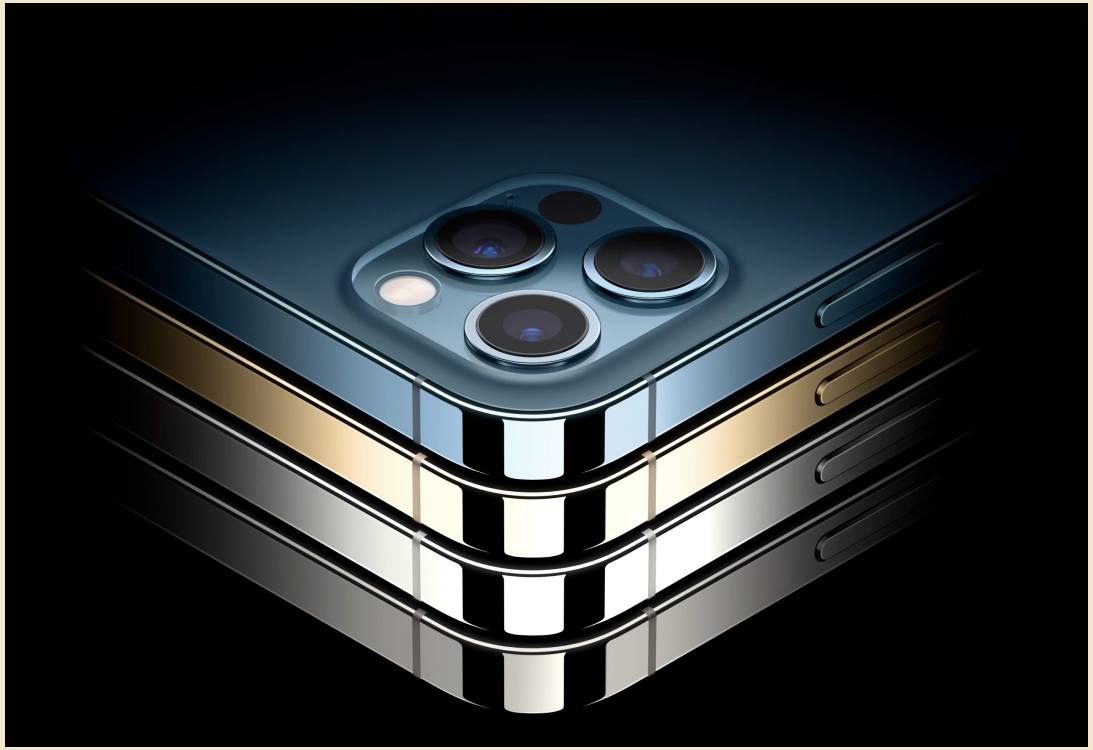
Loops

```
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



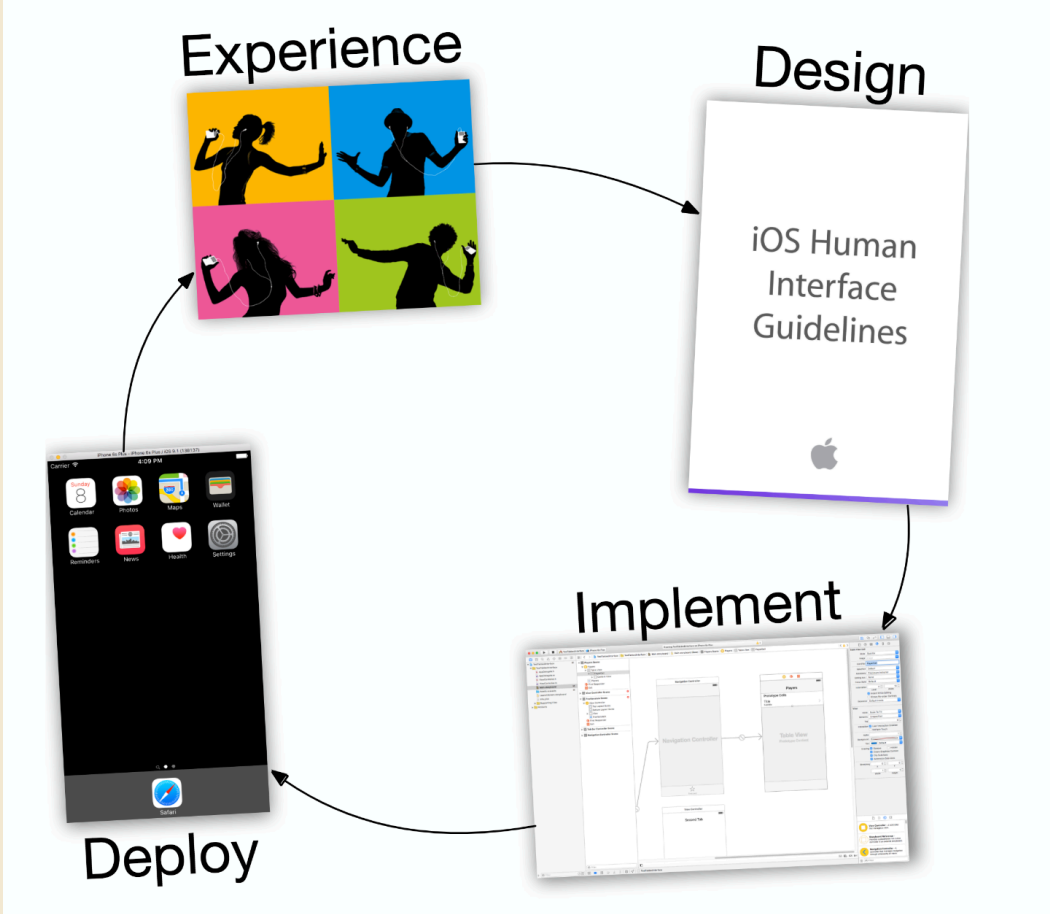
iOS Design Themes

- 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
  - 內容同常會佔滿畫面，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

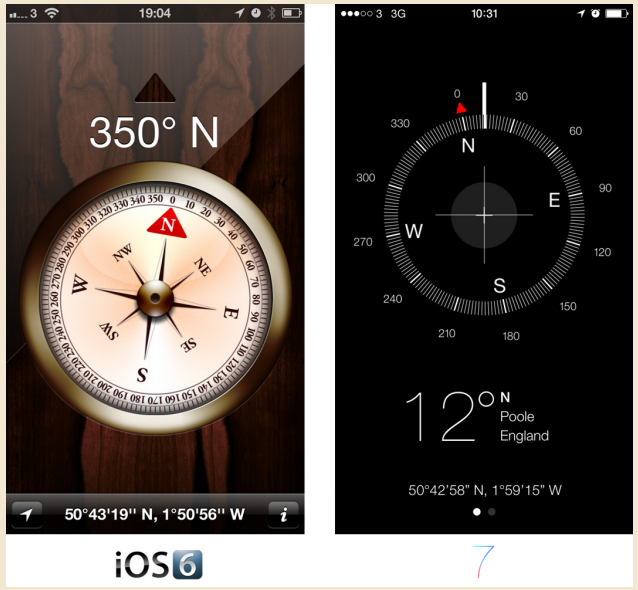
- Aesthetic integrity（設計美學完整性）
  - 外觀與功能的整合程度
  - 利用醒目且清楚的圖形讓使用者專注在功能上
  - 助於預測使用者行為
- Consistency（一致性）
  - 使用系統所提供並為使用者所熟悉的的圖形、介面、文字格式與一致性的術語
  - 提供使用者預期的功能與使用方式
- Direct Manipulation（直接操作）
  - 使用者可以透過畫面直接的操作並看見結果，且能增進理解
  - 例如：觸碰螢幕或是旋轉螢幕，並立刻看到所改變的結果
- Feedback（回饋）
  - 回饋讓使用者知道抱持瞭解狀況
  - 透過內建的聲音或動畫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

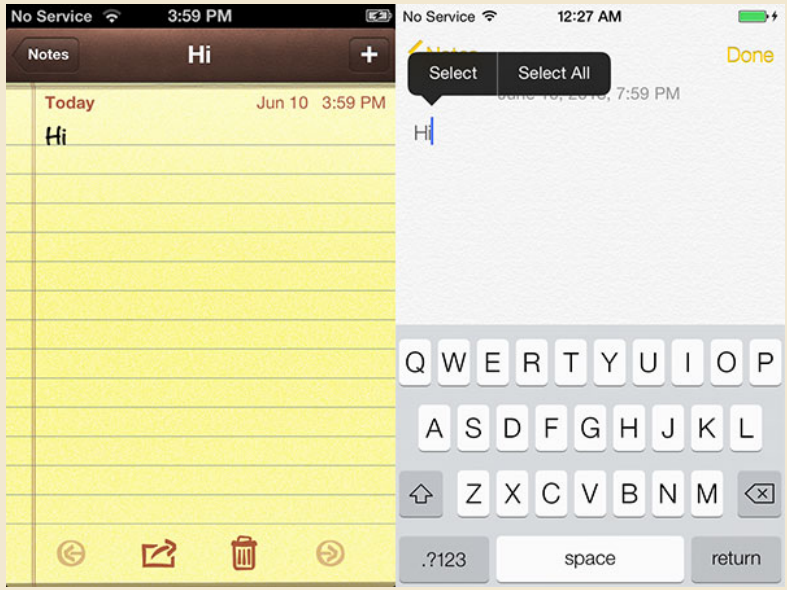
iOS Design Concepts



Apple design concepts

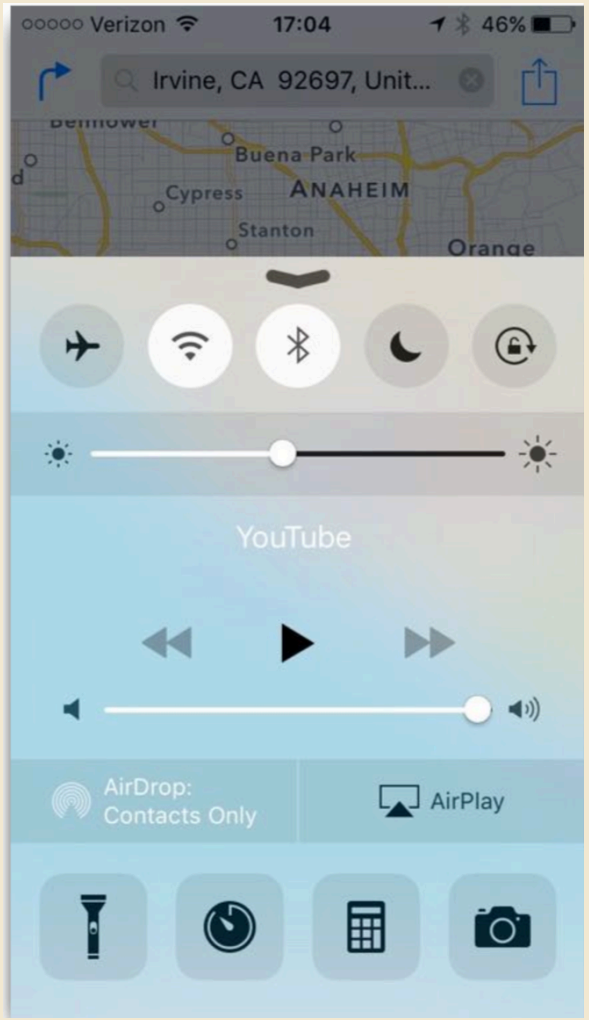
- 時常改變設計的策略
  - 寫實設計到平面設計 (flat design)

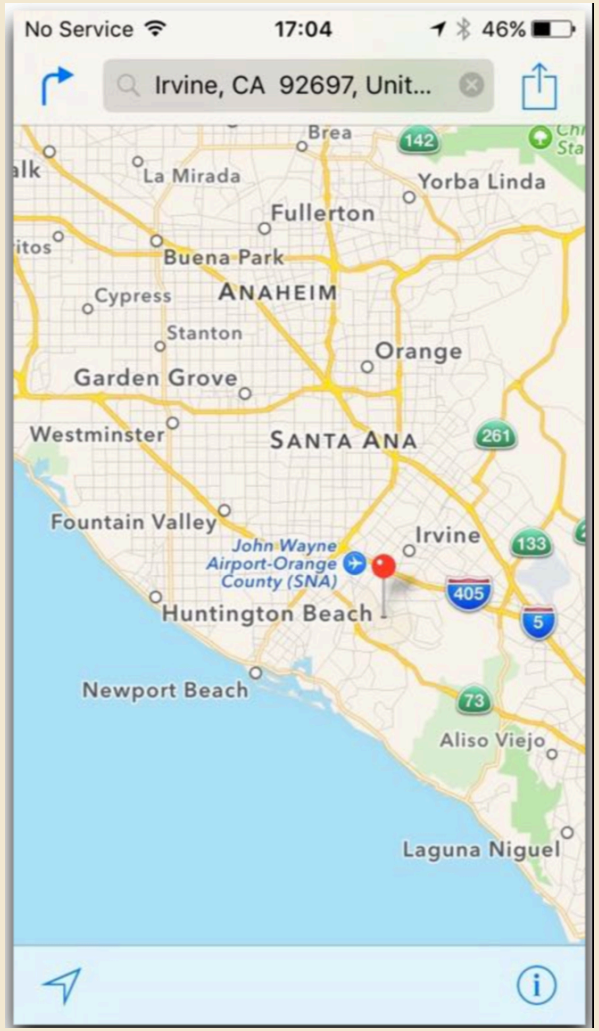




Defer to Content

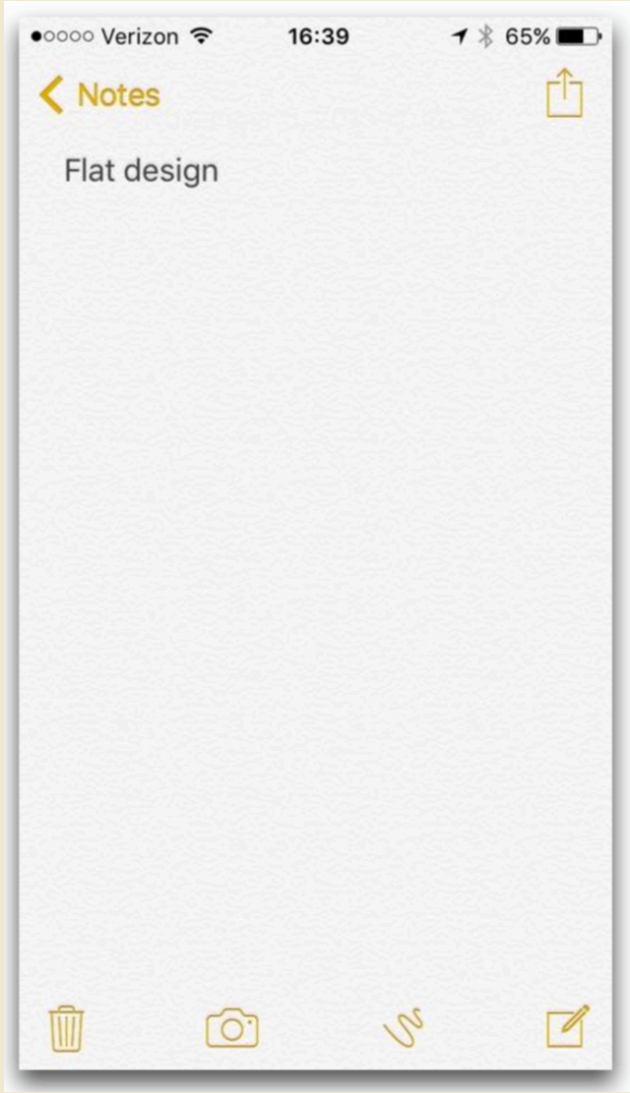
- 使用整個螢幕
- 減少或消除擬真
- 利用半透明

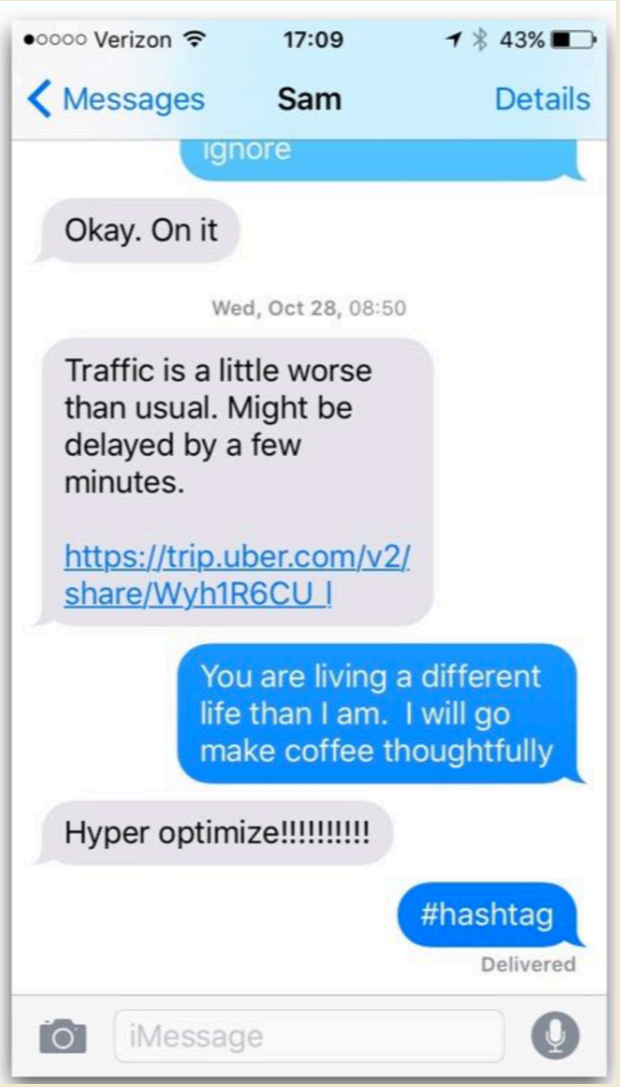




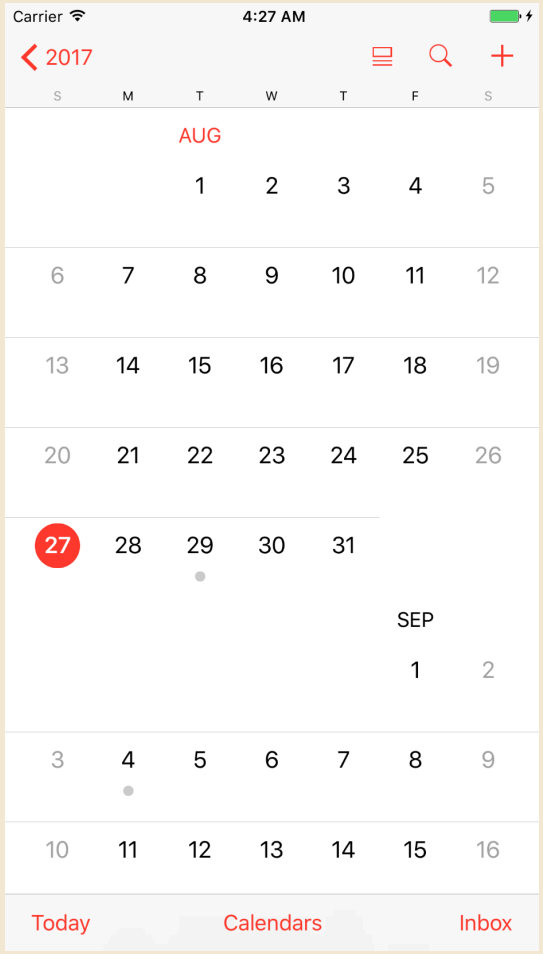
Promote Content Clarity

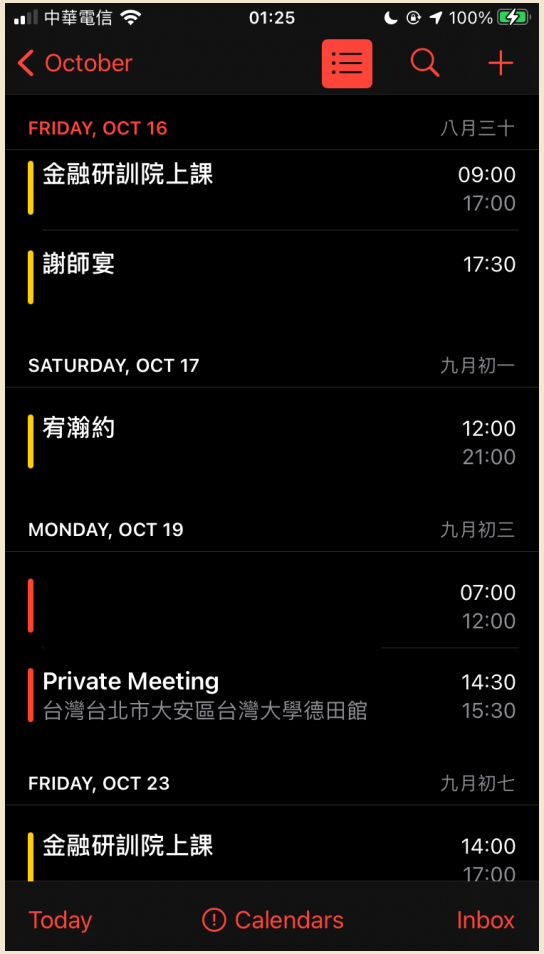
- 利用空白減少干擾
- 利用顏色
  - 相同作用
  - 標明同樣目的
- 利用無邊框按鈕





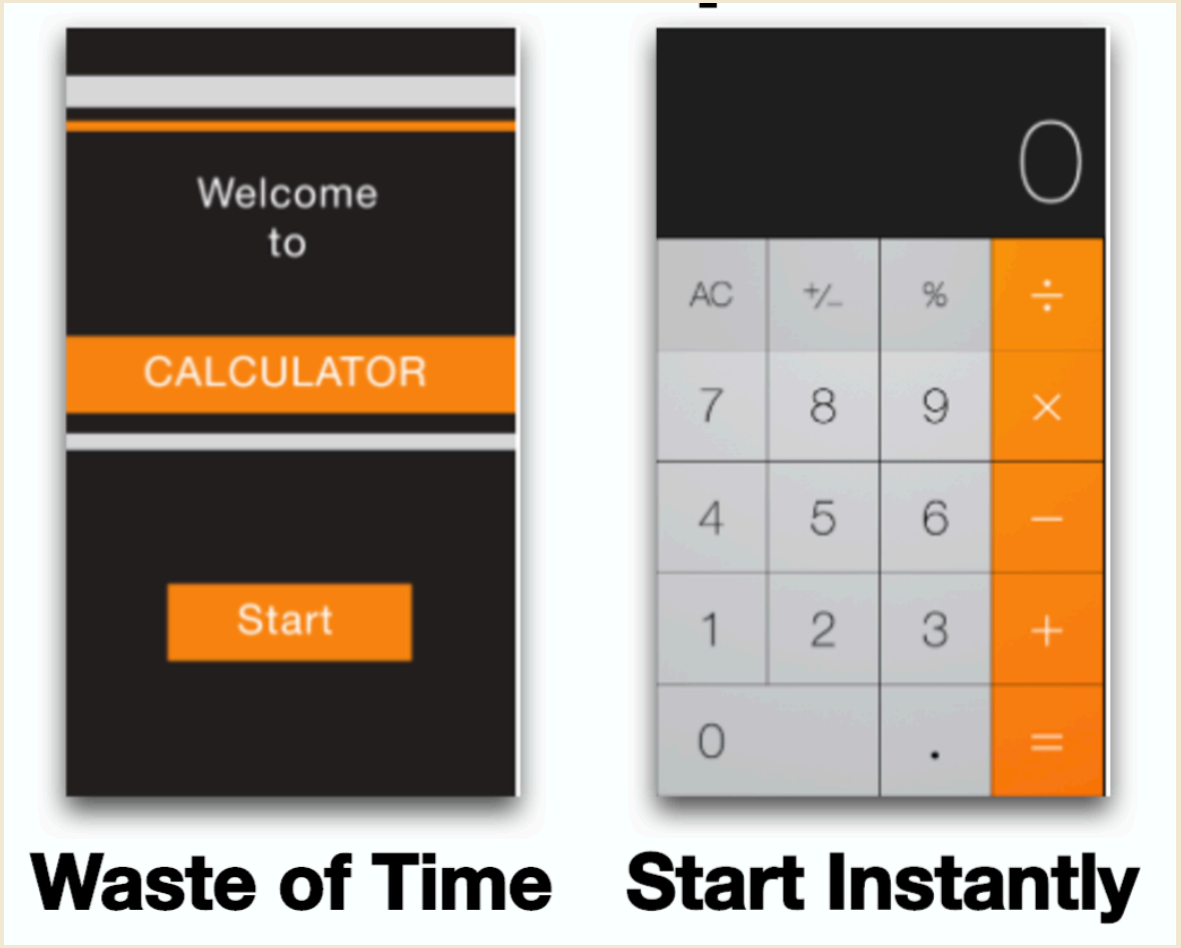
