**IOS**

Contents

[1. Stack View 6](#_Toc132016448)

[2. Timer 11](#_Toc132016449)

[3. BMI Calcurlator 12](#_Toc132016450)

[3.1 Hàm format String: 12](#_Toc132016451)

[3.2 Tạo nhiều màn hình từ code 13](#_Toc132016452)

[3.3 Segues và Navigation for multiple Screen 15](#_Toc132016453)

[4 Build App Weather 18](#_Toc132016454)

[4.1 Dark Mode and working with vector assests 18](#_Toc132016455)

[4.2 Learn to Use UITextField 21](#_Toc132016456)

[4.3 Nói 1 chút về Delegate 25](#_Toc132016457)

[4.4 User URLSession for Networking 27](#_Toc132016458)

[4.5 JSON Decoding 28](#_Toc132016459)

[4.6 Dispatch Queue 32](#_Toc132016460)

[5. Thread, Queue, GCD 32](#_Toc132016461)

[5.1 Grand Central Dispatch(GCD) 32](#_Toc132016462)

[5.2 Serial Queues 33](#_Toc132016463)

[5.3 Concurrent Queue 33](#_Toc132016464)

[5.4 Synchronous and Asynchronous 33](#_Toc132016465)

[6. Flash Chat 34](#_Toc132016466)

[6.1 Navigation Controller Stack and Segues 34](#_Toc132016467)

[6.2 Table View Shell 36](#_Toc132016468)

[Bài 7. Local Data Persitance, Users Default, Core Data and Realm 37](#_Toc132016469)

[7.1 Add cell vào tableView 37](#_Toc132016470)

[7.3 Persitent Local Data Storage Using UserDefault 39](#_Toc132016471)

[7.3 Sử dụng file plist để lưu data iphone 40](#_Toc132016472)

[7.4 Ví dụ file plist 43](#_Toc132016473)

[7.5 CoreData 45](#_Toc132016474)

|  |  |  |
| --- | --- | --- |
| **Tên Bài** | **Phương thức** | **Tác Dụng** |
| **Bài 3: Tính kết quả BMI và chuyển kqBMI tới screen tiếp theo, nhấn back để trở ngược lại** |  | Format 1 String |
|  |  | Chuyển tới Screen có identifier là “gotoResult” |
|  |  | Trước khi navigate sang ViewController khác, hàm này sẽ được gọi |
|  |  | Trở về Screen trước |
| **Bài 4:** | Delegate from UITextFiledDelegate | Nhẫn Button Go trên keyboard thì sẽ vào hàm này |
| **Input là tên city nhập từ textField, request API, parse API và hiển thị kết quả** |  | Keyboard sẽ được ẩn sau khi nhấn **Go(với điều kiện method sau return true)** |
|  |  | Sau khi gọi method endEditing thì system sẽ gọi hàm này để kiểm tra việc endEditting hợp lệ chưa, trả về false thì sẽ ko ẩn keyboard và tiếp tục bắt user nhập |
|  | **Bốn hàm bên trên là đủ cho việc get input từ textfiled.** | Nếu Method bên trên trả về true thì system sẽ trigger và gọi tới hàm này. |
|  |  |  |
|  |  | Đăng kí Request tới url, sau khi request xong chạy func handle() ở background |
|  |  | Bắt đầu thực thi task |
|  |  | Cho phép struct này được Parse. |
|  |  | Khởi tạo 1 biến để ParseJson |
|  |  | Parse data vào biến decodeData.  weatherData: là data ta muốn Parse  WeatherData.self: là dạng data ta muốn Parse |
|  |  | 1 vài tính năng của navigationItem |
|  | Không có mô tả. | Trở về Controller đầu tiên |
| **Bài 6**  **Sử dụng table View làm app Chat** |  | Hàm này đăng kí 1 TableViewShell vào tableView |
|  | Không có mô tả. | Hàm này xem ở bài 6 |
| **Bài 7** |  | Giá trị return là số hàng được thêm vào TableView |
|  | 2 hàm trên cơ bản giống tác dụng hai hàm của ảnh trên | Được sử dụng để setup data vào từng hang của tableView |
|  |  | Hàm này sẽ được gọi khi ta tích vào bất kì cell nào của tableView. Biến indexPath chứa vị trí cell được bấm(indexPath,row là get được hang nào được bấm) |
|  |  | - Khởi tạo alert  -Add textFiled vào trong Alert  -Khởi tạo 1 action  -Add action vào alert và thực thi nó, nó sẽ xuất hiện 1 button có title “Add item”, ta tích vào button đó sẽ trigger action.  - Ta có thể add nhiều action |
|  |  |  |
|  |  | - Lấy đường dẫn trỏ đến thư mục document  - Nối thêm file vào hello.plist vào đường dẫn đó, và trả về đường dẫn được nối  - Tạo file heelo.plist trong folder document  - Read Data trong file hello.plist  - Write data vào file |
|  |  |  |
|  |  |  |
|  |  |  |

# Stack View

Rule Implicit trong Constrains:

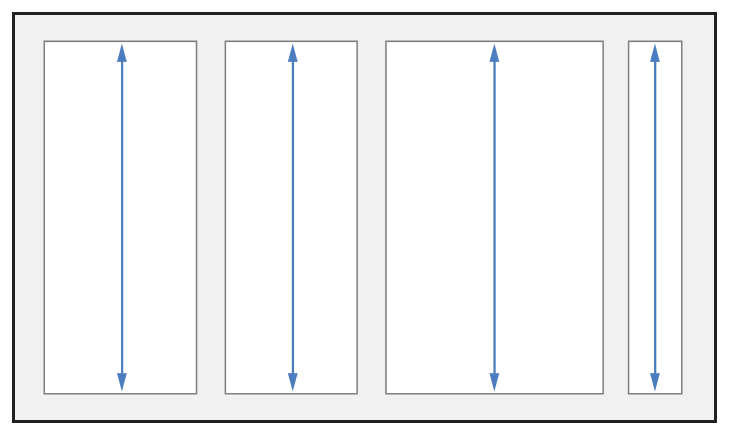
1. Safe Area và SuperView là 2 view đã được Constrain sẵn.
2. Khi ta mới tạo ra 1 subview như imgae, stackview, button, label,….. ta kéo giãn, mở rộng kích thước vô tư và kích thước và vị trí của nó sẽ mãi như vậy( Với điều kiện chưa constrain gì cả)
3. **Nếu ta không sử dụng constrain cho imgae hay bất cứ thứ gì nào thì sẽ không phải constrain cho bất cứ thứ gì trong view đó** 😊))
4. Nếu ta đã constrain cho 1 thứ(Ở đây Vd là image), mà imgae đấy được constrain với 1 ảnh hay labe, thì ta cũng phải constrain cho image hay label đó. Các view khác không liên quan không cần constrain
5. Nếu ta sử dụng stack view, thì lúc ta mới stack các vật lại với nhau, thì đó là do hệ thống làm, ta không cần constrain cho nó
6. Aligment sẽ làm các stack được nằm theo hướng ta chọn theo cái khung của stack view đó, ta cần mở rộng(constrain khung to ra) để nhìn rõ điều đó.
7. Distribution là cách ta phân phối kích cỡ, khoảng cách của các stack trong view đó.

**Deeply understand about stackview: Ta biết là khi mới tạo ra thì nó đã được hệ thống cung cấp 1 size cố định sẵn, ta có thể kéo ra kéo vào để change size đó, hoặc constrain 4 đầu để thay đổi size.** Vì nó có size nên sẽ có ***chiều rộng và chiều dài. Alignemnt và distribution là để chỉnh subview theo chiều rộng và chiều dài đó.***

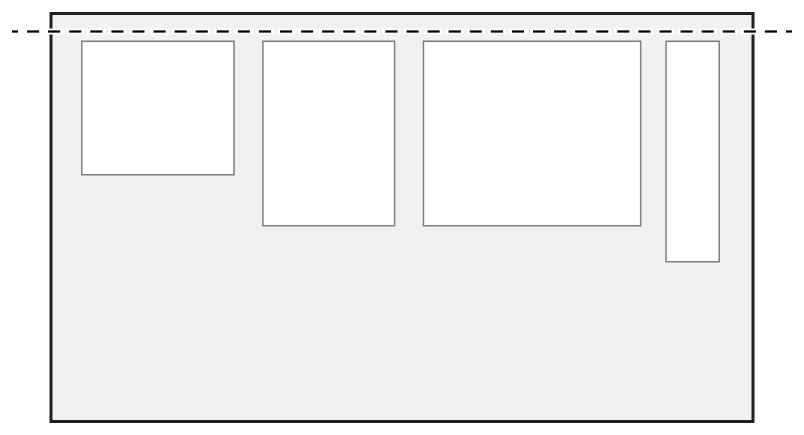
Các thuộc tính của Stack view:

* **Alignment**: Định nghĩa việc căn chỉnh của các View được sắp xếp vuông góc với trục của stack view. Nó có các giá trị như sau:

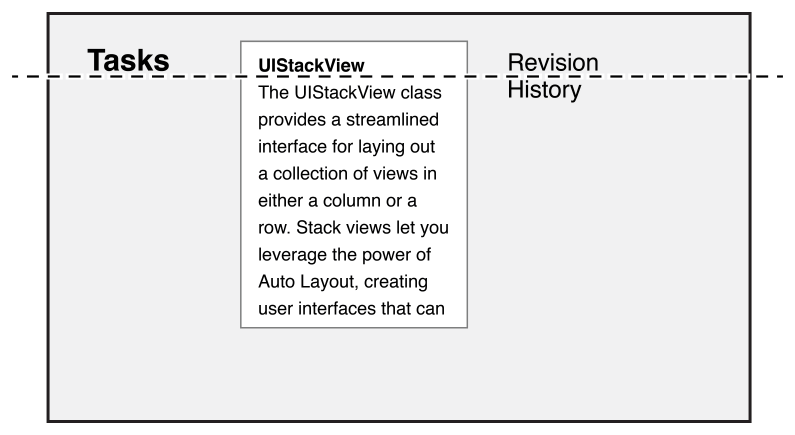
UIStackView.Alignment.**fill**: Làm cho các View bên trong stack view lấp đầy khoảng trống có sẵn trong stack view.(**Kiểu lấp đầy theo chiều rộng**, áp dụng cho những cái gì chưa bị constrain bên trong)



UIStackView.Alignment.**top**: Dùng cho horizontal stack view, nó căn chỉnh cạnh trên của các View trong stack view theo cạnh trên của của nó. .(kiểu bị bay hết lên trên mà ko change size, áp dụng cho những cái gì chưa bị constrain bên trong). Tương tự cho leading và bottom và trailing

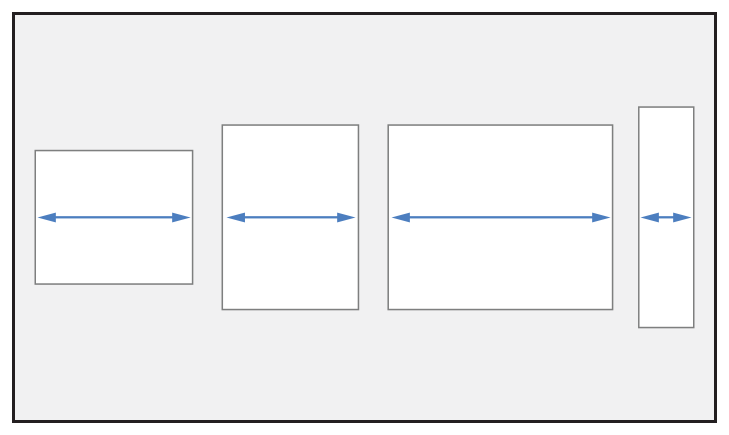


UIStackView.Alignment.**firstBaseline**: Dùng trong horizontal stack view. Nó căn chỉnh các view trong stack view dựa trên đường baseline đầu tiên của chúng.

