1. Behind the Scenes of Custom Transitions

UIKit cho phép ta customize sự xuất hiện của 1 Controller thông qua delegate. Ta chỉ cần cho main controller của chúng ta comform protocol **UIViewControllerTransitioningDelegate.(**Main Controller là kiểu controller có collectionView mà chứa các cell, sau khi tích vào cell thì xuất hiện màn DetailController**).**

Mỗi khi ta present 1 Controller, UIKit hỏi delegate kiệu rằng có nên sử dụng custom transition hay không.

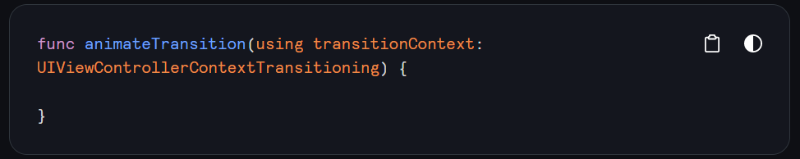
UIKit gọi phương thức **animationController(forPresented:presenting:source:)** để kiểm tra nó có trả về một **UIViewControllerAnimatedTransitioning** object hay không. Nếu phương thức return nil, UIkit sẽ sử dụng build-in transaction. Nếu phương thức return 1 UIViewControllerAnimatedTransitioning **object, UIKit sẽ sử dụng đối tượng đó như là một “animation controller” cho việc transition.**

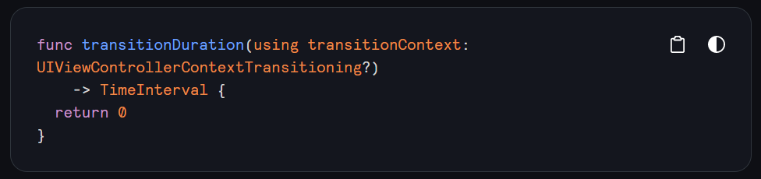
Đầu tiên UIKit sẽ gọi phương thức **animateTransition(using:) của animation controller(animation controller** còn được gọi là **animator).** Trong **animateTransition(using:)** ta sẽ truy cập cả current View Controller và Controller mới mà sẽ được presented. Từ đây ta có thể fade, scale, rotate các View Controller đó.

1. Implementing Transition Delegates

Từ phần trên có thể thấy ta cần một object thuộc kiểu **UIViewControllerAnimatedTransitioning** để quản lý việc animation, nền đầu tiên ta sẽ tạo 1 class mới kế thừa class **UIViewControllerAnimatedTransitioning và NSObject.**

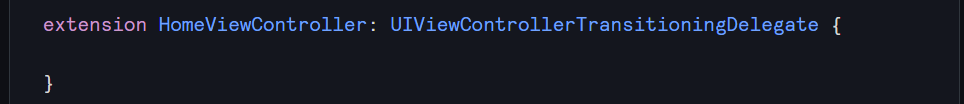


 Từ đây, Xcode sẽ báo lỗi, và ta phải comform 2 phương thức này:



1. Wiring Up the Delegate

Đầu tiên ở Main Controller chúng phải kế thừa Protocol **UIViewControllerTransitioningDelegate.**



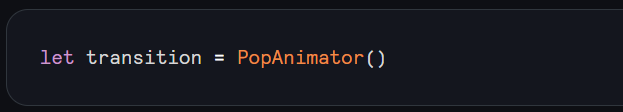
Sau đó, ở Controller mới mà sẽ được animation khi xuất hiện cần delegate transitioningDelegate.