Use Depth to Reflect Content





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
- 進入之後不要要求使用者重新開啟



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

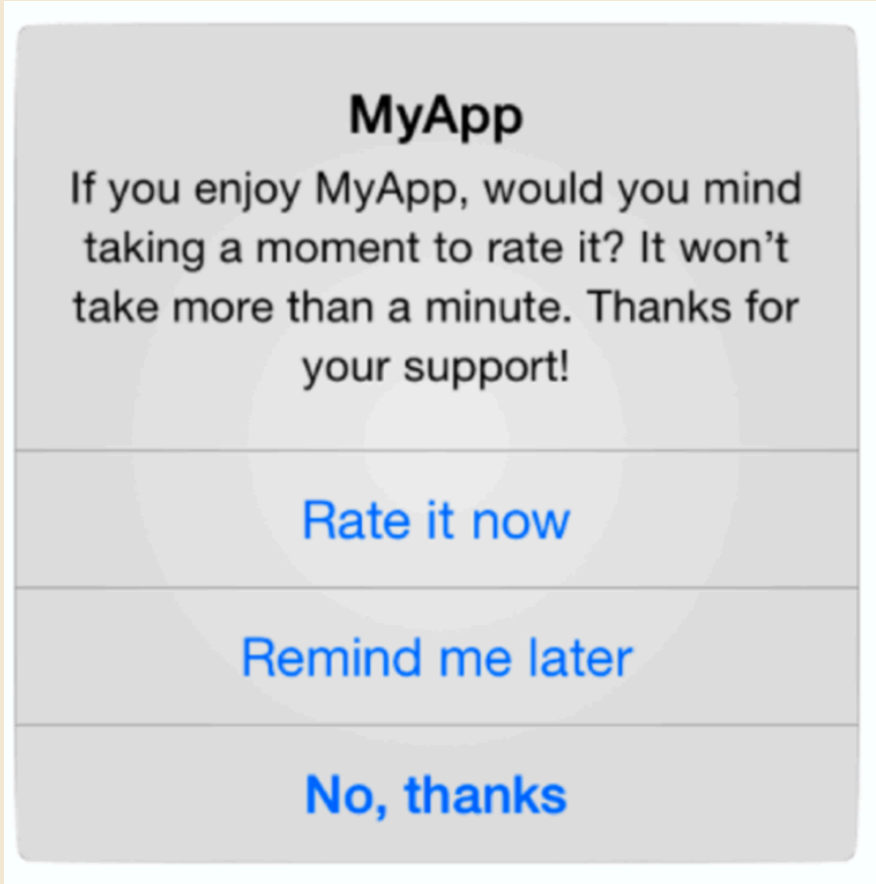
- 等待使用者輸入
  - 沒有必要之前不要求使用者登入
  - 沒有必要之前不詢問系統權限
- 解釋相關權限要求

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
- 提供進入畫面 (launch screen)
  - 設計一個跟app進入後第一個畫面出乎一樣的畫面
  - 可以直接使用單一色彩的畫面
  - 符合手機色彩模式
- 進入畫面不應該包含
  - 關於的視窗
  - 文字描述
  - logo或品牌廣告

Restarting

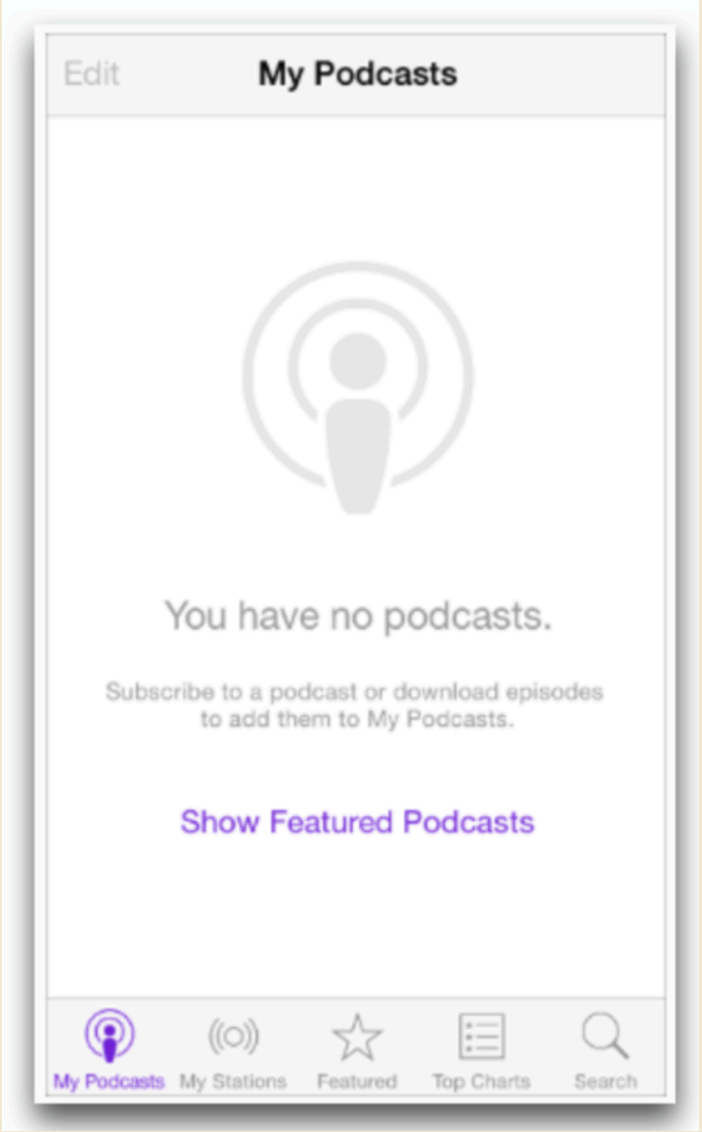
- 重新開始在使用者上次離開的狀態
  - 需要將狀態儲存
- 避免使用中重新啟動
  - 看起來像是當掉

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

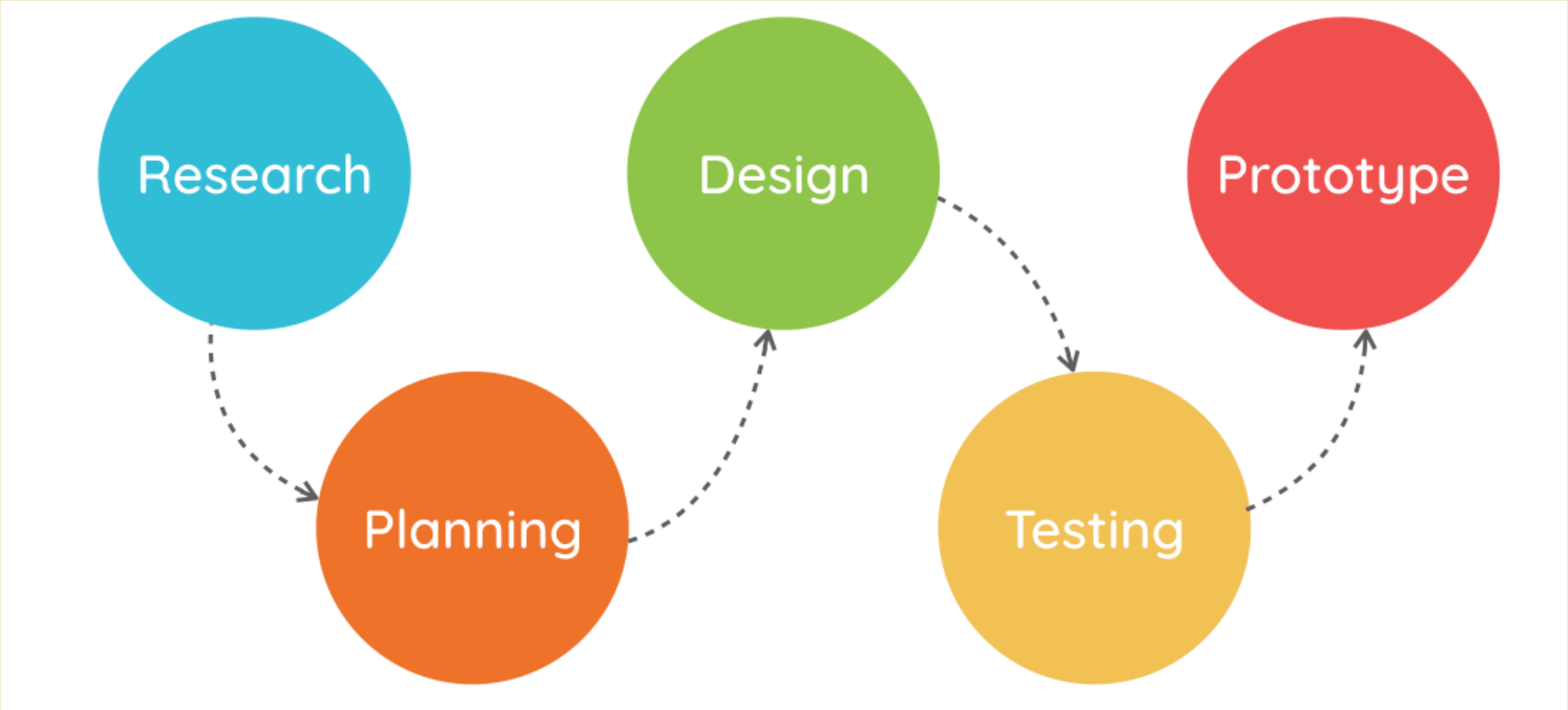
- 否則使用著會覺得疑惑甚至會覺得APP無法運作

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

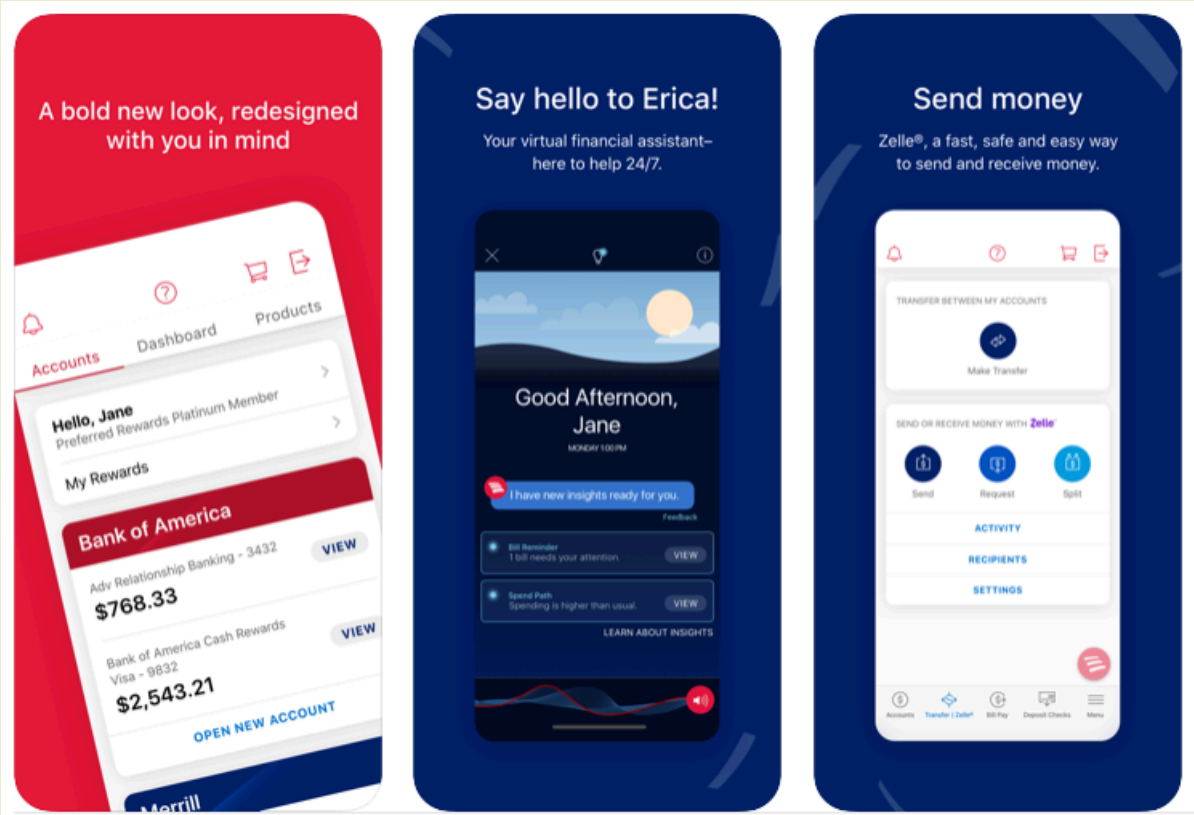


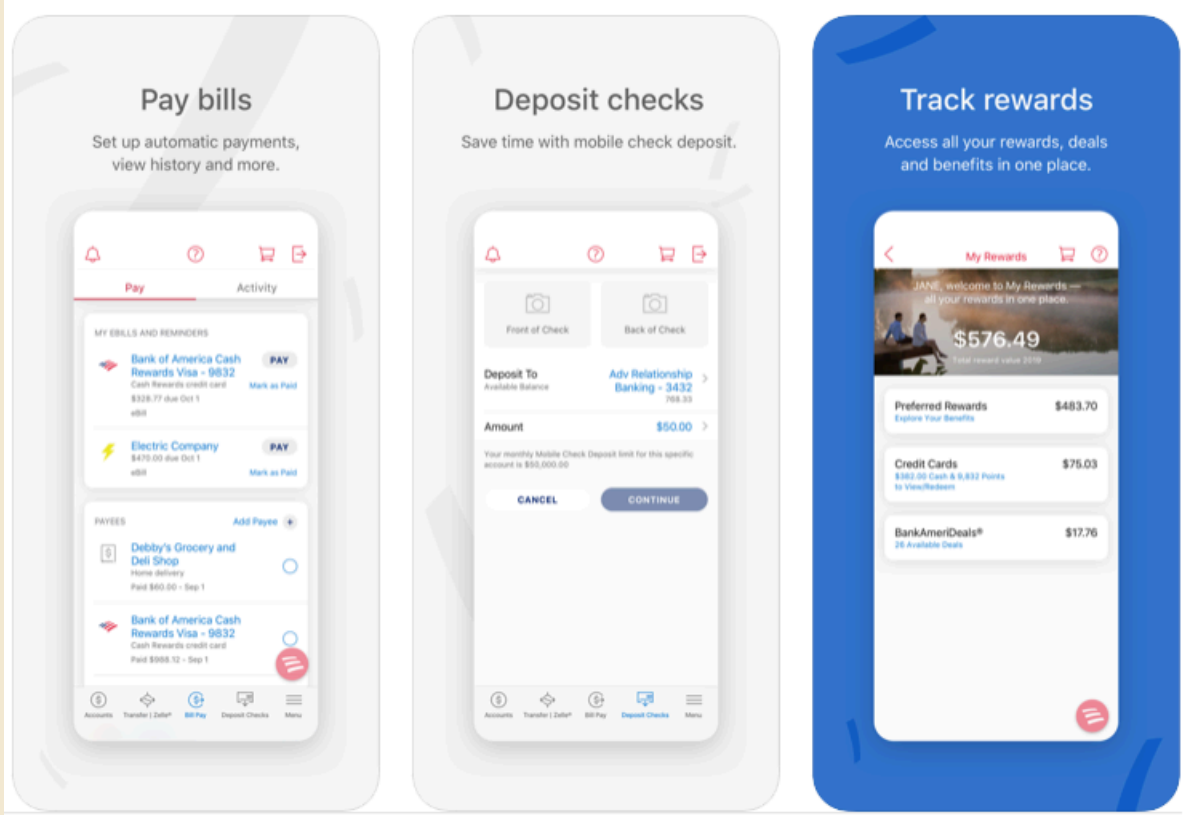
https://blog.prototypr.io/monkiri-ui-case-study-71fcb100f1a1
https://www.sam-solutions.com/blog/the-guide-to-developing-a-banking-app/

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





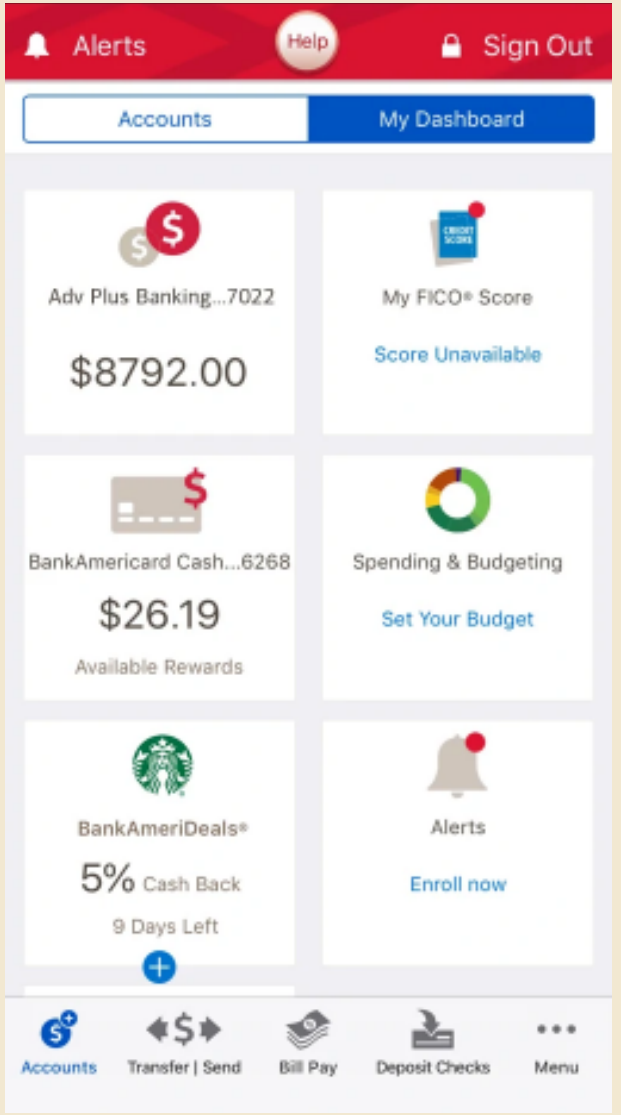
https://apps.apple.com/tw/app/bank-america-mobile-banking/id284847138#?platform=iphone

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

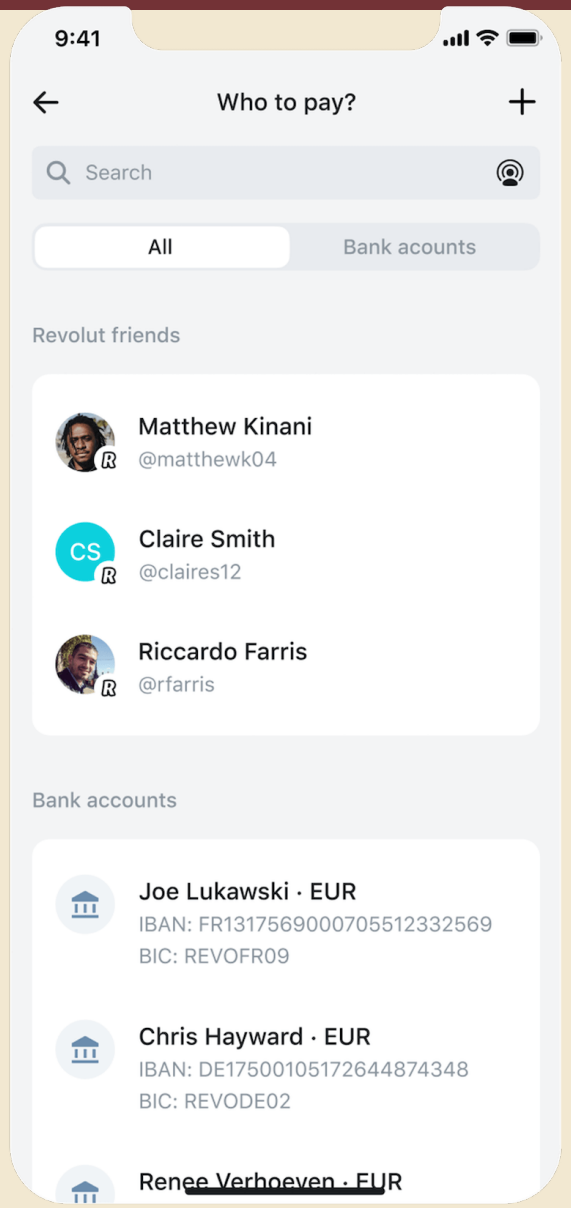
- Simple UI
  - 第一印象
  - 簡單、明確且友善的介面
