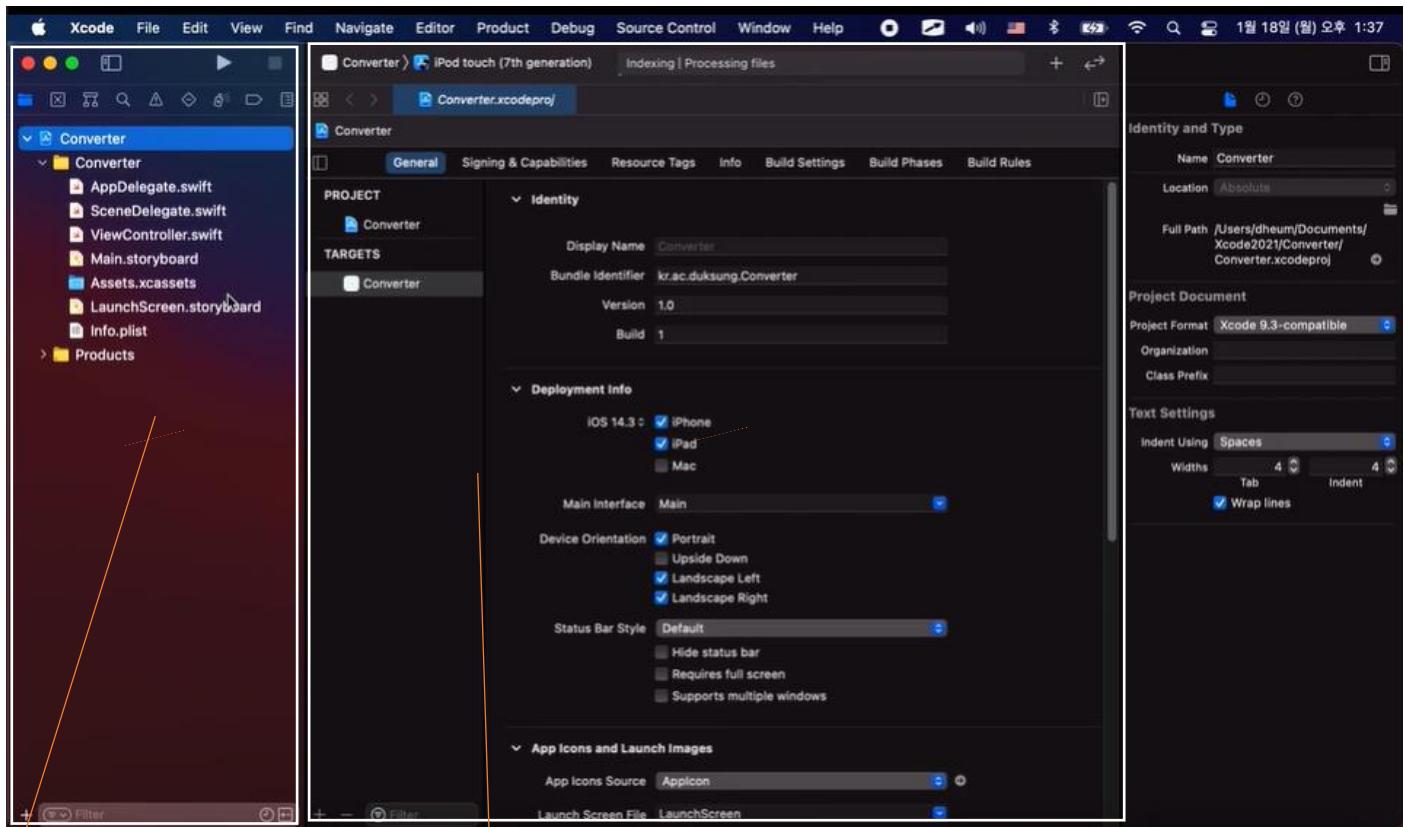


-> include test 오류를 잡기 위한 extra code 생성 유무를 묻는 것이다.

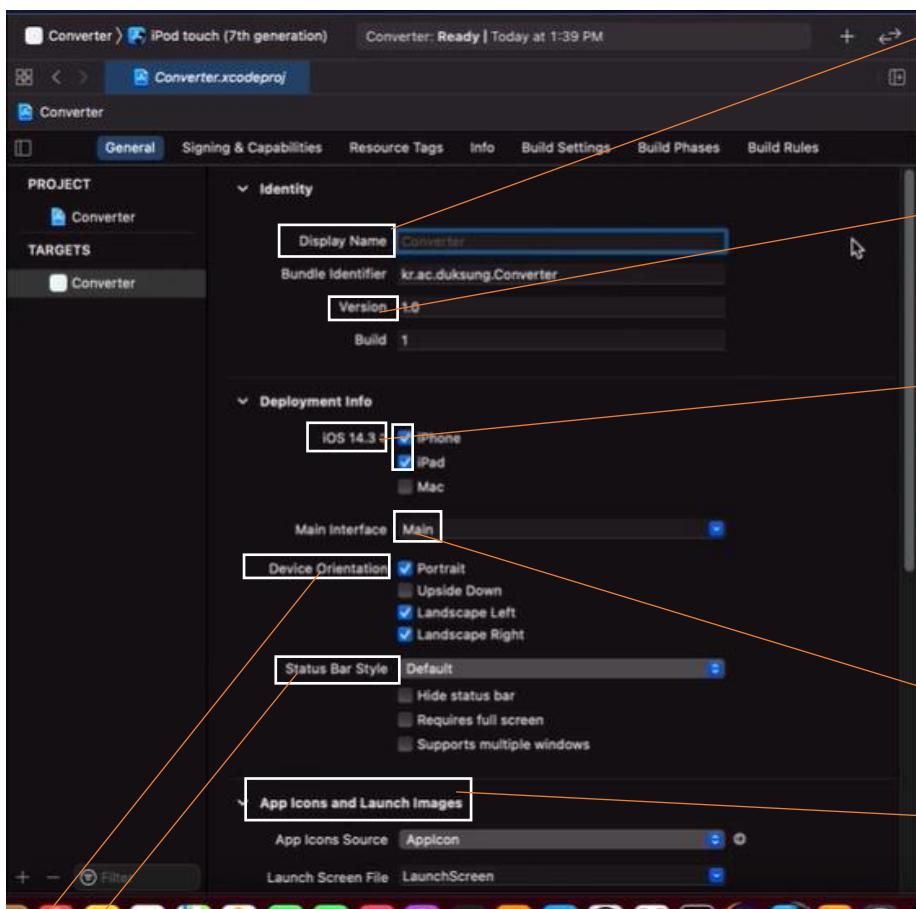
Interface 도구로는 storyboard랑 SwiftUI가 있는 데 SwiftUI는 생긴 지 1년 밖에 되지 않으며 성능을 좋게 만들기

위해 만들었다. 그러나 storyboard 방식이 기본이기 때문에 storyboard 방식으로 선택한다.



프로젝트 탐색기 창

프로젝트 summary 창



Display Name: 아이폰 바탕 화면 아이콘 밑에 들어갈 이름을 설정하는 것이다. Ex)화씨변환기

version: 개발하려는 앱의 버전이다. 기능을 더 넣으면 version 2.0이 되기가 된다.

iOS가 어디부터 지원을 할 것인지를 설정하는 것이다. 지금은 iOS 14.3 버전부터 사용 가능하도록 설정했다. iPhone 용도인지 iPad 용도인지를 체크할 수 있는 것으로, 지금은 iPhone 용도로만 되도록 체크한다.

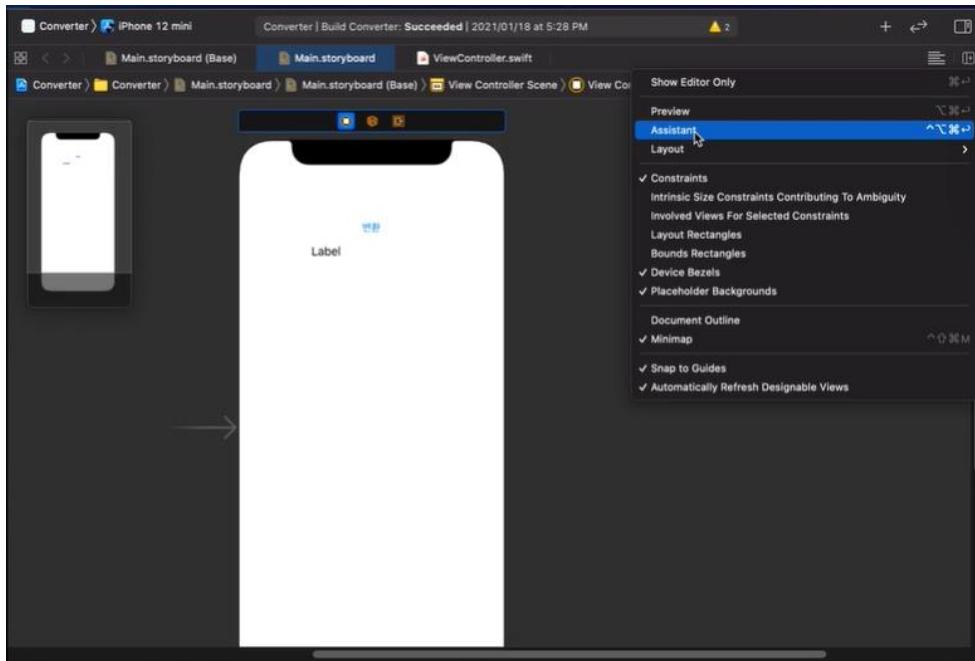
Main.storyboard 파일의 이름을 바꿀 것인지를 설정하는 것이다.

App Icon and Launch Images 앱 아이콘 이미지를 등록하는 것이다.

Device Orientation은 가로, 세로 전환이 되는 지의 여부를 체크하는 것이다.

Status Bar Style은 아이폰 위에 있는 와이파이 세기를 나타내는 것이다.

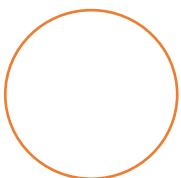
25분 부분 //// -> 시간이 되면 추가하는 것으로....



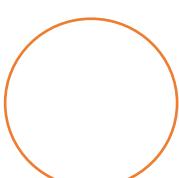
Assistant 클릭 시

viewController.swift 파일이 나오게 된다.

App



windows



View Controller -> viewController 클래스로부터 생성된다.

App 객체에 의해서 View Controller 객체가 생성되면, viewController 클래스로부터 생성

class는 init()이라는 초기화 함수를 자동 생성이 되며, 이때 파라미터를 취하지 않게 된다.

init(){

fahrenheitTextField = nil

celsiusLabel = nil

}

fahrenheitTextField와 celsiusLabel은 옵셔널 변수이지만, 이중방 형태는 아니다. 옵셔널 변수이기 때문에 nil이 들어갈 수 있다.

app 객체가 그 다음으로는 manage해야 할 view 객체를 생성한다.

View 객체를 viewController 객체와 연결하고 viewController는 view 객체를 manage를 한다.

코드에는 직접으로 없는 이유는 UIViewController 클래스에서 view라는 필드가 있고 ViewController는 이를 상속받기 때문이다.

그 다음에 app 객체는 main.storyboard 파일에 간다. -> UI를 배치한다.

```
import UIKit

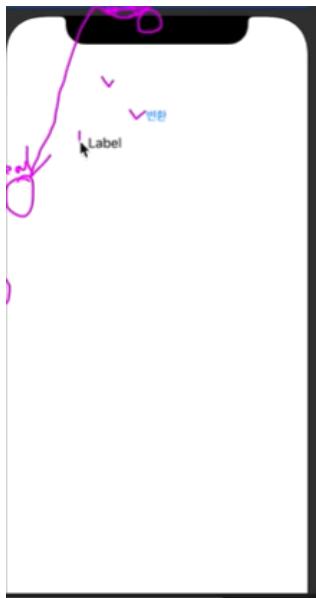
class ViewController: UIViewController {

    @IBOutlet var fahrenTextField: UITextField!
    @IBOutlet var celsiusLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading
        // the view.
    }

    @IBAction func convert(_ sender: Any) {
        /*
        let fahrenStr = fahrenTextField.text
        let fahrenheit = Double(fahrenStr!)
        let celsius = (fahrenheit! - 32.0) / 1.8
        let celsiusStr = String(celsius)
        celsiusLabel.text = "섭씨 " + celsiusStr
        */

        let fahrenStr = fahrenTextField.text
        let fahrenheit = Double(fahrenStr!)
        if let fahren = fahrenheit {
            let celsius = (fahren - 32.0) / 1.8
            let celsiusStr = String(celsius)
            celsiusLabel.text = "섭씨 " +
                celsiusStr
        } else {
            celsiusLabel.text = "화씨 값을 입력하세요"
        }
    }
}
```



이 때, 3개의 객체를 배치한다.

@IBOutlet을 통해 main.storyboard에 있는 객체와 viewController에 있는 fahrenTextField 객체와 celsiusLabel 객체 등을 연결하는 작업을 했다.

변환 버튼에 대해서는 @IBAction으로 convert함수에 연결하는 작업을 했다. 그래서 변환 버튼이 눌려진 순간 값이 변환이 되도록 설정을 했다.

```
import UIKit

class ViewController: UIViewController {

    @IBOutlet var fahrenTextField: UITextField!
    @IBOutlet var celsiusLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading
        // the view.
    }

    @IBAction func convert(_ sender: Any) {
        /*
        let fahrenStr = fahrenTextField.text
        let fahrenheit = Double(fahrenStr!)
        let celsius = (fahrenheit - 32.0) / 1.8
        let celsiusStr = String(celsius)
        celsiusLabel.text = "섭씨 " + celsiusStr
        */

        let fahrenStr = fahrenTextField.text
        let fahrenheit = Double(fahrenStr!)
        if let fahren = fahrenheit {
            let celsius = (fahren - 32.0) / 1.8
            let celsiusStr = String(celsius)
            celsiusLabel.text = "섭씨 " +
                celsiusStr
        } else {
            celsiusLabel.text = "화씨 값을 입력하세요"
        }
    }

    fahrenTextField.resignFirstResponder()
}
```

fahrenTextField와 celsiusLabel 필드가 읍셔널 변수이지만 바인딩을 하지 않는 이유는 벗겨진 채로 정의가 되어 있기 때문이다.

자바 형식으로 선언한 이유는 main.storyboard에서 연결하는 작업에서 이 객체를 가리키는 주소 값을 당연히 가지고 있다는 것을 알고 있기 때문이다.

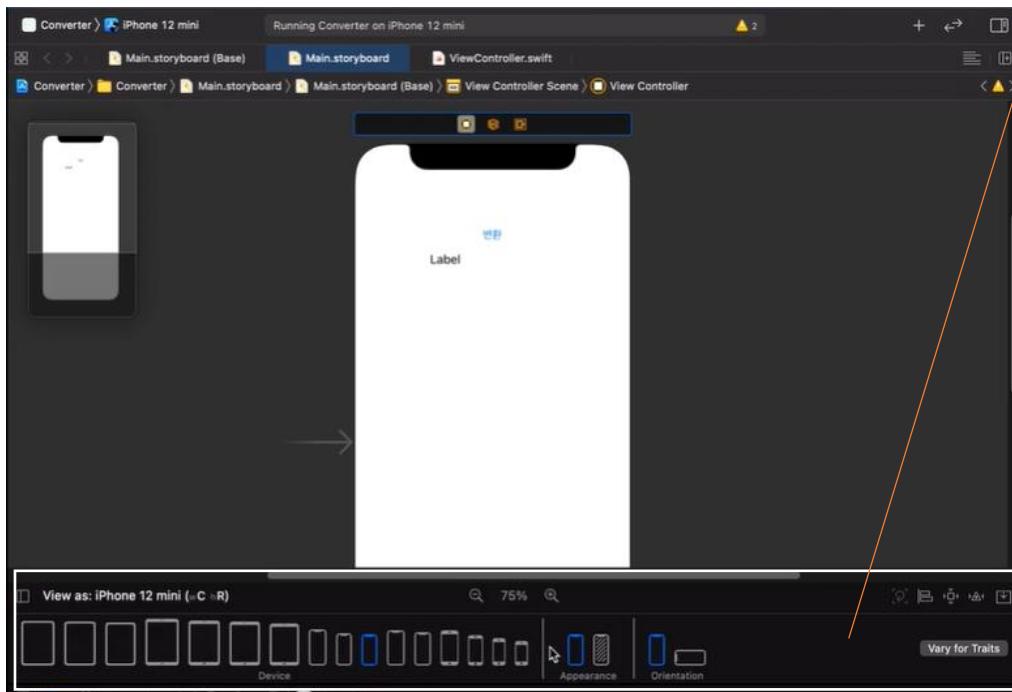
Let Fahrenheit = Double(fahrenStr)->fahrenStr를 double형으로 형변환

`fahrenTextField.resignFirstResponder()` 이 함수는 fahrenTextField 객체에게 키보드의 권한을 내려 놓으라는 것을 나타내는 함수이다.

변환 버튼을 누를 때 키보드의 권한을 내려 놓게 될 것이다.



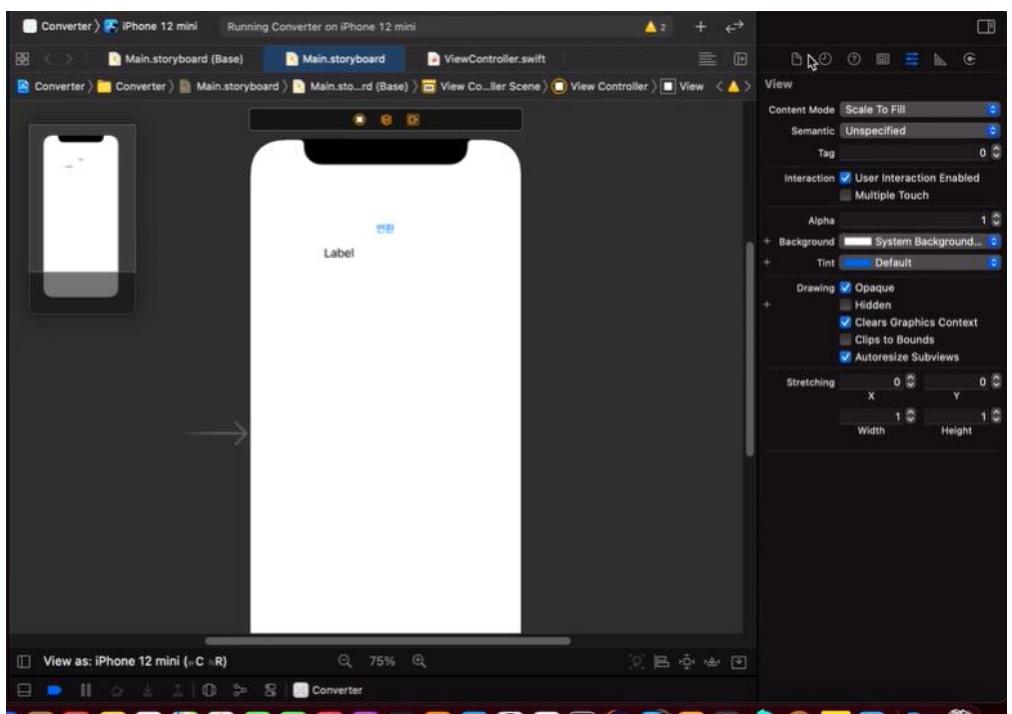
우리가 짠 코드를 그대로 실행시키면 우리가 원하는 배치가 되지 않는다. -> 배치 문제가 발생



View as iPhone을 누르게 되면, 가로모드와 세로모드로 변환할 수 있다.

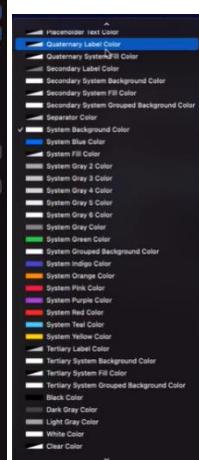
이를 대응할 수 있는 배치를 해야 한다.

다른 디바이스를 선택하면 다르게 배치되는 것을 알 수 있다. 그 이유는 절대값으로 배치했기 때문이다.



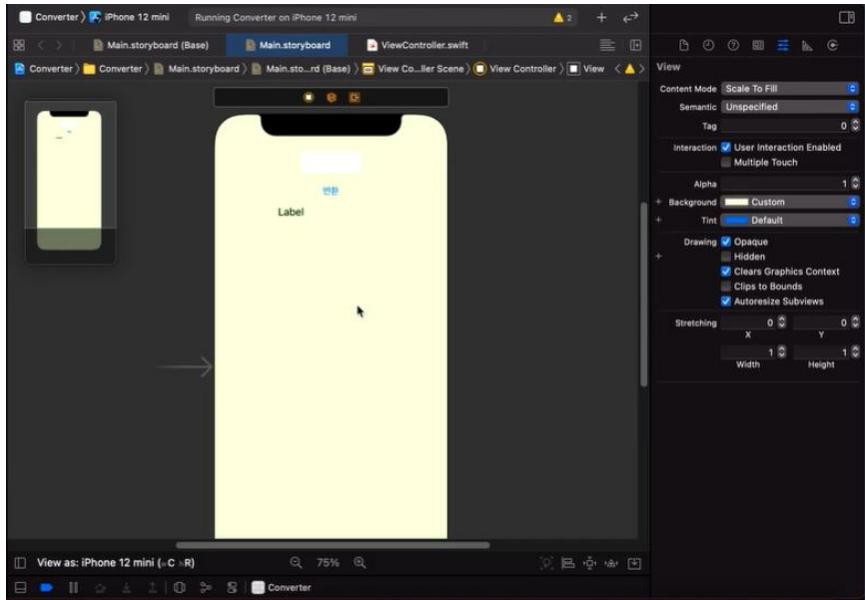
View 객체를 클릭하면 view 속성을 볼 수 있는 창이 뜬다.

View의 색을 변경하고 싶을 때, Background를 클릭한다.



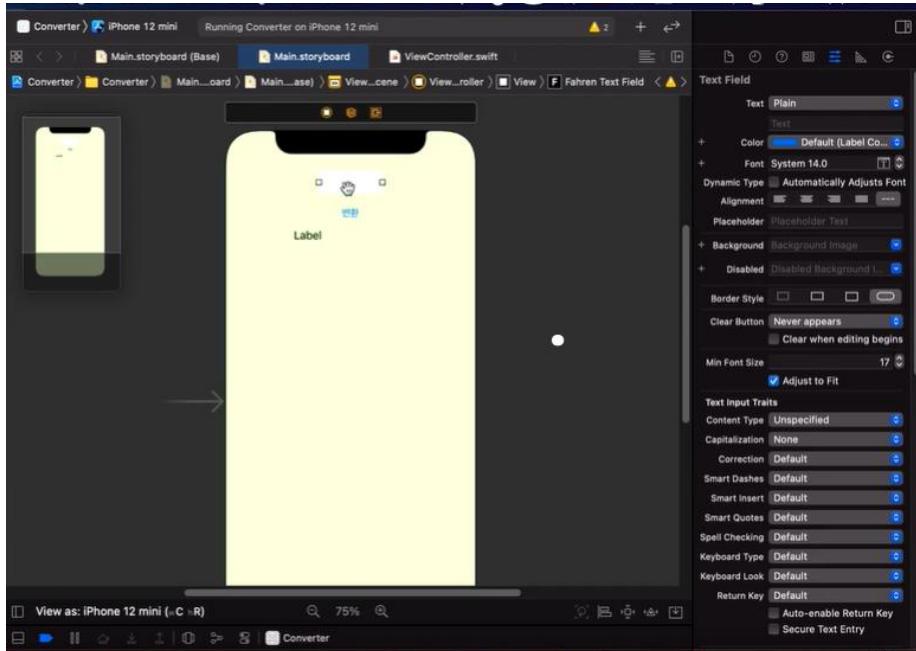
나만의 색을 선택하고자 할 때, custom을 선택하면 된다.

Background 색 변경 완료



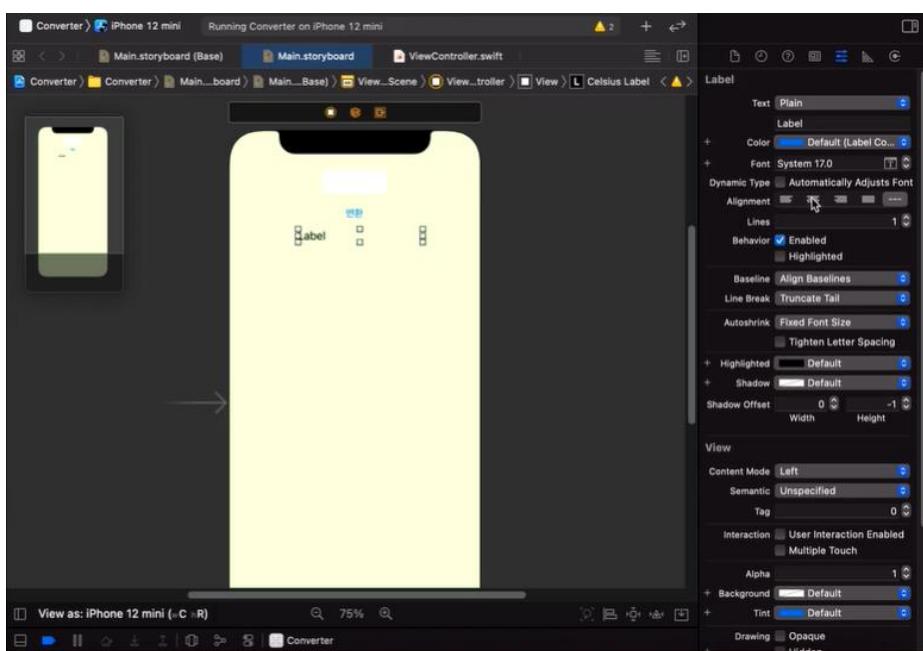
TextField 속성을 보면

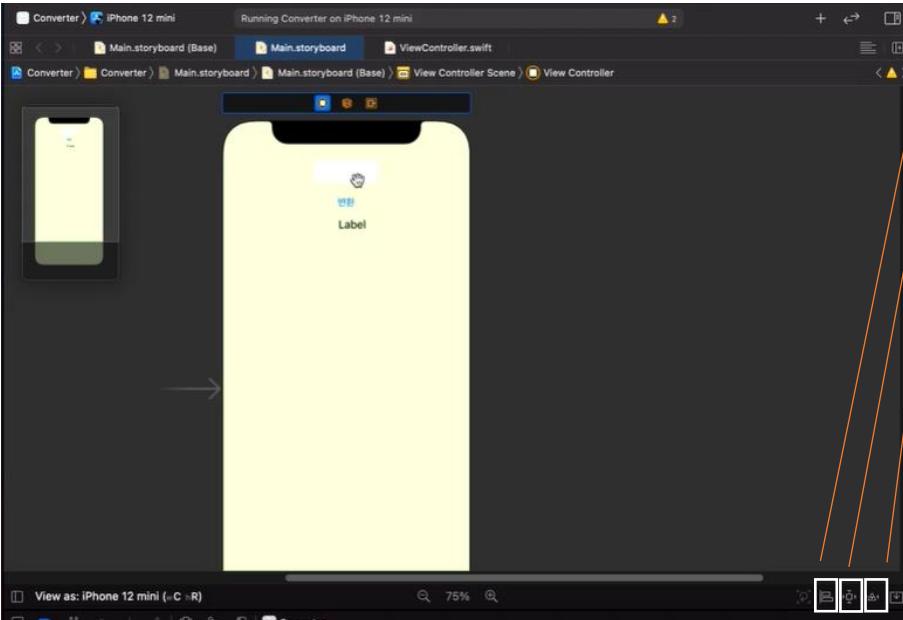
가운데 정렬로 입력을 받고 싶을 때,  
Alignment를 클릭해서 가운데 정렬을  
클릭하면 된다.



Label을 클릭하면 Label에 관한 속성  
창이 나오게 된다.

가운데 정렬을 하고자 할 때  
Alignment를 통해 가운데 정렬을 선택한다.





<배치 문제 해결>

Align 은 정렬

Add New Constraints는 제약조건 설정

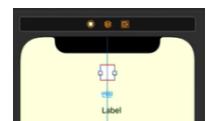
Resolve Auto Layout issues는 전체적인 layout 배치에 관한 작업을 할 수 있는 것이다.



Align 클릭하면, 이와 같은 창이 뜨게 된다.

가로 모드이든, 세로 모드이든 중앙에 배치하기를 하고자 하면, Horizontally in Container를 체크한다. 그리고 Add 1 Constraint 클릭하게 되면, 세로 줄이 추가되면서 가운데 정렬 조건이 추가되었음을 알 수 있다.

하지만 아직도 위아래 부분이 아직 조건에 충족이 되지 않았다는 것을 옆에 있는 그림으로 알 수 있다.



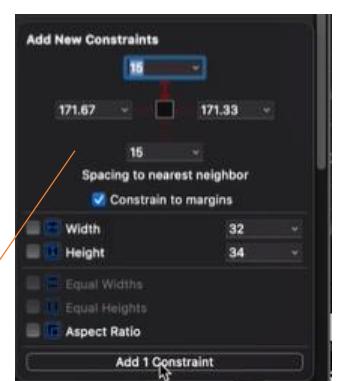
이와 같은 제약조건을 충족하기 위해서 다음과 작업을 할 것이다.

이번에는 Add New Constraints 부분을 클릭하게 되면 다음과 같은 창이 뜬다.

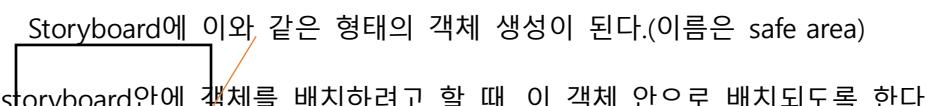
위아래 양옆으로 이와 같은 위치에 있다는 것을 표시하고 있다는 것을 알 수 있다.

Spacing to nearest neighbor로 가장 가까운 객체로부터의 거리를 나타내는 것을 알 수 있습니다. (아래 쪽으로는 변환객체로부터 거리임을 알 수 있음)

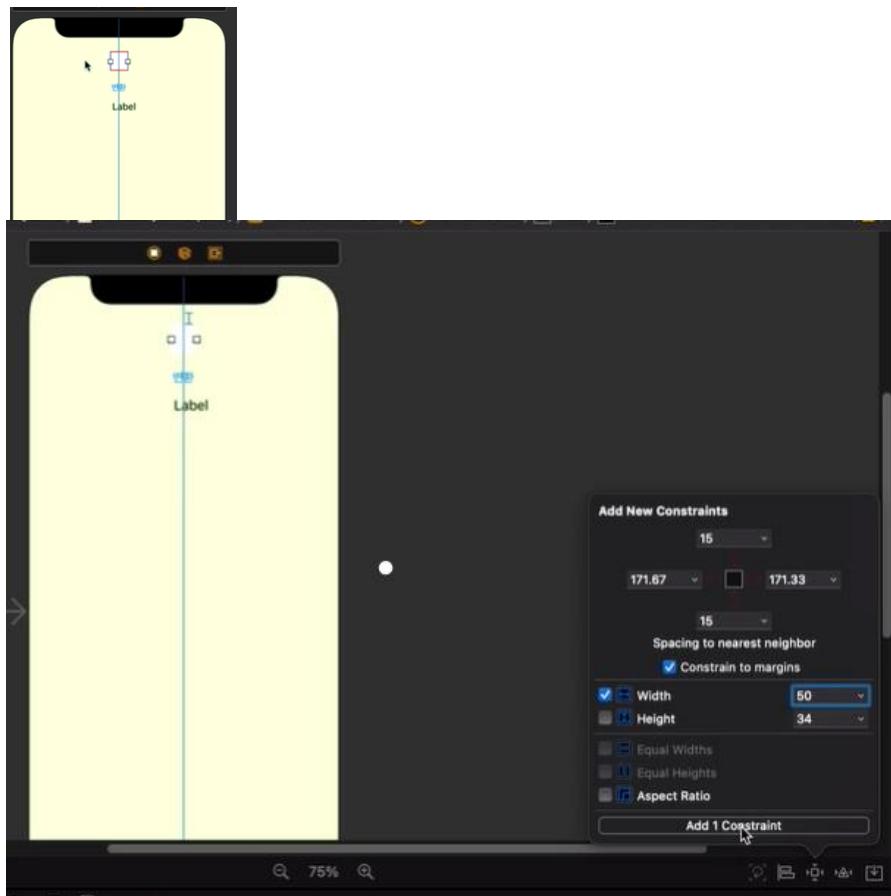
위아래 조건을 주기 위해서는 이와 같이 활성화 시켜주어야 한다. 그리고 나서 Add 1 Constraint를 클릭해준다.



Storyboard에 이와 같은 형태의 객체 생성이 된다.(이름은 safe area)

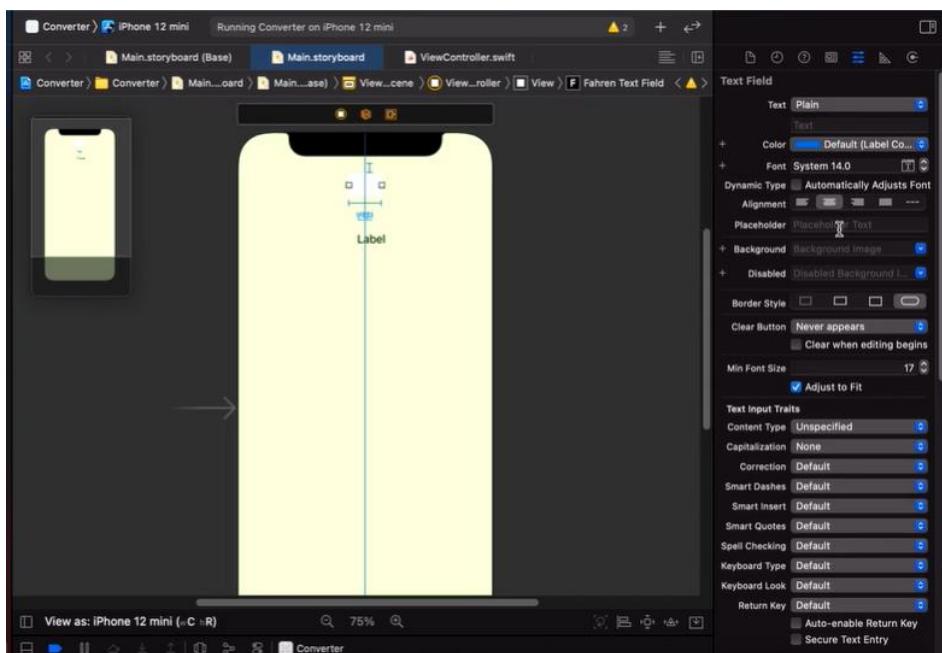


storyboard안에 객체를 배치하려고 할 때, 이 객체 안으로 배치되도록 한다.

