Contents

[1. NavigationController 2](#_Toc132653670)

[2. TabBarController 3](#_Toc132653671)

[3. UIImagePickerController 5](#_Toc132653672)

[4. Auto Layout 6](#_Toc132653673)

[4.1 Anchor 6](#_Toc132653674)

[4.2 Content Size Instrict 10](#_Toc132653675)

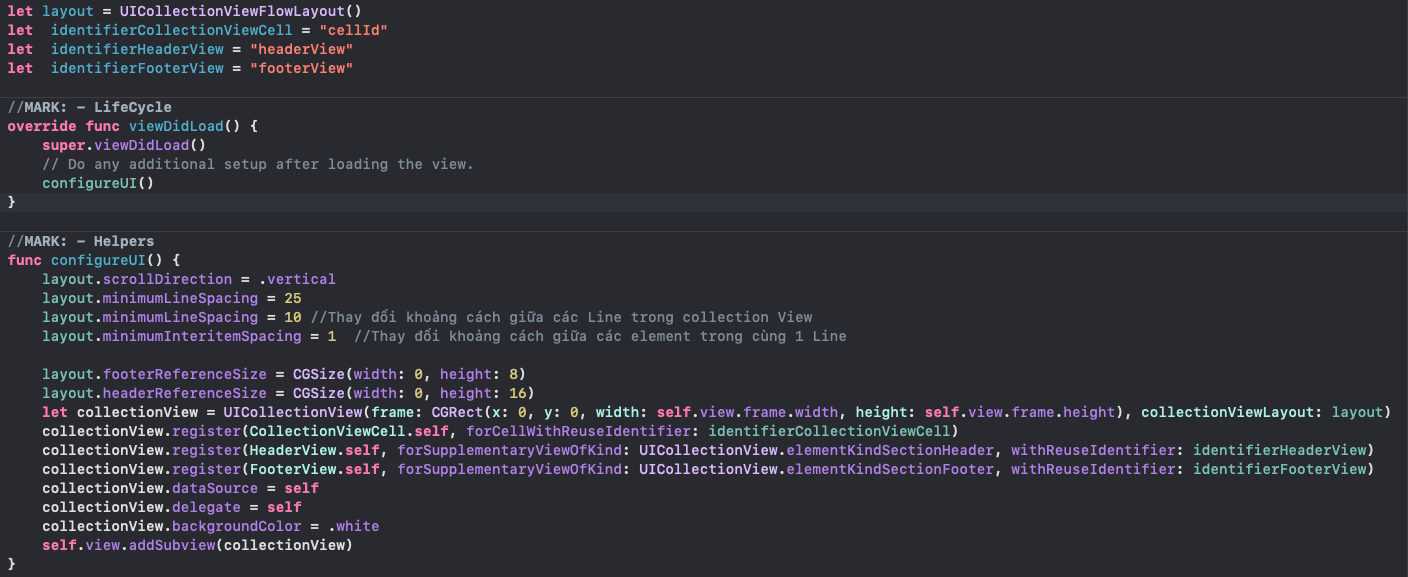
[4.2.1 Content hugging priority 10](#_Toc132653676)

[4.2.2 Compression Resistance Priority 12](#_Toc132653677)

[5.Stack View 13](#_Toc132653678)

[V. Collection View 17](#_Toc132653679)

[5.1 Các bước thực hiện 18](#_Toc132653680)

[ 19](#_Toc132653681)

[VI. SearchController 21](#_Toc132653682)

[VII. Bankey Project 23](#_Toc132653683)

[7.2 Constrain kiểu 2 25](#_Toc132653684)

[7.3 Type Button 27](#_Toc132653685)

[7.4 Page View Controller 28](#_Toc132653686)

[8. CollectionView Composional Layout 28](#_Toc132653687)

# NavigationController

Trọng mọi ViewController ta luôn có các properties như:

* NavigationController
* navigationItem

Khi 1 ViewController được nhúng vào 1 NavigationController, thì biến NavigationController sẽ khác nil, nên từ đây ta có thể truy cập các propertuies bên trong NavigationController đó.

Trong NavigationController có thuộc tính **navigationBar** được sử dụng để thay đổi background, titleAttribute, …

Tóm lại:

* navigationBar: được sử dụng để thay đổi các thuộc tính của thanh navigationBar như color title,sizetitle, backgroundColor,…
* navigationItem: được sử dụng để quản lý các item trên thanh navigationBar đó như: Nội dung title, các button,…

VD:

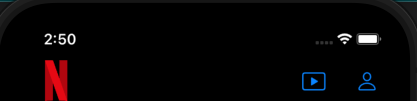


* Có 1 cách khác để thay đổi thuộc tính background color,… đó là sử dụng UINavagationBarAppearance(), khi sử dụng appearance thì mọi ViewController thuộc stackView của NavagationController sẽ mang thuộc tính giống hết với appearance. VD ở ảnh trên.

Xét left hoặc rightBarButton cho thanh navigationBar

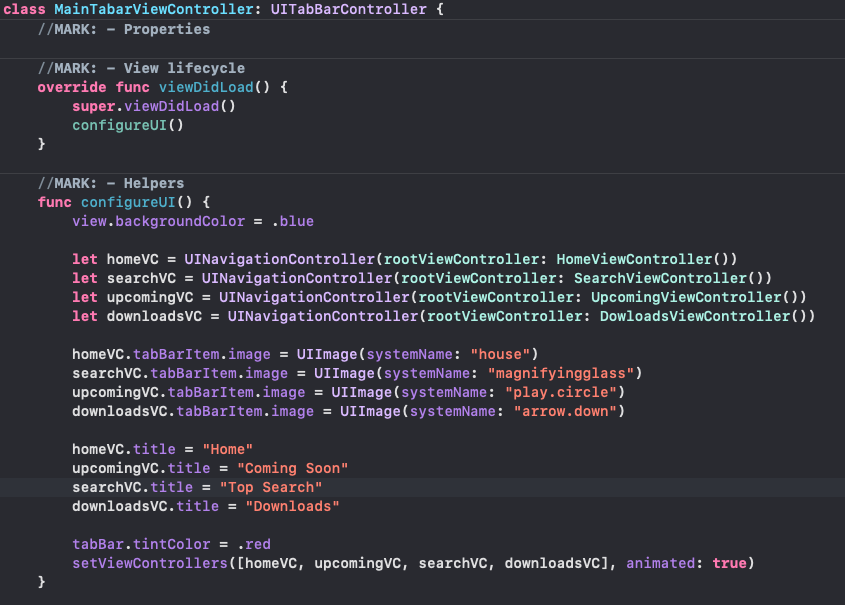


Kết quả:

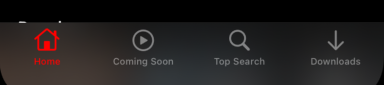


# TabBarController

Ở MainTabBarController: Xét ảnh và title cho menu TabBar

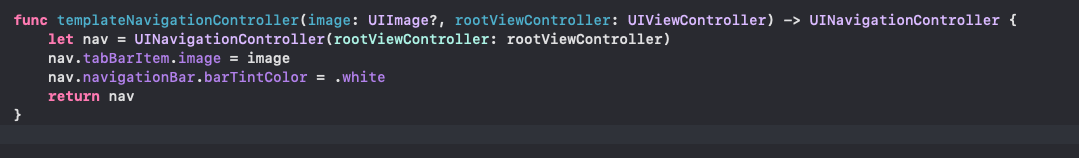


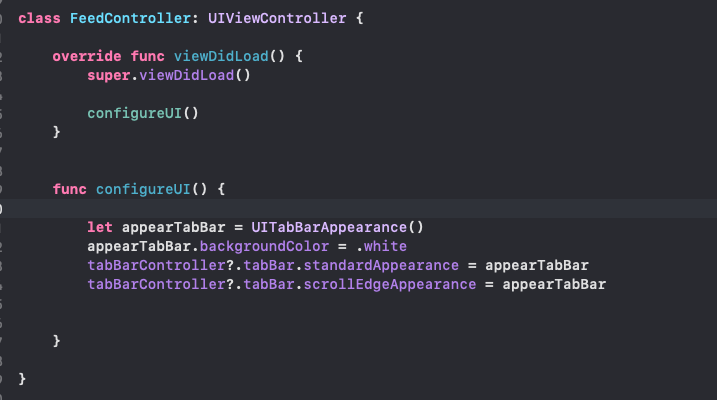
Ta được:



Ta vẫn có trong mọi ViewController ta luôn có các properties như:

* tabBarController
  + Quản lý naviagtionBar, và navigationBar quản lý các thuộc tính như màu sắc, background của tabBar đó.
* tabBarItem 🡪 Được sử dụng để hiển thị image, label ở tabBar





# UIImagePickerController

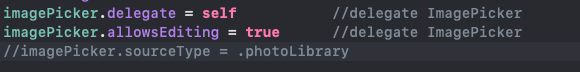
Được sử dụng để lấy image, video,.. từ iphone của mình.

Các bước:

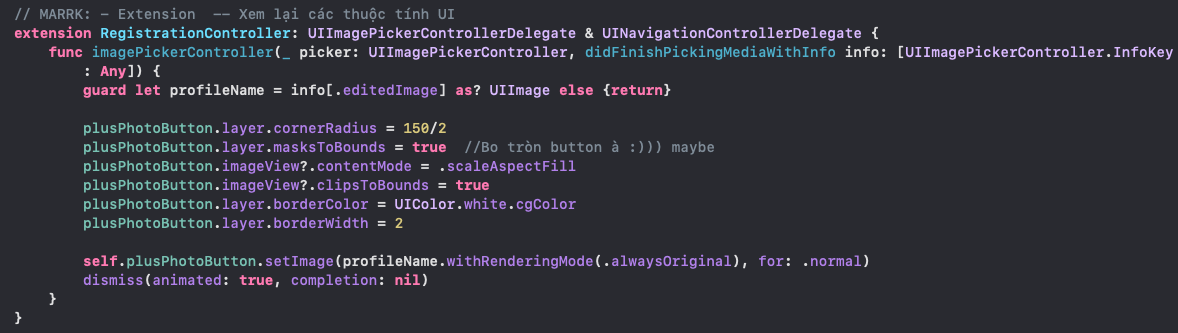
* Khởi tạo 1 biến global PickerView

Không có mô tả.

* Delegate



* Delegate để lấy dữ liệu



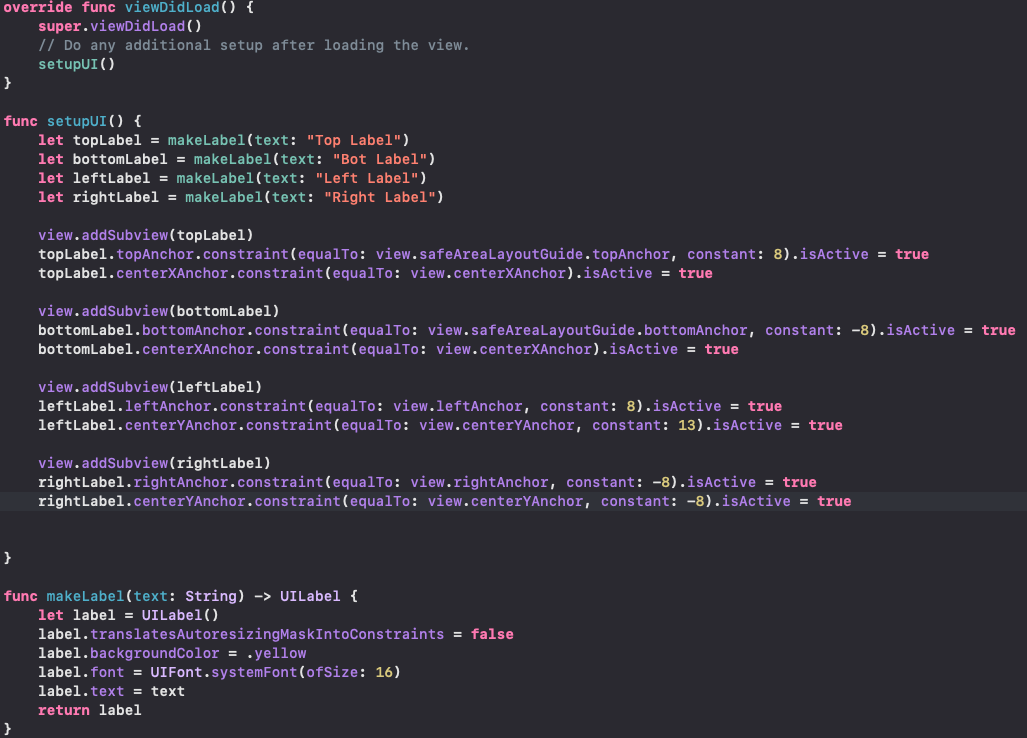
# Auto Layout

Chú ý: Để có thể auto Layout bằng code được thì mỗi element phải triển khai dòng code như sau:

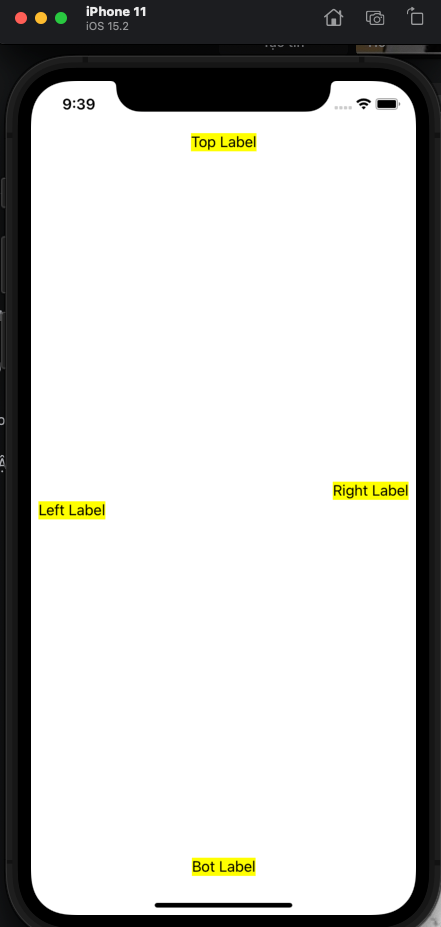
Không có mô tả.

## Anchor

Ta sử dụng các thuộc tính anchor để constrain:



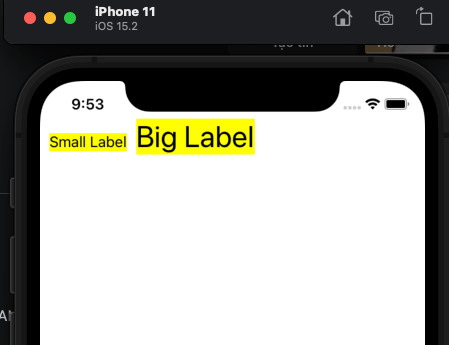
Kết quả:



* Khi ta muốn các text, label ngang hang nhau thì ta có thể sử dụng them thuộc tính firstBaseLineAnchor hoặc lastBaseLineAnchor:



Kết quả:



## Content Size Instrict

Ta thường dùng contraints để định rõ vị trí và size của các view. Tuy nhiên, có một vài view thường có size co giãn theo đúng nội dung mà nó chứa.  Khái niệm này được gọi là Intrinsic content size(kích thước nội dung bên trong).  Vd đơn gỉan hen: 1 button có intrinsic content size là kích thước của toàn bộ nội dung tên button cộng thêm một phần nhỏ của lề.

Ta nên biết rằng, không phải toàn bộ các view đều có intrinsic content size.

UILabel: The size that fits its text

UIImageView: The size of the image,....

UIButton: The size of the buttons label plus some padding

>>>>>>>>>>>>>>>>>> Chú ý: UIView has nó instrict content Size

* Intrinsic content size dựa trên nội dung bên trong của 1 view. Intrinsic content size của 1 label hay button dựa trên số lượng text nội dung và font mà nó sử dụng. Đối với những view khác, intrinsic content size thậm chỉ có thể phức tạp hơn. Ví dụ, một imageView khi không có hình nào thì sẽ không có intrinsic content size. Ngay khi bạn add hình vào nó thì intrinsic content size của imageView đã được set lại là size của hình bạn vừa add.
* Intrinsic content size của một text view có thể khác nhau dựa trên nội dung của nó, khi nó được scroll lên hay không hoặc thậm chí là từ những contraints khác áp dụng vào cho text view này. Ví dụ nha, khi mà bạn enable việc scroll thì text view này sẽ không có intrinsic content size. Khi mà disable việc scroll, thường thì intrinsic content size sẽ được tính toán dựa trên size của đoạn text

### 4.2.1 Content hugging priority

Khi ta muốn 1 view nào đó không tự scale lớn hơn kích thước content của mình thì dùng Content Hugging. Nghĩa là giữa những view với nhau, cái nào có Content Hugging càng lớn thì khả năng bị scale càng nhỏ

Ta có 1 đoạn code như sau:



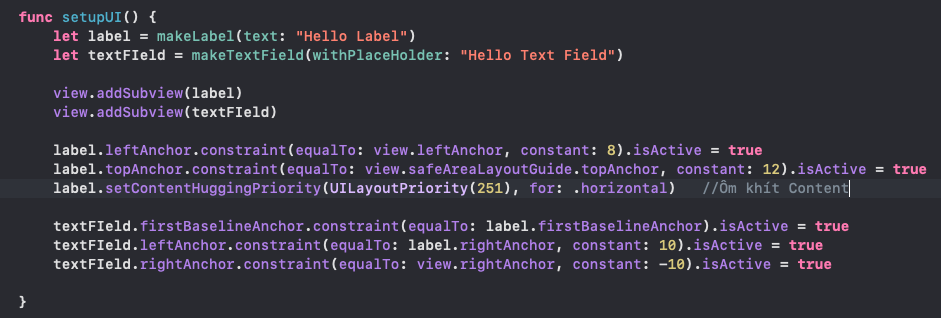
Output ra như sau:



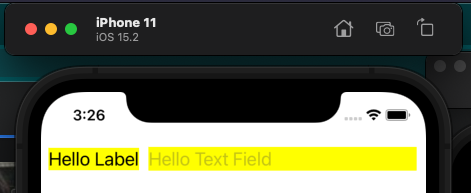
Tại sao Lable lại dài ra như vậy ???