  - 可以透過簡單的按鈕找到需要的功能
- Easy Account Management
- Mobile Transfer & Payment
- Check Deposit
- Security

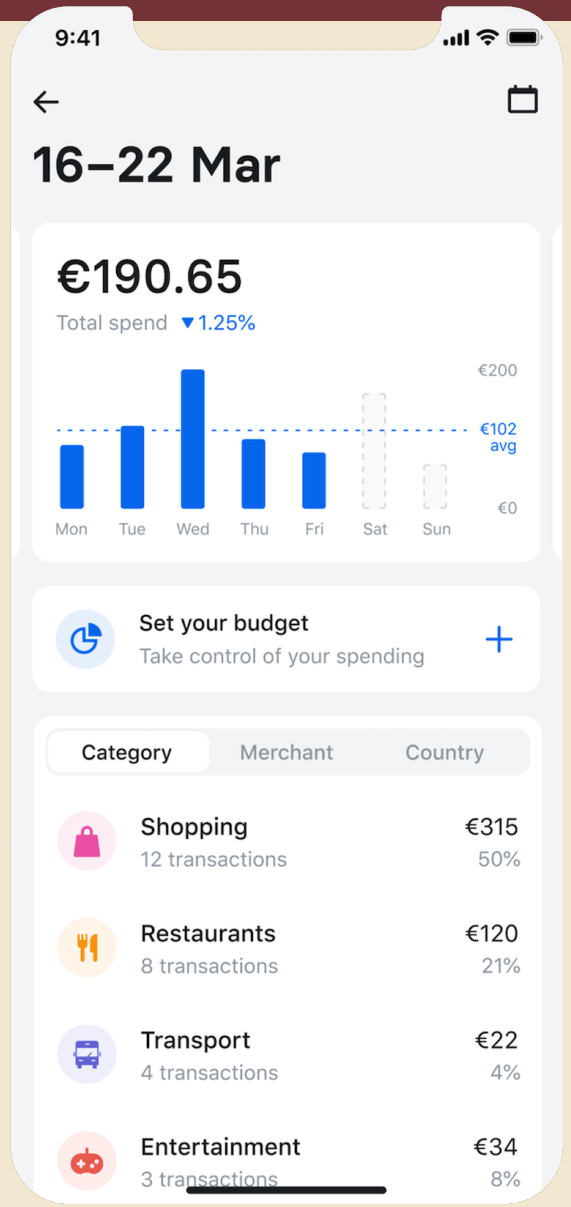


Case Study: App of Taiwan's Banks

Great Patterns and Example

- Revolut
  - Heuristics and trendy aesthetics

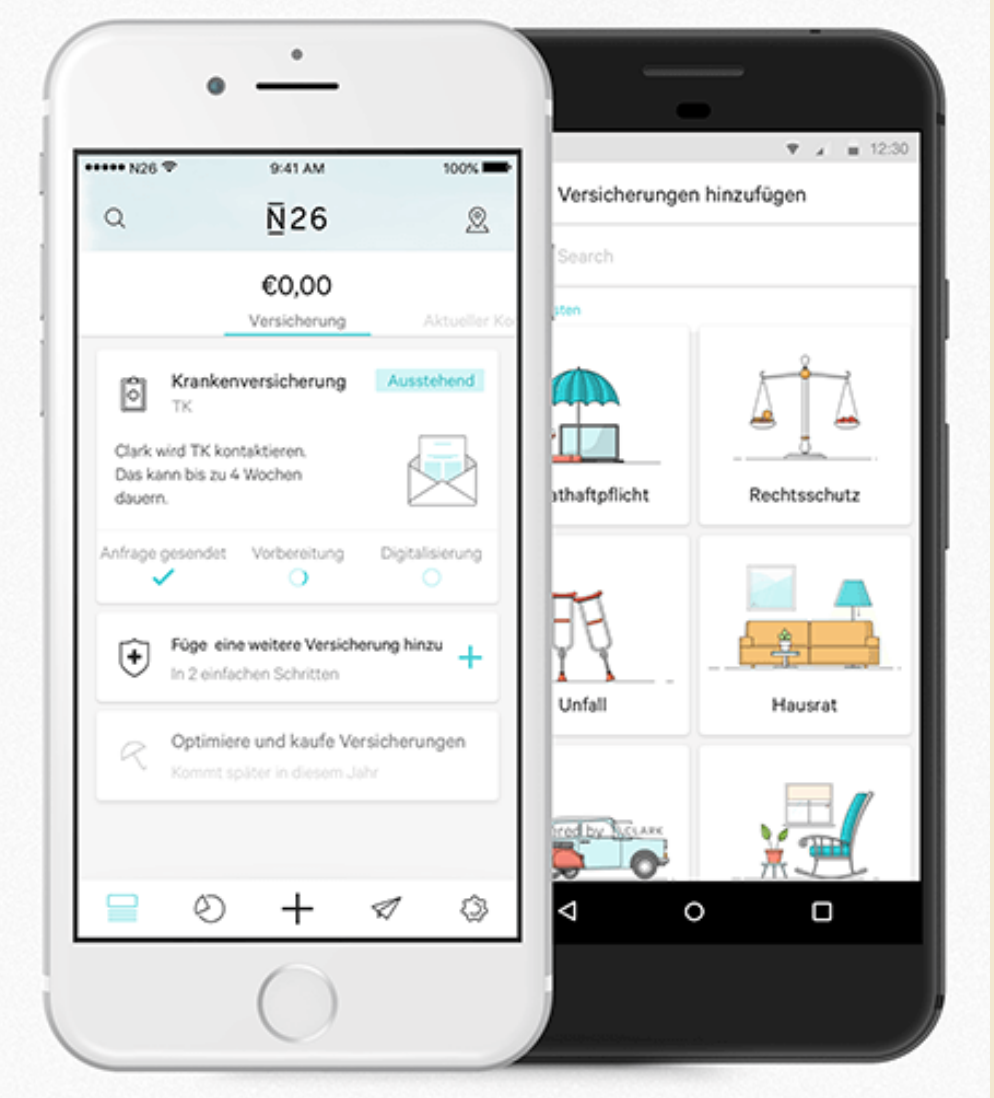




https://www.justinmind.com/blog/banking-app-design/

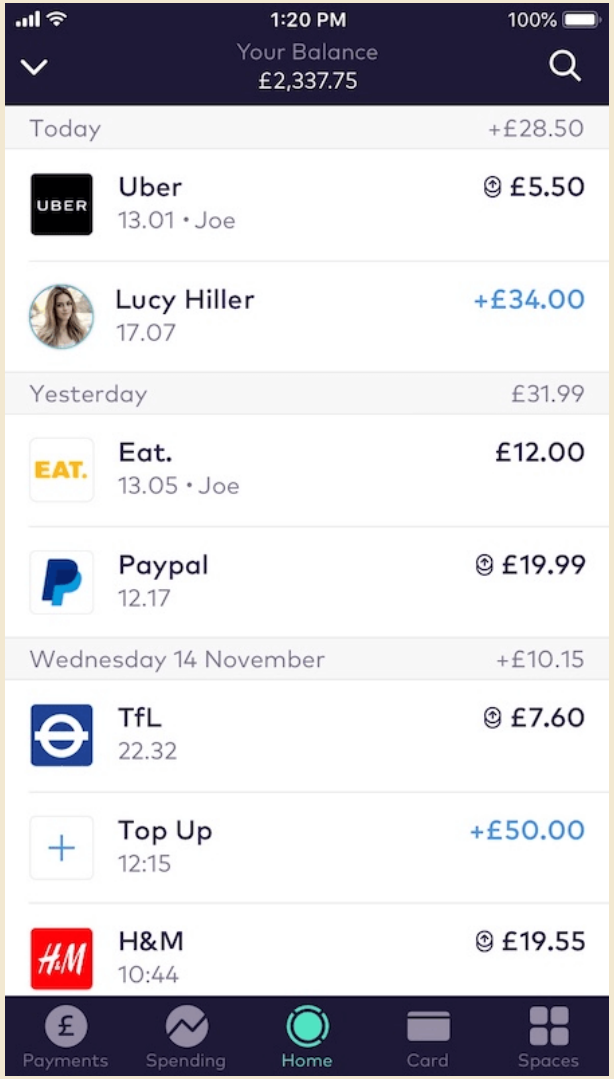
Great Patterns and Example

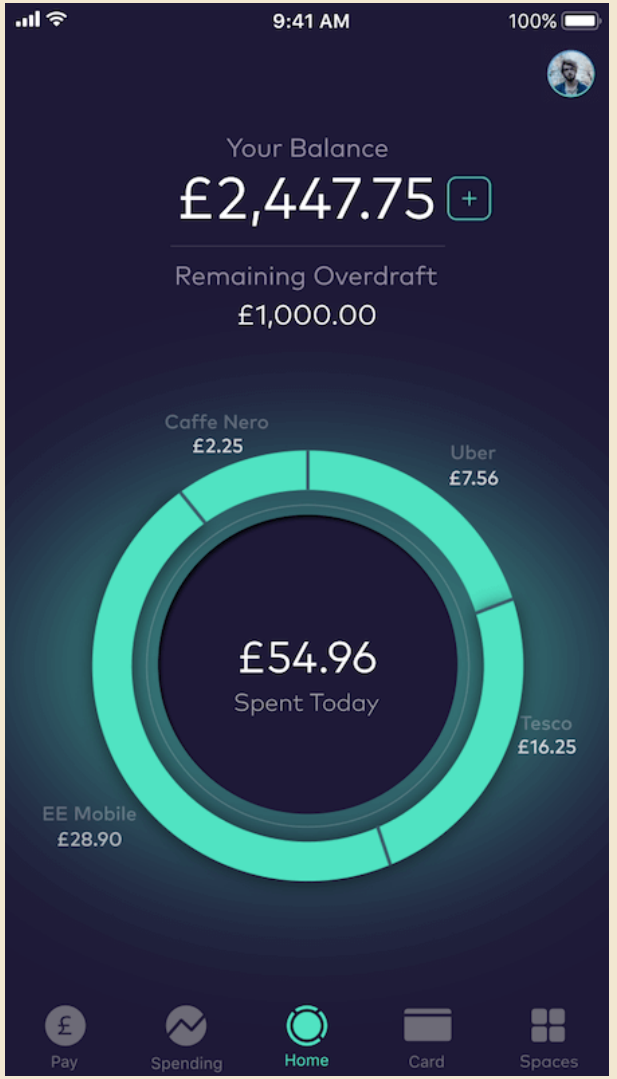
- Cards instead of lists: N26



Great Patterns and Example

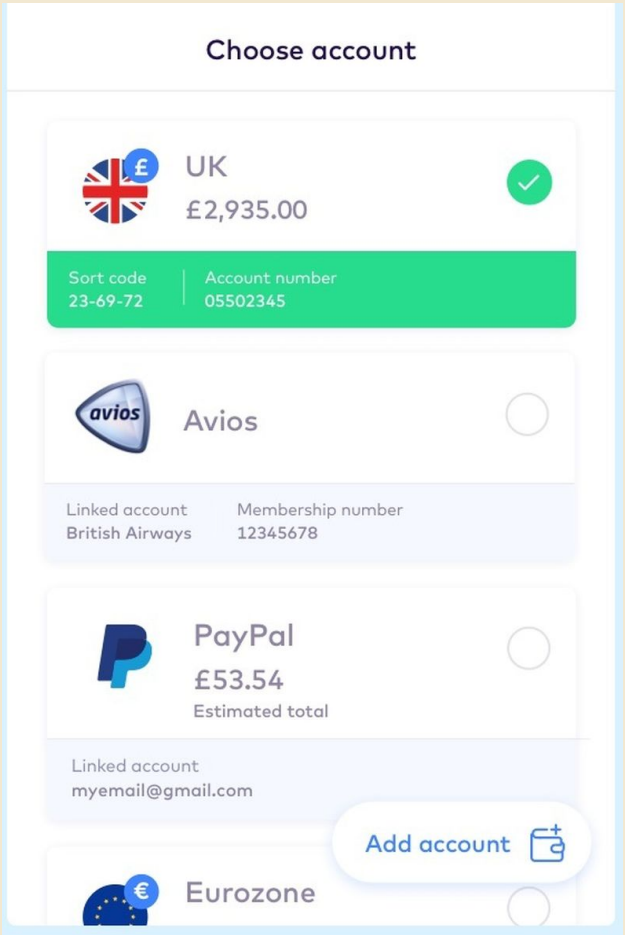
- Landing pages with statistics: Starling Bank

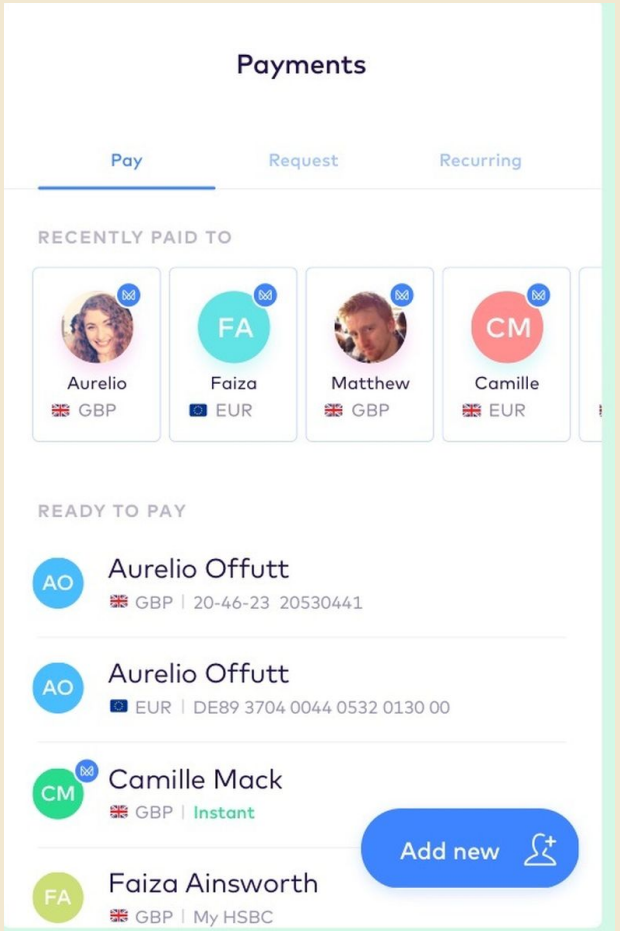




Great Patterns and Example

- Monese



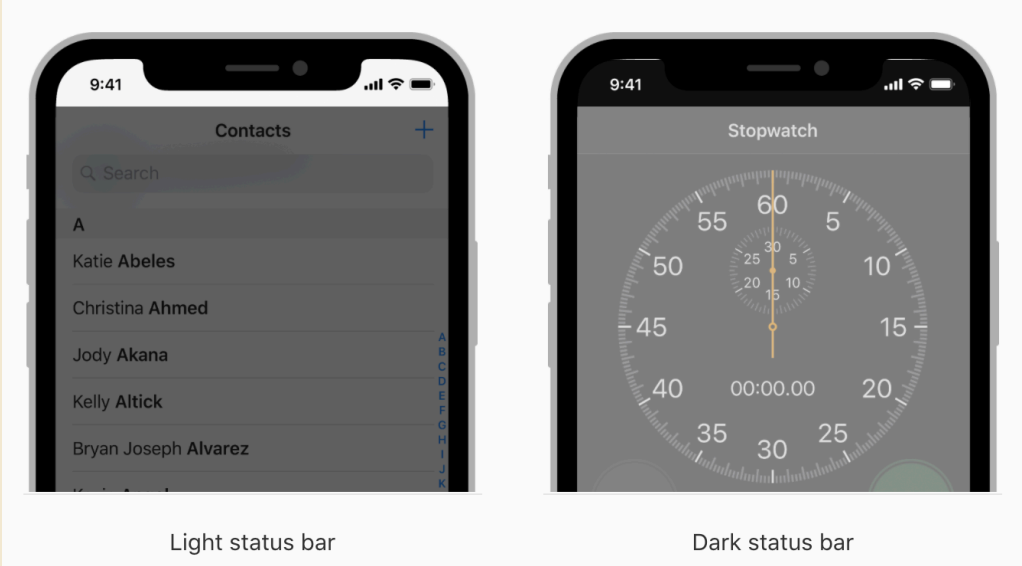


Bars

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

Status Bars

- Use the system-provided status bar
- Set globally in UIApplcation
  - 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

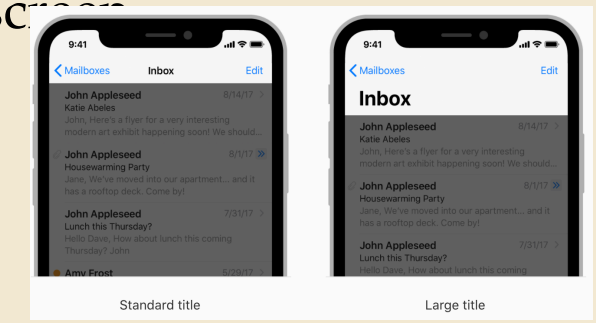


Light status bar

Dark status bar

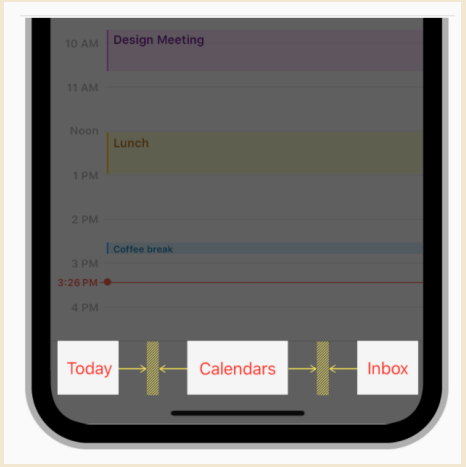
Navigation Bars

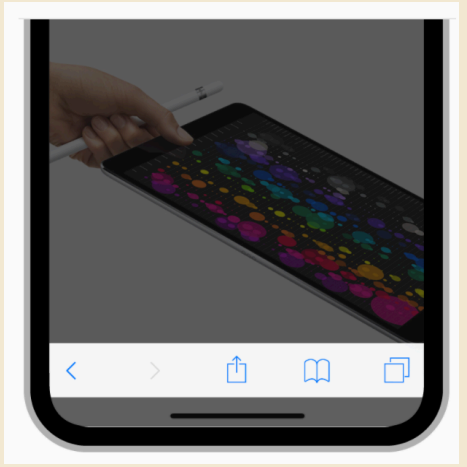
- Appears at the top of an app screen, below the status bar, and enables navigation through a series of hierarchical screens.