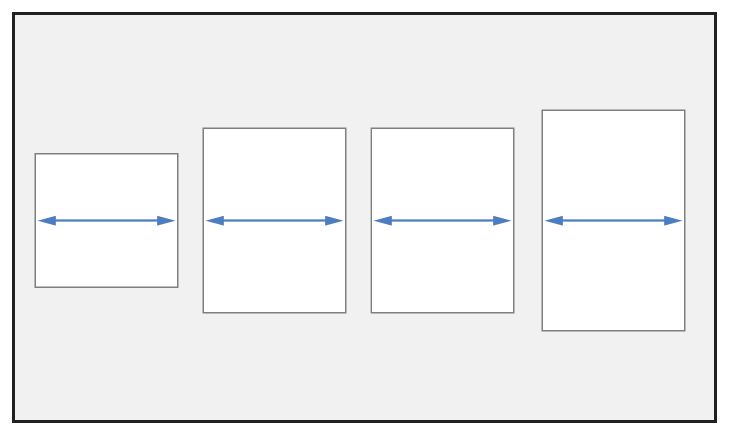


* Distribution: là thuộc tính xác định bố cục của các view được sắp xếp theo chiều ngang của stack view.

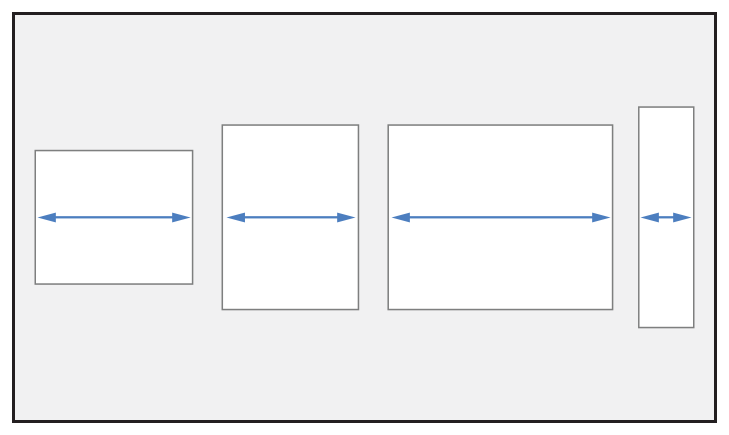
UIStackView.Distribution.**fill**: Nó giúp các view trong stack view lấp đầy không gian có sẵn trong stack view. Khi các view không xếp vừa stack view, nó sẽ thu nhỏ các view dựa trên độ ưu tiên của kháng nén của chúng (Compression resistance priority). Nếu các views được sắp xếp không lấp đầy được stack view, nó sẽ kéo dài các view theo hugging priority của chúng.



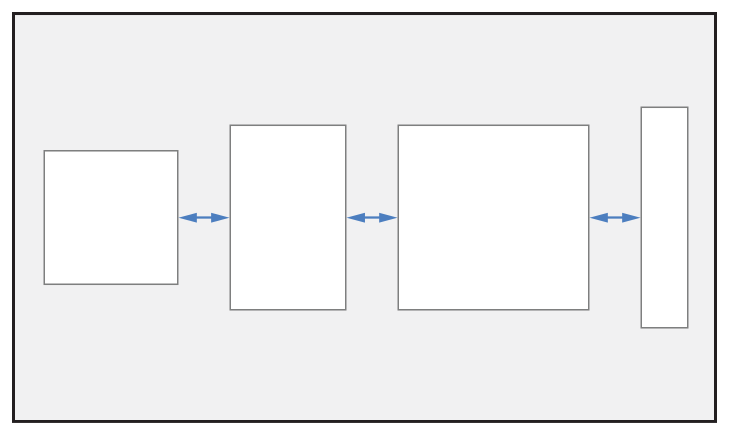
UIStackView.Distribution.**fillEqually**: Thay đổi kích thước các view bên trong stack view sao cho tất cả các view có cùng kích thước theo chiều của nó và lấp đầy không gian có sẵn theo chiều của stack view. Ví dụ: Horizontal stack view thì các view sẽ bằng nhau về chiều rộng, vertical stack view thì bằng nhau về chiều cao.



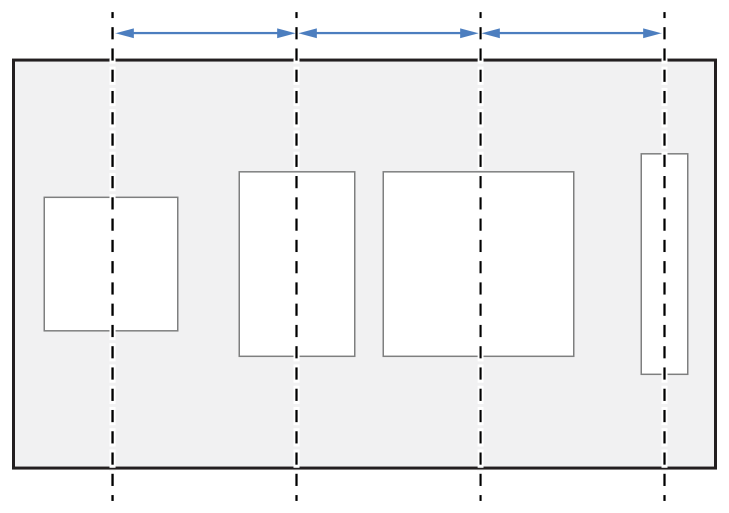
UIStackView.Distribution.**fillProportionally**: Thay đổi kích thước của các view trong stack view sao cho chúng lấp đầy các không gian có sẵn theo trục của stack view. Các view được thay đổi kích thước theo tỉ lệ dựa trên kích thước nội dung bên trong của chúng dọc theo chiều của stack view.



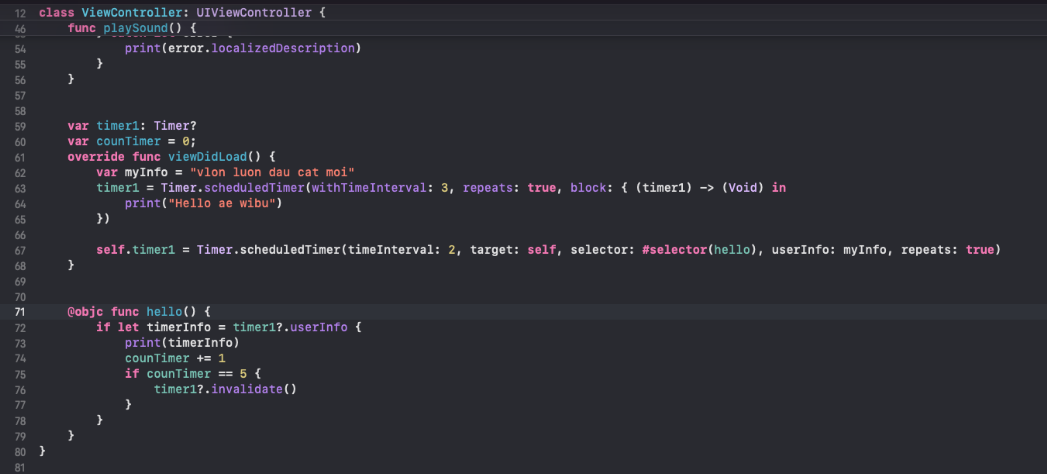
UIStackView.Distribution.**equalSpacing**: Các view trong stack view sẽ được sắp xếp sao cho khoảng cách giữa chúng bằng nhau. Nếu các view được sắp xếp không vừa stack view nó sẽ thu nhỏ các View dưa trên độ ưu tiên kháng nén của chúng(Compression resistance priority).



UIStackView.Distribution.**equalCentering**: Làm cho các view strong stack view được sắp xếp sao cho chúng có khoảng cách từ trung tâm bằng nhau theo chiều của stack view. Trong khi duy trì thuộc tính spacing(khoảng cách) giữa các view. Nếu các view được sắp xếp không vừa stack view, nó sẽ thu hẹp khoảng cách cho đến khi đạt khoảng cách tối thiểu được xác định bới thuộc tính khoảng cách của nó. Nếu các view vẫn không phù hợp, stack view sẽ thu nhỏ các view bên trong nó dựa trên thuộc tính Compression resistance priority.

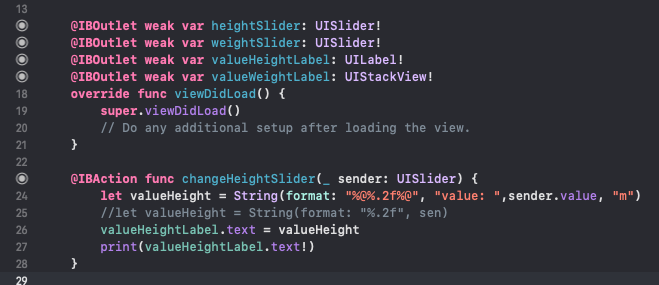


# 2. Timer

**Ta chạy 2 timer với chu kì lần lượt là 2 và 3s và đều lặp lại, timer1 đầu là sử dụng closure, timer1 thứ 2 là truyền function vào. Khi timer1 lần 2 chạy 5 lần thì sẽ dừng lại do chạy method invalidate.**

# 3. BMI Calcurlator

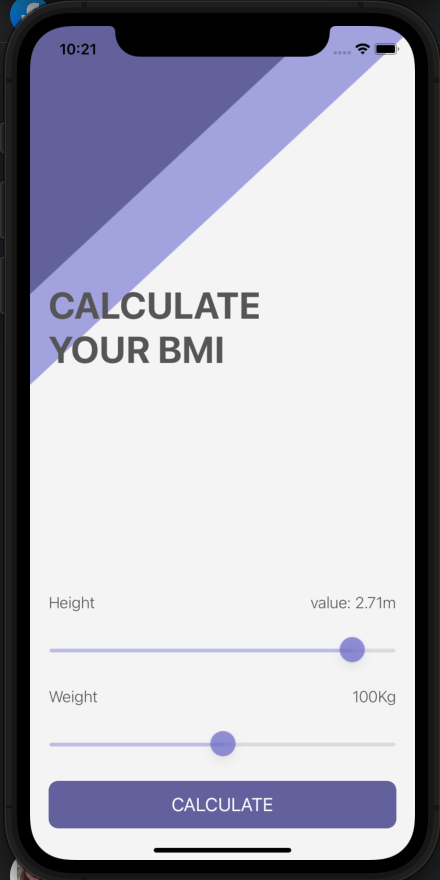
## 3.1 Hàm format String:



%@ : Sẽ lưu cái String vào dạng format

%.2f : Sẽ lưu số float dưới dạng chỉ có 2 số đăng sau dấu phẩy

Output khi ta kéo slider:



## 3.2 Tạo nhiều màn hình từ code

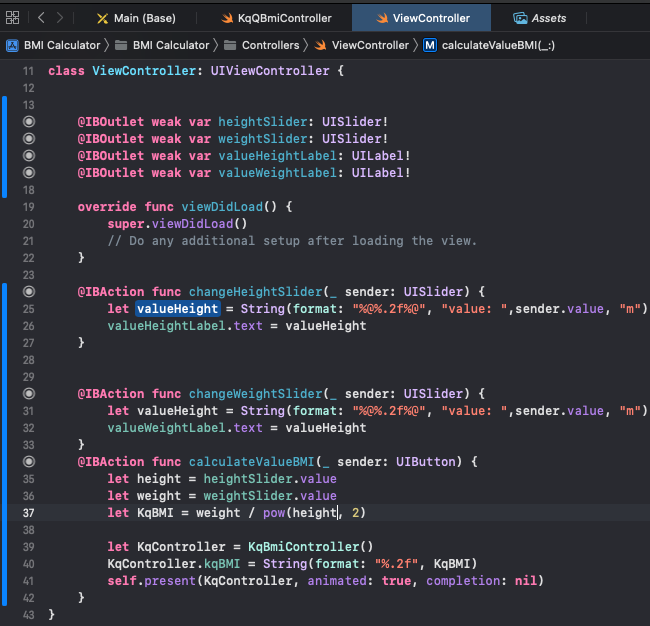
B1: Tạo 1 file SecondController.swift trong folder Controllers

Ở trong file SecondController.swift, ta them các thuộc tính cho view như lable, vịt trí của label, màu background



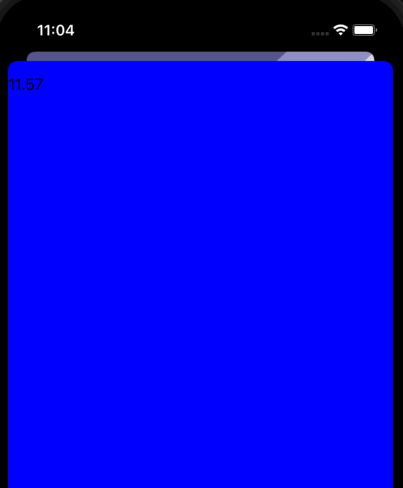
B2. Ở trong file thứ 2, ta khởi tạo object của class SecondController

Let kqController = KqBmiController()



Lúc này để chuyển sang màn hình thứ 2 ta sử dụng self.present(), và trỏ tới object controller ta muốn:

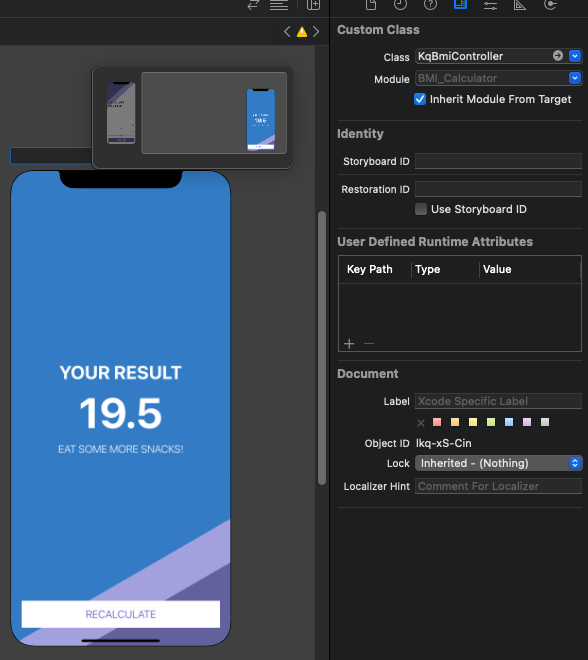
Kết quả:



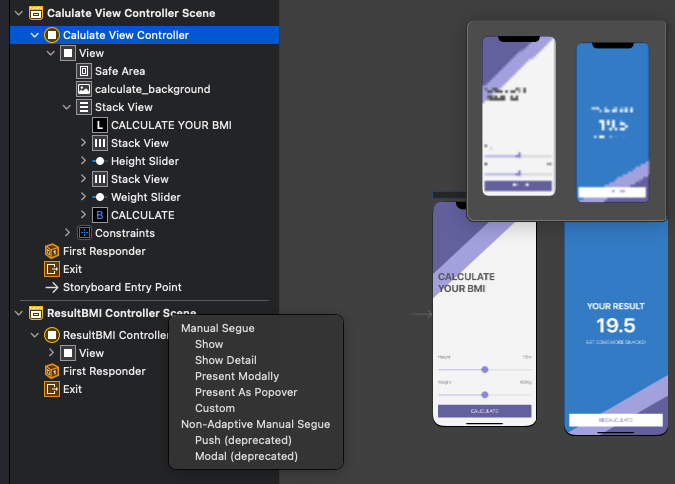
## 3.3 Segues và Navigation for multiple Screen

B1: Tích tạo file mới: Cocoa Touch File trong folder Controllers, ta đặt tên file là ResultBMIComtroller.swift

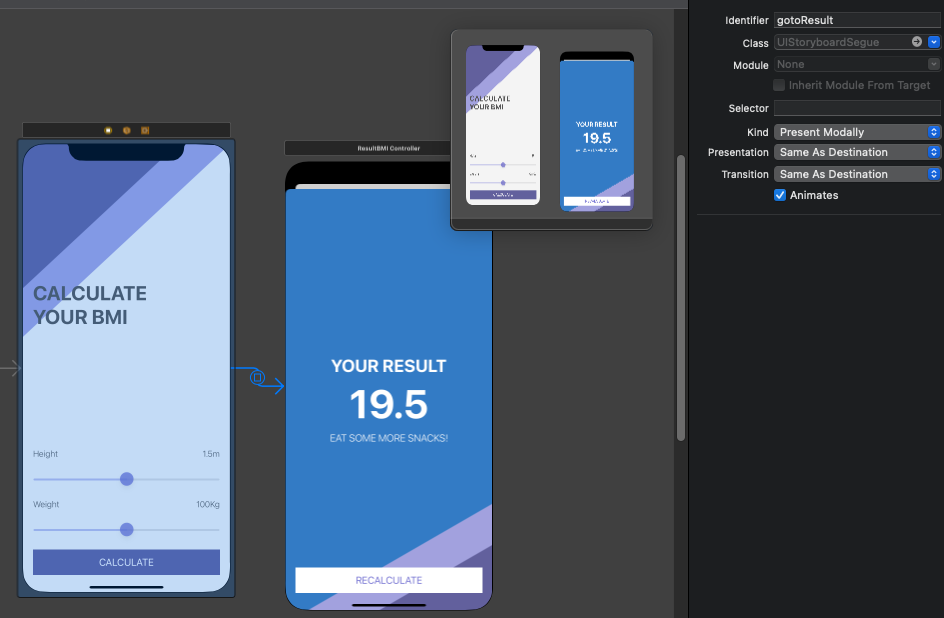
B2: Chuyển sang main.storeyBoard, chọn màn hình ta muốn link tới class đó, đổi tên Class trùng với tên class ta muốn link ở trong Custom Class. Ở đây là ResultBMIComtroller



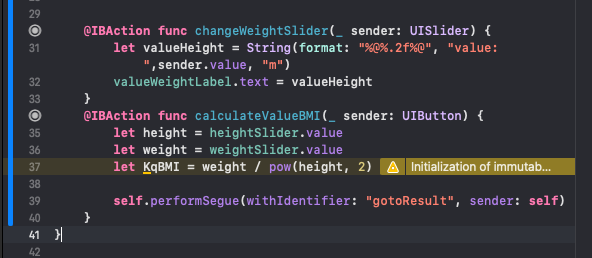
B3: Ta nhấn giữ ctrl và kéo calculate View Controller Scene tới ResulBMI Controller Scene, rồi chọn Present Moally



Tiếp túc ta tích vào cái đoạn lên kết giữa 2 màn hình, sửa identifier thành tên ta muốn(Ở đây là gotoResult)

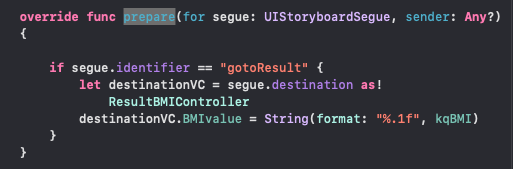


B4: Bên file calculateViewController, ta có



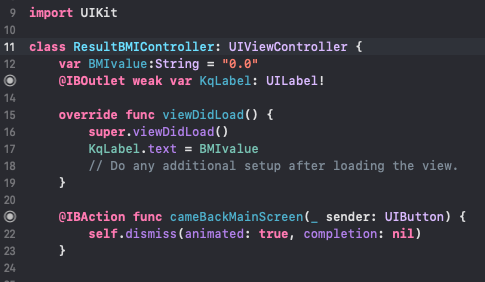
Khi gọi hàm self.performSegue, ta có “gotoReslt” là key ta đặt tên ở phần Bước 3. Bây giờ để gán giá trị cho object ở ResultBMIController.swift ta sang bước 5

B5: Khi ta muốn navigate sang 1 screen mới, ta thường muốn pass giá trị hay làm 1 cái gì đó. Lúc này ta sữ sử dụng hàm pepare(), hàm này được viết bên calculateViewController.swift



Đoạn code segue.destination sẽ Get new UIViewController, mà ta đang muốn nó trả về class ResultBMIController, nên ta phải downcasting

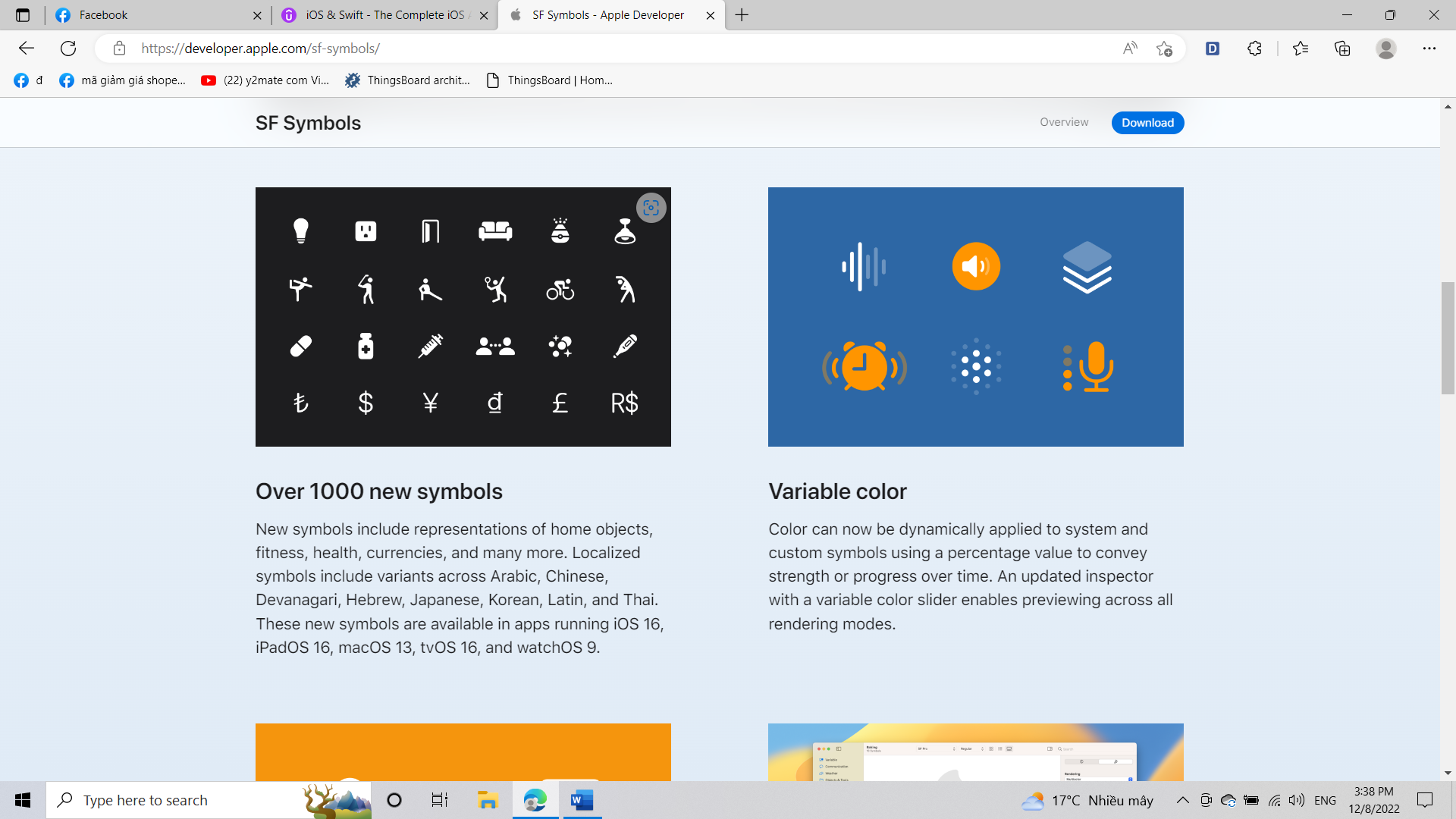
B6: Để trở về screen trước ta sử dụng method dismiss()