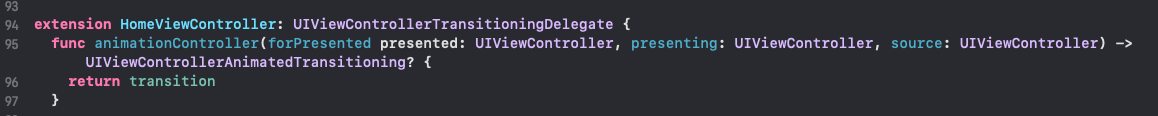
Từ lúc này, UIKit sẽ hỏi HomeViewController cho **animator object** mỗi lần **detail view controller sẽ xuất hiện.**

1. Using Animator.

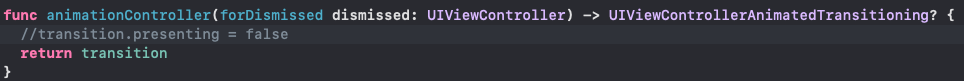
Ta sẽ tạo một instance của PopAnimator ở trong HomeViewController



Từ đây mỗi khi Detail Controller xuất hiện, nó sẽ gọi hàm dưới đây, như ở bên trên, nếu hàm này return nil, thì sẽ sử dụng build-in transition, nếu return 1 object transition, thì sẽ sử dụng đối tượng đó.



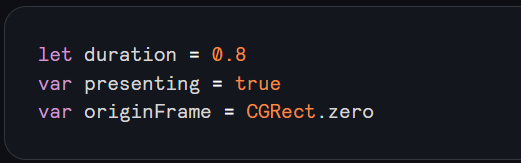
Tương tự cho việc dismiss Detail View Controller.



1. Creating Transition Animator

Các phần trước, ta đã giao việc **animation transition cho** thằng object transition, ở phần này ta sẽ custom class của thằng đối tượng đó.

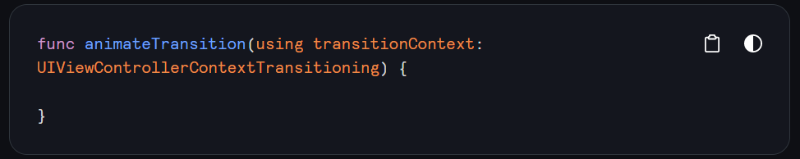
Ta khai báo các thuộc tính này:



* duration: được sử dụng để nói cho UIKit việc transition này được thực hiện trong bao nhiêu
* presenting: Để thông báo liệu ta đang presenting hay dismissing 1 controller.
* originFrame:

Sau đó ta trong hàm **transitionDuration(using:) ta sẽ return duration**

1. Setting your Transition’s Context

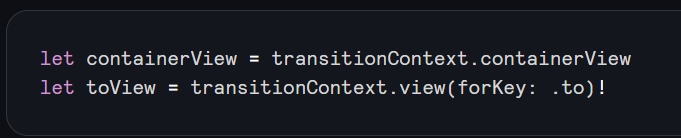
Ta sẽ code 1 vài magic cho phương thức animateTransition(using:), phương thức này có 1 parameter thuộc kiểu **UIViewControllerContextTransitioning.** Thông qua parameter này ta sẽ có thể truy cập vào các **view controllers** của transition.

Đầu tiên ta phải hiểu **animation context** là gì:

* Khi việc transition giữa 2 Controller bắt đầu, cả Main View Controller(gọi là old view) **và** view của View Controller mới(gọi là new view) sẽ được thêm vào **trainsition container view. Lúc này new View controller** đã được tạo nhưng vẫn chưa thể nhìn thấy.
* Trong quá trình transition, hệ thống sẽ kiểm soát view của old view và new view trong **transition container**, nó sẽ tạo ra hiệu ứng animation giữa 2 controller. Khi transition complete, new view controller sẽ được visible, và old controller sẽ được xóa khỏi transition container view