  - Back button, often labeled with the title of the previous screen, appears on the left side of the bar
  - The right side of a navigation bar contains a control, like an Edit 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





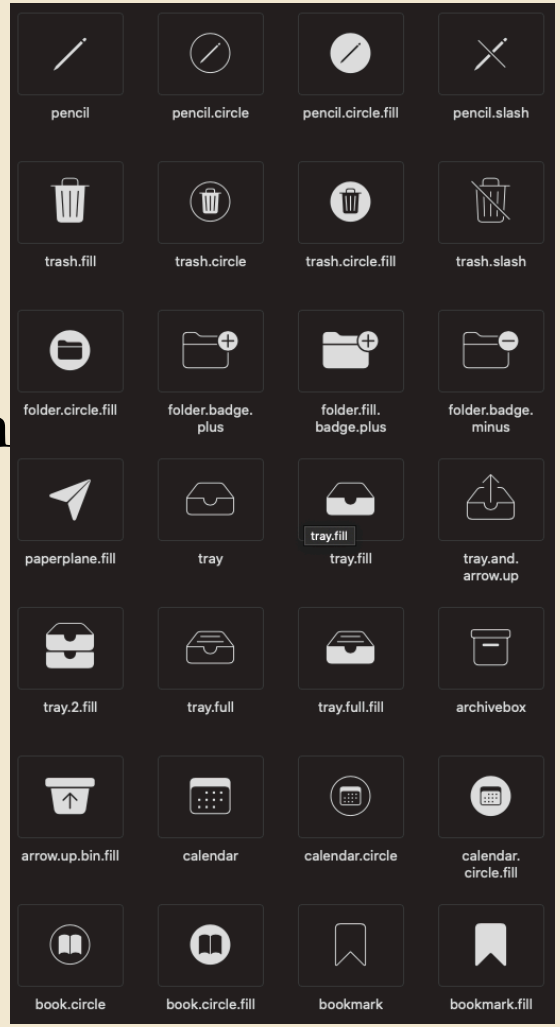
Tab Bars

- 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.

SF Symbols



Toolbar Icons

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

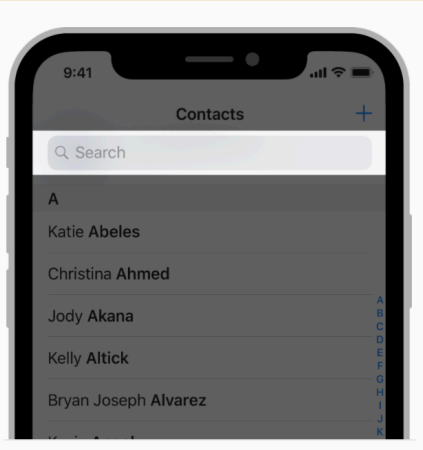
Tab Bar Icons

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

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

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**



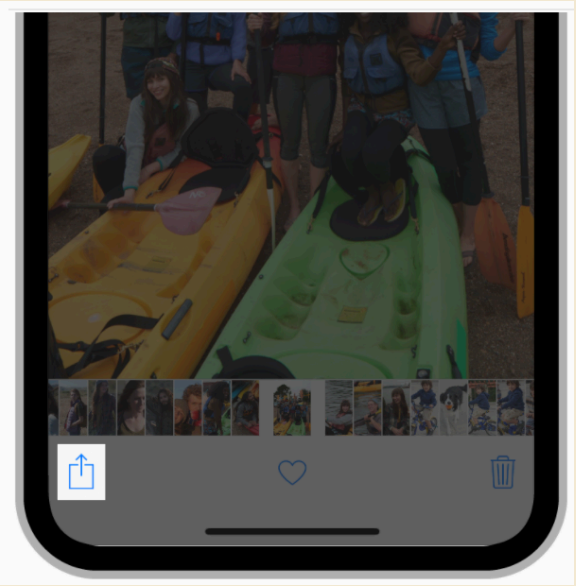
- A scope bar can be added to a search bar to let people refine the scope of a search.

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

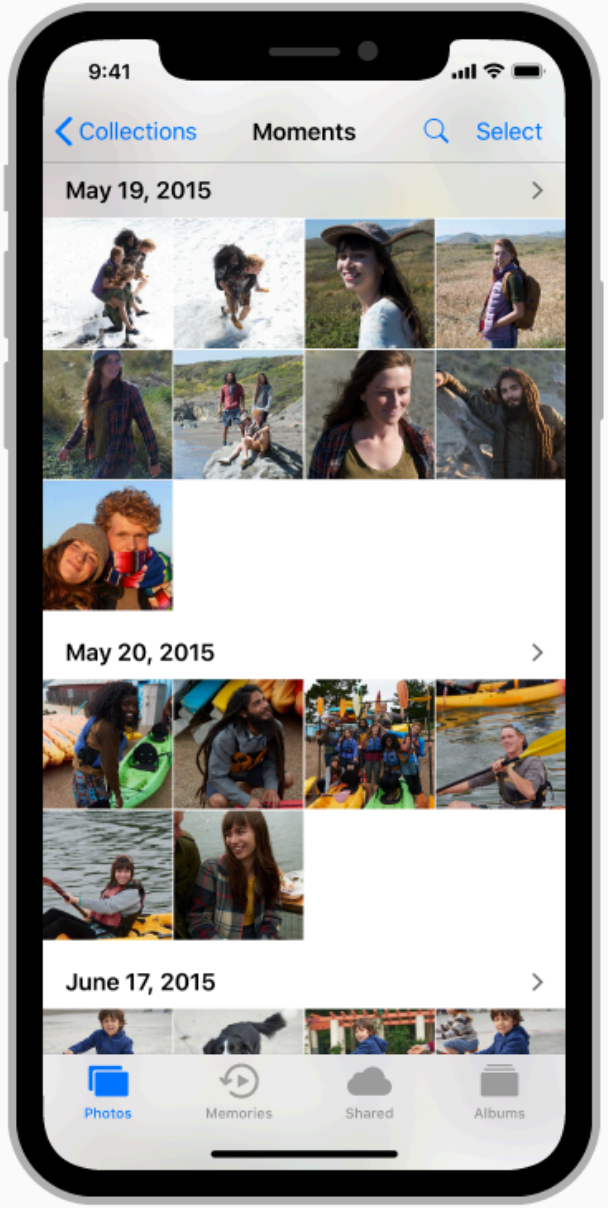
Activity/ Activity View Controller

- 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

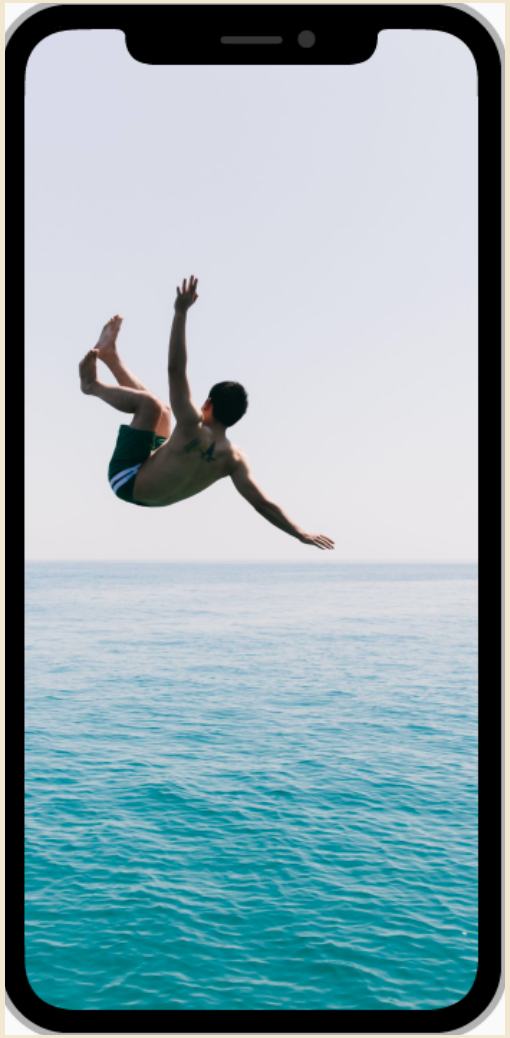


Container View Controller

- 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

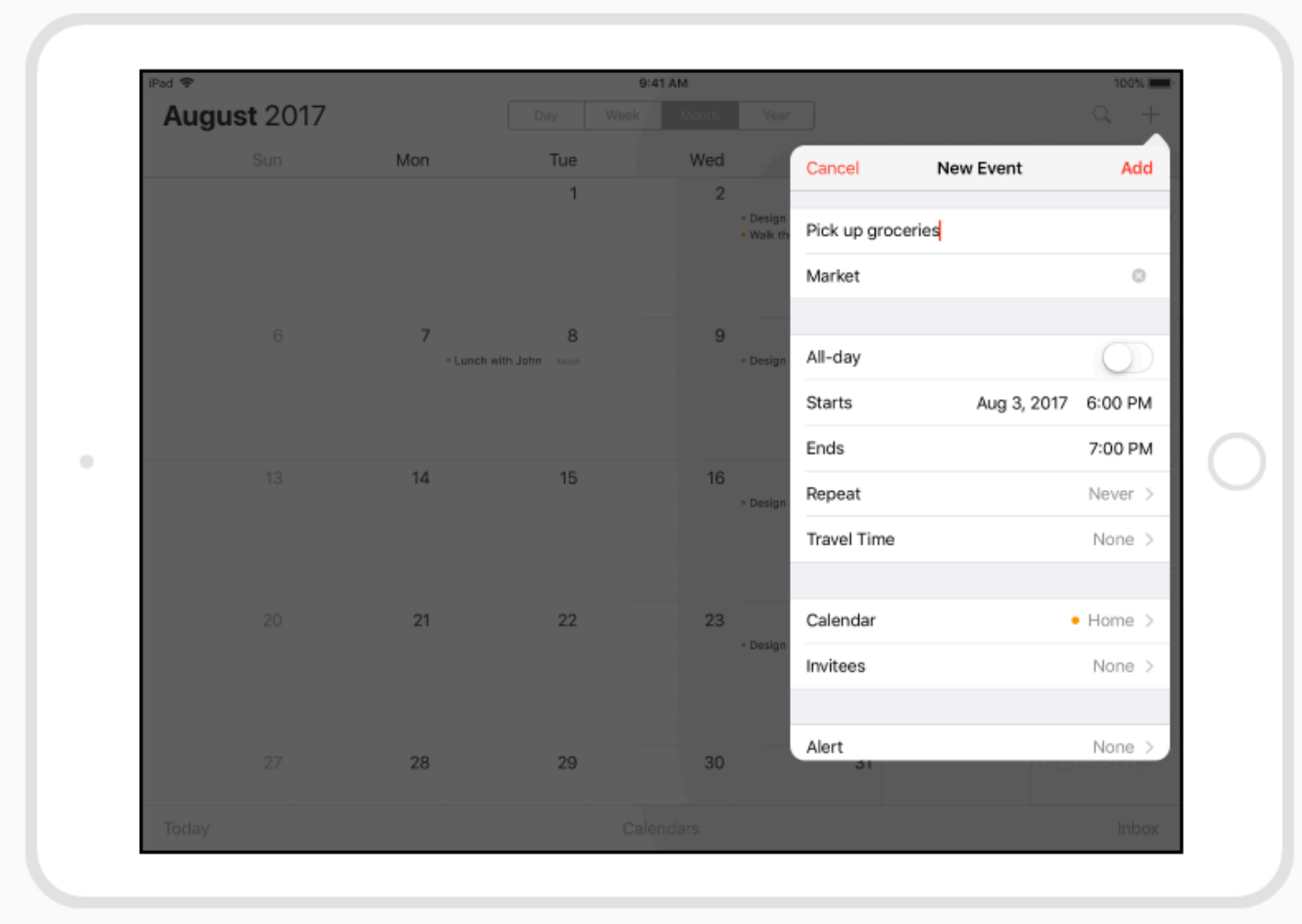
- 顯示地圖的畫面
  - 必須搭配MapKit
- 可以顯示不同方式的顯示
  - 一般地圖
  - 衛星地圖
  - 混合地圖
- 可以標記地點

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



Scroll View

- 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