# 4 Build App Weather

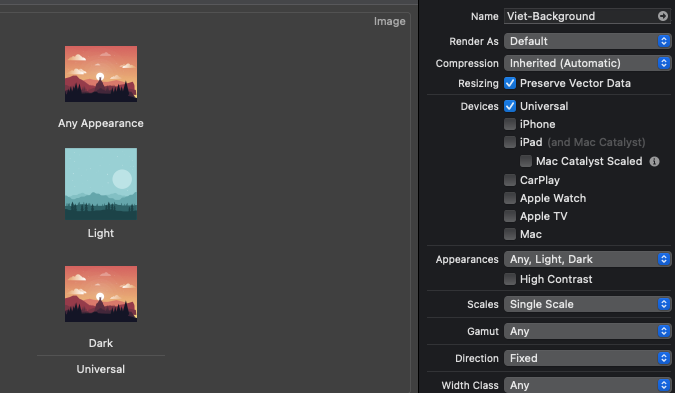
## 4.1 Dark Mode and working with vector assests

Ta có SF symbol đã được làm sẵn để ta them vào app



Hình như mấy ảnh này trong hệ thống rồi ko cần tải 😊 maybe

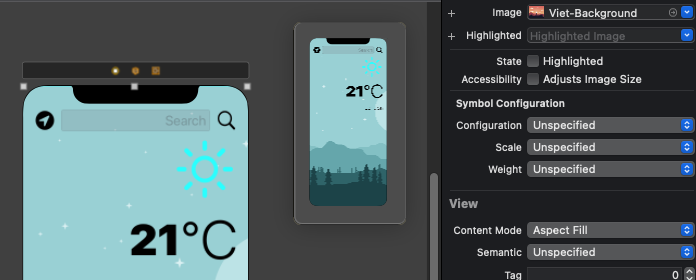
* Thay đổi ảnh background theo font



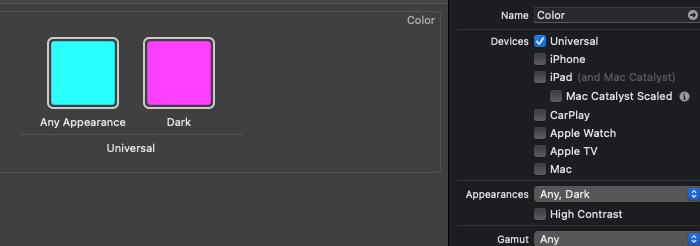
Kéo ảnh ta muốn chọn, tích chọn

* Preserve Vector Data.
* Chọn Appearances: Any, light, dark
* Scale: Single Scale

Sau khi thiết lập xong ở phần image ta chọn tên ảnh là viet-background, khi đó ta tích sáng thì sử dụng ảnh light mà tích tối sử dụng ảnh dark



Tương tự cho màu, ta cấu hình 2 màu là dark và light



Sau đó khi màu nào cần thay đổi theo sáng tối thì ta chọn



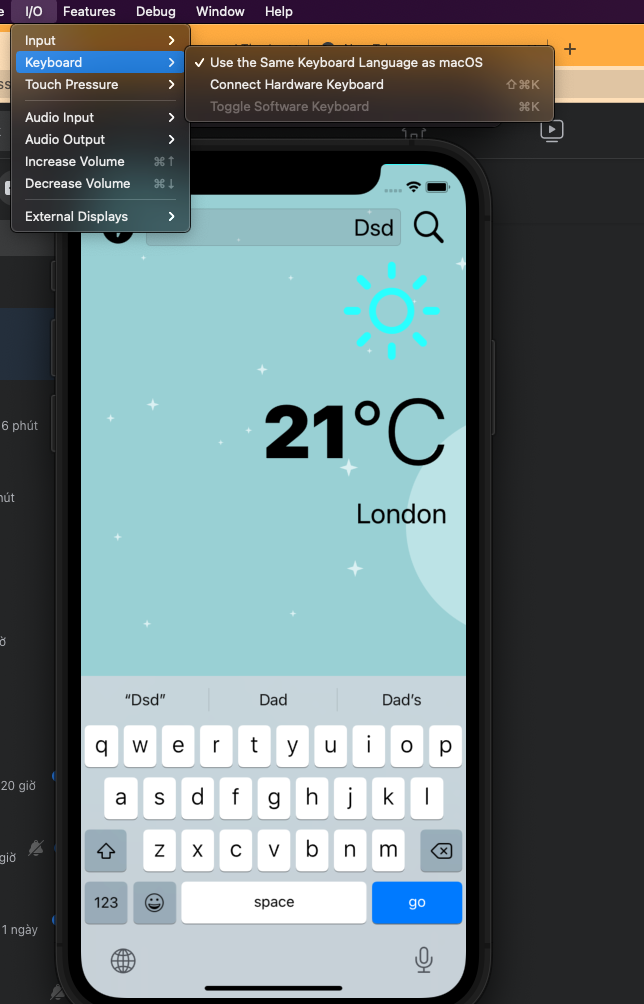
## 4.2 Learn to Use UITextField

Textfiled là vùng để thằng user nhập thông tin(input). Ta sửa

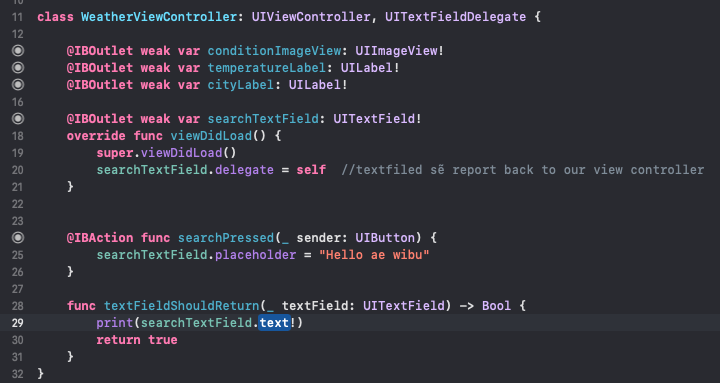
* Capitaliztion: sang word
* Return Key: sang go để hiện như ảnh bên dưới



Sau khi sửa thế ta mô phỏng được như dưới, nhớ bỏ chọn connect hardware



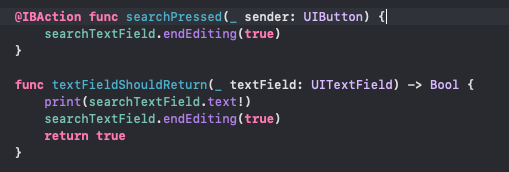
* Câu hỏi đặt ra tiếp theo là làm sao ta sử dụng được button **go.** câu trả lời là dùng UITextFieldDeligate



Self: refer to current view controller

Khi nhấn button go thì sẽ chạy vào function() textFiledShouldReturn(), đây là 1 function keyword.

* Câu hỏi tiếp theo là khi ta nhấn nut **go** nhưng keyboard vẫn hiện ở đó, vậy làm thế nào để tắt nó đi.
* Câu trả lời, ta sử dụng method endEditing()



* Ta có khi sử dụng method endEditing(), thì sẽ trigger cho hệ thống là đã nhập xong(xong ta tích vào textFiled vẫn được nhập tiếp nhé chứ ko phải ko cho nhập nữa đâu) và hệ thống sẽ trigger ra 1 function có tên keyword là **textFieldDidEndEditing** (Nếu ta viết bên dưới). Nghĩa là khi nào chuẩn bị ko nhập nữa thì sẽ gọi func này ra trước

Không có mô tả.

Function này sẽ xóa hết text trên ô nhập textFiled.

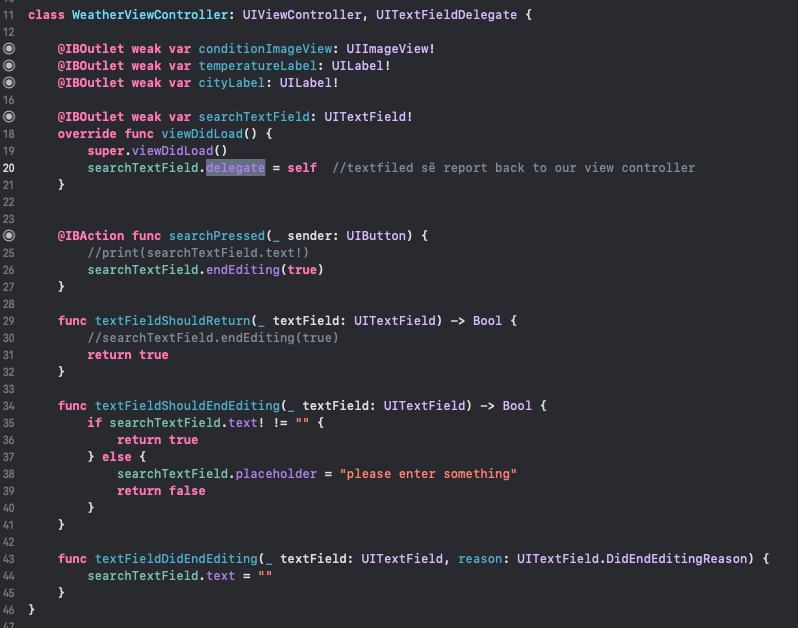
* Một câu hỏi tiếp theo đặt ra, nếu người dụng ko nhập gì mà bấm enter thì sao, data có thể là nul, vậy ta phải làm thế nào
* Câu trả lời là dùng function textFieldShouldEndEditing(), nếu return true nghĩa là cho phép việc nhập kết thúc và gọi func **textFieldDidEndEditing(),** còn return false thì sẽ không cho nhập xong.