Câu trả lời là lúc này ta sử dụng 2 View thuộc dạng Instrict Content Size và được neo ở 2 bên mà. Lúc này sẽ xảy ra hiện tượng ambimous, chương trình không biết ưu tiên cài nào để kéo dài ra. Vì mặc định chúng đang có cùng Content Hugging là 250.

Vậy giờ ta muốn size Label chỉ vừa content bên trong label, ta sẽ tăng Content Hugging lên, để nó có quyền để size view không lớn hơn content view



Kết quả thu được:



### 4.2.2 Compression Resistance Priority

Gán giá trị này càng lớn có nghĩa là ta không muốn view thu nhỏ lại nhỏ hơn intrinsic content size



Với việc ta xét Priority của image.setContentCompressionResistance xuống 749, thấp hơn so với label là 750, nên size của label sẽ được giữ lại và size của image sẽ bị giảm đi

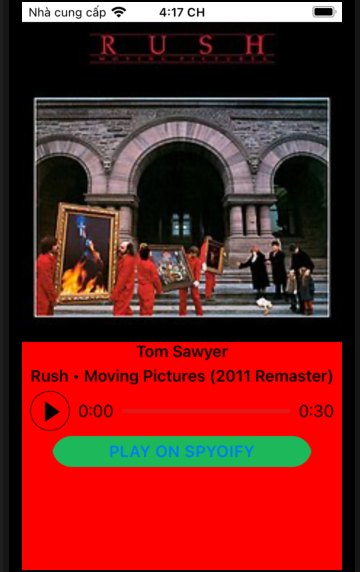
**Tóm lại:**

* **Content Hugging càng lớn: Thì khả năng size view không bị scale lớn hơn size content càng lớn**
* **Compression Resistance càng lớn : Thì khả năng size view không bị thu nhỏ hơn size content càng lớn**

# 5.Stack View

* Đầu tiên Stackview chỉ là 1 cái Container chứa các subviews bên trong và nó chưa có x,y , width, height.
  + Ít nhất ta vẫn cần xét x,y cho stackview.
* Nếu ta chưa constrain width, height cho stackview, thì size của stackview sẽ phụ thuộc vào size của content bên trong.
  + Chiều cao(height) sẽ bằng tổng số size được sắp xếp dọc theo stackview + với space giữa chung
  + Chiều rộng(width) sẽ bằng size lớn nhất(mà nằm theo trục x ý) bên trong stackview.
* Nếu ta constrain height và width thì size của StackView sẽ **luôn là giá trị ta xét.**
* Khi các subviews được add vào Stackview, layout của các subviews đó sẽ tuân theo các properties có sẵn trong stackview(sau đó khi nào ta cần thì ta thay đổi thì sửa value default đó sau) đây là 1 vài default value:
  + Axis: .vertical
  + Distribution: .fill
  + Alignment: .fill
  + Spacing: 0
  + LayoutMarginRelativeArrengment: false 🡪 Nếu xét thành true thì subviews sẽ được layout theo Margin, nghĩa là có margin khoảng cách giữa các Edge Stackview đó

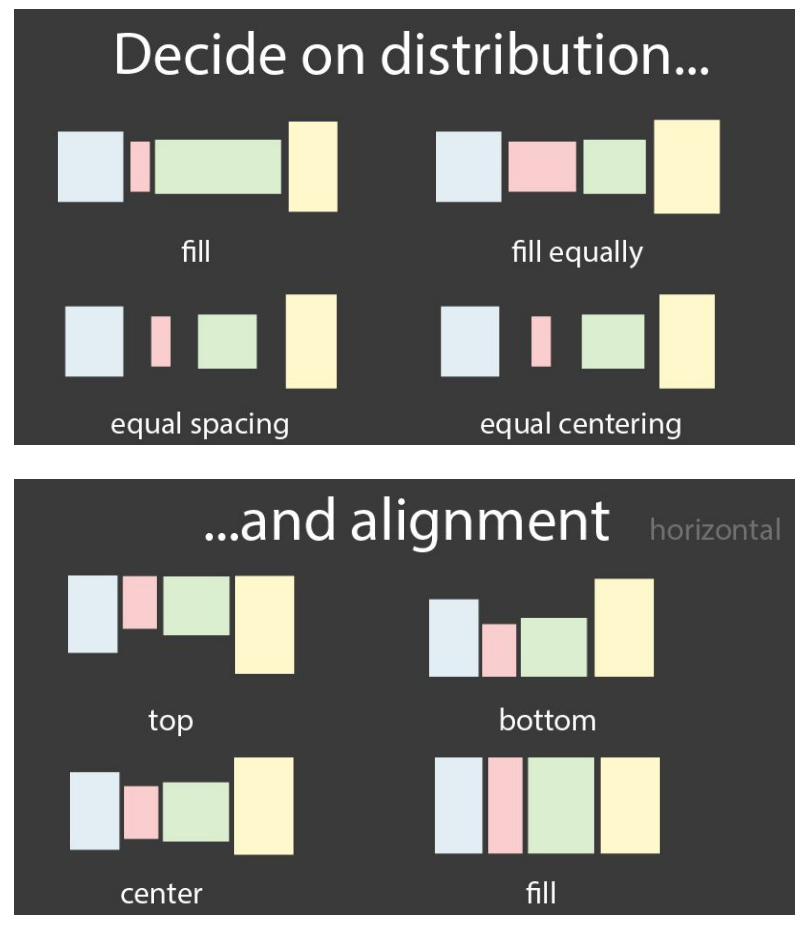
Vì vậy khi ta mới add các subviews vào stackview thì các subviews đó đã được layout theo kiểu Distribution và Alignment là fill rồi 😊))) Nên đừng hỏi tại sao vừa vào nó đã vừa khít.