1. Adding an Expand Transition

Ta sẽ tạo 2 biến mới từ **animation context:**

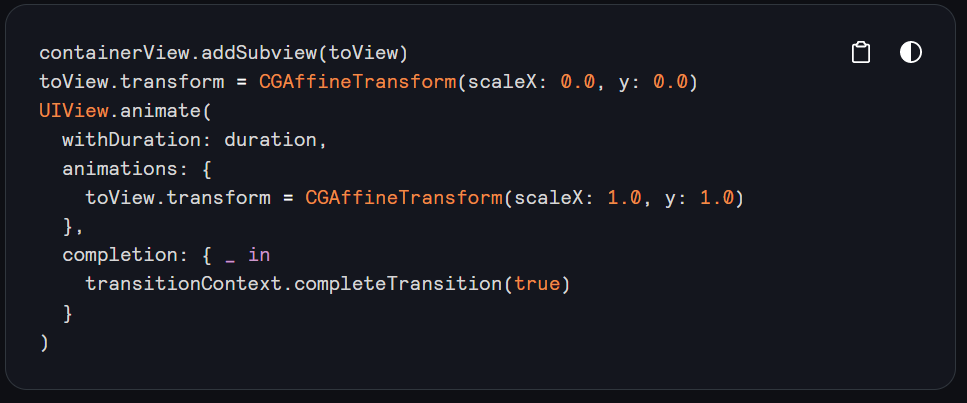
****

* Đầu tiên ta tạo **container view** từ transition context, container view là nơi mà animation sẽ xảy ra. (Như đã trình bày bên trên).
* Khi có **container view,** ta sẽ fetch new view, và sẽ lưu nó trong **toView.**

**Transition context** cung cấp cho ta 2 phương thức để có thể truy cập **new controller và old controller:**

* **view(forKey: ):** Cho phép ta truy cập vào các **views của new view và old view** thông qua argument **UITransitionContextViewKey**.**from** or **UITransitionContextViewKey**.**to** (Cái này là truy cập vào **views**)
* **viewController(forKey):** Cho phép ta truy cập vào **new controller và old controller** thông qua **argument UITransitionContextViewKey.from or UITransitionContextViewKey.to (**Cái này là truy cập vào **Controller)**
  + Tương tự cho bên trên

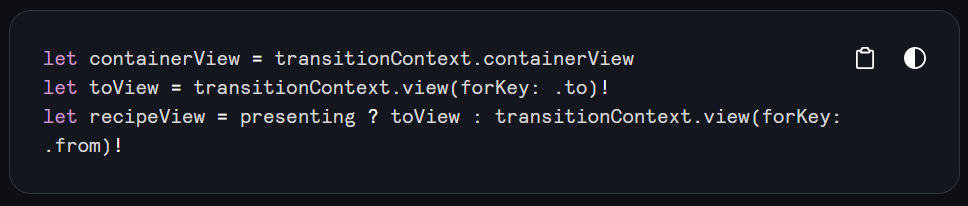
Ở phần trên, ta đã lấy **được view mới mà sẽ được present** ở toView . Tiếp theo, ta cần add view mới đó vào **container view**.



Note that you call completeTransition(\_:) on the transition context in the animation completion block. This tells UIKit that your transition animations are done and that UIKit is free to wrap up the view controller transition.