* Tổng kết: Khi ta bấm nút **go** thì nó sẽ vào trong func **textFiledShouldReturn**(), sau đó khi nó chạy đến lệnh **searchTextField.EndEditing()**, nghĩa là thông báo cho hệ thống là người dùng đã nhập xong, nhưng trước khi hệ thống tắt keyboard, nó sẽ chạy func **textFiledShouldEndEditting**() để kiểm tra đầu vào của Users có hợp lệ không, nếu không thì sẽ không tắt keyboard, nếu có thì sẽ tiếp tục chạy func textFiledDidEndEditting().

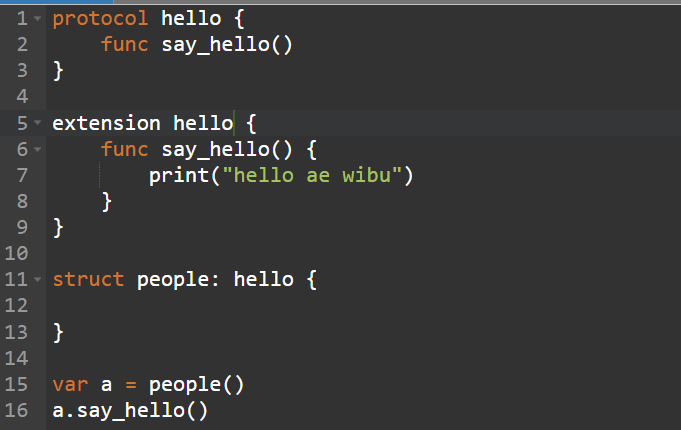
## 4.3 Nói 1 chút về Delegate

Ta có 1 ví dụ như dưới đây

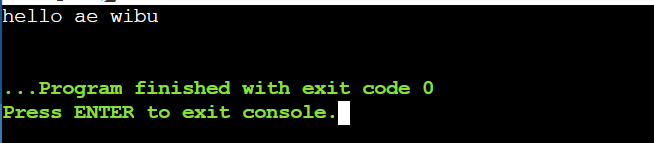


Ta sẽ giải thích cơ chế hoạt động của delegate như thế nào trong đây.

* Đầu tiên: class WeatherviewController kế thừa protocol UITextFeildDelegate, và trong protocol đó liệt kê các func như **textFieldDidEndEditing(), textFieldShouldEndEditing(),…**
* Thứ 2: Ta có 1 biến searchTextFiled thuộc class UITextField, và trong class UITextField đó có 1 biến delegate thuộc kiểu protocol UITextFeildDelegate. Nên khi ta gán searchTextView.delegate = self, nghĩa là ta cho phép biến delegate sử dụng các function **textFieldDidEndEditing(), textFieldShouldEndEditing()**  của class WeatherviewController. Lý do class WeatherviewController không cần viết hết các function của protocol UITextFeildDelegate chắc là do nó triển khai các extension, khi sử dụng extension như default của protocol ta không cần phải triển khai các function đó nữa



Output:



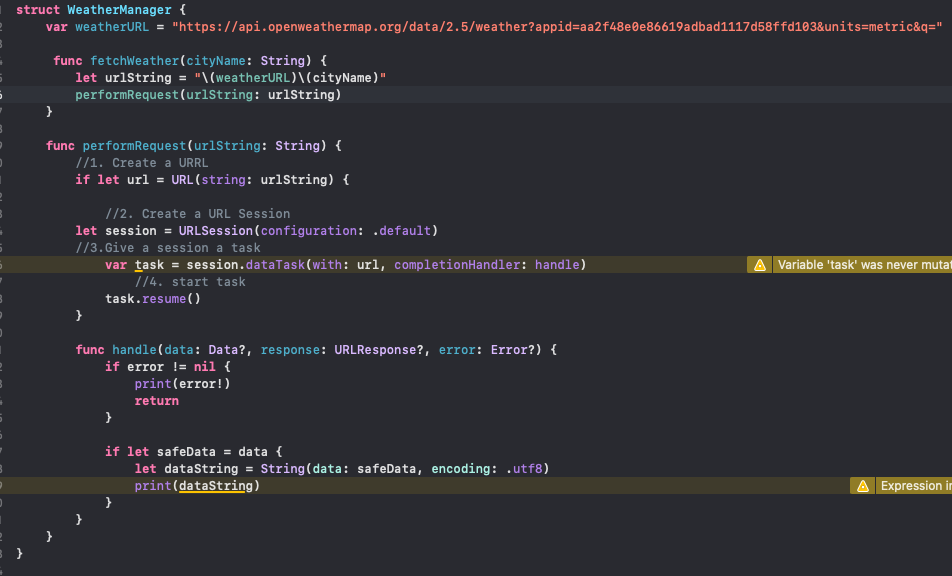
Ta thấy trong struct people ta không cần triển khai fun say\_hello() của protocol hello nữa. Vì func say\_hello() đó đã được triển khai bởi extension rồi.

* Thứ 3: Trong class UITextField, chắc chắn triển khai các func xung quanh biến delegate đó, vì chính nó sẽ trigger tín hiệu, lẫn biến cho nhau.

## 4.4 User URLSession for Networking

Các Bước:

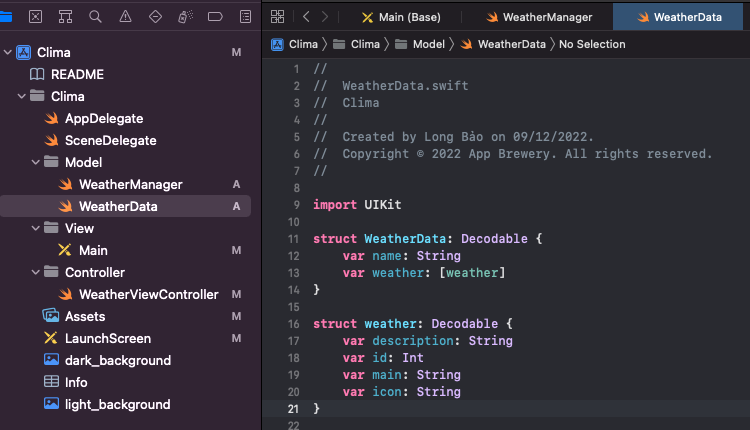
* Create a URL
* Create a URLSession
* Give URLSession a task
* Start this task



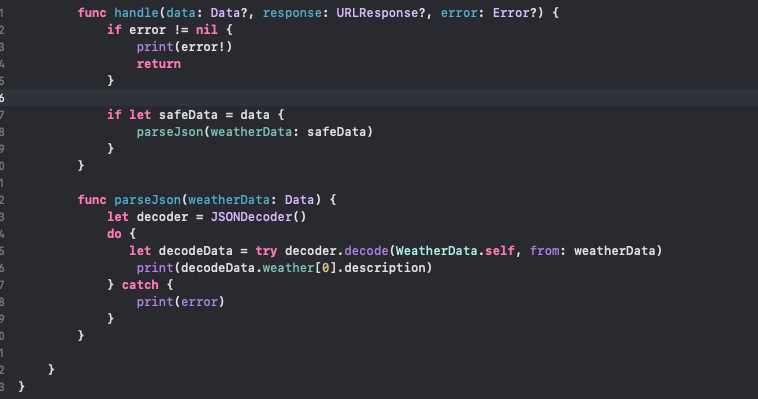
Sau khi task hoàn thành xong thì sẽ nhảy vào completionHandler

## 4.5 JSON Decoding

Trong Model ta tạo



Và trong hàm controller ta có:



Giải thich: Giả sử lúc đó data ta có kiểu nay:



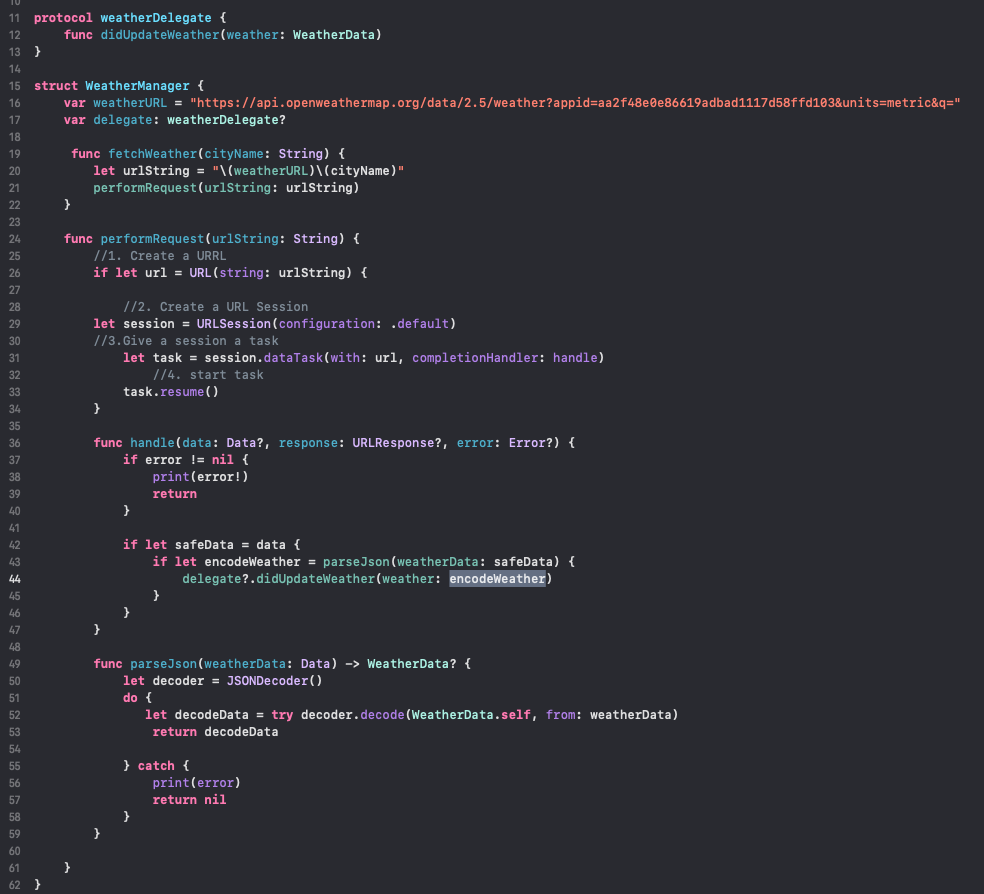
Ta đang muốn lấy thông tin “description” ở mục weather, ta để ý weather này đang có kiểu là mảng chỉ có 1 phần tử. Lúc này ta sẽ viết 2 struct như trên.

Ở mục parse data, method decoder.decode() có throw ra lỗi, nên ta phải sử dụng do, try, catch.