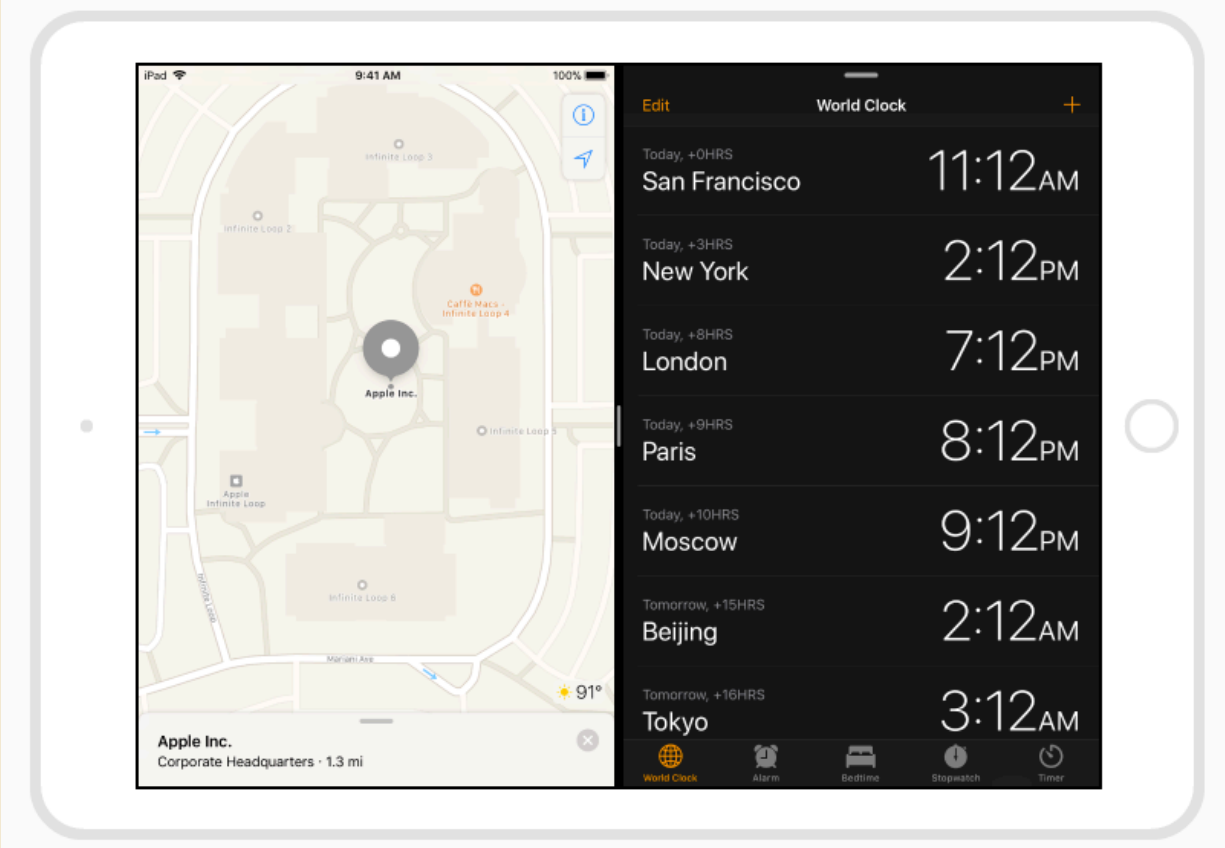
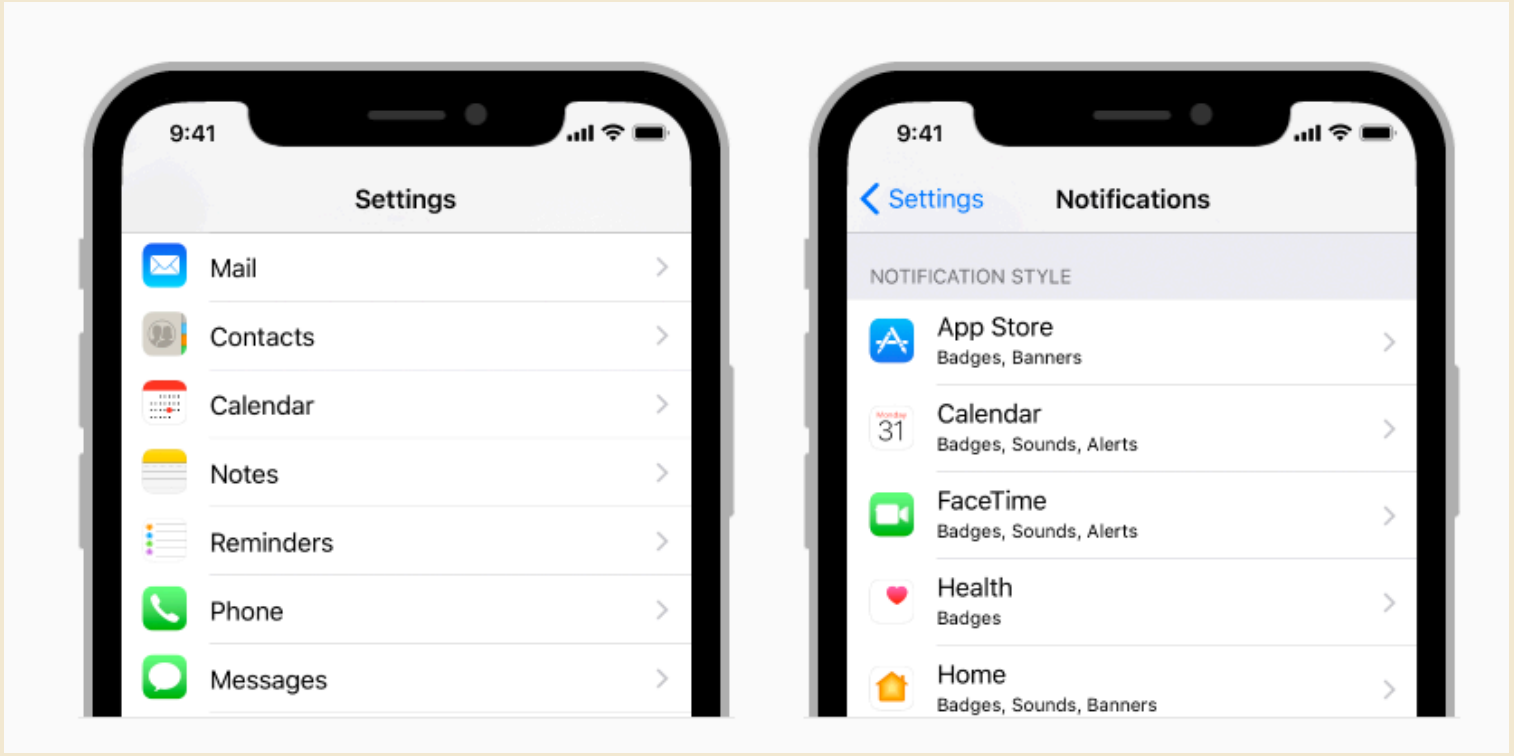


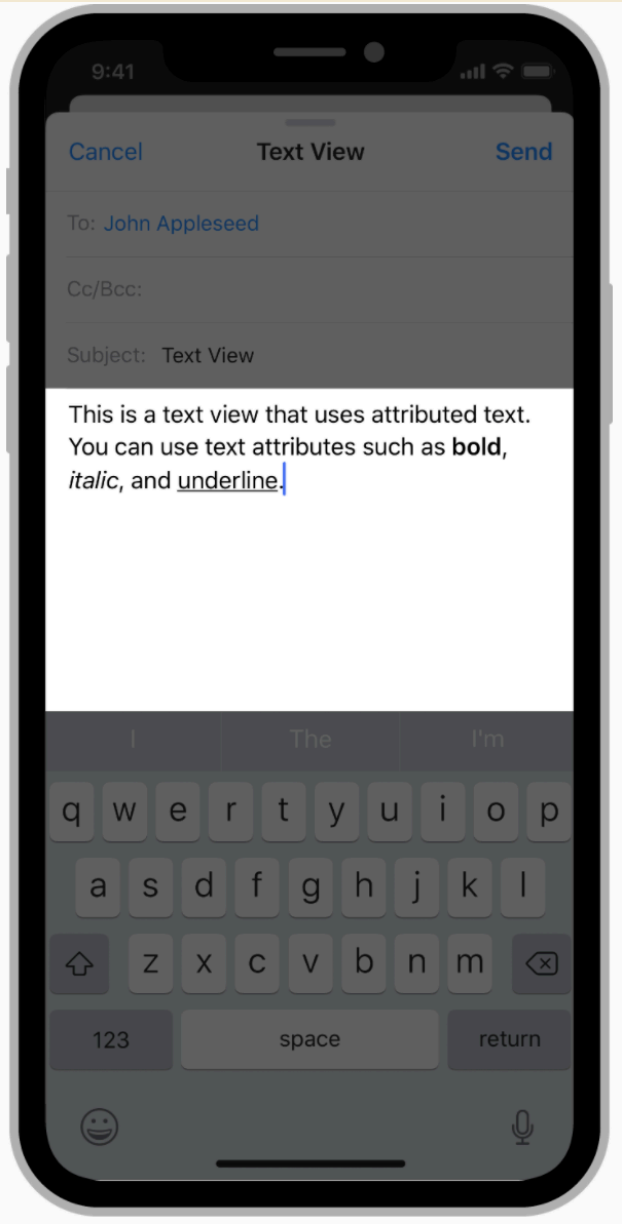
Table View

- 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



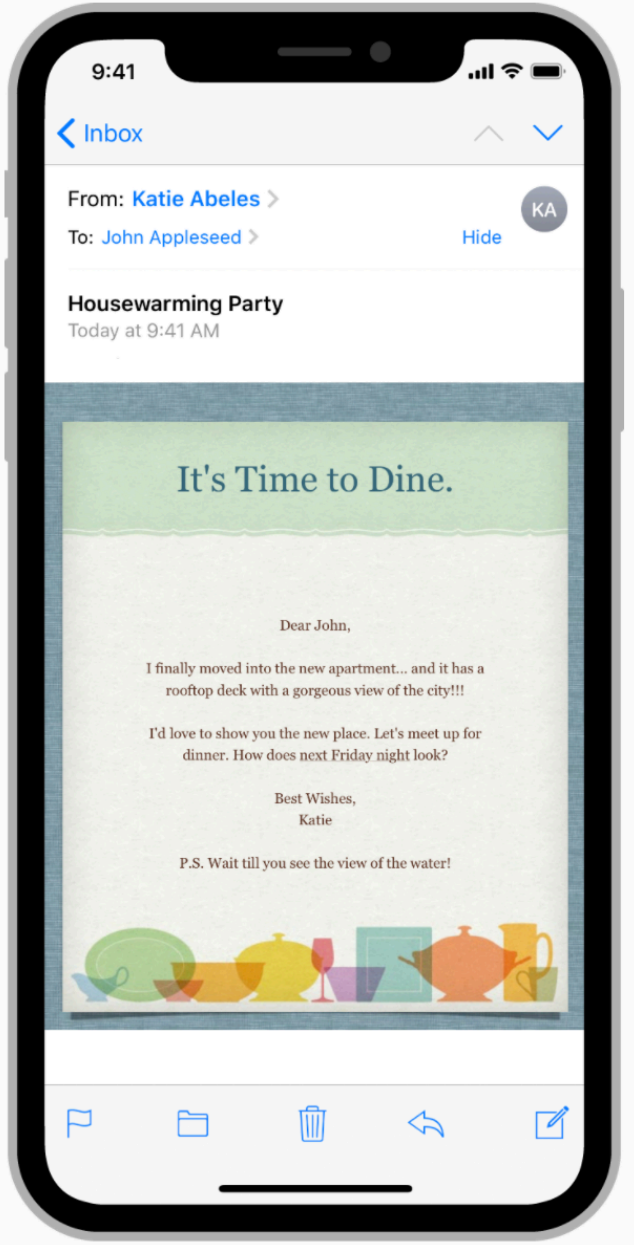
Text View

- 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

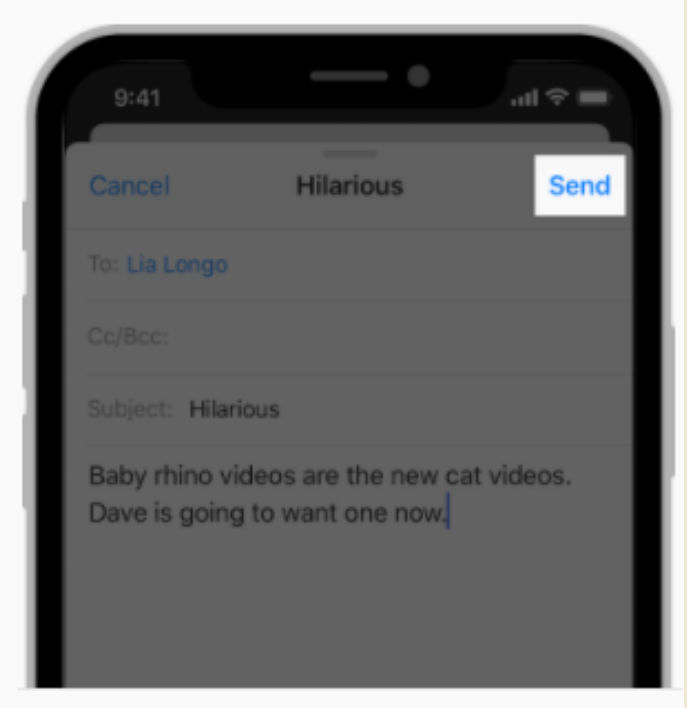


Controls

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

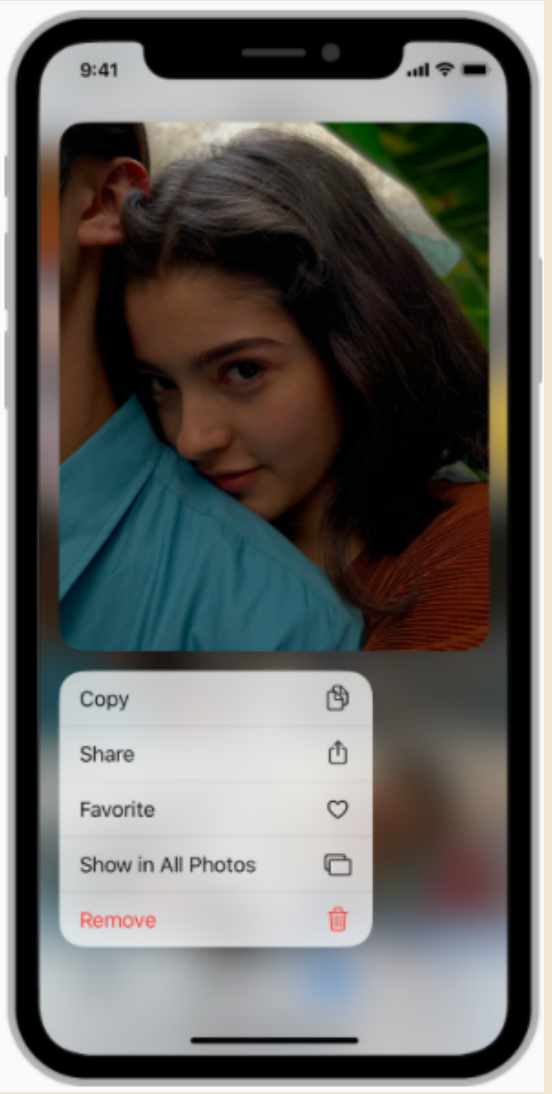
Buttons

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



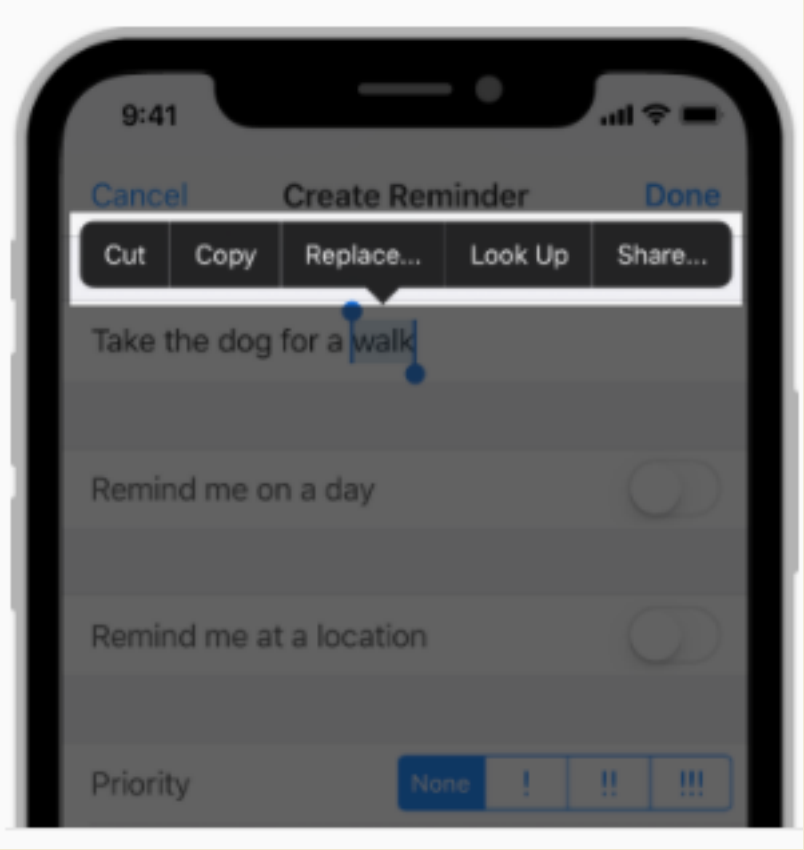
Context Menus

- 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

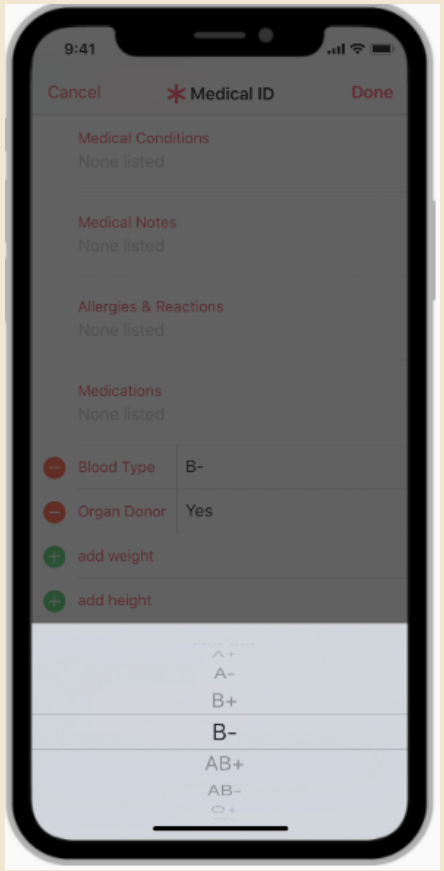
- A label describes an onscreen interface element or provides a short message
- Keep labels legible

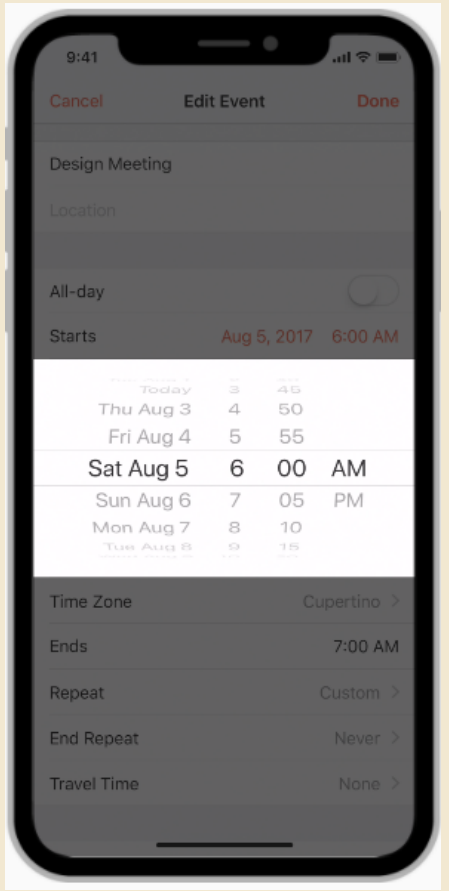
Page Controls

- 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

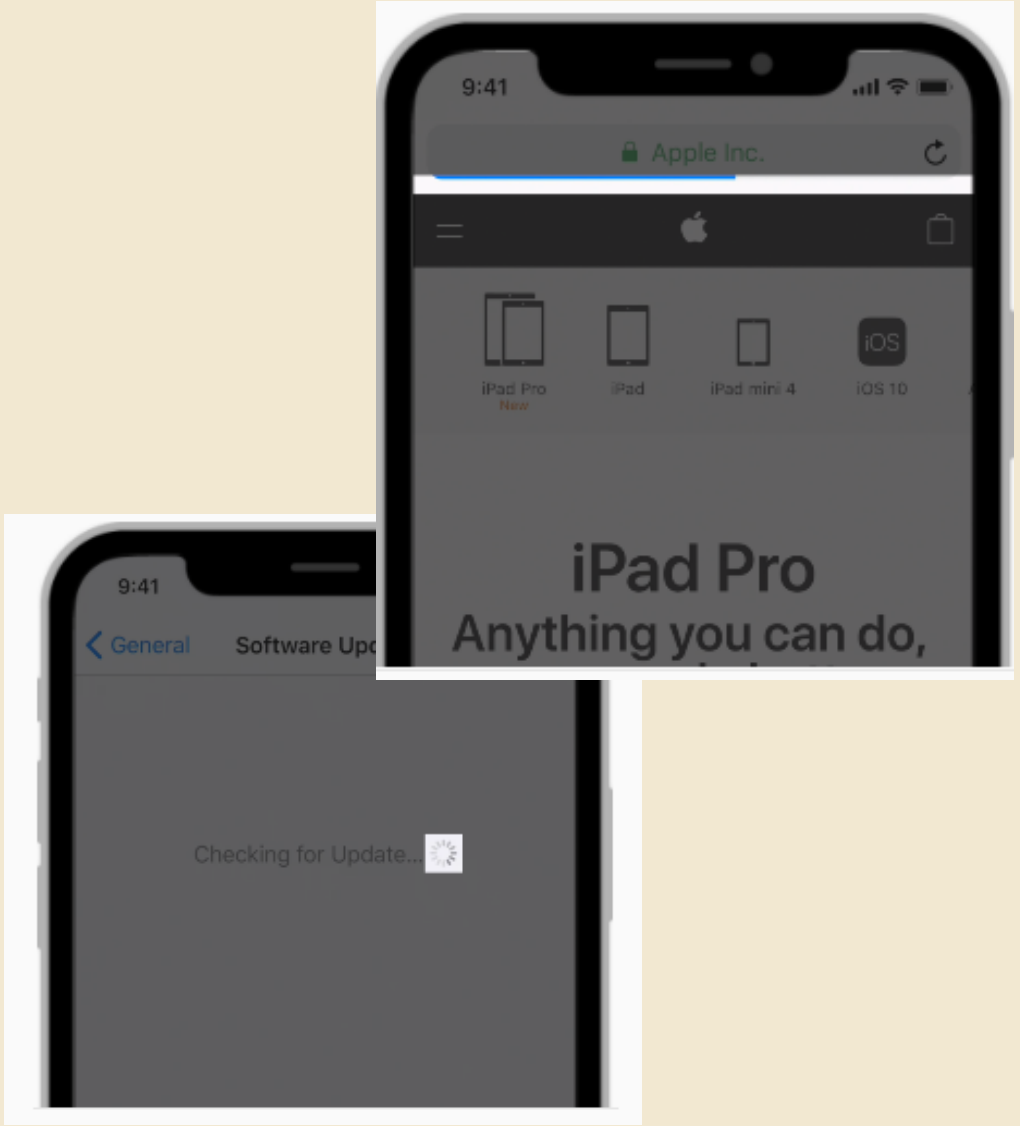
- Pickers tips
  - Use predictable and logically ordered values
  - Avoid switching screens to show a picker
  - Use a table instead of a picker for large value lists
- Date Pickers
  - Consider providing less granularity when specifying minutes





Progress Indicators

- Network Activity Indicators
  - Before iOS 12

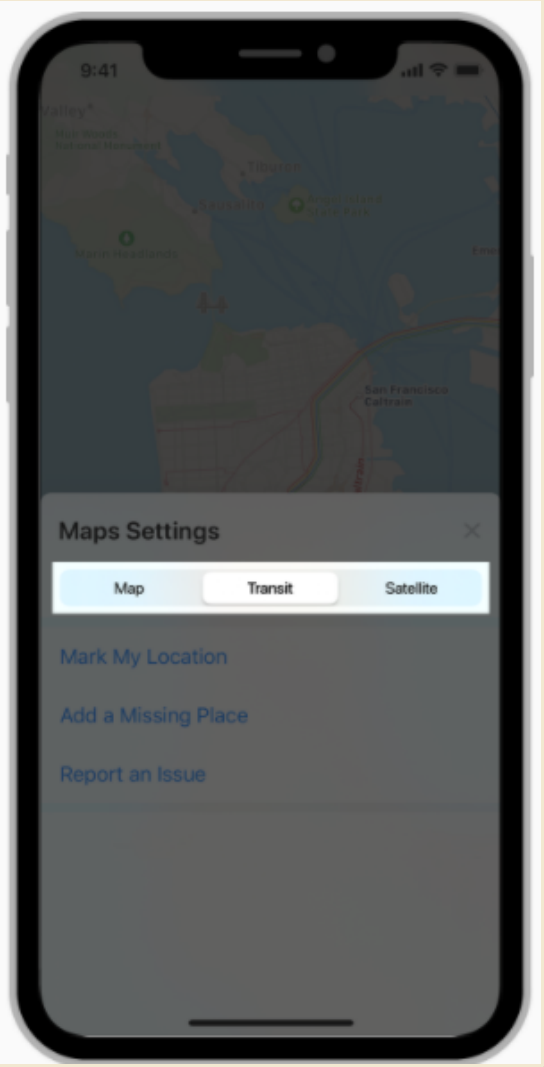


Refresh Content Controls

- 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.