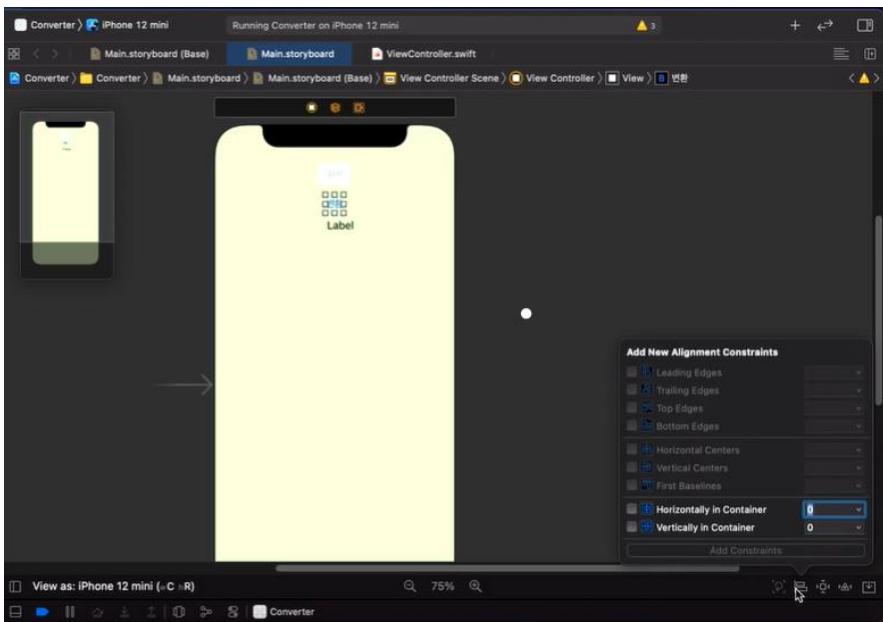


TextField에 설정을 더 추가할 것인데 Width를 체크해주고 50값을 넣어준다. 그리고 나서 Add 1 Constraint를 클릭한다.

그러면 TextField의 값이 늘어나게 된다는 것을 알 수 있다.



TextField의 속성을 다시보면, 이와 같은 차이 나올 것이고, Placeholder에 글자를 넣으면 글자를 흐르게 하는 효과가 나올 것이다.

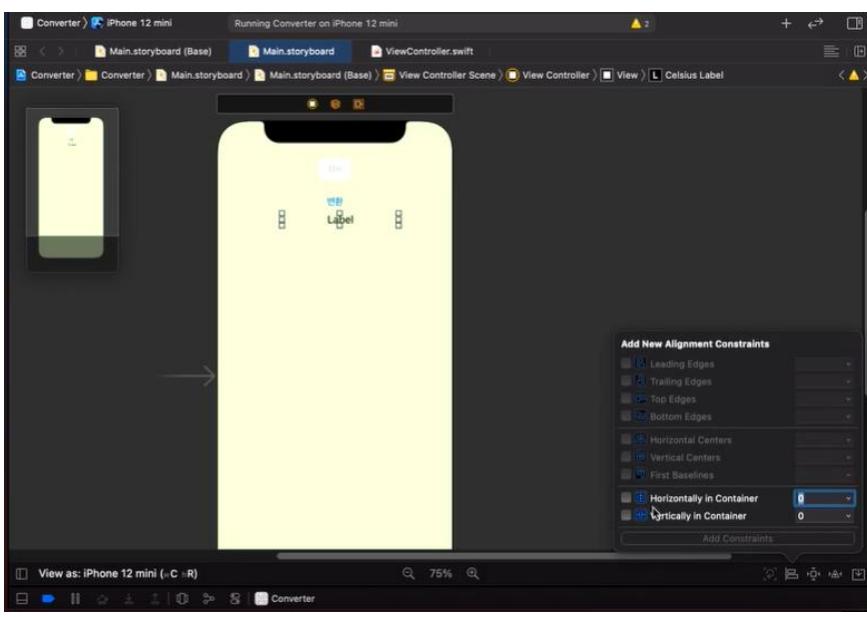


Button에 대해서도 동일하게 작업합니다. 버튼을 클릭하고 Align를 클릭하게 되고, Horizontally in Container를 클릭하여 Add 1 Constraints를 누른다.

그 후에, Add New Constraints 누르고,

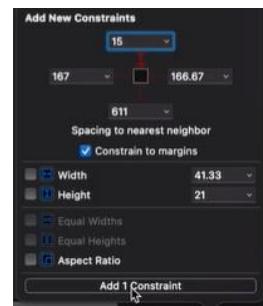


20을 입력하고, 그 다음에 Add 1 Constraint를 눌려서, 해당 속성을 적용한다.



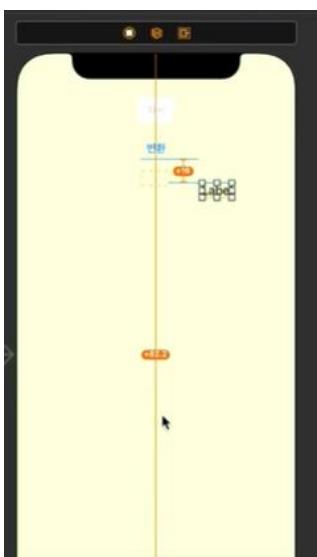
Label에 관한 속성을 설정하자, Align를 클릭한 다음과 같이 설정해주어야 한다. Horizontally in Container를 클릭해주고 Add 1 Constraint를 클릭해준다.

그 후에 Add New Constraints를 누르면,



15를 입력하고, 그 다음에 Add 1 Constraint를 눌려서, 해당 속성을 적용한다.

조건을 설정한 후에, 마음대로 드래깅을 하면, 빨간색 경고가 나오게 됨을 알 수 있다.



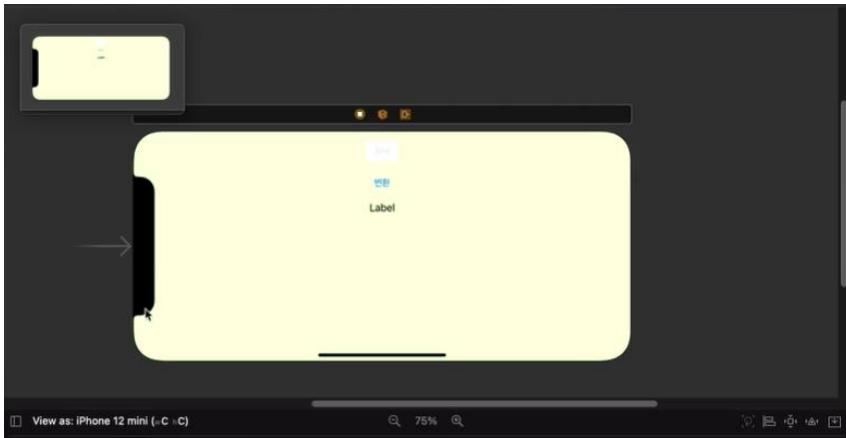
그리고 조건이 맞는 곳으로 점선이 표시가 된다.

조건이 맞는 곳으로 다시 옮기고 싶을 때에는

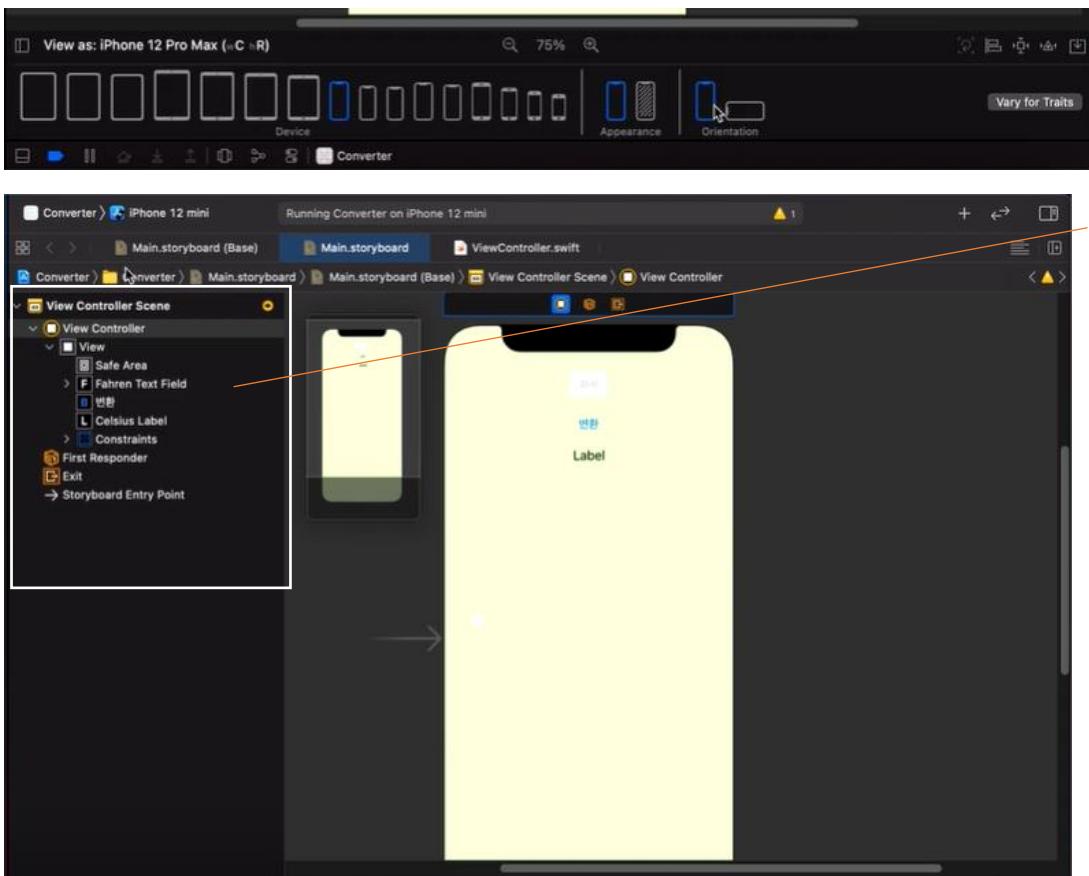


Update Frames를 누르게 되면 조건에 맞는 위치로 가게 된다.

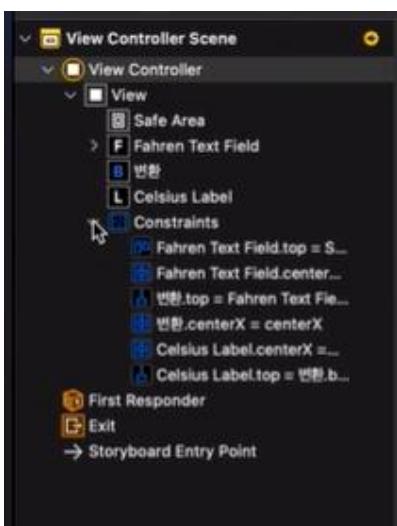
다음과 같이 설정하게 되면, 가로 모드에서도 잘 배치가 되고 있다는 것을 알 수 있다.



그 다음에는 다른 디바이스 환경을 선택하여 잘 배치가 되고 있는지를 확인하면 된다.



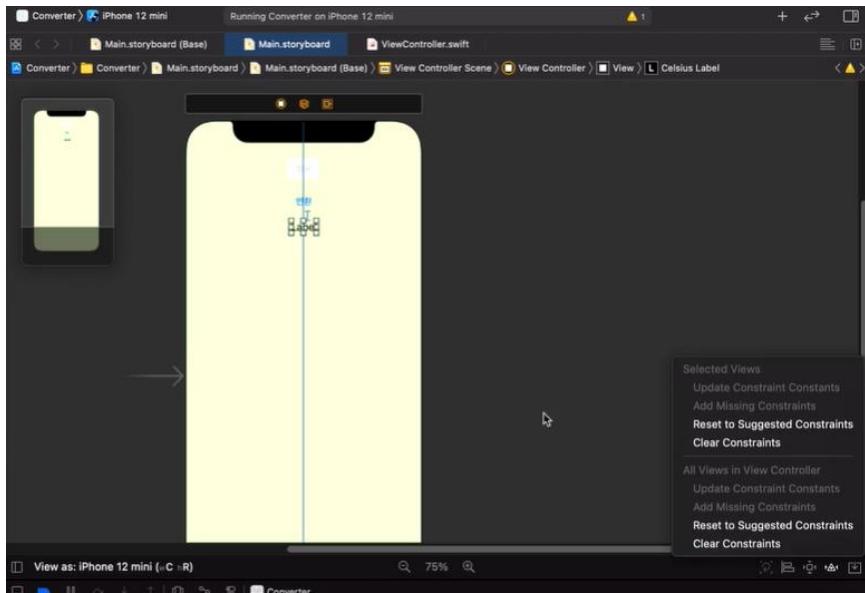
Safe Area는 사면에 있는 얇은 벽으로 Safe Area 안에 객체가 배치되도록 해준다.



Constraints를 전개하면 우리가 조건을 걸어주었던 것이 나오게 된다는 것을 알 수 있다. (조건식이 되어 있다는 것을 알 수 있다.)

Constraints 아래 있는 조건 중 하나를 지우면, 제약 조건이 사라진다는 것을 알 수 있다.

Resolve Auto Layout issues 속성에 대해 살펴보자.

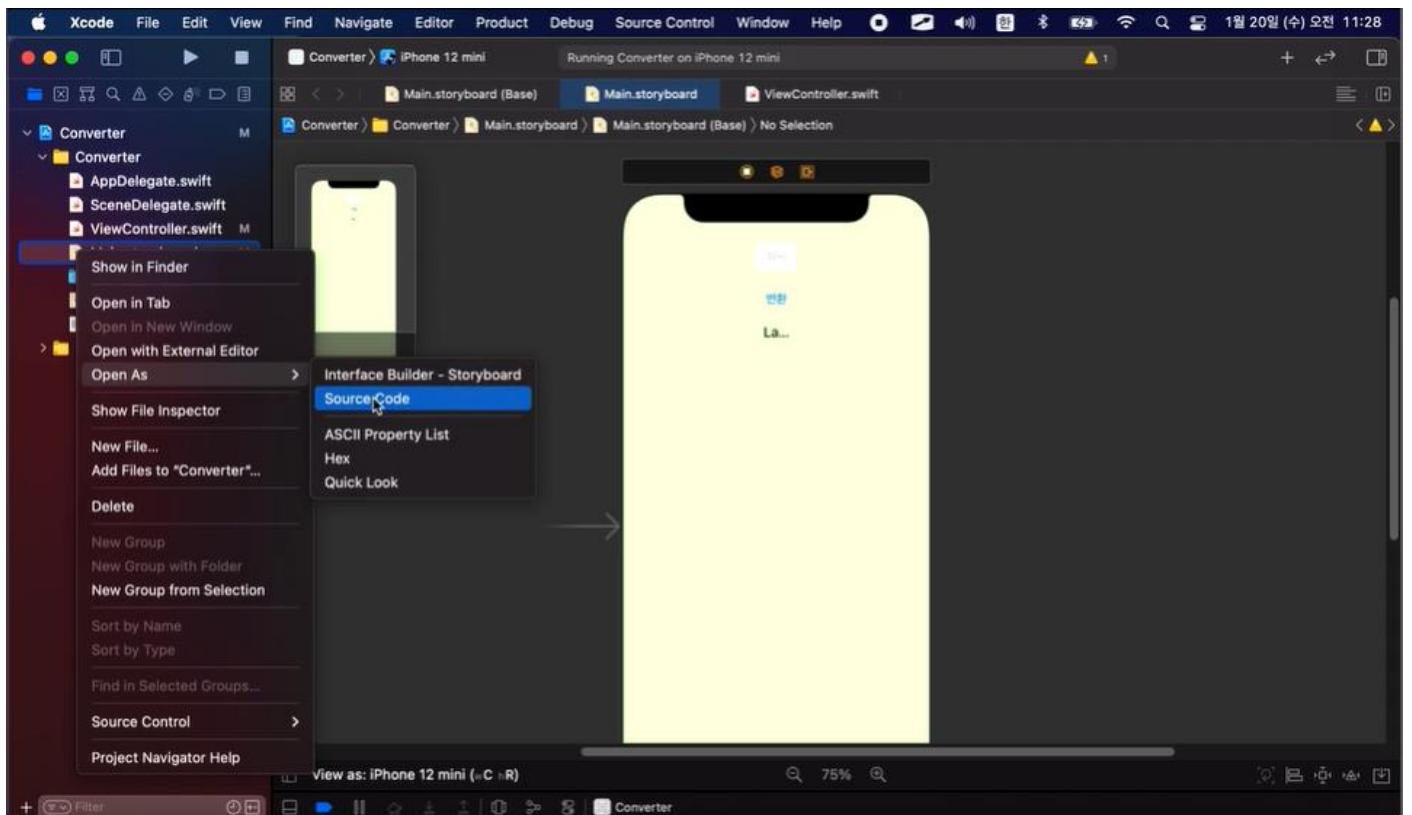


이와 같은 형태가 나오게 된다는 것을 알 수 있다.

Selected Views인 경우에는 선택한 view 객체에 적용할 항목이며, All Views in View Controller는 전체 view 객체에 적용할 항목이다.

Selected Views 부분에서 Reset to Suggested Constraints와 Clear Constraints가 존재한다. Clear Constraints는 Constraints 조건이 사라지게 된다.

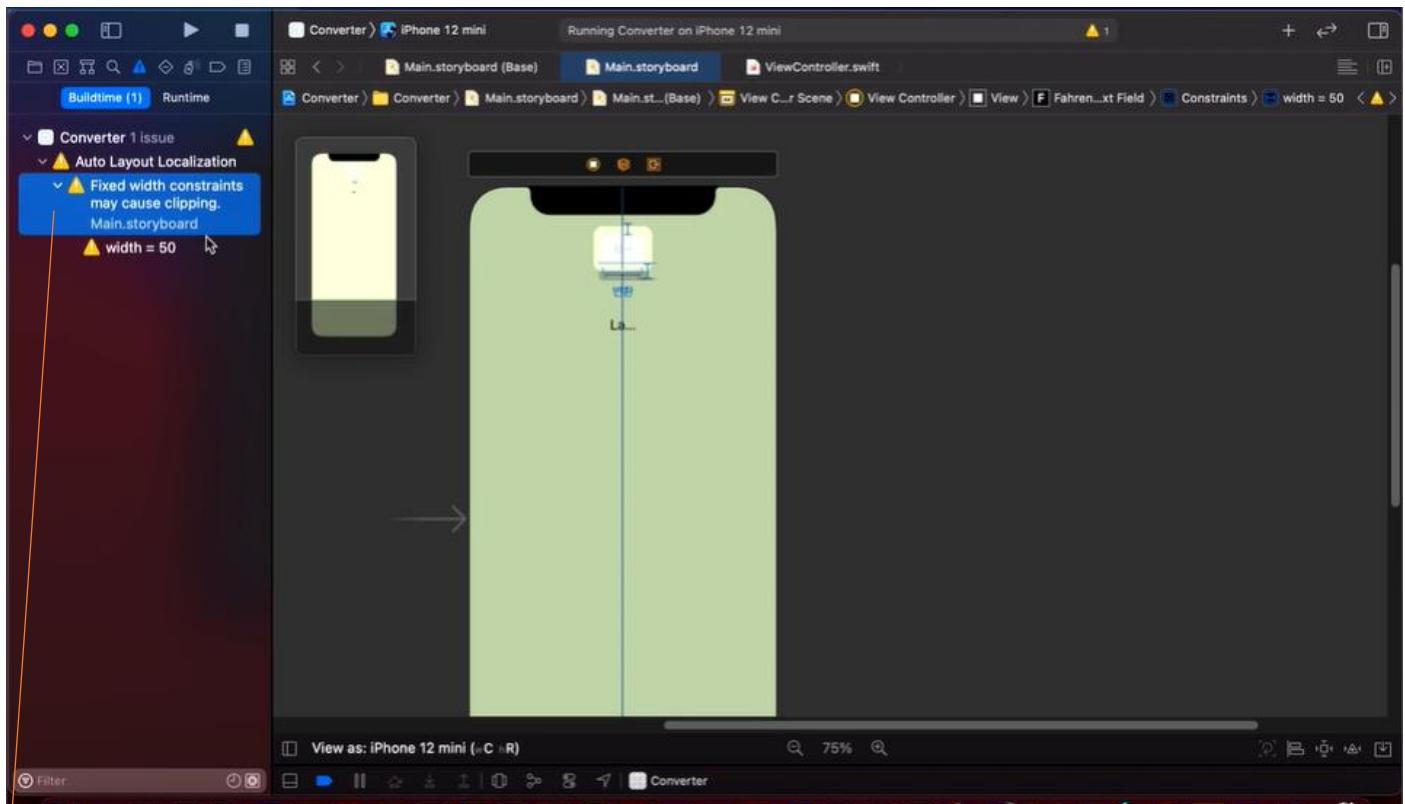
Reset to Suggested Constraints를 클릭하게 되면, 인터페이스가 빌더가 제안한 제약조건이 생성이 되고 있음을 알 수 있다.



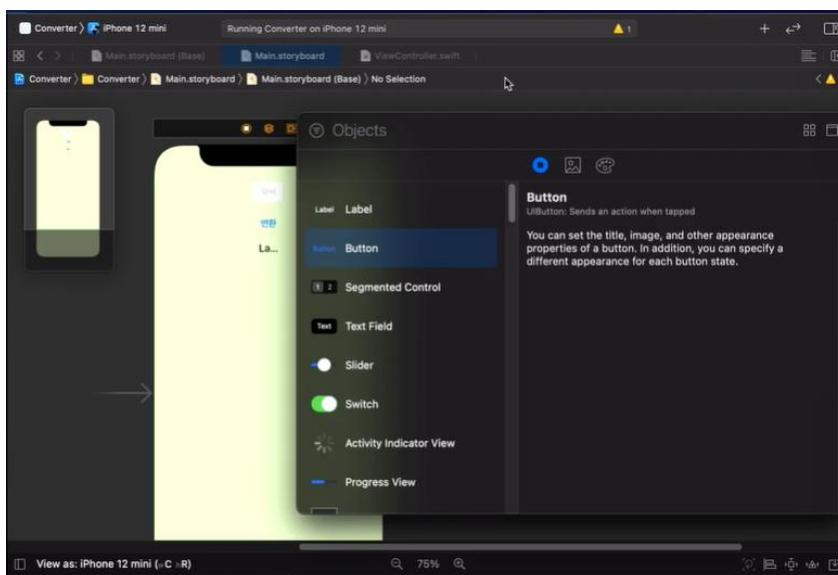
하게 되면 xml 파일이 나오게 된다.

```
30          <action selector="convert:" destinations="BYZ-3B-t0r"
31              eventType="touchUpInside" id="0a2-39-Ru1"/>
32      </connections>
33  </button>
34  <label opaque="NO" userInteractionEnabled="NO" contentMode="left"
35      horizontalHuggingPriority="251" verticalHuggingPriority="251" text="Label"
36      textAlignment="center" lineBreakMode="tailTruncation"
37      baselineAdjustments="alignBaselines" adjustsFontSizeToFit="NO" id="a00-eY-12A">
38      <rect key="frame" x="167" y="158" width="41" height="21"/>
39      <fontDescription key="fontDescription" type="system" pointSize="17"/>
40      <nil key="textColor"/>
41      <nil key="highlightedColor"/>
42  </label>
43  </subviews>
44  <viewLayoutGuide key="safeArea" id="6Tk-0E-BBY"/>
45  <color key="backgroundColor" red="0.93465673923906634" green="1"
46      blue="0.83195120837306481" alpha="1" colorSpace="custom" customColorSpace="sRGB"/>
47  <constraints>
48      <constraint firstItem="a00-eY-12A" firstAttribute="top" secondItem="UeW-ae-NYB"
49          secondAttribute="bottom" constant="15" id="0MS-0d-B45"/>
50      <constraint firstItem="kuv-Ye-aBZ" firstAttribute="top" secondItem="6Tk-0E-BBY"
51          secondAttribute="top" constant="15" id="5w1-Uz-2Yg"/>
52      <constraint firstItem="UeW-ae-NYB" firstAttribute="top" secondItem="kuv-Ye-aBZ"
53          secondAttribute="bottom" constant="20" id="0Rr-Oy-D8I"/>
54      <constraint firstItem="a00-eY-12A" firstAttribute="centerX" secondItem="BbC-Xf-vdC"
55          secondAttribute="centerX" id="Zum-Ey-LD0"/>
56      <constraint firstItem="UeW-ae-NYB" firstAttribute="centerX" secondItem="BbC-Xf-vdC"
57          secondAttribute="centerX" id="aJH-MA-0K1"/>
58      <constraint firstItem="kuv-Ye-aBZ" firstAttribute="centerX" secondItem="BbC-Xf-vdC"
59          secondAttribute="centerX" id="t2G-nk-F0k"/>
60  </constraints>
61  </view>
62  <connections>
63      <outlet property="celsiusLabel" destination="a00-eY-12A" id="COV-RQ-SLX"/>
64  </connections>
```

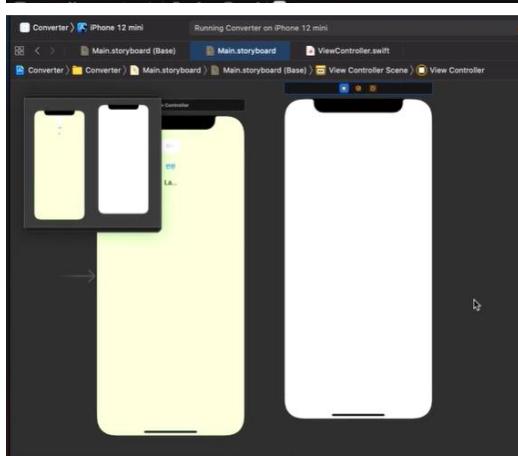
경고창을 보면



이와 같은 경고 또는 것은 입력 값이 길 경우에는 잘릴 수 있다는 것을 나타낸다. -> 우리는 이 경고를 무시하고 진행할 것이다.



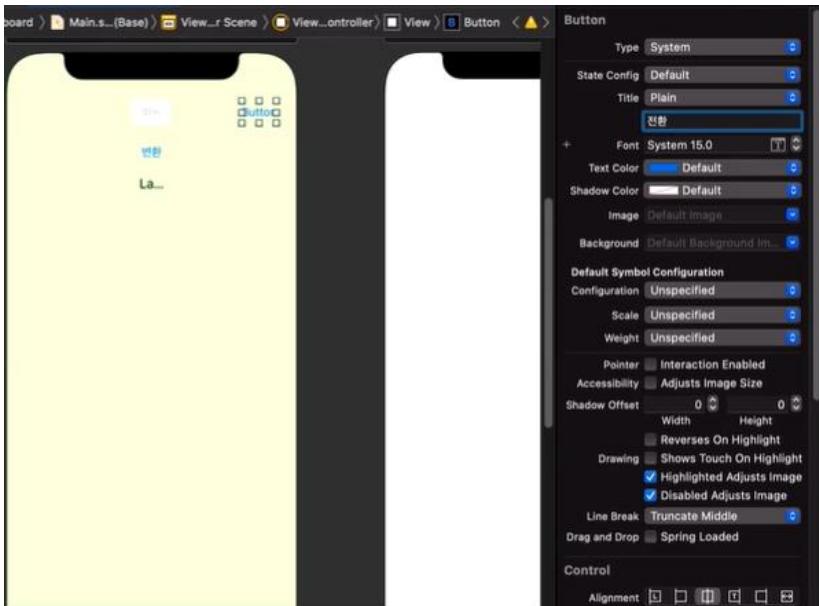
새로운 화면을 생성하기 위해, library인 + 버튼을 누려준다. 그 후에, view controller를 입력하여, 적당한 위치에 드래그를 한다.



View 객체가 새로 생성이 되고 있음을 알 수 있다.

이제, 2 view 객체를 연결해줄 수 있는 게 필요하다.

그래서, 이를 연결해주는 역할을 해주는 버튼 객체를 main view에 배치해준다.

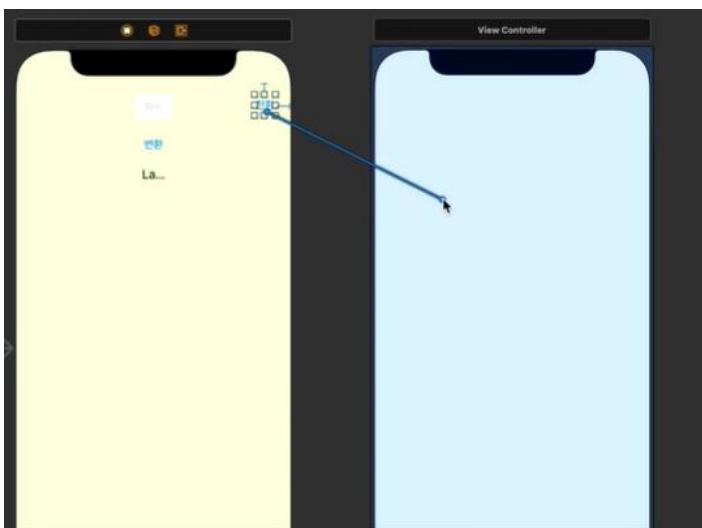


Button의 속성을 설정해보자.

먼저, 버튼의 글자를 전환 글자로 바꾸어 준다.



Add New Constraints에서 제약조건들을 설정해준다.

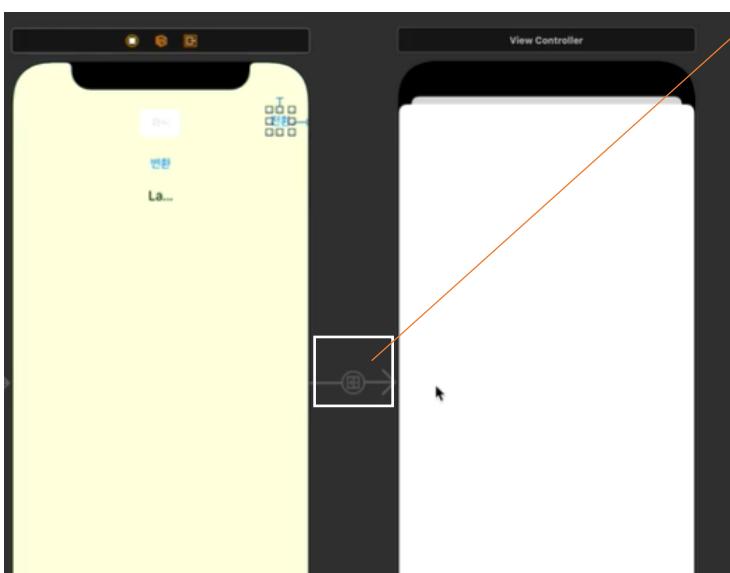


2개 View 객체를 연결하는 방법으로는 다음과 같습니다. 변환 버튼을 **ctrl**를 누른 상태에서 **dragging**을 해줍니다. 2번째 view 객체에 **drop**를 합니다.

그러면 다음과 같은 창이 뜨게 된다.

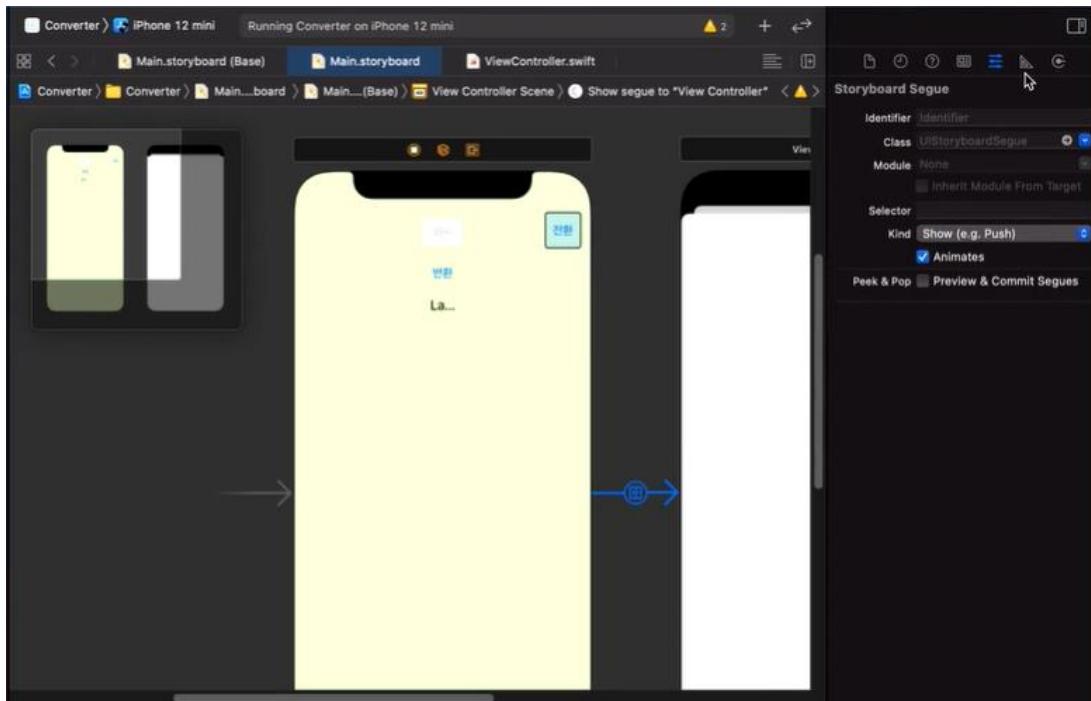
Action Segue는 어떤 방식으로 전환할 것인지를 나타낸다.

기본적인 방식인 **Show**로 클릭하면 아래 그림과 같이 2번째 view가 생성이 된다.



Segue(읽을 때에는 Segway) 객체가 생성이 되었음을 알 수 있고, 변환 버튼으로 인해 2번째 view가 전환이 된다는 것을 알 수 있다.

풀 스크린 모드로 전환하는 방법에 대해서 배워보자.



segue(읽을 때 Segway) 객체를 클릭 된 상태에서 속성창을 열면 다음과 같은 형태가 되고 있음을 알 수 있다.



Kind부분의 창을 클릭하여, 전환 방식을 Show 말고 Present Modally로 바꾸어 준다.



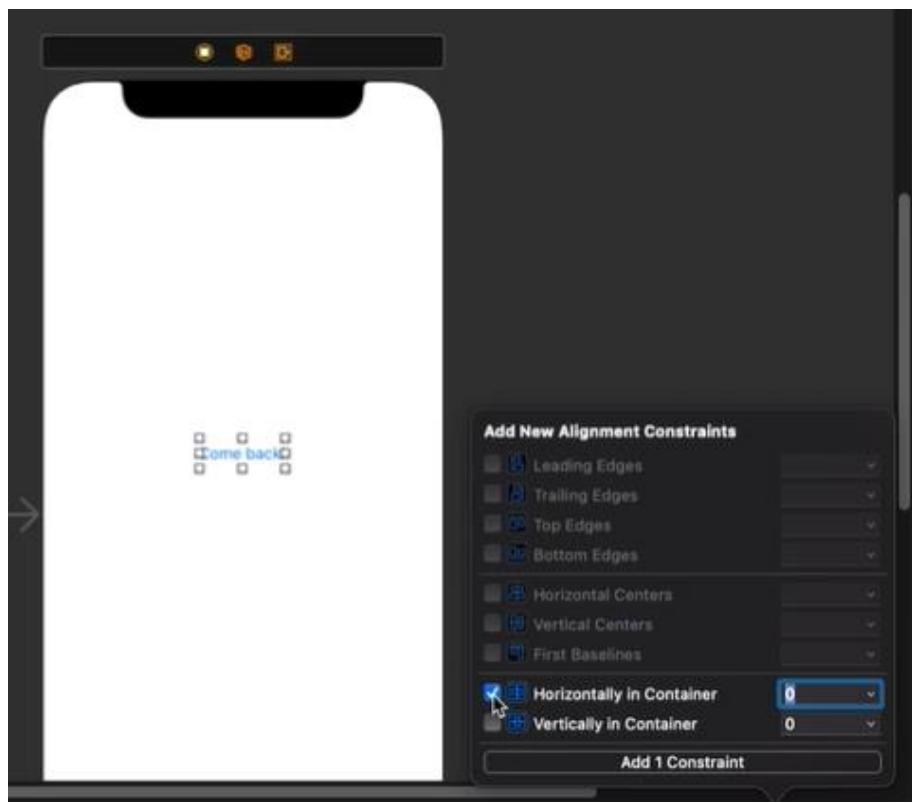
Presentation 스타일을 Full Screen으로 바꾸어 준다.