* Các chú ý khi layout bằng code
* Như đã đề cập bên trên, khi chưa xét gì cho stackview thì Distribution và Alignment đều đã là fill
* Ta gặp 2 trường hợp như này:
  + Khi constrain neo cả 4 góc cho stack view, hay xác định width height cho stackview, sau đó ta add view, label vào stackview thì lúc này stackview **sẽ xóa hết các constrain của view và label ta xét trước đấy**. Dẫn tới lúc này cả label, view chưa được constrain gì, ta bắt buộc phải constrain cho các thuộc tính bên trong đó, như xét height cho view. Nếu ta không constrain gì cho view bên trong, thì vì lúc này size của view và label chưa được xét, stackview chỉ biết làm thế nào cho view và label full màn hình chứ không xác định được height cho từng vật, dẫn tới tình trạng **Ambigous.** Nếu ta xét distribution là fillqually thì có thể ok thôi, vì nó biết chia đều mà
  + Vẫn là constrain cả 4 góc cho stackview, ta add 1 view A chứa 1 view B vào stackview, thì chỉ constrain cho view chứa là A bị xóa(như là height, width, xy, y của A) và quan hệ giữa B và A sẽ không bị xóa. Ta có thể dùng đặc điểm này để constrain nhiều việc.

Ta xem ví dụ bên cạnh, phần bên dưới không hề full, nếu ta add 1 stack nhỏ chứa các label như Tom Sawyer, Rush,… vào stackview chính thì bắt buộc nó **sẽ phải thay đổi để full hết bên đưới.** Vậy để nó không bị thay đổi ta sẽ add stackview nhỏ đó vào view bị constrain 3 góc(dưới trình bày) rồi add view đó vào stackview chính, dẫn tới chỉ thằng view đó bị ảnh hưởng là phải full, còn thằng stackview bên trong không ảnh hưởng gì.

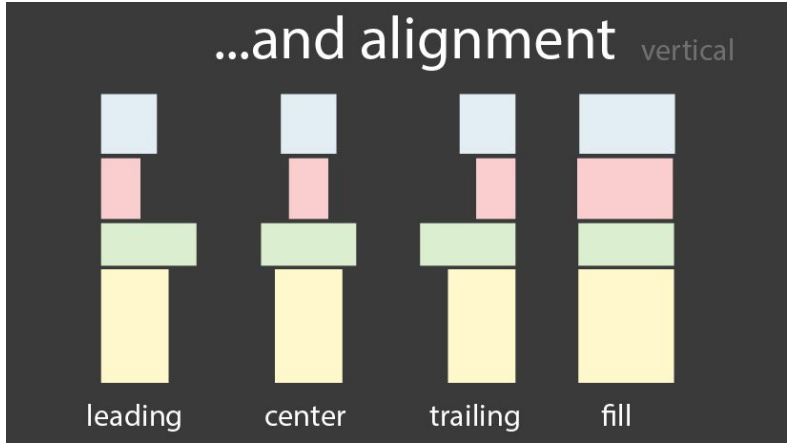
* + Khi ta không constrain cả 4 góc, mà constrain 3 góc như top, left, right để bottom không xác định. Lúc này ta add view và label vào thì stackview sẽ không xóa constrain, height trước đó của view và label nên height của **nó vẫn được giữ nguyên**. Height của stackview lúc này sẽ là height và label và spacing cộng lại.

5.1 Distribution và Alidnment theo kiểu Horizonal StackView



5.1 Distribution và Alidnment theo kiểu Vertical StackView

* Với Alignment: vẫn theo trục dọc, giống trên khá dễ



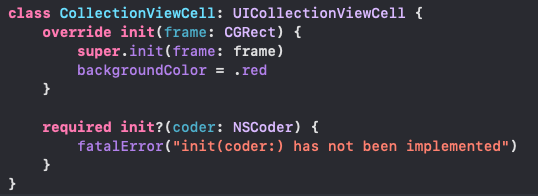
* Với Distribution
* **fill**: The views are stretched to fill the entire stack view.
* **fillEqually**: ***Chiều cao của các views bằng nhau và bằng cái to nhất(maybe to nhất).***
* **fillProportionally**: Thay đổi kích thước của các view trong stack view sao cho chúng lấp đầy các không gian có sẵn 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.
* **equalSpacing**: Khoảng cách bằng nhau
* **equalCentering**: Khoảng cách từ trục giữa các view đến nhau bằng nhau

# V. Collection View

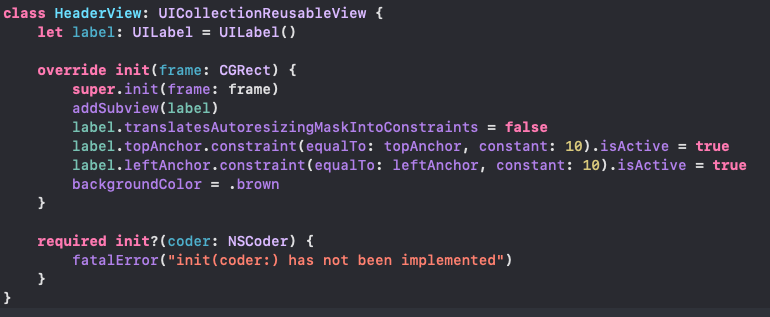
* UICollectionViewFlowLayout: là 1 class cho phép layout các item theo các ô lưới(kiểu hình nhữ chất đó 😊)) với các optional Footer and Header. Nhìn chung nhiệm vụ chính của thằng này hỗ trợ các hàm và thuộc tính để có thể thêm Footer, Header, cấu hình height cho Header, Footer và xét khoảng cách giữa Header và Footer với các line bên trong. Bên cạnh đó UICollectionViewFlowLayout còn có các thuộc tính để xét spacing giữa các Line trong 1 collectionView, xét spacing của các elements trong cùng 1 line.
* UICollectionViewDataSource và UICollectionViewDelegate: Nhiệm vụ chính là xét bao nhiêu element, và cấu hình cell cho collectionView, cấu hình nội dung cho Header và Footer.