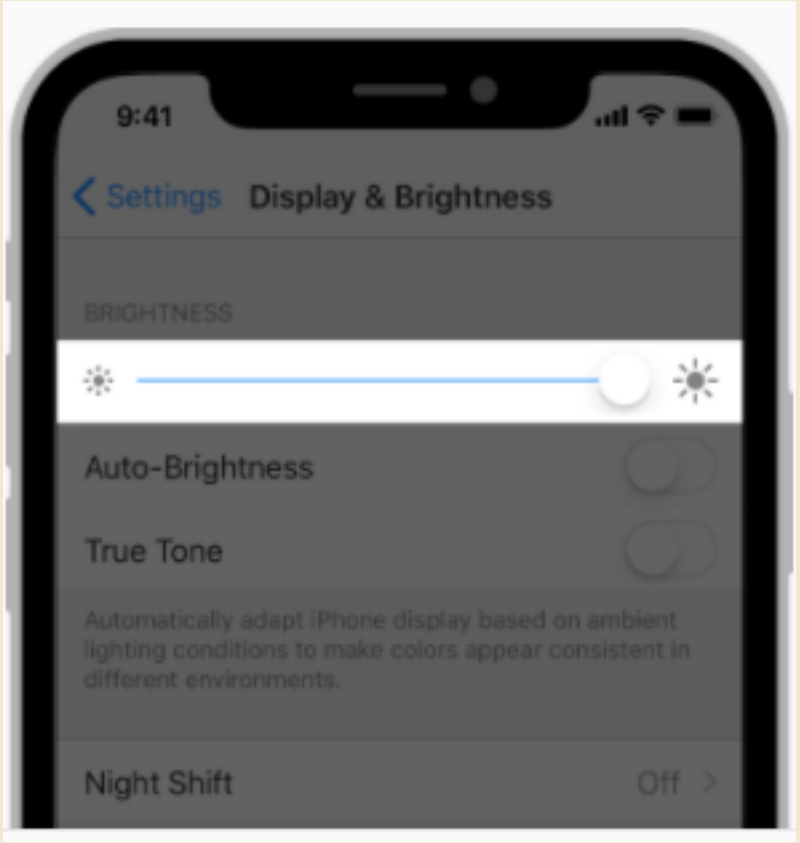
Segmented Controls

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



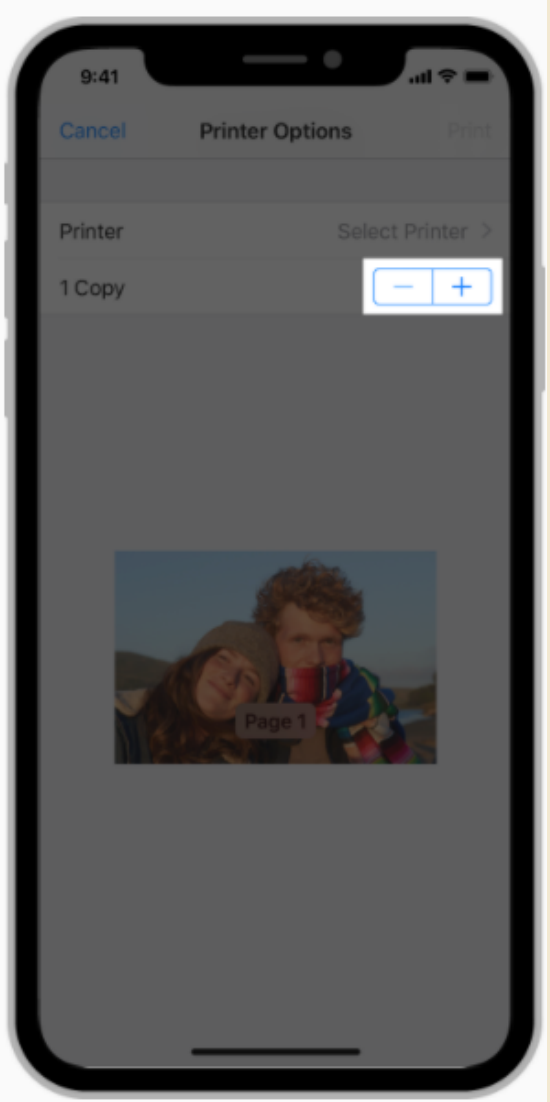
Sliders

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



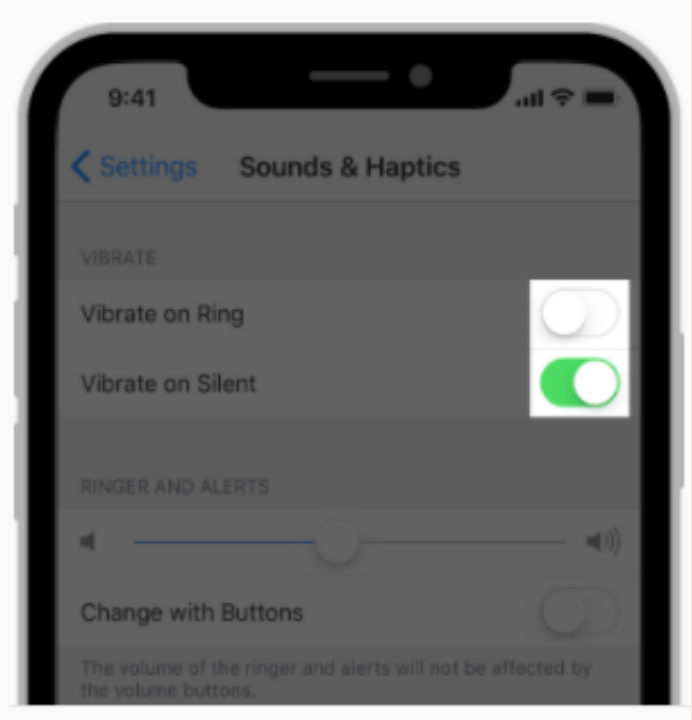
Steppers

- 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.

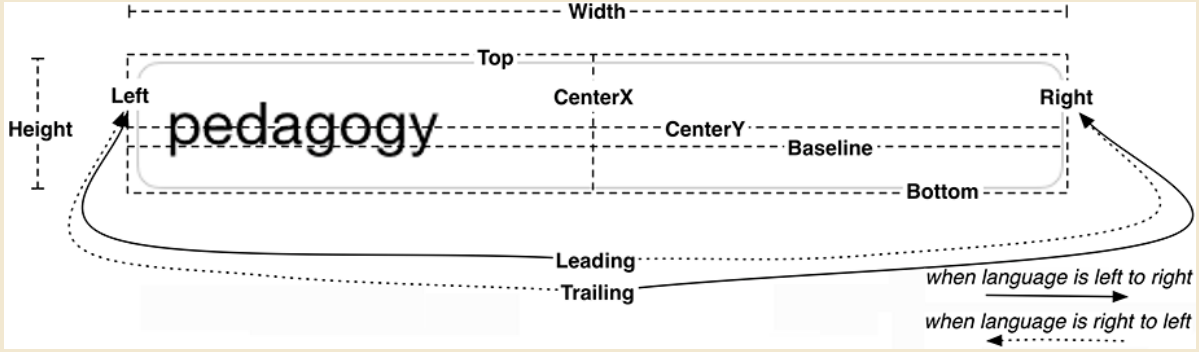


Text Fields

- 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

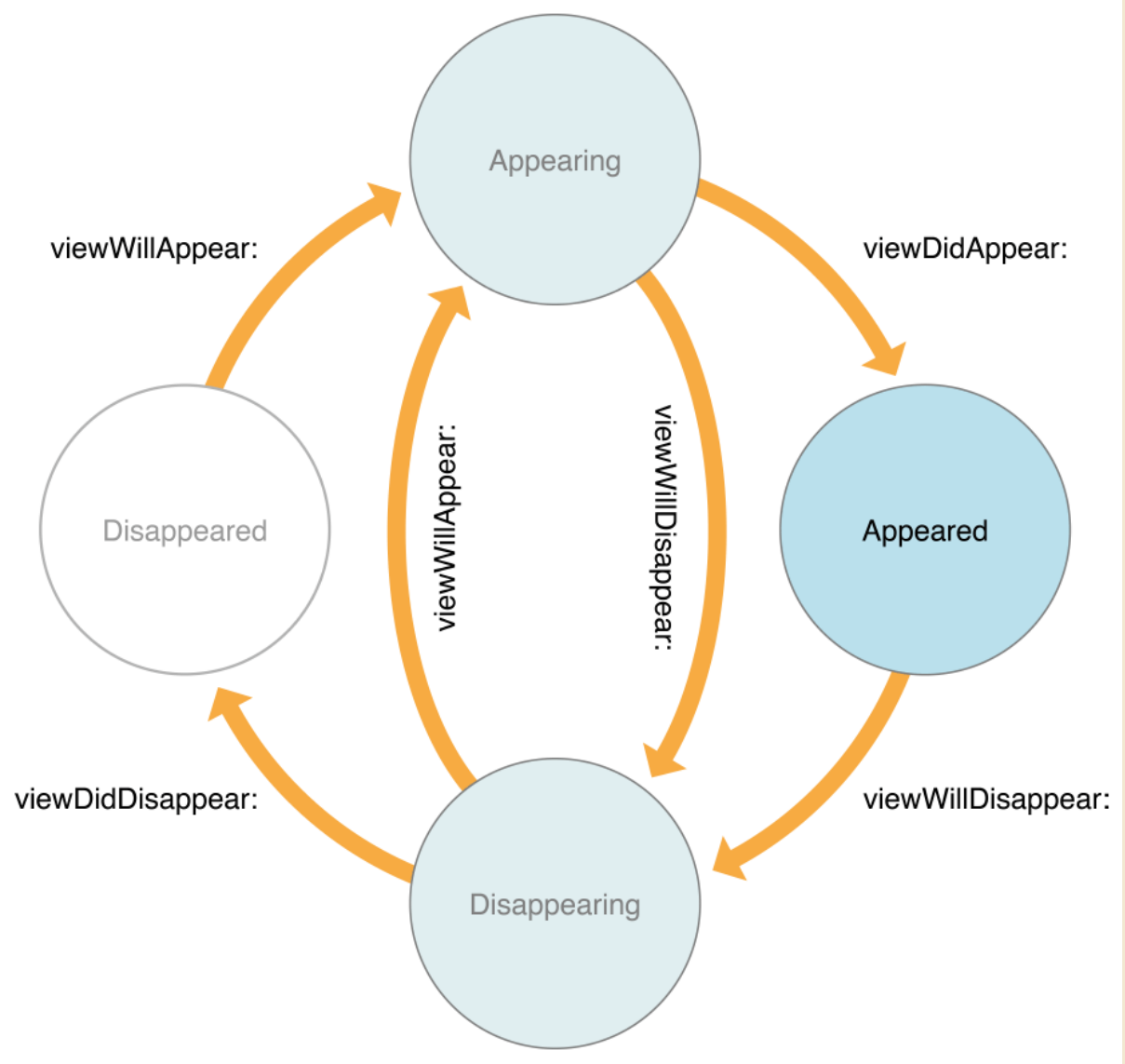
The Auto Layout System

- 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()
- viewWillAppear()
- 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
- Afer iOS 13
  - Can just use SwiftUI

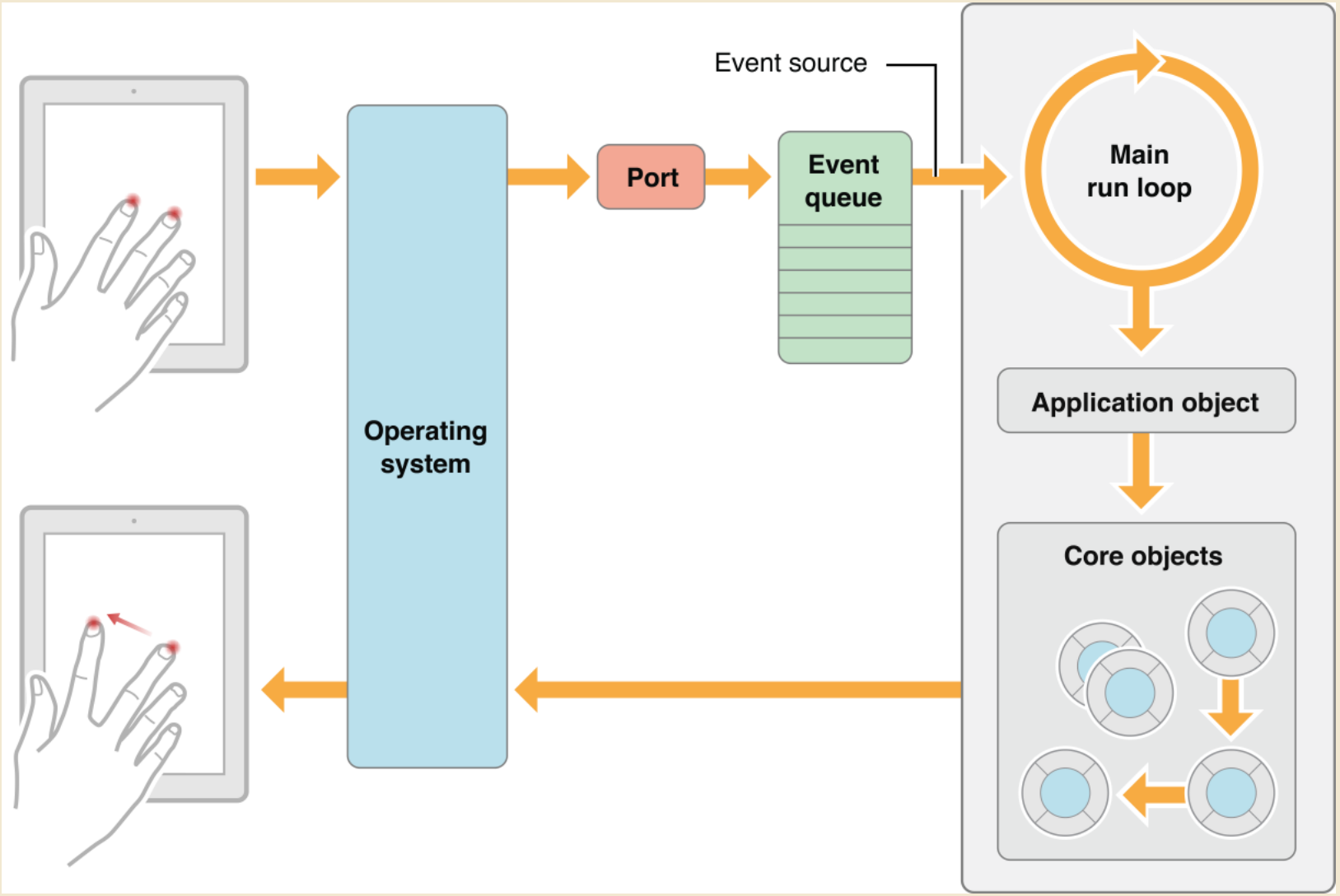
delegate

- 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

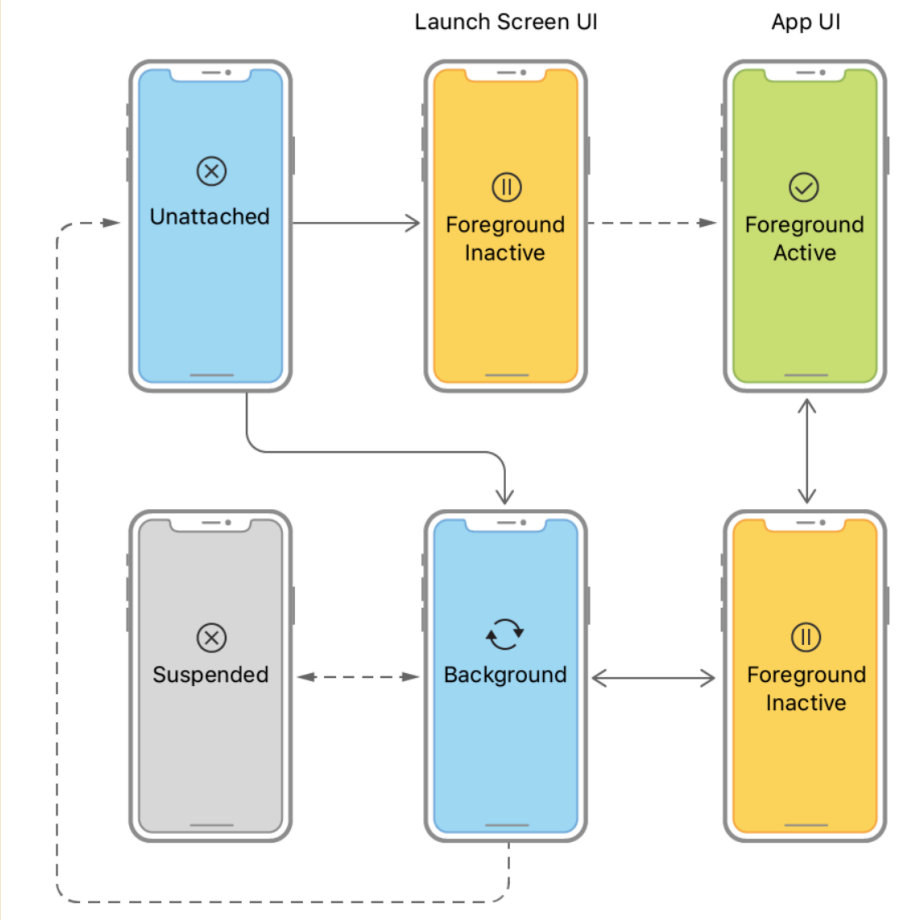
```
int main(int argc, char * argv[]) {
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass
([AppDelegate class]));
    }
}
```

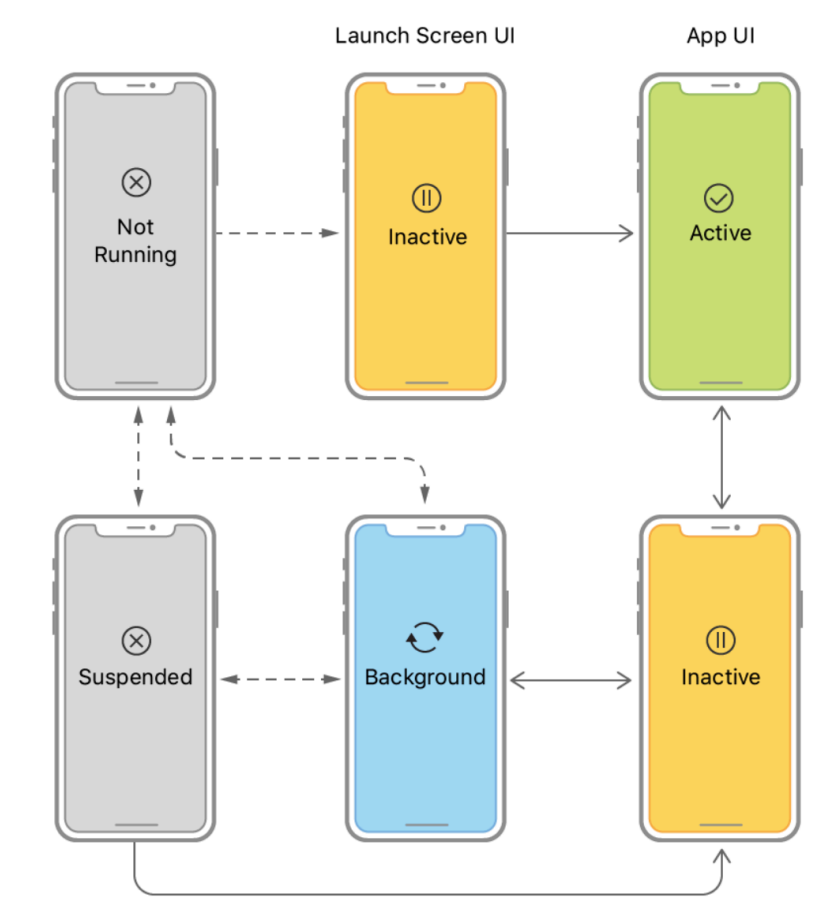
- @autoreleasepool是指使用ARC的意思
- main會在autoreleasepool中呼叫UIApplicationMain
- UIApplicationMain會為app產生兩個用重要的元件
  - UIApplication的物件, 稱為*applcation*
    - 控制app event loop和app的行為
  - AppDelegate的物件，稱為app delegate
    - 產生app window且提供狀態轉變時的回應(設計師可以自行控制的部分)



App Life Cycle

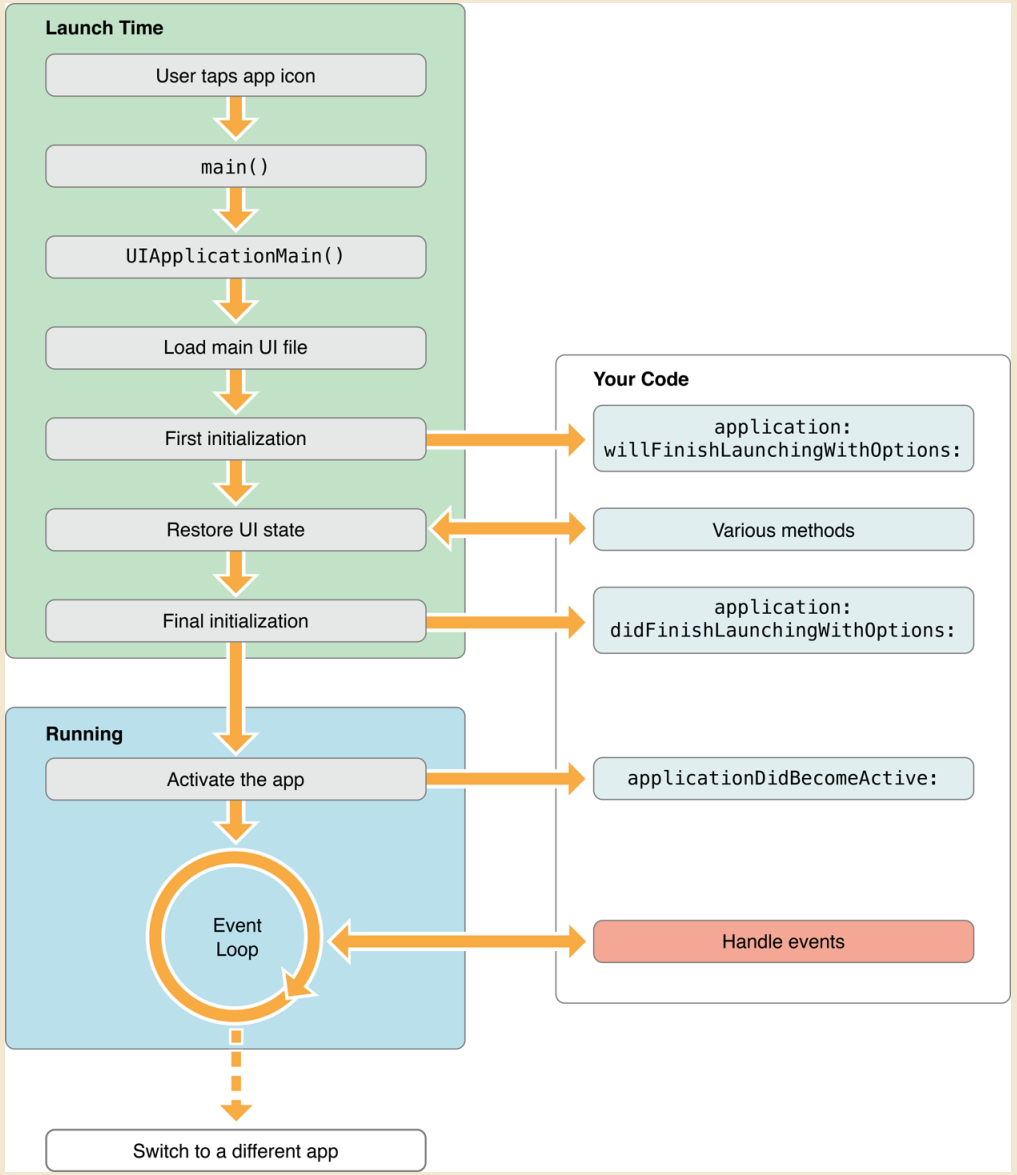
Respond to Scene-Based Life-Cycle Events (in iOS 13 and later)





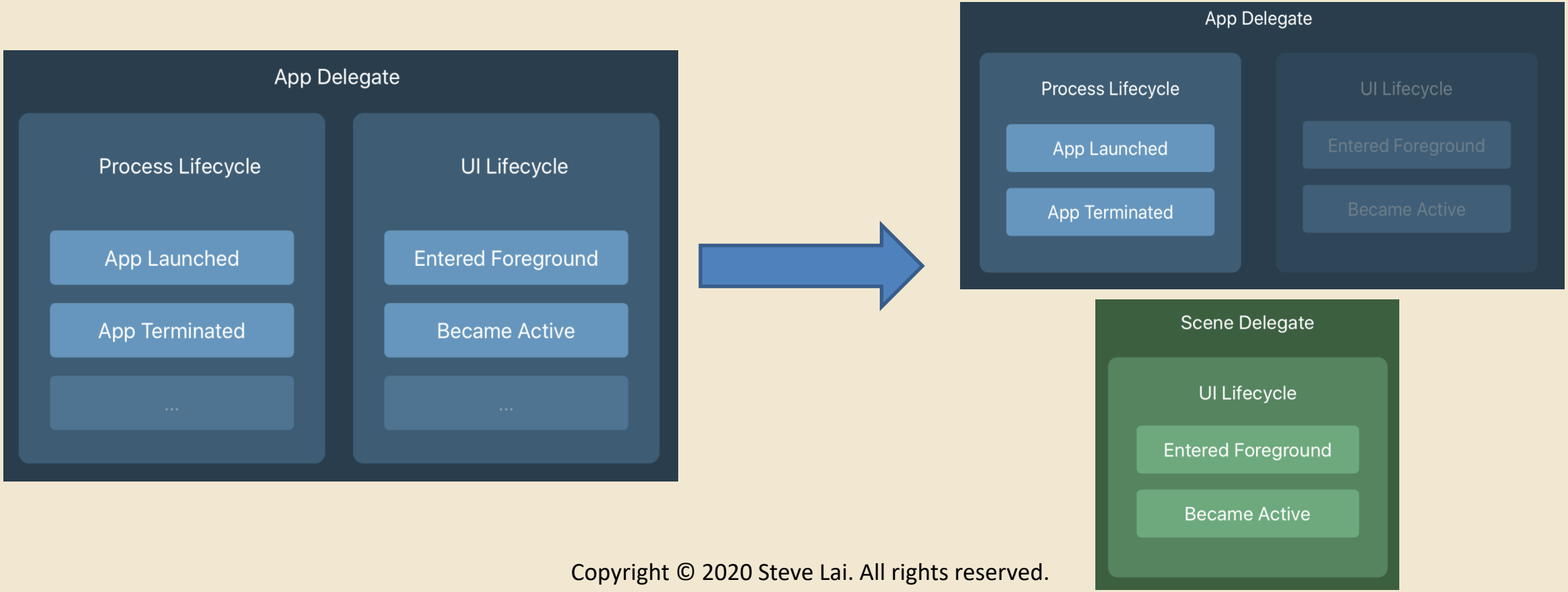
AppDelegate