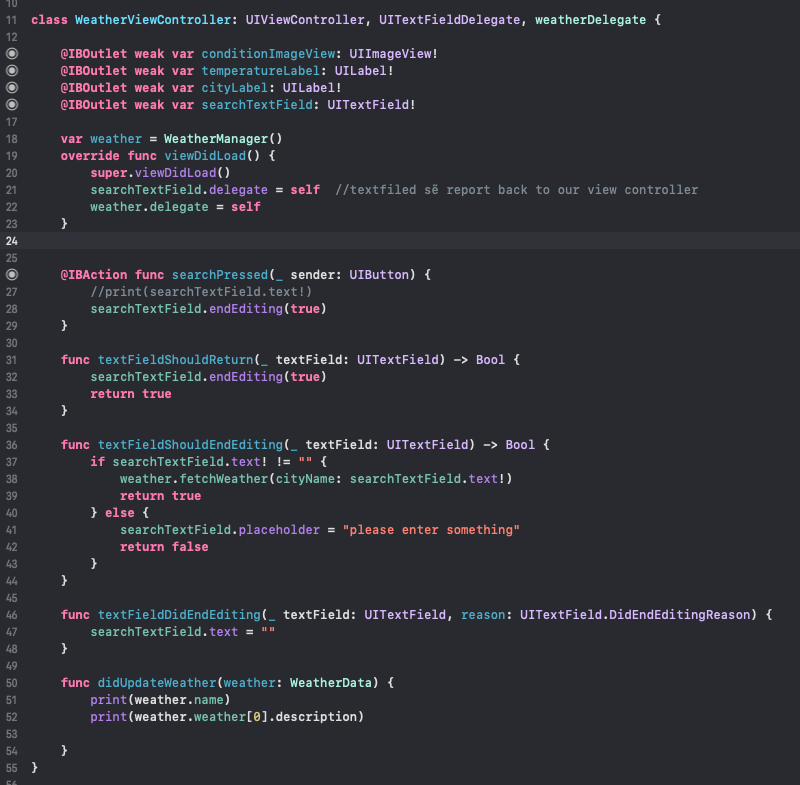
Biến WeatherData.self truyền vào là kiểu data ta muốn parse. Lúc này data json nhận được nó sẽ tự parse tương ứng vào các phần có trong Struct WeatherData.

* Câu hỏi đặt ra là làm thế nào ta truyền dữ liệu json đó tới Controller?? Câu trả lời là sử dụng delegate

File WeatherManager



File WeatherViewController



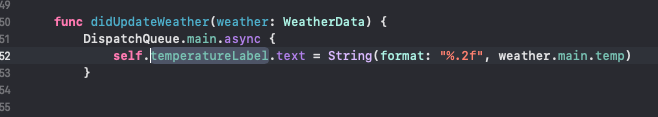
Giải thích:

* Ta tạo 1 biến delegate kiểu protocol weatherDelegate trong struct WeatherManager
* Class WeatherViewController kế thừa protocol weatherDelegate và triển khải func didUpdateWeather() của protocol đó
* Trong struct WeatherManager ta sử dụng biến delegate thông qua các method của class WeatherViewController.

## 4.6 Dispatch Queue

Ta có với giao diện của IOS, thì sẽ được main thread quản lý. Main thread sẽ thực thi các việc liên quan đến UI, như chuyển giao diện, hiện thông báo, zoom to nhỏ màn hình,… Với các task nặng như get data từ networking, sẽ được thực hiện ngầm trên thread background. Vậy lúc ta có hàm handle để thực thi task của session thì có vẻ như task đó được chạy trên background. Nên nó không được phép thay đổi giao diện của UI. Chỉ có thread main thay đổi giao diện của UI.

Vậy để thay đổi giao diện lúc này ta sử dụng Dispatch Queue như sau



Giải thích: Đel có 😊

# 5. Thread, Queue, GCD

Trong ứng dụng iOS hay bất cứ ứng dụng nào, khi chương trình được khởi chạy, hệ thống sẽ khởi động một **Thread ban đầu** cùng với **một tiến trình** thực thi một tác vụ nào đó. Thì Thread đó chính là **Main Thread**.

Khi tiến trình thực hiện xong nhiệm vụ của nó thì Main Thread sẽ kết thúc. Cũng như chương trình của chúng ta sẽ kết thúc theo. Đó là lý do, mà bạn lúc học C/C++ mãi không hình dung ra được. Cho tới khi, bạn bắt đầu nhúng tay vào việc tạo các ứng dụng có giao diện.

Đối với ứng dụng iOS nói riêng, nó thuộc hệ ứng dụng với giao điện đồ hoạ người dùng. Do đó, **Main Thread** sẽ chịu trách nhiệm xử lý giao diện. Nên nhiều lúc, chúng ta còn bắt gặp khái niệm tương tự là Main UI, UI Thread … hoặc khi lười thì nói chữ **Main** là ám chỉ cho toàn bộ khái niệm ở trên.

## 5.1 Grand Central Dispatch(GCD)

Là một **API** phổ biến của Apple được sử dụng để hỗ trợ việc xử lý tác vụ đồng thời trên các phần cứng đa lõi của iOS và OSx. Với **GCD**, chúng ta có thể thực hiện các thao tác sau một cách đơn giản:

* Gửi một tác vụ đến luồng chạy ngầm(background thread) chạy song song với luồng chính(main thread).
* Dùng chung dữ liệu giữa các luồng(mutex).
* Lập lịch chạy cho một tác vụ.
* Xếp các tác vụ chạy theo thứ tự.
* **GCD** hoạt động dựa trên cơ chế Thread Pool: thay vì phải tạo các threads trực tiếp thì chúng ta sẽ đưa các tasks vào **Queues**. **Queues** sẽ tự động tạo và huỷ threads, còn việc tạo bao nhiêu threads thì sẽ do hệ thống quyết định dựa trên công việc của CPU đang thực thi. Do đó công việc của developers đơn giản chỉ là quản lý trên Queues mà thôi.
* **Queues**(hàng đợi):  khái niệm **Queue** được hiểu là các hàng đợi chứa những block code để đợi các thread thực thi.  Queues quản lý những tasks mà bạn submit tới và chạy chúng theo thứ tự FIFO. Điều này đảm bảo rằng task đầu tiên được thêm vào queue sẽ là task đầu tiên được start sau đó lần lượt các task được start theo thứ tự đến hết queue. GCD cung cấp cho chúng ta 2 loại hình của DispatchQueue là: **Serial Queues** và **Concurrent Queues**.

## **5.2 Serial Queues**

Là hàng đợi thực hiện tuần tự tức là chỉ có 1 task được thực hiện trong 1 thời điểm và khi nào task này thực hiện xong thì task khác mới được bắt đầu.

**Main queue**: là queue đặc biệt có sẵn trong hệ thống và là **Serial queue**, nó chạy những tasks trên **main thread** của ứng dụng***. Được sử dụng để update UI của app, thực hiện những task liên quan đến update UIViews hay posting notifications***. Bởi vì nó là serial queue nên chỉ có một task được chạy tại một thời điểm, do đó bạn sẽ bị block UI khi chạy những task nặng trên main queue (vd: download image từ server về). Mặc định hệ thống chỉ tạo duy nhất 1 queue này.

## 5.3 Concurrent Queue

Là hàng đợi thực hiện đồng thời. Trong một thời điểm có thể có nhiều task được thực hiện cùng một lúc. Hệ thống sẽ tuỳ vào tải hiện thời của hệ thống và cấu hình phần cứng thực tế để khởi tạo và cấp phát các Threads để xử lý các tác vụ. **Global queue là concurrent queue và có sẵn trong hệ thống.**

Đến đoạn này chắc nhiều bạn sẽ thắc mắc là Queue hoạt động theo nguyên tắc FIFO thì liệu Concurrent queue có đảm bảo theo nguyên tắc này không nhỉ ?

Câu trả lời là có . Tại sao lại như vậy? Thực chất nguyên tắc này thực hiện trên từng thread và trong concurrent queue thì nó được thực hiện đồng thời trên nhiều threads.