## 5.1 Các bước thực hiện

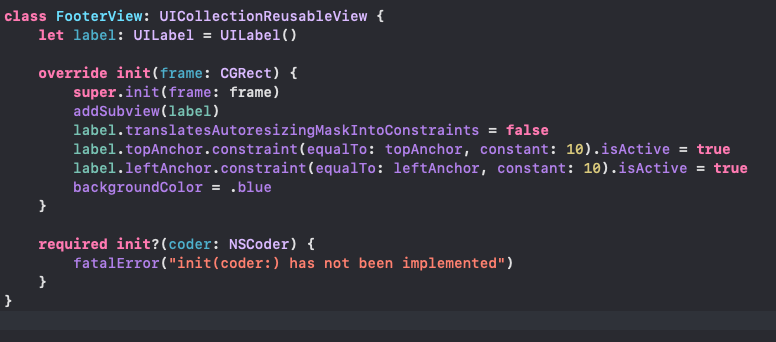
* Ta sẽ triển khai các class Cell, Footer, Header để sử dụng
* Cell



* Header: Type UICollectionReuableView



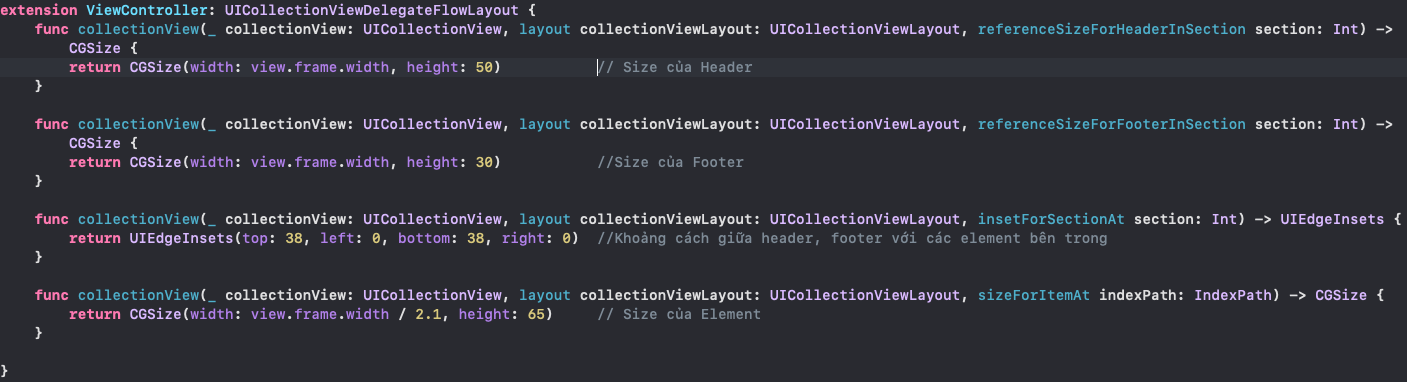
* Footer: Type UICollectionReuableViews



Tiến Hành triển khải code bên trong ViewController

# Không có mô tả.

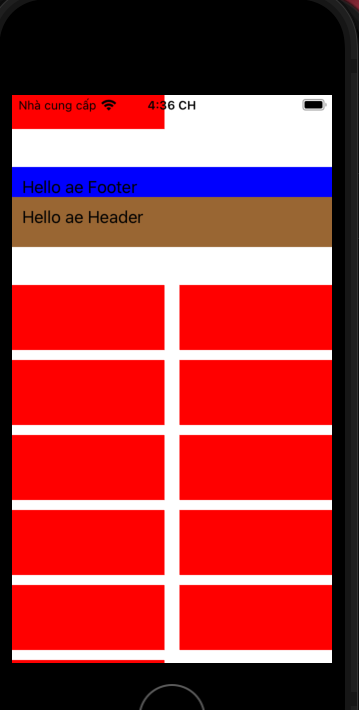
* Khởi tạo 1 instance thuộc kiểu **UICollectionViewFlowLayout**()
* xét scrollDirection là vertical: để ta cuộn theo chiều dọc
* minmumLineSpacing: Là khoảng cách giữa các Line (sẽ bị ghi đẻ bởi method)
* minimumInheritSpacing: Là spacing của các Element trong cùng 1 Line (sẽ bị ghi đẻ bởi method)
* footerReferenceSize: Là size của footer (sẽ bị ghi đè bởi method)
* headerReferenceSize: Là size của header (sẽ bị ghi đẻ bởi method)
* Khởi tạo 1 instance UICollectionView
* Đăng kí cell sử dụng với Class chỉ định kèm với identifier của class đăng kí đó.
* Đăng kí Header, Footer với các Class và identifier chỉ định
* Delegate
* Cấu hình size footer, header, spacing giữa các Line và spacing giữa các elements trong 1 Line



* Cấu hình nội dung cho cell, footer và header

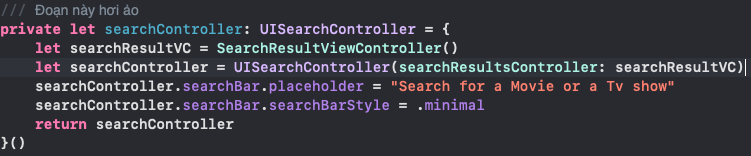


Output:



# VI. SearchController

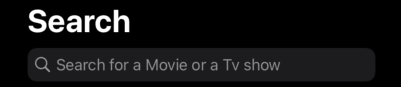
Khởi tạo 1 biến SearchController, sau khi tìm kiếm trên thanh search, nó sẽ đẩy vào Controller khác, ở vd này là searchResultVC.



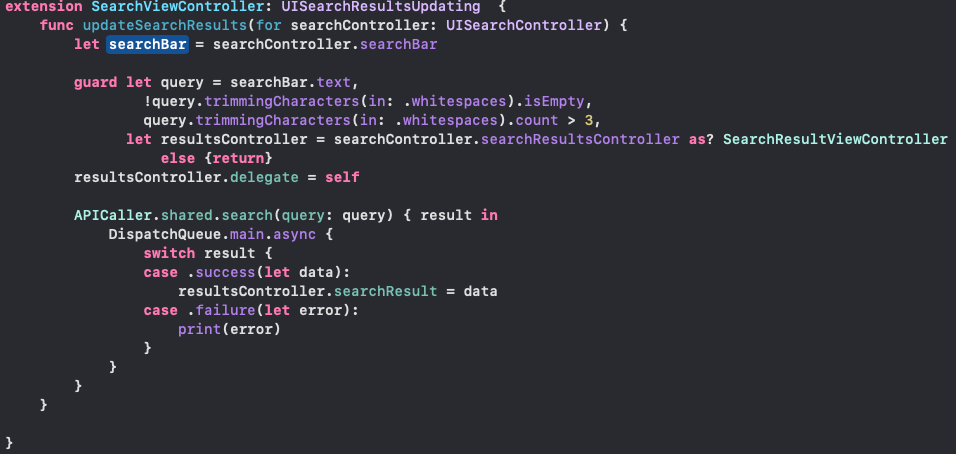
Sau khi khởi tạo xong, ta sẽ gán searchController đó vào navigationItem:

Không có mô tả.