Transition 스타일 즉, 애니메이션을 Flip Horizontal로 바꾸어 준다.

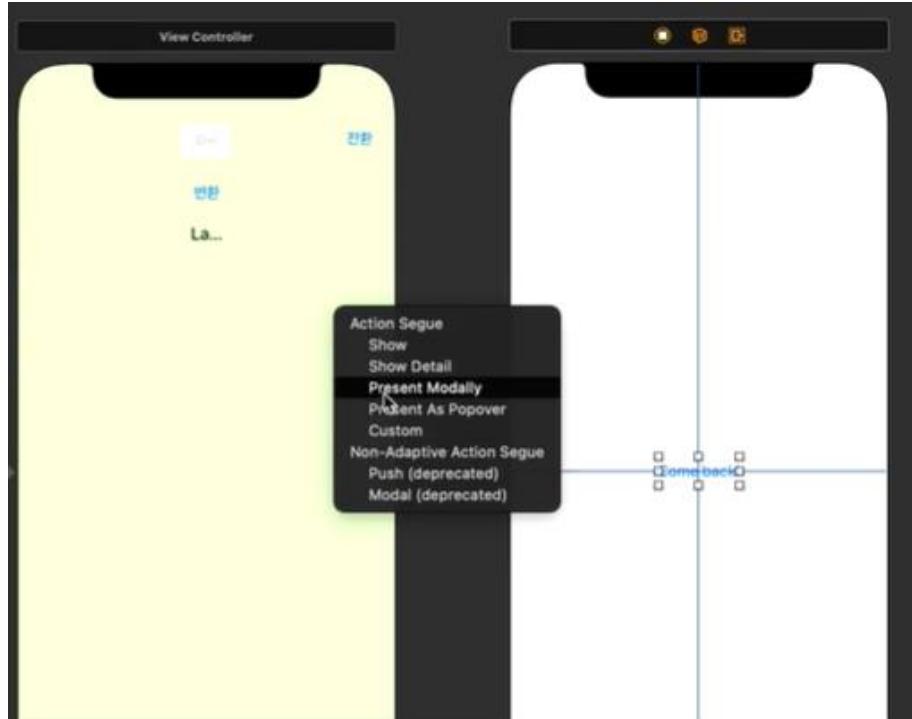
Full Screen으로 하면 첫 화면으로 돌아갈 수 있는 방법이 없어져 버리게 된다. 그 방법을 이제 만들 생각입니다.

첫 번째 화면으로 돌아갈 수 있는 버튼을 2번째 View 생성한다.



library에서 버튼 객체를 2번째 view 객체에 dragging and drop해 놓고, 버튼 이름을 come back!!으로 바꾼다.

그리고, Add New Alignment Constraints 속성에 Horizontally in Container 속성을 체크하고 Vertically in Container 속성을 체크하여 버튼이 정 가운데에 있을 수 있게끔 해준다.



Main view에서 2번째 view로 연결해 준 방식처럼 하게 되면, 두번째 view에서 come back!버튼을 누르게 되면 main view가 나오기는 한데 3번째 view로 생성된다는 문제점이 발생하게 된다.

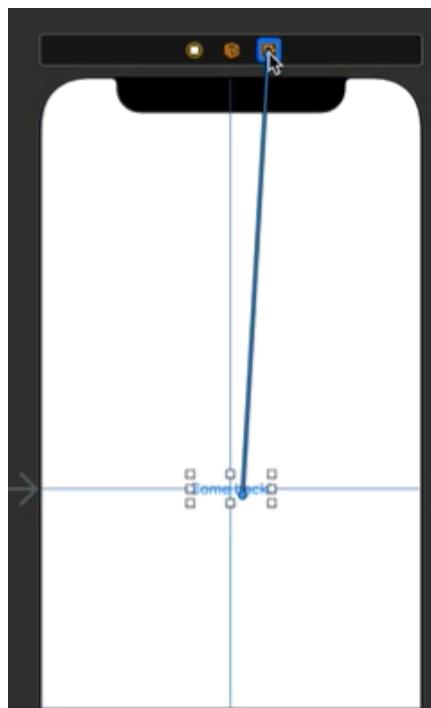
이와 같은 문제점을 해결해보자.

먼저 ViewController에서 이와 같이 타이핑을 해준다.

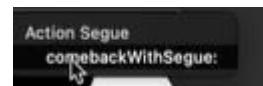
```
@IBAction func comeback(segue: UIStoryboardSegue){
```

```
}
```

그 후에 Main.storyboard로 넘어온다.



come back! 버튼을 마우스 오른쪽 눌려서 ctrl dragging하여 해당 위치에 drop한다.

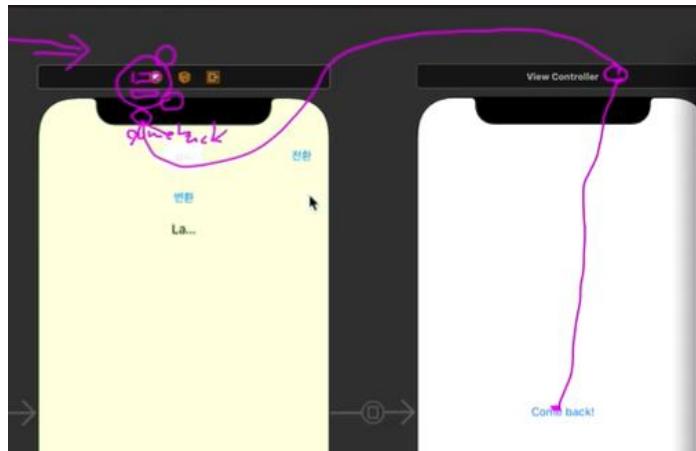


이 정의가 되고, 이를 선택한다.

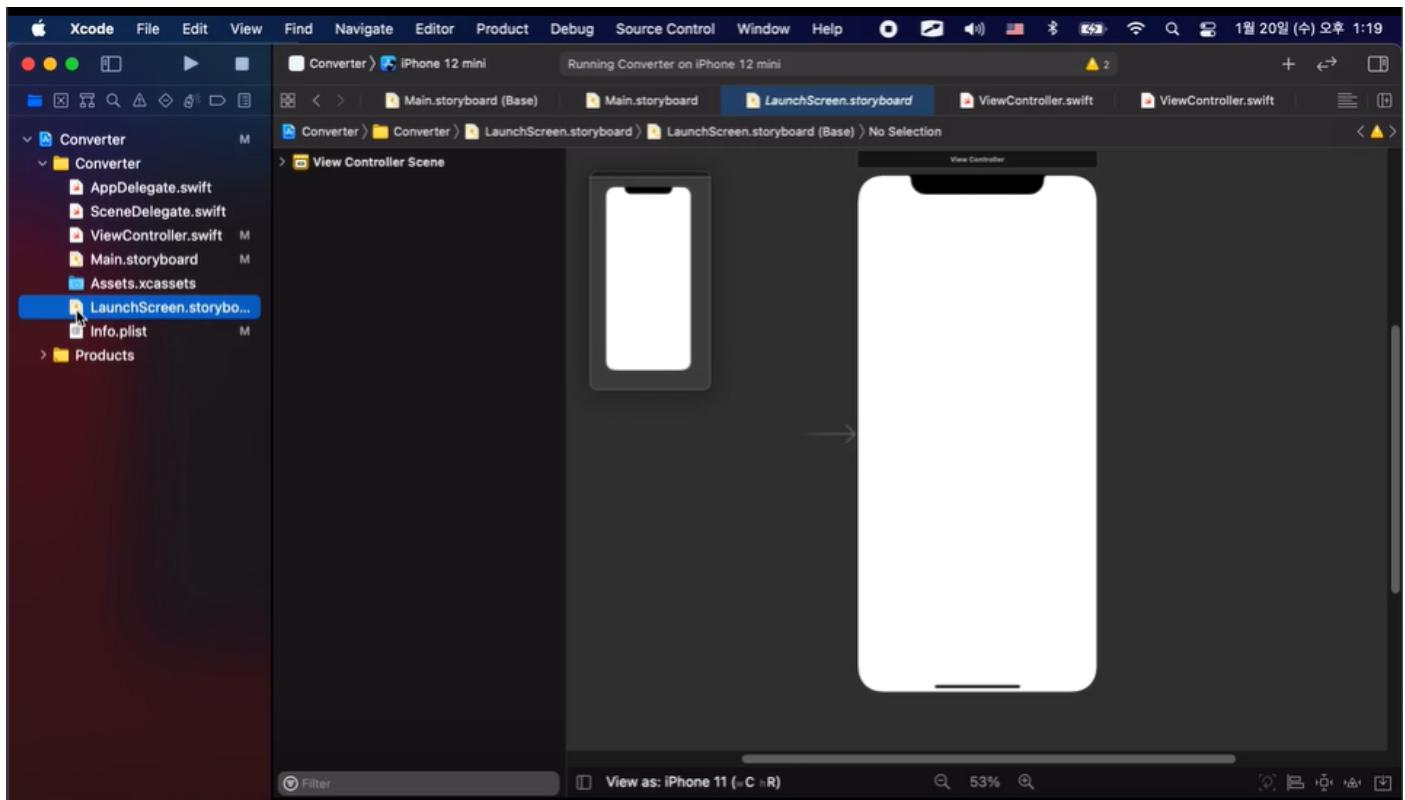
Come back! 버튼을 클릭했을 때, exit를 통해 해당 view를 빠져나가면서 come back 메소드로 연결하게 된다.

Come back! 버튼을 누르게 되면 come back 메서드가 호출이 된다.

즉, 이와 같은 방식으로 해야, 첫 화면으로 돌아가게 된다.

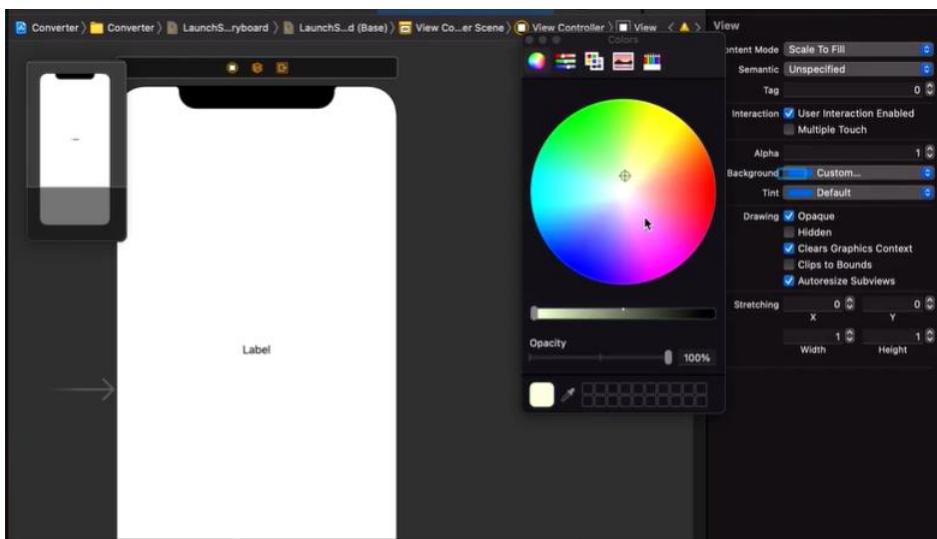


무사히 복귀된다.ㅎ



LaunchScreen.storyboard 클릭하게 되면, 시작화면 모습이 보이게 된다.

Object library를 통해 적당한 곳에 Label 객체를 둔다.

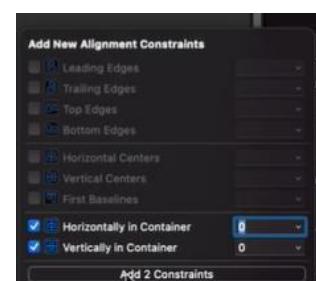


View 객체의 속성 창을 띠워서, background 색을 지정하면 된다. Custom 색을 지정하여, 원하는 색으로 지정하면 된다.

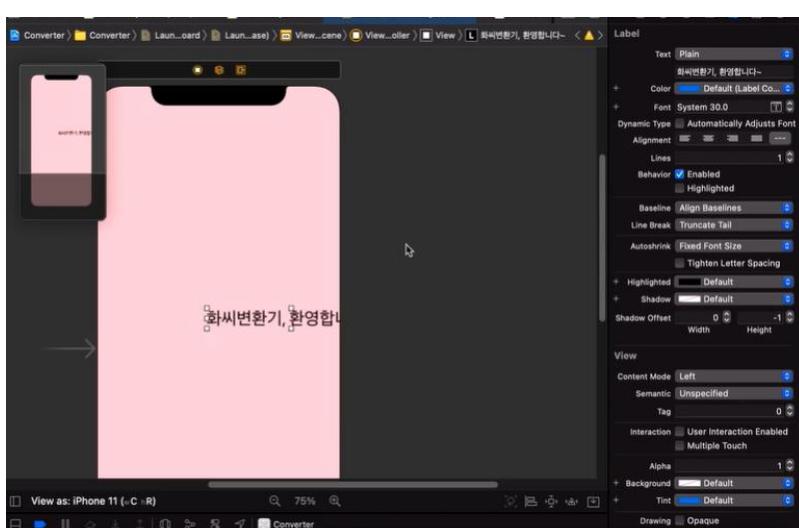


Label 속성 창을 띠워서 Label의 text 값을 “화씨 변환기, 환영합니다”고 변환해주고, Font 크기를 System 30.0으로 해준다.

그 다음에 배치 조건을 준다.



정중앙에 배치가 된다.



초기화면을 보기 위해서 앱 전반의 흐름을 제어하는 AppDelegate.swift를 클릭해준다.

```
func application(_ application: UIApplication,  
didFinishLaunchingWithOptions launchOptions:  
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {  
    // Override point for customization after application launch.  
    return true  
}
```

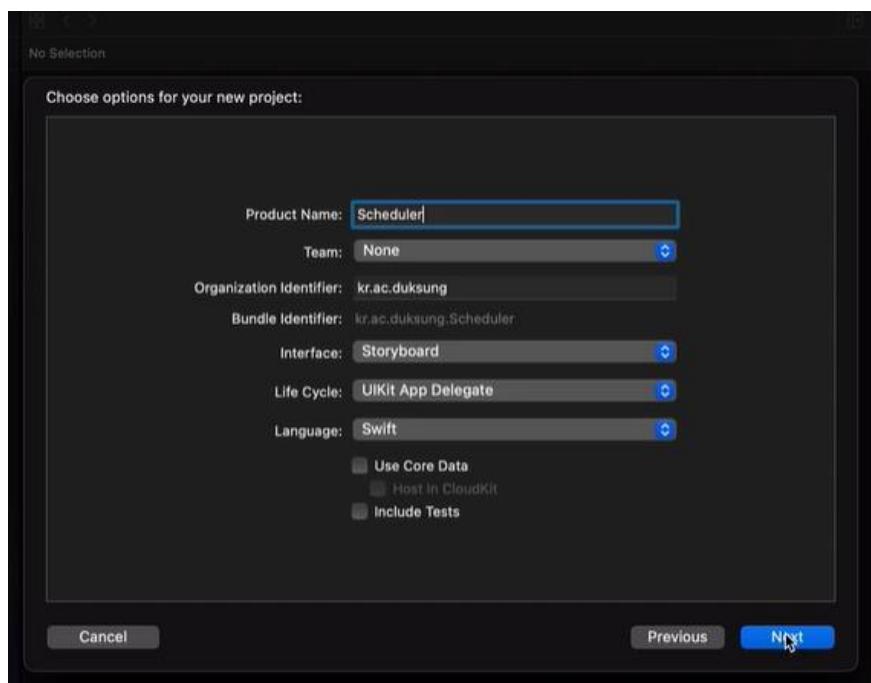
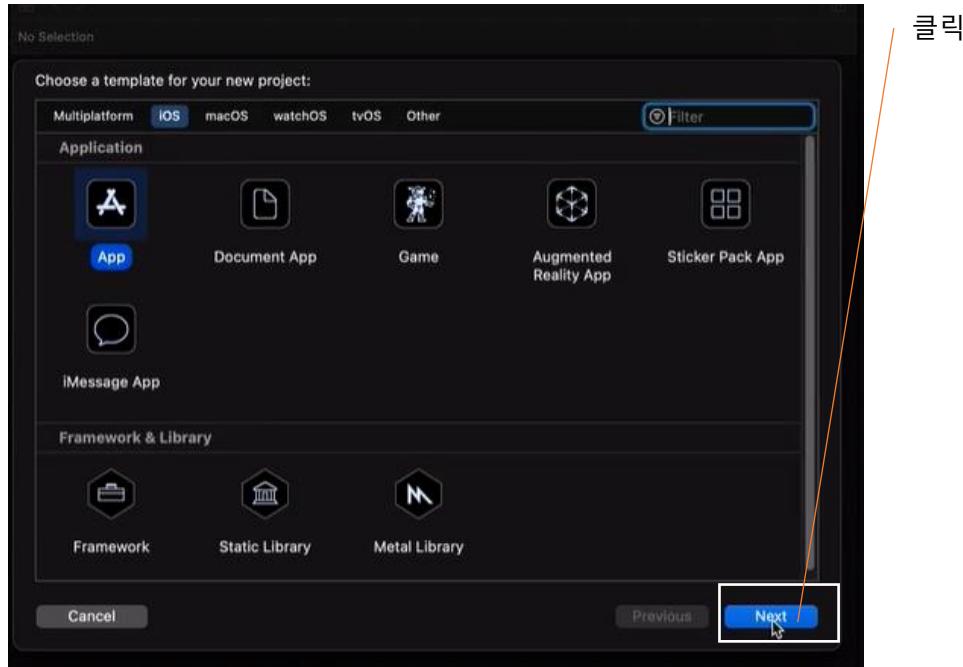
이 함수는 시작화면이 뜨기 전까지 메모리를 올라가야 할 상황을 나타낸다.

시작화면하고 본 화면 사이를 늦춰주고 싶을 때

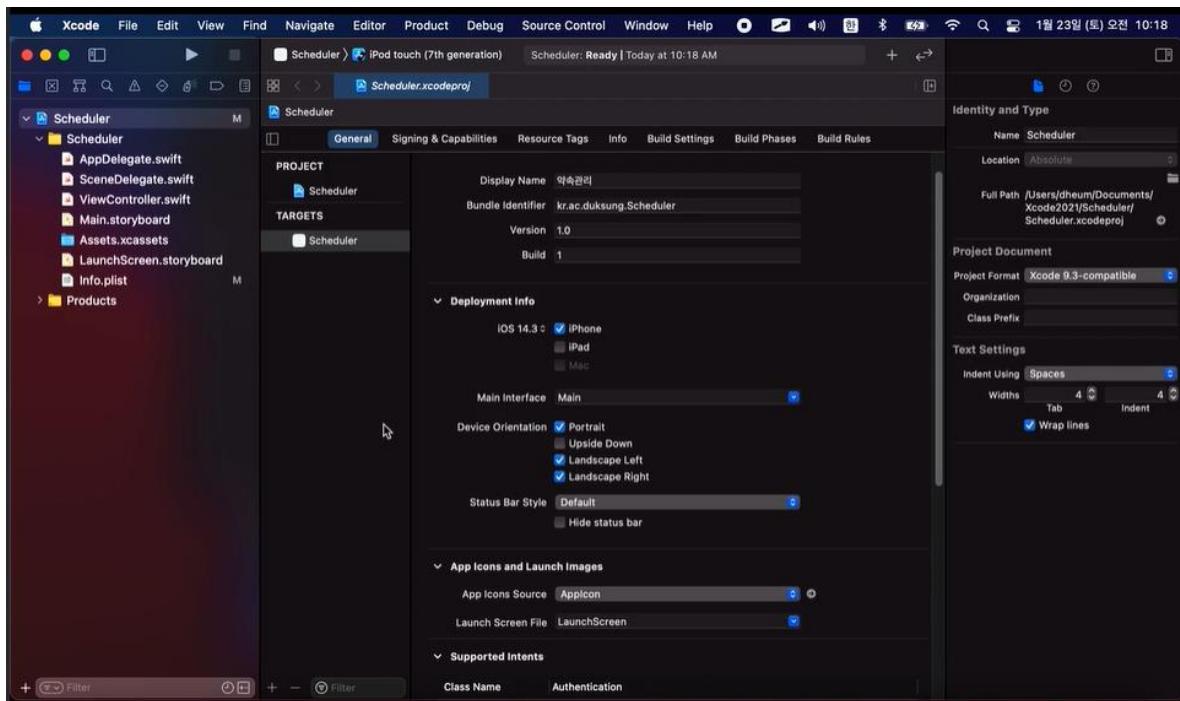
```
func application(_ application: UIApplication,  
didFinishLaunchingWithOptions launchOptions:  
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {  
    // Override point for customization after application launch.  
    sleep(2)  
    return true  
}
```

sleep(2)를 넣어주어 시작화면과 본 화면 사이에 2초동안 간격이 있을 수 있도록 해준다.

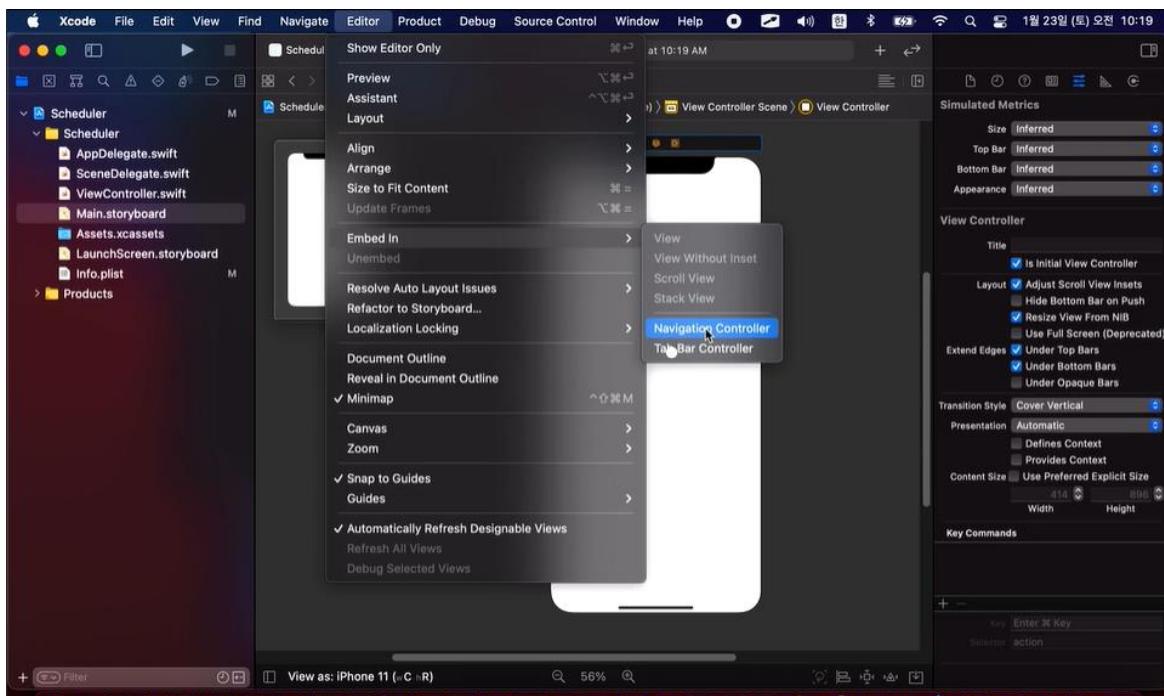
## Iphone 3번째 강의



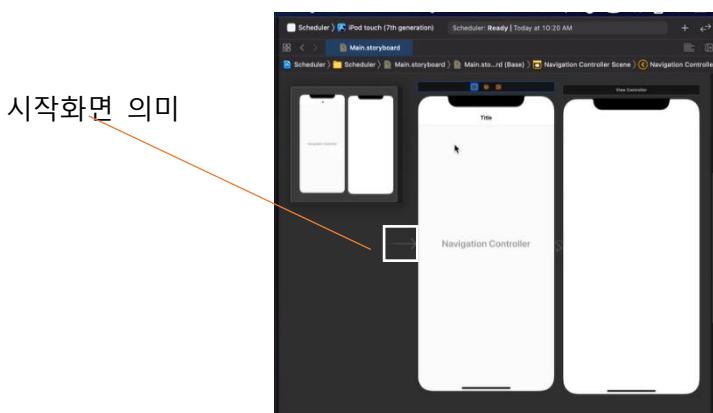
위치는 편한 폴더로 하면 된다.



Display Name: 약속 관리 / ipad 체크 부분을 해제한다.

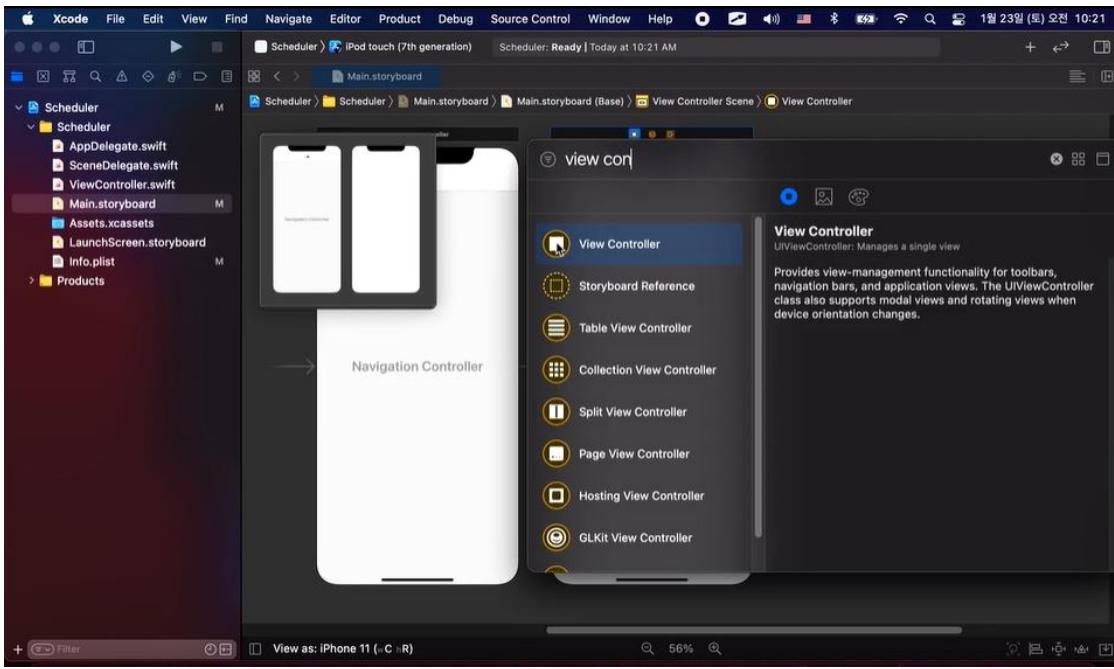


Editor->Embed In -> Navigation Controller를 클릭한다.



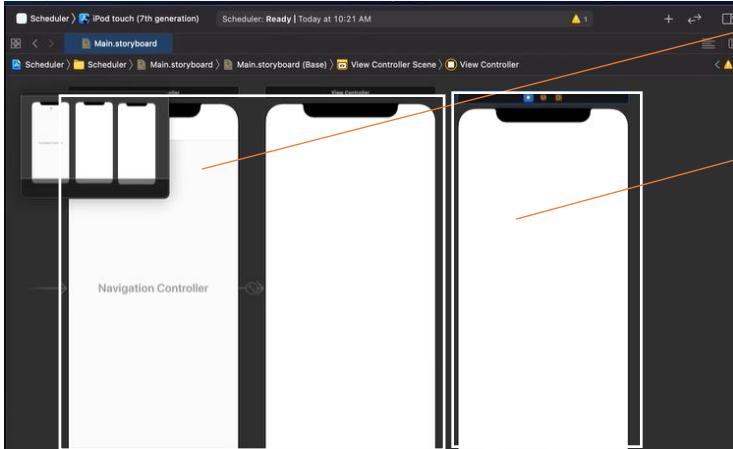
이렇게 화면이 생성이 된다.

시작화면 의미

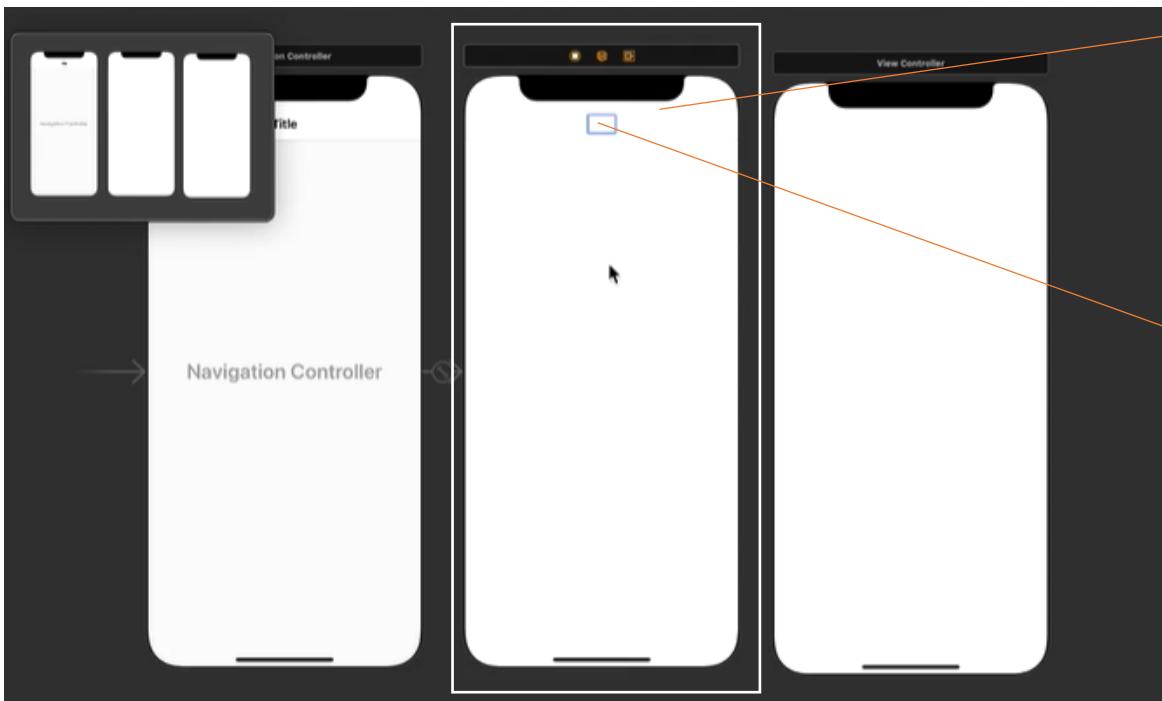


view controller 생성

이렇게 배치



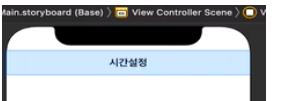
첫번째 화면은 Navigation Controller에 의해서 연결되어 있어서 제어를 받는 입장  
일반화면이다.



중앙 view에 더블 클릭한다.

Navigation bar에 제목을 설정할 수 있다.

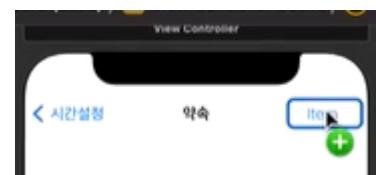
여기에 시간설정을 입력해준다.



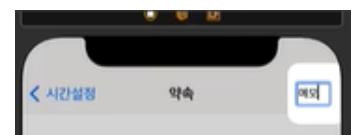




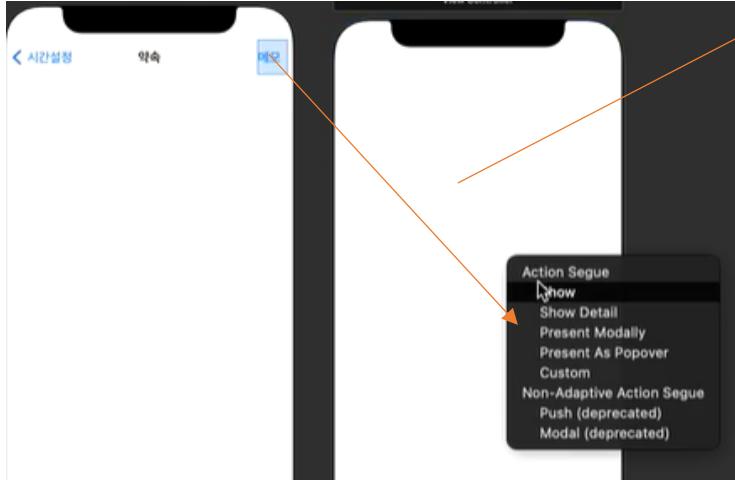
클릭 후



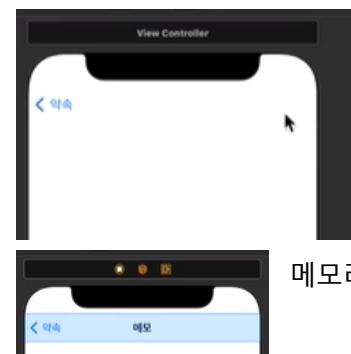
이곳에 배치한다.



버튼 내용물을 메모로 변경한다.



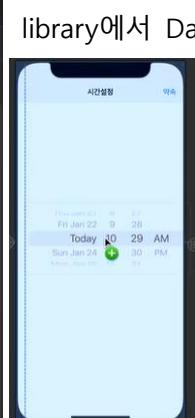
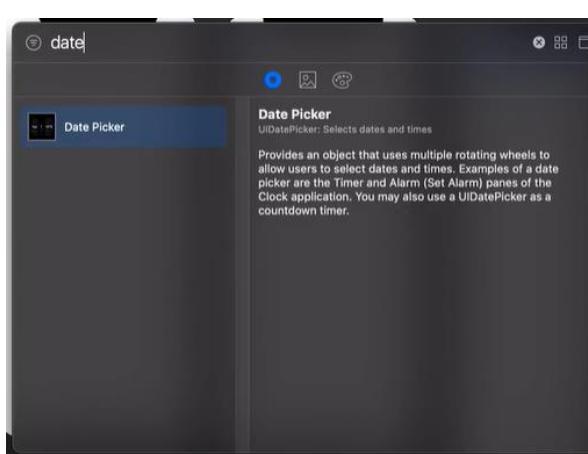
메모 버튼을 control dragging and drop를 하고, Segue 설정을 show로 한다.