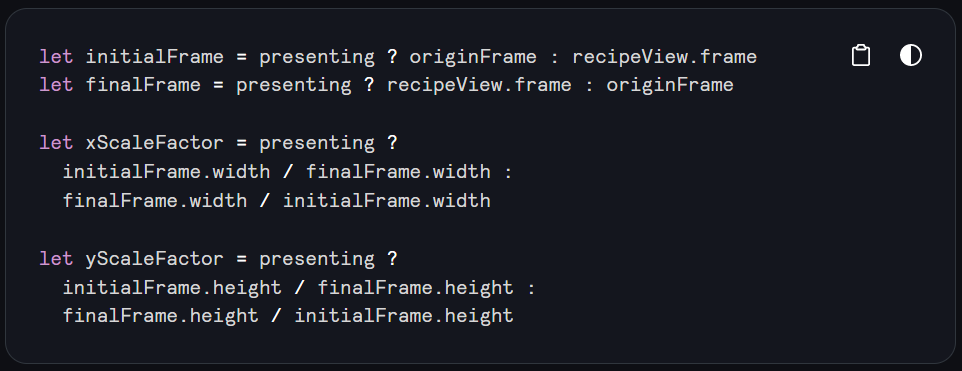
1. Adding a Pop Transition

Ta sẽ tiếp tục structure code

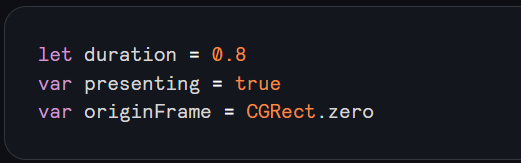


* Như đã nói: **containerView** là nơi animation sẽ xảy ra
* **toView**: là new view mà sẽ được presented(ở đây new View sẽ được presented có thể là Main Controller hoặc Detail Controller)
* Nếu đang presenting, **recipeView** sẽ được gán là toView, tức là view mới sắp được xuất hiện. Nếu đang dismissing, **recipeView** sẽ được gán là **view mà đang được dismissed và trong trường hợp này là detail controller.**

Ta biết rằng, mỗi khi presenting, View sẽ được mở rộng ra, khi dismissed, View sẽ được co lại, đoạn code này thực thi việc đó:

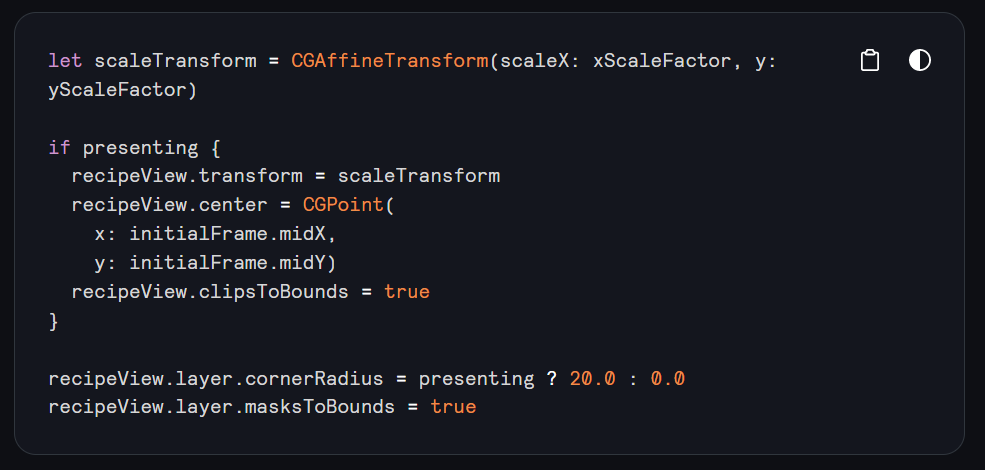


**Nhắc lại**: Ở đầu ta có khai báo các thuộc tính



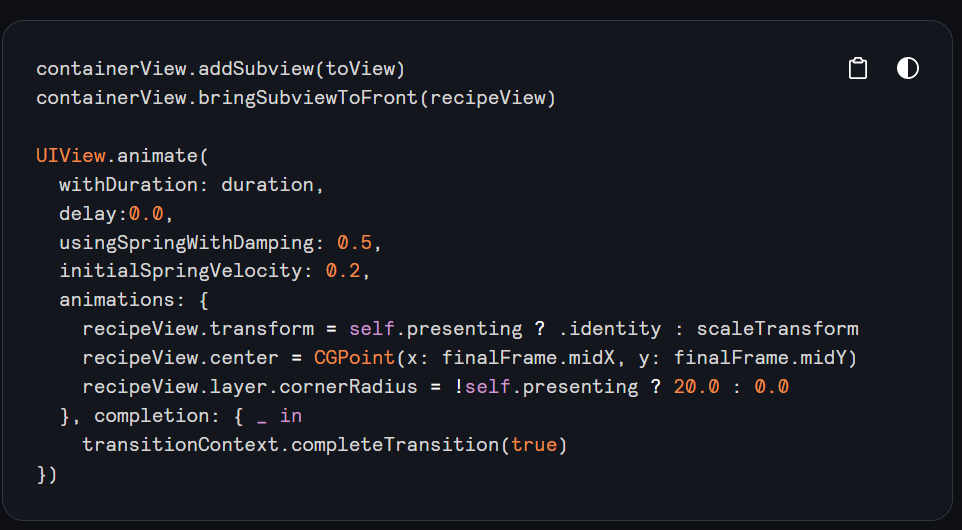
1. Scaling The View

Ta sử dụng CGAffineTransform() để scale view



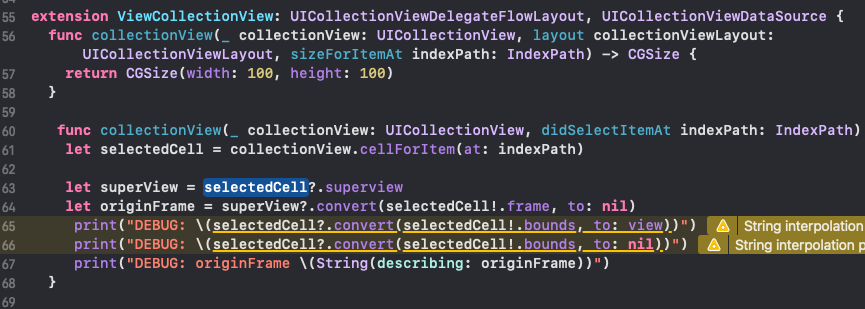
Sau khi scale recipeView, ta sẽ add lần lượt toViewvà recipeViewvào container view. **Ta nhớ rằng recipeView luôn là view sẽ xuất hiện, và là view sẽ được animating, vậy nên recipeView luôn là view ở top của container.**

Sau đó ta sẽ thêm hiệu ứng animation, và xét vị trí cho recipeView.



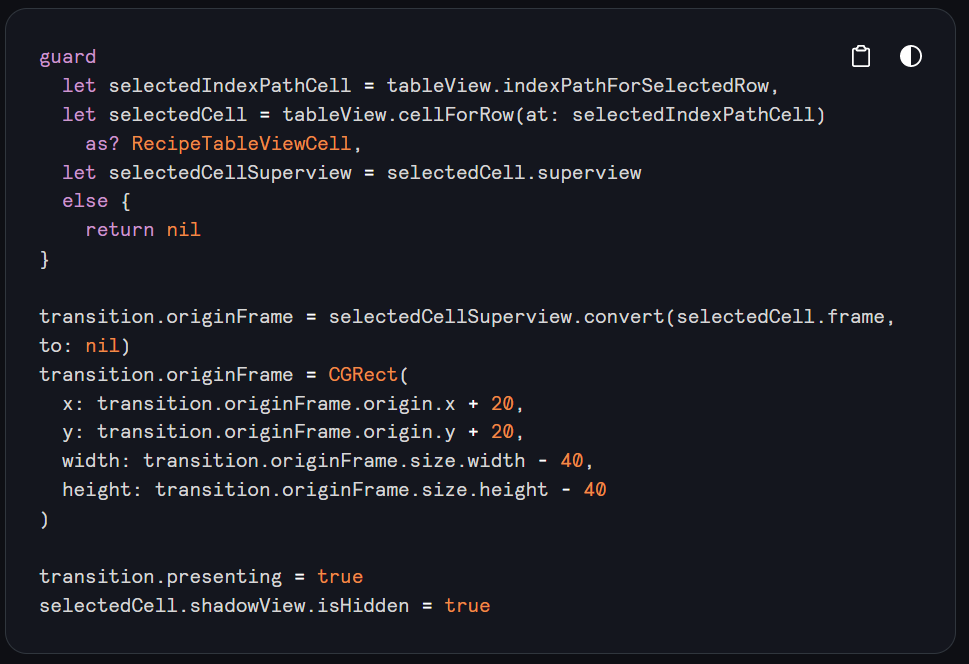
1. Adding Some Polish

Nhiệm vụ bây giờ ta phải xác định được position của Cell ta nhấn vào để tới Detail Controller(ví trị của cell chính là **originFrame**), ta làm như sau:



Ở đây 3 hàm này ta đều sẽ lấy được frame của cell so với Window. Từ đây nhiệm vụ của ta là chỉnh lại position giữa frame và contentView bên trong View. Vì contentView sẽ còn cách trái, cách phải so với frame nữa.

Với bài này, contentView chứa ảnh và được constraint cách top, bottom, left, right 20 pixel, nên cần làm thế này



**Tổng kết:**

Khi 1 Controller chuẩn bị được present, thì hiệu ứng chuyển đổi giữa Controller cũ và Controller mới có thể xảy ra theo 2 trường hợp. Trường hợp 1 là ta sử dụng build in transaction thì animation transaction là default apple cung cấp. Trường hợp 2 là ta tự custom animaion transition.

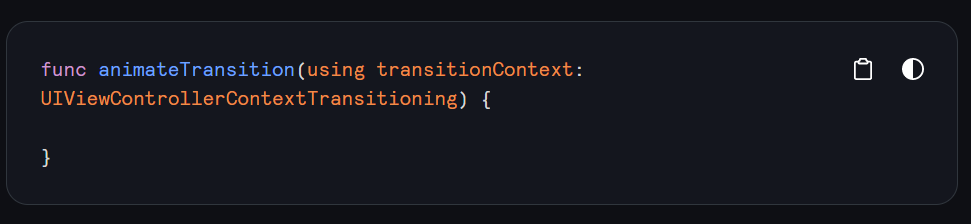
Ta sẽ có 2 Controller:

* **Main** **Controller**: Chứa các cell, image 🡪 View của Main Controller gọi là **mainView**
* **Detail Controller:** Đây là khi được present sẽ có hiệu ứng animation transaction 🡪 View của Detail Controller gọi là **detailView**

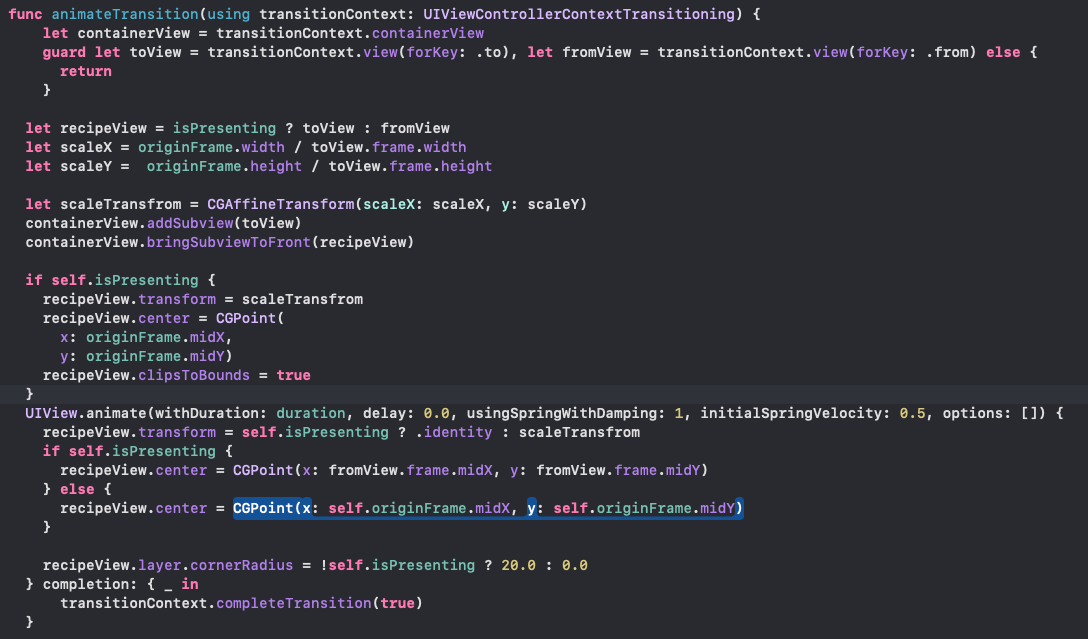
Các bước:

* Khi ta present Detail Controller, thì Detail Controller sẽ hỏi Main Controller liệu rằng nó sẽ sử dụng build in transitions hay custom transitions, nếu hàm animateTransition(using:) trả về nil thì sẽ sử dụng build in, không thì sử dụng custom transition
* Từ đây nhiệm vụ của ta là custom sự xuất hiện của Detail Controller thông qua Transition Object

Ta có hàm quan trọng nhất là hàm dưới đây, một vài điểm cần hiểu rõ



* Ta biết rằng 1 View Controller khi được present luôn đã xác định được **frame** thông qua thuộc tính **modalPresent,** nếu là fullscreen thì là size view bằng size của window.
* Ta cũng biết rằng, thằng **transitionContext** chứa 1 thằng **containerView** và từ **transitionContext** ta sẽ lấy được **toView** và **fromView.** Như bên trên đã nói, vì size của các View khi sắp được present đã được xác định rồi, nên size **toView** và **fromView** hiển nhiên đã xác định, và trong ví dụ trên, ta present fullscreen trên iphone 11, nên frame của cả 2 View này là (0, 0, 414, 896).
* Đầu tiên, thằng **fromView** đã được tồn tại bên trong **containerView**, nhiệm vụ của ta là chỉ cần add thằng **toView** vào **containerView,** thì user lúc này từ sceen đã thấy được **toView** đó. Nên nhớ rằng khi ta add **toView** vào container, lúc này view của toView sẽ fullscreen như đã trình bày bên trên. Cuối cùng ta phải gọi hàm **transitionContext.completion()** để thông báo đã xong quá trình animation transition, người dùng có thể tương tác với screen.
* Về cơ bản đã xong, nhưng ta chưa thấy 1 animation nào cả, bên dưới sẽ tiếp tục thực hiện điều này:
* Tưởng tượng quá trính animation bây giờ là: Bây giờ ta đang ở **Main Controller,**  và ta có 1 Cell, tích vào Cell đấy thì Cell sẽ mở rộng ra. Nhưng đã nói bên trên, khi ta **add toView into containerView,** thì lúc này **toView** đã có size toàn bộ màn hình rồi nên khi nó xuất hiện sẽ không có tí animation nào, nên nhiệm vụ của ta lúc này đó là thu nhỏ size của **toView** về đúng size của **cell**, sau đó ta sẽ phóng to lại view và chèn **animation** vào. Nhìn đoạn code dưới đây thì sẽ hiểu:



* + Ta thấy có đoạn **if self.isPresenting {}** chính là đoạn: nếu ta đang muốn present **detailView** thì ta cần thu nhỏ view của **toView** về trước đã, xong tiếp tục ta phải chỉnh vị trí của **toView** về đúng vị trí cell, xong rồi mới phóng to lại.
  + Ta hiểu rằng **scaleX và scaleY** là giá trị thu nhỏ lại để về đúng size ban đầu của cell.
  + Còn trong trường hợp dismiss **detailView,** thì size hiện tại đang hiện trên sceen của **detailView hiển nhiên sẽ fullscreen** là đúng rồi, chỉ cần thu nhỏ lại đi là được.