Output:

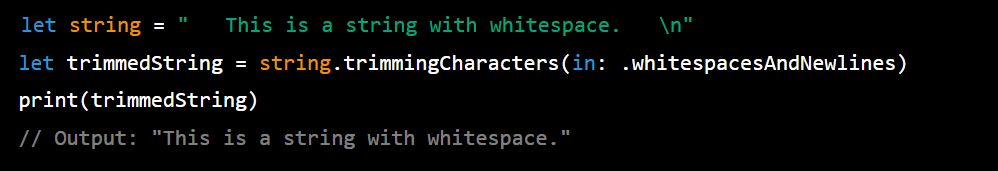


Để có thể lấy được các thông tin khi người dùng nhập search ta delegate:

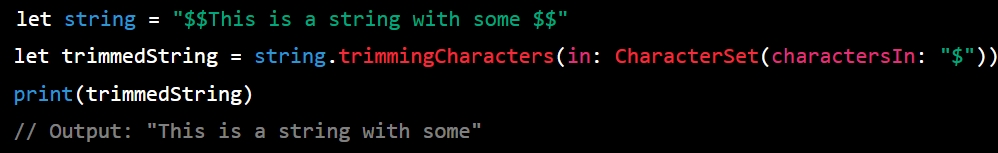


Ta được searchBar.text là lấy được thông tin người dùng tìm kiếm.

* Ta có .trimmingCharacter: Ở đây có tác dụng xóa hết các space trắng ở đầu và cuối String đó



You can also specify a custom character set to trim, by creating a **CharacterSet** instance with the desired characters. For example:



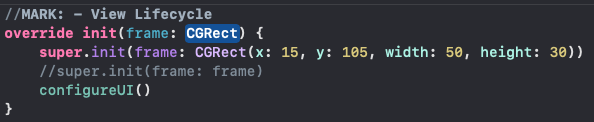
Note that **trimmingCharacters(in:)** does not modify the original string, but instead returns a new string with the desired characters removed.

# VII. Bankey Project

7.1 Intrinsic Contensize

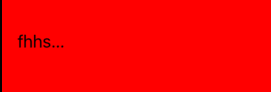


Đầu tiên, khi ta không sử dụng **constraint như set x, or y, or width,…** thì giá trị của intrinsic content size bằng bao nhiêu cũng sẽ ko có ý nghĩa gì, nó sẽ luôn sử dụng giá trị của frame.

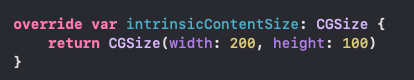


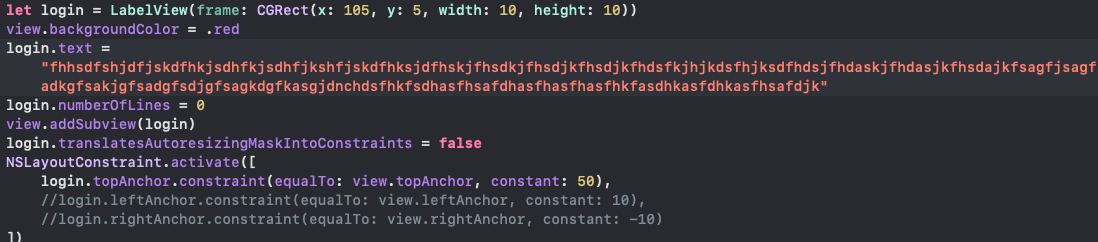
Với ví dụ này khi ko sử dụng constraint thì frame của nó luôn bằng CGRect(x: 15, y: 105, width: 50, height: 30).

Kết quả khi ko sử dụng constraint:



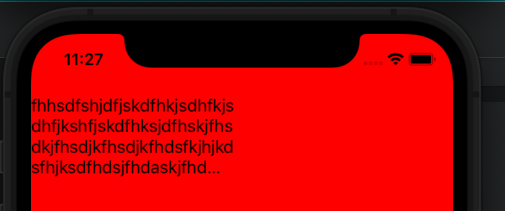
Nhưng khi override **instrictContentSize** và sử dụng **Constraint** cho biến từ instance của class đó. Thì hệ thống sẽ sử dụng giá trị của **instrictContentSize để quyết định size cho label**



**VD1:** 

Ở VD1, ta chỉ **constrain mỗi Top và không động vào width và height** thì **width và height của Label lúc này sẽ luôn luôn bằng giá trị ta xét trong intrinsicContentSize.**

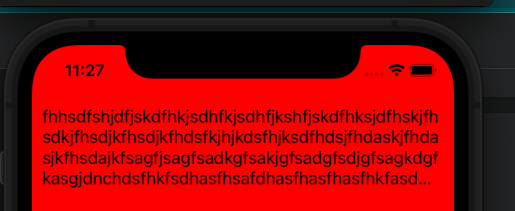
**Kết quả VD1:**



**Ta có VD2:**



**Lúc này khi ta constrain cả width cho View nữa nên dẫn tới Width sẽ được thay đổi theo constrain, và vì ta ko constrain cho height của View nên Height của view vẫn sẽ được giữ theo height của IntrinsictContenSize và là 100, kết quả:**

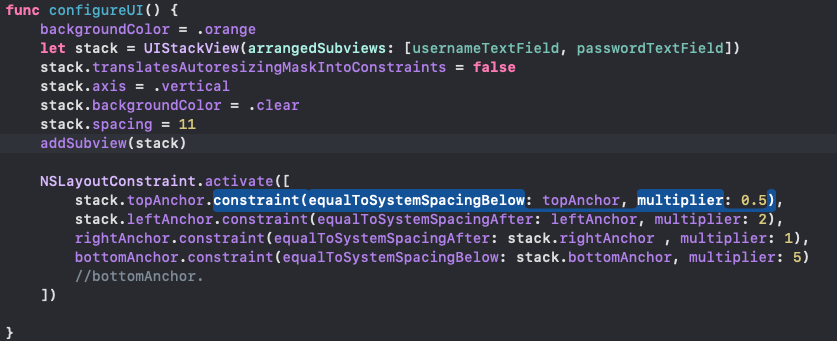


## 7.2 Constrain kiểu 2

Ta có 1 container View chưa 1 View bên trong. Giả sử đó là textField, vì ta có font chữ của TextField nên hệ thống sẽ tự tính được size theo intrinsic contentsize. Nên đôi khi ta ko muốn xét hết height width cho cả Container View đó, mà muốn View đó bám theo size của textField, vì lúc này textField đã có size rồi.