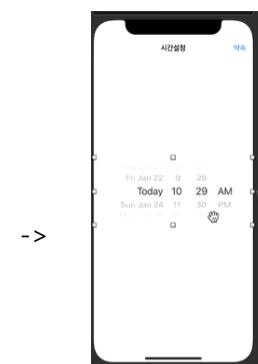
navigation controller  
에 의해 제어를 받는  
view가 된다.



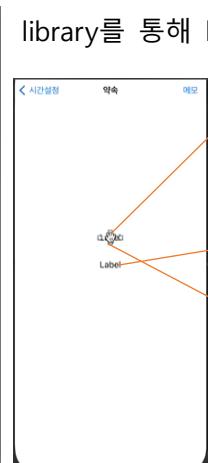
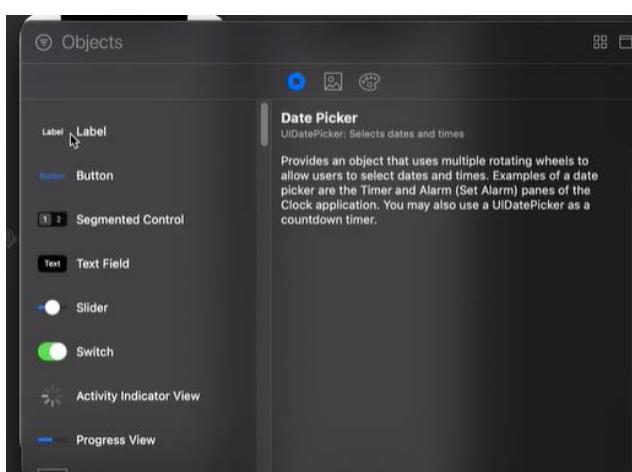
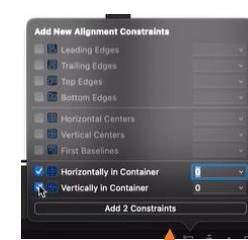
메모라고 설정을 해주면 된다.



library에서 Date Picker를 입력해서 drag를 해준다.



정렬 버튼을 클릭해서 수평,  
수직 중앙으로 해준다.



library를 통해 label 객체 2개를 가지고 와서 배치한다.

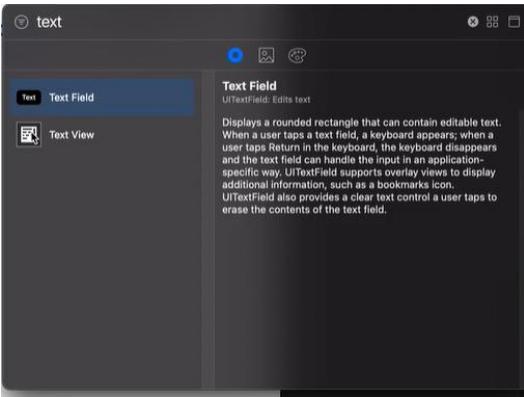
Appointment라고 Label의 내용을 편집한다.

이제 배열 조건을 주자!

이 Label 객체는 정 중앙에 배치하도록 한다.

이 Label은 Horizontally in Container 설정하고,  
수직 위치를 아래에 20만큼  
설정

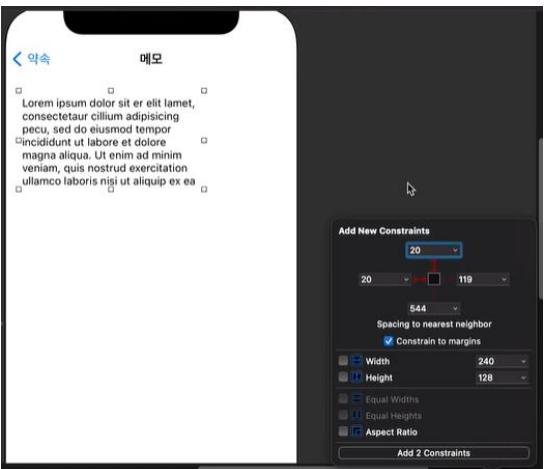




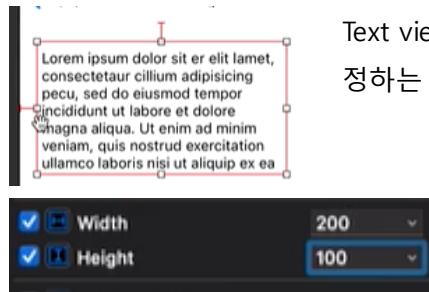
library에서 Text View 클릭하고 dragging and drop한다.

적절한 위치에 둔다.

Text View은 여러 개 text들을 입력할 수 있다.

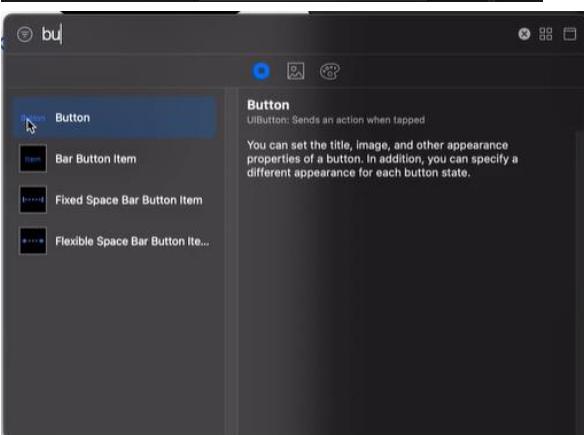


배치 설정을 다음과 같이 해준다.

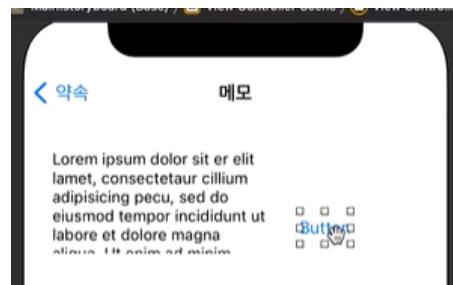


Text view인 경우는 가로, 세로 크기를 설정하는 것이 필요하다.

가로, 세로 조건을 주면 된다.

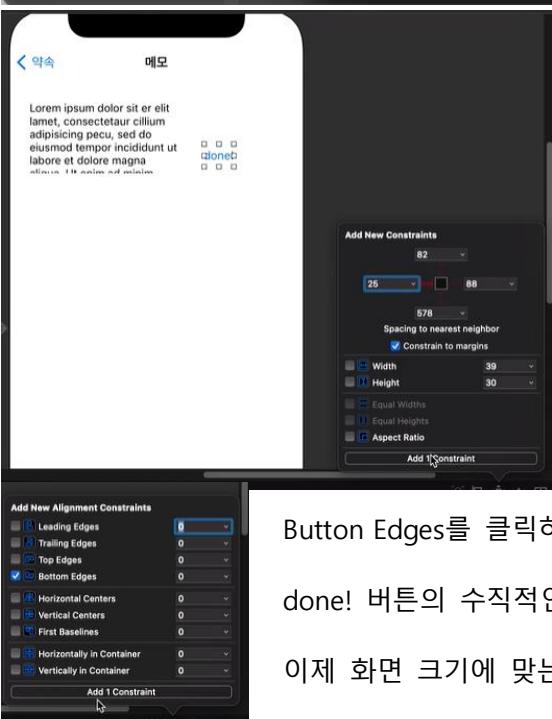


library에서 button 객체를 클릭해준다.



적당한 위치에 dragging and drop을 해준다.

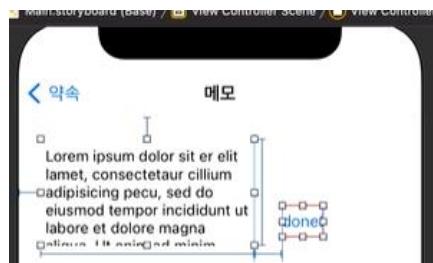
그리고 편집을 해서, done! 이라고 button이름을 변경해준다.



done! 버튼의 배치 조건을 설정해보자!

왼쪽에 25를 넣어 수평적으로 배치 조건을 설정한다.

이제, 수직 조건을 넣어주자!



shift 버튼을 눌려서, 2개 객체를 클릭해준다.

조건주는 버튼을 클릭한다.

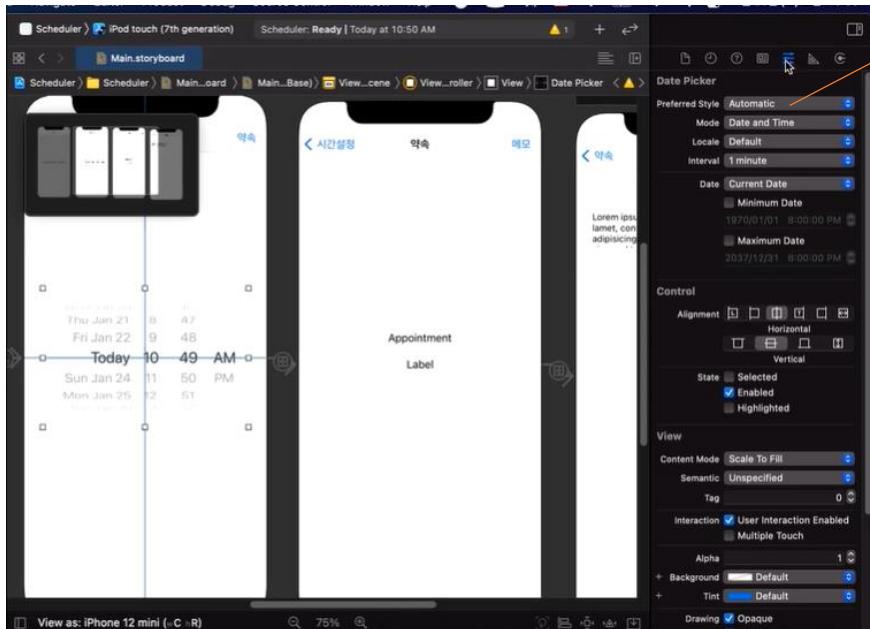
Button Edges를 클릭하여 배치 조건에 추가 시킨다.(Add 1 Constraint 클릭)

done! 버튼의 수직적인 배치 조건을 설정하게 된다.

이제 화면 크기에 맞는지를 확인해보자!

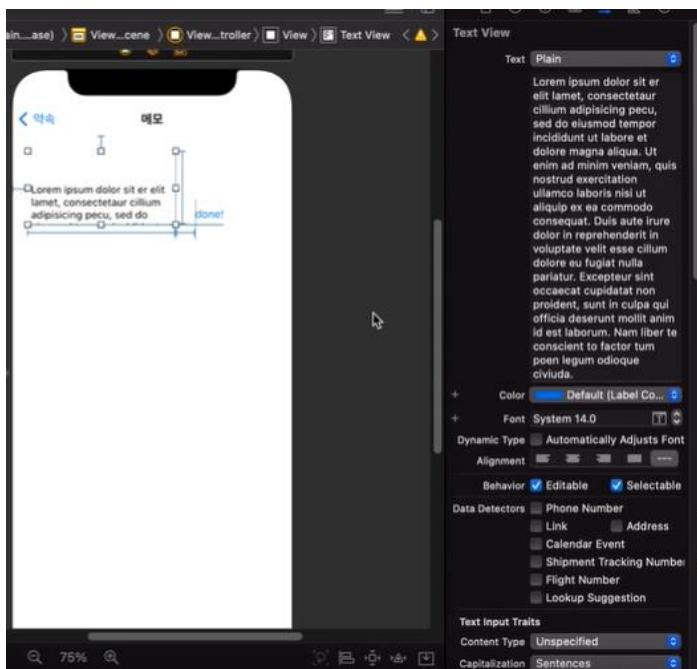
-> 가로모드 사용시 date Picker가 클릭이 안 되는 경우가 있다. 이럴 때에는 date Picker 객체를 지우고 다시 배치해준다. -> 세로모드에서 편집하자!!

Date Picker의 속성 창을 열어보자

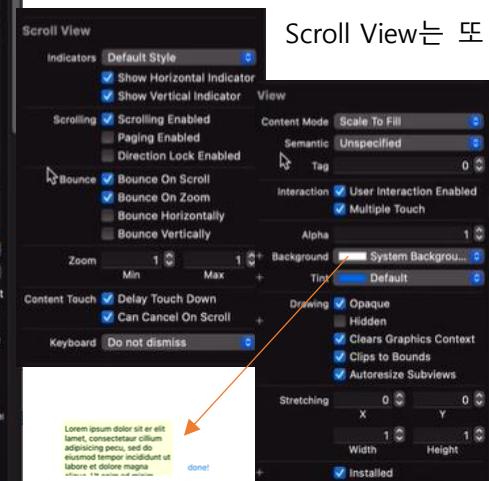


해당 ios에서 톱니바퀴처럼 할 것인지 등등의 선택권을 준다. 우리는 자동모드로 한다.

Text view의 속성창을 열어보자!



속성을 살펴보면 Text view는 Scroll View의 자식으로 되어 있다는 것을 알 수 있다.



ScrollView는 또 View의 자식이다.

view로서의 속성을 설정할 수 있다.

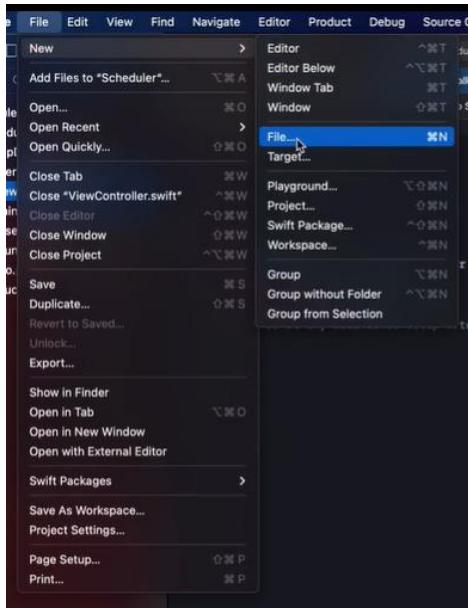
2,3 번째 view controller를 생성하는 파일들이 필요하다.

```

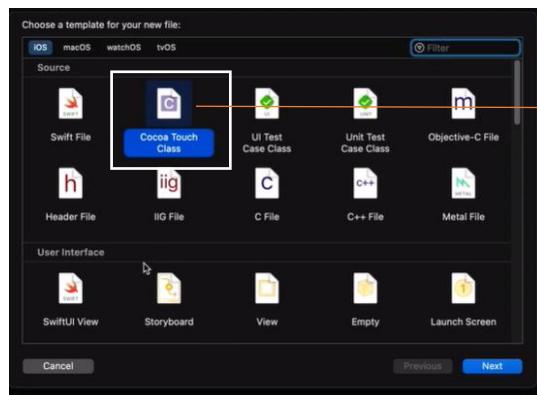
Scheduler
└── Scheduler
    ├── AppDelegate.swift
    └── ViewController.swift
    └── Main.storyboard
    └── Assets.xcassets
    └── LaunchScreen.storyboard
    └── Info.plist
    └── Products
Scheduler
└── ViewController.swift
No Selection
1 //
2 // ViewController.swift
3 // Scheduler
4 //
5 // Created by dheum on 2021/01/23.
6 //
7
8 import UIKit
9
10 class ViewController: UIViewController {
11
12     override func viewDidLoad() {
13         super.viewDidLoad()
14         // Do any additional setup after loading the view.
15     }
16
17
18 }
19
20

```

첫번째 ViewController.swift 파일이라는 것을 알 수 있다.

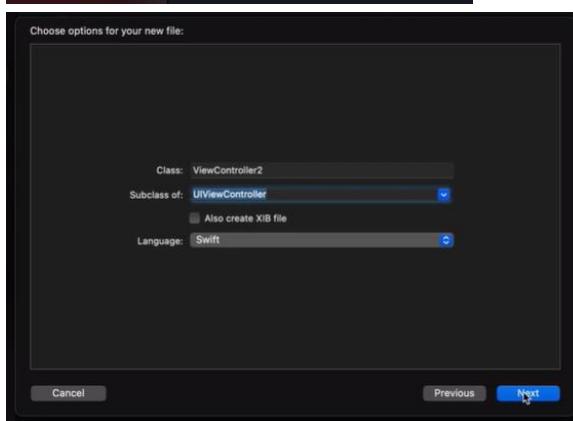


File->New->File 클릭

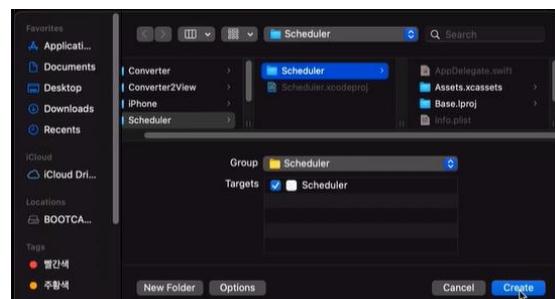


이와 같은 창이 나옴

이게 선택된 상태에서 Next를 클릭해준다.



이와 같은 창이 나오며, class 이름을 ViewController2로 해준다.  
Next를 클릭해준다.



위치 설정이 하는 곳이 나온다. 우리는 이대로 하겠다.

이와 같은 파일이 생성된다.

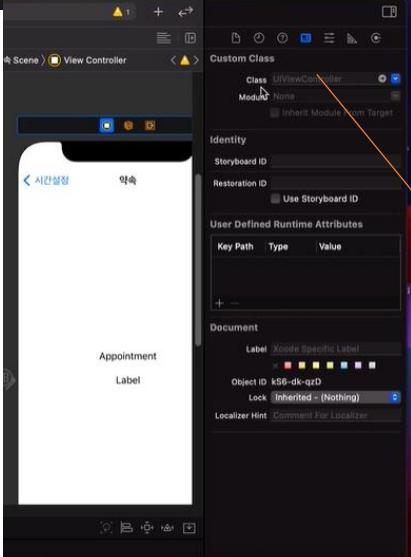
동일하게 파일 하나를 더 만든다. 그리고 파일의 이름을 ViewController3.swift라고 한다.

```

Scheduler [iPhone 12 mini] Running Scheduler on iPhone 12 mini
Main.storyboard ViewController2.swift

Scheduler [Scheduler] ViewController2.swift > No Selection
1 // ViewController2.swift
2 // Scheduler
3 // Created by dhuem on 2021/01/23.
4 //
5 import UIKit
6
7 class ViewController2: UIViewController {
8     override func viewDidLoad() {
9         super.viewDidLoad()
10        // Do any additional setup after loading the view.
11    }
12
13    // MARK: - Navigation
14
15    // In a storyboard-based application, you will often want to do a little preparation before navigation
16    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
17        // Get the new view controller using segue.destination.
18        // Pass the selected object to the new view controller.
19    }
20
21 }

```

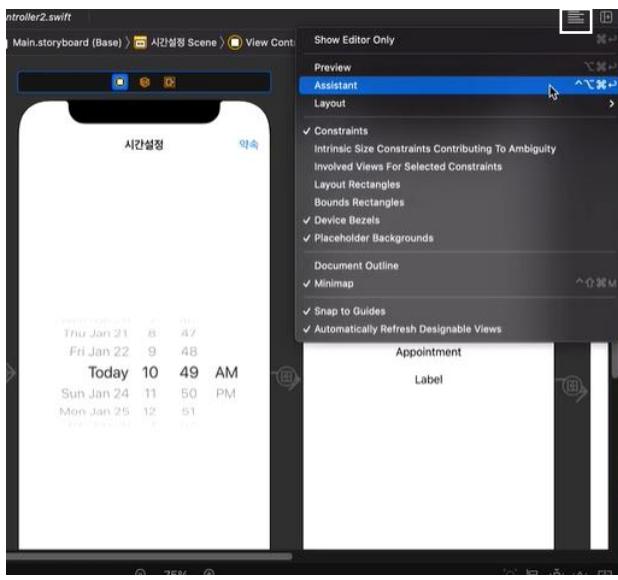


두 번째 화면과 ViewController2를 연계시켜보자.

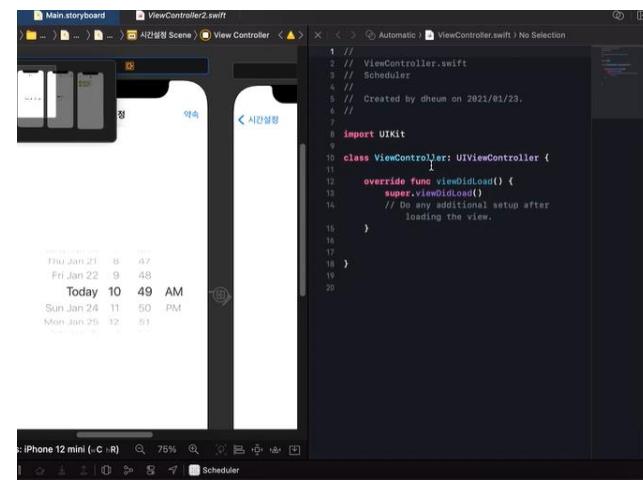
두 번째 view 속성 창을 연다.

Custom class에서 viewController2를 선택한다.

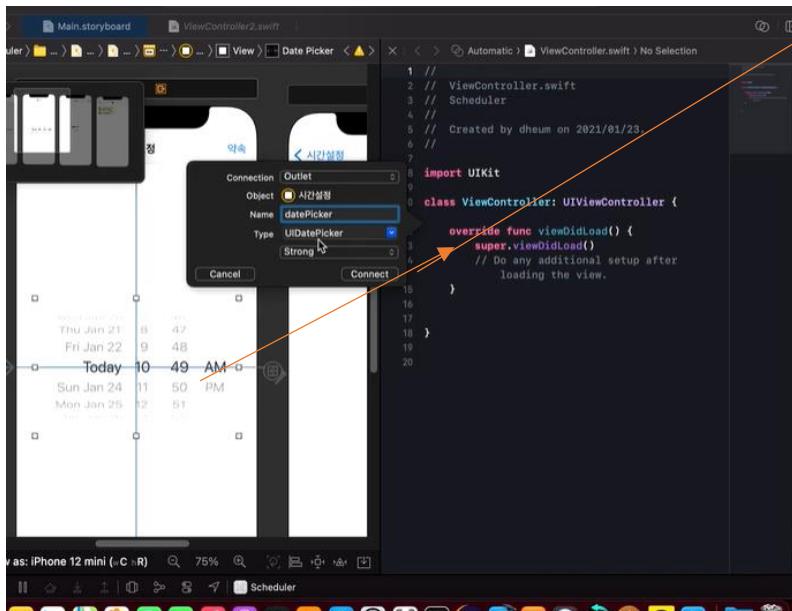
세 번째 화면도 이와 동일하게 하면 된다.



첫 번째 화면에서 view controller를 클릭하고 Assistant 클릭 한다.

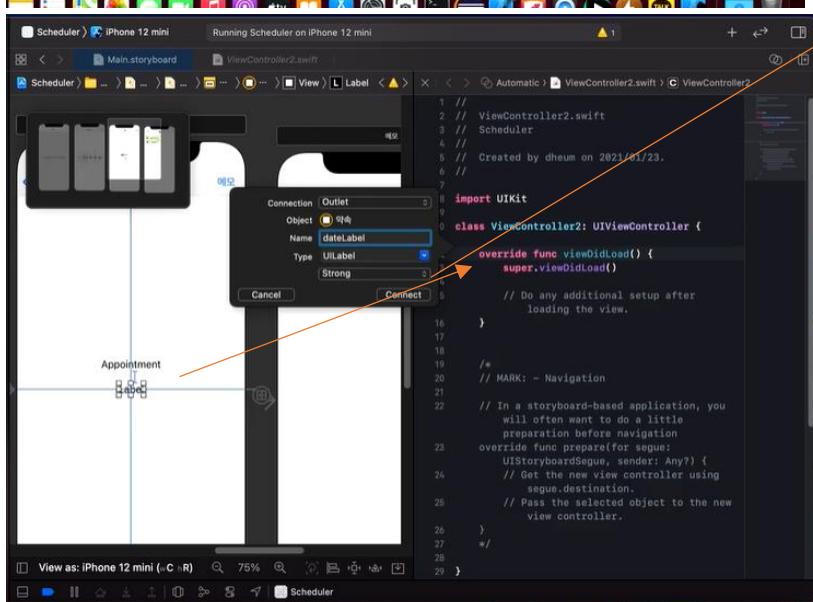


이 화면 됨.

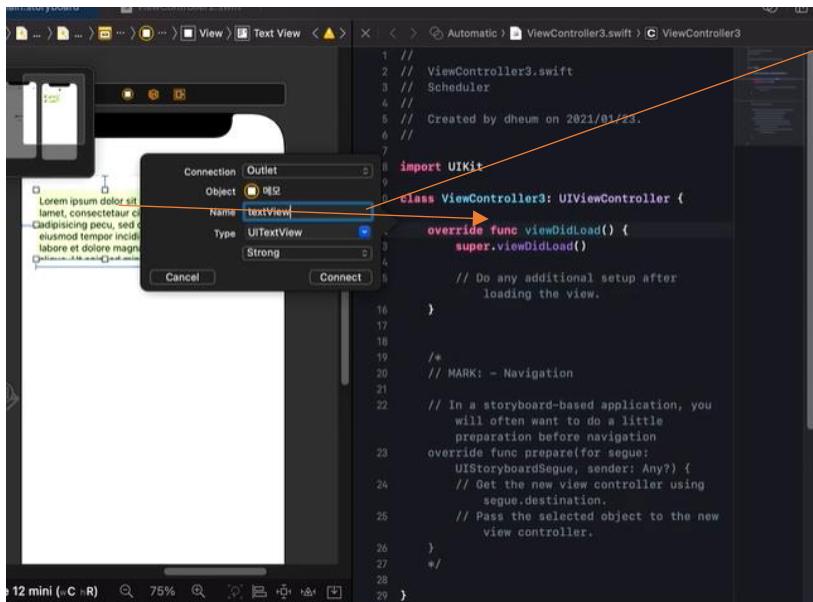


datePicker를 control dragging에서 drop하고 Name을 datePicker로 설정한다.

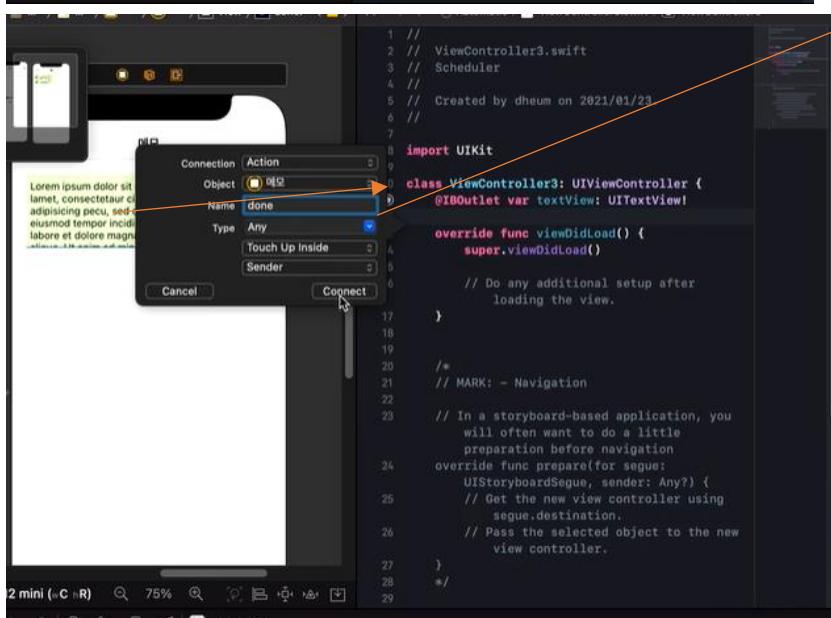
-> 코드와 인터페이스 연결이 된다.



동일하게 보조 editor를 열고 Label 객체를 control dragging을 하고, Name을 dateLabel로 변경해준다.



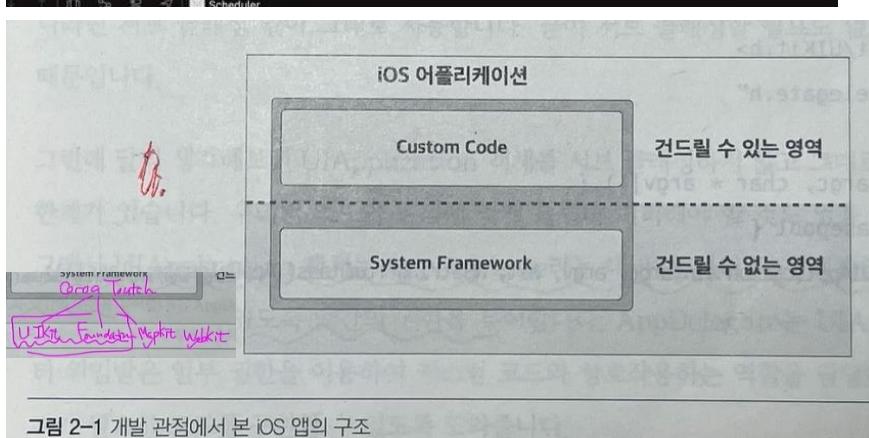
3번째 controller editor를 열고, text view 객체를 control dragging을 한다. 그리고, Name을 textView로 바꾸어준다.



Button을 control dragging을 하고, Connection 정보를 Action으로 하고, Name 을 done으로 해준다.

```
@IBAction func done(_ sender: Any) {
```

함수 형태가 만들어졌다.

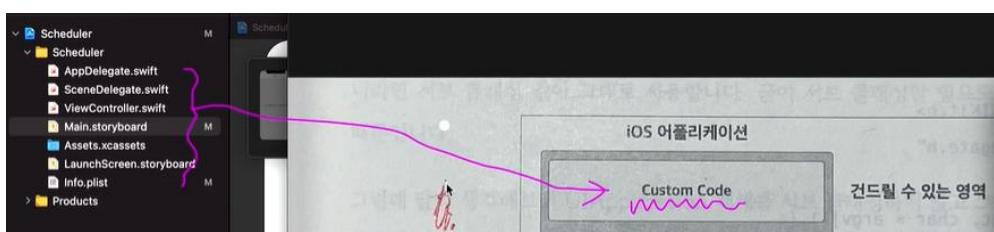


custom code와 system Framework이 유기적으로 연결이 되어, 섬세한 작업을 한다.

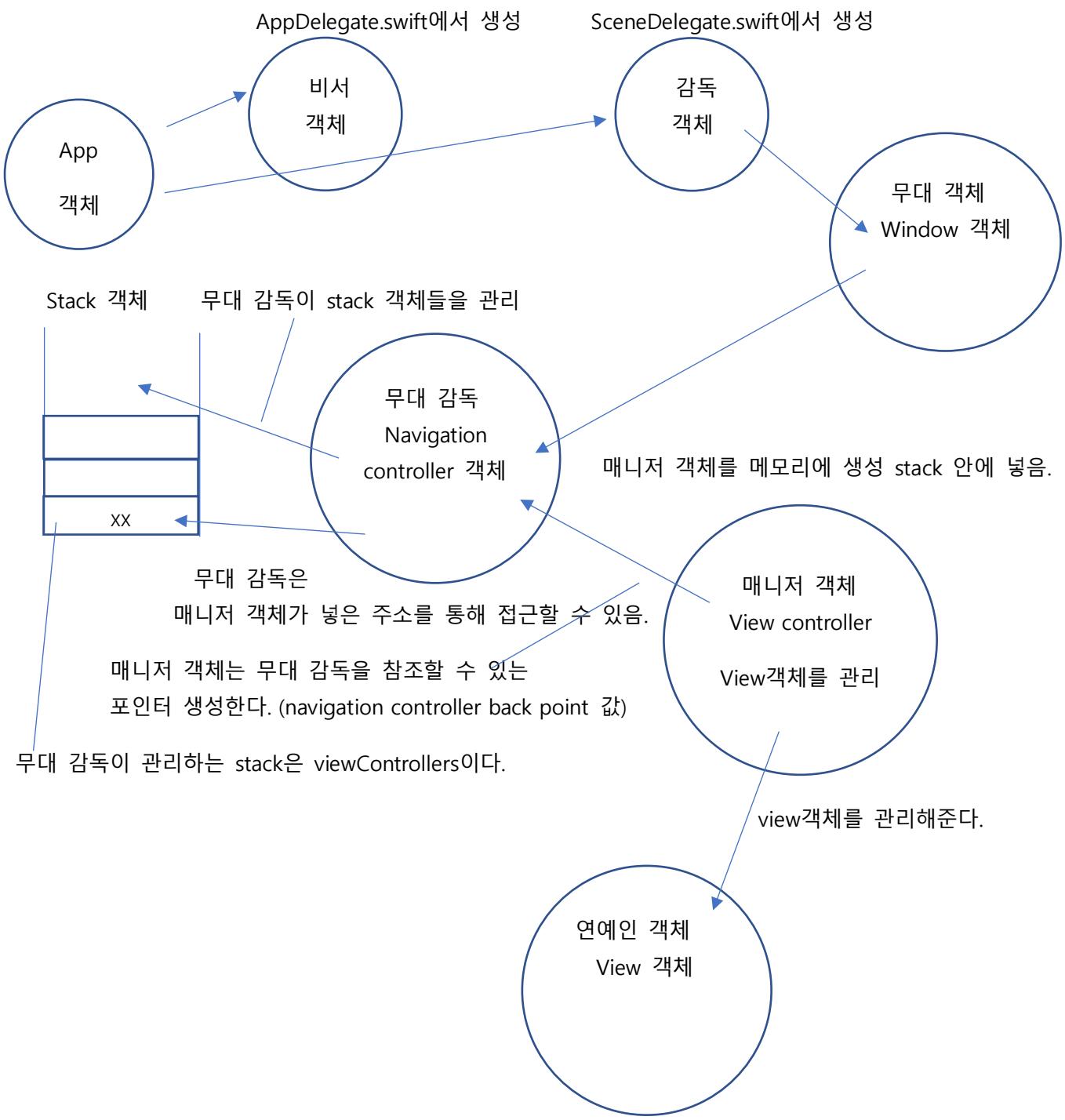
System Framework는 ios인 경우에는 Cocoa Touch이다.

Cocoa Touch는 UIKit, Foundation, MapKit(지도관련), WebKit 등등을 가지고 있다.