## 5.4 Synchronous and Asynchronous

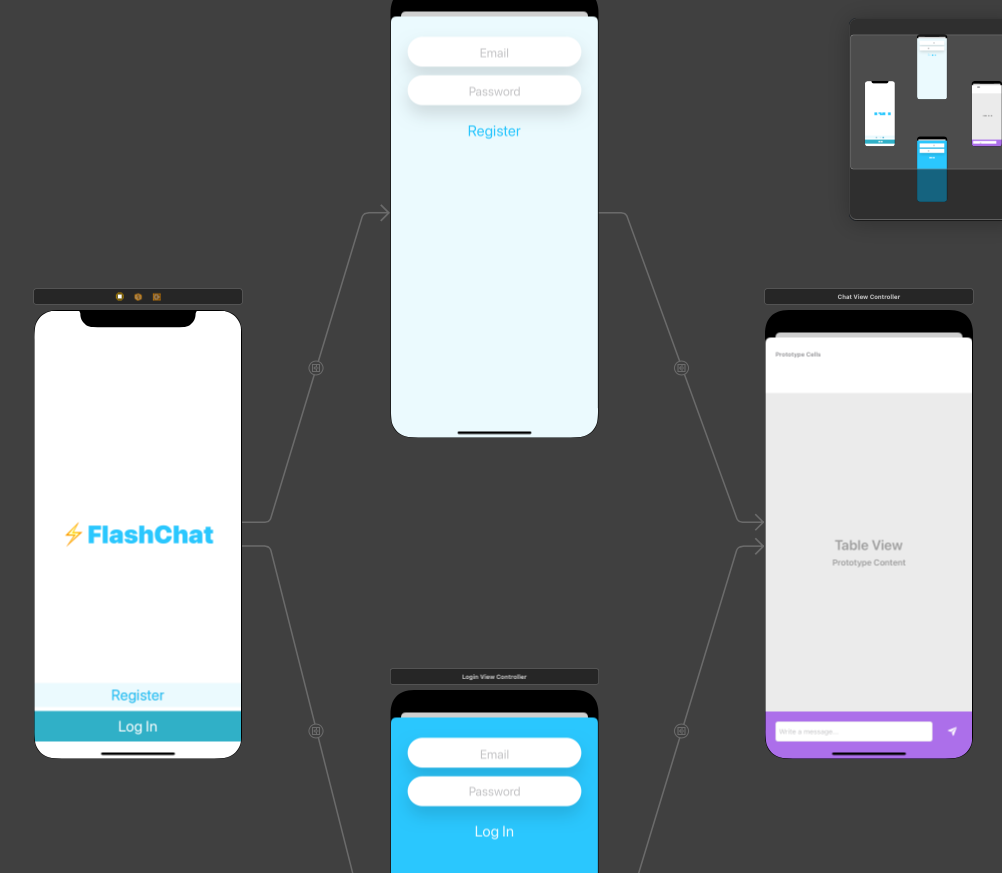
# 6. Flash Chat

## 6.1 Navigation Controller Stack and Segues

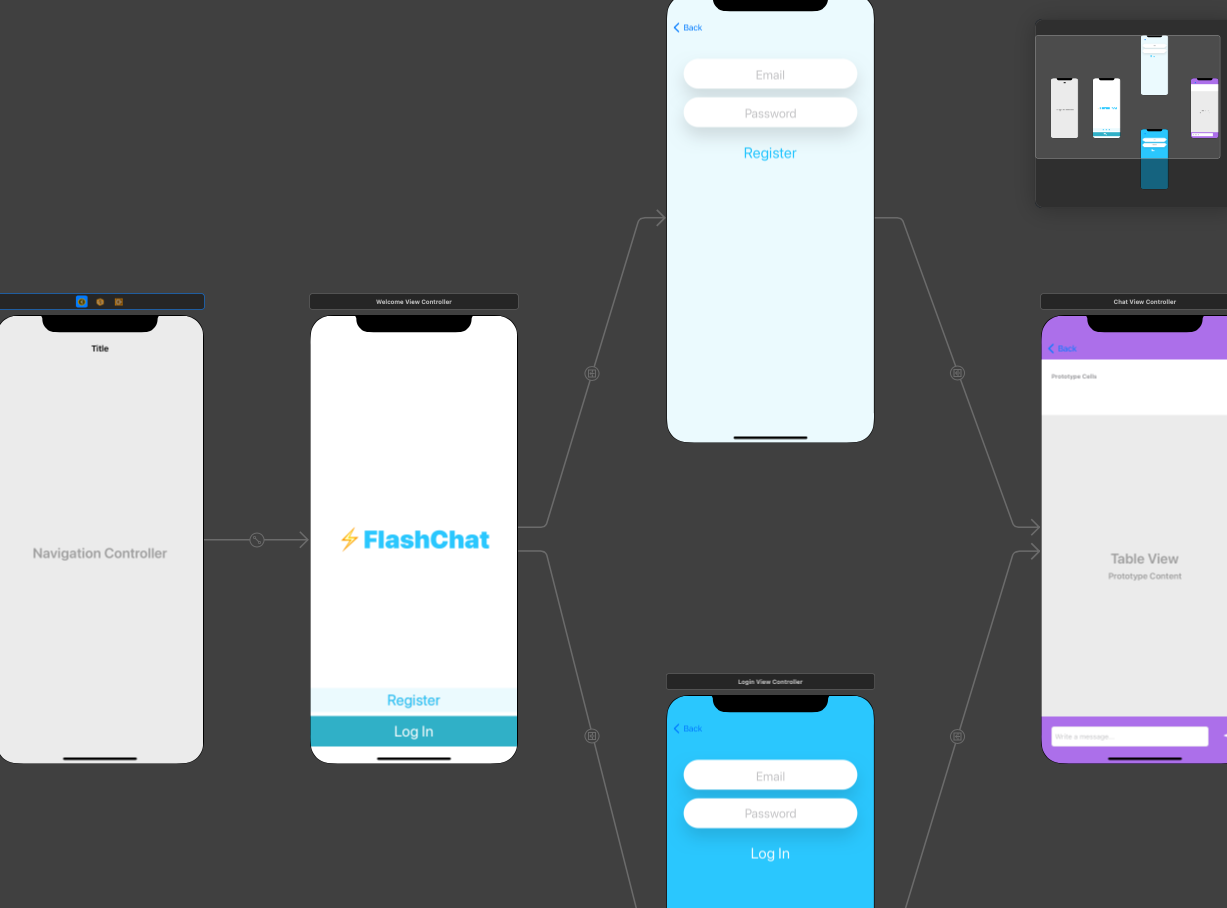
Các types segues action:

* **Show** - Pushes the destination view controller onto the navigation stack**, sliding overtop from right to left*, providing a back button to return*** - if not embedded in a navigation controller it will be presented modally(ta tích vào cái ô vuông 😊)) chọn editor trên thanh taskbar, chọn embedde in và chọn task b  
  Example: Navigating in Settings, for example tapping General > About

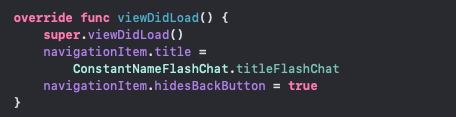
**Không Embedded in**



**Có Embedded in navigation:** Có them nút back trở về vị trí cũ và không bị đè như kia nữa, và các màn hình sẽ theo kiểu stack.



Với việc sử dụng navigation, thì ta được cung cấp các biến xử lý như: navigation để chỉnh các item như nút back, item… navigationController để trở về root giao diện

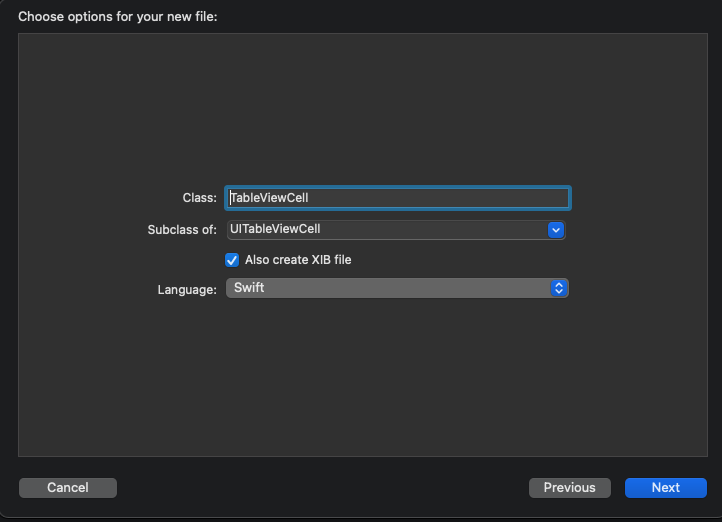


Không có mô tả.

* **Show Detail** - For use in a split view controller, replaces the secondary view controller when in a multi-column interface, or if collapsed to one column it will push in the navigation controller  
  Example: In Messages, tapping a conversation will show the conversation details - replacing the view controller on the right when in a two column layout, or push the conversation when in a single column layout
* **Present Modally** - Presents a view controller overtop the current view controller in various fashions as defined by the modal presentation and transition style - most commonly used to present a view controller in a sheet that animates up from the bottom  
  Example: Selecting Face ID & Passcode in Settings

## 6.2 Table View Shell

B1. Ta khởi tạo 1 file riêng chứa table view shell, chọn cocoa touch, chọn subclass là UITableViewShell, và create file XIB.



B2. Xét tên identifier cho TableViewShell đó

B3. Add TableViewShell vào TableView bằng cách:

* Delegate UITableViewDataSource và nhớ delegate nó bằng: tableView.datasSource = self
* Sử dụng method register của tableView để register TableViewShell đó vào tableView



* ConstantNameFlashChat.reuseable là tên idenfier ta xét bên trên đó còn ConstantNameFlashChat là tên file ta tạo ra mà kế thừa UITableViewShell
* Viết extension để kế thừa:



Func đầu là xét table view sẽ được gọi bao nhiêu lần, Func thứ 2 là xét nội dung hiển thị trên các lần đó.

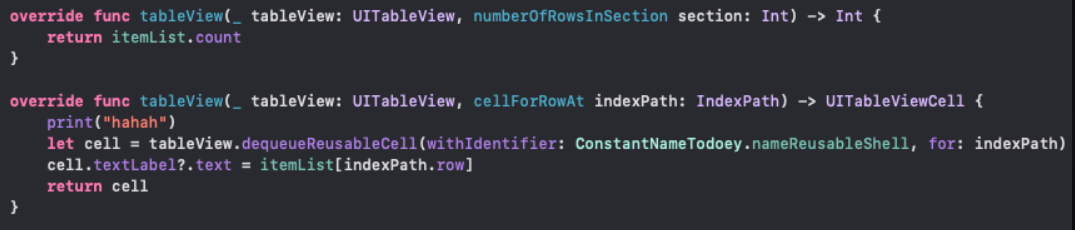
# Bài 7. Local Data Persitance, Users Default, Core Data and Realm

## 7.1 Add cell vào tableView

Ở bài này, ta sẽ sử dụng UITableViewController. UITableViewController là 1 Controller đã được tích hợp sẵn TableView và TableViewCell

Ở bài 6, ta có 2 hàm để xác định số hàng của TableView và hiển thị nội dung của từng hang đó. Ở đây ta có 2 hàm override có tác dụng tương tự như vậy

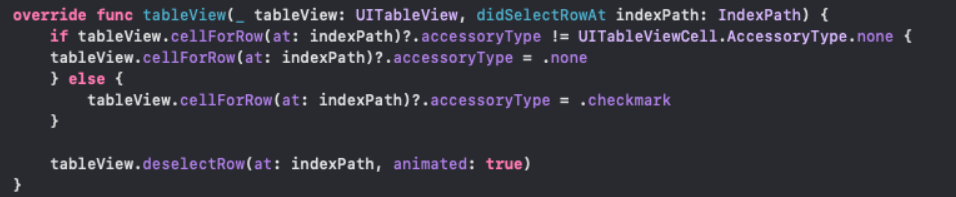
**Chú ý là khi ta kế thừa UITableViewController, ta đã có sẵn biến tableView sử dụng trong toàn class**



Function đầu trả về số lần viewSehll được add vào tableView

Function sau là add nội dung vào hang đó

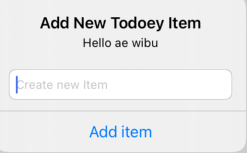
Khi ta tích vào các viewShell sẽ gọi hàm didSelectRowAt.



7.2 Alert trong UITableViewController



Kết quả:

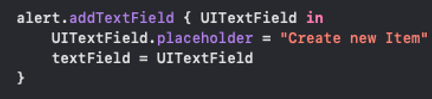


Giải thích:

* 

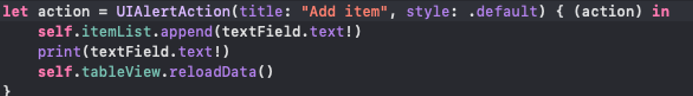
Hàm này sẽ tẹo ra 1 biến thuộc kiểu UIAlertController. Khi ta gọi alert.present(alert,……) thì alert này sẽ được gọi và có bảng thông báo như kia hiện lên.

Bên trong bảng thông báo hiện lên, ta còn có thể thêm các tính năng như là addTextFiled vào trong cảnh báo đó



Data nhận được sẽ được lưu vào biến UITextFiled, nhưng vì đó là biến local của hàm closer nên khi ra khỏi closure thì UITextFiled không thể sử dụng nữa, nên ta phải tạo 1 biến bên ngoài và trỏ tới địa chỉ tham chiếu của UITextFiled đó để lấy giá trị input từ người dùng

Add action vào alert và thực thi nó, nó sẽ xuất hiện 1 button có title “Add item”, ta tích vào button đó sẽ trigger action.



7.3 Persitent Local Data Storage Using UserDefault.

Để lưu 1 biến khi ta tắt chương trình đi, ta có thể sử dụng UserDefault

Không có mô tả.

Không có mô tả.

Không có mô tả.

Lưu ý: UserDefault chỉ lưu được các type như mảng, dictionary,…. Không lưu được các object như class và struct. Và userDefault chỉ được sử dụng để lưu các giá trị ít dung lượng, vì khi lưu các biến lớn sẽ làm chương trình bị châm.

Lưu ý 2: Vì ta sử dụng reuseableviewCell, nên khi ta kéo xuống và ko hiển thị bên trên, thì các thuộc tính của nó sẽ được áp dụng xuống cell ở dưới dưới 😊)))) khó nói vl, xem bài 242 creating Custom Data Model để hiểu. Chính xác là nó sẽ nhảy linh tinh. Nên ta cần gán kiểu giá trị như này

## 7.3 Sử dụng file plist để lưu data iphone

 Here we are gonna discuss about using the **FileManager** instance in swift so that you can **store data in files** and use it for your upcoming projects . **Filemanager** is very handy and its api’s are super easy to understand and use as well . **Filemanager** allows you to store data of your app in files you can create , delete , update and read those you have full control over it 😉.