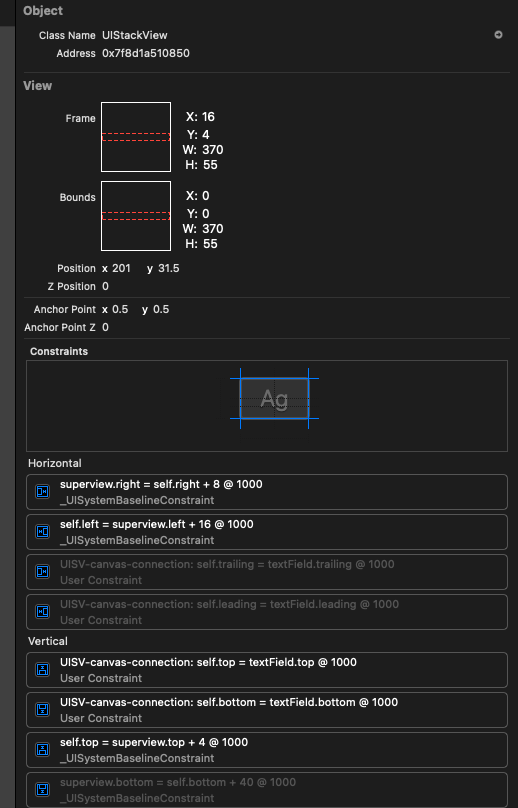
Ta có 2 cách:

**Cách 1:**



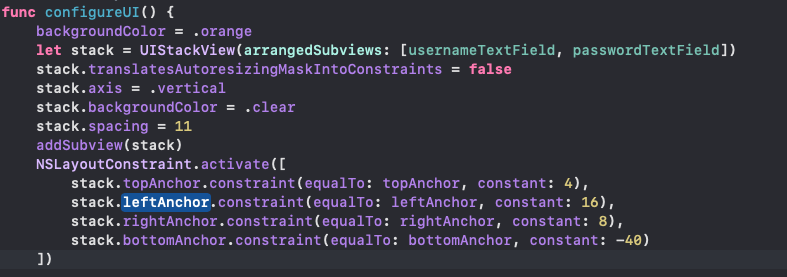
Với giá trị **equalToSystemSpacingBelow và equalToSystemSpacingAfter đều là 8**

Dẫn tới ta xem bảng sau đây:



Constraint của nó stack so với View là: cách left 16 (vì multiplier = 2), cách top 4(vì multiplier = 0.5), …

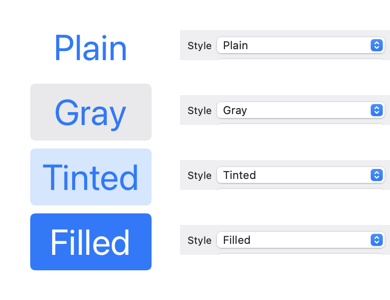
**Cách 2: Ta cũng được kết quả tương tự**

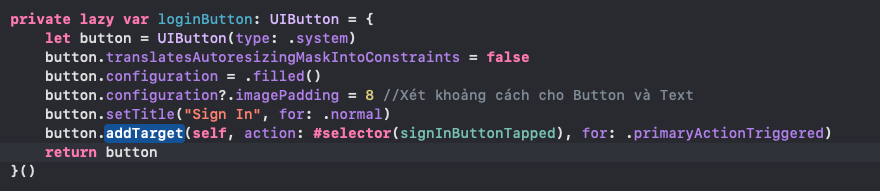


Ta sẽ làm 4 đầu của container sẽ được bám theo 4 đầu của stack, vì size của stack đã được xác định do intrinsic content size rồi nên size **container view cũng sẽ được xác đinh.**

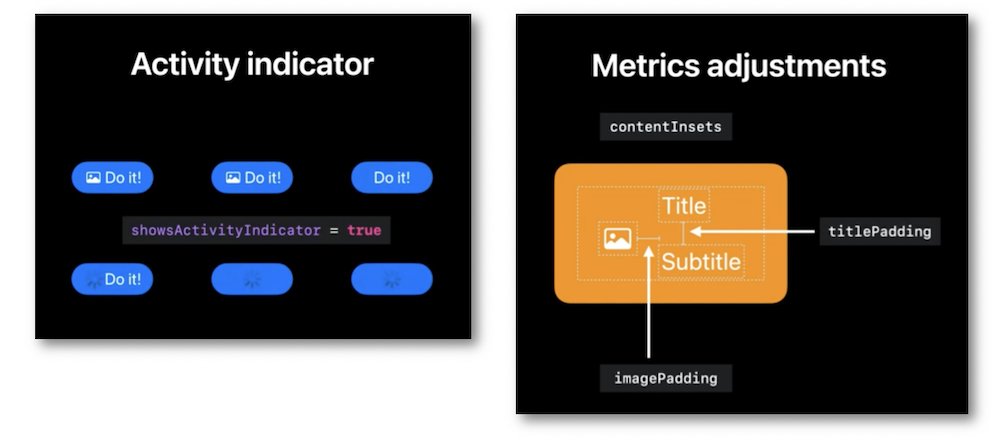
## 7.3 Type Button

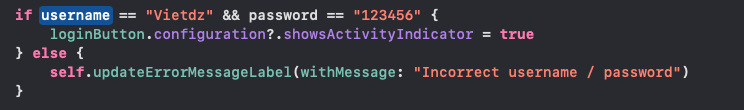
Các kiểu Button:



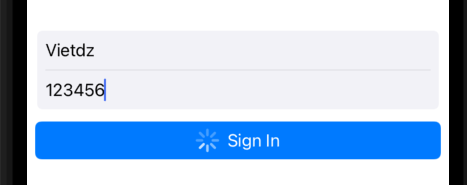


Khi xét **configure.imagePadding = 8** thì khoảng cách của nó so với **image hay so với indicator sẽ bằng 8:**



Và 

Kết quả thu được:



## 7.4 Page View Controller