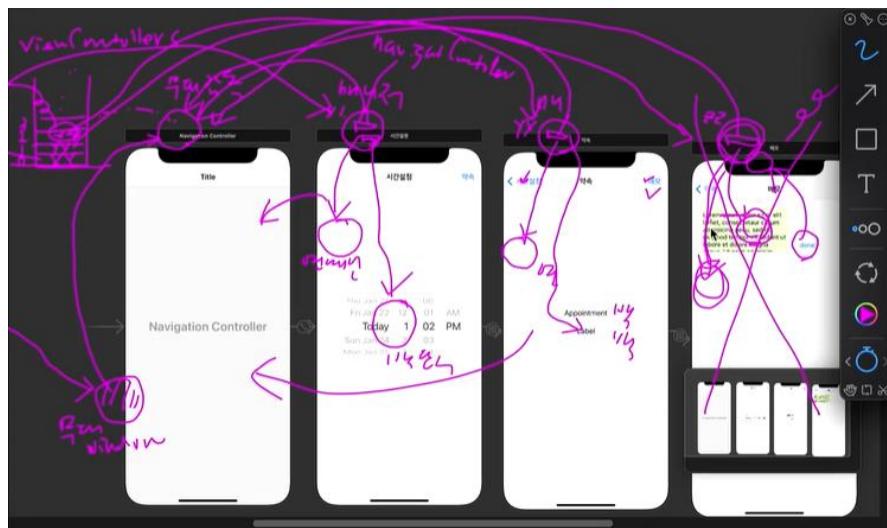
우리는 UIKit, Foundation을 반드시 포함시켜서 사용하고, Mapkit, Webkit는 선택적으로 사용한다. (but 건드리지X)



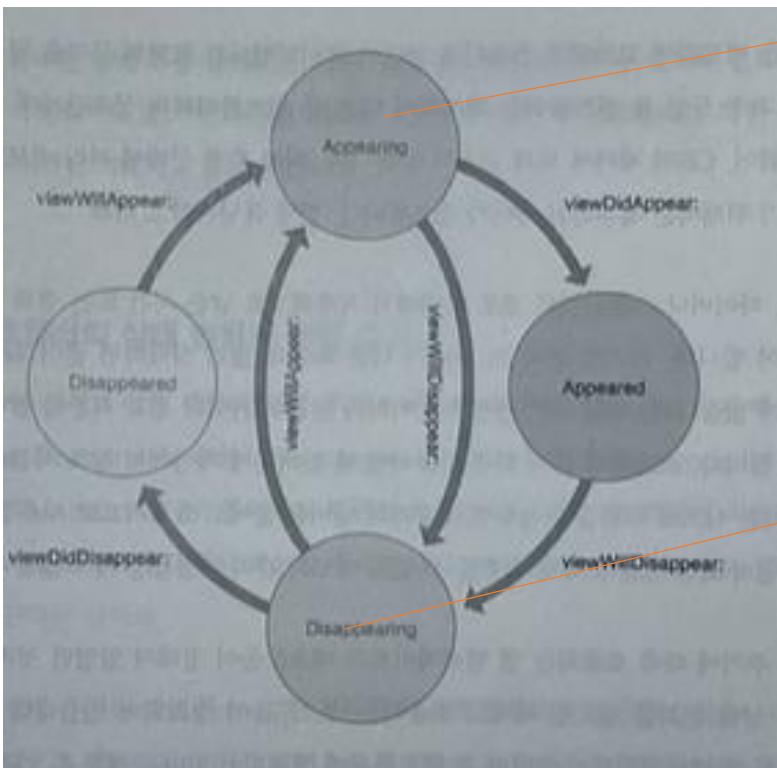
custom Code가 우리가 코딩할 수 있는 부분이다. -> 내가 건드릴 수 있는 부분이다.



시스템은 Main.storyboard에 접근해서, 사용자가 지정한 위치에 view 객체들을 생성한다.



약속 버튼을 누르면 2번째 view가 덮어쓰기 하면서 2번째 화면으로 전환되고, 메모버튼을 누르면 3번째 view가 덮어쓰기 하면서 3번째 화면으로 전환된다. 그러나 약속버튼을 누르게 되면 3번째 view가 pop이 되고, 2번째 화면이 나오게 된다. 시간설정버튼을 누르게 되면 2번째 view가 pop이 되고, 첫번째 화면으로 나오게 된다.



2번째 view는 3번째 view가 사라지게 되면서, appearing 상태가 된다.

3번째 화면에서 약속 버튼을 누르게 되면 disappearing 애니메이션 효과가 생기게 되면서, 서서히 사라지다가, 3번째 view는 disappeared가 된다.

그림 2-24 뷰 컨트롤의 상태 변화에 따른 API -> view controller의 lifecycle이다.

#### ViewDidLoad 메소드



메모 버튼을 누르면 3번째 view controller 객체, view 객체들 생성 그리고 view controller와 view 객체들을 연결한다.(메소드 등등.. 이런 작업들..) 이와 같은 작업들이 다 완성이 되면, viewDidLoad 함수가 호출이 된다.

즉, viewDidLoad 함수는 viewController가 메모리에 적재가 되었다면, 시스템에 의해서 호출이 된다.

Appearing -> Disappearing 애니메이션 효과가 일어날 때가 있다. -> 사용자에 의해서

처음에 마음에 들어서 dragging하다가 별로 라서 drop할 때,

Ex)

```

import UIKit

class ViewController: UIViewController {
    @IBOutlet var datePicker: UIDatePicker!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }
}

```

viewDidAppear를 작업을 하고 싶을 때,

```

    override func viewDidAppear(_ animated: Bool) {
    }

```

이렇게 메소드를 호출하면 된다.

우리는 다음과 같은 메소드를 사용할 것이다. -> ViewController.swift/ViewController2.swift/ViewController3.swift에 넣어준다.

```
override func viewDidLoad() {  
  
    super.viewDidLoad()  
  
    // Do any additional setup after loading the view, typically from a nib.  
  
}
```

```
override func viewWillAppear(_ animated: Bool) {
```

```
    super.viewWillAppear(animated)
```

```
    print("view1 나 들어 가요")
```

```
}
```

```
override func viewDidAppear(_ animated: Bool) {
```

```
    super.viewDidAppear(animated)
```

```
    print("view1 나 들어 왔어요")
```

```
}
```

```
override func viewWillDisappear(_ animated: Bool) {
```

```
    super.viewWillDisappear(animated)
```

```
    print("view1 나 나가요")
```

```
}
```

```
override func viewDidDisappear(_ animated: Bool) {
```

```
    super.viewDidDisappear(animated)
```

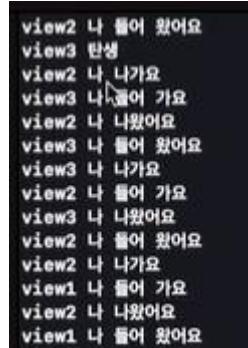
```
    print("view1 나 나왔어요")
```

```
}
```

-> view controller의 lifecycle를 알 수 있음.

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    // Do any additional setup after loading the view, typically from a nib.  
    print("view2 탄생")  
}  
override func viewWillAppear(_ animated: Bool) {  
    super.viewWillAppear(animated)  
    print("view2 나 들어 가요")  
}  
override func viewDidAppear(_ animated: Bool) {  
    super.viewDidAppear(animated)  
    print("view2 나 들어 왔어요")  
}  
override func viewWillDisappear(_ animated: Bool) {  
    super.viewWillDisappear(animated)  
    print("view2 나 나가요")  
}  
override func viewDidDisappear(_ animated: Bool) {  
    super.viewDidDisappear(animated)  
    print("view2 나 나왔어요")  
}
```

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    // Do any additional setup after loading the view, typically from a nib.  
    print("view3 탄생")  
}  
override func viewWillAppear(_ animated: Bool) {  
    super.viewWillAppear(animated)  
    print("view3 나 들어 가요")  
}  
override func viewDidAppear(_ animated: Bool) {  
    super.viewDidAppear(animated)  
    print("view3 나 들어 왔어요")  
}  
override func viewWillDisappear(_ animated: Bool) {  
    super.viewWillDisappear(animated)  
    print("view3 나 나가요")  
}  
override func viewDidDisappear(_ animated: Bool) {  
    super.viewDidDisappear(animated)  
    print("view3 나 나왔어요")  
}
```



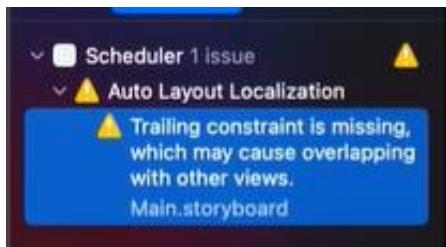
view2 나 들어 왔어요  
view3 탄생  
view2 나 나가요  
view3 나 들어 가요  
view3 나 나왔어요  
view2 나 나왔어요  
view3 나 들어 왔어요  
view3 나 나가요  
view2 나 들어 가요  
view3 나 나왔어요  
view2 나 들어 왔어요  
view2 나 나가요  
view1 나 들어 가요  
view2 나 나왔어요  
view1 나 들어 왔어요

SceneDelegate.swift는 감독 객체를 만드는 코드이다.

참고) scene(\_:willConnectTo:options)이와 같은 형식으로 사용한다.

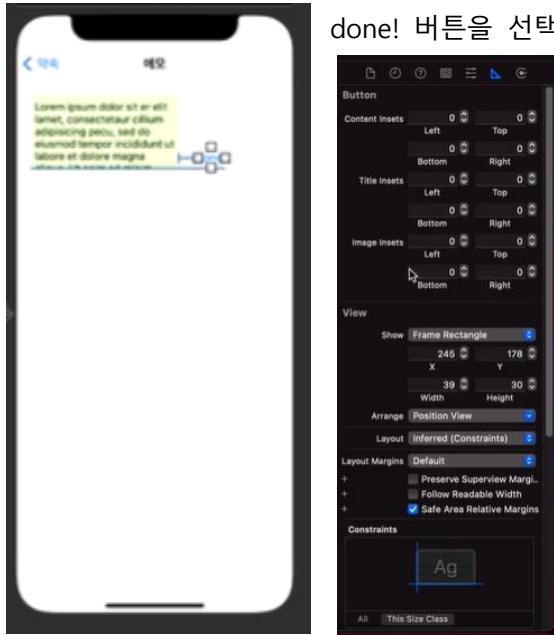
AppDelegate.swift는 비서 객체를 만드는 코드로, 메모리 warning이나, app이 실행되거나 종료 그리고 background되는 상태일 때의 메소드를 가지고 있다. Ex) 예전에 application 메소드 안에 sleep(2)을 사용한 적 있음.

Iphone 4번째 강의



이 부분 경고를 없애기

Done! 버튼 뒤에 있는 객체를 두면 overlapping이 되는 경고창이다.



3번째 Scene의 계층 구조를 보자!

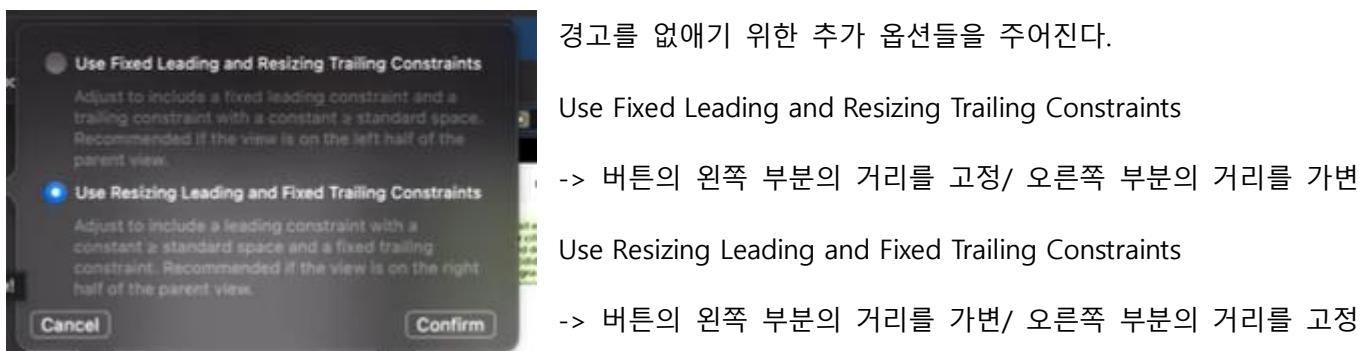


done! 버튼이 클릭된 상태에서



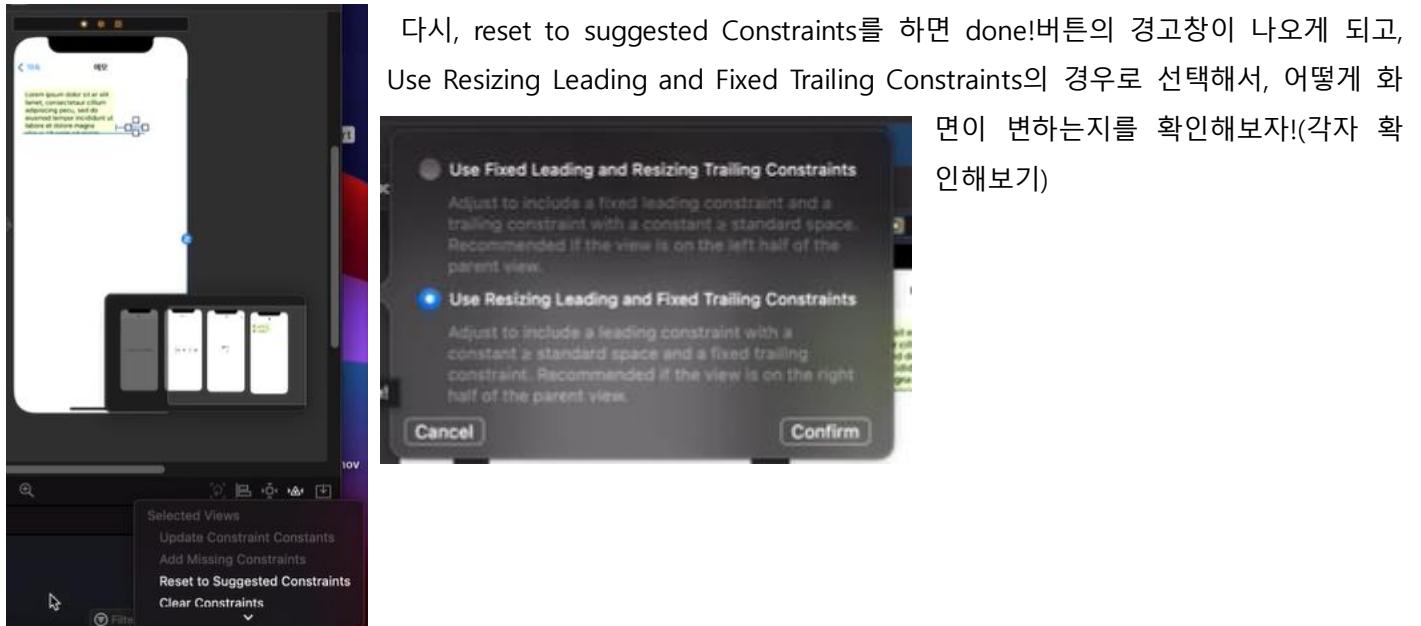
issue 버튼을 클릭하고, Reset to suggested Constraints 버튼을 누른다.

이와 같이 설정이 된다. Text View로부터 얼마큼 떨어져 있는지와 base line이 설정된다. 그러나 아직도, 경고창이 있다.



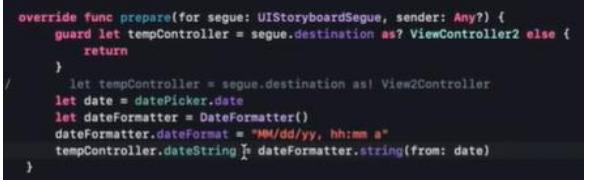
-> 교수님께서는 첫번째 경우를 선택하심.

-> 둘 중에 하나를 선택하게 되면 경고창이 사라지게 된다.



## ViewController.swift에

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
  
    guard let tempController = segue.destination as? ViewController2 else {  
  
        return  
  
    }  
  
    //    let tempController = segue.destination as! View2Controller  
  
    let date = datePicker.date  
  
    let dateFormatter = DateFormatter()  
  
    dateFormatter.dateFormat = "MM/dd/yy, hh:mm a"  
  
    tempController.dateString = dateFormatter.string(from: date)  
  
}
```



이와 같은 코드를 넣어준다. -> tempController.dateString = dateFormatter.string(from: date) 부분에 오류가 생김

## ViewController2.swift에

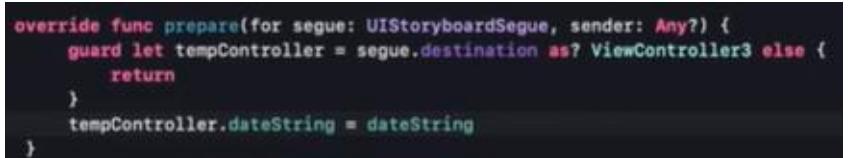
var dateString: String? 변수를 선언

그렇게 하면 tempController.dateString = dateFormatter.string(from: date) 의 오류가 사라진다.

//////////

## ViewController2.swift에

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
  
    guard let tempController = segue.destination as? ViewController3 else {  
  
        return  
  
    }  
  
    tempController.dateString = dateString  
  
}
```



이와 같은 코드를 넣어준다.

그러면 tempController.dateString = dateString 이 부분에서 오류가 발생한다.

이 오류를 없애기 위해 ViewController3.swift에 var dateString: String? 변수를 선언해준다.

그러면 tempController.dateString = dateString 이 부분의 오류가 사라진다.

ViewController3.swift에서 done 메소드 안에

```
textView.resignFirstResponder()
```

```
guard let controller = self.navigationController,
```

```
@IBAction func done(_ sender: Any) {
    textView.resignFirstResponder()
    guard let controller = self.navigationController,
          let tempController = controller.viewControllers[1] as? ViewController2 else {
        return
    }
    tempController.dateString = textView.text
}
```

```
let tempController = controller.viewControllers[1] as? ViewController2 else {
```

```
    return
}
```

```
tempController.dateString = textView.text
```

이 같은 내용을 넣어준다.

//////////

ViewController2.swift에서

viewWillAppear 메소드 안에

dateLabel.text = dateString 부분을 넣어준다.

ViewController3.swift에서

viewWillAppear 메소드 안에

textView.text = dateString 부분을 넣어준다.

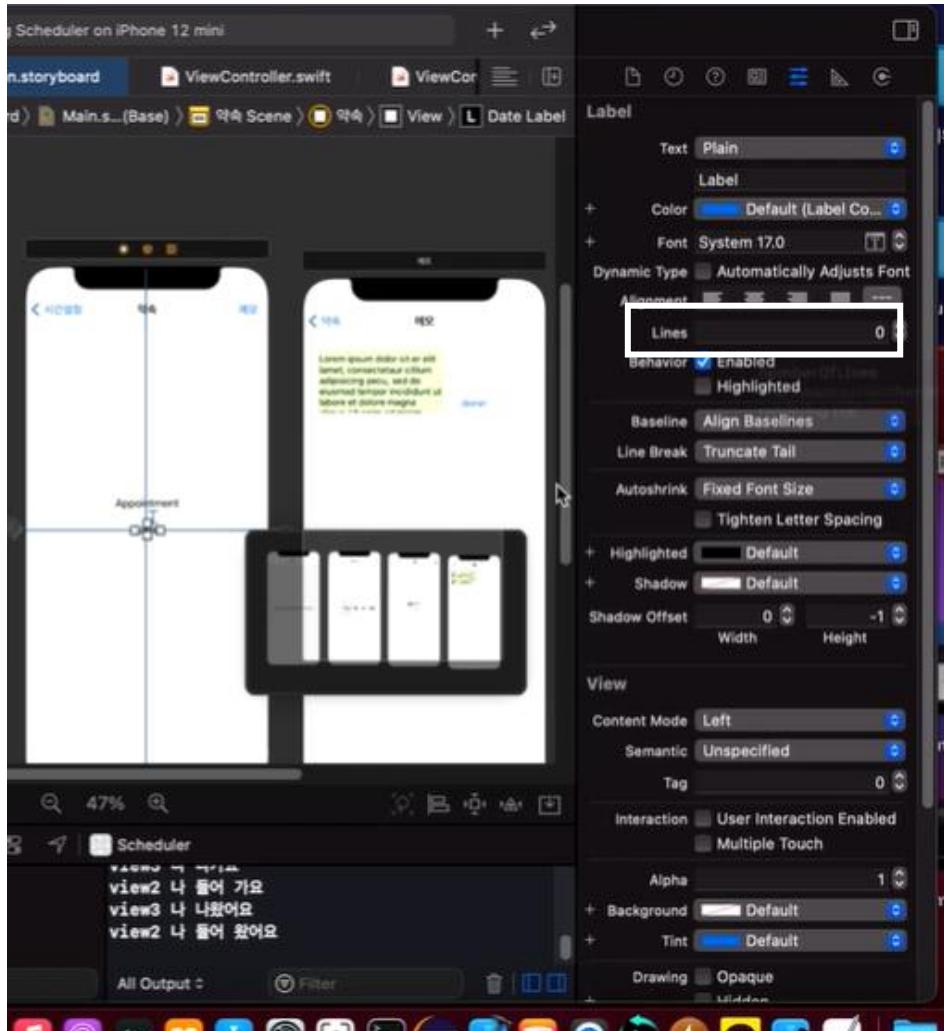
잘 실행이 되고 있다는 것을 알 수 있다.



View 간의 데이터가 공유 되고 있다는 것을 알 수 있다. / done!버튼이 제대로 작동X

Main.storyboard로 들어가서

View 2번째에 있는 Label의 속성창을 열어줍니다.

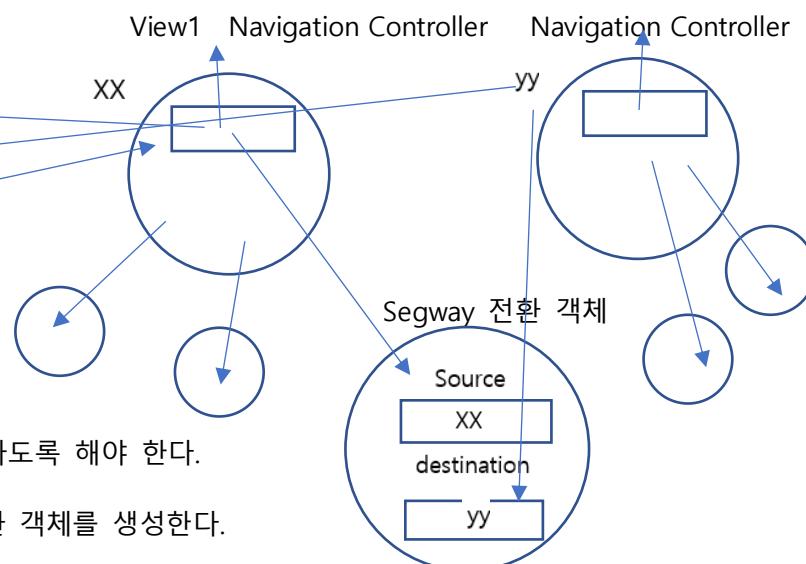
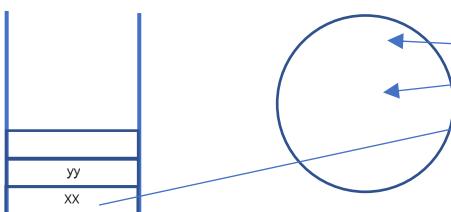


Lines에 0으로 설정

그래서 여러 줄로 setting할 수 있게 된다.

done! 버튼이 이제 잘 작동이 되고 있다는 것을 알 수 있다.

ViewControllers stack    NavigationController



약속버튼을 누르게 되면 Segway 전환이 일어나도록 해야 한다.

Segway전환이 일어나기 위해서는 Segway 전환 객체를 생성한다.

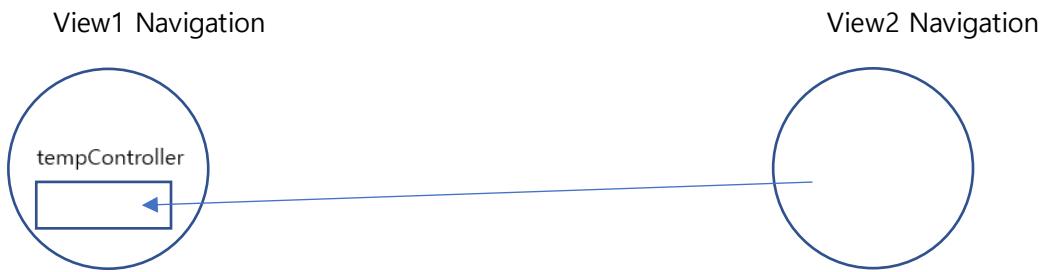
Segway 전환 시 View1에 있는 prepare 함수를 호출하게 된다. 그 다음에 View2에 viewDidLoad 함수가 호출된다.

그 후에 View1에 있는 viewWillDisappear 함수가 호출된다. 등등 시스템 프레임워크에서 일어난 작업들이다.

-> 이와 같은 함수를 이용해서 아이폰 앱을 구성하는 것이다.

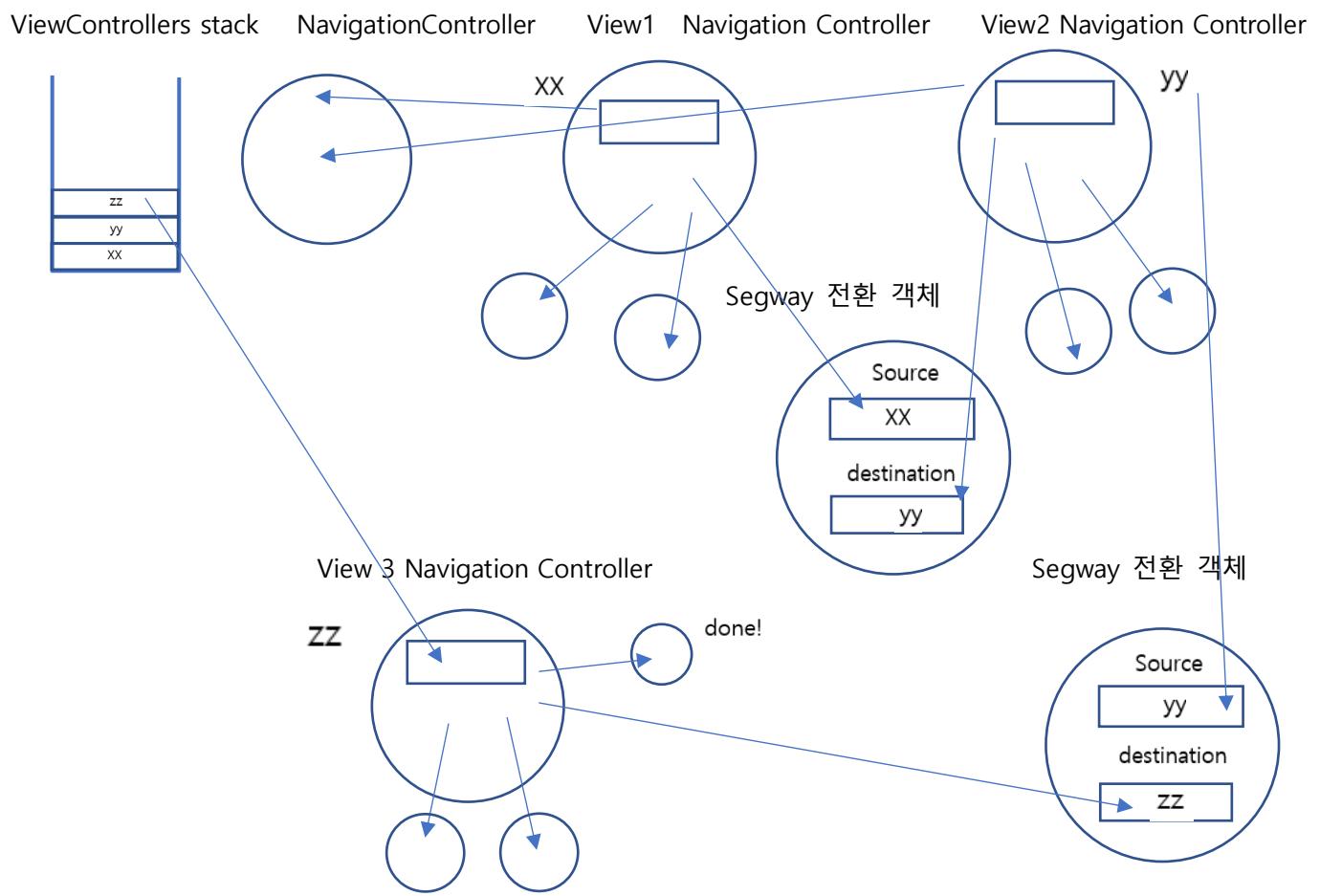
## viewController.swift 코드 일부분

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
  
    guard let tempController = segue.destination as? ViewController2 else {  
  
        // 타임 변환에 성공 시 tempController 안에 넣게 되고, 실패 시 null값으로 들어가게 된다.  
  
        // 타임 변환이 실패가 되면, return이 된다.  
  
        return  
  
    }  
}
```



View2 controller 주소 값을 tempController라는 지역변수 안에 넣게 된다.

```
// let tempController = segue.destination as! ViewController2  
  
let date = datePicker.date //datePicker를 통해 date형의 값을 date변수 안에 넣는다.  
  
let dateFormatter = DateFormatter() //date형의 값을 문자열로 바꾸기 위해, DateFormatter()라는 함수를  
// 통해 date 문자열 형태를 설정할 수 있는 변수를 받아온다.  
  
dateFormatter.dateFormat = "MM/dd/yy, hh:mm a" // date형 문자열 형태를 이와 같이 세팅해준다.  
  
tempController.dateString = dateFormatter.string(from: date) //date형의 문자열이 위에서 설정한 형태로  
// 나온다. 그리고는 view2 controller의 변수  
// 인 dateString 안에 넣어준다.  
}  
  
override func viewWillAppear(_ animated: Bool){  
  
    super.viewWillAppear(animated)  
  
    dateLabel.text = dateString  
}
```



## ViewController2.swift 일부분

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
```

```
guard let tempController = segue.destination as? ViewController3 else {
```

return //tempController 라는 변수 안에 ViewController3 주소를 넣어준다. 실패 시 return해준다.

}

tempController.dateString = dateString //3번째 View안에 있는 tempController의 변수인 dateString 안에 view1에서 받은 값을 넣어준다.

}

### ViewController3.swift 일부분

```
@IBAction func done(_ sender: Any){

    textView.resignFirstResponder()

    guard let controller = self.navigationController, //self.navigationController값이 nil인지 확인

        let tempController = controller.viewControllers[1] as? ViewController2 else {

        //tempController 안에 2번째 View 값을 넣어주게 된다.

        return //2번째 View가 nil인 경우에는 return해준다.

    }

    tempController.dateString = textView.text //2번째 view에 있는 dateString 변수 값을 textView의 내용으로
                                                바꾸어 준다.

}
```

### ViewController2.swift 일부분

```
override func viewWillAppear(_ animated: Bool){

    super.viewWillAppear(animated)

    print("view2 들어가요.")

    dateLabel.text = dateString

}
```

이 함수에 인해서 다시 view2로 돌아갈 때 view3에서 변경된 내용이 view2의 Label에 넣어주게 된다.