- 在AppDelegate對應的Method中標明在狀態改變時要作的動作
- application:willFinishLaunchingWithOptions:—App一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

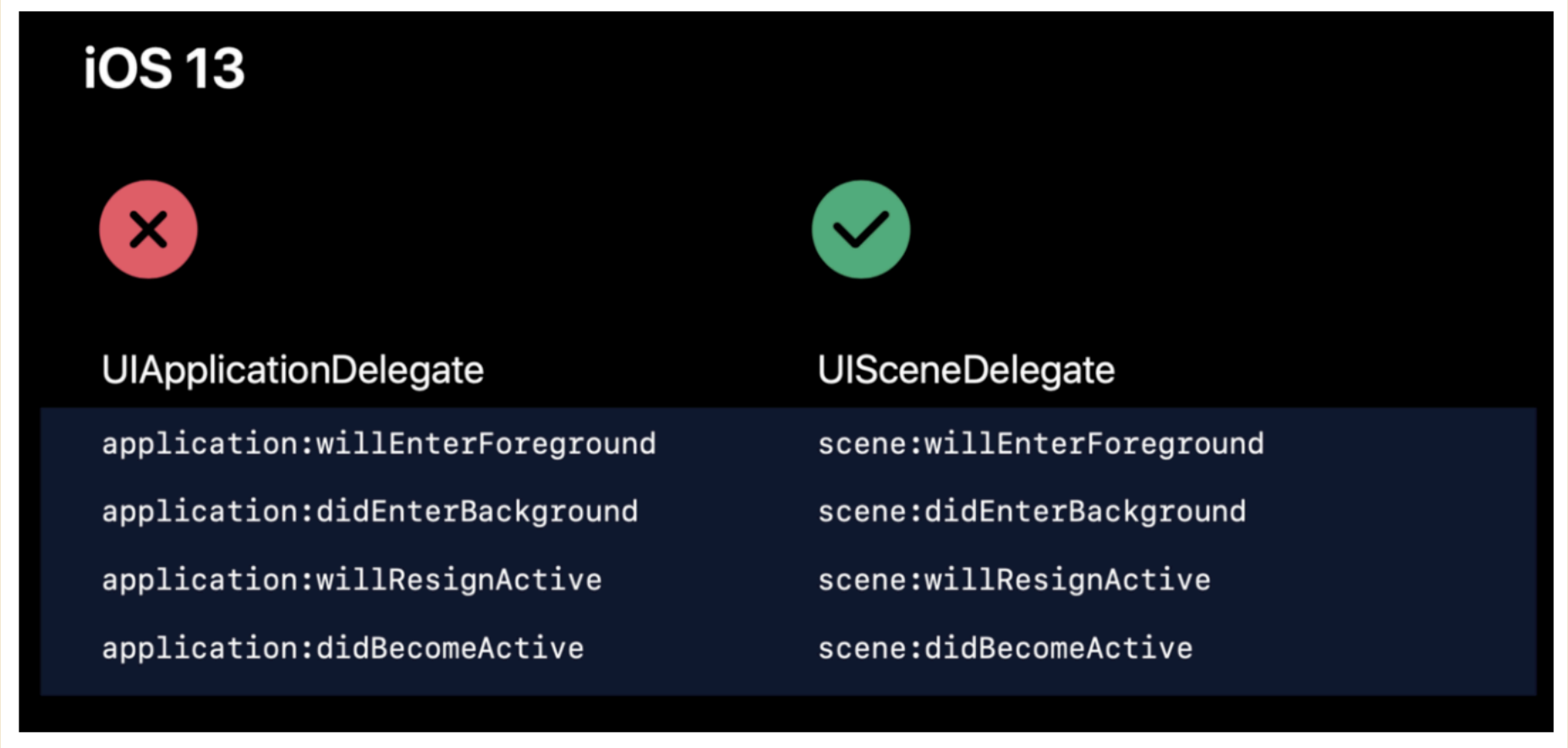


SceneDelegate

- 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

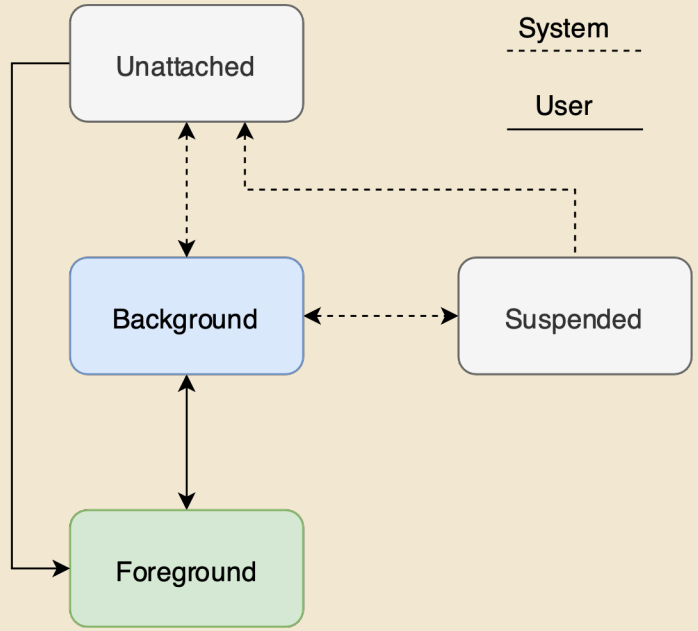


ScnenDelegate



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

- 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

ARC

- 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

Dr. Steve Lai

2020/10/23

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

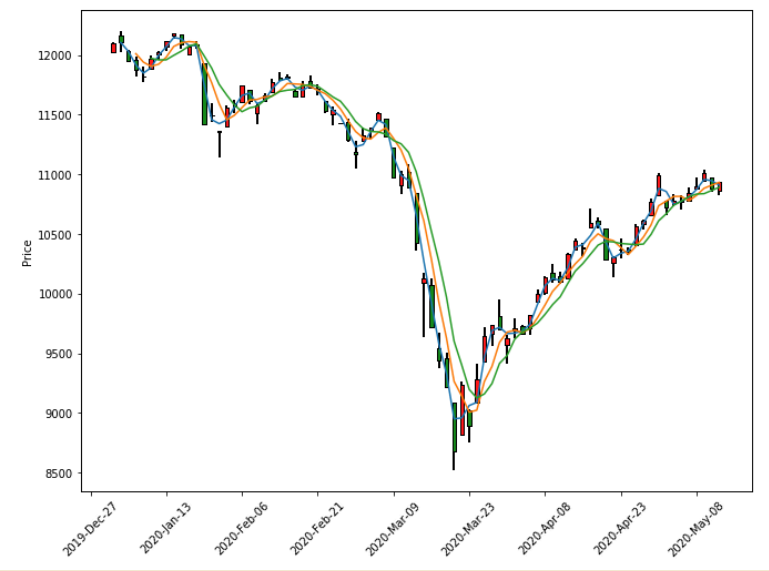
- Environment
  - Jupyter
  - Pycharm
- Python 3.X
  - PEP (Python Enhancement Proposal) 8 style guidelines
- Numpy
- Pandas
- Matplotlib
- Others
  - Scipy
  - Science kit learn
  - Deep learning framework

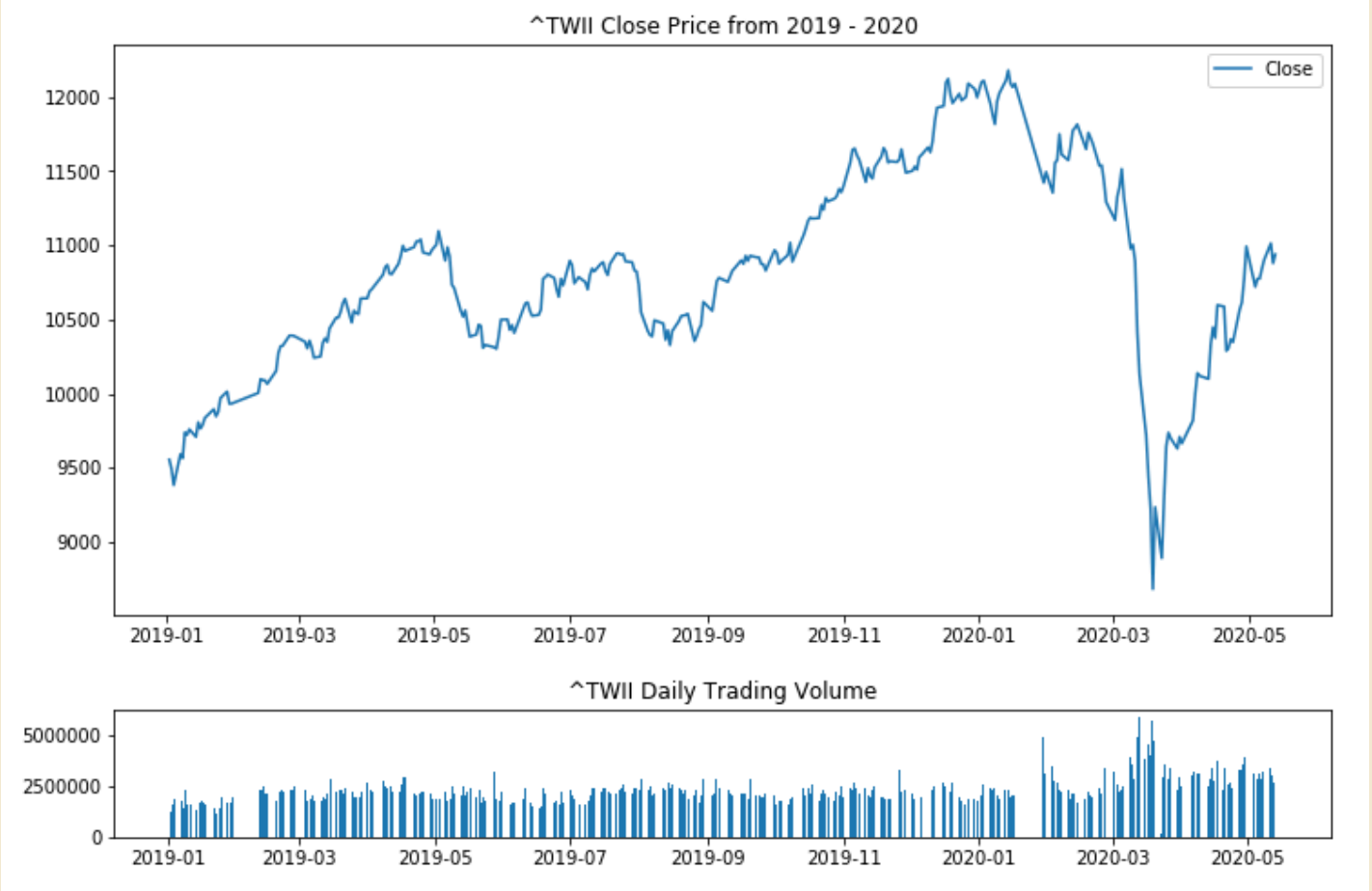
Data Analysis

- 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

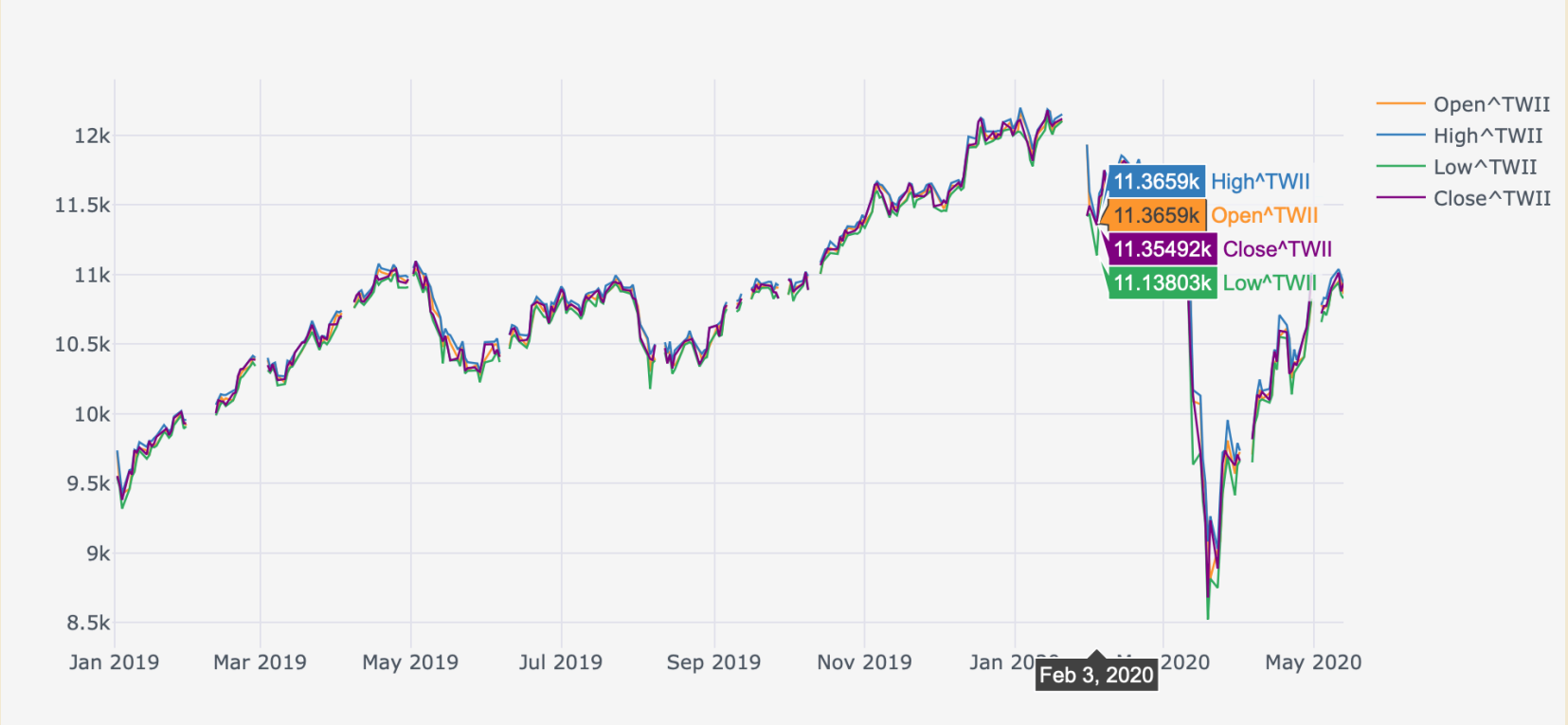
- TWSE
- TAIFEX
- Quandl
  - Need sign up an account
- pandas_datareader
  - yfinance





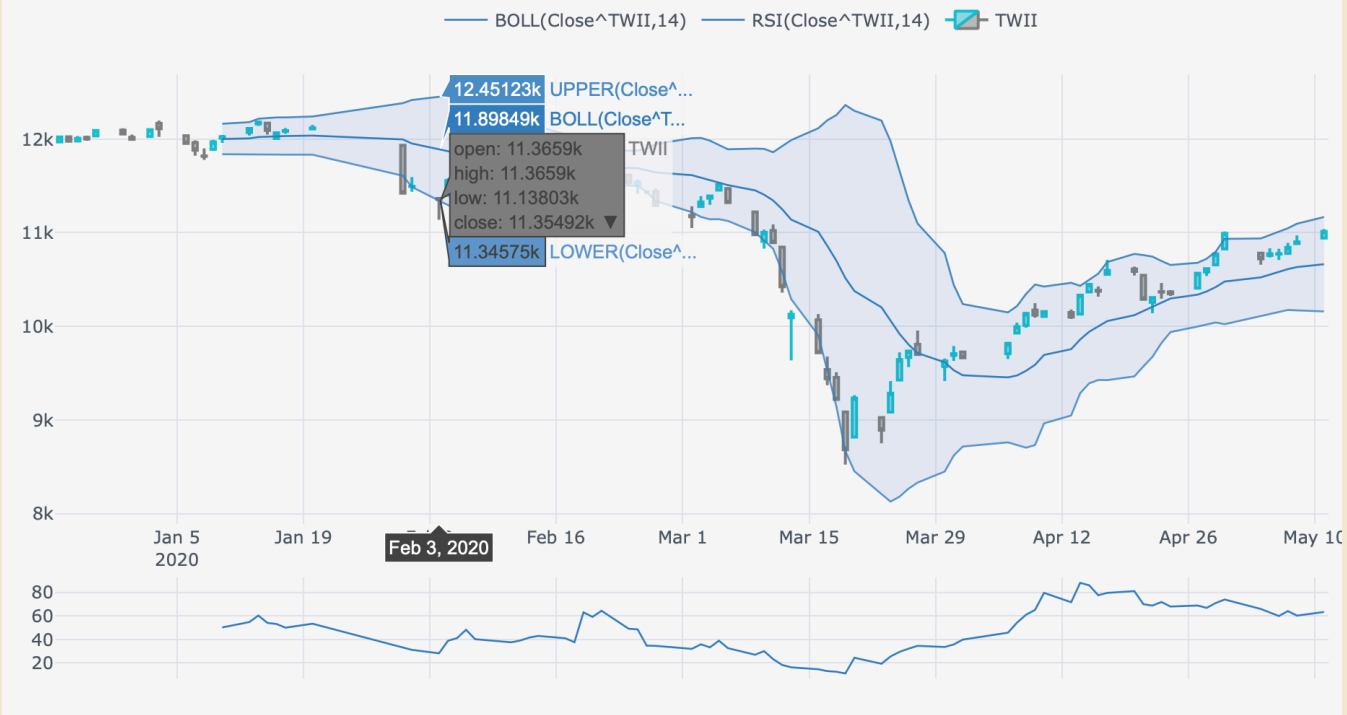
Data Visualization

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



Financial Graphics

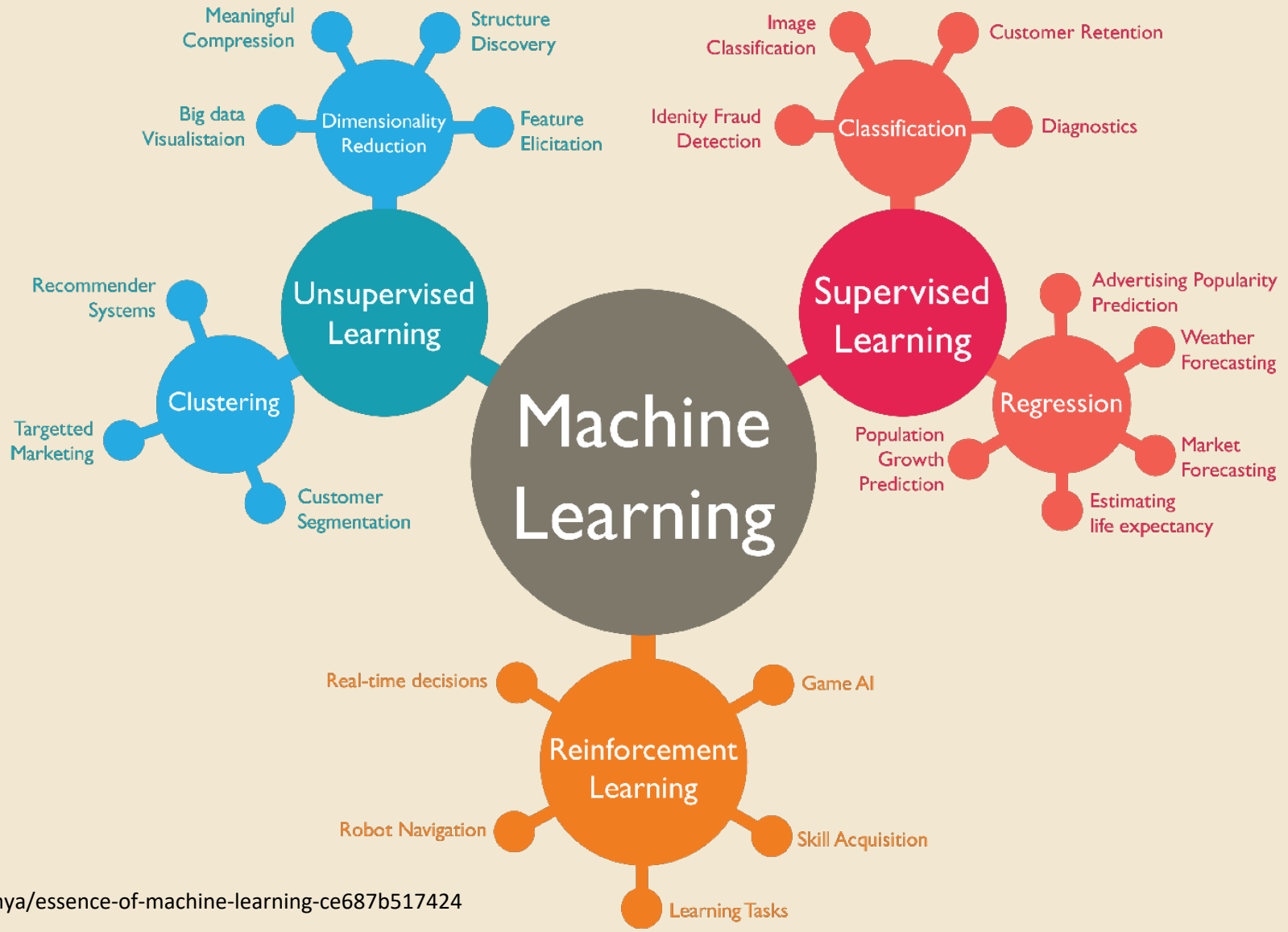
- Requirement packages
  - Cufflinks
  - chart_studio
  - mplfinance
- Graphics Types
  - Candlestick chart
  - Bollinger bands
  - Relative strength indicator
  - Moving average convergence divergence