 You can store you files and folders in document directory of your device , to get started you have to first create a Filemanager instance **Filemanager.default** , default is the shared instance of Filemanager and that’s what we are gonna use



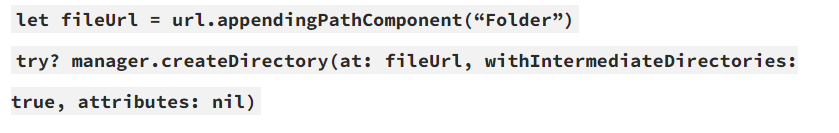
Cái này khởi tạo FileManager thông qua singleton.

Our first step will be to navigate to document directory and you can do that by



**urls** Returns an array of URLs for the specified common directory in the requested domains. In easy language it returns a array with different urls and **the first element in the array is the url for document directory and that’s what we are interested in**  
You can print the path by printing **url.path** and use command **open “ThePrintedPathHere”** in your terminal to open the directory in your **finder**

Now that we have the right directory (Remember you can do anything in this directory), **We are gonna create a folder**



This will **create a folder** named “**Folder**” at specified path of FileUrl (You can print FileUrl.path too and open it as well) if you want another folder inside the folder  You can do it easily by just appending one more path in the FileUrl  
example

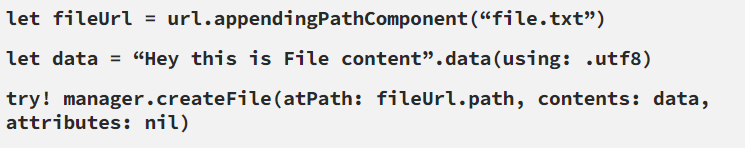


Lets **Delete** this directory , Nothing too fancy you can easily do it



* Creating File

**Same way you can also create any file that you want whether its a text file of .plist file etc**



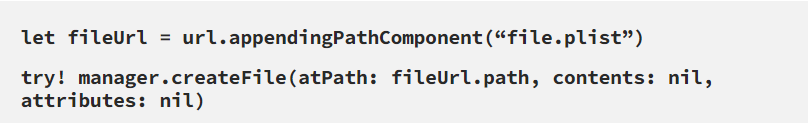
Here we pass the path and contents for that txt file , the data is string which is than converted into type Data using the .utf8 encoding

You can also delete file using

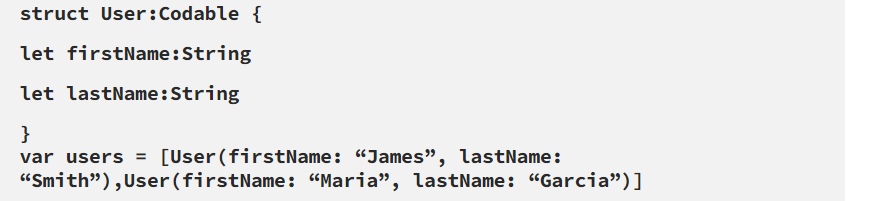


**Now lets get Towards creating .plist files  
what are plist Files? , they are actually xml files used to store data in form of key and values pair (sort of like dictionaries in swift)**

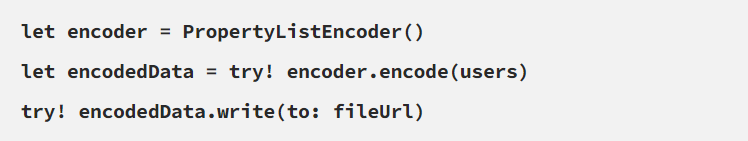
**Create a plist file as shown below**



Also create our data to store in file , we will have a **struct** and store an array of that **struct** types , make sure the struct confirms to **Codable** protocol

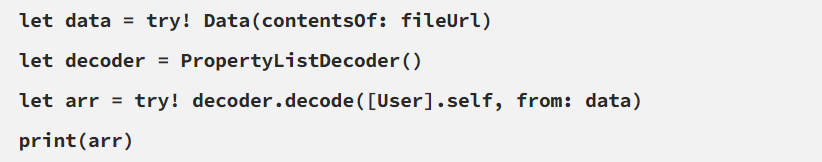


Now we are gonna store that **users** array in plist , for that we need to **encode** out data using the **property list encoder** and same way while retrieving the data we are gonna **decode** the data using **property list decoder , Codable**is typealias for **Encodable** and **Decodable** protocol .



This **encoder.encode**will encode and return data we than write the data to our file , **Note: Here i am using try! this is not a good practice if you are gonna use this code for a production project ,do proper error handling using do{} catch{} blocks to convey errors**

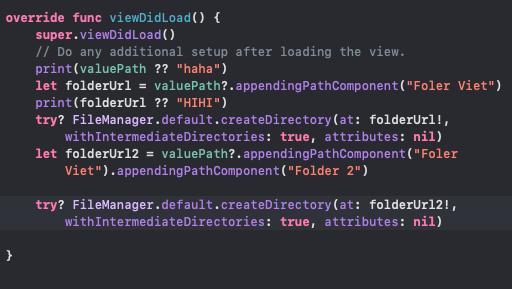
**Retrieving Data from plist**



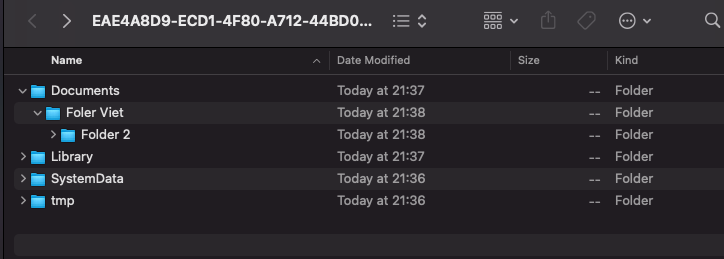
here we retrieve the data and print it , remember the data we stored was of type **[User]** so we pass the **data type** and **data** as **arguments** , that’s it there are more of other methods too

## 7.4 Ví dụ file plist

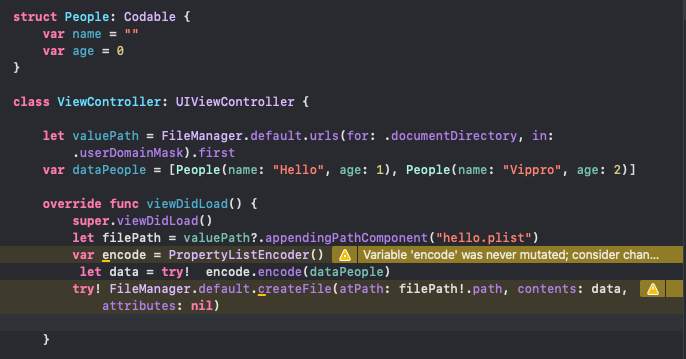
* Tạo folder:



Output:



* Tạo file plist và truyền value



Output:



* Get data từ file plist



Output:

Không có mô tả.

## 7.5 CoreData

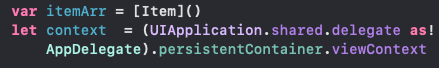
Các bước sử dụng Core data:

B1: Tạo file mới, khi tạo file tích chọn core data, paste phần lazy và saveContext() sang project ta muốn them coreData, đổi name NSPersitentContainer(name : “”) thành tên ta muốn

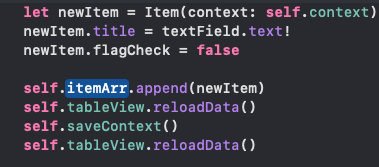
B2. ở project ta add them file CoreData, tên này trùng với tên ta them bên dưới

B3. Giờ qua code chính

Ta sẽ tạo 1 biến context của NSPersitentContainer thông bằng code dưới



Tạo Item mới



Sửa item tại vị trí(Cái này là hàm của Array cmnr)

Không có mô tả.

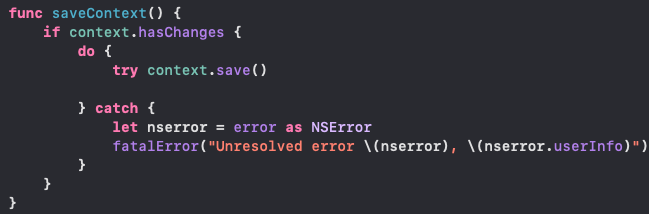
Xóa item

Không có mô tả.

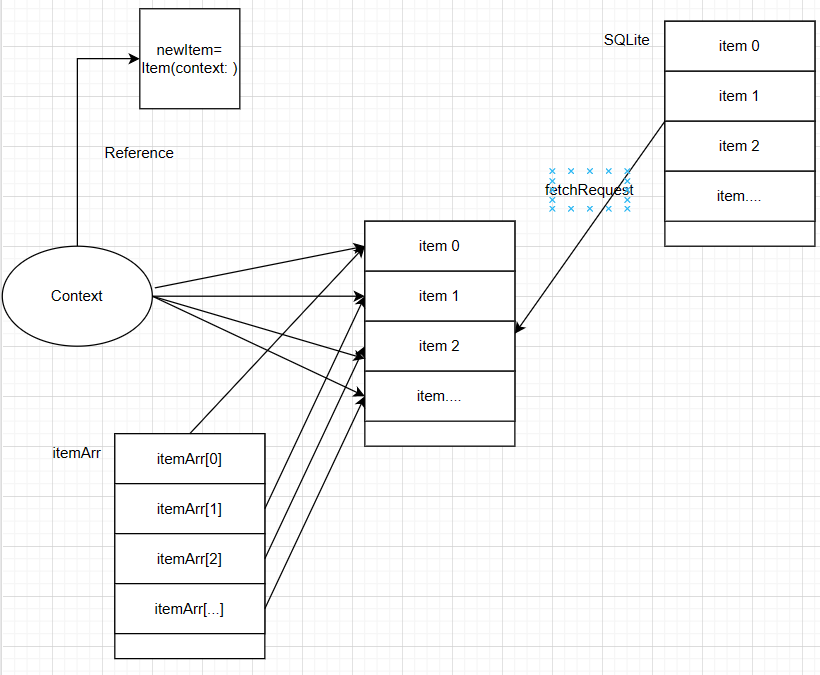
Lấy toàn bộ data từ SQL



Save Data

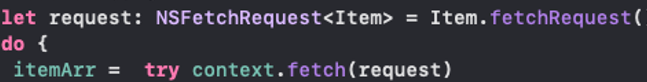


* OK giờ đó là các hàm chính được sử dụng, giờ ta sẽ giải thích



Lưu đồ này cho phép ta cách hoạt động của Context.

* Ta sẽ coi Context là vùng nhớ tạm của NSPersitentContainer có name đặt tên bên trên. Khi ta tạo 1 biến mới Item(context: ) thì sẽ cho phép context Reference đến vùng nhớ đó. Bất kì vùng nhớ nào được Context reference sẽ được push lên database.
* Sau đó ta muốn Read data của database thì ta sẽ gọi hàm fetchRequest()



Lúc này ta sẽ cho phép Context tham chiếu lên toàn bộ vùng nhớ được kéo về, đồng thời biến itemArr cũng tham chiếu lên vùng nhớ đó.

Vì vậy, khi itemArr sửa đổi data của bất kì item nào(itemArr[0], itemArr[1] …) cũng như là việc Context sửa đổi lên vùng nhớ đó. Vì cả 2 đang tham chiếu lên vùng nhớ đó.

Không có mô tả.

Chính vì vậy, tuy ta thay đổi data ở itemArr cũng làm thay đổi ở context.