ViewController2.swift에서 viewDidLoad() 내용을 다음과 같이 바꾸어 준다.

```
override func viewDidLoad() { //처음 viewController를 생성할 때만, alert창이 뜨는 것을 원하기 때문에
                                //viewDidLoad 함수안에 alert 창 생성 코드를 넣어주었다.
```

```
super.viewDidLoad()

// Do any additional setup after loading the view, typically from a nib.

print("view2 탄생")

let c: UIAlertAction -> Void = { action in self.dateLabel.text = "Good!!" } //클로저 형태의 함수이다.

// in이라는 키워드는 함수의 parameter와 함수의 body를 구분하기 위해서 넣어주는 것이다.

let alertController = UIAlertController(title: "약속시간",

                                         message: dateString, preferredStyle: UIAlertController.Style.actionSheet)

//alertController는 View창이기 때문에 UIAlertController함수를 통해 만들게 된다.
```

```

// preferredStyle: UIAlertController.Style.actionSheet에서 action으로 바꾸어 주면, 팝업창이 뜨게 된다.

// 우리는 그냥 actionSheet로 해줄 것이다...

let alertAction = UIAlertAction(title: "Yes", style: UIAlertAction.Style.default, handler: c)

//Yes버튼과 No버튼을 UIAlertAction()를 통해서 만들어지게 된다. -> Yes 버튼을 클릭하게 되면, c 변수
행위가 벌어지게 된다. (c행위를 하기 위해 handler파라미터에 c를 넣어준다.)

이때, Yes를 하게 되면 View controller2 Label의 값이 Good!!으로 바뀌게 된다.

alertController.addAction(alertAction)

/*
alertController.addAction(UIAlertAction(title: "No", style: .cancel,
                                         handler: { action in
    self.dateLabel.text = "Oops!"
    self.navigationController?.popViewController(animated: true)
})) //no버튼이 UIAlertAction생성하고, style의 파라미터의 값이 cancel
임을 알 수 있다. Handler의 파라미터 값을 보면 viewController2
가 pop이 된다는 것을 알 수 있다.

*/



alertController.addAction(UIAlertAction(title: "No", style: .cancel) { action in
    self.dateLabel.text = "Oops!"
    self.navigationController?.popViewController(animated: true)
})

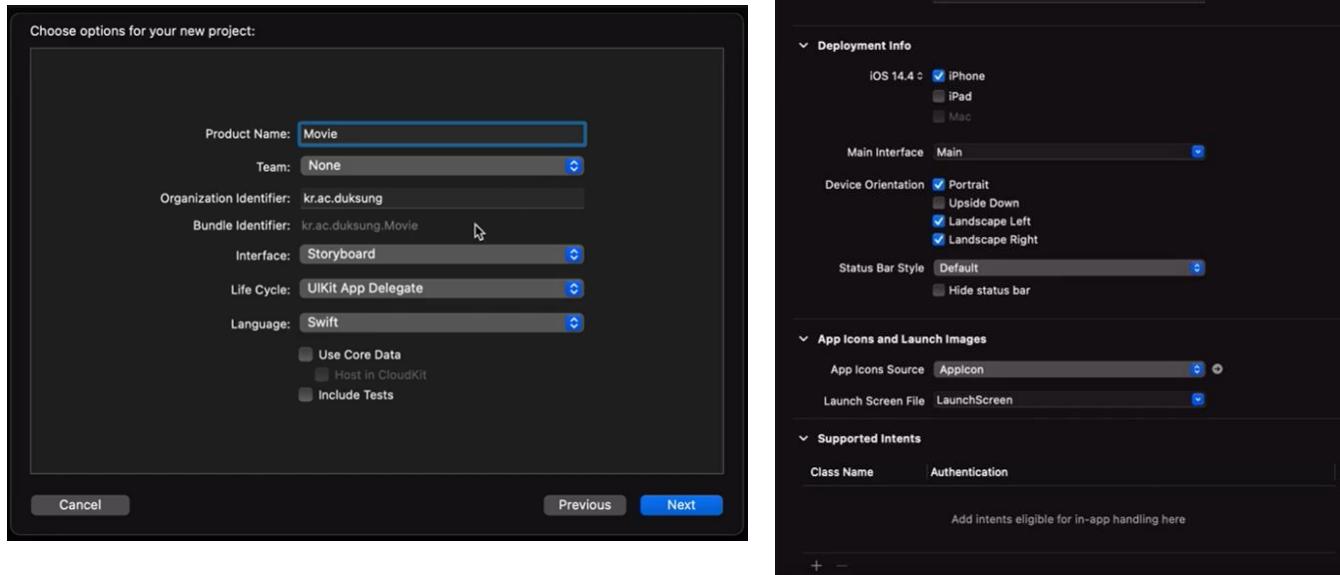
self.present(alertController, animated: true, completion: { () in print("alert shown...") } )

//위에서 생성된 alertController를 생성해달라는 것이며, animation을 해준다.

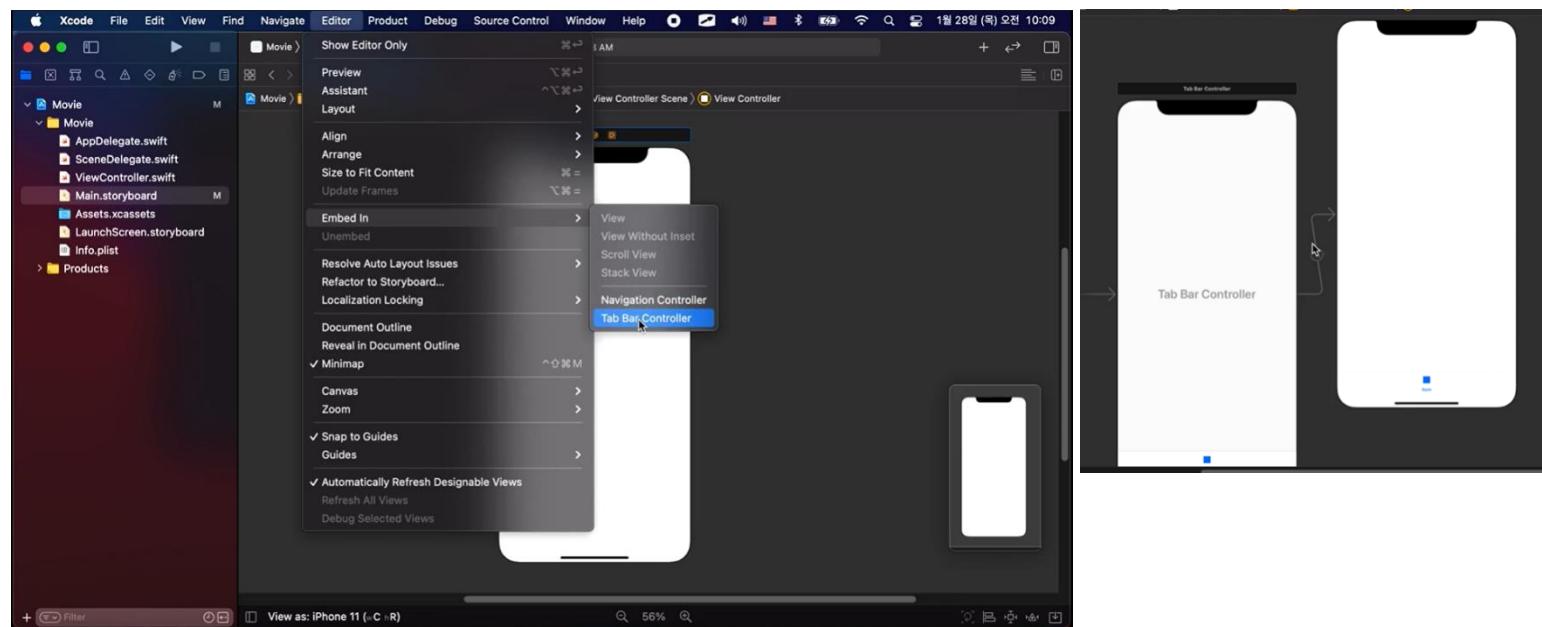
}

```

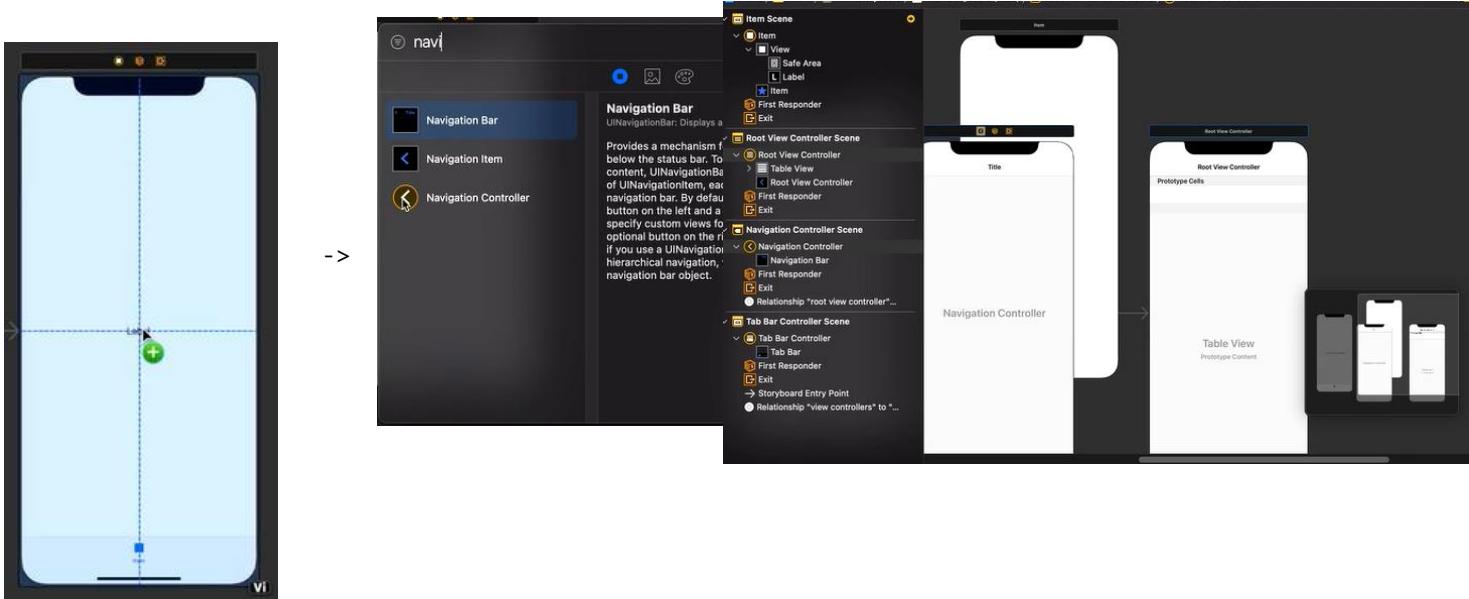
## 기본적인 설정



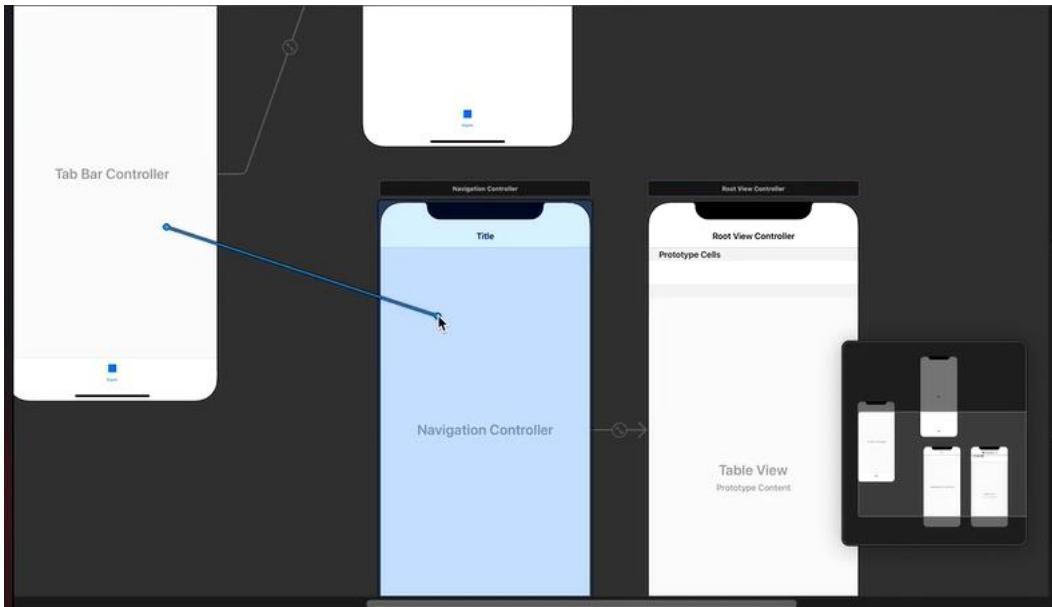
Editor -> Embed in -> Tab Bar Controller를 선택



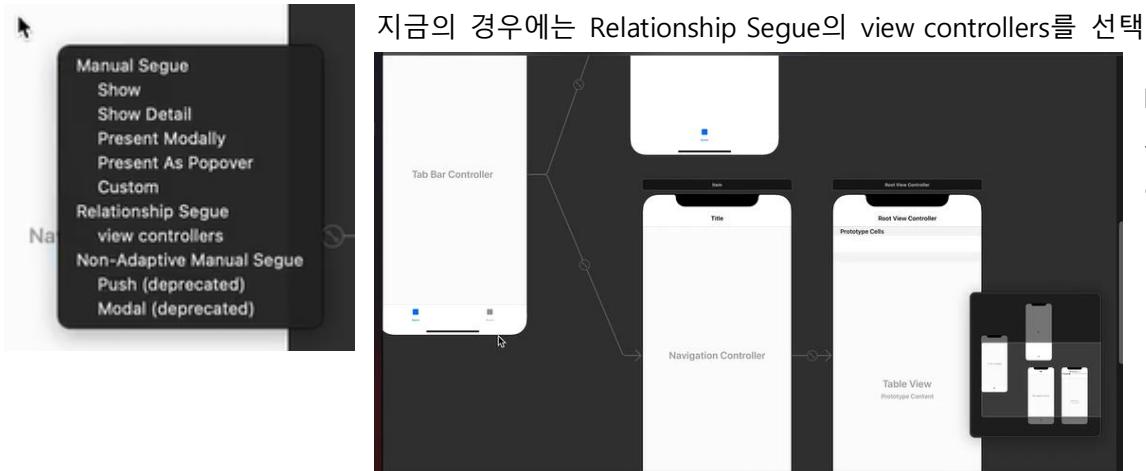
Library에서 Label을 배치 하는 데, Library에서 Navigation controller를  
적절한 위치에 둔다.  
이와 같은 가이드라인이 보일 때, -> dragging해서 꺼냅니다.  
->  
drop한다.



Tab Bar Controller부터 dragging을 시작해서 Navigation Controller에 drop을 한다.



Segue way 종류를 어떤 것으로 할지를 나타나는 팝업창이 뜬다.



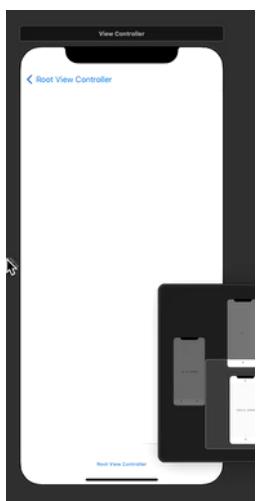
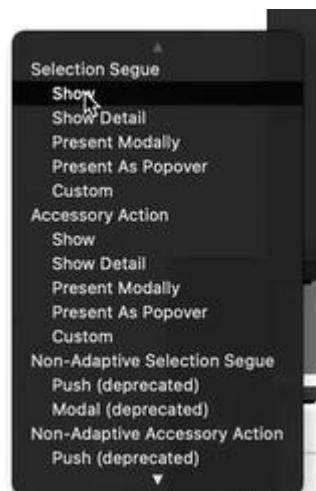
Relationship segue가 생성되었다는 것을 알 수 있다.

그 후에 view Controller를 배치시킨다. 그 다음에 Prototype Cells에서 control를 누른 상태에서 dragging을 해서 새로 배치한 view Controller에 drop을 시킨다.



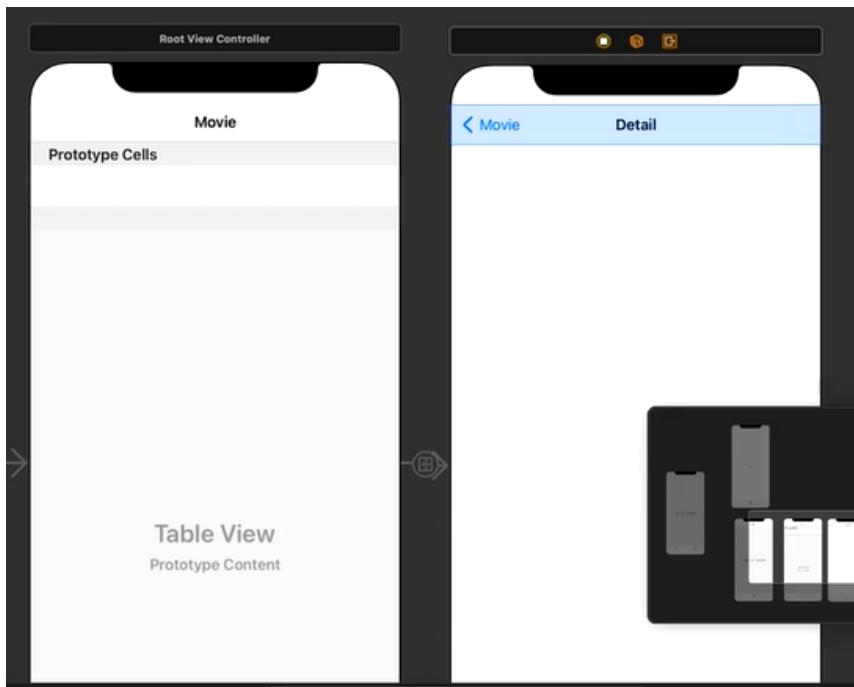
어떤 종류의 Segue 할 것인지를 선택

우리는 Show를 선택한다.

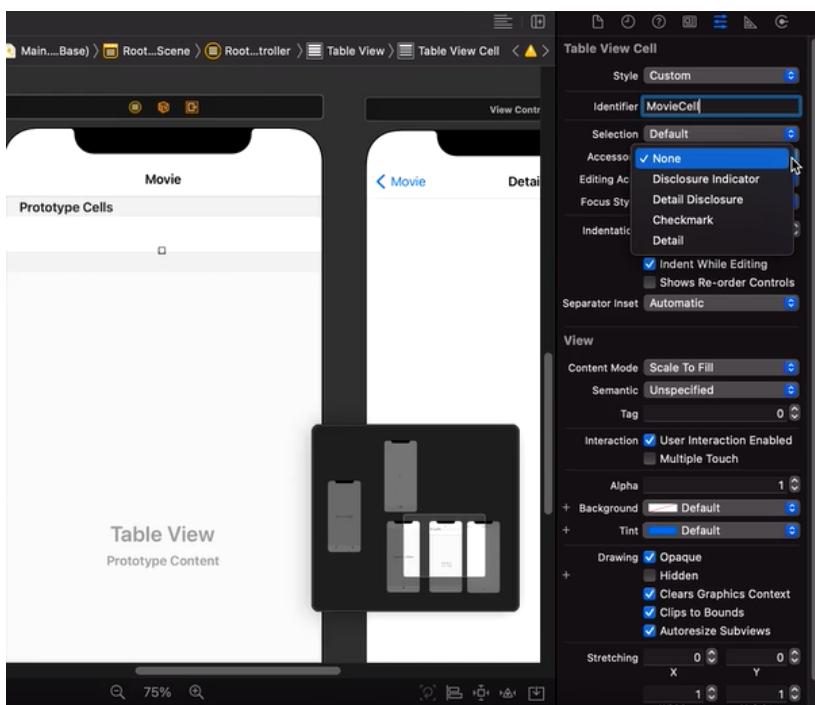


이제, Navigation Bar 타이틀을 Movie로 바꾼다.

두번째 화면의 타이틀을 Detail로 바꾼다.



그 후에, Prototype cells의 속성창을 설정해준다.



Identifier에서 MovieCell로 설정해준다.

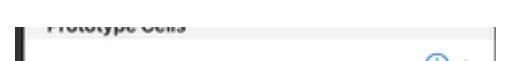
Selection에서 4가지 종류의 액세서리를 고를 수 있다.

#### Disclosure Indicator

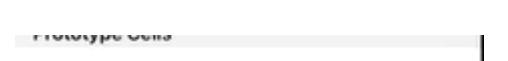


버튼을 누르면 상세한 화면으로 넘어간다는 뜻

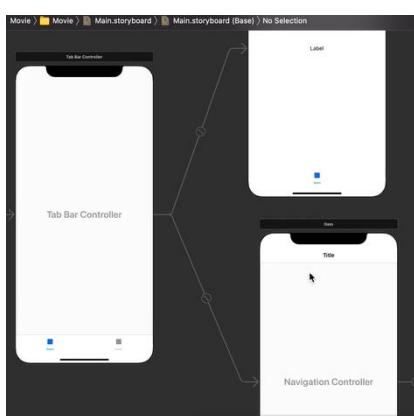
#### Detail Disclosure



#### Checkmark



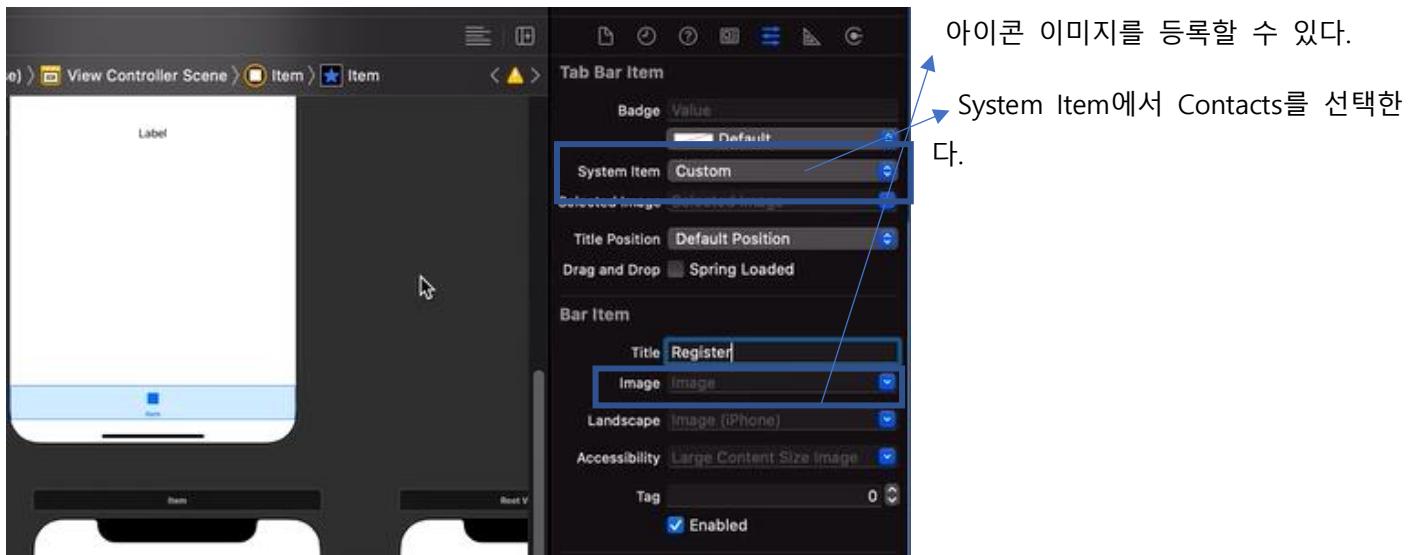
그 중에 우리는 Disclosure Indicator를 선택한다.



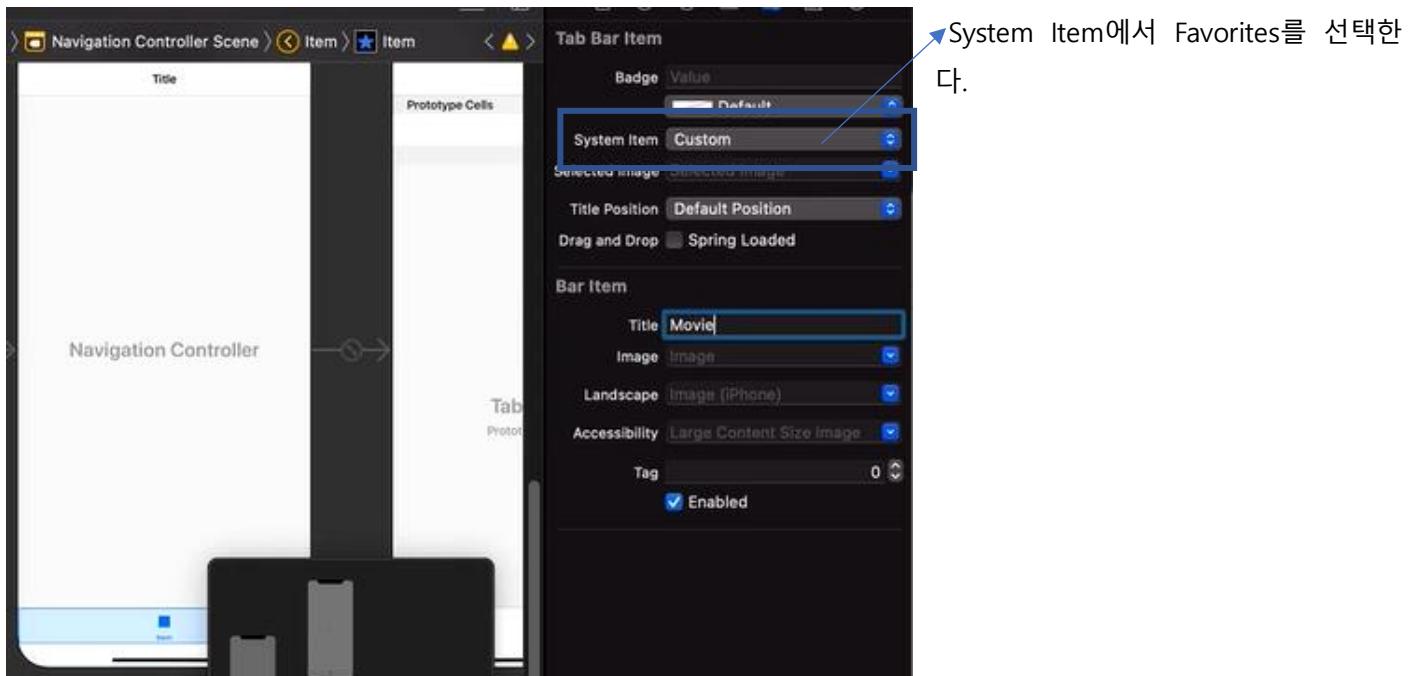
Tab Bar Controller에서 2개 버튼이 있는데, 첫번째 버튼을 누르게 되면, Label을 가지고 있는 view controller로 이동을 하게 되고, 두 번째 버튼을 누르게 되면, Navigation Controller로 이동하게 된다.

우리는 버튼의 타이틀을 바꾸려고 할 때, 연계된 view controller에서 편집해야 된다.

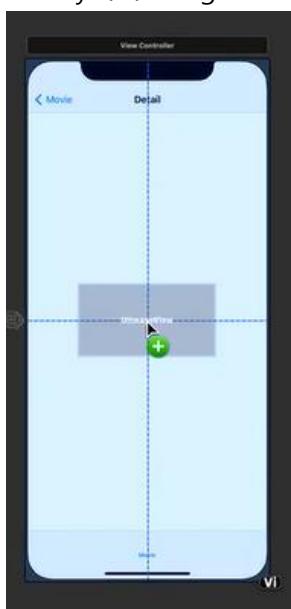
Tab Bar Item 클릭하고 속성창을 열고, Bar Item에서 Title 부분에 Register로 한다.



Navigation controller 탭 부분도 Title을 Movie로 바꾼다.



Library에서 Image View를 선택해서



이쯤에 배치한다.

정렬 배치 조건을 추가해준다.

Add New Alignment Constraints에서



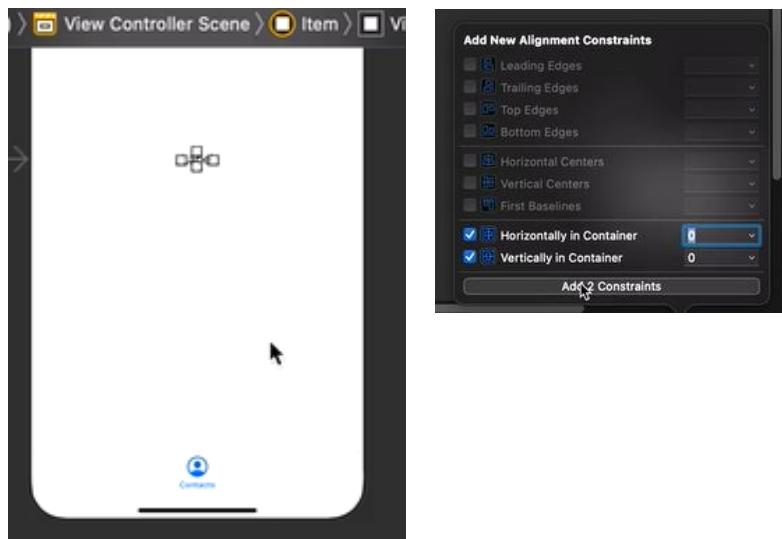
이와 같은 조건을 설정해준다.

아직도 조건이 부족하므로, 가로세로 크기를 설정해야 한다.

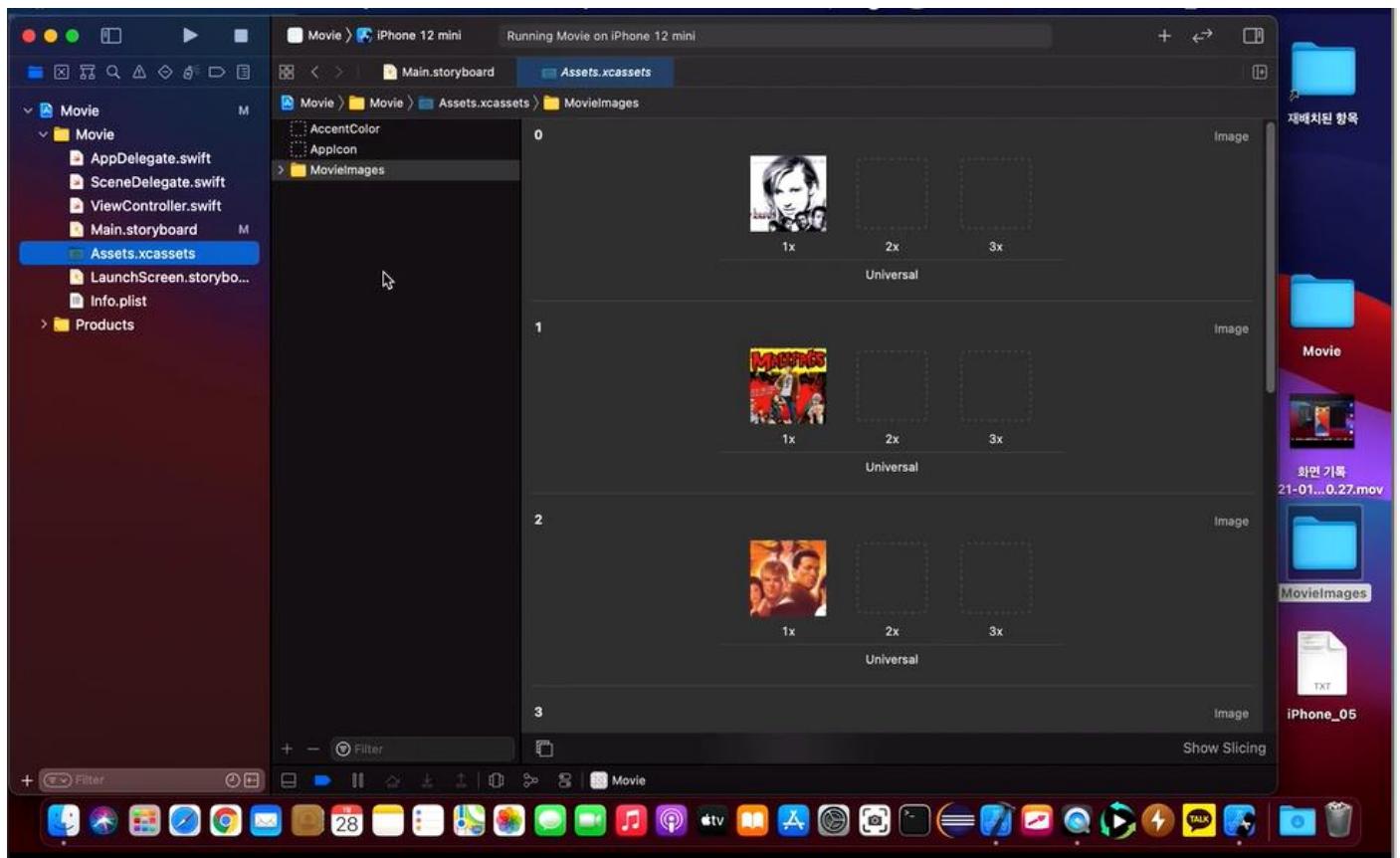
이와 같이 설정해준다.



여기에 아직도 조건을 주지 않았으므로, 정렬조건을 준다.



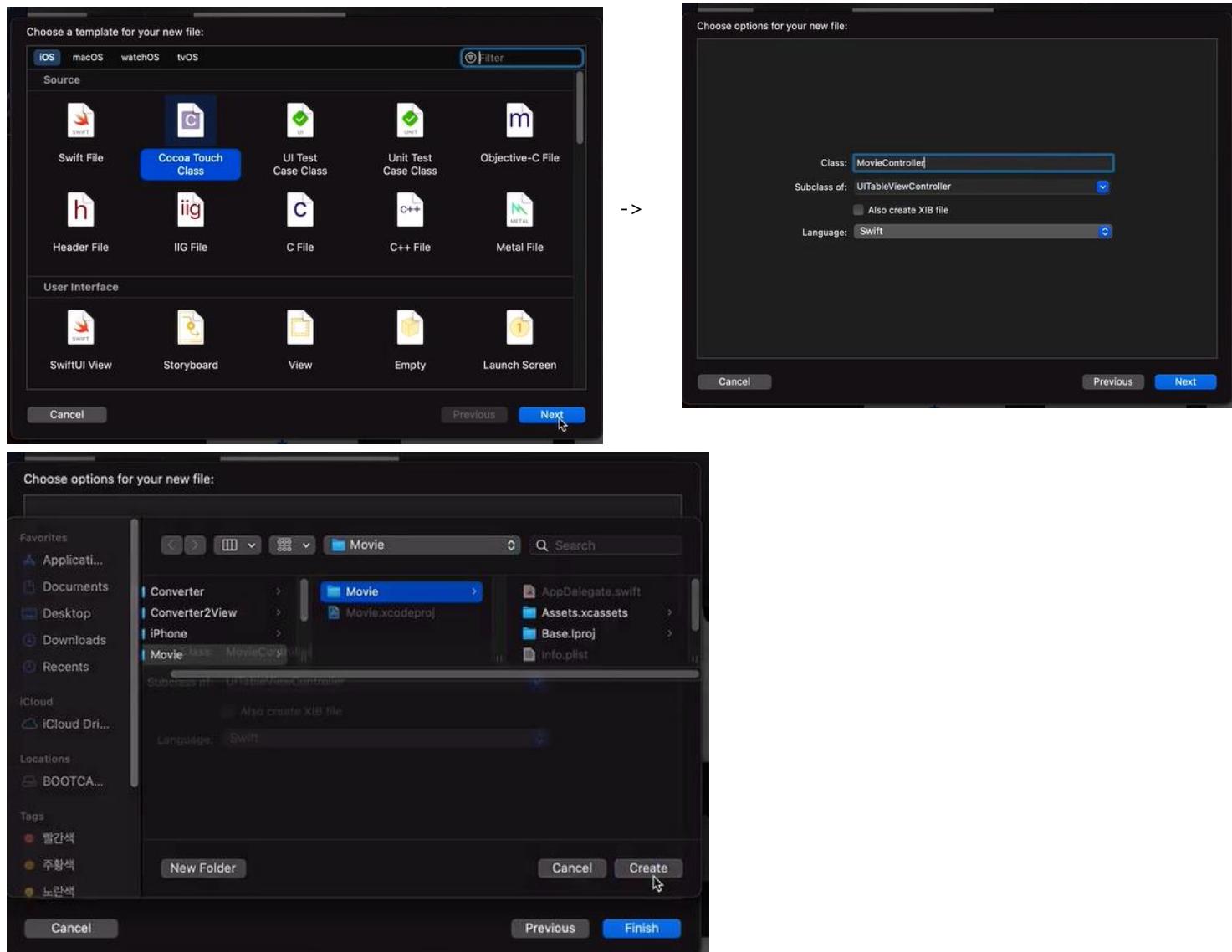
Assets.xcassets은 영상이나 음성파일들을 넣는 공간이다.



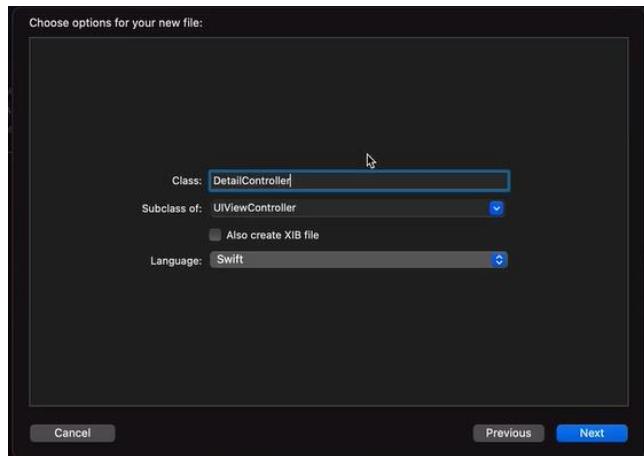
MovieImages을 Assets.xcassets안에 drop시킨다.

## 이제 코딩 작업

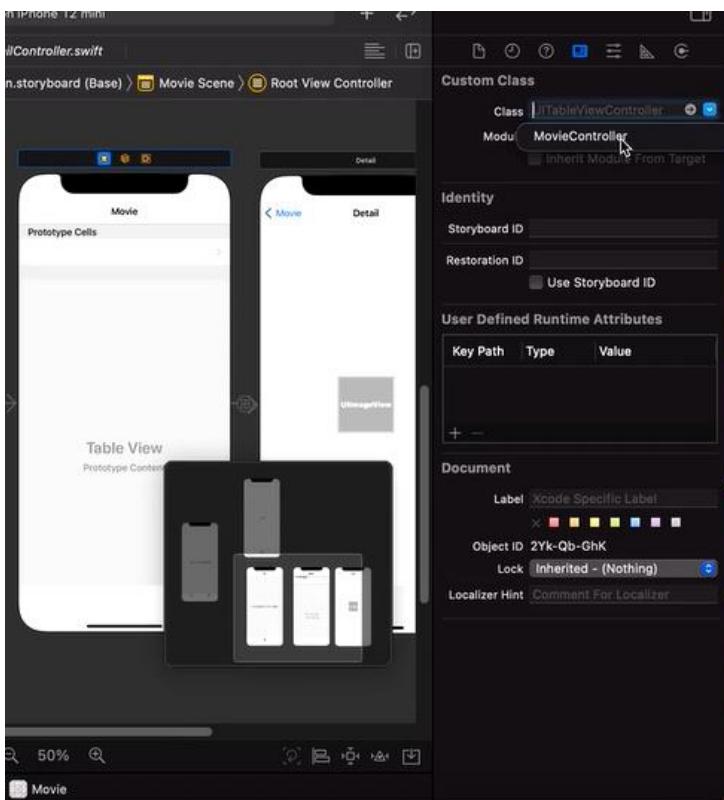
Swift 파일을 생성한다.