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()

Machine Learning



https://medium.com/analytics-vidhya/essence-of-machine-learning-ce687b517424

7 Steps to Machine Learning

Google Cloud Platform
https://www.youtube.com/watch?v=nKW8Ndu7Mjw
The 7 steps of machine learning (AI Adventures)

https://medium.com/dataseries/7-steps-to-machine-learning-how-to-prepare-for-an-automated-future-78c7918cb35d

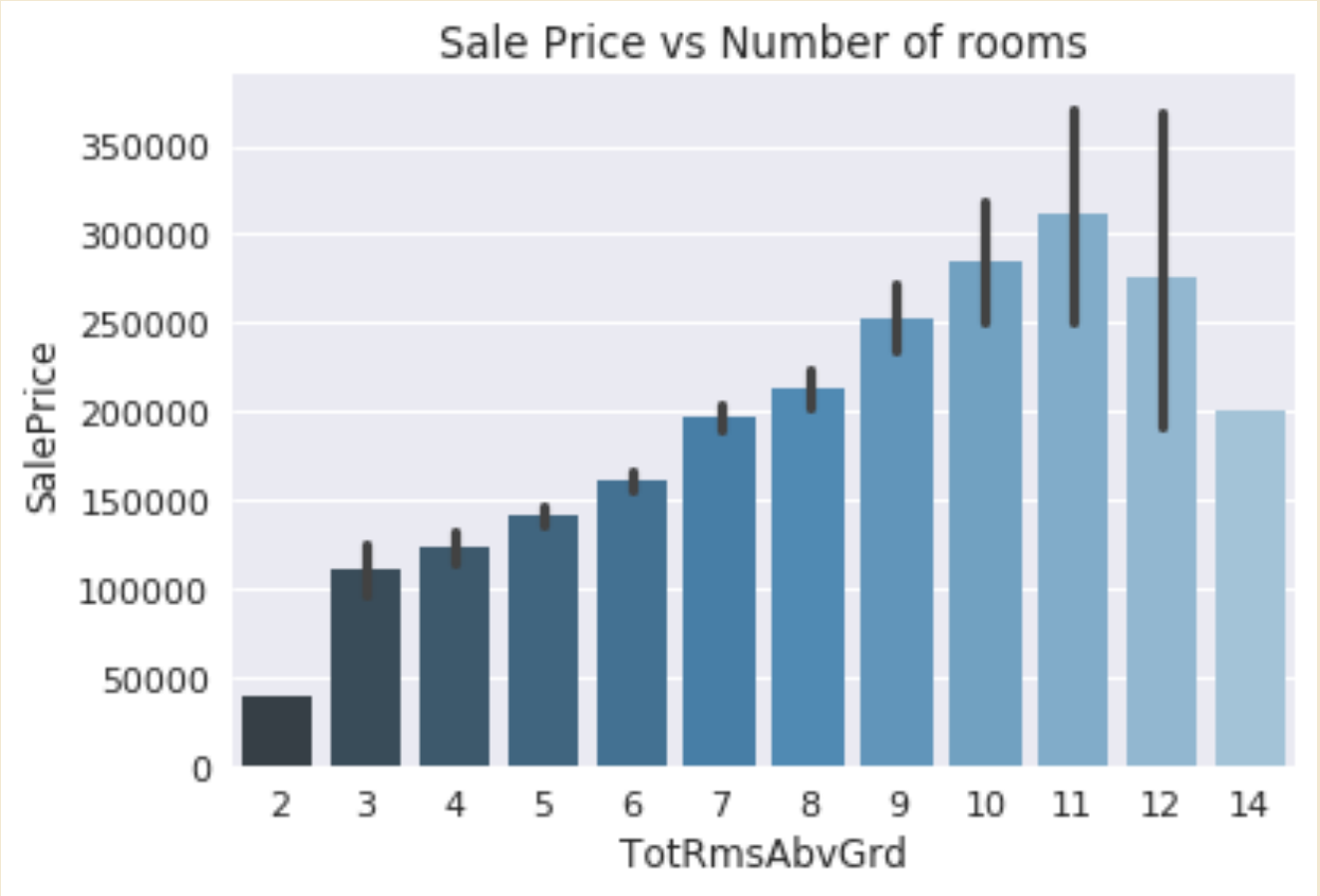
Minimizing Risks for Loan Investments

- Higher credit scores (more trustworthy and less risky) get lower interest rates for their loans
  - lower credit scores (less trustworthy and more risky) get higher rates.
- However, the loans with higher interest rates are more attractive because they provide higher return on investment (ROI)
  - They pose risks of being not returned at all.
- The machine learning model that could predict which of the high interest loans are more likely to be returned, would bring added value by minimizing the associated risks.

- https://www.kaggle.com/wordsforthewise/lending-club

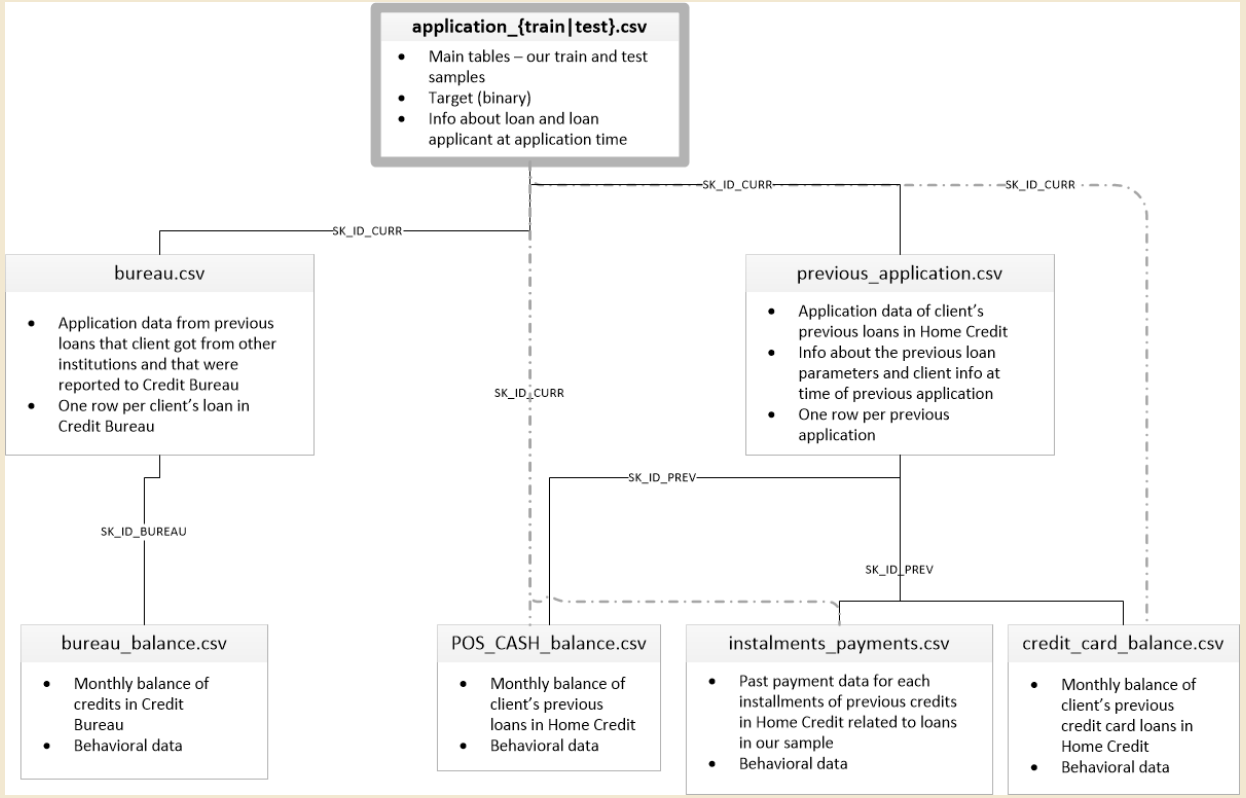
Case Study: House Prices

- Kaggle
  - House Prices: Advanced Regression Techniques
  - https://www.kaggle.com/c/house-prices-advanced-regression-techniques



Case Study: Home Credit Default Risk

- Kaggle
  - Home Credit Default Risk
  - https://www.kaggle.com/c/home-credit-default-risk



Code Reference

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