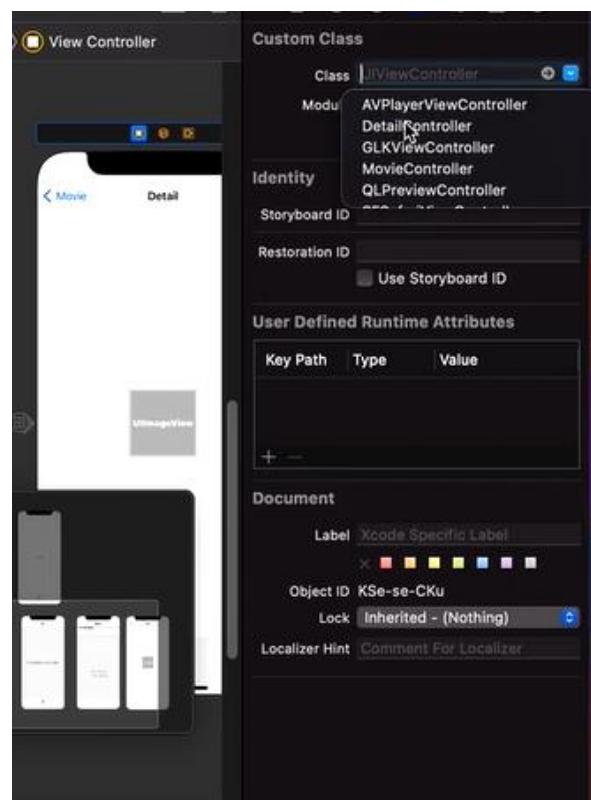
2번째 swift파일 생성하기 이번에는 이와 같이 설정



## 연결해주기 – Movie controller



## Detail Controller 연결해주기



이와 같이 코드 작성하기

```
class MovieController: UITableViewController {
    let movies = ["Chasing Amy", "Mallrats", "Dogma",
                 "Clerks", "Jay & Silent Bob Strike Back", "Red State",
                 "Cop Out", "Jersey Girl"]
    var movieImages: [UIImage] = []

    override func viewDidLoad() {
        super.viewDidLoad()

        // Uncomment the following line to preserve selection between presentations
        // self.clearsSelectionOnViewWillAppear = false

        // Uncomment the following line to display an Edit button in the navigation bar for this
        // view controller.
        // self.navigationItem.rightBarButtonItem = self.editButtonItem

        super.viewDidLoad()

        var i = 0
        for _ in movies {
            let image = UIImage(named: "\((i)).jpg")
            movieImages.append(image!)
            i = i + 1
        }
    }

    override func numberOfSections(in tableView: UITableView) -> Int {
        // #warning Incomplete implementation, return the number of sections
        return 1
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
    {
        // #warning Incomplete implementation, return the number of rows
        return movies.count
    }
}
```

주석부분을 활성화시키고, 일부분을 수정한다.

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "MovieCell", for: indexPath)

    // Configure the cell...
    cell.textLabel?.text = movies[indexPath.row]

    return cell
}
```

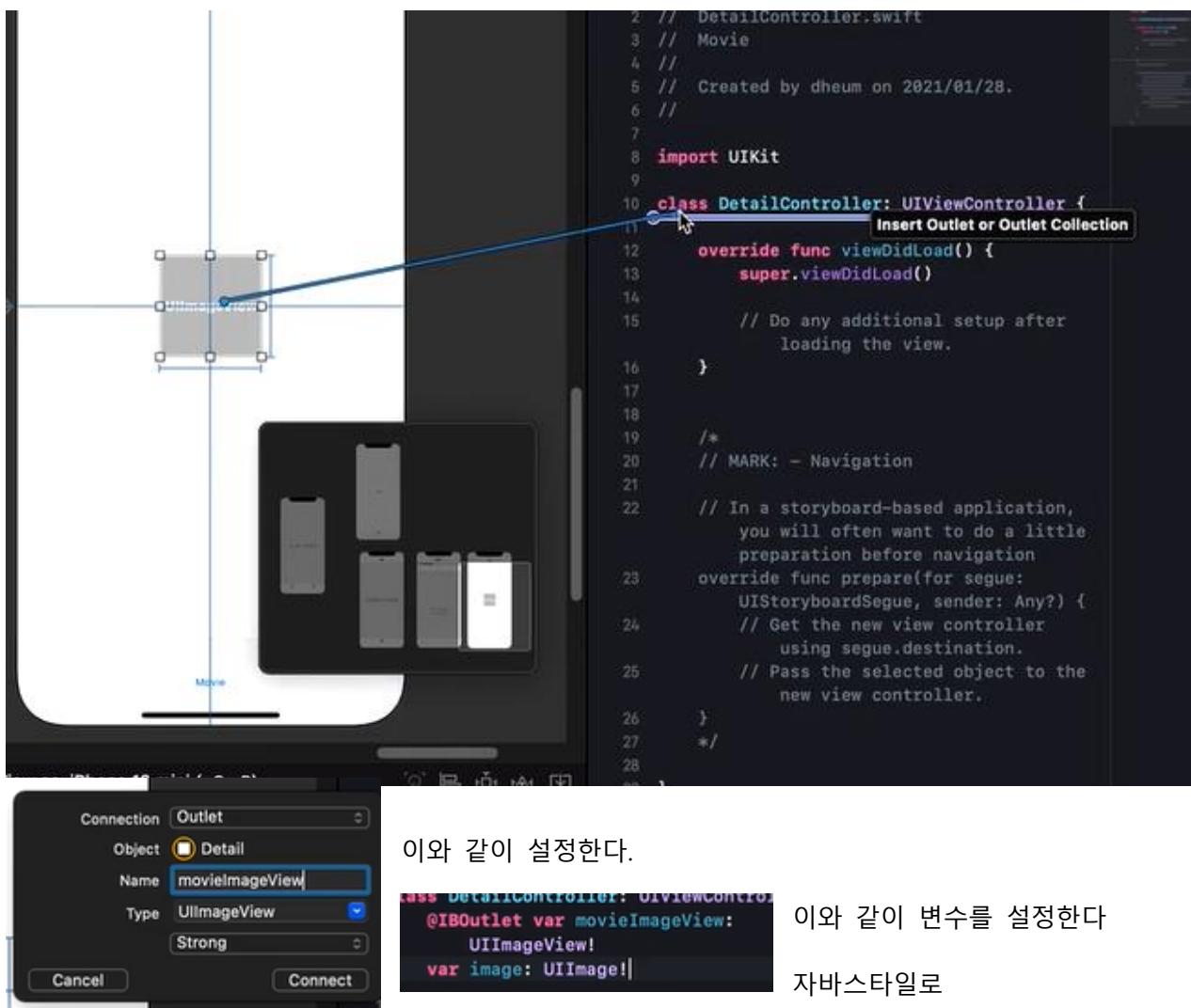
Prepare 메소드 주석해제를 시킨다.

```
navigation
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // Get the new view controller using segue.destination.
    // Pass the selected object to the new view controller.
    let destination = segue.destination as? DetailController
    let index = tableView.indexPathForSelectedRow?.row
    if let destination = destination, let index = index {
        destination.image = self.movieImages[index] // Value of type 'DetailController' has no member 'i...
    }
}
```

오류가 발생이 되는데 이를 해결하고자 Main.storyboard로 이동한다.

Detail view controller로 가서 Assistant를 클릭한다.

ImageView에서 control dragging을 한다.



그리고나서

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Do any additional setup after
    // loading the view.
    movieImageView.image = image
}
```

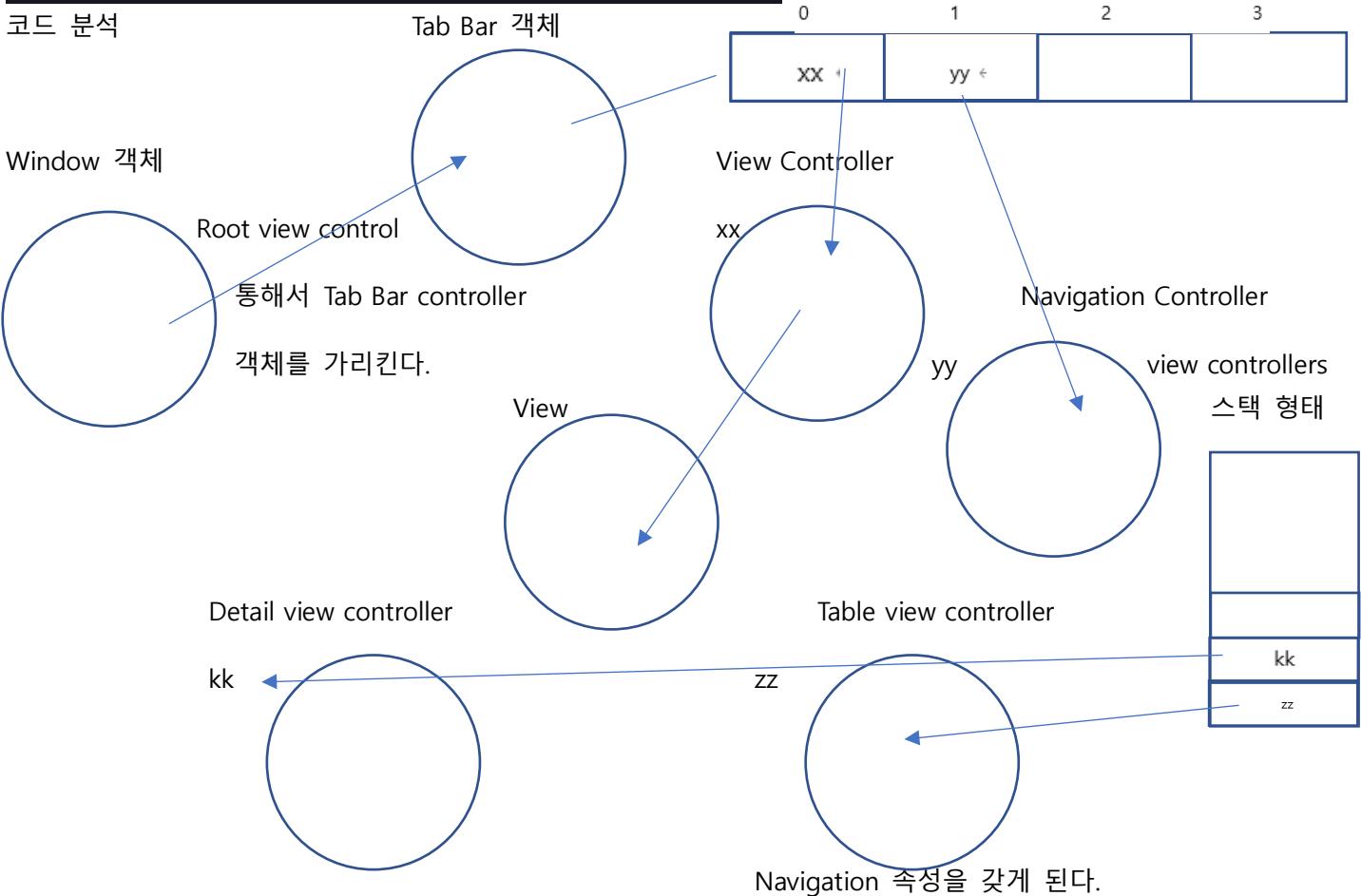
이 부분을 추가한다.

그 후에 Movie Controller로 가면 오류가 사라진다.

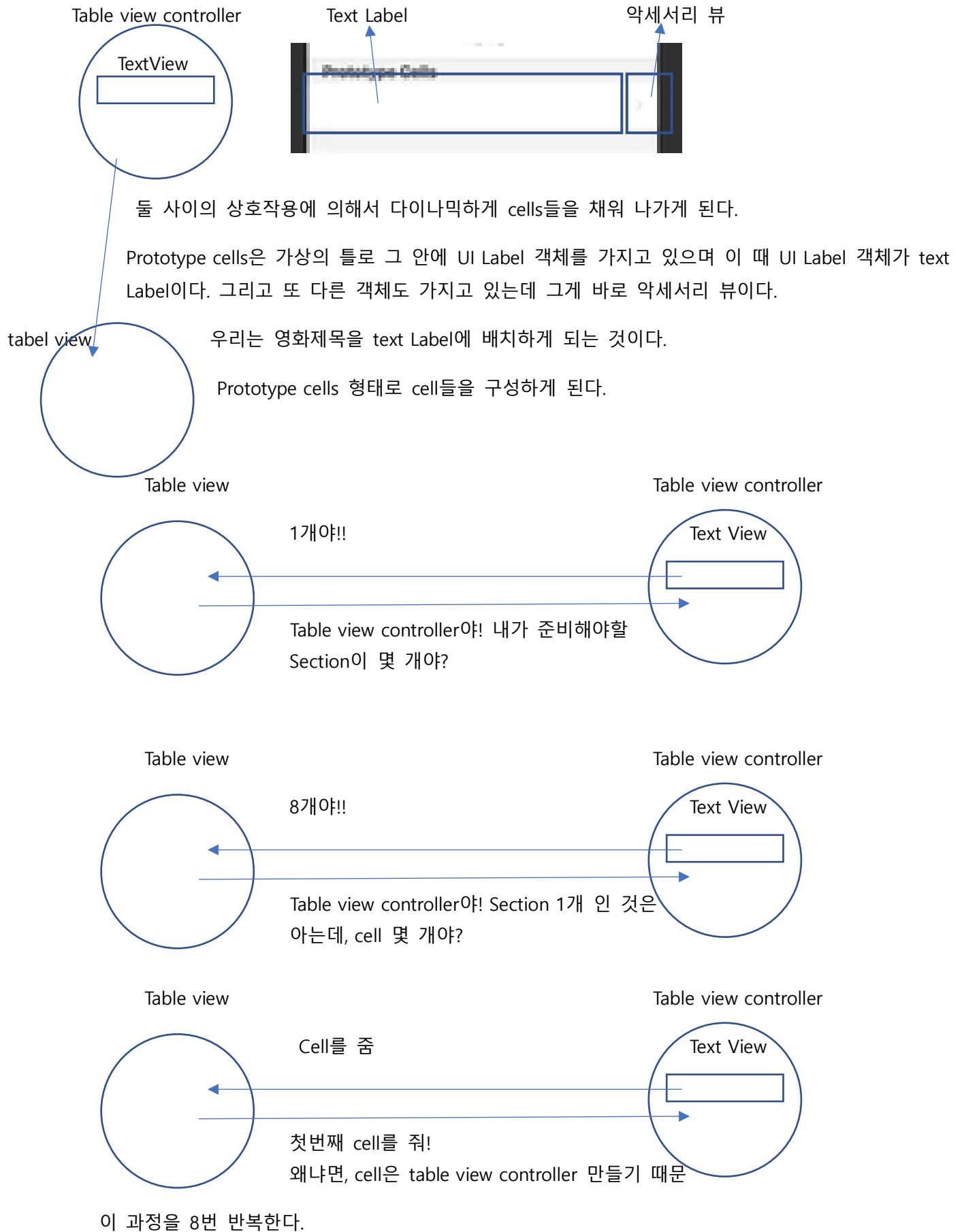
```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // Get the new view controller using segue.destination.
    // Pass the selected object to the new view controller.
    let destination = segue.destination as? DetailController
    let index = tableView.indexPathForSelectedRow?.row
    if let destination = destination, let index = index {
        destination.image = self.movieImages[index]
    }
}
```

viewControllers -> 배열

코드 분석



## Table View controller



## Movie controller.swift

Class MovieController: UITableViewController{

```
let movies = ["Chasing Amy", "Mallrats", "Dogma",
    "Clerks", "Jay & Silent Bob Strike Back", "Red State",
    "Cop Out", "Jersey Girl"]

var movieImages: [UIImage] = [] // 이미지 객체를 담긴 배열

override func viewDidLoad() {
    super.viewDidLoad()

    var i = 0

    for _ in movies {
        let image = UIImage(named: "₩(i).jpg") //UI이미지를 생성한다. -> 옵셔널 타입
                                                //UI이미지 생성하는 데 실패할 수 있으므로
        movieImages.append(image!) //이미지 배열 안에 넣는다.

        //-> 이때 옵셔널을 강제 벗기기를 한다.

        i = i + 1
    }
}

//몇 개 세션을 생성해야 되는지를 묻는 함수이다. -> 지금 세션 하나이므로, return 1로 해준다.

override func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}

//한 개 세션에서 몇 개의 cell를 구성할 것인지를 묻는 함수이다 -> 8개임을 알 수 있다.

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return movies.count
}
```

//8번 호출해서 cell를 생성해서 UITableView에게 전달해준다.

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
  
    let cell = tableView.dequeueReusableCell(withIdentifier: "MovieCell", for: indexPath)  
  
    cell.textLabel?.text = movies[indexPath.row]  
  
    return cell  
}
```

//이미지 객체를 detail controller에게 넘겨주는 상황이다.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
  
    let destination = segue.destination as? DetailController //다운캐스팅 형 변환  
  
    let index = tableView.indexPathForSelectedRow?.row //몇 번째 cell이 사용자한테 touch가 되었는  
    //지를 알아야 한다.  
  
    if let destination = destination, let index = index { //둘 다 옵셔널 형태이기 때문에 if let 사용  
  
        destination.image = self.movieImages[index]  
    }  
  
}  
  
}  
  
let cell = tableView.dequeueReusableCell(withIdentifier: "MovieCell", for: indexPath) 이 부분
```

Table view

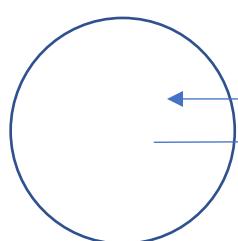
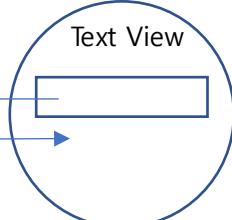


Table view controller



재사용이 가능한 cell를 큐 형태로 관리하고 있다.



Table view은 눈에 안 보이는 cell를 다시 요청할 때, table view controller 큐 안에 있는지 묻고, 없으면 다시 만들어준다.

## Detail controller 부분

Class DetailController: UIViewController{

```
@IBOutlet var movieImage: UIImageView!
```

```
var image: UIImage!
```

```
override func viewDidLoad(){
```

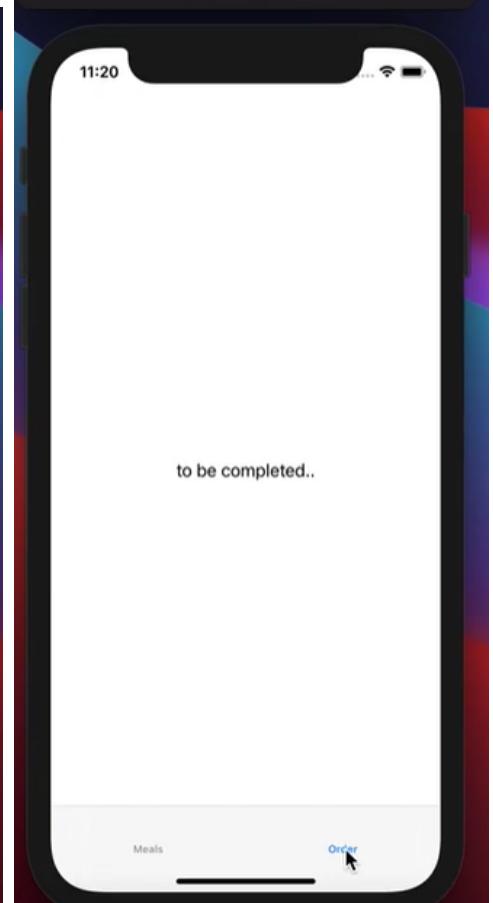
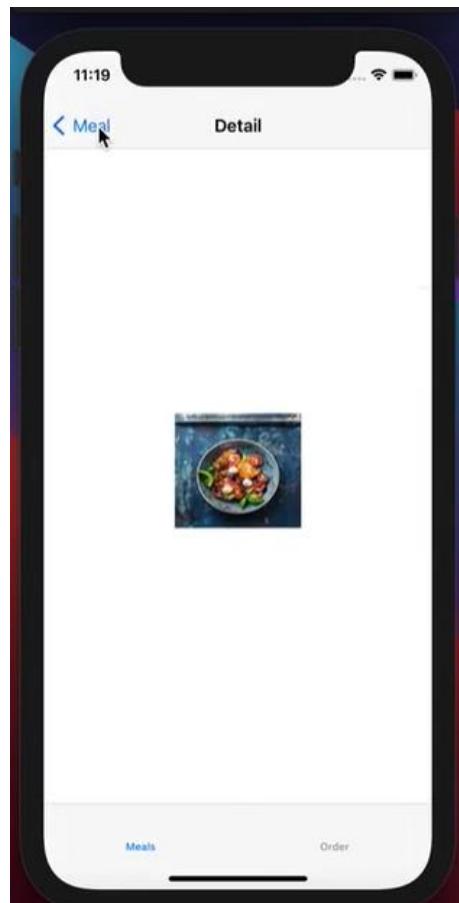
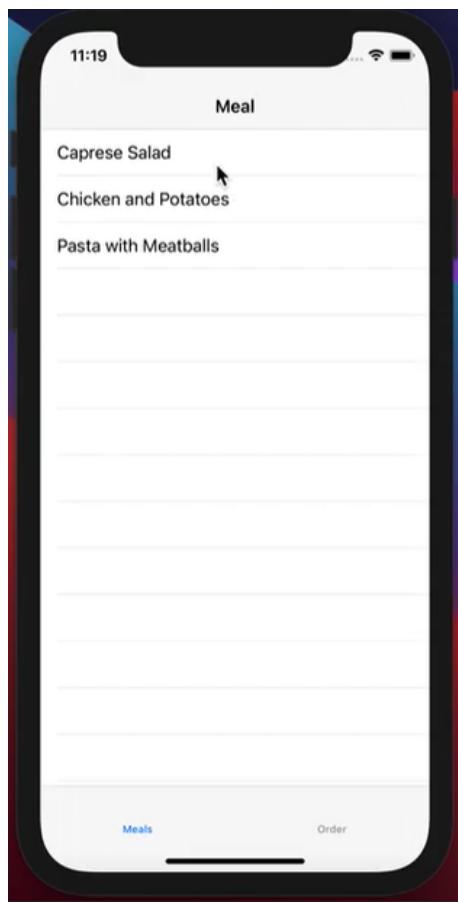
```
super.viewDidLoad()
```

```
movieImage.image = image
```

```
}
```

```
}
```

Homework!!

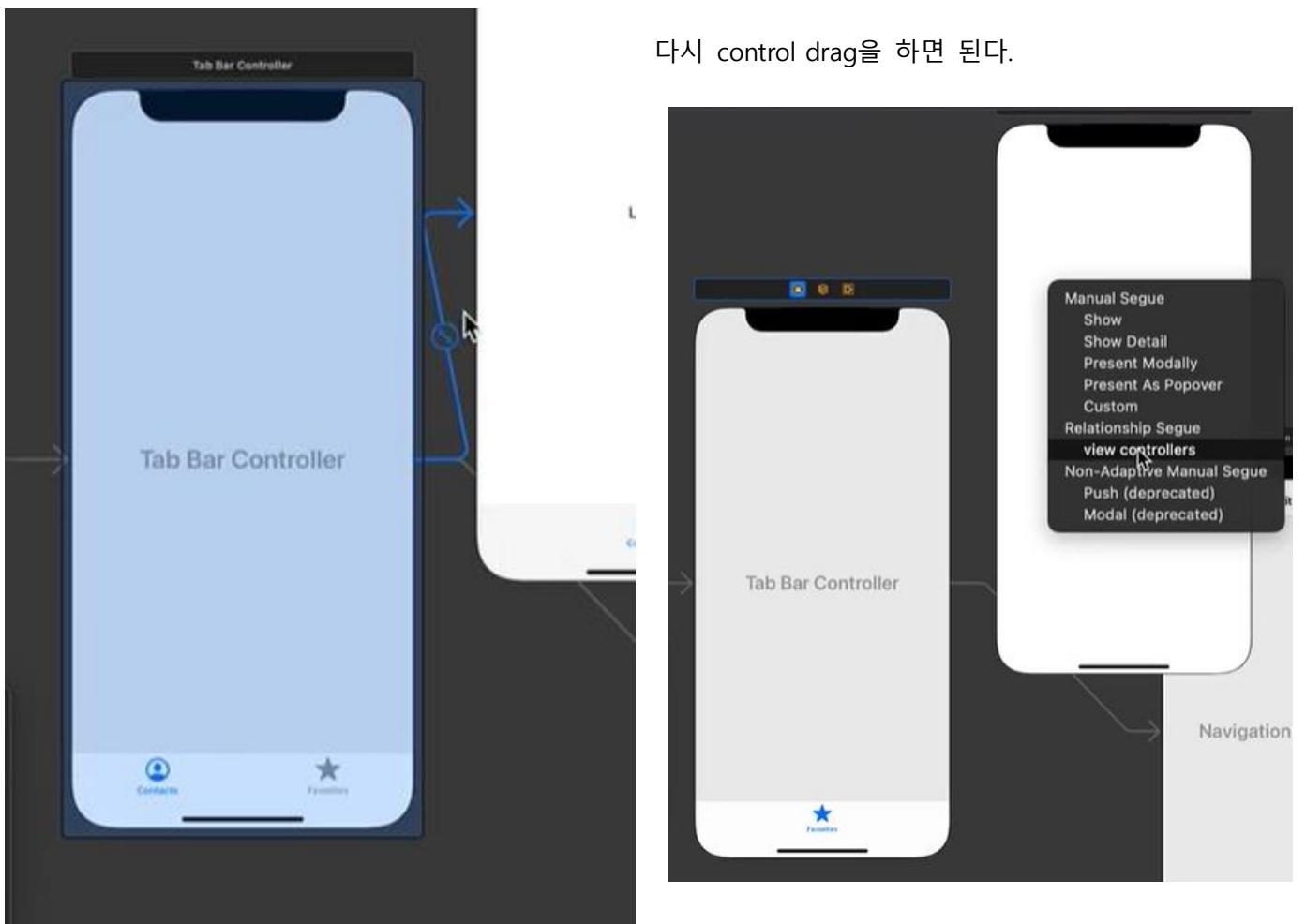


순서바꾸는 법

이것을 지운다.

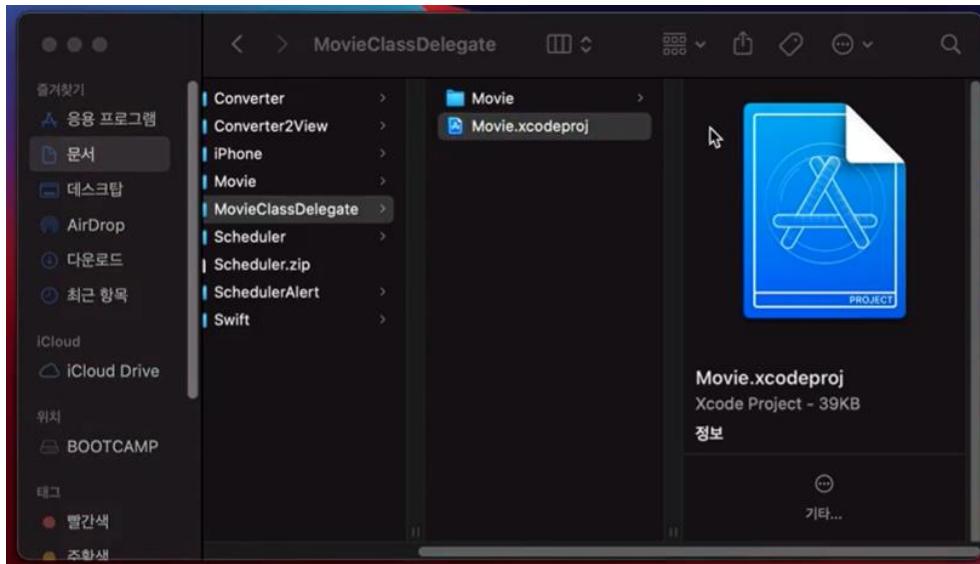
잘 안지우면 다른 view controller 선택하고 다시 지우면 된다.

다시 control drag을 하면 된다.



이것을 하면 된다. 화면의 순서가 변경이 된다.

이와 같이 파일을 변경 -> MovieClassDelegate로



library에서 label를 dragging 해서 적절한

위치에 drop해준다.

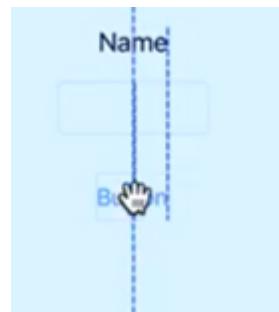
그 후에 더블 클릭해서 label 편집해서 "Name"으로 바꾼다.



그 후에, library에서 text Field를 적절한 위치에 dragging해서 drop한다.

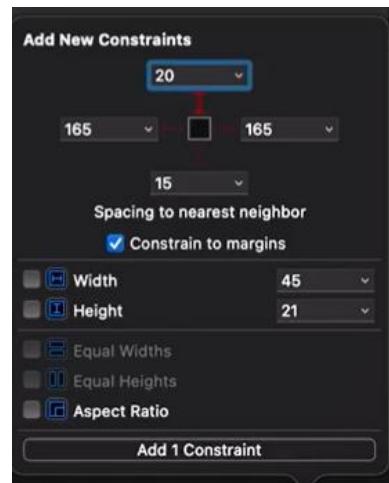


그 후에 library에서 button 하나를 가지고 와서, 적절한 위치에 배치해 둔다.

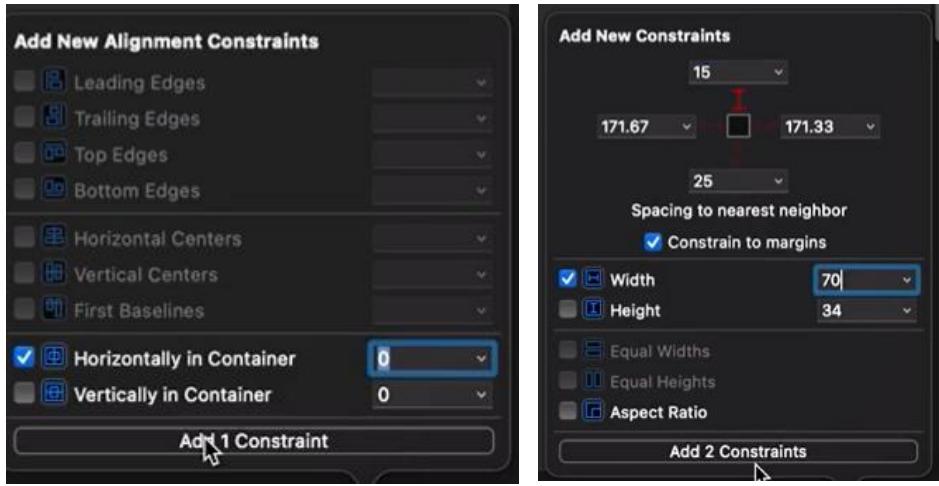


Button도 편집해서 register로 변경한다.

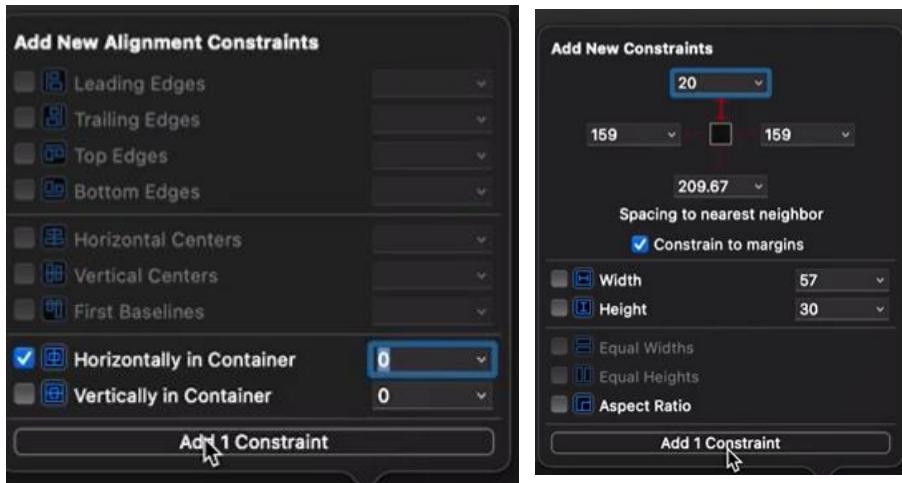
Name Label을 클릭하고 수평으로 정렬한다. 그 후에 위로 20 설정



## Text Field 배치 설정하기

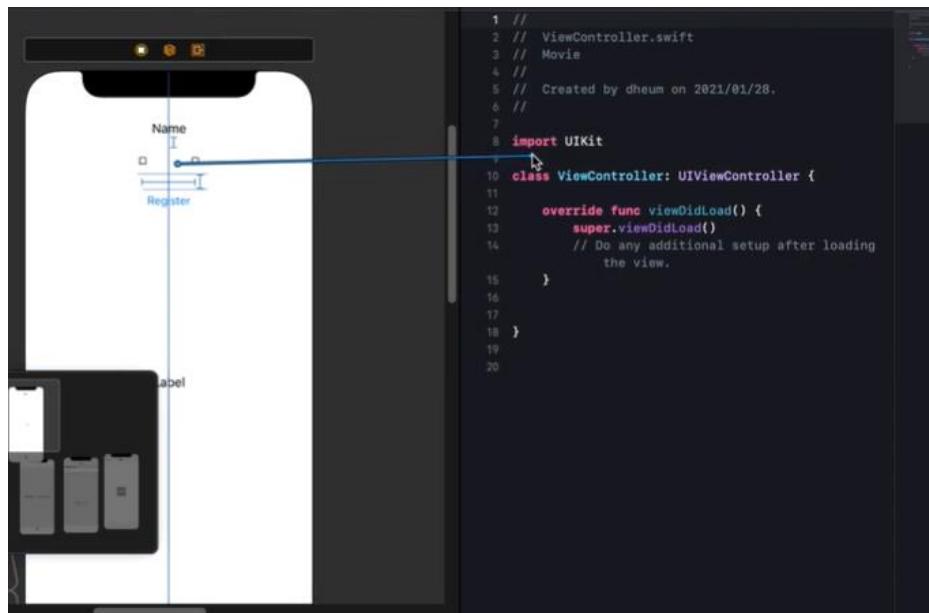


## Register 버튼 설정



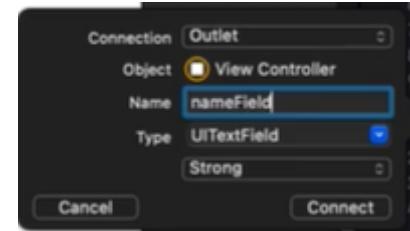
3개다 잘 배치가 되었다.

Label이랑 text field 연결하기 -> 먼저, Assistant를 누르기

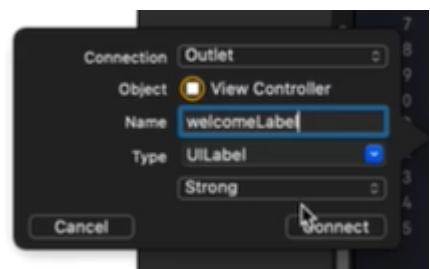


Control를 누른 상태에서 dragging하기

View controller class 안에 drop을 한다.

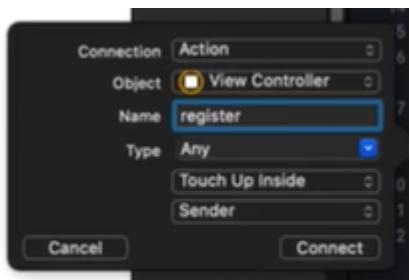


Label을 control를 누른 상태에서 dragging하기 적당한 위치에 drop하기



```
1 //  
2 // ViewController.swift  
3 // Movie  
4 //  
5 // Created by dheum on 2021/01/28.  
6 //  
7  
8 import UIKit  
9  
10 class ViewController: UIViewController {  
11     @IBOutlet var nameField: UITextField!  
12     override func viewDidLoad() {  
13         super.viewDidLoad()  
14         // Do any additional setup after loading  
15         the view.  
16     }  
17  
18 }  
19  
20  
21
```

버튼도 설정해야 하므로, 버튼을 오른쪽 마우스를 누른 상태에서 dragging하고 적절한 위치에 drop한다.



```
1 //  
2 // ViewController.swift  
3 // Movie  
4 //  
5 // Created by dheum on 2021/01/28.  
6 //  
7  
8 import UIKit  
9  
10 class ViewController: UIViewController {  
11     @IBOutlet var nameField: UITextField!  
12     @IBOutlet var welcomeLabel: UILabel!  
13  
14     override func viewDidLoad() {  
15         super.viewDidLoad()  
16         // Do any additional setup after loading  
17         the view.  
18     }  
19  
20 }  
21  
22
```

함수 틀이 만들어진다.

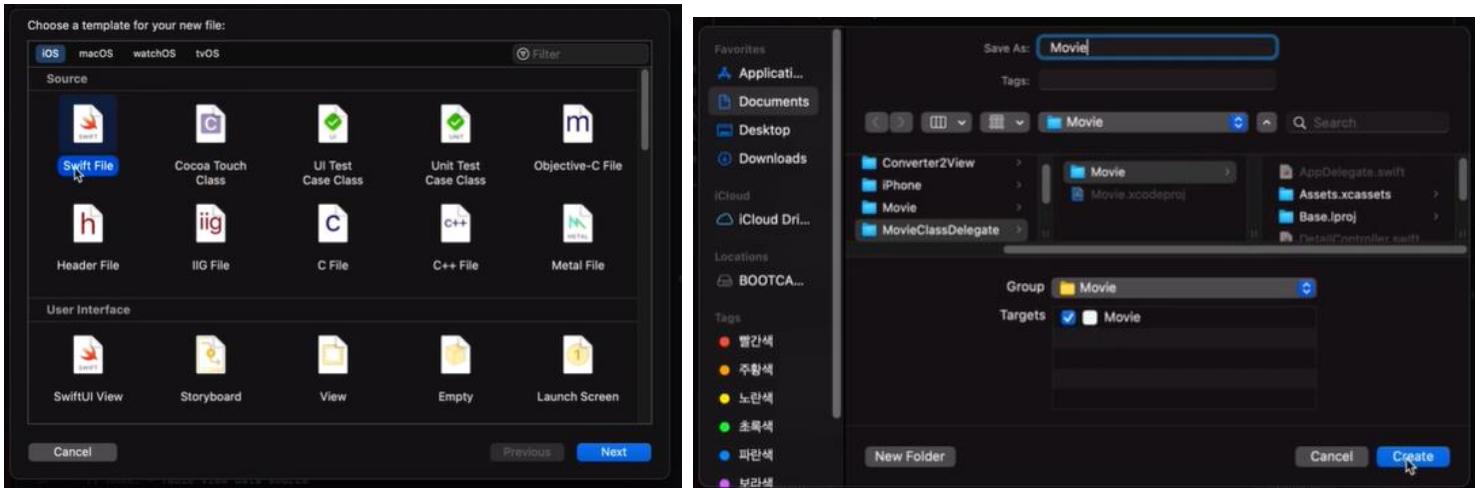
## ViewController.swift 부분

```
@IBAction func register(_ sender: Any) {  
    self.nameField.resignFirstResponder()  
    let text = nameField.text  
    if text!.isEmpty {  
        welcomeLabel.text = "Enter name, please~"  
    } else {  
        welcomeLabel.text = "\(text!) is registered!"  
        let naviController = self.tabBarController?.viewControllers?[1] as? UINavigationController  
        let controller = naviController?.viewControllers[0] as? MovieController  
        controller?.nameString = nameField.text  
    }  
}
```

```
extension ViewController: UITextFieldDelegate {  
  
    func textFieldDidBeginEditing(_ textField: UITextField) {  
        self.nameField.backgroundColor = UIColor.green  
    }  
  
    func textFieldShouldEndEditing(_ textField: UITextField) -> Bool {  
        textField.backgroundColor = UIColor.lightGray  
        return true;  
    }  
}
```

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    // Do any additional setup after loading the view.  
    nameField.delegate = self  
}
```

## 새로운 파일을 만들기



그 후에 Movie.swift에

```
8 import UIKit
9
10 class Movie {
11     var title: String?
12     var description: String?
13     var image: UIImage?
14 }
15
16 let dataset = [
17     ("Chasing Amy", "1997 American romantic comedy-drama film", "0.jpg"),
18     ("Mallrats", "1995 American romantic buddy comedy film", "1.jpg"),
19     ("Dogma", "1999 American fantasy comedy film", "2.jpg"),
20     ("Clerks", "1994 American independent black-and-white buddy comedy film", "3.jpg"),
21     ("Jay & Silent Bob Strike Back", "2001 American comedy film", "4.jpg"),
22     ("Red State", "2011 American independent horror thriller film", "5.jpg"),
23     ("Cop Out", "2010 American buddy cop action-comedy film", "6.jpg"),
24     ("Jersey Girl", "2004 American comedy-drama film", "7.jpg")
25 ]
26
27 var movies: [Movie] = []
28
```

이와 같이 코드 작성한 후에

MovieController.swift로 가서

```
class MovieController: UITableViewController {
    let movies = ["Chasing Amy", "Mallrats", "Dogma",
                  "Clerks", "Jay & Silent Bob Strike Back", "Red State",
                  "Cop Out", "Jersey Girl"]
    var movieImages: [UIImage] = []
}

override func viewDidLoad() {
    super.viewDidLoad()
```

movies배열과 movieImages 배열 부분을 지운다.

그 후, 그 자리에 var nameString: String?이라는 변수를 정의해준다.

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Uncomment the following line to preserve selection between presentations
    // self.clearsSelectionOnViewWillAppear = false

    // Uncomment the following line to display an Edit button in the navigation bar for
    // controller.
    // self.navigationItem.rightBarButtonItem = self.editButtonItem

    var i = 0
    for _ in movies {
        let image = UIImage(named: "\\(i).jpg")
        movieImages.append(image!)
        i += 1
    }
}
```

배열로 사용했던 부분들을 제거해준다.

그 자리에 해당 코드를 넣어준다.

-> view WillAppear 메소드 생성해주고, 메소드 안에 작업하기

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Uncomment the following line to preserve selection
    // self.clearsSelectionOnViewWillAppear = false

    // Uncomment the following line to display an Edit button
    // controller.
    // self.navigationItem.rightBarButtonItem = self.editButtonItem

    for (title, description, imageName) in dataset {
        let movie = Movie()
        movie.title = title
        movie.description = description
        movie.image = UIImage(named: imageName)
        movies.append(movie)
    }
}
```

```
override func viewWillAppears(animated: Bool) {
    super.viewWillAppears(animated)
    if let name = self.nameString {
        self.navigationItem.title = "Movie-\(name)"
    }
}
```

tableView에서 movies[indexPath.row].title이라고 수정

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
    UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "MovieCell", for: indexPath)

    // Configure the cell...
    cell.textLabel?.text = movies[indexPath.row].title

    return cell
}
```

메소드 하나 추가 -> 코드 작성하기

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    // getting the current cell from the index path
    let currentCell = tableView.cellForRow(at: indexPath)
    // getting the text of that cell
    let currentText = currentCell?.textLabel?.text
    let alertController = UIAlertController(title: "Your Choice", message: "You Selected " +
        currentText!, preferredStyle: .alert)
    let defaultAction = UIAlertAction(title: "Close", style: .default, handler: nil)
    alertController.addAction(defaultAction)

    present(alertController, animated: true, completion: nil)
}
```

그 후에 여기서 let alertController = UIAlertController(title: "Your Choice", message: "You Selected " + currentText!, preferredStyle: .actionSheet)로 바꾸어준다.

Alert->actionSheet로

Self를 지우기

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // Get the new view controller using segue.destination.
    // Pass the selected object to the new view controller.
    let destination = segue.destination as? DetailController
    let index = tableView.indexPathForSelectedRow?.row
    if let destination = destination, let index = index {
        destination.image = movies[index].image
    }
}
```

여러 작업을 했으면 해당 오류가 사라진다.

```
IBAction func register(_ sender: Any) {
    self.nameField.resignFirstResponder()
    let text = nameField.text
    if text!.isEmpty {
        welcomeLabel.text = "Enter name, please~"
    } else {
        welcomeLabel.text = "\(text!) is registered!"
        let naviController = self.tabBarController?.viewControllers?[1] as? UINavigationController
        let controller = naviController?.viewControllers[0] as? MovieController
        controller?.nameString = nameField.text
    }
}
```

디자인 패턴 -> 건축학에서 시작 -> 구조물을 어떻게 구성할 것인지 각 구조물들이 어떤 기능을 분담할 것인지를 표준화 시켰다.

-> 객체지향에서는 -> 구성되는 객체들을 어떻게 구성할 것이며 역할분담을 어떻게 할 것인지를 만들었다.

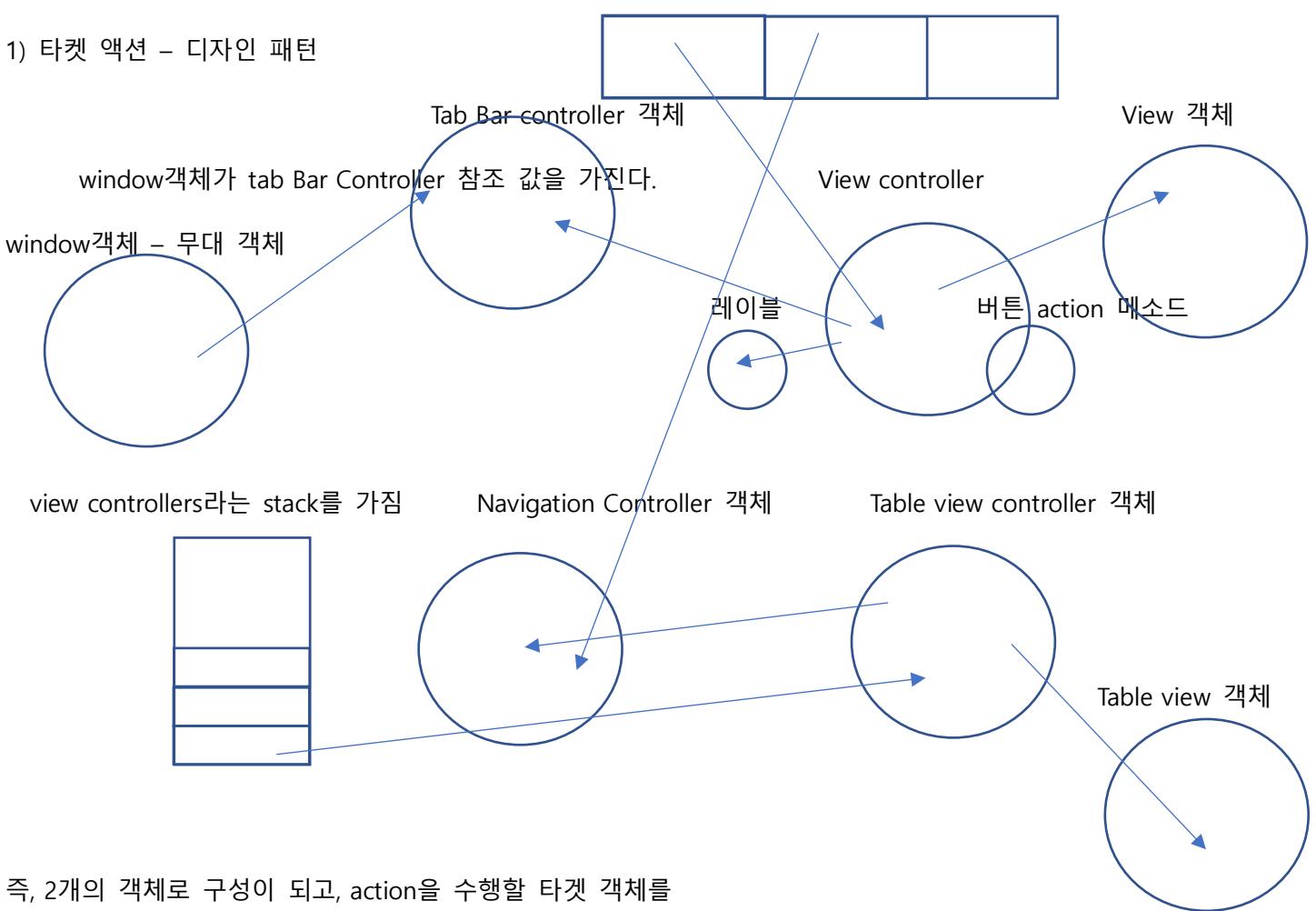
-> 우리가 살펴볼 디자인 패턴을 3가지 살펴볼 것이다.

1) 타겟 액션 - 디자인 패턴

2) MVC 패턴( Model, View, Controller)

3) Delegate

1) 타켓 액션 – 디자인 패턴



즉, 2개의 객체로 구성이 되고, action을 수행할 타겟 객체를

지정하면서, 그 객체를 실행할 메소드를 갖게 된 형태이다.

이해가 되지 않으므로 메소드를 통해 다시 살펴보자! -> 이 메소드가 타켓-액션 패턴이다.

ViewController.swift

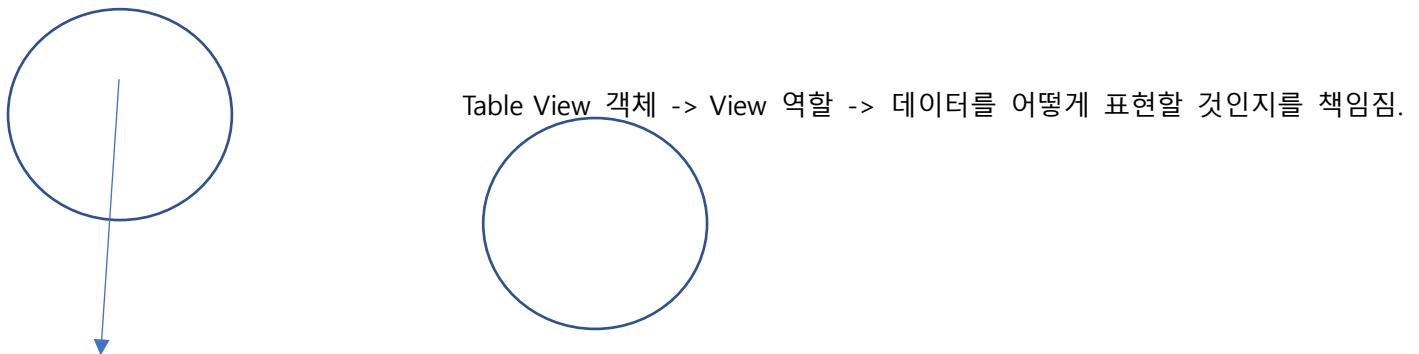
```
@IBAction func register(_ sender: Any) {  
  
    self.nameField.resignFirstResponder() //nameField객체야!! 키보드 잡고 있는 거 포기해라!!  
  
    let text = nameField.text //nameField에 있는 text를 가져온다. -> nil값일 수도 있으므로, 옵셔널 타입이다.  
  
    if text!.isEmpty { //nameField는 빈 문자열이라도 오기 때문에 강제 벗기기가 가능하다!  
  
        //nameField의 문자열이 빈 문자열이면 if문 안으로 진입한다.  
  
        welcomeLabel.text = "Enter name, please~" //welcomeLabel의 문자열을 바꾼다.  
  
    } else {  
  
        welcomeLabel.text = "₩(text!) is registered!" //welcomeLabel의 문자열을 바꾼다.  
  
        let naviController = self.tabBarController?.viewControllers?[1] as? UINavigationController  
  
        //tabBarController의 viewControllers는 배열 형태이지만 옵셔널 형태이다.  
  
        //viewControllers 안에 있는 view Controller가 옵셔널 형태이기 때문에 as?로 코딩함.  
  
        let controller = naviController?.viewControllers[0] as? MovieController  
  
        //naviController의 viewControllers는 viewControllers는 스택 형태이지만 옵셔널 형태가 아니다.  
  
        //viewControllers 안에 있는 view Controller가 옵셔널 형태이기 때문에 as?로 코딩함.  
  
        controller?.nameString = nameField.text //문자열을 MovieController에 있는 nameString에게 전달  
    }  
}
```

이와 같이 설정을 하면 MovieController.swift에 있는 코드 일부분을 보면

```
override func viewWillAppear(_ animated: Bool) {  
  
    super.viewWillAppear(animated)  
  
    //nil이면 pass하지만 nameString 값을 nil이 아니면,  
  
    //navigationItem의 title이 바뀌게 된다는 것을 알 수 있다.  
  
    if let name = self.nameString {  
  
        self.navigationItem.title = "Movie-₩(name)"  
  
    }  
}
```

## 2) MVC 디자인 패턴

Table View Controller 객체 -> Controller



Movie.swift 사용 -> Model은 Movie.swift 안에 있는 배열들을 의미한다.

이와 같이 분리한 이유는 데이터를 변화가 생기거나 view 표현 방식을 바꿀 때 전체코드에 넣게 되면 전부다 건드려야 하는 문제점이 발생하지만 분리하게 되면 그 부분만 수정할 수 있다.

```
class Movie {  
  
    var title: String?  
  
    var description: String?  
  
    var image: UIImage?  
  
}
```

let dataset = [ //실제 데이터 형태 -> 튜플 형식으로 되어 있음.

```
("Chasing Amy", "1997 American romantic comedy-drama film", "0.jpg"),  
(("Mallrats", "1995 American romantic buddy comedy film", "1.jpg"),  
("Dogma", "1999 American fantasy comedy film", "2.jpg"),  
("Clerks", "1994 American independent black-and-white buddy comedy film", "3.jpg"),  
("Jay & Silent Bob Strike Back", "2001 American comedy film", "4.jpg"),  
("Red State", "2011 American independent horror thriller film", "5.jpg"),  
("Cop Out", "2010 American buddy cop action-comedy film", "6.jpg"),  
("Jersey Girl", "2004 American comedy-drama film", "7.jpg")  
]
```

var movies: [Movie] = [] // Movie형 배열을 담을 수 있고, 지금 현재 빈 배열로 선언

```
MovieController.swift
```

```
Import UIKit
```

```
class MovieController: UITableViewController{
```

```
    var nameString: String? //textfield 내용을 전달받기 위한 용도
```

```
    override func viewDidLoad(){
```

```
        super.viewDidLoad()
```

```
        for (title, description, imageName) in dataset { //dataset는 튜플들의 데이터 배열이다.
```

```
            //반복문으로, Movie 데이터를 생성하고 movies 안에 넣어준다.
```

```
            let movie = Movie()
```

```
            movie.title = title
```

```
            movie.description = description
```

```
            movie.image = UIImage(named: imageName) //UIImage를 생성자를 통해서 만들게 된다.
```

```
            movies.append(movie)
```

```
}
```

```
}
```

```
    override func viewDidAppear(_ animated: Bool){
```

```
        super.viewDidAppear(animated)
```

```
        if let name = self.nameString {
```

```
            self.navigationItem.title = "Movie-\u{20}(\name)"
```

```
}
```

```
}
```

```
//몇 개 세션을 생성해야 되는지를 묻는 함수이다. -> 지금 세션 하나이므로, return 1로 해준다.
```

```
    override func numberOfSections(in tableView: UITableView) -> Int {
```

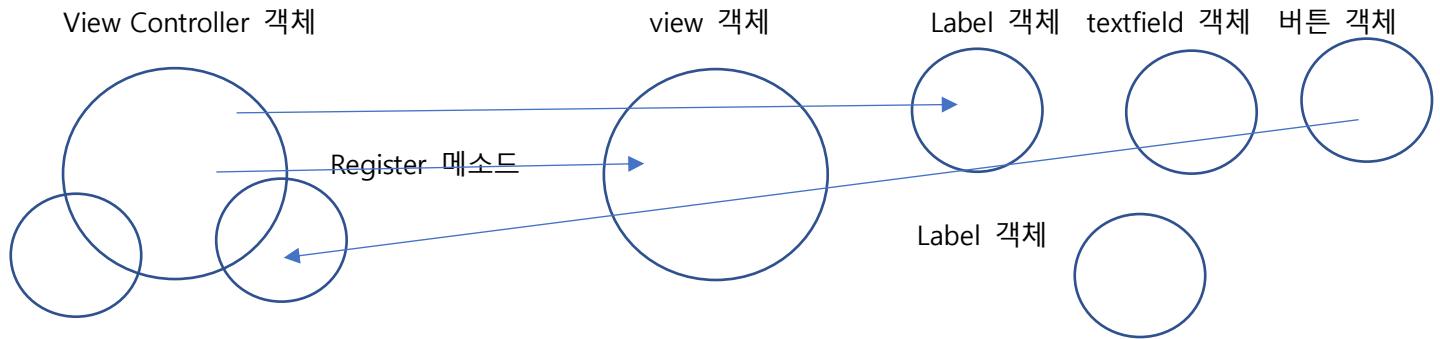
```
        return 1
```

```
}
```

```
//한 개 세션에서 몇 개의 cell를 구성할 것인지를 묻는 함수이다 -> 8개임을 알 수 있다.
```

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
    return movies.count  
}  
  
//8번 호출해서 cell를 생성해서 UITableView에게 전달해준다.  
  
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {  
    let cell = tableView.dequeueReusableCell(withIdentifier: "MovieCell", for: indexPath)  
  
    cell.textLabel?.text = movies[indexPath.row].title // cell만들 때 이 부분을 수정해야 함.  
  
    return cell  
}  
  
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    let destination = segue.destination as? DetailController //다운캐스팅 형 변환  
  
    let index = tableViewindexPathForSelectedRow?.row //몇 번째 cell이 사용자한테 touch가 되었는//지를  
    알아야 한다.  
  
    if let destination = destination, let index = index { //둘 다 옵셔널 형태이기 때문에 if let 사용  
        destination.image = movies[index].image //이 부분을 수정해야 한다. -> 이미지 값을 전달한다.  
    }  
}  
}
```

### 3) Delegate 패턴



이 때, textfield가 해야 할 일이 많을 때 조력자들을 구성해야 하는 상황이 발생

-> 이 때 사용한 패턴이 delegate이다

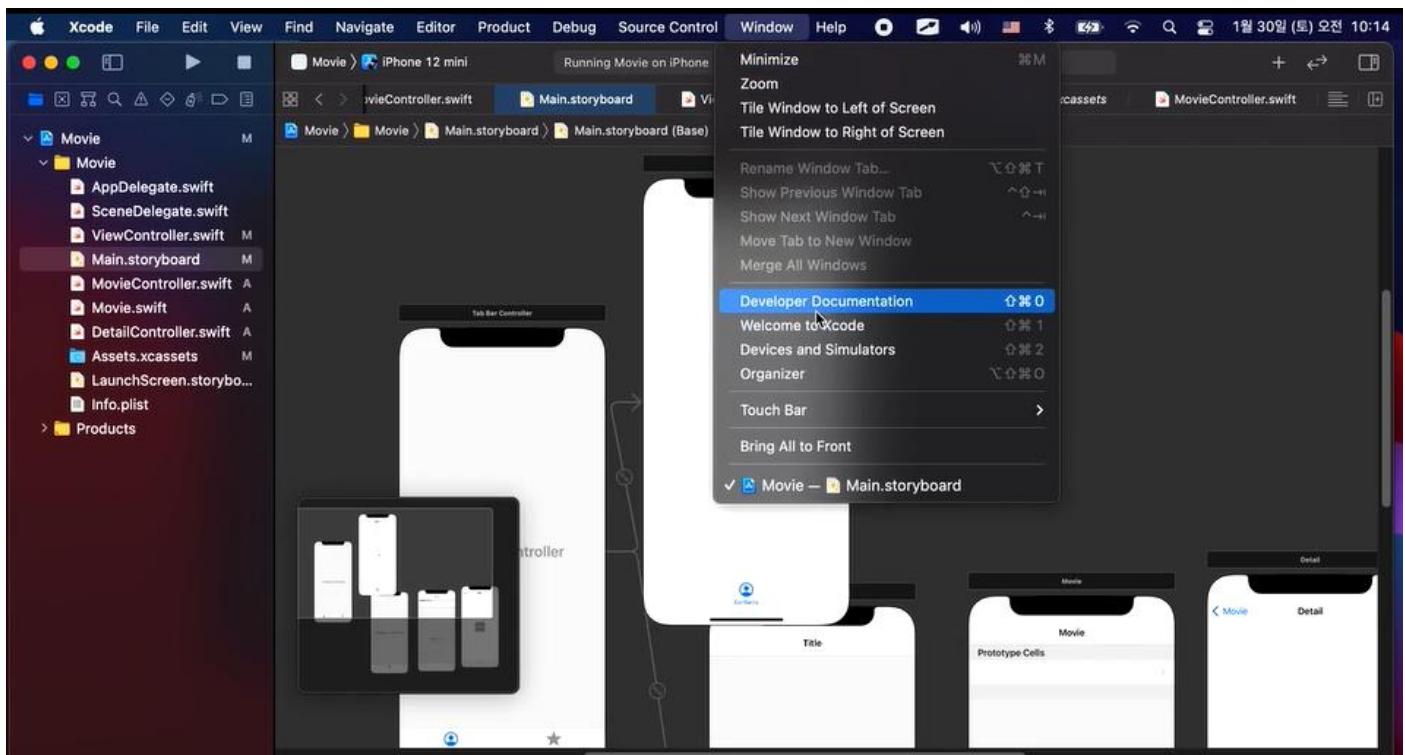
-> 조력자 객체를 따로 생성하는 것보다는 기존에 있는 객체를 조력자로 사용하는 것이 좋다.

그래서 textfield의 delegate 속성을 이용하여 view controller 객체를 조력자로 지정해둔다.

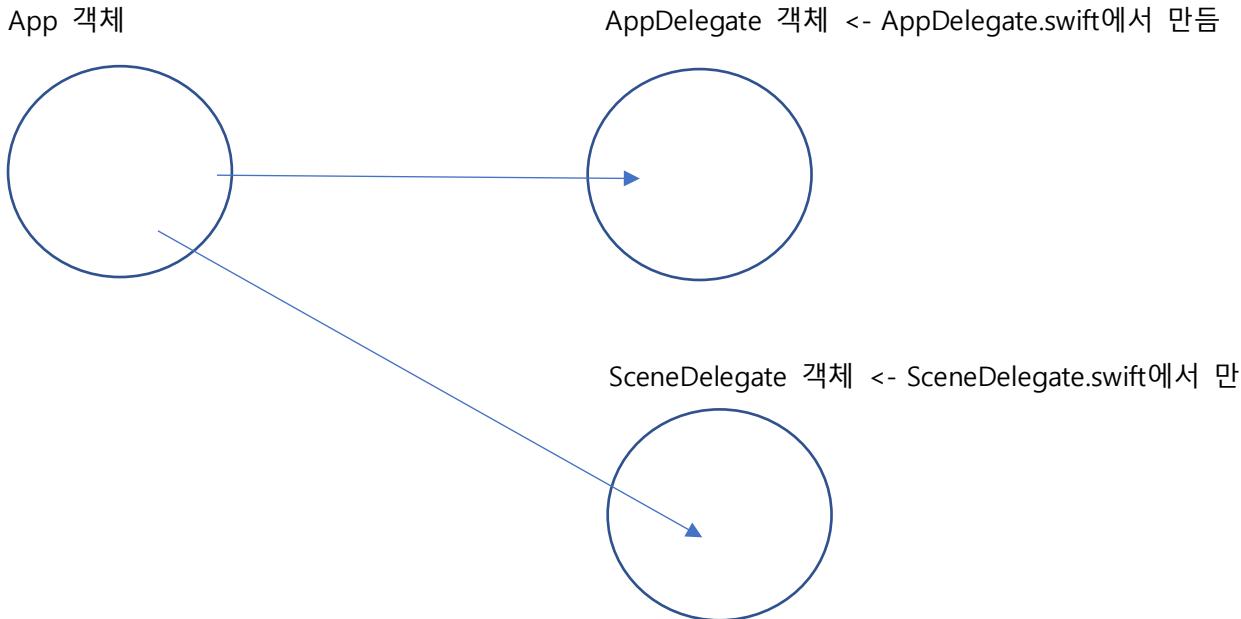
그래서 textfield가 사용자 간의 상호작용을 할 때, view controller 안에 있는 메소드가 호출을하게 된다.

ex) 고급 기능 예는 글자 수 제한하거나 기호 입력에 제한하는 데 이용

그리고 textfield의 조력자로 활동하려면, UITextFieldDelegate 프로토콜 규정에 따라야 한다.



Developer Documentation을 통해 protocol과 struct 명을 검색하면 어떤 필드와 어떤 메소드를 가지고 있는지를 확인할 수 있다.



AppDelegate.swift로 가면 준수해야할 프로토콜이 있다는 것을 알 수 있다.

```
class AppDelegate: UIResponder, UIApplicationDelegate {
```

여기에서는 UIApplicationDelegate 프로토콜을 준수해야 하는 것을 알 수 있다.

SceneDelegate.swift로 가면 준수해야할 프로토콜이 있다는 것을 알 수 있다.

```
class SceneDelegate: UIResponder, UIWindowSceneDelegate {
```

여기에서는 UIWindowSceneDelegate 프로토콜을 준수해야 하는 것을 알 수 있다.

이제 어떻게 textfield에 delegate를 했는지를 알아보기 위해서 ViewController.swift 코드를 보자

Import UIKit

```
class ViewController: UIViewController{
    @IBOutlet var nameField: UITextField!
    @IBOutlet var welcomeLabel: UILabel!

    override func viewDidLoad() {
        super.viewDidLoad()
        nameField.delegate = self
    }
}
```

```

@IBAction func register(_ sender: Any) {

    self.nameField.resignFirstResponder()

    let text = nameField.text

    if text!.isEmpty {

        welcomeLabel.text = "Enter name, please~"

    } else {

        welcomeLabel.text = "₩(text!) is registered!"

        let naviController = self.tabBarController?.viewControllers?[1] as? UINavigationController

        let controller = naviController?.viewControllers[0] as? MovieController

        controller?.nameString = nameField.text

    }

}

}

extension ViewController: UITextFieldDelegate { //viewController를 확장해주는 부분으로

    // UITextFieldDelegate 프로토콜을 상속받는 것을 추가
}

```

```

func textFieldDidBeginEditing(_ textField: UITextField) {

    self.nameField.backgroundColor = UIColor.green // 사용자 textfield를 터치하는 순간 textfield의 색깔이

                                                // 초록색으로 바뀌게 된다.

}

func textFieldShouldEndEditing(_ textField: UITextField) -> Bool {

    textField.backgroundColor = UIColor.lightGray // 사용자 textfield를 editing을 마칠 때, textfield의 색깔이

                                                // 회색으로 바뀌게 된다.

                                                // 보통 register 버튼을 누를 때 이 메서드가 호출이 된다.

    return true;

}

```

Extension을 사용하지 않고

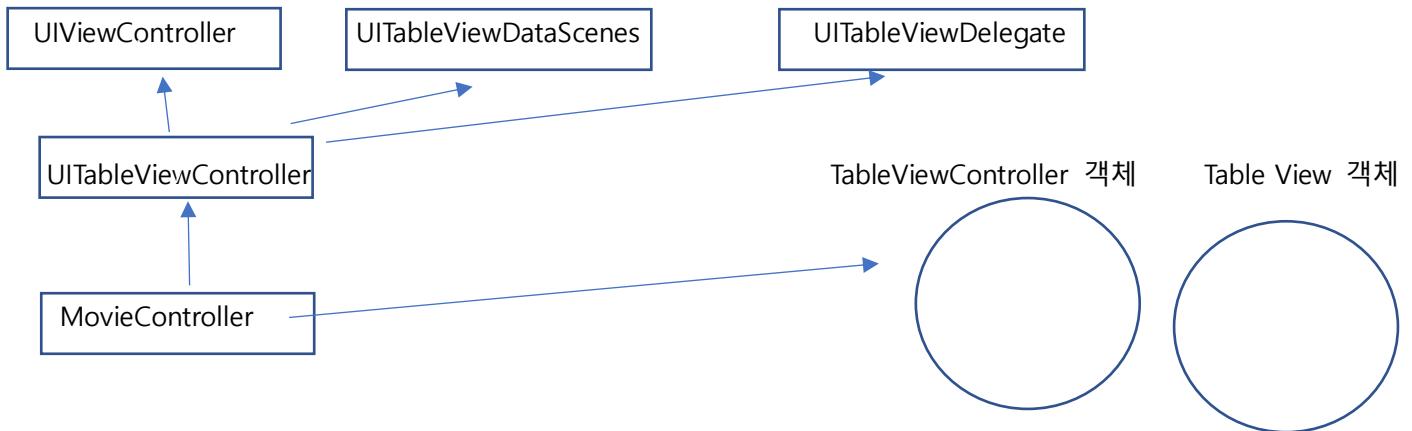
```
class ViewController: UIViewController, UITextFieldDelegate{
```

이와 같은 형식으로 사용하게 되면

그 안에다가, textFieldDidBeginEditing와 textFieldShouldEndEditing를 사용할 수도 있지만.

분리해서 추가 기능을 추가하는 게 코드 유지 보수하는 게 좋다.

이제 MovieController.swift를 살펴보자



상속에 의해서 UITableViewController은 UITableViewDelegate 프로토콜을 준수해야 하는 것을 알 수 있다.

Table를 구성하기 위해서 호출되는 메서드는 UITableViewScenes에서 규정이 되어 있다.

Table view한테 데이터를 제공하는 역할을 한다.

MovieController.swift 내용 일부분 -> 해당 함수은 cell클릭이 할 때, 호출되는 함수

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
    //getting the current cell from the index path
    let currentCell = tableView.cellForRow(at: indexPath) // 선택한 cell의 값을 가져옴
    //getting the text of that cell
    let currentText = currentCell?.textLabel?.text //선택한 cell의 textLabel의 text 값을 가져옴
    let alertController = UIAlertController(title: "Your Choice", message: "You Selected " + currentText! ,
                                           preferredStyle: .actionSheet) //actionSheet를 올라옴.

    let defaultAction = UIAlertAction(title: "Close", style: .default, handler: nil) //alert창을 띄우게 됨.
    alertController.addAction(defaultAction)
    present(alertController, animated: true, completion: nil)
}
```